Efficient Determination of Copper Electroplating Chemistry Additives using Advanced Neural Network Algorithms

by

Charles David Ellis

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 10, 2015

Approved by:

Robert N. Dean, Chair, Associate Professor of Electrical & Computer Engineering
Bogdan M. Wilamowski, Co-chair, Professor of Electrical & Computer Engineering
Michael C. Hamilton, Assistant Professor of Electrical & Computer Engineering
Richard C. Jaeger, Professor Emeritus of Electrical & Computer Engineering

Abstract

Copper plating is the metallization process of choice for modern semiconductor devices. It has proven to be a relatively inexpensive and simple process and easily adaptable to high volume manufacturing. The development of the copper plating process includes the addition of organic components that provide a uniform and smooth surface [1]–[5] These organic components are not only consumed during the plating process, but also decompose over time [6]. To insure a repeatable process these organic components, typically added in parts per million concentrations, must be carefully controlled. To do this, the industry has developed chemical analysis techniques such as Modified Linear Approximation, Dilution-titration, and Response curves to assist in determining the exact concentration of the organics in the plating bath. These techniques, while widely used, are time consuming, wasteful, and inaccurate. A new technique is proposed that will speed up the process, reduce the complexity and waste, and provides a higher accuracy [7]. These techniques will utilize recently introduced second order Advanced Neural Network (ANN) algorithms developed at Auburn University.

Acknowledgements

I would like to thank my wife, Julie, my son David, and my daughter Anna, for their love and support during my professional career. And to the Electrical & Computer Engineering Department who has provided me with stimulating work for the past 29 years.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In the early 1990s IBM introduced a process for replacing aluminum with copper in IC metallization – this process, called Damascene [8], was quickly adopted by most high volume manufacturers and has helped to provide increased speeds and miniaturization that have allowed a continual advancement of IC circuit functionality to the present day.

The Damascene process, shown in Figure 1, contains many steps including seed-layer deposition, pattern delineation, copper plating, and chemical mechanical polishing. In this work a review of the copper plating process is presented in order to familiarize the reader with the complicated chemistry and the reasoning behind the careful additive component control necessary to achieve repeatable plating results. A review of present component analysis techniques is given to show their complexity and wastefulness, and why new techniques would be beneficial. A review of some of the most popular advanced neural network algorithms is presented to show their differences and why new second order techniques are capable of converging for very non-linear datasets. These neural network algorithms are compared and demonstrate the ability to predict minute concentrations of the plating component with much less waste and time.

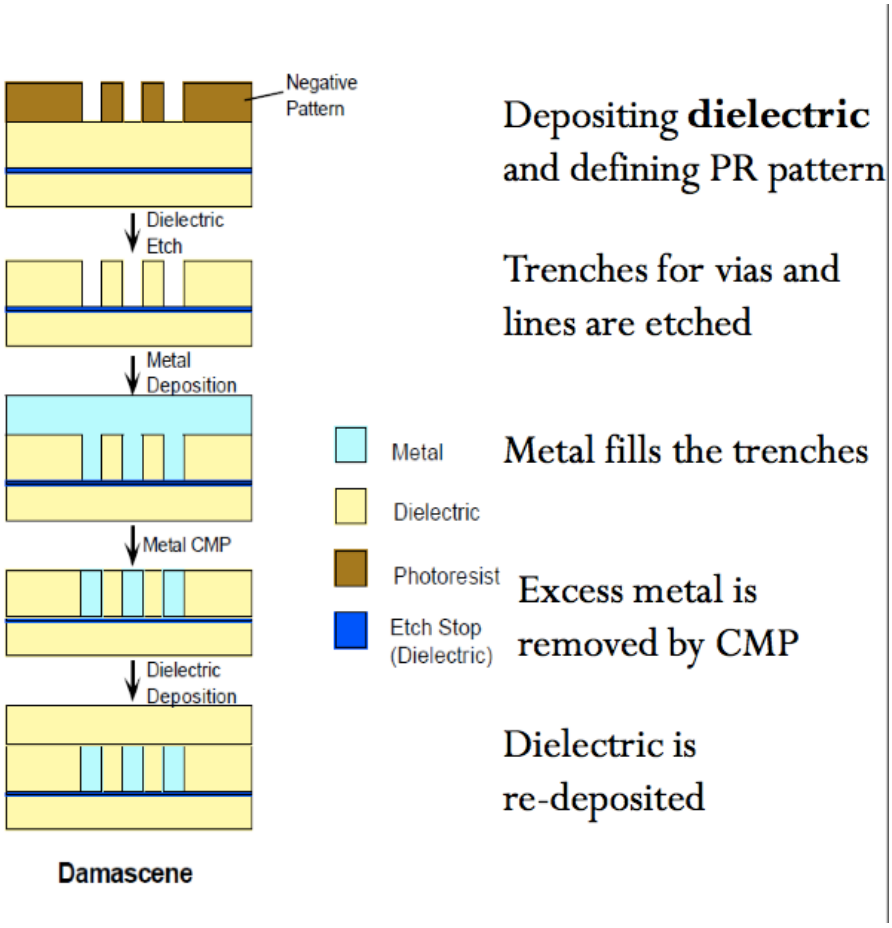


Figure 1. Damascene Process [8]

1.1 The Case for Copper Plating

There are only 4 metals in the periodic table that have electrical resistance low enough for use in thin films in the semiconductor industry. These metals include aluminum, gold, silver, and copper. Aluminum's resistivity is much higher then the other three (aluminum – 2.69 µohm-cm, gold – 2.3 µohm-cm, copper – 1.67 µohm-cm, and silver – 1.60 µohm-cm).

Aluminum:

Aluminum has been used in the IC industry for many years and has many nice properties such as it forms a stable oxide, adheres very well, has almost no diffusion in most dielectrics and other materials used in IC processing, and is easily etched with dry techniques. Unfortunately the

high speed requirements of modern circuits dictates that a metal must have as low a resistance as possible. Since aluminum has the highest resistance of the 4 possible metals it is not the most desirable candidate.

Silver:

Silver would seem to be the best choice, if resistivity is the only consideration, but it suffers from very high diffusivity in other materials such as dielectrics [9]. One of the reasons for this is that it diffuses as an ion and not as an atom, this small ion causes it to be able to quickly diffuse in the relatively large spaces of a dielectric, especially under an electric field. Silver also oxidizes or tarnishes very quickly. These reasons and the fact that Cu is not much different in resistivity leads one to select Cu as the better choice.

Gold:

Gold has many nice properties, such as the fact that it does not oxidize and is very stable, but it does have the same problems with diffusivity as silver [9] (i.e. a very fast diffuser in most materials used in processing). It also has a much higher resistivity and is very expensive. So, it would seem Cu is still the best choice of the 4 metals.

Copper:

Copper metallization has been used in the PCB industry and on MCM substrates for many years and has a mature plating technology for deposition and delineation [10]. This fact along with the fact that the resistivity is only slightly different than that of silver makes copper the best choice for a metal. Although copper still suffers from a high diffusivity in dielectrics [11], it is not as high as silver or gold. So, depending on the dielectric used, there may have to be a barrier metal added to insure a slowing down of the copper diffusion [11], thus increasing the reliability of the structure.

Chapter 2: Copper Interconnect and Via Plating Process

2.1 Platable Region Preparation

There are many uses for copper plating in the IC industry including conductor plating [12], via plating [13], pillar plating [14], through silicon via (TSV) plating [14], and [15], damascene and dual damascene plating. This work will concentrate on the chemistry needed to successfully perform via filling processes such as TSV and Damascene. The organic components are also used in the other plating processes and this new analysis procedure will help determine the organic additives in these plating baths as well. The via processes will be used as examples since the need for the organic components is more obvious and easier to understand with the requirement for a bottom up fill process.

2.2 Damascene Process

The damascene process is shown in figure 1: it shows an additive process for copper as opposed to the typical subtractive process historically used for aluminum. This additive process begins with etching of the underlying dielectric to delineate the regions that will eventually become the copper conductors. After the dielectric etching process (normally a dry etch process), a barrier layer is deposited that provide a barrier between the underlying dielectric and the copper conductors. Since copper is a fast diffuser, in most dielectrics, a thin layer of Ta, TaN, TiN, or TiW is deposited to completely cover the dielectric surface and etched sidewalls, from the previous step. A layer of "seed" copper is next deposited to act as a base layer onto which additional copper will be plated. This seed layer will have to be thick enough to provide a low resistance path for electroplating, yet thin enough to not close off the small hole etched in the

dielectric. Typical thicknesses are 100 nm to 400 nm. This copper layer is sputter deposited to provide optimal step coverage, making sure the copper layer is continuous throughout the etched hole. After the seed layer is deposited copper is electroplated to fill the holes. This plating process must allow the etched hole to be completely filled, which can only be accomplished if the hole is filled from the bottom, otherwise the copper on the surface and at the top edges of the hole will tend to "pinch" off the hole, not allowing additional copper to plate in the center of the hole, leaving a void filled with the copper solution. This could be a reliability problem due to trapped chemistry and a thinning of via metal [17], as well as a reduction in via resistance. A typical via void is shown in figure 2.

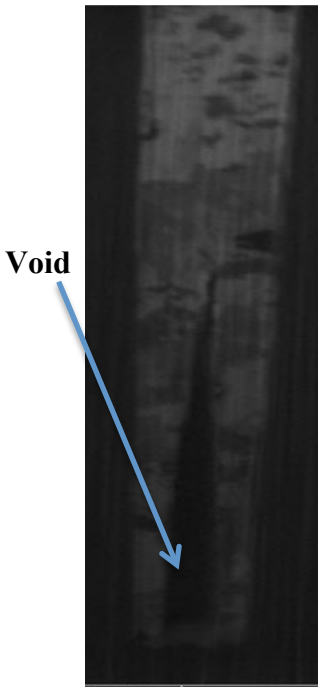


Figure 2. Via void in copper filled via

The next step is a Chemical Mechanical Polish (CMP) [18] that removes excess copper from the surface as well as planarizes the surface delineating the copper conductors. CMP uses the combination of physical and chemical removal processes to provide a planar surface that

stops on the barrier layer and has a minimal effect on the copper conductor surface. As mentioned earlier it is the filling of the etched holes that must be carefully controlled to make sure the plating process proceeds with a bottom up fill: in other words the plating process should plate faster at the bottom of the hole compared to the surface of the substrate. There should also be a reduced rate of plating at the sharp corner at the top of the hole. These steps will allow the hole to fully fill, without voiding.

2.3 Through Silicon Via (TSV) Process

Recent advances in logic switching times and frequency along with reduced circuit voltages have created the need to reduce the attenuation of signals for a given length across an integrated circuit [19]. Some small reductions in attenuation have been realized with lower K dielectrics, but there is still a need to reduce the length of the lines to make significant reductions. The only way to reduce the line length, while maintaining the processing power and complexity, is to break the sections of the processor into separate thin chips and stack them into a 3-D array **[20]–[23]**. This concept is shown in figure 3, and is the only method presently capable of making a significant reduction in the line delays and signal attenuation.

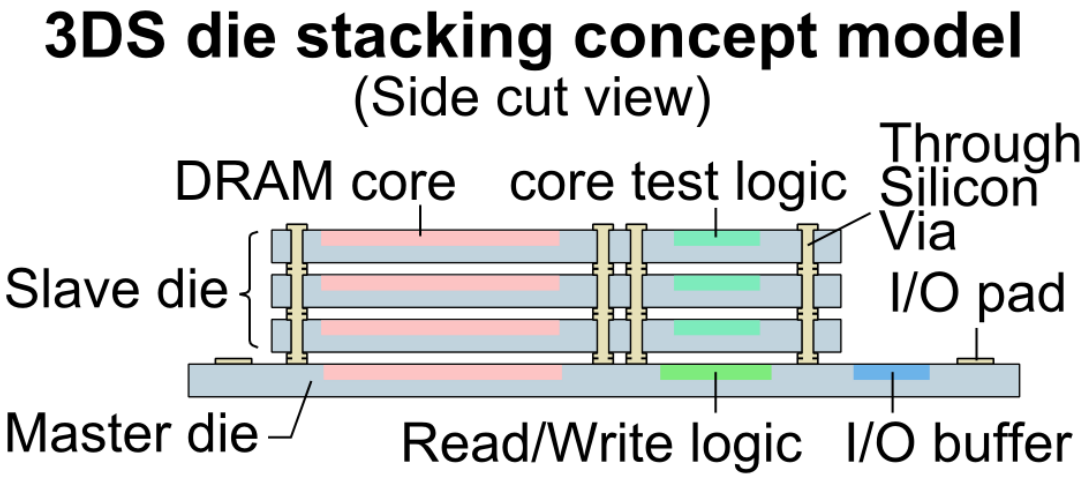


Figure 3. 3-D packaging cross-section showing through silicon vias (TSV) [23]

Through Silicon Vias have been developed to provide interconnection between the stacked die. These vias are filled with copper to provide a low resistance connection and are typically 2 – 50 µms in diameter. The via depth is typically 50 – 150 µms depending on the final thickness of the stacked die. The process for fabricating TSVs is shown in figure 4 and described below.

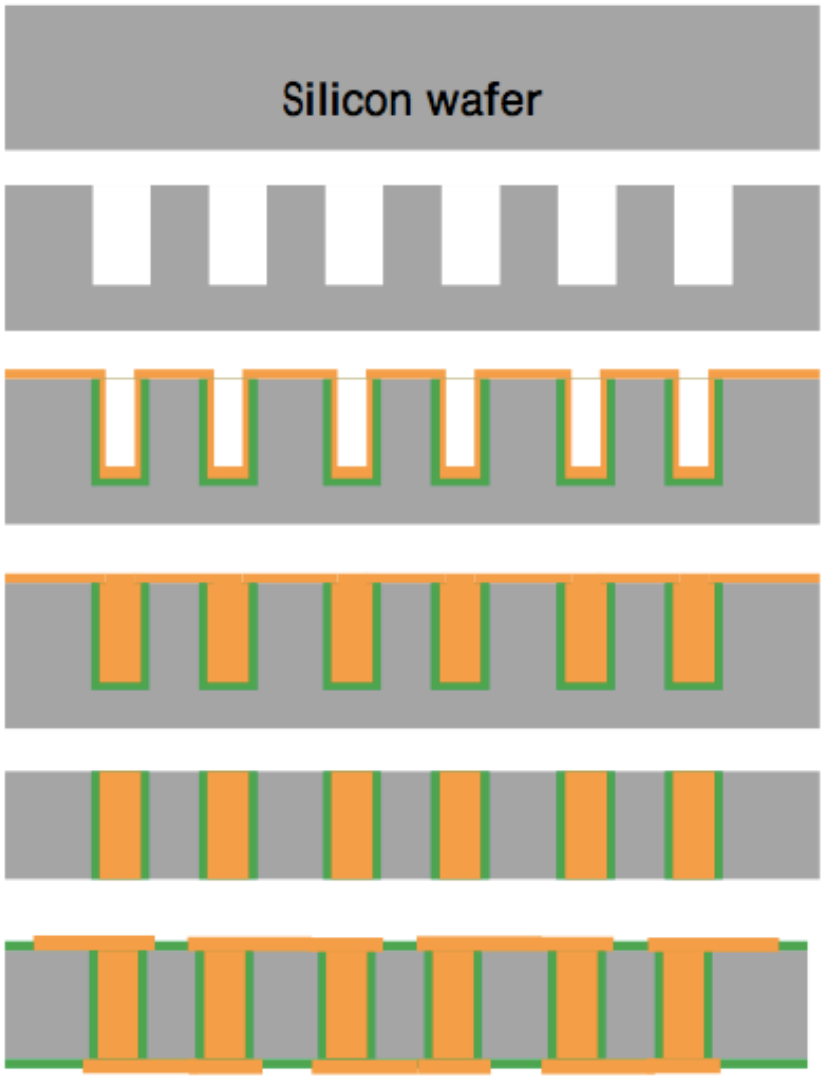


Figure 4. TSV process flow

A silicon wafer is patterned to define the TSV openings and then etched in a deep reactive ion etcher to form high aspect ratio holes. The next step is to strip photoresist and any polymer in the holes. Then the surface of the hole is passivated by thermal oxidation or a deposited oxide layer. The next step is to deposit a barrier layer, typically TiN or TaN, and a plating seed layer, typically copper or ruthenium (in the case of atomic layer deposition (ALD)) [24] . The wafer is then placed in a copper plating bath to fill the vias with copper, as with the

damascene hole fill this process must proceed with a bottom up fill to make sure there are no voids in the copper. Figure 5 shows a wafer after TSV copper plating at Auburn University: in this case the via diameter is 20 μms and the depth is 160 μms.

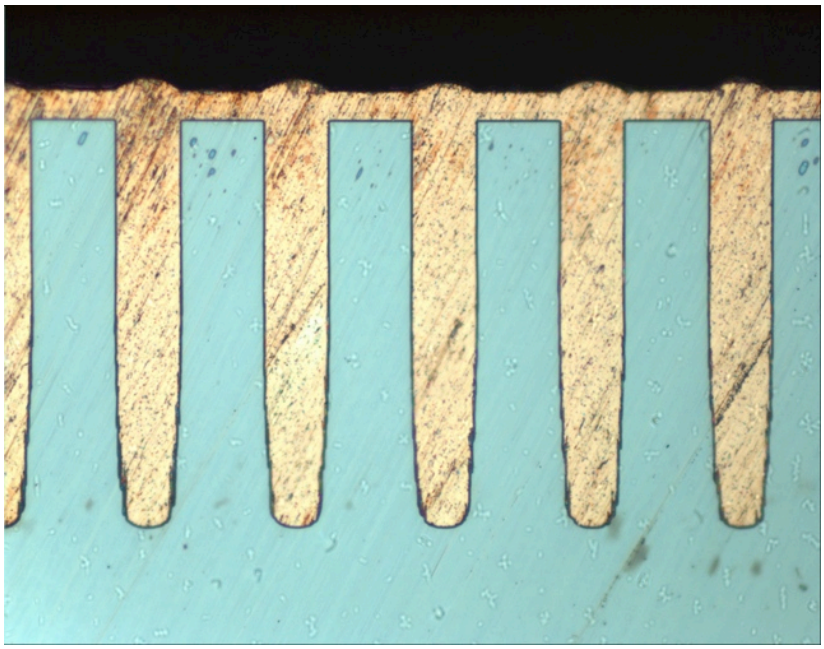


Figure 5. TSV cross-section after via fill process

As shown in figure 5, there is significant over-plating at the top of the holes and on the surface of the wafer that must be removed to isolate the vias from one another. To do this a chemical mechanical polish (CMP) system is used to remove the excess copper [25] from the surface using a combination of chemical and mechanical means. This allows some selectivity to allow the polishing to slow down as the copper is removed and the barrier layer is revealed. The selectivity allows the polishing process to completely remove the copper in shallow areas while the polishing in the exposed higher areas is virtually stopped. A photo of the revealed TSVs,

after a CMP process (at Auburn University), is shown in figure 6: notice the insulating oxide region (black area surrounding the copper).

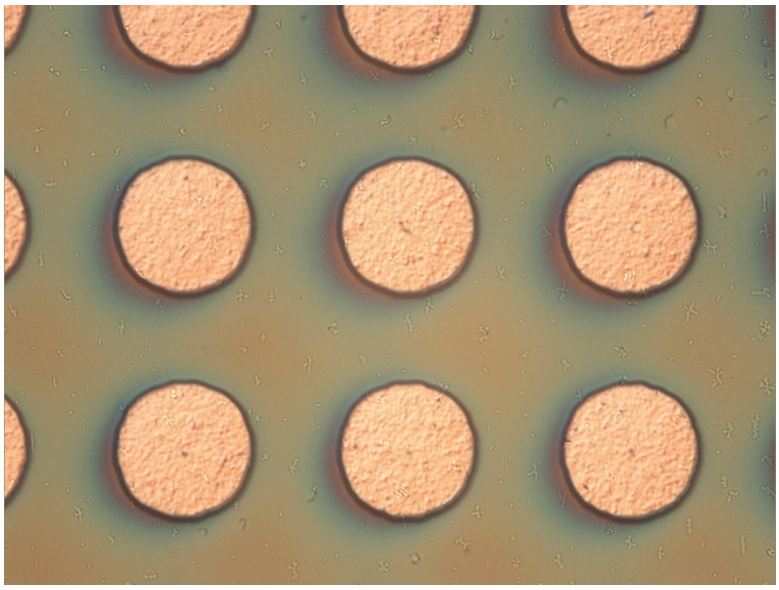


Figure 6. Revealed copper vias after CMP process

Next, the wafer is background to thin the wafer to the final thickness. To do this the wafer is first temporarily attached to a second substrate and the back of the wafer is ground and polished to thin the wafer to a predetermined thickness, exposing the bottom of the TSVs. Finally, a dielectric layer is deposited on both the front and the back surfaces and patterned to form contacts to the vias and the next metal layer to be applied.

2.4 Copper Plating Bath Makeup

A copper plating bath, formulated for damascene or TSVs, consists of various chemicals [5] , both non-organic and organic, to control different aspects of the plating process. The primary component is copper sulfate that contains the copper source. Sulfuric acid is added to

provide an ion source for electrical conductance. The next component is hydrochloric acid that provides a source of chlorine required to activate the organic components. There are typically three organic additives that control the plating rates and other properties at various regions of the plated substrate surface. The three organics are suppressor, accelerator, and leveler. They are normally added in small quantities, a few ml/l of solution, but can have a huge effect on the plating rates. The effects will be discussed in detail in chapter 3.

The concentration of the various components is different for the application, as an example when plating surface conductors and shallow regions, the copper sulfate concentration is relatively low, while when there is a deep or a high aspect ratio feature the copper sulfate concentration is relatively high to insure there is sufficient concentration at the bottom of the deep feature. If you think about plating in a deep feature you are dividing the concentration at the surface of the hole by the surface area interior to the hole, so, as the plating progresses, down the hole, copper is depleted at the surface and copper is plated on the upper sidewalls.

Copper plating solutions are available from many different vendors including Atotech, Enthone, Rhom-Haas, and Moses Lakes, and is sold as a virgin makeup solution (VMS), that contains vendor specified proportions of copper sulfate, sulfuric acid, and hydrochloric acid. These vendors also sale the organic components and give a starting point for their use. Most of the intellectual property (IP) is in the organic additives, which are different for each company.

2.5 TSV Copper Plating Process

2.5.1 Pre-Wetting

The copper plating process begins once the vias have been etched, passivated, and a seed layer has been deposited. The next step is to pre-wet the via hole, typically with water, to ensure

air bubbles do not keep the plating chemistry from entering the via, causing voids. The most promising [26] pre-wetting process begins with placing the TSV wafer into a vacuum chamber, then the air is removed and degassed water is added to the chamber. This process will still have trapped air in the holes, but the composition of the trapped bubble is mostly water vapor (assuming a vacuum level of at least 3 mtorr was achieved prior to adding water). As the vacuum chamber is brought back to atmospheric pressure the trapped bubble is reduced in size according to the ideal gas law PV=nRT (i.e. as the pressure decreases the volume increases, and as the pressure increases the volume decreases). The bubble is still mostly water vapor that will eventually be re-absorbed into the surrounding water further reducing the size of the bubble to a tiny fraction of the original. This process is shown in figure 7.

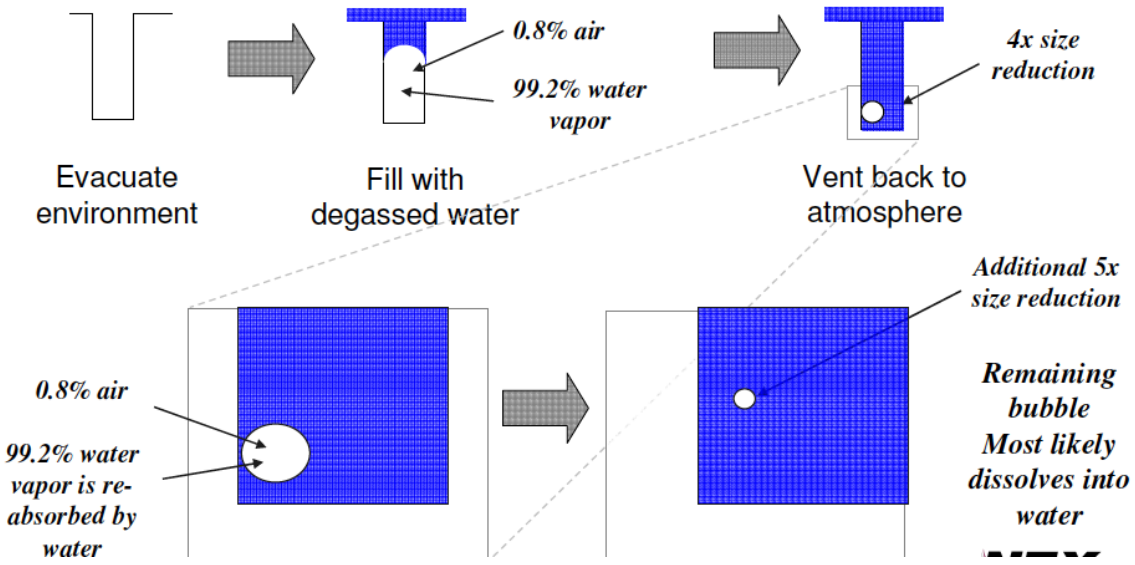


Figure 7. TSV pre-wetting process [26]

Alternative methods to pre-wet include placing a “wet” wafer into a vacuum chamber and pulling vacuum. This process only allows a partial removal of the trapped bubble, since this

bubble is mostly air and upon expanding, during the vacuum phase, it may separate and release some of it's contents into the surrounding water and eventually into the vacuum region. The rest of the bubble stays behind and is still located in the bottom of the hole.

To show the results of the two pre-wetting procedures, several samples were tested and cross-sectioned using the method shown in figure 8. A large array of TSVs are ground at an angle of approximately 3^{o}, exposing different vertical regions of the via with a lateral view. This allows you to see voids without performing time consuming and tedious vertical cross-sectioning.

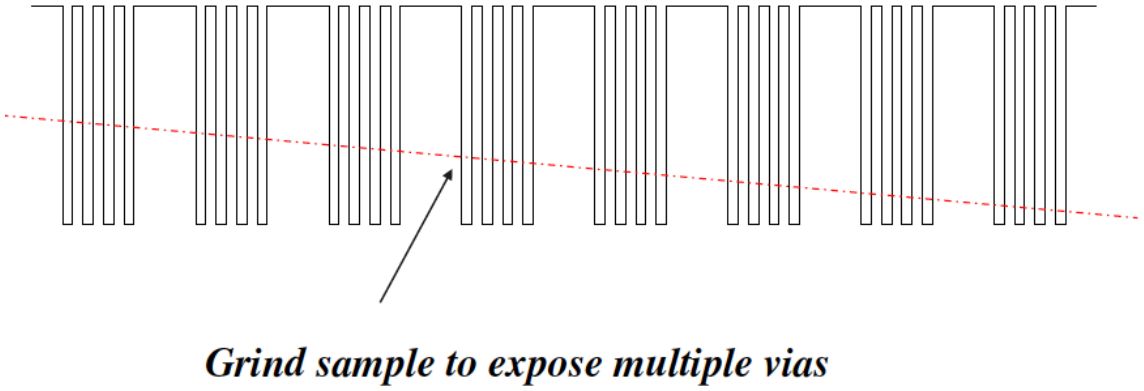
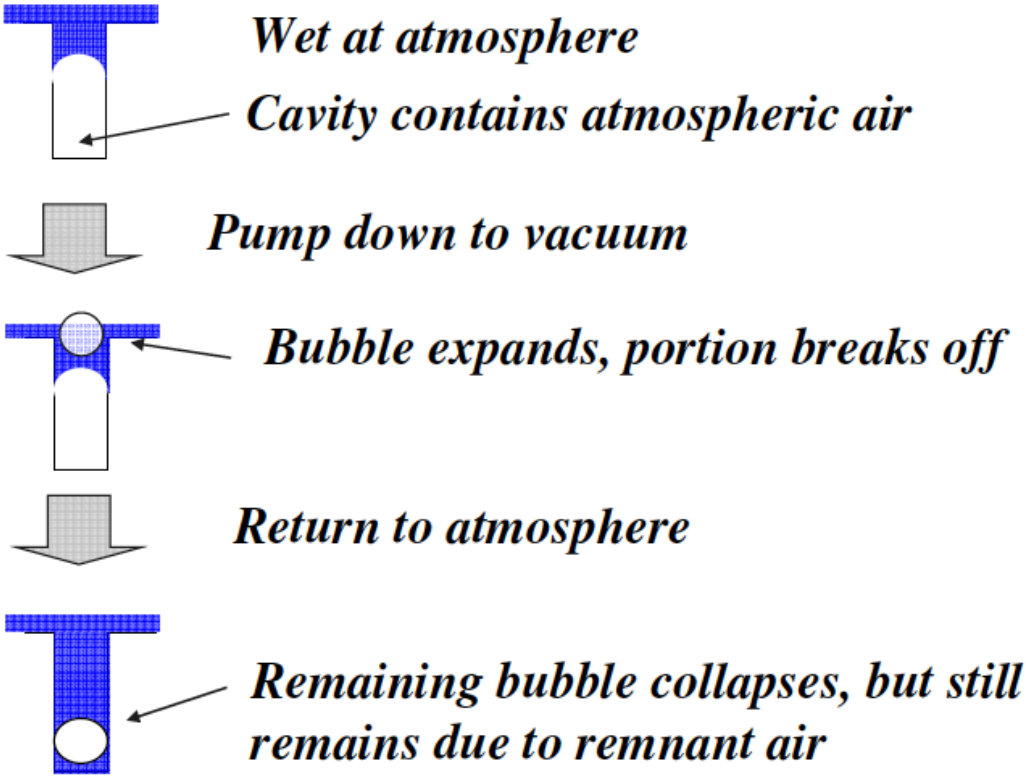


Figure 8. Method to view TSV cross-section

The results are shown in figures 9 & 10 showing the process and the 3^{o} cross-sectioning. Figure 9 shows the process of wetting before applying vacuum and the cross-section results showing voids near the middle (small dark areas in the center of the via). Figure 10 shows the vacuum first process and the results showing the absence of voids.

- Wetting before applying vacuum.



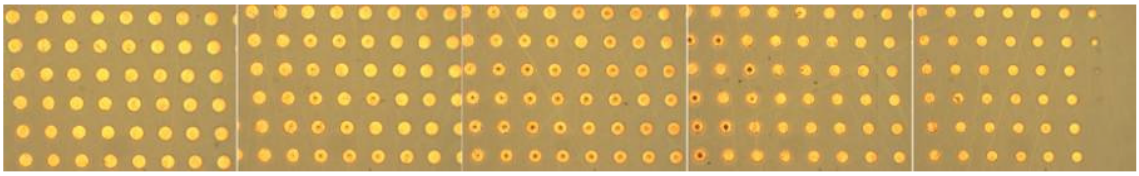


Figure 9. Wet first process and results

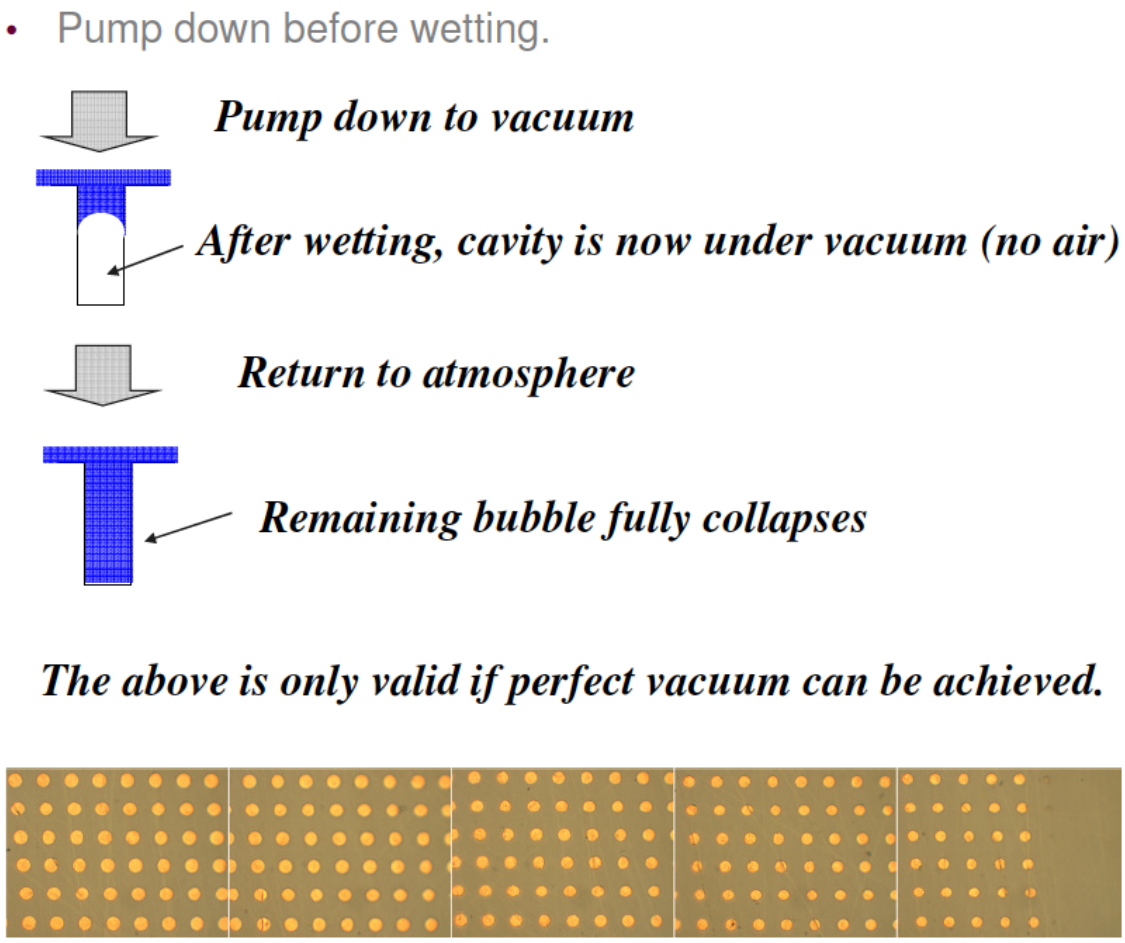


Figure 10. Vacuum first process and results

2.5.2 Plating system schematic

A typical electroplating system is composed of an anode and cathode that serve as the powered electrodes in the ion exchange system [27]. The anode is biased positively and acts as the ion source, while the negative biased cathode acts as the source for electrons. The item to be plated acts as the cathode of the circuit. The setup for a copper plating process is shown in figure 11. It shows a Cu anode that acts as a source for the Cu ions, these copper ions are associated with sulfate ions in the solution to form copper sulfate. At the cathode the Cu2+ ions are reduced, by cathode supplied electrons, to a neutral copper atom once it is deposited on the

surface of the cathode. A copper plating solution will contain both copper sulfate and sulfuric acid, with the copper sulfate acting as the copper source and sulfuric acid permitting electricity to flow. The copper from the copper sulfate is reduced at the cathode, the Cu from the anode will be eroded and maintain the copper sulfate concentration in the plating bath. Therefore, a copper anode is required otherwise the copper sulfate concentration will be reduced with plating and would have to be continuously added. As mentioned previously a relatively low amount of copper sulfate is used for shallow plating, while a larger concentration is used for high aspect ratio plating – such as TSVs.

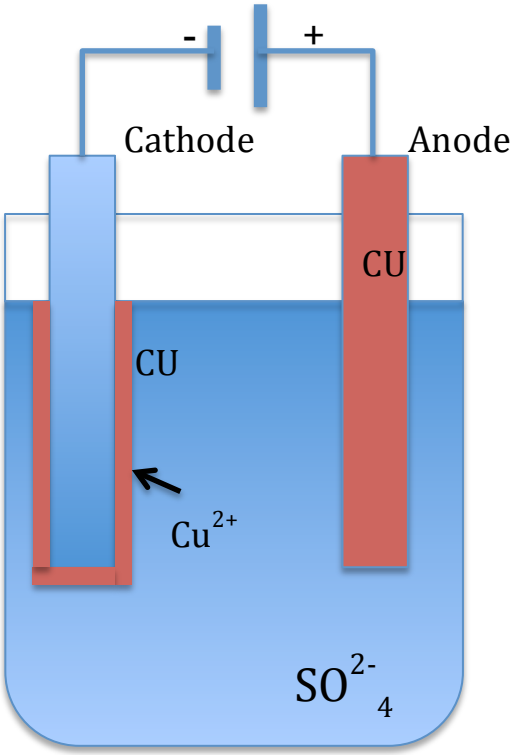


Figure 11. Copper plating bath configuration

2.5.3 DC vs. Pulse Plating

DC and pulse plating [28] are two techniques commonly used in the plating industry for various reasons. Each has advantages and disadvantages. A quick review of pulse plating literature finds that it provides advantages such as grain structure control, leveling of the plating surface, and increased plating in deep recesses. The pulse plating process uses a pulsing power supply that either turns the plating on and off or on to reverse plating in periodic pulses. The process of turning the plating off periodically allows the ion distribution to redistribute, through a diffusion process, and help even out the plating surface. The process of periodically reversing the plating process allows ions to be de-plated replenishing the ion concentration in hard to reach areas, evening out the plating in these regions. It also heavily plates organic compounds in sharp areas, where fields are higher, thereby, suppressing the plating in these areas during the subsequent forward plating pulse. The advantages of pulse plating mirror the advantages of adding organics to the plating bath and, if done properly, reduces the requirement for these additives. The disadvantages of pulse plating is that it is quite complicated to get the pulse timing and amplitudes set for the particular plated surface. The pulse/amplitude requirements can change during the plating process. The cost of the equipment is much higher due to the complexity of the plating power supplies.

DC plating uses a constant current during the plating process. Most modern TSV plating companies use DC plating exclusively due to the simplicity and cost of the plating equipment and process. This causes them to put more technology into controlling organic additives since this is the alternative to providing the same desirable functions as gained with pulse plating. So, all modern production plating systems will include two indispensible items, these are a DC power supply and an additive analysis and dosing system.

2.5.4 Cathode Boundary Layer Formation

During the plating process a boundary layer is formed next to the cathode as shown in figure 12. This region is a result of the reaction of the copper ions with the cathode surface causing a concentration gradient from the bulk concentration to a much smaller concentration where the copper ions are consumed and neutralized at the cathode. This reduction in ion concentration causes a potential change from the bulk region to the grounded cathode as shown in figure 12.

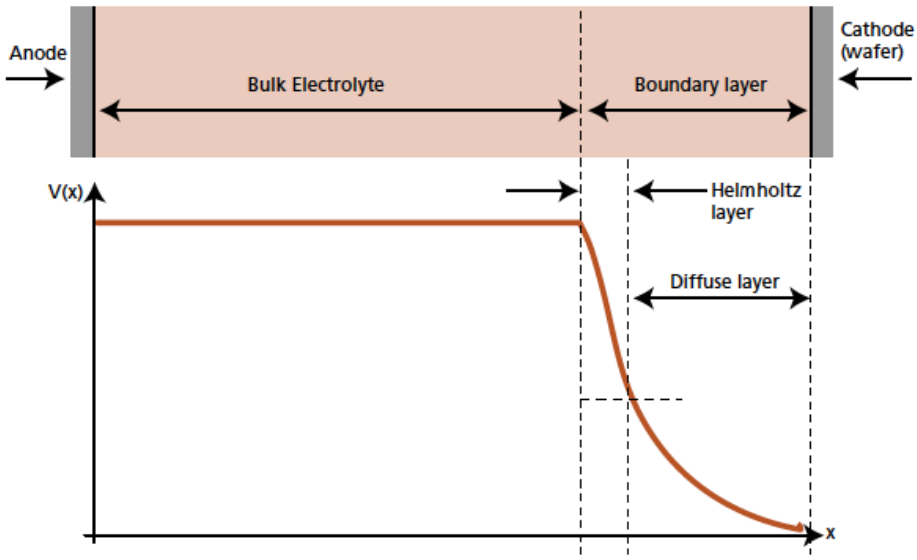


Figure 12. Cathode Boundary Layer and Potential

Since the bulk ion concentration is much higher than the boundary region concentration the potential is relatively constant in the bulk region and all the applied voltage is dropped across this boundary layer. While the positive copper ions drift across this potential difference, the concentration change also causes neutral additive organic molecules to diffuse from the bulk to

the cathode. It is this diffusion that controls the concentrations of the organic additives at different parts of the substrate surface. The different organic additive molecules have different diffusion rates and different functions that will be discussed in more detail in subsequent chapters. This boundary layer is of paramount importance in the plating process, especially in the plating of high aspect ratio vias. The boundary layer thickness and uniformity can be controlled and are conditions that must be understood and manipulated to be able to successfully and repeatedly plate copper across large diameter wafers, and to be able to fully fill a high aspect ratio via.

Chapter 3: Bottom-Up Fill Mechanism

In order to fabricate reliable TSVs a bottom up fill is required. If this bottom up fill is not performed there will be a high probability of a void forming in the center of the via. Shown in figure 13 are three types of filling: (a) sub-conformal, where the top edge of the via is plated more heavily than the sidewalls of the via hole. This causes a large void in the center if the via once plating is complete. The second type of fill is a conformal fill as shown in figure 13 (b), where the sidewalls and top edges of a via are the same thickness. This type of fill will cause a small or thin void to form as the plating is completed. The desired type of fill for a TSV via is bottom up filling as shown in figure 13 (c), where there is increased plating at the bottom of the via as compared to the top edges and upper surfaces of the via. This type of plating will ensure there is a void free, filled via.

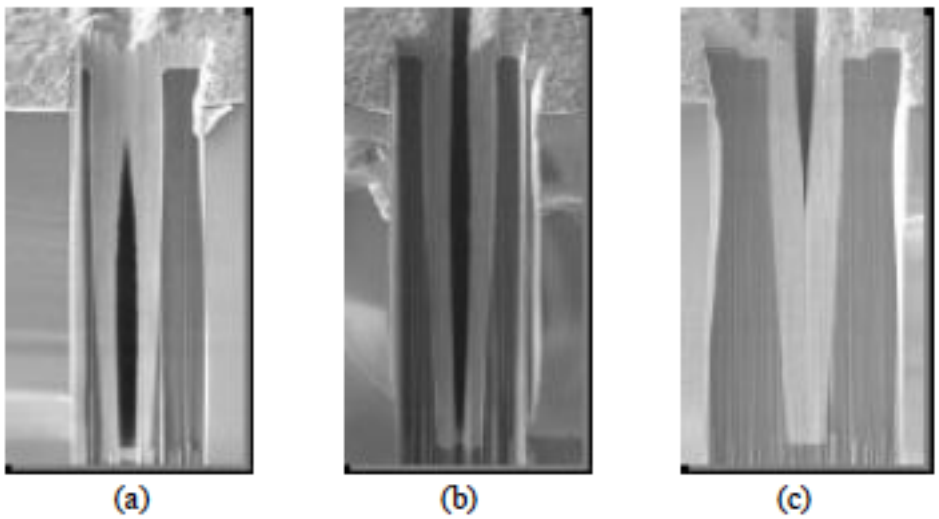


Figure 13. (a) sub-conformal, (b) conformal, (c) bottom up fill mechanisms

There has been a lot of work over the past 15 – 20 years to develop a process for repeatedly and reliably bottom-fill a TSV via. This work has provided processes that utilize several organics including Accelerators, Suppressors, and Levelers that control plating rates in various regions of the TSV substrate. The details of how these organic compounds affect the plating process is explained in the following sections.

3.1 Effect of Suppressors

A suppressor organic additive will inhibit or slow down the plating rate by adsorbing onto the plating surface and physically blocking the plating process. A typical suppressor is Polyethylene Glycol (PEG) [3], a simple polyether with multiple CH_2CH_2O segments. PEG has an affinity for Cu and will readily physically adsorb onto the surface in a matter of seconds. This process will only happen in the presence of chloride in sufficient concentration. Once adsorbed on the Cu surface the plating rate is reduced several orders of magnitude as compared to the plating rate in the absence of PEG and chloride. This suppression effect is dependent on the concentration of PEG and chloride. In both cases there is a maximum reached in the suppression as one component is maintained constant and the other component is increased. This is shown in figure 14 where the chloride concentration is held constant and plating current is measured vs. PEG concentration. It is shown that the plating rate is reduced until a minimum is reached after which further increases in PEG concentration will not reduce the plating rate. This is the point at which the surface is completely covered with PEG.

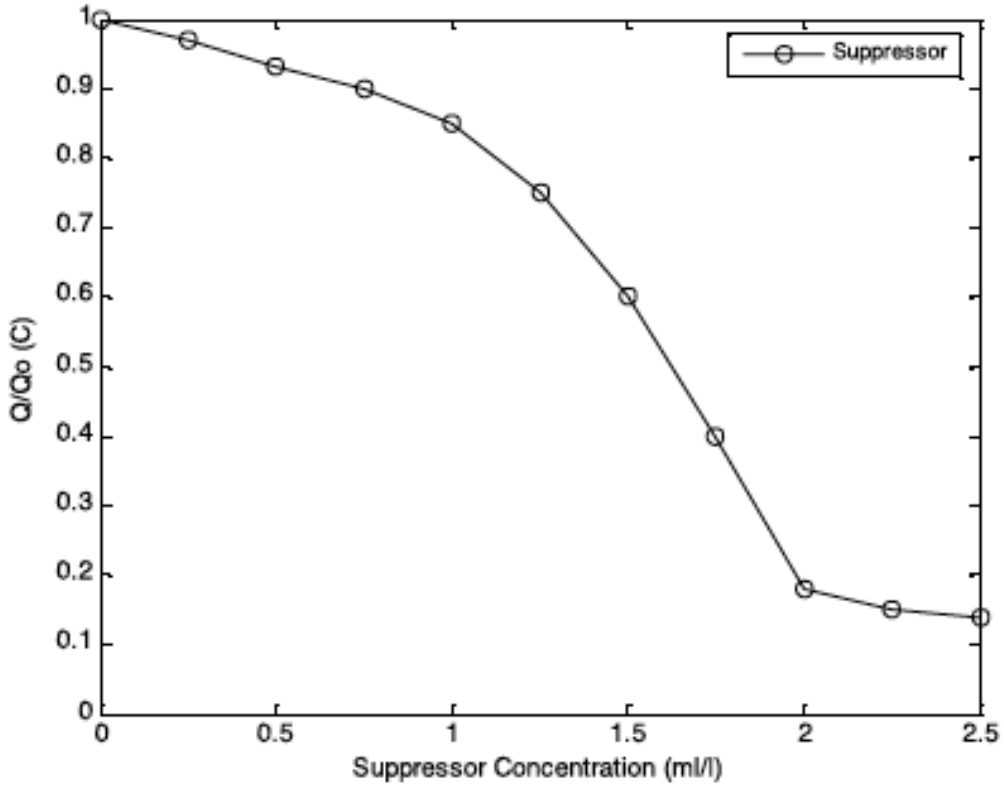


Figure 14. Plating rate vs. PEG concentration

3.2 Effect of Accelerators

In addition to PEG another organic additive utilized in modern TSV plating chemistries is $Na_2[SO_3(CH_2)_3S]_2$ (figure 15) or SPS [29] that acts as an accelerator or catalyst to increase plating in suppressed regions.

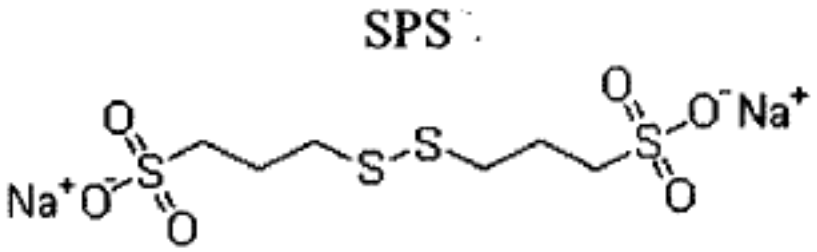


Figure 15. Chemical Structure of SPS

This increase in plating is only in the presence of a suppressor, so, the effect is to reduce the suppression caused by the PEG additive. In the absence of a suppressor, SPS has little effect on the plating rate. This process of anti-suppression is thought to occur by the replacement of the suppressor molecule by an SPS molecule and/or the blocking of a suppressor molecule from absorbing onto the plating surface. This disruption in the suppression is dependent on the plating potential or a more general metric – the over potential. As the over potential is raised the adsorption rate is increased, at a low over potential the replacement of the PEG molecules, and the return to the additive free plating rate, can take several minutes to complete. The relative concentrations of SPS and PEG can also determine the relative plating rates.

To achieve a bottom up fill, the plating rate at the bottom of the via needs to be higher than the other portions of the via. This plating rate enhancement at the bottom of the via is achieved by the relative fast diffusion rate of the SPS molecule vs. the PEG molecule. The SPS molecule has an atomic weight of 354.4 g/mo as compared to several thousand for PEG molecules, which leads to the SPS molecules having diffusion coefficients several orders of magnitude higher than PEG. So, as a pre-wetted wafer, containing TSV vias, is placed in a

plating bath and the surface is wetted with the plating chemistry the bath components will diffuse to all regions of the wafer surface including the bottom of the vias. The relatively fast diffusing SPS molecules will diffuse quickly to the bottom of the vias before the slow diffusing PEG molecules. Since the plating boundary layer is very thin at the surface, the slow moving, but fast adsorbing PEG molecules will saturate the surface of the wafer and suppress the plating on areas other than the bottom of the holes. The SPS molecules will attach to the bottom of the vias and inhibit the PEG molecules from attaching, thus giving additive free plating rates, several orders of magnitude, in these regions. This difference in plating rates is the definition of bottom up fill. The challenge is to balance the concentration of suppressor and accelerator molecules in the bulk to prevent the SPS molecules from completely replacing the PEG molecules on the surface, but do the opposite in the vias. As mentioned previously the removal of PEG molecules by SPS molecules is potential dependent, so along with SPS/PEG concentration control, the plating potential must be carefully controlled to achieve a bottom up fill. Once PEG molecules have been replaced by SPS molecules the plating mechanism goes back to fully conformal and the possibility of a void returns. This balance is very difficult to achieve during the relatively long plating times that are required for TSVs (several minutes to hours). For submicron damascene features, with plating times in several seconds, the process works very will as the plating time is less than the time PEG to SPS replacement time. For these longer plating times, a requirement would be either to significantly increase the SPS adsorption time or somehow keep the PEG from diffusing very deep into the vias. Neither one of these has been accomplished to date in a production environment. To achieve the bottom up plating in deep TSVs a third component has proven to be necessary that is very slow diffusing and is not displaced by SPS or PEG. This component is known as a Leveler and is discussed below.

3.3 Effect of Levelers

The third organic component is leveler [30], a polarized copper plating inhibiting molecule that is a relatively slow diffuser and is not removed by SPS molecules. It is consumed or broken down during the plating process. A nitrogen containing dye, safrananne azo dimethyl aniline (Janus Green B, JGB), figure 16, has been used commercially for many years as a leveler. In the PCB industry JGB is used to reduce the enhanced plating at sharp points or edges, it levels the plating by being a slightly polarized molecule that will preferentially deposit in higher electric field regions and inhibit plating. In TSV plating levelers will preferentially deposit on upper corners of the TSV hole and inhibit plating that would normally cause a pinch-off and voiding. JGB has also been used to reduce the "Accelerator Bump" caused by the fact that SPS is not consumed in the plating process and will tend to concentrate in the hole areas, causing a bump after the TSV is filled.

As mentioned earlier, the addition of a leveler is required for TSV bottom up filling and only needed as an "accelerator bump" reducer in damascene processing, due to the plating time required for the respective plating processes. It has been reported that TSV bottom up filling can be achieved using only leveler, due to the fact that it is consumed during the plating process and it is a relatively slow diffuser. If a good agitation, thin boundary layer, is maintained and the TSV is relatively deep, then it is easy to see why this might be possible. The concentration of leveler is reduced significantly in the hole region and if the concentration in the bulk is balanced properly then the supply of leveler, to make up for the amount consumed during plating, can be high at the surface and very low in the hole, leading to a bottom up fill. From conversations with researchers at Atotech [31], there continues to be a lot of work in this area to see if in fact a

single additive process can be developed. There are a lot of proprietary levelers being introduced recently that shows this is an active area of research.

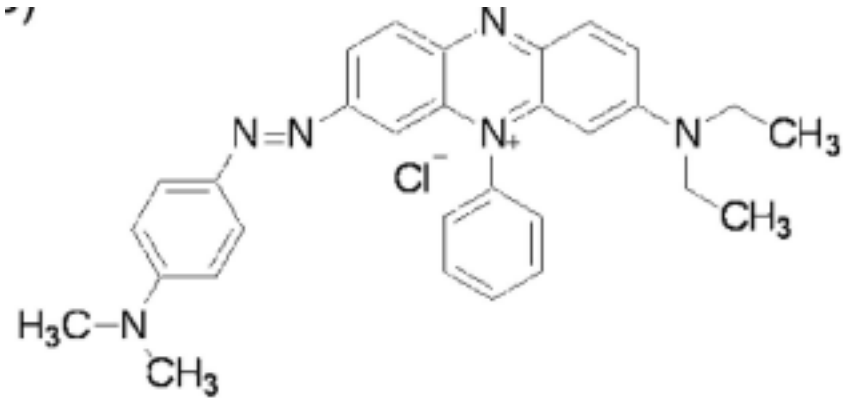


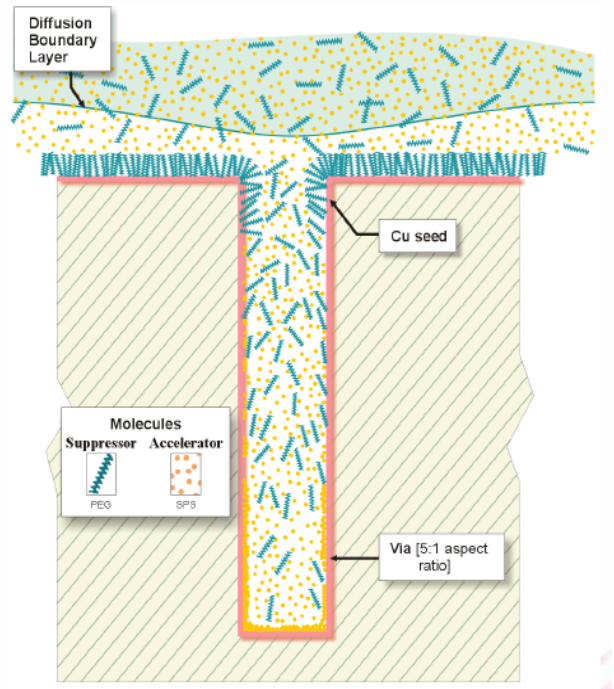
Figure 16. Chemical structure of Janus Green B (JGB) leveler

3.4 Combined effect of organic components

In most literature, reporting TSV and Damascene processing, all three of the organic components mentioned previously are added to a copper plating bath. The combination of these three additives has been shown to provide repeatable and reliable bottom up plating. The process is shown in the following drawings (figure 17 a-h) [32]. The images show the ideal process sequence beginning with a pre-wetted via that is placed in a plating solution containing suppressor and accelerator organic additives. The fast diffusing accelerator molecules quickly cover the surface of the wafer, but they do not react right away, as the adsorption process is slow compared to the PEG adsorption. Since the boundary layer is much thinner at the surface of the wafer, PEG molecules reach the Cu surface quickly and adsorb right away. The slow diffusing PEG molecules will adsorb onto the inner surfaces of the TSV, thus depleting the concentration

deeper into the hole. This low PEG concentration and the extra time it takes for it to diffuse into the deep hole will allow the accelerator time to adsorb and inhibit any PEG that might eventually reach the via bottom from adsorbing. As discussed before the adsorbed PEG will eventually be replaced with SPS given enough time. Also, notice SPS tends to pile up at the bottom of the via, as it is not consumed during the plating process. Upon completion of the plating process the high concentration of accelerator at the moving portion of the bottom-up fill will cause this region to plate faster and eventually cause a "bump" or high spot [32] over the hole. This bump can will cause problems in subsequent CMP processing. To inhibit this bump formation a leveler is typically added to the chemistry to inhibit plating in regions closer to the boundary layer. Leveler molecules are attracted to the higher regions and will readily replace SPS molecules.

A cross-section of an actual partially plated TSV is shown in figure 18. A bottom up fill is shown with very minimal plating in the other regions of the via.



17 (a). Initial step of TSV Cu plating, showing seed layer and relative additive concentrations

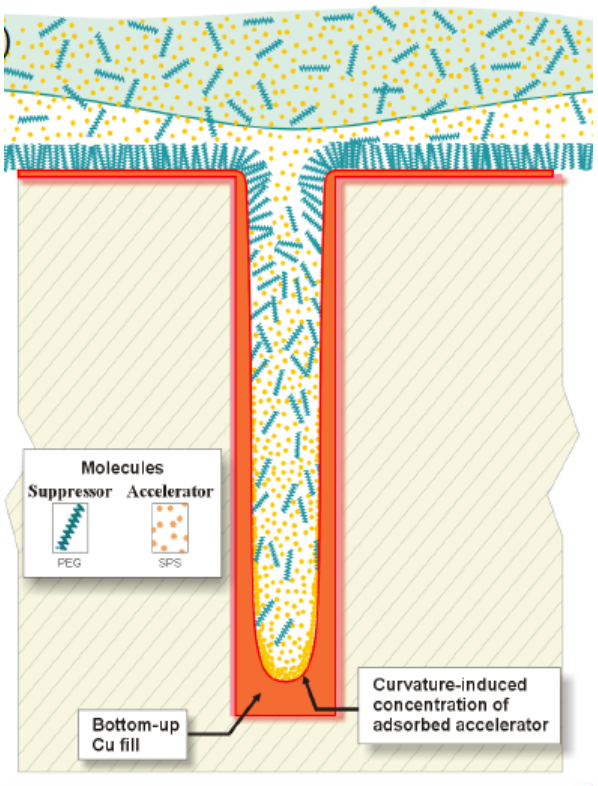


Figure 17 (b). Next step in TSV copper plating showing accelerator concentration at the bottom of the TSV

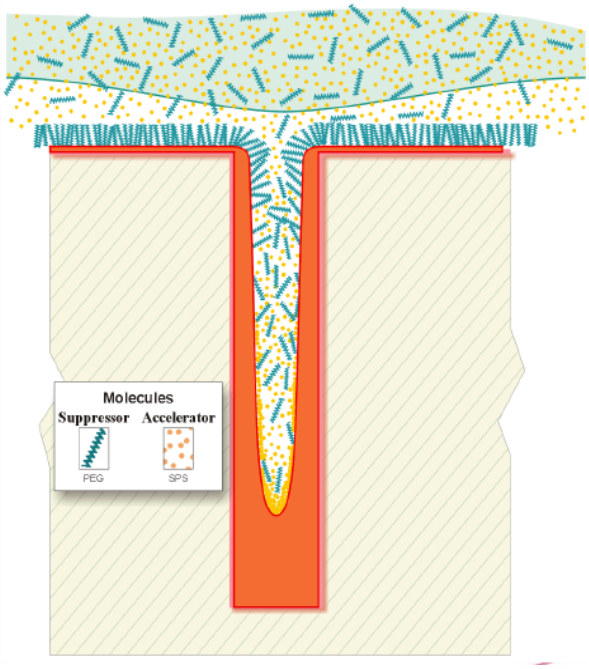


Figure 17 (c). Bottom up process progression

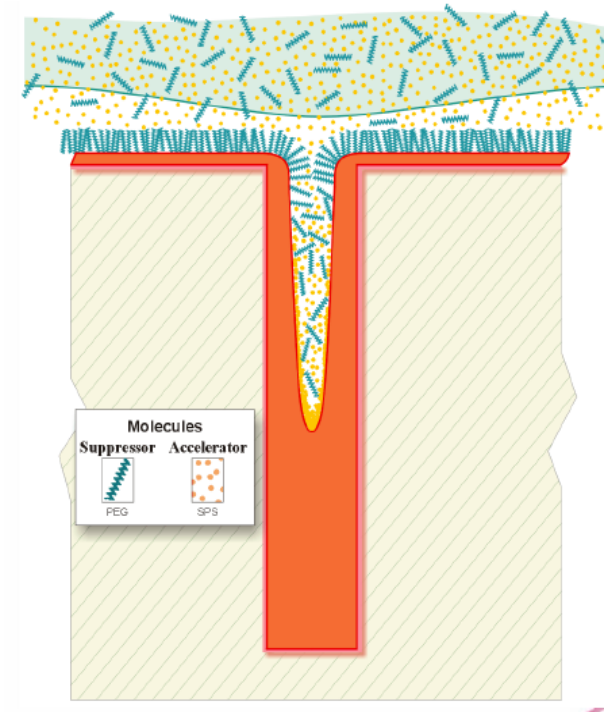


Figure 17 (d). Bottom up process showing lack pinch-off

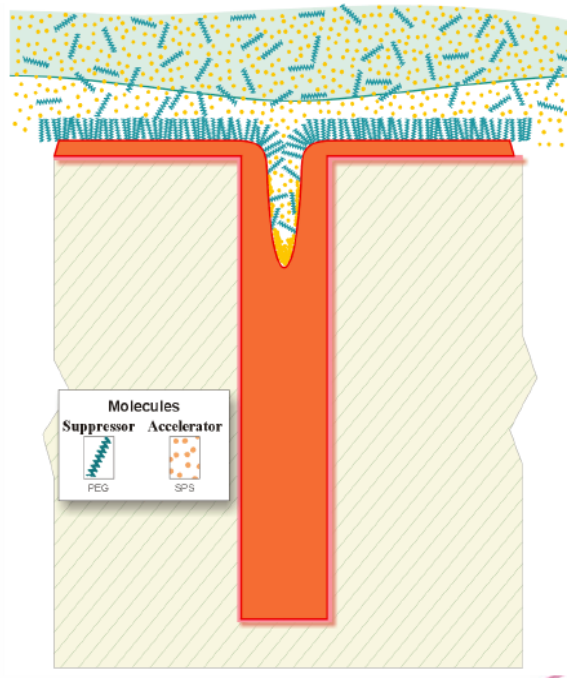


Figure 17 (e). Near end of TSV bottom-up fill process showing high concentration of accelerator

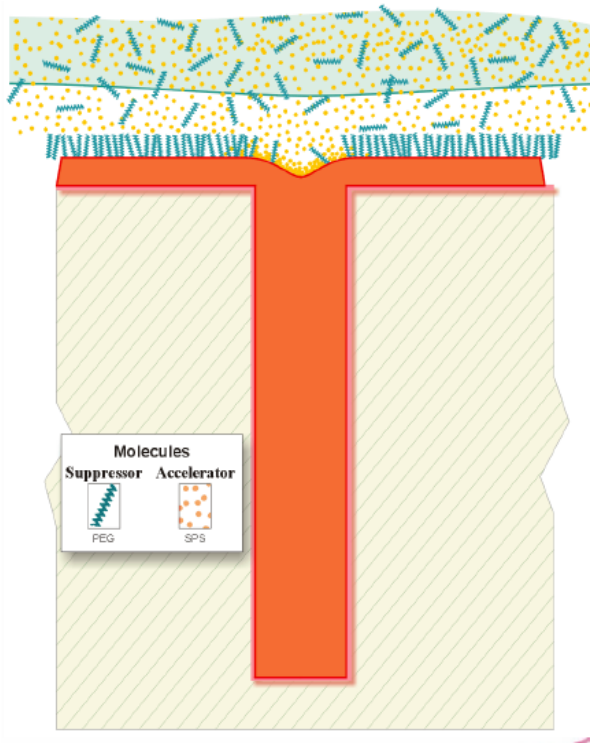


Figure 17 (f). End of TSV plating process showing a high concentration of accelerator in the via region, ideal stopping point

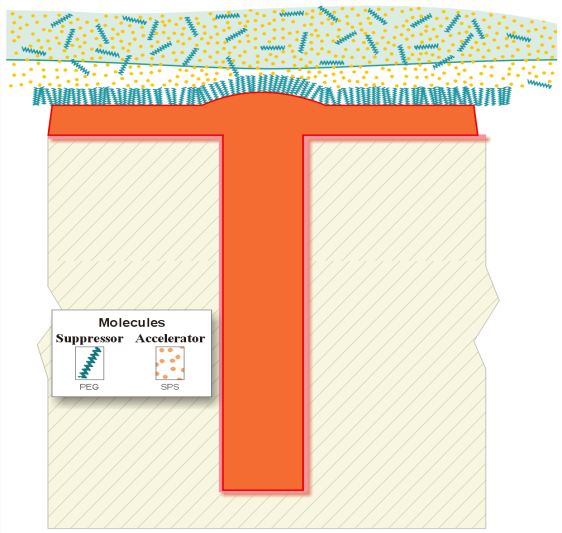


Figure 17 (g). Completed TSV showing “accelerator bump”

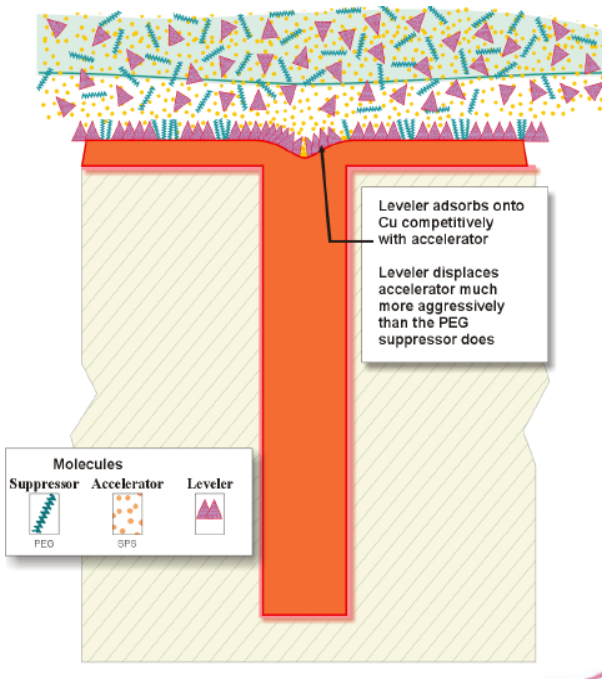


Figure 17 (h).Plated TSV with leveler, showing reduction of "accelerator bump"

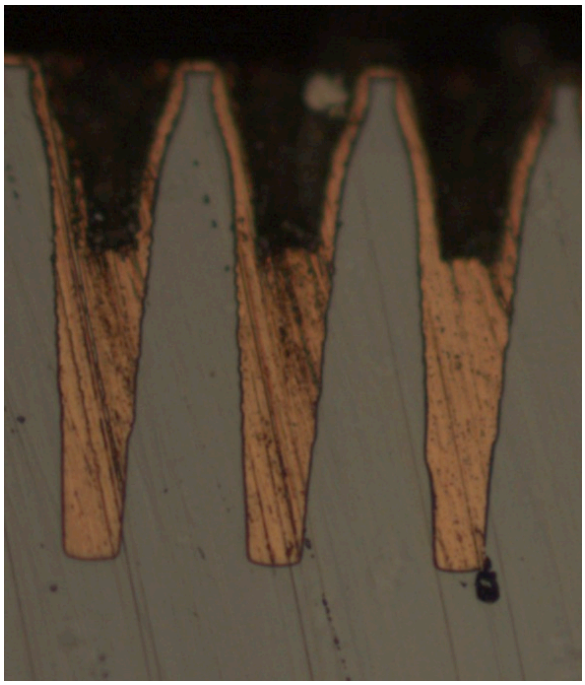


Figure 18. Actual plated TSV showing bottom up fill

Chapter 4: Present Industrial Organic Additive Analysis Methods

In the copper electroplating process organic additives must be maintained within a very tight range in order to insure consistent results from wafer to wafer. Typical copper plating baths contain organic additives to attain a bright, smooth, and level plating surface. These additives include:

- **Accelerators (brightners)** - catalyses and speeds-up plating especially in trenches and vias
- **Suppressors (carriers)** - suppresses plating
- **Levelers** - polar organic molecules that are attracted to higher potential regions, such as sharp points or corners, suppressing plating in these regions.

The combination of these organic additives will allow super-filling of a deep via while suppressing plating on the surface and the top edges of the hole, insuring the hole is completely filled, with copper and without voids.

The organic additives are added in minute quantities and must be maintained within very tight ranges. There are two methods used today to insure consistent plating in production systems. The first is Cyclic Voltrametric Stripping (CVS) [33], [34] , and the other is Bleed & Feed [35] . Often times the two techniques are combined in order to reduce the level of breakdown products, which can lead to a reduction in the ability to control the plating bath. The Bleed & Feed method is a technique whereby a portion (typically around 5%) of the solution is bled from the bath and then a fresh amount, equal to the bled amount, is added at the same rate as the bleed to attain a steady-state solution providing a consistent bath. The CVS method is a standard chemical analysis method that is used to plate a small amount of copper onto a spinning disk electrode and then stripping it from the electrode. The area under the stripping curve is

integrated and this charge is recorded. By performing certain computational methods this charge can be used to determine the amounts of each type of additive.

4.1 Commonly Used Additive Control Methods

Present methods of chemical analysis use CVS to determine the stripping area (Plating charge from integrating the area under the positive portion of the curve), shown in Fig. 19, for plating bath samples. The bath sample stripping data is analyzed using the following techniques to determine the amount of accelerator, suppressor, and leveler present in the plating bath.

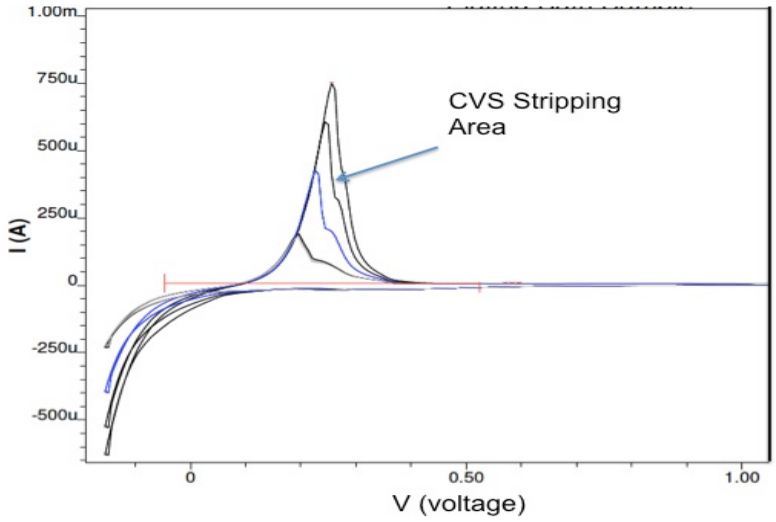


Figure 19. CVS Stripping Curve

4.1.1 Accelerator Determination Using MLAT

To determine the amount of accelerator in a plating sample, a Modified Linear Approximation (MLAT) technique is used [36]. This technique begins with a sample of Virgin Makeup Solution (VMS), which is a solution without any added organics (i.e. zero accelerator, suppressor, leveler). A sufficient amount of 60/40 suppressor/leveler is added to the solution to insure it is completely saturated - see Fig. 20.

The saturated solution will insure that any additional suppressor or leveler will not have an effect on the stripping charge. This minimum charge is recorded and used as the intercept baseline.

Next, a small amount of the plating bath, to be analysed, is added to the saturated solution to give a starting point for the analysis. Next, known amounts of accelerator are added to provide additional points from which a straight line can be drawn.

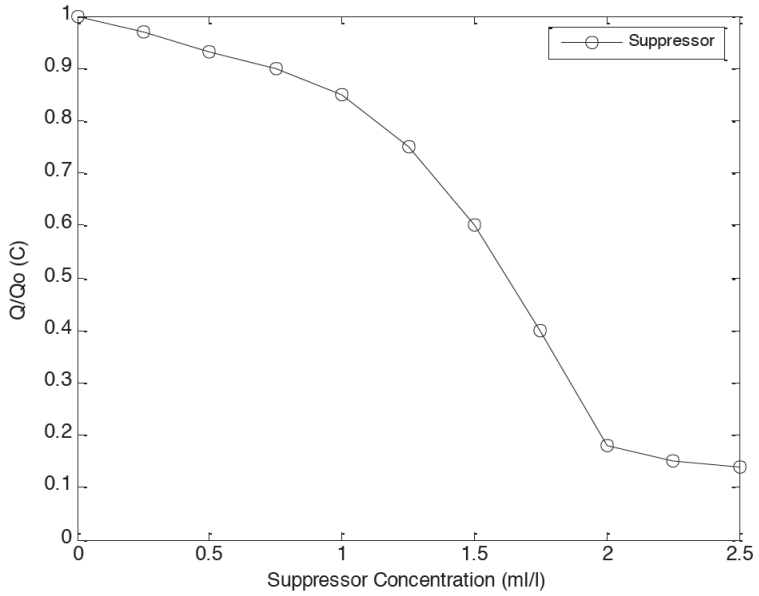


Figure 20. Normalized Stripping Charge vs. Suppressor Concentration

It is known that at small accelerator levels the stripping charge vs. accelerator addition is linear [36]. From the straight line drawn through these points the beginning concentration can be determined from the point where the line crosses the intercept baseline to the origin. This value corresponds to the amount in the first addition (from the actual plating bath) and through some simple calculations can give the number of ml/l of accelerator in the actual plating bath. A typical plot is shown in figure 21.

This technique requires mixing a VMS solution and running at least 4 separate CVS analysis runs (Qint, Qbath, Q1, & Q2). It also requires maintaining test solutions of Accelerator, Suppressor, and Leveler.

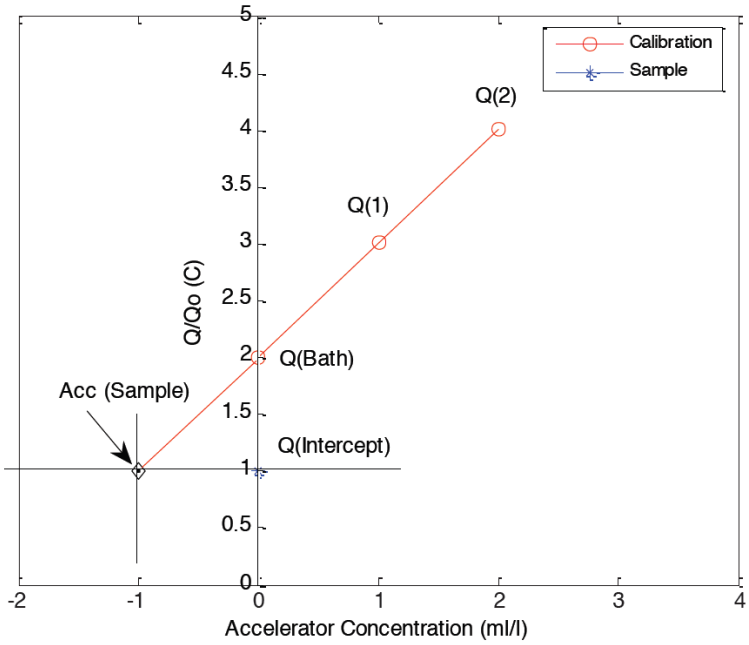


Figure 21. Typical MLAT plot for accelerator determination

4.1.2 Suppressor Determination Using Dilution-Titration

The determination of the suppressor concentration uses a CVS Dilution-Titration (DT) technique [37] . This method requires a calibration curve that must be generated at least once per week. The calibration curve is generated by starting with a VMS solution and adding suppressor additions until the charge per CVS run has dropped to 50% of the VMS solution charge (in other words the amount of plating has been suppressed by 50%). Once the calibration curve is established the VMS solution is replaced with the actual plating bath sample and suppressor is then added until the 50% point is reached - the reduction in the amount of suppressor required to

reach the 50% point is equal to the amount of suppressor in the sample originally. A typical calibration and determination curve is shown in figure 22.

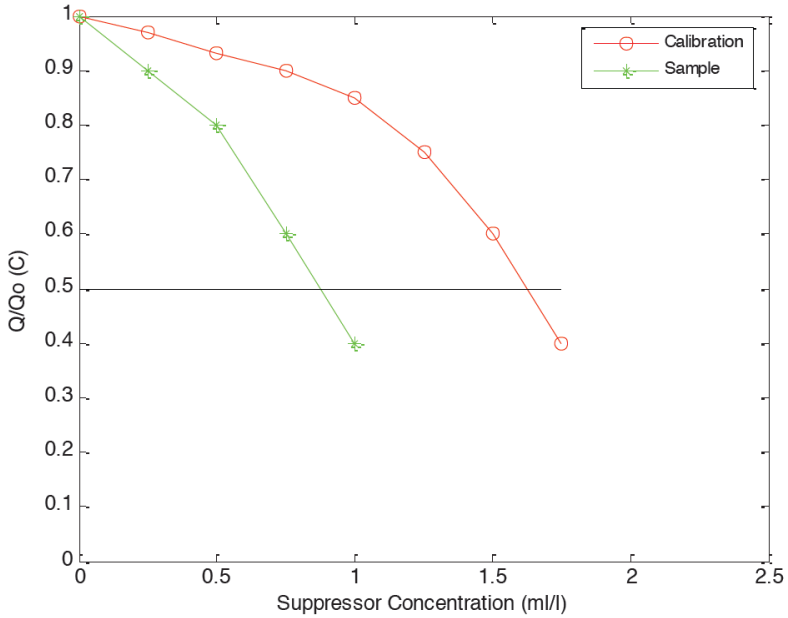


Figure 22. Typical MLAT plot for accelerator determination

The suppressor determination requires a calibration curve and typically 5 − 10 CVS analysis steps, as well as the maintenance of a VMS solution and a test suppressor solution.

4.1.3 Leveler Determination Using Response Curve

The leveler concentration is determined by using a response curve calibration [38]. The first step is to take a VMS solution and add sufficient amounts of suppressor and accelerator to saturate the effects of each. The next step is to generate a calibration curve by adding small amounts of leveler and recording the reduction in stripping charge. Next, the actual sample can be analyzed by adding an amount of it to a fresh solution of the VMS + Suppressor + Accelerator

and recording this point. The reduction of the plating is noted on the calibration plot and from that the amount of leveler, in the sample, can be determined. A typical plot is shown in figure 23.

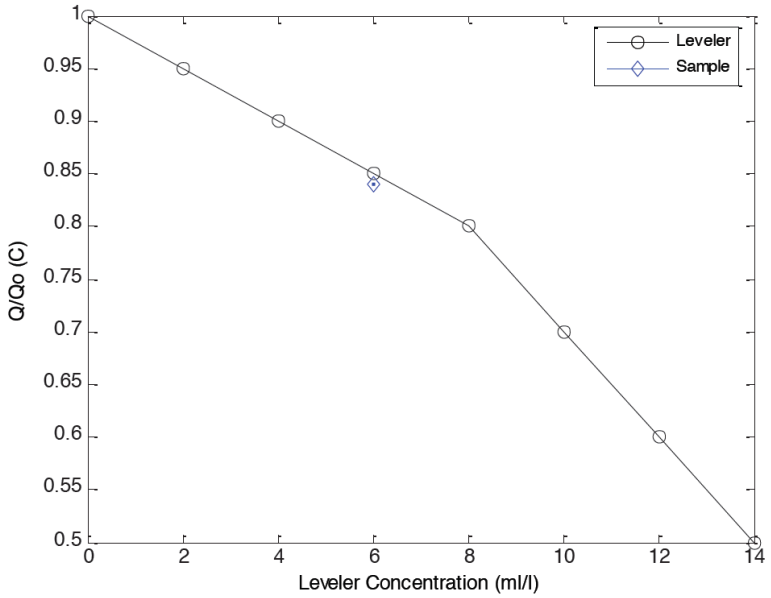


Figure 23. Typical Leveler Calibration and Determination Plot

Leveler determination requires a calibration plot and at least one CVS analysis. These standard analysis steps require calibration plots, at least 10 CVS analysis steps, and the maintenance of several solutions for diluting the VMS for each type of determination.

Chapter 5. Machine Learning Algorithms

Machine learning is a term used to describe the process of training a computer or electrical hardware to provide outputs for a given set of inputs and/or to predict and output from an unknown set of inputs. The most popular machine learning methods in order of decreasing popularity is shown below [39] :

- Decision trees
- Regression
- Cluster analysis
- Time series
- Neural nets
- Factor analysis
- Text mining
- Association rules
- Ensemble models
- Support vector machines
- Bayesian
- Anomaly detection
- Survival analysis
- Rule induction
- Social network analysis
- Genetic algorithms
- Link analysis
- Uplift modeling
- MARS

In this work several of these techniques will be tested and compared to determine if machine learning is a viable method for determining the levels of organic additives in a copper TSV plating bath, and if viable, which technique will give the most accurate results. Polynomial regression, Support Vector Regression (SVM), Extreme Learning Machines (ELM), Error Back Propagation (EBP), Levenberg-Marquardt (LM), and a modified LM – Neuron by Neuron (NBN) techniques will be discussed and compared.

5.1 First Order Algorithms

5.1.1 Polynomial Regression

The first machine learning method evaluated is polynomial regression: it is basically linear regression where the relationship between the independent variable x and the dependent variable y is a polynomial of nth degree**.**

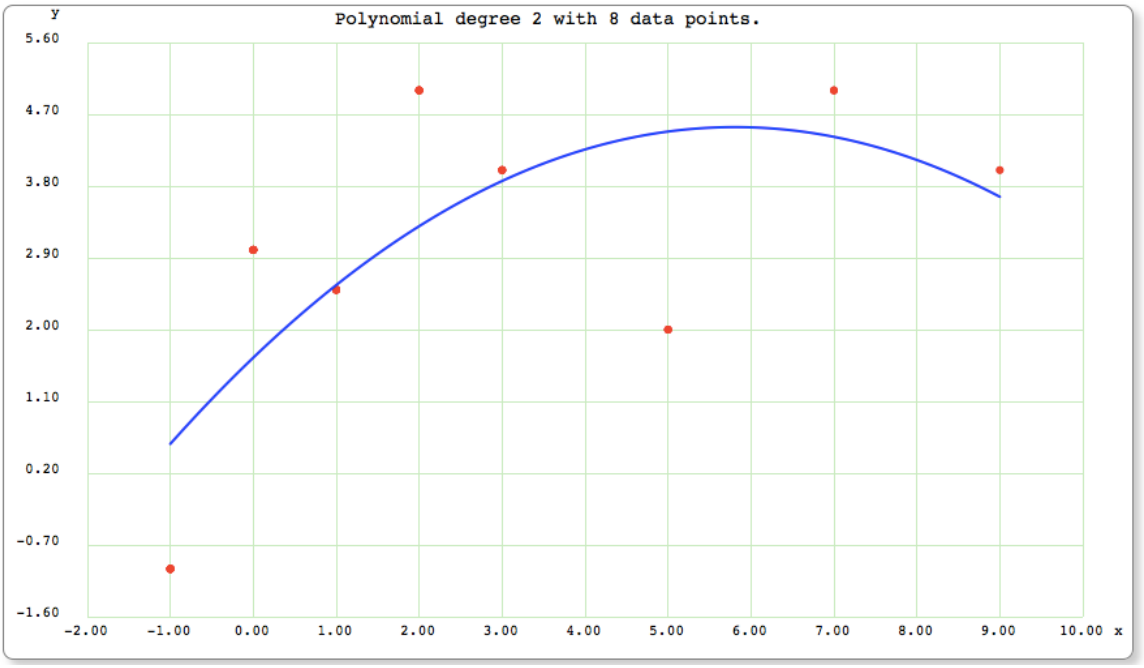


Figure 24. Polynomial regression example

As seen in figure 24, the 2nd order polynomial approximates the given data points. As the polynomial order increases the approximation will fit the data more closely, but at the expense of over-fitting, where the approximation is no longer generalized enough to provide a good prediction, or follow the data trend shown in figure 25. The optimum order must be determined with a validation dataset.

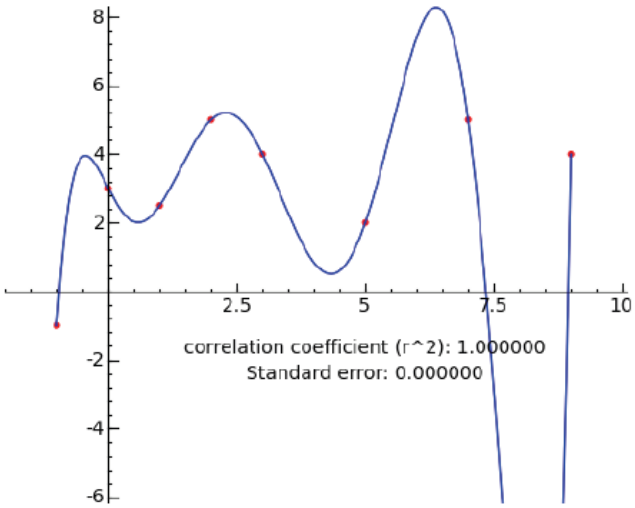


Figure 25. High order polynomial showing the problem with over-fitting

This polynomial least squares is a linear regression since the coefficients are linear and x, x^2, x^3, … are treated as independent variables. A 4 dimensional, 2nd order polynomial has the form of equation 1.

$$y = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_1^2 + a_6x_2^2 + a_7x_3^2 + a_8x_4^2 \quad \textbf{(1)}$$

The polynomial will always have the number of coefficients shown in equation 2 (without cross-multiplication).

$$\#Coefficients = (order * dimension) + 1 \quad (2)$$

Since we are solving for the number of coefficients – the number of unknowns is equal to the number of coefficients and thus there must be at least as many training vectors as the number of coefficients.

To find the coefficients that give the minimum gradient you need to set the gradient of the residuals to 0. The derivation gives an equation known as the "Normal Equation" [40]

$$\boldsymbol{Aa = y \rightarrow A^T Aa = A^T y} \tag{3}$$

To solve for a (the set of coefficients) $\mathbf{A^T A}$ must be inverted.

$$\boldsymbol{(A^T A)^{-1}(A^T A)a = (A^T A)^{-1} A^T y} \tag{4}$$

$$\boldsymbol{a = (A^T A)^{-1} A^t y} \tag{5}$$

For the polynomial shown in equation 1 the A matrix, A matrix transpose, and A^TA is shown below.

Matrix A

1	X_{11}	X_{12}	X_{13}	X_{14}	X_{11}^2	X_{12}^2	X_{13}^2	X_{14}^2
1	X_{21}	X_{22}	X_{23}	X_{24}	X_{21}^2	X_{22}^2	X_{23}^2	X_{24}^2
1	X_{31}	X_{32}	X_{33}	X_{34}	X_{31}^2	X_{32}^2	X_{33}^2	X_{34}^2
1	X_{41}	X_{42}	X_{43}	X_{44}	X_{41}^2	X_{42}^2	X_{43}^2	X_{44}^2
1	X_{51}	X_{52}	X_{53}	X_{54}	X_{51}^2	X_{52}^2	X_{53}^2	X_{54}^2
1	X_{61}	X_{62}	X_{63}	X_{64}	X_{61}^2	X_{62}^2	X_{63}^2	X_{64}^2
1	X_{71}	X_{72}	X_{73}	X_{74}	X_{71}^2	X_{72}^2	X_{73}^2	X_{74}^2
1	X_{81}	X_{82}	X_{83}	X_{84}	X_{81}^2	X_{82}^2	X_{83}^2	X_{84}^2
1	X_{91}	X_{92}	X_{93}	X_{94}	X_{91}^2	X_{92}^2	X_{93}^2	X_{94}^2

Matrix A^T

1	1	1	1	1	1	1	1	1
X_{11}	X_{21}	X_{31}	X_{41}	X_{51}	X_{61}	X_{71}	X_{81}	X_{91}
X_{12}	X_{22}	X_{32}	X_{42}	X_{52}	X_{62}	X_{72}	X_{82}	X_{92}
X_{13}	X_{23}	X_{33}	X_{43}	X_{53}	X_{63}	X_{73}	X_{83}	X_{93}
X_{14}	X_{24}	X_{34}	X_{44}	X_{54}	X_{64}	X_{74}	X_{84}	X_{94}
X_{11}^2	X_{21}^2	X_{31}^2	X_{41}^2	X_{51}^2	X_{61}^2	X_{71}^2	X_{81}^2	X_{91}^2
X_{12}^2	X_{22}^2	X_{32}^2	X_{42}^2	X_{52}^2	X_{62}^2	X_{72}^2	X_{82}^2	X_{92}^2
X_{13}^2	X_{23}^2	X_{33}^2	X_{43}^2	X_{53}^2	X_{63}^2	X_{73}^2	X_{83}^2	X_{93}^2
X_{14}^2	X_{24}^2	X_{34}^2	X_{44}^2	X_{54}^2	X_{64}^2	X_{74}^2	X_{84}^2	X_{94}^2

Matrix A^TA

9	ΣX_{n1}	ΣX_{n2}	ΣX_{n3}	ΣX_{n4}	ΣX_{n1}^2	ΣX_{n2}^2	ΣX_{n3}^2	ΣX_{n4}^2
ΣX_{n1}	ΣX_{n1}^2	$\Sigma X_{n1}X_{n2}$	$\Sigma X_{n1}X_{n3}$	$\Sigma X_{n1}X_{n4}$	ΣX_{n1}^3	$\Sigma X_{n1}X_{n2}^2$	$\Sigma X_{n1}X_{n3}^2$	$\Sigma X_{n1}X_{n4}^2$
ΣX_{n2}	$\Sigma X_{n1}X_{n2}$	ΣX_{n2}^2	$\Sigma X_{n2}X_{n3}$	$\Sigma X_{n2}X_{n4}$	$\Sigma X_{n2}X_{n1}^2$	ΣX_{n2}^3	$\Sigma X_{n2}X_{n3}^2$	$\Sigma X_{n2}X_{n4}^2$
ΣX_{n3}	$\Sigma X_{n1}X_{n3}$	$\Sigma X_{n2}X_{n3}$	ΣX_{n3}^2	$\Sigma X_{n3}X_{n4}$	$\Sigma X_{n3}X_{n1}^2$	$\Sigma X_{n3}X_{n2}^2$	ΣX_{n3}^3	$\Sigma X_{n3}X_{n4}^2$
ΣX_{n4}	$\Sigma X_{n1}X_{n4}$	$\Sigma X_{n2}X_{n4}$	$\Sigma X_{n3}X_{n4}$	ΣX_{n4}^2	$\Sigma X_{n4}X_{n1}^2$	$\Sigma X_{n4}X_{n2}^2$	$\Sigma X_{n4}X_{n3}^2$	ΣX_{n4}^3
ΣX_{n1}^2	ΣX_{n1}^3	$\Sigma X_{n1}^2X_{n1}^2$	$\Sigma X_{n3}X_{n1}^2$	$\Sigma X_{n4}Xn_1^2$	ΣX_{n1}^4	$\Sigma X_{n1}^2X_{n2}^2$	$\Sigma X_{n1}^2X_{n3}^2$	$\Sigma X_{n1}^2X_{n4}^2$
ΣX_{n2}^2	$\Sigma X_{n1}X_{n2}^2$	ΣX_{n2}^3	$\Sigma X_{n3}X_{n2}^2$	$\Sigma X_{n4}X_{n2}^2$	$\Sigma X_{n1}^2X_{n2}^2$	ΣX_{n2}^4	$\Sigma X_{n2}^2X_{n3}^2$	$\Sigma X_{n2}^2X_{n4}^2$
ΣX_{n3}^2	$\Sigma X_{n1}X_{n3}^2$	$\Sigma X_{n2}Xn_3^2$	ΣX_{n3}^3	$\Sigma X_{n4}X_{n3}^2$	$\Sigma X_{n1}^2X_{n3}^2$	$\Sigma X_{n2}^2X_{n3}^2$	ΣX_{n3}^4	$\Sigma X_{n3}^2X_{n4}^2$
ΣX_{n4}^2	$\Sigma X_{n1}X_{n4}^2$	$\Sigma X_{n2}X_{n4}^2$	$\Sigma X_{n3}X_{n4}^2$	ΣX_{n4}^3	$\Sigma X_{n1}2X_{n4}^2$	$\Sigma X_{n2}^2Xn_4^2$	$\Sigma X_{n3}^2X_{n4}^2$	ΣX_{n4}^4

n = number of rows (data vectors)

To determine the coefficients, matrix (A^TA) will have to be inverted. This is not a trivial task even though it is a square matrix. Singular value decomposition (SVD) can be used and is probably the most stable method to perform the inversion. Another method of solving for the coefficients is to use QR factorization. QR factorization substitutes **QR** for **A** and determines the matrices **Q** & **R** where **Q** is orthogonal ($\mathbf{Q^TQ=I}$) and **R** is upper triangular.

Substituting **A=QR** in the normal equation and solving in terms of **R** & **Q** the result is given in equations 6-11:

$$\boldsymbol{A^TAa = A^Ty} \tag{6}$$

$$\boldsymbol{Q^TR^TQRa = Q^TR^Ty} \tag{7}$$

$$\boldsymbol{R^TRa = Q^TR^Ty} \tag{8}$$

$$\boldsymbol{Ra = Q^Ty} \tag{9}$$

$$\boldsymbol{R^{-1}Ra = R^{-1}Q^Ty} \tag{10}$$

$$\boldsymbol{a = R^{-1}Q^Ty} \tag{11}$$

The unknown vector **a** (polynomial coefficients) can be found once **Q** & **R** are determined. **R** is easily inverted since it is an upper rectangular matrix.

5.1.2 Extreme learning machine

Extreme Learning Machine (ELM) **[41]–[43]** is a method of quickly solving single hidden layer forward neural networks without using back propagation. The output weights are analytically determined using inverse matrix operations, while the input weights and hidden layer biases are randomly generated. This method promises to greatly speed up the solution

process and give as good or better generalization. According to the ELM papers referenced above, this method trains two orders of magnitude faster than error back propagation (EBP) and an order of magnitude faster than Support Vector Machine (SVM) and has a success rate better than both. The ELM method will only work for single hidden layer networks, but according to [44], any single hidden layer network can approximate any continuous function and implement any classification application.

The ELM process can be described by the following steps for the Single Hidden Layer Forward Neural Network (SLFN) shown in figure 26.

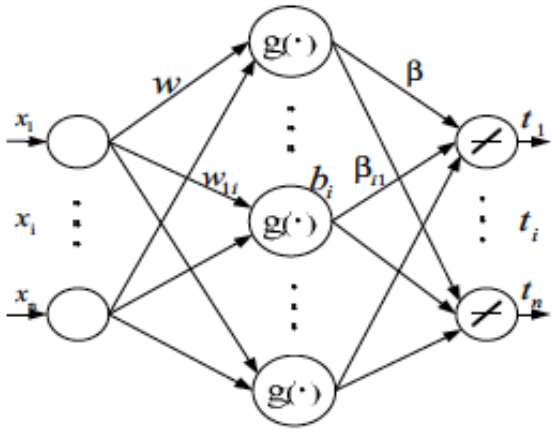


Figure 26. Single Hidden Layer Topology for ELM

For a training dataset (x_i,t_i) – where x is an input, t is an output, and i = (1, …., N)

$g(x)$ – activation function - typically a simple sigmoid function:

$$g(x) = 1/(1 + e^{-x}) \tag{12}$$

L = number of hidden layers

Step 1. Assign random weights, between 0 and 1, $\boldsymbol{w}_i$, and bias b_i, i=1, …. ,L.
Step 2. Calculate the hidden layer output matrix **H**.
Step 3. Calculate the output weight β. ($\boldsymbol{\beta} = \mathbf{H}^{+}\mathbf{T}$), where $\mathbf{H}^{+}$ is the Moore-Penrose generalized

inverse (or Pseudoinverse [45]) of matrix **H**

The output Matrix **H** is defined as:

$$\mathbf{H}(w_1, \cdots, w_L, b_1, \cdots, b_{L,} x_1, \cdots, x_N) = \begin{bmatrix} \boldsymbol{g}(w_1 * x_1 + b_1) \cdots \boldsymbol{g}(w_L * x_1 + b_L) \\ \vdots \quad \cdots \quad \vdots \\ \boldsymbol{g}(w_1 * x_N + b_1) \cdots \boldsymbol{g}(w_L * x_N + b_L) \end{bmatrix} \quad (13)$$

Since **H** is usually not a square matrix, it is not a trivial matter to perform the psuedoinverse of *m*x*n* matrix **H.** There are several methods to do this, but the most useful one is Singlular Value Decomposition (SVD), since it can be used on singular matrices and is not a slow iterative process. The basic steps of the SVD method are summarized below:

For a given $m \times n$ matrix **H**, there exists a factorization of the form

$$\mathrm{H} = \mathbf{U\Sigma V^T} \quad (14)$$

where **U** is an $m \times m$ unitary matrix, **Σ** is a $m \times n$ diagonal matrix with non-negative real numbers on the diagonal, and the $n \times n$ unitary matrix $\mathbf{V^T}$ denotes the transpose of the $n \times n$ unitary matrix **V**. Such a factorization is called a singular value decomposition of **H**. The diagonal entries σ_i of **Σ** are known as the **singular values** of **H**. A common convention is to list the singular values in descending order.

Once the factorization of M is found, the singular value decomposition can be used for computing the pseudoinverse of a matrix. The pseudoinverse of the matrix **H** with singular value decomposition $\mathbf{H} = \mathbf{U\Sigma V^T}$ is

$$\mathbf{H^+ = V\Sigma^+ U} \quad (15)$$

where $\mathbf{\Sigma}^{+}$ is the pseudoinverse of $\mathbf{\Sigma}$, which is formed by replacing every non-zero diagonal entry by its reciprocal and transposing the resulting matrix. Once this matrix is calculated the output weights can be tabulated and the network is complete and ready for testing.

An example of using an ELM network to solve a parity 3 problem is shown in the following.

Table 1. Parity 3 Table

X_1	X_2	X_3	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

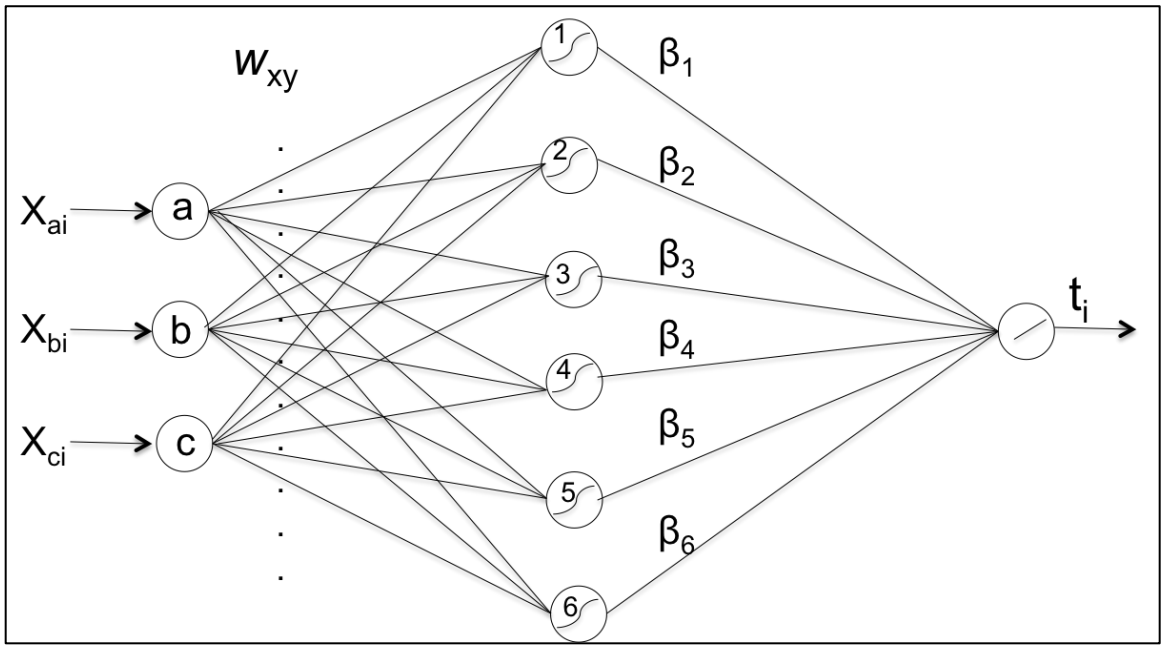


Figure 27. ELM network for parity 3 problem, shown is 6 hidden layer neurons, the actual network, for parity 3, requires 8 hidden layer neurons

A python script written to implement the ELM algorithm is given in Appendix A. The output from this script, for a parity 3 problem, is shown below.

Output:

Output Weights: (8 hidden layer neurons)

[-2.04742978e+02 3.86405547e+02 -9.80026727e-01 -1.00796852e+03 4.98503268e+00
4.87324607e+02 -6.67910173e+01 3.70362928e+02]

Outputs: (8 hidden layer neurons)

```
  [ -5.11590770e-13
     1.00000000e+00
     1.00000000e+00
    -2.84217094e-13
     1.00000000e+00
    -1.98951966e-13
    -1.98951966e-13
     1.00000000e+00 ]
```

(6 hidden layer neurons)

```
  [  0.40873744
     0.49533829
     0.67783524
     0.44787446
     0.53262453
     0.66622452
     0.38109374
     0.39732011 ]
```

As can be seen, from the above discussion and software, as long as a pseudoinversion method is available, ELM can train the weights very quickly without any iterative actions. The parity problem is a highly non-linear problem and it is not a surprise that it takes 9 neurons to solve it.

5.1.3 Support Vector Machines

Support Vector Machines (SVM) [46] learning is a relatively recent addition to the machine learning tools. It uses a technique where hyperplanes, in extended multidimensional space, are defined from a subset of the training vectors that specify the prediction path. Training data within a defined distance Epsilon (ε) of the prediction path is ignored leaving a set of "support vectors". An example of how moving data to a higher dimensional space will help classify the different sets of data is shown in figure 28.

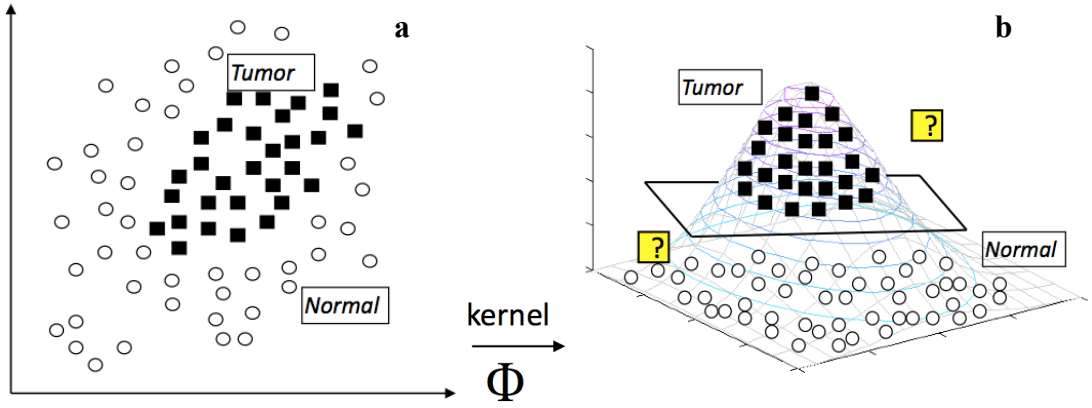


Figure 28. Support Vector Machine extended dimensional conversion

As shown in figure 28 (a), the two sets of data, in 2 dimensional space, are not separable by a plane, but if a third dimension is added to the data Figure 28 (b), the data is separable with a single plane. The vectors closest to the plane from both sides will act as the "support vectors". To perform regression type analysis there is an extension to SVM that allows a set of support vectors to define a least squares type of plane or line that follows the data. The support vector math is extremely intensive and will not be discussed in this work. There are multiple

commercial and open source software packages that will perform the SVM technique for both classification and regression problems. The Rapid-Miner software package [47] will be used to solve for the support vectors using libSVM as the underlying code. It is a powerful tool for setting up a machine learning "project", allowing one to use optimization to find the optimum coefficients and cross-validation to divide the data set into training and validation sets to determine generalization capability. While it is no longer an open source program; it still has an educational set of modules that will perform SVM and SVR (support Vector Regression).

During SVM training there is a couple of parameters that must be adjusted to allow a low error. One of these is the "C" parameter. The C parameter tells the SVM optimization to minimize the chance of misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. A very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable. This C parameter must be optimized for a low error, it is not always easy to choose this value, users typically use a validation set to adjust this parameter.

5.1.4 Error Back Propagation

Error back propagation **[48]–[50]** is the most popular algorithm for training neural networks and is found in virtually all software packages that train Multi Layer Perceptron (MLP) network topologies, shown in figure 29. An MLP network is a feedforward topology that may have several levels of neurons, but it only has connections to neurons on the next level. EBP is a

relatively easy to understand and implement algorithm, which is part of the reason it is so popular.

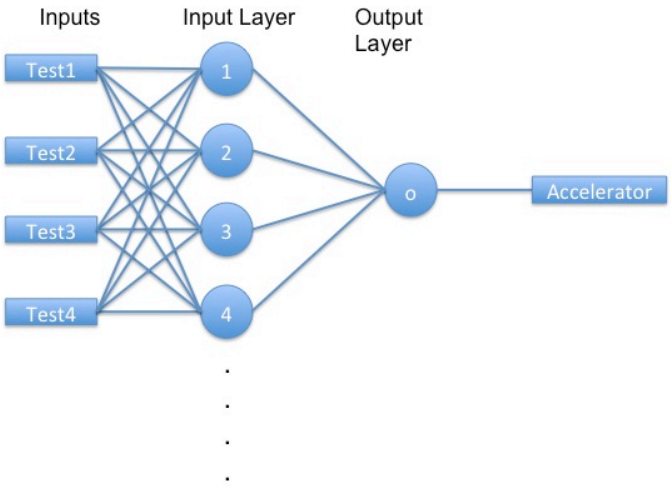


Figure 29. MLP Topology used for EBP Analysis

While it has been determined that EBP can be used on more complicated networks, such as networks that allow connections across layers, or connections that extend beyond the next layer, they are very slow to converge compared with more advanced 2nd order algorithms. Researchers have developed various techniques for speeding up EBP, such as momentum [51] , RPROP [52] , and adaptive learning constant [53] . While these techniques have managed to make a significant difference in the speed of EBP, they are not able to overcome the 1000 times speed difference between the EBP first-order algorithm and several of the 2nd-order algorithms. One advantage of EBP is that it can be used on networks with almost an unlimited number of neurons, which is good because more often than not it takes a lot of neurons to provide a convergent solution.

The EBP algorithm is a gradient based algorithm where the network is trained to find a minimum of the error function. The weights are adjusted to continually reduce the error until an acceptable level is reached. Of course, as mentioned earlier, the network can be over-trained

causing the resulting network, with adjusted weights, to not provide a generalized solution for inputs not contained in the training set.

Figure 30 shows a small network that contains 2 inputs, 2 neurons in the first layer and a single output neuron.

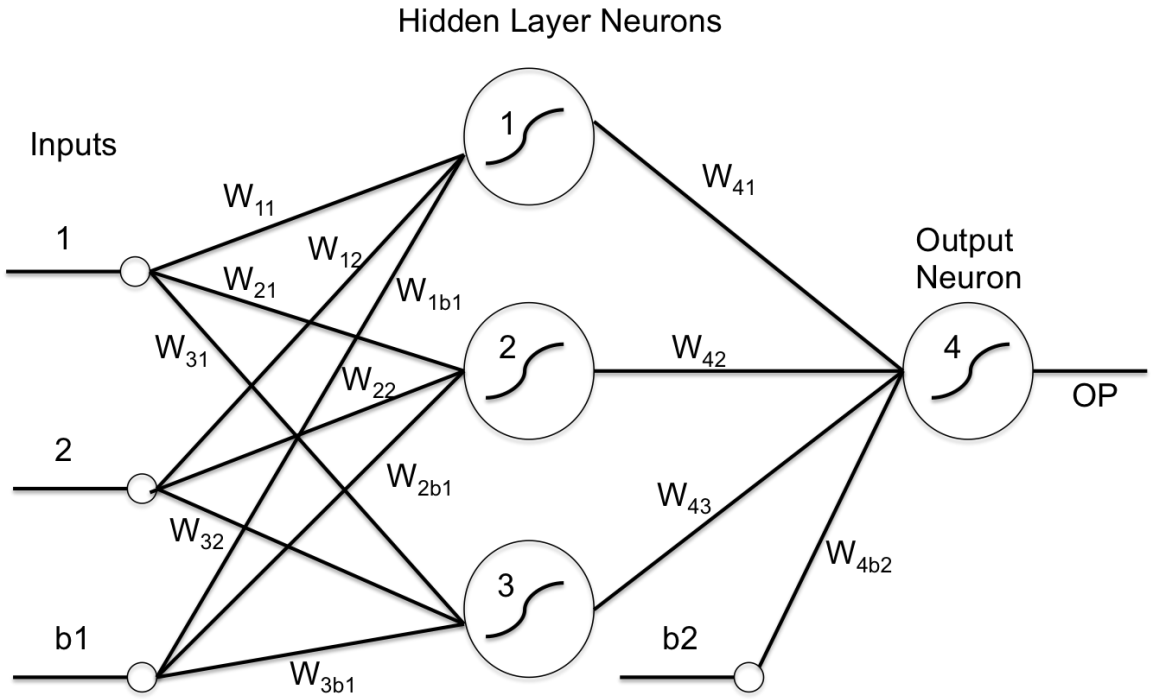


Figure 30. MLP network with weights and biases

There is typically a set of neuron bias lines connected to each neuron for neuron function threshold adjustment. The input, first layer, and bias weights are shown, with a notation $W_{ji,}$ where j is the index of the neuron and i is the input, or if it is between two neurons W_{kj} where j is the index of the neuron closest to the input and k is the index of the neuron toward the output of the network.

The gradient decent method adjusts the network weights in an effort to reduce the resulting error. The total error can be given as follows:

$$\boldsymbol{E} = \frac{1}{2}\sum_{\boldsymbol{k}}^{\boldsymbol{X}}(\boldsymbol{t_k} - \boldsymbol{out_k}) \tag{16}$$

- where $\boldsymbol{t_k}$ is the desired output and $\boldsymbol{out_k}$ is the actual output and x is the number of output neurons.

We want to find a delta weight that will reduce the error:

$$\Delta \boldsymbol{W} \propto -\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial W}} \tag{17}$$

To do this we will look at the change in output E due to a change in a single output W, or, in other words, the sensitivity of the error to a small change in a weight.

$$\Delta W_{kj} \propto -\frac{\partial E}{\partial W_{kj}} \tag{18}$$

Since the error is not a direct function of the weights, we must expand the equation as follows:

$$\Delta W_{kj} = -\mu \frac{\partial E}{\partial out_k}\frac{\partial out_k}{net_k}\frac{net_k}{W_{kj}} \tag{19}$$

Where $\boldsymbol{\mu}$ is a constant. Now we should examine each of the partial terms.

The first term is the partial derivative of the output error with respect to the output:

$$\frac{\partial E}{\partial out_k} = \frac{\partial(\frac{1}{2}(t_{k-out_k})^2)}{\partial out_k} = -(t_k - out_k) \tag{20}$$

The second term is the partial derivative of the output with respect to the output neuron's net input:

$$\frac{\partial out_k}{\partial net_k} = \frac{\partial(1+e^{-net_k})^{-1}}{\partial net_k} = \frac{e^{-net_k}}{(1+e^{-net_k})^2} \tag{21}$$

This can be rewritten in terms of the activation function:

$$1 - \frac{1}{1 + e^{-net_k}} = \frac{e^{-net_k}}{1 + e^{-net_k}} \tag{22}$$

$$\therefore \frac{e^{-net_k}}{(1 + e^{-net_k})^2} = out_k \ (1 - out_k) \tag{23}$$

The last term is the partial derivative of the output neuron's net input with respect to an output weight:

$$\frac{\partial net_k}{\partial W_{kj}} = \frac{\partial (W_{kj} out_j)}{\partial W_{kj}} = out_j \tag{24}$$

Now if we recombine the three derivatives we get:

$$\Delta W_{kj} = \mu (t_k - out_k) out_k (1 - out_k) out_j \tag{25}$$

We can assign terms to the various portions of this equation:

$$err_k = (t_k - out_k) \tag{26}$$

$$Derivative\ of\ activation\ function = A'(out_k) = out_k (1 - out_k) \tag{27}$$

$$out_j = previous\ layer\ output \tag{28}$$

By combining the err and activation function derivative we have a sort of sensitivity factor we'll call del:

$$\delta_k = err_k out_k (1 - out_k) \tag{29}$$

We can now rewrite the output delta weight as:

$$\Delta W_{kj} = \mu \delta_k out_j \tag{30}$$

We will also want to look at how much to change the input weights:

$$\Delta W_{ji} \propto -\left[\sum_k^X \frac{\partial E}{\partial out_k} \frac{\partial out_k}{\partial net_k} \frac{\partial net_k}{\partial out_j} \right] \frac{\partial out_j}{\partial net_j} \frac{\partial net_j}{\partial W_{ji}} = \frac{\partial E}{\partial W_{ji}} \tag{31}$$

From a similar process as shown in the output formula, we can rewrite the above as:

$$\Delta W_{ji} = \mu \left[\sum_{k}^{X} (t_K - out_k) out_k (1 - out_k) W_{kj} \right] out_j (1 - out_j) input_i \quad (32)$$

By rewriting equation (29) as:

$$\delta_k = (t_k - out_k) out_k (1 - out_k) \quad (33)$$

we can rewrite the above formula as:

$$\Delta W_{ji} = \mu \left[\sum_{k}^{X} \delta_k W_{kj} \right] out_j (1 - out_j) input_i \quad (34)$$

We can now define a new variable $\boldsymbol{err_j}$ **and** $\boldsymbol{\delta_j}$ as:

$$err_j = \left[\sum_{k}^{X} \delta_k W_{kj} \right] \quad (35)$$

And

$$\delta_j = err_j out_j (1 - out_j) \quad (36)$$

This gives:

$$\Delta W_{ji} = \mu \delta_j input_i \quad (37)$$

The last step is to add the delta weights to their respective weight.

The weights training process proceeds as follows:

Forward Process

1.) The weights are assigned a random number between 0 and 1.
2.) An input pattern is applied to the inputs
3.) The input patterns are multiplied by the respective weights to provide an input net to the following neuron.
4.) Apply the neuron function to the net and generate an output at each neuron.
5.) Multiply the neuron output by the respective weights to form an input net for the output neuron.
6.) Apply the output neuron function to the output net to get the final output.

7.) Determine the output error by subtracting the desired output from the actual output.

Back Propagation

8.) The output error sensitivity factor is determined and "placed" at the input of the output neuron.
9.) The sensitivity factor is multiplied by the output weights to back propagate the error to the respective outputs of the hidden layer neurons. A delta weight is added to the output weights
10.) The error sensitivity for the hidden neurons are determined and "placed" at the front of the hidden layer neurons.
11.) The error sensitivities are multiplied by the inputs and a new delta input weight is determined.
12.) This process is repeated, beginning at step 2, for each input pattern, and then repeated multiple times (iterations) until the weights are adjusted to give a minimum error for all the outputs.
13.) The whole process, starting from the beginning, may be repeated several times (training times) to reduce the possibility of being stuck in a local minimum of the error function.

It might be beneficial to provide an example to illustrate the process. To keep it readable, a parity-2 table will be used as the inputs and outputs for a simple 3 neuron hidden layer MLP network (shown in figure 32). The parity-2 truth table is shown in table 2.

Table 2. Parity 2 truth table

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

1.) The first step is to assign the random weights:

inputWeights [# inputs + bias, # hidden neurons]

[0.55719336 0.69559482 0.42489087]
[0.34693303 0.04731304 0.19875418]
[0.9520813 0.07458135 0.84528506]

outputWeights [# hidden neurons + bias, # outputs]

[0.19099137]
[0.94340588]
[0.54034952]
[0.38595208]

2.) The second step is to apply the inputs and generate the net/input for each hidden neuron:

net1 [# input patterns, # hidden neurons]
[0.9520813 0.07458135 0.84528506]
[1.29901433 0.12189439 1.04403924]
[1.50927466 0.77017617 1.27017593]
[1.85620769 0.81748921 1.46893011]

3.) The next step is to generate the output of each hidden neuron by applying the sigmoid activation function to the hidden neuron input net.

out1 [# input patterns, # hidden neurons]

[0.72153355 0.5186367 0.69957714]
[0.78566905 0.53043592 0.73962863]
[0.81895369 0.683559 0.78077286]
[0.86485431 0.69370311 0.81289471]

4.) The next step is to add the output neuron bias to the out1 array to form the inputs to the output neuron. Then multiply this new 'ip2' array by the output weights to generate the input net to the output neuron:

net2 [# input patterns, # outputs]
[1.39105985]
[1.43608243]
[1.60912899]
[1.64482265]

5.) Next the output is generated by applying the sigmoid activation function to the output neuron input net.

out2 [# input patterns, # outputs]

[0.80076139]
[0.80784726]
[0.83329042]
[0.83819009]

6.) Determine the errors, (desired output – output):

err2 [# input patterns, # ouputs]

[-0.80076139]
[0.19215274]
[0.16670958]
[-0.83819009]

That completes the first iteration of the forward process – the next steps will back propagate the errors to adjust the weights in a manner to reduce the errors.

7.) The derivative of the output is multiplied by the error to determine the sensitivity factor:

del2 [# input patterns, # outputs]

[-0.12775554]
[0.02982788]
[0.02315888]
[-0.1136816]

8.) Next the sensitivity factor is multiplied by the output weights, backing the error to the hidden layer neurons:

err1 [# hidden neurons + bias, # input patterns]

[-0.02440021 0.00569687 0.00442315 -0.0217122]
[-0.12052533 0.0281398   0.02184822 -0.10724789]
[-0.06903265 0.01611748 0.01251389 -0.0614278]
[-0.04930752 0.01151213 0.00893822 -0.04387565]

The last row of the err1 matrix is the back error propagation on the bias line, and is not needed for further back propagation. So, it is removed before the next steps.

9.) The next step is to determine the sensitivity factor of the back propagated error for each hidden layer by multiplying err1 by the derivative of the hidden layer outputs.

del1 [# input patterns, # hidden neurons]

[-0.00490256 -0.03008947 -0.01450852]
[0.00095931 0.00700888 0.00310387]
[0.00065581 0.0047259   0.00214196]
[-0.00253775 -0.02278793 -0.00934298]

10.) The amount that the output weights are adjusted (delta output weight) is determined by multiplying the del2 matrix by the hidden layer outputs and the bias input. All entries, except the last one, are the sum of the products of del2 X out2, the last entry is the sum of the products of the output bias and the output bias weight for each input pattern.

dw2 [# hidden layers + bias]

[-0.14809704]
[-0.11346775]
[-0.14164265]
[-0.18845038]

11.) Next the amount the input weights are adjusted is determined by multiplying the hidden layer sensitivity factor by the inputs and the bias input. Each delta input weight is the sum of the products of del1 and the input for each input pattern.

dw1 [# inputs + bias, # hidden layers]

[-0.00188194 -0.00157844 -0.00582518]
[-0.01806203 -0.01577905 -0.04114262]
[-0.00720102 -0.0062391 -0.01860566]

12.) dw1 is subtracted from the respective input weights and dw2 is subtracted from the respective output weights. Then the process is repeated (iteration), not including the first step of assigning random weights, until a predetermined error is achieved.

13.) The entire process is repeated (training) to reinitiate the weights to help reduce the possibility of the process becoming "stuck" in a local minima of the error function.

The total error is commonly determined by one of two different methods – summed squared error (SSE), or root mean squared error (RMSE).

A python script that implements the above steps, written by the author, is shown in Appendix B.

A typical output for a parity 3 problem is shown in table 3.

Table 3. EBP program output

```
Training #  1 Iteration #  1 SSE=  1.25064612831
Training #  1 Iteration #  10 SSE=  1.19705651195
Training #  1 Iteration #  100 SSE=  0.774592249767
Training #  1 Iteration #  1000 SSE=  0.00126155535867
Training #  1 Iteration #  2000 SSE=  0.000546331855821
Training #  1 Iteration #  4000 SSE=  0.00025259226898
Training #  1 Iteration #  6000 SSE=  0.000163577401642
Training #  1 Iteration #  8000 SSE=  0.000120781567025
Training #  1 Iteration #  10000 SSE=  9.56729250499e-05
Training #  1 Iteration #  12000 SSE=  7.9179336313e-05
Training #  1 Iteration #  14000 SSE=  6.75222581545e-05
Training #  1 Iteration #  16000 SSE=  5.88491885986e-05
Training #  1 Iteration #  18000 SSE=  5.2145818529e-05
Training #  1 Iteration #  20000 SSE=  4.68103936315e-05
Training #  1 Iteration #  22000 SSE=  4.24634351565e-05
Training #  1 Iteration #  24000 SSE=  3.88538399081e-05
Training #  1 Iteration #  26000 SSE=  3.58088553343e-05
Training #  1 Iteration #  28000 SSE=  3.32057468725e-05
Training #  1 Iteration #  30000 SSE=  3.09549261586e-05
Training #  1 Iteration #  32000 SSE=  2.89894744973e-05
Training #  1 Iteration #  34000 SSE=  2.72584022729e-05
Training #  1 Iteration #  36000 SSE=  2.57221776321e-05
Training #  1 Iteration #  38000 SSE=  2.43496807311e-05
Training #  1 Iteration #  40000 SSE=  2.31160801845e-05
```

```
Training #  1 Iteration #  42000 SSE=  2.20013216834e-05
Training #  1 Iteration #  44000 SSE=  2.09890323855e-05
Training #  1 Iteration #  46000 SSE=  2.0065713521e-05
Training #  1 Iteration #  48000 SSE=  1.92201364948e-05
SSE=  [  1.84432580e-05]
Output:
[[ 0.00230754]
 [ 0.99813799]
 [ 0.99776322]
 [ 0.00215598]]
```

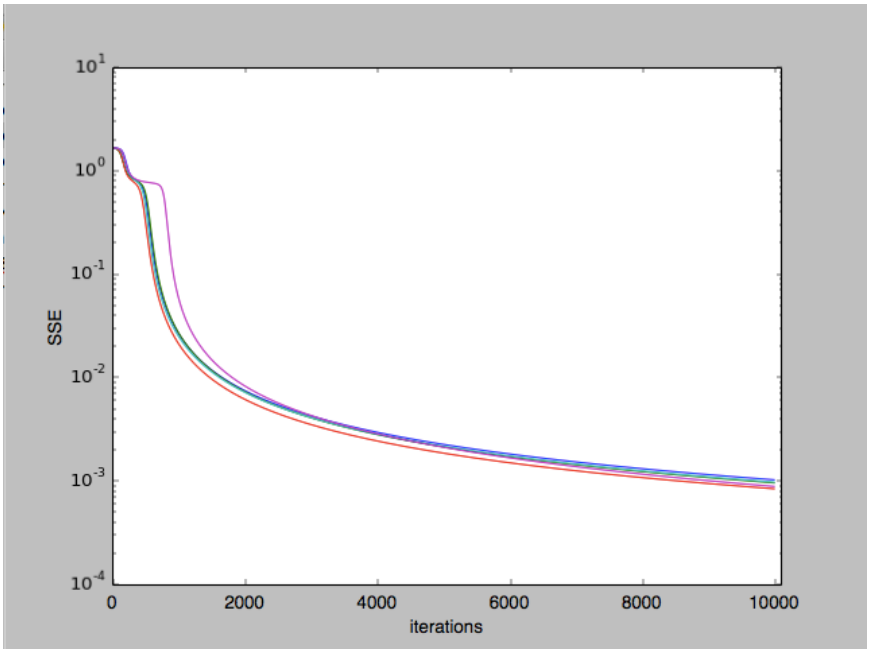


Figure 31. Error Back Propagation (EBP), 5 tries and 10000 iterations - for a parity 3 problem

5.2 Second Order Algorithms

In gradient decent methods, such as EBP, the step size that is used to descend down the gradient of the error function curve must be kept small in order to not move out of the trough. But with the small step size it can take many iterations to reach the bottom of the gradient, slowing down the training process. There have been methods mentioned that will improve this

process by varying the coefficient in order to change the decent rates. In order to speed up the process, several second order algorithms have been used to train neural networks. These include Newton, Gauss-Newton (GN) [54] , Levenburg-Marquardt (LM) [55], [56] , and Neuron by Neuron (NBN) [57]–[59] .

5.2.1 Newton

As seen in the previous section the weights are changed relative to the direction and magnitude of the gradient of the error function and is given by the equation (39):

$$\Delta W_{kj} = -u\frac{\partial E}{\partial W_{kj}} \tag{38}$$

If the gradient is:

$$g = \frac{\partial E}{\partial W_{kj}} \tag{39}$$

Then the new weight is:

$$W_{kj} = W_{kj} + \Delta W_{kj} = W_{kj} - ug_k \tag{40}$$

The newton method begins with the assumption that the gradient is a function of the weights of the system:

$$\begin{cases} g_1 = F_1(w_1 + w_2 + \cdots + w_n) \\ g_2 = F_2(w_1 + w_2 + \cdots + w_n) \\ \cdots \\ g_3 = F_3(w_1 + w_2 + \cdots + w_n) \end{cases} \tag{41}$$

If $\boldsymbol{g_1}$is expanded using the Taylor series, the first order approximation results are:

$$g_1 = g_{1,0} + \frac{\partial g_1}{\partial W_1}\Delta w_1 + \frac{\partial g_2}{\partial W_2}\Delta w_2 + \cdots + \frac{\partial g_n}{\partial W_n}\Delta w_n \tag{42}$$

The same can be done for each of the other gradients.

By noting that the gradient is defined as :

$$\boldsymbol{g} = \frac{\partial E(x,w)}{\partial w} = \left[\frac{\partial E}{\partial w_1}\ \frac{\partial E}{\partial w_2} \dots \frac{\partial E}{\partial w_n}\right]^T \tag{43}$$

and that the minima is found at $\boldsymbol{g} = \boldsymbol{0}$

equation (x) can be rearranged to give:

$$-\boldsymbol{g_1} = \left[\frac{\partial^2 E}{w_1^2}\ \frac{\partial^2 E}{\partial w_1 \partial w_2} \dots \frac{\partial^2 E}{\partial w_1 \partial w_n}\right] \times [\Delta w_1] \tag{44}$$

with the Hessian Matrix defined as:

$$\boldsymbol{H} = \left[\frac{\partial^2 E}{w_1^2}\ \frac{\partial^2 E}{\partial w_1 \partial w_2} \dots \frac{\partial^2 E}{\partial w_1 \partial w_n}\right] \tag{45}$$

which can be written as:

$$-\boldsymbol{g} = \boldsymbol{H}\Delta\boldsymbol{w} \tag{46}$$

or

$$\boldsymbol{w_{k+1}} = \boldsymbol{w_k} - \boldsymbol{H^{-1}g} \tag{47}$$

To determine the components of the Hessian matrix, second order solutions to the total error function must be found. This can get very complicated for even small sized networks. The Newton algorithm can converge on a solution very quickly if the system is almost linear to begin with. This is not the case in most neural networks, causing this method to be mostly useless, due to divergence, for ANNs.

5.2.2 Gauss-Newton

In order to move away from having to solve second order total error functions found in the Hessian matrix, the Gauss-Newton algorithm replaces the need to solve the Hessian matrix with a solution based on a Jacobian matrix. For our Neural Network case the Jacobian matrix will be defined as:

$$\boldsymbol{j} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \cdots & \frac{\partial e_{1,1}}{\partial w_n} \\ \frac{\partial e_{1,2}}{\partial w_1} & \frac{\partial e_{1,2}}{\partial w_2} & \cdots & \frac{\partial e_{1,2}}{\partial w_n} \\ \vdots & & \cdots & \vdots \\ \frac{\partial e_{1,m}}{\partial w_1} & \frac{\partial e_{1,m}}{\partial w_2} & \cdots & \frac{\partial e_{1,m}}{\partial w_n} \\ \vdots & & \cdots & \vdots \\ \frac{\partial e_{p,1}}{\partial w_1} & \frac{\partial e_{p,1}}{\partial w_2} & \cdots & \frac{\partial e_{p,1}}{\partial w_n} \\ \frac{\partial e_{p,2}}{\partial w_1} & \frac{\partial e_{p,2}}{\partial w_2} & \cdots & \frac{\partial e_{p,2}}{\partial w_n} \\ \vdots & & \cdots & \vdots \\ \frac{\partial e_{p,m}}{\partial w_1} & \frac{\partial e_{p,m}}{\partial w_2} & \cdots & \frac{\partial e_{p,m}}{\partial w_n} \end{bmatrix} \tag{48}$$

The rows of the Jacobian matrix are equal to the number of outputs multiplied by the number of patterns, and the number of columns is equal to the number of weights. So, it could potentially get very large for a large number of patterns and a large network (many weights).

The gradient vector $\boldsymbol{g}$ in relation to the above Jacobian matrix can be found by the following derivation:

$$\boldsymbol{g_i} = \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_i}} = \frac{\boldsymbol{\partial}\left(\frac{\mathbf{1}}{\mathbf{2}}\sum_{\boldsymbol{p=1}}^{\boldsymbol{p}}\sum_{\boldsymbol{m=1}}^{\boldsymbol{m}}\boldsymbol{e_{p,m}^2}\right)}{\boldsymbol{\partial w_i}} = \sum_{\boldsymbol{p=1}}^{\boldsymbol{p}}\sum_{\boldsymbol{m=1}}^{\boldsymbol{m}}\left(\frac{\boldsymbol{\partial e_{p,m}}}{\boldsymbol{\partial w_i}}\boldsymbol{e_{p,m}}\right) \tag{49}$$

From the previous 2 equations the gradient vector $\boldsymbol{g}$ is

$$\boldsymbol{g} = \boldsymbol{Je} \tag{50}$$

where $\boldsymbol{e}$ is defined as:

$$\boldsymbol{e} = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ \cdots \\ e_{1,m} \\ \cdots \\ e_{p,1} \\ e_{p,2} \\ \cdots \\ e_{p,m} \end{bmatrix} \tag{51}$$

If we look back at the Hessian Matrix and substitute the sum squared error (SSE):

$$E(w,x) = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} e_{p,m}^2 \tag{52}$$

for the row i and column j entry we get:

$$\boldsymbol{h_{i,j}} = \frac{\boldsymbol{\partial^2 E}}{\boldsymbol{\partial w_i \partial w_j}} = \frac{\boldsymbol{\partial^2}\left(\frac{\mathbf{1}}{\mathbf{2}}\sum_{\boldsymbol{p=1}}^{\boldsymbol{P}}\sum_{\boldsymbol{m=1}}^{\boldsymbol{M}} \boldsymbol{e_{p,m}^2}\right)}{\boldsymbol{\partial w_i \partial w_j}} = \sum_{\boldsymbol{p=1}}^{\boldsymbol{P}}\sum_{\boldsymbol{m=1}}^{\boldsymbol{M}} \frac{\boldsymbol{\partial e_{p,m}}}{\boldsymbol{\partial w_i}}\frac{\boldsymbol{\partial e_{p,m}}}{\boldsymbol{\partial w_j}} + \boldsymbol{S_{i,j}} \tag{53}$$

where $\boldsymbol{S_{i,j}}$ is:

$$\boldsymbol{S_{i,j}} = \sum_{\boldsymbol{p=1}}^{\boldsymbol{P}}\sum_{\boldsymbol{m=1}}^{\boldsymbol{M}} \frac{\boldsymbol{\partial^2 e_{p,m}}}{\boldsymbol{\partial w_i \partial w_j}}\boldsymbol{e_{p,m}} \tag{54}$$

It is assumed that $\boldsymbol{S_{i,j}}$ is close to zero, with this assumption the Hessian matrix can now be written as:

$$\boldsymbol{H} = \boldsymbol{J^T J} \tag{55}$$

The Gauss-Newton weight update equation is:

$$\boldsymbol{w_{k+1}} = \boldsymbol{w_k} - \left(\boldsymbol{J_k^T J_k}\right)^{-1}\boldsymbol{J_k^T e_k} \tag{56}$$

The question now is how to calculate the Jacobian matrix. We must find

$$\frac{\boldsymbol{\partial e_{p,m}}}{\boldsymbol{\partial w_n}} \tag{57}$$

which is the partial derivative of the error at each output for each pattern with respect to the partial derivative of each weight. The Jacobian matrix calculation is found by performing the following steps:

1.) Forward computation to determine the net, output, and slope for each neuron.
2.) Determine $\boldsymbol{\delta}$ for each output: $\boldsymbol{\delta_j} = \frac{\partial f_j(net_j)}{\partial net_j} = \boldsymbol{out_j}(\mathbf{1} - \boldsymbol{out_j})$ = slope
3.) Propagate $\boldsymbol{\delta}$ to the previous layer neurons by multiplying the output $\boldsymbol{\delta}$ by the weight between the previous neuron and the output neuron.
4.) Propagate $\boldsymbol{\delta}$ to the input of the neuron by multiplying by the slope of the neuron.
5.) Continue for each layer
6.) Determine each element of the Jacobian matrix by multiplying : $\frac{\partial e_{p,m}}{\partial w_n} = -\boldsymbol{\delta_{m,j}} \boldsymbol{y_{j,i}}$

The Gauss-Newton algorithm is implemented by the Python code shown in Appendix C. A typical output is shown in figure 32.

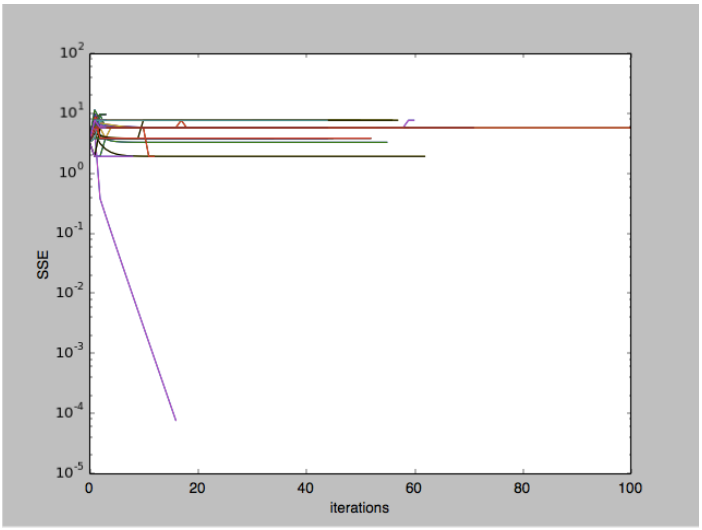


Figure 32. Gauss-Newton algorithm, 300 tries and 100 iterations – parity 3 problem

As seen in figure 32, the success rate is really small, only one out of 300 tries for this run. The initial guess has to be very close to the final solution for the Gauss-Newton algorithm to

converge. It also shows that when there is convergence, it happens very quickly (i.e. very few iterations).

5.2.3 Levenberg-Marquardt (LM) Algorithm

From the above discussions, it seems an ideal algorithm would combine the fast convergence of Gauss-Newton and the high convergence rate of Error Back Propagation. Kenneth Levenberg in 1944 and Donald Marquardt in 1966 both independently developed an algorithm that does combine the best of GN and EBP. It is called the Levenberg-Marquardt (LM) algorithm and is generally regarded as one of the most efficient training algorithms [60] .

The LM algorithm adds a factor (μ) to the Gauss-Newton delta weight equation that, depending on the magnitude of the factor, causes the algorithm to either resemble the Gauss-Newton or a version of the gradient decent method. The LM equation is:

$$\boldsymbol{w_{k+1} = w_k - \left(J_k^T J_k + uI\right)^{-1} J_k^T e_k} \tag{58}$$

Where $\boldsymbol{I}$ is the identity matrix. If μ is small then the LM algorithm becomes:

$$\boldsymbol{w_{k+1} = w_k - \left(J_k^T J_k\right)^{-1} J_k^T e_k} \tag{59}$$

The Gauss-Newton algorithm, but if $\boldsymbol{\mu}$ is large then the LM equation becomes:

$$\boldsymbol{w_{k+1} = w_k - \propto J_k^T e_k} \text{ with } \propto = \frac{\boldsymbol{1}}{\boldsymbol{\mu}} \tag{60}$$

The gradient decent equation. The LM algorithm begins by giving $\boldsymbol{\mu}$ a small value (i.e. 0.01).

The next steps are as follows:

1.) Randomly generate the initial weights and determine the total error (SSE).
2.) Compute the jacobian matrix and update weights using equation X.
3.) Re-evaluate the total error
4.) If the new error is greater then the previous error, then reset the weights to the previous set and increase $\boldsymbol{\mu}$ by a factor of 10. If the new error is less then the previous error, then the new set of weights is kept and $\boldsymbol{\mu}$ is decreased by a factor of 10.

5.) Continue to step 2 until the total error is less than the specified amount.

The LM algorithm is implemented in the Python code found in Appendix D:

A typical output is shown in figure 33, which shows a high success rate for convergence.

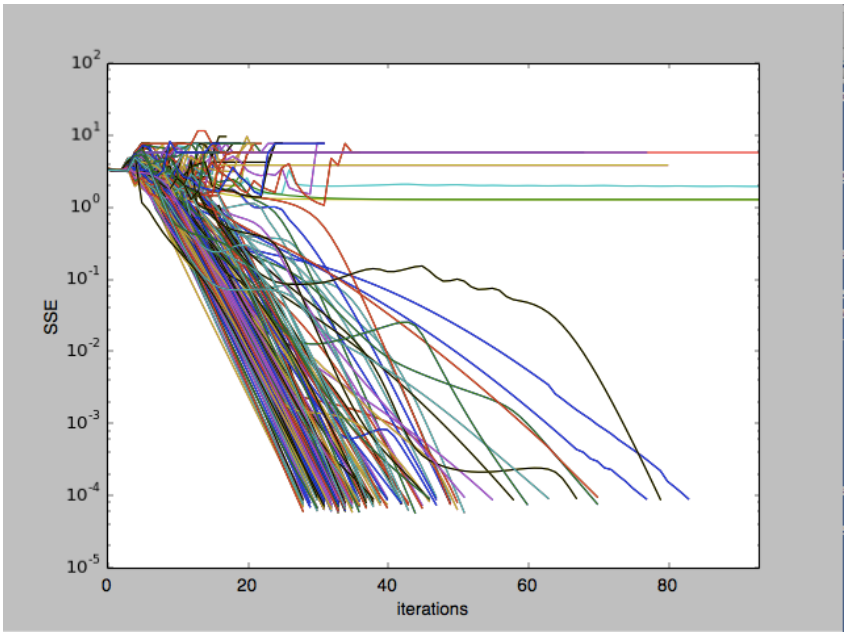


Figure 33. LM algorithm, 200 iterations, parity 3 problem

5.2.4 Neuron by Neuron (NBN) Algorithm

So far all of the neural network algorithms, first and second order, have been capable of utilizing only Multilevel Perceptron (MLP) topologies. It has been shown that arbitrarily connected topologies, such as fully connected cascade (FCC) networks, are more powerful and are capable of convergence with fewer neurons. But the training of such networks will require a new algorithm that is more powerful and able to calculate the jacobian matrix neuron by neuron instead of layer by layer. Such a training algorithm was developed, that utilizes the LM

algorithm to update weights, along with a neuron by neuron determination of the jacobian matrix. It is known as the Neuron by Neuron (NBN) algorithm.

An FCC network is shown in figure 34.

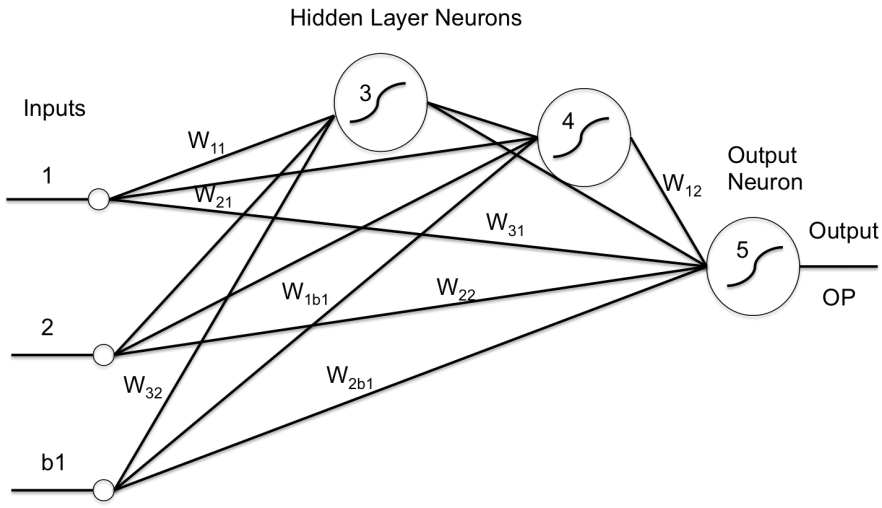


Figure 34. FCC Neural Network Topology

The procedure begins with a forward computation of the nets, outputs, and slopes for each neuron, followed by a back propagation of the delta parameter for each neuron. The next step is to determine the Jacobian row for each pattern. After all patterns have been applied the entire Jacobian matrix is found.

The pseudo code for the basic NBN algorithm is shown in figure 35.

```
for all patterns (np)
% Forward computation
        for all neurons (nn)
                for all weights of the neuron (nx)
                        calculate net;
                end;
                calculate neuron output; % Eq. (3)
                calculate neuron slope; % Eq. (6)
        end;
        for all outputs (no)
                calculate error; % Eq. (2)
%Backward computation
                initial delta as slope;
                for all neurons starting from output neurons (nn)
                        for the weights connected to other neurons (ny)
                                multiply delta through weights
                                sum the backpropagated delta at proper nodes
                        end;
                        multiply delta by slope (for hidden neurons);
                end;
                related Jacobian row computation; %Eq. (12)
        end;
  end;
```

Figure 35. NBN Algorithm Pseudocode

The output is shown in figure 36.

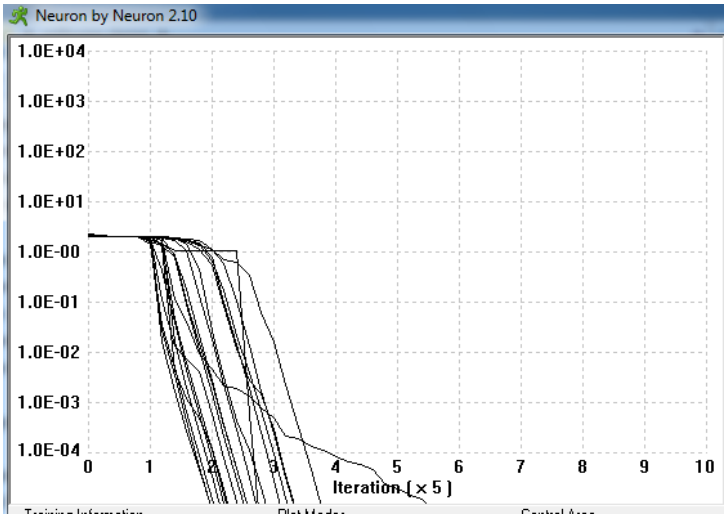


Figure 36. NBN output, parity-3 problem

Notice the mention above of the “basic” NBN algorithm, this is considered the basic NBN since there have been a couple of improvements including a forward only. In this work the basic NBN algorithm is presented.

Chapter 6. Proposed Plating Chemistry Component Determination Method

6.1 Proposed Additive Control Methods

After an examination of the techniques used in the standard analysis methods it seems that by gaining knowledge of the entire system a predictive analysis could be utilized to determine the status of the system. So, if a method could be found by which a set of independent variables could be determined, which are related to the amounts of each additive, then machine learning could be used to provide the capability of predicting the additive amounts.

In an effort to generate a set of independent variables, CVS stripping area response plots for the various additives vs. concentration of the additives were investigated to determine if there are certain conditions that would favor one additive over another. Figure 37 shows a plot of stripping area vs. suppressor concentration [61] for various electrode and test parameter configurations.

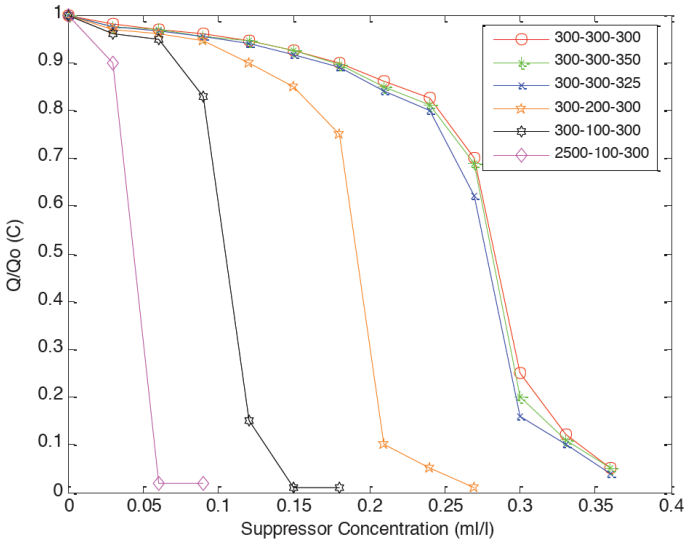


Figure 37. Stripping Charge vs Suppressor Concentration [61]

(Legend: - Rotation speed (RPM) - Scan Rate (V/Sec) - Negative Limit (V))

From the plots in figure 18 it is shown that there is a reduced sensitivity to concentration as the working CVS electrode rotation speed is reduced, and as the scan rate is increased. Both trends make account for the fact that the reactivity of the suppressor is concentration dependent and the concentration is related to the barrier layer thickness (determined by the spin speed of the electrode) that is controlled by mass transport and absorption properties. The barrier layer is thinner as the rotation speed is increased and thus the absorption ability is lowered as the scan rate is increased.

Figure 38 shows a plot of stripping area vs. leveler concentration [61].

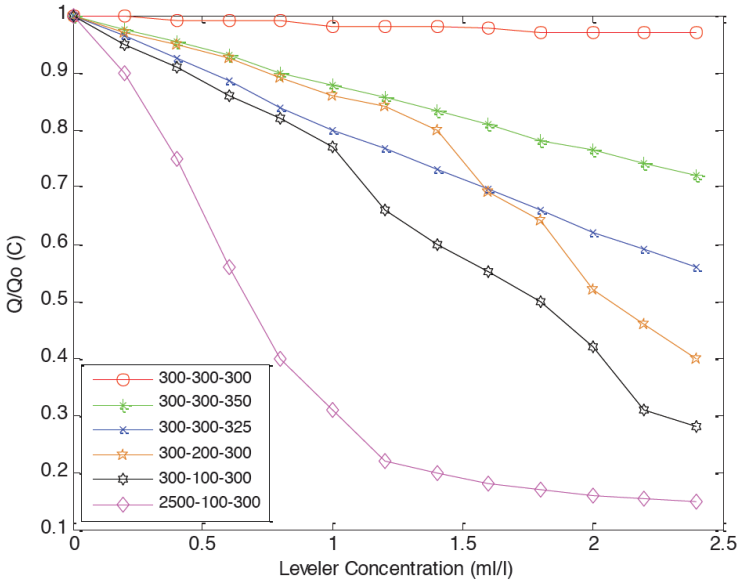


Figure 38. Normalized Charge vs. Leveler Concentration [61]
(Legend: - Rotation speed (RPM) - Scan Rate (V/Sec) - Negative Limit (V))

Figure 38 shows that as the scan rate is increased, the stripping area becomes less sensitive to the leveler concentration. Leveler is a large polar molecule and a slow diffuser. Accelerator is a fast diffuser and thus does not exhibit as large a sensitivity to the barrier layer thickness as the larger slow diffusing suppressor and leveler molecules.

With the above information it was concluded that a set of tests, which included large differences in electrode rotational spin rate and voltage scan rate, would favor or disfavor the various additives and provide a set of stripping areas that would independently define a certain set of additives. Also, once this set of stripping areas is generated, it could be used as a calibration set for determining an amount of additives in an unknown sample. It was further expected that the calibration set of stripping areas could be used to train learning machine systems to estimate the amount of additives in an unknown sample.

The following four CVS analysis methods shown in Table 4 were used to generate a set of stripping curves using a Metrohm CVS system.

Table 4. CVS Test Parameters

CVS Test #	Spin Speed (rpm)	Scan Rate (V/s)
1	800	0.05
2	800	0.5
3	3000	0.05
4	3000	0.5

While there may be many other combinations of tests that would be able to independently characterize the plating additives, this set of tests is well within the capabilities of most available test systems and has proven to give repeatable results after many calibration runs. The intent of this work is to prove that machine learning can predict additive concentration, not to select the optimum set of tests. The selection of an optimum set of tests is a good candidate for future work.

A set of data was generated using the methods described in the previous section. It will be used as a training set to evaluate various machine learning techniques to determine if it is a viable option for additive prediction from unknown plating bath samples. The data typically needs to be scaled to allow it to be used as input for neural network systems. The data will first be normalized to set the magnitude of the input data to the same order as the initial random weights. A subset of the dataset used for this work is shown in table 5. It can be seen that the magnitudes of the values are well above the ranges needed for neural network training (normally between -1 & 1). The data is normalized by dividing each data point by the value with the highest magnitude in each column, thereby normalizing all of the values to numbers between 0 and 1.

Table 5. Typical Measured Plating Bath Data before Normalization

Test1	Test2	Test3	Test4	A	S	L
2390.0	353.5	1540.0	274.7	7.000	3.000	2.000
1680.0	314.1	1130.0	208.1	7.000	3.000	3.800
1150.0	681.7	763.7	275.4	1.000	1.200	4.000
873.5	462.2	750.1	224.7	1.000	1.800	4.000
794.8	350.9	740.5	195.0	1.000	2.400	4.000
767.0	290.8	731.8	175.1	1.000	3.000	4.000
759.6	252.7	724.4	161.6	1.000	3.600	4.000
749.9	228.3	718.5	150.9	1.000	4.200	4.000
740.4	210.0	712.2	143.1	1.000	4.800	4.000

The dataset will be divided into three sets of data (four inputs and one output) one for each of the additives (i.e. accelerator, suppressor, and leveler). This will allow the use of machine learning techniques that are only capable of learning with one output (i.e. Support Vector Regression and Polynomial Regression). Table 6 shows normalized data (same set of data from Table 5).

Table 6. Typical Measured Plating Bath Data After Normalization

Test1	Test2	Test3	Test4	A	S	L
0.996	0.147	1.000	0.415	0.778	0.500	0.100
0.700	0.131	0.734	0.314	0.778	0.500	0.190
0.479	0.284	0.496	0.416	0.111	0.200	0.200
0.364	0.193	0.487	0.339	0.111	0.300	0.200
0.331	0.146	0.481	0.295	0.111	0.400	0.200
0.320	0.121	0.475	0.265	0.111	0.500	0.200
0.317	0.105	0.470	0.244	0.111	0.600	0.200
0.312	0.095	0.467	0.228	0.111	0.700	0.200
0.309	0.088	0.462	0.216	0.111	0.800	0.200

6.2 Machine Learning Software Tools

In this work we will investigate different machine learning techniques to determine which one provides the most accurate predictions for this application:

- Neural Networks (NN) – Traditional Neural Network architectures (MLP) using Error Back Propagation.
- Polynomial Regression.
- Support Vector Regression (SVR) - Support Vector Regression.
- Extreme Learning Machine (ELM).
- Advanced Neural Network Algorithms / Architectures – Fully Connected Cascade (FCC) using Neuron by Neuron algorithm.

For traditional and advanced Neural Network evaluation the neural networks trainer (NBN 2.10) was used, this software can be downloaded at: http://www.eng.auburn.edu/users/wilambm/nnt/.

For Support Vector Regression, Rapid-Miner Software was used, this software can be downloaded at: http://rapid-i.com/content/view/181/190/

For Extreme Learning Machines and Polynomial Regression dedicated custom software was used.

The training data was generated using the "new plating chemistry analysis methods" described in chapter 5. A dataset of 77 vectors was generated over a typical range of Accelerator, Suppressor, and Leveler additive concentrations. The following sections will describe the initial review of the dataset and analysis of the resulting data along with validations for each training run.

6.3 Accelerator Determination

Seven of the vectors were randomly removed from the 77 vector dataset to be used as a validation set, leaving a set of 70 vectors for training. The 70 vector dataset was used as the training dataset for the following algorithms and architectures.

6.3.1 Accelerator Determination using Polynomial Regression

A polynomial regression model with orders between 2 & 10 was used at first. Cross multiplication of the input variables was not utilized to simplify the software needed and to allow the sample dataset to be a reasonable size. Dedicated software was written to perform the linear least squares approach to estimate the coefficients of the polynomial.

This method computes the polynomial coefficients for various order polynomials during accelerator training and used these coefficients to evaluate the accelerator validation set for the different polynomial orders. The results are shown in figure 39. As can be seen that the polynomial regression approach more closely fits the training data as the order increased. As expected, if the order is too high (larger than 7 in this case), the over-fitting phenomenon occurs.

Seven appears to be the optimal order for this dataset and gave a very reasonable fit to the validation data of 0.015 RMSE.

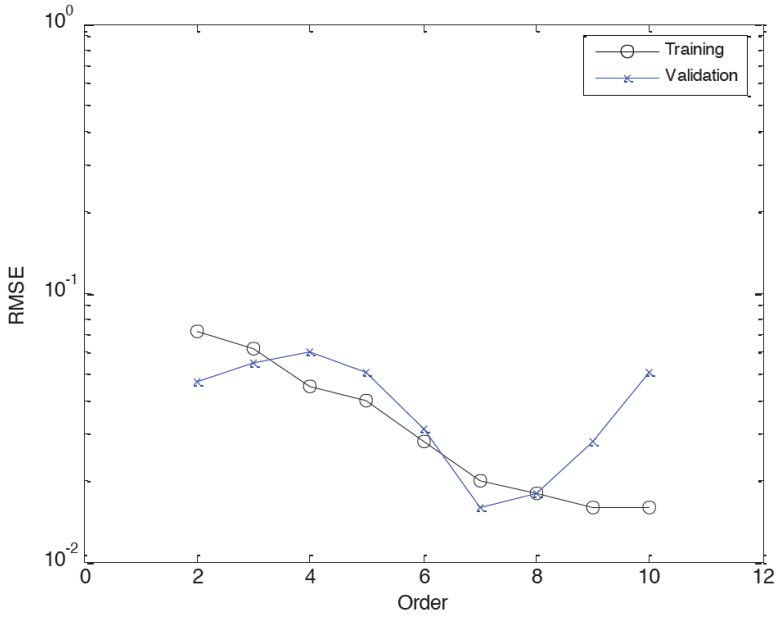


Figure 39. Polynomial Regression Results for Accelerator

6.3.2 Accelerator Determination using **Error back Propagation**

The standard Error Back Propagation (EBP) algorithm was utilized on the training set using a Multi-layer Perceptron (MLP) neuron topology with a unipolar sigmoid activation function (gain of 1 and learning constant of 0.1). The MLP topology is shown in figure 32.

The previously described training and validation datasets are used to train and test generalization for the MLP topology with an increasing number of neurons in the input layer. Results from training and validation, using the NBN 2.10 program, are shown in figure 40.

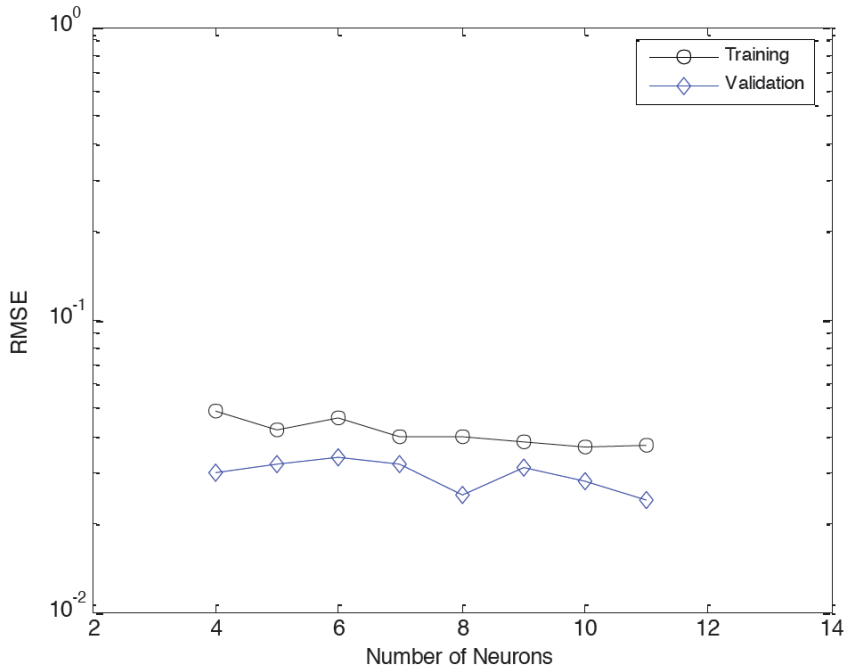


Figure 40. Training and Validation results for the Multilayer Preceptron Topology and EBP Training.

A significant amount of time was spent on trying to optimize the gain, learning constant and other factors to obtain a better convergence. It appears that the validation RMSE is lower than the training RMSE for all neuron numbers. This was not expected, but seems to be the case through all training & validation runs. A possible reason for this is that the data is highly non-linear and more time and effort will be required to optimize the network, thus a reason why this algorithm results in researchers becoming disillusioned with neural networks.

6.3.3 Accelerator Determination using Support Vector Regression (SVR)

The same training set defined above was used for training using SVR. Typically the best method for determining the optimum set of support vectors is to vary the Soft Margin Parameter (C) and validate. A plot of the validation Root Mean Squared Error (RMSE) vs. the soft margin parameter is shown in figure 41. This data shows a general trend downward with increasing C

for both the training and validation data. The validation seems to level out at a RMSE of approximately 0.025 starting around a C value of 80.

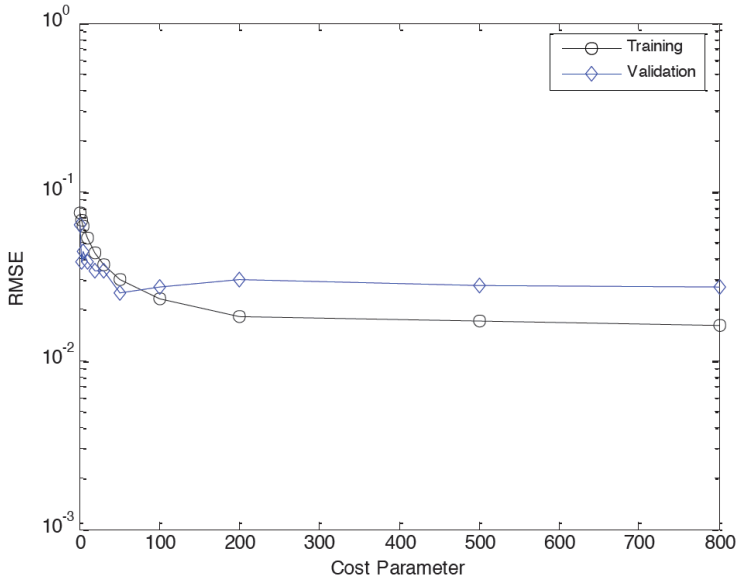


Figure 41. Accelerator Training & Validation using Support Vector Regression

6.3.4 Accelerator Determination using Extreme Learning Machine (ELM)

The extreme learning machine technique has been recently mentioned as a new and novel technique that has the capability to converge very quickly. It is considered an incremental learning algorithm, that is thought to provide universal approximation capability. It uses an algorithm that incrementally constructs a single hidden layer feedforward network with randomly generated activation parameters (i.e. radius, center of RBF, input weight, bias for other functions). The optimum output weights are determined by a least squares minimization. The previously defined dataset was used in experimentation. The ELM algorithm was trained with between 50 and 500 RBF units and 20 training runs. The RMSE results for three different versions of the ELM algorithm are shown in figure 42.

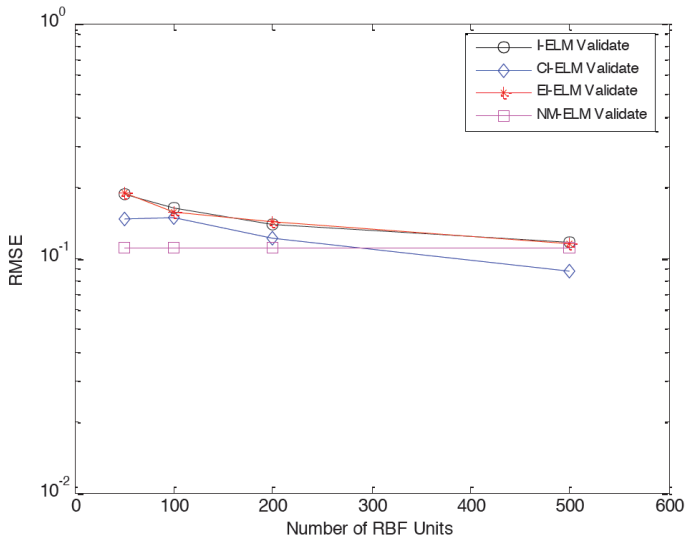


Figure 42. Accelerator Training & Validation using Extreme Learning Machine

An error rate as low as the previous three methods, even after increasing the number of RBF units to as high as 500, was not obtainable. As with EBP, the high non-linearity of the data may cause this algorithm to not converge to a low error rate.

6.3.5 Accelerator Determination using Second Order Algorithms

The advanced second order neural network learning algorithm, neuron by neuron, allows arbitrarily connected topologies to be used. The Fully Connected Cascade (FCC) is one of the most powerful arbitrarily connected topologies for NN training. This topology consists of a series of neurons all fully connected to all previous neurons in a cascade arrangement as shown in figure 43.

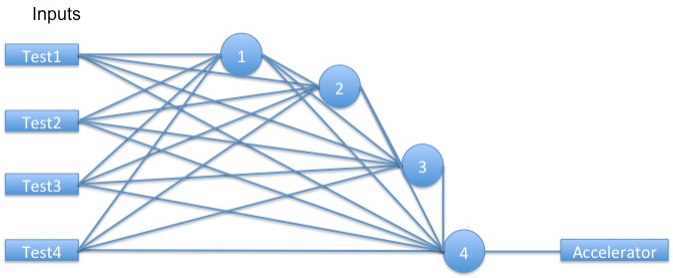


Figure 43. Fully Connected Cascade Topology

The NBN algorithm was used to train MLP and FCC topology networks, using the same training and validation files as was used in the experiments with traditional NNs. The results are shown in figures 44 and 45 respectfully. In figure 44, the MLP topology, it can be seen that there is a very low (as low as 2e-5) root mean square error (RMSE) and a corresponding low validation error as low as 0.001 RMSE) – showing an ability to accurately predict the accelerator concentration within a few percent difference (<4 %).

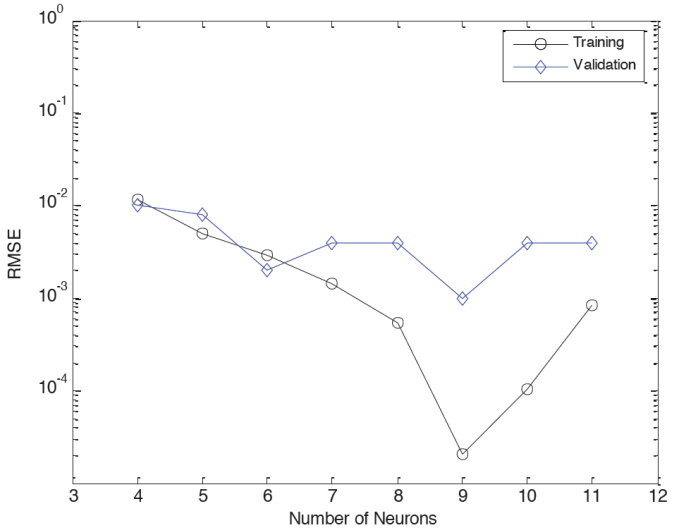


Figure 44. Accelerator Training and Validation results for an MLP Topology with NBN algorithm

Figure 45 shows the training and validation for the FCC topology and the NBN algorithm. It shows a low (1e-5) RMSE for the training and a 1e-3 RMSE for validation, showing that this combination of topology and algorithm can be used to accurately predict the accelerator concentration (within a small percent difference). It should be noticed that overtraining (the validation error begins to increase) is apparent when more than 8 neurons are used. The MLP topology shows similar results.

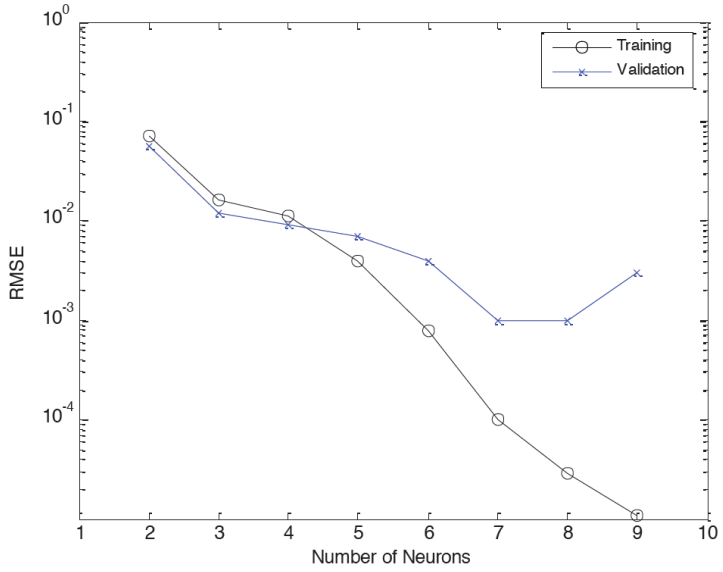


Figure 45. Accelerator Training and Validation results for an FCC Topology with NBN algorithm

6.3.6 Summary of Accelerator Determination Results

The validation results for the tested learning machine techniques are summarized in Table 7. It shows the lowest values for accelerator predictions using the percent difference for the 7 validation vectors. It can be seen that the NBN MLP & FCC algorithm/topology provided the best generalization. While the Poly, SVM, NN/EBP, and ELM techniques gave similar results

that are ~ an order of magnitude worse that the NBN, but still reasonable for this application, although not giving the increase in accuracy desired.

Table 7. Accelerator prediction comparison of machine learning techniques

Predicted Accelerator Values for Different Algorithms/Topologies							
Validate Point #	Actual ml/l	Poly	SVM	ELM	EBP/MLP	NBN/MLP	NBN/FCC
1	2	2.25	2.14	2.23	2.10	2.03	2.01
2	5	5.11	5.04	6.09	5.17	5.02	4.99
3	2	1.91	1.91	1.97	1.71	2.00	2.00
4	9	8.96	9.02	8.92	9.09	9.00	9.00
5	7	6.91	7.56	7.04	7.25	7.00	7.03
6	9	9.12	8.89	8.56	8.84	9.00	8.99
7	2.9	3.08	2.67	2.62	2.51	2.90	2.90

Now that the accelerator has been shown to be predictable using machine learning techniques, in particular the NBN algorithm, an attempt to use the NBN method for predicting the other two additives, suppressor and leveler, will be investigated. Only the validation data is shown in the following analysis (no training data). The neural network NBN algorithm will be used since it has shown to be the most accurate predictor from the previous results. The polynomial regression analysis will also be investigated as a comparison.

6.4 Suppressor Determination

The same training and verification dataset will be used with the accelerator output replaced by the suppressor output. The NBN/FCC and NBN/MLP algorithms/topologies will be used to determine how well the suppressor can be predicted. The addition of the predetermined accelerator value as a fifth input variable will be investigated.

Figures 46 – 48 show the validation results for Polynomial, MLP/NBN, and FCC/NBN learning. These figures compare the 4 inputs along with the inclusion of accelerator as the fifth input variable. Table 8 compares the results from the various NBN topologies and with adding accelerator as an additional input (SA). It is seen that all of the NN techniques gave good results while using accelerator as the fifth input improved the accuracy in all cases. Polynomial regression was used to compare a non-NN learning technique.

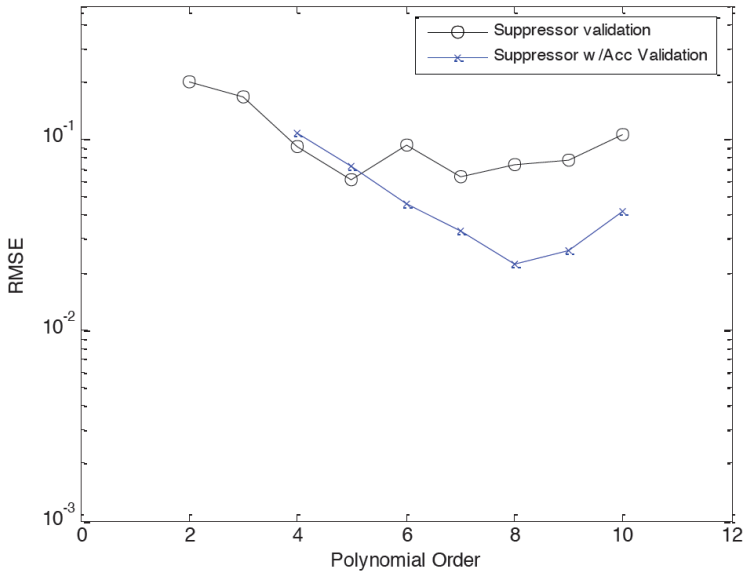


Figure 46. Suppressor Polynomial Validation

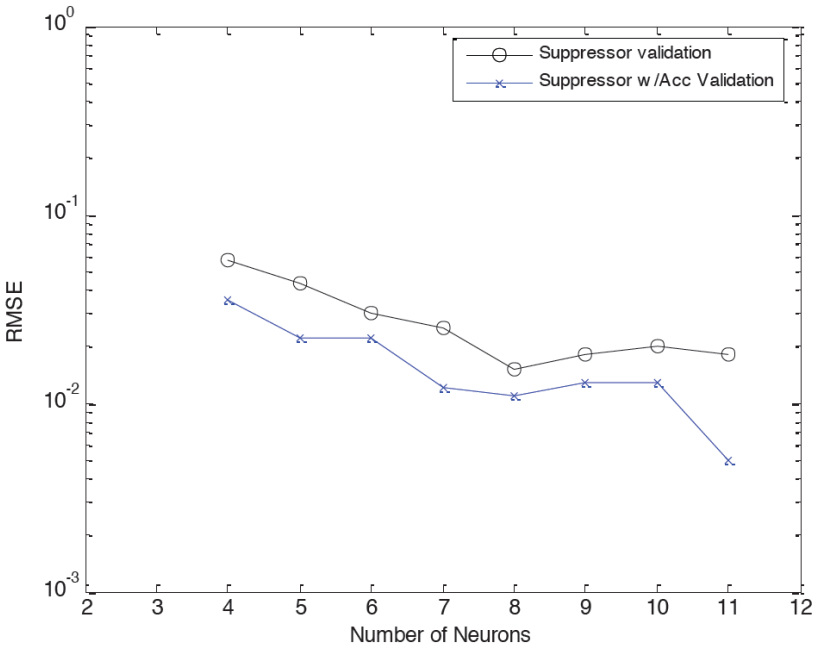


Figure 47. Suppressor MLP/NBN Validation

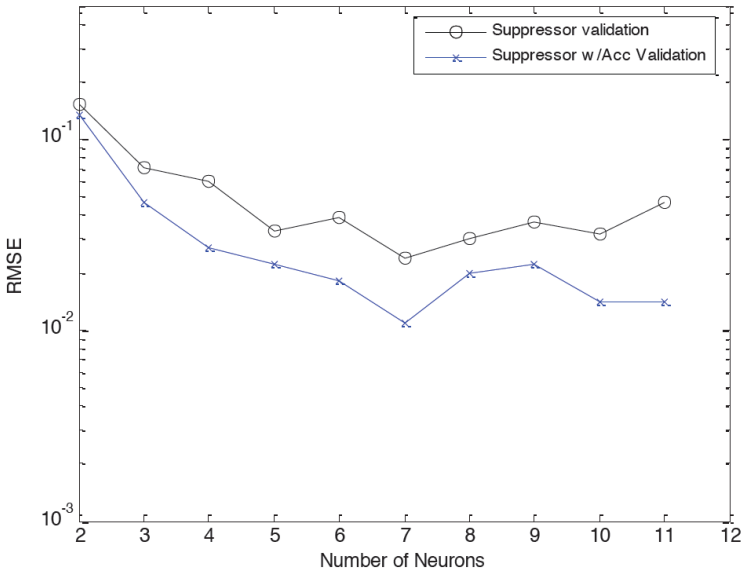


Figure 48. Suppressor FCC/NBN Validation

Table 8. Suppressor prediction comparison of machine learning techniques

Predicted Suppressor Values for Different Algorithms/Topologies							
Validate Point#	Actual ml/l	S NBN/ MLP	SA NBN/ MLP	S NBN/ FCC	SA NBN/ FCC	S Poly	SA Poly
1	6	5.98	5.89	5.79	5.86	5.77	5.80
2	3.6	3.58	3.66	3.94	3.61	3.56	3.85
3	6	5.84	5.85	5.77	5.84	5.09	6.14
4	3.6	3.64	3.57	3.56	3.58	3.40	3.57
5	6	6.00	5.98	6.00	6.00	6.20	6.02
6	4.8	4.49	4.82	4.66	4.73	4.63	4.86
7	3	3.00	3.00	3.00	3.00	3.02	2.99

6.5 Leveler Determination

The Leveler training verification data is shown in figures 49-51 for both leveler validation RMSE for original four test inputs and for adding the accelerator as a fifth input variable. Table 9 shows the percent difference between the predicted and actual leveler amounts for the different algorithms/topologies. The “L” and “AL” designations mean Leveler or Accelerator / Leveler inputs respectively. It shows that all of the NN techniques gave very good predictions. The addition of accelerator as a fifth input showed a slight improvement.

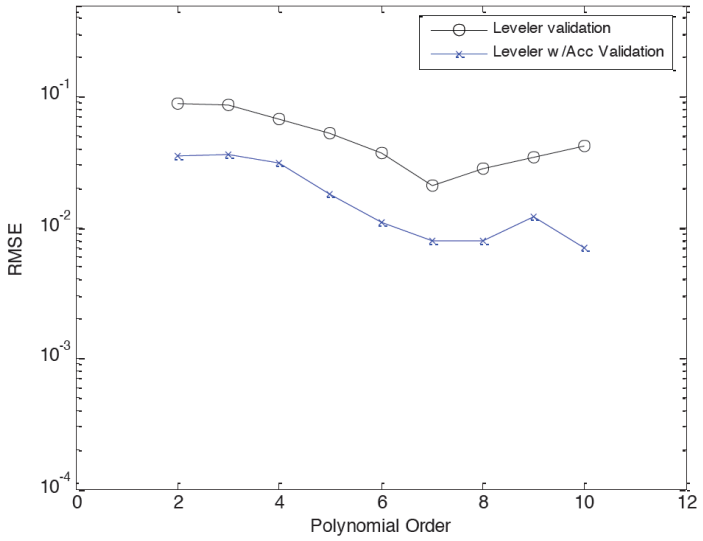


Figure 49. Leveler Polynomial Validation for 4 & 5 Inputs

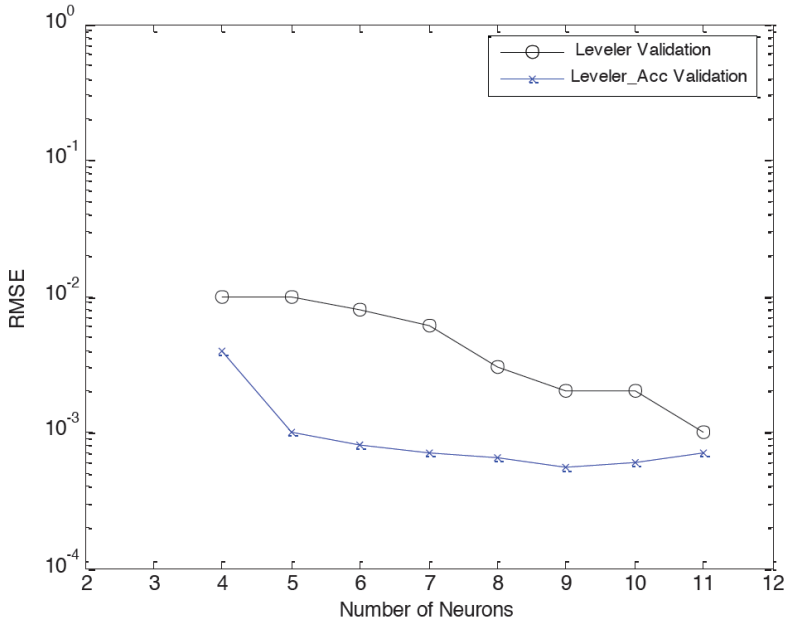


Figure 50. Leveler MLP/NBN Validation for 4 & 5 Inputs

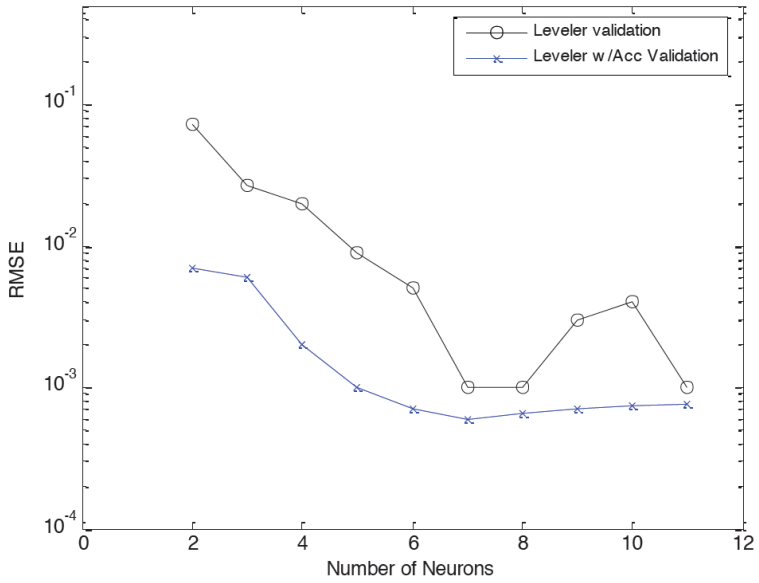


Figure 51. Leveler FCC/NBN Validation for 4 & 5 Inputs

Table 9. Leveler prediction comparison of machine learning techniques

Predicted Leveler Values for Different Algorithms/Topologies							
Vaildate Point#	Actual ml/l	L NBN/ MLP	AL NBN/ MLP	L NBN/ FCC	AL NBN/ FCC	L Poly	AL Poly
1	10	9.85	9.95	10.01	10.00	10.67	10.18
2	5	4.94	4.99	4.95	4.99	4.98	4.70
3	15	14.97	14.99	14.96	15.00	14.32	14.92
4	10	10.00	10.00	9.98	10.00	10.03	9.91
5	15	14.99	15.00	15.00	15.00	14.83	14.99
6	20	20.00	20.00	20.00	20.00	20.40	20.06
7	15	15.00	15.00	15.00	15.00	15.30	15.00

Conclusion

The present methods for analyzing copper plating baths were shown to be very wasteful and inaccurate. These methods are presently implemented using very expensive chemical analysis systems sold by several vendors. These systems normally cost on the order of 100 – 200K and are about the size of a refrigerator. They waste a considerable amount of plating chemistry and need a very experienced operator to understand and maintain them.

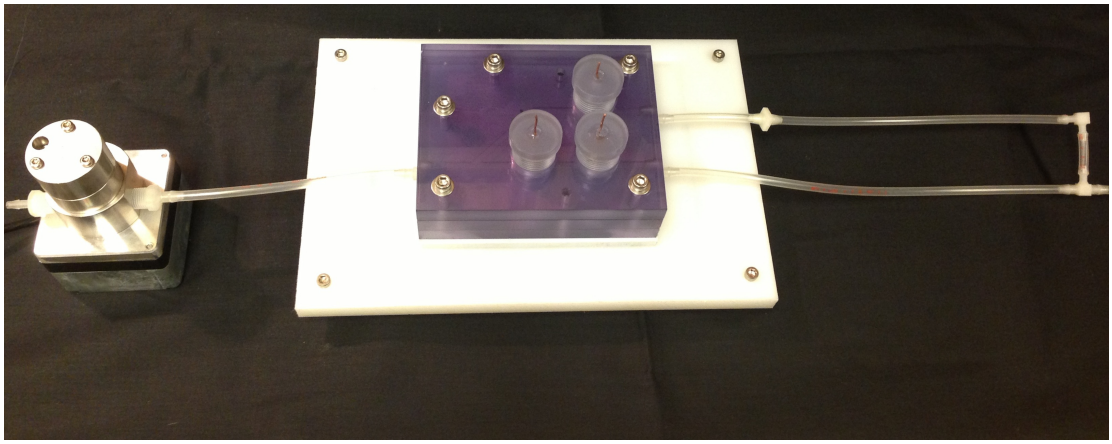
It was thought that it would be beneficial to determine if there is a better and simpler method that will be less wasteful, more accurate and easier to maintain. A decision was made to investigate the use of new algorithms that were under development at Auburn University by Dr. Bogdan Wilamowski's Neural Network Group. These algorithms promised to be much more powerful and easier to use than conventional algorithms.

The next issue was to determine if a set of data could be gathered that would allow the determination of the three different plating organic components using the new algorithms. Through an exhaustive search of the literature and with years of experience in copper plating and CVS analysis, it was decided to the use of set of CVS parameters that generated four charge magnitudes. This set of data, based on these charge magnitudes, were shown to be able to very accurately determine the amounts of accelerator, suppressor, and leveler contained in a copper plating bath using second order neural networks.

It is believed that this method can be easily added to a production copper plating system and will provide a high level of control over the additives, and it can be further adapted to determine decomposition byproducts and other plating components of interest.

Future Work

The obvious next steps would work toward putting this new plating component analysis method into practice. A system would need to be developed that would automate the analysis procedures and allow it to be integrated with a commercial plating bath system. During this work a system was fabricated and tested that should be capable of automating the plating bath testing phase [62]. It is shown in figure 52.



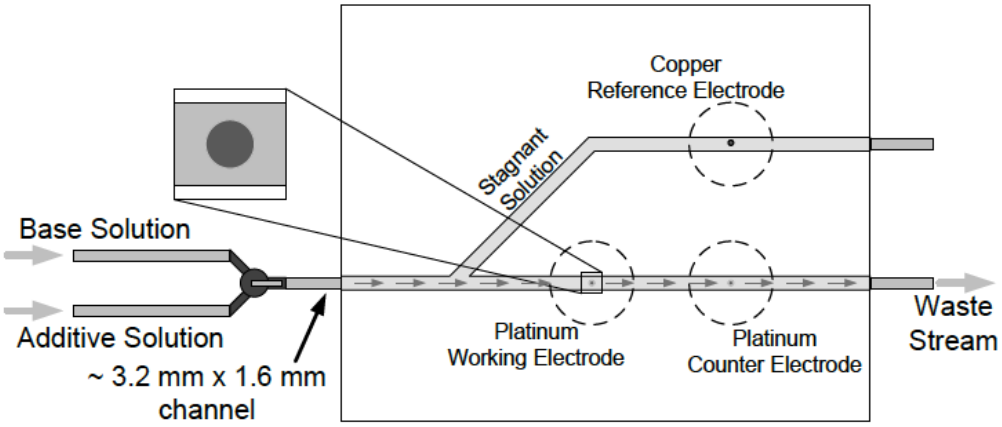


Figure 52. Automating Plating Bath Analysis System

The system contains a working electrode, counter electrode, and reference electrode. The plating solution would be continuously flowing through the system with measurements made periodically to determine the plating bath components. The four measurements required to generate the data needed for the neural network inputs requires two different working electrode spin speeds. This is not available with the system shown, but an alternative method is to change the pumping speed of the solution as this would perform the same function of changing the plating cathode barrier layer. Basically all that is needed is to provide fresh solution at different rates. A flow rate vs. motor voltage is shown in figure 53 for the system shown in figure 52.

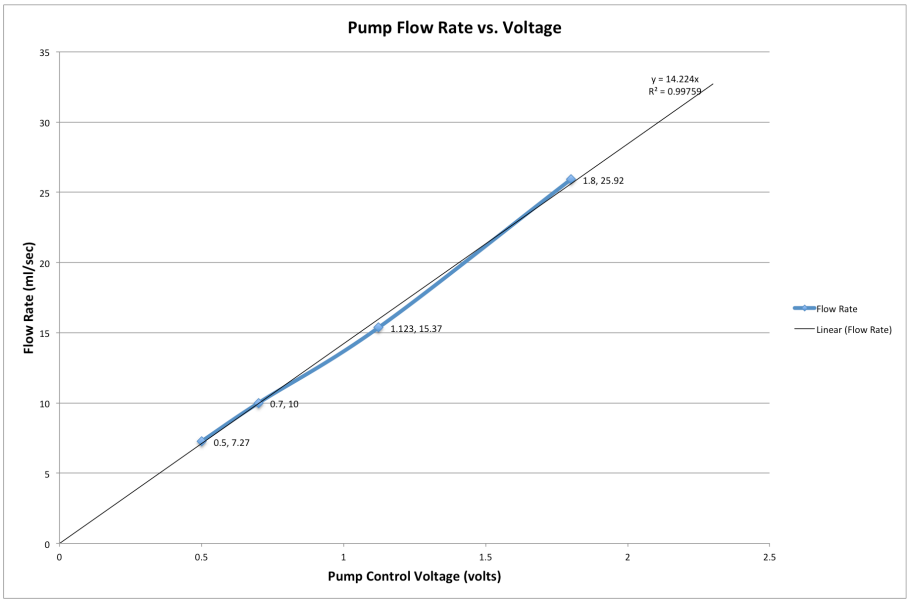


Figure 53. Flow vs. Pump Voltage

Other areas for future research could be looking at the analysis of breakdown components in the plating solution. These breakdown components are important to the makeup of the bath and to determine the end of life, or when the bath needs to be completely changed.

References

[1] P. C. Andricacos, C. Uzoh, J. O. Dukovic, and J. Horkans, "Damascene Copper Electroplating for Chip Interconnections," *IBM J Res Dev*, vol. 42, pp. 567–574, 1998.

[2] R. A. Binstead, R. Mikkola, and J. M. Calvert, "Fundamental Mechanism Controlling Copper Electrodeposition. The Roles of Organic and Inorganic Components in Promoting Superconformal Filling and Self-Leveling of Inlaid Submicron Interconnects," in *Fundamental Challenges in Electrodeposition*, San Francisco, 2003.

[3] M. Yokoi, "Supression Effect and Additive Chemistry," in *Copper Electrodeposition for Nanofabrication of Electronics Devices*, New York: Springer Science+Business Media, pp. 27–43. 2014.

[4] P. M. Vereecken, R. A. Binstead, H. Deligianni, and P. C. Andricacos, "The chemistry of additives in damascene copper plating," *IBM J Res Dev*, vol. 49, no. 1, pp. 3–18, Jan. 2005.

[5] A. C. West, "Theory of Filling High-Aspect Ratio Trenches and Vias in Presence of Additives," *J Electrochem Soc*, vol. 147, pp. 227–232, 2000.

[6] M. H. Lee and J. K. Cho, "A study on the Additive Decomposition Generated during the Via-Filling Process," *J. Korean Inst. Surf. Eng.*, vol. 46, no. 4, pp. 153–157, Aug. 2013.

[7] C. D. Ellis, M. C. Hamilton, J. R. Nakamura, and B. M. Wilamowski, "Efficient Determination of Copper Electroplating Chemistry Additives," *IEEE Trans. Compon. Packag. Manuf. Technol.*, vol. 4, no. 8, pp. 1380–1390, Aug. 2014.

[8] J. Reid, "Damascene Copper Electroplating," in *Handbook of semiconductor manufacturing technology*, 2nd ed., vol. Chapter 16, Boca Raton: CRC Press, 2008.

[9] Ralf Willecke; Franz Faupel, "Diffusion of Gold and Silver in Bisphenol A Polycarbonate," *Am. Chem. Soc.*, vol. 30, pp. 567–573, 1997.

[10] B. Norsworthy, "Advances in Copper Plating Technology." Onboard Technology, Apr-2006.

[11] P. F. Green and L. L. Berger, "Effects of polyimide chemical structure and environment on the diffusivity of copper," *Thin Solid Films*, vol. 224, no. 2, pp. 209–216, Mar. 1993.

[12] K. Shimoto, K. Matsui, and K. Utsumi, "Cu/Photosensitive-BCB Thin-Film Multilayer Technology for High Performance Multichip Modules," *IEEE Trans Compon. Packag. Manuf Technol B Adv Packag.*, vol. 18, pp. 18–22, 1995.

[13] V. M. Ahmed, D. G. Berger, A. Kumar, and S. J. LaMaire, "Selective Plating Method for Forming Integral Vias on Wiring Layers," 5,209,817.

[14] "Thick Copper Pillar Bump Fabrication | Solid State Technology." .

[15] Y. Zhang, T. Richardson, S. Chung, C. Wang, and B. Kim, "Fast copper plating process for TSV fill," in *Microsystems, Packaging, Assembly and Circuits Technology, 2007. IMPACT 2007. International*, 2007, pp. 219–222.

[16] Qi Li; Huiqin Ling; Haiyong Cao; Zuyang Bian; Ming Li; Dali Mao, "Through silicon via filling by copper electroplating in acidic cupric methanesulfonate bath," in *Electronic Packaging Technology & High Density Packaging, 2009. ICEPT-HDP '09*, pp. 68–72, 2009.

[17] B. Banijamali, S. Ramalingam, K. Nagarajan, and R. Chaware, "Advanced reliability study of TSV interposers and interconnects for the 28nm technology FPGA", pp. 285–290. 2011.

[18] J. Chew, U. Mahajan, R. Bajaj, I. Mirshad, and R. Newcomb, “Characterization and optimization of a TSV CMP reveal process using a novel wafer inspection technique for detecting sub-monolayer surface contamination,”, pp. 1–6, 2013.
[19] Jacob, P., McDonald, J.F., “Predicting the Performance of a 3D Processor-Memory Chip Stack,” *IEEE Des. Test Comput.*, vol. 22, no. 6, pp. 540–547, November - December. 2005.
[20] P. Garrou, C. Bower, and P. Ramm, *Handbook of 3D Integration – Technology and Applications of 3D Integrated Circuits*. Weinheim, Germany: Wiley-VCH, 2008.
[21] R. Weerasekera, L.-R. Zheng, D. Pamanuwa, and H. Tenhunen, “Extending Systems-on-Chip to the Third Dimension: Performance, Cost and Technological Tradeoffs,” in *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pp. 212–219, 2007.
[22] E. Beyne and B. Swinnen, “3D System Integration Technologies,” in *Proceedings of IEEE International Conference on Integrated Circuit Design and Technology (ICICDT)* , 2007.
[23] “Three-dimensional integrated circuit.” [Online]. Available: http://en.wikipedia.org/wiki/Three-dimensional_integrated_circuit. [Accessed: 01-Feb-2015].
[24] H. Wang, R. G. Gordon, R. Alvis, and R. M. Ulfig, “Atomic Layer Deposition of Ruthenium Thin Films from an Amidinate Precursor,” *Chem. Vap. Depos.*, p. n/a–n/a, Dec. 2009.
[25] A. Redolfi, D. Velenis, S. Thangaraju, and P. Andricacos, “Implementation of an Industry Compliant, 5×50µm, Via-Middle TSV Technology on 300mm Wafers,” presented at the Electronic Components and Technology Conference, pp. 1384–1388, 2011.
[26] J. S. . Chiu, “The Use of Vacuum Pre-wetting as a Process Aid for TSV Filling,” presented at the 3-D Architectures for Semiconductor Integration and Packaging, San Francisco, CA, 2010.
[27] *Modern electroplating*. Hoboken, N.J.: Wiley, 2013.
[28] M. S. Chandrasekar and M. Pushpavanam, “Pulse and pulse reverse plating—Conceptual,advantages and applications” in *Electrochimica Acta*, vol. 53, pp. 3313–3322, 2008.
[29] Y. Lu, H. Cao, Q. Sun, H. Ling, M. Li, and J. Sun, “Investigation of competitive adsorption between accelerator and suppressor in TSV copper electroplating,” in *2012 13th International Conference on Electronic Packaging Technology and High Density Packaging (ICEPT-HDP)*, pp. 434–437, 2012.
[30] H. Ling, H. Cao, Y. Guo, H. Yu, M. Li, and D. Mao, “Influence of leveler concentration on copper electrodeposition for through silicon via filling,” in *International Conference on Electronic Packaging Technology High Density Packaging, 2009. ICEPT-HDP ’09*, pp. 860–862, 2009.
[31] J. Enloe, “Conversation with Atotech representative concerning Copper plating bath additives,” 29-Aug-2011.
[32] A. Keigler, Z. Liu, J. Chui, and J. Drexler, “Sematech 3D Equipment Challenges: 300mm Copper Plating,” presented at the International Sematech Meeting, 2008.
[33] H. Cao, X. Feng, Q. Sun, W. Luo, H. Ling, J. Sun, and M. Li, “Electrochemical analysis of cathode in TSV copper electroplating,” in *2012 2nd IEEE CPMT Symposium Japan*, pp. 1–4. , 2012.
[34] H. Shen, C. Uzoh, and T. Dinan, “Precise Chemistry Control Using Cyclic Stripping Voltammetry For Improved Through Silicon Via Fill.” Invensas Inc., 2014.

[35] M. West, R. McDonald, M. Anderson, S. Kingston, and R. Mui, “Controlling Copper Electrochemical Deposition (ECD),” presented at the Characterization and Metrology for VLSI Technology: 2003 International Conference, pp. 504–513, 2003.
[36] R. Gluzman, “Brightner Determination using Modified Linear Approximation Technique (MLAT),” presented at the 70Th Am. Electroplaters Soc. Tech. Conf., Indianapolis, IN, pp. 13–18, 1983.
[37] W. O. Freitag, C. Ogden, D. Berger, and J. White, in *Plating Surf. Fin.*, vol. 70, p. 55, 1983.
[38] Application Note V-184, “Determination of leveler in acid copper baths by response curve technique.” Metrohm, Inc.
[39] “Popular Machine Learning Methods,” vol. https://sites.google.com/site/mldmda/popular-methods, last accessed - February 2015.
[40] “Linear least squares,” *http://en.wikipedia.org/wiki/Linear_least_squares_%28mathematics%29*.
[41] G. Haung and L. Chen, “Convex incremental extreme learning machine,” *Neurocomputing*, vol. 70, pp. 3056–3062, 2007.
[42] G. Haung and L. Chen, “Enhanced random search based incremental extreme learning machine”, *Neurocomputing*, vol. 71, pp. 3460–3468, 2008.
[43] G. Haung, L. Chen, and C.-K. Siew, “Universal Approximation Using Incremental Constructive Feedforward Networks With Random Hidden Nodes,” *IEEE Trans. Neural Netw.*, vol. 17, no. 4, Jul. 2006.
[44] G. B. Huang, Q. Y. Zhu, and H. A. Barbi, “Classification ability of single hidden layer feedforward neural networks,” *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 799–801, 2000.
[45] C. R. Rao and S. K. Mitra, *Generalized Inverse of Matrices and its Applications*. New York, 1971.
[46] O. L. Mangasarian and D. R. Musicant, “Robust linear and support vector regression,” *IEEE Trans Pattern Anal. Mach Intell*, vol. 22, pp. 950–955, 2000.
[47] *Rapid Miner, http://rapid-i.com/content/view/181/190/, accessed February 2015*. .
[48] P. J. Werbos, “Back-propagation: Past and future,” in *Proceedings of the IEEE Int. Conf. Neural Netw.*, San Diego, CA, vol. 1, pp. 343–353, 1988.
[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986.
[50] B. M. Wilamowski, “Neural network architectures and learning algorithms: How not to be frustrated with neural networks,” *EEE Ind Electron Mag*, vol. 3, no. 4, pp. 56–63, Dec. 2009.
[51] V. V. Phansalkar and P. S. Sastry, “Analysis of the back-propagation algorithm with momentum,” *IEEE Trans. Neural Netw.*, vol. 5, no. 3, pp. 505–506, May 1994.
[52] L.-H. Zhou, P. Han, S.-M. Jiao, and B.-R. Lin, “Feedforward neural networks using RPROP algorithm and its application in system identification,” in *2002 International Conference on Machine Learning and Cybernetics, 2002. Proceedings*, vol. 4, pp. 2041–2044, 2002.
[53] C.-C. Yu and B.-D. Liu, “A backpropagation algorithm with adaptive learning rate and momentum coefficient,” in *Proceedings of the 2002 International Joint Conference on Neural Networks, 2002. IJCNN*, vol. 2, pp. 1218–1223, 2002.

[54] B. M. Wilamowski and J. D. Irwin, Eds., "Levenberg-Marquardt Training," in *Intelligent systems*, 2nd ed., Boca Raton: CRC Press, pp. 12–1 – 12–16, 2011.
[55] K. Levenburg, "A method for the solution of certain problems in least squares," *Q. Appl. Machematics*, vol. 5, pp. 164–168, 1944.
[56] D. Marquardt, "An Algorithm for the Least-Squares Estimation of Nonlinear Parameters," *SIAM J Ournal Appl. Math.*, vol. 11, no. 2, pp. 431–441, Jun. 1963.
[57] B. M. Wilamowski, N. Cotten, O. Kaynak, and G. Dundar, "Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks," *IEEE Trans Ind. Electron.*, vol. 55, no. 10, pp. 3784–3790. 2010.
[58] H. Yu and B. M. Wilamowski, "Efficient and reliable training of neural networks", in , Catania, Italy, May 21-23, 2009, pp. 109-115.," in *Proc. 2nd IEEE Human System Interaction Conf. HSI 2009*, Catania, Italy, pp. 109–115, 2009.
[59] B. M. Wilamowski, N. Cotten, J. Hewlett, and O. Kaynak, "Neural network trainer with second order learning algorithms". Proc. International Conference on Intelligent Engineering Systems," in *Proc. International Conference on Intelligent Engineering Systems*, pp. 127–132, 2007.
[60] M. T. Hagan and M. Menhaj, "Traing Feed Forward Networks with the Marquardt Algorithm," *IEEE Trans. Neural Netw.*, vol. 5, pp. 989–993, 1994.
[61] Z.-W. Sun and G. Dixit, "Optimized bath control for void-free copper deposition," *Solid State Technol.*, vol. 44, pp. 46–51, Nov. 2001.
[62] J. D. Adolf, "Modeling the role of plating additives in the metallization of semiconductor interconnects: From dual damascene to through silicon vias," Case Western University, 2011.

Appendix

Appendix A. Python Script for parity 3 ELM network

```
import random
import numpy
import math

inputs=[[0,0,0],[0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]]
t=[0,1,1,0,1,0,0,1]

numInputs=3
hidden=8
patterns=len(inputs)

w = numpy.zeros((numInputs,hidden))
b= numpy.zeros((hidden))

Hmatrix=numpy.zeros((patterns,hidden))
for a in range(0,numInputs):
   for c in range(0,hidden):
      w[a,c]=random.random()
      b[c]=random.random()

for k in range(0,patterns):
   nSum=0
   for i in range(0,hidden):
      nSum=0
      for j in range(0,numInputs):
         prod=inputs[k][j]*w[j][i]
         nSum=nSum+prod
      nSum=nSum+b[i]
      gNsum=1/(1+math.exp(-1*nSum))
      Hmatrix[k][i]=gNsum
Hinv=numpy.linalg.pinv(Hmatrix,rcond=1e-15)
outW=numpy.dot(Hinv,t)
print "Output Weights:\n",outW
outArray=numpy.dot(Hmatrix,outW.transpose())
print “Outputs:\n”, outArray
```

Appendix B. Error Back Propagation (EBP) Python Script

```
import random
import numpy
import math

inputs=[[0,0],[0,1],[1,0],[1,1]]
outputs=[0,1,1,0]
hidden = 3 # number hidden layer neurons
numInputs = len(inputs[0]) #number of inputs
numPatterns = len(inputs) #number of input patterns
numOutpatterns = len(outputs) #number of output patterns
numOutputs = 1 #number of output neurons
k2=1 #
acc=1 #
tries=1 #number of training runs
inputs=numpy.insert(inputs,numInputs,values=1,axis=1)#add column of '1's for bias inputs
print inputs

iterations=50000 #number of iterations
sumErrors=numpy.zeros((tries))
RMSE=numpy.zeros((tries))
for t in range (0,tries):
   inputWeights = numpy.random.uniform(0, 1, (numInputs+1,hidden)) #initialize input weights

   outputWeights = numpy.random.uniform(0, 1, (hidden+1,numOutputs)) #initialize output
weights

    print_count=0
   for ite in range (0,iterations):
      sumErrors[t]=0
      print_count=print_count + 1
      net1=numpy.dot(inputs,inputWeights)#Multiply inputs and input weights
      out1=1/(1+numpy.exp(-1*k2*net1))#Hidden Layer sigmoid function output
      ip2 = numpy.insert(out1, hidden, values=1, axis=1)#add column of '1's for hidden layer bias

      net2=numpy.dot(ip2,outputWeights)#Multiply hidden layer output and hidden layer weights
      out2=1/(1+numpy.exp(-1*k2*net2))#Final outputs

      ee2=numpy.subtract(outputs,out2.T)#output error
      del2=numpy.multiply(ee2.T,numpy.multiply(1-out2,out2))#Output error gradient
multiplier
      ee1=numpy.dot(outputWeights,del2.T)#back propagated error

      ee1=numpy.delete(ee1,hidden,axis=0)#delete the bias error - not a neuron to back
```

```
propagate

      del1=numpy.multiply(ee1.T,numpy.multiply((1-out1),out1))#hidden layer error gradient
multipliers

      dw2=numpy.dot(del2.T,ip2)#hidden layer delta weights
      outputWeights=numpy.add(outputWeights,dw2.T)#new output weights
      dw1=numpy.dot(del1.T,inputs)#input delta weights


     inputWeights=numpy.transpose(numpy.add(numpy.transpose(inputWeights),dw1))#new
input weights
      myErrors=numpy.transpose(ee2)#Errors
      for x in range(0,numPatterns):#Sum Squared Errors
         sumErrors[t]=sumErrors[t]+myErrors[x]**2
      #RMSE[t]=math.sqrt(sumErrors[t]/numPatterns)#Root Mean Squared Errors
      #print "Training # ",t+1,"Iteration # ",ite, "RMSE= ",RMSE[t]
     if print_count == 2000:
                    print "Training # ",t+1,"Iteration # ",ite, "SSE= ",sumErrors[t]
                    print_count=0
#print "RMSE=",RMSE
print "SSE= ",sumErrors
print "Output:\n",out2
```

Appendix C. Python implementation of the Gauss-Newton algorithm

```
import numpy
import matplotlib.pyplot as plt

inputs = [[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 1]]
outputs = [0, 1, 1, 0, 1, 0, 1]

hidden = 5  # number hidden layer neurons
numInputs = len(inputs[0])  # number of inputs
numPatterns = len(inputs)  # number of input patterns
numOutpatterns = len(outputs)  # number of output patterns
numOutputs = 1  # number of output neurons
k2 = 0.3
tries = 300  # number of training runs
inputs = numpy.insert(inputs, numInputs, values=1, axis=1)  # add column of '1's for bias inputs
iterations = 100  # number of iterations

sumErrors = numpy.zeros(tries)
del_hidden_out1 = numpy.zeros(hidden+1)
jj_element = numpy.zeros((numPatterns,((numInputs+1)*hidden)))
RMSE = numpy.zeros((tries))
newInputWeights = numpy.zeros((numInputs+1, hidden))
newOutputWeights = numpy.zeros((hidden+1, numOutputs))
jj_identity = numpy.identity(((numInputs+1)*hidden)+((hidden+1)*numOutputs))
savedite = 0
success = 0
for t in range(0, tries):
   inputWeights = numpy.random.uniform(0, 1, (numInputs+1, hidden))  # initialize input
weights
   outputWeights = numpy.random.uniform(0, 1, (hidden+1, numOutputs))  # initialize output
weights
   print_count = -1
   sumError = numpy.zeros(iterations)
   for ite in range(0,iterations):

      sumErrors[t] = 0
      print_count = print_count+1
      net1 = numpy.dot(inputs, inputWeights)  # Multiply inputs and input weights
      out1 = 1/(1+numpy.exp(-1*k2*net1))  # Hidden Layer sigmoid function output
      slope = numpy.multiply(out1, (1-out1))
      ip2 = numpy.insert(out1, hidden, values=1, axis=1)  # add column of '1's for hidden layer
bias
      net2 = numpy.dot(ip2, outputWeights)  # Multiply hidden layer output and hidden layer
weights
      out2 = 1/(1+numpy.exp(-1*k2*net2))  # Final outputs
```

```
    ee2 = numpy.subtract(outputs, out2.T)  # output error
    myErrors = numpy.transpose(ee2)  # Errors
    for x in range(0, numPatterns):  # Sum Squared Errors
        sumErrors[t] = sumErrors[t]+myErrors[x]**2
        eesum = sumErrors
    slope_out = numpy.multiply(out2, (1-out2))
    delout = slope_out
    del_hidden_out = numpy.dot(delout, numpy.transpose(outputWeights))  # need to cut last
column from del  hidden  out & then .* the 4x3 arrays (slope*Del  hidden  out) to get input Del
    del_hidden_out_nobias = numpy.delete(del_hidden_out, hidden, axis=1)  # delete the bias
error - not a neuron to back propagate
    del_input_hidden = numpy.multiply(slope, del_hidden_out_nobias)
    jj_element2 = ip2
    for x in range(0, numPatterns):
        jj_element2[x, :] *= delout[x]
    jj_element2 = jj_element2*-1
    for w in range(0, numPatterns):
        count = 0
        for x in range(0, hidden):
            for y in range (0, numInputs+1):
                jj_element[w, count] = numpy.multiply(inputs[w][y], del_input_hidden[w][x])
                count = count+1
    jj_element = jj_element*-1
    jm = numpy.append(jj_element, jj_element2, axis=1)
    jj_invert = numpy.linalg.pinv(numpy.dot(numpy.transpose(jm), jm))
    Del_weight = numpy.dot(jj_invert, numpy.dot(numpy.transpose(jm), myErrors))
    count = 0
    for y in range(0, hidden):
        for x in range(0, numInputs+1):
            newInputWeights[x][y] = inputWeights[x][y]-Del_weight[count]
            count = count+1
    for x in range(0, numOutputs):
        for y in range(0, hidden+1):
            newOutputWeights[y][x] = outputWeights[y][x]-Del_weight[count]
            count = count+1
    myErrors = numpy.transpose(ee2)  # Errors
    for x in range(0, numPatterns):  # Sum Squared Errors
        sumErrors[t] = sumErrors[t]+myErrors[x]**2
    inputWeights = newInputWeights
    outputWeights = newOutputWeights
    sumError[ite] = sumErrors[t]

    if sumError[ite] == sumError[ite-1]:
        plt.semilogy(sumError)
        if ite >= savedite:
            savedite = ite
```

```
            break
        if sumErrors[t] <= 0.0001:
            plt.semilogy(sumError)
            if ite >= savedite:
                savedite = ite
            success = success+1
            break
        if ite >= savedite:
                savedite=ite
        print "training= ", t+1, "iteration= ", ite+1,"SSE= ", sumErrors[t]
    plt.semilogy(sumError)
    plt.ylabel('SSE')
    plt.xlabel('iterations')
    plt.xlim([0,100])

    print out2
print "Success =  ",success,"out of ",tries
plt.show()
```

Appendix D. Levenberg-Marquardt Algorithm python implementation

```
import numpy
import matplotlib.pyplot as plt

inputs = [[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 1]]
outputs = [0, 1, 1, 0, 1, 0, 1]
hidden = 4  # number hidden layer neurons
numInputs = len(inputs[0])  # number of inputs
numPatterns = len(inputs)  # number of input patterns
numOutpatterns = len(outputs)  # number of output patterns
numOutputs = 1    # number of output neurons
k2 = 0.3
tries = 200  # number of training runs
inputs = numpy.insert(inputs, numInputs, values=1, axis=1)  # add column of '1's for bias inputs
iterations = 100  # number of iterations

sumErrors = numpy.zeros(tries)
del_hidden_out1 = numpy.zeros((hidden+1))
jj_element = numpy.zeros((numPatterns, ((numInputs+1)*hidden)))
RMSE = numpy.zeros(tries)
newInputWeights = numpy.zeros((numInputs+1, hidden))
newOutputWeights = numpy.zeros((hidden+1, numOutputs))
jj_identity = numpy.identity(((numInputs+1)*hidden)+((hidden+1)*numOutputs))
savedite = 0
success = 0
for t in range(0, tries):
    inputWeights = numpy.random.uniform(0, 1, (numInputs+1, hidden))  # initialize input
weights
    outputWeights = numpy.random.uniform(0, 1, (hidden+1, numOutputs))  # initialize output
weights
    print_count = -1
    mu = 0.01
    sumError = numpy.zeros(iterations)
    for ite in range(0, iterations):
        sumErrors[t] = 0
        print_count = print_count+1
        net1 = numpy.dot(inputs, inputWeights)  # Multiply inputs and input weights
        out1 = 1/(1+numpy.exp(-1*k2*net1))  # Hidden Layer sigmoid function output
        slope = numpy.multiply(out1, (1-out1))
        ip2 = numpy.insert(out1, hidden, values=1, axis=1)  # add column of '1's for hidden layer
bias
        net2 = numpy.dot(ip2, outputWeights)  # Multiply hidden layer output and hidden layer
weights
        out2 = 1/(1+numpy.exp(-1*k2*net2))  # Final outputs
        ee2 = numpy.subtract(outputs, out2.T)  # output error
```

```
    myErrors = numpy.transpose(ee2)  # Errors
    for x in range(0, numPatterns):  # Sum Squared Errors
        sumErrors[t] = sumErrors[t]+myErrors[x]**2
        eesum = sumErrors
    slope_out = numpy.multiply(out2, (1-out2))
    delout = slope_out
    del_hidden_out = numpy.dot(delout, numpy.transpose(outputWeights))  # need to cut last
column from del_hidden_out & then .* the 4x3 arrays (slope*Del_hidden_out) to get input Del
    del_hidden_out_nobias = numpy.delete(del_hidden_out, hidden,axis=1)  # delete the bias
error - not a neuron to back propagate
    del_input_hidden = numpy.multiply(slope, del_hidden_out_nobias)
    jj_element2 = ip2
    for x in range(0, numPatterns):
        jj_element2[x, :] *= delout[x]
    jj_element2 = jj_element2*-1
    for w in range(0, numPatterns):
        count = 0
        for x in range(0, hidden):
            for y in range(0, numInputs+1):
                jj_element[w,count] = numpy.multiply(inputs[w][y], del_input_hidden[w][x])
                count = count+1
    jj_element = jj_element*-1
    jm = numpy.append(jj_element, jj_element2, axis=1)
    mu_part = mu*jj_identity
    jj_invert = numpy.linalg.pinv(numpy.dot(numpy.transpose(jm), jm)+mu_part)
    Del_weight = numpy.dot(jj_invert, numpy.dot(numpy.transpose(jm), myErrors))
    count = 0
    for y in range(0, hidden):
        for x in range(0, numInputs+1):
            newInputWeights[x][y] = inputWeights[x][y]-Del_weight[count]
            count = count+1
    for x in range(0,numOutputs):
        for y in range(0, hidden+1):
            newOutputWeights[y][x] = outputWeights[y][x]-Del_weight[count]
            count = count+1
    myErrors = numpy.transpose(ee2)  # Errors
    for x in range(0,numPatterns):  # Sum Squared Errors
        sumErrors[t] = sumErrors[t]+myErrors[x]**2
    pre_inputWeights = inputWeights
    pre_outputWeights = outputWeights
    inputWeights = newInputWeights
    outputWeights = newOutputWeights
    sumError[ite] = sumErrors[t]
    if ite != 0:
        if sumError[ite] >= sumError[ite-1]:
            print sumError[ite], sumError[ite-1]
```

```
            inputWeights = pre_inputWeights
            outputWeights = pre_outputWeights
            mu = mu*10
        else:
            mu = mu/10
        if sumError[ite] == sumError[ite-1]:
            plt.semilogy(sumError)
            break
    if sumErrors[t] <= 0.0001:
        plt.semilogy(sumError)
        if ite >= savedite:
            savedite = ite
        success = success+1
        break


    print "training= ", t+1, "iteration= ", ite+1, "SSE= ", sumErrors[t]

  plt.semilogy(sumError)
  print out2
print "Success =  ", success, "out of ", tries

plt.ylabel('SSE')
plt.xlabel('iterations')
plt.xlim([0, savedite+10])
plt.show()
```

Appendix E. Original Plating Bath Data

Test1	Test2	Test3	Test4	Acc	Sup	Lev
2390	353.5	1540	274.7	7	3	2
1680	314.1	1130	208.1	7	3	3.8
1150	681.7	763.7	275.4	1	1.2	4
873.5	462.2	750.1	224.7	1	1.8	4
794.8	350.9	740.5	195	1	2.4	4
767	290.8	731.8	175.1	1	3	4
759.6	252.7	724.4	161.6	1	3.6	4
749.9	228.3	718.5	150.9	1	4.2	4
740.4	210	712.2	143.1	1	4.8	4
737.5	196.9	707.8	137.1	1	5.4	4
734.1	187.2	702.3	131.4	1	6	4
1290	603.8	973.6	267.2	5	1.2	5
1260	434	947.2	229.4	5	1.8	5
1250	349.1	928.4	206.2	5	2.4	5
1230	300.4	914.2	189.6	5	3	5
1230	269.1	902.8	177.9	5	3.6	5
1220	246.6	893.6	169.3	5	4.2	5
1210	229.7	886.6	162.6	5	4.8	5
1200	217.2	880.3	157	5	5.4	5
1200	207.5	873.4	154.2	5	6	5
1390	281.1	963.1	174.9	7	3	5.6
1220	255	871	154.2	7	3	7.4
1110	232.9	808.4	139.6	7	3	9.2
663.9	405	539.8	188.9	2	0.6	10
969.6	437.3	784.3	214.3	5	0.6	10
1470	438.7	1130	227.1	9	0.6	10
638.1	313.7	516.7	160.2	2	1.2	10
947.4	340.2	763.7	183.3	5	1.2	10
1450	340.8	1110	198.6	9	1.2	10
621.9	261.5	501.4	141.9	2	1.8	10
930.9	284.8	750.1	163.5	5	1.8	10
1450	289.6	1090	180.5	9	1.8	10
610.4	228.2	490.9	129.6	2	2.4	10
918.6	249.6	740.5	149.9	5	2.4	10
1430	257.3	1070	167.6	9	2.4	10
602.6	206	482.4	119.7	2	3	10
908.3	226.2	731.8	139.8	5	3	10
1430	235.3	1060	158.3	9	3	10
594.7	190.5	475.6	112.8	2	3.6	10
900.1	209.2	724.4	132.4	5	3.6	10
1420	220.3	1050	151.3	9	3.6	10
588	178.4	471	107.3	2	4.2	10

893.5	197.2	718.5	126.5	5	4.2	10
1410	209.2	1040	145.2	9	4.2	10
582.9	169.9	466.8	102.6	2	4.8	10
887.2	187.7	712.2	121.7	5	4.8	10
1400	200.7	1030	140.4	9	4.8	10
578.1	162.4	463.4	98.7	2	5.4	10
882	180.2	707.8	117.5	5	5.4	10
1400	193.7	1020	136.4	9	5.4	10
575.2	156.3	459.6	95.5	2	6	10
877.2	173.7	702.3	114.2	5	6	10
1390	187.7	1010	132.8	9	6	10
1030	213.8	760.7	128.8	7	3	11
971.7	196.2	722.5	120.3	7	3	12.8
922.1	179.6	691.6	113.1	7	3	14.6
579.9	357.6	593.5	247.7	2	0	15
553.6	279.7	550.8	193.8	2	0.6	15
998.4	262.3	819.8	153	7	0.6	15
538.6	234.1	525.1	164.5	2	1.2	15
979.2	231.6	804.3	139.5	7	1.2	15
527.1	204.5	510.8	145	2	1.8	15
979.2	201.9	792.7	128.5	7	1.8	15
518.7	184.6	500.8	131.9	2	2.4	15
969.6	184.9	783	120.4	7	2.4	15
349.8	144.3	313.8	90.3	0.5	3	15
422.5	147.6	376.3	93.5	1.3	3	15
512.2	171.3	491.9	122.1	2	3	15
501.4	150.2	437.2	96.4	2.1	3	15
578.8	152.2	497.4	99	2.9	3	15
661	154.1	557.5	102.2	3.7	3	15
740.8	157.1	616.4	105.4	4.5	3	15
820.7	160	674.9	108.5	5.3	3	15
902.8	162.4	731.6	111.7	6.1	3	15
981.8	164.3	788.3	114.6	6.9	3	15
960	171	775.8	114.5	7	3	15
1060	167.6	845.6	117.6	7.7	3	15
1150	169.5	900.6	121	8.5	3	15
507	160.3	484.7	114.9	2	3.6	15
959.8	160.4	768.9	109.7	7	3.6	15
502.5	152.4	478.7	108.8	2	4.2	15
952.5	152.9	762.9	105.6	7	4.2	15
498.4	145.5	473.5	104.2	2	4.8	15
947.7	146.7	756.8	101.8	7	4.8	15
495	140	468.9	100.1	2	5.4	15
942.7	142.4	751.6	98.9	7	5.4	15
492.2	135.1	464.7	96.9	2	6	15

937.3	137.8	747.7	95.9	7	6	15
880.1	168.7	665.6	107	7	3	16.4
842.1	159.4	643.1	102	7	3	18.2
1050	211.5	886.8	138	9	0.6	20
1040	189.8	877	128.7	9	1.2	20
1030	175.1	868.3	121.5	9	1.8	20
1020	164.9	860.1	116.1	9	2.4	20
816.3	150.8	625	97.3	7	3	20
1010	157.1	852.9	111.8	9	3	20
1010	150.5	846.1	108.2	9	3.6	20
1000	145.6	840.6	104.9	9	4.2	20
997.9	140.9	835.2	102.7	9	4.8	20
990.5	137.5	829.8	100.4	9	5.4	20
986.3	134.7	824.7	98.6	9	6	20