Algorithm setup and proof of convergence:

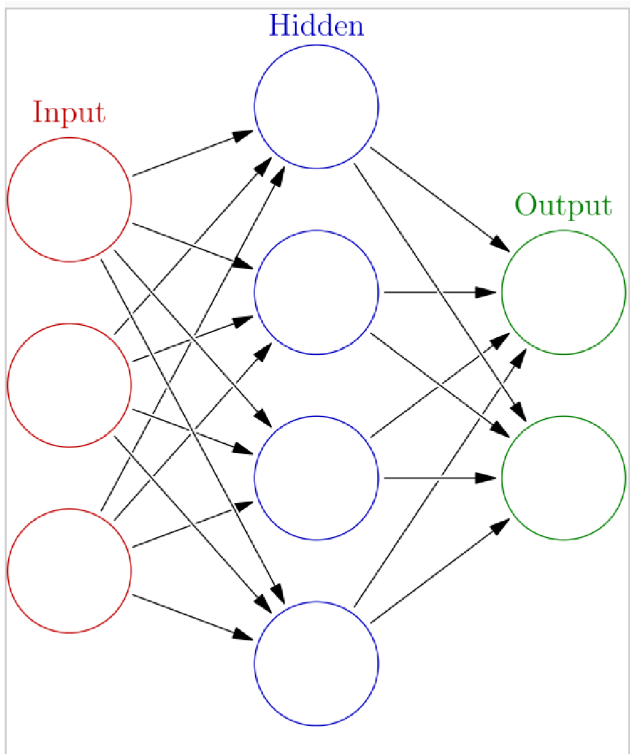


Figure 1

The **method** is as follows:

1) We decide on some group of objects that we want to classify and on the number of classification types.
2) We define how we are going to represent each instance (or non-instance) in terms of real-valued vectors of *size* n.
3) For each classification type we define a desired real-valued output vector of *size* m.
4) We define a size, p, for our hidden layer vector.
5) We initialize a p x n matrix $\theta(1)$ and an m x p matrix $\theta(2)$ (that will represent our connection weights) with random real numbers.
6) We take a *size* n input vector representing an instance of the object we want to classify and we apply $\theta(1)$ to it.
7) We are then left with a *size* p vector that represents the summation of the values of all of the hidden node's inputs.
8) We input each entry in our vector from (7) into a threshold function and get another *size* p vector.
9) We apply steps (6) through (8) to our hidden layer, this time using $\theta(2)$. This results in our *size-*m output vector.
10) We compare the output of our neural net to the desired output.

11) We make adjustments to values in our $\theta(1)$ and $\theta(2)$ matrices in order to get "better" results for this instance.
12) We repeat the process on other instances.

The "**Sigmoid Function**" **method**:
-We use the Sigmoid function as our threshold function:
$S(x) = 1 \;/\; (1 + e^{-c\,x})$

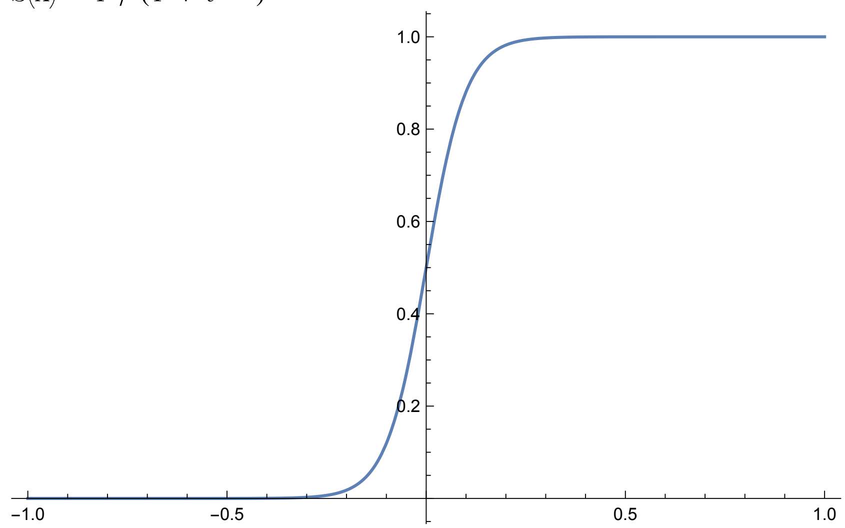


Figure 2: The Sigmoid function for c = 1.

Why the Sigmoid?
We primarily use the Sigmoid because it is "like" a step function yet it is differentiable. The Sigmoid can also be interpreted as a randomization. Lastly, when we use a sigmoid we get an element of confidence in our nets output. Often times an input will cause the net to output more than 1 response ("That looks like a 5 AND an S"). If we used step functions as the threshold device we wouldn't have a notion of which output is the most likely result.

We define the function A: $(R^n \times R^{p*n} \times R^{m*p}) \to R^m$ representing our neural net as follows (We call the function 'A' for "actual output"):
For real valued matrices $\theta(1)$ (p x n) and $\theta(2)$ (m x p) and random input $\xi \in R^n$:

$$A(\theta(1),\ \theta(2), \xi) = S_m(\theta(2)(S_p(\theta(1)(\xi))))$$

Where $S_i : R^i \to R^i$ is the Sigmoid function operating component-wise on a length i column vector. For $S_1 : \mathbb{R} \to \mathbb{R}$ we simply write S.

-We observe the following:

$$\frac{d}{dx} S(x) = \frac{d}{dx}\left(\frac{1}{1 + e^{-c\,x}}\right) = \frac{c\,e^{-c\,x}}{(1+e^{-c\,x})^2} = c\,S(x)\,(1 - S(x))$$

For each ξ we have a corresponding $T \in R^m$ ('T' for "target output") and we wish to minimize:

$$|E[T - A(\theta(1), \theta(2), \xi)]| \text{ or similarly } E[(T - A(\theta(1), \theta(2), \xi))^2]$$

We let $\theta = (\theta(1), \theta(2))$ and define $\phi(\theta, \xi) = (T - A(\theta(1), \theta(2), \xi))^2$ and $J(\theta, \xi) = E[\phi(\theta, \xi)]$.

-**We show that $\phi(\theta, \xi)$ is Lipschitz and bounded**:

-First We show that $A(\theta, \xi)$ is bounded and Lipschitz:

-$A(\theta, \xi)$ is clearly bounded since $S(x) \in (0, 1)$ $\forall x \in \mathbb{R}$

-Now since $\frac{d}{dx}S(x) = c\, S(x)(1 - S(x)) < c$ and since $S \in C^\infty$ we can invoke the mean value theorem and we have $\forall\, x, y \in \mathbb{R}$ $\exists$ M such that:

$$\left|\frac{S(x) - S(y)}{x - y}\right| = |S'(M)| \le c$$

$\Rightarrow$

$$|S(x) - S(y)| \le c|x-y|$$

Thus S is Lipschitz and since the composition of Lipschitz functions is again Lipschitz and since linear operators between finite dimensional spaces are Lipschitz we have that $A(\theta, \xi)$ is Lipschitz.

-Also, since $|T - A(\theta(1), \theta(2), \xi_1) - (T - A(\theta(1), \theta(2), \xi_2))| = |A(\theta(1), \theta(2), \xi_1) - A(\theta(1), \theta(2), \xi_2)|$

$\Rightarrow T - A(\theta(1), \theta(2), \xi)$ is Lipschitz

-Lastly since $f(x) = x^2$ is analytic and has bounded derivative for $x \in (0, 1)$, it is locally Lipschitz. For each of the m positions of the vector: $(T - A(\theta(1), \theta(2), \xi))$ we have $(T_k - A_k(\theta(1), \theta(2), \xi)) \in (0, 1)$ it follows that $\phi(\theta, \xi)$ is Lipschitz and is also clearly bounded.

-For each of our 'm' target outputs, we demand that ξ take on only a finite number of values (in our particular example ξ represents the color information for pixels in an image), and since $\phi(\theta, \xi) \in C^\infty$ is bounded and Lipschitz we have:

$$\nabla_\theta J(\theta, \xi) = \nabla_\theta E[\phi(\theta, \xi)] = \nabla_\theta \sum_{\xi \in \Omega} \xi\, \phi(\theta, \xi) = \sum_{\xi \in \Omega} \xi\, \nabla_\theta \phi(\theta, \xi) = E[\nabla_\theta \phi(\theta, \xi)]$$

We discuss the convergence of our algorithm:

We want to do the following descent algorithm:

$$\theta_{n+1} = \theta_n - \epsilon_n \nabla_\theta J(\theta_n, \xi)$$

-or-

$$\theta(1)_{n+1} = \theta(1)_n - \epsilon_n \nabla_{\theta(1)} J(\theta_n, \xi)$$
$$\theta(2)_{n+1} = \theta(2)_n - \epsilon_n \nabla_{\theta(2)} J(\theta_n, \xi)$$

(i) We let ϵ_n be such that $\sum \epsilon_n = \infty$ and $\sum \epsilon_n^2 < \infty$

(ii) Our model is exogenous since each ξ is independent of θ and so we wish to use the stochastic approximation algorithm:

$$\theta_{n+1} = \theta_n + \epsilon_n\, \Upsilon_n$$
$$\text{Where } \Upsilon_n = -\nabla_\theta\, \phi(\theta, \xi)$$

(iii) We want a feedback function G and sequence of bias terms $\{\beta_k\}$ such that

$E[\Upsilon_n \mid \mathcal{F}_{n-1}] = G(\theta_n) + \beta_n$

By our work above, since $\phi(\theta, \xi)$ is Lipschitz and bounded:

$\nabla_\theta E[\phi(\theta, \xi)] = E[\nabla_\theta\, \phi(\theta, \xi)]$

$\Rightarrow \nabla_\theta E[\phi(\theta_n, \xi) \mid \mathcal{F}_{n-1}] = E[\nabla_\theta\, \phi(\theta_n, \xi) \mid \mathcal{F}_{n-1}]$

$\Rightarrow E[\Upsilon_n \mid \mathcal{F}_{n-1}] = -\nabla_\theta J(\theta_n, \xi)$

Thus $G(\theta_n) = -\nabla_\theta J(\theta_n, \xi)$ and $\beta_n \approx 0 \;\; \forall\, n \in \mathbb{Z}^+$

(iv) $\beta_n = 0 \;\Rightarrow \| \beta_n \| = 0 \;\Rightarrow\; \sum \epsilon_n \| \beta_n \| < \infty$

(v) Since ξ can take on only a finite number of values we have

that $V_n < M$ for some $M \in \mathbb{R}^+$ and $\forall\, n \in \mathbb{Z}^+$

$\Rightarrow \sum \epsilon_n^2\, V_n < \infty$

(vi) Since we are using a negative gradient procedure where $G(\theta_n) = -\nabla_\theta J(\theta_n, \xi)$

we will converge to to a local minimum which depends on the initial weights,

θ_0 and the ODE:

$$\frac{d}{dt}\mathcal{V}(t) = G(\mathcal{V}(t))$$

has a unique limit for each initial condition.

By (i) - (iv) above, the algorithm:

$$\theta_{n+1} = \theta_n - \epsilon_n\, \nabla_\theta\, \phi(\theta_n, \xi)$$
$$\text{-or-}$$
$$\theta(1)_{n+1} = \theta(1)_n - \epsilon_n\, \nabla_{\theta(1)}\, \phi(\theta_n, \xi)$$
$$\theta(2)_{n+1} = \theta(2)_n - \epsilon_n\, \nabla_{\theta(2)}\, \phi(\theta_n, \xi)$$

We have that $\theta_n \to \theta^*$ a.s where θ^* is a local minimum that depends on the initial condition θ_0.

We calculate $\nabla_{\theta(1)}\, \phi(\theta, \xi)$ and $\nabla_{\theta(2)}\, \phi(\theta, \xi)$:

-We define a function called the "hidden output" H: $(R^n \times R^{p*n}) \to R^p$

as $H(\theta(1), \xi) = S_p(\theta(1)\, \xi)$

-We observe:

$$A_k(\theta(1), \theta(2), \xi) = S\left(\sum_{j=1}^{p} \theta(2)_{k,j}\, H_j(\theta(1), \xi)\right)$$

and:

$$H_j(\theta(1), \xi) = S\left(\sum_{s=1}^{n} \theta(1)_{j,s} \cdot \xi_s\right)$$

-For notational simplicity we also define:

$$\overline{H_j}\,(\theta(1), \xi) = \sum_{s=1}^{n} \theta(1)_{j,s} \cdot \xi_s$$

(This is the hidden output before application of the Sigmoid)

Now: $\frac{\partial}{\partial \theta(1)_{uv}}\, \phi(\theta, \xi) = \frac{\partial}{\partial \theta(1)_{uv}}\, (T - A(\theta(1), \theta(2), \xi))^2$

(and dropping the notation with independent variables we have)

$$\frac{\partial}{\partial \theta(1)_{uv}}\,(T_k - A_k)^2 = \frac{\partial}{\partial A_k}(T_k - A_k)^2\, \frac{\partial A_k}{\partial \theta(1)_{uv}} =$$
$$= -2\,(T_k - A_k)\, \frac{\partial}{\partial \theta(1)_{uv}} A_k$$

$= -2\,(T_k - A_k)\,\frac{\partial}{\partial\,\theta(1)_{u\,v}} S\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\, H_j)$
$= -2\,(T_k - A_k)\,S'\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\, H_j)\,\frac{\partial}{\partial\,\theta(1)_{u\,v}}(\sum_{j=1}^{p} \theta(2)_{k,\,j}\, H_j)$
$= -2\,(T_k - A_k)\,S'\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\, H_j)\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\,\frac{\partial}{\partial\,\theta(1)_{u\,v}}\, H_j)$
$= -2\,(T_k - A_k)\,S'\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\, H_j)\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\,\frac{\partial}{\partial\,\theta(1)_{u\,v}}\, S\,(\sum_{s=1}^{n} \theta(1)_{\,j,\,s} \cdot \xi_s))$
$= -2\,(T_k - A_k)\,S'\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\, H_j)\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\;\; S'\,(\sum_{s=1}^{n} \theta(1)_{\,j,\,s} \cdot \xi_s)\,\frac{\partial}{\partial\,\theta(1)_{u\,v}}\,(\sum_{s=1}^{n} \theta(1)_{\,j,\,s} \cdot \xi_s))$
$= -2\,(T_k - A_k)\,S'\,(A_k)\,(\sum_{j=1}^{p} \theta(2)_{k,\,j}\;\; S'\,(\overline{H_j})\,\frac{\partial}{\partial\,\theta(1)_{u\,v}}\,(\sum_{s=1}^{n} \theta(1)_{\,j,\,s} \cdot \xi_s))$

Observe: $\frac{\partial}{\partial\,\theta(1)_{u\,v}}\,(\theta(1)_{\,j,\,s} \cdot \xi_s) = 0$ if j ≠ u and s ≠ v

$= -2\,(T_k - A_k)\,S'\,(A_k)\,\theta(2)_{k,\,u}\;\; S'\,(\overline{H_u})\,\xi_v$
$\Rightarrow$
$\frac{\partial}{\partial\,\theta(1)_{u\,v}}\,(T - A)^2 = -2\sum_{k=1}^{10}(T_k - A_k)\,S'\,(A_k)\,\theta(2)_{k,\,u}\;\; S'\,(\overline{H_u})\,\xi_v$
$\Rightarrow$
$\frac{\partial}{\partial\,\theta(1)_{u\,v}}\,(T - A)^2 = -2\sum_{k=1}^{10} \theta(2)_{k,\,u}\,(T_k - A_k)\,S'\,(\overline{H_u})\,\xi_v\;\; S'\,(A_k)$

(adding back all notation)

$\frac{\partial}{\partial\,\theta(1)_{u\,v}}\,(T - A\,(\theta(1),\;\theta(2),\,\xi\,))^2 =$
$= -2\sum_{k=1}^{10} \theta(2)_{k,\,u}\,(T_k - A_k(\theta(1),\;\theta(2),\,\xi\,))\,S'\left(\overline{H_u}(\theta(1),\,\xi)\right)\xi_v\;\; S'\,(A_k(\theta(1),\;\theta(2),\,\xi\,))$

$$\Rightarrow\; \nabla_{\theta(1)}\,\boldsymbol{\phi}(\,\theta,\,\xi) = -2\begin{pmatrix} E_1 * c\,(H_1)\,(1 - H_1) \\ \ldots \\ E_k * c\,(H_k)\,(1 - H_k) \\ \ldots. \\ E_p * c\,(H_p)\,(1 - H_p) \end{pmatrix} \cdot \xi$$

$$\text{where } E = \theta(2)^T \begin{pmatrix} (T_1 - A_1) * c\,(A_1)\,(1 - A_1) \\ \ldots \\ (T_k - A_k) * c\,(A_k)\,(1 - A_k) \\ \ldots. \\ (T_m - A_m) * c\,(A_m)\,(1 - A_m) \end{pmatrix}$$

Similarly $\Rightarrow \frac{\partial}{\partial\,\theta(2)_{u\,v}}\;\phi(\,\theta,\,\xi) = -2\,(T_u - A_u)\,\frac{\partial A_u}{\partial\,\theta(2)_{u\,v}}$

$= -2\,(T_v - A_v)\;\; S'\,(\sum_{k=1}^{p} \theta(2)_{u\,k} \cdot H_k)\,\frac{\partial}{\partial\,\theta(2)_{u\,v}}\,\sum_{k=1}^{p} \theta(2)_{u\,k} \cdot H_k$

$= -2\,(T_v - A_v)\;\; S'\,(\sum_{k=1}^{p} \theta(2)_{u\,k} \cdot H_k)\;\; H_v$

$$\Rightarrow\; \nabla_{\theta(1)}\,\phi(\,\theta,\,\xi) = -2\begin{pmatrix} (T_1 - A_1) * c\,(A_1)\,(1 - A_1) \\ \ldots \\ (T_k - A_k) * c\,(A_k)\,(1 - A_k) \\ \ldots. \\ (T_m - A_m) * c\,(A_m)\,(1 - A_m) \end{pmatrix} \cdot \mathrm{H}$$

MNIST:
PROCEDURE:

1) We set n = 786 so each input node corresponds to a pixel value
2) We normalize the pixel values so they are in the range (0.01, 1]
3) We set m = 10, where each position corresponds to a possible output
4) For each digit 0 - 9, our desired output is .01 in every position of our output vector except for the position corresponding to the correct digit, which should output .99
(we choose these values over 0 and 1 because the Sigmoid cannot output 0 or 1 and our descent algorithm will be forever chasing impossible outputs)
5) We set a value for p (We tried 397 and 800, these are popular for MNIST)
6) We initialized the weights in $\theta(1)$ and $\theta(2)$ from normal distributions centered at 0 with standard deviations $\frac{1}{\sqrt{n}}$ and $\frac{1}{\sqrt{p}}$ respectively, being careful to reset any weights that get a 0 value.
7) From here we experiment and we need to decide:
    a. our sequence $\{\epsilon_n\}$
    b. Value of c in the Sigmoid function (always set to 1 in our experiments):
    c. How many epochs (number of times to train on each piece of data)

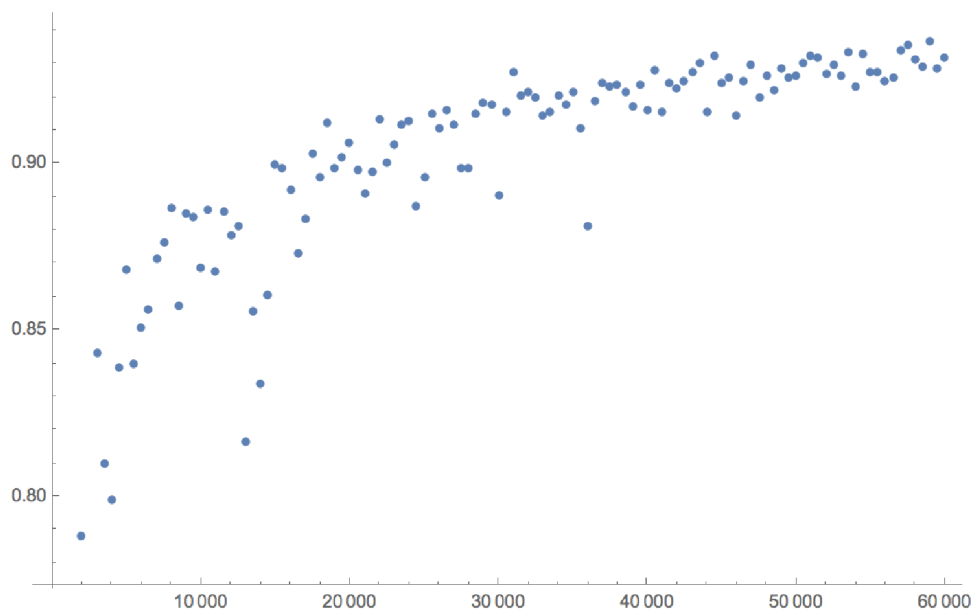
-We clarify a few things here:
(i) A **restart** is when we initialize our net with random entries in the weight matrices $\theta(1)$ and $\theta(2)$.
(ii) An **epoch** is when we run through all 60,000 training instances. If we perform k epochs, then we run through our 60,000 training instances k times without restarting.
(iii) We believe that Theorem 3.3 and empirical results empowers us to perform **multiple epochs** while still maintaining convergence to a local minimum.
It does not matter that training instance $\xi_{k-60,000}$ ($\xi_k \in$ epoch-2 or greater) is correlated with instance ξ_k because:
    a. Our training instances do not depend on the matrices $\theta(i)$.
    b. If we had infinite training instances Theorem 3.3 guarantees convergence to a θ^* (which depends on θ_0). We can view a new epoch as starting from a different set of initial weights which, by construction, must to take us to the same θ^*.
(iv) For some of our training we use a constant sequence $\{\epsilon_n\}$. We have not proven convergence for constant sequence $\{\epsilon_n\}$, however our results strongly suggest convergence. Furthermore we feel that we could make the argument that our constant sequences $\{\epsilon_n\}$ are in fact decreasing at such a rate where $\sum \epsilon_n = \infty$ and $\sum \epsilon_n^2 < \infty$ hold however the first instance k, such that $\epsilon_k - \epsilon_{k-1} \neq 0$ is such that k > N, where N is the total number of training instances.

RESULTS: We chose 60,000 instances for training and 10,000 for testing.
(our decreasing ϵ sequences were always of the form: $\{\epsilon_0, \frac{\epsilon_0}{2}, \frac{\epsilon_0}{3}, \ldots\}$)

Restart	Decreasing $\{\epsilon_n\}$	Decrease every n trains	ϵ_0	# hidden nodes	c	Best Accuracy / # epochs	Accuracy
1	No	–	0.1	397	1	12 / 15	97.01 %
2	No	–	0.01	397	1	15 / 20	97.52 %
3	**Yes**	**28, 000**	**0.05**	**397**	**1**	**23 / 30**	**98.05 %**
4	Yes	240, 000	.025	397	1	33 / 40	97.86 %
5	Yes	180, 000	0.1	800	1	16 / 20	97.28 %
6	Yes	180, 000	.05	397	1	37 / 40	97.73 %

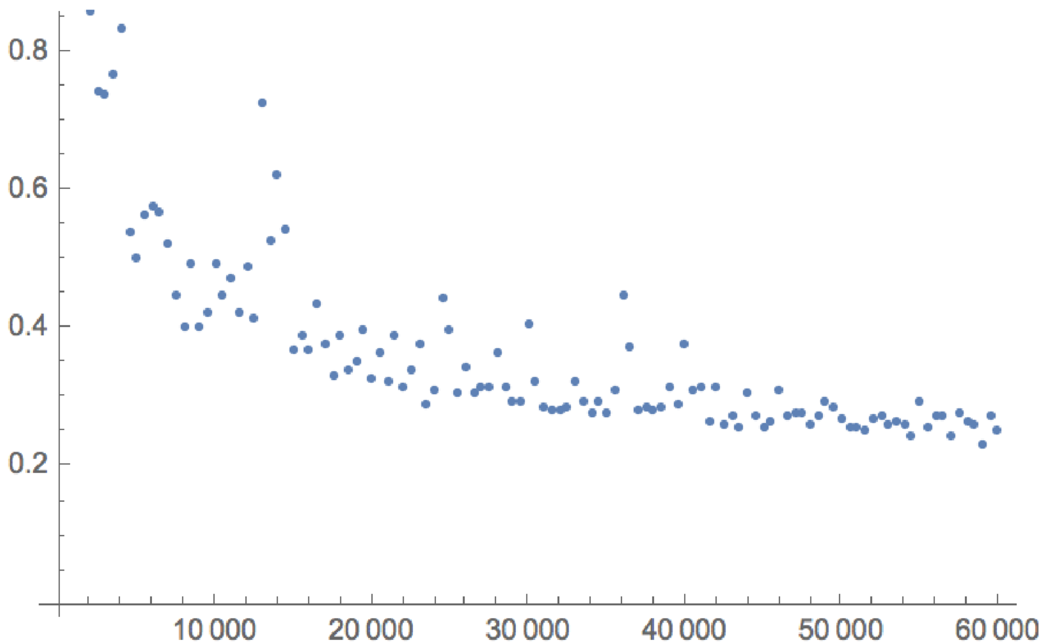
Analysis: Our best net achieved an accuracy of 98.05%. The following analysis is for this net:

The following graph is the accuracy for every 500 training instances over 1 epoch:



(Accuracy (where 1.0 = 100%) Vs. Number of training instances)

The following graph is the average ||T - A|| every 500 train instances over 1 epoch:



(Average Normed error Vs. Number of training instances)
Note: With 0 training the average normed error was 5.003

For each of the 195 failed instances, we have the following:

Target	Number of failed attempts
0	8
1	8
2	25
3	15
4	19
5	26
6	16
7	26
8	22
9	30

-For each of the 195 failed predictions the table below shows which value was the target and the value that was actually predicted:

Target / Prediction →	0	1	2	3	4	5	6	7	8	9
0	–	0	1	0	0	0	3	1	2	1
1	0	–	3	2	0	1	1	0	1	0
2	2	2	–	6	1	2	1	5	6	0
3	1	0	3	–	0	3	0	4	3	1
4	1	0	1	0	–	0	6	1	0	10
5	2	1	0	9	1	–	5	1	3	4
6	5	2	0	0	3	3	–	0	2	1
7	1	5	10	0	0	1	0	–	1	8
8	2	0	0	5	6	4	1	2	–	2
9	2	2	0	4	12	1	1	6	2	–

Below we show the number of times the correct answer was not predicted to be the second most likely:

Target	**# of times the target was not the second highest output**
0	4
1	3
2	6
3	4
4	2
5	8
6	6
7	10
8	9
9	12

In 131 of the 195 instances where the net predicted the wrong digit, the correct digit was considered the second most-likely candidate. For each of these cases the tables below show which value was the target and which value was actually predicted:

Target / Prediction →	0	1	2	3	4	5	6	7	8	9
0	–	0	0	0	0	0	3	0	1	0
1	0	–	3	1	0	0	0	0	1	0
2	2	2	–	6	0	1	1	4	3	0
3	1	0	2	–	0	3	0	2	2	1
4	1	0	1	0	–	0	5	1	0	9
5	0	1	0	7	1	–	3	0	3	3
6	2	2	0	0	3	2	–	0	1	0
7	0	1	6	0	0	0	0	–	1	8
8	1	0	0	1	4	2	1	2	–	2
9	2	1	0	0	9	0	0	4	2	–

Target	**Number of failed attempts**
0	4
1	5
2	19
3	11
4	17
5	18
6	10
7	16
8	13
9	18

-The table below shows how confident (difference between target output and the actual) the net was for each **incorrect** result, for each target.

Target	(0 – .1)	[.1 – .2)	[.2 – .3)	[.3 – .4)	[.4 – .5)	[.5 – .6)	[.6 – .7)	[.7 – .8)	[.8 – .9)	[.9 – 1.0)
0	4	0	0	0	0	0	0	0	1	3
1	1	0	0	3	0	0	0	3	0	1
2	2	1	6	1	1	0	4	1	2	7
3	2	4	0	0	0	2	2	1	0	4
4	5	1	1	1	1	2	1	2	2	3
5	4	2	2	4	2	2	0	2	1	7
6	2	1	2	2	1	0	1	2	1	4
7	5	1	1	4	0	2	1	6	1	5
8	5	3	2	3	2	1	1	1	2	2
9	4	5	1	3	2	2	2	1	2	8

-The table below shows how confident (difference between target output and next highest) the net was for each **correct** result:

Target	(0 – .1)	[.1 – .2)	[.2 – .3)	[.3 – .4)	[.4 – .5)	[.5 – .6)	[.6 – .7)	[.7 – .8)	[.8 – .9)	[.9 – 1.0)
0	3	3	1	3	2	5	5	10	12	928
1	6	2	1	2	2	2	5	10	16	1081
2	6	3	2	4	6	10	9	11	26	930
3	10	3	5	4	0	6	9	11	14	933
4	7	6	3	4	6	5	7	9	21	895
5	8	8	4	5	5	7	10	8	19	782
6	6	6	5	0	4	3	8	6	15	889
7	5	5	2	4	5	4	83	9	13	947
8	10	8	8	7	11	3	9	13	31	852
9	5	4	7	4	6	9	10	8	26	900

-Below is an averaging of all the pixel intensities of all the testing data in which the the net was 90% confident and the prediction was **correct**: