Linear Algebra for Deep Learning:

Matrix Factorization and a Matrix Perspective on Gradient Descent

Nick Kantack

Masters student, University of Virginia

ECE 695, Old Dominion University

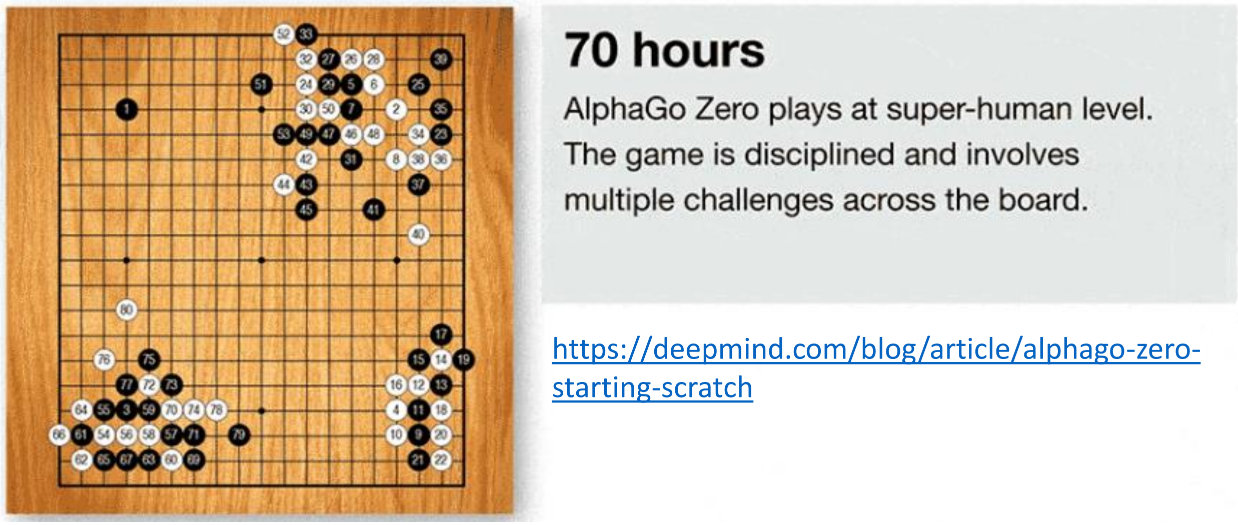
Deep Learning

- **Deep Learning** - machine learning applied to multilayered networks.
- Artificial neural networks provide superhuman problem solving

Artificial neural networks are very powerful because they have the ability to learn how to find patterns in the data by themselves without the need of a human programmer. Often times, these algorithms are taught to learn to take in the data and create new code that mimics the behavior of natural neurons.

- GPT2, and artificial neural network

NATURE: "Artificial intelligence is more accurate than doctors in diagnosing breast cancer from mammograms"
https://www.bbc.com/news/health-50857759#:~:text=Artificial%20intelligence%20is%20more%20accurate,images%20from%20nearly%2029%2C000%20women.



https://deepmind.com/blog/article/alphago-zero-starting-scratch

Gradient Descent

Scalar case

Define a loss function $L(\theta)$ for a parameter θ:

$$L(\theta) = (f(\theta) - y)^2$$

To reduce the loss function, *descend the gradient* of $L(\theta)$:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L$$

In the single parameter case $f(\theta)=\beta\theta$,

$$\nabla_\theta L = \frac{\partial L}{\partial f}\frac{\partial f}{\partial \theta} = 2(f(\theta) - y)\beta$$

Matrix case

For multiple parameters and multiple outputs, define

$$\boldsymbol{f}(\theta_{1,\dots,n}): f_m = \sum_{k=1}^{n} c_k \theta_k \quad \rightarrow \quad \boldsymbol{f}(\boldsymbol{\theta}) = W\boldsymbol{\theta} \quad W \in R^{m \times n}$$

With a new loss function defined as a Euclidean norm squared

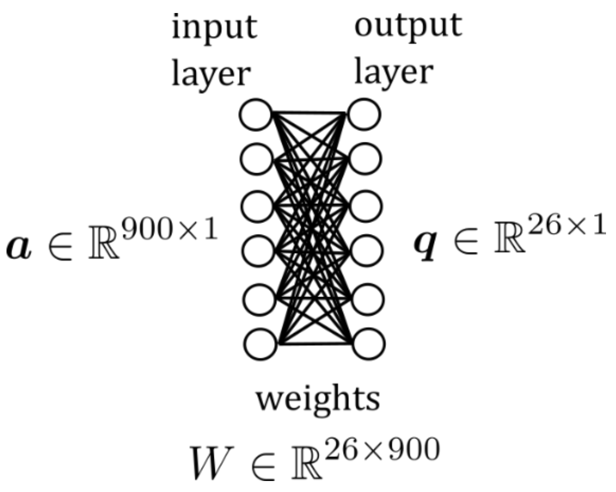
$$L = ||\boldsymbol{f}(\boldsymbol{\theta}) - \boldsymbol{y}||^2 = ||W\boldsymbol{\theta} - \boldsymbol{y}||^2$$

And a gradient of the loss function with respect to each parameter

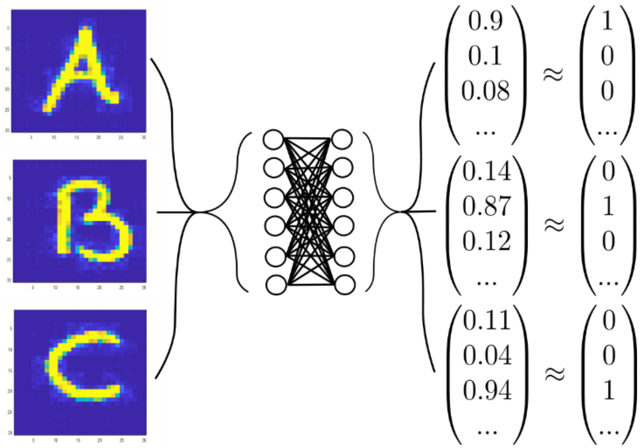
$$\nabla_{\theta_i} L = 2(W\boldsymbol{\theta} - \boldsymbol{y})\boldsymbol{\theta}_i$$

Case study I: A two layer neural network

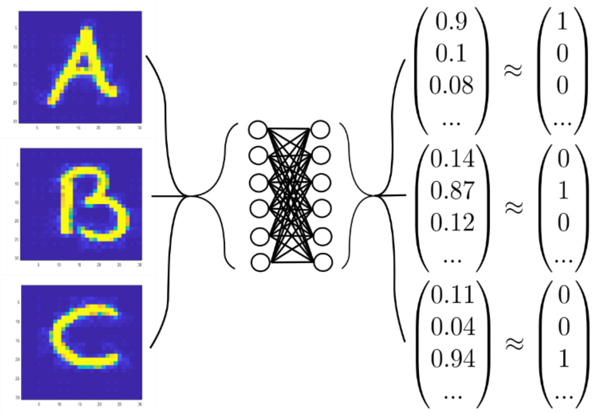
- Neural network for optical character recognition (OCR)
- Performed OCR on a dataset of 520 30px by 30px handwritten capital letters.

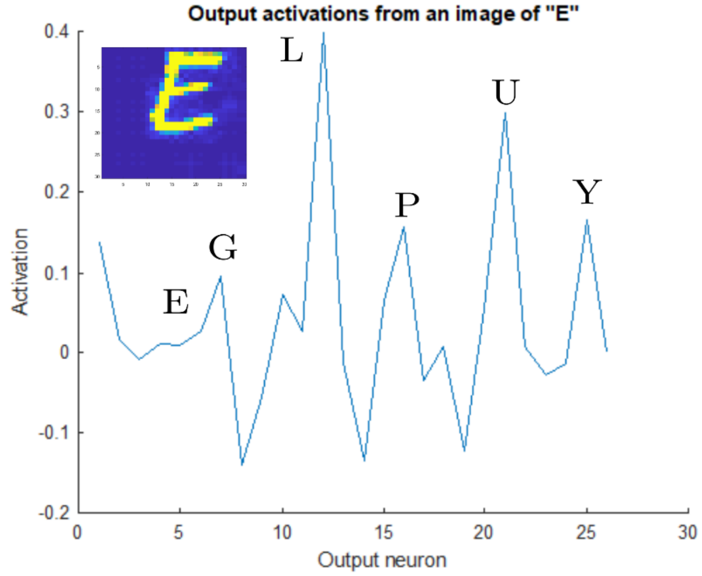


Learning step	Network Implementation
Forward propagation	$\boldsymbol{q} = W\boldsymbol{a}$
Loss function	$L = \lvert\lvert\boldsymbol{c} - \boldsymbol{q}\rvert\rvert^2$
Loss function gradient in weights	$\frac{\partial L}{\partial w_{ij}} = 2(q_i - c_i)a_j$
Back propagation	$W \leftarrow W - 2\alpha(\boldsymbol{q} - \boldsymbol{c})\boldsymbol{a}^T$



Matrix rank and the training samples





Perfect outputs are canonical unit vectors.

Define a matrix, A, of "perfect" letters, then

$$WA = I$$

Then a matrix, $\widetilde{W}$, of perfect weights would be

$$\widetilde{W} = (A^T A)^{-1} A^T$$

Then no learning is needed! But, *<u>$A^T A$ must be an invertible matrix.</u>*

Matrix rank and the training samples

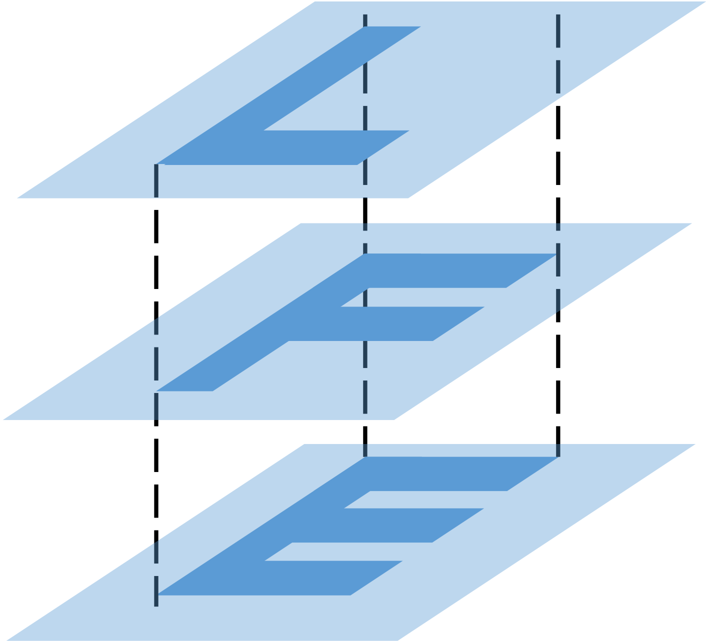
The columns of A are the pixels of handwritten letters.

For $A^T A$ to be invertible, A must be full column rank (rank(A)=26).

No column of A can be a linear combination of other columns.

Is this true for the letters? *No!*

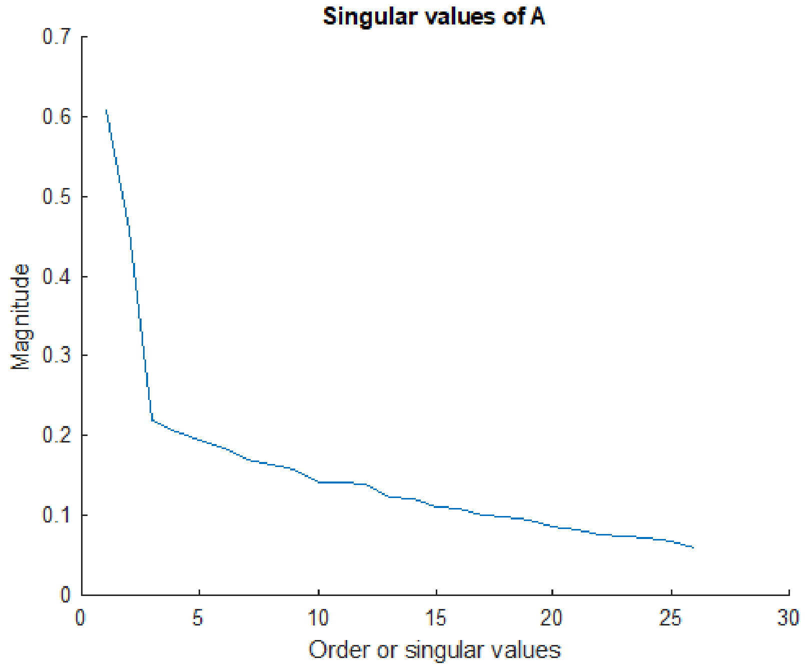
E = F + L



A is effectively low rank. How low?

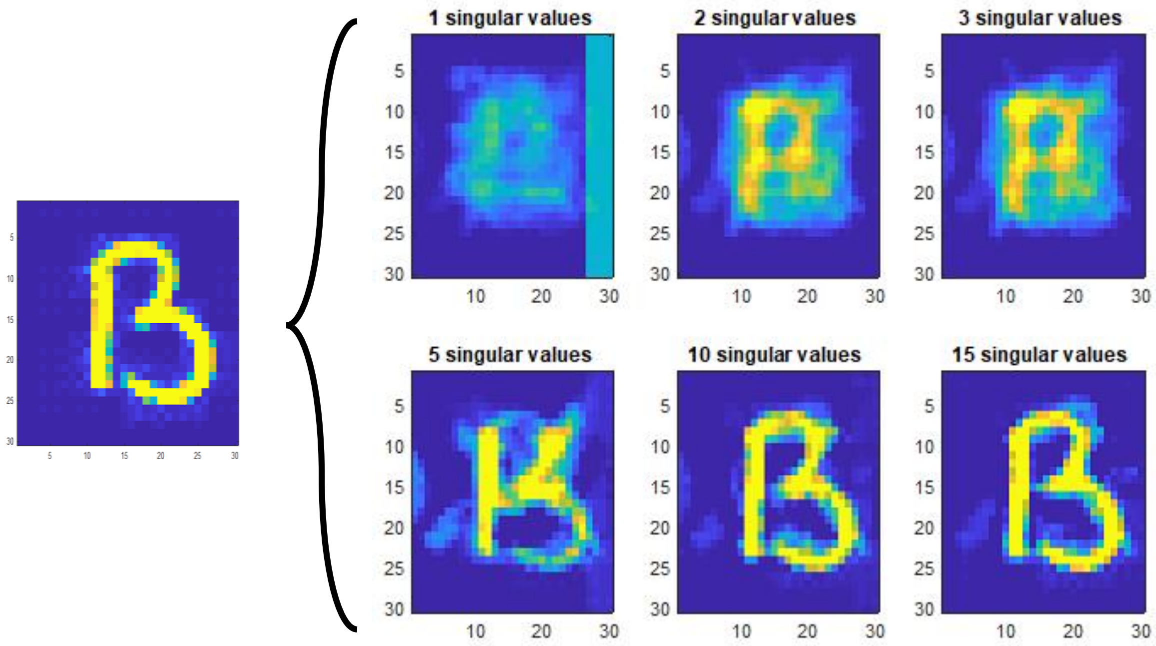
We can find an effective rank for A using SVD

$$A = U\Sigma V^T$$



We can examine low rank approximations for A

$$A \approx U(:,1{:}r)\Sigma(1{:}r,1{:}r)V^T(1{:}r,:)$$



Impact of low rank data

$$\boldsymbol{c}_E = \widetilde{W}\boldsymbol{a}_E \approx \widetilde{W}(\boldsymbol{a}_L + \boldsymbol{a}_F)$$

$$\boldsymbol{c}_E \approx \widetilde{W}\boldsymbol{a}_L + \widetilde{W}\boldsymbol{a}_F = \boldsymbol{c}_F + \boldsymbol{c}_L$$

This is a contradiction. The $\boldsymbol{c}$ vectors are chosen to be orthogonal by design. If A is of low effective rank, then $\widetilde{W} = (A^T A)^{-1} A^T$ is of low effective rank.

$$\widetilde{W}^{-1}\boldsymbol{n} = \epsilon \qquad ||\epsilon|| \ll 1$$

$$\boldsymbol{a}' = \widetilde{W}^{-1}(\boldsymbol{q} + \boldsymbol{n})$$

$$\boldsymbol{q}' = \widetilde{W}^{-1}\boldsymbol{q} + \widetilde{W}^{-1}\boldsymbol{n}$$

$$\boldsymbol{a}' = \boldsymbol{a} + \boldsymbol{\epsilon}$$

Significantly different classification ($\boldsymbol{q} + \boldsymbol{n}$ and $\boldsymbol{q}$) can result from nearly identical input images ($\boldsymbol{a}'$ and $\boldsymbol{a}$). <u>*One possible resolution is to add another layer of neurons.*</u>

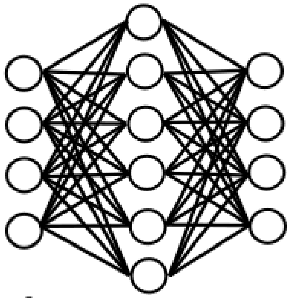
Case study II: A three layer neural network

- Similar to the two layer network, except:
  - Added a hidden layer
  - Added biases for hidden and output layers
  - Added thresholding function
  - Two step back propagation

Learning step	Network Implementation
Forward propagation	$\boldsymbol{q} = \sigma(W_q \sigma(W_h \boldsymbol{a} + \boldsymbol{b}_h) + \boldsymbol{b}_q)$
Loss function	$L = \|\|\boldsymbol{c} - \boldsymbol{a}\|\|^2$
Hidden layer errors	$\boldsymbol{v} = {W_h}^T (\boldsymbol{q} - \boldsymbol{c}) \sigma'(\widetilde{\boldsymbol{h}})$
Back propagation (W_q)	$W_q \leftarrow W_q - 2\alpha(\boldsymbol{q} - \boldsymbol{c})\boldsymbol{h}^T$
Back propagation (W_h)	$W_h \leftarrow W_h - 2\alpha \boldsymbol{v} \boldsymbol{a}^T$

hidden layer $\boldsymbol{h} \in \mathbb{R}^{1800\times 1}$

input layer $\boldsymbol{a} \in \mathbb{R}^{900\times 1}$



output layer $\boldsymbol{q} \in \mathbb{R}^{26\times 1}$

input weights $W_a \in \mathbb{R}^{1800\times 900}$

hidden weights $W_h \in \mathbb{R}^{26\times 1800}$

hidden biases $\boldsymbol{b}_h \in \mathbb{R}^{1800\times 1}$

output biases $\boldsymbol{b}_o \in \mathbb{R}^{26\times 1}$

$\widetilde{\boldsymbol{h}}$ - hidden activations before thresholding

$$\boldsymbol{\sigma}(\boldsymbol{x}):\ \sigma_i = \frac{1}{1 + e^{x_i/150}}$$

Impact of low rank data: hidden layer

Can significantly different outputs arise from similar inputs?

$$\boldsymbol{q} = W_q W_h \boldsymbol{a}$$

$$W_q{}^{-1}\boldsymbol{n} = \epsilon \qquad ||\epsilon|| \ll 1$$

$$\boldsymbol{a}' = W_h{}^{-1} W_q{}^{-1}(\boldsymbol{q} + \boldsymbol{n})$$

$$\boldsymbol{q}' = W_h{}^{-1} W_q{}^{-1}\boldsymbol{q} + W_h{}^{-1} W_q{}^{-1}\boldsymbol{n}$$

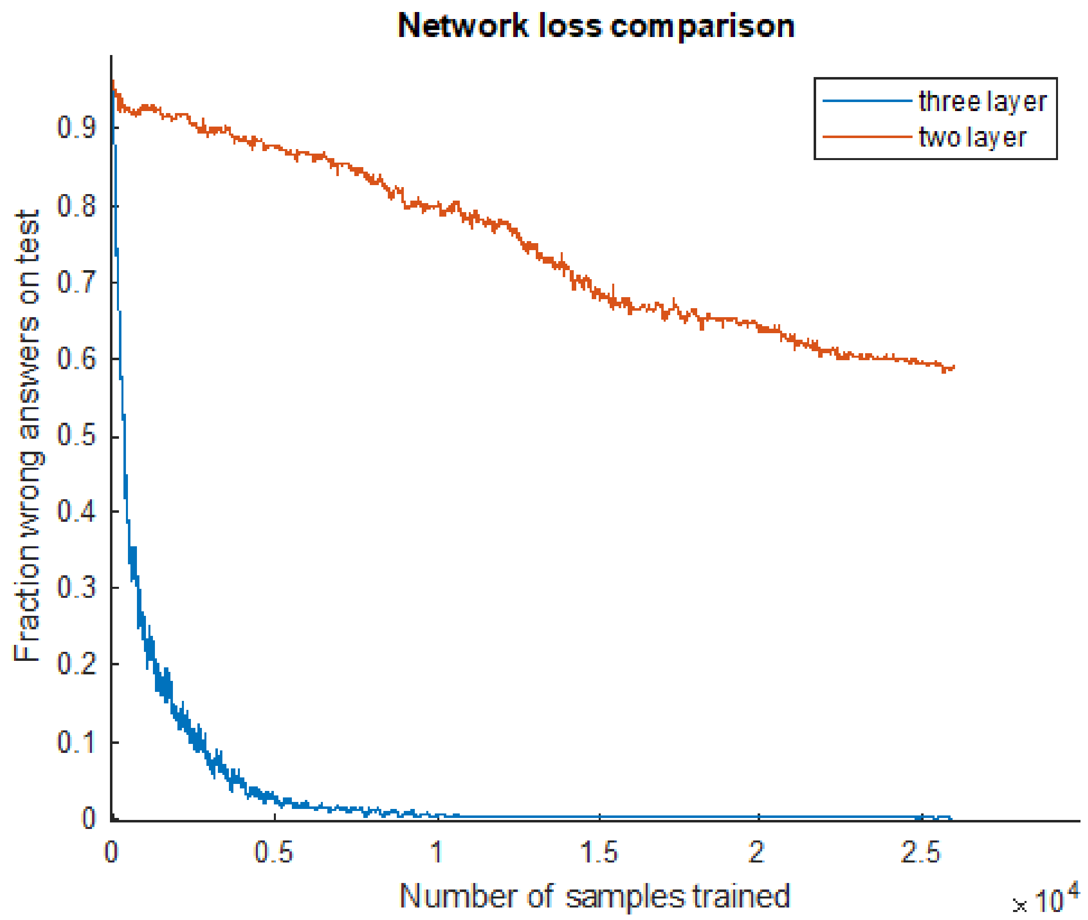
$$\boldsymbol{a}' = \boldsymbol{a} + W_h{}^{-1}\boldsymbol{\epsilon}$$

The matrix $W_h{}^{-1}$ is 1800×1800, so $W_h{}^{-1}\boldsymbol{\epsilon}$ has elements drawn from a normal distribution of mean 0 and standard deviation $\sqrt{1800} \approx 42$. So the hidden layer has the effect of amplifying the rank of A.

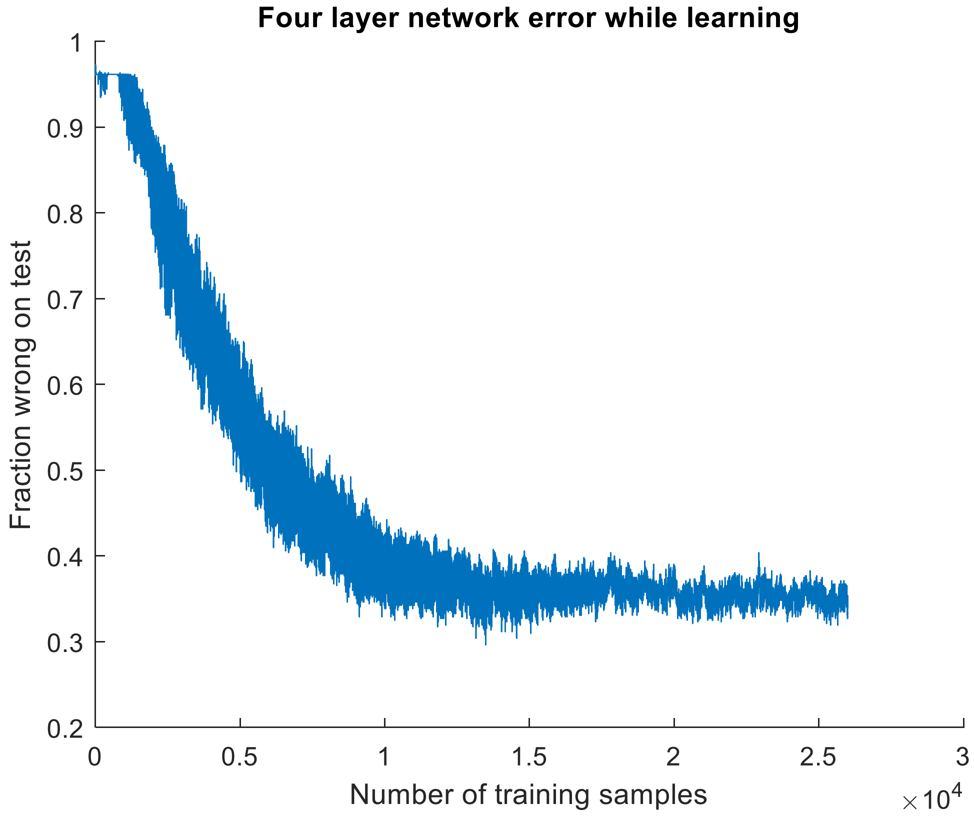
Comparison of network performances

- Three layer network has higher convergence rate in both time and samples

- The accuracy of the three layer network converged to 100% for the sample data test.



A four layer neural network?



- Would adding a fourth layer improve performance?
- Adding a fourth layer raises the number of weights from 5 million to over 8 million.
- A fourth layer slows convergence, without providing more solution flexibility than a three layer network.

Beyond vanilla gradient descent

- More elaborate gradient descent:
  - Stochastic gradient descent
  - Higher order gradient descent
  - Gradient descent gradient descent

Questions?