Weight matrix movement in classification networks

Suggestions on different ways to supervise learning

Volodymyr Boiko
PW MiNI

February 5, 2025

The Basics

Our neural networks take images of handwritten digits from the MNIST dataset and try to classify them.

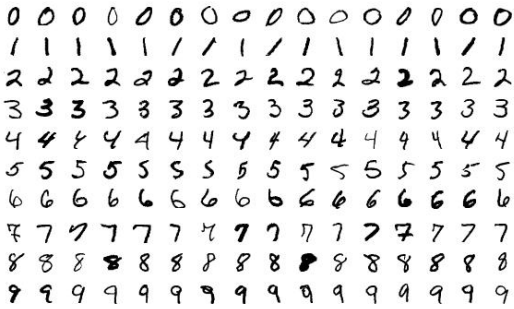


Figure: Example images from MNIST

The Basics 2

Essentially, a fully trained network is a series of matrix multiplications, additions, and non-linear function applications.

Our network uses three layers: the input layer with 784 neuros, one for each pixel of an image from MNIST, one hidden layer with a varying number of neurons, and the output layer with 10 neurons, one for each digit.

Before training a network, we can specify the amount of neurons in the hidden layer, the activation function for it, learning rate and the method of generating initial weights (more on that later).

Relevant definitions

Weight matrices

A weight matrix is composed of multiple vectors, one for each neuron, which determines the weight of every individual input point.

Activation function

An activation function determines the output of a neuron. It is usually nonlinear to ensure non-linearity of a network's output.

Bias

A neuron's bias is a value that is added to the weighted input before applying the activation function.

Forward propagation

Notation

Here, X is input data, W_1 and W_2 are weight matrices, b_1 and b_2 are bias vectors, Z_1 and Z_2 denote weighted input offset by bias and A_1, A_2 are neuron outputs. $ReLU$ and $Softmax$ are example activation functions.

First, multiply all input by weights of the inner layer and shift using the bias vector:

$$Z_1 = W_1 X + b_1$$

Apply the activation function in the first inner layer:

$$A_1 = ReLU(Z_1)$$

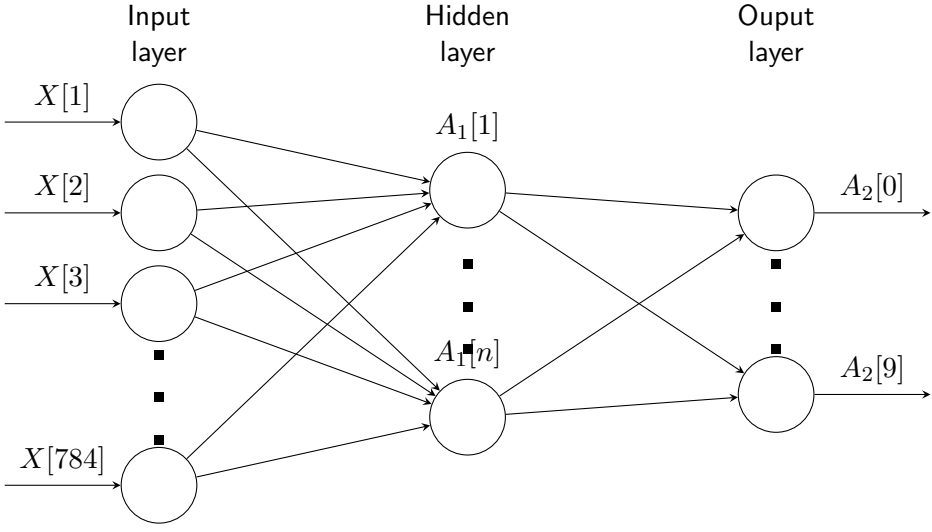
Pass the activated output to the next layer, weight it and adjust for bias again:

$$Z_2 = W_2 A_1 + b_2$$

Activate one last time to get final results (probability of your image being a certain digit):

$$A_2 = Softmax(Z_2)$$

Forward propagation 2



How do we train

Obviously, just generating random weights won't amount to much. The network needs to be trained via a process called *backward propagation* which involves a lot of math and is beyond the scope of this project.

While training, we forward propagate all available data points (excluding the test set), compare our results to what is expected and then adjust weigh matrices and bias vectors based on that information.

The fun stuff

Let's look at the weight distribution of $W1$ (the weight matrix of the inner layer) with 100 neurons trained for 1000 epoches. The starting weights are generated using normal distribution.

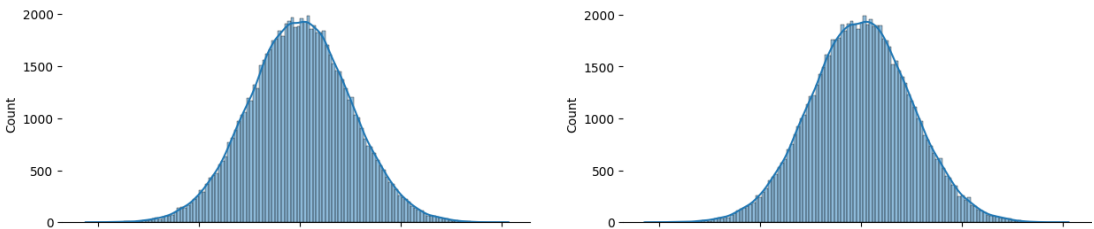


Figure: neurons=100, dist=normal, epoches=1000, $\eta = 0.5$

Weight distribution 2

Let's also look at the weight distribution of $W1$ (the weight matrix of the inner layer) with 100 neurons trained for 2000 epoches. The starting weights are generated using uniform distribution.

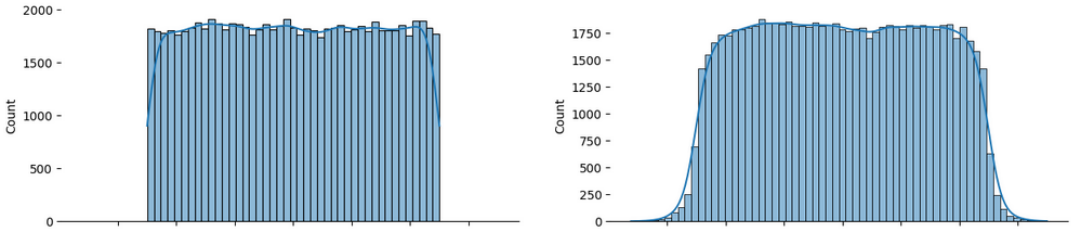


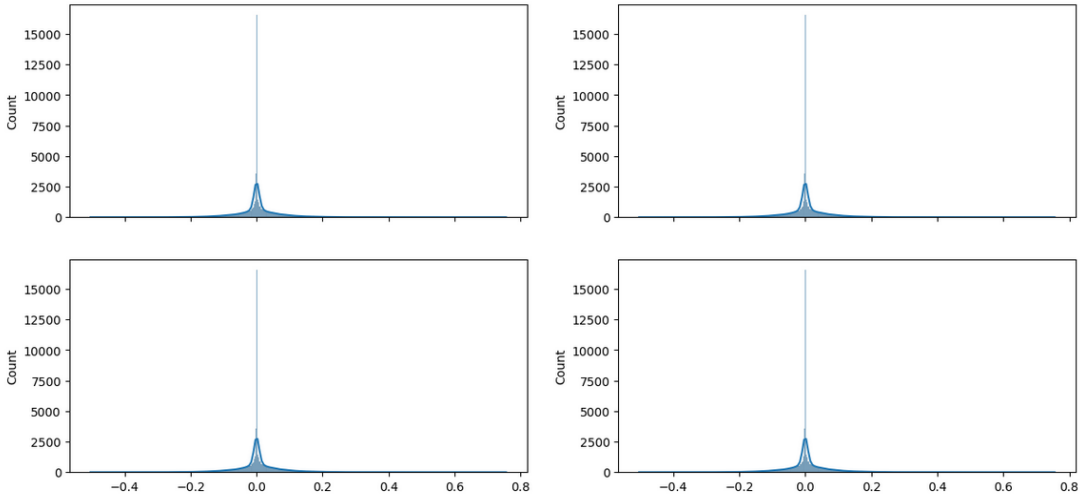
Figure: neurons=100, dist=uni, epoches=2000, $\eta = 0.5$

Weight distribution 3

We can be certainly confident that in these two specific cases, the distribution we used to generate weights is more or less preserved. While the normal distribution appears to be entirely unaffected, the uniform distribution suffers more, but it is not distorted enough to abandon any notion of uniformity.

After training four networks with the same parameters and size (100 neurons, 1000 epoches) but different activation functions and different distributions used for weights, we can build a histogram of how much individual entries of the weight matrix change.

Weight movement distribution



What's all this about then

It is clear that even after a 1000 iterations the weight matrices don't move around too much: the histogram of weight movement distribution overwhelmingly peaks around 0. But how can we quantify this change without resorting to quantitative analysis?

The Basics 3

A matrix $W = (w_{ij})$ with n rows and m columns with entries in $\mathbb{R}$ can be treated as a vector in $\mathbb{R}^{mn}$. Consequently, we can define $|W|$ as the length of its vector in $\mathbb{R}^{mn}$:

$$|W| = \sqrt{\sum w_{ij}^2}$$

Similarly, we can define *distance* between two matrices A and B of the same dimension as the length of their vector difference:

$$d(A, B) = |A - B| = \sqrt{\sum (a_{ij} - b_{ij})^2}$$

Finally, the weighted norm $||W||$ can be defined as $\frac{|W|}{mn}$.

Distance and weighted distance

For the four big networks we trained previously, we can calculate the distance their biggest weight matrix (W_1) traveled after the training was finished.

Distribution	$\eta = 0.5$	$\eta = 2$
Uniform	11.398360	19.430574
Normal	19.270005	19.674065

Table: Distance traveled for different networks

With the exception of the network trained with a learning rate $\eta = 0.5$ on uniformly distributed starting weights, the rest traveled more or less the same distance, suggesting some sort of *convergence* under optimal conditions. If we keep training the network that is lagging behind, it will eventually reach that same distance.

But is this repeatable?

Turns out, yes! Networks trained on the same data set with the same parameters tend to converge to the same distance. For example, for 20 neurons and 200 iterations distances traveled look like this:

W_1	b_1	W_2
16.417	6.532	12.777
16.620	2.439	13.668
16.559	1.484	12.987
16.908	1.125	12.563
16.201	4.499	12.469
17.025	2.129	11.952
16.752	0.159	12.290
16.847	3.873	13.419
16.515	1.214	12.588

Table: Distance traveled for different networks

Repeatability 2

This suggests that for any given number of neurons and initial distribution with a fixed learning rate there may be an optimal distance traveled for weight matrices, after which learning is no longer advisable.

This begs the question: What happens when we allow some of the parameters to change, while fixing other in place? Will distance traveled still converge to some static point?

Distance moved based on epoches (same network)

Let's look at a simple network with 10 hidden neurons with a learning rate $\eta = 0.5$ with $ReLU$ as its activation function. We will continuously train it for 1000 epoches and check what happens to W_1 after every 10 epoches.

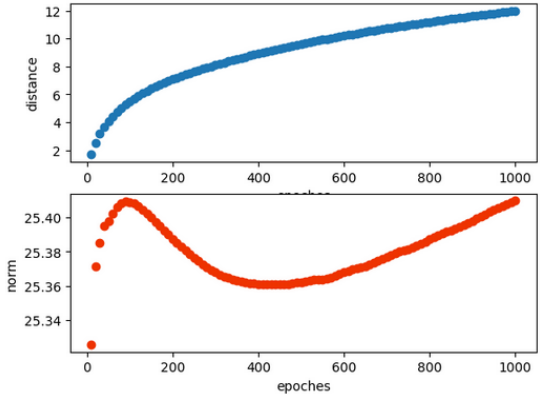


Figure: Distance and norm based on epoches

Distance moved based on epoches (same network)

As we can see, distance traveled increases, with it's rate of growth slowing down as time goes on. This in combination with The Law of Big Numbers suggests that the longer you train a network, the more distorted initial weights become and eventually the distribution loses shape.

Contrary to that, after a sharp initial adjustment, the norm reaches a local minimum, after which it starts growing in a linear manner. This *may* suggest that around 500 epoches would be the optimal amount of training for a network with this configuration.

Let us now look at the same graph, but for networks with a much higher learning rate η.

Distance moved and norm in faulty networks

It is apparent that a maximum is reached much faster than in previous networks, which trained more slowly. After epoch 200 minute changes have been observed only in bias vectors, and weight matrix changes are negligible (almost constant).

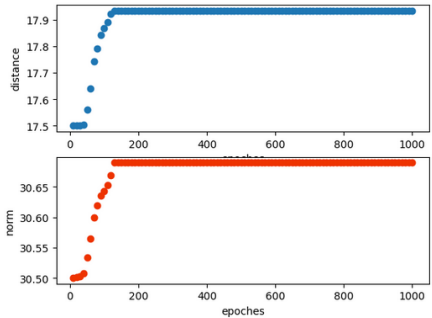


Figure: Distance and norm for a network with a learning rate $\eta = 4$

Distance moved and norm in faulty networks

For higher values of η we can see an even more drastic plateau:

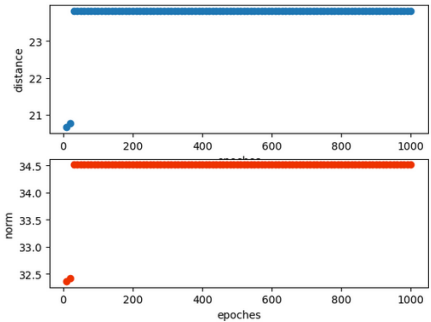


Figure: Distance and norm for a network with a learning rate $\eta = 5$

Even for a learning rate of $\eta = 5$ the convergence occurs much earlier in the training process, while accuracy falls drastically. For $\eta = 10$ the process is almost instantaneous.

Distance moved based on epoches (different networks)

If, instead of training one network continuously, we train a different network with the same configuration for every number of iterations we can still see the convergence in distance traveled, although not as smooth.

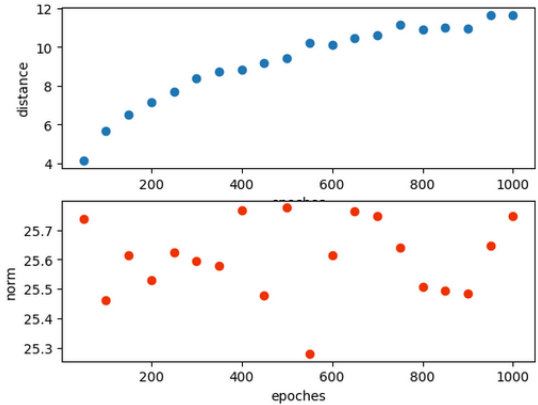


Figure: Distance and norm based on epoches

Distance moved based on epoches (different networks)

Distance traveled here follows the same trend, increasing with a slowly decreasing rate of growth. Norm also achieves a local minimum around 500 epoches, but is much more volatile across different networks overall.

This suggests that distance traveled is a stable metric across different variations of the same network, while norm may be subject to higher variance, but generally follows the same rules.

Use cases (1)

Based on our findings, how do we determine during training if something is not right? Some obvious tells would be:

1. Decrease in distance traveled.
If during training distance traveled from initial weight matrix starts going down instead of up, something may very well be wrong. (This applies to a consistent decrease, not just a single datapoint out of order.) For a correctly configured training process this would mean either overtraining or an overly big learning rate.

Use cases (2)

2. Sharp change in norm/distance.
Generally speaking, for all *well-performing* networks we trained, norm and distance don't really display sharp changes outside of the initial iterations. Even then, norm usually stays within a certain range of the initial norm, which tells us four different things:

- if norm starts jumping around, while staying within range: you may be overtraining.
- if norm starts jumping around, leaving the range: your learning rate may be too high. This should usually be reflected in distance going haywire too, since high learning rates mean bigger changes done to weight matrices.
- if norm keeps growing and leaves the range entirely: there may be something wrong with your training process, or you are overtraining.
- if distance starts jumping around, your learning rate may be too high. If it plateaus, you may have started over.

Use cases (3)

3. Drastic change in distribution.
This only applies to the two distributions we've covered here: uniform and normal. For uniform distribution, the loss is indicated by the histogram becoming more and more bell-like. For normal distribution, loss may be indicated by change in distribution variance, mean or general distortions in the cumulative distribution function.
To enable automatic detection of the loss of distribution you can use different tests, such as the chi square test for uniform distributions, or the Kolmogorov-Smirnov test for normal distributions.

Change in distribution does not mean change in distance

Generally speaking, loss of distribution does not necessarily imply change in distance or norm. Change in norm, however, should mean something changed in the distribution.