Neural Network Notes

Lesson 5: Sentiment Analysis

Framing the Problem

Neural networks search for correlations between datasets
In order for them to train anything, we need to present them with two meaningful datasets: what we know, and what we want to know
Eventually, it'll take one and learn to predict the other

First, we want to develop a predictive theory - "If I were the neural network, where would I look for correlations in the datasets?"

Transforming Text into Numbers

We want to present the words as input into the neural network in such a way that it can look for correlations and make the correct positive/negative prediction of the output

The simplest way is to count each word, and implement those counts as inputs

We represent positiveness and negativeness by 1 and 0
We do this with a single neuron because positive and negative are mutually exclusive
This reduces the number of way a neural network can make a mistake, which reduces the amount it has to learn and helps it learn this particular pattern

In cases where the dataset has 5 different output labels, for example 1-5 stars, this can sometimes hurt the neural network and make it more difficult to predict if it has to predict which start was most likely
This is because it allows it to kind of make double positive predictions
The positive outputs share a lot of signals, and therefore creates ambiguity in the network

In this case, because we only have 2 labels, we can force the network to have to choose between the two of them - thus reducing the escape paths

We want to frame the problem in such a way that it's makes it easiest for the neural network to make a prediction

---

Better Weight Initialization Strategy

The general rules for setting weights in a neural network is to set them close to 0 without being too small

We should start our weights in the range $[-y, y]$ where, $y = \frac{1}{\sqrt{n}}$

Where n is the number of inputs to a given layer

For example, between layers 1 and 2, we should define the weights as a function of the number of nodes n in the hidden layer
Note: Layer 1 is the hidden layer, NOT the input layer, and layer 2 is the output layer

Understanding Neural Noise

The whole game is that we have a ton of data, we know there's a pattern in it, and we want the neural network to be able to find it

Increasing signal and reducing noise is all about making it more obvious for our neural network so it can get to work handling the signal and combining it in interesting ways to look for more difficult patterns

Understanding Inefficiencies in our Network

We want to identify and remove the parts of our network that isn't necessary
A common inefficiency are huge dot products of mostly zero vectors

It's all about stripping out things that the neural network is doing that isn't helping it predict and learn

Further Noise Reduction

The more we can increase the signal to noise ratio, in other words help the neural network cuz through the really obvious stuff so it can focus on the really difficult stuff, the better the neural network will be able to train

We want to heavily weigh the data that have a lot of predictive power

In NLP, it's common to eliminate data that happens really frequently because it happens so much that it doesn't give us much signal, and really infrequently because if it only happens once that's not a pattern and there's no correlation

We're framing the problem to make the correlation as obvious as possible to the neural network so that it has the best chance to ignore noise and find signal

Reducing noise is often the only strategy to both increase the speed and accuracy
Usually there's a trade off

If we have so much training data that we could never train over all of it, we'd also want to increase the speed
Just bring able to run faster over more data, even if we choose a more naive algorithm, makes the accuracy really high

Analysis: What's Going on in the Weights?

When we have a linear layer that's making predictions, the weights before the output layer are the feature detectors - they're trying to detect a certain state in the neurons before the output layer (in the last hidden layer)

For example, if the weight is a big negative number, it's looking for the prior neuron to be a big negative number

There are certain hidden layers that cause the weights to output a high/low number
If all the last weights and all the last neurons are positive, it puts a lot of pressure for the output to be positive
Similarly, if all the last weights and all the last neurons are negative, it puts a lot of pressure for the output to be positive

In other words, when neurons and weights have the same polarity, and have high values in all the same places, it makes a high prediction
Conversely, if the last weights and last neurons have opposite polarities, the output will likely be negative

Groups of positive/negative data points are being trained to manipulate the feature detector weights in the same way, to create a high prediction or low prediction

When we back-propagate gradients and update weights so that groups of positive/negative words exhibit the same phenomenon
More specifically, positive/negative data points are are supposed to affect the output in the same way
As we update their weights to be able to do that, their weights become similar because both of them have the same goal

Back-propagation is all about the output/prediction node being able to tell certain weights hot to be better or more accurate
Since it's sending the same signal to all positive/negative data points, they end up grouping together