Neural Networks: An Exploration

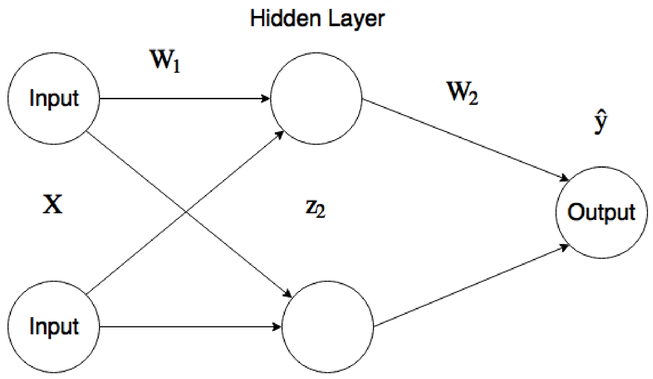
David, Patrick, Philip

More info here: github.com/neuralolin

What is a Neural Network?

A Neural Network is a machine learning model roughly based on the makeup of the brain. A NN is comprised of a series of **layers**. Each of these layers is comprised of **neurons**. Each neuron has an output, known as an **activation**, which is simply a linear combination of the outputs (activations) of the previous layers.

The connections between the neurons are often called **synapses**. The associated values of all the synapses between two layers comprise a **weight matrix**.



Making Predictions

Above we have a simple neural network, which takes in two inputs, and returns one output. Assuming 3 samples, X is a 3x2 matrix. W_1 is a matrix of synapse coefficients. There are four synapses between the first two layers. W_1 is a 2x2 matrix, where each row corresponds to the synapses coming out of one of the input neurons. In this neural net:

$$z_2 = XW_1$$

$$\hat{y} = z_2 W_2$$

We're done! This is now a neural net capable of making predictions.

Activation functions

Perhaps at this point you have astutely observed that currently our Neural Network is basically just a linear regression model, since the output is a linear combination of the inputs. It's time to introduce one of the most important features of neural networks: **nonlinearity**. Most neural network layers apply a activation function to their activations before passing on their output to the next layer. This transformed activation is known as the layer **activity**. All we have to do to make our network capable of predicting nonlinear relationships is update our equations:

$$a_2 = \tanh(z_2) = \tanh(XW_1)$$

The hyperbolic tangent function scales each activation from -1 to 1.

Sources and more information (including explorations into more depth on each topic) can be found in our repo here:
https://github.com/neuralolin/DataScience16FinalProject

The readme functions as the index, with everything explored laid out from there.

Training a Neural Network

Making predictions with NNs is great, but it's not worth much unless we can improve our predictions. Gradient descent to the rescue!

Gradient Descent and Neural Networks

With random weights, a NN is pretty terrible at making predictions. To improve our model, we first need to quantify exactly how wrong our predictions are. We'll do this with a cost function. A cost function allows us to express exactly how wrong or "costly" our models is, given our examples. A common way to compute an overall cost is to take each error value, square it, and sum the values.

$$J = \Sigma \frac{1}{2}(y - \hat{y})^2$$

This cost is a function of two things - the input data, and the weights on the synapses. We can't change the data, so we'll improve the accuracy by modifying the weights!

Conceptually, what we'll be doing is computing derivatives of the cost with respect to each weight matrix. We'll use these derivatives to compile a gradient, which we can then use to change the weights incrementally as our network improves!

Applying Gradient Descent to Train a Network

Let's get to applying gradient descent to improve and train a neural network. Assuming we have a network with one hidden layer and according weight matrices W_1 and W_2.

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\partial \sum \frac{1}{2}(y - \hat{y})^2}{\partial W^{(2)}}$$

Taking the derivative of this expression (temporarily setting aside the summation) simply involves a lot of chain rule. Backpropagation - don't stop doing the chain rule, ever.

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y})\frac{\partial \hat{y}}{\partial W^{(2)}}$$

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y})\frac{\partial \hat{y}}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial W^{(2)}}$$

$$\frac{\partial z^{(3)}}{\partial W^{(2)}} = -(y - \hat{y}) f'(z^{(3)})\frac{\partial z^{(3)}}{\partial W^{(2)}}$$

Finally, we need to investigate the relationship between z^3 and W^2. A way to think about what's going on here is that we're "backpropagating" the error to each weight.

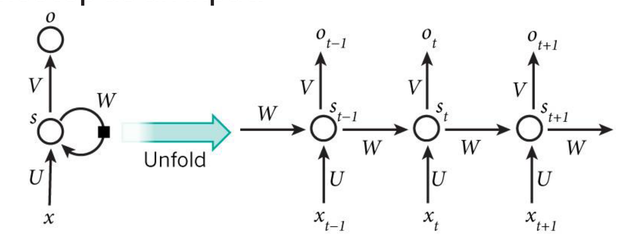
$$\frac{\partial J}{\partial W^{(2)}} = (a^{(2)})^T \delta^{(3)}$$

$$\delta^{(3)} = -(y - \hat{y}) f'(z^{(3)})$$

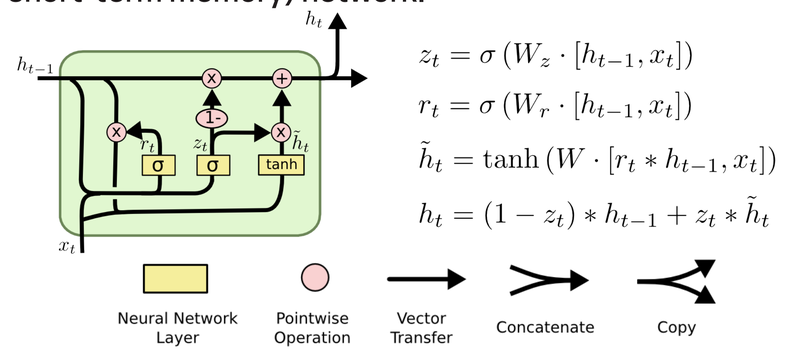
$$\frac{\partial J}{\partial W^{(1)}} = X^T \delta^{(3)} (W^{(2)})^T f'(z^{(2)})$$

LSTMs

NNs have another fascinating use case. They are also able to learn patterns in sequential data. A neural network that learns sequential data is known as a **Recurrent Neural Network**. The only difference is that at each time step, the hidden layer receives the hidden layer from the previous timestep as an input.



A downside to this basic implementation is the fact that the network quickly forgets information from several timesteps ago. To overcome this, many people use a special implementation of an RNN known as an LSTM (long short-term memory) network.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTMs rely on a **cell state**: the horizontal line running through each timestep. This line is like a "conveyor belt," carrying information across timesteps. The information that is stored or changed on the cell state is dictated by a forget gate. In short, the network learns what to remember, and what to forget.

Implementations

Predicting the Weather

One of the neural net implementations we did was training and then running a time-series of daily weather data through a LSTM. Our results were okay (better than a simple baseline model, but not much better). Here's a snapshot of our model's output, after being trained on 14-day sliding windows of weather data.

Text Generation using an LSTM Network

Another neural net implementation we explored was a character-level NLP LSTM network. We trained the network on several different books from Project Gutenberg. Below are some outputs of the network at the later stages of training. Note that we used a seed text of 'The quick brown fox jumps' to give the network a starting point to generate from. Sample outputs below:

"The quick brown fox jumpstence in his friends. The unaveruaties of the period of the family were in town, before I feet the state of his concern, and was as for a man every dear friends in minet seeing the lady for shem allow..."

"The quick brown fox jumpst to be affectionate supposition than he was the compliment of she seemed of her father and more certain that he was not so well groding to ask you, I would not contrive to her as to the particulars o..."

"The quick brown fox jumpster, the children which his advice, as he had been so fortunate as to be dinner with him to say that his addressing with the matter; and the evening, however, soon afterwards in their character, she s..."