Deep Learning - Feed Forward Neural Networks

MInDS @ Mines

Deep learning is a set of tools in machine learning that have recently gained traction due to their ability to achieve significantly superior results compared to other methods. Deep learning is simply the use of neural networks for any of the models we develop and in this lecture we discuss how neural networks work and how they're trained. We will also cover the simplest version of neural networks, the feed forward neural network.

Neural Networks

Deep learning is a set of tools that can be applied to machine learning problems. Deep learning is simply utilizing neural networks to solve machine learning problems. Neural networks are graphs of interconnected neurons with flow from one end to another. Neurons are organized in layers with the first layer being the input layer and the final layer being the output layer. Intermediate layers are called hidden layers and the reason deep learning is named as such is due to our ability to add many, many hidden layers creating "deeper" networks and finding that deeper networks may perform better.

By adding more hidden layers, we increase the learned parameters of the model and reduce its interpretability. This additional model complexity comes at a cost in both interpretability and computation however it is also what allows the model to achieve better results. Given the recent increase in computation power and the efficiency and applicability of GPUs to deep learning, we are able to incur the computation cost in order to create a better model. However, model interpretability is still a cost that we pay with deep learning methods.

Neurons

Neural networks are a graph made up of interconnected neurons where connections represent the passing of data from one neuron to another. Each neuron receives a vector of input values, x and produces an output y. The output of a neuron then becomes the input to the next neuron or may be the output value from the network as a whole. The neuron also has a vector of weight values, w, that are the parameters it learns during training. The output of the neuron can be calculated based on its inputs and weight as,

$$\mathsf{y} = f(\sum_{i=1}^{n} w_i x_i) = f(\mathsf{w}^T \mathsf{x}), \tag{1}$$

where f is the activation function of that neuron. Activation functions and the network architecture are the key hyperparameters to a neural network.

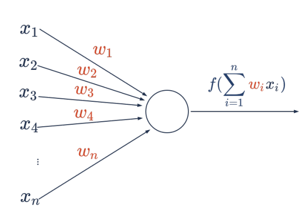

Figure 1: Example of a neuron calculating its output.

Activation Functions

There are many variations of activation functions that we could use for the neurons and each of those will produce varying results. Some commonly used activation functions are the sigmoid function, the step function, softsign, hyperbolic tangent, rectified linear units (ReLU), and leaky ReLU. Each of those have their own advantages and disadvantages.

In practice, most people these days find that hyperbolic tangent and rectified linear units are most effective.

$$f_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}}, \qquad f'_{\text{sigmoid}}(x) = f_{\text{sigmoid}}(x)(1 - f_{\text{sigmoid}}(x)), \tag{2}$$

$$f_{\text{step}}(x) = \begin{cases} 0 & x < 0, \\ 1 & x \geq 0, \end{cases} \qquad f'_{\text{step}}(x) = \begin{cases} 0 & x \neq 0, \\ \text{undefined} & x = 0, \end{cases} \tag{3}$$

$$f_{\text{softsign}}(x) = \frac{x}{1+|x|}, \qquad f'_{\text{softsign}}(x) = \frac{x}{(1+|x|)^2}, \tag{4}$$

$$f_{\text{hyp. tangent}}(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x - e^{-x}}, \qquad f'_{\text{hyp. tangent}}(x) = 1 - f_{\text{hyp. tangent}}(x)^2, \tag{5}$$

$$f_{\text{ReLU}}(x) = \text{max}(0, x), \qquad f'_{\text{ReLU}}(x) = \begin{cases} 0 & x < 0, \\ 1 & x \geq 0 \end{cases} \tag{6}$$

$$f_{\text{Leaky ReLU}}(x) = \begin{cases} 0.01x & x < 0, \\ x & x \geq 0, \end{cases} \qquad f'_{\text{Leaky ReLU}}(x) = \begin{cases} 0.01 & x \neq 0, \\ 1 & x = 0, \end{cases} \tag{7}$$

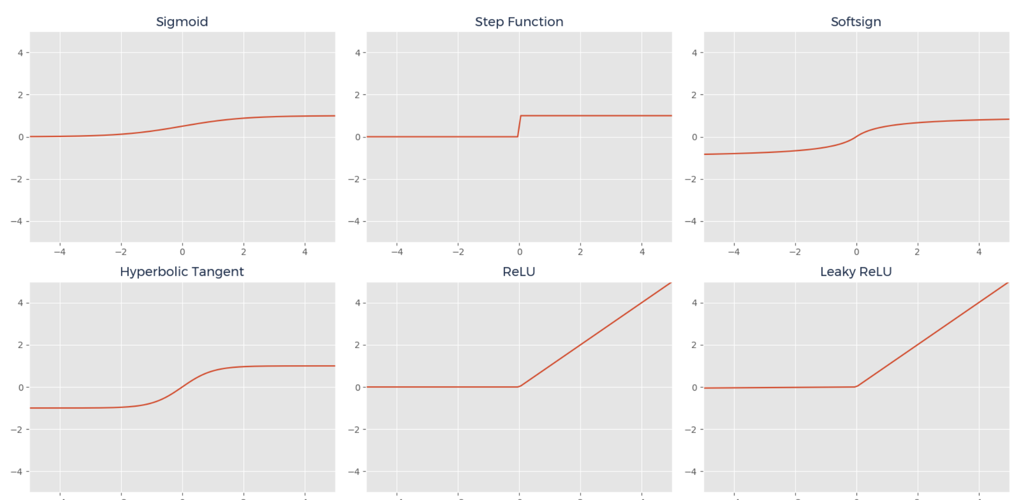

Figure 2: Graphs of activation functions over values from -10 to 10.

There is one special case of activation functions called the softmax function that applies to a layer of neurons instead of a single neuron. Softmax is defined as

$$\sigma(\mathrm{y})_i = \frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}} \tag{8}$$

for a neuron i within a layer with K neurons. Softmax outputs a result that ensures the sum of the output for the neurons in that layer is equal to 1. This is useful to simulate the probabilities of being in one class or another.

Network architecture

In addition to the activation functions used, a key differentiator between neural network models is the chosen architecture of the network. The network architecture is the choice of hidden layer count, number of neurons in each layer, and how they're all connected. The network architecture significantly impacts model performance. The connections specifically not only differentiate between the different instances of a model but also between different types of models.

One important consideration when building a network to perform a specific task is how we construct the output layer. The output layer depends on the task the model is designed to do. The activation functions chosen for the output layer must ensure the output values desired are possible. For supervised learning, the following is generally used,

- **Single-class classification:** 1 neuron output with an output value between 0 and 1 denoting probability.
- **Multi-class classification:** k neurons with Softmax activation where each value is the probability of each of k classes.
- **Regression:** k neurons with an activation that allows the values of the range of values for each of k targets.

Feed Forward Neural Networks

A feed forward neural network (FFNN) is the simplest type of network. FFNN's connections flow in one direction only. This means that neurons in one layer will produce an output that is an input to neurons in the next layer. Data is fed forward and never returns to a previous layer.

A fully connected network is one where the neurons in one layer are each connected to every neuron in the previous and the next layers.

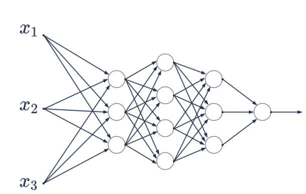

Figure 3: Example of a fully connected feed forward neural network.

Usage & Training

Now that we understand how the neurons are structured, let's look at how they're used and trained. To determine the output of a network, we go through the full flow of the network where each input gets passed to the next neuron until we reach the output neuron producing the desired result we are looking for. Now we look at obtaining the weights of the network so we can use it.

For now, we will go over a simple supervised learning and regression problem. With the initially selected weights, we go through the network and calculate the predicted result, $\hat{y}$. We then determine the error from the correct result using an error function such as the ℓ_2-norm,

$$\epsilon_i = ||\hat{y}_i - y_i||_2. \tag{9}$$

Each neuron is a model that aims to find the weights that minimize its error. It can solve this problem by updating the weights using an iterative approach with a method called stochastic gradient descent,

$$\mathrm{w}_{t+1} = \mathrm{w}_t - \alpha \nabla \epsilon \tag{10}$$

where α is the learning rate per iteration, and $\nabla \epsilon$ is the gradient with respect to that neuron's weights. We can calculate the gradient for neurons that are not directly connected to the output using the chain rule,

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \times \frac{\partial y}{\partial x}. \tag{11}$$

This propagation of the error and gradient from the final output all the way back to the input nodes is called *backpropagation*.

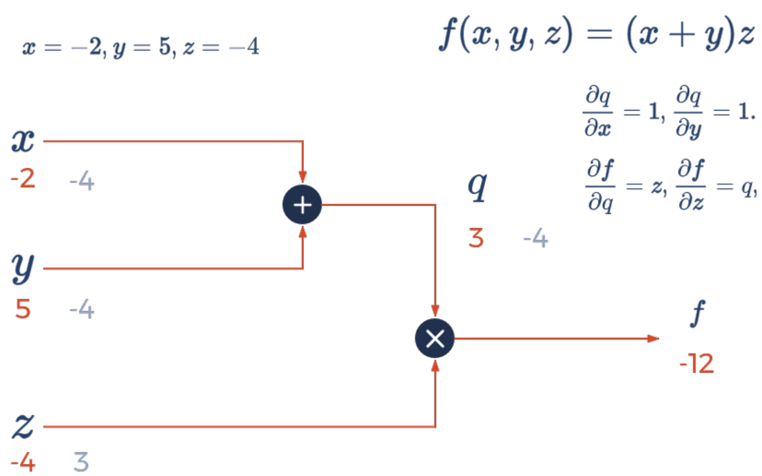

Figure 4: An example of backpropagation gradient calculation.

Overfitting

One issue with a complex model is that it often overfits to the available data. We usually fix this issue by forcing our model to generalize using a regularization method. In linear regression models, we use a regularization term such as an ℓ_p-norm. We can still apply this in neural networks by applying it to the error or cost function of each neuron or layer to force the learned weights to be sparse.

Another regularization method for neural networks is called *dropout*. Dropout is a rate at which we randomly remove connections between some neurons. This forces the model to learn a sufficient understanding of the data without relying on all the neurons in the model, thereby lowering the chance of overfitting.