CME 323: Distributed Algorithms and Optimization, Spring 2015
`http://stanford.edu/~rezab/dao`.
Instructor: Reza Zadeh, Databricks and Stanford.

Lecture 15, May 18. Scribed by Zi Yin, Mingyu Gao and Nicolas Kseib.

15 Neuron Networks

Neural Networks are a class of statistical learning technique which were developed, in parallel, in the fields of computer science, artificial intelligence, and statistics. The semantic of the names derives from the fact that these methods were used to model the activity of the human brain (and other living species) consisting of layers of connected neurons. Practically, neural networks are nothing more than nonlinear statistical techniques whose purpose is to estimate unknown functions that depend on a large number of inputs. Neural networks can be used in both regression and classification problems.

15.1 Single Neuron

As we can see in figure **??** the inputs $x_i \; \forall i = 1, ..., n$ are propagated through the neuron. This neuron takes a linear combination of the inputs $\sum_{i=1}^{n} w_i x_i$ and transforms them through an activation function $f(\sum_{i=1}^{n} w_i x_i)$ which can either be a sigmoid or a tansig function as shown in figure **??**. When the activation function is linear we trivially recover a linear regression problem.

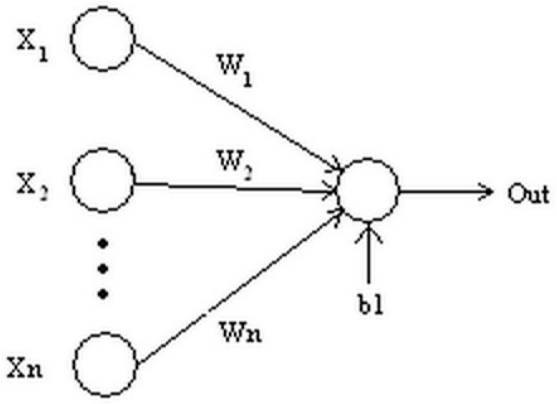

Figure 1: A single neuron example.

Just like in any regression problem the goal is to minimize a cost function. When we would like to minimize for the sum of squared errors we solve the following optimization problem.

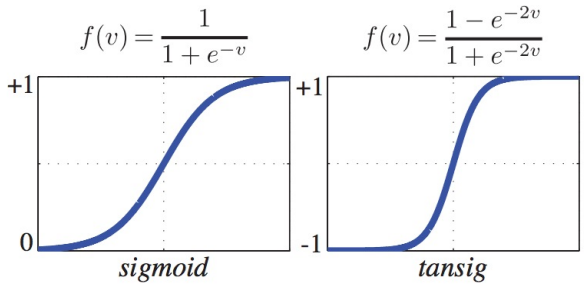

Figure 2: An example of activation functions: sigmoid function (left) and hyperbolic tangent sigmoid (right).

$$\min_{w} \sum_{j=1}^{m} \left(\sum_{i=1}^{n} (w_i x_{ji}) - y_j \right)^2 \tag{1}$$

with y_j the outputs we would like to predict. The dataset is given by $\{y_j, x_j\}_{j=1}^{m}$ in this case.

15.2 Neural Network

Figure **??** shows the usual structure of the single hidden layer neural network, which is a cascade of two regressions. The first is in the hidden layer which gives us $z \in \mathbb{R}^q$ and the second is in the output layer. The hidden layer acts on the input vector $x \in \mathbb{R}^n$, and gives the vector z.

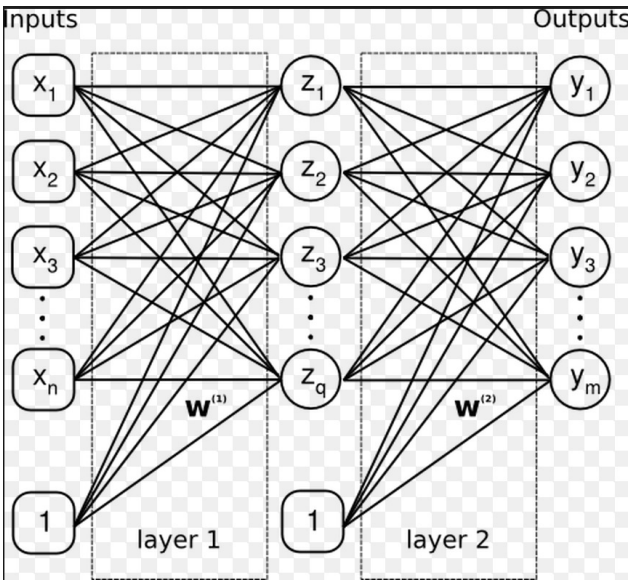

Figure 3: An example of a Neural Network.

The output layer then acts on z to give the outputs $y \in \mathbb{R}^m$. In the hidden layer, each element z_k is a function g of a linear combination of the elements of x:

$$z_k = g\left(W_{0k}^{(1)} + \sum_{i=1}^{n} W_{ik}^{(1)} x_i\right), \forall k = 1, ..., q \tag{2}$$

Furthermore, the vector y is a function f of a linear combination of the elements of the vector z. The bias or intercept terms are given by $W_{0k}^{(1)}$ and $W_{0k}^{(2)}$.

$$y_k = f\left(W_{0k}^{(2)} + \sum_{i=1}^{q} W_{ik}^{(2)} z_i\right), \forall k = 1, ..., m \tag{3}$$

Again if f and g are linear the network becomes a standard linear regression model. For a general neural network we can write the weights in the following form $W_{jk}^{(l)}$ which represents the weight from the k^{th} neuron in the $(l-1)^{th}$ layer to j^{th} neuron in the l^{th} layer.

15.3 Matrix Representation

To better present the back propagation algorithm, we first introduce the matrix representation for the neural network. We denote a^ℓ as the vector of the neurons' output values in the ℓth layer of the network; denote W^ℓ as a matrix of the weights, with the element in row j, column k as w_{jk}^ℓ in layer ℓ; denote b^ℓ as the vector of the biases in layer ℓ. Then we can rewrite the equation as

$$a^\ell = \sigma(W^\ell a^{\ell-1} + b^\ell) \tag{4}$$

It is clear that (**??**) uses the neurons in layer $\ell - 1$ to calculate the values of neurons in layer ℓ. This is called forward pass. As this equation is in linear algebra format, GPU and other hardware accelerators for neural networks can leverage (**??**) and use standard acceleration methods for linear algebra to improve the performance of neural network algorithm.

15.4 Back Propagation

Now we know the algorithm for forward pass. To train the model, we need to get the derivative, which can be obtained using "back propagation" algorithm that does similar computation as forward pass.

First we define another notation for convenience. We let z^ℓ be the vector of the input for the neuron's activation function in layer ℓ, i.e., $z^\ell = W^\ell a^{\ell-1} + b^\ell$, $a^\ell = \sigma(z^\ell)$. Then the back propagation algorithm can be summarized in four equations.
(BP1)

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{5}$$

This equation applies to the last layer L. C is the cost function $C = \sum_j (a_j^L - y_j)^2$; ∇_a is the gradient of C with respect to a^L; σ' is the derivative of the activation function; $\odot$ is element-wise multiplication.

For the layers ℓ other than L, we have

(BP2)

$$\delta^\ell = {W^{\ell+1}}^T \delta^{\ell+1} \odot \sigma'(z^\ell) \tag{6}$$

From all the δ^ℓs, we can calculate the gradient of C with respect to b^ℓ and w^ℓ_{jk}:

(BP3)

$$\frac{\partial C}{\partial b^\ell_j} = \delta^\ell_j \tag{7}$$

(BP4)

$$\frac{\partial C}{\partial w^\ell_{jk}} = a^{\ell-1}_k \delta^\ell_j \tag{8}$$

The back propagation algorithm can also be batched. We just need to turn a^ℓ into a matrix, with the size of a minibatch (say 100).

15.5 Convolutional Neuron Network

Convolutional neuron networks (CNNs) are being widely used in computer vision and natural language processing. The major differences between conventional neuron networks and CNNs are the size and structure. The size of a CNN can be much larger, consisting of tens of layers, each layer has tens on thousands of neurons. And the layers of CNNs are not necessarily fully connected. It is called "convolutional" because each neuron at the input layer can be viewed as a filter working on patches of the image, and as it scans through all patches, it is effectively performing a convolution with the image.

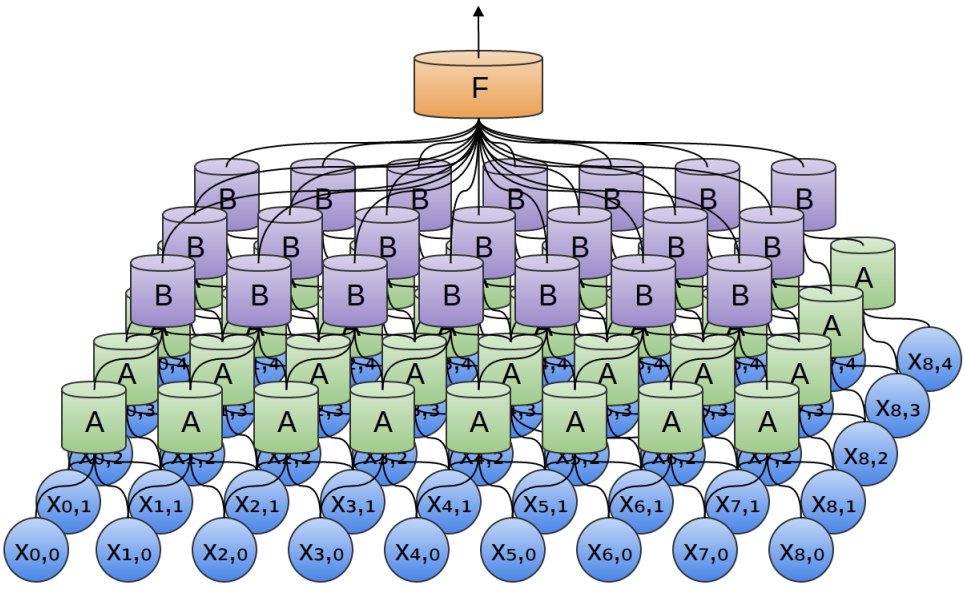

Figure 4: An illustration of a CNN (figure due to Colah)

Since there are too many parameters in a CNN, it is prone to overfitting. There are several heuristics of preventing this from happening. For example, we can use regularization when training the network. Another approach is called dropout, which deactivates some neurons with some probability when training.

15.4 Model and Data Parallelization

Since the CNN models are typically very large, model parallelization is often necessary. In some cases, even a single layer of neurons cannot fit in a single machine's memory.

The first way of parallelizing the model is to store each layer on a different machine. In a forward pass, each machine receive data from the input machine, compute the output and pass it to the machine holding the next layer. In a backward pass, each machine receives gradients from the next machine, compute the gradients of its parameters using chain rule, and feed them back to the machine holding the previous layer.

For a fully connected layer, there can be so many parameters that even splitting them onto several machines is necessary. In this case, communications happen if there is a forward/backward pass across the split boundaries.

To fit large CNNs, it is common to use a parameter server to hold all model parameters. Several machines will then work as a bank, each bank has a replica of model from the parameter server. The bank will ask for the model from the parameter server, compute gradient and send the gradients back to the model server. The server then decide how to use the gradients to update the model.

One thing to notice is that each bank of machines performs computation on a stale model. So it is important for each bank to get an iteration number. When it tries to update the model, the update can be rejected if the stale model is too old. This will prevent the parameters from being messed up by the updates that are too old. Currently Google is implementing a distributed neuron network system of this structure which is called DistBelief.

References

[1] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.