> Describe the basic ideas of multilayer perceptrons – at the level of detail of the hand- out Multilayer Perceptron for lecture 19. in the handout. Why are they better than single layer perceptrons? What is the backpropagation algorithm? What is the main difference between multi-layer perceptrons and deep networks? (At the level of the AlexNet handout in lecture 20).

Multilayer perceptrons:

In order to describe a multilayer perceptron, I need to start with describing a single-layer perceptron. The single-layer perception has two layers: the input layer x and the output layer y. Similar to the concept of a neuron, the output layer unit y receives a set of inputs from x:

$$I = \sum_{j=1}^{d} w_j x_j + w_0$$

To introduce the nonlinearity, we can apply a threshold on top of the sum of the input. We can use the hard threshold as $y = \zeta(I) = 1$ if $y > 0$; $y = \zeta(I) = 0$, otherwise. Or we can use soft threshold

$$y = \sigma(I) = \frac{1}{1 + e^I} = \frac{1}{1 + e^{w^T x}}$$

The multilayer perceptron is a little more complicated than the single-layer one in that it has multiple *hidden* layers. The output of each layer is the input of the next layer. For example, we can have one hidden layer z, which has multiple units z_i. Now, $z_i = \sigma(w_i^T x)$, and $y_i = \sigma(v_i^T z)$. In a similar way, we can add more and more hidden layers as long as we keep the structure as the output of one layer is the input of the next layer with a relationship specified above.

Why the multi-layer perceptrons are better than single-layer perceptrons

The single-layer perceptron cannot represent all decision rules. Famously, the single-layer perceptron cannot perform the XOR operation. The single-layer perceptron can only represent a set of decision rule that separates the space by hyperplanes.

In contrast, the multi-layer perceptrons can represent much more decision rules. By adding the hidden layers, the multi-layer perceptron can almost represent any input-output function. Thus, the multi-perceptron is much more powerful than the single-layer perceptrons in solving complicated problems.

What is the backpropagation algorithm?

The backpropagation algorithm is the process of updating the weighting of connections between successive layers from the output layer back to the early middle layers. Such backpropagation algorithm, in essence, is an application of the chain rule of differentiating a compound function, and it is used to update the weights in the neural network.

Suppose we use only one hidden layer and we define the Error function

$$E[w, v] = \sum_i \left(y_i - \sigma(\sum_h v_{ih} z_h)\right)^2 = \sum_i \left(y_i - \sigma(\sum_h v_{ih} \sigma(\sum_j w_{hj} x_j))\right)^2$$

In order to minimize the error function with respect to w, v, we need to update

$$\Delta w_{hj} = -\frac{\partial E}{\partial w_{hj}}$$

$$\Delta v_{ih} = -\frac{\partial E}{\partial v_{ih}}$$

The second one can be computed directly at the output layer, but the first one needs more computations. if we define $r_k = \sigma(\sum_j w_{kj} xj)$, we can use the chain rule

$$\frac{\partial E}{\partial w_{kj}} = \sum_r \frac{\partial E}{\partial r_k} \frac{\partial r_k}{\partial w_{kj}}$$

In this way, we can update the z layer first by updating $\Delta v_{ih} = -\frac{\partial E}{\partial v_{ih}}$ and $\frac{\partial E}{\partial r_k}$, then we can use the result continue updating the x layer by calculating $\frac{\partial r_k}{\partial w_{kj}}$ and combining the information $\frac{\partial E}{\partial r_k}$ passed from the higher layer. This updating process pass the updating information layer-by-layer backwards, thus termed as backpropagation.

Multi-layer perceptrons and deep network

In my understanding, the deep networks are built based on the ideas of the multi-layer perceptron, and brought in several variations in the original architecture. Let me now use AlexNet as an example of the deep networks.

In terms of architecture, the deep networks have much more layers than multi-layer perceptron, the latter of which usually stopped at 2 layers because of the formidable computations. There are several reasons that the deep network can build so many layers, many of which differentiate the new-era deep networks from the old-era multi-layer perceptrons.

First, deep networks used convolutional layer and max-pooling layer. The convolutional layer, which connects two layers locally, decreased the number of connections between two successive layers, thus decreased the number of parameters. The max-pooling layer also helped to decrease the number of parameters. Although the

AlexNet still used the fully-connected layers, but these are used in quite a late stage and the number of parameters has been decreased.

Second, deep networks, at least AlexNet, used ReLU function for non-linear transformation, instead of the sigmoid function. It turns out that the ReLU function can accelerate training quite a lot and do not suffer from the flat, low gradients that happen a lot when learning with sigmoid functions.

Third, AlexNet used the *Dropout* technique during training. This Dropout technique was used in the fully connected layers, and it randomly and independently set each hidden unit activity to 0 with the probability of 0.5. Such technique also helped to ease the computation burdens in updating the deep network.

Lastly, recent advances of using the GPU and big data set also helped to deep networks, although this feature does not really distinct the multi-layer perceptron from the deep networks.