Section 8

1 Multiple Linear Regression

So far, we have been considering finding functions $f : \mathbb{R}^d \to \mathbb{R}$ that predict a single continuous value for linear regression. Concretely, we find a $w \in \mathbb{R}^d$ predict for a new datapoint $x \in \mathbb{R}^d$ using the following function:

$$f(x) = \begin{bmatrix} -w- \end{bmatrix} \begin{bmatrix} | \\ x \\ | \end{bmatrix} = w^T x$$

Consider if we wanted to find a function $f : \mathbb{R}^d \to \mathbb{R}^k$ to predict k continuous values instead. Then, we can simply learn a w vector for each continuous value we we want to predict. We can express this by making our weights a matrix $W \in \mathbb{R}^{k \times d}$ Concretely,

$$f(x) = Wx = \begin{bmatrix} -w_1- \\ -w_2- \\ \vdots \\ -w_k- \end{bmatrix} \begin{bmatrix} | \\ x \\ | \end{bmatrix} = \begin{bmatrix} w_1^T x \\ w_2^T x \\ \vdots \\ w_k^T x \end{bmatrix}$$

2 Multiple Logistic Regression

In logistic regression, we're interested in classification, and want to find a function $f : \mathbb{R}^d \to [0, 1]$. To achieve this, we simply wrap our linear regression in a sigmoid function $\sigma(x) = \frac{1}{1+exp(-x)}$.

$$f(x) = \sigma(w^T x)$$

The same logic can be applied to multiple linear regression to find a function $f : \mathbb{R}^d \to [0, 1]^k$ to predict k classes.

$$f(x) = \sigma(Wx) = \begin{bmatrix} \sigma(w_1^T x) \\ \sigma(w_2^T x) \\ \vdots \\ \sigma(w_k^T x) \end{bmatrix}$$

3 Learned Feature Maps

We found that feature maps $\phi : \mathbb{R}^d \to \mathbb{R}^k$ allow us to fit more complicated functions. Questions arise: What's a good feature to include? What's not a good feature to include? Wouldn't it be nice if we didn't have to hand engineer features and instead *learn* them? If we made our feature map just *multiple logistic regression* we can do that! Concretely, we can learn some $W_\phi \in \mathbb{R}^{k \times d}$ such that

$$\phi(x) = \sigma(W_\phi x)$$

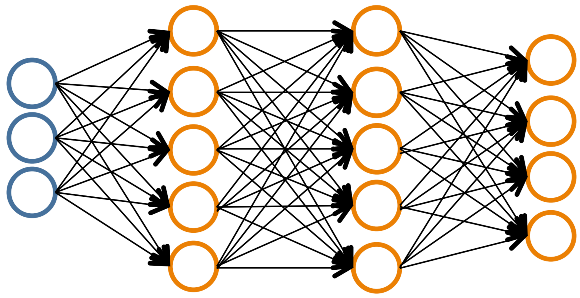
With the above, if we wanted predict r different values using logistic regression as a feature map, our function $f : \mathbb{R}^d \to \mathbb{R}^r$ would look like the following:

$$f(x) = W \cdot \phi(x) = W \cdot \sigma(W_\phi x)$$

We could keep repeating this multiple times (projecting to different feature spaces) and use other elementwise non-linear activation functions $h : \mathbb{R} \to \mathbb{R}$ instead of sigmoid. For instance, if we wanted to learn function $f : \mathbb{R}^3 \to \mathbb{R}^4$, and decided to stack logistic regression three times using $W_3 \in \mathbb{R}^{4\times5}, W_2 \in \mathbb{R}^{5\times5}, W_1 \in \mathbb{R}^{5\times3}$, we get the following

$$f(x) = W_3 \cdot h(W_2 \cdot h(W_1 x))$$

Visually, we can picture each weight as a connection, and input/output as a node.



Congratulations, we have created a basic fully-connected neural network. ("Multi-Layer Perceptron")

The above stems an interesting interpretation of neural networks: one can view all the layers before the final layer as *learning features* the final layer can use to easily classify/predict a given label. Concretely, we can interpret the above feature map as

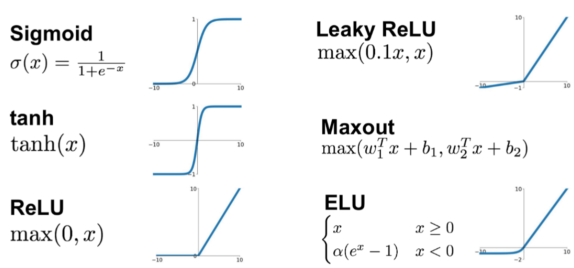
$$\phi(x) = h(W_2 \cdot h(W_1 x))$$

And us doing a simple logistic/linear regression on that feature map

$$f(x) = W_3 \phi(x)$$

The main difference between machine learning and deep learning is that deep learning *learns* the features

4 Activation (Non-Linearity) Importance



Instead of sigmoid, there are many other activations one could use. Let's consider why a non-linearity is necessary. Consider the above example, but instead lets use a linear activation function $h(x) = x$

$$\begin{aligned} f(x) &= W_3 \cdot h(W_2 \cdot h(W_1 x)) \\ &= \underbrace{W_3 W_2 W_1}_{(1)} x \\ &= Ax \end{aligned}$$

Note in (1) that there exists a matrix $A \in \mathbb{R}^{4\times3}$ such that $A = W_3 W_2 W_1$, therefore, a neural network composed of only linear activations can only do as well as simple linear regression!

5 Multi Layer Perceptron Forward Pass

Assume there are T layers of the MLP, then let z_t denote the t^{th} layer, we can then express the t^{th} layer as

$$z_t = h(W_t z_{t-1})$$

where $h(\cdot)$ is the activation function for $2 \leq t \leq T-1$, with the exception that

$$z_1 = x$$
$$z_T = l(y, z_{T-1})$$

where x is the input, l is the loss function and y is the target. Writing out this recursive function we have the following forward pass rule:

$$Z_T = l(y, h(W_{t-1} \cdot h(W_{t-2} \cdot h(\cdots h(W_3 \cdot h(W_2 \cdot x)) \cdots))))$$

6 Backpropagation

An objective of learning the MLP is to

$$\min_{W_{2:T-1}} Z_T = \min_{W_{2:T-1}} l(y, h(W_{t-1} \cdot h(W_{t-2} \cdot h(\cdots h(W_3 \cdot h(W_2 \cdot x)) \cdots))))$$

How do we find the the best $W_2, ..., W_{T-1}$ that minimizes the objective above? Stochastic Gradient Descent!

So our real question now lies in computing each of the gradients $\frac{\partial Z_T}{\partial W_t}$ for $2 \leq t \leq t-1$, so that we can use them to take our gradient steps. More specifically, we can apply chain rule. First we observe that

$$\frac{\partial z_T}{\partial W_t} = \frac{\partial z_T}{\partial z_t} \cdot \frac{\partial z_t}{\partial W_t} = \frac{\partial z_T}{\partial z_t} \cdot \frac{\partial h(W_t z_{t-1})}{\partial W_t} = (\frac{\partial z_T}{\partial z_t} \circ h'(W_t z_{t-1})) \cdot z_{t-1}^T$$

where $A \circ B$ is the element-wise product of A and B.

Clearly, we also need to compute $\frac{\partial z_T}{\partial z_t}$, which by chain rule is:

$$\frac{\partial z_T}{\partial z_t} = \frac{\partial z_T}{\partial z_{t+1}} \cdot \frac{\partial z_{t+1}}{\partial z_t} = \frac{\partial z_T}{\partial z_{t+1}} \cdot \frac{\partial h(W_{t+1} z_t)}{\partial z_t} = (\frac{\partial z_T}{\partial z_{t+1}} \circ h'(W_{t+1} z_t)) \cdot W_{t+1}$$

In forward pass, the sequence of computation is

$$z_1, z_2, ..., z_T$$

In back-propagation, the sequence of computation by the above recursive relationship becomes

$$\frac{\partial z_T}{\partial z_{T-1}}, \frac{\partial z_T}{\partial W_{T-1}}, \frac{\partial z_T}{\partial z_{T-2}}, ..., \frac{\partial z_T}{\partial W_2}$$

Theorem: Assuming each addition or multiplication counts as one operation and h takes roughly the same number operations with h', then Back-propagation takes at most 5 times the operations than the forward pass.

7 Footnote: Bias

Everything in this handout can be extended to have a bias. For multiple linear regression, we'd add a bias term $b \in \mathbb{R}^k$ and define our prediction function f : as the following:

$$f(x) = Wx + b = \begin{bmatrix} -w_1- \\ -w_2- \\ \vdots \\ -w_k- \end{bmatrix} \begin{bmatrix} | \\ x \\ | \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} = \begin{bmatrix} w_1^T x + b_1 \\ w_2^T x + b_2 \\ \vdots \\ w_k^T x + b_k \end{bmatrix}$$

Multiple logistic regression can be written as the following:

$$f(x) = \sigma(Wx + b)$$

The example neural network with biases $b_1 \in \mathbb{R}^3, b_1 \in \mathbb{R}^5, b_3 \in \mathbb{R}^4$ can be written as the following:

$$f(x) = W_3 \cdot h(W_2 \cdot h(W_1 x + b_1) + b_2) + b_3$$