Activation Function

什么是 Activation Function

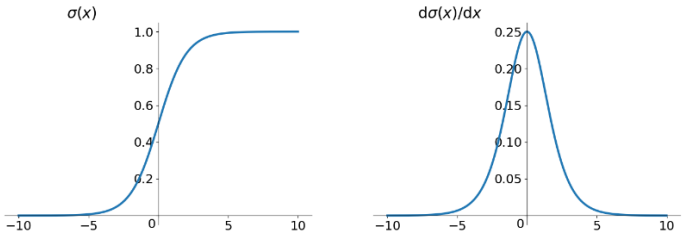
Activation Function is a Function that is added into an Artificial Neural Network in order to **Help the Network Learn Complex Patterns in the Data**. Activation Function takes in the output signal from the Neuron and converts it into some form that can be taken as input to the next Neuron. **Activation Function 有分为 Linear Activation Function 和 Non-Linear Activation Function**

为什么需要 Activation Function

- 在神经网络里，我们是用的 Function 通常是$z = wx + b$，the value if not restricted to a certain limit can go very high in magnitude especially in case of very deep neural networks that have millions of parameters. This will lead to computational issues. 这时候使用 Activation Function 就能限制每个 neuron output 的值，比如说使用 Sigmoid Function，那个值就会被限定在 0 到 1 之间。

- Most important feature in an activation function is its ability to Add Non-Linearity into a neural network. 因为使用的 Function 是$z = wx + b$，而这个 Function 是 Linear Function。只是后如果数据是 Non-Linear 的就不能很好的计算，需要使用 Activation Function 来达到 Non-Linearity 的计算。

Sigmoid Function



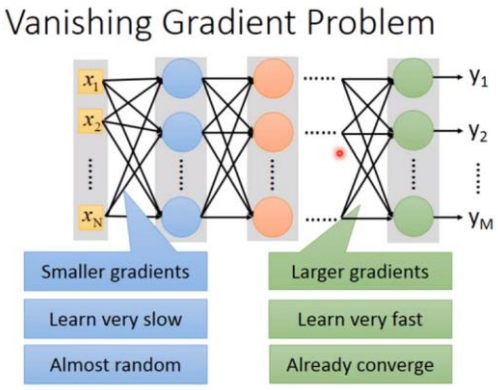
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{d\sigma(z)}{dz} \rightarrow \sigma'(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid Function 是在 Deep Learning 里最常用的 Activation Function. 经过计算后，Sigmoid Function Output 的值会是在 0 到 1 之间。输入越大的值算出来就越接近 1，输入越小的值，算出来就越接近 0。**适合用在需要 Predict Probability as an Output 的网络。**除此之外，由于有着 $(0, 1)$ 的 Range 所以输出值不会爆炸。

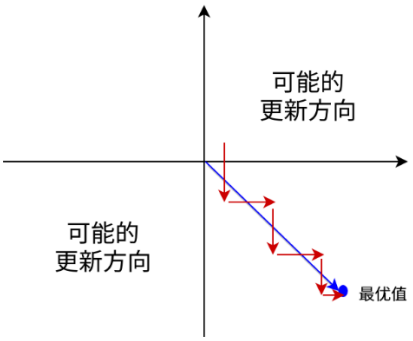
Sigmoid Function 的缺点:

I. 容易出现 Gradient Vanishing



- 在训练深度网络的时候，所有的参数的是 Random Initialize 的。Gradient Vanishing 的发生代表靠近 Output 的 Layers 学习进度比靠近 Input 的 Layers 来的快，因为 Gradient 的值比较大。当靠近 Input 的 Layers 的参数还是 Random 的时候，靠近 Output 的 Layers 的参数已经 Based on 靠近 Input 的 Layers 的值找到了 Local Minimum Point。这时候 Loss 下降的速度是非常慢，而且模型结果是非常差。
- Sigmoid Function 的 Derivative 在 Neural Network 的训练里常常会逐渐变 0，使得 Parameters 没办法 Update，然而导致训练效果不好。这是由 Derivative of Sigmoid Function 的公式导致的。Derivative of Sigmoid Function 最大的值是当输入 $z = 0$ 的时候 $\sigma'(z) = 0.25$。当输入的值越大或者越小，Derivative of Sigmoid Function 给出的值就会很小。随着 Backpropagation 的进行 (当前层的 Derivative 需要和之前各层导数的乘积)，几个小数的相乘，结果会很接近 0，而导致训练卡顿。在 Deep Neural Network 里发生的是接近 Output 的 Layer 训练比较快也比靠近 Input 的 Layer 来的快 Converge。

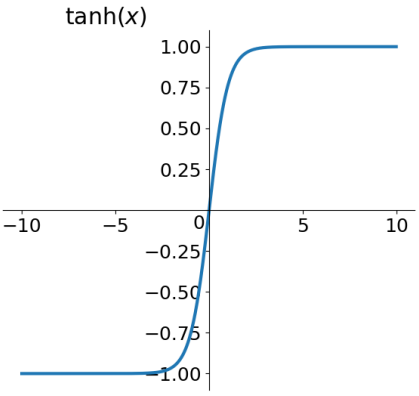
II. Sigmoid Function Output 不是 Zero-Centered

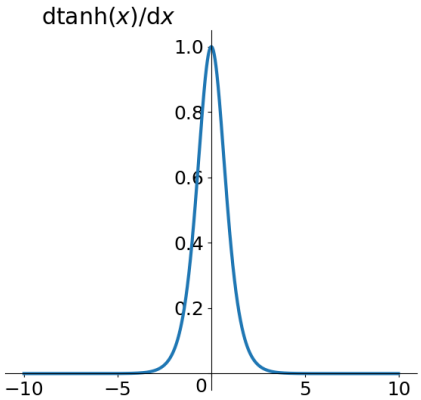


- Sigmoid Function 的 Output 不是 Zero-Centered 的，这回导致 Gradient Updates go too far in different directions。 $0 < Output < 1$，这会导致训练的收敛度变慢。$\sigma(\sum_i w_i x_i + b)$，如果给出的 x_i 全是 Positive 的话算出来的 Derivative 也是 Positive，如果给出的 x_i 全是 Negative 的话算出来的 Derivative 也是 Negative，这会导致没办法找到最好的 Update 路劲，如上图，使用了阶段式更新。

III. 幂运算相对耗时 (Exponential Calculation)

Tanh Function (Hyperbolic Tangent)



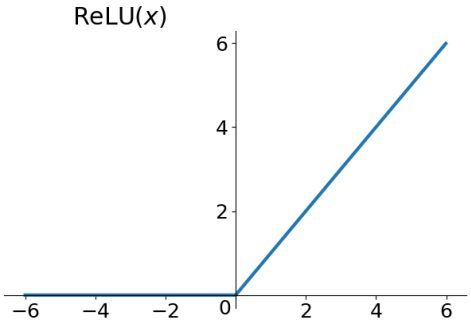


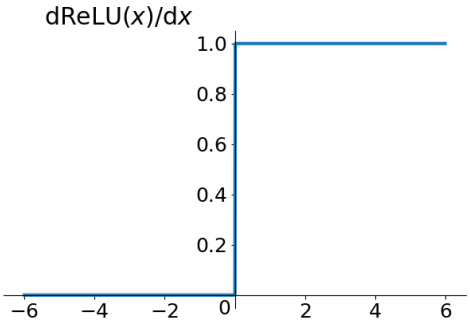
$$\tanh z = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\frac{d(\tanh z)}{dz} = 1 - \tanh^2 z = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2$$

经过计算后，Tanh Function Output 的值会是在 -1 到 1 之间。输入越大的值算出来就越接近 1，输入越小的值，算出来就越接近 -1。**Tanh Function 解决了 Zero-Centered 的输出问题，可是 Gradient Vanishing 的问题和 Exponentiation 的问题依旧存在。**

ReLu Function





$$ReLu = \max(0, z) = \begin{cases} 0 \; if \; z \leq 0 \\ z \; if \; z > 0 \end{cases}$$

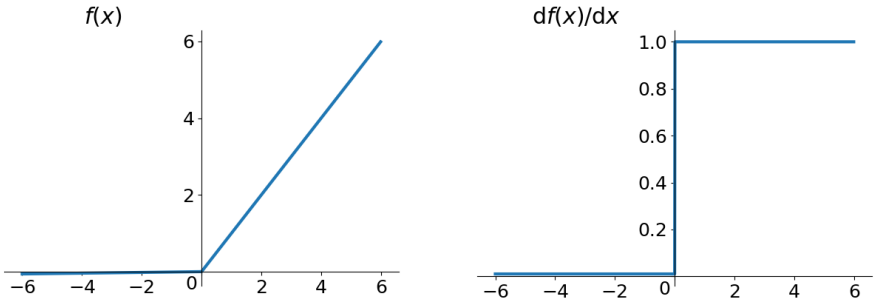
$$\frac{dReLu}{dz} = \begin{cases} 0 \; if \; z < 0 \\ 1 \; if \; z > 0 \end{cases} \; and \; Undefind \; in \; z = 0$$

ReLu Function 的特点就是当 $input\ z$小于 0 的时候 $ReLu\ Output$ 就是 0，当$input\ z$ 大于 0 的时候 $ReLu\ Output$ 就是 z。ReLu Function **成功解决了 Gradient Vanishing 的问题，还加快了计算速度**，只需要计算$input\ z$ 是不是大于 0。再来就是 ReLu 的收敛速度比 Sigmoid Function 和 Tanh Function 来的快。

ReLu 的缺点：

I. **ReLu 的输出不是 Zero-Centered**
II. **ReLu 只适合在 Hidden Layers of Neural Network 里面使用**
III. **Range of ReLu 是 $[0, inf)$ 的，Output 值有可能爆炸大**
IV. **ReLu 存在了 Dead ReLu Problem**
    - Dead ReLu Problem 指的是在 Neural Network 里，有些 Neuron 永远不会被激活，导致相应对的 Neural 里面的参数永远不会被更新。导致 Dead ReLu 的情况可以是非常不幸的参数初始化，但是这个情况比较少见。再来就是 Learning Rate 的值调的太大导致 Update 参数的过程中，参数 Update 到太大，导致 Neural Network 进入这种状态。
      解决 Dead ReLu 的方法可以采用 Xavier 初始化方法，以及避免 Learning Rate 设置太大，或者使用好像 Adagrad 这样的方法。

Leaky ReLu



$$Leaky\ ReLu = \max(\alpha z, z) = \begin{cases} \alpha z\ if\ z \leq 0 \\ z\ if\ z > 0 \end{cases}$$

$$Generally\ \alpha = 0.01$$

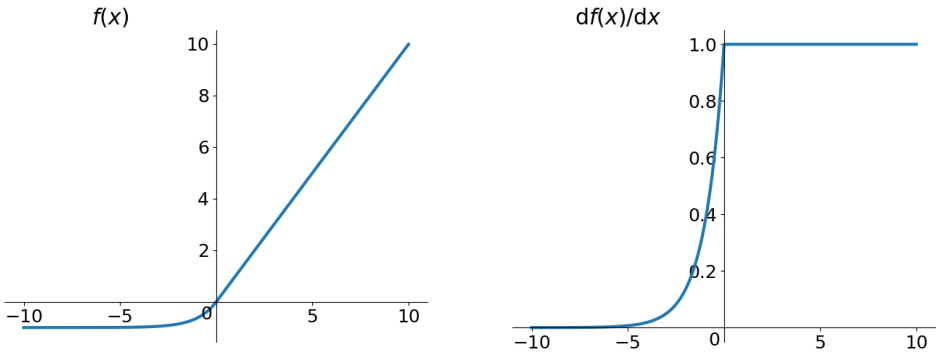
$$\frac{dLeakyReLu}{dz} = \begin{cases} \alpha\ if\ z < 0 \\ 1\ if\ z > 0 \end{cases} and\ Undefind\ in\ z = 0$$

为了解决 Dead ReLu 的问题，在$input\ z \leq 0$ 的时候，让$output = \alpha z$，而非等于 0。理论上来说，Leaky ReLu 有所有 ReLu 该有的特点，外加不会有 Dead ReLu 的问题出现。但实际操作并没有发现 Leaky ReLu 总比 ReLu 好。

Leaky ReLu 的缺点：

I. **由于 Leaky ReLu 是 Linear 的，所以不适合在 Complex Classification 上使用**

ELU (Exponential Linear Units)



$$ELU = \begin{cases} z \; if \; z > 0 \\ \alpha(e^z - 1) \; if \; z \leq 0 \end{cases}$$

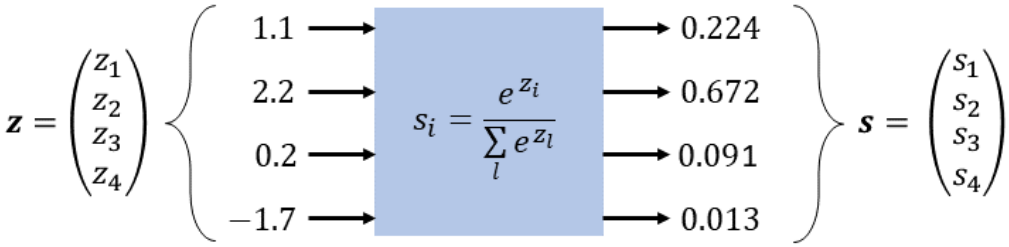
$$\frac{dELU}{dz} = \begin{cases} \alpha e^z \; if \; z < 0 \\ 1 \; if \; z > 0 \end{cases} and \; Undefind \; in \; z = 0$$

ELU 也是为了解决 ReLu 存在的问题，Dead ReLu 和输出不是 Zero-Centered。ELU 的问题在于计算量稍微比 ReLu 大。在理论上，ELU 的表现是比 ReLu 好，但实际操作上，并没有发现 ELU 总是比 ReLu 好。

ELU 的缺点：

I. **For Output of ELU > 0, Output 值有可能爆炸大**

Softmax



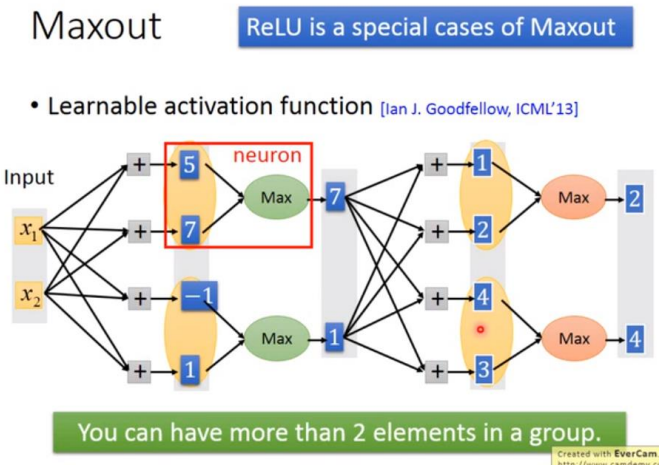
$$Softmax \rightarrow P(y = j | \theta^i) = \frac{e^{\theta^i}}{\sum_{j=0}^{k} e^{\theta_k^i}}$$

Softmax Regression 也叫 Multinomial Logistic Regression 和 Maximum Entropy (MaxEnt) Classifier。Softmax Function 会把 Input Values 转换成几率，Softmax Function 的输出值是 0 到 1。Softmax 通常使用在 Multi-Class Classification 的模型里。

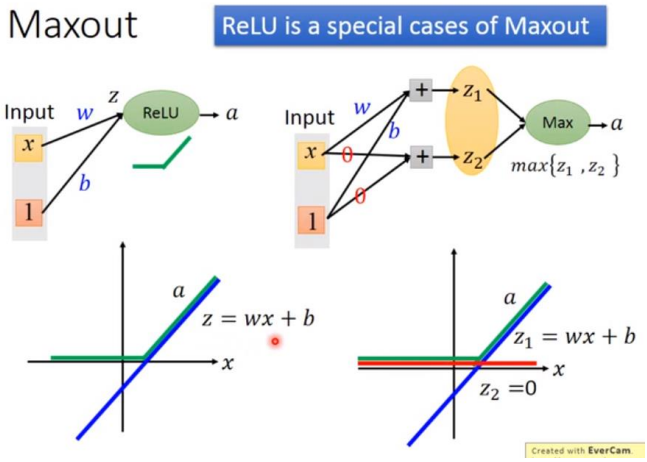
$$-1.7 < 0.2 < 1.1 < 2.2 \rightarrow 0.013 < 0.091 < 0.224 < 0.672$$

Softmax 计算出来的值都是 Positive，而计算出的值也是与 Input 值的 Rank 一样。

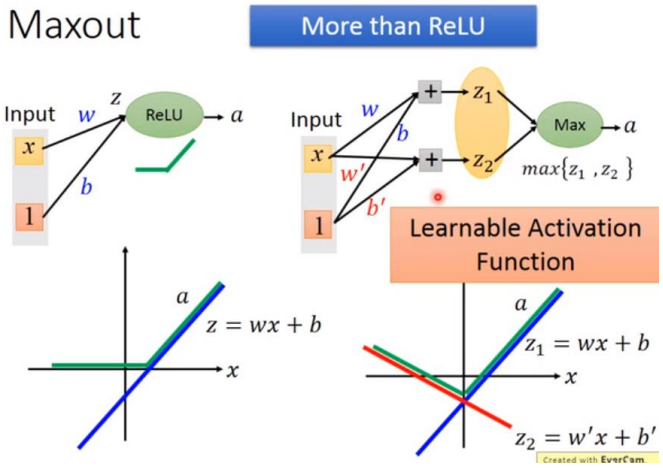
Maxout



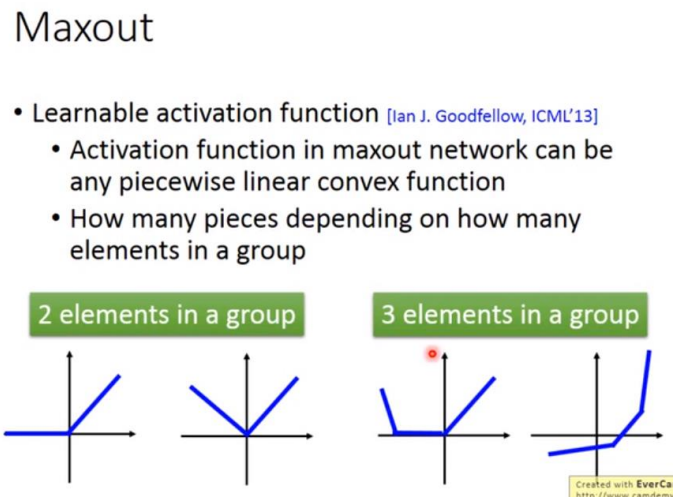
在 Deep Learning 里，有一种训练方法叫 Maxout，就是神经网络里不使用任何的 Activation Function，而是让模型自己学习。Maxout 的方法就是把几个 Neuron 的 Output 事先先 Group 一起。然后在计算的过程中，选出每个 Group 里面最大的值再输入去下一层的 Layer。



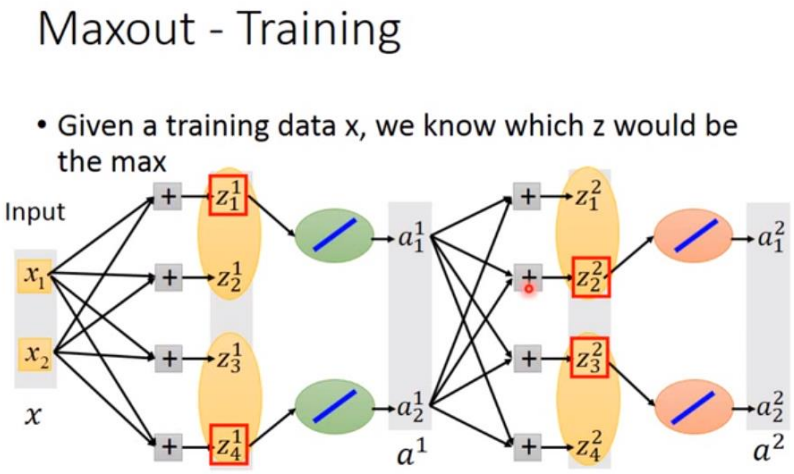
使用 Maxout 的方法在 Neural Network 里是有办法训练出 ReLu 的效果。就是当一个 Group 里如果 z_1 的 $weight\ w$ 和 $bias\ b$ 是正常值，而z_2 的$weight\ w$ 和 $bias\ b$ 是接近 0 的，那么当这个 Group 的 Output 是大于 0 的时候就是使用 z_1 的值，而当 Output 小于 0 的时候就是使用 z_2 的值，而 z_2 的值永远接近 0 因为 z_2 的 $weight\ w$ 和 $bias\ b$ 是接近 0。



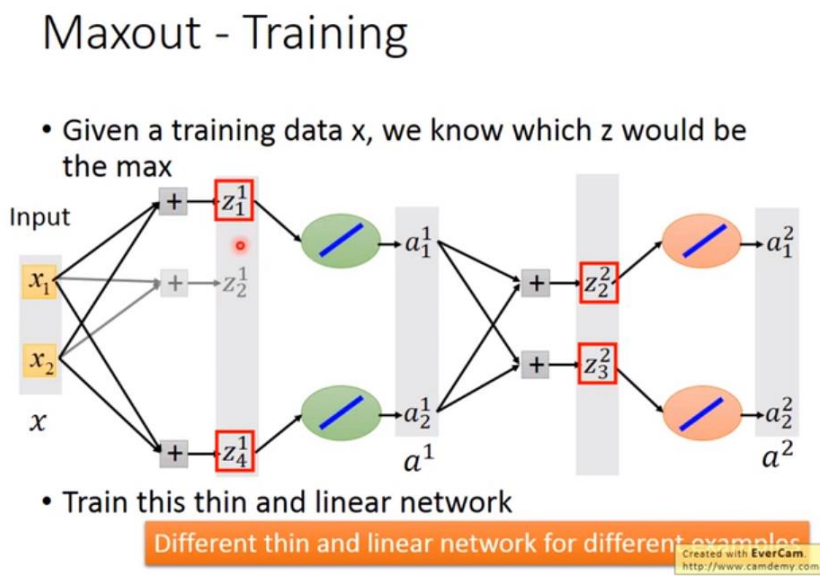
除此之外，不同的数据都能训练出不同的效果，如上图。



更多的 Example，如上图，不同的 Elements 在一个 Group，的出来的效果都不同。



在使用 Maxout 方法来训练模型的时候，其实就更一般的训练一样。



当做完 Forward Pass 之后，那些没被选到的值的参数就不会被 Backpropagation Update 到。但是因为每一笔 Data 都是不一样的，所以不需要担心有些 $weight\ w$ 和 $bias\ b$ 没被训练到。