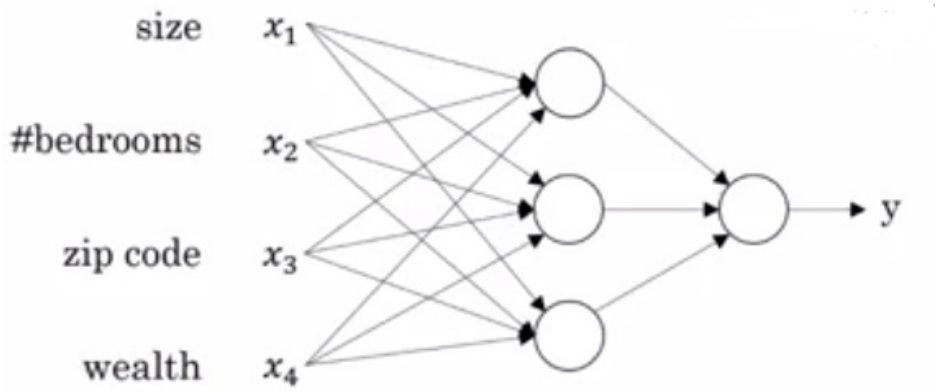
# 1. 神经网络与深度学习

深度学习

## 1. 深度学习介绍

### 1.1 什么是神经网络

神经网络包含输入层、隐藏层和输出层，输入层是原始的特征矩阵、隐藏层的单元数和层数可以自行设置、输出层代表预测的结果。

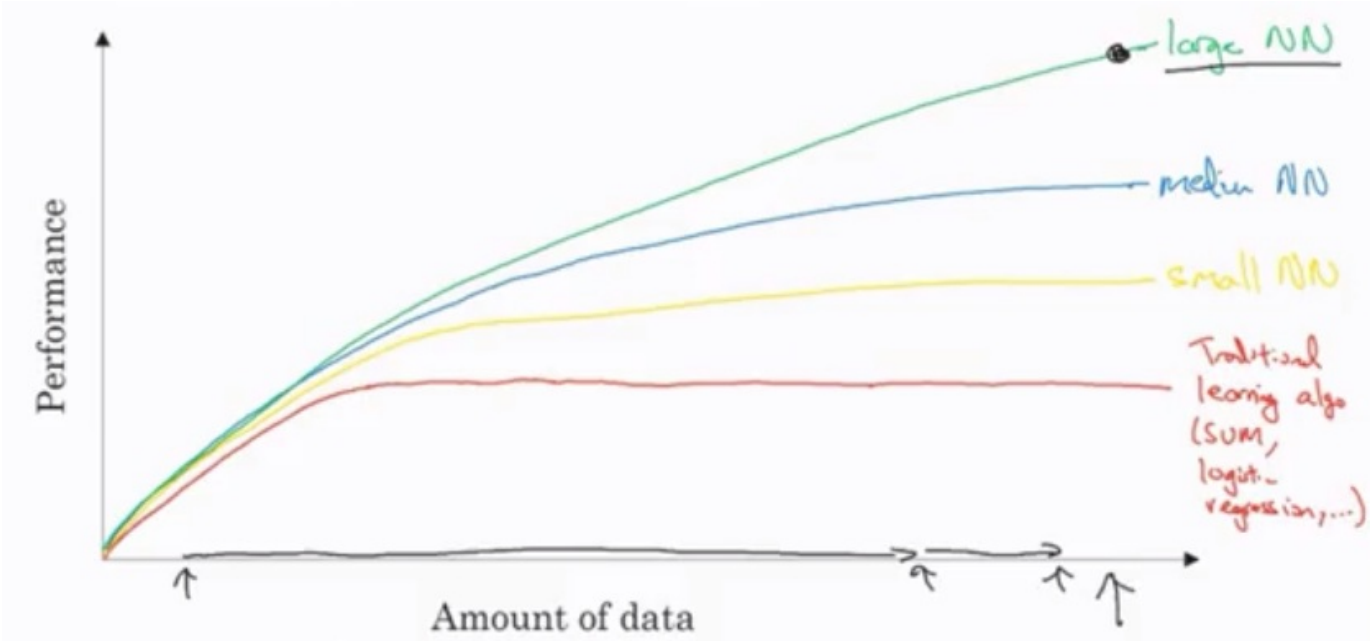



### 1.2 利用神经网络进行监督学习

在监督学习任务中，我们拥有特征X和标签y两类数据。特征也作为模型的输入值，利用特征来预测模型结果；标签也作为输出值，是我们要预测的目标。
监督学习的数据形式分为结构化和非结构化两种：结构化数据拥有较为明确的特征和标签，一般都是数据表形式的，例如房价预测和点击率预测等；非结构化数据则相对难以获取特征，例如图像数据、文本数据、音频数据等。

### 1.3 神经网络的性能优势

对于机器学习项目来说，数据量与模型表现是呈正相关的。随着数据量的增长，模型表现会提升，但模型的表现是有上限的。神经网络优势在于它的上限要高于一般的机器学习算法，随着数据量的增加，神经网络(尤其是深度神经网络)的表现会越来越好。




### 1.4 影响神经网络性能的关键因素

- 数据量
- 计算能力
- 算法性能

## 2. 逻辑回归(无隐藏层神经网络)

### 2.0 符号表示

- m：样本数；n：特征数
- $X_{n*m}$：特征矩阵；$y_{1*m}$：标签
- $x^{(i)}$：第i个样本的特征值；$y^{(i)}$：第i个样本的实际值；w：系数向量；$z^{(i)} = w^T x^{(i)} + b$；
  $\hat{y}^{(i)} = a^{(i)} = \sigma(w^T x^{(i)} + b)$：第i个样本的预测值；$\sigma$：sigmoid激活函数

### 2.1 逻辑回归理解

在逻辑回归中，我们希望通过特征X来预测目标值y。假设预测的目标值是0或1(二分类问题)，输出的预测值是一个概率$\hat{y} = P(y = 1|x)$。对于每一个特征x，我们希望可以找到对应的权重w，那么预测值$\hat{y} = w^T x + b$，其中b是一个偏置项。
但实际上，上述表示很慢保证预测值$\hat{y}$会在0和1之间，因此我们需要为预测值添加一个函数：$\hat{y} = \sigma(w^T x + b)$，这个函数需要保证将y值“压缩”到[0,1]之间。$\sigma(z) = \frac{1}{1+e^{-z}}$就是一个很合适的函数函数，当z接近负无穷时，函数接近0；当z接近正无穷时，函数接近1。为$w^T x + b$添加的转换函数也可以叫做激活函数。

### 2.2 损失函数

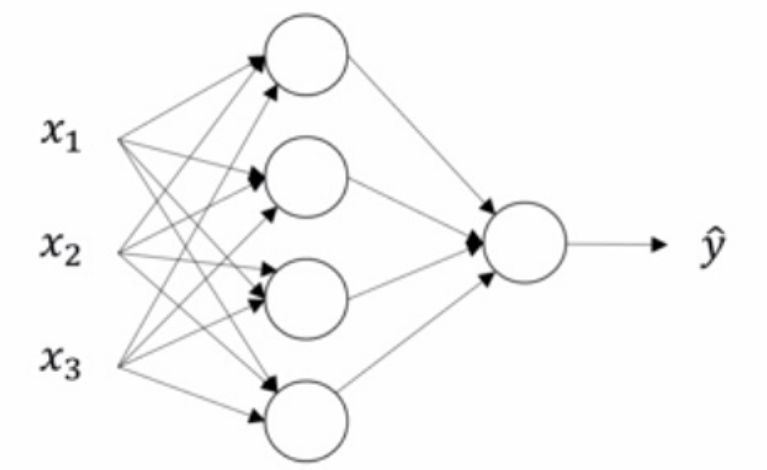
假设我们已经有了权重w，那么对于任意一个样本i，$\hat{y}^i = \sigma(w^T x^i)$，其中$\sigma(z) = \frac{1}{1+e^{-z}}$。
为了使模型更精确，我们希望预测的$\hat{y}^i$能尽可能接近$y^i$。我们使用交叉熵损失函数来定义模型的损失情况：
$L(\hat{y}, y) = -(y * log(\hat{y}) + (1-y)log(1-\hat{y}))$
(1) 当y=1时，后项等于0，前项等于$-log(\hat{y})$，需要使其尽可能小，因此需要使$\hat{y}$尽可能大；
(2) 当y=0时，前项等于0，后项等于$-log(1-\hat{y})$，需要使其尽可能小，因此需要使$\hat{y}$尽可能小。
在实际训练一组样本的过程中，损失函数为：$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m}[y^i * log(\hat{y}^i) + (1-y^i)log(1-\hat{y}^i)]$。最优化的过程实质上是找到一组参数w和b，来使损失函数最小。

## 2.3 梯度下降求解

**梯度下降原理**
为了找到最优解，我们对损失函数在每个参数(w和b)的方向上求导，结果为损失函数在该方向上的梯度。使参数w沿着梯度的负方向变化，能有效地使损失函数下降。我们通过一个权重$\alpha$来确定w在梯度负方向上的变化幅度，$\alpha$也相当于步长。
当特征矩阵拥有d个特征时，每一轮迭代为
$w_d := w_d - \alpha \frac{\partial J(w,b)}{\partial w_d}$
$b := b - \alpha \frac{\partial J(w)}{\partial b}$
**损失函数求导**
逻辑回归模型形式为：$\hat{y} = a = \sigma(z), z = w^T x + b$，损失函数为:
$L(\hat{y}, y) = -(y * log(\hat{y}) + (1-y)log(1-\hat{y}))$。
求解损失函数在各个方向上的偏导数$\frac{dL(\hat{y},y)}{dw_d}$时，可以使用链式法则$\frac{dL(\hat{y},y)}{dw_d} = \frac{dL}{da}\frac{da}{dz}\frac{dz}{dw_d}$。
$\frac{dz}{dw_d} = x_d$
$\frac{da}{dz} = a(1-a)$
$\frac{dL}{da} = -\frac{y}{a} - \frac{1-y}{1-a}$
因此$\frac{dL}{dw_d} = x_d(a-y) = x_d(\hat{y}-y)$
在训练样本的过程中：$\frac{J(w,b)}{w_d} = \frac{1}{m}\sum_{i=1}^{m} x_d(\hat{y}^i - y^i)$

# 3. 浅层神经网络(单隐藏层)

也叫作双层神经网络，注意输入层不算做层数。




## 3.1 符号表示

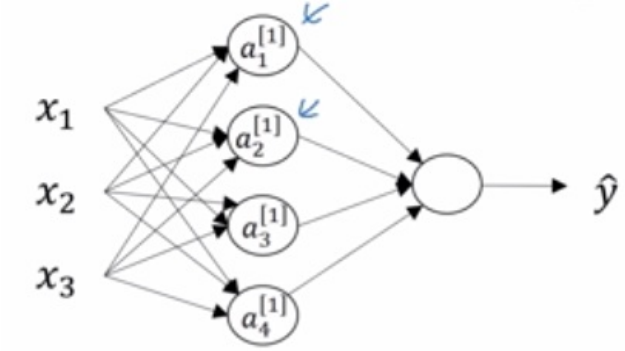
- 输入层：$X = a^{[0]}$，其中每一个特征为$a_i^{[0]}$；隐藏层：$a^{[1]}$，其中每一个特征为$a_i^{[1]}$；输出层：$a^{[2]}$，只有一个单元
- $W^{[l]}$：各层参数矩阵；$Z^{[l]}$：各层参数计算结果；$Z^{[l]}$：各层输出矩阵(激活后)

## 3.2 计算神经网络输出

神经网络中每一个节点都计算都与逻辑回归的方式很类似。假设我们有如下网络结构，输入一个带有三个特征$x_1, x_2, x_3$的样本；网络有四个隐藏层；输出层包含一个单元。


以计算隐藏层的第一个节点为例，我们需要为$x_1, x_2, x_3$找到对应的权重，这三个权重构成的向量可以记为$w_1^{[1]}$，其中上角标l=1代表网络的第一层，下角标i=1代表该层的第一个节点。计算后输出的结果：$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$，再添加一个函数：$a_1^{[1]} = \sigma(z_1^{[1]})$。

依次类推，可以计算出第一层的每一个节点：




$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

该计算过程可以通过逐一计算实现，不过通过向量化的计算方法可以加快速度。第一层中的每一个$(w_i^{[1]})^T$都是一个[1*3]的向量，可以一起构成矩阵$W^{[1]}$，其矩阵结构为[4*3]。因为例子中仅包含一个样本，所以输入的x是[3*1]的向量，那么$W^{[1]}x$会计算出结构为[4*1]的结果$z^{[1]}$，并通过函数$\sigma$计算出相同形式的$a^{[1]}$。
更一般的，假设输入的特征矩阵有m个样本，$n_0$个特征；隐藏层有$n_1$各节点；输出层有$n_2$个节点。那么特征矩阵X的维度为$[n_0, m]$；$W^{[1]}$的维度为$[n_1, n_0]$，$b^{[1]}$为$[n_1, 1]$，$z^{[1]}$和$a^{[1]}$的维度为$[n_1, m]$；$W^{[2]}$的维度为$[n_2, n_1]$，$b^{[1]}$为$[n_2, 1]$，$z^{[2]}$和$a^{[2]}$的维度为$[n_2, m]$。由于输出层只有一个节点，那么$n_2$必然等于1，因此输出结果的形式为[1,m]，恰好也就是每个样本的预测结果。

- $z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$；$a^{[1]} = \sigma^{[1]}(z^{[1]})$
- $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$；$a^{[2]} = \sigma^{[2]}(z^{[2]})$
- 其中输入的特征矩阵X也就是$a^{[0]}$，预测值$\hat{y} = a^{[2]}$

循环计算方法与向量化计算方法的对比

for i = 1 to m

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

**链接**：矩阵计算执行细则

## 3.3 激活函数

### 特点

- 非线性
- 可微分性
- 单调性
- 输出范围有限

### 作用

如果没有激活函数，每一层输出都是上一层的线性函数，因此无论有多少层，最终输出都是线性组合。使用激活函数可以给神经元引入非线性元素，让神经网络可以应用到众多非线性模型中。

**常见激活函数**

**1) sigmoid**

也叫logistics函数，$f(x)=\frac{1}{1+e^{-x}}$

**评价**：计算量大；反向传播时容易出现梯度消失(-4到4以外的区间，基本就梯度消失了)，从而不利于深层训练；中心不为0

**2) tanh**

$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=2sigmoid(2x)-1$

**评价**：解决了sigmoid中均值非0的问题，实际应用中几乎总是好于sigmoid的；但没有解决计算量和梯度消失问题

**3) ReLU**

$f(x)=max(0,x)$

**评价**：计算简便，收敛速度极快；但负数变为0的性质，使神经元很脆弱，极容易永久为0使神经元永久失效。使用ReLU时，需小心设置学习速率

**4) Leaky ReLU**

$f(x)=max(\alpha x,x)$

**评价**：ReLU的改进版本，解决dead神经元过多的问题

**5) softmax**

应用于(多)分类问题，$\sigma(x)_j=\frac{e_j^x}{\sum_{k=1}^{K}e_k^x}$

**激活函数选择**

- 进行分类问题输出时，sigmoid和softmax效果很好
- ReLU只能出现在隐藏层，不能在输出层
- 梯度消失问题严重时，要避免使用sigmoid和tanh
- 死亡神经元数量过多时，考虑将ReLU替换为Leaky ReLU或PReLU
- 通常，隐藏层的选择可以从ReLU开始进行尝试，如果效果不理想再考虑其他激活函数

**激活函数的导数**

- Sigmoid：$g'(z)=g(z)(1-g(z))$
- Tanh：$g'(z)=1-g(z)^2$
- ReLU：$g'(z)=\begin{cases}1, & \text{if z}>0\\0, & \text{if z}<0\end{cases}$
- Leaky Relu：$g'(z)=\begin{cases}1, & \text{if z}>0\\\alpha, & \text{if z}<0\end{cases}$

## 3.4 反向传播算法

构建神经网络模型的过程中，我们希望能够求解最佳参数来使神经网络的损失函数最小，也就是说

$L(\hat{y},y)=\frac{1}{m}\sum_{i=1}^{n}L(\hat{y}_i,y_i)=J(W^{[1]},b^{[1]},W^{[2]},b^{[2]})$

**梯度下降**

对于每个参数，我们沿着其梯度的负方向以一定的步长更新参数，以求解使损失函数最小的参数。

- 计算梯度：$dW^{[1]}=\frac{dJ}{dW^{[1]}},db^{[1]}=\frac{dJ}{db^{[1]}}$；$dW^{[2]}=\frac{dJ}{dW^{[2]}},db^{[2]}=\frac{dJ}{db^{[2]}}$

- 更新参数：$W^{[1]} = W^{[1]} - \alpha dW^{[1]}, b^{[1]} = b^{[1]} - \alpha dWb^{[1]}$；
  $W^{[2]} = W^{[2]} - \alpha dW^{[2]}, b^{[2]} = b^{[2]} - \alpha dWb^{[2]}$

**导数计算**

- $dZ^{[2]} = A^{[2]} - Y$；$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$；$db^{[2]} = \frac{1}{m} sum(dZ^{[2]})$
- $dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(z^{[1]})$；$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[0]T}$；$db^{[1]} = \frac{1}{m} sum(dZ^{[1]})$ (列相加)

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

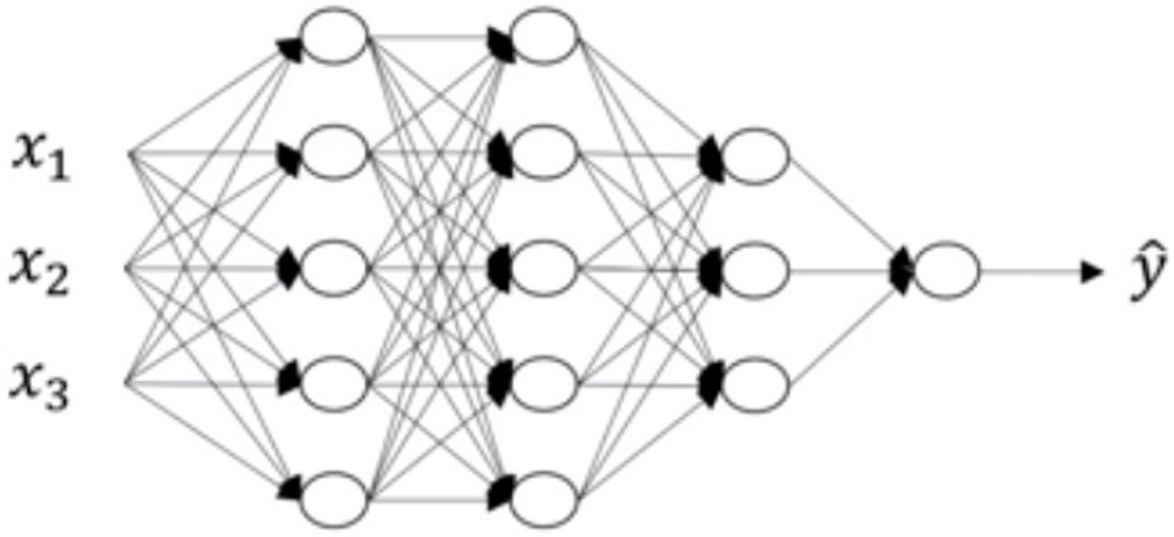
**初始化**

神经网络在进行初始化时，不能将所有参数都设置为0。因为每一个隐藏层中的所有节点都是对称的，参数全部为0会使每一层的所有节点效果都完全一样的。而之后不管进行多少轮迭代，这种情况都不会改变。同理，也不能将所有参数的初始值设置为同一个值。

神经网络的初始化过程一般会选择一个很小的随机数。初始随机数比较小是因为一些激活函数在数值较大或较小时梯度会接近0(例如sigmoid函数)，因此小的随机数可以保证一个足够大的梯度，有较好的训练效果。

# 4. 深层神经网络(多隐藏层)

包括多个隐藏层的神经网络，神经网络的层数等于隐藏层数+1。



深层神经网络包含两个重要要素：模型结构和参数。模型结构一般会在模型训练前就设计好，而参数是需要学习的。

## 4.0 符号表示

- L：神经网络层数；$n^{[l]}$：每层的节点数，其中$n^{[0]}$表示输入层节点数，也就是特征数，$n^{[L]}$表示输出层节点数
- $X = A^{[0]}$：输入层；$A^{[l]}$：隐藏层；$A^{[L]}$：输出层，也就是预测结果；$Z^{[l]}$表示对应的未激活前的计算结果
- $W^{[l]}$，$b^{[l]}$：各层参数

## 4.1 深度神经网络的输出计算

计算深度神经网络时，我们首先使用特征矩阵X和第一层的权重$W^{[1]}$来计算第一个隐藏的结果$Z^{[1]}$，并计算激活后的结果$A^{[1]}$。之后，利用$A^{[1]}$和第二层权重$W^{[2]}$来计算$Z^{[2]}$和$A^{[2]}$。依此类推，不断地使用$A^{[l-1]}$和$W^{[l]}$来计算$Z^{[l]}$和$A^{[l]}$，直至计算完输出层结果。
具体计算过程如下：W代表参数权重、b代表偏置项、Z和A分别代表激活前后的计算结果、g代表每层的激活函数、X是原始特征矩阵，也相当于$A^{[0]}$

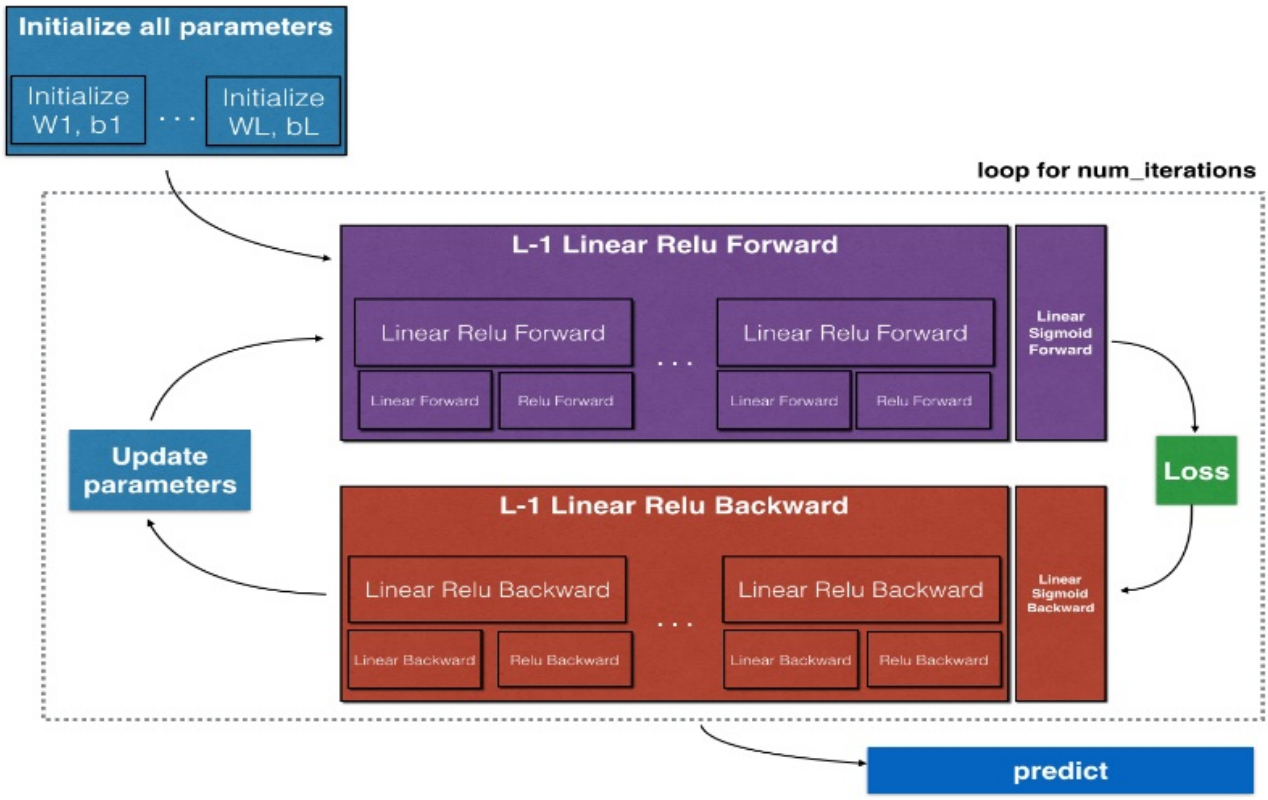
- 计算输入层到第一个隐藏层：$Z^{[1]} = W^{[1]}X + b^{[1]}$；$A^{[1]} = g^{[1]}(Z^{[1]})$
- 计算中间每一个隐藏层直至输出层的结果：$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$；$A^{[l]} = g^{[l]}(Z^{[l]})$
- 输出层计算结果即为预测值：$\hat{y} = A^{[L]}$

假设特征矩阵有m个样本，$n^{[0]}$个特征，隐藏层每一层都有n^{[l]}个节点，则X的形式为$[n^{[0]}, m]$，
$W^{[l]}:[n^{[l]} * n^{[l-1]}]$，$b^{[l]}:[n^{[l]} * 1]$，$Z^{[l-1]}$和$A^{[l-1]}:[n^{[l-1]} * m]$

## 4.2 神经网络参数求解

### 求解过程

获得特征矩阵后，首选通过正向传播逐层计算每一层的$A^{[l]}$，并保存该层参数$W^{[l]}$/$b^{[l]}$和计算值$Z^{[l]}$留作后续使用。计算出$A^{[L]}$后相当于完成了正向传播，计算该层的$dW^{[L]}$和$db^{[L]}$。之后，利用反向传播方法不断向前计算上一层的导数，获得每一层的$dW^{[l]}$和$db^{[l]}$，直至计算到输入层。使用计算出的所有梯度dW和db来更新参数W和b的值。多次重复上述过程进行迭代，直至收敛。




### 反向传播

- 首先利用损失函数L计算输出层L的导数$dA^{[L]}$。例如对于交叉熵损失函数$dA^{[L]}$为$A^{[L]}-y$。
- 求出$dZ^{[L]}=\frac{\partial L}{\partial A}\frac{\partial A}{\partial Z}$，其中$\frac{\partial A}{\partial Z}$也就是激活函数的导数$g'^{[L]}$，例如输出层激活函数为sigmoid时，激活函数g(z)的导数为$g'(z)=z*(1-z)$
- 利用$dZ^{[L]}$计算出：
  - $dW^{[L]}=\frac{\partial L}{\partial ZL}\frac{\partial ZL}{\partial WL}=dZL*A^{[L]T}$
  - $db^{[L]}=\frac{\partial L}{\partial ZL}\frac{\partial ZL}{\partial bL}=dZL*\vec{1}$，其中$\vec{1}$形式为($n_L$,1)，也相当于对dZL沿列相加
- 计算前一层激活后结果的导数 $dA^{[L-1]}=\frac{\partial L}{\partial ZL}\frac{\partial ZL}{\partial AL-1}=W^{[L-1]T}dZL$
- 重复以上过程，按顺序计算之前层的$dZ^{[l]}\rightarrow dW^{[l]}\&db^{[l]}\rightarrow dA^{[l-1]}$
  - $dZ^{[l]}=dA^{[l]}*g'^{[l]}(Z^{[l]})$
  - $dW^{[l]}=\frac{1}{m}dZ^{[l]}A^{[l-1]}$
  - $db^{[l]}=\frac{1}{m}sum(dZ^{[l]})$ (列相加)
  - $dA^{[l-1]}=W^{[l]T}dZ^{[l]}$
- 直至计算出输出层参数$dW^{[1]}$和$db^{[l]}$，结束所有梯度的计算

### 超参数

超参数需要在训练模型前预先设定，它们无法通过训练模型获得，但它们的设定会影响到参数的结果。
神经网络中典型的超参数包括：

- 学习速率
- 迭代轮次
- 隐藏层层数以及每层节点数
- 每层的激活函数选择