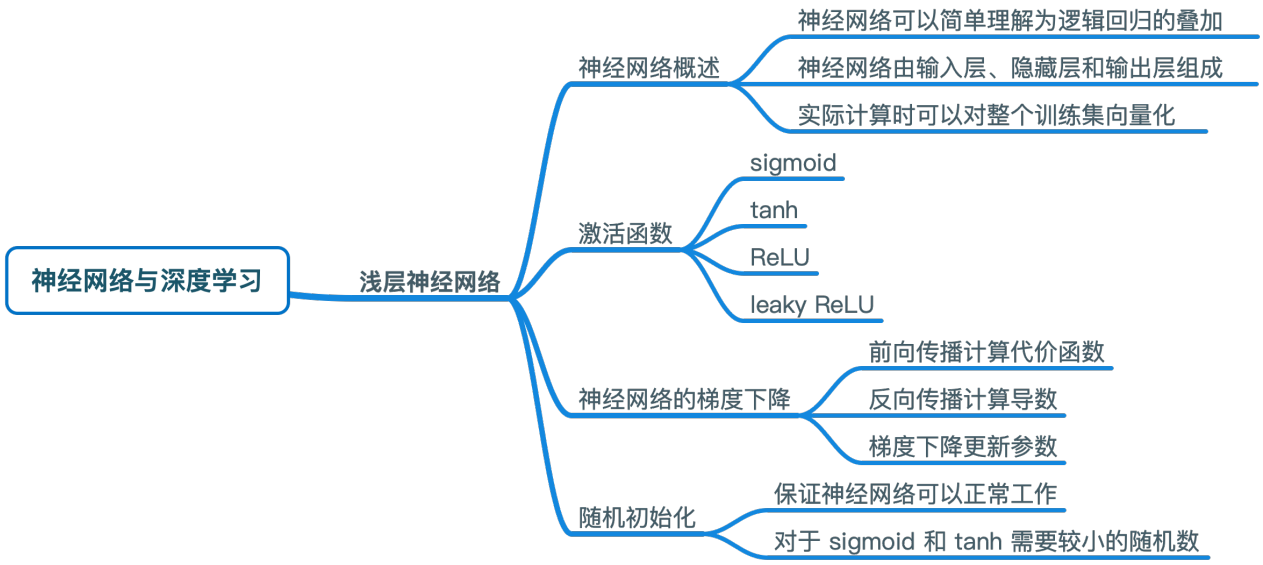
浅层神经网络



神经网络概述

与逻辑回归的对比

- 逻辑回归的结构如下：

```
X1  \
X2   ==>  z = XW + B ==> a = Sigmoid(z) ==> l(a,Y)
X3  /
```

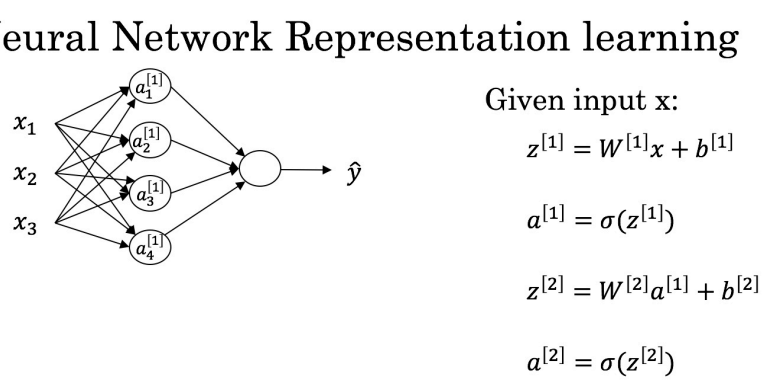
- 而一个单层神经网络的结构如下：

```
X1  \
X2   =>  z1 = XW1 + B1 => a1 = Sig(z1) => z2 = a1W2 + B2 => a2 = Sig(z2)
=> l(a2,Y)
X3  /
```

- 可以将神经网络简单理解为逻辑回归的叠加

表示与计算

- 本节将定义含有一层隐藏层的神经网络
  - `a0 = x` 表示输入层
  - `a1` 表示隐藏层的激活值
  - `a2` 表示输出层的激活值
- 计算神经网络的层数时，我们一般不考虑输入层（即本节讨论的是两层神经网络）
- 下图给出了一个神经网络的前向传播计算公式：



- 在该网络中，隐藏层的神经元数量（`noOfHiddenNeurons`）为 4，输入的维数（`nx`）为 3
- 计算中涉及到的各个变量及其大小如下：
  - `W1` 是隐藏层的参数矩阵, 其形状为 `(noOfHiddenNeurons, nx)`
  - `b1` 是隐藏层的参数矩阵, 其形状为 `(noOfHiddenNeurons, 1)`
  - `z1` 是 `z1 = W1*X + b` 的计算结果，其形状为 `(noOfHiddenNeurons, 1)`
  - `a1` 是 `a1 = sigmoid(z1)` 的计算结果，其形状为 `(noOfHiddenNeurons, 1)`
  - `W2` 是输出层的参数矩阵，其形状为 `(1, noOfHiddenNeurons)`
  - `b2` 是输出层的参数矩阵，其形状为 `(1, 1)`
  - `z2` 是 `z2 = W2*a1 + b` 的计算结果，其形状为 `(1, 1)`
  - `a2` 是 `a2 = sigmoid(z2)` 的计算结果，其形状为 `(1, 1)`

代码实现

- 两层神经网络前向传播的伪代码如下：

```
for i = 1 to m
z[1, i] = W1*x[i] + b1      # shape of z[1, i] is (noOfHiddenNeurons,1)
a[1, i] = sigmoid(z[1, i])  # shape of a[1, i] is (noOfHiddenNeurons,1)
z[2, i] = W2*a[1, i] + b2   # shape of z[2, i] is (1,1)
a[2, i] = sigmoid(z[2, i])  # shape of a[2, i] is (1,1)
```

- 如果对整个训练集进行向量化，得到新的 `X` 形状为 `(Nx, m)`，则新的伪代码如下：

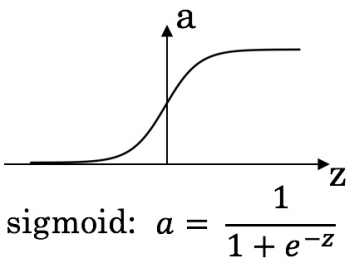
```
Z1 = W1X + b1     # shape of Z1 (noOfHiddenNeurons,m)
A1 = sigmoid(Z1)  # shape of A1 (noOfHiddenNeurons,m)
Z2 = W2A1 + b2    # shape of Z2 is (1,m)
A2 = sigmoid(Z2)  # shape of A2 is (1,m)
```

  - 样本数量 m 始终表示列的维数
  - `X` 可以写为 `A0`

激活函数

常见激活函数

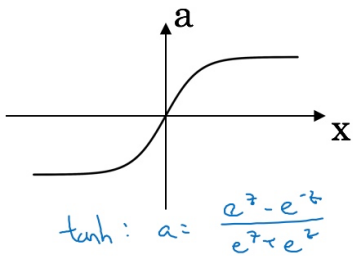
sigmoid



- sigmoid 激活函数的取值范围是 [0,1]
- sigmoid 可能会导致梯度下降时更新速度较慢
- 代码实现：

```
sigmoid = 1 / (1 + np.exp(-z)) # Where z is the input matrix
```

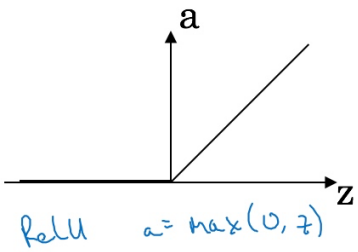
tanh



- tanh 激活函数的取值范围是 [-1,1]（sigmoid 函数的偏移版本）
- 对隐藏层来说，tanh 比 sigmoid 的效果更好，因为其输出的平均值更接近0，这使得下一层数据更加靠近中心（便于梯度下降）
  - tanh 与 sigmoid 存在同样的缺点，即如果输入过大或过小，则斜率会趋近于0，导致梯度下降出现问题
- 代码实现：

```
tanh = (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z)) # Where z is the input matrix
tanh = np.tanh(z)   # Where z is the input matrix
```

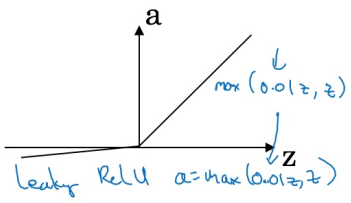
ReLU



- ReLU 函数可以解决梯度下降慢的问题（针对正数）
- 如果你的问题是二元分类（0或1），那么输出层使用 sigmoid，隐藏层使用 ReLU
- 代码实现：

```
ReLU = np.maximum(0,z) # so if z is negative the slope is 0 and if z is positive the slope remains linear.
```

leaky ReLU



- leaky RELU 与 ReLU 的区别在于当输入为负值时，斜率会较小（不为0）
- 它和 ReLU 同样有效，但大部分人使用 ReLU
- 代码实现：

```
leaky_ReLU = np.maximum(0.01*z,z)  #the 0.01 can be a parameter for your algorithm.
```

- 目前激活函数的选择并没有普适性的准则，需要尝试各种激活函数（也可以参考前人的经验）

激活函数的非线性

- 线性激活函数会输出线性的激活值
  - 无论你有多少层隐藏层，激活都将是线性的（类似逻辑回归）
  - 隐藏层会失去意义，无法处理复杂的问题
- 当输出是实数时，可能需要使用线性激活函数，但即便如此如果输出非负，那么使用 ReLU 函数更加合理

激活函数的导数

- sigmoid 函数：

```
A = 1 / (1 + np.exp(-z))
dA = (1 / (1 + np.exp(-z))) * (1 - (1 / (1 + np.exp(-z))))
dA = A * (1 - A)
```

- tanh 函数：

```
A = (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))
dA = 1 - np.tanh(z)^2 = 1 - A^2
```

- ReLU 函数：

```
A = np.maximum(0,z)
dA = { 0   if z < 0
       1   if z >= 0   }
```

- leaky ReLU 函数：

```
A = np.maximum(0.01*z,z)
dA = { 0   if z < 0
       1   if z >= 0   }
```

神经网络的梯度下降

- 反向传播的公式与伪代码如下：

Summary of gradient descent

$dz^{[2]} = a^{[2]} - y$	$dZ^{[2]} = A^{[2]} - Y$
$dW^{[2]} = dz^{[2]}a^{[1]^T}$	$dW^{[2]} = \frac{1}{m} dZ^{[2]}A^{[1]^T}$
$db^{[2]} = dz^{[2]}$	$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(z^{[1]})$	$dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$
$dW^{[1]} = dz^{[1]}x^T$	$dW^{[1]} = \frac{1}{m} dZ^{[1]}X^T$
$db^{[1]} = dz^{[1]}$	$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

Andrew Ng

随机初始化

- 在逻辑回归中随机初始化权重并不重要，而在神经网络中我们需要进行随机初始化
- 如果在神经网络中将所有权重初始化为0，那么神经网络将不能正常工作 (bias初始化为0是可以的):
  - 所有隐藏层会完全同步变化（计算同一个函数）
  - 每次梯度下降迭代所有隐藏层会进行相同的更新
- 为了解决这个问题我们将 W 初始化为一个小的随机数：

```
W1 = np.random.randn((2,2)) * 0.01    # 0.01 to make it small enough
b1 = np.zeros((2,1))                  # its ok to have b as zero
```

  - 对于 sigmoid 或 tanh 来说，我们需要随机数较小
    - 因为较大的值会导致在训练初期线性激活输出过大，从而使激活函数趋向饱和，导致学习速度下降
  - 如果没有使用 sigmoid 或 tanh 作为激活函数，就不会有很大影响
- 常数 0.01 对单层隐藏层来说是合适的，但对于更深的神经网络来说，这个参数会发生改变来保证线性计算得出的值不会过大