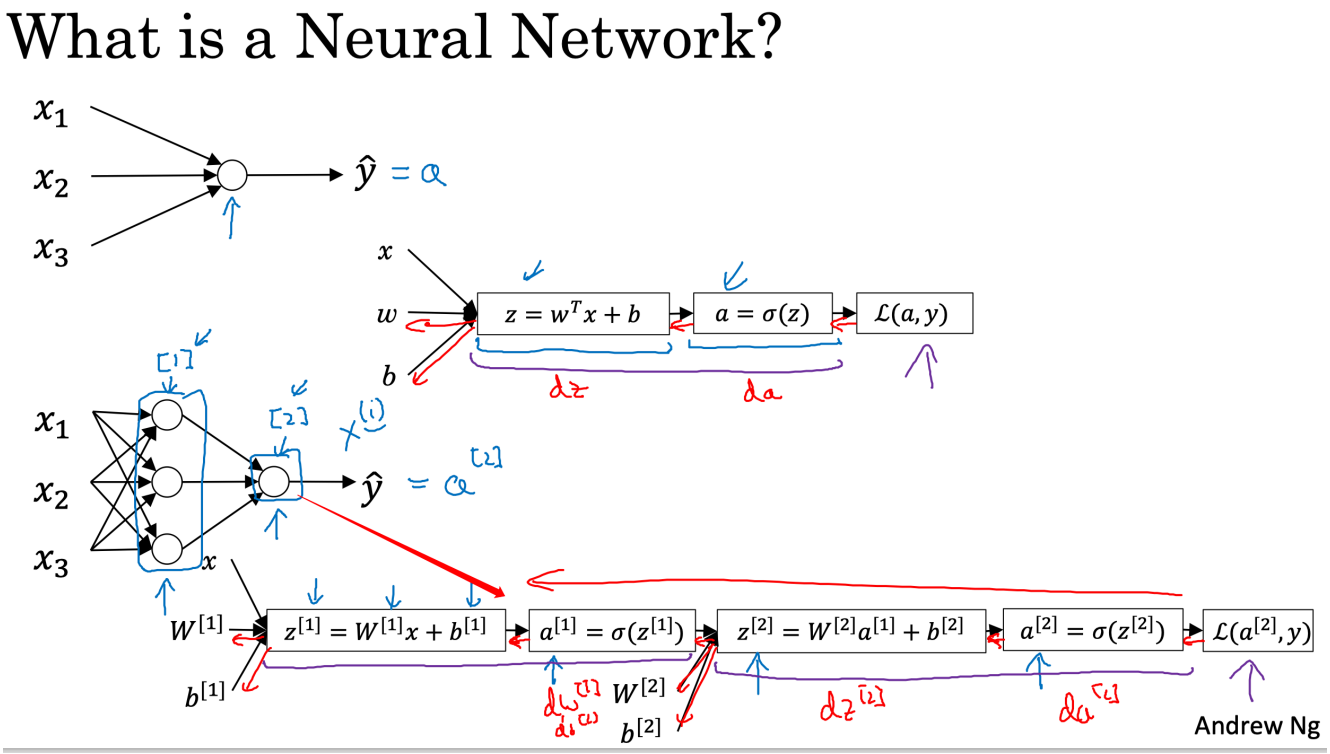
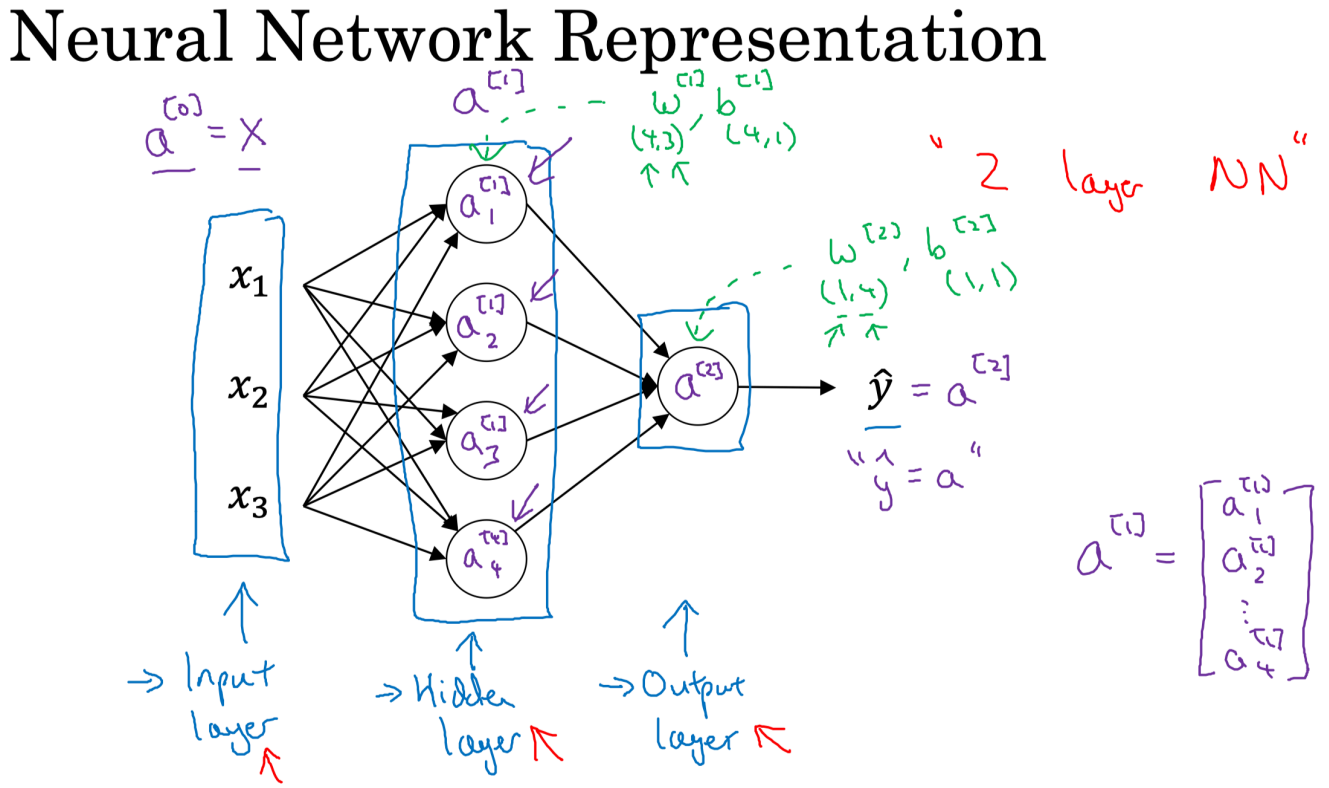
Shallow Neural Network

1. what is a neural network:



每个神经元就是一对z和a

2. Neural Network Representation:



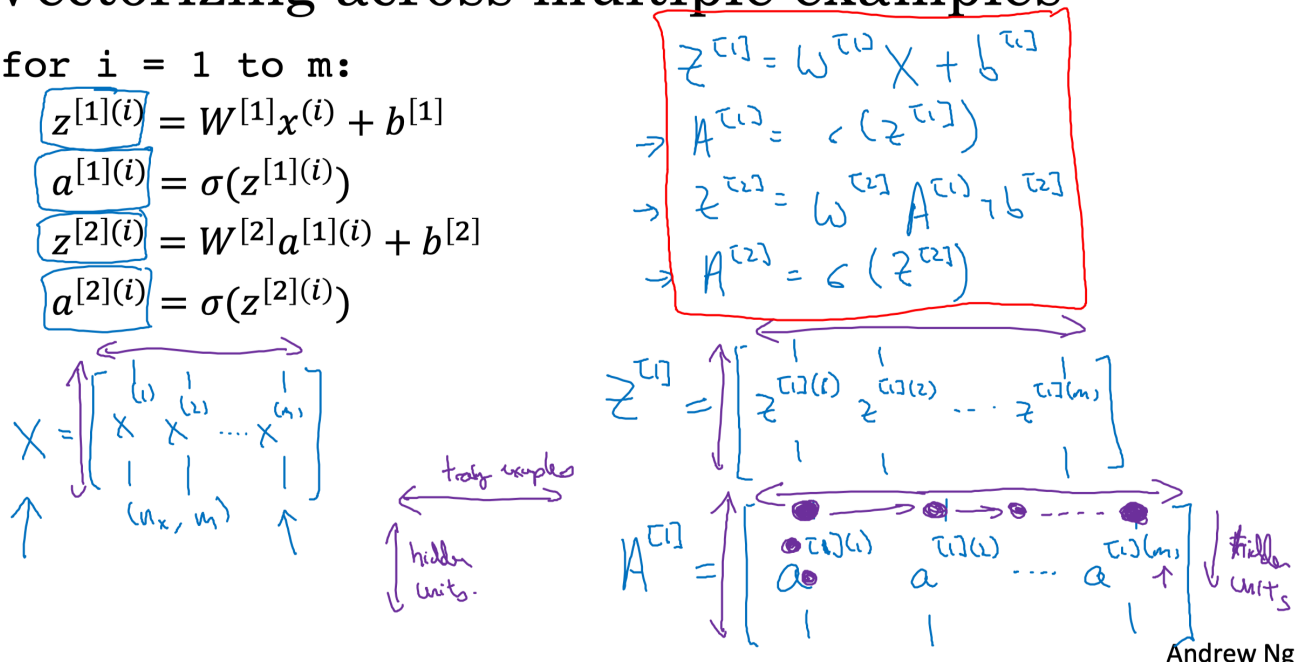
input layer不算一层，Hidden layer才是第一层所以这个是2 layer NN

3. Vectorizing across multiple examples:横向是total example, 纵向是hidden unit

Vectorizing across multiple examples

```
for i = 1 to m:
```

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



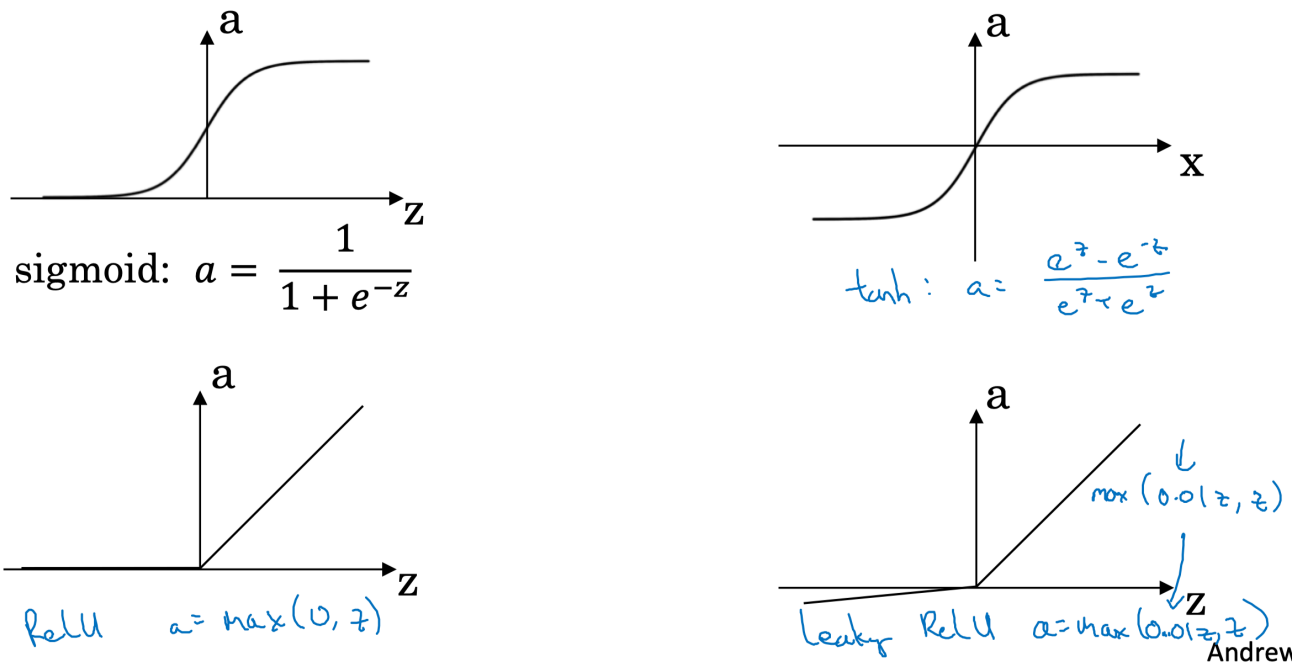
4. Pros and cons of activation functions:

   Choosing the right activation function for a neural network depends on several factors, including the type of problem you're solving, the characteristics of the data, and the specific requirements of the model. Here's a guide on when to use each of the commonly used activation functions:

   1. **Sigmoid (Logistic) Function**
      - Use for:
        - Problems where you need probabilities as output, since the output is in the range (0,1).
        - Output layers of binary classification problems.
      - Avoid for:
        - Hidden layers in deep networks due to the vanishing gradient problem.
   2. **Hyperbolic Tangent (tanh) Function**
      - Use for:
        - Hidden layers in neural networks, especially if the data is centered around zero.
        - Problems where you need outputs in the range (-1, 1).
      - Avoid for:
        - Very deep networks, as it also suffers from the vanishing gradient problem.
   3. **Rectified Linear Unit (ReLU) Function**
      - Use for:
        - Most deep learning networks, especially in hidden layers, due to its computational efficiency and effectiveness in mitigating the vanishing gradient problem.
        - Convolutional Neural Networks (CNNs) and Deep Feedforward Neural Networks.
      - Avoid for:

        - Networks where dead neurons become an issue, unless you have a mechanism to deal with this.
4. **Leaky ReLU**
    - Use for:
        - Situations where you suspect that dying ReLU could be a problem.
        - Deep networks, as it allows for a small gradient when the unit is inactive.
5. **Parametric ReLU (PReLU)**
    - Use for:
        - Networks where you want to give the model the flexibility to learn the appropriate slope of the negative part.
    - Avoid for:
        - Smaller datasets or simpler networks, where the added complexity might lead to overfitting.
6. **Exponential Linear Unit (ELU)**
    - Use for:
        - Networks where you want improved learning characteristics compared to ReLU or Leaky ReLU.
        - Problems where reducing the vanishing gradient effect is important.
    - Avoid for:
        - Situations where computational efficiency is a higher priority, as ELUs are more computationally intensive.
7. **Softmax Function**
    - Use for:
        - The output layer of multi-class classification problems, where you need probabilities for multiple classes.
    - Avoid for:
        - Hidden layers, as its primary purpose is for classification output.

In practice, the choice of an activation function can also be influenced by empirical results; often, the best way to choose is to experiment with a few different functions and see which one performs best for your specific dataset and problem. Additionally, advancements in deep learning might lead to the development of new activation functions that could be more suited to certain types of problems.



如果要求输出是01binary classifcation那么output layer用sigmoid, Relu是最常用的

5. Why do you need Non-Linear Activation Functions for neural network?

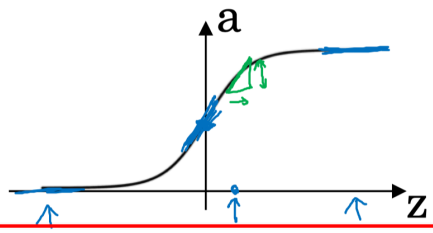
   Non-linear activation functions are crucial in neural networks for several reasons:

   1. **Introducing Non-linearity**: The primary reason is to introduce non-linearity into the network. Without non-linearity, a neural network, regardless of how many layers it has, would behave like a linear model. This limits its ability to model complex data relationships. Non-linear functions allow the network to capture and learn more complex patterns in the data.
   2. **Deep Learning Capabilities**: The introduction of non-linearity allows for deep learning. With non-linear activation functions, deep neural networks can learn features and patterns at various levels of abstraction. As we move deeper into the network, the complexity and abstraction of what the network can learn and represent increases significantly.
   3. **Backpropagation and Gradient Descent**: Activation functions that are differentiable play a key role in enabling backpropagation. In backpropagation, gradients are computed to update the weights of the network, and for this calculation, the derivative of the activation function is necessary. Without a non-linear, differentiable function, this wouldn't be possible.
   4. **Control of Output Range**: Some activation functions like the sigmoid or hyperbolic tangent function squash the output into a bounded range. This is particularly useful in cases like binary classification where we want the output to represent a probability.
   5. **Biological Plausibility**: While not a primary reason for their use in most applications, non-linear activation functions are often seen as a way to mimic the firing patterns of biological neurons, which are not linear in nature.

   Popular non-linear activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh (Hyperbolic Tangent). Each has its own characteristics and is suitable for different types of neural network layers and architectures.

6. Derivatives of activation functions:

Sigmoid activation function



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

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

$z = 10. \quad g(z) \approx 1$

$\frac{d}{dz} g(z) \approx 1\,(1-1) \approx 0$

$z = -10 \quad g(z) \approx 0$

$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) \approx 0$

$z = 0 \quad g(z) = \frac{1}{2}$

$\frac{d}{dz} g(z) = \frac{1}{2}\left(1-\frac{1}{2}\right) = \frac{1}{4}$

$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$

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

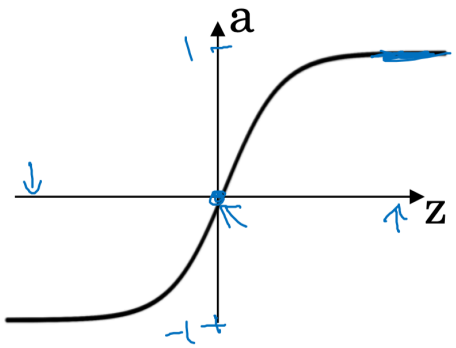
$= g(z)\,(1-g(z)) \leftarrow$

$= a\,(1-a)$

$g'(z) = a(1-a)$

Andre

Tanh activation function



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

$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$

$= 1 - (\tanh(z))^2 \leftarrow$

$a = g(z), \quad g'(z) = 1 - a^2$

$z = 10 \quad \tanh(z) \approx 1$

$g'(z) \approx 0$

$z = -10 \quad \tanh(z) \approx -1$

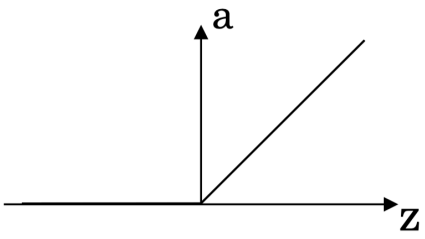
$g'(z) \approx 0$

$z = 0 \quad \tanh(z) = 0$

$g'(z) = 1$

Andr

ReLU and Leaky ReLU

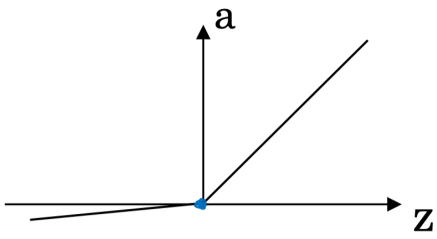


ReLU

$g(z) = \max(0, z)$

$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$

$z = 0.0000\ldots0$



Leaky ReLU

$g(z) = \max(0.01z, z)$

$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

Andrew

7. Neural network gradient:

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y \qquad dZ^{[2]} = A^{[2]} - Y$$

$$J(\cdot) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}, y)$$

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

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

$$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(z^{[1]}) \qquad dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

element-wise product

$$dW^{[1]} = dz^{[1]}x^{T} \qquad dW^{[1]} = \frac{1}{m}dZ^{[1]}X^{T}$$

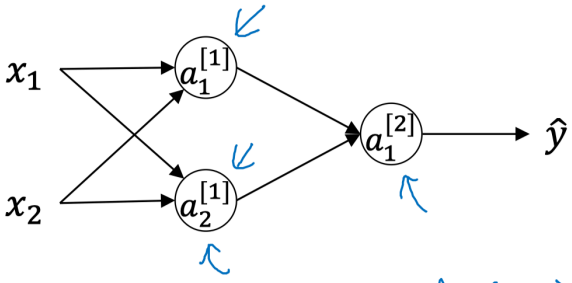
$$db^{[1]} = dz^{[1]} \qquad db^{[1]} = \frac{1}{m}np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

8. Random Initialization:

   If we initialize the w to be 0, then it will have symmetry breaking problem that is all function calculates the same value

   Why to choose 0.01 instead of 100: 如果w太大就会导致z太大，这样z就会在函数两端这样梯度变化就会很慢

Random initialization



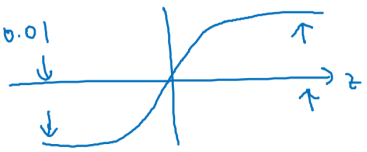
$W^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zeros((2,1))$

$W^{[2]} = np.random.randn((1,2)) * 0.01$

$b^{[2]} = 0$

100?



$z^{[1]} = W^{[1]} x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$