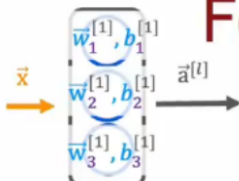


神经网络

输入前一层的激活，给定当前层的参数，它会输出下一层激活值 自定义sequential函数
中间层为网络层、隐藏层，最后输出为激活层

前向传播



Forward prop in NumPy

$$\vec{w}_1^{[1]} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \vec{w}_2^{[1]} = \begin{bmatrix} -3 \\ 4 \end{bmatrix} \quad \vec{w}_3^{[1]} = \begin{bmatrix} 5 \\ -6 \end{bmatrix}$$
$$W = \text{np.array}(\begin{bmatrix} 1 & -3 & 5 \\ 2 & 4 & -6 \end{bmatrix}) \quad 2 \text{ by } 3$$
$$b_1^{[1]} = -1 \quad b_2^{[1]} = 1 \quad b_3^{[1]} = 2$$
$$b = \text{np.array}([-1, 1, 2])$$
$$\vec{a}^{[0]} = \vec{x}$$

```
def dense(a_in, W, b, g):  
    3 units = W.shape[1] [0,0,0]  
    a_out = np.zeros(units)  
    for j in range(units): 0,1,2  
        w = W[:,j]  
        z = np.dot(w, a_in) + b[j]  
        a_out[j] = g(z)  
    return a_out
```

capital W refers to a matrix

```
def sequential(x):  
    a1 = dense(x, W1, b1)  
    a2 = dense(a1, W2, b2)  
    a3 = dense(a2, W3, b3)  
    a4 = dense(a3, W4, b4)  
    f_x = a4  
    return f_x
```

激活函数

二元分类：sigmoid 默认用relu(不丢失有效值) 线性激活一般不使用

hidden layer 隐藏层

除非是二分类问题用sigmoid，一般默认用relu，不能用线性激活函数，否则不能拟合，就相当于一个线性回归了

因为relu计算很快，并且只有右边扁平flat，而sigmoid有两处flat，会影响gd的速度（因为激活函数是计算偏导的一部分）

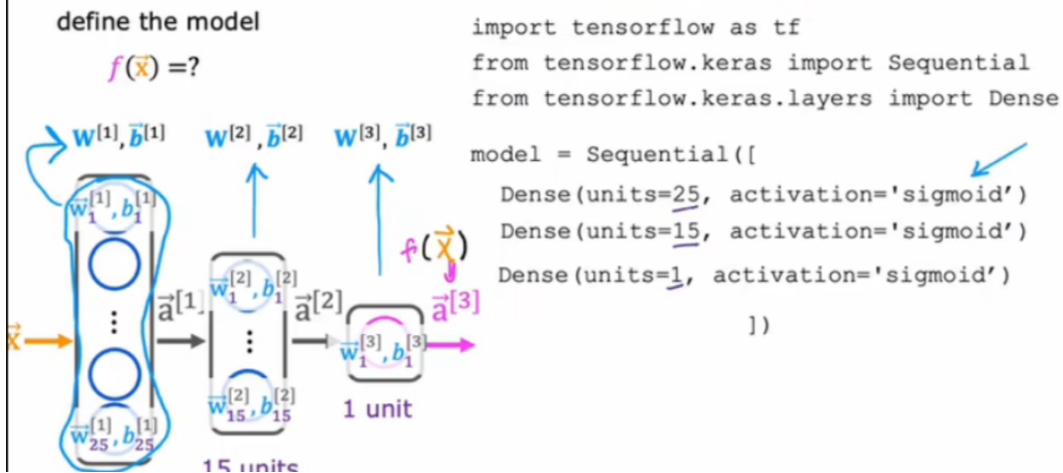
Tensorflow

可以直接调用不同的损失函数来训练模型

模型训练细节

此处二元交叉熵作为损失函数

1. Create the model



2. Loss and cost functions

Mnist digit classification problem \rightarrow binary classification

$$L(f(\vec{x}), y) = -y \log(f(\vec{x})) - (1 - y) \log(1 - f(\vec{x}))$$

Compare prediction vs. target

logistic loss
also known as binary cross entropy

$$J(W, B) = \frac{1}{m} \sum_{i=1}^m L(f(\vec{x}^{(i)}), y^{(i)})$$

$w^{[1]}, w^{[2]}, w^{[3]}$ $b^{[1]}, b^{[2]}, b^{[3]}$ $f_{W,B}(\vec{x})$

```
model.compile(loss= BinaryCrossentropy())
```

from tensorflow.keras.losses import BinaryCrossentropy **K** Keras

regression (predicting numbers and not categories) \rightarrow mean squared error

```
model.compile(loss= MeanSquaredError())
```

from tensorflow.keras.losses import MeanSquaredError

使用二元交叉熵作为损失函数时，通常使用以下公式来计算单个样本的损失：

$$\text{loss} = -[y \log(p) + (1 - y) \log(1 - p)] \quad \text{loss} = -[y \log(p) + (1 - y) \log(1 - p)] \quad \text{loss} = -[y \log(p) + (1 - y) \log(1 - p)]$$

其中， y 是真实标签（0或1）， p 是预测值（0到1之间的概率值）。这个公式表示，当真实标签为1时，我们希望预测值 p 越接近1，此时损失越小，等于 $-\log(p) - \log(p) - \log(p)$ ；

当真实标签为0时，我们希望预测值 p 越接近0，此时损失也越小，等于 $-\log(1 - p) - \log(1 - p) - \log(1 - p)$ 。

softmax 回归算法

Softmax 回归算法的基本思想是将输入数据映射到多个类别之一，并为每个类别分配一个概率。算法的输入是一个 n 维特征向量 x ，输出是 K 个类别的概率分布，其中 K 是类别的数量。（用于分类多元）

$$P(y = k | x, W) = \frac{\exp(w_k^T x)}{\sum_{j=1}^K \exp(w_j^T x)}$$

损失函数：

$$L(W; x, y) = - \sum_{k=1}^K y_k \log P(y = k|x, W)$$

softmax 改进实现，更加精确

理解不同的计算方式的精度是不同的，有的会在计算机存储过程中伴随值精度的损失(计组)
直接带入计算避免公式套用可以避免一些误差

卷积神经网络

本质是特征提取

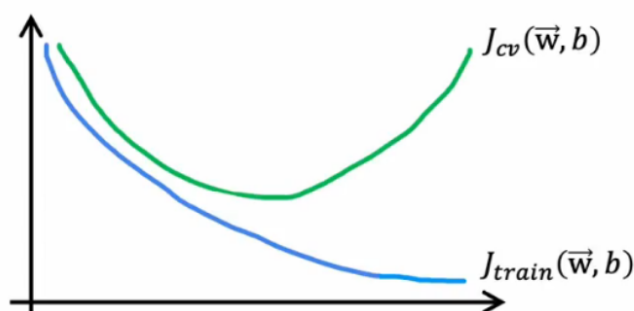
- **参数共享**：卷积操作中使用相同的滤波器减少了参数数量，提高了计算效率。
- **局部连接**：通过局部感受野提取图像局部特征，能够捕捉空间关系。

可保留一部分数据集用于测试集，再一部分作为交叉验证集

先通过训练集拟合 w, b ，然后通过交叉集选择出最优的模型也即参数 d ，最后用选择出来的模型在测试集中估计**泛化能力**，就是一个训练参数 w, b ，一个挑选最合适的一个模型，test用来算误差（敲定最终模型，才能在test集中评估）

Diagnosing bias and variance

How do you tell if your algorithm has a bias or variance problem?



High bias (underfit)

J_{train} will be high \leftarrow
($J_{train} \approx J_{cv}$)

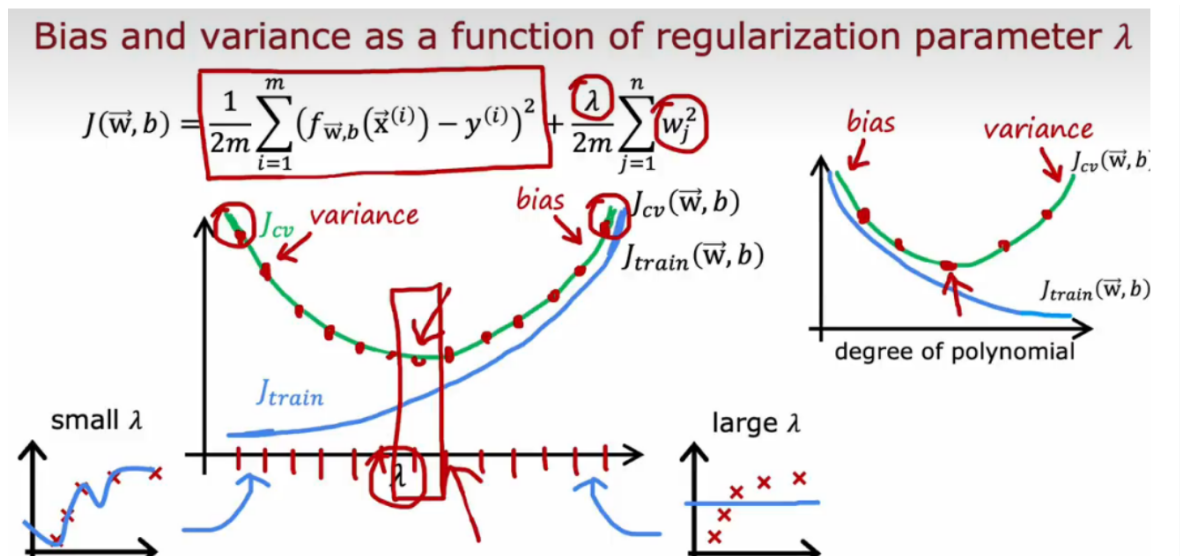
High variance (overfit)

$J_{cv} \gg J_{train}$
(J_{train} may be low)

High bias and high variance

J_{train} will be high
and $J_{cv} \gg J_{train}$

引入正则化



1. 基准baseline performance 和 J_{train} 之间的距离，可以判断是否有一个高偏差bias问题
2. J_{train} 和 J_{cv} 之间的距离可以判断是否有一个高方差variance

偏差为是否拟合训练集 方差为是否拟合交叉验证集 J_{cv}

迁移学习 transfer learning (监督预训练)

1. 微调方法一 (训练集很小)

前4个参数仍然保留，只需要更新输出层的参数，利用gd 或者adam优化算法来最小化代价函数

2. 微调方法二 (训练集很大)

重新训练所有参数，但前四层的参数还是用上面的值作为初始值！