Deep Learning Introduction

CSC Wuxi SDD2-2
Andy Zhou, Cao Fei
2021/08/31

AI、机器学习和深度学习的关系以及深度学习的发展脉络

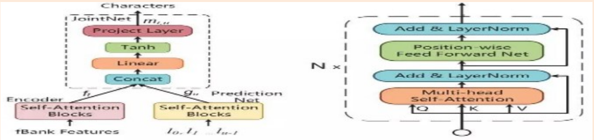
机器学习算法

- Decision Tree（决策树）
- Perceptron（感知器） → MLP（多层感知器）
- Bayes（经典贝叶斯）
- HMM（隐马尔可夫）
- SVM（支持向量机）
- Adaboost
- K-means（K均值）
- 随机森林

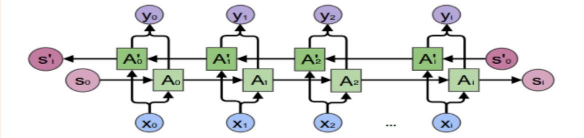
深度学习算法（神经网络系列）

MLP →

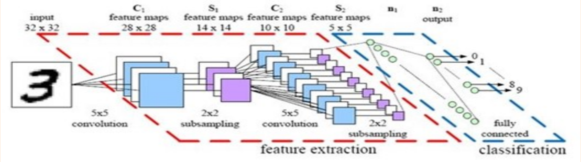
- Self-Attention（注意力机制）



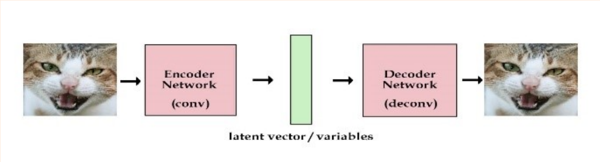
- RNN/LSTM（循环神经网络）



- CNN（卷积神经网络）

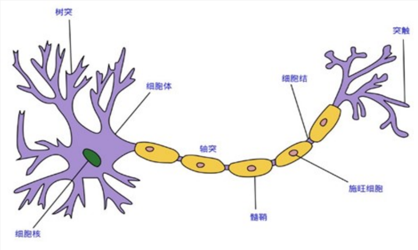


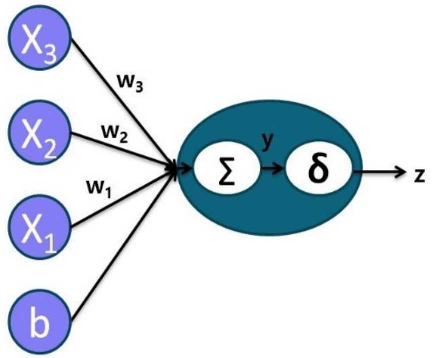
- VAE/GAN（变分自编码器）



深度学习之感知器（人工神经网络如何模拟生物神经网络）

1957年美国神经学家Frank根据大脑神经元的工作原理提出了感知器模型，最简单的前馈式神经网络诞生
#1 X 输入：模拟来神经网络中来自其他神经元的输入
#2 W权重：模拟每个神经元接收突触强度的不同，所以对于外界输入剩以一定的权重
#3 b偏置：模拟每个神经元的一般敏感性，每个神经元的敏感性不同，需要一定的偏差来调整
#4 求和：模拟生物神经网络对外界接收信号的汇总
#5 激活函数：模拟生物神经网络中信号累积到一定程度产生的电位动作，加强或减弱

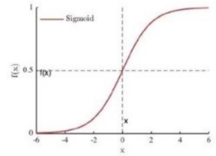




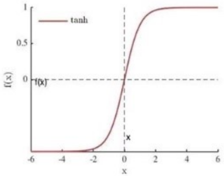
$$y = w_1x_1 + w_2x_2 + w_3x_3 + b = \sum wx + b$$

$$z = \delta(y)$$

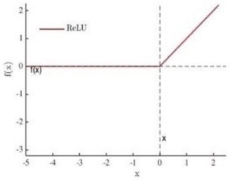
Sigmoid函数 $f(x) = \dfrac{1}{1+e^{-x}}$



Tanh函数 $\tanh(x) = \dfrac{1-e^{-2x}}{1+e^{-2x}}$



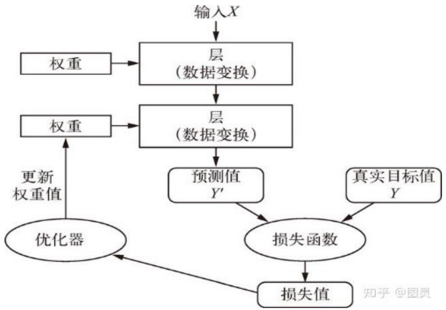
ReLU函数 $y = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$



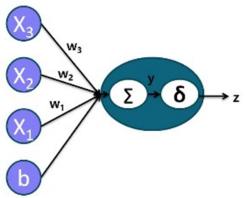
深度学习之MLP （神经网络学习到底怎么学习、训练和预测）

1986年，辛顿等提出了多层感知器模型，以及反馈网络的训练方法，也称BP神经网络，神经网络训练算法开始诞生并采用了sigmoid函数作为激活函数

Training processing



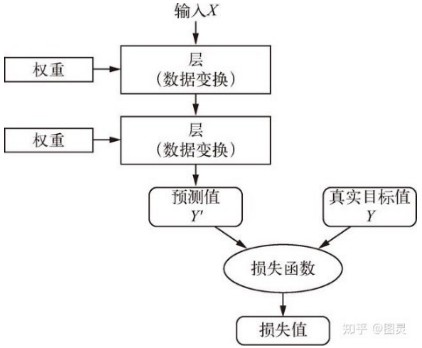
感知器

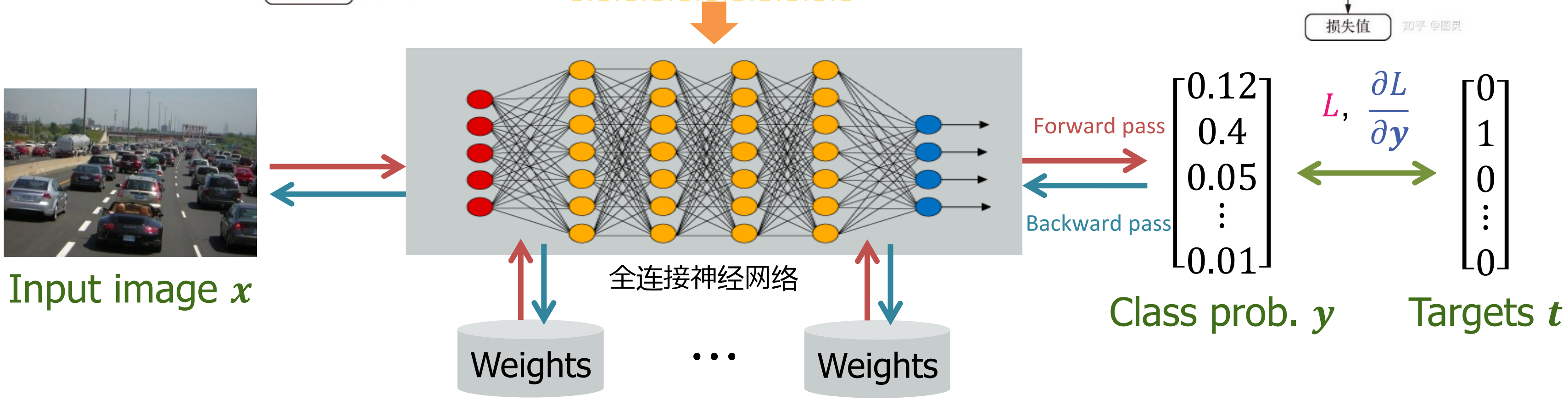


$$y = w_1x_1 + w_2x_2 + w_3x_3 + b = \sum wx + b$$

$$z = \delta(y)$$

Prediction processing

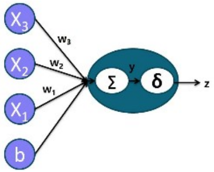




深度学习发展的关键事件

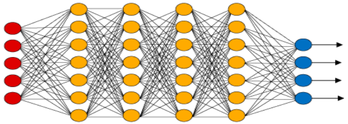
SONY

深度学习之父
辛顿



神经网络第一个20年

神经网络第二个20年的低潮期



Deep Neural Network (Pretraining)

SVM

Multi-layered Perceptron (Backpropagation)

XOR Problem

ADALINE

Perceptron

1989 CNN

1997 LSTM

Golden Age

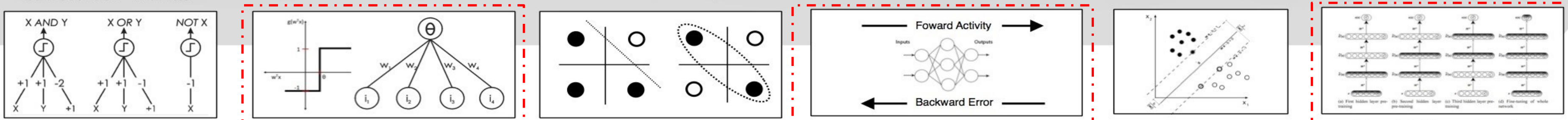
Dark Age ("AI Winter")

Electronic Brain

1943 | 1957 | 1960 | 1969 | 1986 | 1995 | 2006

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

S. McCulloch - W. Pitts | F. Rosenblatt | B. Widrow - M. Hoff | M. Minsky - S. Papert | D. Rumelhart - G. Hinton - R. Wiliams | V. Vapnik - C. Cortes | G. Hinton - S. Ruslan



- Adjustable Weights
- Weights are not Learned

- Learnable Weights and Threshold

- XOR Problem

- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting

- Limitations of learning prior knowledge
- Kernel function: Human Intervention

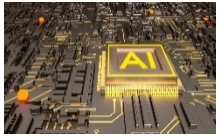
- Hierarchical feature Learning

深度学习为何在2010s开始火了

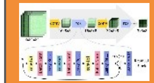
SONY

运算能力

运算能力的大幅度
提升 GPU/TPU



训练算法改进



新的训练方法解决深度
网络训练不成功的问题

#1 2006年，辛顿提出权重初始化方案解决深度
网络梯度消失问题(无监督初始化权重)
#2 2011年，辛顿首次使用ReLU激活函数，
有效的抑制梯度消失问题
#3 2012年，微软首次应用DL进行语音识别，
获得重大突破
#4 2016年的阿尔法go，深度学习进入大众视野

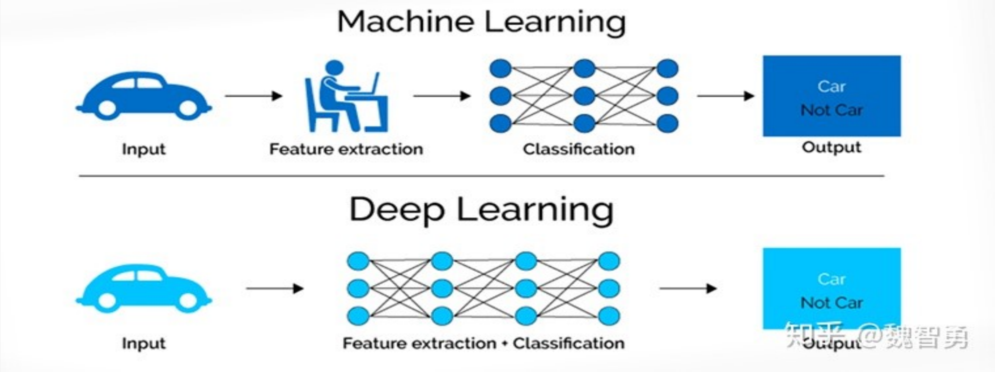
互联网大数据

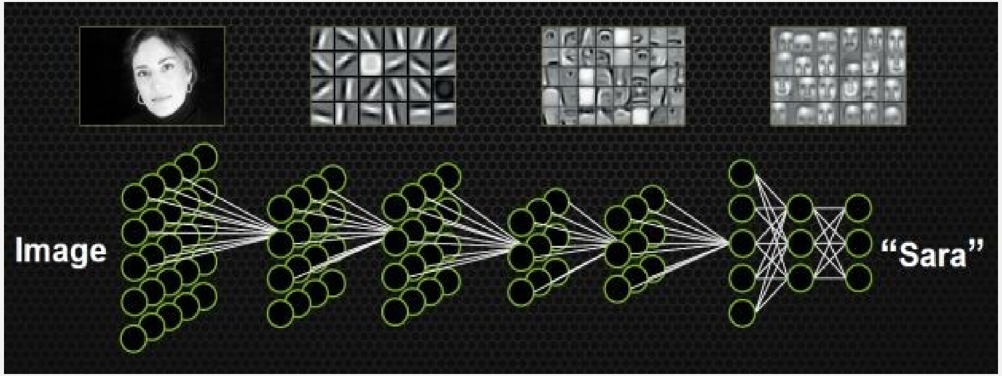
互联网时代
创造大数据

#1 Imagenet 140万的图像标注数据集，1000个类别
#2 维基百科是训练自然语言处理的数据宝库
#3 YouTube 视频也是一座宝库

机器学习和深度学习的区别

SONY

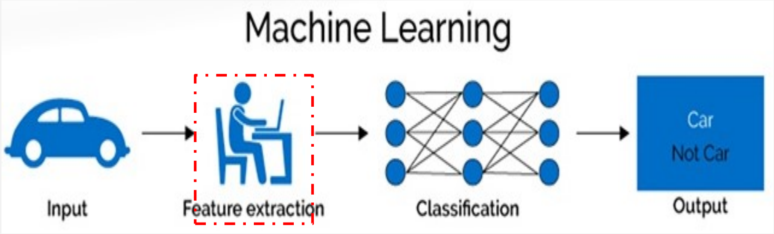




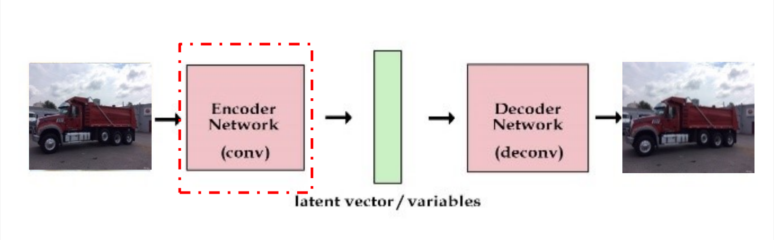
人工统计特征

人工结构特征

人工提取的特征



深度学习的预训练特征表示模型



学习的类别划分（无监督学习、监督学习和强化学习）

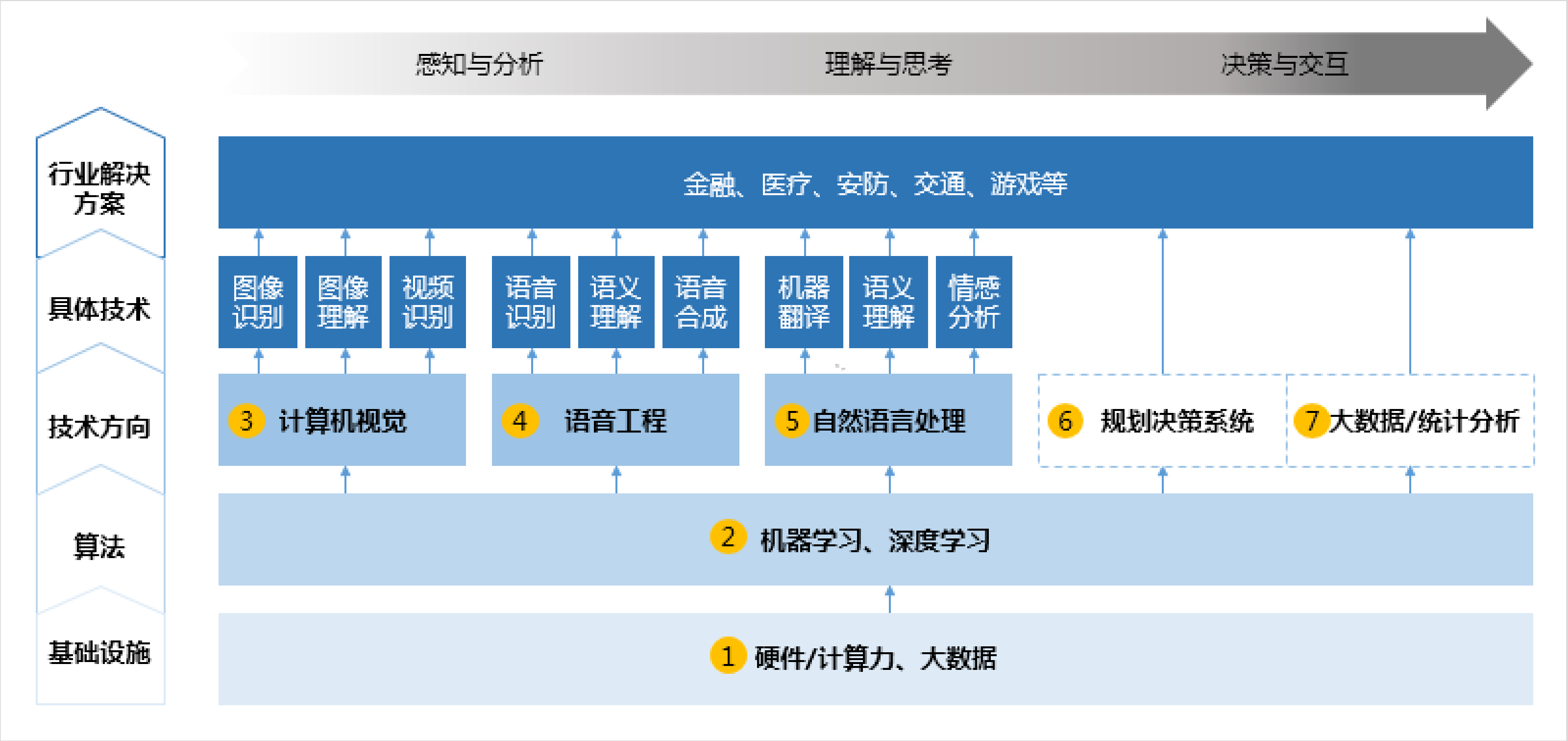
SONY

#监督学习：大多数的机器学习算法＋几乎所有的深度学习算法

#无监督学习：K-means

深度学习的应用场景

#目前，比较成熟的应用主要集中在3大领域，计算机视觉、语音工程和NLP

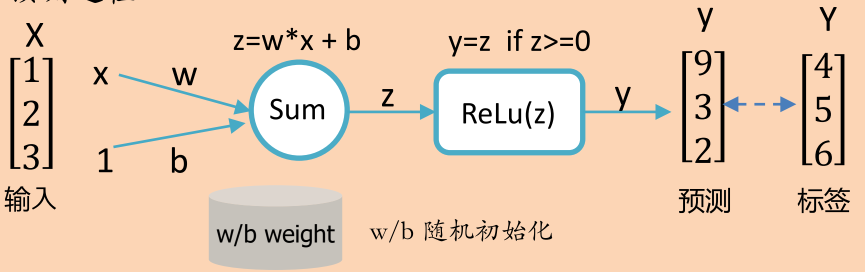


深度学习的学习原理- 线性回归

SONY

给定一些X=[1,2,3] Y=[4,5,6] 训练一个线性传感器神经网络

预测过程

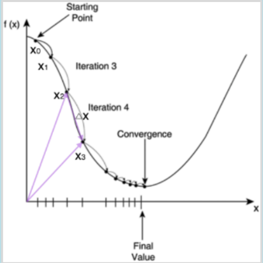


如何训练网络更新权重，使得预测值y与目标标签的误差减少?

训练过程：采用梯度下降法来训练和更新参数，先构建代价或损失函数，代价函数有很多中，但必须是凸函数

$$LossFun=(y-Y)^2/2$$

问题转化为求LossFun的极小点问题，所以沿着导数反方向更新权重则可以逼近极值点



α是学习率，属于超参数，优化训练效果，防止进入局部最优

$$w \leftarrow w - \alpha \frac{\partial L}{\partial w}$$

$$b \leftarrow b - \alpha \frac{\partial L}{\partial b}$$

z= $w * x + b$

y= z

$LossFun=\frac{1}{2}(y-Y)^2$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial z} * \frac{\partial z}{\partial w} = (y-Y) * x$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial z} * \frac{\partial z}{\partial b} = (y-Y)$$

w= $w - \alpha * (y-Y) * x$

b= $b - \alpha * (y-Y)$

深度学习的本质就是通过损失函数的误差反向传播来更新网络的权重参数，采用梯度下降法来实现此过程

深度学习- 线性回归的python代码实现

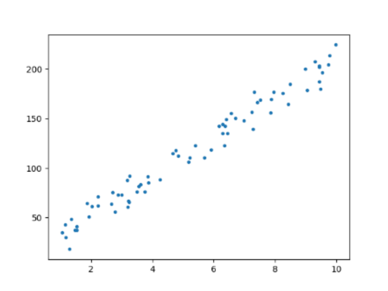
SONY

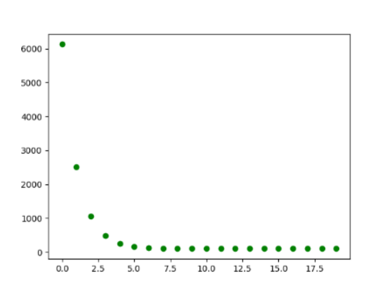
```python
def __init__(self, learning_rate=0.01, max_iter=100, seed=None):
    np.random.seed(seed)
    self.lr = learning_rate
    self.max_iter = max_iter
    self.w = np.random.normal(1, 0.1)
    self.b = np.random.normal(1, 0.1)
    self.loss_arr = []
```

```python
def fit(self, x, y):
    self.x = x
    self.y = y
    print(40*'=')
    print('max_iter=',self.max_iter)
    for i in range(self.max_iter):
        self._train_step(i)
        self.loss_arr.append(self.loss())
```

```python
def loss(self, y_true=None, y_pred=None):
    if y_true is None or y_pred is None:
        y_true = self.y
        y_pred = self.predict(self.x)
    return np.mean((y_true - y_pred)**2)
```

```python
def _f(self, x, w, b):
    return x * w + b

def predict(self, x=None):
    if x is None:
        x = self.x
    y_pred = self._f(x, self.w, self.b)
    return y_pred
```





z= $w * x + b$

y= z

$LossFun = \frac{1}{2}(y - Y)^2$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial z} * \frac{\partial z}{\partial w} = (y - Y) * x$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial z} * \frac{\partial z}{\partial b} = (y - Y)$$

$$w \leftarrow w - \alpha \frac{\partial L}{\partial w}$$

$$b \leftarrow b - \alpha \frac{\partial L}{\partial b}$$

w= $w - \alpha * (y - Y) * x$

b= $b - \alpha * (y - Y)$

```python
def _train_step(self,index):
    d_w, d_b = self._calc_gradient()
    self.w = self.w - self.lr * d_w
    self.b = self.b - self.lr * d_b
    return self.w, self.b
```

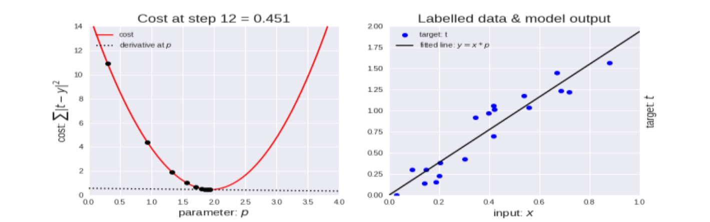
```python
def _calc_gradient(self):
    d_w = np.mean((self.x * self.w + self.b - self.y) * self.x)
    d_b = np.mean(self.x * self.w + self.b - self.y)
    return d_w, d_b
```

深度学习- 线性回归的python代码实现过程打印

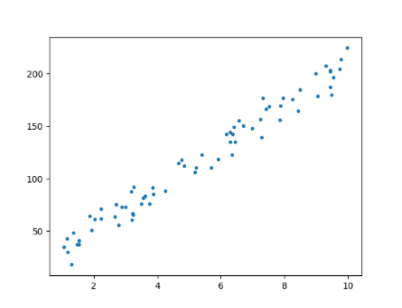

```
('x_train=', array([8.44340561, 7.87504622, 6.16401141, 3.22948482, 9.74028186,
        7.32064404, 1.93090899, 7.94965088, 1.53388415, 6.3675922 ,
        3.7540866 , 9.78171215, 2.88741486, 1.17483877, 7.85282943,
        1.53388668, 6.40979505, 7.53056809, 3.54680345, 5.91926314,
        2.65275974, 9.55238812, 3.85023415, 3.18006603, 9.43819064,
        3.87154281, 1.35479553, 3.21811252, 5.39565557, 2.69809158,
        2.77558845, 8.26495323, 3.25266916, 6.29276673, 3.48792901,
        3.61807215, 9.9880322 , 4.66348509, 5.21980683, 7.27448618,
        9.48460207, 1.04624586, 4.24228745, 4.7533448 , 8.98995125,
        2.21697786, 7.24899019, 6.56105895, 8.48405352, 7.41953419,
        2.02123466, 4.84655189, 6.34184487, 5.70244546, 1.28426277,
        5.18814828, 2.22014269, 6.99344605, 6.29956213, 6.44082128,
        6.71627606, 9.44108164, 3.00154469, 9.29960965, 9.43826648,
        3.19788601, 1.46440793, 1.86626583, 1.15467664, 9.04862078]))
('y_train=', array([164.73407324, 169.51999592, 142.12652646,  65.51657475,
        204.14393052, 176.67516972,  51.01576161, 176.53183029,
         40.9823875 , 142.17970458,  76.03776255, 213.79969288,
         72.82853888,  30.27234335, 155.85468932,  37.60390904,
        148.9399438 , 168.56871208,  81.87202055, 118.63195312,
         63.97333433, 196.45047117,  91.5249247 ,  87.69429952,
        203.25074828,  85.45662878,  48.29643605,  66.90170955,
        122.76863036,  75.63252695,  55.59511534, 175.47187277,
         91.79586416, 134.85666162,  75.94348705,  83.35553782,
        224.46660574, 114.52708478, 110.39994274, 139.11808577,
        180.12131644,  34.6720626 ,  88.2381613 , 118.06845207,
        199.92900887,  62.13748758, 156.37119427, 155.48197233,
        185.09056976, 166.20522842,  61.24036406, 112.18434645,
        122.56688228, 110.59635443,  18.11278399, 106.48057764,
         71.21639458, 148.04676026, 144.19303394, 134.99218941,
        150.29405312, 187.44241492,  73.21964224, 207.56966906,
        202.10856943,  60.92396975,  37.15381523,  64.25305999,
         42.99987739, 178.74176238]))
```

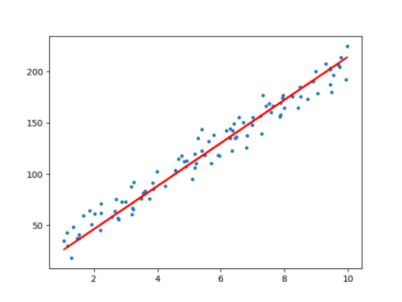
Firefox 网络浏览器

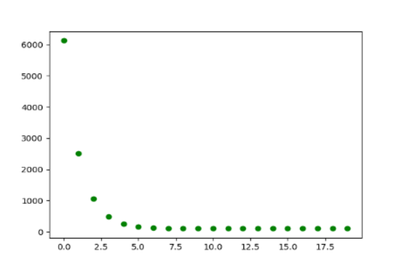
```
cost:     1.04e+02
w:        21.0
b:        4.41
```

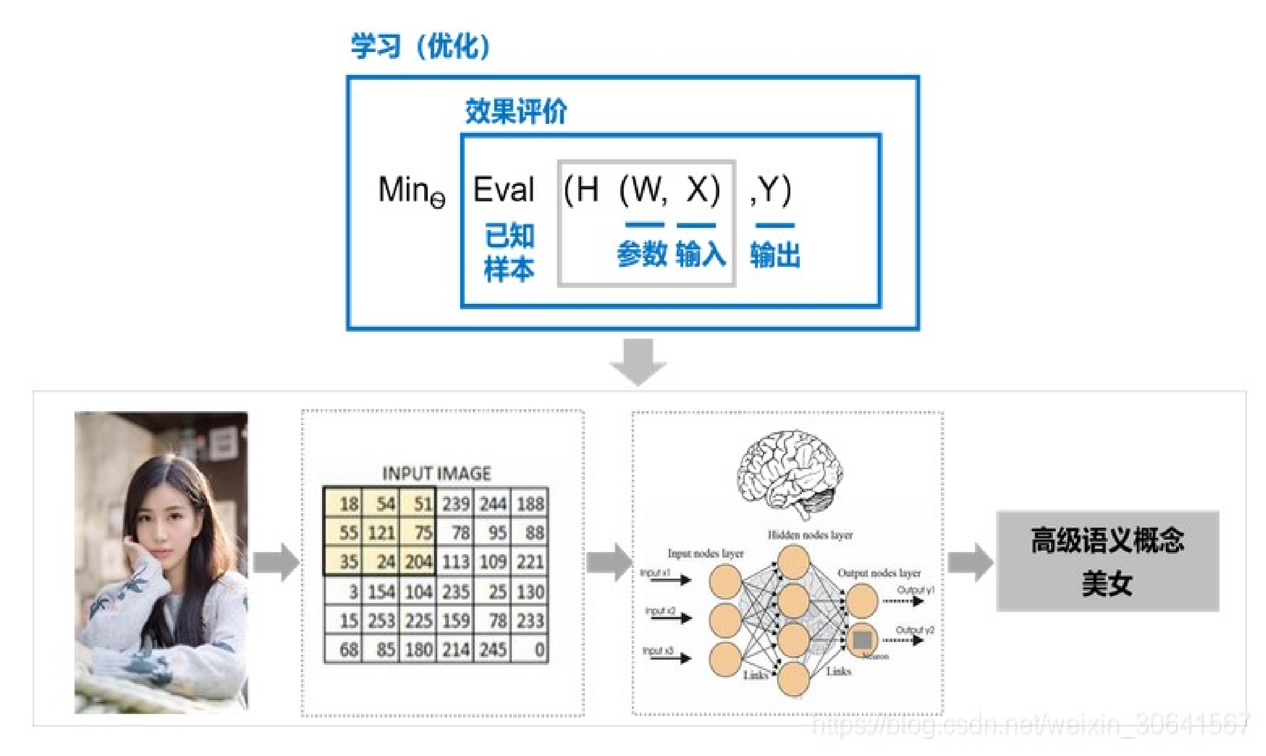



```
('max_iter=', 20)
('training step:', 0)
('update w=', 1.0166085438740533)
('update b=', 1.0781964482511646)

('training step:', 1)
('update w=', 8.380389461325384)
('update b=', 2.1940367397032876)

('training step:', 2)
('update w=', 13.033992588303857)
('update b=', 2.9059031254790675)

('training step:', 3)
('update w=', 15.974518624296376)
('update b=', 3.3624077950208484)

('training step:', 4)
('update w=', 17.832225981342457)
('update b=', 3.6574872089390493)

('training step:', 5)
('update w=', 19.005495349230436)
('update b=', 3.850517789596209)

('training step:', 6)
('update w=', 19.746139587607274)
('update b=', 3.9790308688487133)

('training step:', 7)
('update w=', 20.213327344234976)
('update b=', 4.066749692563257)

('training step:', 8)
('update w=', 20.507667690827667)
('update b=', 4.12866953914428)

('training step:', 9)
('update w=', 20.692754715898122)
('update b=', 4.174268816927319)

('training step:', 10)
('update w=', 20.80878566881368)
('update b=', 4.209538770063646)
```









- **科学定律：** $f($ F=5 $)$ = 2.5
- **图像识别：** $f($  $)$ =“美女”
- **机器翻译：** $f($ “Rain cats and dogs” $)$ = “倾盆大雨”
- **自动问答：** $f($ “姚明是哪里人？” $)$ = “上海”

任何深度神经网络模型，都可以用一个f(x)去表示，都是一个链式嵌套函数，区别在于每一层嵌套函数的复杂程度和嵌套函数的公式不同

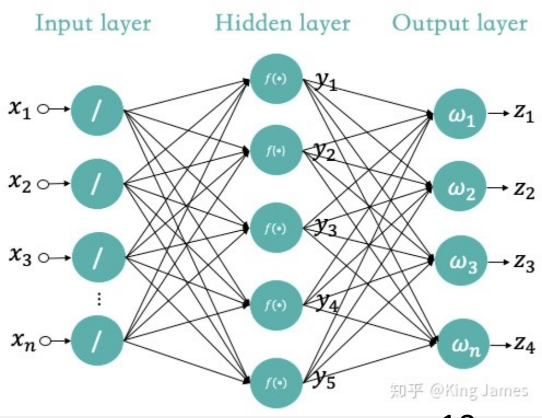
所以，无论模型如何变化，训练过程，都是链式求导，迭代更新网络参数的过程，直至训练结束

深度学习的本质就是通过损失函数的误差反向传播来更新网络参数过程，这种学习是对参数的改变来使得预测值进一步的接近目标值的过程。

深度学习实践案例- 使用MLP完成手写数字识别

SONY

MLP 模型示意图



512 10

Keras 模型创建代码

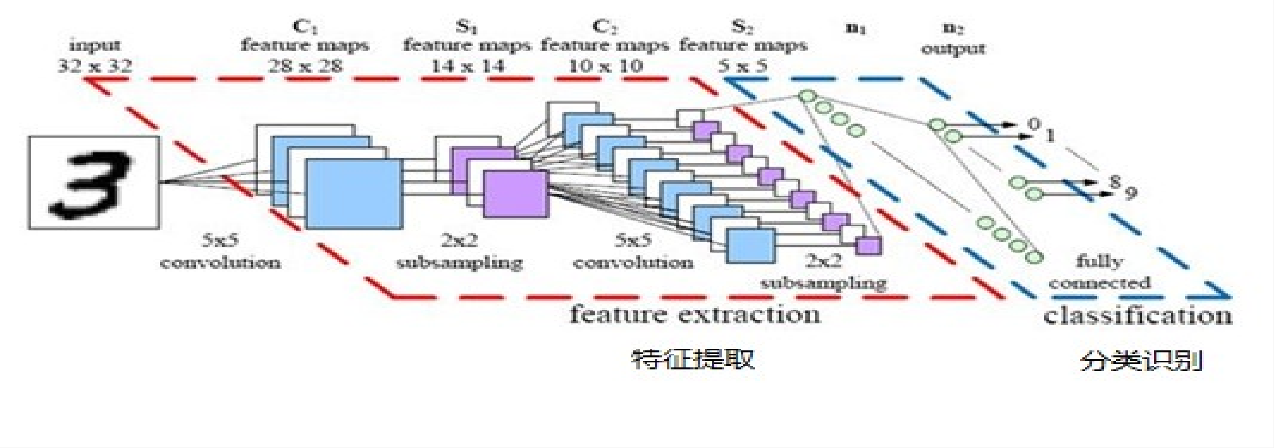
```python
def create_model():
    model=models.Sequential()
    model.add(layers.Dense(512,activation='relu',input_shape=(28*28,)))
    model.add(layers.Dense(10,activation='softmax'))
    model.summary()
    model.save('mlp.h5')
    return model
```

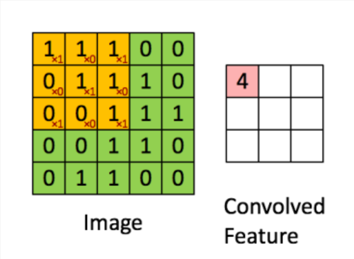
网络参数和训练过程

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
(60000, 784)
(60000, 10)
Epoch 1/10
469/469 [==============================] - 1s 2ms/step - loss: 0.2547 - accuracy: 0.9250
Epoch 2/10
469/469 [==============================] - 1s 2ms/step - loss: 0.1040 - accuracy: 0.9686
Epoch 3/10
469/469 [==============================] - 1s 2ms/step - loss: 0.0678 - accuracy: 0.9799
Epoch 4/10
469/469 [==============================] - 1s 2ms/step - loss: 0.0490 - accuracy: 0.9849
Epoch 5/10
469/469 [==============================] - 1s 2ms/step - loss: 0.0370 - accuracy: 0.9890
Epoch 6/10
469/469 [==============================] - 1s 2ms/step - loss: 0.0282 - accuracy: 0.9917
Epoch 7/10
469/469 [==============================] - 1s 2ms/step - loss: 0.0217 - accuracy: 0.9937
Epoch 8/10
469/469 [==============================] - 1s 2ms/step - loss: 0.0171 - accuracy: 0.9951
Epoch 9/10
469/469 [==============================] - 1s 2ms/step - loss: 0.0130 - accuracy: 0.9963
Epoch 10/10
469/469 [==============================] - 1s 2ms/step - loss: 0.0100 - accuracy: 0.9969
313/313 [==============================] - 0s 451us/step - loss: 0.0704 - accuracy: 0.9810
loss= 0.0703979954123497
acc= 0.9810000061988831
(10,)
1.0
7
```

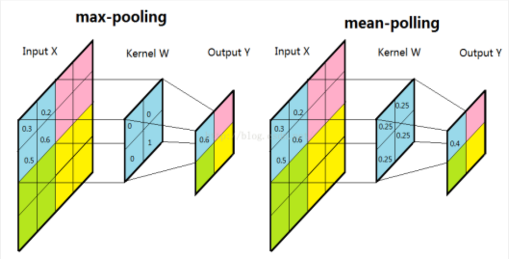
深度学习实践案例- 使用CNN完成手写数字识别

CNN 示意图

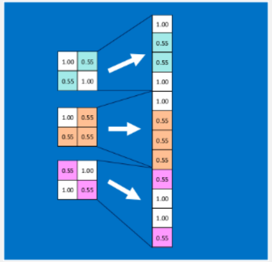




卷积

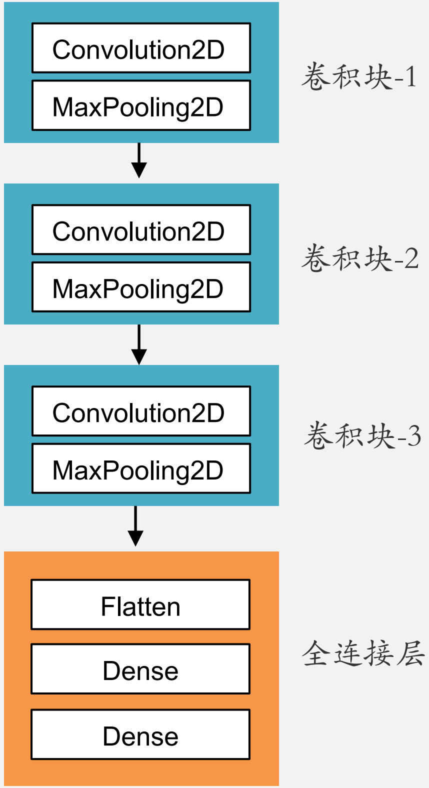


池化



展平 3D->1D

卷积网络设计



深度学习实践案例- 使用CNN完成手写数字识别

手写数字训练数据库

CNN模型创建代码

```python
def create_model():
    model=models.Sequential()
    model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(3,3),activation='relu'))
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(3,3),activation='relu'))
    model.add(layers.Flatten()) #3D->1D
    model.add(layers.Dense(64,activation='relu'))
    model.add(layers.Dense(10,activation='softmax'))
    model.summary()
    model.save('cnn.h5')
    return model
```

模型参数和训练过程

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
(60000, 28, 28, 1)
(60000, 10)
Epoch 1/10
938/938 [==============================] - 8s 8ms/step - loss: 0.1781 - accuracy: 0.9445
Epoch 2/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0479 - accuracy: 0.9854
Epoch 3/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0330 - accuracy: 0.9903
Epoch 4/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0255 - accuracy: 0.9919
Epoch 5/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0201 - accuracy: 0.9939
Epoch 6/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0163 - accuracy: 0.9952
Epoch 7/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0128 - accuracy: 0.9961
Epoch 8/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0113 - accuracy: 0.9966
Epoch 9/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0103 - accuracy: 0.9970
Epoch 10/10
938/938 [==============================] - 8s 9ms/step - loss: 0.0085 - accuracy: 0.9975
313/313 [==============================] - 0s 1ms/step - loss: 0.0363 - accuracy: 0.9922
loss= 0.036258965730667114
acc= 0.9922000169754028
(10,)
1.0
7
```

Thank You