



Python程序设计（混合式）



杨尚东

南京邮电大学计算机学院，数据科学与工程系

shangdongyang.github.io

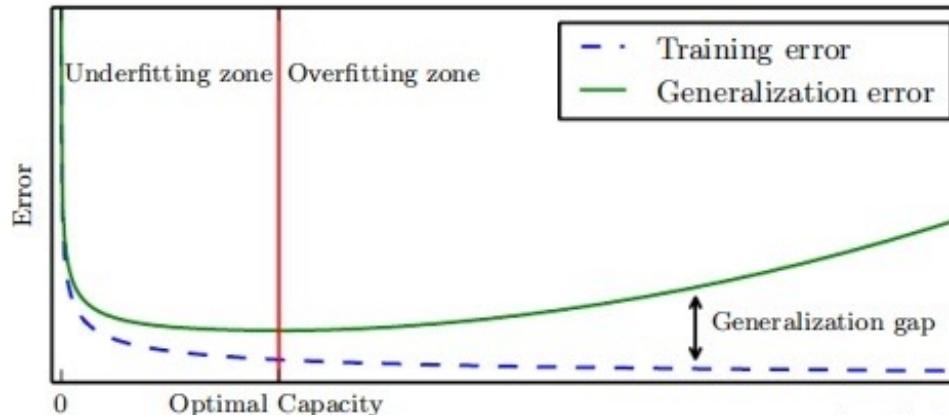
2023/12/8

目录

- 神经网络简介
- TensorFlow
- Keras
- 全连接神经网络及Keras实现
- 全连接神经网络的自定义编码实现
- 卷积神经网络及TensorFlow实现
- AlexNet编码实现

没有免费的午餐 (No free lunch)

- 在训练模型的过程中，对于有参模型而言，选择模型的容量对应着可以得到的假设空间。假若模型容量不够大，或者没有涵盖到真实的目标函数 f ，就会导致欠拟合.
- 反之，若模型的容量过大（自由度过高），就会产生过拟合情况。

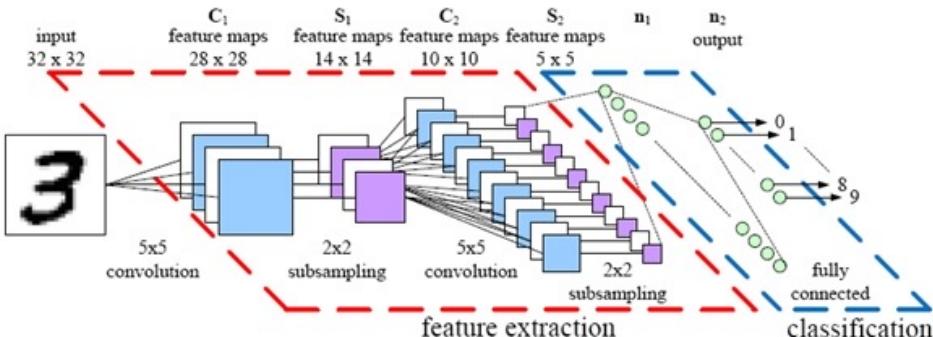


- No Free Lunch Theorem

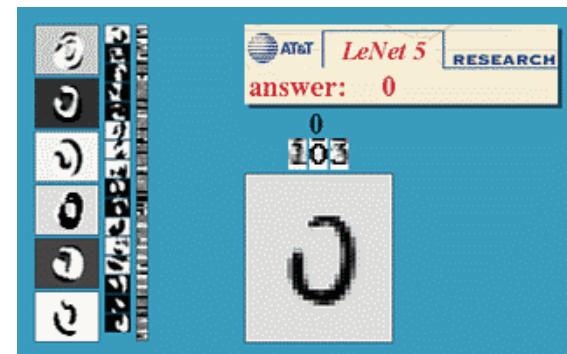
- ✓ 没有一种通用的学习算法可以在各种任务中都有很好的表现
- ✓ 一种算法（算法A）在特定数据集上的表现优于另一种算法（算法B）的同时，一定伴随着算法A在另外某一个特定的数据集上有着不如算法B的表现；

神经网络简介

- 如何分类图片？
- 神经网络：通过从样本中学习特征和标签间的关系，便可以根据新样本的特征计算其标签。



0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9

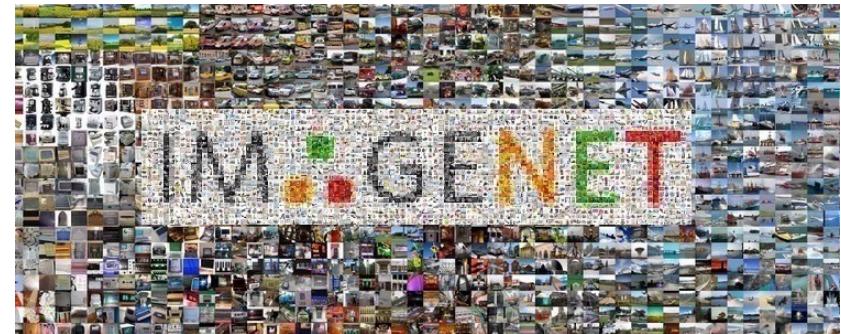


原因1：大数据

□ 随着信息化、互联网的高速发展，数据量爆发式地增长，数据维度越来越丰富

□ ImageNet

- ✓ ImageNet项目是一个用于视觉对象识别软件研究的大型可视化数据库。超过1400万的图像URL被ImageNet手动注释，以指示图片中的对象；



- 互联网数据
- 用户行为数据
- 结构数据

数据量大

- 用户属性
- 商品维度
- 时间维度
- 多媒体
- ...

维度高

- 图像
- 视频
- 文本
- 语音
- ...

类型多样

原因2：运算能力提升

口 摩尔定律：集成电路上可以容纳的晶体管数目在大约每经过18个月便会增加一倍

2013 780Ti

流处理器: 512个
核心频率:1206MHz
显存频率:7000MHz
显存规格:2G GDDR5
显存位宽:128bit
输出接口:DVI-D/DP/HDMI



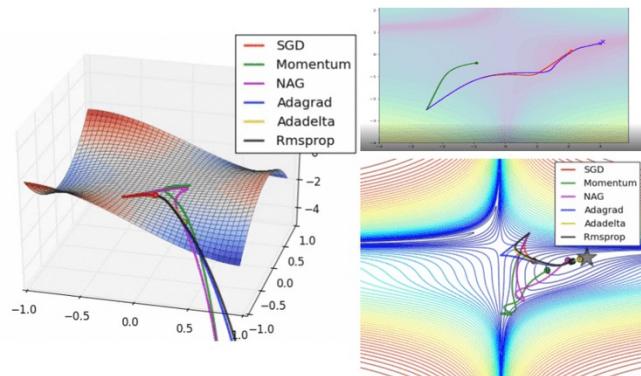
产品参数	
显示核心	GeForce RTX 3090
CUDA数量	10496
核心频率	1695Mhz
一键超频	1785Mhz
显存容量	24GB
显存类型	GDDR6X
显存位宽	384Bit
电源接口	3*8Pin
推荐电源	750W及以上
散 热	三风扇散热+6*8mm热管
产品尺寸	323*158*60mm
输出接口	3*DP+HDMI

GeForce RTX 4090	
Architecture	Ada (TSMC N4)
GPU	AD102-300
Board Number	PG139-SKU330
SMs	126 -> 128
CUDA Cores	16128 -> 16384
CUDA vs Predecessor	+56% (vs 3090)
Memory	24 GB G6X
Memory Bus	384-bit
Memory Speed	21 Gbps
TDP	-450W
Launch Date	September-October 2022

原因3：神经网络设计与优化方法

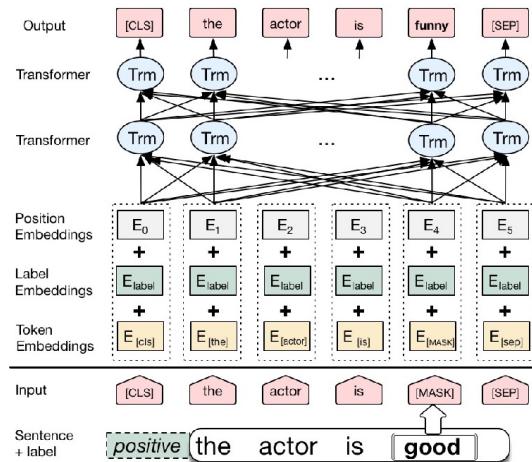
口 优化方法

- ✓ 神经网络的目标函数是非凸的



口 网络结构

- ✓ 面对大量的数据，需要更复杂的网络结构



神经网络简介

- 以图像分类为例，给定若干个 $N \times N$ 维像素矩阵组成的图像样本集，其中 s 为 S 中的一个样本， **x 为样本 s 的特征**。这里将像素作为图像样本的特征。神经网络根据样本学习得到一个函数 $f()$ ，该函数的值便是当前函数对图片样本的分类结果，公式如下：

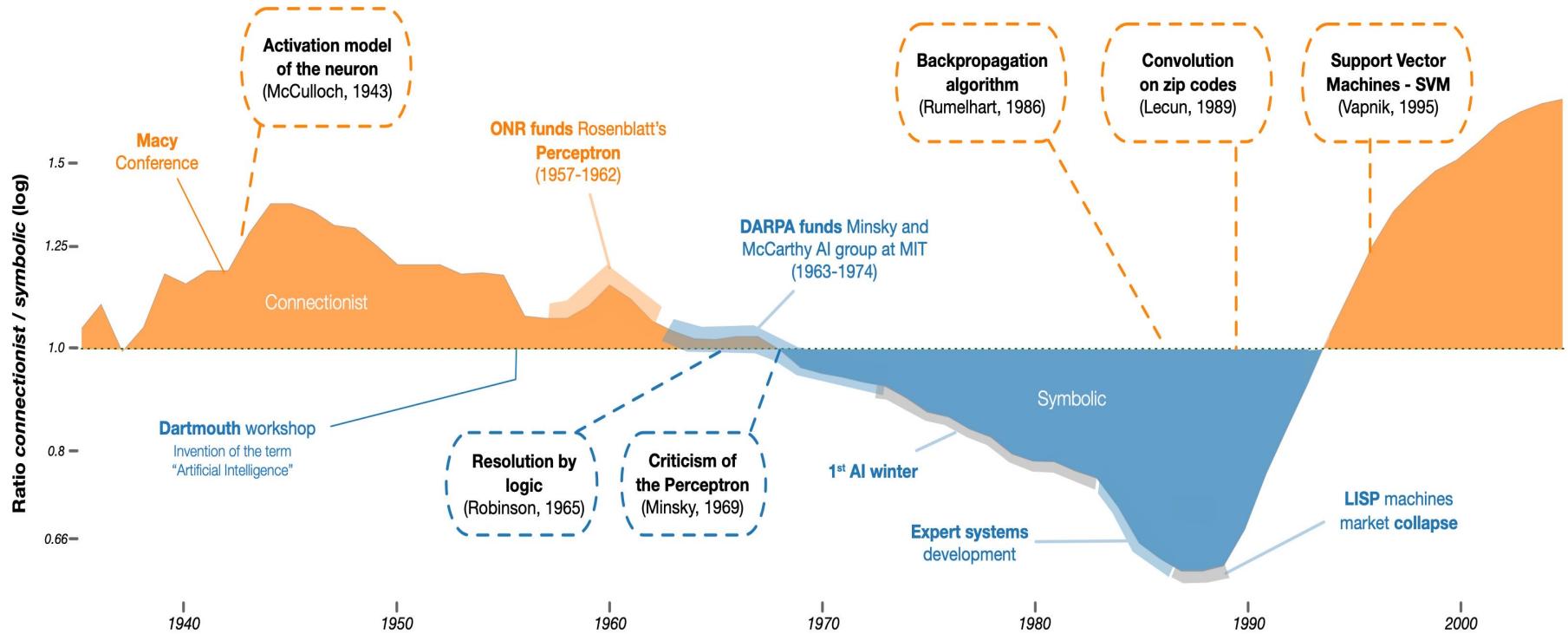
$$\hat{y} = f(x) = f(Wx + \theta)$$

- 该公式就是需要学习的模型。其中， $f()$ 为神经网络函数， W 为分类函数中的投影矩阵， θ 为偏置项。神经网络要学习和调整的参数为 W 和 θ 。训练神经网络的过程就是在寻找一条合适的曲线的过程，使得定义的目标损失函数最小。

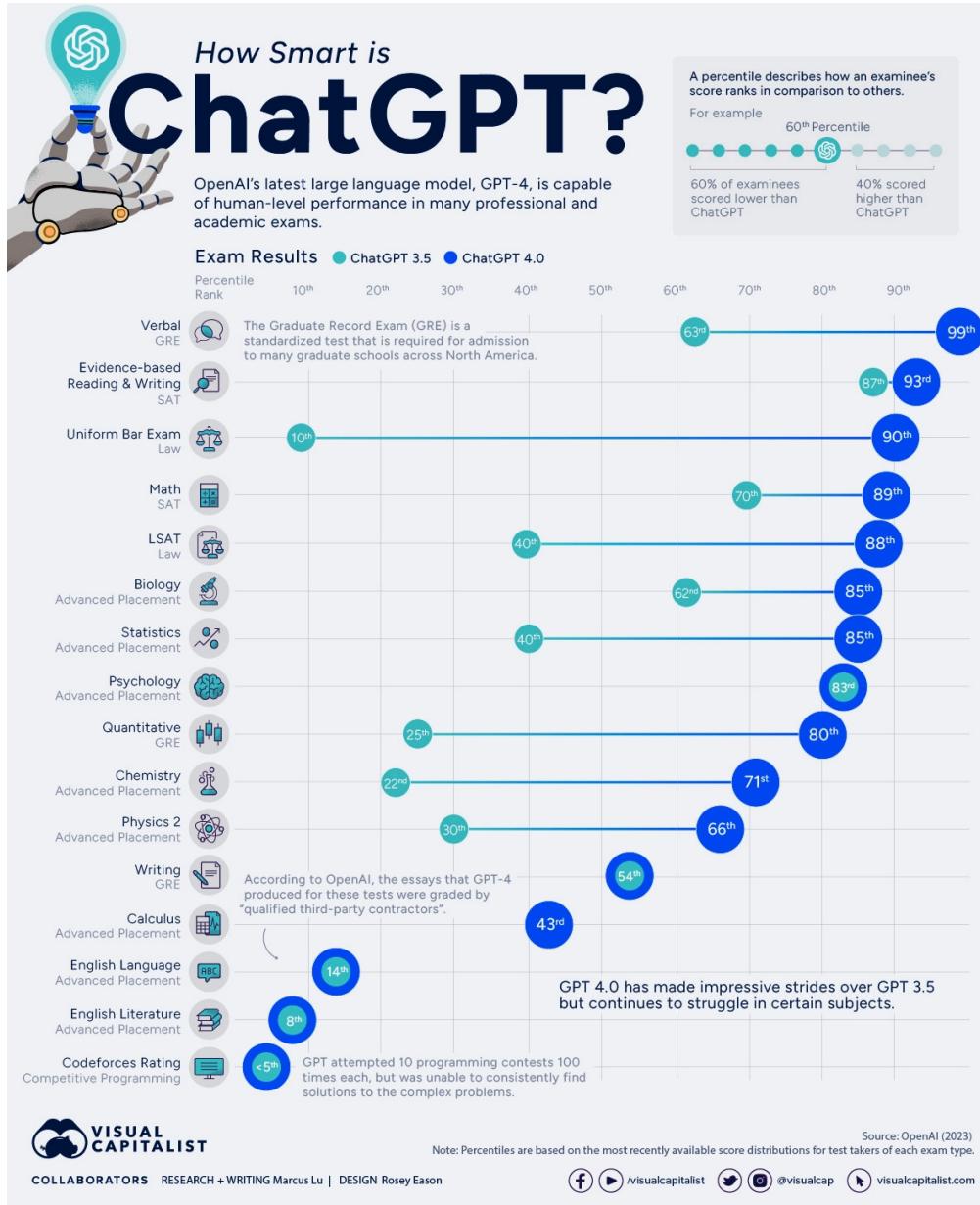
□ 特点

- ✓ 模型参数多、容量大、非线性
- ✓ 高维特征处理能力
- ✓ 更容易过拟合

神经网络发展



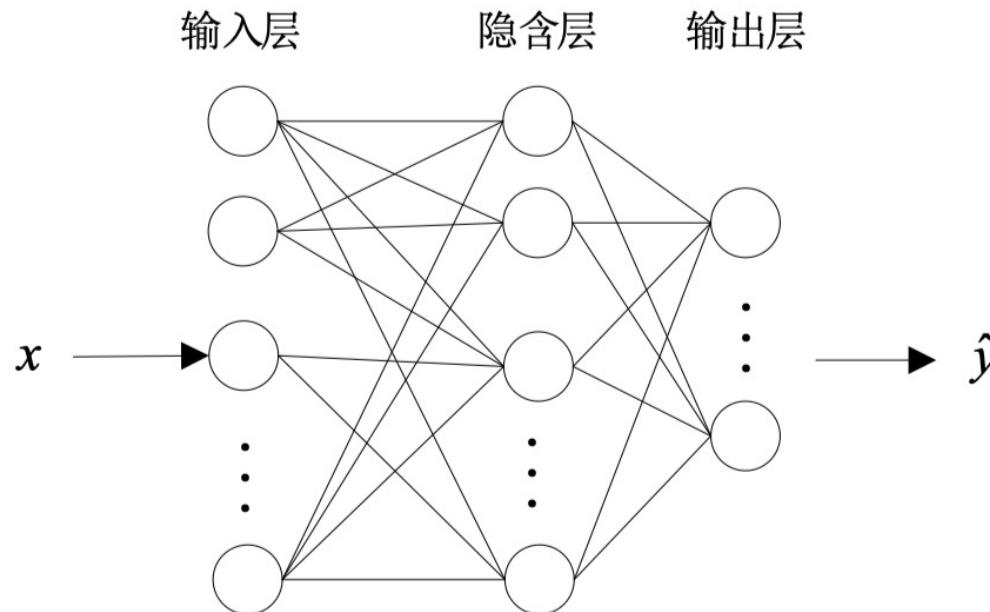
ChatGPT



全连接神经网络的基本原理

□ 全连接神经网络包含三层结构：**输入层、隐含层（一层或多层）、输出层**。所谓全连接，就是**每一层中的神经元与后面一层的各个神经元都有连接**，同层神经元之间不存在连接，也不存在跨层连接。

- ✓ **输入层**用来接收样本的特征输入；
- ✓ **隐含层**用于对输入的特征进行加工处理，通过其内部神经元的激活函数实现；
- ✓ **输出层**用于接收隐含层的加工信息并输出分类结果。

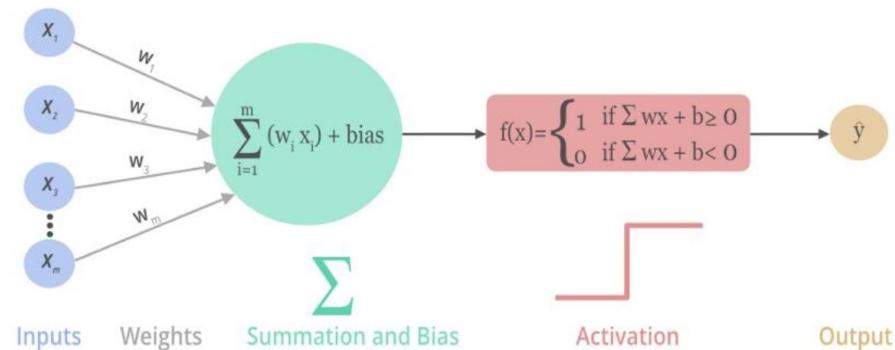


全连接神经网络的基本原理

□ 前向传播（计算过程）

$$h(x) = g(W_1 x + \theta_1)$$

$$\hat{y} = f(W_2 h(x) + \theta_2)$$

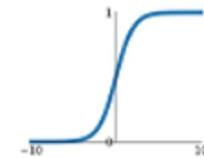


□ 激活函数： g, f 为**激活函数**，常用激活函数如下

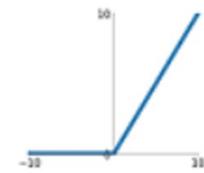
激活函数	表达式	特点
sigmoid	$S(x) = \frac{1}{1 + e^{-x}}$	将数值映射至(0, 1)的区间；在数值相差不大时效果较好
ReLU	$\text{ReLU}(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$	计算过程简单，目前最受欢迎的激活函数
Softmax	$S_i = \frac{e^i}{\sum_j e^j}$	将数值映射至(0, 1)的区间；通常用于多分类神经网络输出

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



ReLU
 $\max(0, x)$



全连接神经网络的基本原理

- 反向传播（训练过程）

$$Loss = \frac{1}{2K} \sum_{k=1}^K (\hat{y}^k - y^k)^2$$

（当损失函数为均方误差时）

- 反向传播基于梯度下降算法，以**目标的负梯度方向**对参数进行更新。

TensorFlow

- TensorFlow 是由谷歌在 2015 年 11 月发布的深度学习开源工具，可用来快速构建神经网络，并训练深度学习模型。
- TensorFlow 的底端由 C++ 实现，计算速度更快。TensorFlow 有一个高级机器学习 API (tf.contrib.learn)，可以更容易地配置、训练和评估大量的机器学习模型。

	PyTorch	TensorFlow
计算图分类	动态计算图	静态计算图
计算图定义	计算图在运行时定义	计算图需提前定义
调试	较简单，可用任何 python 开发工具 (例如：PyCharm)	较复杂，只能用专为 Tensorflow 开发的工具 (例如：tfdbg)
可视化	支持 Tensorboard	支持 Tensorboard
数据并行	极其简单，只需一行代码	较复杂，需要手动配置
支持硬件	CPU, GPU	CPU, GPU
支持语言	Python, C++	Python, C++
开发公司	Facebook	Google

TensorFlow

□ TensorFlow2.0的安装

- ✓ pip install tensorflow #安装cpu版
- ✓ pip install tensorflow-gpu #安装gpu版
- ✓ import tensorflow as tf

TensorFlow

□ TensorFlow2.0的张量

- ✓ 与NumPy类似，1维的数组称之为向量，2维的数组称之为矩阵，
N维数组则称之为张量（Tensor）。
- ✓ TensorFlow 中，数据都使用张量进行表述。
- ✓ tf.Variable : Tensor变量
- ✓ tf.constant : Tensor常量
- ✓ 基本运算
- ✓ 动微分和梯度计算

TensorFlow

□ TensorFlow2.0的常用模块

- ✓ tf: 包含了张量定义、变换等常用函数和类；
- ✓ tf.data: 输入数据处理模块，提供了如 tf.data.Dataset 等类，用于封装输入数据、指定批量大小等；
- ✓ tf.image: 图像处理模块，提供了如图像裁剪、变换、编码、解码等类；
- ✓ tf.keras: 原 Keras 框架高阶 API，包含原 tf.layers 中的高阶神经网络层；
- ✓ tf.linalg: 线性代数模块，提供了大量线性代数计算方法和类；
- ✓ tf.losses: 损失函数模块，便于定义神经网络的损失函数；
- ✓ tf.math: 数学计算模块，提供了大量数学计算函数；
- ✓ tf.saved_model: 模型保存模块，可用于模型的保存和恢复；
- ✓ tf.train: 提供用于训练的组件，例如优化器、学习率衰减策略等；
- ✓ tf.nn: 提供用于构建神经网络的底层函数，以帮助实现深度神经网络的各类功能层；
- ✓ tf.estimator: 高阶 API，提供了预创建的 Estimator 或自定义组件。

Keras

- Keras是由纯Python编写的面向对象的基于TensorFlow的深度学习库。
- Keras包含许多常用神经网络构建块的实现，例如层、目标函数、激活函数、优化器和一系列工具，其代码托管在Github上。
- Keras里有两种搭建神经网络模型的方式，一种是 **Sequential 模型**，另一种是 **Model 模型**。

Keras

□ Keras的安装

- ✓ pip install keras
- ✓ import keras

Keras

□ Keras的序列模型

1. 构建模型

```
from keras import models # 导入models模块用于组装各个组件
```

```
from keras import layers # 导入layers模块用于生成神经网络层
```

```
neural_net = models.Sequential() # 组合层级叠加的网络架构
```

```
# 添加模型的网络层
```

```
neural_net.add(layers.Dense(units=64, activation='relu', input_shape=(28*28, )))
```

```
neural_net.add(layers.Dense(units=64, activation='relu'))
```

```
neural_net.add(layers.Dense(units=10, activation='softmax'))
```

```
sequential.summary() # 输出网络初始化的各层的参数状况
```

Keras

□ Keras的序列模型

2. 训练模型

```
neural_net.compile(optimizer= keras.optimizers.SGD(lr = 0.1), loss='mse',
                     metrics=['accuracy'])      # 配置网络的训练方法
history = neural_net.fit(x, y, epochs, batch_size) # 执行模型的训练
```

3. 测试模型

```
test_loss, test_acc = neural_net.evaluate(data, label, batch_size) # 测试模型
```

Keras

□ Keras的函数式模型

1. 构建模型

```
from keras.layers import Input, Dense      # Input用于接收模型的输入； Dense  
                                         # 用于构建全连接层  
  
from keras.models import Model            # 用于构建Model 模型  
  
inputs = Input(shape=(784,))              # 设置模型输入数据的格式  
  
x = Dense(64, activation='relu')(inputs)   # 全连接层的输入层  
  
x = Dense(64, activation='relu')(x)        # 全连接层的隐藏层  
  
predictions = Dense(10, activation='softmax')(x) # 全连接层的输出层  
  
model = Model(inputs=inputs, outputs=predictions) # 构建模型并设置模型的  
                                                 # 输入和输出
```

Keras

□ Keras的函数式模型

2. 训练模型

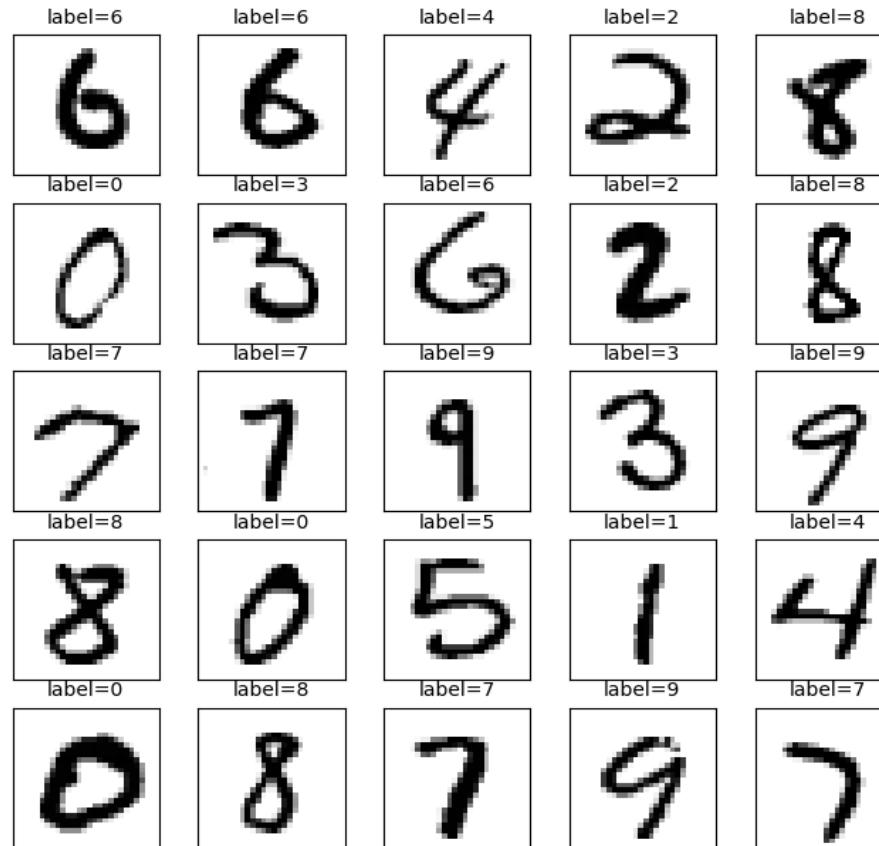
```
# 设置模型的优化器、损失函数、性能指标  
model.compile(optimizer='rmaprop',  
                loss='categorical_crossentropy',  
                metrics=['accuracy'])  
  
model.fit(data,labels) # 训练模型
```

3. 测试模型

```
model.predict(self, x, batch_size=32, verbose=0) # 测试模型
```

基于Keras库构建全连接神经网络

口以手写数字图片数据集MNIST为例，实现对手写数字图片的分类与识别



基于Keras库构建全连接神经网络

- 导入所用框架和库
- 获取并处理训练集和测试集
- 构建全连接神经网络

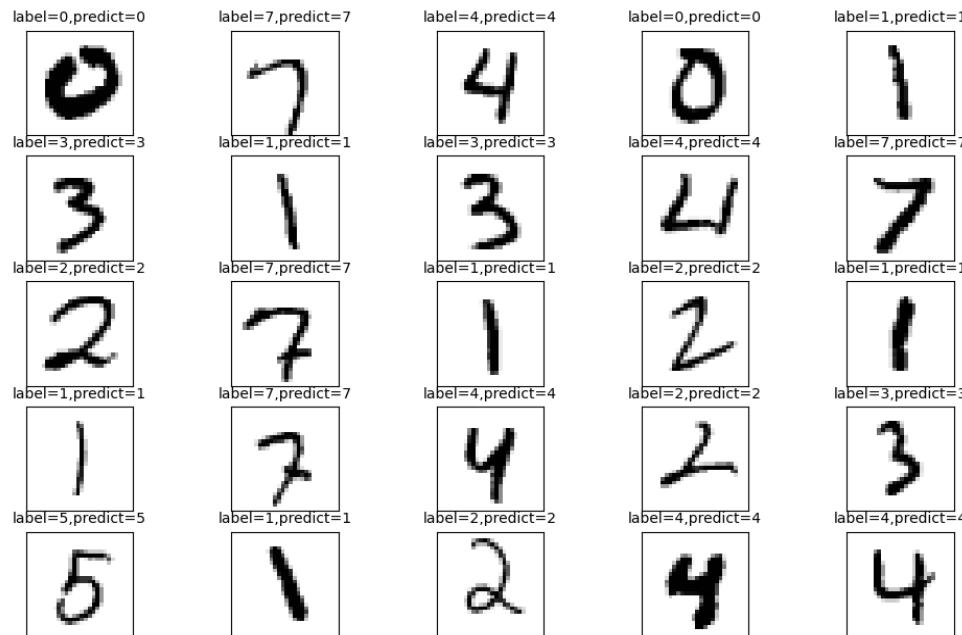
```
# 组合层级叠加的网络架构
neural_net = models.Sequential()
# 设置隐含层的神经元 # 数量和激活函数
neural_net.add(layers.Dense(units=512, activation='relu',
                            input_shape=(28*28, )))
# 设置输出层的神经元数量和激活函数
neural_net.add(layers.Dense(units=10, activation='softmax'))
# 输出网络初始化的各层的参数状况
neural_net.summary()
```

基于Keras库构建全连接神经网络

```
# 配置网络的训练方法
neural_net.compile(
    optimizer= tf.keras.optimizers.SGD(lr = 0.1), # 设置优化算法为随机梯度下降,
                                                    # 学习率为0.1
    loss='mse',                                     # 设置损失函数为均方误差
    metrics=['accuracy']                           # 设置变量accuracy用于存储分类准确率
)

# 执行模型的训练
history = neural_net.fit(train_images, train_labels, epochs=50, batch_size=512)
# 测试训练后的模型
test_loss, test_acc = neural_net.evaluate(test_images, test_labels)
print("Train Epoch:", '%02d' % (epoch + 1), "Loss=", "{:.9f}".format(loss), "
Accuracy=", acc)
```

全连接神经网络实现图片识别



```
Train Epoch: 45 Loss= 0.080445871 Accuracy= 0.8864000082015991
Train Epoch: 46 Loss= 0.080929279 Accuracy= 0.8861999750137329
Train Epoch: 47 Loss= 0.081528306 Accuracy= 0.8860000014305115
Train Epoch: 48 Loss= 0.082433820 Accuracy= 0.8855999946594239
Train Epoch: 49 Loss= 0.083543174 Accuracy= 0.8861999750137329
Train Epoch: 50 Loss= 0.085390568 Accuracy= 0.8858000278472901
acc: 0.87049997
```

全连接神经网络的自定义编码实现

□ 定义主函数

```
# 在主函数中调用模型的训练和测试函数
if __name__ == '__main__':
    # 训练模型
    model = train(mnist.train.images, mnist.train.labels, mnist.validation.images,
                  mnist.validation.labels)
    # 测试模型并得到最优的模型
    accuracy = test(model, mnist.test.images, mnist.test.labels)
    print('test the best model, accuracy=%.2f' % (accuracy)) # 输出准确率
```

全连接神经网络的自定义编码实现

- 导入所用框架和库
- 获取并处理训练集、验证集和测试集
- 定义全连接神经网络类

```
class FullConnectionLayer():  
    def __init__(self): # 用于初始化类实例  
        self.mem = {} # 定义在实例化对象中，以便类全局使用  
  
    def forward(self, X, W): # 前向传播，X为输入数据，W为网络的连接权重矩阵  
        self.mem["X"] = X # 接收输入数据  
        self.mem["W"] = W # 接收网络参数  
        H = np.matmul(X, W) # 计算网络输出  
        return H  
  
    def backward(self, grad_H): # 反向传播，grad_H为损失函数关于H的梯度  
        X = self.mem["X"] # 输入数据  
        W = self.mem["W"]  
        grad_X = np.matmul(grad_H, W.T) # 损失函数关于X的梯度  
        grad_W = np.matmul(X.T, grad_H) # 损失函数关于W的梯度  
        return grad_X, grad_W
```

全连接神经网络的自定义编码实现

□ 定义激活函数和损失函数

- ✓ 定义ReLU类用于隐含层（非线性）
- ✓ 定义交叉熵损失函数类用于训练
- ✓ 定义Softmax类用于输出分类结果

□ 构建全连接神经网络模型

```
class FullConnectionModel(): # 全连接神经网络模型类
    def __init__(self, latent_dims): # 构造函数
        self.W1 = np.random.normal(loc=0, scale=1, size=[28*28+1, latent_dims])/np.sqrt((28*28+1)/2) # 初始化W1
        self.W2 = np.random.normal(loc=0, scale=1, size=[latent_dims, 10])/np.sqrt(latent_dims/2) # He 初始化W2
        self.mul_h1 = FullConnectionLayer() # 隐含层
        self.relu = Relu() # 激活函数
        self.mul_h2 = FullConnectionLayer() # 输出层
        self.softmax = Softmax() # 激活函数
        self.cross_en = CrossEntropy() # 损失函数
```

全连接神经网络的自定义编码实现

```
def forward(self, X, labels): # 前向传播
    bias = np.ones(shape=[X.shape[0], 1]) # 生成权值均为1的矩阵作为偏置
    X = np.concatenate([X, bias], axis=1) # 合并矩阵
    self.h1 = self.mul_h1.forward(X, self.W1) # 隐含层的计算
    self.h1_relu = self.relu.forward(self.h1) # 经过隐含层激活函数处理
    self.h2 = self.mul_h2.forward(self.h1_relu, self.W2) # 输出层的计算
    self.h2_soft = self.softmax.forward(self.h2) # 经过输出层激活函数处理
    self.loss = self.cross_en.forward(self.h2_soft, labels) # 计算交叉熵损失函数

def backward(self, labels): # 反向传播
    self.loss_grad = self.cross_en.backward(labels) # 计算损失函数在输出层的梯度
    self.h2_soft_grad = self.softmax.backward(self.loss_grad) # 对softmax求导
    self.h2_grad, self.W2_grad = self.mul_h2.backward(self.h2_soft_grad) # 求导
    self.h1_relu_grad = self.relu.backward(self.h2_grad) # 对relu求导
    self.h1_grad, self.W1_grad = self.mul_h1.backward(self.h1_relu_grad) # 计算隐含层的梯度
```

全连接神经网络的自定义编码实现

□ 定义训练函数

- ✓ 定义计算准确率的函数
- ✓ 定义单步训练函数
- ✓ 定义批量训练函数

□ 定义测试函数

```
def test(model, x, y): # 测试函数
    model.forward(x, y) # 运行模型
    accuracy = computeAccuracy(model.h2_soft, y) # 获取准确率
    return accuracy
```

全连接神经网络的自定义编码实现

□ 程序运行结果:

```
Start searching the best parameter...

Parameter latent_dims=100, epoch=20 , loss=0.61235563, accuracy=0.798364: 100%|██████████| 20/20 [00:22<00:00,  1.15s/it]
Parameter latent_dims=100, validation_loss=0.5979167190591659, validation_accuracy=0.797.

Parameter latent_dims=200, epoch=20 , loss=0.5519253 , accuracy=0.851945: 100%|██████████| 20/20 [00:25<00:00,  1.28s/it]
Parameter latent_dims=200, validation_loss=0.5074925231099131, validation_accuracy=0.853.

Parameter latent_dims=300, epoch=20 , loss=0.7119555 , accuracy=0.773982: 100%|██████████| 20/20 [00:31<00:00,  1.59s/it]
Parameter latent_dims=300, validation_loss=0.5305907522703492, validation_accuracy=0.842.

The best parameter is 200.

Start training the best model...
Training the best model, epoch=50 , loss=0.31931881, accuracy=0.91      : 100%|██████████| 50/50 [01:07<00:00,  1.35s/it]
Evaluate the best model, test loss=0.30098108, accuracy=0.916000.
```

回顾整个过程

1. 问题建模

- 回归、分类、聚类 ...

2. 收集数据

- 回归：特征数据X，连续数据y
- 分类：特征数据X，离散数据y

3. 特征预处理

- 归一化
- 类别特征
- 时间特征
- 图像数据、序列数据、图结构数据

4. 构建模型

模型选择：线性回归、Logistic Regression
KNN、K-Means、神经网络

损失函数：均方误差，交叉熵

5. 模型验证&参数调优

使用训练数据训练模型，使用验证数据进行参数调优

验证指标：回归 (wmape、R2、均方误差)

6. 模型上线/AB测试

在测试数据上进行模型测试（测试数据和训练数据来自于同一分布）