# 1.1 深度学习与机器学习的区别

## 1.1.1.1 特征提取方面

- 机器学习：
  - **要靠手动完成的，而且需要大量领域专业知识**
- 深度学习：**不需要人工设计特征提取环节**
- 数据量

第一、它们需要大量的训练数据集

第二、是训练深度神经网络需要大量的算力

## 1.1.2 算法代表

- 机器学习
  - 朴素贝叶斯、决策树等
- 深度学习
  - 神经网络

图像识别

自然语言处理技术

# 1.2 深度学习框架介绍

## 1.2.2 TensorFlow的特点

## 1.2.3 TensorFlow的安装

- ubuntu安装

```
pip install tensorflow==1.12 -i https://mirrors.aliyun.com/pypi/simple
```

- MacOS安装

```
pip install tensorflow==1.12 -i https://mirrors.aliyun.com/pypi/simple
```

**tf.keras** 构建、训练和验证您的模型

tf相关API用于损失计算修改,tensorflow提供模型训练模型部署

tensorflow + tf.keras , pytorch

# 2.1 TF数据流图

## 2.1.1.2 TensorFlow结构分析

**一个构建图阶段和一个执行图阶段。**

- 构件图：数据与操作的执行步骤被描述成一个图
- 执行图：使用会话执行构建好的图中的操作
- 图和会话：
  - 图：这是 TensorFlow 将计算表示为指令之间的依赖关系的一种表示法
  - 会话：TensorFlow 跨一个或多个本地或远程设备运行数据流图的机制
- 张量：TensorFlow 中的基本数据对象
- 节点：提供图当中执行的操作

## 2.1.2 数据流图介绍

- 图(graph)与会话(session)， 节点(operation)和张量(Tensor)

# 2.2 图与TensorBoard

- 图包含了**一组tf.Operation代表的计算单元对象**和**tf.Tensor代表的计算单元之间流动的数据**。

## 2.2.2 图相关操作

- 获取tf程序的默认图：tf.get_default_graph()
- op, tensor, session：所在的都是默认的这个程序的图

## 2 创建图

- tf.graph
- 会话只运行默认那张图，如果有多个图需要多个会话开启

```
with tf.Session(graph=new_g)
```

## 2.2.3 TensorBoard:可视化学习

为了更方便 TensorFlow 程序的理解、调试与优化

- 1 数据序列化-events文件
- 2启动TensorBoard

## 2.2.4 OP

一个操作对象（Operation）是TensorFlow图中的一个节点,**接收0个或者多个输入Tensor, 并且可以输出0个或者多个Tensor**

- **打印出来的是张量值，可以理解成OP当中包含了这个值**
- "Const_1:0"
  - "<OP_NAME>" 是生成该张量的指令的名称
  - *"" 是一个整数，它表示该张量在指令的输出中的索引,都是0*
- *怎么修改op的名称：*
  - *name参数*
  - *tf.constant(42.0, name="answer")*

# *2.3 会话*

## *2.3.1 会话*

- *tf.Session：用于完整的程序当中*
  - *config：此参数允许您指定一个 tf.ConfigProto 以便控制会话的行为。例如，ConfigProto协议用于打印设备使用信息*

    ```
    /job:worker/replica:0/task:0/device:CPU:0
    ```

- ***2.3.1.2 会话的run()***
  - *placeholder:在运行时候填充数据，通过feed_dict*
  - *用处：图定义好，数据没有固定，在运行时每次填充然后计算*
- *tf.InteractiveSession：用于交互式上下文中的TensorFlow ， 例如shell*
- *eval()直接运行结果*

# *2.4 张量*

## *2.4.1 张量(Tensor)*

*TensorFlow 的张量就是一个 n 维数组， 类型为tf.Tensor*

## *2.4.2 创建张量的指令*

## *2.4.3 张量的变换*

- *类型变换：*
  - *tf.cast()*
- *形状变换：图像识别检测，*
  - *静态形状*
    - *转换静态形状的时候，1-D到1-D，2-D到2-D，不能跨阶数改变形状*
    - *对于已经固定的张量的静态形状的张量，不能再次设置静态形状*
  - *动态形状*

        - *tf.reshape()动态创建新张量时，张量的元素个数必须匹配*
- ## *2.4.4 张量的数学运算*

# *2.5 变量OP*

- ***存储持久化***
- ***可修改值***
- ***可指定被训练***
- *特点：手动初始化*
    - *init_op = tf.global_variables_initializer()*
- ## *2.5.2 使用tf.variable_scope()修改变量的命名空间*

# *2.7 案例：实现线性回归*

## *2.7.1 线性回归原理复习*

- *样本：特征值，目标值*
    - *随机生成100个点，特征只有1个*
    - *y = 0.8 * x + 0.7，目标值（100， 1）*
- *w, b x [w] + [b] = y_predict*
- *1 准备好数据集：y = 0.8x + 0.7 100个样本*
    - *x, y*
- *2 建立线性模型*
    - *随机初始化W1和b1*
    - *y = W·X + b，目标：求出权重W和偏置b*
    - *tf.Variable()：权重和偏置需要被训练*
- *3 确定损失函数（预测值与真实值之间的误差）-均方误差*
    - *tf.square(), tf.reduce_mean()*
- *4 梯度下降优化损失：需要指定学习率（超参数）*
    - *tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(loss)*
- *tf.Variable(initial_value=tf.random_normal([1, 1]), trainable=False)*
    - *迁移学习（指定某些网络结构不被训练）*

## *2.7.3 增加其他功能*

- ***增加命名空间***
    - *tf.variable_scope("original_data"):*
    - *添加损失，权重，偏置的训练观察（tensorfboard）*
    - *tf.summary.scalar("error", error)*

        - *收集零维度的值*
    - *tf.summary.histogram("weights", weights)*
        - *收集高纬度的张量值结果*
    - *summary = sess.run(merge)*
    - *filewriter.add_summary(summary, i)*
- ***命令行参数设置***
    - *tf.app.flags.DEFINE_integer("max_step", 1000, "train step number")*

      *FLAGS = tf.app.flags.FLAGS*
- *模型的保存与加载*
    - ***tf.train.Saver***
    - *tf.train.latest_checkpoint("./tmp/model/")*
    - *saver.save(sess, '/tmp/ckpt/test/myregression.ckpt')*
    - *saver.restore(sess, '/tmp/ckpt/test/myregression.ckpt')*

# *1.2 神经网络基础*

## *1.2.1 Logistic回归*

### *1.2.1.2 逻辑回归损失函数*

- *linear:均方误差*

## *1.2.3 导数*

### *1.2.3.1 导数*

*导数也可以理解成某一点处的斜率*

- *f(a) = 4a， 斜率可以理解为当一个点偏移一个不可估量的小的值，所增加的为4倍。*
- *f(a)=a^2， 导数为2a*

### *1.2.3.2 导数计算图*

### *1.2.3.3 链式法则*

### *1.2.3.4 逻辑回归的梯度下降*

***计算损失函数的某个点相对于w_1,w_2,bw1,w2,b的导数之后，就可以更新这次优化后的结果***

- *w1, w2, b: 10， 20， 5---> dw1:2, dw2: 1, db: 3*
- *w1- (0.001)*(2)*
- *w2-(0.001)*(1)*
- *b - (0.001)*(3)*
- *w1', w2',b*

*1.2.4 向量化编程 每更新一次梯度时候，在训练期间我们会拥有m个样本，那么这样每个样本提供进去都可以做一个梯度下降计算。所以我们要去做在所有样本上的计算结果、梯度等操作*

- 1000个样本都计算一次梯度，得到1000个梯度
- 求出平均梯度
- 两步合而为1，

#### 1.2.4.2 向量化实现伪代码-逻辑回归的梯度下降实现

## 1.2.5 案例：实现逻辑回归

```
def propagate(w, b, X, Y):
    """
    参数：w,b,X,Y：网络参数和数据
    Return:
    损失cost、参数W的梯度dw、参数b的梯度db
    """
    m = X.shape[1]

    # 从前往后计算损失（前向传播）
    A = basic_sigmoid(np.dot(w.T, X) + b)
    # 计算损失
    cost = -1 / m * np.sum(Y * np.log(A) + (1 - Y) * np.log(1 - A))
    # 从后往前求出梯度（反向传播）
    dZ = A - Y
    dw = 1 / m * np.dot(X, dZ.T)
    db = 1 / m * np.sum(dZ)

    grads = {"dw": dw, "db": db}
    return grads, cost
```

w1

w2

w3

w4

w5