deep learning

1. 本质上就是通过一系列函数的线性组合去模拟出最终数值变化的函数
2. 欠拟合：神经元太少，模型太简单，不能很好地拟合数据
3. 过拟合：神经元太多，模型太复杂，对训练数据过度拟合，泛化能力差
4. 所以根据问题的规模、选择合适的模型大小
5. sigmod 的缺点就是梯度消失，（因为最大的一层的梯度为 0.25，然后再往前传的时候，梯度就会越来越小，导致前面的神经元的权重更新很慢）
6. 最后的模式里面、蓝色的是权重小的、红色的是权重大的（好像这个图不太对，因为有很多层，如果把隐藏层减掉就是差不多的了）
7. 卷积几次之后、特征已经变得更复杂了（模式的组合），所以需要池化降低维度
8. 填充：为了结果不要太小
9. 通道数：会影响到特征的提取，如果通道数太少，可能会丢失一些特征，如果通道数太多，可能会增加一些噪声，但是也能提取更多特征
10. 池化：最大值池化、平均值池化
    - 最大值池化：保留最大值，减少噪声，保留特征
11. 卷积神经网络：参数更少，适合图像识别任务

NLP

1. NLP 的六个场景：
    - 信息获取
    - 文本分类
    - 文档相似度推荐
    - 机器翻译
    - 聊天机器人/问答系统
    - 文本生成
2. NLP 研究领域 ≈ 应用领域
3. 如果一个 word 用一个 one-hot vector 表示，那么这个向量的维度就是词典的大小，这样的向量维度太大，不利于计算，而且会有很多个 0，浪费空间
4. 第一个问题是怎么把一个 word 变成一个 vector（因为上面的问题、所以需要降维）
5. 第二个问题可以用一个 BOW（bag of words）来解决（这个单词间的顺序问题如何去解决？）
6. 上面就是一句话怎么把它数值化的一个方法，数值化之后，每个单词都是一个向量
7. CNN 就是可以用于 NLP 中的文本分类
8. 然后用卷积核去卷的时候、就能发现三个单词之间有什么关系、然后再池化、就能找到 5 个单词之间有什么关系、所以 CNN 是可以去做 NLP 的、只不过需要进行修改
9. 有 CNN 和 TextCNN、TextCNN 是用卷积核去卷的时候、拼接了短、中、长的三个不同范围的单词的特征、再去做训练，效果会更好（简单的 CNN 一定要卷到一个长的句子为止才终止）

RNN，LSTM

1. 可以做时序预测，文本的生成（NLP）

2. 相当于每次输入的时候都拿到了前面一段时间的信息
3. 缺点是这个程序是不能并行的(RNN，LSTM)、因为一个点算之前需要算之前的点，只能串行
4. 如果这个时间的依赖非常长期的话、那么它这个网络就会很弱、学习不到这个特征
5. 让这个时间序列变得灵活的话、就有了 LSTM（长短期记忆网络）
6. cell 状态就是它的状态，h 就是输出
7. LSTM 有三个门：遗忘门、输入门、输出门
8. 遗忘门：决定哪些信息是需要遗忘的（就是用一个输入的数据、sigmoid 之后乘以上一个时间的输出）
9. 输入门：决定哪些信息是需要输入的（就是用一个输入的数据、sigmoid 之后乘以上一个时间的输出，然后再用一个 tanh 之后乘上一个时间的输出）
10. 输出门：决定哪些信息是需要输出的（就是用一个输入的数据、sigmoid 之后乘以上一个时间的输出，然后再用一个 tanh 之后传给下一个时间），输出了 h
11. ppt 上有具体的 LSTM 的公式
12. RNN 是一种递归神经网络 每一次的输出由上一次的输出和当前的输入共同决定 即 yi=f(yi-1, xi) 也就带上了之前所有数据的特征 适合预测时间序列数据与自然语言处理
13. RNN 的缺点是无法并行 因为每一次的输出都依赖于上一次的输出 另外 RNN 固定了依赖的长度 模型不够灵活 导致长期记忆的学习比较弱 这点可以通过 LSTM 来解决
14. LSTM（Long Short-Term Memory）：长短期记忆网络 是 RNN 的一种变体 通过门控机制来控制信息的流动 具体而言 LSTM 有三个门：遗忘门、输入门、输出门 遗忘门决定了之前的状态有多少会被遗忘 输入门决定了新的状态有多少会被加入 输出门决定了这个状态有多少会被输出 虽然 LSTM 仍不能并行 但是单个 LSTM 单元内部的计算的并行度较好 比较适合长期记忆的学习

```
# 预处理数据 相当于将第i个数据的输入变为第i-look_back到i-1个数据
def create_dataset(data, look_back):
  X, y = [], []
  for i in range(len(data) - look_back):
      X.append(data[i:i + look_back])
      y.append(data[i + look_back])
  return np.array(X), np.array(y)
X, y = create_dataset(scaled_data, look_back)
# 激活函数是tanh 所以需要将数据缩放到-1到1之间
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(time_series.reshape(-1, 1))
# 构建RNN模型 神经元数为50 输入形状为look_back 输出形状为1
model = Sequential([
  SimpleRNN(50, input_shape=(look_back, 1), return_sequences=False,
activation='tanh'),
  Dense(1)  # Output layer for regression
])
# LSTM模型基本一样
model = Sequential([
  LSTM(50, input_shape=(look_back, 1), return_sequences=False),
  Dense(1)
])
```

chatGPT

1. 由于先前的计算效率不够高、所以需要 transformer 这些加快运算

2. 为什么每次问相同的问题的时候、会给出不同的答案。因为其中会有一个温度的概念、当温度不一样的时候、这个就有一定的概率去选择概率不是那么高的预测结果、这就显示出了很多的灵活性。（然后是每次选择一个单词、然后再去预测下一个单词，所以得到的结果不是千篇一律的而且会有差异）（所以如果温度为 0 的话，那么每次内容都会一样）
3. 这个 n-gram 就是对 n 个字母进行预测、n 越高的时候它越像一个单词了
4. 然后这个从字母推广到词的时候、就能产生类似的效果。
5. 但是词的组合会有很多个、所以会有很多的组合、所以这个模型的参数会很大
6. 所以需要去涉及到做降维、就是做 embeddings（相当于把一个 one-hot 编码变成一个词的 vector）
7. 然后再经过 transformer（和 attention 机制）、会得到后续结果

Transformer

1. TextCNN、卷积本身是在找相同的特征（用相同的模式）（有小范围、中范围和大范围的，最后把这个特征给合并了，准确率比单一的好）（但是也有缺点、如果三个词的顺序发生了变化、卷积神经网络也会找不到这个特征）（而且前面的词都会对后面的词产生影响、这个初始模型太简单了）
2. 所以有了 RNN。RNN 的问题是很难学到长时间的依赖
3. 所以有了 LSTM、会有记忆和遗忘的机制（但是还是有缺点、计算的时候必须拿到上一个的结果、就是计算效率低，只能串行）
4. 所以有了 transformer、分为编码器和解码器。编码器：找到特征图，解码器，翻译这个编码内容然后输出内容结果
5. multi head attention、就是把 attention 的结果拆成多个头、然后再合并起来（因为自然语言处理的时候、一个词可能有多个意思、所以要多个头，多学一点出来）
6. 掩码：一开始进去的时候只有句子的开始。
7. Attention：有一个用来算自注意力的公式；每一个编码器里有一个前馈网络层和多头注意力层；这个前馈层是非线性的层（线性代数里的非线性变换，因为线性变换没有意义）
8. Attention(Q, K, V) = softmax(QK^T / sqrt(d_k))V
9. self-attention：就是这个 it 为什么指的是它这个句子里的前面的狗？有三个矩阵、查询矩阵 Q、键矩阵 K 和值矩阵 V。然后这个权重 W 是算出来的。最后的任何一个位置上的值、就是两个单词的相关性的值。（而且还不是一个对称矩阵、因为语言的顺序会有不一样的意思）所以 Q*KT 就是算的是这个东西，然后要让里面的所有的值都除以下面这个 dk、防止学习的时候的梯度消失问题、然后放到 softmax 里、做一个归一化、最后再乘一个 V 矩阵。最后的 Z 就是注意力矩阵。
10. 多头就是把多个上面的结果合并起来。
11. 重要的是再这个 Self-attention 的 Z1、乘相关度之后、得到了一个单词在一个句子里的特征是怎么样的
12. positional encoding：位置编码(ppt 里是一个矩阵)。就是每个单词、可能单词都是一样的、但是位置不一样、所以要不同。原始嵌入值输入加上位置编码之后、再做训练。
13. 解码器就是反向得出这句话、所以会有一个 mask、只能看到前面的内容。然后就是把这个编码器的输出和解码器的输出做 attention、然后再做一个前馈网络层、最后输出。（就是一个负无穷大）
14. 这个 transformer 可以并行计算（因为是矩阵计算），解码器可以不是并行计算。
15. NLP、CV 都可以用 transformer。