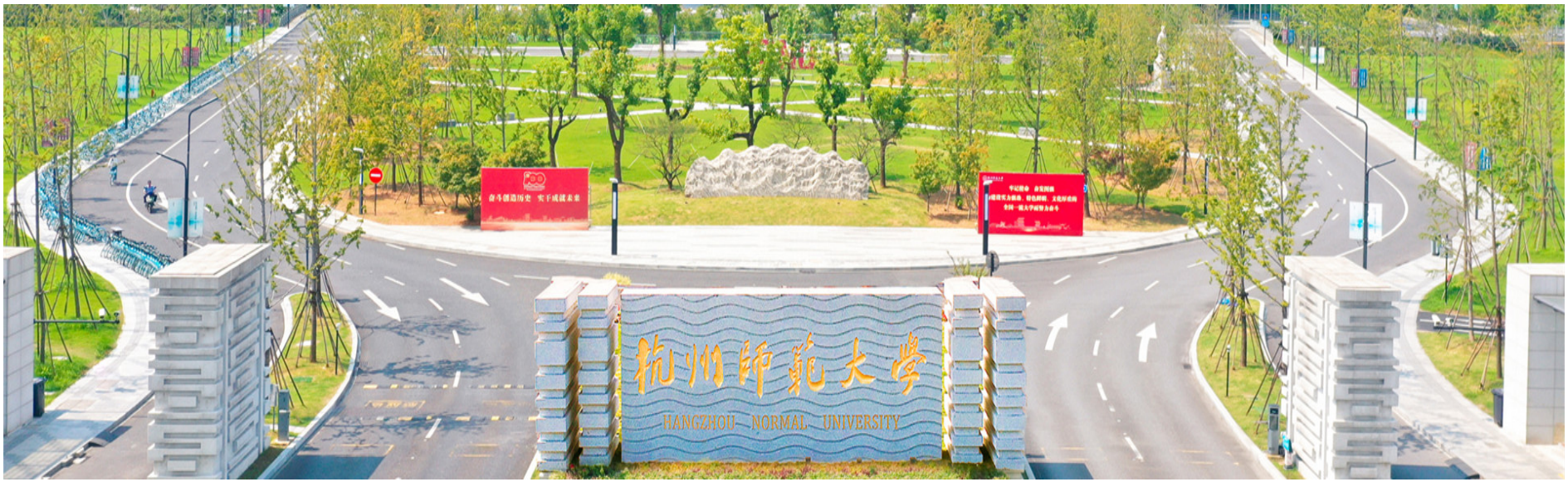
第七讲 - 基于神经网络的文本挖掘方法

张建章

阿里巴巴商学院
杭州师范大学

2024-03-01



过去的 10 多年间，计算机**算力**、互联网**数据**和深度学习**算法**的快速发展，推动自然语言处理、计算图形学等人工智能领域取得了一系列突破性进展。

图 1: 深度学习三巨头

2018 年，ACM（国际计算机学会）决定将计算机领域的最高奖项图灵奖颁给 Yoshua Bengio、Yann LeCun 和 Geoffrey Hinton，以表彰他们在计算机深度学习领域的贡献。

DNN4NLP 代表性论文

[1] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space.” arXiv preprint arXiv:1301.3781, 2013. [pdf]

[2] Yoon Kim. “Convolutional Neural Networks for Sentence Classification.” In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1746−1751, 2014. [pdf]

[3] Huang, Zhiheng, Wei Xu, and Kai Yu. “Bidirectional LSTM-CRF models for sequence tagging.” arXiv preprint arXiv:1508.01991, 2015. [pdf]

[4] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate.” ICLR, 2015. [pdf]

[5] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need.” Advances in neural information processing systems, 2017. [pdf]

DNN4NLP 学习方法

1. 读课本，看论文，学理论

① 《神经网络与深度学习》，邱锡鹏，2020，机械工业出版社，[pdf]；

② 《大语言模型》，赵鑫等，2024，eprint，[pdf

③ 《Natural Language Processing with Transformers》，Lewis Tunstall 等，2022，O'Reilly，[中译本]

④ ACL，EMNLP，ICML 等 CCF 推荐的人工智能国际会议论文；

⑤ 温习回顾高等数学、线性代数、概率论与数理统计等课程；

2. 学编程，看源码，敲代码

① 熟练掌握 Python 编程语言；

② 运行、学习论文作者提供的源代码；

③ 使用 Jupyter Notebook，PyCharm，Linux 编写代码；

前馈神经网络 (feedforward Neural Networks) 是最基础的神经网络结构，其典型代表为多层感知机 (Multilayer Perceptron, MLP)。

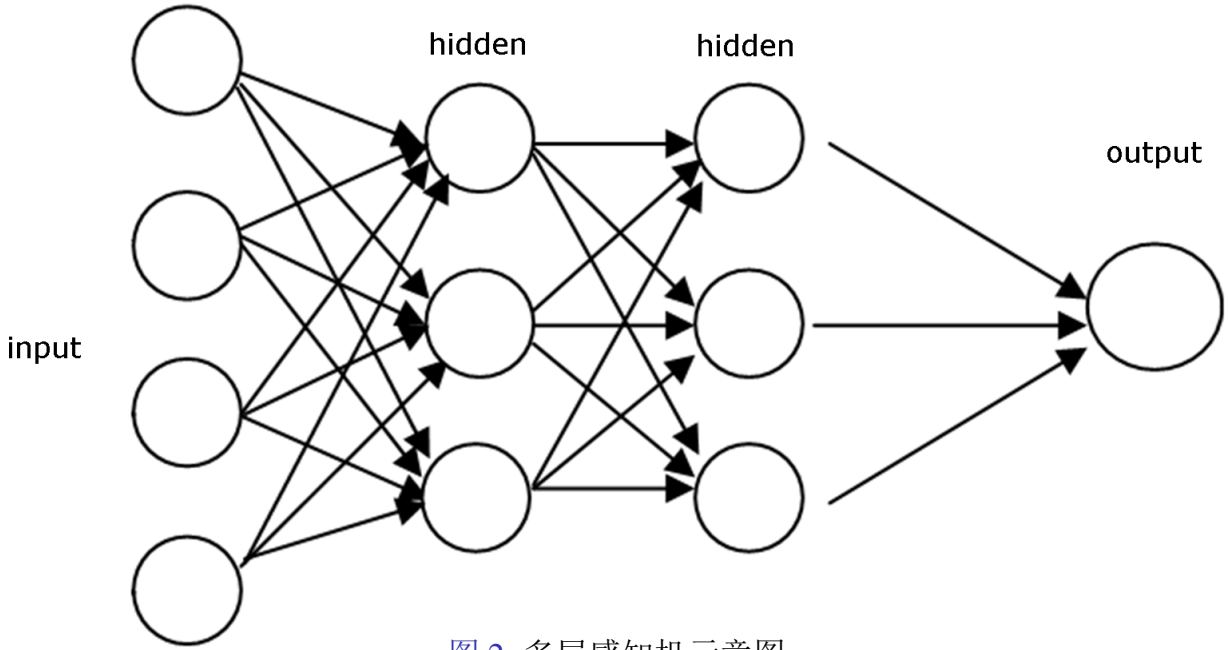


图 2: 多层感知机示意图

MLP 包含一个输入层、一个或多个隐藏层和一个输出层。除输入层外，每一层的每个节点 (神经元) 都与前一层的所有节点连接，每个连接都有相应的权重，每个节点的输出值使用非线性激活函数 (sigmoid，ReLU 函数等) 计算得到，每个激活函数的输入值为上一层节点的线性组合。

工作原理： 前向传播计算预测和反向传播求参数。

多层感知机前向传播计算预测过程的数学表示如下：

- **输入层：** n 维向量 $\boldsymbol{x} = [x_1, x_2, ..., x_n]^T$；
- **隐藏层 (共 m 层)：** 令第 l 层的权重矩阵为 $\boldsymbol{W}^{(l)}$ 和偏置向量 $\boldsymbol{b}^{(l)}$。

输入为:

$$\boldsymbol{Z}_l = \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}$$

输出为:

$$\boldsymbol{h}^{(l)} = \sigma(\boldsymbol{Z}_l)$$

其中，$l \in [1, m]$，$\boldsymbol{h}^{(0)} = \boldsymbol{x}$，$\sigma$ 是激活函数，如 ReLU 或 sigmoid；

- **输出层：**

$$\boldsymbol{y} = f(Z_o) = f(\boldsymbol{w}^{(o)}\boldsymbol{h}^{(m)} + b^{(o)})$$

其中，f 是输出层的激活函数，如 sigmoid 用于二分类，或 softmax 用于多分类。

Sigmoid 函数

sigmoid 函数是指具有 S 形曲线的函数，神经网络中通常使用逻辑函数：

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

$$S'(x) = \frac{d}{dx}S(x) = S(x)(1-S(x))$$

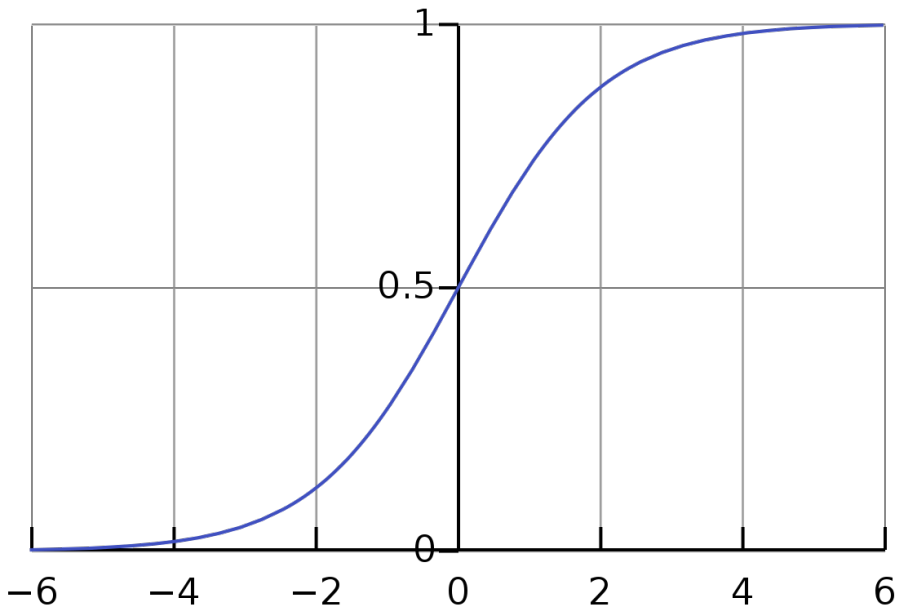


图 3: 逻辑 (logistic) 函数图像

以图2为例，输入为文档 d 的 TF-IDF 向量 $\boldsymbol{x}$，激活函数为 sigmoid 函数：

$$\boldsymbol{x} = \begin{bmatrix} 0.1 \\ 0.3 \\ 0.4 \\ 0.2 \end{bmatrix}$$

令，第一个隐藏层的权重矩阵和偏置向量为：

$$\boldsymbol{W}_1 = \begin{bmatrix} 0.2 & 0.8 & 0.6 & 0.4 \\ 0.6 & 0.3 & 0.5 & 0.7 \\ 0.5 & 0.6 & 0.7 & 0.8 \end{bmatrix} \quad \boldsymbol{b}_1 = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix}$$

则，第一个隐藏层的输入为：

$$\boldsymbol{Z}_1 = \boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1 = \begin{bmatrix} 0.56 & 0.66 & 0.89 \end{bmatrix}^T$$

输出为：$\boldsymbol{H}_1 = \text{sigmoid}(\boldsymbol{Z}_1) = \begin{bmatrix} 0.636 & 0.659 & 0.709 \end{bmatrix}^T$

令，第二个隐藏层的权重矩阵和偏置向量为：

$$\mathbf{W}_2 = \begin{bmatrix} 0.4 & 0.6 & 0.8 \\ 0.2 & 0.5 & 0.7 \\ 0.6 & 0.3 & 0.4 \end{bmatrix} \quad \boldsymbol{b}_2 = \begin{bmatrix} 0.3 \\ 0.4 \\ 0.5 \end{bmatrix}$$

则，第二个隐藏层的输入为：

$$\boldsymbol{Z}_2 = \boldsymbol{W}_2\boldsymbol{H}_1 + \boldsymbol{b}_2 = \begin{bmatrix} 0.972 & 0.834 & 0.781 \end{bmatrix}^T$$

输出为：

$$\boldsymbol{H}_2 = \text{sigmoid}(\boldsymbol{Z}_2) = \begin{bmatrix} 0.726 & 0.697 & 0.686 \end{bmatrix}^T$$

令，输出层的权重矩阵和偏置向量为：

$$\mathbf{W}_o = \begin{bmatrix} 0.5 & 0.7 & 0.2 \end{bmatrix} \boldsymbol{b}_o = 0.6$$

则，输出层的输入为：

$$Z_o = \boldsymbol{W}_o \boldsymbol{H}_2 + b_o = 1.118$$

输出为：

$$y = \text{sigmoid}(Z_o) = 0.753$$

表示文档 d 属于正类的概率，根据阈值 (如，0.5) 进行决策，可将文档分类为正类。

其他激活函数，如，ReLU，softmax 见本讲后续内容。

令神经网络的损失函数 (成本) 函数为 L，则使用标注数据集 $(\boldsymbol{X}, \boldsymbol{y}) = (\boldsymbol{x}_1, y_1, ..., \boldsymbol{x}_m, y_m)$ 训练神经网络的目标即为求解参数 W 和 b，以最小化 L，L 通常定义为预测值 $\boldsymbol{y'}$ 与真实值 $\boldsymbol{y}$ 的偏离程度：

$$\min_{\boldsymbol{W},\boldsymbol{b}} L(\boldsymbol{y}, \boldsymbol{y'})$$

通常使用梯度下降法，迭代更新参数 $\boldsymbol{W}$ 和 $\boldsymbol{b}$。

反向传播：就是用链式法则以网络每层的权重为变量计算损失函数的梯度，以更新参数来最小化损失函数。

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{y'}} \frac{\partial \boldsymbol{y'}}{\partial \boldsymbol{W}}$$

在每一层的输入列向量的第 0 维添加一个元素 1，偏置向量 $\boldsymbol{b}$ 添加到参数矩阵 $\boldsymbol{W}$ 第 0 列，即可用一个统一的矩阵表示参数矩阵。

梯度下降

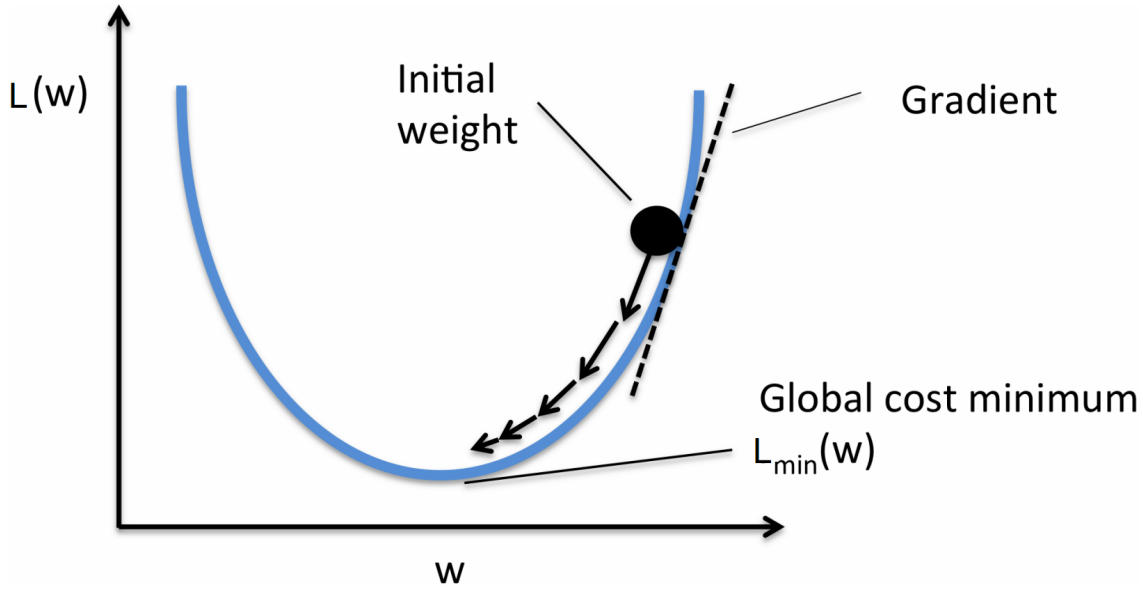


图 4: 梯度下降示意图 (一元函数)

权重参数 $\boldsymbol{W}$ 随机初始化，在神经网络计算一遍数据集后更新一次参数 (weights update for each epoch)。梯度下降参数更新公式如下：

$$\boldsymbol{W} := \boldsymbol{W} + \Delta \boldsymbol{W} = \boldsymbol{W} - \eta \frac{\partial L}{\partial \boldsymbol{W}}$$

其中，η 表示学习率，又称步长，$-\frac{\partial L}{\partial \boldsymbol{W}}$ 表示负梯度方向。

本质上，可以将梯度下降优化想象成一个徒步旅行者 (权重参数) 想要从山上 (损失函数) 爬进山谷 (损失最小)，每一步都由坡度 (梯度) 的陡度和步长 (学习率) 确定，即朝着最陡的方向迈一步。

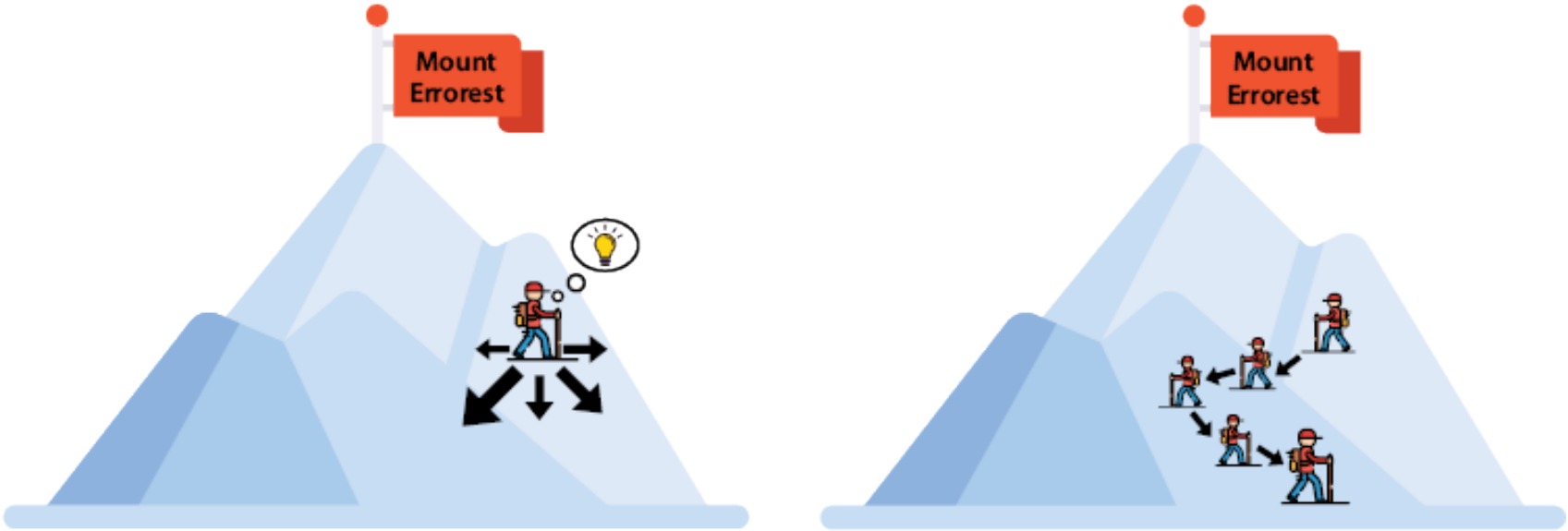


图 5: 梯度下降原理示意图 (下山)

梯度下降基于全部数据集计算梯度更新参数，即，先将每个样本求得的梯度进行累积，然后更新参数，数据集越大，参数更新越慢，参数收敛越慢。**随机梯度下降**基于每个训练样本更新一次参数，虽然参数更新快，但更新过程损失函数值会存在震荡。**小批量梯度下降**基于一组训练样本更新一次参数，迭代次数少，并且可以借助向量运算提高计算性能。

梯度下降计算实例

以图2为例，应用链式法则求输出层参数梯度 $\frac{\partial L}{\partial \boldsymbol{w}}$，使用交叉熵损失函数：

$$L = -[y\log(y') + (1-y)\log(1-y')]$$

激活函数使用 sigmoid 函数。

输出层的输入值为：

$$z = w_0 + h_1 * w_1 + h_2 * w_2 + h_3 * w_3$$

输出层的输出值为：

$$y' = \frac{1}{1+e^{-z}}$$

梯度为：

$$\frac{\partial L}{\partial \boldsymbol{w}} = \frac{\partial L}{\partial y'}\frac{\partial y'}{\partial z}\frac{\partial z}{\partial w}$$

第一项，交叉熵求导：

$$\frac{\partial L}{\partial y'} = \frac{y' - y}{y'(1 - y')}$$

第二项，sigmoid 函数求导：

$$\frac{\partial y'}{\partial z} = y'(1 - y')$$

第三项，线性函数求导：

$$\frac{\partial z}{\partial w} = h$$

因此可得输出层参数的梯度 (导数) 为：

$$(1 - y')h$$

类似地，应用链式求导法则，可计算其它隐藏层参数的梯度。

梯度下降更新参数：

$$w := w - \eta \frac{1}{m} \sum_{y'} (1 - y')h$$

其中，m 表示训练集的样本量。

随机梯度下降更新参数：

$$w := w - \eta (1 - y')h$$

小批量梯度下降更新参数：

$$w := w - \eta \frac{1}{k} \sum_{y'} (1 - y')h$$

其中，k 表示小批量的样本量。

1. **初始化**：随机初始化网络的权重和偏置，可通过多种方式，如，使用高斯分布的随机值或一些特定的初始化策略 (如 Xavier 初始化或 He 初始化)；
2. **前向传播**：给定一个 (批) 输入，通过网络进行前向传播，计算每个神经元的输出，以及输出层的输出；
3. **计算损失**：计算输出层的预测值和真实值之间的损失，常见的损失函数有均方误差 (回归) 和交叉熵损失 (分类)；
4. **反向传播**：从输出层开始，反向遍历网络，计算损失函数相对于每个权重和偏置的梯度；
5. **参数更新**：对于每个参数，将其当前值减去学习率与梯度的乘积。学习率是超参数，决定了参数更新的步长；
6. **迭代执行**：以上过程迭代执行，直到达到设定的迭代次数，或者模型的损失函数值下降到一定的范围，或者当模型的性能在验证集上不再提高时，停止训练，保存模型。

以上为基本框架，对于实操 (俗称炼丹)，还需要考虑其他因素，如，加入正则项防止过拟合、使用动量等加速收敛、动态调整学习率等。

基本原理

卷积神经网络 (CNN) 适合处理具有网格结构的数据，如图片，亦可用于处理 (如，分类) 词嵌入向量表示的文本。

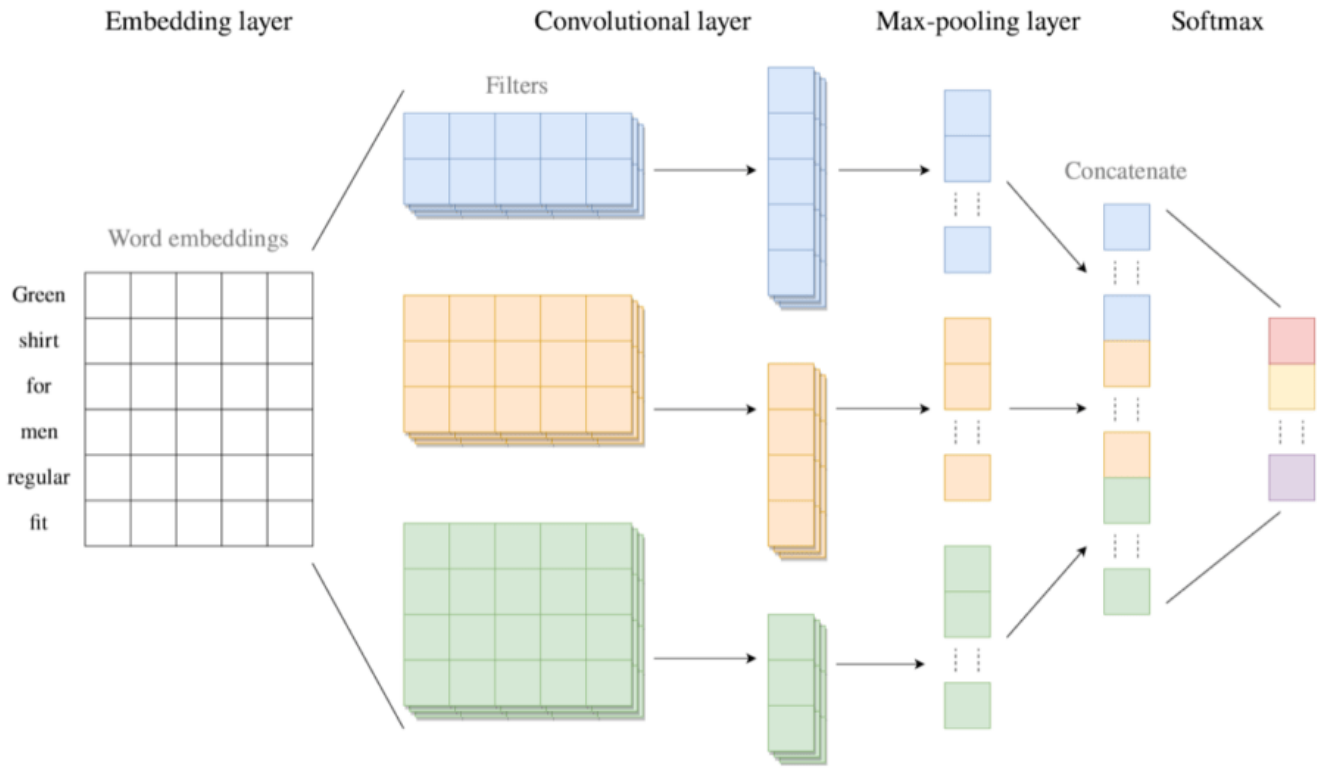


图 6: convolutional neural network (CNN) 文本分类示意图

CNN 的基本结构：输入层 - 卷积层 - 激活函数 - 池化层 - 全连接层 - 输出层。在实际应用中，卷积层和池化层可以有多个。

卷积

使用滤波器 (filter) 进行卷积操作 (同 MLP 中的前向传播过程 $\boldsymbol{W}\boldsymbol{h}+\boldsymbol{b}$)，如下图所示：

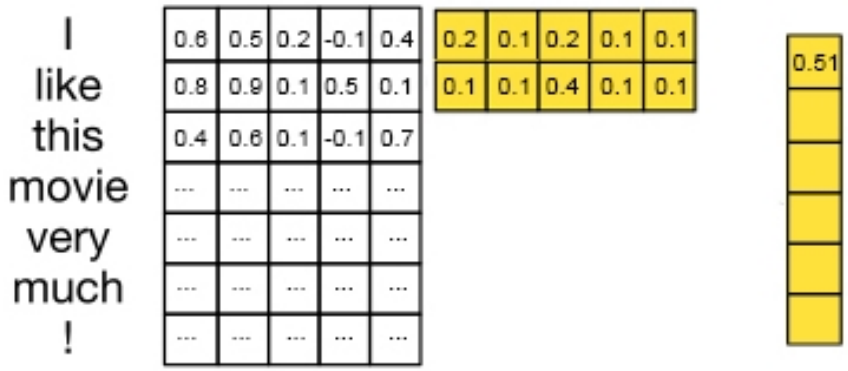


图 7: 文本向量卷积操作示意图

上图中，滤波器的尺寸 (filter size) 为 $(2,5)$，步长 (stride) 为 1，相当于取 2-gram，即，$\{I\ like, like\ this, this\ movie, ...\}$。(点我看卷积操作动图)。

① 对词向量和卷积核进行逐元素乘积-求和操作，例如：

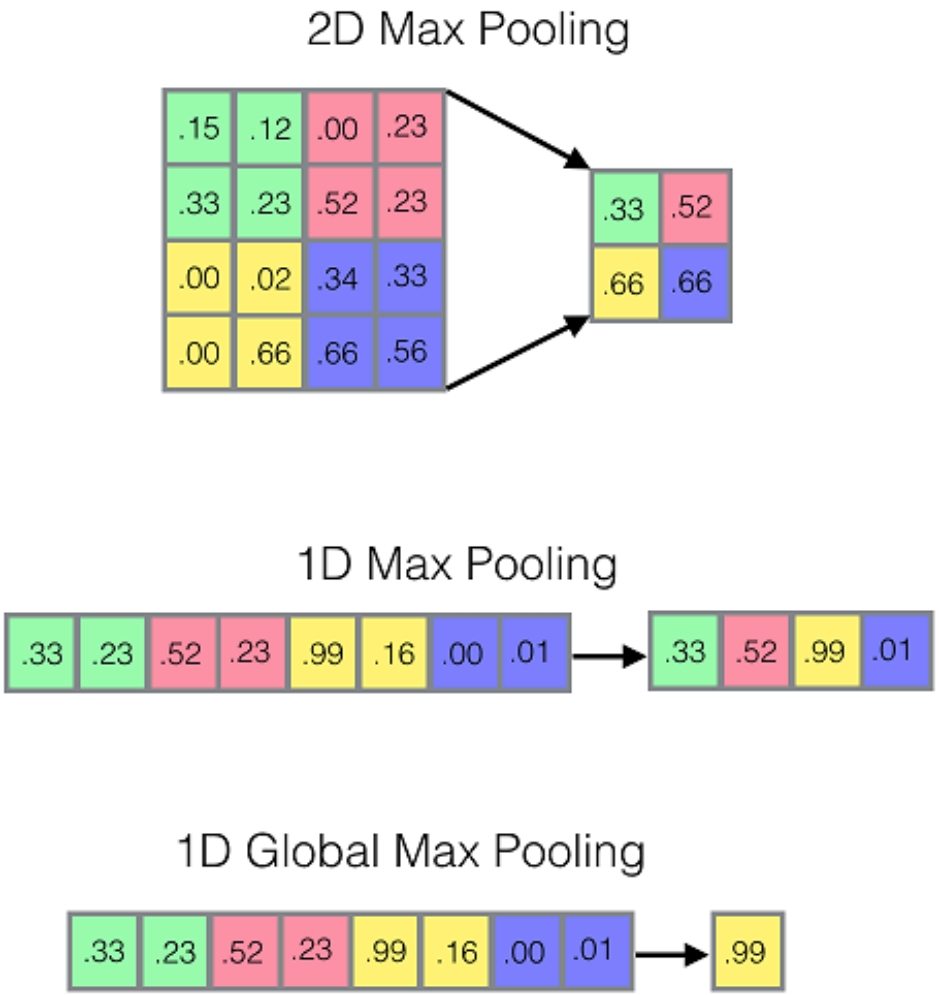
$$0.6 \times 0.2 + 0.5 \times 0.1 + ... + 0.1 \times 0.1 = 0.51$$

② 对卷积之后的结果添加偏置项后，应用激活函数 (如，ReLU)。

输出结果称为特征映射 (feature map)。

池化

池化 (pooling) 通过平均或取最大值，对特征映射中的信息进行聚合，如下图 (最大池化) 所示：



池化

池化操作中也有滤波器尺寸和步长的概念：

① 第一幅图中，filter size 为 $(2,2)$，stride 为 $(2,2)$;

② 第二幅图中，filter size 为 $(1,2)$，stride 为 2;

③ 第三幅图中，取向量的最大值;

相应地，图6中的池化操作，就是取每个特征映射向量中的最大值，然后，将取出的最大值拼接 (concatenate, 即首尾相连) 成一个向量。

最后，将池化的结果通过全连接层进行线性变换，转化为输出层接收的尺寸，输出层计算输出结果。输出层为 softmax 函数时，接收多维向量 (维度同类别数) 做多分类。

点我看卷积-池化操作动图

softmax 函数

softmax 函数的定义如下：

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K.$$

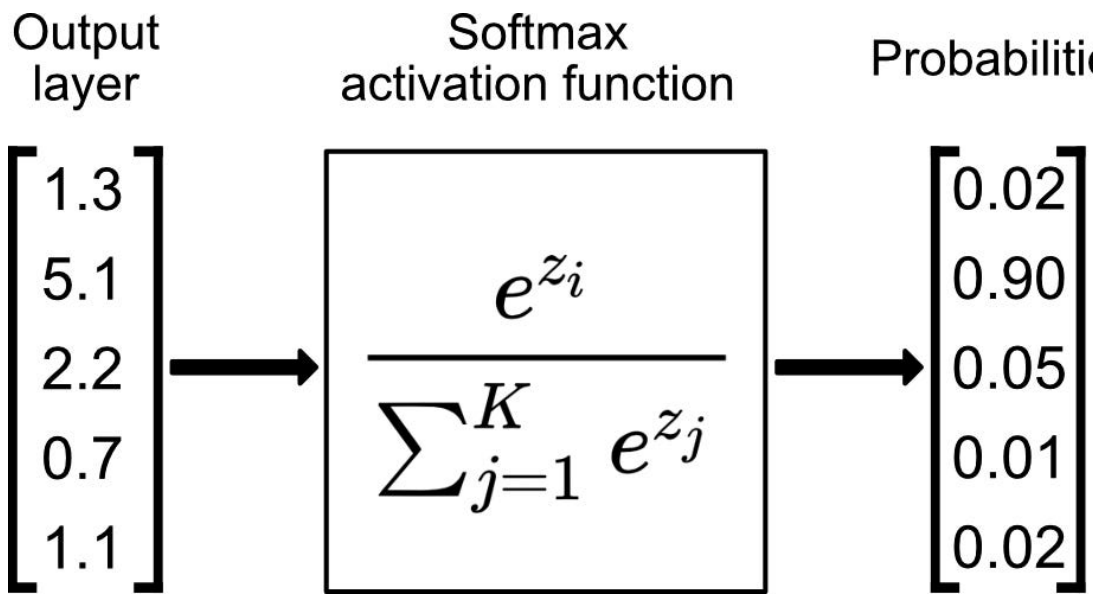


图 8: softmax 函数计算示例

在多分类任务中，K 表示类别数，$\sigma(\mathbf{z})_i$ 表示样本属于类别 i 的概率。

ReLU 函数

ReLU 函数的定义和图像如下图：

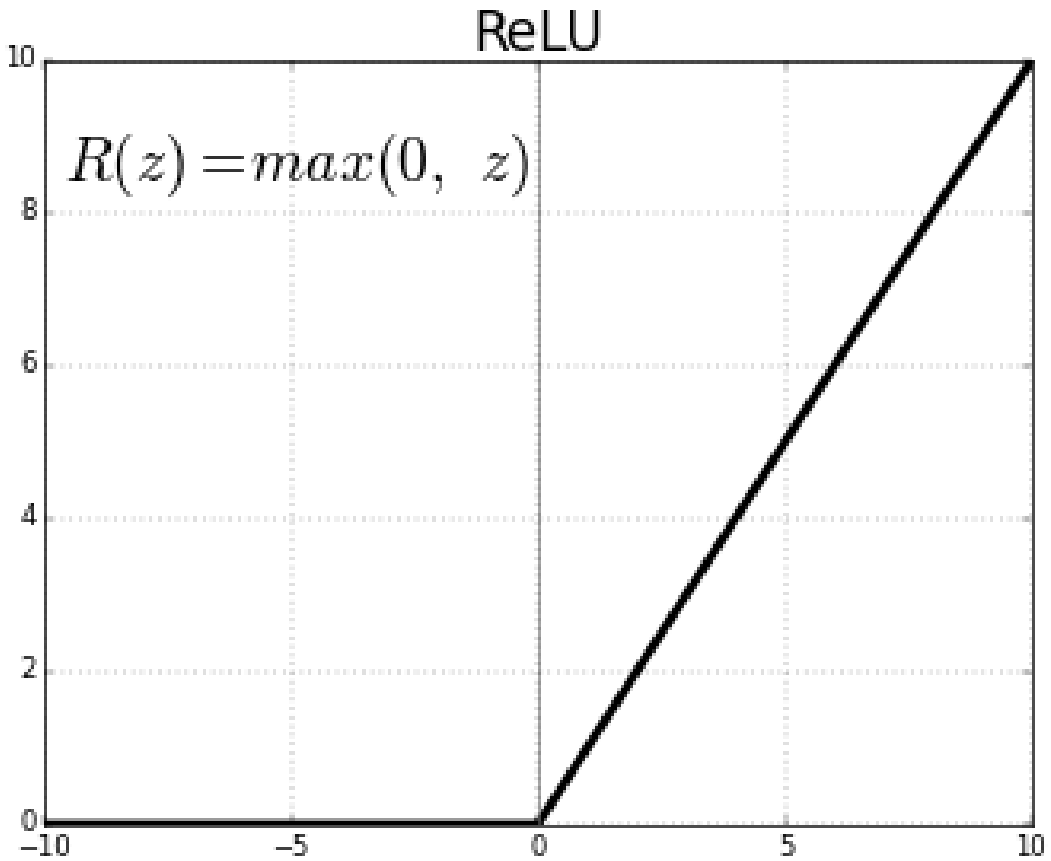


图 9: ReLU 函数图像

RelU 函数的结果只保留非负值。

循环神经网络 (RNN) 用于处理序列数据类型，如语音识别、自然语言处理等、时间序列分析等，非常适合于处理变长序列的任务。

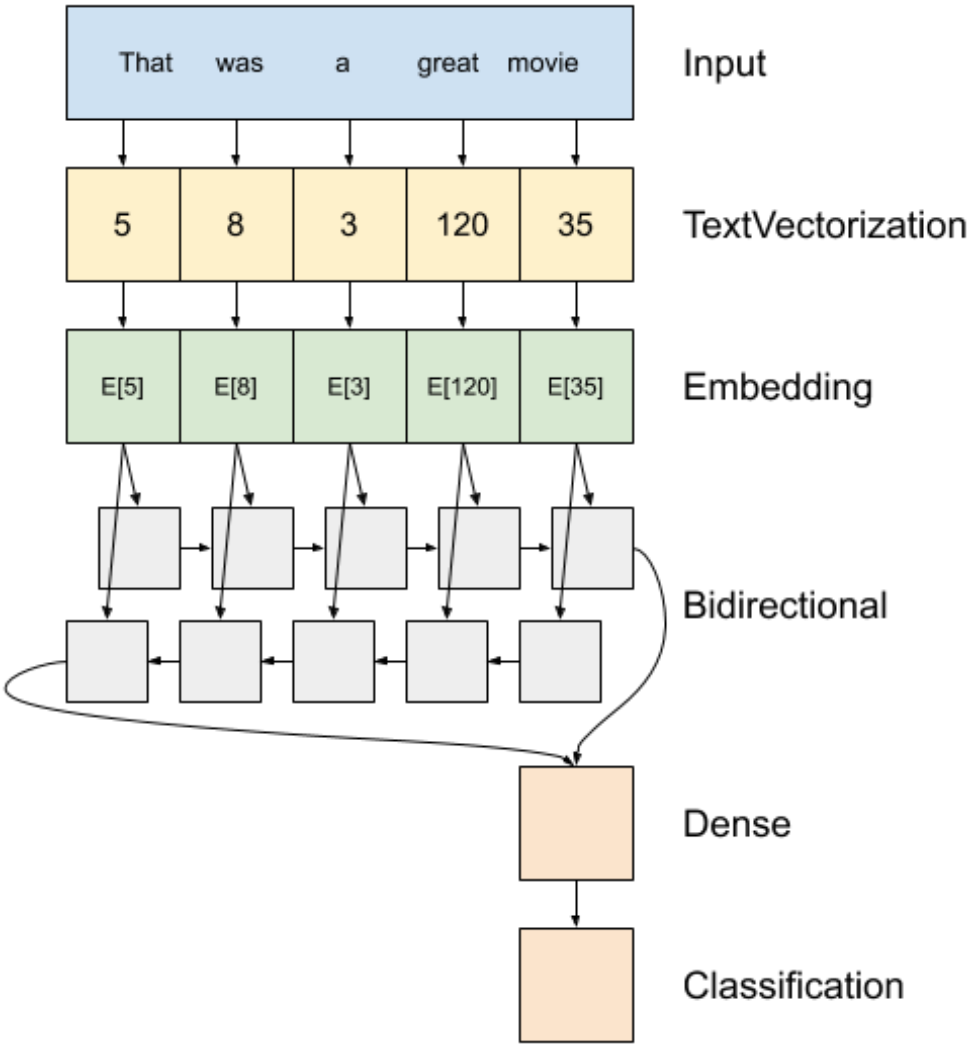


图 10: Bidirectional Recurrent Neural Networks 文本分类示意图

循环单元

RNN 的基本组成模块为循环单元 (recurrent cell)，结构如下图：

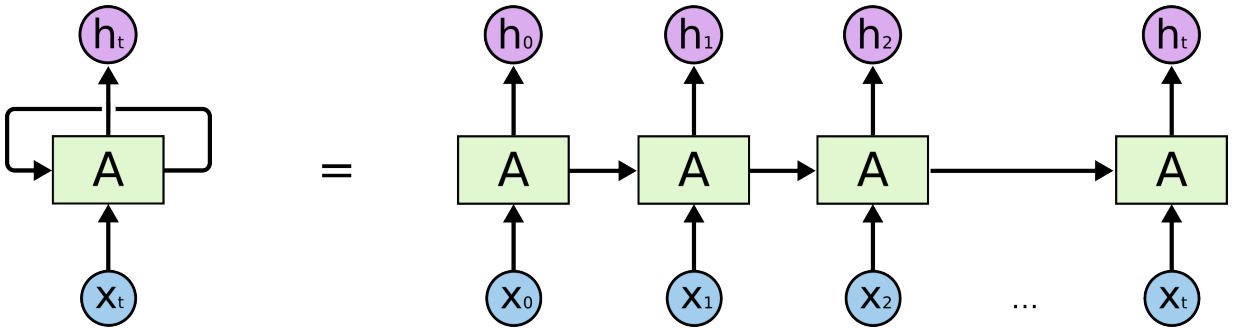


图 11: RNN 循环单元结构示意图 (点我看 RNN 动图)

$\boldsymbol{x}_t$ 为时刻 t 的输入值，$\boldsymbol{h}_t$ 是该时刻输出值，A 表示一个 RNN 单元：

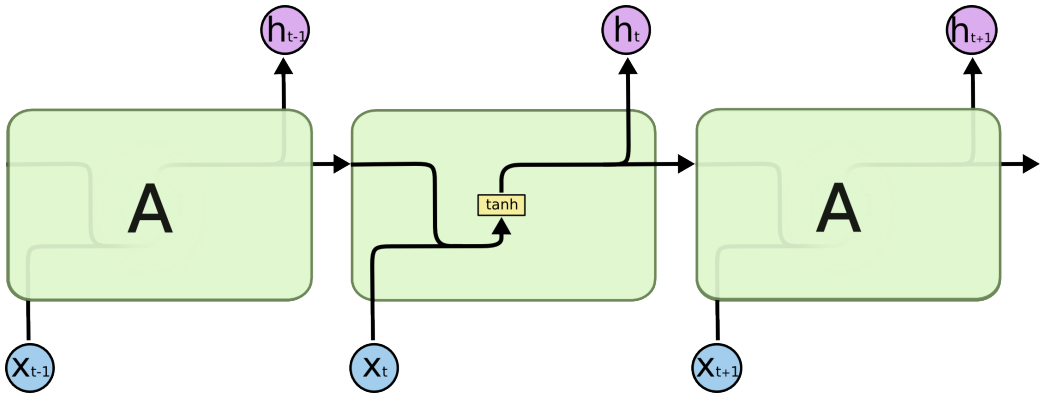


图 12: RNN 循环单元计算示意图

tanh 函数

双曲正切函数 (hyperbolic tangent, tanh) 的定义和图像如下图：

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

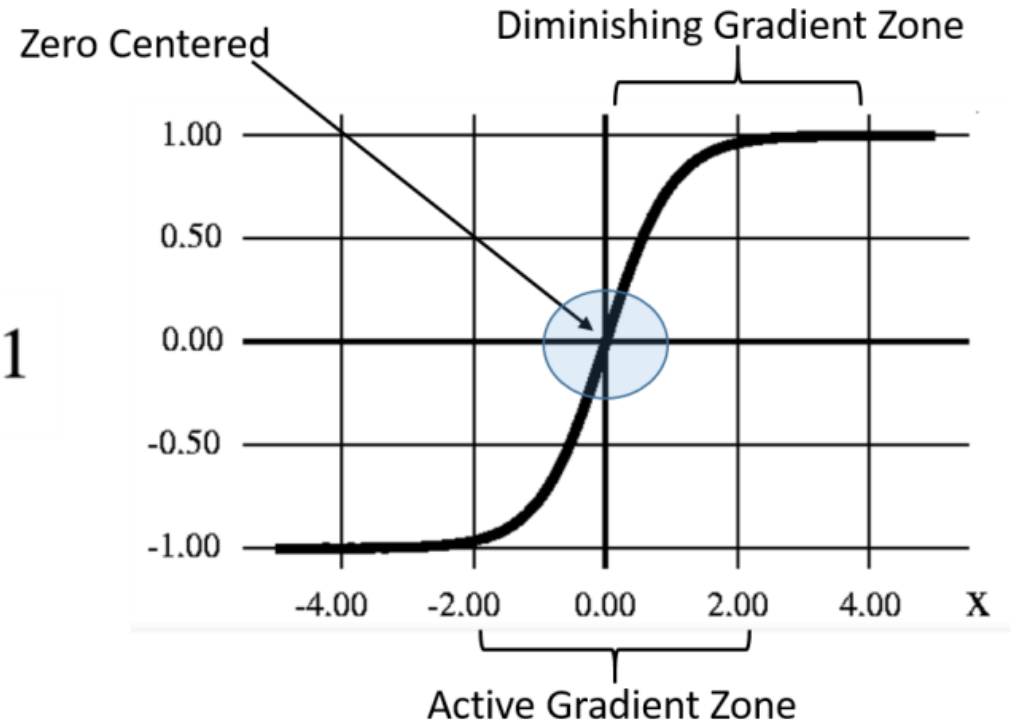


图 13: tanh 函数定义和图像

双曲正切函数可视为 sigmoid 函数 (logistic 函数) 的变体，以 0 为中心 (logistic 函数平移、伸缩后)。

LSTM

基本的 RNN 模型存在梯度消失和梯度爆炸问题，在实际应用中的效果并不理想。一些改进的 RNN 模型，如长短期记忆网络（Long Short-Term Memory, LSTM）和门控循环单元（Gated Recurrent Unit, GRU）通过引入门机制来控制信息的流动，有效地解决了梯度消失和梯度爆炸的问题，能够处理更长的序列和更复杂的任务。

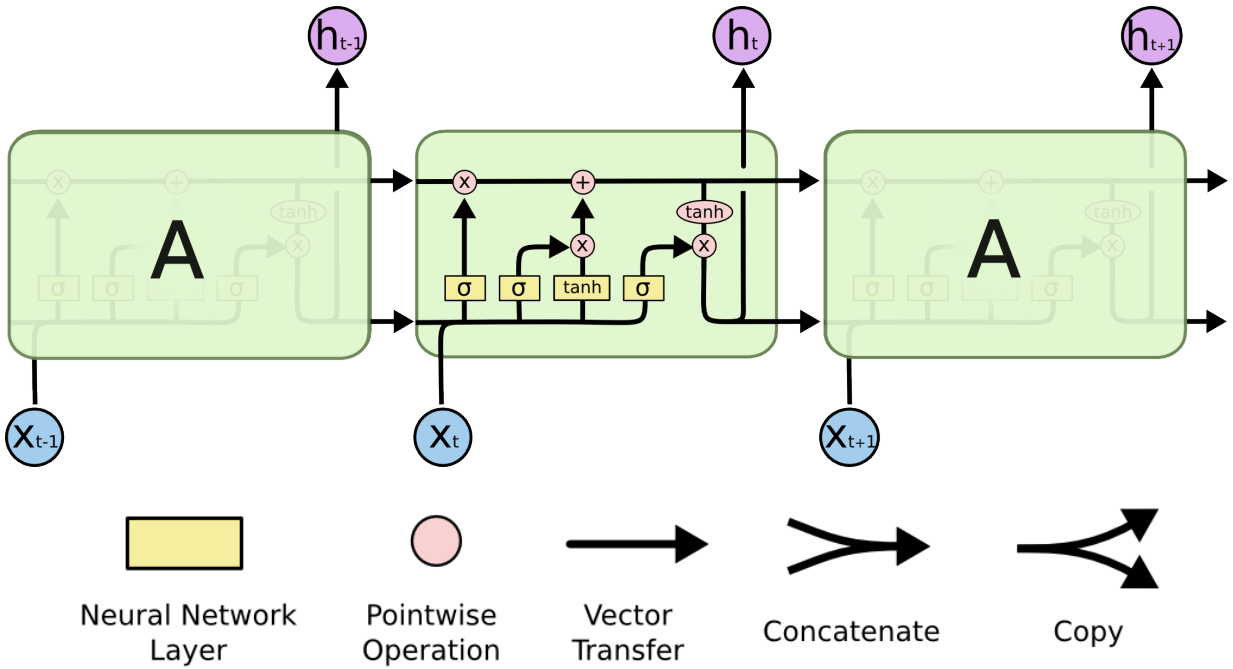


图 14: LSTM 模型计算示意图

LSTM

LSTM 背后的核心思想是通过门 (gate) 控机制调节单元状态 (cell state) 中的信息。

下图所示的水平黑线表示 LSTM 的单元状态，它贯穿整个循环链条，像传送带一样搭载着信息沿着时间步向前流动。LSTM 通过门控机制更新 (新增或删除) 单元状态中的信息。

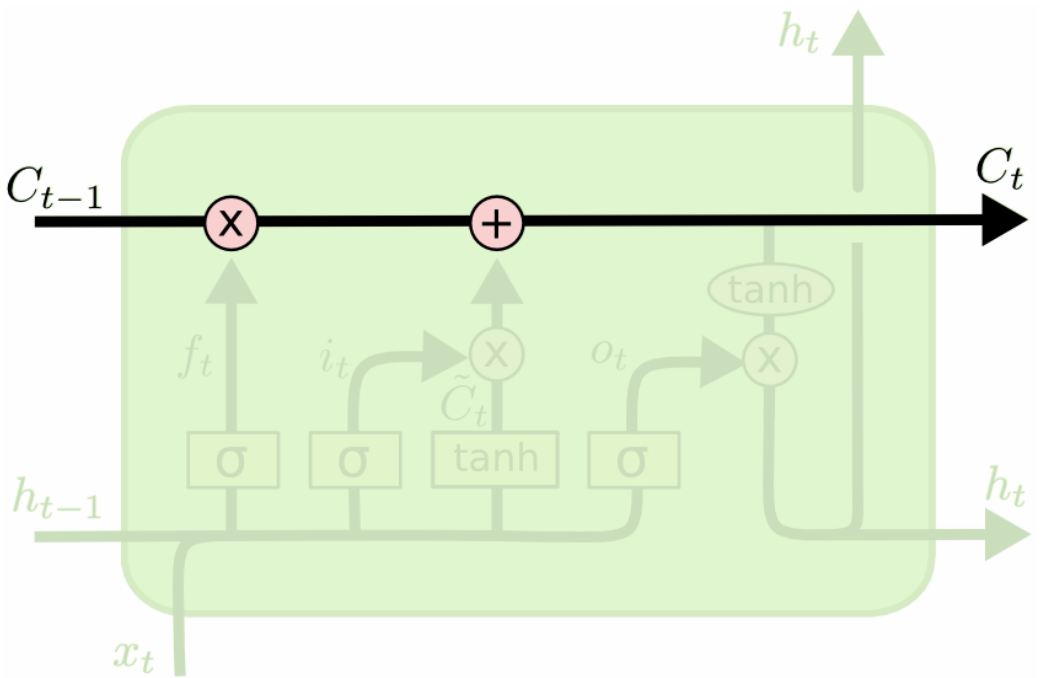


图 15: LSTM 的单元状态

LSTM

门控机制是一种选择性地让信息通过的方式，由 sigmoid 神经网络层和逐点乘法运算组成。

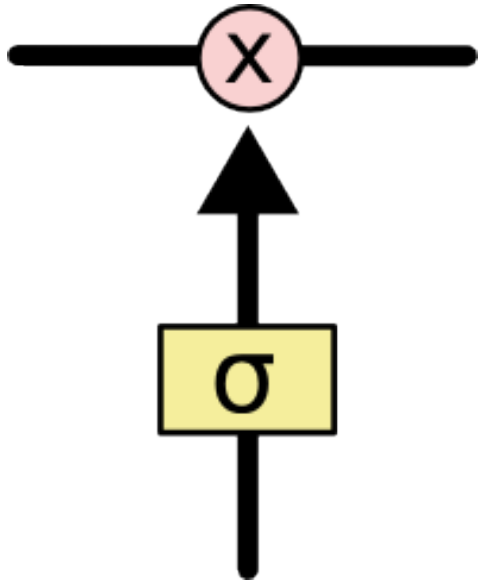
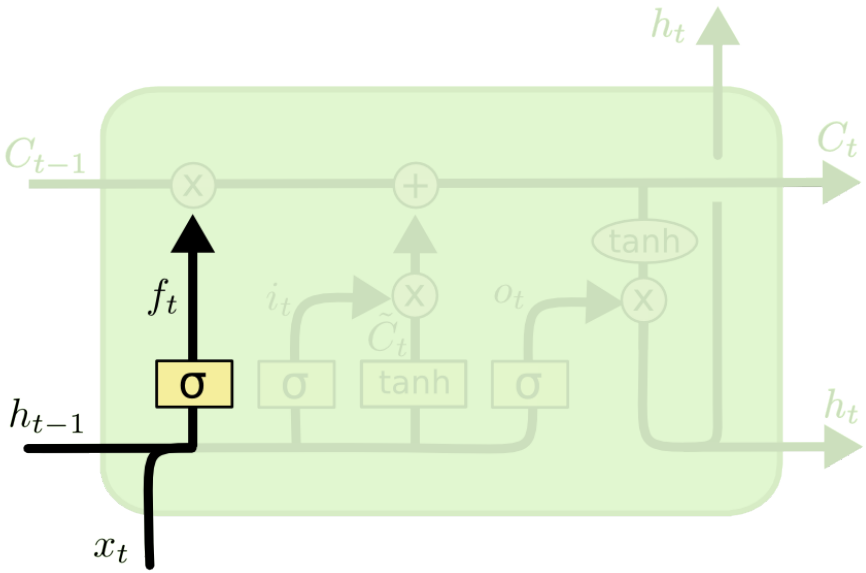


图 16: LSTM 的门控机制

sigmoid 层输出 0 到 1 之间的数字，描述每个组件应该有多少比例的信息通过。LSTM 包含三个门来控制单元状态中的信息。

LSTM

遗忘门 (forget gate) 确定从单元状态中丢弃哪些信息，通过 sigmoid 层为上一个单元状态 C_{t-1} 中的每一个数值计算出一个 0-1 之间的通过比例，0 表示完全丢弃，1 表示完全保留。输入为上一个时间步的隐藏状态 h_{t-1} 和当前时间步的输入 x_t。

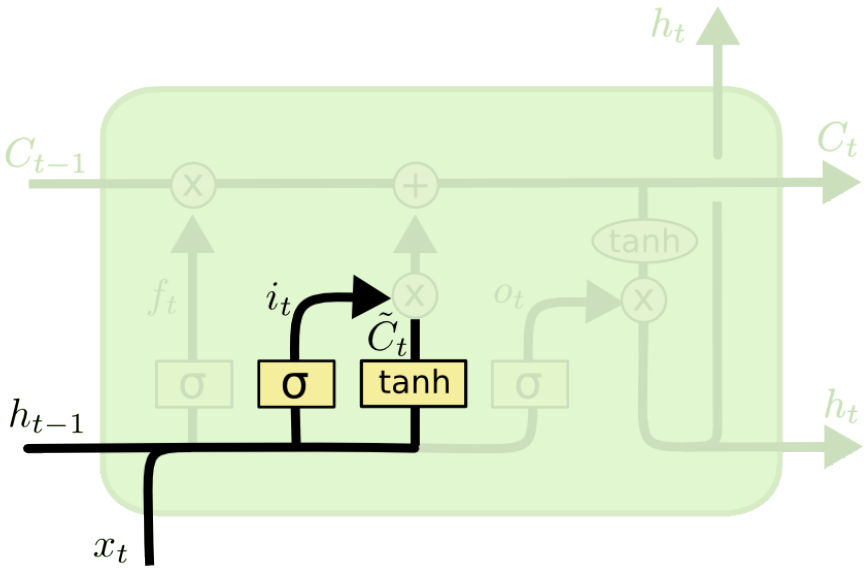


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

图 17: LSTM 的遗忘门

LSTM

输入门 (input gate) 确定为单元状态新增哪些信息，包括两部分：① sigmoid 层为单元状态中的每一个值计算新增信息的比例; ② tanh 层计算当前时间步的新增信息 $\tilde{C}$。



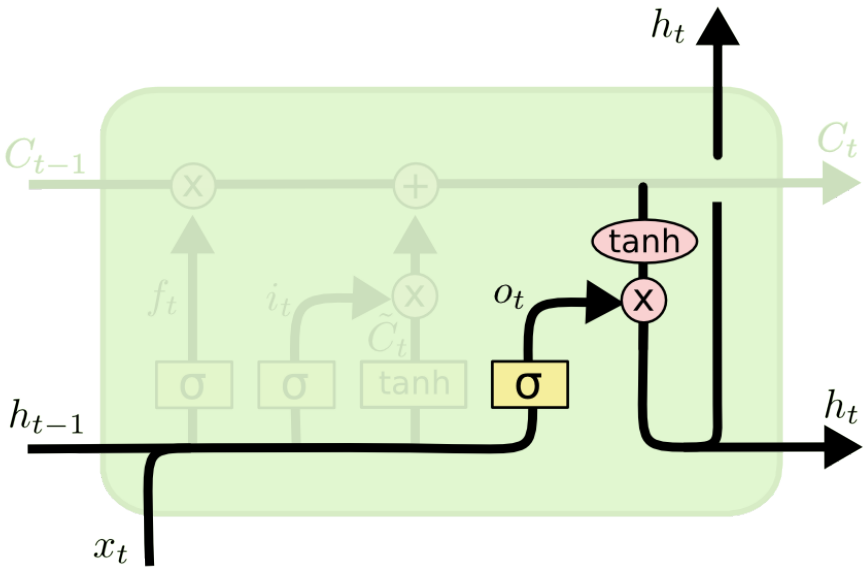
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

图 18: LSTM 的输入门

LSTM

输出门 (output gate) 确定从新的单元状态中输出哪些信息，包括两部分：① sigmoid 层为单元状态中的每一个值计算输出的比例; ② 为单元状态 C_t 应用 tanh 层，将数值压缩到 $[-1, +1]$ 区间。组合两部分计算得到当前时间步的输出值 (隐藏层值)h_t:



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

图 19: LSTM 的输出门

MLP 文本分类

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.datasets import fetch_20newsgroups
# MLP 分类器，包含两个隐藏层，维度均为 300
# 使用随机梯度下降优化参数
# 学习率设置为 0.1
mlp = MLPClassifier(hidden_layer_sizes=(300, 300), max_iter=200,
↪  alpha=1e-4, solver='sgd', verbose=10, tol=1e-4,
↪  random_state=1, learning_rate_init=.1)
# 使用训练数据训练模型
mlp.fit(X_train, y_train)
# 对测试集进行预测
y_pred = mlp.predict(X_test)
# 计算准确率
accuracy = (y_pred == y_test).mean()
print(f"Accuracy: {accuracy}")
```

CNN/RNN 文本分类

使用 PyTorch 框架，应用 CNN/LSTM 模型进行文本分类的基本步骤：

1. **初始化**：将文本转换为预训练词向量 (如，word2vec 等)。;
2. **前向传播**：将词向量作为 CNN/RNN 的输入，设定 CNN 中卷积层数、池化层数、卷积核尺寸、步长等参数 (设定 RNN 的层数，是否双向，使用使用 LSTM、GRU 等 RNN 变体);
3. **结果输出**：在 CNN/RNN 之后添加一个全连接层以产生分类输出;
4. **反向传播**：应用交叉熵计算模型损失函数值、计算梯度、更新参数;
5. **模型评估**：在测试集上对训练得到的模型进行性能评估，如查全率、查准率、F 值;

更多代码细节，详见课程网站本讲配套代码。

MLP 文本分类代码，请修改贝叶斯中文新闻二分类代码中的分类器部分代码

1. 参考技术博文*Understanding LSTM Networks*学习 LSTM 神经网络；
2. 借助大模型对话工具，学习使用深度学习框架 PyTorch：
   (1) 文心一言;
   (2) Kimi;
   (3) 通义千问;
   (4) 其他大模型 (请自行试用);
3. 结合该项目，学习使用 PyTorch 进行文本挖掘；
4. 使用本课程提供的中文新闻语料库，练习使用 PyTorch 构建深度学习文本分类器；

THE END