目录

Chapter 1

1.1

1.1.1 Estimating class probabilities in multiclass classification via the softmax function

The softmax function is a soft form of the argmax function; instead of giving a single class index, it provides the probability of each class. Therefore, it allows us to compute meaningful class probabilities in multiclass settings (multinomial logistic regression).

In softmax, the probability of a particular sample with net input z belonging to the ith class can be computed with a normalization term in the denominator, that is, the sum of the exponentially weighted linear functions:

$$p(z) = \sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{M} e^{z_j}} \tag{1.1}$$

1.1.2 Broadening the output spectrum using a hyperbolic tangent

Another sigmoidal function that is often used in the hidden layers of artificial NNs is the hyperbolic tangent (commonly known as tanh), which can be interpreted as a rescaled version of the logistic function:

$$\begin{aligned} \sigma_{logistic}(z) &= \frac{1}{1+e^{-z}} \\ \sigma_{tanh}(z) &= 2 \times \sigma_{logistic}(2z) - 1 = \frac{e^z - e^{-z}}{e^z + e^{-z}} \end{aligned} \tag{1.2}$$

The advantage of the hyperbolic tangent over the logistic function is that it has a broader output spectrum ranging in the open interval $(-1, 1)$, which can improve the convergence of the backpropagation algorithm.

Note that using torch.sigmoid(x) produces results that are equivalent to torch. nn.Sigmoid()(x). torch.nn.Sigmoid is a class to which you can pass in parameters to construct an object in order to control the behavior. In contrast, torch.sigmoid is a function.

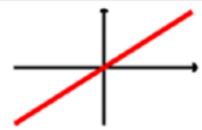
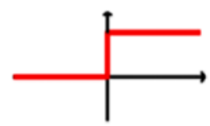
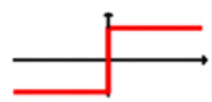
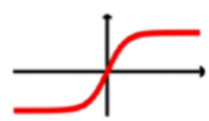
Activation function	Equation	Example	1D graph
Linear	$\sigma(z) = z$	Adaline, linear regression	
Unit step (Heaviside function)	$\sigma(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\sigma(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise linear	$\sigma(z) = \begin{cases} 0 & z \le -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \le z \le \frac{1}{2} \\ 1 & z \ge \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\sigma(z) = \dfrac{1}{1 + e^{-z}}$	Logistic regression, multilayer NN	
Hyperbolic tangent (tanh)	$\sigma(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\sigma(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

图 1.1: The activation functions covered

Chapter 2

Classifying Images with Deep Convolutional Neural Networks

2.1 The building blocks of CNNs

2.1.1 Discrete convolutions in one dimension

Determining the size of the convolution output

The output size of a convolution is determined by the total number of times that we shift the filter, $\boldsymbol{w}$, along the input vector. Let's assume that the input vector is of size n and the filter is of size m. Then, the size of the output resulting from $\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{w}$, with padding p and stride s, would be determined as follows:

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1 \tag{2.1}$$

2.1.2 Subsampling layers

Subsampling is typically applied in two forms of pooling operations in CNNs: max-pooling and mean-pooling (also known as average-pooling). The pooling layer is usually denoted by $P_{n_1 \times n_2}$.

The advantage of pooling is twofold:

- Pooling (max-pooling) introduces a local invariance. This means that small changes in a local neighborhood do not change the result of max-pooling. Therefore, it helps with generating features that are more robust to noise in the input data.

- Pooling decreases the size of features, which results in higher computational efficiency. Furthermore, reducing the number of features may reduce the degree of overfitting as well.

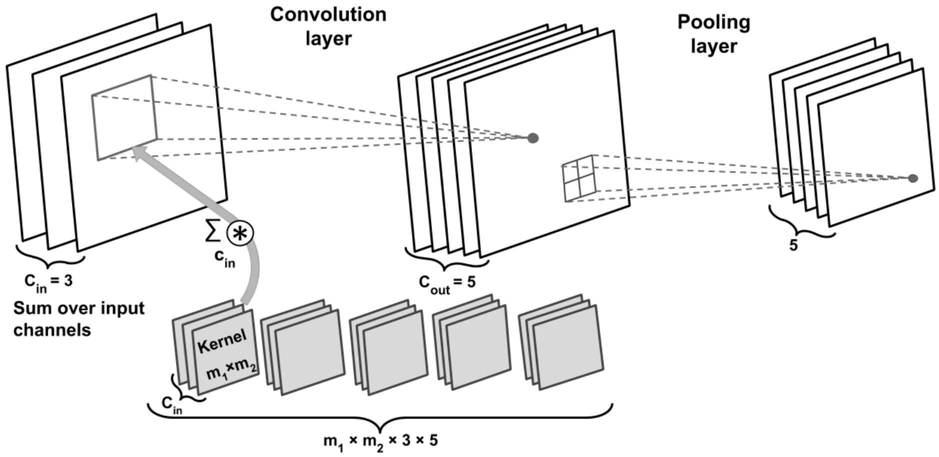


图 2.1: Implementing a CNN

Overlapping versus non-overlapping pooling

Traditionally, pooling is assumed to be non-overlapping. Pooling is typically performed on non-overlapping neighborhoods, which can be done by setting the stride parameter equal to the pooling size. For example, a non-overlapping pooling layer, $P_{n_1 \times n_2}$, requires a stride parameter $s = (n_1, n_2)$. On the other hand, overlapping pooling occurs if the stride is smaller than the pooling size.

2.2 构建卷积神经网络

2.2.1 处理多个输入通道

2.3 使用卷积神经网络对人脸图像进行微笑分类

2.3.1 图像转化和数据增广

以用五种不同类型的转换：

- 用一个边界框剪裁图像 cropping an image to a bounding box
- 水平翻转图像 flipping an image horizontally
- 调整对比度 adjusting the contrast
- 调整亮度 adjusting the brightness
- 剪裁中心图像并将生成的图像调整为原始大小 center-cropping an image and resizing the resulting image back to its original size

Chapter 3

用循环神经网络对序列数据建模

3.1 用于序列数据建模的循环神经网络

3.1.1 循环神经网络的循环机制

相邻时刻隐藏层信息的流动时神经网络可以记住过去的信息。通常用回路来表示相邻时刻隐藏层信息的流动。在表示循环神经网络的图中，这个回路也被称为**循环边**。

3.1.2 循环神经网络激活值计算

单层循环神经网络所有的权重矩阵如下：

- $\boldsymbol{W}_{xh}$: 输入层 $\boldsymbol{x}^{(t)}$ 和隐藏层 $\boldsymbol{h}$ 之间的权重矩阵
- $\boldsymbol{W}_{hh}$: 与循环边相关联的权重矩阵
- $\boldsymbol{W}_{ho}$: 隐藏层与输出层之间的权重矩阵

对于隐藏层，通过输入值的线性组合方式计算净输入 $\boldsymbol{z}_h$，即计算两个权重矩阵与对应输入向量的乘法的和，再加上偏置单元 $\boldsymbol{b}_h$：

$$\boldsymbol{z}_h^{(t)} = \boldsymbol{W}_{xh}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h \tag{3.1}$$

然后，计算隐藏层在 t 时刻的激活值：

$$\boldsymbol{h}^t = \sigma_h(\boldsymbol{z}_h^{(t)}) = \sigma_h(\boldsymbol{W}_{xh}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h) \tag{3.2}$$

其中，$\sigma_h(\cdot)$ 为隐藏层的激活函数

如使用组合权重矩阵 $\boldsymbol{W}_h = [\boldsymbol{W}_{xh}; \boldsymbol{W}_{hh}]$，那么公式将变为：

$$\boldsymbol{h}^t = \sigma_h\left([\boldsymbol{W}_{xh}; \boldsymbol{W}_{hh}]\begin{bmatrix}\boldsymbol{x}^{(t)}\\ \boldsymbol{h}^{(t-1)}\end{bmatrix} + \boldsymbol{b}_h\right) \tag{3.3}$$

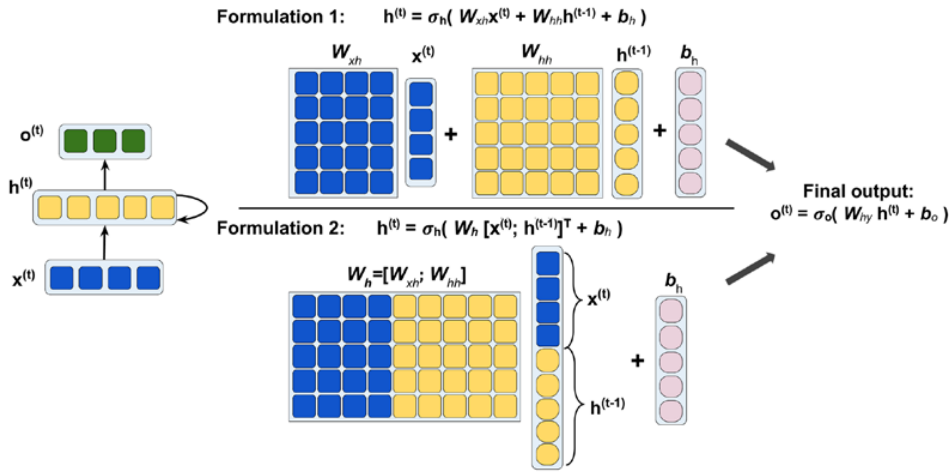


图 3.1: 计算单层循环神经网络隐藏层和输出层的激活值

在获得当前时刻隐藏层的激活值后，就可以计算输出层的激活值：

$$\boldsymbol{o}^{(t)} = \sigma_o(\boldsymbol{W}_{ho}\boldsymbol{h}^{(t)} + \boldsymbol{b}_o) \tag{3.4}$$

3.1.3 隐藏层循环与输出层循环

当存在输出层循环连接时，前一时刻输出层的激活值 $\boldsymbol{o}^{(t-1)}$ 可以通过以下两种方式循环：

- 添加到当前时刻的隐藏层 $\boldsymbol{h}^{(t)}$
- 添加到当前时刻的输出层 $\boldsymbol{o}^{(t)}$

3.1.4 远距离学习面临的问题

在实践中，解决序列中远距离依赖问题常用的三种解决方案如下：

1. 梯度剪裁
2. 截断时序方向传播（TBPTT）
3. 长短期记忆网络

梯度剪裁是指为梯度设定一个截止值或阈值，如果梯度超过该值则将梯度设为此截止值。而 TBPTT 限制反向传播时刻数量。

3.1.5 长短期记忆网络

在长短期记忆网络中，前一时刻的单元状态 $\boldsymbol{C}^{(t-1)}$ 直接参与当前时刻单元状态 $\boldsymbol{C}^{(t)}$ 的计算。记忆单元中的信息流由多个计算单元（通常称为门）控制。在 Figure 3.3 中，$\boldsymbol{x}^{(t)}$ 表示 t 时刻的输入数据，

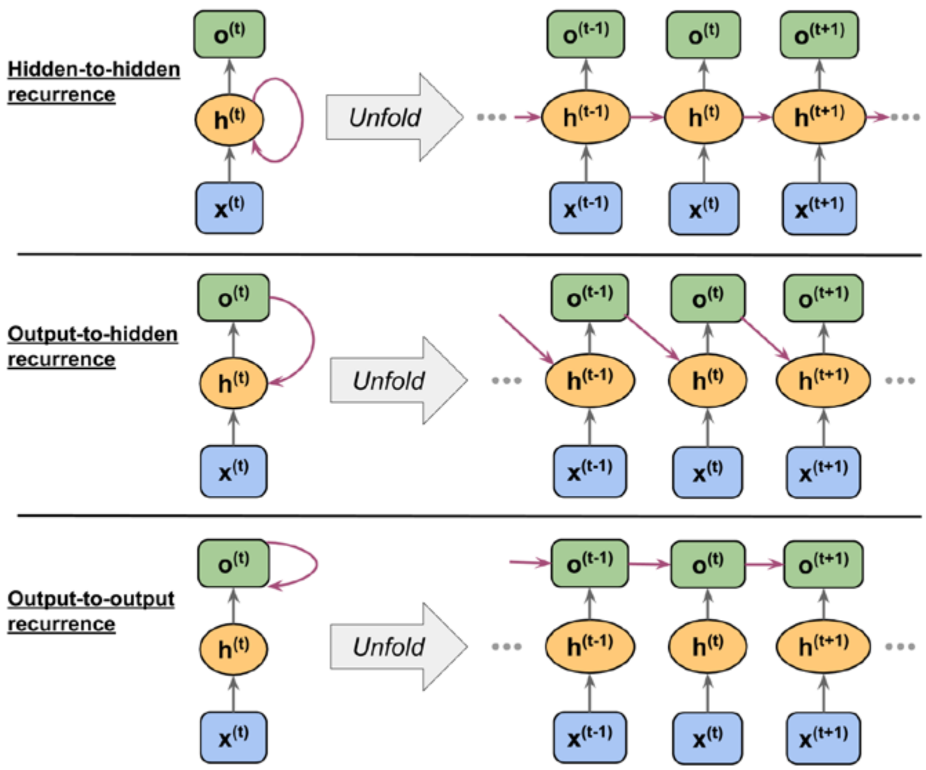


图 3.2: 不同循环连接模型

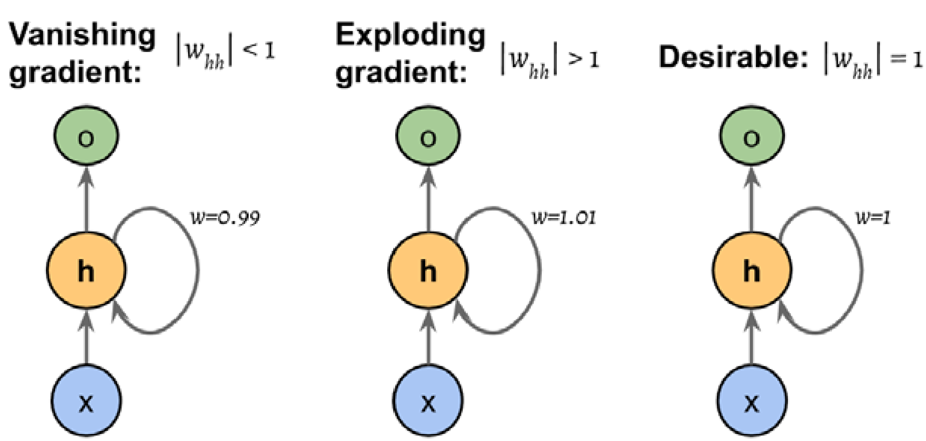


图 3.3: 计算损失函数梯度时出现的问题

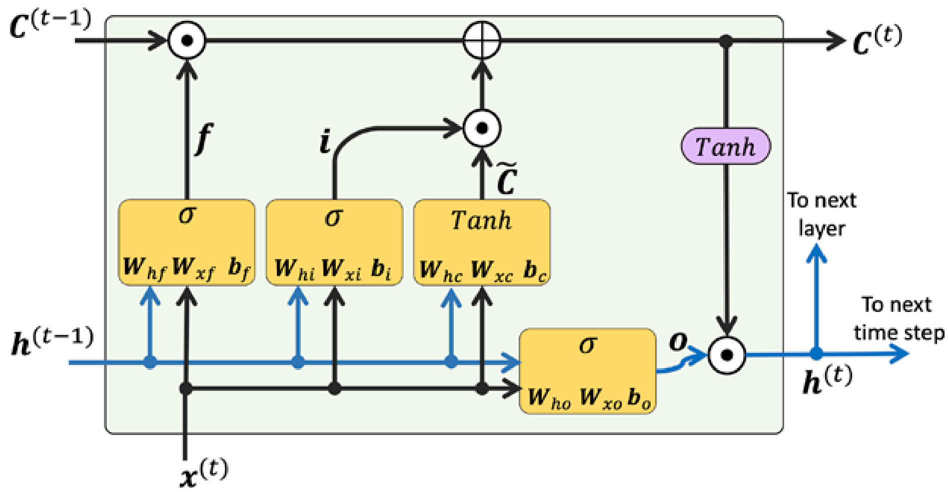


图 3.4: 长短期记忆网络单元的内部结构

$\boldsymbol{h}^{(t-1)}$ 表示 $t-1$ 时刻隐藏层输出。图中有 4 个框，每个框内给出了激活函数（sigmoid 函数 σ 或双曲正切函数 tanh、权重和偏置项。在每个框中，需要计算 $\boldsymbol{h}^{(t-1)}$ 和 $\boldsymbol{x}^{(t)}$ 向量与矩阵的乘积加上 $\boldsymbol{b}^{(t)}$，在通过激活函数得到输出值。输出的值经过运算（称为一个门)。$\bigodot$ 代表两个向量对应元素相乘，$\bigoplus$ 代表两个向量元素相加。

长短期基于网络有 3 种不同类型的门，分别为遗忘门、输入门和输出门。

遗忘门 $\boldsymbol{f}_t$ 可以重置单元状态，以防状态无限增长。事实上，遗忘门的任务就是决定保留哪部分信息，遗忘哪部分信息。计算公式如下：

$$\boldsymbol{f}_t=\sigma\left(\boldsymbol{W}_{xf}\boldsymbol{x}^{(t)}+\boldsymbol{W}_{hf}\boldsymbol{h}^{(t-1)}+\boldsymbol{b}_f\right) \tag{3.5}$$

输出门 $\boldsymbol{i}_t$ 和候选值 $\tilde{\boldsymbol{C}}_t$ 负责更新单元状态，计算公式如下：

$$\begin{aligned}\boldsymbol{i}_t&=\sigma\left(\boldsymbol{W}_{xi}\boldsymbol{x}^{(t)}+\boldsymbol{W}_{hi}\boldsymbol{h}^{(t-1)}+\boldsymbol{b}_i\right)\\ \tilde{\boldsymbol{C}}_t&=\tanh\left(\boldsymbol{W}_{xc}\boldsymbol{x}^{(t)}+\boldsymbol{W}_{hc}\boldsymbol{h}^{(t-1)}+\boldsymbol{b}_c\right)\end{aligned} \tag{3.6}$$

t 时刻的单元状态计算如下：

$$\boldsymbol{C}^{(t)}=\left(\boldsymbol{C}^{(t-1)}\bigodot\boldsymbol{f}_t\right)\bigoplus\left(\boldsymbol{i}_t\bigodot\tilde{\boldsymbol{C}}_t\right) \tag{3.7}$$

输出门 $\boldsymbol{o}_t$ 决定如何更新隐藏层的值：

$$\boldsymbol{o}_t=\sigma\left(\boldsymbol{W}_{xo}\boldsymbol{x}^{(t)}+\boldsymbol{W}_{ho}\boldsymbol{h}^{(t-1)}+\boldsymbol{b}_o\right) \tag{3.8}$$

综上所述，当前时刻隐藏层的输出计算如下：

$$\boldsymbol{h}^{(t)}=\boldsymbol{o}_t\bigodot\tanh\left(\boldsymbol{C}^{(t)}\right) \tag{3.9}$$