deep-learning 笔记

徐世桐

1 基本定义

label 标签：输出结果，$\hat{y}$ 为计算得到的结果，y 为实际测量结果

feature 特征：用于预测标签的输入变量，$x_j^{(i)}$ 为第 i 组 sample 第 j 号特征

sample 样本：一组特征的取值和对应的标签输出

batch：batch size 个 sample 被分为一组，进行向量化的计算，称 B

hyperparameter 超参数：人为设定的参数。如样本个数（批量大小 batch size）$|B|$，学习率 η。少数情况下通过学习得到

W 一层 layer 的权重矩阵：行数 = 前层节点数，列数 = 当前层节点数

全连接层 fully-connected layer/稠密层 dense layer：此层所有节点都分别和前一层所有节点连接

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

tanh 函数：$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

softmax 函数：$softmax(Y) = \frac{exp(y)}{\sum_{y' \in Y} exp(y')}$，将数值输出转化为概率值，1. 值为正 2. 值总和为 1

catagorical cross entropy 交叉熵

定义：分部 p 和 分部 q 间的 cross entropy $H(p,q) = -E_p(\log(q))$。为 expected value of $log(q)$ with respect to distribution p

公式：对一样本的输出 $\hat{y}^{(i)}$, $H(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{|B|}\sum_{j \in B} y_j^{(i)} log(\hat{y}_j^{(i)})$

对一批量的输出使用 cross entropy：对每一样本 i 的 $H(y^{(i)}, \hat{y}^{(i)})$ 求和

使用：联系两个值概率分部间的差异，即可将数值输出 $\hat{y}$ 和分类结果 y 直接做对比

适用于 multi-class classification，仅有唯一类别作为输出

仍可和 softmax 同时使用，softmax 将可能性先转换为正数并和为 1，随后使用 cross entropy

binary cross entropy

公式：$H(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{|B|}\sum_{j \in B} y_j^{(i)} log(\hat{y}_j^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}_j^{(i)})$

$\hat{y}^{(i)}, y^{(i)}$ 为单一标量

使用：

使用于单个输出的 logistic regression，或 multi-label regression

multi-label regression 代价值为每一 label 预测值的 binary cross entropy 求和

指数加权移动平均

$y_t = \gamma y_{t-1} + (1-\gamma) x_t$

小批量乘法

对 n 个形状为 (a,b) 矩阵 $X_1, X_2, ..., X_n$，n 个形状为 (b,c) 矩阵 $Y_1, Y_2, ..., Y_n$，小批量乘法结果为 $X_1Y_1, X_2Y_2, ..., X_nY_n$

* 特指被 word2vec 使用的乘法，其他模型仍根据矩阵乘法，非所有批量乘法都使用 小批量乘法

Sigmoid 二元交叉熵损失函数

得到向量 $P=[p_1,...,p_n]$，向量 $L=[l_1,...,l_n]$，掩码向量 $M=[m_1,...,m_n]$

p_i 标记 i 位置事件的可能性，l_i 标记期望 i 位置的事件发生 ($l_i=1$) 或不发生 ($l_i=0$)

计算:

1. 遍历 L，若 $l_i=0$，$p'_i=-p_i$
2. 对 P' 中每一 p'_i 取 sigmoid 值，对所有 $m_i=1$ 的 p'_i 求平均值

整体计算为: $f(P,L,M)=\frac{\sigma(P)\odot(-1)^L\odot M}{sum(M)}$

2 perceptron 分类算法

前向传播

$\hat{y}=h(W^Tx+b)$

激活函数 h(x)

当 $x>0$ 时 $h(x)=1$

当 $x\le 0$ 时 $h(x)=0$

迭代

$\theta_i=\theta_i+\alpha(y-h(x))x_i$

α 为学习率

3 反向传播公式推导

向量求导定义

标量对向量求导: $\frac{ds}{d\mathbf{v}}=\begin{bmatrix}\frac{ds}{dv_0} & \frac{ds}{dv_1} & \dots & \frac{ds}{dv_n}\end{bmatrix}$

向量对向量求导: $\frac{d\mathbf{u}}{d\mathbf{v}}=\begin{bmatrix}\frac{du_0}{dv_0} & \dots & \frac{du_0}{dv_n}\\ \dots & \dots & \dots\\ \frac{du_m}{dv_0} & \dots & \frac{du_m}{dv_n}\end{bmatrix}$

对 $z=Wx$ 有:

$\frac{z}{x}=W$

$\frac{dL}{dW}=\frac{dL}{dz}x$

对 $z=xW$ 有:

$\frac{dz}{dx}=W^T$

$\frac{dL}{dW}=x^T\frac{dL}{dz}$

对一层前向传播 $g(Z)$, $Z=XW+\vec{b}$

X 为前一层输入，即前一层 g(X) 矩阵

g 此处为广播操作，对矩阵 Z 中每一元素求 activation 值

+ 此处为广播操作，对每行 XW 加偏差

求权重斜率

单一权重值求导: $\frac{dL}{dw_{ij}}=\sum_k\frac{dL}{dZ_{kj}}X_{ki}$

对权重矩阵求导: $\frac{dJ}{dW}=X^T\frac{dL}{Z}$

对前一层输出 X 求导

$\frac{dL}{dX_{ij}} = \sum_k W_{jk} \frac{dL}{dZ_{ik}}$
$\frac{dL}{dX} = \frac{dL}{dZ} W^T$

对偏差求导

$\frac{dL}{db_i} = \sum_k \frac{dL}{dZ_{ki}}$
$\frac{dL}{d\vec{b}} = \vec{1}^T \frac{dL}{dZ}$

激活函数求导

反向传播：$\frac{dL}{dx} = \frac{dL}{dg(x)} \cdot g'(x)$

$g', \cdot$ 为按元素操作

sigmoid $g(z) = \frac{1}{1+e^{-z}}$ 求导：$g'(z) = g(z)(1-g(z))$

tanh $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ 求导：$g'(z) = 1 - g(z)^2$

softmax $\hat{y}_i = softmax(z_i)$ $J = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} log(\hat{y}_{ic})$,

求导：$\frac{dL}{dz} = \frac{1}{N}(\hat{y} - y)$

验证 gradient 计算

1. 得到斜率 $\frac{dL}{dW}$
2. 微调 $W + \epsilon$，计算 $L(W)$，$\frac{dL}{dW} \approx \frac{L(W+\epsilon) - L(W-\epsilon)}{2\epsilon}$
3. 比较两 $\frac{dL}{dW}$ 值是否相近

4 linear regression 线性回归

平方代价函数：$J(\theta) = \frac{1}{|B|} \sum_{i=1}^{|B|} J^{(i)}(\theta) = \frac{1}{2|B|} \sum_{i=1}^{|B|} (\hat{y}^{(i)} - y^{(i)})^2$，为所有样本误差的平均值

迭代：$\theta_i = \theta_i - \frac{\eta}{|B|} \sum_{j \in B} \frac{dJ^{(j)}(\theta)}{d\theta_i}$，即对所有 sample 训练一次，得到 label 差值，对每一参数减 斜率 * 学习率 的平均值

当使用平方代价函数：

$\theta_i = \theta_i - \frac{\eta}{|B|} \sum_{i \in B} x_i^{(j)} (x_1^{(j)} \theta_1 + x_2^{(j)} \theta_2 + ... + const - y^{(j)}) = \theta_i - \frac{\eta}{|B|} \sum_{i \in B} x^{(i)} (\hat{y}^{(j)} - y^{(j)})$

$const = const - \frac{\eta}{|B|} \sum_{i \in B} (x_1^{(j)} \theta_1 + x_2^{(j)} \theta_2 + ... + const - y^{(i)}) = const - \frac{\eta}{|B|} \sum_{i \in B} (\hat{y}^{(j)} - y^{(j)})$

对样本 i 的偏导数向量为 $\nabla_\theta J^{(i)}(\theta) = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ ... \\ 1 \end{bmatrix} (\hat{y}^{(i)} - y^{(i)})$

交叉熵代价函数：$J(\theta) = \frac{1}{|B|} \sum_{i \in B} H(y^{(i)}, \hat{y}^{(i)})$

softmax 线性回归：单层神经网络，使用 softmax 函数得到分类，使用 cross entropy 计算代价

过拟合问题

1. 权重衰减 regularisation：在代价函数中惩罚高权重的值，尽可能使所有权重值减小

每一权重的正则化值都会影响其余所有权重的斜率，由于正则化项为标量加入代价函数

L2 正则化代价函数 $= J(\theta) + \frac{\lambda}{2} \sum_{w \in W} w^2$，即 $J(\theta) + \frac{\lambda}{2} *$ 所有权重的平方和。λ 为超参数，决定权重衰减的程度

求导 $= \frac{dL}{dw} + \lambda w$

L1 正则化代价函数 $= J(\theta) + \lambda \sum_{w \in W} |w|$，即 $J(\theta) + \frac{\lambda}{2} *$ 所有权重绝对值和。

求导 $= \frac{dL}{dw} + \lambda sign(w)$

2. 丢弃法 dropout

每一权重（不包括 const）有 p 的几率 $\theta' = 0$，有 1-p 的几率 $\theta' = \frac{\theta}{1-p}$

常用 p = 0.5

仅将输出值部分设为 0，不删除原权重

反向传播时被隐藏的神经元向上一层传斜率为 0

为了得到确切的值，在测试模型时较少使用

初始化参数

1.MXNet 默认随机初始化：所有权重 $\sim N(0,1)$ 的 normal distribution，所有 const 取 0

2.Xavier 随机初始化：对一全连接层，输入个数 a，输出个数 b，则所有参数 $\sim U(-\sqrt{\frac{6}{a+b}}, \sqrt{\frac{6}{a+b}})$

预处理数据集

1. 特征标准化 standardization/z-normalization：$x' = \frac{x-\mu}{\sigma}$，即统计中 z 值

2. 离散值转换成指示特征：对于一个可取值为 A, B, C 的离散输入值，转换成 3 个数值输入。即如果原输入为 A，转换后 3 个数值输入为 1，0，0。原离散值为 B 则转换后为 0，1，0

3.Min-max normalisation：将每一特征值拉伸为 [a, b]，取一特征的最值 X_{min}, X_{max}，则拉伸后每一特征值 $X' = a + \frac{(X-X_{min})(b-a)}{X_{max}-X_{min}}$

存储 a, b, X_{min}, X_{max} 值可将数据逆操作得到原数据集

结构

- 将训练集分组，每组 batch_size 个 sample。
- 对这个 batch 的数据进行向量化计算，计算 loss，**斜率清零**，计算斜率，调用优化函数。
- 即每一 batch 使用相同的权重 偏差。一次训练一共历多次所有 sample，一次遍历进行 $\frac{sample\_size}{|B|}$ 次向量化计算

隐藏层必定使用激活函数，输出层可选使用激活函数

梯度下降的过程中一直使用真实梯度，无需减小学习率

5 convolutional neural network 卷积神经网络

互相关运算:

输入一个二维数组，和**二维核 kernel** 进行互相关运算，得到二维数组

二维核/卷积核/filter 过滤器：在输入数组上滑动，每次和二维数组矩阵一部分按元素相乘求和，作为输出矩阵的元素

二维卷积层:

将输入和卷积核做互相关运算，结果加上 const 作为输出

特征图：输出矩阵可看做是输入矩阵的表征，称特征图

感受野 receptive field:

对输出矩阵一元素 x，所有可能影响其值的输入矩阵元素称感受野

感受野可能大于实际输入的矩阵边界

填充 padding:

在输入矩阵外侧添加全零元素，使得输出矩阵的维度增加，由于可用的感受野增加。

常使用奇数 kernel，添加 $\lfloor \frac{kernel}{2} \rfloor$ 的填充，使得输出矩阵和输入矩阵纬度一样

步幅 stride:

定义每次感受野向左/向下移动的纬度

多通道输入输出:

当输入的数据包含多个矩阵，即多通道输入，例：RGB 图像有 3 个输入通道

对 c_i 输入, c_o 输出的卷积层，kernel shape 为 (c_o, c_i, 行数，列数)

每一输入通道有唯一的 kernel (c_i, 行数，列数) 对应，进行互相关运算后结果矩阵相加，作为一条输出通道的结果

多组 (c_i, 行数，列数) 分别产生输出通道的结果矩阵，则有 c_o 条输出通道

池化层:

作用：1 为了防止当输入变化时，输出立即随之更改。2 减少计算量

池化窗口，同卷积层的感受野。限定某块输入被同时考虑，同样有 stride，可对输入 padding

1. 最大池化层：取池化窗口内最大的输入
2. 平均池化层：取池化窗口平均值

多输入通道间池化结果不相加，即 输入通道数 = 输出通道数

LeNet 卷积神经网络

1. 使用 2 组 卷积计算层 激活函数层 池化层

输出通道数分别为 6，16。卷积层 kernel 为 (5, 5), 步幅为 1, 无 padding

激活函数层 对每一元素做 sigmoid

最大池化层 窗口 (2, 2)，步幅为 2

2. 使用 3 组全连接层

节点数 120, 84, 输出节点数。除输出层使用 sigmoid 激活函数，即 120 84 节点层

将 (批量大小, 通道数, height, width) 看做 (批量大小，通道数 * height * width) 处理

AlexNet 深度卷积神经网络:

除输出层和丢弃层，全部使用 relu 做激活函数

卷积部分

- 2 组 卷积层 + 最大池化层

```
nn.Conv2D(96, kernel_size=11, strides=4, activation='relu')
nn.MaxPool2D(pool_size=3, strides=2)
nn.Conv2D(256, kernel_size=5, padding=2, activation='relu')
nn.MaxPool2D(pool_size=3, strides=2)
```

- 3 卷积层 + 1 最大池化层，高输出通道，低卷积窗口

```
nn.Conv2D(384, kernel_size=3, padding=1, activation='relu')
nn.Conv2D(384, kernel_size=3, padding=1, activation='relu')
nn.Conv2D(256, kernel_size=3, padding=1, activation='relu')
nn.MaxPool2D(pool_size=3, strides=2)
```

全连接层部分

- 两 hidden layer 全连接层 使用丢弃法

```
nn.Dense(4096, activation="relu"), nn.Dropout(0.5)
nn.Dense(4096, activation="relu"), nn.Dropout(0.5)
nn.Dense(10) // 根据需求改变输出层节点，原论文为 1000
```

VGG 使用重复元素网络

VGG 基础块

数个 (3, 3)kernel 1 填充卷积层 + 1 个 (2, 2) 窗口 2 步幅最大池化层

卷积层 层数 通道数为超参数，一 VGG 块中每一卷积层有相同通道数

VGG 神经网络由 数个 VGG 块 + 数个全连接层 组成

例：**VGG-11**

1. (1, 64) (1, 128) (2, 256) (2, 512) (2, 512) 5 层 VGG 块
   (n, m) 代表此 VGG 块使用 n 层卷积层，各有 m 通道
2. 3 层全连接层，实现同 AlexNet 的全连接层部分

共 8 层卷积层 +3 层全连接层，所以称 VGG-11

NiN 神经网络

NiN 块

1 个自定义卷积层 + 2 层 (1, 1)kernel 卷积层，3 层卷积层都不包含池化层

自定义卷积层可设置 kernel，步幅，填充。

(1, 1) 卷积层可设置通道数 (= 自定义层通道数)，其余固定为默认值

NiN 神经网络有多组 (NiN 块 + 池化层)

例：**NiN 模型**

- NiN 块部分

```
nin_block(96, kernel_size=11, strides=4, padding=0)
nn.MaxPool2D(pool_size=3, strides=2)
nin_block(256, kernel_size=5, strides=1, padding=2)
nn.MaxPool2D(pool_size=3, strides=2)
nin_block(384, kernel_size=3, strides=1, padding=1)
nn.MaxPool2D(pool_size=3, strides=2)
```

- 在 NiN 块部分结束后加入丢弃层

```
nn.Dropout(0.5)
```

- 转化为对应分类个数的输出

```
nin_block(10, kernel_size=3, strides=1, padding=1)
nn.GlobalAvgPool2D() // 全局平均池化层，每一通道取矩阵所有元素的平均值
nn.Flatten() // 平均池化层结果即分类结果，flatten 仅用于改变 shape
```

GoogLeNet 含并行结构神经网络

Inception 块

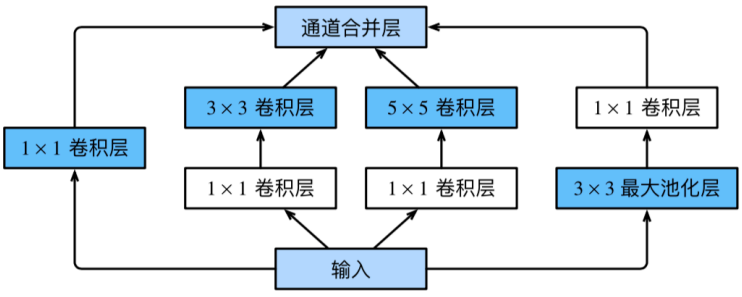


图 5.8: Inception块的结构

inception 块结构表示：$(n_1, (n_{21}, n_{22}), (n_{31}, n_{32}), n_4)$

第一线路使用 n_1 通道

第二线路第一卷积层使用 n_{21} 通道，第二层卷积层使用 n_{22} 通道 1 填充

第三线路第一卷积层使用 n_{31} 通道，第二层卷积层使用 n_{32} 通道 2 填充

第四线路第一最大池化层使用 (3, 3) 窗口 1 填充，第二层卷积层使用 n_4 通道

每一卷积层都使用 relu 激活函数

所有层输出作为不同通道结果，即最终有 $n_1 + n_{22} + n_{32} + n_4$ 通道

GoogLeNet 结构：

5 个串联模块，每一卷积层使用 relu 激活函数，每一模块间使用 (3, 3) 窗口 步幅 2 1 填充 池化层

1. 64 通道 (7, 7)kernel 2 步幅 3 填充卷积层 + 模块间池化层
2. 64 通道 (1, 1)kernel 卷积层 + 64 * 3 通道 (3, 3)kernel 1 填充卷积层 + 模块间池化层
3. 串联 2 inception 块 + 模块间池化层，分别有结构

   (64, (96, 128), (16, 32), 32)，通道比 `2:4:1:1`

   (128, (128, 192), (32, 96), 64)，通道比 `4:6:3:2`
4. 串联 5 inception 块 + 模块间池化层，

   (192, (96, 208), (16, 48), 64)

   (160, (112, 224), (24, 64), 64)

   (128, (128, 256), (24, 64), 64)

   (112, (144, 288), (32, 64), 64)

   (256, (160, 320), (32, 128), 128)
5. 串联 2 inception 块 + 全局平均池化层

   (256, (160, 320), (32, 128), 128)

   (384, (192, 384), (48, 128), 128)
6. 全连接层，节点数和分类类别数相同

6 CNN 优化方法

批量归一化 batch normalization

1. 对全连接层做批量归一

处于 输入的仿射变换 和 激活函数间，即 输出 $= \phi(BN(x))$ BN 为批量归一计算

1. 对于批量仿射 $x = Wu + b$，求标准化 $\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$。$\mu$ 和 σ 都为此组仿射变换的结果
2. $BN(x) = \gamma * \hat{x}_i + \beta$，$\gamma$ 拉伸 β 偏移。* + 为按元素加法乘法

2. 对卷积层做批量归一

处于 卷积计算 和 激活函数 间，卷积计算 + 批量归一 + 激活函数 + 池化层

各通道独立计算，各有独立拉伸 γ 偏移 β。

σ, μ 为此通道 一批量内 所有通道 的所有元素 的总体方差，平均值

得到 σ, μ 后对此通道此批量内所有元素求标准化

最终对此通道 偏移

ResNet 残差网络

残差块

训练时期望输出为 f(x) - x，而非直接使用 f(x) 期望输出。得到 f(x) - x 后 +x 得到 f(x)

1. 卷积层 (批量归一) + relu + 卷积层 (批量归一) 得到 $f(x) - x$
2. $[f(x) - x]$ + [(1, 1) 卷积层对 (x) 卷积结果] + relu 激活函数

   第一卷积层：(3, 3)kernel 1 填充 (通道数 步幅自定义)

   第 234 组残差组第一残差块 第一卷积层步幅为 2，否则为 1

第二卷积层：(3, 3)kernel 1 填充 1 步幅 (通道数自定义)

+x 步骤的 (1, 1) 卷积层：(通道数 步幅自定义)

第 234 组残差组 第一残差块使用 (1, 1) 卷积层，步幅为 2，否则直接 +x

3 层卷积层通道数共享同一自定义值，**要求 2 层卷积层输入输出通道数一致**

ResNet-18 模型：共 18 卷积层

1. 64 通道 (7, 7)kernel 2 步幅 3 填充 批量归一卷积层 + (3, 3) 窗口 2 步幅 1 填充最大池化层
2. 4 组 残差块，每组包含多个残差块

第一组 2 个残差块 输出通道数和 1 中输出通道数一致

第二三四组 各 2 个残差块 输出通道数为前一层通道数 ∗2

3. 全局平均池化层 + 对应输出结果数全连接层

DenseNet 稠密连接网络

类似 ResNet 残差网络，+x 步变为 concat x 连在输出结果后，即 x 直接传向下一层

稠密块

多组 (批量归一 + relu + (3, 3)kernel 1 填充卷积层 + concat x) 卷积层通道数相同

concat 操作为在通道纬度的 concat，即输入 x 作为额外输出通道。

增长率 = 输出通道 - 输入通道 = 卷积层通道数

过渡层

批量归一 + relu + (1, 1) 卷积层 + (2, 2) 窗口 2 步幅平均池化层

使用 (1, 1) 卷积层减小通道数，2 步幅平均池化层减小矩阵大小

卷积层通道数 = 输出通道数 / 2

DenseNet 模型

1. 64 通道 (7, 7)kernel 2 步幅 3 填充 批量归一卷积层 + (3, 3) 窗口 2 步幅 1 填充最大池化层
2. 4 组稠密块，由 3 个过渡层分隔

4 层稠密块卷积层数可以不相同

3. 批量归一 + relu + 全局平均池化层 + 对应输出结果数全连接层

7 RNN 循环神经网络

记录数据状态，根据以往状态和当前输入决定输出

n 阶马尔科夫链：一个词的出现仅和前 n 个词有关

语言模型：词序 (w_1, w_2, ..., w_T) 的出现可能性为

$P(w_1, w_2, ..., w_T) \approx \prod_{t=1}^{T} P(w_t|w_{t-(n-1)}, ..., w_{t-1})$

称 n 元语法，每一 w_t 为一时间步中出现的词

处理语言模型：将每一文字转化为索引，使用索引做训练参数集

one_hot 表示：索引为 i 的词对应 one_hot 向量 $[v_0 = 0, v_1 = 0, ..., v_i = 1, v_{i+1} = 0, ...]$

采样方式:

$BATCH\_SIZE$ 每次采集的样本数

NUM_STEPS 每个样本包含的时间步数，

1 随机采样:

[1 2 3 4] [5 6 7 8] [9 10 11 12] [13 14 15 16]

将所有样本分为头尾相连的组，每组有相等可能性被取值，每次随机取 BATCH_NUM 组

训练来自不同批量的样本时不能将前一次隐藏层结果纳入计算

2 相邻取样:

[1 2 3 4] [5 6 7 8] [9 10 11 12] [13 14 15 16]

[17 18 19 20] [21 22 23 24] [25 26 27 28] [29 30 31 32]

将样本填入 BATCH_NUM 行矩阵，再分为 (BATCH_NUM, NUM_STEPS) 的小矩阵，每次每个小矩阵有等可能性被选择

仅需在训练一开始初始化隐藏层结果，而非在每一批量开始初始化

裁剪梯度

将所有参数拼接成向量 g，进行裁剪：$g' = min(\frac{\theta}{||g||}, 1)g$

困惑度

= exp(交叉熵损失函数值)

RNN 实现

模型:

输入 X_t，输出 O：时间步数 个 (批量大小，词典大小) 矩阵

1. 隐藏层 $H_t = \phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$

X_t 为上一时间步的词

H_{t-1} 项为隐藏层前一次输出，称隐藏状态，第一个时间步使用全零 H_{t-1} 矩阵

简化计算：$X_t W_{xh} + H_{t-1} W_{hh} = [X_t, H_{t-1}][W_{xh}, W_{hh}]^T$

激活函数 ϕ 为 tanh

2. 输出层 $O = HW_{hq} + b_q$

实现:

训练时输入一段词序 $[w_0, w_l]$，输出即为预测 $[w'_1, w'_N]$ 的词

批量输入：(批量大小, 时间步数 l) 每一元素为单个词，即一批量的句子前缀

输入转换为 时间步数 个 (批量大小, 词典大小) 矩阵，第 i 矩阵第 j 行对应 批量中第 j 样本 第 i 时间步的词的 one_hot 向量

计算：预测 T 长度的词序

时间步 t 从 1 开始

1.$t = 0$ 的隐藏状态 H_0 设为全 0 向量

2.$t \leq T$：使用 (prefix 在 t 位置的词，隐藏状态 H_{t-1}) 计算 (t 位置预测词，H_t)

输出词 $[w'_1, w'_T]$ **不一定和输入** $[w_1, w_T]$ **一致**，但在 $[1, T]$ 时间段内使用输入词 w_{t-1}，而非上一预测词 w'_{t-1} 进行预测

3.$t > T$：使用 (上一预测词，H_{t-1}) 预测

训练时标签长度和 prefix 长度相同，即到达 t = T 时预测停止

代价函数对每组 预测词和目标词的 one_hot 使用 softmax 交叉熵，求和

通过时间反向传播

有关时间步的损失函数：$L = \frac{1}{T} \sum_{t=1}^{T} l(o_t, y_t)$ o_t **此处为单一批量的 t 时间步输出**

反向传播公式

针对单一样本，假设隐藏层不使用激活函数

$h_t = W_{xh} x_t + W_{hh} h_{t-1} + b_h$

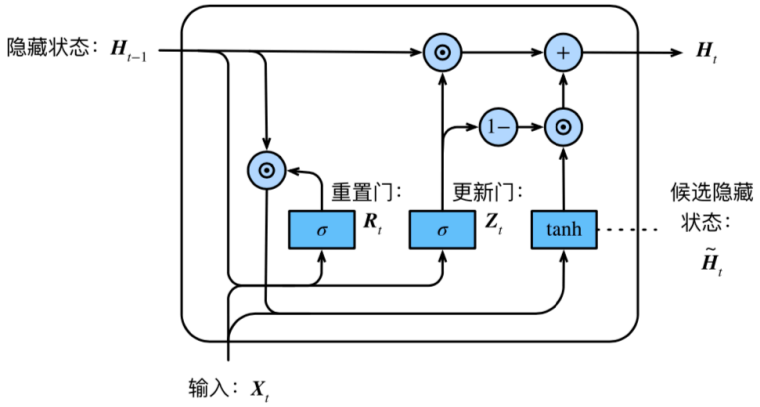
$o = W_{hq} h_t$

$\frac{dL}{do_t} = \frac{1}{T} \frac{dl(o_t, y_t)}{do_t}$

$\frac{dL}{dW_{qh}} = \sum_{t=1}^{T} \frac{dL}{do_t} h_t^T$
$\frac{dL}{dh_T} = W_{qh}^T \frac{dL}{do_T}$
当 t < T: $\frac{dL}{dh_t} = W_{hh}^T \frac{dL}{dh_{t+1}} + W_{qh}^T \frac{dL}{do_t}$
$= \sum_{i=t}^{T} (W_{hh}^T)^{T-i} W_{qh}^T \frac{dL}{do_{T-i+t}}$
$\frac{dL}{dW_{hx}} = \sum_{t=1}^{T} \frac{dL}{dh_t} x^T$
$\frac{dL}{dW_{hh}} = \sum_{t=1}^{T} \frac{dL}{dh_t} h_{t-1}^T$

GRU 门控循环单元

替代原计算隐藏状态方法，应对梯度衰减



reset gate 重置门 update gate 更新门:

得到上一层影藏层结果 H_{t-1} 当前时间步输入 X_t

重置门 $R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$

更新门 $Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$

W 为权重参数 b 为偏差参数 σ 为 sigmoid 函数

候选隐藏状态

候选隐藏状态 $\tilde{H}_t = tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$

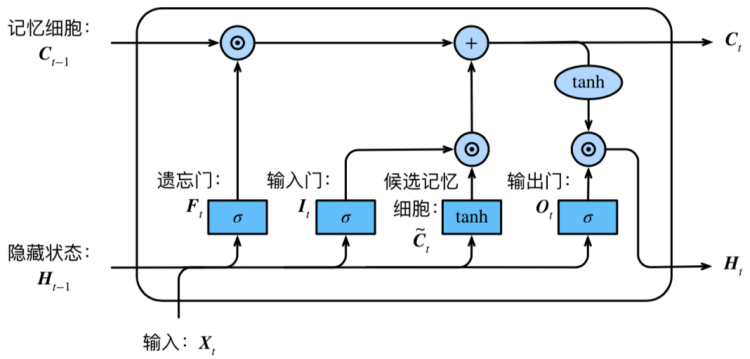
$\odot$ 为按元素相乘，使得重置门中对应位置值为 0 的元素被丢弃

隐藏状态

$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$

输出公式不变: $O_t = H_t W_{hq} + b_q$

LSTM 长短期记忆门控循环网络



input gate 输入门 forget gate 遗忘门 output gate 输出门候选记忆细胞

输入门 $I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$

遗忘门 $F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$

输出门 $O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$

候选记忆细胞 $\tilde{C}_t = tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$

记忆细胞

$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$

隐藏状态

$H_t = O_t \odot tanh(C_t)$

深度循环神经网络

包含多个隐藏层的 RNN

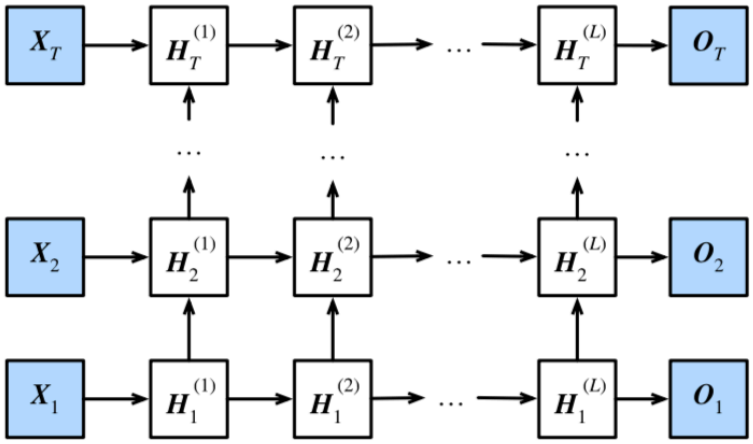


图 6.11: 深度循环神经网络的架构

结构:

隐藏层**输出**为 $H_i^{(1)} H_i^{(2)} ... H_i^{(n)}$，第 i 次 forward 对应图中一行，即 $X_i \to H_i^{(1)} \to ... \to H_i^{(n)} \to O_i$

第 i 次计算中:

$l = 1$ 隐藏层输出 $H_i^{(1)} = \phi(X_i W_{xh}^{(1)} + H_{i-1}^{(l)} W_{hh}^{(1)} + b_h^{(1)})$

和单层隐藏层 RNN 的 隐藏层输出公式 一致

$l = 2, 3 ... n$ 隐藏层输出 $H_i^{(l)} = \phi(H_i^{(l-1)} W_{xh}^{(l)} + H_{i-1}^{(l)} W_{hh}^{(l)} + b_h^{(l)})$

即 $l = 1$ 公式中将输入变为前一隐藏层输出

输出层 $O_i = H_i^{(n)} W_{hq} + b_q$

双向循环神经网络

允许神经网络根据前后的词序决定当前时间步的词

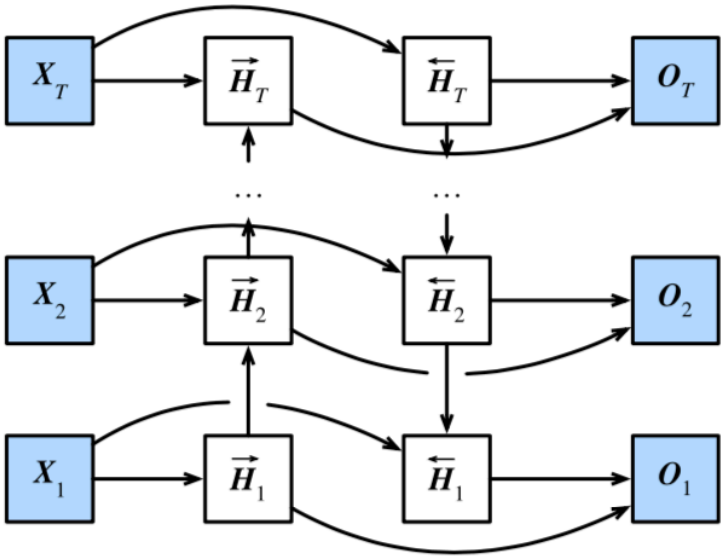


图 6.12: 双向循环神经网络的架构

结构:

仅有 2 隐藏层，分为 正向隐藏层 反向隐藏层，分别输出 $H^{(f)}$ $H^{(b)}$

第 i 次计算中

$H_i^{(f)} = \phi(X_i W_{xq}^{(f)} + H_{i-1}^{(f)} W_{hh}^{(f)} + b_h^{(f)})$

$H_i^{(b)} = \phi(X_i W_{xq}^{(b)} + H_{i+1}^{(b)} W_{hh}^{(b)} + b_h^{(b)})$

$O_i = H_i W_{hq} + b_q$

$H_i = (H_i^{(f)}, H_i^{(b)})$，为正向 反向隐藏层的输出 concat

8 代码算法优化

Stochastic Gradient Decent 随机梯度下降：采样方法

每次迭代随机选择一个**样本**，得到权重针对此样本的斜率进行梯度下降 而不是求参数针对所有样本的代价函数斜率。开销从 O(n) 变为 O(1)

即任取的样本 j，SGD 迭代为 $\theta_i = \theta_i - \eta \frac{dJ^{(j)}(\theta)}{d\theta_i}$

训练时 noisy

batch gradient descent 小批量随机梯度下降：采样方法

每次迭代随机选择一组样本 B，计算参数关于 B 的代价函数斜率

分组目的为减少 noisy，并避免需要迭代过整个训练集才能计算斜率

重复采样：允许 B 中重复出现同一样本。反之不重复采样，

随迭代次数增加，学习率可减小，使学习率和斜率的乘积方差减小

动量法：迭代方法

解决固定一学习率值无法同时满足多个参数的学习率范围，导致某些参数发生学习太慢，某些参数不断越过最优解

定义:

向量 v_t 为第 t 次迭代每一参数的速度变量

向量 x_t 为第 t 次迭代参数向量

向量 g_t 为第 t 次迭代每一参数的斜率

迭代:

$v_t = \gamma v_{t-1} + \eta_t g_t$

$x_t = x_{t-1} - v_t$

adaptive learning rate

对每一参数使用不同 learning rate，当参数更新较慢时增加学习率，反之减少

以下迭代方法均为 adaptive learning rate 算法

AdaGrad 算法：迭代方法

根据参数斜率调整学习率

定义向量 s_t 为第 t 次迭代累加变量，累计每一参数的斜率平方和

迭代:

$s_t = s_{t-1} + g_t \odot g_t$

$x_t = x_{t-1} - \frac{\eta}{\sqrt{s_t+\epsilon}} \odot g_t$

即每一参数有变量记录所有斜率历史，每次迭代时 学习率/斜率历史 l2 norm

由于累加变量斜率历史，学习率始终降低，造成学习缓慢

RMSProp 算法：迭代方法

解决 AdaGrad 末期学习率过低问题

迭代:

$s_t = \gamma s_{t-1} + (1-\gamma) g_t \odot g_t$ 即对 s_t 按元素平方做指数加权

x_t 同 AdaGrad 算法

AdaDelta 算法：迭代方法

解决 AdaGrad 末期学习率过低问题，不使用超参数

定义:

向量 g'_t 记录参数变化量

向量 Δx_t 对 g'_t 按平方做指数加权

迭代:

$s_t = \gamma s_{t-1} + (1-\gamma) g_t \odot g_t$ 同 RMSProp

$\Delta x_t = \gamma \Delta x_{t-1} + (1-p) g'_t \odot g'_t$ Δx_t 初始化为全零向量

$g'_t = \sqrt{\frac{\Delta x_t+\epsilon}{s_t+\epsilon}} \odot g_t$

$x_t = x_{t-1} - g'_t$ g'_t 即参数变化率

Adam 算法：迭代方法

定义:

v_t t 时间步的 动量变量

s_t t 时间步的 指数加权移动平均

g_t t 时间步 小批量**随机梯度**

$0 \le \beta_1 < 1$ 所有 动量变量的权重超参数。常取 0.9

$1 \le \beta_2 < 1$ 所有 指数加权移动平均变量超参数。常取 0.999

迭代:

$v_t = \beta_1 v_{t-1} + (1-\beta_1) g_t$

$s_t = \beta_2 s_{t-1} + (1-\beta_2) g_t \odot g_t$

$\hat{v}_t = \frac{v_t}{1-\beta_1^t}$

$\hat{s}_t = \frac{s_t}{1-\beta_2^t}$

使用 $\hat{v}_t$, $\hat{s}_t$ **偏差修正**

使得每一时间步 t 前的权重和为 1，使得在时间步数较小时动量值不受权重影响

例：不使用偏差修正：$v_1 = 0.1 g_1$

$g'_t = \frac{\eta \hat{v}_t}{\sqrt{\hat{s}^t + \epsilon}}$

ϵ 常取 10^{-8}

$x_t = x_{t-1} - g'_t$

9 计算机视觉

提升模型泛化能力方法：图像增广 图像微调

图像增广

对图像随机变换，产生相似样本。扩大训练集

方法：

翻转：有固定几率上下/左右翻转

裁剪：随机裁剪原图 10%-100% 面积的图像，宽高比 0.5-2，并拉伸至固定像素大小

变化颜色：调整亮度 色调 对比度 饱和度

微调

假设：

源模型包含的知识和目标模型紧密相连

源模型输出层不能直接用于目标模型

目标数据集远小于源数据集

方法：

1. 在源数据集上训练一神经网络，称源模型
2. 创建一新神经网络，称目标模型，复制源模型除了输出层的所有结构和参数
3. 为目标模型添加输出层，节点数对应输出种类，随机初始化模型参数
4. 在目标数据集上训练目标模型，微调 hidden layer 参数，从头训练输出层参数

通过提高输出层学习率，降低隐藏层学习率达到重新训练输出层，微调隐藏层的目的。学习率相差可达 1000 倍

目标检测 创建 匹配锚框算法

锚框：标记一个物体的范围框

针对参数 $s = s_1, ..., s_n$, $r = r_1, ..., r_m$

为避免生成过多锚框，将图片分为不同区域 grid，以每一区域中心生成锚框。区域 grid 可重叠

每一 grid 生成 $(s_1, r_1), (s_1, r_2), ..., (s_1, r_m), (s_2, r_1), ..., (s_n, r_m)$ 共 $n + m - 1$ 个锚框

对 H, W 大小的图片，h 行 w 列个区域情况中，每一区域中以 (s_i, r_j) 为参数的锚框有 高 $= s_i * \sqrt{r_j} * \frac{H}{h}$，宽 $= \frac{s_i}{\sqrt{r_j}} * \frac{W}{w}$

`contrib.ndarray.MultiBoxPrior` 和 `d2l.torch.multibox_prior` 得到高 $= \frac{s_i}{\sqrt{r_j}}$ 宽 $= s_i * \sqrt{a_j}$

交并比 IoU：两锚框 相交面积/相并面积

训练集：每一锚框有对应的目标的标签 相对目标范围框的偏移

赋目标框

1. 对锚框组 $A_1, A_2, ...A_n$，目标框组 $B_1, B_2, ...B_m$ 定义矩阵 X，
   X 为 (n, m)，包含每一锚框相对每一目标框的交并比
2. 找到 X 中值最大项 x_{ij}，则 A_i 对应目标 B_j，移除 X 中 i 行 j 列
3. 重复找到剩余矩阵中的最大项并移除，最终有 m 锚框对应目标，剩余 $n-m$ 锚框
4. 对每一未对应锚框 A_i，寻找 X 中 i 行最大交并比，若值大于阈值，为 A_i 分配对应目标框，若没有交并比大于阈值，目标框为整个图片。

被赋予目标框的锚框称正类锚框，否则为负类锚框

赋偏移量

对锚框 A_i 有坐标 + 长宽 (x_a, y_a, h_a, w_a)，对应目标框 B_j 有 (x_b, y_b, h_b, w_b)

设置常数 $\mu_x = \mu_y = \mu_w = \mu_h = 0$, $\sigma_x = \sigma_y = 0.1$, $\sigma_w = \sigma_h = 0.2$

偏移量为 $(\frac{\frac{x_b - x_a}{w_a} - \mu_x}{\sigma_x}, \frac{\frac{y_b - y_a}{h_a} - \mu_y}{\sigma_y}, \frac{log(\frac{w_b}{w_a}) - \mu_w}{\sigma_w}, \frac{log(\frac{h_b}{h_a}) - \mu_h}{\sigma_h})$

非极大值抑制：non-maximum suppression

在显示结果阶段，去除重复对一个物体分类的锚框。**在训练结束后输出目标时使用，训练时直接使用所有锚框和标签集对比**

锚框置信度：一个锚框 A_i 针对所有目标锚框 B 计算概率，A_i 最大概率符合的目标锚框对应 A_i 的预测类别，此最大概率为 p。概率不是锚框和目标锚框的交并比，而是输出层输出的 每一锚框与每一类别 的可能性

算法：

1. 计算每一锚框置信度，选取最高的锚框 A_i
2. 将所有和 A_i 交并比高于阈值的锚框删除

重复 1.2. 步，直至没有锚框剩余

多尺度目标检测：仅适用部分像素点作为锚框的中心，减少锚框数量，减少计算

SSD 单发多框检测：一种目标检测算法

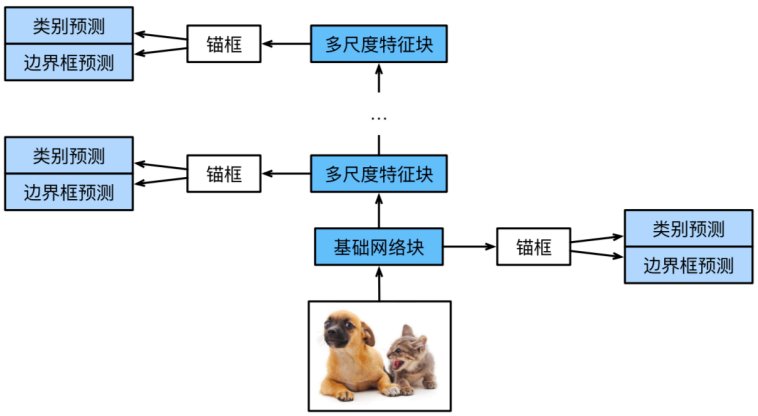


图 9.4: 单发多框检测模型主要由一个基础网络块和若干多尺度特征块串联而成

类别预测层：

得到卷积网络的张量，形状 (批量大小，通道数，高，宽)，处于同一高宽坐标的元素对应原图片中一区域 grid 的像素，即原图区域为此元素的感受野。此区域中可包含多个锚框。

假设锚框中心像素有 (h, w) 个，每个中心点生成 a 个锚框，每个锚框需匹配进 $(q+1)$ 个分类，多 1 分类对应负类分类

将锚框看做多层，每层 $h*w$ 个，共 a 层，层编号为 $c_1, c_2, ..., c_a$

方法 0：对每一锚框使用全连接层分类，即每一像素输入全连接网络，最终有匹配类别个数的输出层，判断锚框分类

造成参数过多，无法训练

方法 1：使用卷积层通道数输出类别

包含一个保持输入高 宽的卷积层，如 1 填充 (3, 3)kernel 的卷积层

输出通道数为 $a*(q+1)$，每一通道输出仍为矩阵，第 $(i-1)*(q+1)+j$ 通道包含锚框层 c_i 中所有锚框对分类 j 的匹配可能性。**由于输入张量每一元素对应一区域 grid，卷积层输出 (h, w) 位置的值的可能性即为此锚框层属于 (h, w) 区域的锚框对一类别的预测可能性**

边界框预测层：

输入结构同类别预测层，输出通道数为 $a*4$，4 对应一个锚框有 4 个偏差值，每一通道输出仍为矩阵，第 $(i-1)*(q+1)+j$ 通道包含锚框层 c_i 中所有锚框的第 i 号偏差值

连接多尺度预测：

将不同多尺度特征块和基础网络块产生的 类别 边界预测层结果 合并

(批量大小, 通道数, 高, 宽) 转为 (批量大小, 通道数 * 高 * 宽)

高宽减半块：

1. 两组 [(3, 3)kernel 1 填充 卷积层 + 批量归一化 + relu]

2.(2, 2) 窗口 2 步幅 最大池化层

基础网络块

串联 3 块高宽减半块，分别有 16, 32, 64 通道

整体模型

结构对应图例，每一模块 l_i 对结果张量 (批量大小, h_i, w_i) 有 类别 边界预测层

1. 基础网络块

2-4. 高宽减半块，通道数都为 128

5. 仅包含全局最大池化层 高宽降至 1

得到输出集

5 层 类别 边界预测结果分别通过 concat 得到总体类别预测 边界预测，用于计算代价。

类别分类 concat 结果：(批量大小，$\sum_i h_i*w_i$ * 锚框数，类别数 +1)。

边界预测 concat 结果：(批量大小，$\sum_i h_i*w_i$ * 锚框数 * 4)，4 对应偏移量个数

类别预测使用 `SoftmaxCrossEntropyLoss`，边界预测使用 `L1Loss`。**由于所有坐标通过百分比表示，可直接和标签集一一对应，不受卷积层对矩阵大小影响**

得到标签集

对于 5 层中所有使用过的锚框，使用**目标检测**章节的算法对每一锚框添加标签 偏移量。

结果得到 3 个张量：

每一锚框的类别，此张量所有元素和输出的类别预测进行交叉熵代价计算

锚框数 *4 大小的 bitmap，代表哪些锚框的偏移量被纳入代价函数计算。即 将此 bitmap 和第三张量按元素相乘 结果取平方和，为偏移量代价值

锚框数 *4 大小的张量，代表每一锚框的偏移量。负类锚框的偏移量被忽略

针对同一网络，不同迭代中产生的锚框位置 大小不变。由于目标框计算基于交并比，偏移量计算值不变。计算偏移量代价作用为：检查网络对偏移量的预估是否正确

R-CNN 区域卷积神经网络

对每一提议区域做卷积神经网络训练，耗时长

Fast R-CNN

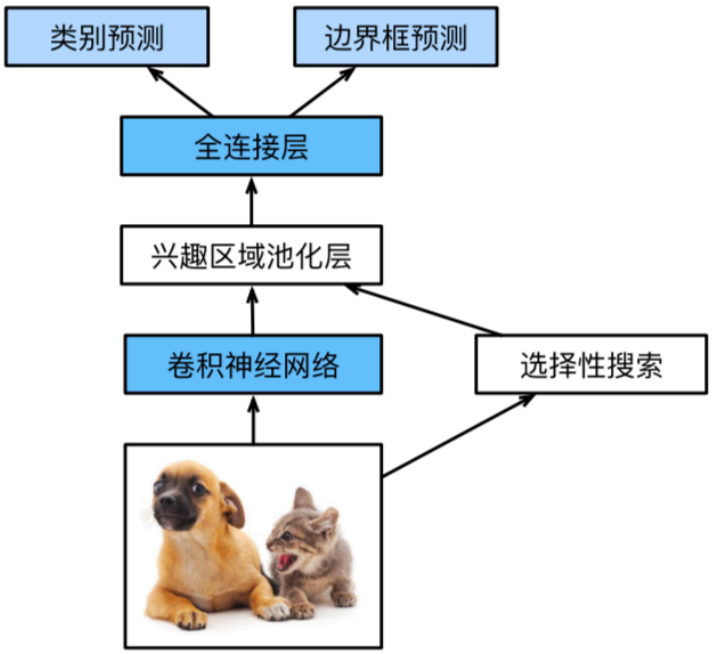


图 9.6: Fast R-CNN模型

与 R-CNN 区别：

1. 提取特征的神经网络为整个图像
   当输入为一张图片时，输出形状为 $(1, c, h_1, w_1)$
2. 选择性搜索 n 个提议区域，
3. 兴趣区域池化层 输出 (n, c, h_2, w_2)
   可指定池化窗口任意矩形区域作为一个池化结果的源，所以可产生任意形状池化结果
   例：$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$ 可取 $\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$ 和 $\begin{bmatrix} x_{31} & x_{32} \end{bmatrix}$ 做输出 y_{11} 和 y_{21} 的来源
4. 全连接层 输出变为 (n, d)，d 为超参数
5. 预测类型使用 $softmax$，输出 (n, q)
   预测边界框时，输出 $(n, 4)$

图像分割

区分图片中一个物体的不同部分，不将每一物体和标签对应

实例分割

区分一个像素属于哪一物体，即使两物体为同一标签下的实例

semantic segmentation 语义分割

图像预处理：不使用拉伸，仅适用随机图片裁剪，裁剪得到图片一块固定形状的小图，作为训练集

FCN 全卷积网络

结构：

1. 使用卷积神经网络 抽取图像特征

2.(1, 1) 卷积层 通道数变为类别个数

3. 转置卷积层 将图像的高宽变为输入图像的尺寸，作用仅为将 2. 中的图片分类反卷积得到图片表示，所以输入通道数 = 输出通道数 = 类别个数

例：

1.ResNet-18 预先训练神经网络抽取图像特征 丢弃最后全局平均池化层 + 全连接层

2.(1, 1) 卷积层，通道数 = 类别数 n

3. 转置卷积层将 (1, 1) 卷积层每一通道的输出转换回图片大小的矩阵，每一元素为类别对应像素的 index

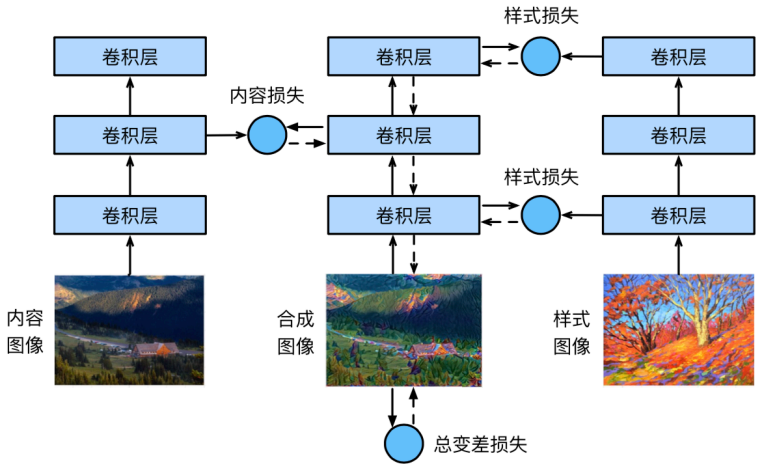
输出图片：

1. 对一个图片的 n 个通道中结果 合并为唯一矩阵，新矩阵每一元素 = n 个矩阵中对应位置最大值

即 输出的 n 个矩阵代表图像符合每一类别的区域，对一个位置的元素取最大值即判断此位置最优先属于哪一类别

2. 替换 1. 中输出的矩阵元素，每一元素替换为此位置的 RGB 向量

样式迁移



得到 内容图像 样式图像，输出合成图像

图像预处理：将输入图片在 RGB 3 通道内分别做标准化

图像后处理：将输出矩阵标准化的值 转回 像素值

抽取特征：

使用 VGG-19 预训练参数抽取图像特征

选择 VGG 靠近输出的层，称内容层，避免保留过多内容图像的细节

选择多个中间层的输出匹配样式，称样式层

损失函数

损失函数为内容 样式 总变差损失函数的加权和

内容损失函数：

使用平均平方代价函数，计算 合成图像 和 内容图像 在 内容特征上的误差

输入张量为 内容图像和合成图像 通过抽取特征后内容层的输出

样式损失函数：

输入张量为样式层输入

对每一样式层：

1. 将 c 通道 (h, w) 的样式层输出转为 (c, hw) 的矩阵 X

2. 计算 X 的 Gram matrix, XX^T。元素 $(XX^T)_{ij}$ 即 $x_i \cdot x_j$，代表通道 i 和 j 的相关性。可对 Gram matrix 每一元素先除以 $|X|$ 元素值，避免样式损失值过高

3. 传入内容图像 合成图像，对每一样式层输出做平均平方代价函数，每层代价相加为总样式损失值

总变差损失：

输入仅为合成图像，对合成图像使用总变差降噪

$J = \sum_{i,j} |x_{i,j} - x_{i+1,j}| + |x_{i,j} - x_{i,j+1}|$

即 对每一合成图片的像素，求其和右方 下方像素的差值。最后求和

样式迁移模型：

使用每一 VGG 块的第一卷积层做样式层，第四卷积块的最后一卷积层做内容层

更改的参数仅有生成的图片，不对网络参数进行更改

10 自然语言识别

一. 词嵌入

词赋予两个向量 v_o, v_c，分别为将词作为中心词和作为背景词使用的词向量

词间夹角 $\frac{u_o^T v_c}{||u_o|| ||v_c||} \in [-1, 1]$ 为词的相似度

定义：

S：词序集合，包含多个词序，包含重复的词

$\mathcal{V}$：所有词的索引集合，即 $nub(S)$

w_i：一个词，索引为 i

v_i：词 w_i 对应的实数向量

向量夹角余弦值 $\frac{x^T y}{||x|| ||y||}$ 代表两个词的关联程度

背景窗口大小 n：定义对一个中心词 w_i，背景词取值范围 $=$ [中心词 $-n$, 中心词 $+n$]。

范围为词序内位置关系，不是索引序号关系。背景词和中心词必须在同一词序内

当中心词一侧背景词个数 $l < n$，填充 $n - l$ 个无效词，随后由掩码舍弃不进入代价函数计算。

给定中心词 w_c，有单个背景词 w_o 的概率 $P(w_o|w_c) = \frac{exp(u_o^T v_c)}{\sum_{i \in \mathcal{V}} exp(u_i^T v_c)}$

$log(P(w_o|w_c)) = u_o^T v_c - log(\sum_{i \in \mathcal{V}} exp(u_i^T v_c))$

跳字模型：

基于一个**中心词**生成才其周围的多个**背景词**

似然函数：代表一段词序发生的可能性

对中心词 w_c,产生所有窗口内的背景词的概率为 (假设背景词间 $independent$)$P(c) = \prod_{c-n<j<c+n, j \neq c} P(w_j|w_c)$

(假设每一中心词产生背景词的可能性 independent) 产生整个词序 S 的可能性：

$P(S) = \prod_{0 \leq c \leq |S|} P(c) = \prod_{c=0}^{|S|-1} \prod_{j=c-n, j \neq c}^{c+n} P(w_j|w_c)$

代价函数：基于似然函数，高似然代表低代价函数值

$J = -log(P(S)) = -\sum_{c=0}^{|S|-1}\sum_{j=c-n,j\neq c}^{c+n} log(P(w_j|w_c))$

求导：$\frac{dJ}{dv_c} = -\sum_{c=0}^{|S|-1}\sum_{j=c-n,j\neq c}^{c+n}\frac{dlog(P(w_j|w_c))}{dv_c}$

对于每一中心词 v_c 有 $\frac{dP(w_j|w_c)}{dv_c} = u_o^T - \sum_{j\in\mathcal{V}} P(w_j|w_c)u_j$

连续词袋模型

与跳字模型不同处：中心词由背景词产生，和跳字模型相反

给定背景词 $W_o = w_{o_1}, w_{o_2}, ..., w_{o_{2m}}$，中心词 w_c 有出现概率：

$P(w_c|w_{o_1}, ..., w_{o_{2m}}) = \frac{exp(\frac{1}{2m}u_c^T(v_{o_1}+...+v_{o_{2m}}))}{\sum_{i\in\mathcal{V}} exp(\frac{1}{2m}u_i^T(v_{o_1}+...+v_{o_{2m}}))}$

$= \frac{exp(u_c^T\bar{v}_o)}{\sum_{i\in\mathcal{V}} exp(u_i^T\bar{v}_o)}$

1. 使用 `postierior estimate` 计算得到，忽视 $P(w_c)$ 项

2.$\frac{1}{2m}$ 项为额外添加，使 $\bar{v}_o = \frac{(v_{o_1}+...+v_{o_{2m}})}{2m}$

3. 背景词使用背景向量 w_o 而非 w_c

似然函数：

$\prod_{c\in\mathcal{V}} P(w_c|W_o)$

代价函数：基于似然函数，高似然代表低代价函数值

$J = -log(\prod_{c\in\mathcal{V}} P(w_c|W_o))$

$= -\sum_{c\in\mathcal{V}}(u_c^T\bar{v}_o - log(\sum_{i\in\mathcal{V}} exp(u_i^T\bar{v}_o)))$

求导：$\frac{dJ}{dv_{o_i}} = -\frac{1}{2m}(u_c - \sum_{j\in\mathcal{V}} P(w_j|W_o)u_j)$

近似训练：对跳字模型和词袋模型的优化

避免每次梯度计算都包含词典大小的项数计算

负采样

定义：

背景词 w_o 出现在 w_c 背景窗的概率为 $P(D=1|w_c, w_o) = \sigma(u_o^T v_c)$

σ 为 sigmoid 激活函数，$D=1$ 代表事件发生，$D=0$ 代表未发生

更改 $P(w_o|w_c)$ 定义：$P(w_o|w_c) = P(D=1|w_c, w_o)\prod_{k=1,w_k\sim P(w)}^{K} P(D=0|w_c, w_k)$

根据 $P(w)$ 分部取 K 个反样本，称噪声词，反样本不能为背景词

根据 word2vec 论文，选择每一噪声词 w 的几率为 w 出现几率的 0.75 次方

加入反样本的原因：若最大化 $P(D=1|w_c, w_o) = \sigma(u_o^T v_c)$，则所有词向量方向相同且长度极大

代价函数：$-log(P(w_o|w_c)) = -log(\sigma(u_o^T v_c)) - \sum_{k=1,w_k\sim P(w)}^{K} log(1-\sigma(u_k^T v_c))$

$= -log(\sigma(u_o^T v_c)) - \sum_{k=1,w_k\sim P(w)}^{K} log(\sigma(-u_k^T v_c))$ **层序 softmax**

对整个词典有二叉树，每一分支节点 i 有背景词向量 u_i，每一叶节点对应一词

定义：

$L(w)$：w 在二叉树中的深度，包括根节点和叶节点

$n(w,i)$：从根节点到 w 叶节点的路径上第 j 个节点，有背景词向量 $u_{n(w,i)}$

w 在此为背景词，非中心词

一节点只有唯一背景词向量，当一节点出现在多个路径上时背景词向量被共享

更改 $P(w_o|w_c) = \prod_{j=1}^{L(w_o)-1}\sigma(isLeftChild(n(w_o, j+1)))\cdot u_{n(w_o,j)}^T v_c)$

$$isLeftChild(x) = \begin{cases} 1 & x < 0 \\ -1 & otherwise \end{cases}$$

对固定中心词 w_c，所有词的几率和为 $\sum_{o\in\mathcal{V}} P(w_o|w_c) = 1$

证明：选择任意节点 k 使得其左右子节点 i, j 都为叶节点

$P(w_i|w_c) + P(w_j|w_c) = \prod ... \cdot (\sigma(x) + \sigma(-x))$

$= \prod ... \cdot 1$

即，任意仅有 2 页子节点的节点，子节点可能性和都为 1。循环合并子节点，最终得到跟节点可能性为 1

二次采样

作用：对于一个词 w_0，和低频词同时出现的情况 比 和高频词同时出现 对模型训练更加有用

实现：

取样的背景词中每个词有 $P(w_i) = max(1 - \sqrt{\frac{t}{f(w_i)}}, 0)$ 几率被丢弃

$f(w_i) =$ 出现 w_i 个数 / 总词数，即词 w_i 在整个数据集中出现频率

t 为超参数

对每个 $w \in S$，使用二次采样随机丢弃 w，创建新的 S' 作为训练集

word2vec 词嵌入模型实现

目标：**得到每一词的词向量，使有关联的词间向量夹角最小**

嵌入层

得到词嵌入的层，输入词 w_i 索引 i，输出权重矩阵第 i 行作为词向量

嵌入层有权重矩阵，形状（词典大小，每个词向量纬度）

前向计算

输入：中心词索引矩阵 $\begin{bmatrix}[c_1]\\ ...\\ [c_b]\end{bmatrix}$ + (背景词, 噪声词) 索引矩阵 $\begin{bmatrix}[o_{11}, ..., o_{1n}, q_{11}, ..., q_{1k}]\\ ...\\ [o_{b1}, ..., o_{bn}, q_{b1}, ..., q_{bk}]\end{bmatrix}$

b 为批量大小

n 为窗口大小

k 为噪声词数量

两个输入矩阵元素皆为常数，非向量

1. 通过嵌入层变换为 中心词向量张量（b, 1, 词向量长） $\begin{bmatrix}[u_{c_1}]\\ ...\\ [u_{c_b}]\end{bmatrix}$ 背景噪声词向量张量（b, n+k, 词向量长） $\begin{bmatrix}[v_{o_{11}}, ..., v_{o_{1n}}, v_{q_{11}}, ..., v_{q_{1k}}]\\ ...\\ [v_{o_{b1}}, ..., v_{o_{bn}}, v_{q_{b1}}, ..., v_{q_{bk}}]\end{bmatrix}$

2. 对两张量小批量相乘，即得到（b, 1, n+k）张量 $\begin{bmatrix}u_{c_1}^T[o_{11}, ..., o_{1n}, q_{11}, ..., q_{1k}]\\ ...\\ u_{c_b}^T[o_{b1}, ..., o_{bn}, q_{b1}, ..., q_{bk}]\end{bmatrix}$

乘法为矩阵乘法，行向量 $u_{c_1}^T$ 乘 矩阵 $[o_{11}, ..., o_{1n}, q_{11}, ..., q_{1k}]$

输出张量 (b, 1, k+n)，每一元素为背景词 噪声词和中心词的点乘，对应计算词向量夹角

代价函数

使用 Sigmoid 二元交叉熵函数，传入前向传播结果 P，label$L$ 代表样本是否为正样本，掩码 M 代表样本是否有效

目标为使预测结果 P 每一有效位置和 L 对应位置相同

fastText 子词嵌入：基于 word2vec 优化

产生子词：在单词首尾添加 < >，取所有长度为 s 的子字符串 + 单词本身

如，对单词'where' 有子词 {'<wh', 'whe', 'her', 'ere', 're>', '<where>'}

每一子词都存在词典中，单个词对应的向量为子词向量和

GloVe 全局向量的词嵌入：基于 word2vec 优化

对每一中心词 w_i，合并所有 w_i 的背景词，称多重集 $\mathcal{C}_i$。多重集直接合并原集合，不删除重复项

一个背景词 w_j 在 $\mathcal{C}_i$ 中的出现次数称 重数，记做 x_{ij}

定义 $x_i = |\mathcal{C}_i|$，为多重集集合大小

代价函数 1：$J = -\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{V}} x_{ij} log(P(x_j|x_i))$

$= -\sum_{i\in\mathcal{V}} x_i \sum_{j\in\mathcal{V}} \frac{x_{ij}}{x_i} log(P(x_j|x_i))$

第二层求和即 实际样本中 w_j 出现在 w_i 背景集合中的概率 $\frac{x_{ij}}{x_i}$ 和 模型预测的出现概率 $P(x_j|x_i)$ 的交叉熵

* 较少使用，计算开销较大。并包含大量生僻词，导致预测准确率较低

代价函数 2：使用平方代价函数 $J = \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{V}} h(x_{ij})(u_j^T v_i + b_i + c_j - log(x_{ij}))^2$

$h(x_{ij})$ 为权重，在 $[0,1]$ 单调递增

例：

$$h(x) = \begin{cases} 0 & x = 0 \\ (x/c)^{0.75} & x < c \\ 1 & x > c \end{cases}$$

c 可取 100

b_i 为中心词偏差项，c_j 为背景词偏差项

$u_j^T v_i$ 即 x_{ij}，$log(x_{ij})$ 即 $P(x_j|x_i)$ 除去分子

GloVe 每一词的背景词向量和中心词向量相同，由于每对词互为背景词。最终每一词的背景词向量 = 中心词向量 = 所求的 $u_c + u_o$

预测 $P(w_i|w_j)$ 公式不变

证明代价函数

针对词向量，定义函数 $f(u_i, u_j, u_k)$ 使得对一个词 w_k，其余两个词相对出现的几率 $\frac{P(w_i|w_k)}{P(w_j|w_k)} \approx f(u_i, u_j, u_k)$

函数 $f(u_i, u_j, u_k)$ 可定义为标量函数 $g((u_i - u_j)^T u_k)$

$g(x)$ 需满足 $g(x)g(-x) = 1$，则 g 可取 $g(x) = exp(x)$

则需要 $\frac{exp(u_i^T u_k)}{exp(u_j^T u_k)} \approx \frac{P(w_i|w_k)}{P(w_j|w_k)}$

则 $P(w_i|w_k) = \frac{x_{ki}}{x_k} = \alpha \cdot exp(u_i^T u_k)$

$u_i^T u_k = log(\alpha) + log(x_{ki}) - log(x_k)$

中心词偏差项 b_k 和背景词偏差项 c_i 之和 $b_k + c_i$ 模拟 $-log(\alpha) + log(x_k)$

则最小化 $u_i^T u_k + b_k + c_i - log(x_{ki})$，使用平方代价函数

求近义词：word2vec 应用

使用 KNN，在训练结束的词向量中寻找余弦值最小的 K 个向量

求类比词：word2vec 应用

给定词 w_a, w_b, w_c，求 w_d 使得 $w_a : w_b$ 关系类似 $w_c : w_d$

算法：使用 KNN，寻找向量临近 $v_b - v_a + v_c$

二．文本情感分类

分析文本作者的情感

输入：多组 词序长度 `n` + 标记的情感类型

使用循环神经网络：

1. 使用预先训练的嵌入层得到词向量

2. 双向循环神经网络

每一隐藏状态分别有形状 (批量大小，2 * 隐藏单元个数)，*2 由于为双向网络

3. 全连接层，得到分类的情感

输入为双向循环神经网络 第一和最后一隐藏状态 连接后的张量，有形状 (批量大小，4 * 隐藏单元个数)

使用卷积神经网络 textCNN

1. 使用预先训练的嵌入层得到词向量，即输入形状为 (词向量长度 d，词数 L)

2. 卷积计算：多个一维卷积核 k_i，每一卷积核为 (输出通道个数 c_i, 词向量长度 d，卷积核宽 w_i) 形状的矩阵

对一个卷积核一通道的计算，将每一卷积核窗口内的输入按元素相乘求和。输出形状 (1, 词数 - 核宽 + 1)

总输出为一列表矩阵，有形状 $[(c_0, L-w_0+1), (c_1, L-w_1+1), ...]$

同一维卷积核有唯一核宽长度相同，不同一维卷积核可使用不同核宽

3. 时序最大池化层：类似一维全局最大池化层

针对单一一维卷积核的输出 $(c_i, L-w_i+1)$，对 c_i 个输入通道各得出 $L-w_i+1$ 时序内最大值。即输出一向量，长度为 c_i

4. 将所有时序最大池化层结果 `concat`

5. 全连接层，将 `concat` 结果分类为情感

seq2seq 编码器-解码器

对不定长输入 $[x_1, ..., x_T]$ 允许输出为不定长序列 $[y_1, ...y_{T'}]$，使用定长的背景向量 c 做连接 解码器-编码器 的中间向量

定义：输出词集合为 $\mathcal{Y}$，其中包含一个 $<eol>$ 特殊词代表词序末尾

编码器：将不定长输入序列变为定长背景变量 c

对每一时间步 i，字符 w_i，使用循环神经网络隐藏层计算隐藏状态 h_i。即输出 $[h_0, ..., h_T]$

长度为 T 序列 s_i 有背景向量 $c = q(h_1, ..., h_T)$。q 为自定义函数

解码器：根据前一输出序列和 c，输出结果序列

根据解码器隐藏状态 $s_{t'} = g(y_{t'-1}, c, s_{t'-1})$ 计算 $P(y_i|y_1, .., y_{i-1}, c)$

对一个序列的输出，总可能性为 $P(y_1, ..., y_{T'}|x_1, ..., x_T)$。此值目标为此值最大化

代价函数

$-log(P(y_1, ..., y_{T'}|x_1, ..., x_T)) = -log(\prod_{i=1}^{T'} P(y_i|x_1, ..., x_T))$

$= -\sum_{i=1}^{T'} log(P(y_i|x_1, ..., x_T))$

解码器算法：

贪婪搜索：每一 $y_i = argmax_{y \in \mathcal{Y}} P(y|y_1, ..., y_{i-1})$

即每次取 y_i 使得 $[1, i]$ 范围内 P 值最高，直至 $i = T'$

y_{i-1} 对 y_i 的取值造成影响，若最优解 y_i 位置的词非第 i 时间步的 $argmax$ 则贪婪算法无法取到最优解

beam search 束搜索

定义：beam size 束宽 k

计算：

1. 时间步 $i=1$，选取条件概率最大 k 个值，做 k 个可选序列的首词

2. 时间步 $i=2,3...,L$，对每一可选输出序列考虑整个 $\mathcal{Y}$，将 $k*\mathcal{Y}$ 个新时间序列看做一整体，选出 P 值最高的 k 个作为此序列的下一词。最终每一时间步的输出序列都作为可选序列进入第三步，共 $L*K$ 个时间序列。

即，永远保持 k 个序列，而一个时间序列可能添加不同的 $\hat{y}$ 而在下一时间步预测中有多个分支。

3. 将 $L*k$ 个序列筛选，仅保留包含 $<eol>$ 的词序，舍弃 $<eol>$ 后的词

4. 在 3. 的序列集合中选择值 $\frac{1}{l^a}log(P(y_1,...,l))=\frac{1}{l^a}\sum_{i=1}^{l}log(P(y_i|y_1,...,y_{i-1},c))$ 最大序列作为输出

l 为每一序列长度，a 为参数，常取 0.75。$\frac{1}{l^a}$ 惩罚长度较大项

注意力机制

编码器对每一时间步产生背景向量 c_i' 而非对整个词序产生唯一背景向量 c

计算背景变量 $c_{t'}$：

计算每一编码器隐藏状态 h_t 的权重 $\alpha_{t't}=\frac{exp(e_{t't})}{\sum_{t=1}^{T}exp(t't)}$，即对不同 t 的 $e_{t't}$ 求 softmax

$e_{t't}$ 同时基于编码器时间步 t 和解码器时间步 t'，则可定义 $e_{t't}=a(s_{t'-1},h_t)$。

a 函数例：当 $s_{t'-1},h_t$ 长度相等，$a(s_{t'-1},h_t)=s_{t'-1}^{T}h_t$

注意力机制论文：$a(s_{t'-1},h_t)=v^Ttanh(W_ss+W_hh)$，$v,W_s,W_h$ 为可学习参数

对所有时间步内的编码器隐藏状态 h_t 求加权平均，即背景变量 $c_{t'}=\sum_{t=1}^{T}\alpha_{t't}h_t$

计算解码器隐藏状态：(类似 GRU 算法，加入背景变量项)

$s_{t'}=z_{t'}\odot s_{t'-1}+(1-z_{t'})\odot\tilde{s}_{t'}g(y_{t'-1},c_{t'},s_{t'-1})$

重置门 $r_{t'}=\sigma(W_{yr}y_{t'-1}+W_{sr}s_{t'-1}+W_{cr}c_{t'}+b_r)$

更新门 $z_{t'}=\sigma(W_{yz}y_{t'-1}+W_{sz}s_{t'-1}+W_{cz}c_{t'}+b_z)$

候选隐藏状态 $\tilde{s}_{t'}=tanh(W_{ys}y_{t'-1}+W_{ss}(s_{t'-1}\odot r_{t'})+W_{cs}c_{t'}+b_s)$

强制学习 teaching forcing：对自然语言人工智能通用

当预测第 i 时间步 y_i 时使 y_{i-1} 为样本实际上一 label，而不是上一次预测的计算结果 $\hat{y}_i$

11 推荐算法

12 Deep Feedforward Network (Deep Learning 第 6 章笔记)

SVM 支持向量机

仍通过 w^Tx+b 得到输出，输出仅表示 identity，正值说明有 identity，负值说明没有

依据：一个平面的公式为 $\beta_0+\beta_1x_1+\beta_2x_2=0$，则当计算 w^Tx+b 得到值后，>0 则为平面上方的数据点，<0 为下方数据点

kernel trick

kernel method 将数据集表示成相近的两个数据点一组的集合 (x_i,x_j)，kernel method 将一对数据变为单一数据点 $x=k(x_i,x_j)=\langle\phi(x_i),\phi(x_j)\rangle$

kernel method 使用 ϕ 转换数据的纬度，而点乘化简后无需先计算 $\phi(x_i), \phi(x_j)$ 即可得到新数据点 x

manifold hypothesis:

当训练数据集合包含大量无规律的数据，则将其中大部分视为无效数据，并只关心落在一个 manifold 上的数据。

例：生成图像文字声音时数据大多很集中，当像素文字随机分布时生成图像大多无意义

deep feedforward network/feedforward neural network/multilayer perceptrons MLP：

找到 θ 使得 $f(x;\theta)$ 最接近数据 y 值。f^* 为最理想的 f，即 $f^*(x) = y$。θ 可为多个参数，如 $f(x;w,b) = x^T w + b$

$f^*(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$，$f^{(1)}$ 为 network 第一层。每一 $f^{(i)}(x) = \phi(x;\theta)^T w$

神经网络

1. 结构：

   输入层没有 weight，第一 hidden layer 得到所有输入层的值。

   hidden layer 和输出层所有输出都为 0/1，非连续的值

2. 一层 hidden layer 计算方法：$f^{(i)}(x;W,c) = \sigma(W^T x + c)$

   x 为前一层的输出向量，输入层 x 即为训练参数向量。

   c 为此层常数向量

   z = $W^T x$ 为一层 hidden layer 对输入取得的中间值向量，称 logit。a = $\sigma(z + c)$ 为对 z + c 每一元素取 σ 的结果向量，a 即此层的输出。

   W 为此层参数矩阵，行数 = 前层节点数，列数 = 当前层节点数

   X 为多个参数点的训练集中前一层的输出矩阵，行数为数据点个数，列数为前一层节点数

   XW 当 W 对参数集矩阵操作时，每行向量 z_i^T 此时为一层 hidden layer 各节点对第 i 参数点的中间值向量。对每行 $+c^T$ 并分别取 σ 得到输出矩阵，a_{ij} 为当使用第 i 个参数点时此层第 j 节点的输出

cross entropy

分部 p 和分部 q 间的 cross entropy $H(p,q) = -E_p(\log(q))$。为 expected value of $log(q)$ with respect to distribution p

cost function

当使用 maximum likelihood 估计参数时，cost function$J(\theta)$ 为训练输入参数的分部和训练结果参数的分部间 cross-entropy：$J(\theta) = -E_{x,y\sim training\_dataset}(log(p_{model}(y|x)))$

对于每一在训练集内的 (x, y)，求 $log(p_{model}(y|x))$，并求 expected value。$p_{model}(y|x)$ 即训练得到的 y 关于 x 的分部

例：当 model 为 $y = N(f(x;\theta), 1)$ 正则分部时，$J(\theta) = -E_{x,y\sim data}(y - f(x;\theta))^2 + const$

output layer

当输出层的结果和不为 1 时，代表数据没有被准确分到某一类中，使用 exponentiation and normalisation

normalisation 后结果 $p = \frac{\tilde{p}}{\sum \tilde{p'}}$，为 $\tilde{p}$ 在所有结果中占的比例。$\tilde{p}$ 为未 normalise 值

假设输出层结果 $\tilde{P}(y|x)$ 有 $log(\tilde{P}(y|x)) = yz$

$\tilde{P}(y|x) = exp(yz)$

$P(y|x) = \frac{exp(yz)}{\sum_{y'=0}^{1} y'z}$，称 **softmax function**

$P(y|x) = \sigma((2y-1)z)$，y，y' 为训练目标结果，所以 $\sum_{y'=0}^{1}$ 包含所有 y'

对 softmax function 使用 log likelihood 原因：log $softmax(z)_i = z_i - log\sum_j exp(z_j)$。

当 z_i 为 dominant，并对应期望的输出项。log $softmax(z)_i = 0$。则此项不产生高 cost，否则产生 cost。

hidden unit

代表一个 hidden layer 节点的激发函数。

1. rectified linear unit: $g(x) = max(0, x)$

   无法用于 gradient based learning，由于一阶导为 0

   基于 rectified linear unit 的优化：$g(x) = max(0, x) + a * min(0, x)$

   a = -1：absolute value rectifier

   a 为极小值：leaky ReLU

   a 为可学习值：Parametric ReLU, PReLU

2. Maxout units

   将 x 分为多组，每组 h(x) 为组内最高值

backward propagation

一种计算 gradient 的方法，区别于使用 gradient 进行学习的 stochastic gradient descent 算法：

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \ldots, 1$ **do**
  Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
  Compute gradients on weights and biases (including the regularization term, where needed):
  $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
  $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
  Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$
end for