

# 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  $\eta$ 。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上一层所有节点连接

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

**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:  $H(y^{(i)}, \hat{y}^{(i)}) = -\sum_{j \in B} y^{(i)} \log(\hat{y}^{(i)})$

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

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

## 2 linear regression 线性回归

**平方代价函数**:  $J(\theta) = \frac{1}{n} \sum_{i=1}^n J^{(i)}(\theta)$ , 为所有样本误差的平均值

**迭代**:  $\theta_i = \theta_i - \frac{\eta}{|B|} \sum_{i \in B} \frac{dJ^{(i)}(\theta)}{d\theta_i}$ , 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:

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

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

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

**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 代价函数

过拟合问题

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

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

2. **丢弃法**

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

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**:  $x' = \frac{x-\mu}{\sigma}$ , 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

**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, 调用优化函数。

- 即每一 batch 使用相同的权重偏差。一次训练一共只遍历一次所有 sample, 共  $\frac{\text{sample\_size}}{|B|}$  次向量化计算

**activation function** 只出现在 hidden layer 的输出, 输出层无需 activation function

### 3 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  $\lfloor \frac{\text{kernel}}{2} \rfloor$  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  $(c_o, c_i, \text{行数}, \text{列数})$

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

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 10, 使用 sigmoid 激活函数

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 组包含 pooling 的卷积计算

```
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 组仅 pooling 一次的卷积计算, 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 步幅池化层

VGG 神经网络由数个 VGG 块 + 数个全连接层组成

例: **VGG-11**

包含结构为 (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  
和 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，都有同样通道数  
自定义卷积层可设置 kernel，步幅，填充。(1, 1) 除了通道数可设置，其余固定为默认值  
NiN 神经网络有多组 (NiN 块 + 池化层) 每一 NiN 块后有池化层

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() // 将四维的输出转成二维的输出，其形状为 (批量大小, 10)
```

## GoogLeNet 含并行结构神经网络

### Inception 块

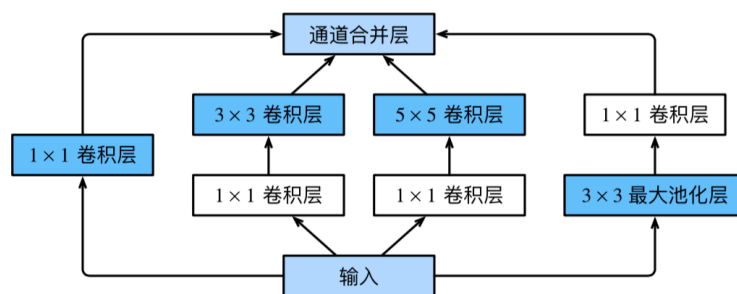


图 5.8: Inception 块的结构

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，每一模块间使用步幅 2 (3, 3) 窗口 1 填充池化层
1. 64 通道 (7, 7)kernel 2 步幅 3 填充卷积层 + 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 + 池化层
3. 串联 2 inception 块 + 池化层，分别有结构
  - (64, (96, 128), (16, 32), 32)
  - (128, (128, 192), (32, 96), 64)
4. 串联 5 inception 块 + 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

## 4 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  $= \phi(BN(x))$

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

$\sigma, \mu$  为此通道一批量内所有矩阵的所有元素的总体方差，平均值

得到  $\sigma, \mu$  后对此通道此批量内所有元素求标准化

最终对此通道每一 sample 标准化的结果拉伸偏移

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)

(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

## 5 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  $(w_1, w_2, \dots, w_T)$  的出现可能性为

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

称 n 元语法，每一  $w_t$  为一时间步中出现的词

### 循环神经网络

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

$H_{t-1} W_{hh}$  项将隐藏层前一次输出纳入此次计算

输出  $O = H W_{hq} + b_q$

**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

**采样方式**：

$BATCH\_SIZE$  每次采集的样本数

$NUM\_TEPS$  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

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

### SVM 支持向量机

仍通过  $w^T x + b$  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  $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$ , 则当计算  $w^T x + b$  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  $\phi$  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:

当训练数据集包含大量无规律的数据, 则将其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  $\theta$  使得  $f(x; \theta)$  最接近数据  $y$  值。  $f^*$  为最理想的  $f$ , 即  $f^*(x) = y$ 。  $\theta$  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  $\sigma$  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  $\sigma$  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:  $J(\theta) = -E_{x, y \sim \text{training\_dataset}}(\log(p_{\text{model}}(y|x)))$

对于每一在训练集内的  $(x, y)$ , 求  $\log(p_{\text{model}}(y|x))$ , 并求 expected value。  $p_{\text{model}}(y|x)$  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,  $J(\theta) = -E_{x, y \sim \text{data}}(y - f(x; \theta))^2 + \text{const}$

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

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

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

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

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

当  $z_i$  为 dominant, 并对应期望的输出项。  $\log \text{softmax}(z)_i = 0$ 。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:
 $\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$ 
for  $k = l, l-1, \dots, 1$  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):
     $\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot f'(\mathbf{a}^{(k)})$ 
    Compute gradients on weights and biases (including the regularization term,
    where needed):
     $\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\theta)$ 
     $\nabla_{\mathbf{W}^{(k)}} J = \mathbf{g} \mathbf{h}^{(k-1)\top} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\theta)$ 
    Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
     $\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = \mathbf{W}^{(k)\top} \mathbf{g}$ 
end for

```

---