作业 3

211300024 石睿

1，下面是计算一个 layer 的“forward propagation”操作的函数，这一层使用了 sigmoid activation 函数，请找出代码的错误并写出修正的结果

```
import numpy as np

def forward_prop(W, a_prev, b):
    z = W*a_prev + b
    a = 1/(1+np.exp(-z)) #sigmoid
    return a
```

答案：

由神经网络 forward propagation 操作可知，如若抽象成一个函数，则其中的 W 和 a_prev 应分别是一个向量，b 是一个标量。而向量间点积应使用矩阵乘法操作符“@”。综上，代码应修改成

```
import numpy as np

def forward_prop(W, a_prev, b):
    z = W @ a_prev + b
    a = 1/(1 + np.exp(-z))
    return a
```

2，请解释 Momentum 相比于标准的 Gradient Descent, 它是如何加速学习过程的，并解释它在优化求解中的优势/作用？

答案：

$$m_t = \gamma \cdot m_{t-1} + \alpha \cdot \frac{\partial L}{\partial w_t}$$

$$w_{t+1} = w_t - m_t$$

本式即引入 Momentum 后的参数更新情况。其中 m 是动量，同时受到历史动量以及当前梯度的影响。下标 t 表示是 t 时刻的动量 m 和参数 w，γ 代表对历史动量信息的参考程度，一般取 0.9。

（1）加速

1.1：t 时刻动量可以不断累积 t 时刻之前的动量信息，期望情况下动量会逐步增大（同时取决于当前梯度）

1.2：相比于标准的梯度下降算法，如若遇到函数较平坦的位置（梯度很小），则更新较慢。但在动量法中，尽管遇到函数较为平坦处，但仍有相对可观的动量可以更新参数

（2）优势

2.1 稳定性高：其他批次的梯度信息或本次更新中的历史信息，使参数更新更接近真实的梯度，可以增加稳定性。同时如果梯度方向发生改变（和动量方向相反），参数更新幅度下降，可以有效避免参数更新时的震荡而不收敛的困境。

2.2 跳出局部最优解：相比于标准的梯度下降算法，如若遇到局部最小值以及函数波谷则很难跳出。但动量法中在梯度方向不正确（朝局部最优方向），依靠动量的正确方向有可能跳脱出局部最优解。

2.3 加速学习过程：由（1）中分析

3，有人认为“single hidden layer”的神经网络已经足够逼近任何连续的函数，那我们为什么很多时候却会使用“multiple layers”的神经网络？请分析两种设计的优缺点。

答案：

3.1

神经网络“万有逼近性”在理论上来说是可以以单层 hidden layer，加以无限多的神经元和具有 squeeze 性质的激活函数，来逼近任何连续函数，也即多层感知机 MLP 可以拟合任意的函数。

但实际上，这样做出来效果不佳，因为受限于两个条件：

① 优化算法：基于反向传播的优化算法可能无法找到单层 hidden layer 网络的最优情况

② 拟合情况：为了拟合目标函数，单层神经网络可能会非常大，其参数 features 很多，由 Hoeffding’s inequality 可得，在 feature 很多的神经网络中，很容易 overfitting。

3.2

① 单 hidden layer：

- 优点：实现简单、在限制神经元一定数量上限的情况下，反向传播更快（不需要链式法则不断回溯），训练速度更快
- 缺点：
  - （限制神经元数量时）可表示的函数集合很小，只能对简单函数近似；无法通过多层网络来逐层抽取复杂特征
  - （不限制数量）缺点如 3.1 所述

② 多 hidden layers：

- 优点：
  - 可表示的函数集合很大，在神经网络架构设置正确的前提下，可以逼近很复杂的函数；
  - 使用多个隐层网络可以对复杂特征逐步拆解来学习（如 deep VAE 中每层的“code”通常分别代表这输入的一些特征）
  - 和通用的优化求解过程契合，反向传播梯度下降会更适合多层神经网络
- 缺点：
  - 梯度消失或爆炸：求梯度涉及链式法则，随着网络的深度增加，每层特征值分布会逐渐的向激活函数的输出区间的上下两端（激活函数饱和区间）靠近，长此以往则会导致梯度消失。如使用 sigmoid 函数，在层数很深，链式法则链很长的情况下很容易出现导数趋向于 0，进而导致训练停滞。可以通过更换激活函数为 ReLU 或 tanh、梯度裁剪、batch nomorlization 等方式来解决
  - 过拟合：参数 features 很多，由 Hoeffding’s inequality 可得，在 feature 很多的神经网络中，很容易 overfitting。可以通过早停、加入结构化风险正则化项等方式来解决。

4，请简要描述 batch normalization，以及在 standard normalization 后为什么要做“scaling” 和“shifting” ？

答案：

4.1

批归一化依靠两次连续的线性变换，希望转化后的数值满足一定的分布（基于标准正态分布变换），不仅可以加快了模型的收敛速度，也一定程度缓解了特征分布较散的问题。

因为随着网络的深度增加，每层特征值分布会逐渐的向激活函数的饱和区间靠近，长此以往则会导致梯度消失。BN 通过将该层特征值分布重新拉回标准正态分布，特征值将落在激活函数对于**输入较为敏感的区间（如 sigmoid 函数输入靠近 0 的小区间）**，输入的小变化可导致损失函数较大的变化，使得梯度变大，避免梯度消失，同时也可加快收敛。

主要可以分为以下四步：

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Mini-batch mean 和 mini-batch variance 计算每个 batch 上样本均值和方差
- Normalize：对每个样本进行正则化操作，转换为符合 N(0,1) 分布的情况

- Scale and shift：这是最重要的一步，引入两个需要神经网络自己学习的参数 gamma（负责放缩）和 beta（负责平移）。因为正则化后样本只符合 N(0,1)，和输入特征分布未必符合，也没有达到使输入靠近激活函数饱和区间的目的。如若此时继续训练，会使网络表达能力下降。

4.2

引入 scale 和 shift 有以下原因：

- 还原原始数据分布特征，恢复表达能力：如 4.1 所述。此外可以降低内部协变量偏移，在恢复到原有数据分布特征时，有一定程度放大缩小的自由度，可以获得和原分布更好的相似性。
- 使输入靠近激活函数的敏感区域，并有效利用激活函数非线性功能：如 4.1 所述

5，假设你决定在你的神经网络中使用 ReLU 作为 hidden layer 的 activation 函数，你在 sigmoid activation 前面也插入了一个 ReLU 函数$\hat{y} = \sigma(\text{ReLU}(z))$。请分析这样做会有什么问题？

答案：

经过 ReLU 处理后，得到输出都≥0，使得传入 sigmoid 函数的 input 都集中在右侧区域。在多层不断处理中，原始输入为负数的数据，激活后不断靠向 sigmoid 函数函数值为+1 的地方。反向传播时，会使梯度消失问题更加严重!

6，为什么 sigmoid activation function 可能会导致 gradient vanishing 的问题？请列举至少 2 个可以克服这个问题的 activation function。

答案：

6.1

sigmoid 函数存在饱和区域，在输入趋向于正负无穷时，经 sigmoid 函数激活后再求导，其导数可以由函数值简化计算。可以看到如果函数值趋向于 0 和 1 的时候，导数值趋向于 0。同时下面等式的最大值可以求得最大为 0.25，使得反向传播时链式法则多项 sigmoid 的导数相乘结果趋向于 0，也即梯度消失问题。

$$f'(x) = f(x)(1 - f(x))$$

6.2

- ReLU(x) = max(0, x)

- LeakyReLU(x):对 ReLU 函数的改进，使得就算输入小于 0，也不会使神经元死亡。（反向传播求梯度时，对此神经元下一层的权重梯度恒等于 0，之后就不能对这个参数更新啦）

$$LeakyReLU\,(x) = \begin{cases} x, & if\ x \geq 0 \\ \alpha x, & if\ x < 0 \end{cases}$$

7，假设你为一位客户开发一个基于指纹的验证系统使客户能够用自己的指纹登录各种设备。你收集了一个开发用的数据集，训练出一个在测试集上能达到>99%准确率的模型。但是为客户部署模型后，你却遭到了客户的投诉说模型的准确率很糟。请列出一些可能导致以上情况发生的因素，并提出可能的解决办法。

答案：

- **测试集、验证集比例过小 or 没有交叉验证**：因为是在测试集上性能达到正确 99%以上，过拟合可能存在，但题目中未给出相关表述。能达到 99%以上正确率的模型确实很恐怖，可能的原因是，没有交叉验证，选择出的测试集全和训练集中样本特征相同；测试集过小，导致无法反应真实泛化性能。

  解决：采用交叉验证、调整测试集比例，同时也可以使用正则化方法缓解过拟合问题

- **数据集分布不同，进而导致对模型的 inductive bias 不同**："没有免费的午餐"，是最有可能出现的情况。开发用的数据集和客户所持有的的数据集不同，在做模型训练中做出的偏好和假设在真实使用中并不相符!

  解决：不断和客户沟通，获取部分客户持有的数据情况，进而修改已有模型的 inductive bias

8，训练完一个神经网络模型后，你发现 training accuracy (100%) 和 test accuracy (42%)之间有一个巨大的差异。请列举可能的原因和可能的解决办法？

答案：

- **过拟合**：因为特征过多（NN 层数太深）、训练数据过少，或训练 epoch 过多，导致模型泛化性能下降。解决：对以上三种情况可以分别尝试：减少 NN 层数或 neuron 个数、增加训练

数据（数据增广）、训练时不断记录 loss，采用早停技术

- **未采用交叉验证、数据不平衡：**训练集和测试集之间数据所有类别以及特征可能不同，导致在训练集上训练的模型未必拥有较好的泛化性能（对所有特征都可预测）。解决：shuffle（随机打乱）数据，并重新训练；进行交叉验证，选出泛化性能好的模型。

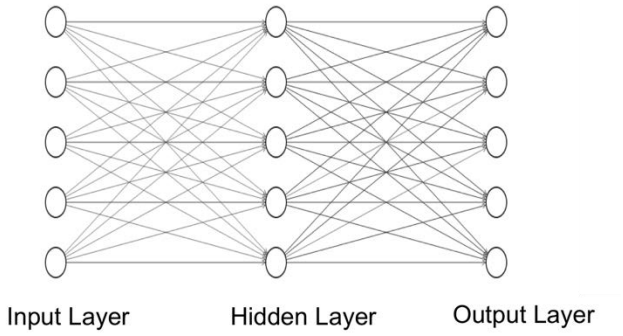
9，什么时候可以使用 multi-task learning 技术？

答案：

- **数据集相关且数据不足 or 数据集质量不高时：**MTL 可以帮助进行隐式的数据增强。多任务数据集合并，并通过共享模型参数来共同训练；数据集都会有一些噪音，当针对任务 A 训练模型时，目标是为该任务学习到一个好的表达，在理想情况下能忽视数据相关的噪音并具有良好的泛化性。由于不同任务具有不同的噪音模式，一个同时学习两个任务的模型能够学习到一个更为通用的表达，从而减少模型在单个任务上过拟合的风险。
- **正则化降低过拟合风险：**MTL 可看作是一个引入归纳偏置的正则化措举，能够减少模型的复杂度和过拟合的风险。
- **单任务学习效果不佳：**可能存在一些特征，这些特征对于任务 B 来说可以轻易的学习到，但对于任务 A 来说却极为困难。这可能是任务 A 以一种更为复杂的方式与特征交互，或是其

他特征阻碍了其学习特征。同时，MTL 迫使模型倾向于学习其他任务也需要的表达，这会帮助模型更好地泛化到新任务上。

10，假设你构建了如下结构的 2 层神经网络，请问它一共有多少参数（包括 bias）？



答案：

一共 60 个，w：5*5*2，b：5+5

11，请列举 3 个能够克服模型 overfitting 的方法？

答案：

- **早停：**训练时划分训练集和验证集，每次 epoch 训练后在验证集上看效果，不断记录在验证集上 loss。当验证集上 loss 达到最低点时停止训练。

- **加入结构风险-正则化项：**加入惩罚项也即加入了对模型的 bias，如可以通过加入 L1 或 L2 正则化项使结果参数稀疏，达到降低模型复杂度的作用。
- **Dropout 层：**在神经元中随机凋亡一些，减少了神经元间依赖关系，使模型在不同的神经元子集上训练，提高泛化能力。

12，请列举 2 个能够解决或减轻 gradient exploding 问题的方法？

答案：

- **梯度裁剪：**当梯度对应范数超过预定阈值时，对梯度进行缩放使其缩小到一定范围内。从而稳定训练过程，避免因梯度过大而使模型参数更新过于剧烈。
- **批归一化 batch normalization：**具体细节同第 4 题，对数据分布做重新调整，具体调整稀疏 gamma 和 beta 由神经网络学习得到。期望上保证经激活函数后的值稳定在合理范围内，既不会引起梯度消失，也不会导致提到爆炸。

13，尽管 Pooling 操作会使模型丢失掉一些 feature map 信息，那我们为什么常在 CNN 里面使用 Pooling 操作？

答案：

- **有效特征提取**：pooling 操作在规定 field 内保留了值最大的 pixel，可以认为是保留了主要细节，并丢弃了其他不必要细节；

- **特征降维：**pooling 只保留值最大的 pixel，减少了神经网络 hidden layer 中 input feature 的个数，使得后续神经网络参数减少，降低过拟合风险，加快模型训练。

14，假设你在开发一个分类模型，你首先在一个有 20 个样本的数据上训练模型，模型训练收敛后，你发现 training loss 仍然挺高；所以你打算到一个有 10000 个样本的数据上重新训练模型。请问这样你能够获得满意的训练模型么？如果可以请解释原因和可能会得到的结果；如果不能，请解释原因并提供解决方法。

答案：

由于模型已经训练收敛，而此时 training loss 仍然很高，也就是发生了欠拟合 underfitting。欠拟合产生的主要原因是模型复杂度不够、模型的 inductive bias 错误或数据特征很弱，和数据数量无关。所以到 10000 个样本数据上重新训练数据，并不能得到满意结果。

解决：增加模型复杂度，如增加更多隐层、神经元数量；更改模型的 inductive bias，如更改核函数；做数据的特征工程，如进行归一化处理、重新采样质量更高特征更明显的数据。

15，请介绍 Self-Attention，并比较其和 LSTM 的相同点和不同点。

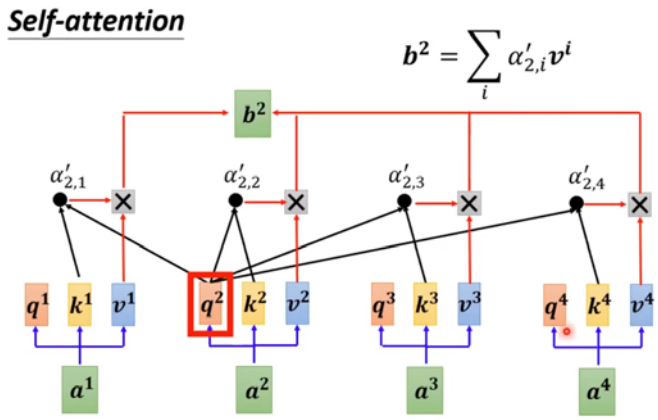
答案：

15.1

传统的 Attention 机制发生在 **Target 的元素和 Source 中的所有元素**之间。在一般任务的 Encoder-Decoder 框架中，输入 Source 和输出 Target 内容是不一样的。Attention 任务可以看成是对原数据特征 X 加权求和后的结果（Q,K,V 都是由 X 线性变换得到，由机器自己来学如何变幻），计算的结果可以看做是数据间的相关度信息（可以从 $\mathrm{softmax}(XX^{T})X$ 来理解）

$$Attention(Q,K,V)=softmax(\frac{QK^{T}}{\sqrt{d_k}})V$$

Self-attention 源自于 attention。Self - Attention 是 **Source 内部元素之间或者 Target 内部元素之间发生的 Attention 机制**。Self-attention 计算每一个输入样本和其他样本的内积，这和 attention 计算过程是一样的，只是计算对象发生了变化，相当于是 **Query=Key=Value**，计算过程与 attention 一样。



15.2

LSTM 源自 RNN，为了解决 RNN 接受长 sequence 时梯度消失和爆炸的问题，进而引入了输入、遗忘、输出门。

- **相同点：**

  二者都是用于处理 sequence 输入问题，并捕捉长文本上下文间信息。

- **不同点：**
  - **位置信息：**二者对于输入文本的位置信息捕获方式不同。self-attention 各个位置可以分别计算（并行性），没有考虑序列之间的先后关系（在序列中随意变换词与词之间的位置，得到的结果一样）；而 LSTM 通过顺序排列的 cell 以及选择是否以往，捕获输入文本位置相对顺序并决定是否保留。
  - **训练难度：**尽管 LSTM 部分解决了 RNN 的问题，但其在训练时梯度消失和爆炸的问题并未完全解决。同时 LSTM 也是一个 RNN，在序列很长其参数量很大，难以训练。

16，为什么 deep network 需要 non-linear 的 activation function?

答案：

NN 的产生原因是为了拟合更复杂、非线性的函数，其具有的非线性拟合能力正是通过 non-linear 的激活函数实现的。若使用 linear 的激活函数，神经网络只能对输入数据做线性变换，无法捕获数据中隐含的复杂非线性关系。

17，请列举 mini-batch 的优点和缺点。

答案：

Mini-batch 方法在随机梯度下降中进行使用，其估算权重梯度，并使用估算得到的梯度进行下降优化，可以分为以下四步：

step1：所有数据随机排列 Shuffle

step2：随机排列的数据抽取一定数量数据，大小为 Batch size

step3：计算此 batch 中每个数据的损失函数，并对损失函数求平均

step4：用平均后的损失函数（下式左）反向传播，计算对应梯度权重。

$$\frac{\partial \frac{\sum L_i}{N}}{\partial w} = \frac{1}{N}\sum \frac{\partial L_i}{\partial w}$$

此时，平均的权重梯度和数据梯度（上式左） = batch 中所有数据的单个损失函数对应权重梯度和数据梯度的平均（上式右）

- 优点：
  - **降低运算器的内存需求：**在一些中低端 gpu 中，其缓存很

小，而数据集通常很大，无法支持对所有数据的存储。而 mini-batch 只需要存储当前 batch size 数量的数据即可，使得模型训练的要求下降。

- **可以并行计算**：在并行多处理器中，如 COW、SMP 上，mini-batch 可以在多核上分别计算当前 batch 的梯度情况。最终只需进行通讯（SM、LM、消息传递），即可在一个核上完成参数的多次更新。当然也可以采用流水线技术。
- **随机性**：由于并不是由整个数据集求梯度得到的，在不同的数据子集上并不能代表全部数据的梯度信息。但从好的角度来看其引入了随机性，帮助模型避免陷入局部最优（类似模拟退火算法解决避免陷入局部最优）

- 缺点：
  - **随机性**: 在不同 mini-batch 上计算得到的梯度可能会差别很大，需要不断依据此时的大小调整学习率以保证稳定性和收敛性

18，我们有一个做 binary classification 的单层神经网络。Input $X \in R^{n\times m}$, Output $\hat{y} \in R^{1\times m}$, true lable $y \in R^{1\times m}$ 以及具体的计算如下：

$$z^{[1]} = W^{[1]}X + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$\hat{y} = a^{[1]}$$
$$\mathcal{J} = -\sum_{i=1}^{m} y^{(i)} \log(\hat{y}^{[i]}) + (1 - y^{(i)}) \log(1 - \hat{y}^{[i]})$$

请以矩阵向量计算的形式写出$\frac{\partial J}{\partial W^{[1]}}$的计算结果。

答案：

本题中损失函数 J 是对 m 个数据共同定义的，每个 n 维样本经过单层神经网络后得到标量 y_hat。由于对矩阵 $W^{[1]}$ 求导，也就是对其各个分量求导的组合，故以下先对各个分量求导，并最终组合成矩阵形式。【也可通过转化目标函数为矩阵式，并采用矩阵微分直接求解】

$$\frac{\partial J}{\partial w_i} = \left(\sum_{j=1}^{m} \frac{\partial J}{\partial \hat{y}^{[j]}} \cdot \frac{\partial \hat{y}^{[j]}}{\partial z_j^{[1]}} \cdot \frac{\partial z_j^{[1]}}{\partial w_i} \right)$$

$$= \sum_{j=1}^{m} \frac{\hat{y}^{[j]} - y^j}{\hat{y}^{[j]} \cdot (1 - \hat{y}^{[j]})} \cdot \sigma(z_j^{[1]}) \cdot (1 - \sigma(z_j^{[1]})) \cdot X_{ij}$$

$$\text{由于 } \frac{\partial J}{\partial W^{[1]}} = \left(\frac{\partial J}{\partial W_1^{[1]}}, \cdots, \frac{\partial J}{\partial W_n^{[1]}} \right)$$

$$\text{所以 } \frac{\partial J}{\partial W^{[1]}} = (y - \hat{y}) \cdot X^T$$

以上乘法都是矩阵向量乘法

19，假设你训练一个 GAN 模型，在每一个 epoch 的结尾你都记录了 Generator 和 Discriminator 的 loss 值。 你发现第 1 个和第 100 个 epoch 结尾的 losses 非常接近，没啥变化。请问在这种情况下生成的图片的质量是否是一样的？为什么？

答案：

不一样，原因有以下几点：

- **生成的分布只是高维空间中的低维映射：**GAN 产生一个分布，

可能原本输出所对应的空间维度很大（如图片每个 pixel 对应一个维度），真实分布和生成的分布的重叠部分在高维空间中都只占了很小的一部分。如想要 GAN 产生目标类型图片可能只是高维空间的一个低维映射，真实分布和生成分布很难有很多重叠，进而训练时 loss 可能不会发生很大变化。

- **可能真实、生成分布重合很大，但进行 sample 出的测试样本很少：**空间很大，sample 样本很少，最终可能也会认为两重合率很大的分布相差很大，也即 loss 在训练前后变化很小。
- **衡量分布差异的泛函选择不佳：**如使用 JS 散度，只要两分布不重合，其值一直是 log2。可以更换为 Wasserstein distance 来衡量两分布差异（WGAN）
-

20，假设你用一个 2 层 hidden layer 的神经网络来解决 K-class 的分类问题。具体定义如下：

$$\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$
$$\mathbf{a}^{[1]} = \mathrm{LeakyReLU}(\mathbf{z}^{[1]}, \alpha = 0.01)$$
$$\mathbf{z}^{[2]} = W^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$
$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{z}^{[2]})$$
$$L = -\sum_{i=1}^{K} \mathbf{y}_i \log(\hat{\mathbf{y}}_i)$$

其中输入 $X \in R^{D_x \times 1}$, one-hot encoded 标签$y \in \{0,1\}^K$， $z^{[1]} \in R^{D_a \times 1}$。softmax 定义如下：

$$\hat{\mathbf{y}} = [\frac{\exp(\mathbf{z}_1^{[2]})}{Z}, \ldots, \frac{\exp(\mathbf{z}_K^{[2]})}{Z}]$$

$$Z = \sum_{j=1}^{K} \exp(\mathbf{z}_j^{[2]})$$

i. 请问$W^{[2]}$和$b^{[2]}$的维度是多少（dimensionality）？如果我们一次给一个 batch 的输入数据 $X \in R^{D_x \times m}$（m 个样本）,请问最后一层输出的结果的维度是多少？

答案：

$$W^{[2]} \in \mathcal{R}^{K \times D_a}$$
$$b^{[2]} \in \mathcal{R}^{K \times 1}$$
输出的维度是 $K \times m$

ii. 请计算$\partial \hat{y}_k / \partial z_k^{[2]}$？请尽量用$\hat{y}$里的元素表示最终结果。

答案：

$$\begin{aligned}
\frac{\partial \hat{y}_k}{\partial z_k^{[2]}} &= \frac{\partial}{\partial z_k^{[2]}} \left(\frac{e^{z_k^{[2]}}}{(\sum_{j=1}^{K} e^{z_j^{[2]}})^2} \right) \\
&= \frac{e^{z_k^{[2]}} \cdot Z - (e^{z_k^{[2]}})^2}{Z^2} \\
&= \frac{e^{z_k^{[2]}} \cdot \left(\sum_{j=1}^{K} e^{z_j^{[2]}} - e^{z_j^{[2]}} \right)}{Z^2} \\
&= \frac{e^{z_k^{[2]}}}{Z} \cdot \frac{\sum_{j=1}^{K} e^{z_j^{[2]}} - e^{z_j^{[2]}}}{Z} \\
&= \hat{y}_k \cdot (1 - \hat{y}_k)
\end{aligned}$$

iii. 如果$i \neq k$，请给出$\partial \hat{y}_k / \partial z_i^{[2]}$的计算结果，请尽量用$\hat{y}$里的元素表示最终结果。

答案：

$$\begin{aligned}\frac{\partial \hat{y}_k}{\partial z_i^{[2]}} &= \frac{\partial}{\partial z_i^{[2]}}\left(\frac{e^{z_k^{[2]}}}{(\sum_{j=1}^{K} e^{z_j^{[2]}})^2}\right) \\ &= \frac{0 - e^{z_k^{[2]}} e^{z_i^{[2]}}}{Z^2} \\ &= -\hat{y}_i \cdot \hat{y}_k\end{aligned}$$

iv. 假设 y 向量里的第 k 个元素值是 1，其它都是 0。请分情况计算$\partial L / \partial z_i^{[2]}$？请尽量用$\hat{y}$里的元素表示最终结果。

答案：

由题，$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) = -\mathbf{y} \cdot \log(\hat{\mathbf{y}}) = -log(y_k)$

所以 $\frac{\partial L}{\partial z_i^{[2]}} = -\frac{\partial(log(\hat{y}_k))}{\partial z_i^{[2]}}$

①$i = k$

$$\frac{\partial L}{\partial z_i^{[2]}} = -\frac{\partial(log(\hat{y}_k))}{\partial z_k^{[2]}} = -\frac{\partial(z_k^{[2]} - log(Z))}{\partial z_k^{[2]}} = \hat{y}_k - 1$$

①$i \neq k$

$$\frac{\partial L}{\partial z_i^{[2]}} = -\frac{\partial(log(\hat{y}_k))}{\partial z_i^{[2]}} = -\frac{\partial(z_k^{[2]} - log(Z))}{\partial z_i^{[2]}} = \hat{y}_i$$