机器学习导论 习题四

学号, 姓名, 邮箱

2023 年 6 月 2 日

作业提交注意事项

1. 请在 LaTeX 模板中第一页填写个人的学号、姓名、邮箱;
2. 本次作业需提交作答后的该 pdf 文件、编程题 .ipynb 文件; **请将二者打包为 .zip 文件上传**. 注意命名规则, 三个文件均命名为“学号_姓名”+ “. 后缀”(例如 211300001_张三”+ “.pdf”、“.ipynb”、“.zip”);
3. 若多次提交作业, 则在命名 .zip 文件时加上版本号, 例如 211300001_ 张三 _v1.zip”(批改时以版本号最高的文件为准);
4. 本次作业提交截止时间为 **5 月 24 日 23:59:59**. 未按照要求提交作业, 提交作业格式不正确, **作业命名不规范**, 将会被扣除部分作业分数; 除特殊原因 (如因病缓交, 需出示医院假条) 逾期未交作业, 本次作业记 0 分; **如发现抄袭, 抄袭和被抄袭双方成绩全部取消**;
5. 本次作业提交地址为 here, 请大家预留时间提前上交, 以防在临近截止日期时, 因网络等原因无法按时提交作业.

1 [15pts] Vanishing Gradient Problem

在使用梯度下降与反向传播训练深度神经网络时, 可能会出现梯度消失的问题, 即网络参数的梯度非常小, 导致网络更新非常缓慢, 甚至停止更新. 该问题的成因较为复杂, 有很多因素可能导致该问题的出现. 本题将主要讨论激活函数与该问题之间的联系.

(1) **[5pts]** 当在深度神经网络中采用 Sigmoid 激活函数时, 网络训练容易出现梯度消失问题. 为分析此现象, 请先求解 Sigmoid 导函数的值域, 并根据该范围, 进一步对该现象进行分析.

(2) **[5pts]** 当前深度神经网络大多采用 ReLU 激活函数, 试分析相较于 Sigmoid, ReLU 对梯度消失问题的缓解作用, 同时思考其可能带来的一些问题.

(3) **[5pts]** 请从激活函数之外的角度, 列举三项缓解梯度消失问题的措施.

Solution. 此处用于写解答 (中英文均可)

(1) Sigmoid 函数形式为 $f(x) = 1/(1 + e^{-x})$, 其导函数为 $f'(x) = f(x)(1 - f(x))$. 由于 $f(x)$ 的值域为 $(0, 1)$, 因此其导函数的值域为 $(0, 0.25]$, 当 $x = 0$ 时, $f'(x)$ 取得最大值 0.25. 由于 $0 < f'(x) \leq 0.25 < 1$, 因此在反向传播时, 梯度值将会随深度指数衰减, 致使网络浅层参数的梯度趋近于 0, 出现梯度消失.

(2) 对于 ReLU 激活函数, 当输入为正数时, 其梯度为 1, 虽然有一部分神经元可能不会被激活, 但对于激活的神经元, 其梯度在后向传播的过程中, 不会像 Sigmoid 那样发生梯度的指数衰减, 进而对梯度消失问题有一定的缓解作用.

   虽然将 ReLU 作为激活函数可以缓解梯度消失问题, 但也会导致一些其他问题. 例如当输入为负数时, ReLU 的输出将为 0, 对应的梯度也为 0, 如果网络初始化不当或者学习率过大, 会导致部分神经元始终无法被激活, 出现 “The Dying ReLU Problem”.

(3)
- 更改网络结构, 加入残差连接;
- 在网络中加入 Batch Normalization, 即对网络中各层的输入进行批量归一化;
- 采用不同的网络训练方法, 例如先逐层训练神经网络, 最后再对整体进行微调.

2 [15pts] Derivation and Analysis of PCA

记中心化样本 $\mathbf{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) \in \mathbb{R}^{d\times n}$ 满足 $\sum_i \boldsymbol{x}_i = \mathbf{0}$; 投影变换 $\mathbf{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{d'}) \in \mathbb{R}^{d\times d'}$, 每维是正交基向量, 满足 $\|\boldsymbol{w}_i\|_2 = 1, \boldsymbol{w}_i^\top \boldsymbol{w}_j = 0 (\forall i \neq j)$.

(1) **[5pts]** 用拉格朗日乘子法求解 PCA 的优化问题.

$$\begin{aligned} \max_{\mathbf{W}} \quad & \operatorname{tr}\left(\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W}\right) \\ \text{s.t.} \quad & \mathbf{W}^{\mathrm{T}}\mathbf{W} = \mathbf{I} \end{aligned}$$

(2) **[5pts]** 对于以下三个样本点: $\boldsymbol{x}_1 = (-1, 1)^{\mathrm{T}}, \boldsymbol{x}_2 = (0, -2)^{\mathrm{T}}, \boldsymbol{x}_3 = (1, 1)^{\mathrm{T}}$, 试用 (1) 中得到的结果求解最大主成分对应的 $\boldsymbol{w}_1$.

(3) **[5pts]** 设原样本 $\mathbf{X}$ 的协方差矩阵对应的 d' 个特征值组成的投影变换为 $\mathbf{W}$. 考虑以下三种变换: 平移 (每个样本沿向量 $\boldsymbol{q}$ 方向移动距离 s)、放缩 (每个样本乘以放大率 α) 和旋转 (样本围绕点 $\boldsymbol{p}$ 顺时针旋转 θ). 试求解变换后的样本 $\hat{\mathbf{X}}$ 对应的 $\hat{\mathbf{W}}$.

Solution.

(1) 写出凸优化问题的标准形式:

$$\begin{aligned} \min_{\mathbf{W}} \quad & -\operatorname{tr}\left(\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W}\right) \\ \text{s.t.} \quad & \mathbf{W}^{\mathrm{T}}\mathbf{W} = \mathbf{I} \end{aligned}$$

写出优化目标的 Lagrange 函数为:

$$\begin{aligned} L(W, \Theta) &= -\operatorname{tr}\left(W^T X X^T W\right) + \left\langle \Theta, W^T W - I \right\rangle \\ &= -\operatorname{tr}\left(W^T X X^T W\right) + \operatorname{tr}\left(\Theta^T \left(W^T W - I\right)\right) \end{aligned}$$

考虑约束 $w_i^T w_i = 1$, 则只需对 $W^T W$ 的对角线进行约束. 因此此时的拉格朗日乘子矩阵为对角矩阵, 不妨记为 $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \cdots, \lambda_{d'})$ 此时 Lagrange 函数写为

$$L(W, \Theta) = -\operatorname{tr}\left(W^T X X^T W\right) + \operatorname{tr}\left(\Lambda^T \left(W^T W - I\right)\right)$$

对 W 求偏导使其为 0 :

$$\begin{aligned} \frac{\partial L}{\partial W} &= -\frac{\partial}{\partial W}\operatorname{tr}\left(W^T X X^T W\right) + \frac{\partial}{\partial W}\operatorname{tr}\left(\Lambda^T \left(W^T W - I\right)\right) \\ &= -2XX^T W + W\left(\Lambda + \Lambda^T\right) \\ &= -2XX^T W + 2W\Lambda = 0 \end{aligned}$$

可得

$$XX^T \omega_i = \lambda_i \omega_i \tag{2.1}$$

又 XX^T 是实对称阵, 其不同特征值所属的特征向量正交, 同一特征值的可以 Schmidt 正交

化, 故上述限制并不是必要. 因此我们有

$$\begin{aligned}\max_{\mathbf{W}} \operatorname{tr}\left(\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W}\right) &= \max_{\mathbf{W}} \sum_{i=1}^{d'} w_i^T X X^T w_i \\ &= \max_{\mathbf{W}} \sum_{i=1}^{d'} w_i^T \lambda_i w_i \\ &= \max_{\mathbf{W}} \sum_{i=1}^{d'} \lambda_i w_i^T w_i \\ &= \max_{\mathbf{W}} \sum_{i=1}^{d'} \lambda_i\end{aligned}$$

优化的目标函数最终的最优值是 XX^T 的前 d′ 个最大的特征值之和, 其对应的 W 是由 XX^T 前 d′ 大的特征值对应的特征向量组成, $W^* = (w_1, w_2, \cdots, w_{d'})$.

(2) $\boldsymbol{X} = \begin{pmatrix} -1 & 0 & 1 \\ 1 & -2 & 1 \end{pmatrix}$, $\boldsymbol{X}\boldsymbol{X}^\top = \begin{pmatrix} 2 & 0 \\ 0 & 6 \end{pmatrix}$, 容易得到 $\lambda_{11} = 6, \boldsymbol{w}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

(3) 平移和旋转变换对应的 $\hat{W} = W$.

(a) 平移变换, 此平移变换为:

$$x_i' = x_i + s\frac{q}{|q|}$$

对于变换后的均值有

$$\bar{x}' = \frac{1}{d}\sum_{i=1}^{d} x_i' = \frac{1}{d}\sum_{i=1}^{d} x_i + s\frac{q}{|q|} = \bar{x} + s\frac{q}{|q|}$$

变换后的协方差矩阵有

$$\begin{aligned}\Sigma' &= \sum_{i=1}^{d} \left(x_i' - \bar{x}'\right)\left(x_i' - \bar{x}'\right)^T \\ &= \sum_{i=1}^{d} \left(x_i + s\frac{q}{|q|} - \bar{x} - s\frac{q}{|q|}\right)\left(x_i + s\frac{q}{|q|} - \bar{x} - s\frac{q}{|q|}\right)^T \\ &= \sum_{i=1}^{d} \left(x_i - \bar{x}\right)\left(x_i - \bar{x}\right)^T = \sum_{i=1}^{d} x_i x_i^T = XX^T = \Sigma\end{aligned}$$

因此, 协方差矩阵没有变化, 变换后的样本 $\hat{\mathrm{X}}$ 对应的 $\hat{\mathrm{W}}$ 也没有变化, $\hat{W} = W$

(b) 对于伸缩变换变换为:

$$x_i' = \alpha x_i$$

对均值有

$$\bar{x}' = \alpha\bar{x} = 0.$$

对变换后的协方差矩阵有

$$\begin{aligned}\Sigma' &= \sum_{i=1}^{d} \left(x_i' - \bar{x}'\right)\left(x_i' - \bar{x}'\right)^T \\ &= \sum_{i=1}^{d} \alpha^2 \left(x_i - \bar{x}\right)\left(x_i - \bar{x}\right)^T \\ &= \sum_{i=1}^{d} \alpha^2 x_i x_i^T = \alpha^2 X X^T = \alpha^2 \Sigma\end{aligned}$$

容易发现变换后的协方差矩阵和变换前的协方差矩阵的特征向量不变, 特征子空间相同. 因此变换后的样本 $\hat{\mathrm{X}}$ 对应的 $\hat{\mathrm{W}}$ 没有变化, $\hat{W} = W$

(c) 旋转变换

用旋转矩阵 M 来表示旋转操作. 先将向量平移到以 p 点为原点的坐标系中旋转, 再还原坐标系, 且旋转矩阵是正交矩阵. 于是有如下变换表达形式

$$x_i' = M\left(x_i - p\right) + p,$$

易得均值有

$$\bar{x}' = M(\bar{x} - p) + p = -Mp + p$$

协方差矩阵有

$$\begin{aligned}\Sigma' &= \sum_{i=1}^{d} \left(x_i' - \bar{x}'\right)\left(x_i' - \bar{x}'\right)^T \\ &= \sum_{i=1}^{d} \left(M\left(x_i - p\right) + p + Mp - p\right)\left(M\left(x_i - p\right) + p + Mp - p\right)^T \\ &= \sum_{i=1}^{d} \left(Mx_i\right)\left(Mx_i\right)^T \\ &= \sum_{i=1}^{d} Mx_i x_i^T M^T \\ &= MXX^T M^T = M\Sigma M^T\end{aligned}$$

现在求解

$$\begin{aligned}&\max_{\hat{\boldsymbol{W}}} \operatorname{tr}\left(\hat{W}^T (MX)(MX)^T \hat{W}\right) \\ &\text{s.t. } \hat{W}^T \hat{W} = I\end{aligned}$$

可以发现:

$$\begin{aligned}W^T XX^T W &= W^T IXX^T IW \\ &= W^T M^T MXX^T M^T MW \\ &= (MW)^T MXX^T M^T (MW)\end{aligned}$$

且

$$(MW)^T (MW) = W^T M^T MW = I$$

因此可知, 对于上面的优化问题

$$\begin{aligned} &\max_{\mathbf{W}} \operatorname{tr}\left(\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W}\right) \\ &\text{s.t. } \mathbf{W}^{\mathrm{T}}\mathbf{W} = \mathbf{I} \end{aligned}$$

通过映射 $\Phi(X) = MX$ 对目标函数和约束函数进行复合可以构造原优化问题的等价问题:

$$\begin{aligned} \max_{\mathbf{W}} \quad & \operatorname{tr}\left((MW)^T(MX)(MX)^T(MW)\right) \\ \text{s.t.} \quad & (MW)^T(MW) = I \end{aligned}$$

由于等价问题的可行集和最优解相同, 设原样本 $\mathbf{X}$ 的协方差矩阵对应的 d' 个特征值组成的投影变换为 $\mathbf{W}$, 新的优化问题的解也为 W, 而 M 是本来我们已经固定的矩阵, 进行优化变量的代换后, 易得我们要新求解的优化问题的解为 $\hat{W} = MW$.

3 [35pts] Theoretical Analysis of k-means Algorithm

给定样本集 $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, k-means 聚类算法希望获得簇划分 $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$, 使得最小化欧氏距离

$$J(\gamma, \mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \sum_{j=1}^{k} \gamma_{ij} \|\mathbf{x}_i - \mu_j\|^2, \tag{3.1}$$

其中 $\mu_1, \ldots, \mu_k$ 为 k 个簇的中心 (means), $\gamma \in \mathbb{R}^{n \times k}$ 为指示矩阵 (indicator matrix). γ 具体定义如下: 若 $\mathbf{x}_i$ 属于第 j 个簇, 则 $\gamma_{ij} = 1$, 否则为 0. 算法 1 中所示为经典 k-means 聚类算法的具体流程 (与课本中描述稍有差别, 但实际上是等价的).

Algorithm 1: k-means Algorithm

1 Initialize $\mu_1, \ldots, \mu_k$.

2 **repeat**

3 **Step 1**: Decide the class memberships of $\{\mathbf{x}_i\}_{i=1}^n$ by assigning each of them to its nearest cluster center.

$$\gamma_{ij} = \begin{cases} 1, & \|\mathbf{x}_i - \mu_j\|^2 \le \|\mathbf{x}_i - \mu_{j'}\|^2, \forall j', \\ 0, & \text{otherwise.} \end{cases}$$

4 **Step 2**: For each $j \in \{1, \cdots, k\}$, recompute μ_j using the updated γ to be the center of mass of all points in C_j:

$$\mu_j = \frac{\sum_{i=1}^n \gamma_{ij} \mathbf{x}_i}{\sum_{i=1}^n \gamma_{ij}}.$$

5 **until** *the objective function J no longer changes*;

(1) **[5pts]** 试证明, 在算法 1 中, **Step 1** 和 **Step 2** 都会使目标函数 J 的值降低.

(2) **[5pts]** 试证明, 算法 1 会在有限步内停止.

(3) **[5pts]** 试证明, 目标函数 J 的最小值是关于 k 的非增函数, 其中 k 是聚类簇的数目.

(4) **[10pts]** 记 $\hat{\mathbf{x}}$ 为 n 个样本的中心点, 定义如下变量:

total deviation	$T(X) = \sum_{i=1}^n \|\mathbf{x}_i - \hat{\mathbf{x}}\|^2 / n$
intra-cluster deviation	$W_j(X) = \sum_{i=1}^n \gamma_{ij} \|\mathbf{x}_i - \mu_j\|^2 / \sum_{i=1}^n \gamma_{ij}$
inter-cluster deviation	$B(X) = \sum_{j=1}^k \frac{\sum_{i=1}^n \gamma_{ij}}{n} \|\mu_j - \hat{\mathbf{x}}\|^2$

试探究以上三个变量之间有什么样的等式关系? 基于此, 请证明, k-means 聚类算法可以认为是在最小化 intra-cluster deviation 的加权平均, 同时近似最大化 inter-cluster deviation.

(5) **[10pts]** 在公式 3.1 中, 我们使用 ℓ_2-范数来度量距离 (即欧氏距离), 下面我们考虑使用 ℓ_1-范数来度量距离:

$$J'(\gamma, \mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \sum_{j=1}^{k} \gamma_{ij} \|\mathbf{x}_i - \mu_j\|_1. \tag{3.2}$$

- **[5pts]** 请仿效算法 1 (k-means-ℓ_2 算法), 给出新的算法 (命名为 k-means-ℓ_1 算法) 以优化公式 3.2 中的目标函数 J'.
- **[5pts]** 当样本集中存在少量异常点 (outliers) 时, 对于上述的 k-means-ℓ_2 和 k-means-ℓ_1 算法, 我们应该如何选择? 请从算法鲁棒性的角度分析, 说明哪个算法具有更好的鲁棒性?

Solution.

(1) 在 **Step 1** 中, $\forall i$, 令

$$\hat{j} = \min_{1 \le j \le k} \|\mathbf{x}_i - \mu_j\|^2$$

又因为 γ_i 是指示向量,

$$\sum_{j=1}^{k} \gamma_{ij} \|\mathbf{x}_i - \mu_j\|^2 \ge \left\|\mathbf{x}_i - \mu_{\hat{j}}\right\|^2$$

故 $J(\gamma, \mu_1, \ldots, \mu_k)$ 若在第一步发生改变, 一定下降.
在 **Step 2** 中, 令 $X_{\in j} = \{\mathbf{x}_i \mid \gamma_{ij} = 1\}$ 代表在族 j 中点的集合, $n_j = |X_{\in j}|$ 为该集合大小, $\overline{\mathbf{x}}$ 为该集合的均值. 则 $\forall j, \forall \mathbf{a}$:

$$\begin{aligned}\sum_{x \in X_{\in|}} \|\mathbf{x} - \mathbf{a}\|^2 &= \sum_{x \in X_{\in j}} \left(\mathbf{x}^{\mathrm{T}}\mathbf{x} + \mathbf{a}^{\mathbf{T}}\mathbf{a} - 2\mathbf{x}^{\mathbf{T}}\mathbf{a}\right) \\ &= \sum_{x \in X_{\in j}} \mathbf{x}^{\mathbf{T}}\mathbf{x} + n_j \mathbf{a}^{\mathrm{T}}\mathbf{a} - 2n_j \overline{\mathbf{x}}^{T}\mathbf{a}\end{aligned}$$

而当 $\mathbf{a} = \overline{\mathbf{x}}$ 时上式取得最小值. 故 $J(\gamma, \mu_1, \ldots, \mu_k)$ 若在第二步发生改变, 一定下降.

(2) 由于不同的 $J(\gamma, \mu_1, \ldots, \mu_k)$ 对应不同的 γ, 且每次更新时 J 均下降 (不下降时终止), 故 γ 不会与先前重复. 又由于 $\gamma \in \{0, 1\}^{n \times k}$ 最多有 2^{kn} 种可能取值, 所以算法会在有限步终止.

(3) 令 $k = k_0$ 时取得 J 的最小值的指示矩阵 γ 不变, $k = k_1$ 时将 μ_{k+1} 设为 $\mathcal{D}$ 中任意一个点, 则 J 的值必不上升. 从而 J 的最小值不上升.

(4) 令 $X_{\in j} = \{\mathbf{x}_i \mid \gamma_{ij} = 1\}$ 代表在簇 j 中点的集合, $n_j = \sum_{i=1}^{n} \gamma_{ij}$ 为该集合大小, $\hat{\mathbf{x}}$ 为该集合的均值. 则:

$$\begin{aligned} n_j W_j(X) + n B_j(X) &= \sum_{x \in X_{\in j}} \|\mathrm{x} - \mu_j\|^2 + n_j \|\mu_j - \hat{\mathbf{x}}\|^2 \\ &= \sum_{x \in X_{\in j}} \mathrm{x}^{\mathrm{T}}\mathbf{x} - n_j \|\boldsymbol{\mu}_j\|^2 + n_j \left(\|\mu_j\|^2 + \|\hat{\mathbf{x}}\|^2 - 2\mu_j^T \hat{\mathbf{x}}\right) \\ &= \sum_{x \in X_{\in j}} \mathrm{x}^{\mathrm{T}}\mathbf{x} + n_j \|\hat{\mathbf{x}}\|^2 - 2n_i \boldsymbol{\mu}_j^T \hat{\mathbf{x}} \\ &= \sum_{x \in X_{\in j}} \|\mathbf{x} - \hat{\mathbf{x}}\|^2 \end{aligned}$$

从而:

$$\sum_{j=1}^{k} \frac{n_j}{n} W_j(X) + B(X) = T(X)$$

由于算法迭代过程中 $T(X)$ 不变, 而目标函数 J 的值下降, 即 $\sum_{j=1}^{k} \frac{n_j}{n} W_j(X)$ 的值下降, 所以 $B(X)$ 上升. 所以可以认为“是在最小化 intra-cluster deviation 的加权平均, 同时近似最大化 inter-cluster deviation”.

(5) 设 d 代表维度. 当样本集中存在少数异常点时, k-means- ℓ_1 算法具有更好的鲁棒性. 因为采用平均时, 与这些异常点最近的笑的中心点很可能受到较大影响从而偏离本应在的中心, 而采用中位数时, 该中心点不会发生太大变化.

Algorithm 2: k-means- ℓ_1 Algorithm

Initialize $\mu_1, \ldots, \mu_k$.

repeat

  Step 1: Decide the class memberships:

$$\gamma_{ij} = \begin{cases} 1, & \|\mathbf{x}_i - \mu_j\|_1 \leq \|\mathbf{x}_i - \mu_{j'}\|_1, \forall j' \\ 0, & \text{otherwise} \end{cases}$$

  Step 2: For each $j \in \{1, \cdots, k\}$, recompute μ_j using the updated γ :

$$\forall, \mu_j[d] = \text{median of } \{\mathbf{x_i} \mid \gamma_i j = 1\}$$

until *the objective function J no longer changes*;

4 [35pts] Neural Network in Practice

本题需编程实现多层前馈神经网络, 且不使用现有神经网络库, 具体内容见 lab.ipynb 文件.

4.1 任务要求

在不使用现有神经网络库的前提下, 复现 lab.ipynb 中利用 PyTorch 编写的一段多层前馈神经网络代码, 需注意:

- 保持网络结构一致;
- 保持损失函数一致;
- 保持 batch_size, learning_rate, epochs 超参一致.

编写时可参考 lab.ipynb 文件中给出的代码框架, 也可以将其删除, 编写你自己的代码. 任务完成后, 需提交对应的 .ipynb 文件.

4.2 评分标准

我们会重新运行你所提交的 .ipynb 文件, 具体评分标准如下:

- 5/35: 复现的代码可正常运行;
- 10/35: 复现的代码保持了网络结构、损失函数以及各超参一致;
- 15/35: 复现的代码可在 5min 内迭代完规定的 epochs 轮数;
- 25/35: 运行结果中 running_loss 整体为下降趋势;
- 35/35: 运行结果中最后一轮的 running_acc 超过 90%.

4.3 测试环境

以下为测试环境采用的版本:

- Python: 3.8.16
- PyTorch: 2.0.0
- Numpy: 1.24.2

你可以使用你喜欢的任意版本, 只需确保你的代码能顺利运行.

lab-sol

June 2, 2023

1 MNIST Lab

1.1 MNIST reference:

- http://yann.lecun.com/exdb/mnist/

1.2 PyTorch reference:

- https://pytorch.org/docs/stable/index.html

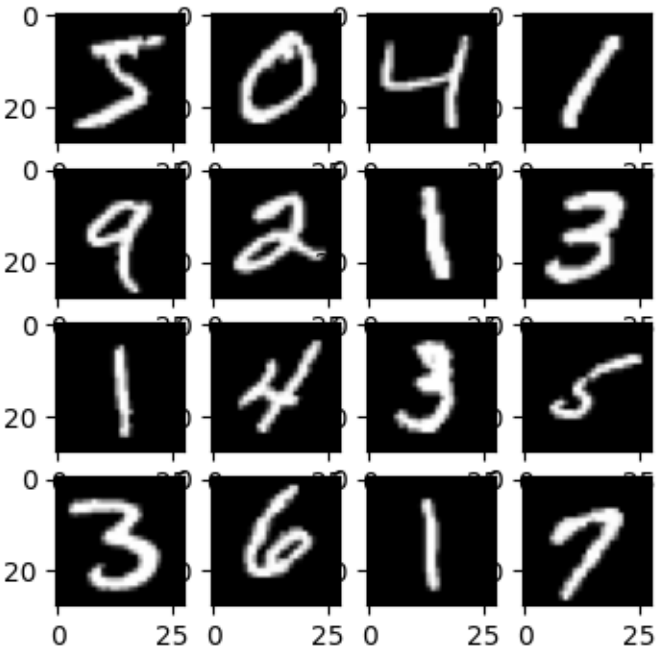
[3]:
```
# read the csv file (i.e., ./data/mnist.csv) with pandas
# store data into numpy arrays

import numpy as np
import pandas as pd

%time data = list(pd.read_csv("./data/mnist_train.csv", header=None).values)
data = np.array(data)

img = data[:, 1:].reshape((-1, 28, 28))
lab = data[:, 0]
one = np.zeros((len(lab), 10))
one[range(len(lab)), lab] = 1
data = None

batch_size, learning_rate, epochs = 200, 0.1, 20
```

```
CPU times: total: 3.39 s
Wall time: 3.4 s
```

[4]:
```
# check loaded data by showing some key properties of the numpy arrays

print(img.shape, lab.shape, np.mean(one, axis=0))
```

```
(60000, 28, 28) (60000,) [0.09871667 0.11236667 0.0993     0.10218333 0.09736667
0.09035
 0.09863333 0.10441667 0.09751667 0.09915   ]
```

[5]:
```
# check loaded data by showing a few images

import matplotlib.pyplot as plt
%matplotlib inline

size = 4
plt.figure(figsize=(size, size))
for i in range(size*size):
  plt.subplot(size, size, i+1)
  plt.imshow(img[i], cmap='gray')
```



[6]:
```
# check loaded data by showing a few labels

print(lab[:size*size])
print(one[:size*size])
```

```
[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1 7]
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

```
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
```

[7]:
```
# split the dataset into training and validation datasets
from sklearn.model_selection import train_test_split

img = img / 256

img_train, img_test, lab_train, lab_test, one_train, one_test =␣
 ↪train_test_split(img, lab, one, test_size=0.2, random_state=2020)
print(img_train.shape, img_test.shape)
print(lab_train.shape, lab_test.shape)
print(one_train.shape, one_test.shape)
```

```
(48000, 28, 28) (12000, 28, 28)
(48000,) (12000,)
(48000, 10) (12000, 10)
```

2 Lab Requirements

Rewrite the following cells to train and to evaluate the same neural network without using any Deep Learning APIs. That means you can only use NumPy or Tensors, but you cannot use torch.nn or torch.optim.

[8]:
```
import torch as pt
from tqdm.notebook import tqdm
from torch import nn, optim
from torch.autograd import Variable
from torch.utils.data import TensorDataset, DataLoader

# define a simple neural network model
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.dense = nn.Sequential(nn.Flatten(),
                                   nn.Linear(28 * 28, 256), nn.ReLU(),
                                   nn.Linear(256, 256), nn.ReLU(),
                                   nn.Linear(256, 10))

    def forward(self, x):
        return self.dense(x)
```

```
# prepare the datasets for training
img_train = pt.from_numpy(img_train).float()
lab_train = pt.from_numpy(lab_train).long()
data_train = DataLoader(TensorDataset(img_train, lab_train),␣
 ↪batch_size=batch_size, shuffle=True, drop_last=True)

img_test = pt.from_numpy(img_test).float()
lab_test = pt.from_numpy(lab_test).long()
data_test = DataLoader(TensorDataset(img_test, lab_test),␣
 ↪batch_size=batch_size, shuffle=False, drop_last=True)

# train a simple neural network model
dev = pt.device('cpu')
model = MyModel().to(dev)
loss = nn.CrossEntropyLoss()
opt = optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(epochs):
    running_loss = []
    for x, y in data_train:
        x = Variable(x).to(dev)
        y = Variable(y).to(dev)
        model.train()
        yy = model(x)

        l = loss(yy, y)
        opt.zero_grad()
        l.backward()
        opt.step()
        running_loss.append(float(l))

    running_acc = []
    for x, y in data_test:
        x = Variable(x).to(dev)
        y = Variable(y).to(dev)
        model.eval()
        yy = model(x)
        yy = pt.argmax(yy, dim=1)

        acc = float(pt.sum(yy == y)) / batch_size
        running_acc.append(acc)

    print("#summary:% 4d %.4f %.2f%%" % (epoch, np.mean(running_loss), np.
 ↪mean(running_acc)*100))
```

```
#summary:   0 0.9591 88.85%
#summary:   1 0.3377 90.87%
```

```
#summary:   2 0.2717 92.22%
#summary:   3 0.2292 93.36%
#summary:   4 0.1966 94.33%
#summary:   5 0.1710 94.61%
#summary:   6 0.1498 95.18%
#summary:   7 0.1328 95.62%
#summary:   8 0.1194 95.95%
#summary:   9 0.1077 95.94%
#summary:  10 0.0977 96.44%
#summary:  11 0.0886 96.65%
#summary:  12 0.0810 97.02%
#summary:  13 0.0740 97.08%
#summary:  14 0.0683 97.07%
#summary:  15 0.0631 97.27%
#summary:  16 0.0576 97.37%
#summary:  17 0.0536 97.44%
#summary:  18 0.0488 97.31%
#summary:  19 0.0455 97.52%
```

[10]:
```python
# rewrite the code to train and to evaluate the same neural network
# you can fill in the TODO part of the code below, or just delete it and write your own code from scratch

import os
import random
from copy import deepcopy as dco


def set_seed(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    pt.manual_seed(seed)
    pt.cuda.manual_seed(seed)
    pt.cuda.manual_seed_all(seed)
    pt.backends.cudnn.benchmark = False
    pt.backends.cudnn.deterministic = True


class Neural_Network:
    def __init__(self, dims, data_train, data_test, epoch, batch_size, l_rate):
        # initialization
        # variable definitions: x_2 = ReLU(z_2), z_2 = x_1w_1 + b_1
        set_seed(2023)
        self.w, self.b, self.x = [], [], []
        self.x.append(np.zeros([batch_size, dims[0]], 'float64'))
        for i in range(1, len(dims)):
```

```
            self.w.append(np.random.uniform(-0.1, 0.1, (dims[i-1], dims[i])))
            self.b.append(np.zeros(dims[i], 'float64'))
            self.x.append(np.zeros((batch_size, dims[i]), 'float64'))

        # define the parameters
        self.layer_num = len(dims) - 1
        self.data_train = data_train
        self.data_test = data_test
        self.epoch = epoch
        self.batch_size = batch_size
        self.l_rate = l_rate
        self.z = dco(self.x)
        self.grad_x, self.grad_w, self.grad_b = dco(self.x), dco(self.w),␣
↪dco(self.b)

    # define the activation function
    def actFunction(self, x):
        return np.maximum(x, 0)

    # define the derivative of the activation function
    def actDerivation(self, x):
        # TODO: return the the derivative
        return np.where(x > 0, 1, 0)

    # define the forward function
    def forward(self):
        # update rule: x_2 = ReLU(z_2), z_2 = x_1w_1 + b_1
        # TODO: forward update z, x
        for k in range(self.layer_num):
            self.z[k + 1] = self.x[k] @ self.w[k] + self.b[k]
            if k != self.layer_num - 1:
                self.x[k + 1] = self.actFunction(self.z[k + 1])
            else:
                self.x[k + 1] = np.exp(self.z[k + 1])
                self.x[k + 1] /= self.x[k + 1].sum(1).reshape(-1, 1).
↪repeat(self.x[k + 1].shape[1], axis=1)

    # define the loss function
    def loss(self, y_pred, y_real):
        # TODO: return the cross entropy loss
        return -np.log2(y_pred[np.arange(self.batch_size), y_real]).mean()

    # define the backward function
    def backward(self, y_real):
        # TODO: backward update grad_x, grad_w, grad_b
        # loss -> last layer
        one_hot = np.zeros((self.batch_size, self.x[-1].shape[1]))
```

```
        one_hot[np.arange(self.batch_size), y_real] = 1
        self.grad_x[-1] = self.x[-1] - one_hot

        # (k+1)th layer -> kth w, x
        for k in range(self.layer_num - 1, -1, -1):
            term = self.grad_x[k + 1] * self.actDerivation(self.z[k + 1])
            self.grad_w[k] = self.x[k].T @ term / self.batch_size
            self.grad_b[k] = term.mean(0)
            self.grad_x[k] = term @ self.w[k].T

    # define the update function
    def update(self):
        self.b = [self.b[i] - self.l_rate * self.grad_b[i] for i in range(self.
↪layer_num)]
        self.w = [self.w[i] - self.l_rate * self.grad_w[i] for i in range(self.
↪layer_num)]

    # train the neural network model
    def train(self):
        running_loss = []
        running_acc = []

        for epoch in range(self.epoch):
            for train_x, train_y in data_train:
                train_x = train_x.numpy()
                train_y = train_y.numpy()
                self.x[0] = train_x.reshape((self.batch_size, -1))
                self.forward()
                batch_loss = self.loss(self.x[-1], train_y)
                self.backward(train_y)
                self.update()
                running_loss.append(batch_loss)

            for test_x, test_y in data_test:
                test_x = test_x.numpy()
                test_y = test_y.numpy()
                self.x[0] = test_x.reshape((self.batch_size, -1))
                self.forward()
                out = np.argmax(self.x[-1], axis=1)
                running_acc.append(np.mean(out == test_y))

            print("#summary:% 4d %.4f %.2f%%" % (epoch, np.mean(running_loss),␣
↪np.mean(running_acc) * 100))

# train the network
NN = Neural_Network([28 * 28, 256, 256, 10], data_train, data_test, epochs,␣
↪batch_size, learning_rate)
```

```
NN.train()
```

```
#summary:   0 1.0297 90.00%
#summary:   1 0.7227 91.13%
#summary:   2 0.5915 91.94%
#summary:   3 0.5125 92.56%
#summary:   4 0.4571 92.98%
#summary:   5 0.4151 93.40%
#summary:   6 0.3815 93.76%
#summary:   7 0.3539 94.05%
#summary:   8 0.3306 94.31%
#summary:   9 0.3106 94.55%
#summary:  10 0.2932 94.75%
#summary:  11 0.2778 94.94%
#summary:  12 0.2640 95.11%
#summary:  13 0.2516 95.26%
#summary:  14 0.2404 95.40%
#summary:  15 0.2302 95.52%
#summary:  16 0.2209 95.63%
#summary:  17 0.2122 95.74%
#summary:  18 0.2043 95.82%
#summary:  19 0.1969 95.91%
```