Mathematical Foundations of Deep Neural Networks, M1407.001200
E. Ryu
Fall 2022

Midterm Exam
Saturday, October 15, 2022, 8:00am–12:00pm
4 hours, 7 questions, 100 points, 11 pages

This exam is open-book in the sense that you may use any non-electronic resource.
While we don't expect you will need more space than provided,
you may continue on the back of the pages.

Name: ______________________________

Do not turn to the next page until the start of the exam.

1. (10 points) $\{$*RMSProp, sign SGD*$\} \subset$ *Adam without bias correction.* Let $g^0, g^1, \ldots \in \mathbb{R}^d$ be a sequence of stochastic gradients. Define Adam without bias correction as

$$m_1^{k+1} = \beta_1 m_1^k + (1-\beta_1) g^k, \qquad m_2^{k+1} = \beta_2 m_2^k + (1-\beta_2)(g^k \circledast g^k)$$
$$\theta^{k+1} = \theta^k - \alpha m_1^{k+1} \oslash \sqrt{m_2^{k+1} + \epsilon}$$

for $k = 0, 1, \ldots$. Recall that RMSProp has the form

$$m_2^{k+1} = \beta_2 m_2^k + (1-\beta_2)(g^k \circledast g^k), \qquad \theta^{k+1} = \theta^k - \alpha g^k \oslash \sqrt{m_2^{k+1} + \epsilon}$$

for $k = 0, 1, \ldots$. Here, $\circledast$ and $\oslash$ denote element-wise multiplication and division.

(a) Show that RMSProp is an instance of Adam without bias correction.

(b) The optimizer signSGD is defined as

$$\theta^{k+1} = \theta^k - \alpha \operatorname{sign}(g^k)$$

for $k = 0, 1, \ldots$, where the sign function

$$\operatorname{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \end{cases}$$

is applied elementwise to $g^k \in \mathbb{R}^d$. For simplicity, assume $(g^k)_i \neq 0$ for all $i = 1, \ldots, d$ and $k = 0, 1, \ldots$. Show that signSGD is an instance of Adam without bias correction.

Clarification. You are being asked to identify particular choices of Adam's parameters β_1, β_2, and ϵ such that Adam reduces to the two other optimizers.

2. (10 points) *Composing conv, max-pool, ReLU.* Show that the compositions

   - conv-MP-ReLU
   - conv-ReLU-MP
   - conv-ReLU-MP-ReLU

   are equivalent. More precisely, show that the following three models are equivalent.

```
class model1(nn.Module):
  def __init__(self, in_chan, out_chan):
    super(model1, self).__init__()
    self.conv = nn.Conv2d(in_channels=in_chan,
                out_channels=out_chan, kernel_size=3)
    self.mp = nn.MaxPool2d(2, stride=2)

  def forward(self, x):
    return torch.relu(self.mp(self.conv(x)))

class model2(nn.Module):
  def __init__(self, in_chan, out_chan):
    super(model2, self).__init__()
    self.conv = nn.Conv2d(in_channels=in_chan,
                out_channels=out_chan, kernel_size=3)
    self.mp = nn.MaxPool2d(2, stride=2)

  def forward(self, x):
    return self.mp(torch.relu(self.conv(x)))

class model3(nn.Module):
  def __init__(self, in_chan, out_chan):
    super(model3, self).__init__()
    self.conv = nn.Conv2d(in_channels=in_chan,
                out_channels=out_chan, kernel_size=3)
    self.mp = nn.MaxPool2d(2, stride=2)

  def forward(self, x):
    return torch.relu(self.mp(torch.relu(self.conv(x))))
```

3. (10 points) *Convolution can represent identity.* Consider a 2DConv layer with 3×3 filter, padding 1, C input channels, and C output channels. If we want the layer to represent the identity map, i.e., if we want the 2DConv layer to map any $X \in \mathbb{R}^{C\times v\times h}$ to X itself without modification, what should the filter w and bias b be?

   Clarification. Precisely specify all elements of w and b.

4. (10 points) *Logistic regression via BCE loss.* Consider the supervised learning setup with data $X_1, \dots, X_N \in \mathbb{R}^p$ and labels $Y_1, \dots, Y_N \in \{-1, +1\}$. Recall that logistic regression uses the model

$$f_{a,b}(x) = \begin{bmatrix} \frac{1}{1+e^{a^\intercal x + b}} \\ \frac{1}{1+e^{-(a^\intercal x + b)}} \end{bmatrix}$$

and solves

$$\underset{a\in\mathbb{R}^d,\, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} D_{\text{KL}}(\mathcal{P}(Y_i) \| f_{a,b}(X_i)),$$

where

$$\mathcal{P}(Y_i) = \begin{cases} \begin{bmatrix} 1 & 0 \end{bmatrix}^\intercal \text{ if } Y_i = -1 \\ \begin{bmatrix} 0 & 1 \end{bmatrix}^\intercal \text{ if } Y_i = +1. \end{cases}$$

The *binary cross-entropy* (BCE) loss is defined as

$$\ell^{\text{BCE}}(x, y) = -(y \log x + (1-y)\log(1-x))$$

for $x \in (0, 1)$ and $y \in \{0, +1\}$. Let

$$\sigma(r) = \frac{1}{1+e^{-r}}$$

be the sigmoid activation function. Define $\tilde{f}_{a,b}(x) = a^\intercal x + b$ and

$$\tilde{Y}_i = \begin{cases} 0 & \text{if } Y_i = -1 \\ 1 & \text{if } Y_i = +1 \end{cases}$$

for $i = 1, \dots, N$. Show that

$$\underset{a\in\mathbb{R}^d,\, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} \ell^{\text{BCE}}(\sigma(\tilde{f}_{a,b}(X_i)), \tilde{Y}_i)$$

is equivalent to logistic regression.

5. (20 points) *MLP with zero-initialization.* Let $\sigma\colon \mathbb{R} \to \mathbb{R}$ be the hyperbolic tangent activation function, i.e.,

$$\sigma(r) = \frac{e^{2r} - 1}{e^{2r} + 1}.$$

Consider the MLP

$$\begin{aligned} f_\theta(x) &= y_L \\ y_L &= A_L y_{L-1} + b_L \\ y_{L-1} &= \sigma(A_{L-1} y_{L-2} + b_{L-1}) \\ &\vdots \\ y_2 &= \sigma(A_2 y_1 + b_2) \\ y_1 &= \sigma(A_1 x + b_1), \end{aligned}$$

where $x \in \mathbb{R}^{n_0}$, $A_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$, $b_\ell \in \mathbb{R}^{n_\ell}$, and $n_L = 1$. (To clarify, σ is applied element-wise.) Use the notation $\theta = (A_1, b_1, A_2, b_2, \ldots, A_L, b_L)$. Assume $A_1, \ldots, A_L, b_1, \ldots, b_L$ are all initialized to zero. Let $X_1, \ldots, X_N \in \mathbb{R}^{n_0}$ and $Y_1, \ldots, Y_N \in \mathbb{R}$. Consider training f_θ by solving

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{N} \frac{1}{2} (f_\theta(X_i) - Y_i)^2.$$

For simplicity, assume we use gradient descent (GD) for training.

(a) Show that $A_1, \ldots, A_L$ and $b_1, \ldots, b_{L-1}$ do not move throughout the training, i.e., they all stay at the initial values of 0.

(b) Assume the learning rate of GD is chosen appropriately. What does b_L converge to?

6. (20 points) *He initialization.* Consider the layer

$$y^+ = Ax + b$$
$$x = \sigma(y),$$

where $x, y \in \mathbb{R}^{n_{\text{in}}}$, $y^+ \in \mathbb{R}^{n_{\text{out}}}$, and $\sigma(r) = \max\{0, r\}$ is the ReLU activation function. Assume the elements of A and b are initialized IID as $A_{ij} \sim \mathcal{N}(0, \sigma_A^2)$ and $b_i = 0$. Assume y is a random vector, independent of A and b, such that y_j has mean 0 and variance 1 for $j = 1, \ldots, n_{\text{in}}$. Assume $y_1, \ldots, y_{n_{\text{in}}}$ are uncorrelated. Finally, assume y is symmetric, i.e., y and $-y$ have the same distribution. Show the following:

(a) $\mathbb{E}[(x_j)^2] = \frac{1}{2}$ for $j = 1, \ldots, n_{\text{in}}$.

(b) $\mathbb{E}[y^+] = 0$.

(c) $\mathbb{E}[(y_i^+)^2] = \frac{n_{\text{in}}\sigma_A^2}{2}$ for $i = 1, \ldots, n_{\text{out}}$.

(d) $y_1^+, \ldots, y_{n_{\text{out}}}^+$ are uncorrelated.

(e) y^+ is symmetric.

Hint. For (e), use the fact that A and $-A$ have the same distribution.

7. (20 points) *Is NiN shift-invariant?* Let $X \in \mathbb{R}^{3\times32\times32}$ be such that

$$X_{ijk} = \begin{cases} 1 & \text{for } i=1,\ j=19,\ k=19 \\ 0 & \text{otherwise.} \end{cases}$$

Let $\tilde{X} \in \mathbb{R}^{3\times32\times32}$ be X shifted along the j coordinate by 4 pixels, i.e.,

$$\tilde{X}_{ijk} = \begin{cases} 1 & \text{for } i=1,\ j=23,\ k=19 \\ 0 & \text{otherwise.} \end{cases}$$

Consider the NiN network, precisely defined below, in `eval` mode. For simplicity, do not use biases. We use 0-based indexing, so the j and k indices for X_{ijk} range from 0 to 31.

(a) When X and $\tilde{X}$ are provided as inputs to NiN, are the outputs approximately equal?

(b) When X is provided as input to NiN, which elements of `out1` can be nonzero?

(c) When X is provided as input to NiN, which elements of `out2` can be nonzero?

(d) When X is provided as input to NiN, which elements of `out3` can be nonzero?

(e) When X and $\tilde{X}$ are provided as inputs to NiN, are the outputs exactly equal?

Justify your answers.

```
class NiN(nn.Module):
  def __init__(self):
    super(NiN, self).__init__()
    self.mlpconv_layer1 = nn.Sequential(
      nn.Conv2d(3, 192, kernel_size=5, padding=2, bias=False),
      nn.ReLU(),
      nn.Conv2d(192, 160, kernel_size=1, bias=False),
      nn.ReLU(),
      nn.Conv2d(160, 96, kernel_size=1, bias=False),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=3, stride=2, ceil_mode=True),
      nn.Dropout()
    )
    self.mlpconv_layer2 = nn.Sequential(
      nn.Conv2d(96, 192, kernel_size=5, padding=2, bias=False),
      nn.ReLU(),
      nn.Conv2d(192, 192, kernel_size=1, bias=False),
      nn.ReLU(),
      nn.Conv2d(192, 192, kernel_size=1, bias=False),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=3, stride=2, ceil_mode=True),
      nn.Dropout()
    )
    self.mlpconv_layer3 = nn.Sequential(
      nn.Conv2d(192, 192, kernel_size=3, padding=1, bias=False),
      nn.ReLU(),
      nn.Conv2d(192, 192, kernel_size=1, bias=False),
      nn.ReLU(),
      nn.Conv2d(192, 10, kernel_size=1, bias=False),
      nn.ReLU()
    )

  def forward(self, x) :
    out1 = self.mlpconv_layer1(x)
    out2 = self.mlpconv_layer2(out1)
    out3 = self.mlpconv_layer3(out2)
    out4 = nn.AvgPool2d(kernel_size=8)(out3)
    return out4.view(-1, 10)
```