Mathematical Foundations of Deep Neural Networks, M1407.001200
E. Ryu
Spring 2021

Midterm Exam
Thursday, October 21, 2021, 3:30–7:30 pm
4 hours, 7 questions, 100 points, 10 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: ______________________________

Teaching staff signature: ______________________________

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

1. (10 points) *Color-wise finite difference as convolution.* Given a color image $X \in \mathbb{R}^{3\times m\times n}$, we wish to compute the x- and y-direction derivatives for each color channel. Define $Y \in \mathbb{R}^{6\times m\times n}$ with

$$\begin{aligned}
Y_{1,i,j} &= X_{1,i+1,j} - X_{1,i,j}\\
Y_{2,i,j} &= X_{1,i,j+1} - X_{1,i,j}\\
Y_{3,i,j} &= X_{2,i+1,j} - X_{2,i,j}\\
Y_{4,i,j} &= X_{2,i,j+1} - X_{2,i,j}\\
Y_{5,i,j} &= X_{3,i+1,j} - X_{3,i,j}\\
Y_{6,i,j} &= X_{3,i,j+1} - X_{3,i,j}
\end{aligned}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. We define $X_{:,m+1,:} = 0$ and $X_{:,:,n+1} = 0$, i.e., we define the out-of-bounds elements to have 0 value. How can we represent the mapping $X \mapsto Y$ as a convolution with a 3×3 filter and zero padding of 1? (The stride is 1.) More specifically, what should the filter $w \in \mathbb{R}^{6\times 3\times 3\times 3}$ be?

2. (15 points) *Duplicate neurons.* Consider the 2-layer neural network

$$f_\theta(x) = u^\intercal \sigma(ax + b) = \sum_{j=1}^{p} u_j \sigma(a_j x + b_j),$$

where $x \in \mathbb{R}$ and $a, b, u \in \mathbb{R}^p$. Let σ be a differentiable activation function. Using the data $X_1, \dots, X_N \in \mathbb{R}$ and labels $Y_1, \dots, Y_N \in \mathcal{Y}$, we train the neural network by solving

$$\underset{\theta \in \mathbb{R}^{3p}}{\text{minimize}} \quad \tfrac{1}{N} \textstyle\sum_{i=1}^{N} \ell(f_\theta(X_i), Y_i)$$

with Adam. Assume $\ell(f, y)$ is differentiable in f. Initialize with $\theta^0 = (a_1^0, \dots, a_p^0, b_1^0, \dots, b_p^0, u_1^0, \dots, u_p^0)$ such that $a_{p-1}^0 = a_p^0$, $b_{p-1}^0 = b_p^0$, and $u_{p-1}^0 = u_p^0$, i.e., the $(p-1)$-th and p-th neuron's parameters are equal at initialization. Show that $a_{p-1}^k = a_p^k$, $b_{p-1}^k = b_p^k$, and $u_{p-1}^k = u_p^k$ throughout the training.

3. (15 points) *Dropout-ReLU=ReLU-Dropout.* Consider the following convolutional layer

```
class myLayer(nn.Module):
  def __init__(self, input_size, output_size):
    super(myLayer, self).__init__()
    self.linear = nn.Linear(input_size,output_size)
    self.sigma = nn.ReLU()
    # self.sigma = nn.Sigmoid()
    # self.sigma = nn.LeakyReLU()
    self.dropout= nn.Dropout(p=0.4)
  def forward(self, x):
    return dropout(sigma(linear))
    # return sigma(dropout(linear))  # Is this is equivalent?
```

In which of the three following cases are the operations linear-dropout-σ and linear-σ-dropout equivalent?

(a) `self.sigma = nn.ReLU()`

(b) `self.sigma = nn.Sigmoid()`

(c) `self.sigma = nn.LeakyReLU()`

Justify your answers.

Clarification. The Leaky ReLU activation function is defined as

$$\sigma(z) = \begin{cases} z & \text{for } z \geq 0 \\ \alpha z & \text{otherwise,} \end{cases}$$

where α is a fixed parameter (α is not trained) often set to $\alpha = 0.01$.

4. (15 points) Consider the layer

$$\begin{aligned} y &= \sigma(\tilde{y}) \\ \tilde{y} &= Ax + b, \end{aligned}$$

where $x \in \mathbb{R}^{n_{\text{in}}}$ and $y, \tilde{y}, \in \mathbb{R}^{n_{\text{out}}}$. Let σ be the sigmoid, i.e., $\sigma(z) = (1 + e^{-z})^{-1}$. Initialize the weights with $A_{ij} \sim \mathcal{N}(0, \sigma_A^2)$ and $b_i = 0$. Assume the approximations $\sigma(\tilde{y}) \approx \frac{1}{2} + \frac{\tilde{y}}{4}$ and $\sigma'(\tilde{y}) \approx \frac{1}{4}$ are accurate

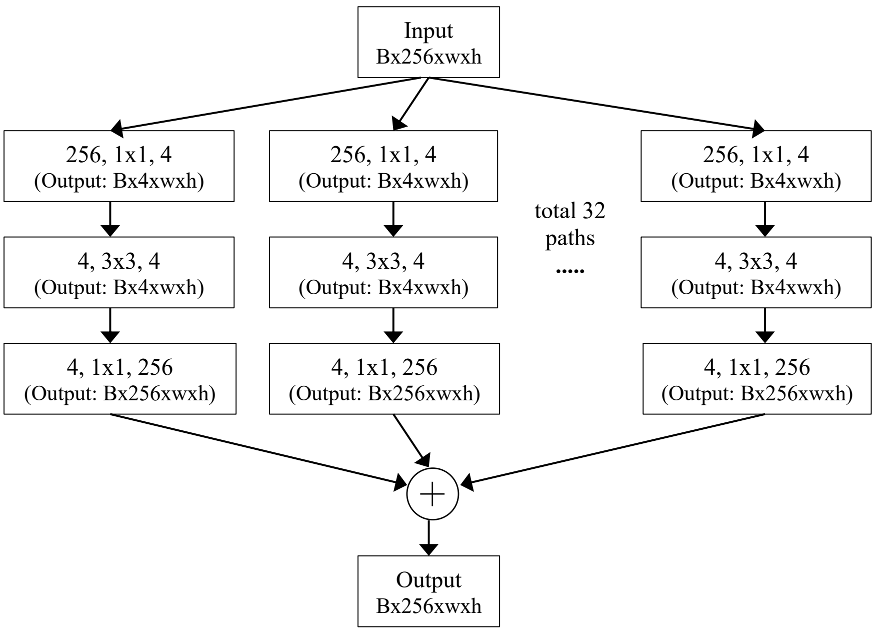
(a) Assume $x_1, \ldots, x_{n_{\text{in}}}$ have mean 1/2, have variance 1, and are uncorrelated. What is the mean and variance of the $y_1, \ldots, y_{n_{\text{out}}}$?

(b) Consider the gradient with respect to some loss $\ell(y)$. Assume $\left(\frac{\partial \ell}{\partial y}\right)_i$ for $i = 1, \ldots, n_{\text{out}}$ have mean 0, have variance 1, are uncorrelated, and are independent from A. What is the mean and variance of $\left(\frac{\partial \ell}{\partial x}\right)_j$ for $j = 1, \ldots, n_{\text{in}}$?

5. (15 points) *Split-transform-merge convolutions.* Consider a series of 1×1, 3×3, 1×1 conv-ReLU operations with 256–128–128–256 channels:

```
class MyConvLayer(nn.Module):
  def __init__(self):
    super(MyConvLayer, self).__init__()
    self.conv1 = nn.Conv2d(256, 128, 1,)
    self.conv2 = nn.Conv2d(128, 128, 3, padding=1)
    self.conv3 = nn.Conv2d(128, 256, 1)
  def forward(self, x):
    out = torch.nn.functional.relu(self.conv1(x))
    out = torch.nn.functional.relu(self.conv2(out))
    out = torch.nn.functional.relu(self.conv3(out))
    return out
```

An issue with this construction, however, is that it has too many trainable parameters. To reduce the number of trainable parameters, we use the following *split-transform-merge* structure: [apply a series of 1×1, 3×3, 1×1 conv-ReLU operations with 256–4–4–256 channels] a total of 32 times and sum the 32 outputs. The following figure illustrates this construction.



To clarify, all convolutions use biases and the strides are all equal to 1. ReLU is not applied after the sum operation.

(a) How many trainable parameters are present in both constructions?

(b) In the following page, implement this convolution with the split-transform-merge structure.

Remark. For part (a), you will perform some lengthy hand calculations. I apologize for making you do this without the aid of a calculator. We will not deduct points for simple calculation mistakes.

```
class STMConvLayer(nn.Module):
  def __init__(self):
    super(STMConvLayer, self).__init__()
    #----------------------------------------
    # Fill in code here















    #----------------------------------------
  def forward(self, x):
    # [apply 1x1conv with 4 output channels
    #  apply 3x3conv with 4 output channels (with padding=1)
    #  apply 1x1conv with 256 output channels] X 32
    # Add all 32 outputs
    #----------------------------------------
    # Fill in code here
















    #----------------------------------------

    return out
```

6. (15 points) *Two forms of Nesterov momentum SGD.* There are two forms for the Nesterov momentum SGD. Form I is

$$\begin{aligned}\psi^{k+1} &= \theta^k - \alpha g^k \\ \theta^{k+1} &= \psi^{k+1} + \beta(\psi^{k+1} - \psi^k)\end{aligned}$$

for $k = 0, 1, \ldots,$ where $\psi^0 = \theta^0$. Form II is

$$\begin{aligned}m^{k+1} &= g^k + \beta m^k \\ v^{k+1} &= \beta m^{k+1} + g^k \\ \theta^{k+1} &= \theta^k - \alpha v^{k+1}\end{aligned}$$

for $k = 0, 1, \ldots,$ where $m^0 = 0$. In both algorithms, g^k represents stochastic gradients computed with θ^k. Form II is the form implemented in PyTorch with the option `Nesterov=True`. Show that the two forms are equivalent in the sense that given a starting point $\theta^0 \in \mathbb{R}^n$ and a sequence of stochastic gradients $g^0, g^1, \ldots \in \mathbb{R}^n$, Forms I and II produce the same $\theta^1, \theta^2, \ldots$ sequence.

7. (15 points) *Backprop for MLP with residual connections.* Let $\sigma\colon \mathbb{R} \to \mathbb{R}$ be a differentiable activation function and consider the following MLP with residual connections

$$\begin{aligned}
y_L &= A_L y_{L-1} + b_L \\
y_{L-1} &= \sigma(A_{L-1} y_{L-2} + b_{L-1}) + y_{L-2} \\
&\vdots \\
y_3 &= \sigma(A_3 y_2 + b_3) + y_2 \\
y_2 &= \sigma(A_2 y_1 + b_2) + y_1 \\
y_1 &= \sigma(A_1 x + b_1),
\end{aligned}$$

where $x \in \mathbb{R}^n$, $A_1 \in \mathbb{R}^{m\times n}$, $b_1 \in \mathbb{R}^m$, $A_\ell \in \mathbb{R}^{m\times m}$, $b_\ell \in \mathbb{R}^m$ for $\ell = 2, \ldots, L-1$, and $A_L \in \mathbb{R}^{1\times m}$, $b_L \in \mathbb{R}^1$. (To clarify, σ is applied element-wise.) For notational convenience, define $y_0 = x$.

(i) Find formulae for

$$\frac{\partial y_\ell}{\partial y_{\ell-1}}$$

for $\ell = 2, \ldots, L$.

(ii) Find formulae for

$$\frac{\partial y_L}{\partial b_\ell}, \qquad \frac{\partial y_L}{\partial A_\ell}$$

for $\ell = 1, \ldots, L$.

(iii) The gradients

$$\frac{\partial y_L}{\partial b_i}, \quad \frac{\partial y_L}{\partial A_i}$$

for $i = 1, \ldots, \ell$ need not vanish when [$A_j = 0$ for some $j \in \{\ell+1, \ldots, L-1\}$] or [$\sigma'(A_j y_{j-1} + b_j) = 0$ for some $j \in \{\ell+1, \ldots, L-1\}$]. Explain why.