**University of Illinois**
**ECE 364** **Profs. Kamalabadi, Snyder**

# Midterm Exam 2

9:30-10:50am, Tuesday, December 10, 2024

**Name:** _____

**NetID:** _____

**Score:** _____

| Problem | Pts. | Score |
|---|---|---|
| 1 | 30 | |
| 2 | 20 | |
| 3 | 20 | |
| 4 | 15 | |
| 5 | 15 | |
| Total | 100 | |

**Netid:** _____

**(30 Pts.)**

1. Select **True** or **False** to each of the following statements.

   (a) Multi-class logistic regression models perform "one-vs-one" classification to generate the final class prediction. **True/<u>False</u>**

   (b) A given multi-layer perceptron model requires inputs of a specific size, i.e. the size of the data cannot vary in the dataset during training or testing. **<u>True</u>/False**

   (c) Pooling functions in convolutional neural networks must be non-linear functions. **True/<u>False</u>**

   (d) Recurrent neural networks are not an example of feed-forward deep net models. **<u>True</u>/False**

   (e) Long short-term memory networks achieve long-term and short-term time-dependencies by applying short and long skip connections between nearby and far away model layers, respectively. **True/<u>False</u>**

   (f) Dropout is a technique for preventing overfitting in deep net models. **<u>True</u>/False**

   (g) Batch norm is applied to normalize the color channels of input images to convolutional neural networks. **True/<u>False</u>**

   (h) A training epoch is one full pass of model updates through the training dataset. **<u>True</u>/False**

   (i) Suppose we have a collection of $N$ 3-dimensional data points. The vectors

   $$v_1 = \begin{bmatrix} 1/2 & \sqrt{3}/2 & 0 \end{bmatrix}^\top,\ v_2 = \begin{bmatrix} 0 & 1/2 & \sqrt{3}/2 \end{bmatrix}^\top,\ v_3 = \begin{bmatrix} \sqrt{3}/2 & 0 & 1/2 \end{bmatrix}^\top$$

   represent a possible set of principal component vectors for the data. **True/<u>False</u>**

   (j) K-Means clustering can represent clusters as ellipsoids (ellipses in higher dimensions) while Gaussian Mixture Models can only represent spherical clusters. **True/<u>False</u>**

**(20 Pts.)**

2. The following parts are unrelated.

   (a) State the three required methods for implementing a Pytorch `Dataset` class and **provide a brief explanation of what each method is used for.**

   A Pytorch `Dataset` class must have:

   i. `__init__` method: This is the constructor for the dataset that initializes any attributes or methods.

   ii. `__len__` method: This method overrides the `len()` function in Python to establish the length of the dataset.

   iii. `__getitem__` method: This method defines what data the dataset returns when accessed with an integer index like an array or list would. For example, a dataset named `dataset` is called with `dataset[idx]` to produce the example in the dataset at index "`idx`".

   (b) Suppose we would like to create separate PyTorch DataLoaders for the training and validation set of a dataset named `my_dataset`. The full dataset has 72 data points and we would like each dataloader to produce four batches on each complete pass through each dataloader. Complete the below function to implement the two dataloaders. Note: you may assume that `train_indices` and `validation_indices` are equal length.

```
from torch.utils.data import DataLoader, SubsetRandomSampler
def split_dataset(my_dataset, training_indices, validation_indices):
    # complete this function in available space below
    return train_loader, validation_loader
```

   The remaining code could be (among other possible implementations) as follows. Note that each split of the data will have 36 data points and thus need a batch size of 9 for each dataloader to produce 4 equal sized batches. (We will not be overly strict about syntax, the main points are batch size, use of sampler, and calling the DataLoader class)

```
train_sampler = SubsetRandomSampler(train_indices)
validation_sampler = SubsetRandomSampler(validation_indices)
train_loader = DataLoader(my_dataset, batch_size=9,
                          sampler=train_sampler)
validation_loader = DataLoader(my_dataset, batch_size=9,
                               sampler=validation_sampler)
```

   (c) For each of the following options, state whether the parameter is required, optional, or not used in order to initialize a `torch.optim.SGD` optimizer. (Circle one for each parameter)

   i. Number of training epochs — **Required/Optional/<u>Not Used</u>**

   ii. Momentum — **Required/<u>Optional</u>/Not Used**

   iii. Learning rate — **<u>Required</u>/Optional/Not Used**

   iv. Trainable model parameters — **<u>Required</u>/Optional/Not Used**

   v. Weight decay — **Required/<u>Optional</u>/Not Used**

   vi. Loss function — **Required/Optional/<u>Not Used</u>**

**(20 Pts.)**

3. Consider the following implementation of a block to be used in a deep neural network. The block takes input $x \in \mathbb{R}^N$ and produces an output vector $z \in \mathbb{R}^D$. Let $z = f(x)$ denote the final output, $\sigma$ be an arbitrary activation function, and $\odot$ be the element-wise product of two vectors.

$$y_1 = \sigma(W_1 x) \odot x$$
$$y_2 = \sigma(W_2 y_1) \odot x$$
$$z = W_3 y_2$$

(a) Each of $W_1, W_2, W_3$ represent parameter matrices of the block. Assuming the use of bias terms, determine (i) the dimensions of $W_1, W_2, W_3$ and (ii) the number of learnable parameters in this block in terms of $N$ and $D$.

i. $W_1 \in \mathbb{R}^{N \times N}, W_2 \in \mathbb{R}^{N \times N}, W_3 \in \mathbb{R}^{D \times N}$.

ii. Number of parameters $= (N \cdot N + N) + (N \cdot N + N) + (D \cdot N + D) = 2N^2 + 2N + ND + D$.

```
import torch.nn as nn
class CustomBlock(nn.Module):
    def __init__(self, N, D, activation_fn):
        super().__init__()
        # Complete constructor in Part (b)
        self.W1 = A
        self.W2 = B
        self.W3 = C
        self.activation = D
    def forward(self, x):
        # Implement the forward function in Part (c)
        return z
```

(b) Using the given inputs to the `CustomBlock` constructor above and the `torch.nn.Linear` class, complete the `__init__` method by stating the lines of code for `A, B, C` and `D`.

i. `A`: nn.Linear(N, N) (bias=True is optional since this is the default)

ii. `B`: nn.Linear(N, N)

iii. `C`: nn.Linear(N, D)

iv. `D`: activation_fn

(c) Implement the `forward` method to implement the function $f(x)$.

```
def forward(self, x):
    # Implement the forward function in Part (c)
    y1 = self.activation(self.W1(x))*x
    y2 = self.activation(self.W2(y1))*x
    z = self.W3(y2)
    return z
```

**Netid:** _____

**(15 Pts.)**

4. Consider the below line of code implementing a convolutional layer.

```
import torch.nn as nn
conv_layer = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5,
                       padding=0, stride=1, bias=False)
```

(a) Suppose the input to the above layer is a feature map of shape $(H, W) = 40 \times 40$. What will be the output shape of this convolutional layer?

The output shape will be $(H, W) = 36 \times 36$.

(b) Determine the number of learnable parameters in the above convolutional layer.

The number of parameters will be $3 \cdot 16 \cdot 5 \cdot 5 = 1{,}200$

(c) Consider the below channel of a feature map in a convolutional neural network.

$$\mathbf{Z} = \begin{bmatrix} 4 & 1 & 8 & 2 & 3 & 7 \\ 0 & 0 & 2 & 5 & 6 & 5 \\ 1 & 4 & 9 & 2 & 5 & 5 \\ 2 & 6 & 2 & 3 & 3 & 6 \\ 0 & 6 & 6 & 7 & 6 & 3 \\ 6 & 2 & 1 & 1 & 9 & 3 \end{bmatrix}$$

Suppose we pass $\mathbf{Z}$ as input to a max pooling layer with $2 \times 2$ kernel size and stride 3. Determine the output of this max pooling layer.

The output of this layer will be

$$\begin{bmatrix} 4 & 6 \\ 6 & 7 \end{bmatrix}$$

**Netid:** _____

**(15 Pts.)**

5. (a) Explain the significance of activation functions to neural networks and why they allow us to build deep networks.

   Activation functions provide non-linear functions inside of neural networks that interrupt the (often) linear operations applied by trainable parameters, e.g. matrix multiplications and convolution. The use of activation functions prevents multiple linear operations from collapsing to a single linear operation and thus allows us to stack multiple separate layers for feature learning. Therefore, activation functions allow us to make neural networks with depth, i.e. deep neural nets.

   (b) Suppose we have an element-wise activation function given by $\text{ReLU}(x) = \max\{0, x\}$ and a mean-pooling operation $g(x)$ of stride 3 with no overlap. Is the below statement true? Please provide justification for your answer.

$$\text{ReLU}(g(x)) = g\left(\text{ReLU}(x)\right)$$

   This statement is **False**. Consider a one-dimensional input of $x = \{-2, 3, -1, 0, -3, 6\}$. Comparing the two sides of the statement:

$$g(x) = \{0, 1\} \implies \text{ReLU}(g(x)) = \{0, 1\}$$
$$\text{ReLU}(x) = \{0, 3, 0, 0, 0, 6\} \implies g(\text{ReLU}(x)) = \{1, 2\}$$

   The above outputs are clearly different in this example.

   (c) Generative adversarial networks (GANs) are a popular type of deep generative neural network composed of two neural networks: a generator network and a discriminator network. Explain how we train the two networks together. Please be specific about how each model is updated at each training step.

   At each training step, the discriminator model receives either a real input example from the training dataset or a fake example produced by the generator. The discriminator predicts if the example is real or fake. The **discriminator is updated to minimize the classification loss** while the the **generator is updated to maximize the classification loss** for any examples it generates for the discriminator.