# Problem Set 2

Jungwoong Yoon

March 3, 2023

## 1 Exercise 1

(a)$\rightarrow$ (d) $\rightarrow$ (f)$\rightarrow$ (c) $\rightarrow$ (b) $\rightarrow$ (e)

## 2 Exercise 2

1. (g), (h), (i)
2. (g), (h), (j)
3. (g)
4. (j)

## 3 Exercise 3

$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$

- For Gaussian A, $p(-7) = \frac{1}{1\sqrt{2\pi}} e^{\frac{-(-7-0)^2}{2(1)^2}} \approx 9.13 \times 10^{-12}$

- For Gaussian B, $p(-7) = \frac{1}{2\sqrt{2\pi}} e^{\frac{-(-7-6)^2}{2(2)^2}} \approx 1.33 \times 10^{-10}$

$p(-7)$ for Gaussian B is higher, thus it is more likely the datapoint was generated by Gaussian B.

# 4 Exercise 4

## 4.1 Loading the Data

```python
# Load the dataset
data = np.load('lab2_dataset.npz')
train_feats = torch.tensor(data['train_feats'])
test_feats = torch.tensor(data['test_feats'])
train_labels = torch.tensor(data['train_labels'])
test_labels = torch.tensor(data['test_labels'])
phone_labels = data['phone_labels']

# Set up the dataloaders
train_dataset = torch.utils.data.TensorDataset(train_feats, train_labels)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)

test_dataset = torch.utils.data.TensorDataset(test_feats, test_labels)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=128, shuffle=False)
```

## 4.2 Defining your model

```python
# Define the model architecture
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        # TODO: Fill in the model's layers here
        self.fnn = nn.Sequential(nn.Linear(11 * 40, 4096), nn.Sigmoid(), nn.Linear(4096, 48))


    def forward(self, x):
        # TODO: Fill in the forward pass here
        x = x.view(x.size(0), -1)
        x = self.fnn(x)

        return x


# Instantiate the model, loss function, and optimizer
model = MyModel()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

## 4.3 Implementing the training loop

```python
def train_network(model, train_loader, criterion, optimizer):
    # TODO: fill in
    print("Begin training network")
    for epoch in range(10):
        for i, (inputs, labels) in enumerate(train_loader, 0):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

    print("Training finished.\n")
```

### 4.4 Evaluating and improving the model's performance

```
label_list = []
correct_count = defaultdict(int)

def test_network(model, test_loader):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)

            # For computing accuracy for each class
            label_list.append(labels)
            for i, label in enumerate(labels):
                if predicted[i] == label:
                    correct_count[phone_labels[label]] += 1

            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print('Test accuracy: %d %%' % (100 * correct / total))

train_network(model, train_loader, criterion, optimizer)
test_network(model, test_loader)
```

```
Begin training network
Training finished.

Test accuracy: 58 %
```

```
labels_counter = Counter(torch.cat(label_list).tolist())
total_acc = 0.0
individual_acc = {}

for k, v in labels_counter.items():
    individual_acc[phone_labels[k]] = 100 * correct_count[phone_labels[k]] / v

three_highest = Counter(individual_acc).most_common(3)
three_lowest = Counter(individual_acc).most_common()[-3:]

print(three_highest)
print(three_lowest)
print()

phonemes_to_check = ['sh', 'p', 'm', 'r', 'ae']
for phoneme in phonemes_to_check:
    print("Accruacy for {}: {}".format(phoneme, individual_acc[phoneme]))
```

```
[('sh', 90.0), ('sil', 89.0), ('s', 88.0)]
[('uh', 27.0), ('ih', 26.0), ('zh', 20.54794520547945)]

Accruacy for sh: 90.0
Accruacy for p: 41.0
Accruacy for m: 66.0
Accruacy for r: 59.0
Accruacy for ae: 57.0
```

1. Final accuracy: 58 %
2. Three phoneme classes with the highest accuracy

- 'sh' - 90 %
- 'sil' - 89 %
- 's' - 88 %

3. Three phoneme classes with the lowest accuracy

   - 'uh' - 27 %
   - 'ih' - 26 %
   - 'zh' - 21 %

4. most commonly mis-classified phoneme classes is 's'. This makes sense because both 'sh' and 's' seem to be unvoiced fricatives but just have different place of articulation.

5. Same reason as the previous question, they all make sense because each of phoneme and its most commonly mis-classified phoneme pairs has different place of articulation but shares the same voicing feature.

   - 'p' - 'k'
   - 'm' - 'n'
   - 'r' - 'er'
   - 'ae' - 'eh'

   - Accruacy for 'sh': 90 %
   - Accruacy for 'p': 41 %
   - Accruacy for 'm': 66 %
   - Accruacy for 'r': 59 %
   - Accruacy for 'ae': 57 %