Name: Marco Ravelo

Problem Set 2: Acoustic Models
The University of Texas at Austin
CS395T: Spoken Language Technologies / Fall 2022
Instructor: David Harwath

---

1. Write down the logical sequence of states that follow from (a):

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

2.1. $\sum_1 = \sum_2$: $(g),\ (h),\ (i)$

2.2. $For\ both\ \sum_1\ and\ \sum_2,\ j \neq k \Rightarrow \sigma_{jk} = 0$: $(g),\ (h),\ (j)$

2.3. $\sum_1 = \sum_2 = \sigma\ I$: (g)

2.4. $\sum_1 = \sigma_1\ I\ and\ \sum_2 = \sigma_2\ I\ for\ some\ \sigma_1,\ \sigma_2$: $(g),\ (j)$

---

3. Given the datapoint x = −7, is it more likely that the datapoint was generated by Gaussian A or by Gaussian B?

x = -7 is 7 standard deviations away from Gaussian A's μ = 0 and 6.5 standard deviations from Gaussian B's μ = 6, therefore it is more likey that the datapoint is from Gaussian B.

4.1. Loading the data:

```
data = np.load('lab2_dataset.npz')
# training and testing features
train_feats = torch.tensor(data['train_feats'], requires_grad=True)
test_feats = torch.tensor(data['test_feats'], requires_grad=True)
# training and testing labels
train_labels = torch.tensor(data['train_labels'])
test_labels = torch.tensor(data['test_labels'])
# phonemes
phone_labels = data['phone_labels']

# set up dataloaders
train_dataset = torch.utils.data.TensorDataset(train_feats, train_labels)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size = batch_size, shuffle =
True)
test_dataset = torch.utils.data.TensorDataset(test_feats, test_labels)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size = batch_size, shuffle = False)
```

4.2. Defining the model:

my feed forward model:

```
# 3.1 Feed-Forward Neural Network

class MyFFNN(nn.Module):
    def __init__(self, model_type, input_dim, output_dim):
        super(MyFFNN, self).__init__()
        # store model type
        self.model_type = model_type
        # store input size
        self.input_size = input_dim
        # batch 1
        self.linear1 = nn.Linear(input_dim, 2048)
        self.relu1 = nn.ReLU()
        # batch 2
        self.linear2 = nn.Linear(2048, 2048)
        self.relu2 = nn.ReLU()
        # batch 3
        self.linear3 = nn.Linear(2048, 2048)
        self.relu3 = nn.ReLU()
        # batch 4
        self.linear4 = nn.Linear(2048, 2048)
        self.relu4 = nn.ReLU()
        # batch 5 output
        self.linearOut = nn.Linear(2048, output_dim)

    def forward(self, x):
        # reshape data to work with model
        out = x.reshape(-1, self.input_size)
        # batch 1
        out = self.linear1(out)
        out = self.relu1(out)
        # batch 2
        out = self.linear2(out)
        out = self.relu2(out)
        # batch 3
        out = self.linear3(out)
        out = self.relu3(out)
        # batch 4
        out = self.linear4(out)
        out = self.relu4(out)
        # batch 5 output
        out = self.linearOut(out)
        return out
```

my convolutional model:

```
# 3.2 Convolutional Neural Network

class MyCNN(nn.Module):
    def __init__(self, model_type, output_dim):
        super(MyCNN, self).__init__()
        # store model type
        self.model_type = model_type
        # batch 1
        self.conv1 = nn.Conv2d(1, 128, kernel_size=3)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2)
        # batch 2
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2)
        # batch 3
        self.flatten = nn.Flatten()
        self.linear3 = nn.Linear(1024, 1024)
        self.relu3 = nn.ReLU()
        # batch 4
        self.linear4 = nn.Linear(1024, 1024)
        self.relu4 = nn.ReLU()
        # batch 5 output
        self.linear5 = nn.Linear(1024, output_dim)

    def forward(self, x):
        # reshape input
        x = x[:, None, :, :]
        # batch 1
        out = self.conv1(x)
        out = self.relu1(out)
        out = self.pool1(out)
        # batch 2
        out = self.conv2(out)
        out = self.relu2(out)
        out = self.pool2(out)
        # batch 3
        out = self.flatten(out)
        out = self.linear3(out)
        out = self.relu3(out)
        # batch 4
        out = self.linear4(out)
        out = self.relu4(out)
        # batch 5 output
        out = self.linear5(out)
        return out
```

4.3. Implementing the training loop:

```
# 5. Train the model with stochastic gradient descent, iterating over the training dataset
several times

def train_network(epochs, iterations, current_model, train_loader, criterion, optimizer,
print_iteration):

    startTime = time.time()

    print ("device name: ", torch.cuda.get_device_name(0))
    print ("model.type: ", current_model.model_type)
    print ("model.device: ", next(current_model.parameters()).device)

    current_iteration = 0

    # lists for data collection
    iteration_list = []
    accuracy_list = []
    loss_list = []

    for epoch in range(epochs):
        print ("epoch: ", epoch)
        for i, (inputs, labels) in enumerate(train_loader, 0):
            current_iteration = i + ((epoch) * iterations)
            # forward pass
            outputs = current_model(inputs)
            loss = criterion(outputs, labels)
            # backward pass and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            # calculate accuracy
            if current_iteration % 100 == 0:
                test_accuracy = test_network(current_model, test_loader)
                accuracy_list.append(test_accuracy)
                iteration_list.append(current_iteration)
                loss_list.append(loss.item())
            # print stats during training
            if current_iteration % print_iteration == 0:
                test_accuracy = test_network(current_model, test_loader)
                print(f'\t iteration: {current_iteration}\t loss: {loss.item():.3f}\t accuracy:
{test_accuracy:.3f} %')

            current_iteration += 1
    # print out stats
    print_stats(current_model, iteration_list, accuracy_list, loss_list)
    print ("time elapsed: ", round((time.time() - startTime), 2), " sec")
```

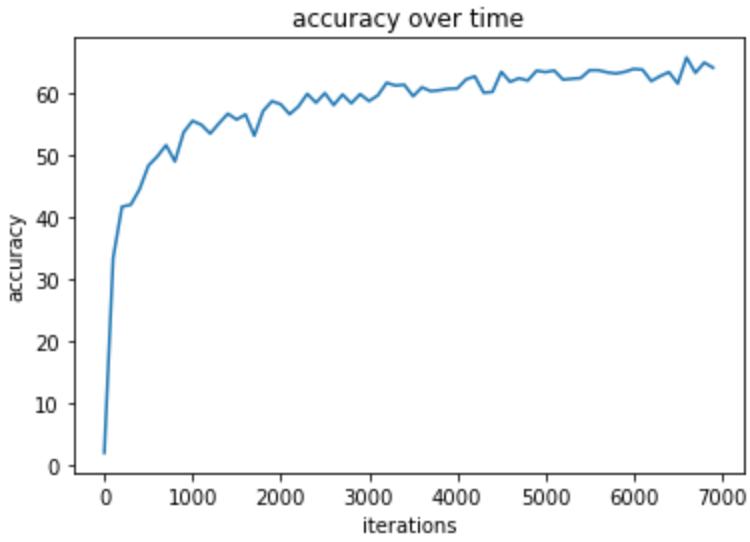
training a feed forward and a convolution model:

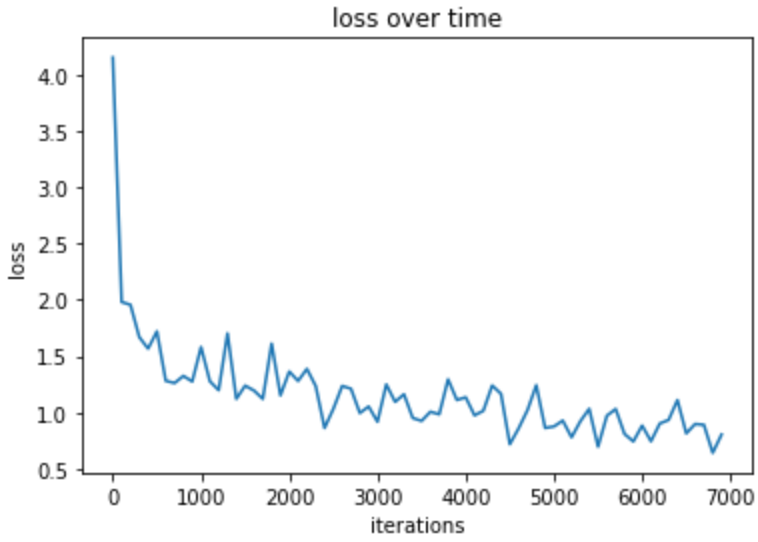
```
# all together now!

epochs = 10
iterations = total_examples / batch_size
iterations = int(iterations)
input_size =  train_feats.shape[1] * train_feats.shape[2]
output_size = 48

print ("epochs: ", epochs)
print ("total_examples: ", total_examples)
print ("iterations per epoch: ", iterations)
print ("batch_size: ", batch_size)
print ("input_size: ", input_size)
print ("output_size: ", output_size)
print ("--------------------------------")


# feed forward
myModel = MyFFNN("FFNN", input_size, output_size)
myModel = myModel.to(device)
myCriterion = nn.CrossEntropyLoss()
mOptimizer = optim.SGD(myModel.parameters(), lr = 0.001, momentum = 0.9)
train_network(epochs, iterations, myModel, train_loader, myCriterion, mOptimizer, int(iterations
/ 3))

print ("--------------------------------")

# convolutional
myModel = MyCNN("CNN", output_size)
myModel = myModel.to(device)
myCriterion = nn.CrossEntropyLoss()
mOptimizer = optim.SGD(myModel.parameters(), lr = 0.001, momentum = 0.9)
train_network(epochs, iterations, myModel, train_loader, myCriterion, mOptimizer, int(iterations
/ 3))
```

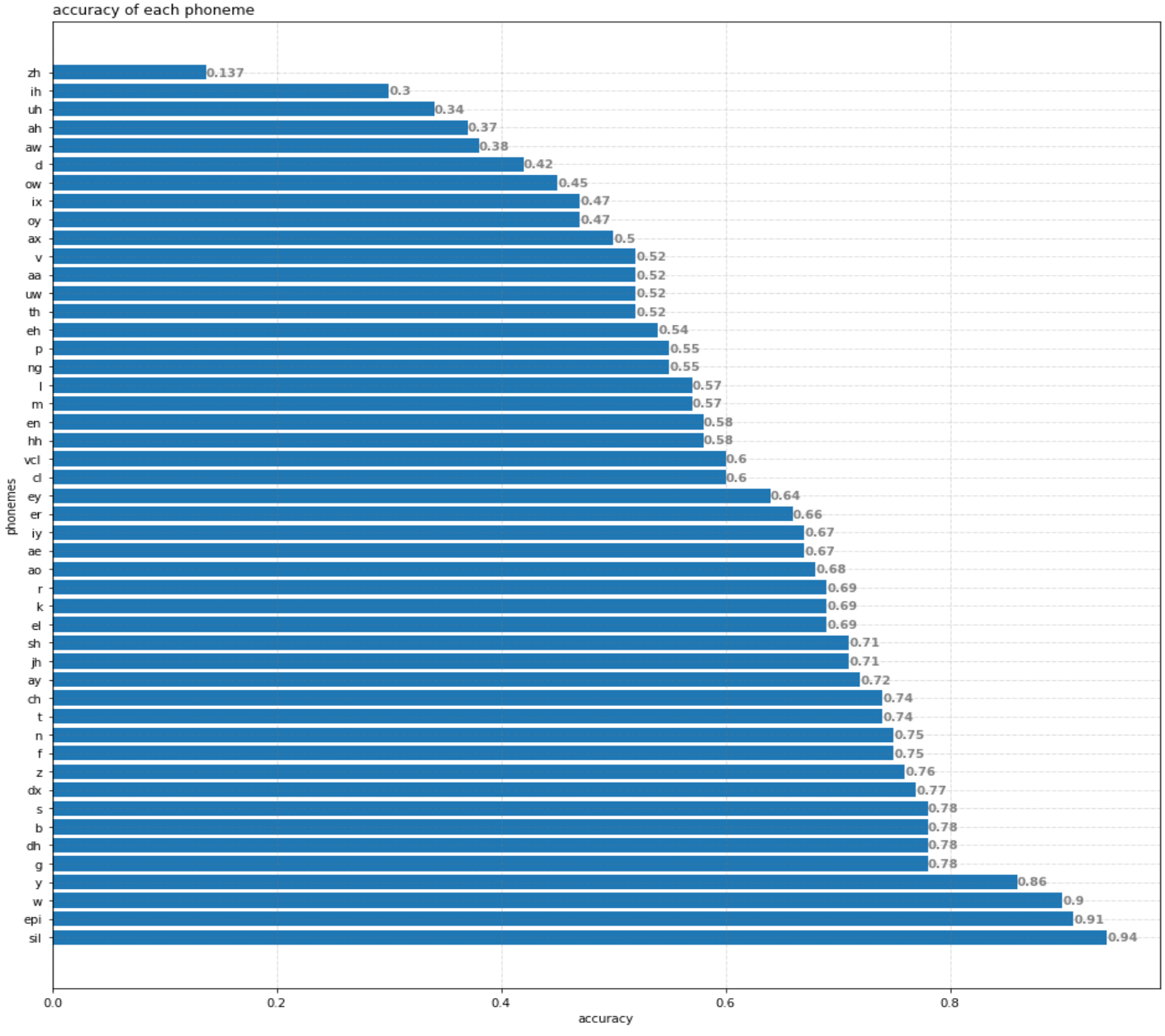
output of the feed forward model being trained:

```
epochs:  10
total_examples:  44730
iterations per epoch:  698
batch_size:  64
input_size:  440
output_size:  48
--------------------------------
device name:  NVIDIA GeForce GTX 1660 Ti
model.type:  FFNN
model.device:  cuda:0
epoch:  0
        iteration: 0           loss: 4.155    accuracy: 2.095 %
        iteration: 232         loss: 1.825    accuracy: 39.849 %
        iteration: 464         loss: 1.372    accuracy: 48.355 %
        iteration: 696         loss: 1.515    accuracy: 51.666 %
epoch:  1
        iteration: 928         loss: 1.434    accuracy: 55.919 %
        iteration: 1160        loss: 1.196    accuracy: 55.269 %
        iteration: 1392        loss: 1.481    accuracy: 55.500 %
epoch:  2
        iteration: 1624        loss: 1.027    accuracy: 55.437 %
        iteration: 1856        loss: 1.347    accuracy: 58.957 %
        iteration: 2088        loss: 1.544    accuracy: 57.867 %
epoch:  3
        iteration: 2320        loss: 1.039    accuracy: 57.846 %
        iteration: 2552        loss: 1.322    accuracy: 59.899 %
        iteration: 2784        loss: 1.388    accuracy: 59.376 %
epoch:  4
        iteration: 3016        loss: 0.786    accuracy: 60.549 %
        iteration: 3248        loss: 1.027    accuracy: 60.549 %
        iteration: 3480        loss: 1.171    accuracy: 60.214 %
epoch:  5
        iteration: 3712        loss: 1.139    accuracy: 61.115 %
        iteration: 3944        loss: 1.225    accuracy: 62.267 %
        iteration: 4176        loss: 1.075    accuracy: 60.842 %
epoch:  6
        iteration: 4408        loss: 0.955    accuracy: 61.596 %
        iteration: 4640        loss: 0.996    accuracy: 60.779 %
        iteration: 4872        loss: 1.264    accuracy: 63.335 %
epoch:  7
        iteration: 5104        loss: 1.241    accuracy: 60.465 %
        iteration: 5336        loss: 1.185    accuracy: 61.806 %
        iteration: 5568        loss: 0.732    accuracy: 63.419 %
epoch:  8
        iteration: 5800        loss: 0.809    accuracy: 63.021 %
        iteration: 6032        loss: 0.967    accuracy: 62.602 %
        iteration: 6264        loss: 0.849    accuracy: 64.027 %
epoch:  9
        iteration: 6496        loss: 0.684    accuracy: 61.073 %
        iteration: 6728        loss: 0.797    accuracy: 64.341 %
        iteration: 6960        loss: 0.813    accuracy: 62.854 %
time elapsed:  912.19  sec
```

graphs for the feed forward model being trained:



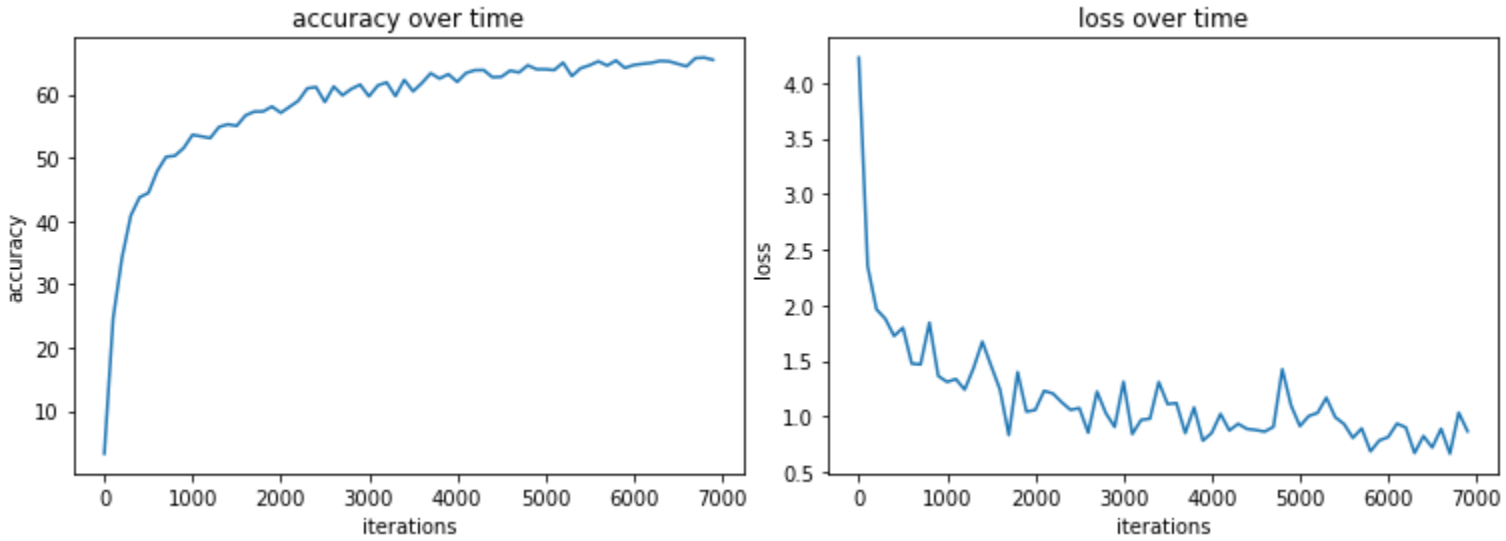


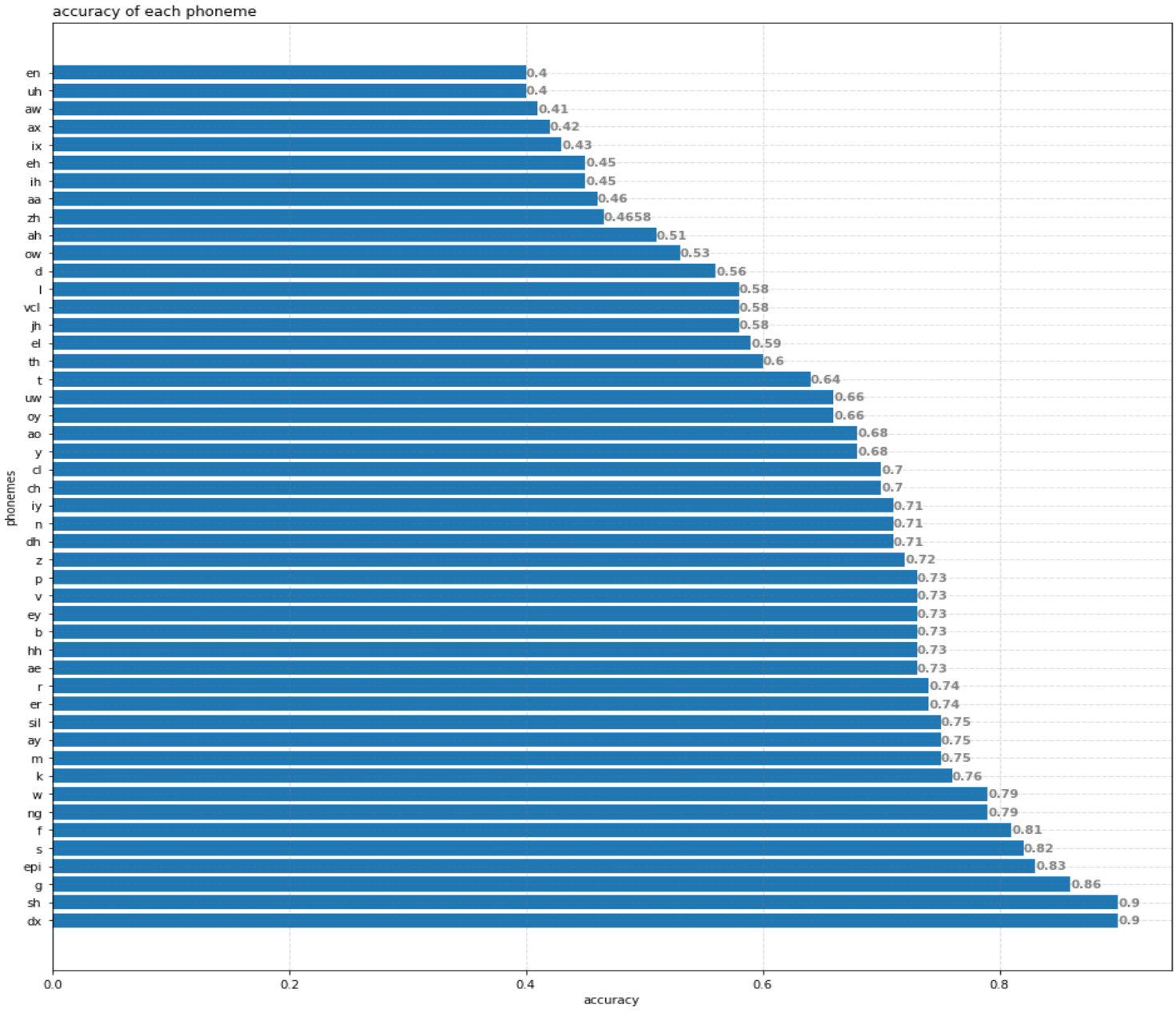


output of the convolution model being trained:

```
--------------------------------
device name:  NVIDIA GeForce GTX 1660 Ti
model.type:  CNN
model.device:  cuda:0
epoch:  0
        iteration: 0           loss: 4.227    accuracy: 3.247 %
        iteration: 232         loss: 1.998    accuracy: 34.737 %
        iteration: 464         loss: 1.892    accuracy: 46.533 %
        iteration: 696         loss: 1.326    accuracy: 51.058 %
epoch:  1
        iteration: 928         loss: 1.517    accuracy: 52.671 %
        iteration: 1160        loss: 1.459    accuracy: 54.829 %
        iteration: 1392        loss: 1.340    accuracy: 56.359 %
epoch:  2
        iteration: 1624        loss: 1.449    accuracy: 57.218 %
        iteration: 1856        loss: 1.525    accuracy: 57.050 %
        iteration: 2088        loss: 1.042    accuracy: 57.239 %
epoch:  3
        iteration: 2320        loss: 1.378    accuracy: 60.298 %
        iteration: 2552        loss: 0.802    accuracy: 60.821 %
        iteration: 2784        loss: 1.088    accuracy: 60.109 %
epoch:  4
        iteration: 3016        loss: 1.357    accuracy: 60.591 %
        iteration: 3248        loss: 1.213    accuracy: 60.423 %
        iteration: 3480        loss: 0.962    accuracy: 61.596 %
epoch:  5
        iteration: 3712        loss: 1.295    accuracy: 61.995 %
        iteration: 3944        loss: 1.088    accuracy: 63.314 %
        iteration: 4176        loss: 1.308    accuracy: 62.036 %
epoch:  6
        iteration: 4408        loss: 1.077    accuracy: 62.686 %
        iteration: 4640        loss: 0.874    accuracy: 63.503 %
        iteration: 4872        loss: 0.675    accuracy: 62.979 %
epoch:  7
        iteration: 5104        loss: 1.039    accuracy: 63.922 %
        iteration: 5336        loss: 0.738    accuracy: 64.090 %
        iteration: 5568        loss: 0.739    accuracy: 64.530 %
epoch:  8
        iteration: 5800        loss: 0.685    accuracy: 65.347 %
        iteration: 6032        loss: 0.716    accuracy: 65.703 %
        iteration: 6264        loss: 0.845    accuracy: 64.865 %
epoch:  9
        iteration: 6496        loss: 1.119    accuracy: 64.215 %
        iteration: 6728        loss: 0.999    accuracy: 65.326 %
        iteration: 6960        loss: 0.811    accuracy: 65.598 %
time elapsed:  938.54  sec
```

graphs for the convolutional model being trained:





4.4. Evaluating the model's performance:

4.4.1. Report the final accuracy achieved by your model.

$feed\ forward\ model\ accuracy$: ~63% $with$ 10 $epochs$
$convolution\ model\ accuracy$: ~65% $with$ 10 $epochs$

4.4.2. Write some code that computes the accuracy for each dierent phoneme class individually. What are the 3 phoneme classes that your model predicts with the highest accuracy?

$feed\ forward\ model\ top\ phonemes$: (sil), (epi), (w)
$convolution\ model\ top\ phonemes$: (dx), (sh), (g)

4.4.3. What about the 3 classes that have the lowest accuracy?

$feed\ forward\ model\ lowest\ phonemes$: (zh), (ih), (uh)
$convolution\ model\ lowest\ phonemes$: (en), (uh), (aw)

4.4.3. For phoneme segments that have the ground-truth label 'sh', what other phoneme class are they most commonly mis-classied as? Does this make sense? Why or why not?

$The\ phoneme$ "sh" ($also\ known\ as$ /š/) $sounds\ very\ similar\ to$ "zh" ($also\ known\ as$ /ž/). $This\ can\ be\ easily\ illustrated\ by\ comparing\ their\ spectrograms\ when\ they\ are\ spoken.$ $Because\ of\ this,\ the\ model\ will\ often\ mis\text{–}classify$ "zh" as "sh" ($most\ likely\ because$ "sh" $is\ more\ common\ than$ "zh").

4.4.5. 5. Repeat the previous question for the 'p', 'm', 'r', and 'ae' phoneme classes.

$The\ phoneme$ "p" ($/p/$) $will\ be\ mis\text{–}classifed\ with\ phoneme$ "b" ($/b/$).
$The\ phoneme$ "m" ($/m/$) $will\ be\ mis\text{–}classifed\ with\ phoneme$ "n" ($/n/$).
$The\ phoneme$ "r" ($/r/$) $will\ be\ mis\text{–}classifed\ with\ phoneme$ "er" ($/ɝ/$).
$The\ phoneme$ "ae" ($/æ/$) $will\ be\ mis\text{–}classifed\ with\ phoneme$ "a" ($/a/$).

Taking the models to their limits - 100 epochs:

I wanted to see how high an accuracy I could reach with each model. Since I was able to use my GPU to train my models, I decided to run each one with 100 epochs and see what the results were.

```
FFNN accuracy limit
epoch:  99
         iteration: 69136       loss: 0.000     accuracy: 68.217 %
         iteration: 69368       loss: 0.001     accuracy: 68.091 %
         iteration: 69600       loss: 0.001     accuracy: 68.322 %

...

CNN accuracy limit
epoch:  99
         iteration: 69136       loss: 0.000     accuracy: 67.337 %
         iteration: 69368       loss: 0.000     accuracy: 67.232 %
         iteration: 69600       loss: 0.000     accuracy: 67.358 %
```