ECE421

2021

Assignment Five

Chase McDougall

Engineering Science

2023

Question 1
Fit_on_texts loops through the words in the given “messages’ variable and updates the frequency tracking dictionary for each of the words it encounters. Either incrementing the count by 1 or initializing it to 1 if the word is contained within the dictionary keys or not respectively.

Question 2
Takes in a list of words and returns a list of the given words indices. These indices correspond to their placement within the frequency dictionary, where smaller indices correspond to higher frequency elements.

Question 3
Pad_sequences takes in a list of sequences and adds padding to the beginning of the sequence such that all sequences in the returned list are of the same length. This will be the length of the longest sequence unless a length limit is passed into the function. The default value for padding is 0 and the 0th index in the frequency dictionary is reserved for padding, though the padding value can be adjusted.

Question 4
We have a 2D array where each element of the array is a sequence of integers. These integers do not exceed 1999 which is the expected bound of sequence due to our set variables. We see that the shorter sequences have been padded with 0s at the beginning of the sequence.
Therefore the acquired array for messages_train is of the form we expected.

Question 5
Depends on our definition of a “sentence”:
Since we cannot initiate or call using a string as input, we will assume that by “sentence” we mean a sequence of integers representing word indices.
However, even this is not an acceptable input for __init__ of the class.
Therefore the __init__ function will not work if a “sentence” is passed in, as it expects only initial integer inputs.

On initialization the class takes in two integer values which are used to define the dimensions of the initial weight matrix.
Calling the class function will use our input for indexing our initial weight matrix.
Our input sequence will be used to return a new matrix as follows:
Each element of the sequence will correspond to a row in the output matrix.
If the i^{th} element in this sequence is equal to k, then the i^{th} row in the output will be equal to the k^{th} row of the initial weight matrix. If k exceeds the maximum index of the initial weight matrix, then the corresponding output row will be equal to the last row of the initial weight matrix.

Question 6

```
update_gate =
objax.functional.sigmoid(jn.dot(update_w,x)+jn.dot(update_u,state)+update_
b)

            # fill this in r_t
            reset_gate =
objax.functional.sigmoid(jn.dot(reset_w,x)+jn.dot(reset_u,state)+reset_b)

            # fill this in h_hat_t
            output_gate =  objax.functional.tanh(jn.dot(output_w, x) +
jn.dot(output_u, jn.multiply(reset_gate, state)) + output_b)
```

Question 7

```
def cumsum_norm(nums):
  # Calculate Cumulative Sum of nums in array
  return_array = []
  sum = 0
  for num in nums:
    sum += num
    return_array.append(sum)
  return jn.array(return_array)
x = jn.array([1,2,3,4,5])
print('Normal Cumulative Sum:')
print(cumsum_norm(x))

# Now Using lax.scan:
def cumsum_lax(sum, num):
  new_sum = sum + num
  return new_sum, new_sum

output, results = lax.scan(cumsum_lax, 0, x)
print('Lax Cumulative Sum:')
print(results)
```

Output:

```
Normal Cumulative Sum: [1 3 6 10 15]
Lax Cumulative Sum:    [1 3 6 10 15]
```

lax.scan:
```python
def scan(f, init, xs, length=None):
  if xs is None:
    xs = [None] * length
  carry = init
  ys = []
  for x in xs:
    carry, y = f(carry, x)
    ys.append(y)
  return carry, np.stack(ys)
```

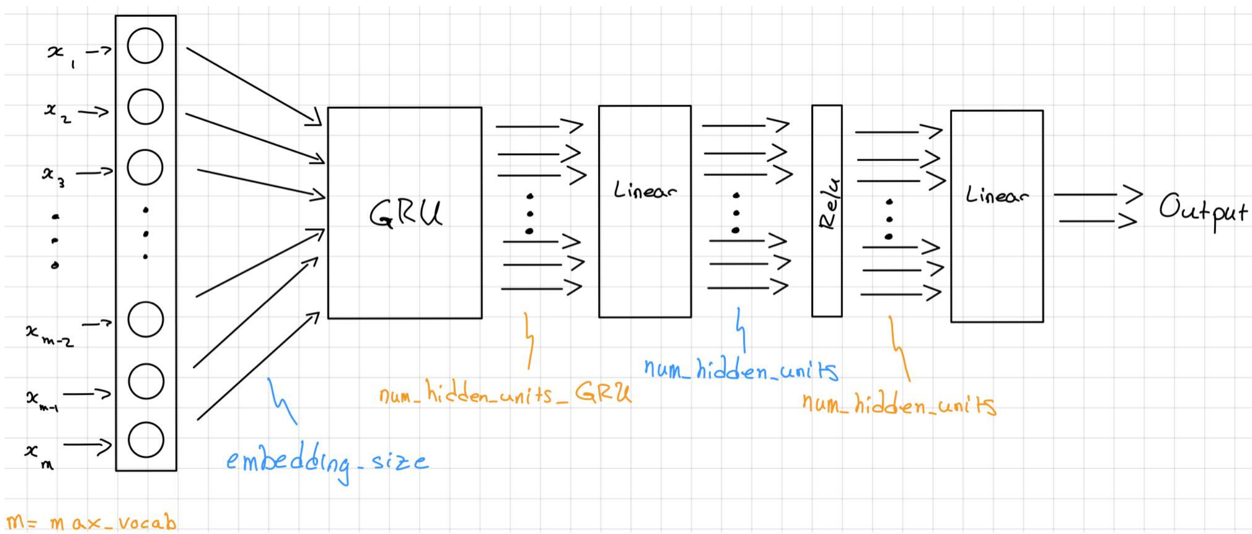
The lax.scan function takes in a minimum of three inputs: a function, an initial value, and an array. The initial value will be used as our starting carry value and the preceding operations will manipulate it during the process. The array x will be iterated through in the passed in function augmenting the provided initial value.

In the given example of a cumulative sum, we see that the lax.scan function can be used to replace a traditional Python loop for improved performance and a reduction in written code. We will notice that the function we pass into lax.scan needs to take in two inputs as well as return two outputs. These inputs represent the current carry as well as the next value from the input array. In the case of the cumulative sum these two values are simply added together and their sum is returned as both the new carry and value at that point in time to be appended to our result array.

Question 8

```
gru_rnn = objax.nn.Sequential([
    Embed(max_vocab, embedding_size),
    GRU(embedding_size, num_hidden_units_GRU),
    objax.nn.Linear(num_hidden_units_GRU, num_hidden_units),
    objax.functional.relu,
    objax.nn.Linear(num_hidden_units, 2)
])
```

Question 9



Question 10

```
opt = objax.optimizer.SGD(gru_rnn.vars())
```

Question 11

```
def train(EPOCHS = 20, BATCH = 20, LEARNING_RATE = 9e-4):
  avg_train_loss_epoch = []
  avg_val_loss_epoch = []
  train_acc_epoch = []
  val_acc_epoch = []

  for epoch in range(EPOCHS):
      avg_train_loss = 0 # (averaged) training loss per batch
      avg_val_loss =  0  # (averaged) validation loss per batch
      train_acc = 0      # training accuracy per batch
      val_acc = 0        # validation accuracy per batch

      # shuffle the examples prior to training to remove correlation
      train_indices = np.arange(len(messages_train))
      np.random.shuffle(train_indices)
      for it in range(0, messages_train.shape[0], BATCH):
          batch = train_indices[it:it+BATCH]
          avg_train_loss += float(train_op(messages_train[batch],
labels_train[batch], LEARNING_RATE)[0]) * len(batch)
          train_prediction = predict(messages_train[batch]).argmax(1)
          train_acc += (np.array(train_prediction).flatten() ==
labels_train[batch]).sum()
      train_acc_epoch.append(train_acc/messages_train.shape[0])
      avg_train_loss_epoch.append(avg_train_loss/messages_train.shape[0])

      # run validation
      val_indices = np.arange(len(messages_valid))
      np.random.shuffle(val_indices)
      for it in range(0, messages_valid.shape[0], BATCH):
          batch = val_indices[it:it+BATCH]
          avg_val_loss += float(loss_function(messages_valid[batch],
labels_valid[batch])) * len(batch)
          val_prediction = predict(messages_valid[batch]).argmax(1)
          val_acc += (np.array(val_prediction).flatten() ==
labels_valid[batch]).sum()
      val_acc_epoch.append(val_acc/messages_valid.shape[0])
      avg_val_loss_epoch.append(avg_val_loss/messages_valid.shape[0])
```

```
      print('Epoch %04d  Training Loss %.2f Validation Loss %.2f Training 
Accuracy %.2f Validation Accuracy %.2f' % (epoch + 1, 
avg_train_loss/messages_train.shape[0], 
avg_val_loss/messages_valid.shape[0], 
100*train_acc/messages_train.shape[0], 
100*val_acc/messages_valid.shape[0]))

  # Print Test Accuracy
  print(f"Test Accuracy: {accuracy(test_data)}")

  #Plot training loss
  plt.title("Train vs Validation Loss")
  plt.plot(avg_train_loss_epoch, label="Train")
  plt.plot(avg_val_loss_epoch, label="Validation")
  plt.xlabel("Epoch")
  plt.ylabel("Loss")
  plt.legend(loc='best')
  plt.show()

  plt.title("Train vs Validation Accuracy")
  plt.plot(train_acc_epoch, label="Train")
  plt.plot(val_acc_epoch, label="Validation")
  plt.xlabel("Epoch")
  plt.ylabel("Accuracy (%)")
  plt.legend(loc='best')
  plt.show()
```

Question 12

Using SGD.

In the following table we have the accuracy after training for 20 epochs.

Validation Accuracy	Test Accuracy
81.40%	82.16%

The following Figure depicts the Training vs Validation accuracies during training over 20 epochs using SGD.

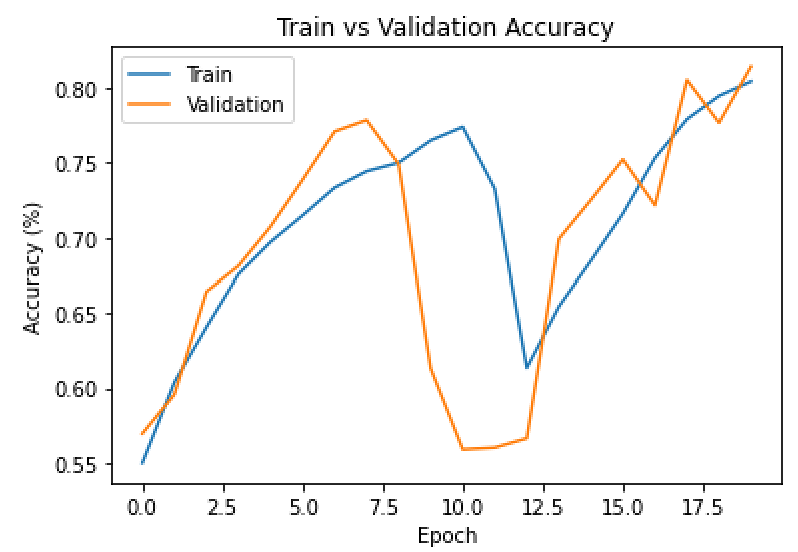


Figure 1: Train vs Validation Accuracy for SGD

We have a relatively smaller generalization gap of $82.16\% - 81.40\% = 0.76\%$, which means that the model generalizes well to unseen data. Especially considering that the larger of the two accuracies is from the accuracy on the test set. Ie unseen data.

Question 13

Reused code from previous parts with the exception of changing opt to:

```
opt2 = objax.optimizer.Adam(gru_rnn2.vars())
```

Question 14

Using Adam.

In the following table we have the accuracy after training for 20 epochs.

Validation Accuracy	Test Accuracy
88.30%	87.28%

The following Figure depicts the Training vs Validation accuracies during training over 20 epochs using Adam.

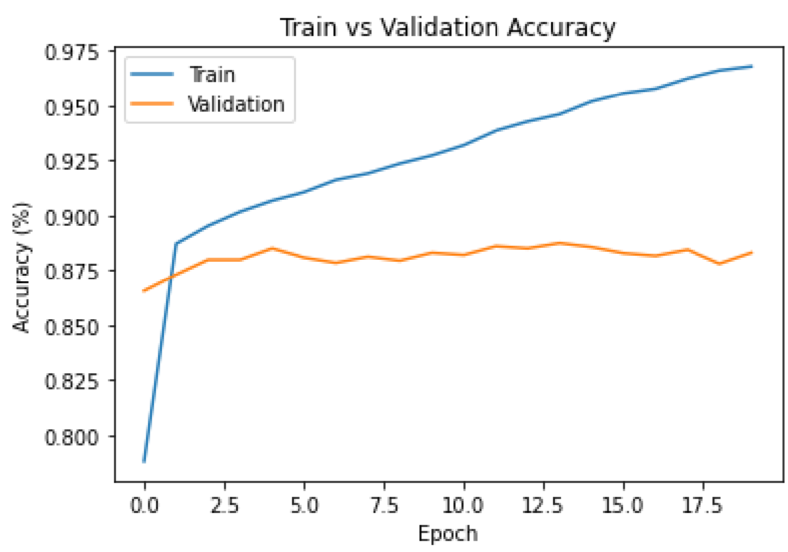


Figure 2: Train vs Validation Accuracy for Adam

We now see a generalization gap of 1.02% with Validation accuracy now being greater than Test accuracy. While this is still a relatively small gap meaning our model generalizes well, it is now in the opposite direction of what we found with SGD. Furthermore, it is worth noting that while we see large fluctuations in accuracy with SGD, these fluctuations are not as present in our Adam training model. Specifically we see that the Validation remains relatively constant, while only the training accuracy climbs. Looking at a single epoch we obtain the following:

Validation Accuracy	Test Accuracy
85.74%	85.70%

It is interesting that when using Adam instead of SGD we are able to see such large accuracy values at such low numbers of epochs. It is also interesting to note that as we train the model over more epochs we see an increase in the generalization gap as it goes from roughly 0.04% to 1.02%.

Question 15

The following formulas illustrate how the Adam optimizer makes adjustments to our w parameters, with the last equation being the overall update step and other equations being used within it.

$$v_k = \beta_1 v_{k-1} + (1-\beta_1)\nabla f(.; w_{k-1})$$
$$s_k = \beta_2 s_{k-1} - (1-\beta_2)(\nabla f(.; w_{k-1}))^2$$
$$\hat{v}_k = \frac{v_k}{1-\beta_1^k}$$
$$\hat{s}_k = \frac{s_k}{1-\beta_2^k}$$
$$w_k = w_{k-1} - \eta\frac{\hat{v}_k}{\sqrt{\hat{s}_k} + \epsilon}$$

Alternatively the following represents the update approach of SGD:

$$w_k = w_{k-1} - \eta\nabla f(.; w_{k-1})$$

We see that both options follow similar conventions with only the term following η differing between them.

Adam is said to be an extension of SGD which combines the advantages of two other advantages:

- Adaptive Gradient Algorithm (AdaGrad)
  Maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
- Root Mean Square Propagation (RMSProp)
  Maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

[1]

Adam maintains the benefits of both of these by not only adapting learning rates based on the average first moment, but also incorporating the average second moment of the gradient.

Question 16

Early Stopping Code:

```
if (avg_val_loss_epoch[len(avg_val_loss_epoch)-1] >
avg_val_loss_epoch[len(avg_val_loss_epoch)-2]):
        if loss_increase:
          loss_seq += 1
        else:
          loss_increase = True
          loss_seq = 1

        if (loss_seq >= max_patience_window):
          print('Epoch %04d  Training Loss %.2f Validation Loss %.2f
Training Accuracy %.2f Validation Accuracy %.2f' % (epoch + 1,
avg_train_loss/messages_train.shape[0],
avg_val_loss/messages_valid.shape[0],
100*train_acc/messages_train.shape[0],
100*val_acc/messages_valid.shape[0]))
          print("Validation Error Increased. BREAK!")
          break
else:
          loss_increase = False
```

Validation Accuracy	Test Accuracy
87.96% at Epoch 4	87.76% at Epoch 10

Using the Adam Optimizer and a patience window of 5 epochs we get the accuracies shown in the above table. The early stopping algorithm stopped training at Epoch 10 after 5 epochs of increasing Validation Loss.

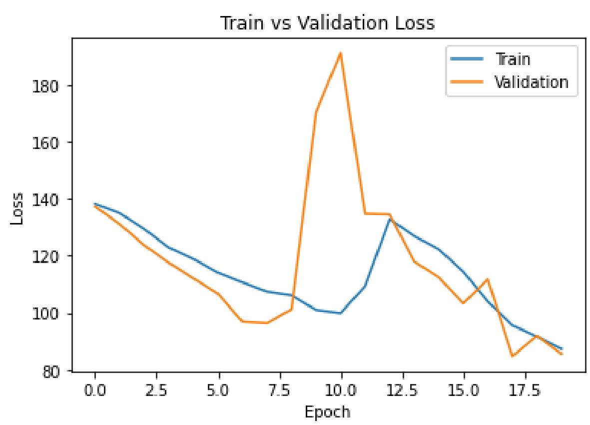
References

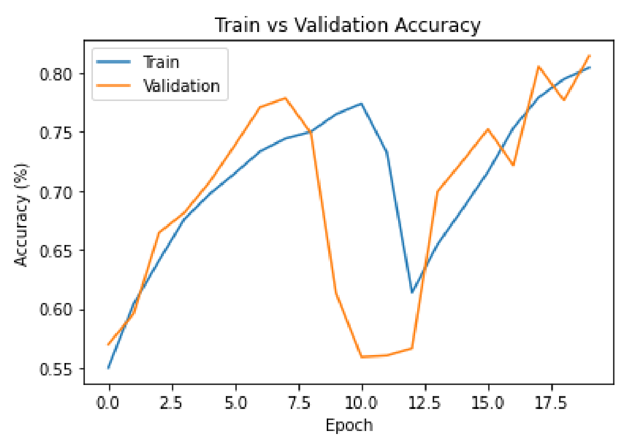
1. J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning," *Machine Learning Mastery*, 12-Jan-2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed: 30-Nov-2021].

Appendix

Question 11/12 Training Info and Plots:

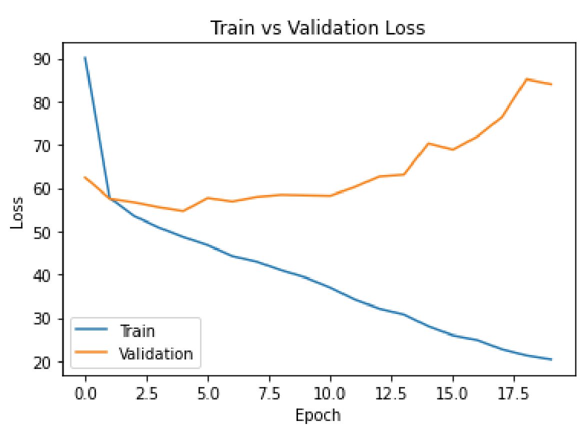
```
Epoch 0001  Training Loss 138.15 Validation Loss 137.26 Training Accuracy 55.02 Validation Accuracy 56.98
Epoch 0002  Training Loss 134.92 Validation Loss 131.05 Training Accuracy 60.40 Validation Accuracy 59.58
Epoch 0003  Training Loss 129.35 Validation Loss 123.64 Training Accuracy 64.07 Validation Accuracy 66.42
Epoch 0004  Training Loss 122.83 Validation Loss 117.52 Training Accuracy 67.57 Validation Accuracy 68.12
Epoch 0005  Training Loss 118.88 Validation Loss 112.06 Training Accuracy 69.70 Validation Accuracy 70.72
Epoch 0006  Training Loss 114.04 Validation Loss 106.69 Training Accuracy 71.46 Validation Accuracy 73.84
Epoch 0007  Training Loss 110.64 Validation Loss 96.85 Training Accuracy 73.32 Validation Accuracy 77.06
Epoch 0008  Training Loss 107.38 Validation Loss 96.30 Training Accuracy 74.41 Validation Accuracy 77.82
Epoch 0009  Training Loss 106.11 Validation Loss 101.11 Training Accuracy 74.98 Validation Accuracy 74.88
Epoch 0010  Training Loss 100.92 Validation Loss 170.34 Training Accuracy 76.47 Validation Accuracy 61.32
Epoch 0011  Training Loss 99.80 Validation Loss 191.08 Training Accuracy 77.36 Validation Accuracy 55.92
Epoch 0012  Training Loss 109.32 Validation Loss 134.68 Training Accuracy 73.24 Validation Accuracy 56.04
Epoch 0013  Training Loss 132.79 Validation Loss 134.45 Training Accuracy 61.35 Validation Accuracy 56.66
Epoch 0014  Training Loss 126.89 Validation Loss 117.93 Training Accuracy 65.43 Validation Accuracy 69.90
Epoch 0015  Training Loss 122.21 Validation Loss 112.42 Training Accuracy 68.46 Validation Accuracy 72.50
Epoch 0016  Training Loss 114.38 Validation Loss 103.24 Training Accuracy 71.59 Validation Accuracy 75.20
Epoch 0017  Training Loss 103.94 Validation Loss 111.68 Training Accuracy 75.30 Validation Accuracy 72.14
Epoch 0018  Training Loss 95.55 Validation Loss 84.72 Training Accuracy 77.88 Validation Accuracy 80.50
Epoch 0019  Training Loss 91.60 Validation Loss 91.84 Training Accuracy 79.44 Validation Accuracy 77.64
Epoch 0020  Training Loss 87.34 Validation Loss 85.57 Training Accuracy 80.40 Validation Accuracy 81.40
```

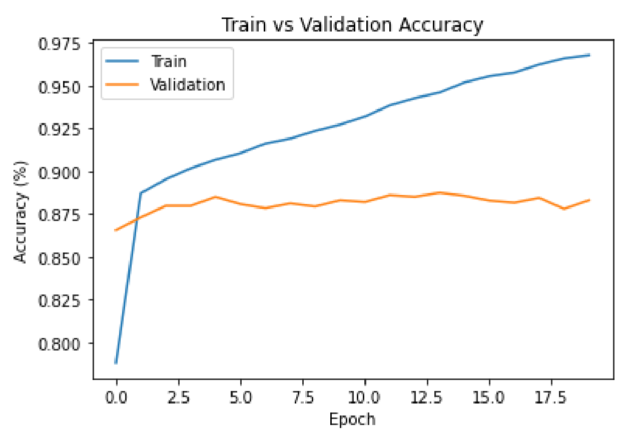




<u>Question 14 Training Info and Plots:</u>

```
Epoch 0001  Training Loss 90.10 Validation Loss 62.42 Training Accuracy 78.83 Validation Accuracy 86.56
Epoch 0002  Training Loss 57.72 Validation Loss 57.53 Training Accuracy 88.71 Validation Accuracy 87.30
Epoch 0003  Training Loss 53.50 Validation Loss 56.69 Training Accuracy 89.52 Validation Accuracy 87.98
Epoch 0004  Training Loss 50.88 Validation Loss 55.56 Training Accuracy 90.16 Validation Accuracy 87.98
Epoch 0005  Training Loss 48.69 Validation Loss 54.65 Training Accuracy 90.66 Validation Accuracy 88.50
Epoch 0006  Training Loss 46.78 Validation Loss 57.71 Training Accuracy 91.05 Validation Accuracy 88.08
Epoch 0007  Training Loss 44.27 Validation Loss 56.88 Training Accuracy 91.60 Validation Accuracy 87.84
Epoch 0008  Training Loss 42.96 Validation Loss 57.95 Training Accuracy 91.89 Validation Accuracy 88.12
Epoch 0009  Training Loss 40.98 Validation Loss 58.45 Training Accuracy 92.34 Validation Accuracy 87.94
Epoch 0010  Training Loss 39.28 Validation Loss 58.33 Training Accuracy 92.71 Validation Accuracy 88.30
Epoch 0011  Training Loss 36.97 Validation Loss 58.21 Training Accuracy 93.18 Validation Accuracy 88.20
Epoch 0012  Training Loss 34.20 Validation Loss 60.24 Training Accuracy 93.83 Validation Accuracy 88.60
Epoch 0013  Training Loss 32.12 Validation Loss 62.67 Training Accuracy 94.25 Validation Accuracy 88.50
Epoch 0014  Training Loss 30.77 Validation Loss 63.10 Training Accuracy 94.58 Validation Accuracy 88.74
Epoch 0015  Training Loss 28.02 Validation Loss 70.31 Training Accuracy 95.17 Validation Accuracy 88.56
Epoch 0016  Training Loss 25.94 Validation Loss 68.84 Training Accuracy 95.54 Validation Accuracy 88.28
Epoch 0017  Training Loss 24.85 Validation Loss 71.90 Training Accuracy 95.75 Validation Accuracy 88.16
Epoch 0018  Training Loss 22.72 Validation Loss 76.40 Training Accuracy 96.22 Validation Accuracy 88.44
Epoch 0019  Training Loss 21.27 Validation Loss 85.13 Training Accuracy 96.58 Validation Accuracy 87.80
Epoch 0020  Training Loss 20.35 Validation Loss 84.01 Training Accuracy 96.76 Validation Accuracy 88.30
Test Accuracy: 87.28%
```





Early Stopping:

```
Epoch 0001  Training Loss 92.26 Validation Loss 60.74 Training Accuracy 77.94 Validation Accuracy 87.06
Epoch 0002  Training Loss 59.01 Validation Loss 58.38 Training Accuracy 88.47 Validation Accuracy 87.26
Epoch 0003  Training Loss 53.33 Validation Loss 58.06 Training Accuracy 89.46 Validation Accuracy 87.50
Epoch 0004  Training Loss 50.57 Validation Loss 56.18 Training Accuracy 90.25 Validation Accuracy 87.96
Epoch 0005  Training Loss 47.90 Validation Loss 55.65 Training Accuracy 90.68 Validation Accuracy 88.02
Epoch 0006  Training Loss 45.91 Validation Loss 56.63 Training Accuracy 91.16 Validation Accuracy 87.68
Epoch 0007  Training Loss 43.90 Validation Loss 57.14 Training Accuracy 91.72 Validation Accuracy 87.62
Epoch 0008  Training Loss 42.54 Validation Loss 59.16 Training Accuracy 92.02 Validation Accuracy 87.64
Epoch 0009  Training Loss 40.61 Validation Loss 59.40 Training Accuracy 92.48 Validation Accuracy 87.92
Epoch 0010  Training Loss 38.47 Validation Loss 60.62 Training Accuracy 92.85 Validation Accuracy 87.32
Validation Error Increased. BREAK!
Test Accuracy: 87.76%
```