Lab 2: Cats vs Dogs

Deadline: Oct 8, 11:59pm

Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted. Quercus submission time will be used, not your local computer time. You can submit your labs as many times as you want before the deadline, so please submit often and early.

Marking TA: Tinglin (Francis) Duan

This lab is partially based on an assignment developed by Prof. Jonathan Rose and Harris Chan.

In this lab, you will train a convolutional neural network to classify an image into one of two classes: "cat" or "dog". The code for the neural networks you train will be written for you, and you are not (yet!) expected to understand all provided code. However, by the end of the lab, you should be able to:

1. Understand at a high level the training loop for a machine learning model.
2. Understand the distinction between training, validation, and test data.
3. The concepts of overfitting and underfitting.
4. Investigate how different hyperparameters, such as learning rate and batch size, affect the success of training.
5. Compare an ANN (aka Multi-Layer Perceptron) with a CNN.

What to submit

Submit a PDF file containing all your code, outputs, and write-up from parts 1-5. You can produce a PDF of your Google Colab file by going to **File > Print** and then save as PDF. The Colab instructions has more information.

Do not submit any other files produced by your code.

Include a link to your colab file in your submission.

Please use Google Colab to complete this assignment. If you want to use Jupyter Notebook, please complete the assignment and upload your Jupyter Notebook file to Google Colab for submission.

With Colab, you can export a PDF file using the menu option `File -> Print` and save as PDF file. **Adjust the scaling to ensure that the text is not cutoff at the margins.**

Colab Link

Include a link to your colab file here

Colab Link: https://drive.google.com/file/d/1TMRj0IwWv7w-pMCGXIjLQkEreErQYw6a/view?usp=sharing

```
import numpy as np
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torch.utils.data.sampler import SubsetRandomSampler
import torchvision.transforms as transforms
```

Part 0. Helper Functions

We will be making use of the following helper functions. You will be asked to look at and possibly modify some of these, but you are not expected to understand all of them.

You should look at the function names and read the docstrings. If you are curious, come back and explore the code *after* making some progress on the lab.

```
###############################################################################
# Data Loading

def get_relevant_indices(dataset, classes, target_classes):
    """ Return the indices for datapoints in the dataset that belongs to the
    desired target classes, a subset of all possible classes.

    Args:
        dataset: Dataset object
        classes: A list of strings denoting the name of each class
        target_classes: A list of strings denoting the name of desired classes
                        Should be a subset of the 'classes'
    Returns:
        indices: list of indices that have labels corresponding to one of the
                 target classes
    """
    indices = []
    for i in range(len(dataset)):
        # Check if the label is in the target classes
        label_index = dataset[i][1] # ex: 3
        label_class = classes[label_index] # ex: 'cat'
        if label_class in target_classes:
            indices.append(i)
    return indices
```

```
def get_data_loader(target_classes, batch_size):
    """ Loads images of cats and dogs, splits the data into training, validation
    and testing datasets. Returns data loaders for the three preprocessed datasets.

    Args:
        target_classes: A list of strings denoting the name of the desired
                        classes. Should be a subset of the argument 'classes'
        batch_size: A int representing the number of samples per batch

    Returns:
        train_loader: iterable training dataset organized according to batch size
        val_loader: iterable validation dataset organized according to batch size
        test_loader: iterable testing dataset organized according to batch size
        classes: A list of strings denoting the name of each class
    """

    classes = ('plane', 'car', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
    ########################################################################
    # The output of torchvision datasets are PILImage images of range [0, 1].
    # We transform them to Tensors of normalized range [-1, 1].
    transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    # Load CIFAR10 training data
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transform)
    # Get the list of indices to sample from
    relevant_indices = get_relevant_indices(trainset, classes, target_classes)

    # Split into train and validation
    np.random.seed(1000) # Fixed numpy random seed for reproducible shuffling
    np.random.shuffle(relevant_indices)
    split = int(len(relevant_indices) * 0.8) #split at 80%

    # split into training and validation indices
    relevant_train_indices, relevant_val_indices = relevant_indices[:split], relevant_
    train_sampler = SubsetRandomSampler(relevant_train_indices)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                               num_workers=1, sampler=train_sampler)
    val_sampler = SubsetRandomSampler(relevant_val_indices)
    val_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                             num_workers=1, sampler=val_sampler)
    # Load CIFAR10 testing data
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=transform)
    # Get the list of indices to sample from
    relevant_test_indices = get_relevant_indices(testset, classes, target_classes)
    test_sampler = SubsetRandomSampler(relevant_test_indices)
    test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                              num_workers=1, sampler=test_sampler)
    return train_loader, val_loader, test_loader, classes
```

```
    return train_loader, val_loader, test_loader, classes

###############################################################################
# Training
def get_model_name(name, batch_size, learning_rate, epoch):
    """ Generate a name for the model consisting of all the hyperparameter values

    Args:
        config: Configuration object containing the hyperparameters
    Returns:
        path: A string with the hyperparameter name and value concatenated
    """
    path = "model_{0}_bs{1}_lr{2}_epoch{3}".format(name,
                                                   batch_size,
                                                   learning_rate,
                                                   epoch)
    return path

def normalize_label(labels):
    """
    Given a tensor containing 2 possible values, normalize this to 0/1

    Args:
        labels: a 1D tensor containing two possible scalar values
    Returns:
        A tensor normalize to 0/1 value
    """
    max_val = torch.max(labels)
    min_val = torch.min(labels)
    norm_labels = (labels - min_val)//(max_val - min_val)
    return norm_labels

def evaluate(net, loader, criterion):
    """ Evaluate the network on the validation set.

     Args:
         net: PyTorch neural network object
         loader: PyTorch data loader for the validation set
         criterion: The loss function
     Returns:
         err: A scalar for the avg classification error over the validation set
         loss: A scalar for the average loss function over the validation set
     """
    total_loss = 0.0
    total_err = 0.0
    total_epoch = 0
    for i, data in enumerate(loader, 0):
        inputs, labels = data
        labels = normalize_label(labels)  # Convert labels to 0/1
        outputs = net(inputs)
        loss = criterion(outputs, labels.float())
        corr = (outputs > 0.0).squeeze().long() != labels
```

```
            total_err += int(corr.sum())
            total_loss += loss.item()
            total_epoch += len(labels)
        err = float(total_err) / total_epoch
        loss = float(total_loss) / (i + 1)
        return err, loss

###############################################################################
# Training Curve
def plot_training_curve(path):
    """ Plots the training curve for a model run, given the csv files
    containing the train/validation error/loss.

    Args:
        path: The base path of the csv files produced during training
    """
    import matplotlib.pyplot as plt
    train_err = np.loadtxt("{}_train_err.csv".format(path))
    val_err = np.loadtxt("{}_val_err.csv".format(path))
    train_loss = np.loadtxt("{}_train_loss.csv".format(path))
    val_loss = np.loadtxt("{}_val_loss.csv".format(path))
    plt.title("Train vs Validation Error")
    n = len(train_err) # number of epochs
    plt.plot(range(1,n+1), train_err, label="Train")
    plt.plot(range(1,n+1), val_err, label="Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Error")
    plt.legend(loc='best')
    plt.show()
    plt.title("Train vs Validation Loss")
    plt.plot(range(1,n+1), train_loss, label="Train")
    plt.plot(range(1,n+1), val_loss, label="Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend(loc='best')
    plt.show()
```

Part 1. Visualizing the Data [7 pt]

We will make use of some of the CIFAR-10 data set, which consists of colour images of size 32x32 pixels belonging to 10 categories. You can find out more about the dataset at https://www.cs.toronto.edu/~kriz/cifar.html

For this assignment, we will only be using the cat and dog categories. We have included code that automatically downloads the dataset the first time that the main script is run.

```
# This will download the CIFAR-10 dataset to a folder called "data"
# the first time you run this code.
train_loader, val_loader, test_loader, classes = get_data_loader(
```

```
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=1) # One image per batch
```

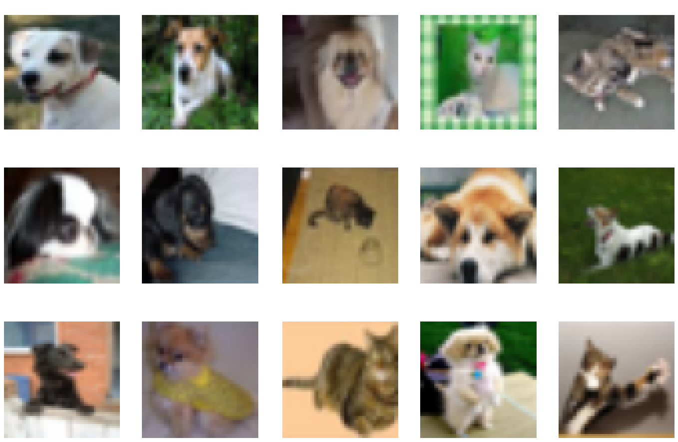
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/ci
                                    170500096/? [00:20<00:00, 61685932.97it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Part (a) -- 1 pt

Visualize some of the data by running the code below. Include the visualization in your writeup.

(You don't need to submit anything else.)

```
import matplotlib.pyplot as plt

k = 0
for images, labels in val_loader:
    # since batch_size = 1, there is only 1 image in `images`
    image = images[0]
    # place the colour channel at the end, instead of at the beginning
    img = np.transpose(image, [1,2,0])
    # normalize pixel intensity values to [0, 1]
    img = img / 2 + 0.5
    plt.subplot(3, 5, k+1)
    plt.axis('off')
    plt.imshow(img)

    k += 1
    if k > 14:
        break
```



Part (b) -- 3 pt

How many training examples do we have for the combined `cat` and `dog` classes? What about

```
print(len(train_loader))
print(len(test_loader))
print(len(val_loader))

# There are 8000 training examples and 2000 each of the testing and validation example
```

```
8000
2000
2000
```

Part (c) -- 3pt

Why do we need a validation set when training our model? What happens if we judge the performance of our models using the training set loss/error instead of the validation set loss/error?

It is important to have a validation set as you want to verify that the model is trained well. There is always the risk of overfitting / underfitting the model, as it may seem like the model works if there is small error or loss. We use the validation set to introduce unseen data to run the model such that we can confirm that it works.

Part 2. Training [15 pt]

We define two neural networks, a `LargeNet` and `SmallNet`. We'll be training the networks in this section.

You won't understand fully what these networks are doing until the next few classes, and that's okay. For this assignment, please focus on learning how to train networks, and how hyperparameters affect training.

```
class LargeNet(nn.Module):
    def __init__(self):
        super(LargeNet, self).__init__()
        self.name = "large"
        self.conv1 = nn.Conv2d(3, 5, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(5, 10, 5)
        self.fc1 = nn.Linear(10 * 5 * 5, 32)
        self.fc2 = nn.Linear(32, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 10 * 5 * 5)
```

```
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        x = x.squeeze(1) # Flatten to [batch_size]
        return x


class SmallNet(nn.Module):
    def __init__(self):
        super(SmallNet, self).__init__()
        self.name = "small"
        self.conv = nn.Conv2d(3, 5, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(5 * 7 * 7, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv(x)))
        x = self.pool(x)
        x = x.view(-1, 5 * 7 * 7)
        x = self.fc(x)
        x = x.squeeze(1) # Flatten to [batch_size]
        return x


small_net = SmallNet()
large_net = LargeNet()
```

Part (a) -- 2pt

The methods `small_net.parameters()` and `large_net.parameters()` produces an iterator of all the trainable parameters of the network. These parameters are torch tensors containing many scalar values.

We haven't learned how how the parameters in these high-dimensional tensors will be used, but we should be able to count the number of parameters. Measuring the number of parameters in a network is one way of measuring the "size" of a network.

What is the total number of parameters in `small_net` and in `large_net`? (Hint: how many numbers are in each tensor?)

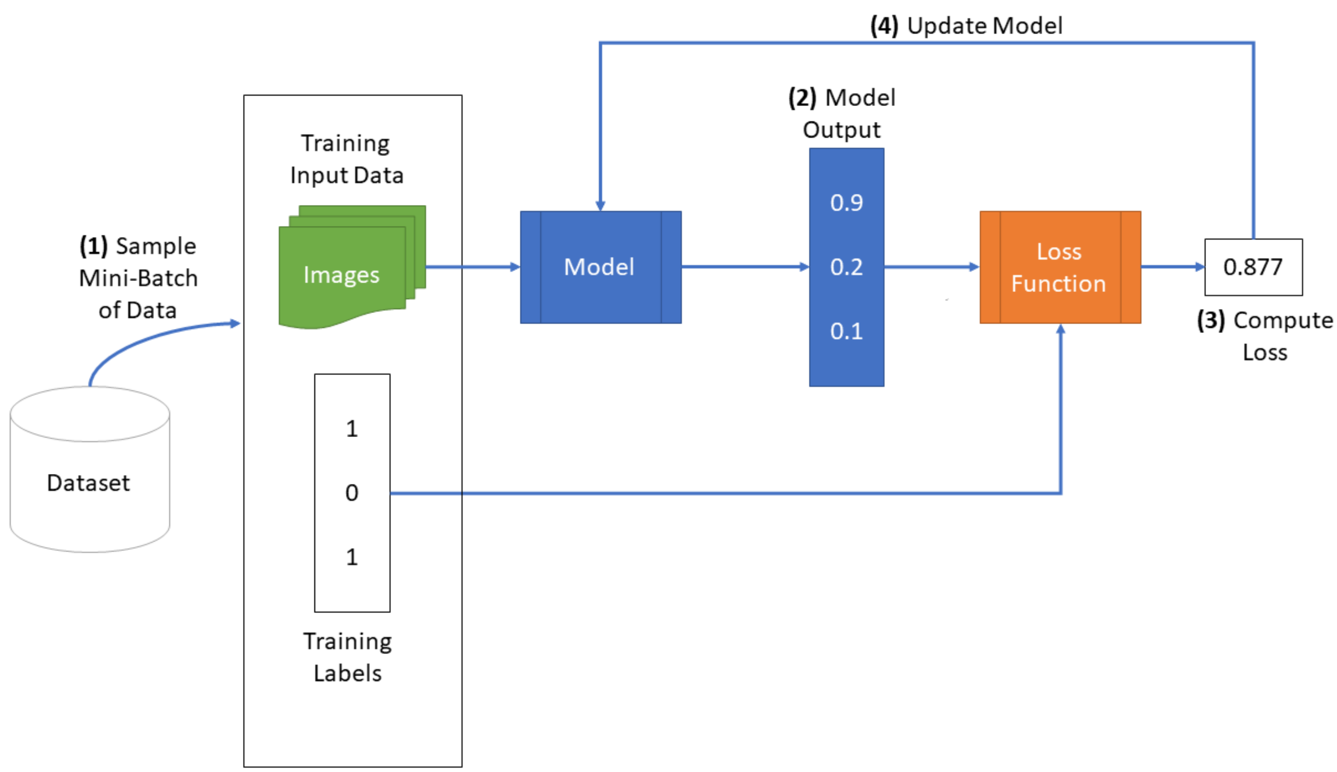
```
accum = 1
for param in small_net.parameters():
    for i in range(0, len(param.shape)):
      accum = accum*param.shape[i]

print("There are " + str(accum) + " parameters in the small_net")

accum = 1
for param in large_net.parameters():
    for i in range(0, len(param.shape)):
      accum = accum*param.shape[i]
```

```
print("There are " + str(accum) + " parameters in the small_net")
```

```
There are 165375 parameters in the small_net
There are 192000000000000 parameters in the small_net
```

The function train_net

The function `train_net` below takes an untrained neural network (like `small_net` and `large_net`) and several other parameters. You should be able to understand how this function works. The figure below shows the high level training loop for a machine learning model:




```
def train_net(net, batch_size=64, learning_rate=0.01, num_epochs=30):
    ########################################################################
    # Train a classifier on cats vs dogs
    target_classes = ["cat", "dog"]
    ########################################################################
    # Fixed PyTorch random seed for reproducible result
    torch.manual_seed(1000)
    ########################################################################
    # Obtain the PyTorch data loader objects to load batches of the datasets
    train_loader, val_loader, test_loader, classes = get_data_loader(
            target_classes, batch_size)
    ########################################################################
    # Define the Loss function and optimizer
    # The loss function will be Binary Cross Entropy (BCE). In this case we
```

```
    # will use the BCEWithLogitsLoss which takes unnormalized output from
    # the neural network and scalar label.
    # Optimizer will be SGD with Momentum.
    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
    ########################################################################
    # Set up some numpy arrays to store the training/test loss/erruracy
    train_err = np.zeros(num_epochs)
    train_loss = np.zeros(num_epochs)
    val_err = np.zeros(num_epochs)
    val_loss = np.zeros(num_epochs)
    ########################################################################
    # Train the network
    # Loop over the data iterator and sample a new batch of training data
    # Get the output from the network, and optimize our loss function.
    start_time = time.time()
    for epoch in range(num_epochs):  # loop over the dataset multiple times
        total_train_loss = 0.0
        total_train_err = 0.0
        total_epoch = 0
        for i, data in enumerate(train_loader, 0):
            # Get the inputs
            inputs, labels = data
            labels = normalize_label(labels) # Convert labels to 0/1
            # Zero the parameter gradients
            optimizer.zero_grad()
            # Forward pass, backward pass, and optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels.float())
            loss.backward()
            optimizer.step()
            # Calculate the statistics
            corr = (outputs > 0.0).squeeze().long() != labels
            total_train_err += int(corr.sum())
            total_train_loss += loss.item()
            total_epoch += len(labels)
        train_err[epoch] = float(total_train_err) / total_epoch
        train_loss[epoch] = float(total_train_loss) / (i+1)
        val_err[epoch], val_loss[epoch] = evaluate(net, val_loader, criterion)
        print(("Epoch {}: Train err: {}, Train loss: {} |"+
               "Validation err: {}, Validation loss: {}").format(
                   epoch + 1,
                   train_err[epoch],
                   train_loss[epoch],
                   val_err[epoch],
                   val_loss[epoch]))
        # Save the current model (checkpoint) to a file
        model_path = get_model_name(net.name, batch_size, learning_rate, epoch)
        torch.save(net.state_dict(), model_path)
    print('Finished Training')
    end_time = time.time()
    elapsed_time = end_time - start_time
```

```
    print("Total time elapsed: {:.2f} seconds".format(elapsed_time))
    # Write the train/test loss/err into CSV file for plotting later
    epochs = np.arange(1, num_epochs + 1)
    np.savetxt("{}_train_err.csv".format(model_path), train_err)
    np.savetxt("{}_train_loss.csv".format(model_path), train_loss)
    np.savetxt("{}_val_err.csv".format(model_path), val_err)
    np.savetxt("{}_val_loss.csv".format(model_path), val_loss)
```

Part (b) -- 1pt

The parameters to the function `train_net` are hyperparameters of our neural network. We made these hyperparameters easy to modify so that we can tune them later on.

What are the default values of the parameters `batch_size`, `learning_rate`, and `num_epochs`?

```
# batch_size = 64
# learning_rate = 0.01
# num_epochs = 30
```

Part (c) -- 3 pt

What files are written to disk when we call `train_net` with `small_net`, and train for 5 epochs? Provide a list of all the files written to disk, and what information the files contain.

```
train_net(small_net, 64, 0.01, 5)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.418125, Train loss: 0.6704066848754883 |Validation err: 0.:
Epoch 2: Train err: 0.364875, Train loss: 0.6426583008766175 |Validation err: 0.:
Epoch 3: Train err: 0.353375, Train loss: 0.6322054648399353 |Validation err: 0.:
Epoch 4: Train err: 0.342375, Train loss: 0.6217929873466491 |Validation err: 0.:
Epoch 5: Train err: 0.33425, Train loss: 0.6123823966979981 |Validation err: 0.3:
Finished Training
Total time elapsed: 18.50 seconds
```

The following files written to disk:

model_small_bs64_lr0.01_epoch0.

model_small_bs64_lr0.01_epoch1

model_small_bs64_lr0.01_epoch2

model_small_bs64_lr0.01_epoch3

model_small_bs64_lr0.01_epoch4

model_small_bs64_lr0.01_epoch4_train_err

model_small_bs64_lr0.01_epoch_train_loss

model_small_bs64_lr0.01_epoch_val_loss
model_small_bs64_lr0.01_epoch_val_err

Part (d) -- 2pt

Train both `small_net` and `large_net` using the function `train_net` and its default parameters. The function will write many files to disk, including a model checkpoint (saved values of model weights) at the end of each epoch.

If you are using Google Colab, you will need to mount Google Drive so that the files generated by `train_net` gets saved. We will be using these files in part (d). (See the Google Colab tutorial for more information about this.)

Report the total time elapsed when training each network. Which network took longer to train? Why?

```
# Since the function writes files to disk, you will need to mount
# your Google Drive. If you are working on the lab locally, you
# can comment out this code.

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
train_net(small_net, 64, 0.01, 30)
## 111.05 seconds to train

train_net(large_net, 64, 0.01, 30)
## 124.64 seconds to train

# The large_net took longer as there are more parameters to train
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.32825, Train loss: 0.6051286149024964 |Validation err: 0.3
Epoch 2: Train err: 0.31775, Train loss: 0.5988569338321685 |Validation err: 0.3
Epoch 3: Train err: 0.3205, Train loss: 0.5974532868862152 |Validation err: 0.33
Epoch 4: Train err: 0.320125, Train loss: 0.5951218860149383 |Validation err: 0.
Epoch 5: Train err: 0.31175, Train loss: 0.5874869222640992 |Validation err: 0.3
Epoch 6: Train err: 0.305375, Train loss: 0.5835547342300414 |Validation err: 0.
Epoch 7: Train err: 0.307, Train loss: 0.5832296054363251 |Validation err: 0.329
Epoch 8: Train err: 0.306125, Train loss: 0.5778174822330475 |Validation err: 0.
Epoch 9: Train err: 0.30175, Train loss: 0.5817056109905243 |Validation err: 0.3
Epoch 10: Train err: 0.298875, Train loss: 0.5740554056167603 |Validation err: 0
Epoch 11: Train err: 0.295625, Train loss: 0.5715680284500122 |Validation err: 0
Epoch 12: Train err: 0.297125, Train loss: 0.5660525941848755 |Validation err: 0
Epoch 13: Train err: 0.29225, Train loss: 0.5667166328430175 |Validation err: 0.
Epoch 14: Train err: 0.293125, Train loss: 0.5630647814273835 |Validation err: 0
Epoch 15: Train err: 0.291625, Train loss: 0.5624129445552826 |Validation err: 0
Epoch 16: Train err: 0.294875, Train loss: 0.5652327179908753 |Validation err: 0
Epoch 17: Train err: 0.293625, Train loss: 0.5638533592224121 |Validation err: 0
Epoch 18: Train err: 0.285125, Train loss: 0.555136979341507 |Validation err: 0.
Epoch 19: Train err: 0.28125, Train loss: 0.5518331458568573 |Validation err: 0.
Epoch 20: Train err: 0.287375, Train loss: 0.5528540678024292 |Validation err: 0
Epoch 21: Train err: 0.277625, Train loss: 0.5527849159240723 |Validation err: 0
Epoch 22: Train err: 0.281875, Train loss: 0.5523663868904114 |Validation err: 0
Epoch 23: Train err: 0.276875, Train loss: 0.5489318742752075 |Validation err: 0
Epoch 24: Train err: 0.275875, Train loss: 0.5477618153095245 |Validation err: 0
Epoch 25: Train err: 0.274375, Train loss: 0.5441374328136445 |Validation err: 0
Epoch 26: Train err: 0.273625, Train loss: 0.542892305135727 |Validation err: 0.
Epoch 27: Train err: 0.278, Train loss: 0.5421670744419098 |Validation err: 0.29
Epoch 28: Train err: 0.278375, Train loss: 0.5446225323677063 |Validation err: 0
Epoch 29: Train err: 0.275625, Train loss: 0.542576758146286 |Validation err: 0.
Epoch 30: Train err: 0.278, Train loss: 0.5443889632225036 |Validation err: 0.29
Finished Training
Total time elapsed: 108.04 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.471625, Train loss: 0.6914801979064942 |Validation err: 0.
Epoch 2: Train err: 0.435125, Train loss: 0.6820778837203979 |Validation err: 0.
Epoch 3: Train err: 0.406, Train loss: 0.6679732670783997 |Validation err: 0.373
Epoch 4: Train err: 0.36675, Train loss: 0.6430274744033814 |Validation err: 0.3
Epoch 5: Train err: 0.350125, Train loss: 0.6276090312004089 |Validation err: 0.
Epoch 6: Train err: 0.330625, Train loss: 0.6105496971607208 |Validation err: 0.
Epoch 7: Train err: 0.322125, Train loss: 0.6005237801074982 |Validation err: 0.
Epoch 8: Train err: 0.30875, Train loss: 0.5852175650596618 |Validation err: 0.3
Epoch 9: Train err: 0.309125, Train loss: 0.5838861713409423 |Validation err: 0.
Epoch 10: Train err: 0.294875, Train loss: 0.5673127889633178 |Validation err: 0
Epoch 11: Train err: 0.284, Train loss: 0.5562010607719421 |Validation err: 0.32
Epoch 12: Train err: 0.27275, Train loss: 0.5404626941680908 |Validation err: 0.
Epoch 13: Train err: 0.27075, Train loss: 0.5338843922615051 |Validation err: 0.
Epoch 14: Train err: 0.262125, Train loss: 0.5188266251087189 |Validation err: 0
Epoch 15: Train err: 0.256125, Train loss: 0.5122005236148834 |Validation err: 0
Epoch 16: Train err: 0.247, Train loss: 0.5076880040168762 |Validation err: 0.29
Epoch 17: Train err: 0.2425, Train loss: 0.49078083753585816 |Validation err: 0.
```

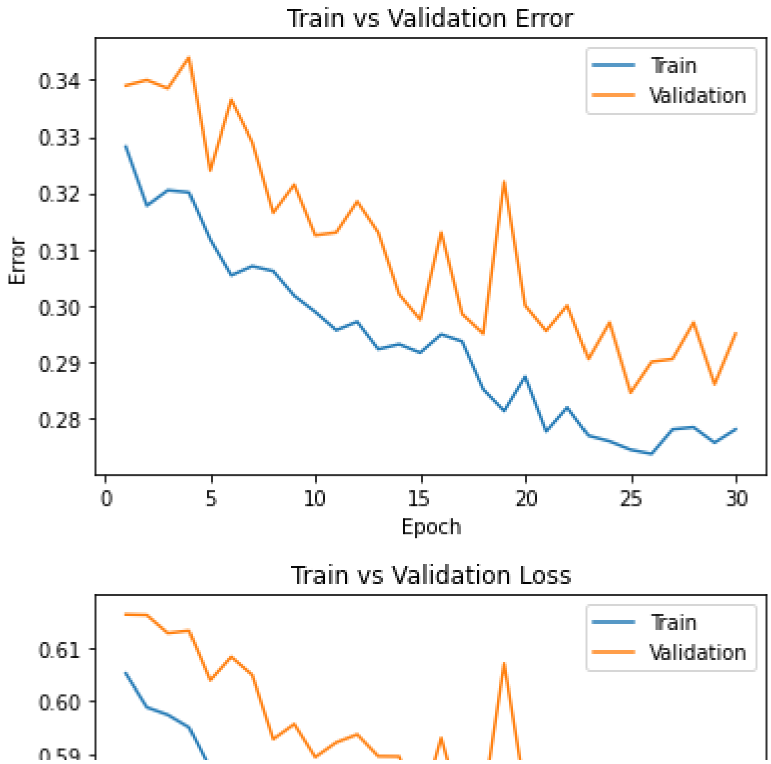
Part (e) - 2pt

Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

Do this for both the small network and the large network. Include both plots in your writeup.

```
Epoch 28: Train err: 0.16225, Train loss: 0.3586689592599869 |Validation err: 0.:
```

```
model_path = get_model_name("small", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)

model_path = get_model_name("large", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```



Part (f) - 5pt

Describe what you notice about the training curve. How do the curves differ for `small_net` and `large_net`? Identify any occurences of underfitting and overfitting.

For the small_net, the training curves for both the error and the loss are lower than the validation curves, meaning less error and loss. In both cases, the training curves also trend downward as the epoch increase to smaller values while the validation curves plateau.

For the large_net, the training curves match up very well with the validation curves while the epoch number is low. As the epochs increase, the two curves tend to diverge, where the training curve continues on a downward linear trend and the validation curve trend in the opposite direction.

As the training error decreases and the validation error does not, these cases are considered to be overfitting. This can be seen in the areas where Epoch > 15. In the areas where Epoch < 10, the model is underfitting as the training error continues to decrease even though the complexity is increasing with the epochs.

Part 3. Optimization Parameters [12 pt]

For this section, we will work with `large_net` only.

Part (a) - 3pt

Train `large_net` with all default parameters, except set `learning_rate=0.001`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *lowering* the learning rate.


```
# Note: When we re-construct the model, we start the training
# with *random weights*. If we omit this code, the values of
# the weights will still be the previously trained values.
large_net = LargeNet()

train_net(large_net, 64, 0.001, 30)
model_path = get_model_name("large", batch_size=64, learning_rate=0.001, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360004425049 |Validation err: 0.4
Epoch 2: Train err: 0.448625, Train loss: 0.6922589735984802 |Validation err: 0.
Epoch 3: Train err: 0.43575, Train loss: 0.6916067390441895 |Validation err: 0.4
Epoch 4: Train err: 0.430125, Train loss: 0.6908613820075988 |Validation err: 0.
Epoch 5: Train err: 0.434125, Train loss: 0.6899198398590088 |Validation err: 0.
Epoch 6: Train err: 0.435875, Train loss: 0.6887419700622559 |Validation err: 0.
Epoch 7: Train err: 0.436625, Train loss: 0.6873781762123108 |Validation err: 0.
Epoch 8: Train err: 0.43725, Train loss: 0.6859267873764038 |Validation err: 0.4
Epoch 9: Train err: 0.424375, Train loss: 0.6844043645858765 |Validation err: 0.
Epoch 10: Train err: 0.42425, Train loss: 0.6828487544059754 |Validation err: 0.
Epoch 11: Train err: 0.42525, Train loss: 0.681236349105835 |Validation err: 0.4
Epoch 12: Train err: 0.419875, Train loss: 0.6796339492797852 |Validation err: 0
Epoch 13: Train err: 0.41475, Train loss: 0.6777917776107788 |Validation err: 0.
Epoch 14: Train err: 0.412375, Train loss: 0.6761110095977784 |Validation err: 0
Epoch 15: Train err: 0.40925, Train loss: 0.674471221446991 |Validation err: 0.4
Epoch 16: Train err: 0.4065, Train loss: 0.6727405357360839 |Validation err: 0.4
Epoch 17: Train err: 0.4015, Train loss: 0.6713058156967163 |Validation err: 0.4
Epoch 18: Train err: 0.39925, Train loss: 0.6696760263442993 |Validation err: 0.
Epoch 19: Train err: 0.400875, Train loss: 0.6679068713188171 |Validation err: 0
Epoch 20: Train err: 0.392375, Train loss: 0.6657861313819885 |Validation err: 0
Epoch 21: Train err: 0.3895, Train loss: 0.6646247744560242 |Validation err: 0.3
Epoch 22: Train err: 0.38875, Train loss: 0.6623678812980652 |Validation err: 0.
Epoch 23: Train err: 0.384, Train loss: 0.6601416163444519 |Validation err: 0.39
Epoch 24: Train err: 0.382625, Train loss: 0.6583918170928955 |Validation err: 0
```

The model takes slightly longer to train than before. We notice with the lower learning rate that the training and validation curves are much more similar. The overall error continues to decrease with the increasing epochs, but even with the lower learning rate of 0.001, the error was still higher.

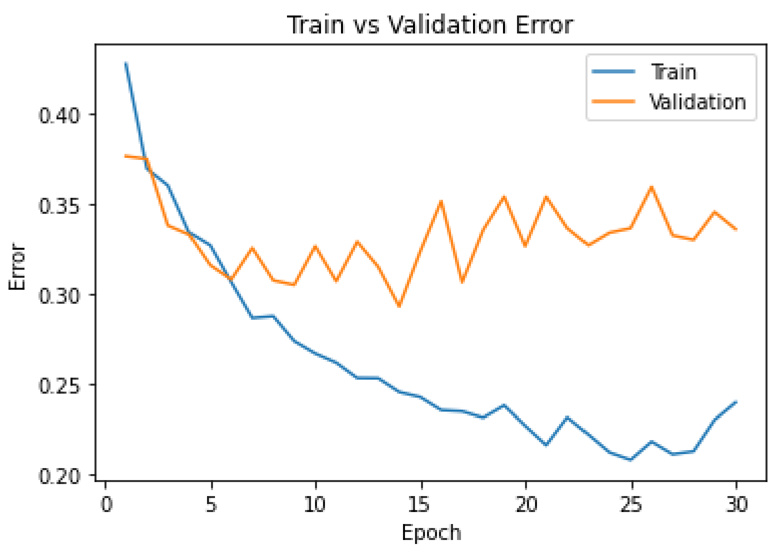
Essentially, having a lower learning rate does not increase efficiency, as the run time is greater and the error is actually greater, so this could be considered to be ineffective.

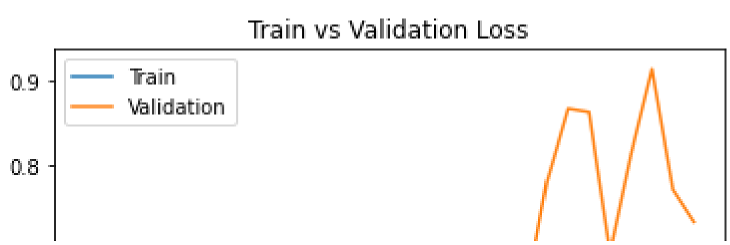
Part (b) - 3pt

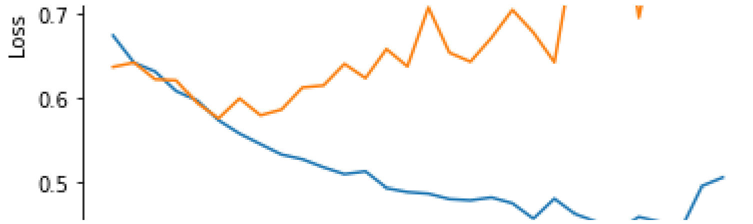
Train `large_net` with all default parameters, except set `learning_rate=0.1`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the learning rate.

```
large_net = LargeNet()
train_net(large_net, 64, 0.1, 30)
model_path = get_model_name("large", batch_size=64, learning_rate=0.1, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4275, Train loss: 0.6742977547645569 |Validation err: 0.37
Epoch 2: Train err: 0.369625, Train loss: 0.6419942178726197 |Validation err: 0.
Epoch 3: Train err: 0.36025, Train loss: 0.6315926456451416 |Validation err: 0.3
Epoch 4: Train err: 0.334125, Train loss: 0.6084355125427247 |Validation err: 0.
Epoch 5: Train err: 0.327, Train loss: 0.5968132178783416 |Validation err: 0.316
Epoch 6: Train err: 0.3065, Train loss: 0.5736211636066437 |Validation err: 0.30
Epoch 7: Train err: 0.286625, Train loss: 0.557759182214737 |Validation err: 0.3
Epoch 8: Train err: 0.287625, Train loss: 0.5451903882026672 |Validation err: 0.
Epoch 9: Train err: 0.27375, Train loss: 0.5328604147434235 |Validation err: 0.3
Epoch 10: Train err: 0.266875, Train loss: 0.5272168016433716 |Validation err: 0
Epoch 11: Train err: 0.26175, Train loss: 0.5176891093254089 |Validation err: 0.
Epoch 12: Train err: 0.25325, Train loss: 0.5095298163890839 |Validation err: 0.
Epoch 13: Train err: 0.253125, Train loss: 0.5128449490070343 |Validation err: 0
Epoch 14: Train err: 0.2455, Train loss: 0.49335162925720216 |Validation err: 0.
Epoch 15: Train err: 0.24275, Train loss: 0.48878711795806884 |Validation err: 0
Epoch 16: Train err: 0.2355, Train loss: 0.48690388107299803 |Validation err: 0.
Epoch 17: Train err: 0.234875, Train loss: 0.48044825553894044 |Validation err:
Epoch 18: Train err: 0.231375, Train loss: 0.4789571635723114 |Validation err: 0
Epoch 19: Train err: 0.238125, Train loss: 0.4823964536190033 |Validation err: 0
Epoch 20: Train err: 0.226875, Train loss: 0.4755194764137268 |Validation err: 0
Epoch 21: Train err: 0.216125, Train loss: 0.45714300179481504 |Validation err:
Epoch 22: Train err: 0.2315, Train loss: 0.48096809661388396 |Validation err: 0.
Epoch 23: Train err: 0.222, Train loss: 0.46243434739112854 |Validation err: 0.3
Epoch 24: Train err: 0.212125, Train loss: 0.4531472542285919 |Validation err: 0
Epoch 25: Train err: 0.208, Train loss: 0.4432792100906372 |Validation err: 0.33
Epoch 26: Train err: 0.218125, Train loss: 0.45924149703979494 |Validation err:
Epoch 27: Train err: 0.211125, Train loss: 0.4533423124551773 |Validation err: 0
Epoch 28: Train err: 0.21275, Train loss: 0.44430617237091063 |Validation err: 0
Epoch 29: Train err: 0.230125, Train loss: 0.4953969626426697 |Validation err: 0
Epoch 30: Train err: 0.23975, Train loss: 0.5056364586353302 |Validation err: 0.
Finished Training
Total time elapsed: 122.79 seconds
```







The model takes nearly the same amount of time to run with the higher learning rate. We see a trend of decreasing validation loss and error up until about epoch = 5, when it then begins to reverse its trend to an upward direction diverging from the training error and loss.

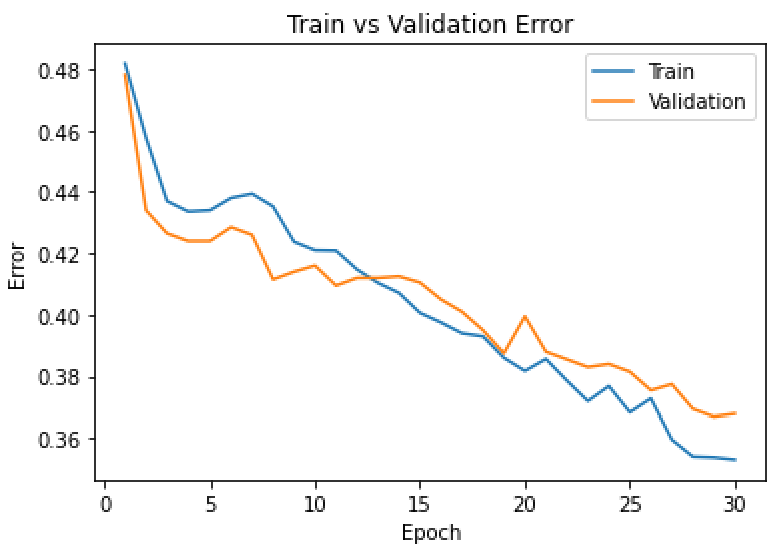
We also see that the change in error and loss as the epochs increase is much more significant with this higher learning rate of 0.1, compared to the learning rate of 0.01.

Part (c) - 3pt

Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=512`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the batch size.

```
large_net = LargeNet()
train_net(large_net, 512, 0.01, 30)
model_path = get_model_name("large", batch_size=512, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```

```
Epoch 7: Train err: 0.439375, Train loss: 0.6866871640086174 |Validation err: 0.
Epoch 8: Train err: 0.43525, Train loss: 0.6849770732223988 |Validation err: 0.4
Epoch 9: Train err: 0.42375, Train loss: 0.6832009218633175 |Validation err: 0.4
Epoch 10: Train err: 0.421, Train loss: 0.6811090856790543 |Validation err: 0.41
Epoch 11: Train err: 0.420875, Train loss: 0.6794028840959072 |Validation err: 0
Epoch 12: Train err: 0.41475, Train loss: 0.676804918795824 |Validation err: 0.4
Epoch 13: Train err: 0.410375, Train loss: 0.6749706864356995 |Validation err: 0
Epoch 14: Train err: 0.407125, Train loss: 0.6730913147330284 |Validation err: 0
Epoch 15: Train err: 0.400625, Train loss: 0.6706849597394466 |Validation err: 0
Epoch 16: Train err: 0.3975, Train loss: 0.6691824123263359 |Validation err: 0.4
Epoch 17: Train err: 0.394, Train loss: 0.6675699539482594 |Validation err: 0.40
Epoch 18: Train err: 0.393, Train loss: 0.6647988669574261 |Validation err: 0.39
Epoch 19: Train err: 0.386, Train loss: 0.6627439819276333 |Validation err: 0.38
Epoch 20: Train err: 0.38175, Train loss: 0.6596207581460476 |Validation err: 0.
Epoch 21: Train err: 0.385625, Train loss: 0.6584848575294018 |Validation err: 0
Epoch 22: Train err: 0.378625, Train loss: 0.6551308929920197 |Validation err: 0
Epoch 23: Train err: 0.372, Train loss: 0.650898166000843 |Validation err: 0.383
Epoch 24: Train err: 0.376875, Train loss: 0.6488463133573532 |Validation err: 0
Epoch 25: Train err: 0.368375, Train loss: 0.6446415856480598 |Validation err: 0
Epoch 26: Train err: 0.372875, Train loss: 0.6429142691195011 |Validation err: 0
Epoch 27: Train err: 0.359625, Train loss: 0.6373121663928032 |Validation err: 0
Epoch 28: Train err: 0.354125, Train loss: 0.6338207013905048 |Validation err: 0
Epoch 29: Train err: 0.353875, Train loss: 0.6311671584844589 |Validation err: 0
Epoch 30: Train err: 0.353125, Train loss: 0.6283805817365646 |Validation err: 0
Finished Training
Total time elapsed: 107.30 seconds
```



Now, by increasing the batch size to 512, the model takes significantly less time to train, it being approximately 20% more efficient. The training and validation curves for both error and less are very similar and they trend consistently negative and appear to be quite smooth.

Part (d) - 3pt

Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=16`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *decreasing* the batch size.

```
large_net = LargeNet()
train_net(large_net, 16, 0.01, 30)
model_path = get_model_name("large", batch_size=16, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.432625, Train loss: 0.6775410465598106 |Validation err: 0.
Epoch 2: Train err: 0.367875, Train loss: 0.6389183864593506 |Validation err: 0.
Epoch 3: Train err: 0.340375, Train loss: 0.6114947981238366 |Validation err: 0.
Epoch 4: Train err: 0.31175, Train loss: 0.5834803532958031 |Validation err: 0.3
Epoch 5: Train err: 0.301125, Train loss: 0.5638278910517692 |Validation err: 0.
Epoch 6: Train err: 0.28475, Train loss: 0.5464649626612663 |Validation err: 0.3
Epoch 7: Train err: 0.270625, Train loss: 0.5266695675551891 |Validation err: 0.
Epoch 8: Train err: 0.254, Train loss: 0.5092709631323814 |Validation err: 0.322
Epoch 9: Train err: 0.245625, Train loss: 0.4968788513541222 |Validation err: 0.
Epoch 10: Train err: 0.231875, Train loss: 0.4726636277139187 |Validation err: 0
Epoch 11: Train err: 0.2195, Train loss: 0.45416154369711875 |Validation err: 0.
Epoch 12: Train err: 0.2135, Train loss: 0.43952479952573775 |Validation err: 0.
Epoch 13: Train err: 0.198875, Train loss: 0.41390844713151453 |Validation err:
Epoch 14: Train err: 0.185375, Train loss: 0.3907764309346676 |Validation err: 0
Epoch 15: Train err: 0.174625, Train loss: 0.37488927601277827 |Validation err:
Epoch 16: Train err: 0.163125, Train loss: 0.3653211060985923 |Validation err: 0
```

With a batch size of 16 this time, the validation curves diverged from the training curves by the time the epoch was about 5. This is not necessarily a good sign as it could signal that the model was overfitting as the training error continues to decrease while the validation error is about 3 times as large at 30 epochs.

Part 4. Hyperparameter Search [6 pt]

Part (a) - 2pt

Based on the plots from above, choose another set of values for the hyperparameters (network, batch_size, learning_rate) that you think would help you improve the validation accuracy. Justify your choice.

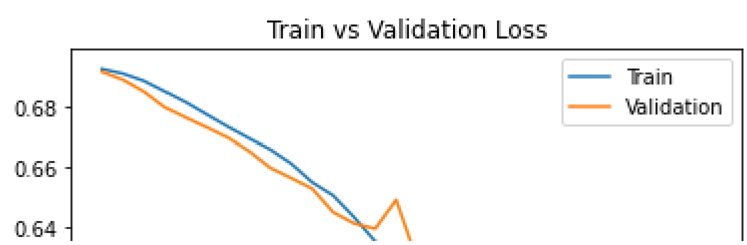
Based on the plots above, I would choose a moderate batch size of 128 with a lower learning rate of 0.005. Lowering the batch size earlier didn't really help and setting it extremely high wasn't amazing either, so I'd like to try a moderate level. Additionally, a lower learning rate would move things in a smaller step which could improve a accuracy

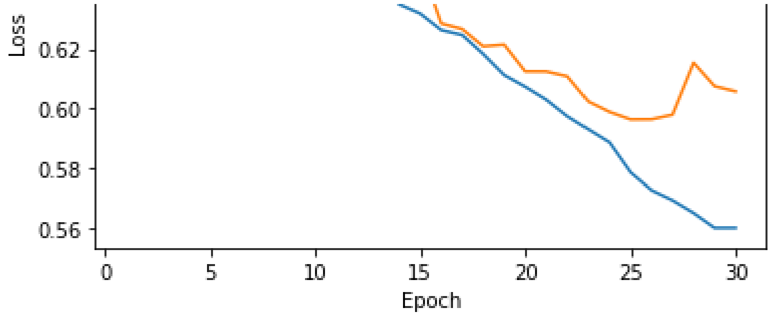
Part (b) - 1pt

Train the model with the hyperparameters you chose in part(a), and include the training curve.

```
large_net = LargeNet()

train_net(large_net, batch_size=128,learning_rate=0.005, num_epochs=30)
model_path = get_model_name("large", batch_size=128, learning_rate=0.005, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.466625, Train loss: 0.6925613265188914 |Validation err: 0.
Epoch 2: Train err: 0.45025, Train loss: 0.6910346150398254 |Validation err: 0.4
Epoch 3: Train err: 0.4305, Train loss: 0.6885887715551589 |Validation err: 0.41
Epoch 4: Train err: 0.430625, Train loss: 0.6850015464283171 |Validation err: 0.
Epoch 5: Train err: 0.420875, Train loss: 0.6813878568391951 |Validation err: 0.
Epoch 6: Train err: 0.415625, Train loss: 0.6772933602333069 |Validation err: 0.
Epoch 7: Train err: 0.405125, Train loss: 0.6732233772202144 |Validation err: 0.
Epoch 8: Train err: 0.4005, Train loss: 0.6694089571634928 |Validation err: 0.40
Epoch 9: Train err: 0.39075, Train loss: 0.665704798130762 |Validation err: 0.39
Epoch 10: Train err: 0.384125, Train loss: 0.6609769766292874 |Validation err: 0
Epoch 11: Train err: 0.374875, Train loss: 0.6547541533197675 |Validation err: 0
Epoch 12: Train err: 0.378625, Train loss: 0.6503568007832482 |Validation err: 0
Epoch 13: Train err: 0.368625, Train loss: 0.6430440866757953 |Validation err: 0
Epoch 14: Train err: 0.353875, Train loss: 0.6349754617327735 |Validation err: 0
Epoch 15: Train err: 0.353375, Train loss: 0.631892790870061 |Validation err: 0.
Epoch 16: Train err: 0.351625, Train loss: 0.6263627977598281 |Validation err: 0
Epoch 17: Train err: 0.34975, Train loss: 0.6247141408541846 |Validation err: 0.
Epoch 18: Train err: 0.343625, Train loss: 0.6183376870458088 |Validation err: 0
Epoch 19: Train err: 0.337, Train loss: 0.6112656962303888 |Validation err: 0.34
Epoch 20: Train err: 0.3355, Train loss: 0.6073562540705242 |Validation err: 0.3
Epoch 21: Train err: 0.32925, Train loss: 0.6030320581935701 |Validation err: 0.
Epoch 22: Train err: 0.327125, Train loss: 0.5974628300893874 |Validation err: 0
Epoch 23: Train err: 0.32125, Train loss: 0.5930387093907311 |Validation err: 0.
Epoch 24: Train err: 0.317125, Train loss: 0.5886963378815424 |Validation err: 0
Epoch 25: Train err: 0.305875, Train loss: 0.5786937678617144 |Validation err: 0
Epoch 26: Train err: 0.3065, Train loss: 0.5727116952812861 |Validation err: 0.3
Epoch 27: Train err: 0.3025, Train loss: 0.5693432674521491 |Validation err: 0.3
Epoch 28: Train err: 0.2995, Train loss: 0.5651203800761511 |Validation err: 0.3
Epoch 29: Train err: 0.29225, Train loss: 0.5601372870187911 |Validation err: 0.
Epoch 30: Train err: 0.292375, Train loss: 0.5601449537844885 |Validation err: 0
Finished Training
Total time elapsed: 109.62 seconds
```





Part (c) - 2pt

Based on your result from Part(a), suggest another set of hyperparameter values to try. Justify your choice.

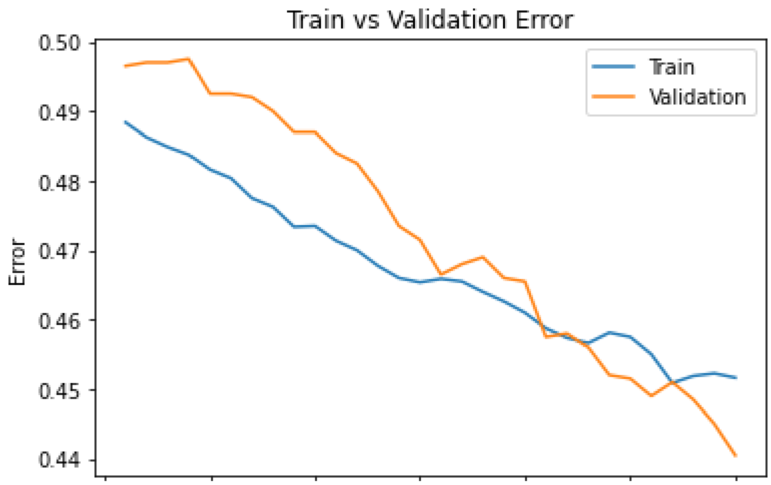
Based on the results above, this time I'd like to try lowering the learning rate a little bit more to see if there is any more improvement. This time I will try 0.0001. The validation curve looks good until about the 25 epoch mark where it slightly diverges and plateaus.

Part (d) - 1pt

Train the model with the hyperparameters you chose in part(c), and include the training curve.

```
large_net = LargeNet()

train_net(large_net,batch_size=128,learning_rate=0.0001,num_epochs=30)
model_path = get_model_name("large", batch_size=128, learning_rate=0.0001, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.488375, Train loss: 0.6930624028993031 |Validation err: 0.
Epoch 2: Train err: 0.48625, Train loss: 0.6930225396913196 |Validation err: 0.4
Epoch 3: Train err: 0.484875, Train loss: 0.6930104787387545 |Validation err: 0.
Epoch 4: Train err: 0.48375, Train loss: 0.6929737159184047 |Validation err: 0.4
Epoch 5: Train err: 0.481625, Train loss: 0.6929437924945165 |Validation err: 0.
Epoch 6: Train err: 0.480375, Train loss: 0.6929201266122242 |Validation err: 0.
Epoch 7: Train err: 0.4775, Train loss: 0.6928898332610963 |Validation err: 0.49
Epoch 8: Train err: 0.47625, Train loss: 0.6928518338808938 |Validation err: 0.4
Epoch 9: Train err: 0.473375, Train loss: 0.6928283364053757 |Validation err: 0.
Epoch 10: Train err: 0.4735, Train loss: 0.6927830217376588 |Validation err: 0.4
Epoch 11: Train err: 0.471375, Train loss: 0.692752634722089 |Validation err: 0.
Epoch 12: Train err: 0.47, Train loss: 0.6927369047725012 |Validation err: 0.482
Epoch 13: Train err: 0.46775, Train loss: 0.6927011656382728 |Validation err: 0.
Epoch 14: Train err: 0.466, Train loss: 0.6926825481747824 |Validation err: 0.47
Epoch 15: Train err: 0.465375, Train loss: 0.6926437039223928 |Validation err: 0
Epoch 16: Train err: 0.465875, Train loss: 0.6926113017021663 |Validation err: 0
Epoch 17: Train err: 0.4655, Train loss: 0.6925884409556313 |Validation err: 0.4
Epoch 18: Train err: 0.464, Train loss: 0.6925563386508397 |Validation err: 0.46
Epoch 19: Train err: 0.462625, Train loss: 0.6925268693575783 |Validation err: 0
Epoch 20: Train err: 0.461, Train loss: 0.6924817524259053 |Validation err: 0.46
Epoch 21: Train err: 0.45875, Train loss: 0.6924625351315453 |Validation err: 0.
Epoch 22: Train err: 0.457375, Train loss: 0.6924205725155179 |Validation err: 0
Epoch 23: Train err: 0.456625, Train loss: 0.6924104567558046 |Validation err: 0
Epoch 24: Train err: 0.458125, Train loss: 0.692374030749003 |Validation err: 0.
Epoch 25: Train err: 0.4575, Train loss: 0.6923392566423567 |Validation err: 0.4
Epoch 26: Train err: 0.455, Train loss: 0.6923072669241164 |Validation err: 0.44
Epoch 27: Train err: 0.450875, Train loss: 0.6922921983022539 |Validation err: 0
Epoch 28: Train err: 0.451875, Train loss: 0.6922409875052316 |Validation err: 0
Epoch 29: Train err: 0.45225, Train loss: 0.6922226188674806 |Validation err: 0.
Epoch 30: Train err: 0.451625, Train loss: 0.6921830574671427 |Validation err: 0
Finished Training
Total time elapsed: 109.62 seconds
```



Part 4. Evaluating the Best Model [15 pt]

Part (a) - 1pt

Choose the **best** model that you have so far. This means choosing the best model checkpoint, including the choice of `small_net` vs `large_net`, the `batch_size`, `learning_rate`, **and the epoch number**.

Modify the code below to load your chosen set of weights to the model object `net`.

```
net = LargeNet()
model_path = get_model_name("large", batch_size=128, learning_rate=0.005, epoch=29)
state = torch.load(model_path)
net.load_state_dict(state)
```

```
<All keys matched successfully>
```

Part (b) - 2pt

Justify your choice of model from part (a).

The large net seemed to perform just slightly better in terms of validation errors, although both nets were pretty similar. Aside from this, the test with the moderate batch size of 128 and learning rate of 0.005 produced good results so these were the parameters I chose.

Part (c) - 2pt

Using the code in Part 0, any code from lecture notes, or any code that you write, compute and report the **test classification error** for your chosen model.

```
# If you use the `evaluate` function provided in part 0, you will need to
# set batch_size > 1
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=128)

testing = nn.BCEWithLogitsLoss()

test_err, test_loss = evaluate(net, test_loader, testing)
print(test_err, test_loss)

val_err, val_loss = evaluate(net, val_loader, testing)
print(val_err, val_loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
0.312 0.5874807238578796
0.3275 0.6071126200258732
```

The test classification error for the model is 0.312

Part (d) - 3pt

How does the test classification error compare with the **validation error**? Explain why you would expect the test error to be *higher* than the validation error.

The test error is slightly less than the validation error which contradicts our expectations of a higher validation error. We fit the hyperparameters to the training data so technically the validation error would be less since the model has been trained to take on this data which hasn't been seen before.

Part (e) - 2pt

Why did we only use the test data set at the very end? Why is it important that we use the test data as little as possible?

When a model is being trained it is very important that the test data is only used when you are actually ready to test the model. You want to feed your model fresh unseen data to see how it really performs in an unbiased way. You don't want to risk over/under fitting with the testing data because then you are just fitting the model to work with this data.

Part (f) - 5pt

How does the your best CNN model compare with an 2-layer ANN model (no convolutional layers) on classifying cat and dog images. You can use a 2-layer ANN architecture similar to what you used in Lab 1. You should explore different hyperparameter settings to determine how well you can do on the validation dataset. Once satisified with the performance, you may test it out on the test data.

Hint: The ANN in lab 1 was applied on greyscale images. The cat and dog images are colour (RGB) and so you will need to flatted and concatinate all three colour layers before feeding them into an ANN.

```
# Below is the ANN from Lab 1

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
import matplotlib.pyplot as plt # for plotting
import torch.optim as optim

torch.manual_seed(1) # set the random seed
```

```
# define a 2-layer artificial neural network
class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.layer1 = nn.Linear(3*32*32, 60)
        self.layer2 = nn.Linear(60,30)
        self.layer3 = nn.Linear(30,1)
        self.name = "pigeon"

    def forward(self, img):
        flattened = img.view(-1,3 * 32 * 32)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        activation2 = F.relu(activation2)
        activation3 = self.layer3(activation2)
        return activation3.squeeze(1)

pigeon = Pigeon()

train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=128)

train_net(pigeon,batch_size=128,learning_rate=0.005,num_epochs=30)

testing = nn.BCEWithLogitsLoss()

test_err, test_loss = evaluate(net, test_loader, testing)
print(test_err)
```