Lab 2: Cats vs Dogs

Deadline: May 22, 9pm

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.

TA: Huan Ling

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.

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.

Colab Link

Include a link to your colab file here

Colab Link: https://colab.research.google.com/drive/16dYAhUku4e7KrJFXd0UWOlfpCqf0vJ6K?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):
    """ Returns 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 the desired
                        classes. Should be a subset of the argument 'classes'
    Returns:
        indices: list of indices that have labels corresponding to one of the
                 target classes
    """
    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))])
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transform)
    # Get the list of indices to sample from
    relevant_train_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_train_indices)
    split = int(len(relevant_train_indices) * 0.8)
    relevant_train_indices, relevant_val_indices = relevant_train_indices[:split], relevant_t
    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)
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=transform)
    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


################################################################################
# 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(
    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/cifar-10-p
                                                                    170499072/? [00:02<00:00, 62187290.52it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Part (a) -- 1 pt

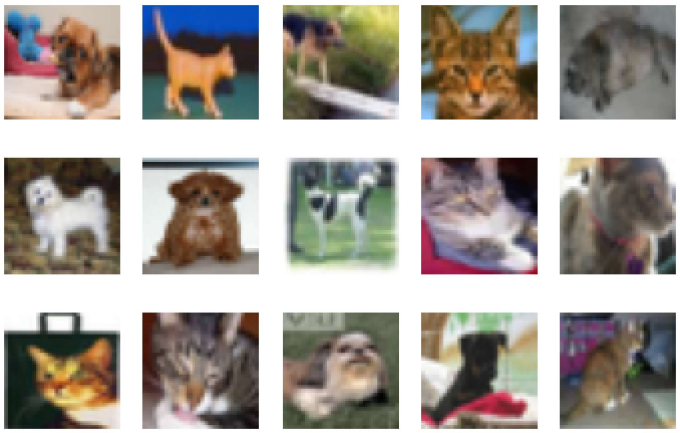
Visualize some of the data by running the code below. Include the visualization in your writeup.

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

```
import matplotlib.pyplot as plt

k = 0
for images, labels in train_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 validation examples? What about test examples?

```
i = 0
for images, labels in train_loader:
  i += 1
print("Training sample size:", i)

i = 0
for images, labels in val_loader:
  i += 1
print("Validation sample size:", i)

i = 0
for images, labels in test_loader:
  i += 1
print("Testing sample size:", i)
```

```
Training sample size: 8000
Validation sample size: 2000
Testing sample size: 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?

```
# We need our validation set to ensure that our model is learning well by giving it unbiased
# If we use test set for this task then the model will see the data during test and it will d

# Its like Homework problems are training data
# Tutorial problems are evaluation data
# Final exam is test data
```

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?)

```
total = 0
```

```
for param in small_net.parameters():
    prod = 1
    for dim in list(param.shape):
      prod *= dim
    total += prod
    print(param.shape, prod)
print(total)
```

```
torch.Size([5, 3, 3, 3]) 135
torch.Size([5]) 5
torch.Size([1, 245]) 245
torch.Size([1]) 1
386
```

```
total = 0
for param in large_net.parameters():
    prod = 1
    for dim in list(param.shape):
      prod *= dim
    total += prod
    print(param.shape, prod)
print(total)
```

```
torch.Size([5, 3, 5, 5]) 375
torch.Size([5]) 5
torch.Size([10, 5, 5, 5]) 1250
torch.Size([10]) 10
torch.Size([32, 250]) 8000
torch.Size([32]) 32
torch.Size([1, 32]) 32
torch.Size([1]) 1
9705
```

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, batch_size=64, learning_rate=0.01, num_epochs=5)
```

```
    Files already downloaded and verified
    Files already downloaded and verified
    Epoch 1: Train err: 0.4405, Train loss: 0.6775434913635254 |Validation err: 0.392, Valic
    Epoch 2: Train err: 0.384875, Train loss: 0.6595904388427735 |Validation err: 0.4045, Va
    Epoch 3: Train err: 0.365125, Train loss: 0.6470252327919006 |Validation err: 0.3625, Va
    Epoch 4: Train err: 0.35375, Train loss: 0.6328100733757019 |Validation err: 0.3595, Val
    Epoch 5: Train err: 0.347875, Train loss: 0.6245655851364136 |Validation err: 0.3545, Va
    Finished Training
    Total time elapsed: 18.68 seconds
```

```
# Training error - np.savetxt("{}_train_err.csv".format(model_path), train_err)
# Training Loss - np.savetxt("{}_train_loss.csv".format(model_path), train_loss)
# Validation error - np.savetxt("{}_val_err.csv".format(model_path), val_err)
```

```
# Validation Loss - np.savetxt("{}_val_loss.csv".format(model_path), val_loss)

# It also has a file for every epoch which stores the weights and biases used in that particu
```

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')


small_net = SmallNet()
train_net(small_net, batch_size=64, learning_rate=0.01, num_epochs=30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.43925, Train loss: 0.6837761998176575 |Validation err: 0.3915, Val
Epoch 2: Train err: 0.381, Train loss: 0.6527652173042298 |Validation err: 0.3935, Valid
Epoch 3: Train err: 0.342125, Train loss: 0.6253658514022827 |Validation err: 0.335, Val
Epoch 4: Train err: 0.324, Train loss: 0.6055522425174713 |Validation err: 0.3375, Valid
Epoch 5: Train err: 0.314, Train loss: 0.5930070235729218 |Validation err: 0.32, Validat
Epoch 6: Train err: 0.299875, Train loss: 0.5810152497291565 |Validation err: 0.326, Val
Epoch 7: Train err: 0.297125, Train loss: 0.5772572119235992 |Validation err: 0.332, Val
Epoch 8: Train err: 0.293625, Train loss: 0.5692502632141113 |Validation err: 0.3085, Va
Epoch 9: Train err: 0.28975, Train loss: 0.5679050085544586 |Validation err: 0.3235, Val
Epoch 10: Train err: 0.289625, Train loss: 0.5611771538257598 |Validation err: 0.318, Va
Epoch 11: Train err: 0.28275, Train loss: 0.5590389950275421 |Validation err: 0.3215, Va
Epoch 12: Train err: 0.278125, Train loss: 0.552915855884552 |Validation err: 0.322, Val
Epoch 13: Train err: 0.280875, Train loss: 0.5536699004173279 |Validation err: 0.31, Val
Epoch 14: Train err: 0.278, Train loss: 0.5482110195159912 |Validation err: 0.314, Valid
Epoch 15: Train err: 0.276375, Train loss: 0.5475579555034638 |Validation err: 0.316, Va
Epoch 16: Train err: 0.280875, Train loss: 0.5506727433204651 |Validation err: 0.312, Va
Epoch 17: Train err: 0.276625, Train loss: 0.549470397233963 |Validation err: 0.307, Val
Epoch 18: Train err: 0.271, Train loss: 0.5432693302631378 |Validation err: 0.3175, Vali
Epoch 19: Train err: 0.271625, Train loss: 0.5406748006343841 |Validation err: 0.3225, \
Epoch 20: Train err: 0.272375, Train loss: 0.5401509728431702 |Validation err: 0.2995, \
Epoch 21: Train err: 0.273625, Train loss: 0.5404087448120117 |Validation err: 0.3095, \
```

```
Epoch 22: Train err: 0.272, Train loss: 0.5392598848342895 |Validation err: 0.3075, Vali
Epoch 23: Train err: 0.27, Train loss: 0.5399896326065063 |Validation err: 0.3035, Valid
Epoch 24: Train err: 0.26875, Train loss: 0.538470311164856 |Validation err: 0.3095, Val
Epoch 25: Train err: 0.269875, Train loss: 0.5359496214389801 |Validation err: 0.3125, V
Epoch 26: Train err: 0.27, Train loss: 0.535960717201233 |Validation err: 0.3, Validatio
Epoch 27: Train err: 0.27075, Train loss: 0.5362830855846406 |Validation err: 0.3145, Va
Epoch 28: Train err: 0.269625, Train loss: 0.5371881670951844 |Validation err: 0.2975, V
Epoch 29: Train err: 0.272125, Train loss: 0.5361146366596222 |Validation err: 0.3025, V
Epoch 30: Train err: 0.264625, Train loss: 0.5357177419662476 |Validation err: 0.32, Val
Finished Training
Total time elapsed: 113.29 seconds
```

```
train_net(large_net, batch_size=64, learning_rate=0.01, num_epochs=30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.32675, Train loss: 0.6035628664493561 |Validation err: 0.3335, Val
Epoch 2: Train err: 0.3205, Train loss: 0.5906253323554993 |Validation err: 0.325, Valid
Epoch 3: Train err: 0.311, Train loss: 0.5773048672676087 |Validation err: 0.3125, Valid
Epoch 4: Train err: 0.3015, Train loss: 0.5659988174438476 |Validation err: 0.3245, Vali
Epoch 5: Train err: 0.28525, Train loss: 0.5473730721473694 |Validation err: 0.307, Vali
Epoch 6: Train err: 0.273625, Train loss: 0.5374099099636078 |Validation err: 0.2925, Va
Epoch 7: Train err: 0.268625, Train loss: 0.5293472776412964 |Validation err: 0.2975, Va
Epoch 8: Train err: 0.263375, Train loss: 0.5205994968414307 |Validation err: 0.3055, Va
Epoch 9: Train err: 0.257, Train loss: 0.5155899045467377 |Validation err: 0.3055, Valid
Epoch 10: Train err: 0.24975, Train loss: 0.5026589710712432 |Validation err: 0.2905, Va
Epoch 11: Train err: 0.240375, Train loss: 0.48654758977890017 |Validation err: 0.289, V
Epoch 12: Train err: 0.22975, Train loss: 0.4714341549873352 |Validation err: 0.2945, Va
Epoch 13: Train err: 0.22775, Train loss: 0.4681701395511627 |Validation err: 0.296, Val
Epoch 14: Train err: 0.219875, Train loss: 0.45684660840034486 |Validation err: 0.292, V
Epoch 15: Train err: 0.2155, Train loss: 0.44868142414093015 |Validation err: 0.282, Val
Epoch 16: Train err: 0.211, Train loss: 0.44487254643440244 |Validation err: 0.291, Vali
Epoch 17: Train err: 0.202, Train loss: 0.4277032092809677 |Validation err: 0.299, Valid
Epoch 18: Train err: 0.193, Train loss: 0.41114472031593324 |Validation err: 0.29, Valid
Epoch 19: Train err: 0.191375, Train loss: 0.40760666036605836 |Validation err: 0.2975,
Epoch 20: Train err: 0.18175, Train loss: 0.39629838228225706 |Validation err: 0.3035, V
Epoch 21: Train err: 0.172125, Train loss: 0.38442246985435485 |Validation err: 0.29, Va
Epoch 22: Train err: 0.1645, Train loss: 0.36488783526420593 |Validation err: 0.292, Val
Epoch 23: Train err: 0.158125, Train loss: 0.35017412078380583 |Validation err: 0.291, V
Epoch 24: Train err: 0.1575, Train loss: 0.34482338225841525 |Validation err: 0.294, Val
Epoch 25: Train err: 0.146625, Train loss: 0.32657413685321807 |Validation err: 0.291, V
Epoch 26: Train err: 0.132375, Train loss: 0.3087524787187576 |Validation err: 0.298, Va
Epoch 27: Train err: 0.132, Train loss: 0.29883666390180585 |Validation err: 0.293, Vali
Epoch 28: Train err: 0.12925, Train loss: 0.2927038335800171 |Validation err: 0.3085, Va
Epoch 29: Train err: 0.1115, Train loss: 0.2640171468257904 |Validation err: 0.3195, Val
Epoch 30: Train err: 0.1245, Train loss: 0.28156155967712404 |Validation err: 0.314, Val
Finished Training
Total time elapsed: 123.37 seconds
```

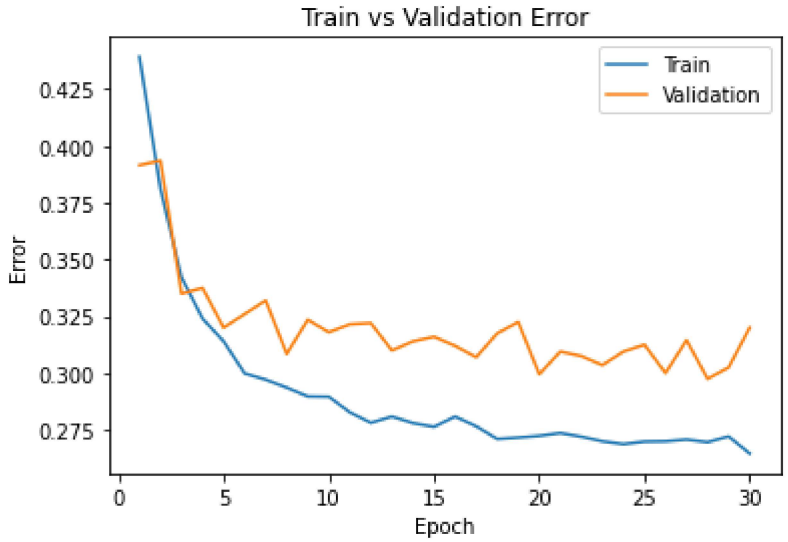
ANS: The total time elapsed in small_net is 113.29 seconds and large_net required 123.37 seconds. As expected the large_net took more time as it has more parameter values to learn

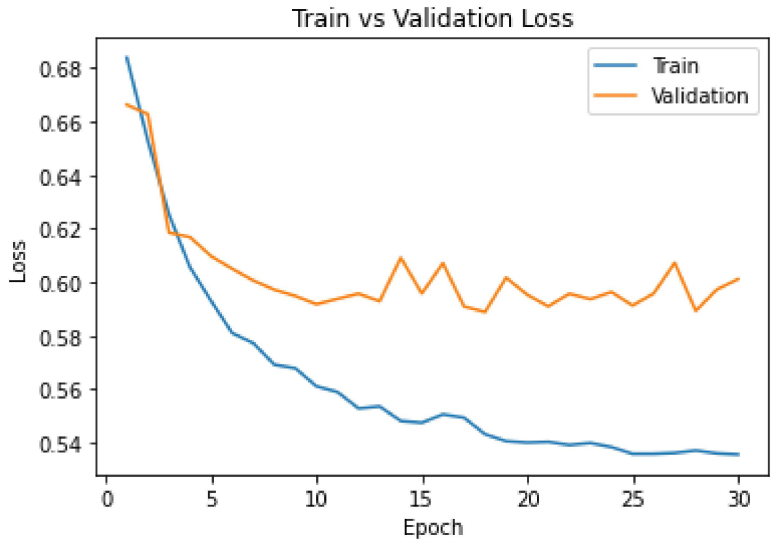
Part (e) - 2pt

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.

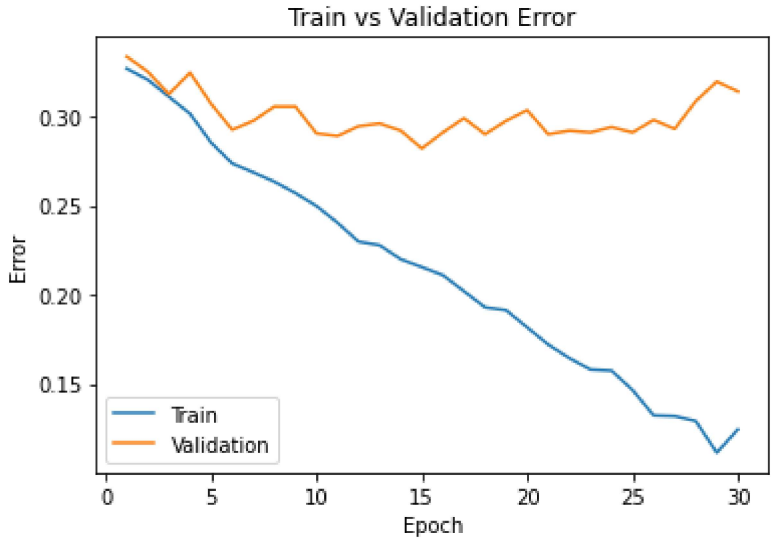
Do this for both the small network and the large network. Include both plots in your writeup.

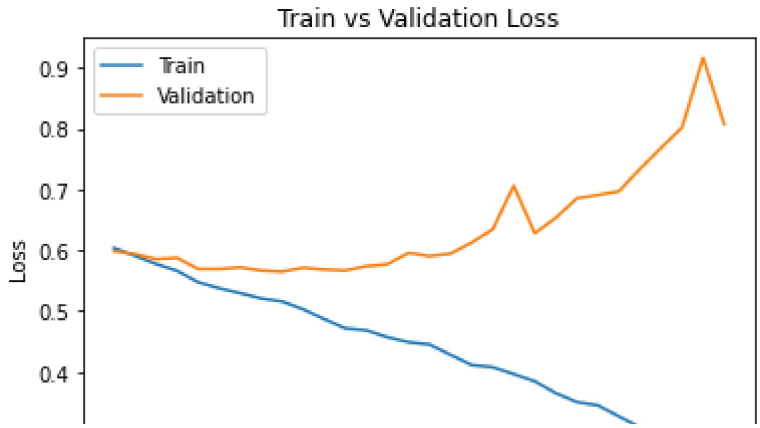
```
model_path = get_model_name("small", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```






```
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.

ANS: In the small net graphs we can see that the validation error and loss show a net decrease with more epochs as does the training error. This indicates that the model is learning. However, for the large_net the validation loss starts increasing around epoch 10, while the training loss continuous to decrease. This is clearly an indication of overfitting as the model is memorizing the training data and not effectively learning the features.

Part 3. Optimization Parameters [12 pt]

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

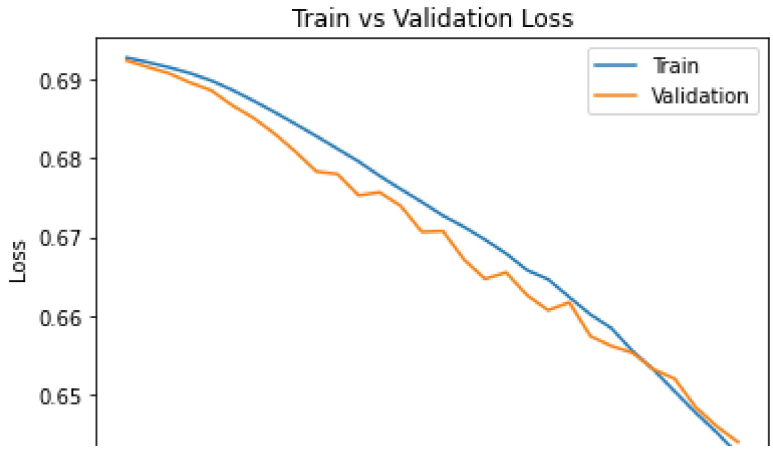
Part (a) - 3pt

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, batch_size=64, learning_rate=0.001, num_epochs=30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360013961792 |Validation err: 0.467, Vali
Epoch 2: Train err: 0.448625, Train loss: 0.6922589712142945 |Validation err: 0.4305, Va
Epoch 3: Train err: 0.43575, Train loss: 0.6916067280769348 |Validation err: 0.4285, Val
Epoch 4: Train err: 0.43, Train loss: 0.690861343383789 |Validation err: 0.424, Validati
Epoch 5: Train err: 0.434125, Train loss: 0.6899195008277893 |Validation err: 0.4195, Va
Epoch 6: Train err: 0.43575, Train loss: 0.6887411961555481 |Validation err: 0.4195, Val
Epoch 7: Train err: 0.437125, Train loss: 0.6873774147033691 |Validation err: 0.4185, Va
Epoch 8: Train err: 0.4375, Train loss: 0.6859278454780579 |Validation err: 0.412, Valid
Epoch 9: Train err: 0.424375, Train loss: 0.6844058036804199 |Validation err: 0.411, Val
Epoch 10: Train err: 0.424, Train loss: 0.6828502931594849 |Validation err: 0.408, Valid
Epoch 11: Train err: 0.425375, Train loss: 0.6812348766326904 |Validation err: 0.4125, \
Epoch 12: Train err: 0.42, Train loss: 0.6796319708824158 |Validation err: 0.4125, Valid
Epoch 13: Train err: 0.414875, Train loss: 0.6777918744087219 |Validation err: 0.415, Va
Epoch 14: Train err: 0.412375, Train loss: 0.6761112003326416 |Validation err: 0.412, Va
Epoch 15: Train err: 0.40925, Train loss: 0.674472680568695 |Validation err: 0.415, Vali
Epoch 16: Train err: 0.406375, Train loss: 0.6727448840141297 |Validation err: 0.4105, \
Epoch 17: Train err: 0.4015, Train loss: 0.6713076601028443 |Validation err: 0.4045, Val
Epoch 18: Train err: 0.3995, Train loss: 0.6696742882728577 |Validation err: 0.4055, Val
Epoch 19: Train err: 0.40075, Train loss: 0.6679086356163025 |Validation err: 0.396, Val
Epoch 20: Train err: 0.392375, Train loss: 0.665787980556488 |Validation err: 0.405, Val
Epoch 21: Train err: 0.38975, Train loss: 0.6646300601959229 |Validation err: 0.394, Val
Epoch 22: Train err: 0.388875, Train loss: 0.662373058795929 |Validation err: 0.393, Val
Epoch 23: Train err: 0.38425, Train loss: 0.6601516346931458 |Validation err: 0.3975, Va
Epoch 24: Train err: 0.382375, Train loss: 0.6584009389877319 |Validation err: 0.386, Va
Epoch 25: Train err: 0.37875, Train loss: 0.6554971766471863 |Validation err: 0.388, Val
Epoch 26: Train err: 0.376625, Train loss: 0.6531173253059387 |Validation err: 0.3875, \
Epoch 27: Train err: 0.375, Train loss: 0.6503696331977844 |Validation err: 0.387, Valid
Epoch 28: Train err: 0.371375, Train loss: 0.6476435809135437 |Validation err: 0.3875, \
Epoch 29: Train err: 0.368375, Train loss: 0.6451257643699646 |Validation err: 0.3825, \
Epoch 30: Train err: 0.362625, Train loss: 0.6423329524993896 |Validation err: 0.3785, \
Finished Training
Total time elapsed: 122.20 seconds
```

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



Part (b) - 3pt

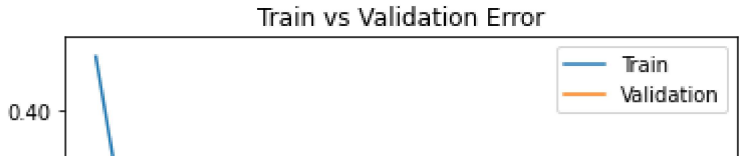
Train `large_net` with all default parameters, except set `learning_rate=0.1`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the learning rate.

```
# 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, batch_size=64, learning_rate=0.1, num_epochs=30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4295, Train loss: 0.67437779712677 |Validation err: 0.3595, Valida
Epoch 2: Train err: 0.36075, Train loss: 0.6411805458068848 |Validation err: 0.3535, Val
Epoch 3: Train err: 0.365125, Train loss: 0.6321813461780548 |Validation err: 0.3385, Va
Epoch 4: Train err: 0.352625, Train loss: 0.6233456182479858 |Validation err: 0.3575, Va
Epoch 5: Train err: 0.34075, Train loss: 0.6108013873100281 |Validation err: 0.3305, Val
Epoch 6: Train err: 0.323375, Train loss: 0.5921835997104645 |Validation err: 0.317, Val
Epoch 7: Train err: 0.3145, Train loss: 0.5817317583560944 |Validation err: 0.3365, Vali
Epoch 8: Train err: 0.29825, Train loss: 0.5660300073623658 |Validation err: 0.3285, Val
Epoch 9: Train err: 0.290875, Train loss: 0.552809501171112 |Validation err: 0.3315, Val
Epoch 10: Train err: 0.278625, Train loss: 0.539032607793808 |Validation err: 0.306, Val
```

```
Epoch 11: Train err: 0.272375, Train loss: 0.5236025826931 |Validation err: 0.33, Valida
Epoch 12: Train err: 0.267375, Train loss: 0.5220149435997009 |Validation err: 0.2925, \
Epoch 13: Train err: 0.266, Train loss: 0.5160510110855102 |Validation err: 0.3125, Vali
Epoch 14: Train err: 0.24875, Train loss: 0.4951590054035187 |Validation err: 0.3145, Va
Epoch 15: Train err: 0.264625, Train loss: 0.519231944322586 |Validation err: 0.314, Val
Epoch 16: Train err: 0.252625, Train loss: 0.5020012385845184 |Validation err: 0.3225, \
Epoch 17: Train err: 0.23875, Train loss: 0.481714787364006 |Validation err: 0.357, Vali
Epoch 18: Train err: 0.23375, Train loss: 0.47645506453514097 |Validation err: 0.3375, \
Epoch 19: Train err: 0.218125, Train loss: 0.45134368968009947 |Validation err: 0.3445,
Epoch 20: Train err: 0.217875, Train loss: 0.45516350817680357 |Validation err: 0.3245,
Epoch 21: Train err: 0.23275, Train loss: 0.47897080445289614 |Validation err: 0.3255, \
Epoch 22: Train err: 0.234875, Train loss: 0.4808810565471649 |Validation err: 0.334, Va
Epoch 23: Train err: 0.21575, Train loss: 0.4563647754192352 |Validation err: 0.316, Val
Epoch 24: Train err: 0.2355, Train loss: 0.47718250966072084 |Validation err: 0.327, Val
Epoch 25: Train err: 0.22025, Train loss: 0.4583414270877838 |Validation err: 0.3315, Va
Epoch 26: Train err: 0.209625, Train loss: 0.4519626965522766 |Validation err: 0.3435, \
Epoch 27: Train err: 0.22175, Train loss: 0.4636160457134247 |Validation err: 0.3215, Va
Epoch 28: Train err: 0.219375, Train loss: 0.46314777398109436 |Validation err: 0.348, \
Epoch 29: Train err: 0.235875, Train loss: 0.49053542733192446 |Validation err: 0.326, \
Epoch 30: Train err: 0.22, Train loss: 0.4623157248497009 |Validation err: 0.3165, Valid
Finished Training
Total time elapsed: 135.61 seconds
```

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



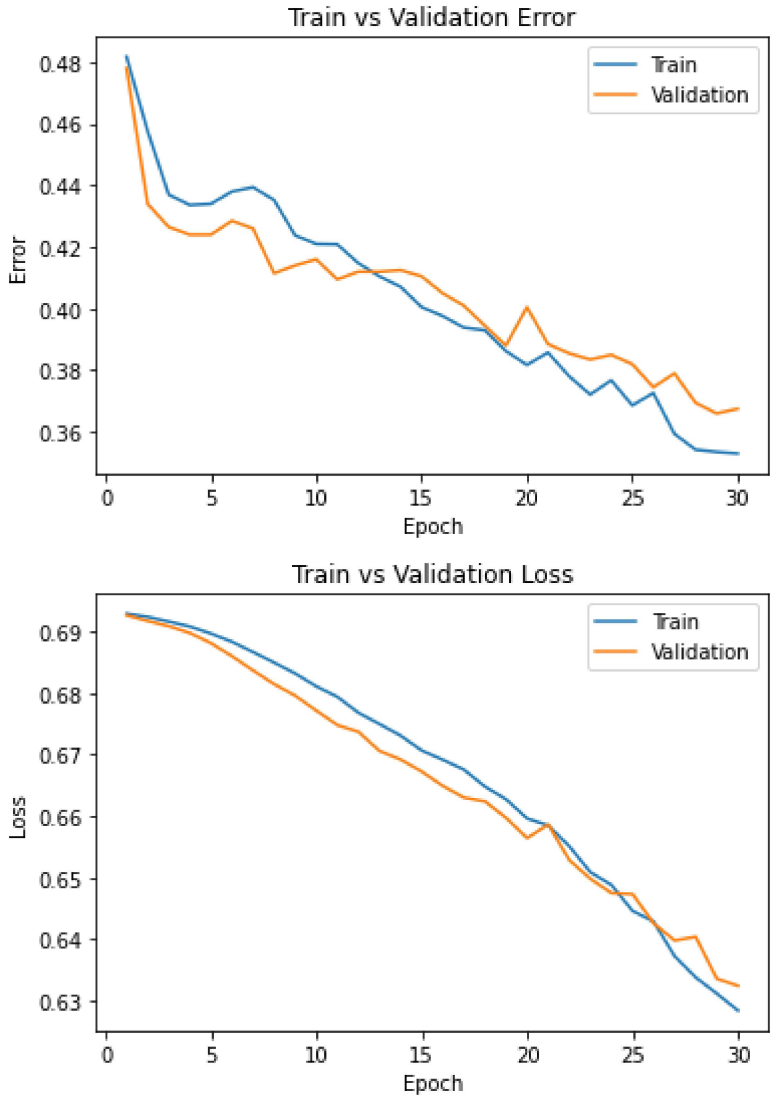
Part (c) - 3pt

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.

```
# 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, batch_size=512, learning_rate=0.01, num_epochs=30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48175, Train loss: 0.6929379552602768 |Validation err: 0.478, Vali
Epoch 2: Train err: 0.457625, Train loss: 0.6924104019999504 |Validation err: 0.434, Val
Epoch 3: Train err: 0.437, Train loss: 0.6916500590741634 |Validation err: 0.4265, Valic
Epoch 4: Train err: 0.433625, Train loss: 0.6908449940383434 |Validation err: 0.424, Val
Epoch 5: Train err: 0.434, Train loss: 0.6896935552358627 |Validation err: 0.424, Valida
Epoch 6: Train err: 0.438, Train loss: 0.688353206962347 |Validation err: 0.4285, Valida
Epoch 7: Train err: 0.439375, Train loss: 0.6866871677339077 |Validation err: 0.426, Val
Epoch 8: Train err: 0.43525, Train loss: 0.6849770769476891 |Validation err: 0.4115, Val
Epoch 9: Train err: 0.42375, Train loss: 0.6832009293138981 |Validation err: 0.414, Vali
Epoch 10: Train err: 0.421, Train loss: 0.6811089366674423 |Validation err: 0.416, Valic
Epoch 11: Train err: 0.420875, Train loss: 0.6794026419520378 |Validation err: 0.4095, V
Epoch 12: Train err: 0.41475, Train loss: 0.6768048219382763 |Validation err: 0.412, Val
Epoch 13: Train err: 0.4105, Train loss: 0.6749702803790569 |Validation err: 0.412, Vali
Epoch 14: Train err: 0.407125, Train loss: 0.6730880849063396 |Validation err: 0.4125, V
Epoch 15: Train err: 0.4005, Train loss: 0.6706806942820549 |Validation err: 0.4105, Val
Epoch 16: Train err: 0.397625, Train loss: 0.6691771410405636 |Validation err: 0.405, Va
Epoch 17: Train err: 0.393875, Train loss: 0.6675694733858109 |Validation err: 0.401, Va
Epoch 18: Train err: 0.393, Train loss: 0.6648042872548103 |Validation err: 0.3945, Vali
Epoch 19: Train err: 0.38625, Train loss: 0.662746611982584 |Validation err: 0.388, Vali
Epoch 20: Train err: 0.38175, Train loss: 0.6596181839704514 |Validation err: 0.4005, Va
Epoch 21: Train err: 0.38575, Train loss: 0.6584899798035622 |Validation err: 0.3885, Va
Epoch 22: Train err: 0.378125, Train loss: 0.655123382806778 |Validation err: 0.3855, Va
Epoch 23: Train err: 0.372125, Train loss: 0.6508794128894806 |Validation err: 0.3835, V
Epoch 24: Train err: 0.37675, Train loss: 0.6488028429448605 |Validation err: 0.385, Val
Epoch 25: Train err: 0.368625, Train loss: 0.6445869170129299 |Validation err: 0.382, Va
Epoch 26: Train err: 0.372625, Train loss: 0.6428566053509712 |Validation err: 0.3745, V
Epoch 27: Train err: 0.359375, Train loss: 0.6372117549180984 |Validation err: 0.379, Va
Epoch 28: Train err: 0.35425, Train loss: 0.6337667480111122 |Validation err: 0.3695, Va
Epoch 29: Train err: 0.3535, Train loss: 0.6311353109776974 |Validation err: 0.366, Vali
Epoch 30: Train err: 0.353, Train loss: 0.6283832415938377 |Validation err: 0.3675, Vali
Finished Training
Total time elapsed: 108.30 seconds
```

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



Part (d) - 3pt

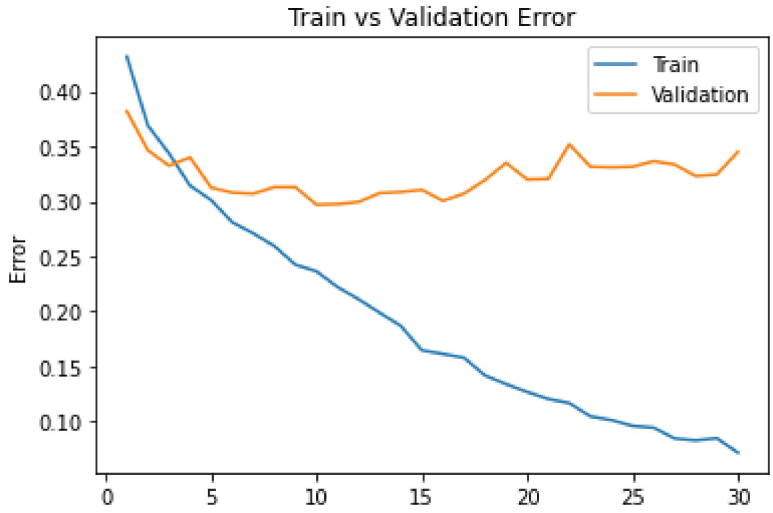
Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=16`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *decreasing* the batch size.

```
# 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, batch_size=16, learning_rate=0.01, num_epochs=30)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
Epoch 1: Train err: 0.43175, Train loss: 0.6774994022846222 |Validation err: 0.382, Vali
Epoch 2: Train err: 0.369, Train loss: 0.639639899969101 |Validation err: 0.3465, Valida
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err: 0.3325, Val
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err: 0.34, Vali
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err: 0.3125, Va
Epoch 6: Train err: 0.281, Train loss: 0.5452213581204415 |Validation err: 0.308, Valida
Epoch 7: Train err: 0.270875, Train loss: 0.5272981298565864 |Validation err: 0.307, Val
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err: 0.313, Val
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err: 0.313, Val
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation err: 0.297, Va
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation err: 0.2975, \
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err: 0.2995, Vali
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err: 0.3075, Va
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err: 0.3085, Va
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err: 0.3105, Val
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err: 0.3005, Va
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err: 0.307, Val
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation err: 0.3195,
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation err: 0.335, Va
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation err: 0.32, Val
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation err: 0.3205, \
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err: 0.352, Val
Epoch 23: Train err: 0.104375, Train loss: 0.2467898527495563 |Validation err: 0.3315, \
Epoch 24: Train err: 0.101, Train loss: 0.23970085787773132 |Validation err: 0.331, Vali
Epoch 25: Train err: 0.09575, Train loss: 0.23643119425699116 |Validation err: 0.3315, \
Epoch 26: Train err: 0.094125, Train loss: 0.2325953512713313 |Validation err: 0.3365, \
Epoch 27: Train err: 0.08425, Train loss: 0.21040759468451142 |Validation err: 0.3335, \
Epoch 28: Train err: 0.0825, Train loss: 0.20643112615589052 |Validation err: 0.323, Val
Epoch 29: Train err: 0.0845, Train loss: 0.21273409337876364 |Validation err: 0.3245, Va
Epoch 30: Train err: 0.071375, Train loss: 0.18387044295761734 |Validation err: 0.345, \
Finished Training
Total time elapsed: 176.38 seconds
```

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



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.

```
network = small_net # Small net achieved a validation loss of 0.60 which is lower than the be
batch_size = 512 # We saw earlier than increasing batch size resolves the issue of overfittin
learning_rate = 0.001 # A smaller learning rate also helped us prevent overfitting and reduce
```

Part (b) - 1pt

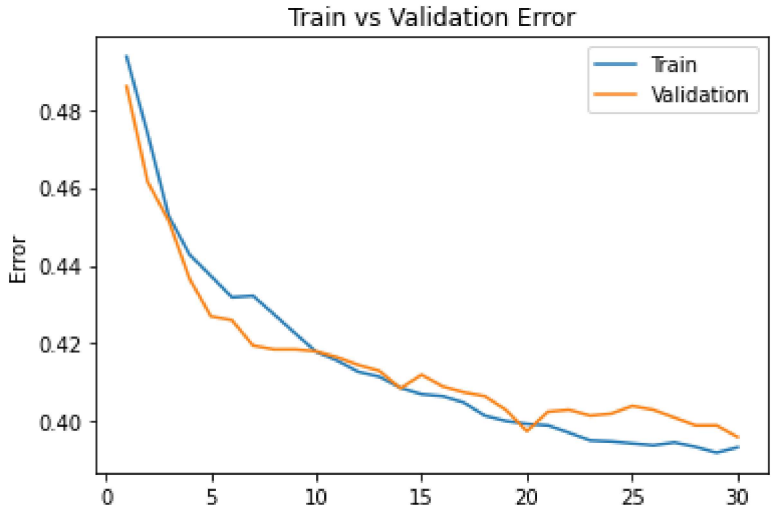
Train the model with the hyperparameters you chose in part(a), and include the training curve.

```
# 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.
small_net = SmallNet()
train_net(small_net, batch_size=512, learning_rate=0.001, num_epochs=30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.493625, Train loss: 0.6933990158140659 |Validation err: 0.486, Val
Epoch 2: Train err: 0.473875, Train loss: 0.6907263100147247 |Validation err: 0.4615, Va
Epoch 3: Train err: 0.45275, Train loss: 0.6885315254330635 |Validation err: 0.4515, Val
Epoch 4: Train err: 0.44275, Train loss: 0.6870227605104446 |Validation err: 0.4365, Val
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7fa66bcb7a7(
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py", line 132
    self._shutdown_workers()
  File "/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py", line 132
```

```
    if w.is_alive():
  File "/usr/lib/python3.7/multiprocessing/process.py", line 151, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
Epoch 5: Train err: 0.437375, Train loss: 0.6858981251716614 |Validation err: 0.427, Val
Epoch 6: Train err: 0.431875, Train loss: 0.6847453452646732 |Validation err: 0.426, Val
Epoch 7: Train err: 0.43225, Train loss: 0.6837133355438709 |Validation err: 0.4195, Val
Epoch 8: Train err: 0.4275, Train loss: 0.6826400235295296 |Validation err: 0.4185, Vali
Epoch 9: Train err: 0.422625, Train loss: 0.6817480809986591 |Validation err: 0.4185, Va
Epoch 10: Train err: 0.417875, Train loss: 0.68068478256464 |Validation err: 0.418, Vali
Epoch 11: Train err: 0.415625, Train loss: 0.6797182895243168 |Validation err: 0.4165, \
Epoch 12: Train err: 0.41275, Train loss: 0.678427692502737 |Validation err: 0.4145, Val
Epoch 13: Train err: 0.4115, Train loss: 0.6776212379336357 |Validation err: 0.413, Vali
Epoch 14: Train err: 0.408625, Train loss: 0.6766696572303772 |Validation err: 0.4085, \
Epoch 15: Train err: 0.407, Train loss: 0.6753812283277512 |Validation err: 0.412, Valid
Epoch 16: Train err: 0.4065, Train loss: 0.6748527996242046 |Validation err: 0.409, Vali
Epoch 17: Train err: 0.404875, Train loss: 0.6738493144512177 |Validation err: 0.4075, \
Epoch 18: Train err: 0.4015, Train loss: 0.6730030737817287 |Validation err: 0.4065, Val
Epoch 19: Train err: 0.400125, Train loss: 0.6724236235022545 |Validation err: 0.403, Va
Epoch 20: Train err: 0.399375, Train loss: 0.6712068282067776 |Validation err: 0.3975, \
Epoch 21: Train err: 0.399, Train loss: 0.6705421060323715 |Validation err: 0.4025, Vali
Epoch 22: Train err: 0.397125, Train loss: 0.6702669113874435 |Validation err: 0.403, Va
Epoch 23: Train err: 0.395125, Train loss: 0.6690297685563564 |Validation err: 0.4015, \
Epoch 24: Train err: 0.394875, Train loss: 0.6685625687241554 |Validation err: 0.402, Va
Epoch 25: Train err: 0.394375, Train loss: 0.6675844378769398 |Validation err: 0.404, Va
Epoch 26: Train err: 0.393875, Train loss: 0.6672164835035801 |Validation err: 0.403, Va
Epoch 27: Train err: 0.394625, Train loss: 0.6664052307605743 |Validation err: 0.401, Va
Epoch 28: Train err: 0.3935, Train loss: 0.6663550063967705 |Validation err: 0.399, Vali
Epoch 29: Train err: 0.392, Train loss: 0.6649808846414089 |Validation err: 0.399, Valid
Epoch 30: Train err: 0.393375, Train loss: 0.6648307293653488 |Validation err: 0.396, Va
Finished Training
Total time elapsed: 98.13 seconds
```

```
model_path = get_model_name("small", batch_size=512, learning_rate=0.001, epoch=29)
plot_training_curve(model_path)
```



Part (c) - 2pt

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

```
# We notice that the model does learn over time but the final val loss of 0.68 is much larger
# This actually makes sense, the parameters we tuned helped to reduce the overfitting in larg
# To solve this we increase the learning rate to 0.005 and decrease the batch size to 128. Th
# This could lead to overfitting as it has a greater opportunity to learn the pecularities of
```

Part (d) - 1pt

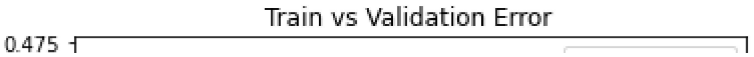
Train the model with the hyperparameters you chose in part(c), and include the training curve.

```
# 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.
small_net = SmallNet()
train_net(small_net, batch_size=128, learning_rate=0.005, num_epochs=30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.468625, Train loss: 0.6908236969084967 |Validation err: 0.436, Val
Epoch 2: Train err: 0.433, Train loss: 0.6834235077812558 |Validation err: 0.4095, Valic
Epoch 3: Train err: 0.41625, Train loss: 0.6766141917970445 |Validation err: 0.3865, Val
Epoch 4: Train err: 0.395, Train loss: 0.6682078951881045 |Validation err: 0.3955, Valic
Epoch 5: Train err: 0.384375, Train loss: 0.659145544445704 |Validation err: 0.3865, Val
Epoch 6: Train err: 0.369, Train loss: 0.6487330179365854 |Validation err: 0.3635, Valic
Epoch 7: Train err: 0.35875, Train loss: 0.6391496005512419 |Validation err: 0.354, Vali
Epoch 8: Train err: 0.3495, Train loss: 0.630687625635238 |Validation err: 0.35, Validat
Epoch 9: Train err: 0.341, Train loss: 0.6234588537897382 |Validation err: 0.3525, Valic
Epoch 10: Train err: 0.335, Train loss: 0.6173147142879547 |Validation err: 0.348, Valic
Epoch 11: Train err: 0.33225, Train loss: 0.6117481873148963 |Validation err: 0.342, Val
Epoch 12: Train err: 0.33, Train loss: 0.6065104121253604 |Validation err: 0.3415, Valic
Epoch 13: Train err: 0.323875, Train loss: 0.603826661904653 |Validation err: 0.3365, Va
```

```
Epoch 14: Train err: 0.31725, Train loss: 0.5995143103221107 |Validation err: 0.3395, Va
Epoch 15: Train err: 0.31575, Train loss: 0.5970706154429724 |Validation err: 0.3345, Va
Epoch 16: Train err: 0.314, Train loss: 0.5923306790609209 |Validation err: 0.3285, Vali
Epoch 17: Train err: 0.311625, Train loss: 0.592100812802239 |Validation err: 0.335, Val
Epoch 18: Train err: 0.308625, Train loss: 0.5882551859295557 |Validation err: 0.323, Va
Epoch 19: Train err: 0.3075, Train loss: 0.5867307205048818 |Validation err: 0.33, Valid
Epoch 20: Train err: 0.300125, Train loss: 0.5818465238525754 |Validation err: 0.3225, V
Epoch 21: Train err: 0.299, Train loss: 0.5800543271359944 |Validation err: 0.319, Valid
Epoch 22: Train err: 0.297, Train loss: 0.5789775091504293 |Validation err: 0.3205, Vali
Epoch 23: Train err: 0.294875, Train loss: 0.575637506114112 |Validation err: 0.3155, Va
Epoch 24: Train err: 0.29925, Train loss: 0.5777310189746675 |Validation err: 0.319, Val
Epoch 25: Train err: 0.290625, Train loss: 0.5701041108085996 |Validation err: 0.3155, V
Epoch 26: Train err: 0.293375, Train loss: 0.5706725721321408 |Validation err: 0.3215, V
Epoch 27: Train err: 0.288875, Train loss: 0.5677874637028527 |Validation err: 0.311, Va
Epoch 28: Train err: 0.28775, Train loss: 0.5681796575349475 |Validation err: 0.3075, Va
Epoch 29: Train err: 0.2915, Train loss: 0.5682089754513332 |Validation err: 0.314, Vali
Epoch 30: Train err: 0.29025, Train loss: 0.5657869260462504 |Validation err: 0.3085, Va
Finished Training
Total time elapsed: 113.64 seconds
```

```
model_path = get_model_name("small", batch_size=128, learning_rate=0.005, epoch=29)
plot_training_curve(model_path)
```



Part 4. Evaluating the Best Model [10 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 = SmallNet()
model_path = get_model_name(net.name, batch_size=128, learning_rate=0.005, epoch=10) # After
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).

```
# Based on the validation loss and validation error, the small net with batch size of 128 and
# best result (lowest loss and error)
```

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=64)

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

print(evaluate(net, test_loader, nn.BCEWithLogitsLoss()))
```

```
Files already downloaded and verified
Files already downloaded and verified
(0.3315, 0.6112697850912809)
```

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.

ANS: It is slightly higher than the validation error achieved at the end of training (0.593). We do expect it because the performance test results were shown to the model to improve upon however it had never seen the test set and hence it might not be able to produce the same accuracy (or loss)

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?

Using it during the training phase defeats its very purpose. It's like showing the final exam paper to a student so he performs well in the finals! Test data is used to predict how well the model is

behave on 'unseen' data. The emphasis on 'unseen' suggests that it is recommended that we use it as little as possible when fine-tuning