hw4

May 31, 2022

1 HW4: Autoencoders & RNNs

The homework is broken up as follows:

1. Question 1: Masked Autoencoder, Revisited (three points total)
2. Question 2: A Simple Sequence Task (three points total)

3. Question 3: Sequence-to-Sequence Translation (three points total)
4. Question 4: Short Response (two points total)

1.1 How to Submit This Homework

This homework has been provided to you as an `.ipynb` file. I expect you to complete the notebook and submit the following on Canvas:

1. The completed notebook itself with your answers filled in (including any source code used to answer the question)
2. A `.pdf` file of the notebook that shows all your answers. This will make grading easier and I will appreciate that.

If you submit only one of these, you will lose points. No late homework will be graded. No submissions via email or other media will be graded.

Finally, each question specifies exactly what should be provided in your answer – for example, if no code is requested, do not provide it. Your submitted notebook (and PDF) should not contain any additional content or cells beyond what is requested.

1.2 Setup

If you do not have access to a CUDA-compatible NVIDIA GPU, it is recommended that you run this notebook in Google Colab. There, you will have the option to enable GPU acceleration with `Runtime » Change runtime type » Hardware accelerator » GPU » Save`. Note that you may have to re-connect or restart the notebook after changing runtime types.

[10]:
```
# helper code from the course repository
%%capture
!git clone https://github.com/interactiveaudiolab/course-deep-learning.git
# install common pacakges used for deep learning
!cd course-deep-learning/ && pip install -r requirements.txt
```

[2]:
```
%matplotlib inline
%cd course-deep-learning/

import datetime
import math
import torch
import matplotlib.pyplot as plt
import numpy as np
import IPython.display as ipd
from matplotlib.animation import FuncAnimation
from pathlib import Path
from torch.utils.tensorboard import SummaryWriter
from torchsummary import summary
from tqdm import tqdm
import torchvision

from utils.gan import *
```

```
/content/course-deep-learning
```

1.3 Question 1: Masked Autoencoder, Revisited (three points total)

1.3.1 1.1 A Bigger Mask (one point)

Train the MLP-based inpainting autoencoder from notebook 7 but this time, re-write `apply_mask()` (and your training code) to **deterministically zero-out the *left half* of every input image. Train for 30 epochs, and place a plot of your reconstruction loss vs. epochs below.**

Hint: make sure that you apply the mask to a copy of each input rather than destructively editing the input itself, as the original input still has to serve as a target!

[11]:
```
class MLPEncoder(torch.nn.Module):

    def __init__(self,
                 number_of_hidden_layers: int,
                 input_size: int,
                 hidden_size: int,
                 latent_size: int,
                 activation: torch.nn.Module):
        """Construct a simple MLP encoder"""

        super().__init__()

        assert number_of_hidden_layers >= 0, "Encoder number_of_hidden_layers␣
↪must be at least 0"

        dims_in = [input_size] + [hidden_size] * number_of_hidden_layers
```

```
        dims_out = [hidden_size] * number_of_hidden_layers + [latent_size]  # final output should be latent size
        layers = []
        for i in range(number_of_hidden_layers + 1):
            layers.append(torch.nn.Linear(dims_in[i], dims_out[i]))

            if i < number_of_hidden_layers:
                layers.append(activation)

        self.net = torch.nn.Sequential(*layers)

    def forward(self, x: torch.Tensor):
        return self.net(x)


class MLPDecoder(torch.nn.Module):

    def __init__(self,
                 number_of_hidden_layers: int,
                 latent_size: int,
                 hidden_size: int,
                 input_size: int,
                 activation: torch.nn.Module):
        """Construct a simple MLP decoder"""

        super().__init__()

        assert number_of_hidden_layers >= 0, "Decoder number_of_hidden_layers must be at least 0"

        dims_in = [latent_size] + [hidden_size] * number_of_hidden_layers
        dims_out = [hidden_size] * number_of_hidden_layers + [input_size]  # final output is an image

        layers = []
        for i in range(number_of_hidden_layers + 1):
            layers.append(torch.nn.Linear(dims_in[i], dims_out[i]))

            if i < number_of_hidden_layers:
                layers.append(activation)

        # apply Tanh after final layer to bound pixels to range [-1, 1]
        layers.append(torch.nn.Sigmoid())

        self.net = torch.nn.Sequential(*layers)

    def forward(self, x: torch.Tensor):
```

```
        return self.net(x)


class MLPAutoencoder(torch.nn.Module):

    def __init__(self,
                 number_of_hidden_layers: int,
                 input_size: int,
                 hidden_size: int,
                 latent_size: int,
                 activation_encoder: torch.nn.Module = torch.nn.ReLU(),
                 activation_decoder: torch.nn.Module = torch.nn.LeakyReLU(0.2)
                ):
        """Construct a simple MLP autoencoder

        number_of_hidden_layers: An int. Must be >=0. Defines the number of
                hidden layers for both the encoder E and decoder D
        latent_size:  An int. Defines the size of the latent representation␣
↪produced by
                        the encoder.
        hidden_size: An int. The size of each hidden layer for the encoder E and
                        the decoder D.
        input_size: An int. Determines the size of the input and output images
        activation_encoder: A torch.nn.Module defining the activation function␣
↪in every
                                hidden layer of the encoder.
        activation_decoder: A torch.nn.Module defining the activation function␣
↪in every
                                hidden layer of the decoder.
        """
        super().__init__()

        self.encoder = MLPEncoder(
            number_of_hidden_layers=number_of_hidden_layers,
            input_size=input_size,
            hidden_size=hidden_size,
            latent_size=latent_size,
            activation=activation_encoder
        )

        self.decoder = MLPDecoder(
            number_of_hidden_layers=number_of_hidden_layers,
            input_size=input_size,
            hidden_size=hidden_size,
            latent_size=latent_size,
            activation=activation_decoder
        )
```

```
    def encode(self, x: torch.Tensor):
        return self.encoder(x)

    def decode(self, x: torch.Tensor):
        return self.decoder(x)

    def forward(self, x: torch.Tensor):
        return(self.decode(self.encode(x)))
```

[12]:
```
%load_ext tensorboard
%tensorboard --logdir logs --port 6006
```

```
<IPython.core.display.Javascript object>
```

[13]:
```
%%capture
# autoencoder training hyperparameters
image_size = 28
batch_size = 64
latent_size = 64
hidden_size = 256
number_of_hidden_layers = 2
lr = 0.0002
epochs = 30

# fix random seed
torch.manual_seed(0)

# select device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# load MNIST dataset
mnist = load_mnist(batch_size=batch_size)

# initialize the model
model = MLPAutoencoder(
    number_of_hidden_layers=number_of_hidden_layers,
    latent_size=latent_size,
    hidden_size=hidden_size,
    input_size=image_size*image_size,
).to(device)

# use an optimizer to handle parameter updates
opt = torch.optim.Adam(model.parameters(), lr=lr)

# loss function: mean-squared error between original and reconstructed images
loss = torch.nn.MSELoss()
```

```
# save all log data to a local directory
run_dir = "logs"

# to clear out TensorBoard and start totally fresh, we'll need to
# remove old logs by deleting them from the directory
!rm -rf ./logs/

# timestamp the logs for each run so we can sort through them
run_time = datetime.datetime.now().strftime("%I:%M%p on %B %d, %Y")

# initialize a SummaryWriter object to handle all logging actions
logger = SummaryWriter(log_dir=Path(run_dir) / run_time, flush_secs=20)

# log reconstructions of a fixed set of images during training
example_batch, _ = next(iter(mnist))
example_batch = example_batch.to(device)
logger.add_image("training_images", make_grid(example_batch, math.floor(math.
 ↪sqrt(batch_size))), 0)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
./data/MNIST/raw/train-images-idx3-ubyte.gz
```

```
  0%|          | 0/9912422 [00:00<?, ?it/s]
```

```
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
./data/MNIST/raw/train-labels-idx1-ubyte.gz
```

```
  0%|          | 0/28881 [00:00<?, ?it/s]
```

```
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
./data/MNIST/raw/t10k-images-idx3-ubyte.gz
```

```
  0%|          | 0/1648877 [00:00<?, ?it/s]
```

```
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
```

```
  0%|          | 0/4542 [00:00<?, ?it/s]
```

```
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

[14]:
```python
def apply_mask(x: torch.Tensor, image_size: int):
    """Apply mask that zero-out the left-half of each input in batch"""

    n_batch = x.shape[0]

    x_masked = x.clone()

    # apply masks to each image in batch
    for i in range(n_batch):
        for j in range(int(image_size / 2)):
            x_masked[
            i, :, :, j
            ] = 0
    return x_masked
```

[15]:
```python
for epoch in range(epochs):

    # weight batch losses/scores proportional to batch size
    iter_count = 0
    loss_epoch = 0

    for batch_idx, batch_data in enumerate(mnist):

        # we only care about inputs, not labels
        x_real, _ = batch_data

        # obtain batch of inputs and move to device
        x_real = x_real.to(device)
        n_batch = x_real.shape[0]

        # apply random masks
        x_masked = apply_mask(x_real, image_size)
        x_masked = x_masked.reshape(n_batch, -1)

        # flatten input images
        x_real = x_real.reshape(n_batch, -1)

        model.zero_grad()

        # train on a batch of inputs
        x_reconstructed = model(x_masked)
        loss_batch = loss(x_real, x_reconstructed)
        loss_batch.backward()
        opt.step()
```

```
        # log loss
        loss_epoch += loss_batch.detach().item() * n_batch
        iter_count += n_batch

    # plot loss
    loss_epoch /= iter_count
    logger.add_scalar("mse_loss", loss_epoch, epoch)

    # plot example generated images
    with torch.no_grad():
        masked_example = apply_mask(example_batch, image_size)
        logger.add_image("masked_images", make_grid(masked_example, math.
↪floor(math.sqrt(batch_size)), title="Masked Images"), epoch)
        reconstructed_batch = model(masked_example.reshape(batch_size, -1)).
↪reshape(batch_size, 1, image_size, image_size)
        logger.add_image("reconstructed_images", make_grid(reconstructed_batch,␣
↪math.floor(math.sqrt(batch_size)), title="Reconstructed Images"), epoch)

    if not epoch % 10:
        print(f"Epoch: {epoch + 1}\tMSE Loss: {loss_epoch :0.4f}")
```

```
Epoch: 1        MSE Loss: 0.0634
Epoch: 11       MSE Loss: 0.0186
Epoch: 21       MSE Loss: 0.0157
```

(YOUR LOSS PLOT HERE)

1.3.2 1.2 Reconstruction (one point)

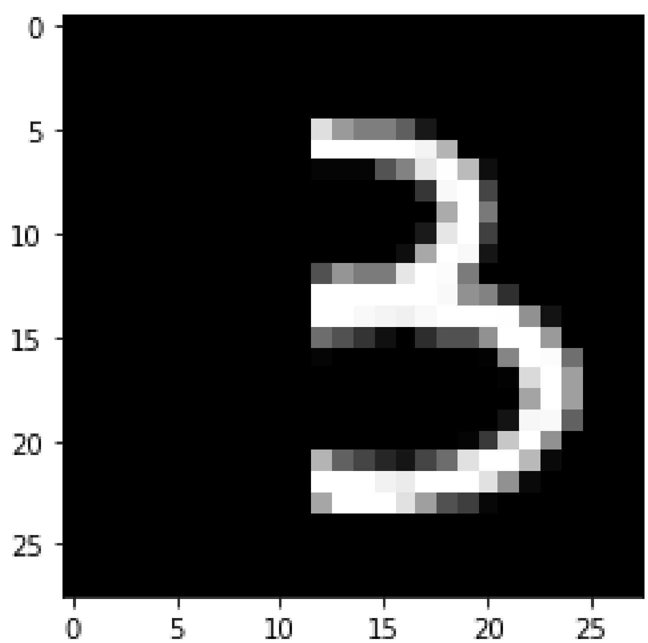
The code below provides a hard-coded example image of a "3." **Pass this image through your new inpainting autoencoder and include the reconstructed image below. Does your autoencoder faithfully reconstruct the input (i.e. produce a similar-looking image of the same class)? If not, briefly explain why you think the reconstruction looks the way it does.**

[12]:
```
x = torch.as_tensor(
    [[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
```

```
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.8667, 0.6000, 0.4941, 0.4941, 0.3765, 0.0941, 0.
↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.9961, 0.9961, 0.9961, 0.9961, 0.9961, 0.9490, 0.
↪7059, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0157, 0.0157, 0.0157, 0.3255, 0.5059, 0.8941, 0.
↪9922, 0.7294, 0.0549, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.2118, 0.
↪9765, 0.9961, 0.2706, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪6627, 0.9961, 0.4706, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1020, 0.
↪8980, 0.9961, 0.2471, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0549, 0.6588, 0.
↪9961, 0.9725, 0.0863, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
      [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.3216, 0.5843, 0.4863, 0.4863, 0.8980, 0.9961, 0.
↪9804, 0.4863, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
```

```
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.9961, 0.9961, 0.9961, 0.9961, 0.9961, 0.9961, 0.
↪9765, 0.5647, 0.5059, 0.1882, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.9961, 0.9961, 0.9725, 0.9490, 0.9373, 0.9686, 0.
↪9961, 0.9961, 0.9961, 0.9922, 0.5647, 0.0706, 0.0000, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.4314, 0.3176, 0.2000, 0.0627, 0.0000, 0.1765, 0.
↪3176, 0.3176, 0.5608, 0.9961, 0.9961, 0.6000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0157, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0118, 0.5176, 0.9961, 0.9843, 0.4353, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0000, 0.0118, 0.8471, 0.9961, 0.6157, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0000, 0.0000, 0.6392, 0.9961, 0.6157, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0000, 0.0000, 0.0745, 0.9608, 0.9725, 0.3843, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.0039, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000, 0.0157, 0.2196, 0.7765, 0.9961, 0.5608, 0.0000, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.6980, 0.3804, 0.2667, 0.1529, 0.0824, 0.2667, 0.
↪4235, 0.8745, 0.9961, 0.9961, 0.7255, 0.0392, 0.0000, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.9961, 0.9961, 0.9961, 0.9451, 0.9176, 0.9961, 0.
↪9961, 0.9961, 0.8745, 0.5608, 0.0392, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
     [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
↪0.0000, 0.0000, 0.0000, 0.6471, 0.9961, 0.9961, 0.9961, 0.8784, 0.6157, 0.
↪3216, 0.2431, 0.0235, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
↪0000],
```

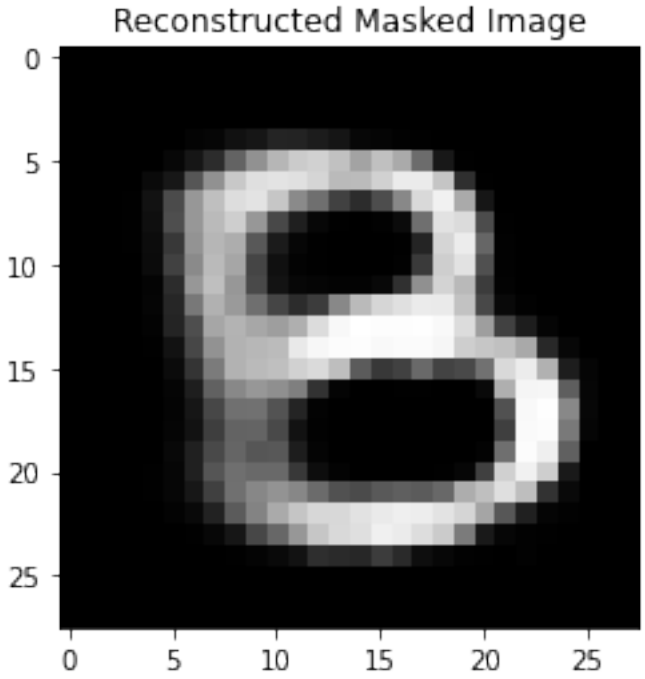
```
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
 ↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
 ↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
 ↪0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
 ↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
 ↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
 ↪0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
 ↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
 ↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
 ↪0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,␣
 ↪0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
 ↪0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.
 ↪0000]]])

plt.imshow(x.squeeze(), cmap='gray')
plt.show()
```



(YOUR RECONSTRUCTED IMAGE HERE)

```
[13]: reconstructed_image = model(x.flatten().unsqueeze(0).to(device)).reshape(1,␣
 ↪image_size, image_size).detach().cpu()
plt.imshow(reconstructed_image.squeeze(), cmap='gray')
```

```
plt.title("Reconstructed Masked Image")
plt.show()
```



(YOUR EXPLANATION HERE)

Instead of reconstructing a 3, my autoencoder faithfully reconstructed an 8. Becauses 3 looks like the right-half of an 8, and our autoencoder is trained to fill in the masked left-half of an input image, the input image 3 will be reconstructed with its left-half filled, and the resulting image looks like an 8.

1.3.3 1.3 Augmentation (one point)

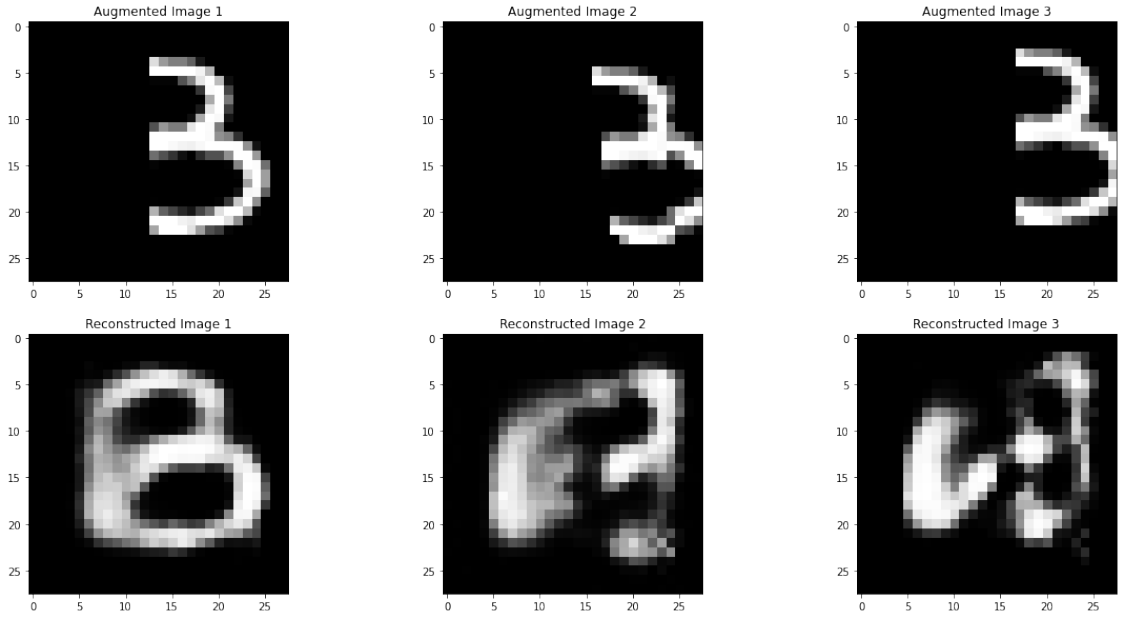
In notebook 4, we learned how to apply random augmentations to image data. **Create a `RandomAffine` transform that applies: * Rotations between -10 and 10 degrees * Horizontal and vertical translations of at most 0.2 * Shear between -5 and 5 degrees**

Using this randomized transform, create three augmented versions of the hard-coded image above. For each augmented version obtain a reconstruction by passing it to your inpainting autoencoder. Include the three augmented images and three corresponding reconstructions below.

Finally, answer the following question: does your autoencoder reconstruct the augmented images correctly and consistently? If not, explain why you think this may be the case.

[29]:
```
fig, axis = plt.subplots(2,3, figsize=(20, 10))
for i in range(3):
```

```
affine_aug = torchvision.transforms.RandomAffine(degrees=(-10, 10),␣
↪translate=(0.2, 0.2), shear=(-5, 5))
augmented = affine_aug(x)
axis[0][i].title.set_text("Augmented Image {}".format(i+1))
axis[0][i].imshow(augmented.squeeze(), cmap='gray')
axis[1][i].title.set_text("Reconstructed Image {}".format(i+1))
reconstructed_image = model(augmented.flatten().unsqueeze(0).to(device)).
↪reshape(1, image_size, image_size).detach().cpu()
axis[1][i].imshow(reconstructed_image.squeeze(), cmap='gray')
```



(YOUR AUGMENTED IMAGES HERE)

(YOUR RECONSTRUCTED IMAGES HERE)

(YOUR EXPLANATION HERE)

It turns out that the autoencoder does not produce consistent and correct reconstructed image. I think it might be caused by the transformation on the input image that cause the displacement of the right-half of the original image. For example, for our augmented and reconstructed image 1, the result looks similar to an 8 because there is not so much transformation on the original 3, and that the autoencoder could locate the right-half (3) correctly. However, for our second and third sets, the transformations are significant, and the autoencoder does not know how to properly fill in the left-half of these images with such a big gap in the middle. Therefore, the later two reconstructions are not recognizable and not consistent.

1.4 Question 2: A Simple Sequence Task (three points total)

In this question, we'll explore the differences between standard recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) using a simple addition task akin to the one used in the original LSTM paper.

1.4.1 2.1 Theoretical Comparison (0.5 points)

Explain the argument for why we would expect an LSTM to outperform a standard RNN on problems that involve processing long sequences.

(YOUR ANSWER HERE)

The LSTM provides "constant error back proporgation" in its memory cells. LSTMs deal with vanishing and exploding gradient problem by introducing new gates, such as input gates, new candidate gates, and forget gates, which allow for a better control over the gradient flow and enable better preservation of "long-range dependencies".

1.4.2 2.2 Empirical Comparison (1.5 points)

Use the code provided below to compare the performance of a standard RNN to an LSTM on the task defined by the `Add2DigitsDataset`. Do this for three different lengths of sequence (5, 10, and 30). For all experiments, use a hidden size of 128 and 2 layers. Keep your network architecture the same, except for the substitution of the LSTM for an RNN. Provide the following graphs for each sequence length:

1. **Loss as a function of the number of training steps**

2. **Training accuracy & validation accuracy as a function of the number of training steps.**

The following utility code is provided to help with the question. Note that the addition task is formulated as a *classification* rather than *regression* problem: because the operands are digits, their sum must be a positive integer in [0, 18], allowing us to output a vector of "class scores" over these potential values.

[5]:
```
def MakeItHot(data, num_tokens):
  """ Make the one hot encoding of the input."""
  i_size, j_size = data.shape
  one_hot_data = torch.zeros(i_size, j_size, num_tokens)
  for i in range(0,i_size):
    for j in range(0,j_size):
      one_hot_data[i,j,data[i,j]] = 1
  return one_hot_data

def KeepItCool(data):
  """Just use the data as-is...but with a new dimension added"""
  return torch.unsqueeze(data,2)
```

```
class Add2DigitsDataset(torch.utils.data.Dataset):
    """Add2DigitsDataset"""

    def __init__(self, num_examples=3, seq_length=10):
        """Create a set of examples, where each example is a sequence of single
        positive digits (0 through 9) interspersed with 1 instance of a negative
        one. The game is to sum the final digit in the sequence with the
        digit after the -1. The correct label y for the sequence x is the sum␣
↪of
        these two digits. Here are 3 examples.

        x = [1, -1, 3, 7, 8, 9] y = 12
        x = [9, 1, 2, 4, -1, 7, 2, 3, 1, 0] y = 7
        x = [9, -1, 9, 2 , 1, 3, 0, 5, 6, 4, 7, 8, 5, 1] y = 10


        PARAMETERS
        ----------
        num_examples    A non-negative integer determining how much data we'll␣
↪generate
        seq_length      A non-negative integer saying how long the sequences␣
↪all are.

        EXAMPLE USE
        -----------
        dataset = Add2DigitsDataset(num_examples = 100, seq_length = 9)
        data, one_hot_data, labels  = dataset[:]
        print(f'Train instances: one_hot_data (shape {one_hot_data.shape}),␣
↪labels (shape {labels.shape})')
        loader =  torch.utils.data.DataLoader(dataset, batch_size=10,␣
↪shuffle=True)
        """

        assert seq_length >=3, "Seq_length must be a minimum of 3"
        self.seq_length = seq_length
        self.num_tokens = 11

        # make random sequences of integers of the right length
        data = torch.randint(10, (num_examples, seq_length))
        # the labels will go here...
        label = torch.ones(num_examples,1)

        # Now insert our special tokens
        for x in range(0, num_examples):
          # figure out where to place our special token
          a_idx = torch.randint(0, seq_length-2, (1,1)).squeeze()
```

```
          # insert the special_tokens
          data[x,[a_idx]] = -1
          # create the label by summing the digit after the special -1 token
          # with the final digit in the sequence
          label[x] = data[x,a_idx+1]+data[x,-1]

        # OK. Store data for later.
        self.data = KeepItCool(data)
        self.one_hot_data = MakeItHot(data, num_tokens=self.num_tokens)
        self.label = label.squeeze().long()
        self.lengths = self.seq_length * torch.ones_like(self.label)
        self.lengths = self.lengths.squeeze().long().tolist()

    def __len__(self):
      return len(self.label)

    def __getitem__(self, idx):
      return self.data[idx], self.one_hot_data[idx], self.label[idx] #, self.
 ↪lengths[idx]



# Make sure our dataset looks correct by looking at a few examples...
dataset = Add2DigitsDataset(num_examples=100,seq_length=5)
data, one_hot_data, labels = dataset[:3]
# inputs have shape (B, MAX_L, V) where MAX_L is largest length in batch
print(f'data.shape: {data.shape}, one_hot_data.shape: {one_hot_data.shape},␣
 ↪labels.shape: {labels.shape}')
print(f'data: {data}, one_hot_data: {one_hot_data}, labels: {labels}')
```

```
data.shape: torch.Size([3, 5, 1]), one_hot_data.shape: torch.Size([3, 5, 11]),
labels.shape: torch.Size([3])
data: tensor([[[ 8],
         [-1],
         [ 2],
         [ 6],
         [ 1]],

        [[ 3],
         [ 0],
         [-1],
         [ 4],
         [ 6]],

        [[ 7],
         [-1],
         [ 2],
```

```
         [ 9],
         [ 6]]]), one_hot_data: tensor([[[0., 0., 0., 0., 0., 0., 0., 0., 1.,
0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
         [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
         [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
         [1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
         [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
         [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
         [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]]]), labels: tensor([ 3,
10,  8])
```

[6]:
```python
class MyRNN(nn.Module):
  """ A simple RNN that that lets you select architectural features when␣
 ↪creating
  a new instance """

  def __init__(self,
               input_size: int,
               hidden_size: int,
               output_size: int,
               num_layers: int = 1,
               use_LSTM: bool = True):

    super().__init__()

    if use_LSTM:
      self.rnn = nn.LSTM(
        input_size=input_size,
        hidden_size=hidden_size,
        num_layers=num_layers,
        batch_first=True)
    else:
      self.rnn = nn.RNN(
        input_size=input_size,
        hidden_size=hidden_size,
        num_layers=num_layers,
        batch_first=True)
```

```
    self.out_proj = nn.Linear(hidden_size, output_size)


  def forward(self, x: torch.Tensor):

    # require batched inputs: (B, MAX_L, V)
    # Here b = the batch size, l = sequence length , v = input token size (i.e.
    # the number of positiions in a one-hot-encoding array)
    assert x.ndim == 3
    b, l, v = x.shape

    # built-in PyTorch layer handles loop along sequence dimension,
    # including passing hidden state back each time step. It also
    # handles creating a new initial state for each batch!
    output, hidden = self.rnn(x)

    # for each item in batch, take final output state
    output = output[:,-1,:]

    # apply final linear layer to get predictions
    output = self.out_proj(output)

    return output


# declare the model and try out th untrained model on a sequence
model = MyRNN(
    input_size=11,
    hidden_size= 128,
    output_size=19, # allow outputs in range [0, 18]
    num_layers=2,
    use_LSTM=False)

# you should use `num_token` = 11: ten tokens for the digits, plus the special␣
 ↪-1 token
input =  MakeItHot(torch.tensor([[1,2,3,1,5,6,-1,9,3,4,5,0]]), num_tokens=11)
prediction = model(input)
print('My prediction is ', torch.argmax(prediction))
```

```
My prediction is  tensor(2)
```

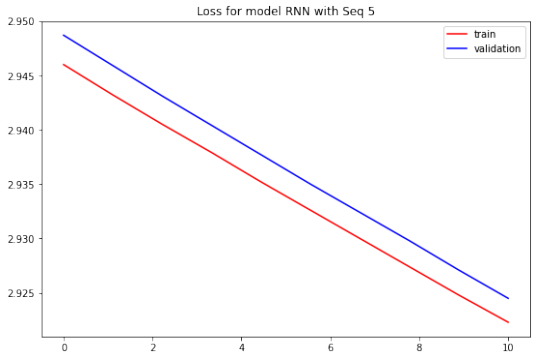
[13]:
```
def train_model(model, data, target, optimizer, criterion):
    model.train()

    optimizer.zero_grad()
```

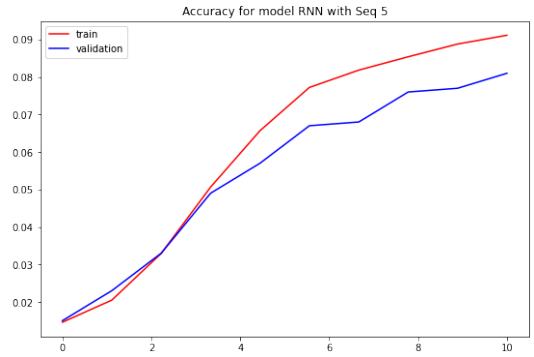
```
    output = model(data)
    loss = criterion(output.squeeze(), target)

    loss.backward()
    optimizer.step()


def test_model(model, data, target, criterion):
    model.eval()

    with torch.no_grad():

        output = model(data)
        output = output.squeeze()

        loss = criterion(output, target)

        correct = ((output.argmax(dim=-1) == target) > 0).sum().item()
        accuracy = correct / len(target)

        loss = round(loss.item(), 4)

    return accuracy, loss
```

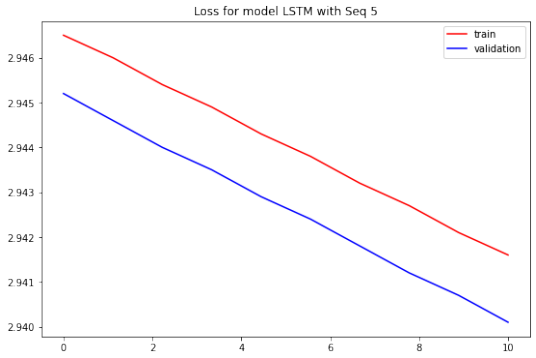
[8]:
```
def experiment(seq, epochs, model, optimizer, criterion, modelType):
  dataset = Add2DigitsDataset(num_examples=50000,seq_length=seq)
  data, one_hot_data, labels = dataset[:]
  v_dataset = Add2DigitsDataset(num_examples=1000,seq_length=seq)
  v_data, v_one_hot_data, v_labels = v_dataset[:]
  accs = []
  losses = []
  v_accs = []
  v_losses = []

  for epoch in range(epochs):
    train_model(model, one_hot_data, labels, optimizer, criterion)
    a, l = test_model(model, one_hot_data, labels, criterion)
    v_a, v_l = test_model(model, v_one_hot_data, v_labels, criterion)
    accs.append(a)
    losses.append(l)
    v_accs.append(v_a)
    v_losses.append(v_l)

  fig, ax = plt.subplots(1, 2, figsize=(20,6))
  ax[0].set_title("Loss for model {model} with Seq {s}".format(model=modelType,␣
↪s=str(seq)))
```

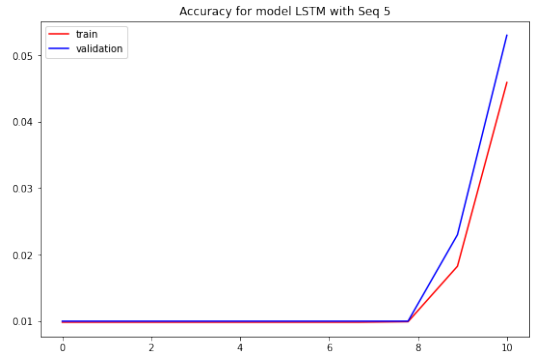
```
  ax[1].set_title("Accuracy for model {model} with Seq {s}".
↪format(model=modelType, s=str(seq)))
  ax[0].plot(np.linspace(0, epochs, epochs), losses, 'r',label="train")
  ax[0].plot(np.linspace(0, epochs, epochs), v_losses, 'b', label="validation")
  ax[0].legend()
  ax[1].plot(np.linspace(0, epochs, epochs), accs, 'r',label="train")
  ax[1].plot(np.linspace(0, epochs, epochs), v_accs, 'b', label="validation")
  ax[1].legend()

  plt.show()
```

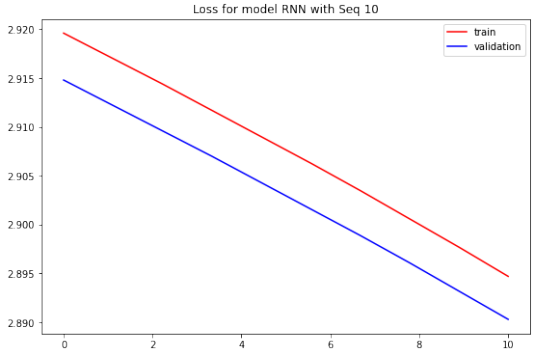
[14]:
```
modelRNN = MyRNN(
  input_size=11,
  hidden_size= 128,
  output_size=19, # allow outputs in range [0, 18]
  num_layers=2,
  use_LSTM=False)

modelLSTM = MyRNN(
  input_size=11,
  hidden_size= 128,
  output_size=19, # allow outputs in range [0, 18]
  num_layers=2,
  use_LSTM=True)

optimizerRNN = torch.optim.Adam(modelRNN.parameters(), lr=1e-4)
optimizerLSTM = torch.optim.Adam(modelLSTM.parameters(), lr=1e-4)
criterion = nn.CrossEntropyLoss()

#sequences = [5]
sequences = [5, 10, 30]
epochs = 10
for seq in sequences:
  experiment(seq, epochs, modelRNN, optimizerRNN, criterion, 'RNN')
  experiment(seq, epochs, modelLSTM, optimizerLSTM, criterion, 'LSTM')
```

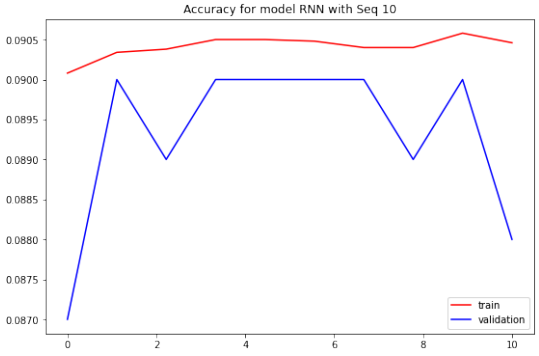


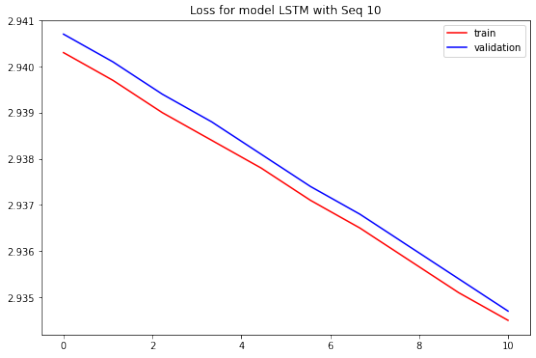


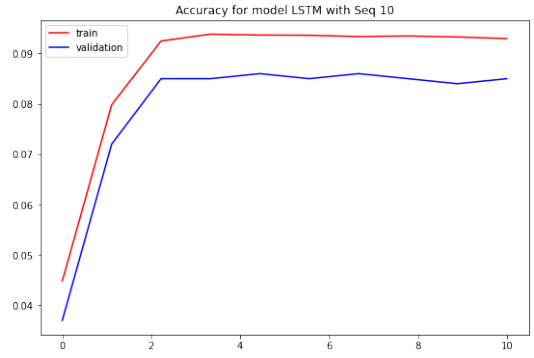

Loss for model LSTM with Seq 5
train
validation
2.946
2.945
2.944
2.943
2.942
2.941
2.940
0
2
4
6
8
10

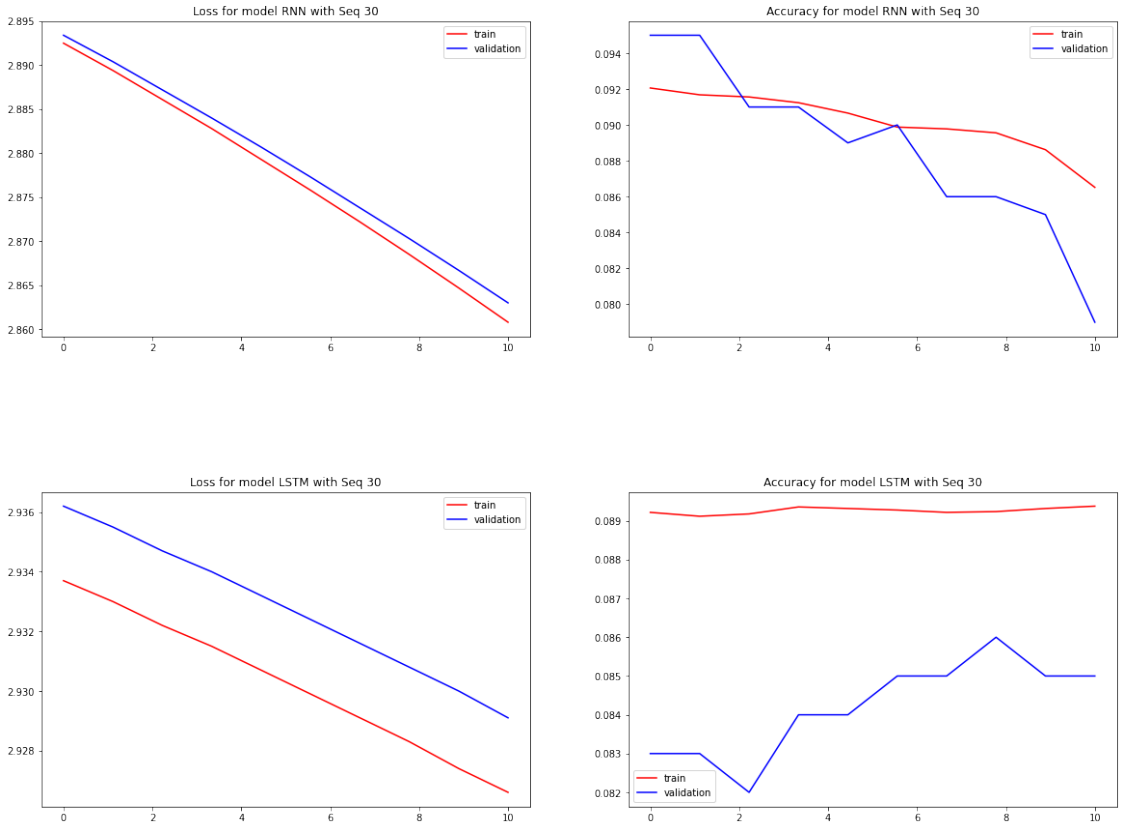

Accuracy for model LSTM with Seq 5
train
validation
0.05
0.04
0.03
0.02
0.01
0
2
4
6
8
10


Loss for model RNN with Seq 10
train
validation
2.920
2.915
2.910
2.905
2.900
2.895
2.890
0
2
4
6
8
10


Accuracy for model RNN with Seq 10
train
validation
0.0905
0.0900
0.0895
0.0890
0.0885
0.0880
0.0875
0.0870
0
2
4
6
8
10


Loss for model LSTM with Seq 10
train
validation
2.941
2.940
2.939
2.938
2.937
2.936
2.935
0
2
4
6
8
10


Accuracy for model LSTM with Seq 10
train
validation
0.09
0.08
0.07
0.06
0.05
0.04
0
2
4
6
8
10



1.4.3 2.3 Analysis (one point)

Provide your analysis of the above experiment. Was the hypothesis from question 2.1 supported by your data? Why or why not? What might be the limitations of your experiment?

(YOUR ANALYSIS HERE)

Comparing the performance of RNN and LSTM on three different sequence lengths trained on 10 epochs with 50,000 training examples and 1,000 validation examples, we observe that for low sequence length (5), RNN actually performs better than LSTM. RNN could achieve a validation accuracy of 8%, whereas LSTM could only achieve 5%. However, for longer sequence length (10 and 30), the training and validation accuracy for RNN becomes unstable and they even display a decreasing trend, where as the accuracies for LSTM display a generally increasing trend with respect to the training steps. In general, I think my data supports my hypothesis in 2.1 because LSTM performs better on longer memory tasks. One limitation of my experiment is that the training epochs are not large enough. Also, I should perform this experiment on larger datasets with a more precise discretization on the sequence length to produce a generalizable conclusion.

1.5 Question 3: Sequence-to-Sequence Translation (three points total)

Sequence-to-sequence tasks such as language translation require a model capable of converting variable-length inputs to variable-length outputs, with no guarantee that each position in the input will map directly to a position in the output. This can be thought of as a "many-to-many" task, as illustrated by the second figure from the right in Andrej Karpathy's diagram:

Image source: "The Unreasonable Effectiveness of Recurrent Neural Networks" (Karpathy)

For example, consider the following English-to-Spanish translation:

`I like chocolate` –> `Me gusta el chocolate`

The input sequence contains three words, while the output sequence contains four. Moreover, the individual words and grammatical structure of the sentences do not cleanly correspond, as `chocolate` is preceded by the definite article `el` and the subject switches from the speaker to the chocolate. It's easy to see why sequence-to-sequence translation can be a challenging task!

To overcome these difficulties, Sutskever et al. proposed to use recurrent **encoder** and **decoder** networks. First, special **start and stop tokens** are added to the ends of all sentences. Then, the encoder RNN loops over an input sequence, and its final hidden state is taken as a representation of the entire input "context." This context is passed to the decoder RNN, which generates one word at a time **autoregressively** (taking its own past predictions as inputs) until it produces a stop token. This allows for arbitrary input and output lengths, as (a) the only representation of the input that the decoder sees is a single context vector, and (b) the decoder is free to generate for as long as it wants to before emitting a stop token.

Image source: "Sequence to Sequence Learning with Neural Networks" (Sutskever et al. 2014). "EOS" is the end-of-sentence stop token.

In this question, we'll explore how to use the encoder-decoder architecture to perform German-to-English translation. This will require a bit of setup, which is documented in the next section. To simplify the process and avoid messing with your local Python environment, it's recommended that you use Google Colab.

As a final note, elements of this question are adapted from Ben Trevett's tutorials on sequence modeling in PyTorch. You are welcome to look at these tutorials for additional information.

1.5.1 3.1 Installation & Data Preparation (zero points)

Run these cells to prepare your environment and load data. You may be prompted to approve the installation by typing `y`, as well as to restart your notebook environment once the installation is complete.

[12]:
```
# the torchtext library provides a number of utilites for handling text data
!pip uninstall torchtext
!pip install torchtext==0.9.0 --no-cache-dir
!pip install torch==1.8.1+cu111 torchvision==0.9.1+cu111 torchaudio==0.8.1 -f␣
↪https://download.pytorch.org/whl/torch_stable.html
```

```
Found existing installation: torchtext 0.12.0
Uninstalling torchtext-0.12.0:
  Would remove:
```

```
    /usr/local/lib/python3.7/dist-packages/torchtext-0.12.0.dist-info/*
    /usr/local/lib/python3.7/dist-packages/torchtext/*
Proceed (y/n)? y
  Successfully uninstalled torchtext-0.12.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting torchtext==0.9.0
  Downloading torchtext-0.9.0-cp37-cp37m-manylinux1_x86_64.whl (7.1 MB)
     |                  | 7.1 MB 7.9 MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-
packages (from torchtext==0.9.0) (4.64.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from torchtext==0.9.0) (1.21.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from torchtext==0.9.0) (2.23.0)
Collecting torch==1.8.0
  Downloading torch-1.8.0-cp37-cp37m-manylinux1_x86_64.whl (735.5 MB)
     |                  | 735.5 MB 149.6 MB/s
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from torch==1.8.0->torchtext==0.9.0)
(4.2.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->torchtext==0.9.0)
(1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->torchtext==0.9.0) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->torchtext==0.9.0) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->torchtext==0.9.0)
(2022.5.18.1)
Installing collected packages: torch, torchtext
  Attempting uninstall: torch
    Found existing installation: torch 1.11.0+cu113
    Uninstalling torch-1.11.0+cu113:
      Successfully uninstalled torch-1.11.0+cu113
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
torchvision 0.12.0+cu113 requires torch==1.11.0, but you have torch 1.8.0 which
is incompatible.
torchaudio 0.11.0+cu113 requires torch==1.11.0, but you have torch 1.8.0 which
is incompatible.
Successfully installed torch-1.8.0 torchtext-0.9.0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.8.1+cu111
  Downloading https://download.pytorch.org/whl/cu111/torch-1.8.1%2Bcu111-cp37-cp
37m-linux_x86_64.whl (1982.2 MB)
     |                              | 834.1 MB 91.8 MB/s eta
0:00:13tcmalloc: large alloc 1147494400 bytes == 0x39d42000 @  0x7f1dafd3c615
0x592b76 0x4df71e 0x59afff 0x515655 0x549576 0x593fce 0x548ae9 0x51566f 0x549576
0x593fce 0x548ae9 0x5127f1 0x598e3b 0x511f68 0x598e3b 0x511f68 0x598e3b 0x511f68
0x4bc98a 0x532e76 0x594b72 0x515600 0x549576 0x593fce 0x548ae9 0x5127f1 0x549576
0x593fce 0x5118f8 0x593dd7
     |                              | 1055.7 MB 67.4 MB/s eta
0:00:14tcmalloc: large alloc 1434370048 bytes == 0x7e398000 @  0x7f1dafd3c615
0x592b76 0x4df71e 0x59afff 0x515655 0x549576 0x593fce 0x548ae9 0x51566f 0x549576
0x593fce 0x548ae9 0x5127f1 0x598e3b 0x511f68 0x598e3b 0x511f68 0x598e3b 0x511f68
0x4bc98a 0x532e76 0x594b72 0x515600 0x549576 0x593fce 0x548ae9 0x5127f1 0x549576
0x593fce 0x5118f8 0x593dd7
     |                            | 1336.2 MB 1.2 MB/s eta
0:08:47tcmalloc: large alloc 1792966656 bytes == 0x31ca000 @  0x7f1dafd3c615
0x592b76 0x4df71e 0x59afff 0x515655 0x549576 0x593fce 0x548ae9 0x51566f 0x549576
0x593fce 0x548ae9 0x5127f1 0x598e3b 0x511f68 0x598e3b 0x511f68 0x598e3b 0x511f68
0x4bc98a 0x532e76 0x594b72 0x515600 0x549576 0x593fce 0x548ae9 0x5127f1 0x549576
0x593fce 0x5118f8 0x593dd7
     |                         | 1691.1 MB 1.1 MB/s eta
0:04:15tcmalloc: large alloc 2241208320 bytes == 0x6dfb2000 @  0x7f1dafd3c615
0x592b76 0x4df71e 0x59afff 0x515655 0x549576 0x593fce 0x548ae9 0x51566f 0x549576
0x593fce 0x548ae9 0x5127f1 0x598e3b 0x511f68 0x598e3b 0x511f68 0x598e3b 0x511f68
0x4bc98a 0x532e76 0x594b72 0x515600 0x549576 0x593fce 0x548ae9 0x5127f1 0x549576
0x593fce 0x5118f8 0x593dd7
     |                        | 1982.2 MB 1.1 MB/s eta
0:00:01tcmalloc: large alloc 1982177280 bytes == 0xf3914000 @  0x7f1dafd3b1e7
0x4a3940 0x4a39cc 0x592b76 0x4df71e 0x59afff 0x515655 0x549576 0x593fce 0x511e2c
0x549576 0x593fce 0x511e2c 0x549576 0x593fce 0x511e2c 0x549576 0x593fce 0x511e2c
0x549576 0x593fce 0x511e2c 0x593dd7 0x511e2c 0x549576 0x593fce 0x548ae9 0x5127f1
0x549576 0x593fce 0x548ae9
tcmalloc: large alloc 2477727744 bytes == 0x169b6e000 @  0x7f1dafd3c615 0x592b76
0x4df71e 0x59afff 0x515655 0x549576 0x593fce 0x511e2c 0x549576 0x593fce 0x511e2c
0x549576 0x593fce 0x511e2c 0x549576 0x593fce 0x511e2c 0x549576 0x593fce 0x511e2c
0x593dd7 0x511e2c 0x549576 0x593fce 0x548ae9 0x5127f1 0x549576 0x593fce 0x548ae9
0x5127f1 0x549576
     |                        | 1982.2 MB 2.8 kB/s
Collecting torchvision==0.9.1+cu111
  Downloading https://download.pytorch.org/whl/cu111/torchvision-0.9.1%2Bcu111-c
p37-cp37m-linux_x86_64.whl (17.6 MB)
     |                        | 17.6 MB 47.2 MB/s
Collecting torchaudio==0.8.1
  Downloading torchaudio-0.8.1-cp37-cp37m-manylinux1_x86_64.whl (1.9 MB)
```

```
     |                | 1.9 MB 8.0 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-
packages (from torch==1.8.1+cu111) (1.21.0)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from torch==1.8.1+cu111) (4.2.0)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.7/dist-
packages (from torchvision==0.9.1+cu111) (7.1.2)
Installing collected packages: torch, torchvision, torchaudio
  Attempting uninstall: torch
    Found existing installation: torch 1.8.0
    Uninstalling torch-1.8.0:
      Successfully uninstalled torch-1.8.0
  Attempting uninstall: torchvision
    Found existing installation: torchvision 0.12.0+cu113
    Uninstalling torchvision-0.12.0+cu113:
      Successfully uninstalled torchvision-0.12.0+cu113
  Attempting uninstall: torchaudio
    Found existing installation: torchaudio 0.11.0+cu113
    Uninstalling torchaudio-0.11.0+cu113:
      Successfully uninstalled torchaudio-0.11.0+cu113
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
torchtext 0.9.0 requires torch==1.8.0, but you have torch 1.8.1+cu111 which is
incompatible.
Successfully installed torch-1.8.1+cu111 torchaudio-0.8.1
torchvision-0.9.1+cu111
```

We'll use the spaCy library to "tokenize" our input sentences (break them up into word units), but the library is also widely used for a variety of natural language processing tasks – feel free to take a look if that's of interest.

[3]:
```
# the spacy library will allow us to "tokenize" text datasets into individual␣
 ↪word units
!python -m spacy download en_core_web_sm
!python -m spacy download de_core_news_sm
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting en_core_web_sm==2.2.5
  Downloading https://github.com/explosion/spacy-
models/releases/download/en_core_web_sm-2.2.5/en_core_web_sm-2.2.5.tar.gz (12.0
MB)
     |                | 12.0 MB 9.4 MB/s
Requirement already satisfied: spacy>=2.2.2 in
/usr/local/lib/python3.7/dist-packages (from en_core_web_sm==2.2.5) (2.2.4)
```

```
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->en_core_web_sm==2.2.5) (7.4.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (2.23.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->en_core_web_sm==2.2.5) (57.4.0)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (1.0.0)
Requirement already satisfied: blis<0.5.0,>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (0.4.1)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->en_core_web_sm==2.2.5) (1.21.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (3.0.6)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (2.0.6)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (0.9.1)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (4.64.0)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (1.0.7)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (1.0.5)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (1.1.3)
Requirement already satisfied: importlib-metadata>=0.20 in
/usr/local/lib/python3.7/dist-packages (from
catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5) (4.11.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5)
(3.8.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/usr/local/lib/python3.7/dist-packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5)
(4.2.0)
Requirement already satisfied: certifi>=2017.4.17 in
```

```
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (2022.5.18.1)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5)
(2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (1.24.3)
 Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting de_core_news_sm==2.2.5
  Downloading https://github.com/explosion/spacy-
models/releases/download/de_core_news_sm-2.2.5/de_core_news_sm-2.2.5.tar.gz
(14.9 MB)
     |                  | 14.9 MB 1.3 MB/s
Requirement already satisfied: spacy>=2.2.2 in
/usr/local/lib/python3.7/dist-packages (from de_core_news_sm==2.2.5) (2.2.4)
Requirement already satisfied: blis<0.5.0,>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (0.4.1)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (1.0.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (3.0.6)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (2.0.6)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (1.1.3)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->de_core_news_sm==2.2.5) (1.21.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (4.64.0)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (1.0.7)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->de_core_news_sm==2.2.5) (57.4.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
```

```
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (2.23.0)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (1.0.5)
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->de_core_news_sm==2.2.5) (7.4.0)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->de_core_news_sm==2.2.5) (0.9.1)
Requirement already satisfied: importlib-metadata>=0.20 in
/usr/local/lib/python3.7/dist-packages (from
catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->de_core_news_sm==2.2.5) (4.11.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->de_core_news_sm==2.2.5)
(3.8.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/usr/local/lib/python3.7/dist-packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->de_core_news_sm==2.2.5)
(4.2.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->de_core_news_sm==2.2.5) (2022.5.18.1)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests<3.0.0,>=2.13.0->spacy>=2.2.2->de_core_news_sm==2.2.5)
(2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->de_core_news_sm==2.2.5) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->de_core_news_sm==2.2.5) (3.0.4)
  Download and installation successful
You can now load the model via spacy.load('de_core_news_sm')
```

[5]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchtext.legacy.datasets import Multi30k
from torchtext.legacy.data import Field, BucketIterator
import spacy
import numpy as np
import random
import math
import time
import en_core_web_sm
```

```
import de_core_news_sm
from typing import Union

# set device (default to GPU if available)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

We'll start by loading German and English text datasets. Then, we'll preprocess and reformat our string data so that we can pass it to PyTorch neural network models.

[9]:
```
# load spacy tokenizers for English and German text
spacy_de = de_core_news_sm.load()
spacy_en = en_core_web_sm.load()

def tokenize_de(text): return [tok.text for tok in spacy_de.tokenizer(text)]
def tokenize_de_reverse(text): return [tok.text for tok in spacy_de.
 ↪tokenizer(text)][::-1]
def tokenize_en(text): return [tok.text for tok in spacy_en.tokenizer(text)]
def tokenize_en_reverse(text): return [tok.text for tok in spacy_en.
 ↪tokenizer(text)][::-1]

# boilerplate for torchtext dataset formatting
source_field = Field(tokenize = tokenize_de, init_token = '<sos>', eos_token =␣
 ↪'<eos>', lower = True)
target_field = Field(tokenize = tokenize_en, init_token = '<sos>', eos_token =␣
 ↪'<eos>', lower = True)

# create datasets; by torchtext convention, instances are stored under the␣
 ↪`examples` attribute
train_data, valid_data, test_data = Multi30k.splits(exts = ('.de', '.en'),␣
 ↪fields = (source_field, target_field))

print(f"Dataset size (train/val/test): {len(train_data.examples)} /␣
 ↪{len(valid_data.examples)} / {len(test_data.examples)}")
```

```
Dataset size (train/val/test): 29000 / 1014 / 1000
```

We can see that for each instance, our dataset stores the input sentence (`src`) and output sentence (`trg`):

[6]:
```
print(vars(train_data.examples[0]))
```

```
{'src': ['zwei', 'junge', 'weiße', 'männer', 'sind', 'im', 'freien', 'in',
'der', 'nähe', 'vieler', 'büsche', '.'], 'trg': ['two', 'young', ',', 'white',
'males', 'are', 'outside', 'near', 'many', 'bushes', '.']}
```

Rather than one-hot-encode individual characters, we'll encode words. We'll associate each word with an index in the **vocabulary** of all words present in the data. To limit the size of this vocabulary (and thus our model's inputs), we'll associate all words that show up at most once with a single

"unknown" index. Note that we only consider words present in the training data – we can't expect our model to predict words it has never seen!

[7]:
```
# min_freq = 2 --> all words appearing only once are mapped to "unknown"
source_field.build_vocab(train_data, min_freq = 2)
target_field.build_vocab(train_data, min_freq = 2)

print(f"Unique tokens in source (de) vocabulary: {len(source_field.vocab)}")
print(f"Unique tokens in target (en) vocabulary: {len(target_field.vocab)}")

# rather than deal with DataLoader objects and collation, we'll simply
# use Python iterators to walk through our datasets
train_iterator, valid_iterator, test_iterator = BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = 128,
    device = device)
```

```
Unique tokens in source (de) vocabulary: 7855
Unique tokens in target (en) vocabulary: 5893
```

Our data preparation is now complete. When training, we can fetch one input/output pair at a time from our iterator, encode the input, and pass it to our model to make predictions and compute losses.

[8]:
```
# demonstration of how data loading will work in training
for i, batch in enumerate(train_iterator):

  # note that each batch is already padded to the maximum sequence length!
  x = batch.src
  y = batch.trg

  print(f'`x` is a {type(x)} of shape {x.shape} (in_seq_len, batch_size) \n`y`␣
↪is a {type(y)} of shape {y.shape} (out_seq_len, batch_size)')
  break
```

```
`x` is a <class 'torch.Tensor'> of shape torch.Size([27, 128]) (in_seq_len,
batch_size)
`y` is a <class 'torch.Tensor'> of shape torch.Size([33, 128]) (out_seq_len,
batch_size)
```

1.5.2 3.2 RNN Translation (one point)

Finally, we can train an encoder-decoder model to perform sequence-to-sequence translation. We'll operate the encoder similar to the "layer" format we discussed in notebook 8, while the decoder will predict for one step at a time – more akin to the "cell" format.

Modify the definitions in the cell below as instructed by the accompanying comments. Then, run the following cell to train and evaluate your encoder-decoder network for German-to-English translation. Finally, include a plot of your training and validation lossses by epoch where instructed below.

If your implementation is "correct," you should not see loss values far above `5.0` as your model trains. You can also check your implementation by running the example translation provided after the training code. Don't worry if your model isn't perfrect – as long as a few predicted words roughly correspond to the actual translation, it indicates that your implementation is mostly correct:

Actual Translation	Predicted Translation
`a dog walked by the park`	`a a animal a walked EOS EOS`

(YOUR LOSS PLOT HERE)

DO modify the following cell as instructed.

[9]:
```
class RNNEncoder(nn.Module):

    def __init__(self,
                 input_size: int,
                 embedding_size: int,
                 hidden_size: int,
                 num_layers: int):

        super().__init__()

        self.hidden_size = hidden_size
        self.num_layers = num_layers

        # embedding layer: similar to one-hot encoding, we will map
        # integer indices to vectors
        self.embedding = nn.Embedding(input_size, embedding_size)

        ############################################################
        # YOUR CODE STARTS HERE (1/5)
        ############################################################

        # define the RNN
        self.rnn = nn.RNN(
            input_size=embedding_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            dropout=0.5)

        ############################################################
        # YOUR CODE ENDS HERE (1/5)
        ############################################################

        # a droupout layer can improve generalization
```

```
        self.dropout = nn.Dropout(0.5)

    def forward(self, x: torch.Tensor):

        assert x.ndim == 2
        seq_len, batch_size = x.shape
        # convert integer indices to vector representations
        embedded = self.dropout(self.embedding(x))  # (seq_len, batch_size,␣
↪embedding_size)
        ###############################################################
        # YOUR CODE STARTS HERE (2/5)
        ###############################################################

        # pass the embedded input to the RNN and obtain the hidden state
        # for the final time step; dimension of hidden state should be
        # (num_layers, batch_size, hidden_size)
        #???, ??? = ???
        output, hidden = self.rnn(embedded)

        # return only the final hidden state
        return hidden

        ###############################################################
        # YOUR CODE ENDS HERE (2/5)
        ###############################################################


class RNNDecoder(nn.Module):

    def __init__(self,
                 output_size: int,
                 embedding_size: int,
                 hidden_size: int,
                 num_layers: int):

        super().__init__()

        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.output_size = output_size

        # embedding layer: similar to one-hot encoding, we will map
        # integer indices to vectors
        self.embedding = nn.Embedding(output_size, embedding_size)

        ###############################################################
        # YOUR CODE STARTS HERE (3/5)
```

```
        ############################################################

        # define the RNN
        self.rnn = nn.RNN(
            input_size=embedding_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            dropout=0.5)

        # define a fully-connected layer that maps vectors of dimension
        # `hidden_size` to vectors of dimension `output_size`
        self.out_proj = nn.Linear(in_features=hidden_size,␣
↪out_features=output_size)

        ############################################################
        # YOUR CODE ENDS HERE (3/5)
        ############################################################

        # a droupout layer can improve generalization
        self.dropout = nn.Dropout(0.5)

    def forward(self, x: torch.Tensor, hidden: torch.Tensor):

        # take a single time-step of input and a single hidden state
        assert x.ndim == 1
        x = x.unsqueeze(0)
        _, batch_size = x.shape

        # convert integer indices to vector representations
        embedded = self.dropout(self.embedding(x))  # (1, batch_size,␣
↪embedding_size)

        ############################################################
        # YOUR CODE STARTS HERE (4/5)
        ############################################################

        # call the RNN on our single-step input and prior hidden state
        # to obtain the output and hidden state; the dimension of the
        # output should be (1, batch_size, hidden_size), and the
        # dimension of the hidden state should be
        # (num_layers, batch_size, hidden_size)
        output, hidden_state = self.rnn(embedded, hidden)

        # apply fully-connected layer to map output dimension to (1,␣
↪batch_size, output_size)
        prediction = self.out_proj(output)
        # squeeze output to remove first (sequence) dimension
```

```
        prediction = prediction.squeeze(0)  # (batch_size, output_size)
        # return final output and hidden state
        return prediction, hidden_state

        ##########################################################
        # YOUR CODE ENDS HERE (4/5)
        ##########################################################


class RNNTranslator(nn.Module):

    def __init__(self,
                 encoder: nn.Module,
                 decoder: nn.Module,
                 device: Union[str, torch.device]):

        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.device = device

        assert encoder.hidden_size == decoder.hidden_size, \
        f"Mismatch: `encoder.hidden_size` = {encoder.hidden_size}, `decoder.
↪hidden_size` = {decoder.hidden_size}"

        assert encoder.num_layers == decoder.num_layers, \
        f"Mismatch: `encoder.num_layers` = {encoder.num_layers}, `decoder.
↪num_layers` = {decoder.num_layers}"

    def forward(self, x: torch.Tensor, y: torch.Tensor, teacher_forcing_ratio:␣
↪float = 0.5):

        # inputs should have shape (in_seq_len, batch_size)
        assert x.ndim == 2
        in_seq_len, batch_size = x.shape

        # targets should have shape (out_seq_len, batch size)
        assert y.ndim == 2
        out_seq_len, _ = y.shape
        out_vocab_size = self.decoder.output_size

        # prepare to store decoder outputs. We only need to allocate
        # the "true" number of time steps during training, but for inference
        # we can simply pass in an empty `y` of any length; this will serve as
        # a maximum length constraint on the output translation (which may end
        # sooner if an <EOS> token is predicted).
```

```
        outputs = torch.zeros(out_seq_len, batch_size, out_vocab_size).to(self.
↪device)

        ###############################################################
        # YOUR CODE STARTS HERE (5/5)
        ###############################################################

        # pass inputs to encoder to obtain final hidden state
        hidden = self.encoder(x)

        ###############################################################
        # YOUR CODE ENDS HERE (5/5)
        ###############################################################

        # first input to the decoder is the <SOS> start token
        input = y[0,:]

        for t in range(1, out_seq_len):

            # given input for current time step (either previous decoder
            # prediction or <SOS> token) and previous hidden state, compute
            # new output and hidden states
            output, hidden = self.decoder(input, hidden)

            # write predicted word probabilities to output buffer
            outputs[t] = output

            # during training, we may randomly decide whether to use
            # the decoder's previous predictions or the ground-truth
            # tokens for the previous time step. When we do the latter,
            # it is called "teacher-forcing." We will randomly apply
            # teacher-forcing during training only.
            teacher_force = random.random() < teacher_forcing_ratio

            # take highest-scoring token from decoder's previous prediction
            top1 = output.argmax(1)

            # optionally apply teacher-forcing
            input = y[t] if teacher_force else top1

        return outputs
```

DO NOT modify the following cell.

[13]:
```
# training hyperparameters
INPUT_DIM = len(source_field_.vocab)
OUTPUT_DIM = len(target_field.vocab)
```

```
ENC_EMB_DIM = 256
DEC_EMB_DIM = 256
HID_DIM = 512
N_LAYERS = 2
ENC_DROPOUT = 0.5
DEC_DROPOUT = 0.5
TRG_PAD_IDX = target_field.vocab.stoi[target_field.pad_token]
N_EPOCHS = 10
CLIP = 1

# initialize encoder-decoder model
enc = RNNEncoder(INPUT_DIM, ENC_EMB_DIM, HID_DIM, N_LAYERS)
dec = RNNDecoder(OUTPUT_DIM, DEC_EMB_DIM, HID_DIM, N_LAYERS)
model = RNNTranslator(enc, dec, device).to(device)

# initialize model weights
def init_weights(m):
    for name, param in m.named_parameters():
        nn.init.uniform_(param.data, -0.08, 0.08)
model.apply(init_weights)

# parameter count
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'The model has {count_parameters(model):,} trainable parameters')

# optimizer and loss function
optimizer = optim.Adam(model.parameters())
criterion = nn.CrossEntropyLoss(ignore_index = TRG_PAD_IDX)

# track training and validation losses
training_loss = []
validation_loss = []

def train(model, iterator, optimizer, criterion, clip):

    model.train()
    epoch_loss = 0

    for i, batch in enumerate(iterator):

        src = batch.src
        trg = batch.trg

        optimizer.zero_grad()
        output = model(src, trg)
```

```
        output_dim = output.shape[-1]
        output = output[1:].view(-1, output_dim)
        trg = trg[1:].view(-1)

        loss = criterion(output, trg)
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        epoch_loss += loss.item()

    return epoch_loss / len(iterator)

def evaluate(model, iterator, criterion):

    model.eval()
    epoch_loss = 0

    with torch.no_grad():

        for i, batch in enumerate(iterator):

            src = batch.src
            trg = batch.trg

            output = model(src, trg, 0) #turn off teacher forcing
            output_dim = output.shape[-1]

            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)

            loss = criterion(output, trg)
            epoch_loss += loss.item()

    return epoch_loss / len(iterator)

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):
```

```
    start_time = time.time()

    train_loss = train(model, train_iterator, optimizer, criterion, CLIP)
    valid_loss = evaluate(model, valid_iterator, criterion)

    training_loss.append(train_loss)
    validation_loss.append(valid_loss)

    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'rnn-model.pt')

    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f}')
    print(f'\t Val. Loss: {valid_loss:.3f}')

model.load_state_dict(torch.load('rnn-model.pt'))
test_loss = evaluate(model, test_iterator, criterion)

print(f'| Test Loss: {test_loss:.3f}')

# generate loss plot
import matplotlib.pyplot as plt
plt.plot(training_loss, label="training")
plt.plot(validation_loss, label="validation")
plt.legend()
plt.xlabel("epoch")
plt.ylabel("loss")
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-13-19779f805c44> in <module>
      1 # training hyperparameters
----> 2 INPUT_DIM = len(source_field.vocab)
      3 OUTPUT_DIM = len(target_field.vocab)
      4 ENC_EMB_DIM = 256
      5 DEC_EMB_DIM = 256

AttributeError: 'Field' object has no attribute 'vocab'
```

We can examine the model's translations over it's training data to get a sense of how well it is performing (you don't have to turn anything in for this part).

[11]:
```
def str_to_tensor(s: str, tokenize_fn, field):
  """Convert string to tensor of shape (seq_len, 1)"""

  s_tokenized = tokenize_fn(s)
  s_encoded = []

  for word in s_tokenized:

    if word in field.vocab.stoi:
      s_encoded.append(field.vocab.stoi[word])
    else:
      s_encoded.append(field.vocab.unk_index)

  return torch.as_tensor(s_encoded).reshape(-1, 1)

def tensor_to_str(t: torch.Tensor, field):
  """Convert tensor of shape (seq_len, 1) to string)"""

  s_encoded = t.flatten().tolist()
  s_tokenized = []

  for idx in s_encoded:
    s_tokenized.append(field.vocab.itos[idx])  # ignore "unknown" tokens

  return " ".join(s_tokenized)

# select a random training translation
german_phrase = " ".join(train_data.examples[0].src)
english_phrase = " ".join(train_data.examples[0].trg)
german_phrase_encoded = str_to_tensor(german_phrase, tokenize_de, source_field).
 ↪to(model.device)

german_phrase, english_phrase
```

[11]:
```
('zwei junge weiße männer sind im freien in der nähe vieler büsche .',
 'two young , white males are outside near many bushes .')
```

[13]:
```
model.eval()

with torch.no_grad():

    output = model(german_phrase_encoded, torch.
 ↪zeros_like(german_phrase_encoded), 0) #turn off teacher forcing
    output_dim = output.shape[-1]
    output = output[1:].view(-1, output_dim).argmax(dim=-1)

tensor_to_str(output, target_field)
```

[13]: `'a man in a a a a . <eos> . <eos> .'`

1.5.3 3.3 LSTM Translation (one point)

Next, we'll substitute `nn.RNN` for `nn.LSTM` in our encoder and decoder networks. This means we'll also have to deal with both hidden *and* cell states, as covered in notebook 8.

Modify the definitions in the cell below as instructed by the accompanying comments. Then, run the following cell to train and evaluate your encoder-decoder network for German-to-English translation. Finally, include a plot of your training and validation lossses by epoch where instructed below.

If your implementation is "correct," you should not see loss values far above `5.0` as your model trains. You can also check your implementation by running the example translation provided after the training code. Don't worry if your model isn't perfrect – as long as a few predicted words roughly correspond to the actual translation, it indicates that your implementation is mostly correct:

Actual Translation	Predicted Translation
`a dog walked by the park`	`a a animal a walked EOS EOS`

(YOUR LOSS PLOT HERE)

Test Loss: 3.843

DO modify the following cell as instructed.

[6]:
```
class LSTMEncoder(nn.Module):

    def __init__(self,
                 input_size: int,
                 embedding_size: int,
                 hidden_size: int,
                 num_layers: int):

        super().__init__()

        self.hidden_size = hidden_size
        self.num_layers = num_layers

        # embedding layer: similar to one-hot encoding, we will map
        # integer indices to vectors
        self.embedding = nn.Embedding(input_size, embedding_size)

        ############################################################
        # YOUR CODE STARTS HERE (1/5)
        ############################################################
```

```
        # define the LSTM
        self.rnn = nn.LSTM(
            input_size=embedding_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            dropout=0.5)

        ############################################################
        # YOUR CODE ENDS HERE (1/5)
        ############################################################

        # a droupout layer can improve generalization
        self.dropout = nn.Dropout(0.5)

    def forward(self, x: torch.Tensor):

        assert x.ndim == 2
        seq_len, batch_size = x.shape

        # convert integer indices to vector representations
        embedded = self.dropout(self.embedding(x))  # (seq_len, batch_size,␣
↪embedding_size)

        ############################################################
        # YOUR CODE STARTS HERE (2/5)
        ############################################################

        # pass the embedded input to the LSTM and obtain both the
        # hidden state and cell state for the final time step; the
        # dimension of both should be (num_layers, batch_size, hidden_size)
        output, (hidden, cell) = self.rnn(embedded)

        # return only the final hidden state and cell state
        return hidden, cell

        ############################################################
        # YOUR CODE ENDS HERE (2/5)
        ############################################################


class LSTMDecoder(nn.Module):

    def __init__(self,
                 output_size: int,
                 embedding_size: int,
                 hidden_size: int,
```

```
                 num_layers: int):

        super().__init__()

        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.output_size = output_size

        # embedding layer: similar to one-hot encoding, we will map
        # integer indices to vectors
        self.embedding = nn.Embedding(output_size, embedding_size)

        ############################################################
        # YOUR CODE STARTS HERE (3/5)
        ############################################################

        # define the LSTM
        self.rnn = nn.LSTM(
            input_size=embedding_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            dropout=0.5)

        # define a fully-connected layer that maps vectors of dimension
        # `hidden_size` to vectors of dimension `output_size`
        self.out_proj = nn.Linear(in_features=hidden_size,␣
↪out_features=output_size)

        ############################################################
        # YOUR CODE ENDS HERE (3/5)
        ############################################################

        # a droupout layer can improve generalization
        self.dropout = nn.Dropout(0.5)

    def forward(self, x: torch.Tensor, hidden: torch.Tensor, cell: torch.
↪Tensor):

        # take a single time-step of input, a single hidden state, and a single␣
↪cell state
        assert x.ndim == 1
        x = x.unsqueeze(0)
        _, batch_size = x.shape

        # convert integer indices to vector representations
        embedded = self.dropout(self.embedding(x))  # (1, batch_size,␣
↪embedding_size)
```

```
        ##############################################################
        # YOUR CODE STARTS HERE (4/5)
        ##############################################################

        # call the LSTM on our single-step input and a tuple containing
        # the hidden and cell states. This should return an output state,
        # a final hidden state, and a final cell state. The dimension of
        # the output should be (1, batch_size, hidden_size), and the
        # dimension of the updated hidden and cell states should be
        # (num_layers, batch_size, hidden_size)
        output, (hidden_state, cell_state) = self.rnn(embedded, (hidden, cell))

        # apply fully-connected layer to map output dimension to (1,␣
↪batch_size, output_size)
        prediction = self.out_proj(output)

        # squeeze output to remove first (sequence) dimension
        prediction = prediction.squeeze(0)  # (batch_size, output_size)

        # return final output, hidden state, and cell state
        return prediction, hidden_state, cell_state

        ##############################################################
        # YOUR CODE ENDS HERE (4/5)
        ##############################################################


class LSTMTranslator(nn.Module):

    def __init__(self,
                 encoder: nn.Module,
                 decoder: nn.Module,
                 device: Union[str, torch.device]):

        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.device = device

        assert encoder.hidden_size == decoder.hidden_size, \
        f"Mismatch: `encoder.hidden_size` = {encoder.hidden_size}, `decoder.
↪hidden_size` = {decoder.hidden_size}"

        assert encoder.num_layers == decoder.num_layers, \
```

```
        f"Mismatch: `encoder.num_layers` = {encoder.num_layers}, `decoder.
↪num_layers` = {decoder.num_layers}"

    def forward(self, x: torch.Tensor, y: torch.Tensor, teacher_forcing_ratio:␣
↪float = 0.5):

        # inputs should have shape (in_seq_len, batch_size)
        assert x.ndim == 2
        in_seq_len, batch_size = x.shape

        # targets should have shape (out_seq_len, batch size)
        assert y.ndim == 2
        out_seq_len, _ = y.shape
        out_vocab_size = self.decoder.output_size

        # prepare to store decoder outputs. We only need to allocate
        # the "true" number of time steps during training, but for inference
        # we can simply pass in an empty `y` of any length; this will serve as
        # a maximum length constraint on the output translation (which may end
        # sooner if an <EOS> token is predicted).
        outputs = torch.zeros(out_seq_len, batch_size, out_vocab_size).to(self.
↪device)

        ############################################################
        # YOUR CODE STARTS HERE (5/5)
        ############################################################

        # pass inputs to encoder to obtain final hidden state and
        # cell state
        hidden, cell = self.encoder(x)

        ############################################################
        # YOUR CODE ENDS HERE (5/5)
        ############################################################

        # first input to the decoder is the <SOS> start token
        input = y[0,:]

        for t in range(1, out_seq_len):

            # given input for current time step (either previous decoder
            # prediction or <SOS> token) and previous hidden state and
            # cell state, compute new output, hidden, and cell states
            output, hidden, cell = self.decoder(input, hidden, cell)

            # write predicted word probabilities to output buffer
            outputs[t] = output
```

```
            # during training, we may randomly decide whether to use
            # the decoder's previous predictions or the ground-truth
            # tokens for the previous time step. When we do the latter,
            # it is called "teacher-forcing." We will randomly apply
            # teacher-forcing during training only.
            teacher_force = random.random() < teacher_forcing_ratio

            # take highest-scoring token from decoder's previous prediction
            top1 = output.argmax(1)

            # optionally apply teacher-forcing
            input = y[t] if teacher_force else top1

        return outputs
```

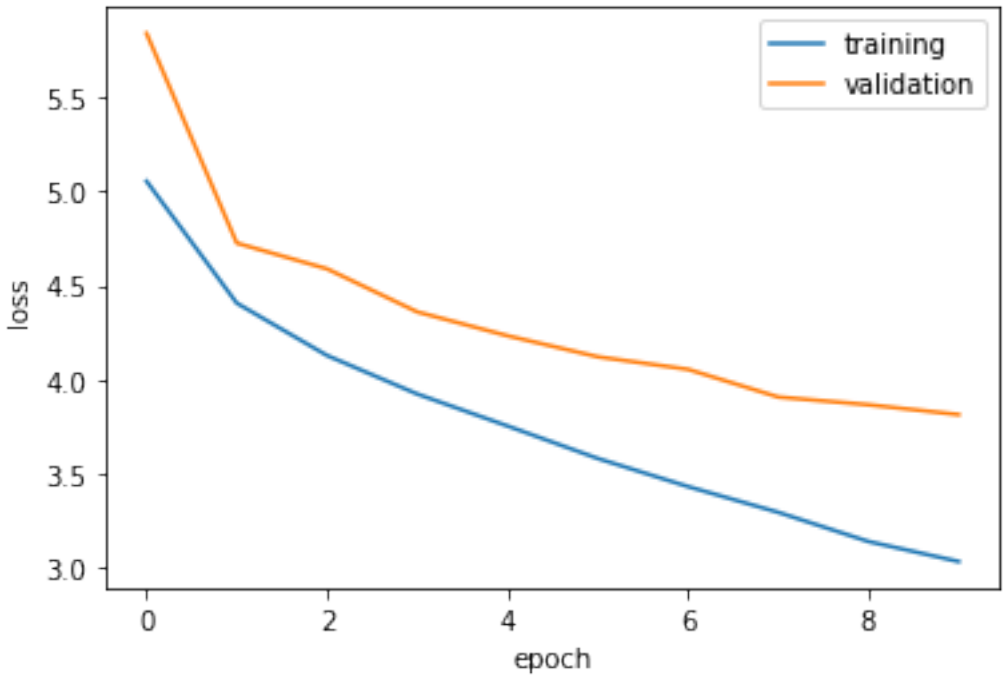
DO NOT modify the following cell.

[31]:
```
# initialize encoder-decoder model
enc_lstm = LSTMEncoder(INPUT_DIM, ENC_EMB_DIM, HID_DIM, N_LAYERS)
dec_lstm = LSTMDecoder(OUTPUT_DIM, DEC_EMB_DIM, HID_DIM, N_LAYERS)
model_lstm = LSTMTranslator(enc_lstm, dec_lstm, device).to(device)

# initialize model weights
def init_weights(m):
    for name, param in m.named_parameters():
        nn.init.uniform_(param.data, -0.08, 0.08)
model_lstm.apply(init_weights)

# parameter count
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'The model has {count_parameters(model_lstm):,} trainable parameters')

# optimizer and loss function
optimizer = optim.Adam(model_lstm.parameters())
criterion = nn.CrossEntropyLoss(ignore_index = TRG_PAD_IDX)

# track training and validation losses
training_loss = []
validation_loss = []

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()
```

```
    train_loss = train(model_lstm, train_iterator, optimizer, criterion, CLIP)
    valid_loss = evaluate(model_lstm, valid_iterator, criterion)

    training_loss.append(train_loss)
    validation_loss.append(valid_loss)

    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model_lstm.state_dict(), 'lstm-model.pt')

    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f}')
    print(f'\t Val. Loss: {valid_loss:.3f}')

model_lstm.load_state_dict(torch.load('lstm-model.pt'))
test_loss = evaluate(model_lstm, test_iterator, criterion)

print(f'| Test Loss: {test_loss:.3f}')

# generate loss plot
import matplotlib.pyplot as plt
plt.plot(training_loss, label="training")
plt.plot(validation_loss, label="validation")
plt.legend()
plt.xlabel("epoch")
plt.ylabel("loss")
plt.show()
```

```
The model has 13,899,013 trainable parameters
Epoch: 01 | Time: 0m 28s
        Train Loss: 5.053
         Val. Loss: 5.837
Epoch: 02 | Time: 0m 28s
        Train Loss: 4.405
         Val. Loss: 4.724
Epoch: 03 | Time: 0m 28s
        Train Loss: 4.127
         Val. Loss: 4.587
Epoch: 04 | Time: 0m 28s
        Train Loss: 3.923
         Val. Loss: 4.359
Epoch: 05 | Time: 0m 28s
        Train Loss: 3.753
```

```
         Val. Loss: 4.232
Epoch: 06 | Time: 0m 28s
        Train Loss: 3.581
         Val. Loss: 4.120
Epoch: 07 | Time: 0m 28s
        Train Loss: 3.432
         Val. Loss: 4.054
Epoch: 08 | Time: 0m 28s
        Train Loss: 3.295
         Val. Loss: 3.907
Epoch: 09 | Time: 0m 28s
        Train Loss: 3.140
         Val. Loss: 3.866
Epoch: 10 | Time: 0m 28s
        Train Loss: 3.034
         Val. Loss: 3.813
| Test Loss: 3.843
```



[32]:
```
# select a random training translation
german_phrase = " ".join(train_data.examples[0].src)
english_phrase = " ".join(train_data.examples[0].trg)
german_phrase_encoded = str_to_tensor(german_phrase, tokenize_de, source_field).
 ↪to(model.device)

model_lstm.eval()
```

```
with torch.no_grad():

    output = model_lstm(german_phrase_encoded, torch.
 ↪zeros_like(german_phrase_encoded), 0) #turn off teacher forcing
    output_dim = output.shape[-1]
    output = output[1:].view(-1, output_dim).argmax(dim=-1)

tensor_to_str(output, target_field)
```

[32]:
```
'two young men are in in a kitchen . <eos> <eos> <eos>'
```

1.5.4 3.4 Reversing Sequences (one point)

Sutskever et al. also performed experiments in which the source sequence was reversed before being passed to the encoder network. We

Modify the data-loading code above so that all references to `tokenize_de` are replaced with `tokenize_de_reverse`. Re-generate the data and re-train your LSTM model from question 3.3. Include a plot of your training and validation lossses by epoch where instructed below. Finally, give a brief explanation of how (if at all) your network's performance changed, and why (if a change occurs).

(YOUR LOSS PLOT HERE)

Test Loss: 4.036

(YOUR EXPLANATION HERE)

[7]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# load spacy tokenizers for English and German text
spacy_de = de_core_news_sm.load()
spacy_en = en_core_web_sm.load()
```

[10]:
```
# boilerplate for torchtext dataset formatting with reverse
source_field_re = Field(tokenize = tokenize_de_reverse, init_token = '<sos>',␣
 ↪eos_token = '<eos>', lower = True)
target_field_re = Field(tokenize = tokenize_de_reverse, init_token = '<sos>',␣
 ↪eos_token = '<eos>', lower = True)

train_data_re, valid_data_re, test_data_re = Multi30k.splits(exts = ('.de', '.
 ↪en'), fields = (source_field_re, target_field_re))
```

[11]:
```
#code for reverse dataset
# min_freq = 2 --> all words appearing only once are mapped to "unknown"
source_field_re.build_vocab(train_data_re, min_freq = 2)
target_field_re.build_vocab(train_data_re, min_freq = 2)

print(f"Unique tokens in source (de) vocabulary: {len(source_field_re.vocab)}")
```

```
print(f"Unique tokens in target (en) vocabulary: {len(target_field_re.vocab)}")

# rather than deal with DataLoader objects and collation, we'll simply
# use Python iterators to walk through our datasets
train_iterator_re, valid_iterator_re, test_iterator_re = BucketIterator.splits(
    (train_data_re, valid_data_re, test_data_re),
    batch_size = 128,
    device = device)
```

```
Unique tokens in source (de) vocabulary: 7855
Unique tokens in target (en) vocabulary: 5923
```

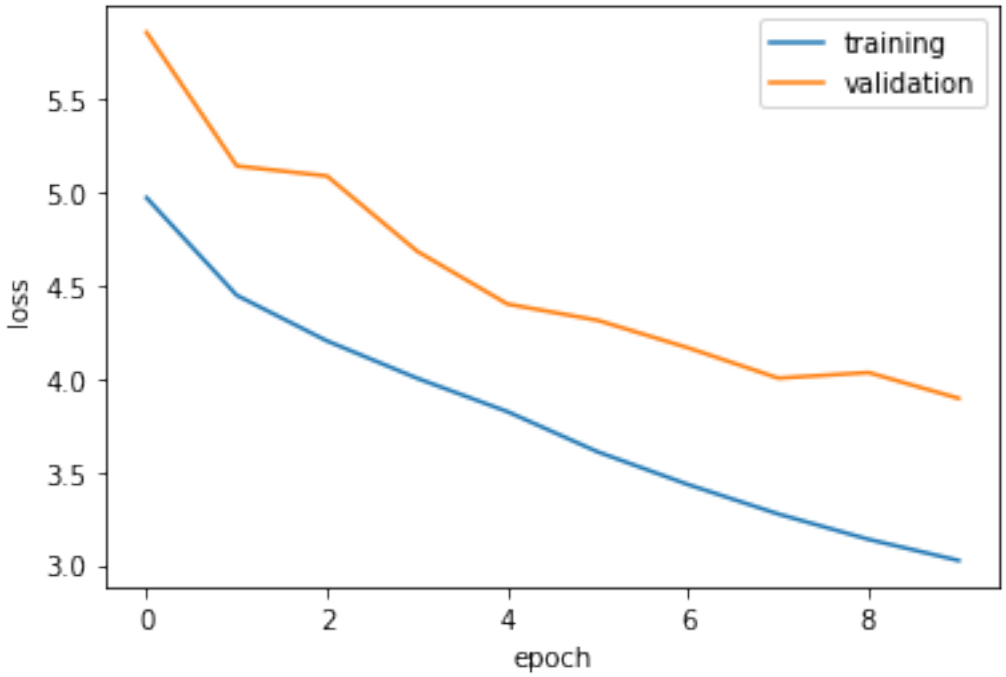
[20]:
```
# training hyperparameters
INPUT_DIM = len(source_field_re.vocab)
OUTPUT_DIM = len(target_field_re.vocab)
ENC_EMB_DIM = 256
DEC_EMB_DIM = 256
HID_DIM = 512
N_LAYERS = 2
ENC_DROPOUT = 0.5
DEC_DROPOUT = 0.5
TRG_PAD_IDX = target_field_re.vocab.stoi[target_field.pad_token]
N_EPOCHS = 10
CLIP = 1
# initialize model weights
def init_weights(m):
    for name, param in m.named_parameters():
        nn.init.uniform_(param.data, -0.08, 0.08)

# parameter count
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)


# track training and validation losses
training_loss = []
validation_loss = []

def train(model, iterator, optimizer, criterion, clip):

    model.train()
    epoch_loss = 0

    for i, batch in enumerate(iterator):

        src = batch.src
        trg = batch.trg
```

```
        optimizer.zero_grad()
        output = model(src, trg)

        output_dim = output.shape[-1]
        output = output[1:].view(-1, output_dim)
        trg = trg[1:].view(-1)

        loss = criterion(output, trg)
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        epoch_loss += loss.item()

    return epoch_loss / len(iterator)

def evaluate(model, iterator, criterion):

    model.eval()
    epoch_loss = 0

    with torch.no_grad():

        for i, batch in enumerate(iterator):

            src = batch.src
            trg = batch.trg

            output = model(src, trg, 0) #turn off teacher forcing
            output_dim = output.shape[-1]

            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)

            loss = criterion(output, trg)
            epoch_loss += loss.item()

    return epoch_loss / len(iterator)

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

[21]:
```
# initialize encoder-decoder model
enc_lstm = LSTMEncoder(INPUT_DIM, ENC_EMB_DIM, HID_DIM, N_LAYERS)
dec_lstm = LSTMDecoder(OUTPUT_DIM, DEC_EMB_DIM, HID_DIM, N_LAYERS)
model_lstm = LSTMTranslator(enc_lstm, dec_lstm, device).to(device)

# initialize model weights
def init_weights(m):
    for name, param in m.named_parameters():
        nn.init.uniform_(param.data, -0.08, 0.08)
model_lstm.apply(init_weights)

# parameter count
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'The model has {count_parameters(model_lstm):,} trainable parameters')

# optimizer and loss function
optimizer = optim.Adam(model_lstm.parameters())
criterion = nn.CrossEntropyLoss(ignore_index = TRG_PAD_IDX)

# track training and validation losses
training_loss = []
validation_loss = []

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss = train(model_lstm, train_iterator_re, optimizer, criterion,␣
↪CLIP)
    valid_loss = evaluate(model_lstm, valid_iterator_re, criterion)

    training_loss.append(train_loss)
    validation_loss.append(valid_loss)

    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model_lstm.state_dict(), 'lstm-model.pt')

    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f}')
    print(f'\t Val. Loss: {valid_loss:.3f}')
```

```
model_lstm.load_state_dict(torch.load('lstm-model.pt'))
test_loss = evaluate(model_lstm, test_iterator_re, criterion)

print(f'| Test Loss: {test_loss:.3f}')

# generate loss plot
import matplotlib.pyplot as plt
plt.plot(training_loss, label="training")
plt.plot(validation_loss, label="validation")
plt.legend()
plt.xlabel("epoch")
plt.ylabel("loss")
plt.show()
```

```
The model has 13,922,083 trainable parameters
Epoch: 01 | Time: 0m 28s
        Train Loss: 4.969
         Val. Loss: 5.855
Epoch: 02 | Time: 0m 28s
        Train Loss: 4.446
         Val. Loss: 5.139
Epoch: 03 | Time: 0m 28s
        Train Loss: 4.201
         Val. Loss: 5.085
Epoch: 04 | Time: 0m 28s
        Train Loss: 4.001
         Val. Loss: 4.682
Epoch: 05 | Time: 0m 28s
        Train Loss: 3.822
         Val. Loss: 4.399
Epoch: 06 | Time: 0m 28s
        Train Loss: 3.607
         Val. Loss: 4.312
Epoch: 07 | Time: 0m 28s
        Train Loss: 3.432
         Val. Loss: 4.165
Epoch: 08 | Time: 0m 28s
        Train Loss: 3.274
         Val. Loss: 4.002
Epoch: 09 | Time: 0m 28s
        Train Loss: 3.138
         Val. Loss: 4.032
Epoch: 10 | Time: 0m 28s
        Train Loss: 3.025
         Val. Loss: 3.894
| Test Loss: 4.036
```



1.6 Question 4: Short Response (two points total)

1.6.1 4.1 State Initialization (one point)

Assume we are given the task from notebook 8, in which we must predict the language from which a given name originates. Assume we are training an RNN in "cell" configuration, meaning we must manually handle the propagation of states, and that we pass our network one training example at a time. Finally, recall that in notebook 8, we demonstrate how to re-initialize the RNN cell's hidden state with a zero vector.

What kind of information does the RNN cell's hidden state hold *after* processing a full name input (but before re-initializing)?

(YOUR ANSWER HERE)

It holds the one-hot-encoded tensor for the last letter in the full name input and some information about the previous letters.

Assume that we do not re-initialize the RNN cell's hidden state after processing a name input. What does this mean for the hidden state when we start processing the next name input?

(YOUR ANSWER HERE)

We will be passing the information from the last full name input as a hidden state tensor to the current name input, and the RNN cell will learn information from both names.

Based on your answers above, when does it make most sense to re-initialize the hidden state for our task?

1. **Only once, at the beginning of training**
2. **After processing each individual name input**
3. **After processing each epoch (i.e. after iterating over all names once)**

Include a brief explanation supporting your answer.

(YOUR ANSWER HERE)

We should re-initialize the hidden state after processing each individual name input since we want to learn about the information of the letters in one name, but we don't want to be bothered by other names' information.

1.6.2 4.2 Network Size (one point)

Assume we are given a simplified version of our task from notebook 8, in which we must predict the language from which a given name originates. In this simplified task, names are at most 10 characters long, and characters are drawn only from the standard english alphabet (26 possibilities). Names are drawn from the same 18 languages. Assume all inputs are one-hot encoded.

We consider two different network architectures:

- Network A pads all inputs to the maximum sequence length, flattens them, and passes them through two fully connected layers with sizes 256 and 18, respectively.
- Network B is a recurrent neural network identical to `RNNCell` in notebook 8, with hidden size 256 and output size 18.

How many parameters does each network contain? Explain your reasoning.

(YOUR ANSWER HERE)

Assuming there are no bias terms in both networks. Network A has 71,168 parameters, Network B has 77,268 parameters.

What if the maximum input length is 20 characters? What does your answer say about the sizes of fully-connected and recurrent networks as the sequence length grows?

(YOUR ANSWER HERE)

The Network A now has 137,728 parameters, while the Network B's parameters stay the same. Fully-connected network have fewer parameters when the sequence length is small, but its size grows with the sequence length; recurrent networks are not impacted by the sequence length.

[7]:
```
class RNNCell(nn.Module):

  def __init__(self,
               input_size: int,
               hidden_size: int,
               output_size: int):

    super().__init__()

    self.hidden_size = hidden_size
```

```
    self.i2h = nn.Linear(input_size + hidden_size, hidden_size, bias=False)
    self.i2o = nn.Linear(input_size + hidden_size, output_size, bias=False)

  def forward(self, input, hidden):
    if input.ndim > 2:
      L, B, V, *_ = input.shape
      assert L == 1
      input = input.squeeze(0)

    combined = torch.cat((input, hidden), -1)
    hidden = self.i2h(combined)
    output = self.i2o(combined)
    return output, hidden


class Net(nn.Module):
  def __init__(self, input_size, output_size):
    super().__init__()
    self.f1 = nn.Linear(input_size, 256, bias=False)
    self.f2 = nn.Linear(256, output_size, bias=False)
  def forward(self, x):
    x = self.f1(x)
    x = self.f2(x)
    return x
```

[8]:
```
hidden_size = 256
n_languages = 18
n_letters = 26

rnncell = RNNCell(
    input_size=n_letters,    # inputs are one-hot-encoded (one entry per␣
 ↪possible letter)
    hidden_size=hidden_size, # pick a large enough hidden size
    output_size=n_languages  # output a vector of class scores, one per language
)
net = Net(input_size=260, output_size=n_languages)
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'The cell has {count_parameters(rnncell):,} trainable parameters')
print(f'The net has {count_parameters(net):,} trainable parameters')
```

```
The cell has 77,268 trainable parameters
The net has 71,168 trainable parameters
```

[9]:
```
net2 = Net(input_size=520, output_size=n_languages)
rnncell2 = RNNCell(
```

```
    input_size=n_letters,    # inputs are one-hot-encoded (one entry per␣
 ↪possible letter)
    hidden_size=hidden_size, # pick a large enough hidden size
    output_size=n_languages  # output a vector of class scores, one per language
)
print(f'The cell has {count_parameters(rnncell2):,} trainable parameters')
print(f'The net has {count_parameters(net2):,} trainable parameters')
```

```
The cell has 77,268 trainable parameters
The net has 137,728 trainable parameters
```

[ ]: