Lior Ben Ishay - 313135261 Keren Druker - 208674487

EX 2 – RNN

Please read the submission guidelines before you start.

Theory

1. In RNN, and specifically when working on sequence to sequence tasks, one problem is the varying length of the sequences.
   a. Describe two ways (or more) to deal with variable-length **input** sequences.
   b. Describe two ways (or more) to deal with variable-length **output** sequences.

Variable-length Input sequences

1. use a parameter: sequence-length. can be used in static/dynamic rnn it allows to handle sequnces of varying lengths without padding and adapts the computation to the acutal input length.

2. padding - most common approach all sequnces are padded with special symbol to match the length of the longest input this symbol is ignored in training.

Variable-length Output sequences

1. EOS token: An End-Of-Sequence token indicates when the model should stop generating output.

2. Teacher Forcing - During training insted of feeding the model the previous output as next input. the actual output from training set is used. This way the model learns the correct sequences length.

2. Name two advantages of GRU over LSTM.

LSTM- long Short-Term Memory - was designed to overcome the venishing gradient problem in traditional RNN. It contains memory cell that maintain its state over time. it has 3 types of gates (input, forget, output) to controll the information.

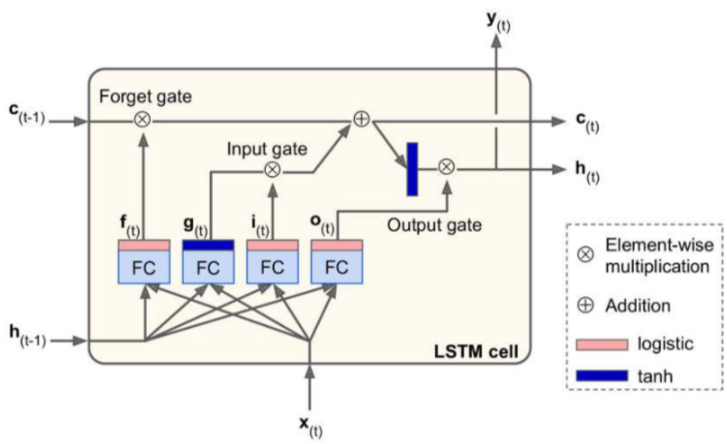
GRU (Gated Recurrent Unit) simpler version of LSTM. Combines the forget and input gates into update gate

GRU is simpler architecture with fewer gates, fewer parameters, therefore easier to train and understanding and it is more efficient and faster.

3. The LSTM cell equations are as follows:

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{i})$$
$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{f})$$
$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{o})$$
$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{g})$$
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$
$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

An illustration of an LSTM cell:



A network based on a single LSTM cell uses a vector of size 200 to describe the current state. Its input size are 200 sized vectors. Considering only parameters related to the cell, how many parameters does it have?

An LSTM cell has three gates (input, forget, output) and a state cell, each gate has its own weight matrices and bais vector. each one has 2 weight metrices on for the input and on for the previous hidden state

for example: $W_1^T X(t) + W_2^T h(t-1) + b$

h is the hidden state vector (200)

X is the input vector (200)

each state has input matrix size: $h*X$ and hidden state matrix size: $h*h$

in total 4 components (3 gates 1 cell state) $4*((h*x)+(h*h)+h) = 160{,}800$

$x = 200$
$h = 200$

in addion each state has FC layer with $200*200$ parameter

$4*200*200 = 160{,}000$

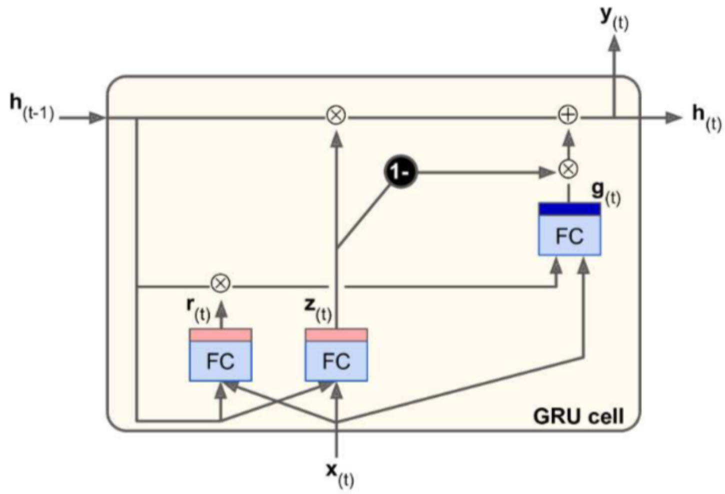
in total $160.000 + 160.800 = 320{,}800$

4. The GRU equations are as follows:

$$
\begin{aligned}
\mathbf{z}_{(t)} &= \sigma(\mathbf{W}_{xz}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z) \\
\mathbf{r}_{(t)} &= \sigma(\mathbf{W}_{xr}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r) \\
\mathbf{g}_{(t)} &= \tanh\,(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g) \\
\mathbf{h}_{(t)} &= \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}
\end{aligned}
$$

Where $\sigma(x) = \frac{1}{1+e^{-x}}$

An illustration of a GRU cell:



$$\frac{\partial \sigma(x)}{\sigma(x)} = \sigma(x)(1-\sigma(x))$$

$$\frac{\partial \tanh x}{\partial x} = 1 - \tanh^2 x$$

Consider GRU network with two timestamp (e.g. two iterations of the GRU cell), with a defined loss $\epsilon_{(t)}$ (e.g., the l_2 loss: $\frac{1}{2}\left(h_{(t)} - y_t\right)^2$).

Assume the gradient $\frac{\partial \epsilon_{(2)}}{\partial h_{(2)}}$ is given.

We would like to calculate the gradients of GRU for back propagation. For simplicity, you may ignore the bias, <u>and calculate the gradients of the second time stamp only</u>. Using the chain rule, Calculate:

a. $\frac{\partial \epsilon_{(2)}}{\partial W_{xz}} = \frac{\partial E}{\partial h} \cdot \frac{\partial h}{\partial z} \cdot \frac{\partial z}{\partial W_{xz}} = \frac{\partial E}{\partial h}\left(h(t-1) - g(t)\right) \cdot \sigma_{z_2}(1-\sigma_{z_2}) X(2)$

b. $\frac{\partial \epsilon_{(2)}}{\partial W_{hz}} = \frac{\partial E}{\partial h} \cdot \frac{\partial h}{\partial z} \cdot \frac{\partial z}{\partial W_{xz}} = \frac{\partial E}{\partial h}\left(h(t-1) - g(t)\right) \cdot \sigma_{z_2}(1-\sigma_{z_2}) h(1)$

c. $\frac{\partial \epsilon_{(2)}}{\partial W_{xg}} = \frac{\partial E}{\partial h} \cdot \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial W_{hg}} = \frac{\partial E}{\partial h}(1 - z(2))(1 - \tanh^2_{g_t}) X(2)$

d. $\frac{\partial \epsilon_{(2)}}{\partial W_{hg}} = \frac{\partial E}{\partial h} \cdot \frac{\partial h}{\partial g} \cdot \frac{\partial g}{W_{hg}} = \frac{\partial E}{\partial h}(1 - z(2))\left(1 - \tanh^2_{g_t}\right) r(2) \cdot h(1)$

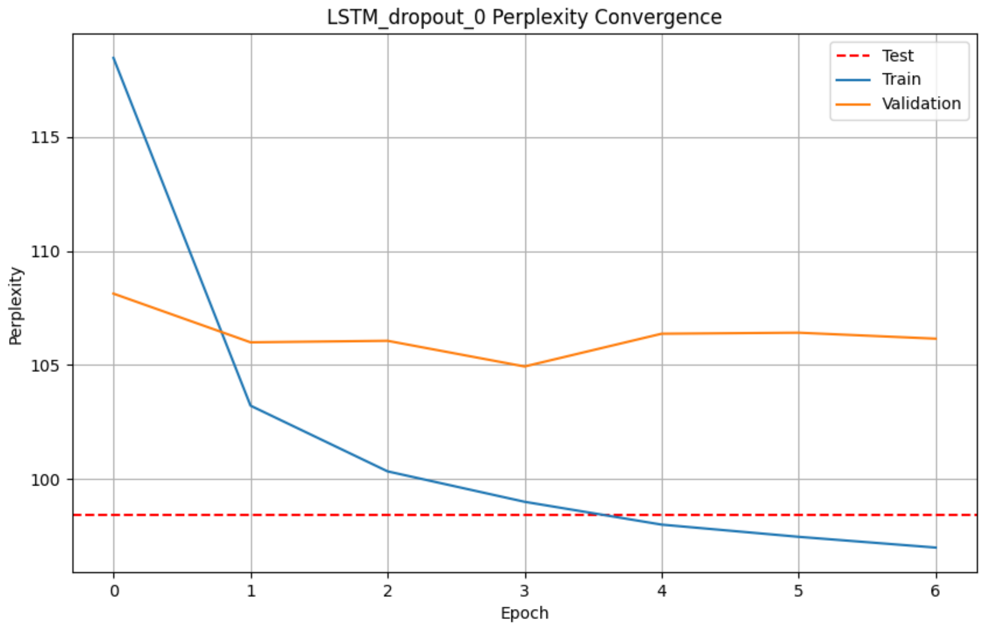
e. $\frac{\partial \epsilon_{(2)}}{\partial W_{xr}} = \frac{\partial E}{\partial h} \cdot \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial r} \cdot \frac{\partial r}{\partial W_{xr}} = \frac{\partial E}{\partial h}(1-z(2))(1-\tanh^2_{g_t})(W_{hg} \cdot h(1))(\sigma_{r_2}(1-\sigma_{r_2}) \cdot X(2))$

f. $\frac{\partial \epsilon_{(2)}}{\partial W_{hr}} = \frac{\partial E}{\partial h} \cdot \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial r} \cdot \frac{\partial r}{\partial W_{hr}} = \frac{\partial E}{\partial h}(1 - z(2))(1 - \tanh^2_{g_t})(W_{hg} \cdot h(1))(\sigma_{r_2}(1-\sigma_{r_2}) \cdot h(1))$

Practical Part

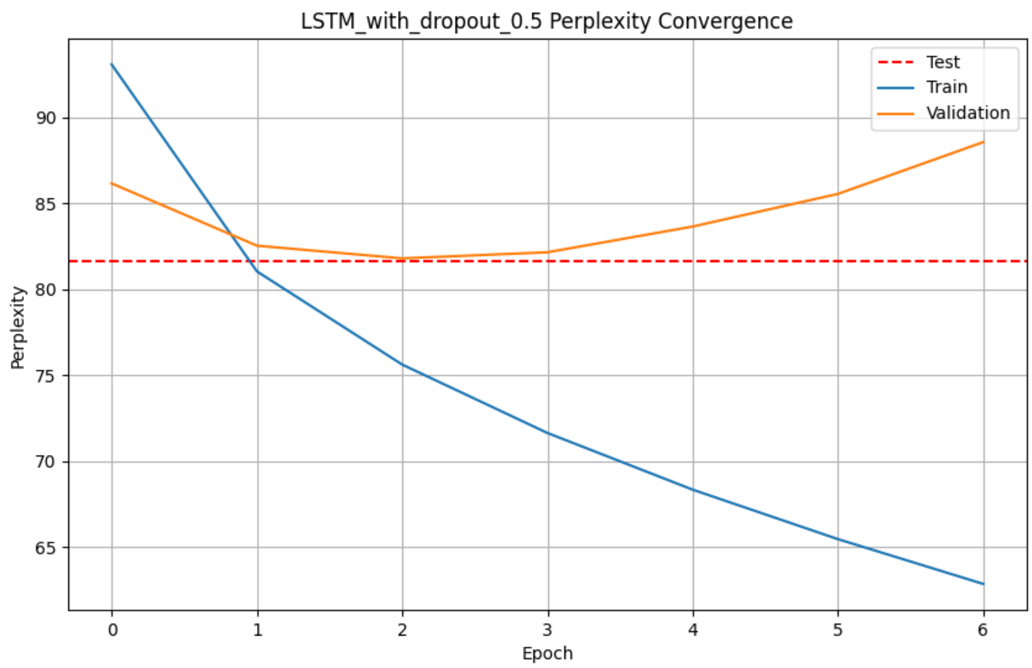
Our Results:

LSTM without Dropout:



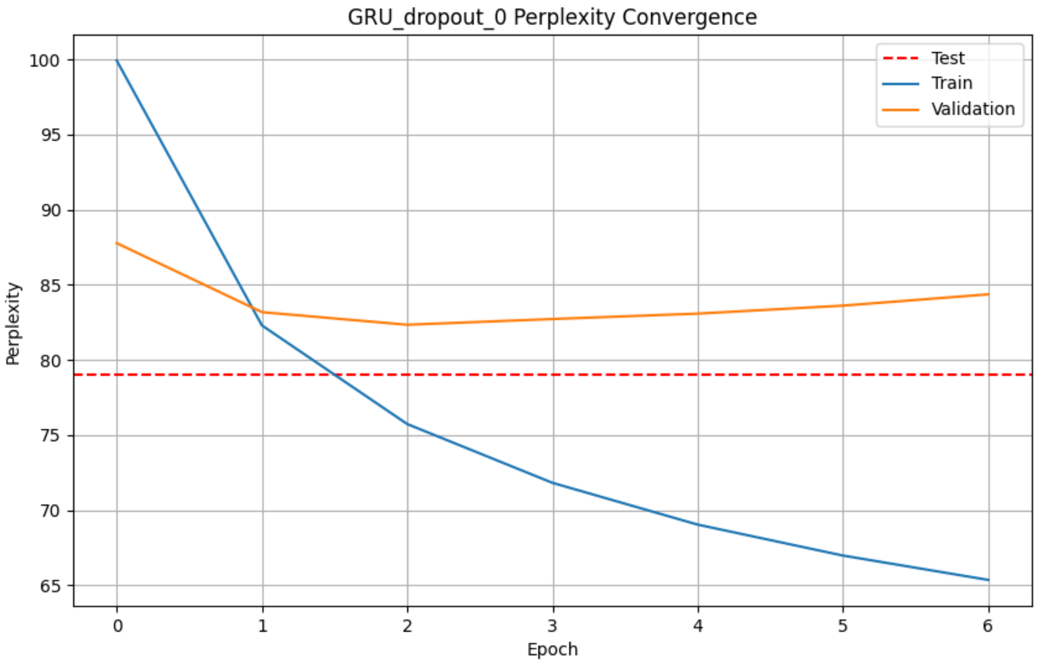
Training perplexity – 94.5464 , Validation perplexity – 106.0245

LSTM with Dropout (0.5):



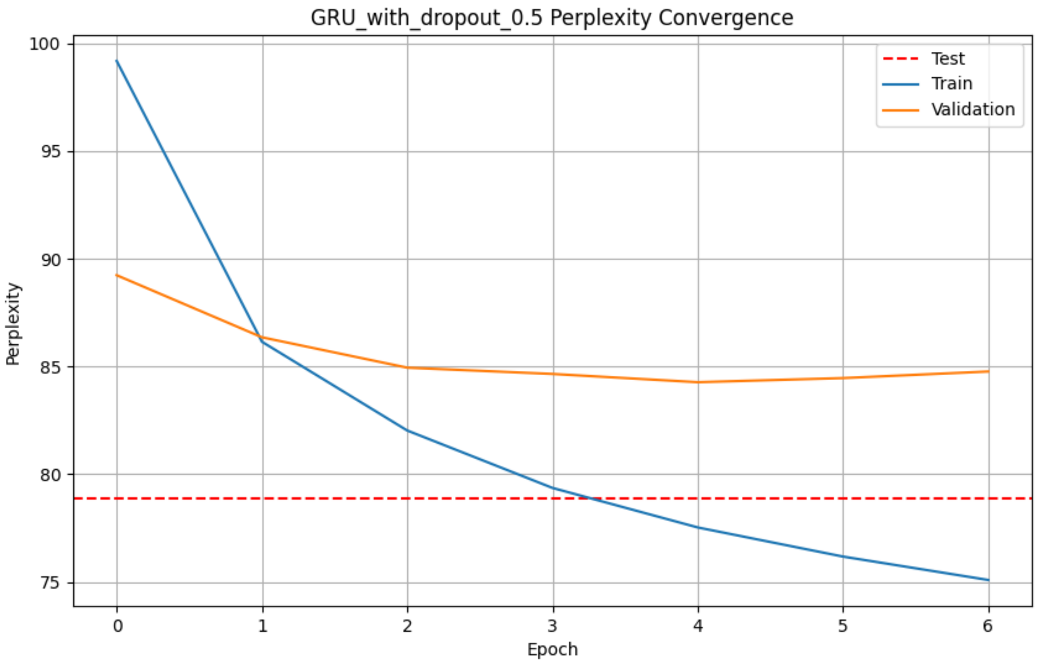
Training perplexity – 62.8612 , Validation perplexity – 88.5461

GRU without Dropout:



Training perplexity – 65.3495 , Validation perplexity – 84.3590

GRU with Dropout (0.5):



Training perplexity – 75.0906 , Validation perplexity – 84.7690

Results:

Model	Training perplexity	Validation perplexity	Test perplexity
LSTM without Dropout	102.5464	106.0245	96.8409
LSTM with Dropout (0.5)	62.8612	88.5461	81.6674
GRU without Dropout	65.3495	84.3590	79.6721
GRU with Dropout (0.5)	75.0906	84.7690	79.0265

Training Goals: All models achieved perplexities below the specified targets without dropout (<125) and with dropout (<100) as required.

Best Model: GRU with Dropout (0.5) achieved the best test perplexity of 79.0265, indicating it performed best on unseen data.

Impact of Dropout: The model based on LSTM shows significant improvement (train, valid, test) when dropout was used and matched the theory as we expected and learnt in class. Regarding the model based on GRU, we are observing the same or even worse performance in train and validation. In term of test we see slight improvement, which is expected given that dropout is a regularization technique that can help prevent overfitting, which can improve generalization performance on unseen data. We expected to see a similar behavior to the one showed for the LSTM (significant improvement). One possible explanation is that the learning rate we have used was static, and it might get stuck in a local minima.

Training vs Validation vs Test Perplexity: In all cases, training perplexity is the lowest, followed by validation perplexity, and then test perplexity. This is expected as the model is trained on the training data, validated on the validation data, and tested on the unseen test data.

README File:

Introduction

This code implements various configurations of recurrent neural networks (RNNs) for next-word prediction on the Penn Tree Bank dataset, aiming to achieve perplexities below the specified targets without dropout (<125) and with dropout (<100). It includes detailed explanations for training, testing, and interpreting the results, along with clear instructions for customization.

Dependencies

This work based on the following:

- PyTorch (torch)
- NumPy (numpy)
- Matplotlib (matplotlib.pyplot as plt)
- tqdm (optional, for progress bars)

Instructions

1. Download the Penn Tree Bank dataset:
   - Access the dataset from your course Moodle and extract it to the specified location (base_path).
   - Ensure that the extracted files (ptb.train.txt, ptb.valid.txt, ptb.test.txt) reside within the base_path directory.
2. Modify Hyperparameters:
   - Modify the script to experiment with different hyperparameters (learning rate, dropout, epochs) and architectures (LSTM/GRU, number of layers, units) to potentially improve performance.
3. Run the script:
   - Execute the script using Python: python next_word_prediction.py.
   - The script will train and evaluate each RNN configuration (LSTM/GRU with/without dropout) and generate convergence graphs and a summary table.

Code Breakdown

1. Imports and Constants:

Import relevant libraries, Define dataset path, Set hyperparameters.

2. Data Processing (PTBDataset class):

- Loads and preprocesses the Penn Tree Bank data, using:
  - **path:** data path
  - **seq_len:** sequence length
  - **vocab:** optional predefined vocabulary
- Tokenizes text and builds vocabulary.
- Converts tokens to indices.
- Creates batches of training, validation, and test data.

3. NextWordPredict Model:

Define the NextWordPredict Model structure, including:

- **vocab_size:** size of the dictionary of embeddings
- **embd_dim:** the size of each embedding vector
- **n_hidden:** Hidden layer dimension
- **n_layers:** Number of layers
- **dropout:** dropout value, '0' if no dropout
- **is_lstm:** TRUE if LSTM based, FALSE for GRU based

4. Training and Evaluation Functions:

- train: Performs training with early stopping based on validation perplexity.
- evaluate: Evaluates the model on a given dataset and calculates perplexity.

5. Experimentation Loop:

- Iterates through four configurations (LSTM/GRU with/without dropout).
- For each configuration, trains and evaluates the model, generating a convergence graph.

Code

Import Libraries

```
import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from collections import Counter
import os
from tqdm import tqdm

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

Model Structure

vocab_size: size of the dictionary of embeddings

embd_dim: the size of each embedding vector

n_hidden: Hidden layer dimension

n_layers: Number of layers

dropout: dropout value, '0' if no dropout

is_lstm: TRUE if LSTM based, FALSE for GRU based

```
class NextWordPredict(nn.Module):
      def __init__(self, vocab_size, embd_dim, n_hidden, n_layers, dropout=0, is_lstm=True):
        super().__init__()
        # save init values
        self.vocab_size = vocab_size
        self.embd_dim = embd_dim
        self.n_hidden = n_hidden
        self.n_layers = n_layers


        # define the model
        self.rnn = nn.LSTM(embd_dim, n_hidden, n_layers, dropout=dropout) if is_lstm else nn.GRU(embd_dim, n_hidden, n_layers, dropout=dropout) # Will be define by eac
        self.embd = nn.Embedding(vocab_size, embd_dim)
        self.drop = nn.Dropout(dropout) if dropout else None
        self.fc = nn.Linear(n_hidden, vocab_size)

      def forward(self, input):
        out = self.embd(input)
        out, hidden = self.rnn(out)
        # Be aware that the dropout is taking place inside the RNN!
        out = self.fc(out)
        return out
```

Load Data

path: data path

seq_len: sequence length

vocab: optional predefined vocabulary

```
class PTBDataset(Dataset):
    def __init__(self, path, seq_len, vocab=None):
        # Tokenize the text (simple space-based tokenization)
        with open(path) as f:
            text = f.read()
        self.tokens = text.split()
        self.seq_len = seq_len

        # Build vocabulary if not provided
        if vocab is None:
            self.vocab = self.build_vocab(self.tokens)
        else:
            self.vocab = vocab

        self.vocab_size = len(self.vocab)
        # Convert tokens to indices
        self.token_indices = [self.vocab.get(token, '<unk>') for token in self.tokens]

    def build_vocab(self, tokens):
        # Count the tokens and add <unk> for unknown tokens
        vocab = {'<unk>': 0}
        token_counts = Counter(tokens)
        token_counts.pop('<unk>', None)
        vocab.update({token: idx + 1 for idx, (token, _) in enumerate(token_counts.items())})
        return vocab

    def __len__(self):
        # Return the number of tokens (minus one since we predict the next token)
        return len(self.tokens) - (1 + self.seq_len)

    def __getitem__(self, idx):
        # Return the current token and the next token (as indices)
        current_token_idx = self.token_indices[idx:self.seq_len+idx]
        # next_token_idx = self.token_indices[idx+1:self.seq_len+idx+1]  if want labels to be all the sequence with offset 1
        next_token_idx = self.token_indices[self.seq_len+idx+1]
        return torch.tensor(current_token_idx), torch.nn.functional.one_hot(torch.tensor(next_token_idx), self.vocab_size).type(torch.float)
```

Data Location

```
base_path = '/content/drive/MyDrive/ex2_313135261_208678827/PTB/'
train_path = os.path.join(base_path, 'ptb.train.txt')
valid_path = os.path.join(base_path, 'ptb.valid.txt')
test_path = os.path.join(base_path, 'ptb.test.txt')
```

Functions Definition

```
# Save the current trained model
def save_model(model, save_path):
  torch.save(model.state_dict(),save_path, )

# Load Pre-trained model
def load_model(model, path):
  sd = torch.load(path)
  model.load_state_dict(sd)

# Define function to calculate perplexity
def calculate_perplexity(loss):
    return 2 ** loss

# Plot graph
def plot(train_perps, val_perps, test_perps, model_name):
      # Plot convergence graphs (modify as needed for multiple lines)
      plt.figure(figsize=(10, 6))
      plt.axhline(y = test_perps, color = 'r', linestyle = 'dashed', label="Test")
      plt.plot(train_perps, label="Train")
      plt.plot(val_perps, label="Validation")
      plt.xlabel("Epoch")
      plt.ylabel("Perplexity")
      plt.title(f"{model_name} Perplexity Convergence")
      plt.legend()
      plt.grid(True)
      plt.savefig(f"{model_name}_perplexity.png")
      plt.show()
      plt.close()
```

Model Parameters

Instructions:

- **LSTM or GRU:** in case of model training, set "is_lstm" to TRUE if the network is based on LSTM, and FALSE if it based on GRU.
- **Pre-trained model:** in case of using pre-trained model, set "load_pretrained" to TRUE, else FALSE
- **Dropout:** if using dropout - set "dropout" to the desired value, if not using dropout - set to '0'
- List item

```
# Set params
embd_dim = 128
n_hidden = 200
n_layers = 2
seq_len = 20
learning_rate = 0.001
dropout = 0
batch_size = 128
is_lstm = True

# Process Data
train_data = PTBDataset(train_path, seq_len)
valid_data = PTBDataset(valid_path, seq_len, train_data.vocab)
test_data = PTBDataset(test_path, seq_len, train_data.vocab)

train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
valid_dataloader = DataLoader(valid_data, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=False)

vocab_size = len(train_data.vocab)

# Set the model
model = NextWordPredict(vocab_size, embd_dim, n_hidden, n_layers, dropout=dropout, is_lstm=is_lstm)

# Set criterion
criterion = nn.CrossEntropyLoss()

# Check if GPU is available or not
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")
```

```
Using device: cuda
```

Training Structure

```
def train(model, epochs, device, train_dataloader, valid_dataloader, learning_rate):
    model.train()
    model.to(device)
    # Set optimizer
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    train_perplexity = []
    valid_perplexity = []
    test_perplexity = []
```

```
    test_perplexity = []
    epochs_perp = []
    val_perps = []
    losses = []
    for epoch in range(epochs):
      epoch_loss = []
      for inputs, targets in tqdm(train_dataloader) :
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        preds = model(inputs)
        preds = preds[:, -1, :]
        loss = criterion(preds, targets)
        loss.backward()
        # Add gradient clipping
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        epoch_loss.append(loss.item())
        losses.append(loss.item())

      # Calculate loss
      total_epoch_loss = np.mean(epoch_loss)
      train_epoch_perp = calculate_perplexity(total_epoch_loss)
      epochs_perp.append(train_epoch_perp)

      valid_perp = evaluate(model, valid_dataloader, device, criterion)
      val_perps.append(valid_perp)

      # Print and store epoch results
      print(f"Epoch {epoch+1}: Train Perplexity = {train_epoch_perp:.4f}, Val Perplexity = {valid_perp:.4f}")
    return epochs_perp, val_perps
```

Evaluate Structure

```
def evaluate(model, dataloader, device, criterion):
    model.eval()
    loss = []
    perplexity = []
    for inputs, targets in tqdm(dataloader):
      # turn off gradients
      with torch.no_grad():
        # set model to evaluation mode
        inputs, targets = inputs.to(device), targets.to(device)
        preds = model(inputs)
        preds = preds[:,-1,:]
        loss.append(criterion(preds, targets).item())

    model.train()

    # calculate perplexity
    total_loss = np.mean(loss)

    # return total_loss
    return calculate_perplexity(total_loss)
```

Training the model

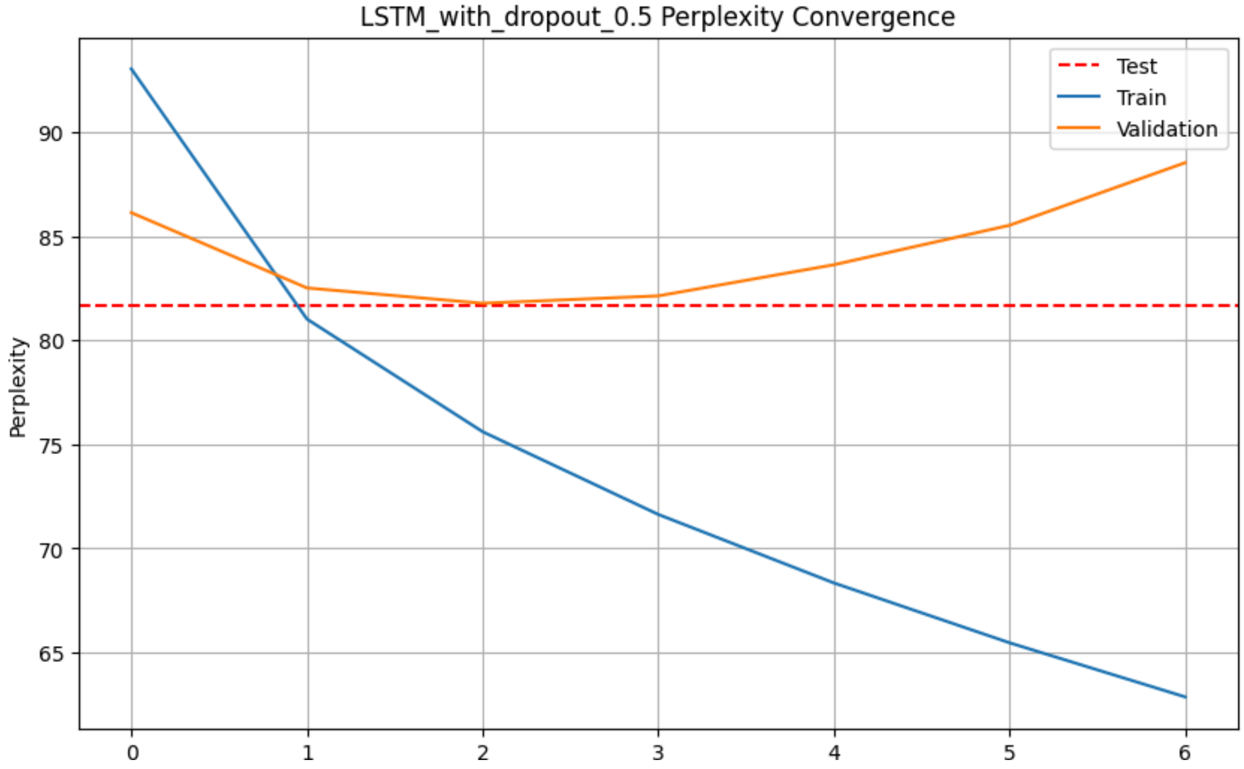
LSTM with Dropout - Training

```
# Define model
LSTM_with_dropout_model = NextWordPredict(vocab_size, embd_dim, n_hidden, n_layers, dropout=0.5, is_lstm=True)
LSTM_with_dropout_train_perps, LSTM_with_dropout_val_perps = train(LSTM_with_dropout_model, 7, device, train_dataloader, valid_dataloader, learning_rate)

model_name = 'LSTM_with_dropout_0.5'
saved_models_path = '/content/drive/MyDrive/ex2_313135261_208678827/'
save_model(LSTM_with_dropout_model, os.path.join(saved_models_path,f'model_{model_name}.pt'))

# Test and plot
test_perps = evaluate(LSTM_with_dropout_model, test_dataloader, device, criterion)
plot(LSTM_with_dropout_train_perps, LSTM_with_dropout_val_perps, test_perps, model_name)
```

```
100%|██████████| 6934/6934 [03:32<00:00, 32.65it/s]
100%|██████████| 550/550 [00:08<00:00, 63.24it/s]
Epoch 1: Train Perplexity = 93.0592, Val Perplexity = 86.1445
100%|██████████| 6934/6934 [03:32<00:00, 32.64it/s]
100%|██████████| 550/550 [00:08<00:00, 64.06it/s]
Epoch 2: Train Perplexity = 81.0299, Val Perplexity = 82.5250
100%|██████████| 6934/6934 [03:32<00:00, 32.70it/s]
100%|██████████| 550/550 [00:08<00:00, 63.60it/s]
Epoch 3: Train Perplexity = 75.6133, Val Perplexity = 81.7869
100%|██████████| 6934/6934 [03:32<00:00, 32.60it/s]
100%|██████████| 550/550 [00:08<00:00, 63.41it/s]
Epoch 4: Train Perplexity = 71.6358, Val Perplexity = 82.1425
100%|██████████| 6934/6934 [03:32<00:00, 32.69it/s]
100%|██████████| 550/550 [00:08<00:00, 64.37it/s]
Epoch 5: Train Perplexity = 68.3406, Val Perplexity = 83.6391
100%|██████████| 6934/6934 [03:32<00:00, 32.70it/s]
100%|██████████| 550/550 [00:08<00:00, 64.16it/s]
Epoch 6: Train Perplexity = 65.4640, Val Perplexity = 85.5368
100%|██████████| 6934/6934 [03:31<00:00, 32.79it/s]
100%|██████████| 550/550 [00:08<00:00, 64.63it/s]
Epoch 7: Train Perplexity = 62.8612, Val Perplexity = 88.5461
100%|██████████| 615/615 [00:09<00:00, 64.59it/s]
```



Epoch

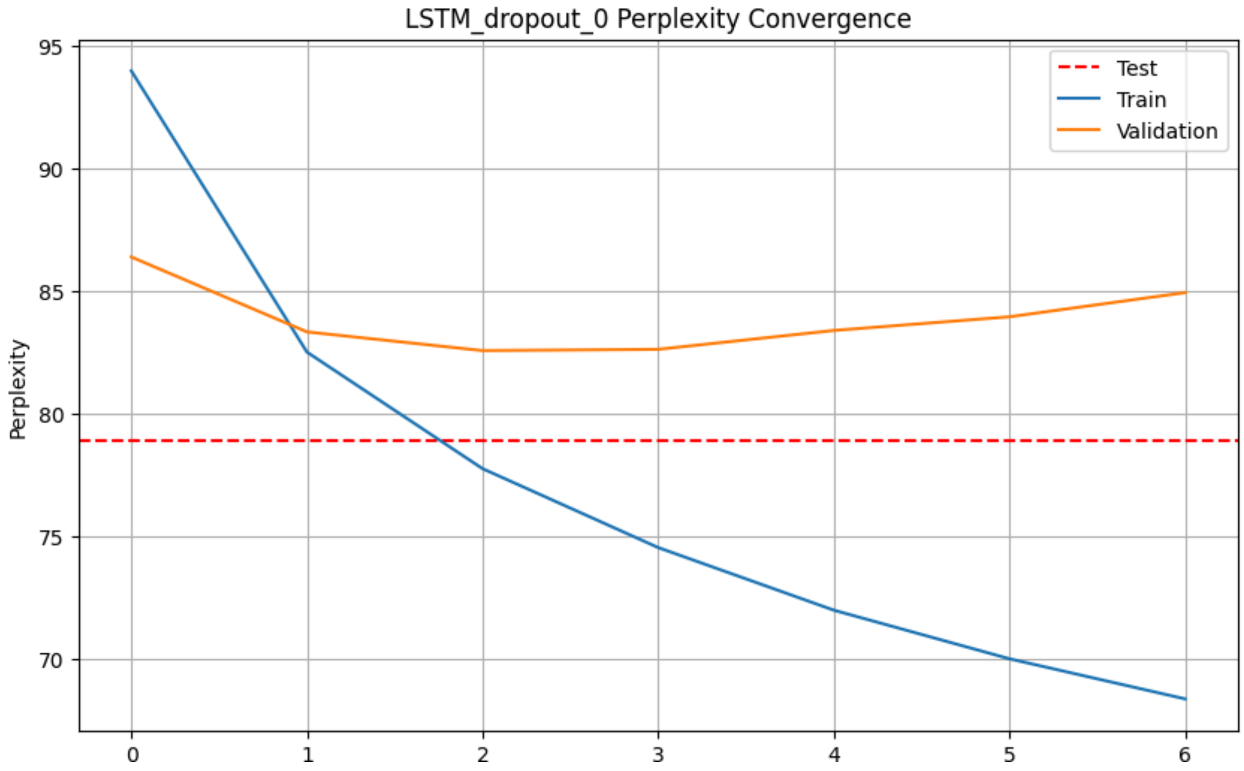
LSTM without Dropout - Training

```
# Define model
LSTM_no_dropout_model = NextWordPredict(vocab_size, embd_dim, n_hidden, n_layers, dropout=0, is_lstm=True)
LSTM_no_dropout_train_perps, LSTM_no_dropout_val_perps = train(LSTM_no_dropout_model, 7, device, train_dataloader, valid_dataloader, learning_rate)

model_name = 'LSTM_dropout_0'
saved_models_path = '/content/drive/MyDrive/ex2_313135261_208678827/'
save_model(LSTM_no_dropout_model, os.path.join(saved_models_path,f'model_{model_name}.pt'))

# Test and plot
test_perps = evaluate(LSTM_no_dropout_model, test_dataloader, device, criterion)
plot(LSTM_no_dropout_train_perps, LSTM_no_dropout_val_perps, test_perps, model_name)
```

```
100%|██████████| 6934/6934 [03:35<00:00, 32.25it/s]
100%|██████████| 550/550 [00:08<00:00, 65.02it/s]
Epoch 1: Train Perplexity = 93.9527, Val Perplexity = 86.3682
100%|██████████| 6934/6934 [03:35<00:00, 32.22it/s]
100%|██████████| 550/550 [00:08<00:00, 64.77it/s]
Epoch 2: Train Perplexity = 82.4931, Val Perplexity = 83.3152
100%|██████████| 6934/6934 [03:35<00:00, 32.12it/s]
100%|██████████| 550/550 [00:08<00:00, 63.99it/s]
Epoch 3: Train Perplexity = 77.7425, Val Perplexity = 82.5534
100%|██████████| 6934/6934 [03:37<00:00, 31.90it/s]
100%|██████████| 550/550 [00:08<00:00, 63.16it/s]
Epoch 4: Train Perplexity = 74.5266, Val Perplexity = 82.6056
100%|██████████| 6934/6934 [03:36<00:00, 32.05it/s]
100%|██████████| 550/550 [00:08<00:00, 65.41it/s]
Epoch 5: Train Perplexity = 71.9754, Val Perplexity = 83.3740
100%|██████████| 6934/6934 [03:35<00:00, 32.14it/s]
100%|██████████| 550/550 [00:08<00:00, 64.04it/s]
Epoch 6: Train Perplexity = 69.9950, Val Perplexity = 83.9316
100%|██████████| 6934/6934 [03:35<00:00, 32.13it/s]
100%|██████████| 550/550 [00:08<00:00, 64.64it/s]
Epoch 7: Train Perplexity = 68.3605, Val Perplexity = 84.9166
100%|██████████| 615/615 [00:09<00:00, 64.89it/s]
```



Epoch

GRU without Dropout - Training

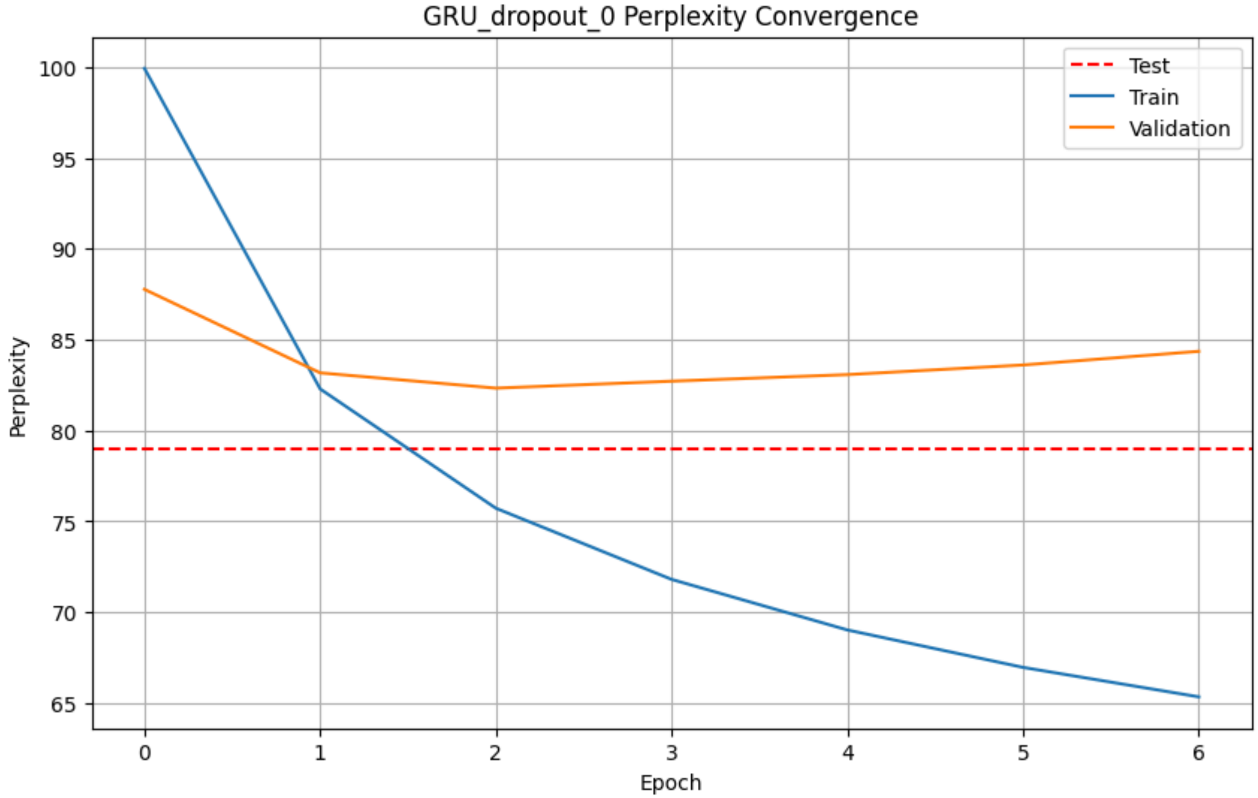
```
GRU_no_dropout_model = NextWordPredict(vocab_size, embd_dim, n_hidden, n_layers, dropout=0, is_lstm=False)
GRU_no_dropout_train_perps, GRU_no_dropout_val_perps = train(GRU_no_dropout_model, 7, device, train_dataloader, valid_dataloader, learning_rate)
lstm_or_gru = 'LSTM' if is_lstm else 'GRU'
model_name = f'{lstm_or_gru}_dropout_{dropout}'
saved_models_path = '/content/drive/MyDrive/ex2_313135261_208678827/'
save_model(model, os.path.join(saved_models_path,f'model_{model_name}.pt'))
test_perps = evaluate(GRU_no_dropout_model, test_dataloader, device, criterion)
plot(GRU_no_dropout_train_perps, GRU_no_dropout_val_perps, test_perps, model_name)
```

```
100%|██████████| 6934/6934 [03:30<00:00, 32.91it/s]
100%|██████████| 550/550 [00:08<00:00, 65.70it/s]
Epoch 1: Train Perplexity = 99.9244, Val Perplexity = 87.7678
100%|██████████| 6934/6934 [03:30<00:00, 32.89it/s]
100%|██████████| 550/550 [00:08<00:00, 65.51it/s]
Epoch 2: Train Perplexity = 82.2895, Val Perplexity = 83.1739
100%|██████████| 6934/6934 [03:30<00:00, 32.94it/s]
100%|██████████| 550/550 [00:08<00:00, 64.68it/s]
Epoch 3: Train Perplexity = 75.7257, Val Perplexity = 82.3335
100%|██████████| 6934/6934 [03:30<00:00, 32.97it/s]
100%|██████████| 550/550 [00:08<00:00, 65.66it/s]
Epoch 4: Train Perplexity = 71.8116, Val Perplexity = 82.7151
100%|██████████| 6934/6934 [03:30<00:00, 33.01it/s]
100%|██████████| 550/550 [00:08<00:00, 64.70it/s]
Epoch 5: Train Perplexity = 69.0297, Val Perplexity = 83.0762
100%|██████████| 6934/6934 [03:30<00:00, 33.00it/s]
100%|██████████| 550/550 [00:08<00:00, 65.41it/s]
Epoch 6: Train Perplexity = 66.9703, Val Perplexity = 83.6068
100%|██████████| 6934/6934 [03:30<00:00, 32.99it/s]
100%|██████████| 550/550 [00:08<00:00, 64.94it/s]
Epoch 7: Train Perplexity = 65.3495, Val Perplexity = 84.3590
100%|██████████| 615/615 [00:09<00:00, 65.18it/s]
```

Epoch

```
model_name = 'GRU_dropout_0'
saved_models_path = '/content/drive/MyDrive/ex2_313135261_208678827/'
save_model(model, os.path.join(saved_models_path,f'model_{model_name}.pt'))
test_perps = evaluate(GRU_no_dropout_model, test_dataloader, device, criterion)
plot(GRU_no_dropout_train_perps, GRU_no_dropout_val_perps, test_perps, model_name)
```

```
100%|██████████| 615/615 [00:09<00:00, 65.34it/s]
```



GRU with Dropout - Training