CS7.401: Introduction to NLP | Assignment 3

Author: Aditya Kumar Singh
ID: 2021701010

April 11, 2022

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Contents

Results

Overview

NNLM (Neural Network Language Model) is a basic language model with neural backbone that tries to predict the word (or it's corresponding index indirectly from a dictionary or vocabulary) given some fixed length of context words (in form of list of indices).

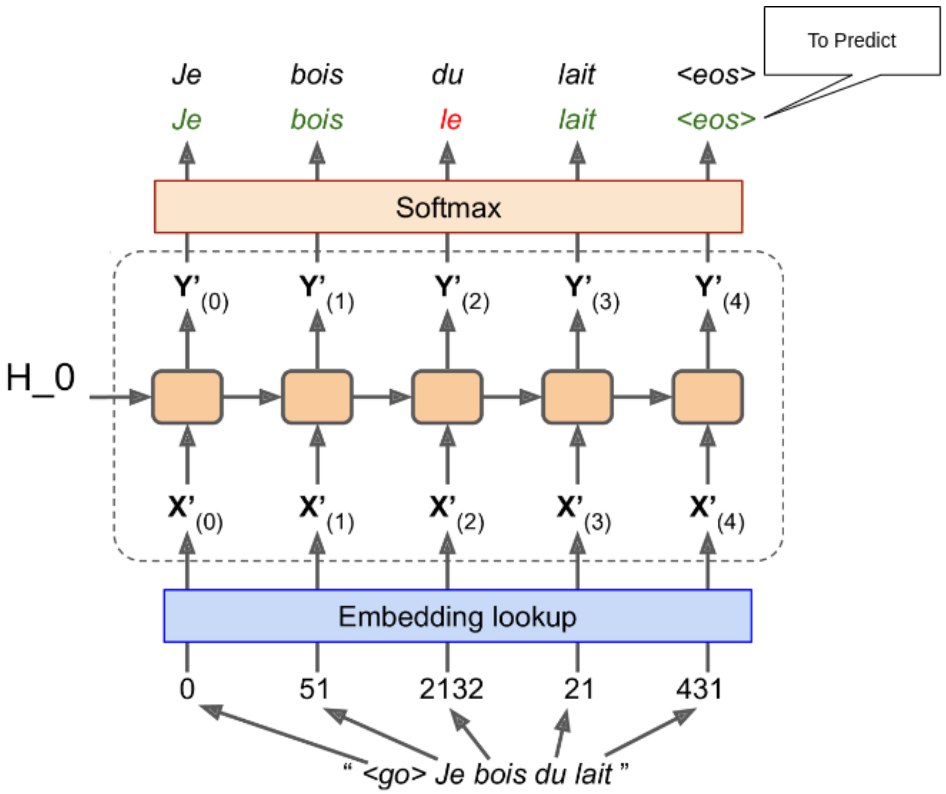


Figure 1: A basic sketch of NNLM: In our case the recurrent unit is **GRU** with *bidirectional* = *False* and *num_layers* = 2 i.e., no. of stacks of GRU. *Embedding dimension* = 512; *Hidden Dimension* = 256; *Output layer* = Linear layer of size = vocab_size, followed by *softmax*

Perplexity scores for NNLM

- Average PP score for English NNLM (train): 74.331
- Average PP score for English NNLM (test): 250.411

BLEU scores for MT models

- Corpus-level BLEU score for MT1 model (train data): 19.082
- Corpus-level BLEU score for MT1 model (test data): 19.582
- Corpus-level BLEU score for MT2 model (train data): 3.747
- Corpus-level BLEU score for MT2 model (test data): 3.125

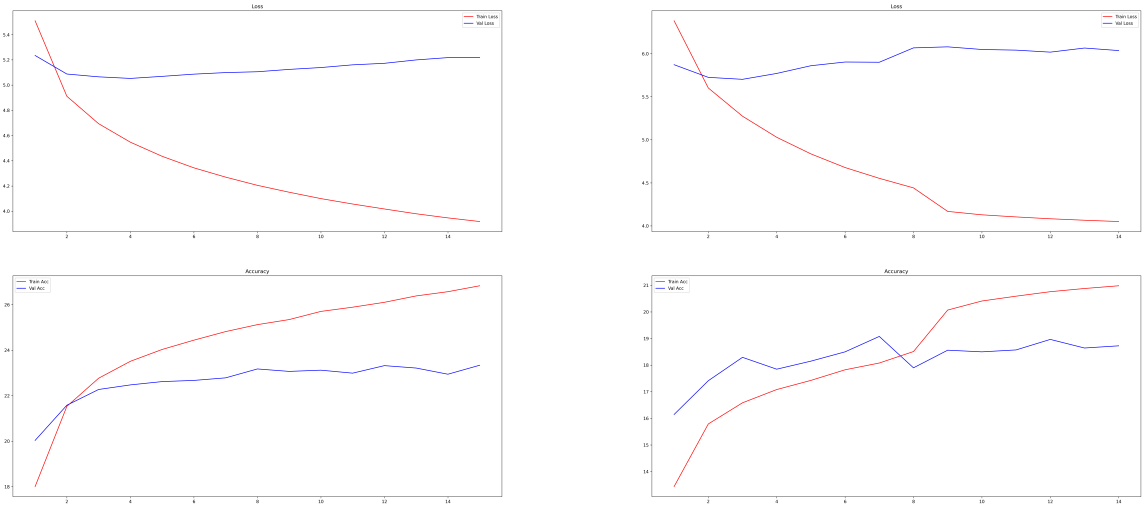
Analysis

The given tasks were computationally (GPU intensive) heavy and has put a limit to our exploration within hyper-parameter space.

Let's have a model-by-model case study and later we'll compare them in terms of their performance. (*Note* that we used `spacy` tokenizer and `build_vocab` function from `torchtext` library to construct vocabulary.)

NNLM

- For both *English* and *French* we managed to accommodate all the data, "Euoparl" for English (the English version of the Europarl v7 dataset) and "News" (the News Crawl 2013 corpus for French language) data for French, and tried to train our models, wherein some of them shows saturation behaviour in terms of learning.
- Both training and validation curve goes flat after, approximately, 10-15 epochs for both LM.
- Even though French vocabulary is quite richer (containing almost $39,003$ words) than one for the English (containing about $16,062$ words), in terms of performance (as shown in learning curve below) its not that promising.



(a) English NNLM curve

(b) French NNLM curve

Figure 2: Learning Curve

The step-wise pattern in learning curve is due to the `lr_scheduler` which in our case is `ReduceLROnPlateau()` that reduces learning rate when a metric stops improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'PATIENCE' number of epochs, the learning rate is reduced.

- The fact that both the model stagnates at a particular loss is mainly due to the "simplistic" architecture (with one two stacks of **GRU** only and no **Linear or Dense** layer) as well as less amount of data. And hence the model is not able to learn (i.e., overfit) perfectly on training data.
- In terms of model, both of them have the same architecture as shown in summary form below.

```
NNLM(
  (embedding): Embedding(16062, 512)
  (gru): GRU(512, 256, num_layers=2, batch_first=True)
```

```
  (drop_layer): Dropout(p=0, inplace=False)
  (out1): Linear(in_features=256, out_features=16062, bias=True)
  (log_softmax): LogSoftmax(dim=1)
  )
```

- Below mentioned are the hyperparameters used to train both the models.

```
Language = FRENCH
Batch Size = 256
n (no. of grams) = 5
Learning Rate = 0.001
Patience to Early Stop the Model = 10
Vocab Size = 39003
Embedding Dimension = 512
Encoding Hidden Dimension = 256
No. of Stacked Layers in GRU = 2
Dropout = 0.5
Early Stopping happened at Step = 14
Number of Epochs = 100
```

```
Language = ENGLISH
Batch Size = 256
n (no. of grams) = 5
Learning Rate = 0.001
Patience to Early Stop the Model = 10
Vocab Size = 16062
Embedding Dimension = 512
Encoding Hidden Dimension = 256
No. of Stacked Layers in GRU = 2
Dropout = 0.5
Early Stopping happened at Step = 15
Number of Epochs = 100
```

MT1

- For MT (machine translation) task, we have been provided with "ted-talks" corpus (the English-French translation task in IWSLT 2016) to train, validate and test our model.
- Let's view the summary of the architecture used (*note that the order of "dropout" layer shown is not always correct*):

1. ENCODER

```
Encoder(
  (embedding): Embedding(55000, 128)
  (rnn): GRU(128, 256, bidirectional=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=512, out_features=256, bias=True)
)
```

2. ATTENTION

```
Attention(
  (attn): Linear(in_features=768, out_features=32, bias=True)
)
```

3. DECODER

```
Decoder(
  (attention): Attention(
    (attn): Linear(in_features=768, out_features=32, bias=True)
  )
  (embedding): Embedding(32000, 128)
  (rnn): GRU(640, 256)
  (dropout): Dropout(p=0.5, inplace=False)
  (out): Linear(in_features=896, out_features=32000, bias=True)
)
```

4. Seq2Seq (the combination of all of them)

```
Seq2Seq(
  (encoder): Encoder(
    (embedding): Embedding(55000, 128)
    (rnn): GRU(128, 256, bidirectional=True)
    (fc): Linear(in_features=512, out_features=256, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (decoder): Decoder(
    (attention): Attention(
      (attn): Linear(in_features=768, out_features=32, bias=True)
    )
    (embedding): Embedding(32000, 128)
    (rnn): GRU(640, 256)
    (dropout): Dropout(p=0.5, inplace=False)
    (out): Linear(in_features=896, out_features=32000, bias=True)
  )
)
```

5. Parameters used for MT1

```
Batch Size = 8
Learning Rate = 0.001
Patience to Early Stop the Model = 6
ENCODER Vocab Size = 25231
DECODER Vocab Size = 31890
ENCODER Embedding Dimension = 32
DECODER Embedding Dimension = 32
ENCODER Encoding Hidden Dimension = 64
DECODER Encoding Hidden Dimension = 64
Attention dimension = 8
ENCODER Dropout = 0.5
DECODER Dropout = 0.5
Early Stopping happened at Step = 21
Number of Epochs = 40
```

- Following are the learning curves for `seq2seq` model trained from scratch (**MT1**). The same reason (as seen in NNLM) also accounts here for inefficient learning for MT1. Though it has an "attention" module, its not able to show a significant improvement over NNLM. Hence this indicates we can improve upon the existing model and infer where it is going wrong and why can't it improve its learning curve.

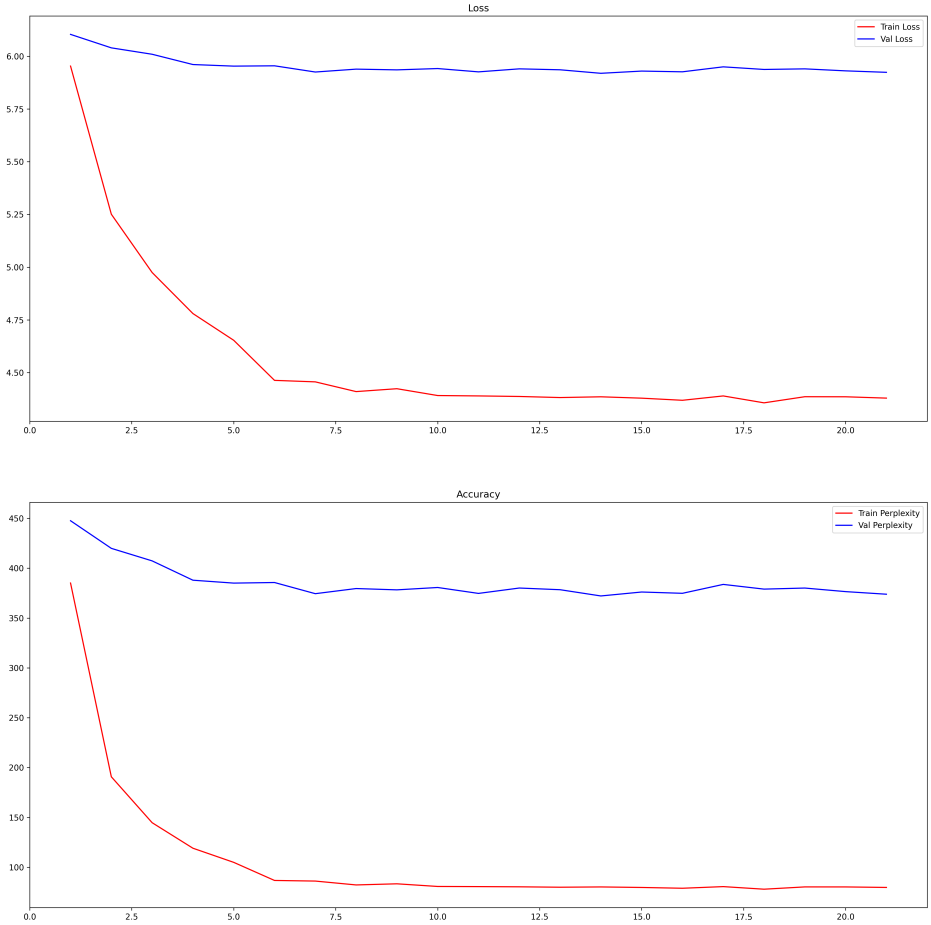


Figure 3: Cross-Entropy Loss Curve for MT1

MT2

- The task here was to bring in use the model trained in NNLM subsection (i.e., both French and English LM) for translation task (from English to French).
- This practice of Transfer Learning (using pre-trained models) where the main model has learnt language representation on general tasks has proved to be fruitful by gearing the model towards the desired downstream task. And this method, overall, known as Fine Tuning that requires significantly fewer data points than training a model from scratch for the same task.
- Below is the "performance" curve for the model:

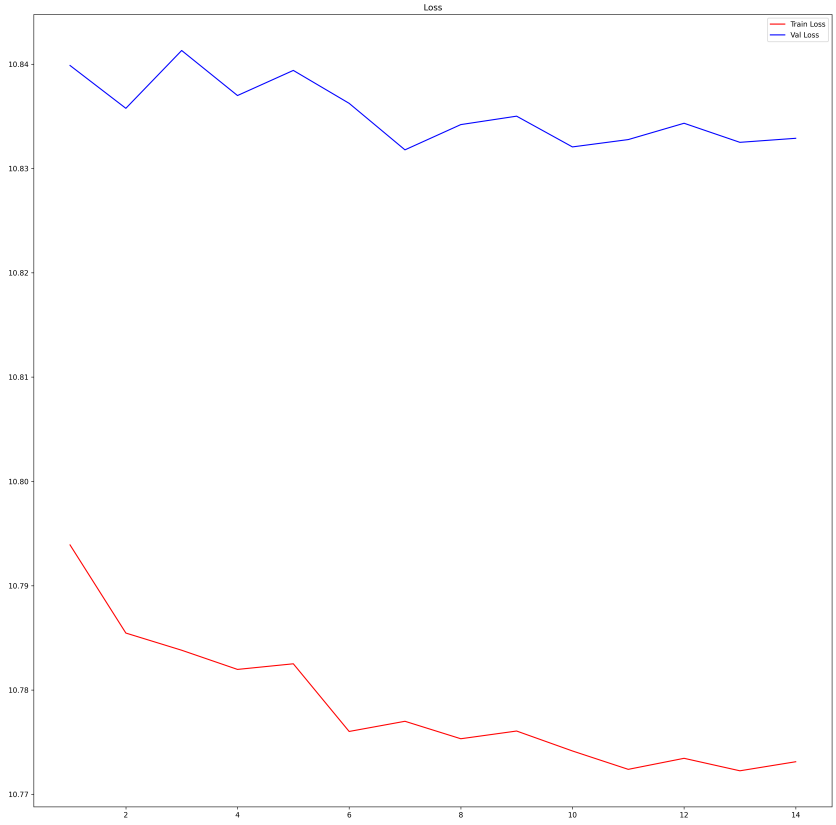


Figure 4: Cross-Entropy Loss curve for MT2

- Its architecture is pretty simple (a basic figure has been illustrated below) and hence it's performance. Again, since we're constrained by computational resources we tried to make model as simple as possible to let it run on our system. Observing the above graph we can infer that the learning is happening at a very slow pace. With over *70,000,000* parameters (i.e., almost between 7-8 crores parameters) the learning becomes computationally heavy and temporally slow. Hence for an epic-level performance we not only need to have heavy architecture but also heavy computational resources.

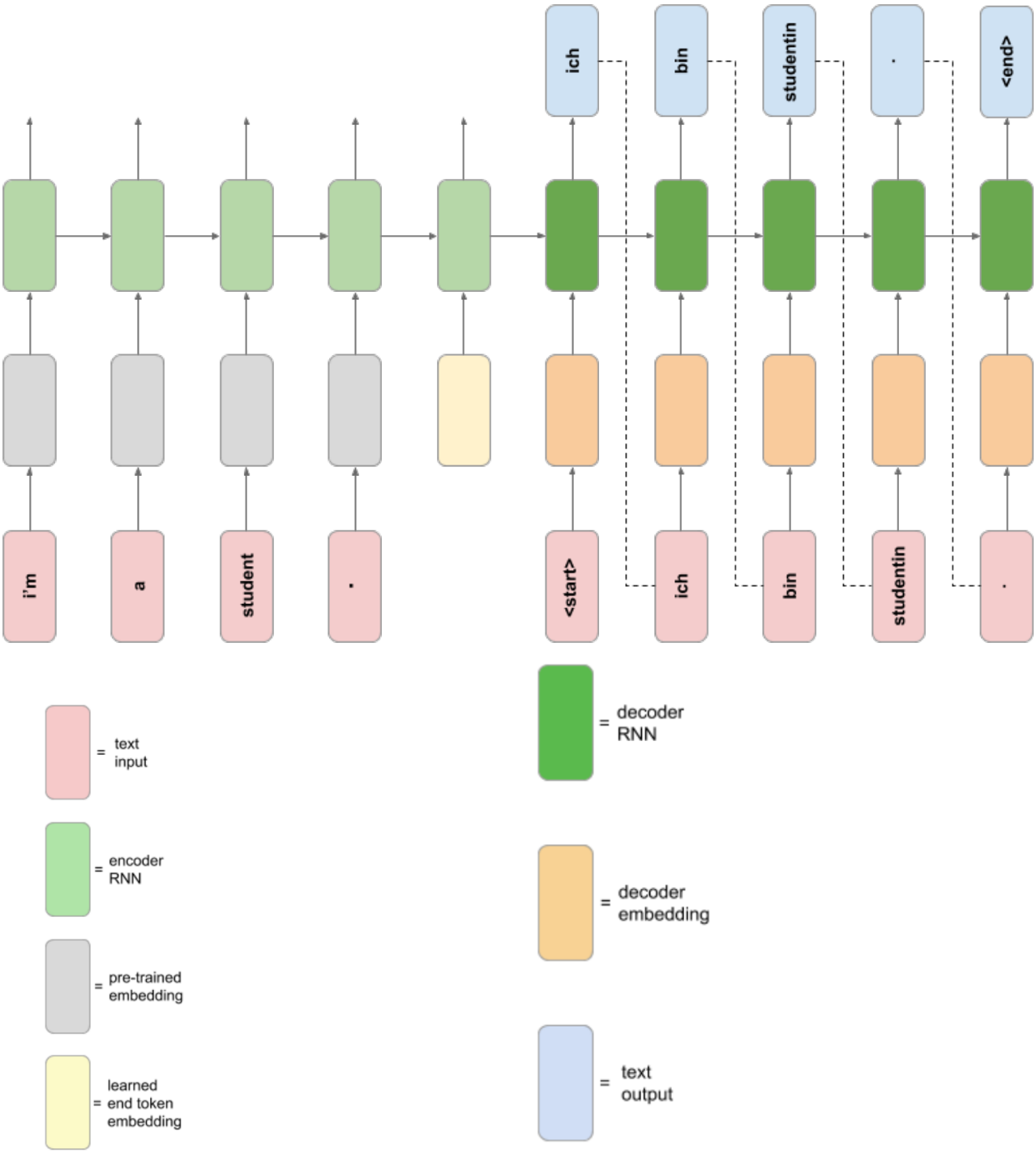


Figure 5: Seq2Seq Task with One Layer of Recurrent Unit (In our case it's 2 layer of units.)

- Below is the summary of model and hyper-parameters used for the same.

  1. ENCODER

```
encoder(
  (enc): NNLM(
    (embedding): Embedding(32085, 512)
    (gru): GRU(512, 256, num_layers=2, batch_first=True)
    (drop_layer): Dropout(p=0, inplace=False)
    (out1): Linear(in_features=256, out_features=32085, bias=True)
    (log_softmax): LogSoftmax(dim=1)
  )
  (drop_layer): Dropout(p=0, inplace=False)
)
```

2. DECODER

```
decoder(
  (dec): NNLM(
    (embedding): Embedding(32085, 512)
    (gru): GRU(512, 256, num_layers=2, batch_first=True)
    (drop_layer): Dropout(p=0, inplace=False)
    (out1): Linear(in_features=256, out_features=32085, bias=True)
    (log_softmax): LogSoftmax(dim=1)
  )
  (out2): Linear(in_features=256, out_features=55104, bias=True)
  (softmax): Softmax(dim=1)
  (drop_layer): Dropout(p=0, inplace=False)
)
```

3. Seq2Seq (the combination of all of them)

```
Seq2Seq_MT2(
  (encoder): encoder(
    (enc): NNLM(
      (embedding): Embedding(32085, 512)
      (gru): GRU(512, 256, num_layers=2, batch_first=True)
      (drop_layer): Dropout(p=0, inplace=False)
      (out1): Linear(in_features=256, out_features=32085, bias=True)
      (log_softmax): LogSoftmax(dim=1)
    )
    (drop_layer): Dropout(p=0, inplace=False)
  )
  (decoder): decoder(
    (dec): NNLM(
      (embedding): Embedding(32085, 512)
      (gru): GRU(512, 256, num_layers=2, batch_first=True)
      (drop_layer): Dropout(p=0, inplace=False)
      (out1): Linear(in_features=256, out_features=32085, bias=True)
      (log_softmax): LogSoftmax(dim=1)
    )
    (out2): Linear(in_features=256, out_features=55104, bias=True)
    (softmax): Softmax(dim=1)
    (drop_layer): Dropout(p=0, inplace=False)
  )
)
```

- Parameters used for MT2

```
Batch Size = 6
Learning Rate = 0.001
Patience to Early Stop the Model = 6
ENCODER Vocab Size = 32085
DECODER Vocab Size = 55104
ENCODER Embedding Dimension = 512
DECODER Embedding Dimension = 512
ENCODER Encoding Hidden Dimension = 256
DECODER Encoding Hidden Dimension = 256
ENCODER Dropout = 0.5
DECODER Dropout = 0.5
Early Stopping happened at Step = 14
Number of Epochs = 50
```

Overall Say

- With regards to NNLM, seeing the training and test scores one can infer that a heavy overfitting has hapenned, which is apparent from the learning curve. *Data Augmentation* is surely a way out from the overfitting which we've kept for future try outs. Even after 50% `dropout` in between `embedding` layer and `GRU` layer, we unable to prevent overfitting.
- In case of MT1, since `attention` is used we're very much sure to get better results than MT2. While in case of MT2 we are constrained to use simple pretrained models with no attention.
- In terms of *improvement* if we, for the time being, put away the GRU part of pre-trained decoder, and train a new fresh GRU in decoder with same architecture as MT1, then we believe to have a better performing translation model than MT1.

REMARKS: To run the code, please refer to the `Readme.md` file.