

MANDATORY ASSIGNMENT 2 - IN4310

DANIEL FREMMING

1. INTRODUCTION/CODE SETUP

The necessary code can be found in the as listed:

- Task 1: `train1.sh` for loading right modules and run the training-script.
 - `train.py`
 - `out1.log.log`
 - Save checkpoints before running training on task 2
create a separate copy of `model.py` with a new filename,
and a copy of `train.py` with a new filename which imports the
- Task 2: Instructions on saving a separate copy of `model.py` with a new filename, (`model_task2.py`) and a copy of `train.py` with a new file `train_task2.py`
 - `train_task2.py`
 - `model_task2.py`
 - `out1.log`
- Task 3: Same procedure as in Task 2
 - `train_task3.py`
 - `model_task3.py`
 - `out1.log`
 - `save_model.pth` - pretrained model
 - 5 example images with predicted captions using the `generate_captioned_images_top_bottom`-method for the attention model.

Additionally, `plot.py` was used to access `out1.log` to visualize the loss and metric plots for each task since it was not possible to download the slurm module on the ML-nodes.

2. IMPLEMENTATION DETAILS

For this mandatory assignment I used the UiO's ML-nodes, with setting up a remote Python interpreter in VScode to be necessary. Trying to run the training on my Lenovo Yoga laptop proved to be a difficult task as it froze, rendering it unusable(*the training schedule was probably using up the RAM*). This was different from mandatory 1 where I ran the training schedule on locally on the very same hardware.

I followed the step-by-step in section 3.1. Here I created a bash file using `vim` and modified the file accordingly. This file is attached as `train1.sh`. For easy file upload and download, I used the [UiO's ML node guide](#), and later combined the task for file-transferring with WinSCP - a graphic user-face. I used WinSCP interchangeably

with the remote interpreter on my laptop. I ran into problems with the import of the torch library. Import this module did not help. What helped was switching the order of importing modules in the script(`train1.sh`):

From:

```
1 module load PyTorch-bundle/1.10.0-MKL-bundle-pre-optimised
2 module load matplotlib/3.4.3-foss-2021b
```

To:

```
1 module load matplotlib/3.4.3-foss-2021b
2 module load PyTorch-bundle/1.10.0-MKL-bundle-pre-optimised
```

. The `train1.sh`-script ws used for task 1-3 and `generate_and_capture.sh` was used to generate 5 example images with predicted captions obtained from the model used in task 3. Also see ->>

2.1. Justification and modified parameters.

For this mandatory assignment I modified following `config.py`-file:

```
1 # Optimisation
2     self.learning_rate = 1e-4
3     self.weight_decay = 1e-5
4     self.num_epochs = 10
5     self.batch_size = 16
```

The rest remained unchanged except for the brief mentions about training parameters in each task. The justification for this parameter-choice is to save time with training and works ideally when you are pressed on time. This way I reduced runtime for the training schedule from est. 20hours to maybe 2-4 hours using the nvidia RTX3090 m19 for task 1. For task 2 and task 3 these times were significantly longer. In addition, I spent days on setting up training on the UiO's ML nodes as this task posed a real challenge. I wish this part was better facilitated as I probably spent most of the my time setting up this. Also, I felt the latter deviates from this course's curriculum, while it gave me new knowledge about SSH tunneling and the use of ML infrastructure, I felt, at times, lost, even pondering that I will not submit mandatory 2.

3. TASK 1: BASIC RNN(VANILLA) IMPLEMENTATION


This task involved working on a basic vanilla RNN implementation for the image capture. The handed coded accommodates both LSTM cells and RNN cells in the same architecture, such that each hidden state tensor has the shape `[batch_size, 2 * self.hidden_state_size]`. As the vanilla RNN cell doesn't need the cell state, only the first half of the hidden state of size `hidden_state_size` is used:

```
h_old = state_old[:, : state_old.size(1) // 2].
```

3.1. Setting `self.num_layers=1`.

Task 1 forms the initial development of an image captioning model, therefore it was reasonable to set `self.num_layers=1` for simplicity. This way the complexity of managing different input dimensions between layers is reduced, which saves development time. Below the reader can see the modifications used in `config.py` for task 1.

```
1 self.use_attention = False
2 self.num_layers = 1
3 self.cell_type = 'RNN'
```

 Python

3.2. Inputs in RNN cells and layers.

The code displays how inputs are handled for the different layers. The vanilla RNN uses a single layer, but it could be nice to highlight how the inputs are processed for multiple layers. The provided code-snippet displays how:

```
input_sizes = []
for i in range(num_layers):
    if i == 0:
        input_sizes.append(embedding_size + hidden_state_size)
    else:
        input_sizes.append(hidden_state_size)
```

First layer (`i=0`): takes concatenated word embedding and the image features as a vector, adding them together so the resulting input size is `embedding_size + hidden_state_size`.

Subsequent layers (`i>0`), from layer 2 and onwards: receives the hidden state from previous layer as input, thus input size is just the `hidden_state_size`.

3.3. Input Size Dimensions.

For the single-layer RNN the `input_sizes` captures only the first entry, i.e. `embedding_size + hidden_state_size`. During the first time step, the start token embedding is concatenated with the image feature vector as we are not using attention. Subsequent time steps use either the ground truth token embeddings or the predicted token embeddings, where the latter is still concatenated with image features from the first time step.

3.4. Loss Graph.

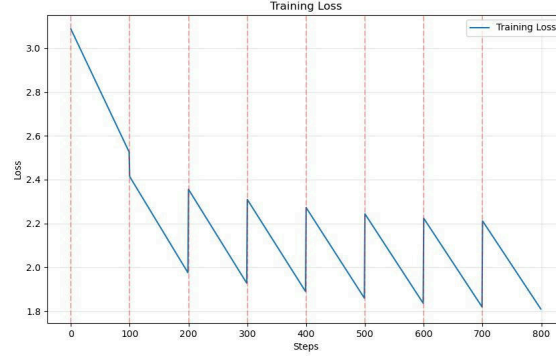


FIGURE 1. Training Loss curve for a vanilla RNN using a single layer.

Figure 1 shows a training loss curve with a sawtooth pattern after around step 200, which is a typical pattern in machine learning (<https://arxiv.org/abs/2410.10056>). A possible source for this “Epochal sawtooth Effect”(see link) is when evaluations are performed periodically. The reason number of steps are in the hundreds is because how the steps for the loss is calculated: steps for loss is based on number of epochs, where steps per epoch was set to 100. Looking at the curve in Figure 1 the loss starts high ~ 3.1 and decreases over time (to ~ 1.8). I observe the trend to be overall downward which suggest the model is learning effectively.

3.5. Validation Metrics (BLEU@4 and CIDEr scores).

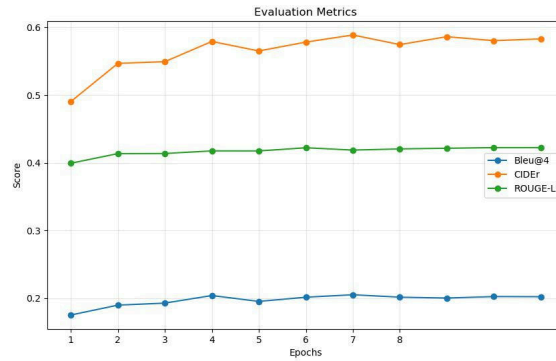


FIGURE 2. Evaluation metrics(Cider, Bleu,Rouge) for a single-layer vanilla RNN.

Figure 2 displays evaluation metrics across 11 epochs for three different natural language processing(NLP) evaluation metrics. I don’t know why the plot displays 11 epochs and stops at epoch 8, but it still executes its task and visualizes the curve beyond epoch 8. The evaluation metrics are read to be as follows:

- CIDEr (orange) shows the highest scores (0.49-0.59)
- ROUGE-L (green) shows moderate scores (0.40-0.42)
- BLEU@4 (blue) shows the lowest scores (0.17-0.20)

The trend for all metrics shows successive improvements across all epochs. Some notes on the metrics:

- CIDEr often yields higher scores than ROUGE-L.
- BLEU scores tend to be the lowest of the two.

Considering these remarks and the gradual improvement of the loss curve, this information suggests the model is learning appropriately.


Accessing the out1.log the best reported scores for the relevant metrics are listed below:

- Best CIDEr score in epoch 9: 0.5876004149962719
- Best BLEU score in epoch 5: 0.20525868801860056

4. TASK 2: REPLACING VANILLA RNN WITH A LSTM

In task 2 I turn my focus to LSTM, thus replacing the vanilla RNN cell type with a LSTM cell type, without attention. Attention isn't implemented before task 3. The parameters in `config.py` are set accordingly:

```
1 self.use_attention = False
2 self.num_layers = 2
3 self.cell_type = 'LSTM'
```

 Python

4.1. Layer Input Differences.

First layer (layer 0): Receives concatenated token embeddings and the raw image feature vector when used without attention. This can be examined by studying the 2 parameters accepted by the LSTMCell class in its forward method:

- `x`: Input tensor with shape `[batch_size, input_size]`
- `hidden_state`: Tensor with shape `[batch_size, 2 * hidden_state_size]`

For the first layer (layer 0): The `x` input is a concatenation of token embeddings and feature vectors, where `input_size = embedding_size + hidden_state_size`. We can see that this result matches the `input_sizes` list from Section 3.2 in the CaptionRNN initialization. For the second and subsequent layers: The `x` input is just the hidden state from the previous layer, so `input_size=hidden_state_size`, which also matches the result in Section 3.2.

4.2. Hidden State Shape.

The shape of the hidden state tensor when using the LSTM cell type is `[batch_size, 2 * hidden_state_size]`, where first half contains the actual hidden state: `[:, :hidden_state_size]`. Second half contains the memory cell state

`[:, hidden_state_size:]`, which is omitted for the RNN cell type. The output of the cell should return a tensor in the same format (*1st half, 2nd half*) containing the new hidden state and new cell state.

4.3. Loss Graph.

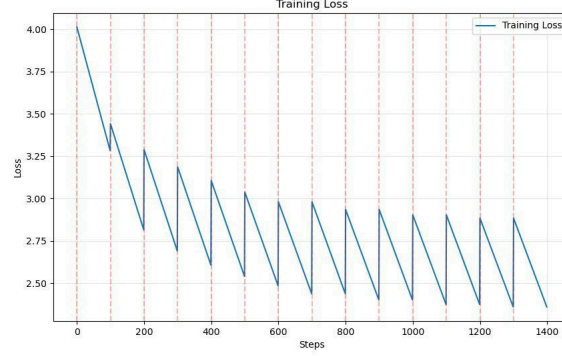


FIGURE 3. Training Loss curve for a double layer LSTM without attention.

Figure 3 displays the same decreasing sawtooth pattern in the loss curve as in RNN implementation. The loss-value decreases from 4.0 to 2.3, which is higher compared to Figure 1(3.1 – 1.8). We can observe that the lowest point of each sawtooth steadily decrease, suggesting the model is still improving and has not plateaued. A suggestion could be run the training for more epochs as this schedule was stopped after 10 epochs.

4.4. Validation metrics.

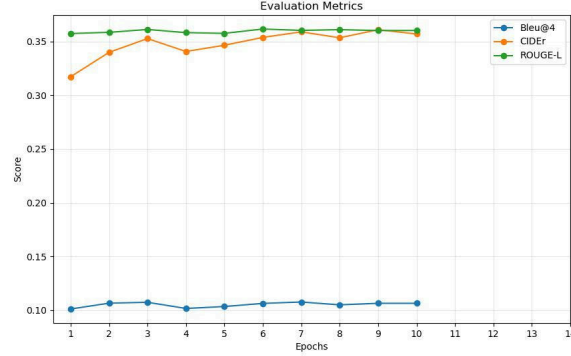


FIGURE 4. Evaluation metrics(Cider, Bleu,Rouge) for a double layer LSTM without attention.

From Figure 4 we observe evaluation metric to be:

- CIDEr (orange) shows the highest scores (0.31-0.35)
- ROUGE-L (green) shows moderate scores (0.35-0.36)
- BLEU@4 (blue) shows the lowest scores (0.10-0.11)

The ROUGE-L metric (green line) performs best at a relatively steady level. On the other hand, CIDEr (orange) is lower than ROUGE-L across epochs, which is

different from what can be seen in Figure 2. This metric starts off worse than ROUGE-L, but it manages to catch up with the latter at the end of the training. (BLEU@4 (blue line) scores much lower (around 0.10-0.11) and shows minimal improvement. The lower BLEU score compared to ROUGE-L suggest the model might be better at sentence structure(ROUGE-L) and more flexible with exact phrasing.

Accessing the out1.log the best reported scores for the relevant metrics are listed below:

- Best CIDEr score in epoch 8: 0.36089558491512846
- Best BLEU score in epoch 6: 0.10766230253280189

4.5. Comparison (Task 1 and Task 2).

The RNN shows (Figure 1) a smoother initial decline in loss compared to the LSTM (Figure 3).

4.5.1. Evaluation Metrics.

The RNN reaches a lower final loss value (1.8) than shown in your previous LSTM plot (2.3) Both show similar sawtooth patterns, but the RNN appears to have fewer total steps (800 vs 1400) which suggests inconsistencies in the out1.log, as I used the same plot.py. Other mentions:

- CIDEr scores are significantly higher with the RNN (0.49-0.59) compared to the LSTM (0.31-0.36)
- ROUGE-L scores are also improved with the RNN (0.40-0.42) versus the LSTM (0.35-0.36)
- BLEU scores nearly doubled with the RNN (0.17-0.20) compared to the LSTM (0.10-0.11)

The metrics show more pronounced improvement across epochs with the RNN. This result is surprising since I got the notion that LSTMs typically outperforms simple RNNs. Both architectures were ran before implementing the attention-mechanism. The only deviation between the when training was the use of `self.num_layers`. In task 1 this parameter was set to 1. As for task 2 I followed instructions in the task and set it to 2.

The significantly higher CIDEr scores with the RNN indicate it's producing more novel and diverse text that matches reference texts better than the LSTM model.

5. TASK 3: SIMPLE ATTENTION MODEL

For the simple attention model using the LSTM cell type the parameters stayed the same.

5.1. Loss graph.

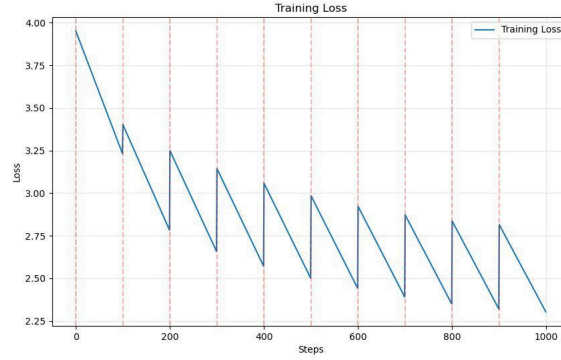


FIGURE 5. Training Loss curve for a double layer LSTM with attention.

Figure 5 displays the same decreasing sawtooth pattern in the loss curve as in both RNN and LSTM implementation without attention. The loss-value decreases from ~ 4.0 to ~ 2.3 , where introducing attention mechanism seem to introduce a marginal improvement to the LSTM without attention(see Figure 3). Figure 5 shows the same trend as the others, suggesting the model is still improving and has not plateaued.

5.2. Validation metrics.

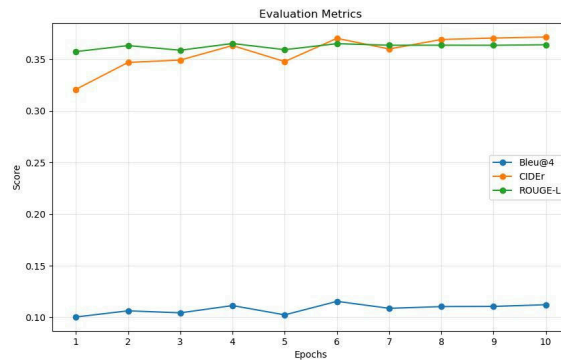


FIGURE 6. Evaluation metrics(Cider, Bleu,Rouge) for a double layer LSTM with attention.

The RNN reaches a lower final loss value (1.8) than shown in your previous LSTM plot (2.3) Both show similar sawtooth patterns, but the RNN appears to have fewer

total steps (800 vs 1400) which suggests inconsistencies in the `out1.log`, as I used the same `plot.py`. Other mentions:

- CIDEr scores start lower than ROUGE-L but improves and stays marginal higher after 8 epochs: 0.32 to 0.37
- ROUGE-L scores start out best (0.36-0.37) and stays around this value across the epochs
- BLEU scores are low but steady across all epochs (0.10-0.11)

5.2.1. *Comparison.*

Introducing attention mechanism to the LSTM cell type provided only marginal improvements in metrics compared to the LSTM without attention mechanism. This is depicted in both the loss graph (Figure 3) and validation metrics (Figure 4). Though the numbers of epochs seem to be greater for task 3 the CIDEr scores show slightly better convergence with attention, but both the loss graph and evaluation metrics seem similar both for task 2 and 3 in terms of graph trajectories and forms. The ROUGE-L scores are identical suggesting similar sentence structure capture. From the results we can observe that LSTM with the attention mechanism provides marginal improvements to LSTM without attention. As for the RNN architecture, it performs surprisingly well, even outperforms the LSTM with attention. These findings may cast light upon factors such as hyperparameters, model size and implementation are having significant effects, favoring the RNN implementation.

5.2.2. *Comment on production of results.*

There were problems to plotting when training the different architectures using the UiO's ML-nodes. Plots were not generated and as a result, I had to implement a plotting function which read from the training-score-history text-file, which can lead to errors and inconsistencies. This can be seen from how the the number of epochs are plotted.

Accessing the `out1.log` the best reported scores for the relevant metrics are listed below:

- Best CIDEr score in epoch 10: 0.3714443516481644
- Best BLEU score in epoch 6: 0.11546264533209777

6. ANALYSIS OF PREDICTION EXAMPLES

For the 5 example images with predicted captions were generated using the “top-bottom”-method for LSTM with attention. These are shown in Figure 7.

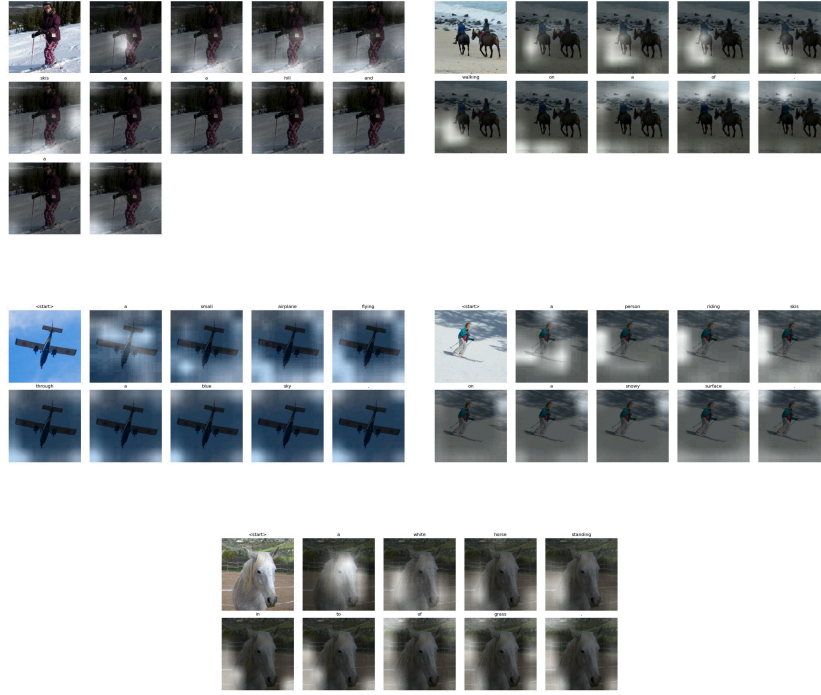


FIGURE 7. Top row displays predicted captions belonging to the “bottom”-category, as for the 3 remaining, they belong to the “top”-category. Numbered from top-left to bottom-right(1-5)

6.1. Observed Limitations.

Starting out with the less accurate captions, we have the following:

- Image 1: a person on skis skis a a hill and a (incomplete but mostly accurate)
- Image 2: a herd of cattle walking on a of (also incomplete and grammatical error)

For the bottom cases the model runs into grammatical issues, caption completeness and repetitive words, even though this seem minor, the caption completeness is more interesting as it is terminated prematurely or without punctuation. In image 2 we run into the awkward phrasing or grammatical error: “a herd of cattle walking *on a of*”. Other issues related to image 1 and 2 are scene complexity where the model struggles with complex scenes and giving a context. For instance, image 2 depicts two men riding horses on a beach with some rocks in the background. It also confuses cattle with horses. As for image 1 the scene belongs to a simpler composition. Another issue is the repetitive words, the model shows a tendency to repeat articles “a person on skis skis *a a hill and a*”. The 2 examples suggest that the model has attention limitations since it is not fully capturing the visual details in a complex scene.

6.2. Observed Strengths.

As for the more accurate captions:

- Image 3: a small airplane flying through a blue sky (highly accurate)
- Image 4: a person riding skis on a snowy surface (highly accurate)
- Image 5: a white horse standing in to of grass (mostly accurate with minor grammatical error “in to of”)

The observed strengths can be summarized as following:

- Object Recognition: The model does well identifying primary subjects (person, airplane, horse)
- Action Recognition: Correctly identifies activities like “riding,” “flying,” and “standing”
- Context Understanding: Recognizes relevant environmental elements like “snowy surface,” “blue sky,” and “grass”
- Attribute Detection: Successfully identifies attributes like “small” for the airplane and “white” for the horse

The LSTM with attention performs reasonably well for simpler scenes. This is seen in the visualization as bright areas, which highlight parts of the image that are used for generating the caption. However, the attention maps also focus on irrelevant areas of the image, which may suggest some of the caption inaccuracies. This may show the model’s shortcomings with complex compositions and syntax. These results align with the metrics that can be found in Task 1-3, showing modest but not dramatic improvements over the base LSTM model.