IN5400 - Mandatory 2

Alex

March 29, 2022

Submission due 19th of April 23:59 = 11:59PM Oslo time.

1 The mandatory exercise 2

Learning goals:

- Work with image captioning and outputs beyond simple fixed classes
- Complete a Recurrent Neural Net (RNN) which outputs a sequence of variable length
- Complete a 2-layer RNN
- Complete GRU and LSTM cell equations
- Implement a simple attention model on top

About the tasks: They are smaller than task1 and task2 from mandatory1, hopefully with less failpoints too.

- Task1 is a relatively simple effort of completing code.
- Task2 needs a larger change. It is probably the biggest task because you deal with two layers now. You implement GRU equations in this task.
- Task3 is just another RNN cell type (LSTM) over task 2. It is a quick effort to code, once task 2 works. The main difference to GRU is that you sometimes need the first half of a hidden state vector, and sometimes the second half, depending on whether you need the memory cell or the actual hidden state.
- Task4 is easier than task2 in terms of complexity, but it will need minor interface changes (a different CNN feature processor - you are dealing with unpooled features, an attention model head, and different features entering the RNN on the second level affecting the dimensionality on the second level). I kept the attention model head simple for this year.

1.1 Task 1

NOTE: **DO NOT** change the variable names, argument names or the function definitions.

- Use the code with

  `self.simplifiedrnn = True`

  in the

  `class ImageCaptionModel(nn.Module)`

  in cocoSource_xcnnfused.py

- Complete in

  `class ImageCaptionModel(nn.Module):`

  the

  `self.inputLayer`

  as a sequence of

  - dropout with set to zero probability of 0.25
  - a linear layer with

    `self.nnmapsize`

    many outputs
  - a leaky ReLU

- Complete in

  `class ImageCaptionModel(nn.Module):`

  the remaining parts

- Complete in

  `class RNNOneLayerSimplified(nn.Module):`

  the remaining parts

- Initialize the hidden states with zeros! The content gets infused into the RNN as input only

- Take note of the hidden state shape $(1, batchsize, hidden\_dim)$

- Train your model using these maxpooled features:

  `'featurepathstub': 'detectron2_lim10maxfeatures' ,`

  their actual path is in: /itf-fi-ml/shared/IN5400/dataforall/mandatory2/data/coco/

On training speed:

You will attain above 0.25 METEOR over 100 iterations, however the single layer RNN can be a bit unstable. After 10 iters you maybe have $0.08 - 0.1$, it can stay there for a while. After 25 iters maybe you have 0.18 or 0.12! After 40 iters you might be still a bit below 0.2.

On features:

- I used object detection features from the Detectron2 framework, however i did not use FB's official models on MSCoCo. I used object detection models pretrained on the visual genome dataset which gives stronger features, as they detect many more object classes. It would be worth playing with the LVIS models as well.
- Extracted features from Detectron2 are the top-10 detections, which gives per image a feature of shape $(10, 2048)$, which is as dataset of size around 10 Gbyte. I max-pooled it into a single feature of shape (2048), which is as dataset of size 1.3 Gbyte. Note that max-pooling is much stronger than average pooling (METEOR maxpooling: ≈ 0.255 METEOR avgpooling: ≈ 0.243). If you think that every element of the $(10, 2048)$-dim features is a detection for one single bounding box, then you can understand easily, why max-pooling is the better choice when you don't need the gradient to flow back into the feature generation stub.

On models:

For some better models see e.g., `https://paperswithcode.com/task/image-captioning`

1.2 Task 2

NOTE: **DO NOT** change the variable names, argument names or the function definitions.

- Goal of this task is to have a 2-layer RNN instead of 1 AND replace vanilla RNN with a GRU.
- We won't be using vanilla (simplified) RNN anymore. So it'll be false for this task and the rest of the tasks.
  Set

  ```
  self.simplifiedrnn = False
  ```

  in the

  ```
  class ImageCaptionModel(nn.Module)
  ```

  in cocoSource_xcnnfused.py
- It is advisable to create a separate copy of cocoSource_xcnnfused.py with a new filename, and a copy of Exercise_Train_an_image_captioning_network_test.py with a new filename which imports the

`class ImageCaptionModel`

from the new copy of cocoSource*

- implement a 2-layer GRU. There are two ways to take on this:

  Way one: at first implement the 2-layer structure reusing the RNNsimpleCell in

  `class RNN(nn.Module):`

  Way two: at first implement the GRUCell and plug it into the simplernn class in a one layer structure, then extend it to two layers.

  NOTE: Use the equations from the slides.

- Complete in

  `class RNN(nn.Module):`

  the remaining parts

- Complete in

  `class GRUCell(nn.Module):`

  the remaining parts

- Take note that inputs to the RNNcells are different in layer 1 and layer2 !
- Take note what to put in as dimensions into

  `input_size_list`

  for each layer

- Take note of the hidden state shape $(num\_layers, batchsize, hidden\_dim)$
- Write the code that it can be used also for 3 layers
- You will run into trouble if you would code like this:

  `hiddenstate[a,:,0:b] = rnncell(inputs,hiddenstate)`

  due to backward in place modification errors.

- Train your model using these maxpooled features:

  `'featurepathstub': 'detectron2_lim10maxfeatures' ,`

you will attain around 0.25 METEOR over 80 iterations,

1.3 Task 3

NOTE: DO NOT change the variable names, argument names or the function definitions.

- Use the code with

  ```
  self.simplifiedrnn = False
  ```

  in the

  ```
  class ImageCaptionModel(nn.Module)
  ```

  in cocoSource_xcnnfused.py

- Implement a 2-layer LSTM

- Complete in

  ```
  class LSTMCell(nn.Module):
  ```

  the remaining parts. Once you are done with task 2, it is just another RNN cell to be implemented.

- Take note of the hidden state shape for an LSTM is $(num\_layers, batchsize, 2*hidden\_dim)$!!!

  Reason: hidden state tensor in LSTM is the concatenation of memory cell and the actual hidden state along the third dimension. The first half represents the hidden state and the second half represents the memory cell state. You must return the output in the same format: 1^{st} half - new hidden state, 2^{nd} half - new cell state.

- Train your model using these maxpooled features:

  ```
  'featurepathstub': 'detectron2_lim10maxfeatures' ,
  ```

you will attain around 0.25 METEOR over 80 iterations,

1.4 Task4

- Use the code with

  ```
  self.simplifiedrnn = False
  ```

  in `class ImageCaptionModel(nn.Module)` in cocoSource_xcnnfused.py

- Goal: implement a simple attention model. Refer to the YouTube links in the slides to get an overview of what attention mechanism is. The video uses translation as an example but you can think of each input word in the video as a patch in image so instead of attending to different words in the input text, here you're instead attending to different parts of the image.

- Train your model using the following not pooled yet features:

  ```
  'featurepathstub': 'detectron2_lim10features',
  ```

- Create a modified version of `class ImageCaptionModel(nn.Module)` and of `class RNN(nn.Module)` – in a separate py-file for cocoSource*.py and a separate Exercise*.py file

- Model details:
  - Input feature shape is now: $(batchsize, 10, 2048)$ because you're using a different `featurepathstub` now (check above). Therefore, you need a different input layer to deal with these features.
  - Input layer will be a sequence of Dropout with a drop probability of 0.25, then a 1x1 1d-convolution with output channel size equal to self.nnmapsize,then a 1d BatchNorm, then a leaky ReLU.

    You can think why I want to use a 1x1 1d-conv instead of a fully connected layer here.
  - The first layer of the LSTM will take a max-pooled version of those features as input which pools over and removes the 10 in the shape (the number of detected windows). (If you would use average pooling, then you would get for a standard LSTM only 24.3 meteor and for attention LSTM maybe +0.5 .)
  - The last layer of the LSTM will take as input **a concatenation of** the previous layer hidden state **and** an attention-weighted sum of the feature with a weight of size 10.
  - you will need an attention layer which computes the weights, and you will need to pass this layer into the forward function as argument, so that it can be used inside the modified LSTM. Here a proposal for the attention layer: 2 layer MLP with dropout

    it takes the hidden state and the memory cell of the first layer as input, and processes it by: a dropout with a drop probability of 0.25, then a linear layer with 50 outputs, then a leaky ReLU, then a linear layer with 10 outputs, then a softmax

    This attention head used here is rather simple. More modern models use heads which are based on similarities between the 10 sub features and the first layer hidden state. They may calculate 2 or more different attention weights, and concatenate 2 or more attention-weighted features.
  - The inputsize for the RNN now has 3 possible sizes: input, intermediate size and last layer size (where one inputs the hidden state and the weighted image feature). Therefore the RNN initializer/constructor

    `def __init__(...)`

    must take one parameter more (last_layer_state_size) as input compared to the standard imagecap model, and this will affect the construction of `input_size_list`
  - The RNN forward will need to take at least the attention weights network as additional input

It is advisable to run this task on the cluster or a machine with 32 Gbyte Ram at least. You don't need to train this for 100 iterations. 30 or 50 suffice as you don't need to set new high scores.

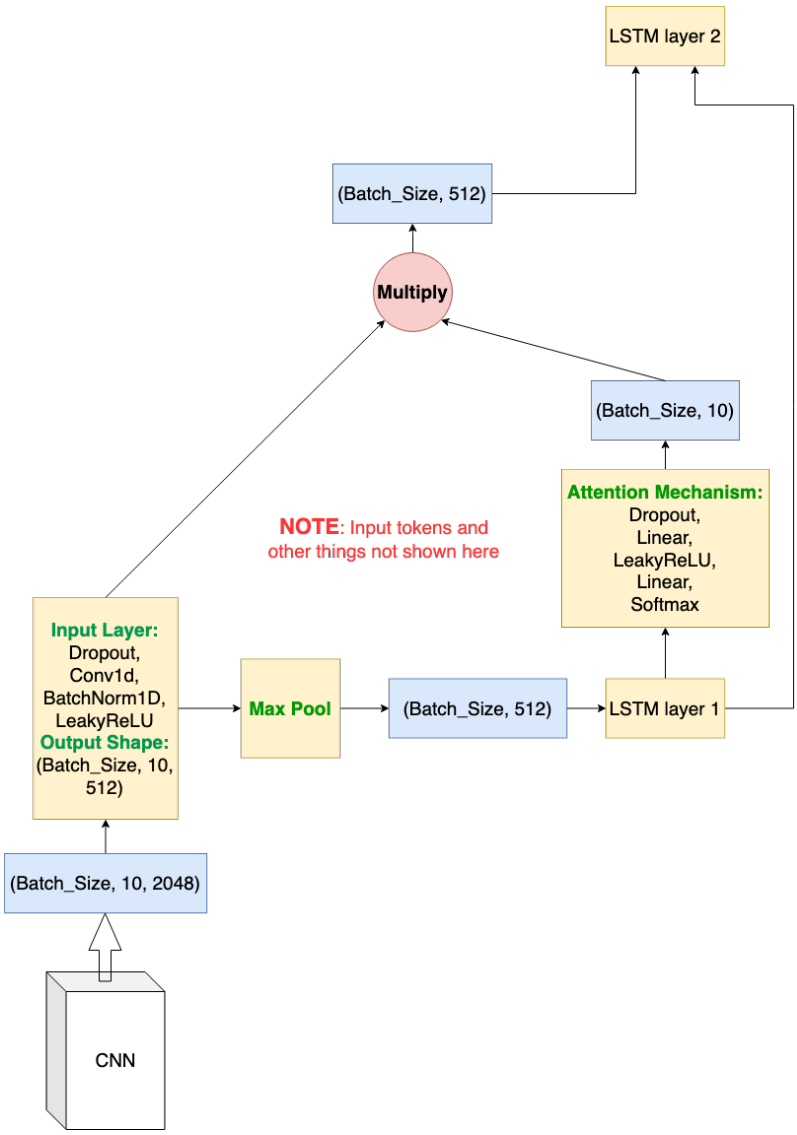

Figure 1: A diagram showing the important components of task4

1.5 Expected performance

in METEOR scores, when you selected the best score per epoch by peeking on your validation data:

- one layer RNN 25.2 to 25.6
- two layer GRU 25.3 to 25.7

- two layer LSTM 25.5 to 25.8
- two layer attention LSTM 25.7 to 26.0

Note that for reproducible research it is wrong to select on your test set.

Why these scores are so similar? Three main reasons

The pooled features are taken over 10 detection windows only, thus we extracted mostly high scoring and informative detection windows in any image. There is little potential to extract windows which could be sometimes more helpful for some images and uninformative for other images.

Max-pooling loses no information and creates a strong feature. It has disadvantages in terms of gradient flow, but we wont retrain the detector, so that is no argument against it.

For getting closer to the state of the art one would extract 36 or 50 windows per image. Then one would have higher potential for improved scores with models which can choose features, but this dataset would take 36 GB or 50 GB and not 10 GB.

For getting closer one would also code more complicated models, which demands more load from the cluster (I did not want to risk to overstress it, even though it looks well done), ... and more time from the coders (I did not want to risk to make you more stressed, even though your skills look like they have improved ;)). In short: lets keep it easy on you and the GPUs.

1.6 debugging ugly errors appearing during the backward pass

if you are not sure what shape you have somewhere, then just insert:

```
print(sometensor.shape)

exit()
```

if you run into errors saying that backward detected an inplace modification of a tensor,

```
Warning: Error detected in *
RuntimeError: one of the variables needed for gradient computation
has been modified by an inplace operation.
```

then the solution is usually to clone but not detach an offending tensor (bcs detach prevents gradient flow backwards!). `https://discuss.pytorch.org/t/backward-error-after-in-place-modification-only-if-using-tanh/78291`

if you fail to debug it, but you have really tried it on 2 different days, then reach out to me and we can check your code.

the most common error of this type would occur where you use the hidden state to compute the attention layer in Task4. Use then a .clone():

```
w= attlayer(sometensor.clone())
```

1.7 Running unit tests

This year we're providing you with some unit tests for LSTM and GRU equations. The idea is to help you ensure that your implementation is likely to be correct. We might have missed some cases so if unit tests pass, it does not guarantee that your equations are correct. However, if unit tests fail, your implementation is definitely incorrect.

We are using python's pytest library (needs to be installed separately in your python environment). We recommend you run unit tests locally on your PC by installing pytest as `pip install pytest`. The unit tests won't invoke any model training so they will run pretty fast (less than a minute). The tests are located in the `tests` directory. You can check pytest's documentation (`https://docs.pytest.org/en/7.1.x/how-to/usage.html`) for different ways of running unit tests via command line.

Steps for running unit tests:

1. Install pytest locally via `pip install pytest`.
2. Open the test files (test_lstm.py, test_gru.py) and modify the import statement to import your implementation of the LSTMCell and the GRUCell.
3. In the shell/terminal, navigate to the root directory of the project.
4. Run `PYTHONPATH=<PATH TO THE PROJECT> pytest tests/test_lstm.py` to invoke pytest for lstm. Please note that you might need to replace the slash here with the backward slash depending on your OS and the PYTHONPATH you set would depend on how you're importing the LSTMCell inside the test file test_lstm.py. Do the same for test_gru.py. You can also invoke pytest tests/. to run both LSTM and GRU tests.

1.8 Deliverables

Read carefully!

- Deliver for each task, one separate training start file. If you have to create separate versions of cocoSource_xcnnfused.py for different tasks, then you can do that too.
- For each task, report train-test loss curves/graphs and the METEOR validation score per epoch.
- For each task, report the best BLEU-4 and METEOR score which you achieved (they could be in different epochs)
- For any 1 task of your choise, deliver the pretrained model and one validation-only (training code not included) file, which takes the pretrained model and computes the validation error.
- Deliver a report as pdf-file with your full name – to identify the submitter
- Write in your report: the training parameters
- Write in your report: 5 example images with predicted captions which you obtained using your code – for one of the models

- Put everything : codes, saved models, the pdf and everything else you want to add **into one single zip file**.

Code guidelines:

- **DO NOT** change the variable names, argument names or the function definitions.
- One should only be required to change the path to the data folder in your code. All other paths should be relative to your main folder of your .py files, no absolute paths except for the dataset.
- path manipulations: os.path.join, os.path.basename, os.path.dirname, os.path.is*, , os.makedirs(...)
- Reproducibility: set all involved seeds to fixed values (python, numpy, torch). Check PyTorch's page on how to make PyTorch code reproducible: `https://pytorch.org/docs/stable/notes/randomness.html`
- A requirements.txt and write it into the pdf for any additional packages beyond numpy, scipy, matplotlib, pytorch and their dependencies
- Your deliverable should work with the following steps:
  - Unpack the zip files
  - Set **one single path** for the root of the dataset. This must be documented. Nothing else should need to set it up.
- Code should run without typing any parameters on the command line!!

```
CUDA_VISIBLE_DEVICES=x TMP=./tmp python blafile.py
```

- Python scripts.

1.9 GPU resources and data

You have two options: use your own GPU, or use the university provided resources

- ml6.hpc.uio.no

How to use them ?

- log in using ssh and your ifi username:

```
ssh proffarnsworth@ml6.hpc.uio.no
```

On windows PuTTY or MobaXterm may help you. On mac you can use ssh as is. I do use windows, but for games :D.

- each of these nodes has 8 GPUs, each with 11 Gbyte GPU Ram. The critical resource will be GPU ram. if you go over the limit, your script will die with a mem allocation error.
- use `n`vidia-smi to see which on which GPUs scripts are running and how much memory is used on each GPU. Choose a GPU such which has still 2.5 Gbyte RAM unused.
- **load your environment:** you need Java here too!

1.10 running your script on the cluster

Note the differences to mandatory1:

- setting a TMP environment variable
- a memory footprint of 2500 Mbytes
- module load needs also Java (for score evaluation)

```
module load PyTorch-bundle/1.10.0-MKL-bundle-pre-optimised
```

- to start a script on a specific GPU with numerical number $x \in \{0, \ldots, 7\}$ use the following command below. However this will stop when you log out of ssh. Thus this makes sense only to debug your code.

```
CUDA_VISIBLE_DEVICES=x TMP=./tmp  python yourscript.py
```

- to start a script which does not hang up on logout (on a specific GPU with numerical number $x \in \{0, \ldots, 7\}$), please use

```
CUDA_VISIBLE_DEVICES=x TMP=./tmp nohup python yourscript.py > out1.log 2> error1.log &
```

What does this do?

- nohup starts the command without hangup
- > out1.log redirects normal output onto out1.log
- 2 > error1.log redirects error messages onto error1.log
- & places the job in the background

The alternative to nohup is to use screen:

- first time: screen gets into screen
- next time: screen -r reattaches to an active screen
- Ctrl-a c creates a new screen
- Ctrl-a n switches to the next screen
- Ctrl-a n switches to the previous screen
- Ctrl-a d exists screen (see above how to reattach)
- Do not start a script when there are already 4 jobs running on it or when it is foreseaable that your 2.5Gbyte wont fit into this GPU RAM.
- Ensure that you only have 1 main python process running at a time. Run `pstree -p -U $USER` to check your process tree. When you're training a model, this should show you a tree with 1 main python process and some other python processes spun by it as branches of the tree. If you see some old python process you ran then kill the stale process with `kill -9 <PID>` where $\langle PID \rangle$ is the process ID shown in the parentheses. Refer to `https://man7.org/linux/man-pages/man1/pstree.1.html` for more details on pstree.

How to kill your own process?

ps -u proffarnsworth

↑ shows only the processes of the user proffarnsworth `https://en.wikipedia.org/wiki/Professor_Farnsworth`

ps -u proffarnsworth | grep -i python

↑ shows only the processes of user proffarnsworth which are python. The -i makes a case sensitive grep search. If you see nothing, then you may have mistyped your command, or you are not using python, or your process has already finished.

- both of these will show you process ids (PID)s

kill -9 PID

↑ kills your process with pid PID

1.11 I use my own GPUs, where to get the data?

in the folder /itf-fi-ml/shared/IN5400/dataforall/mandatory2/data/coco/

- ...

Writing a mandatory exercise ... you can guess how long it took for that pdf, now it is your time :).