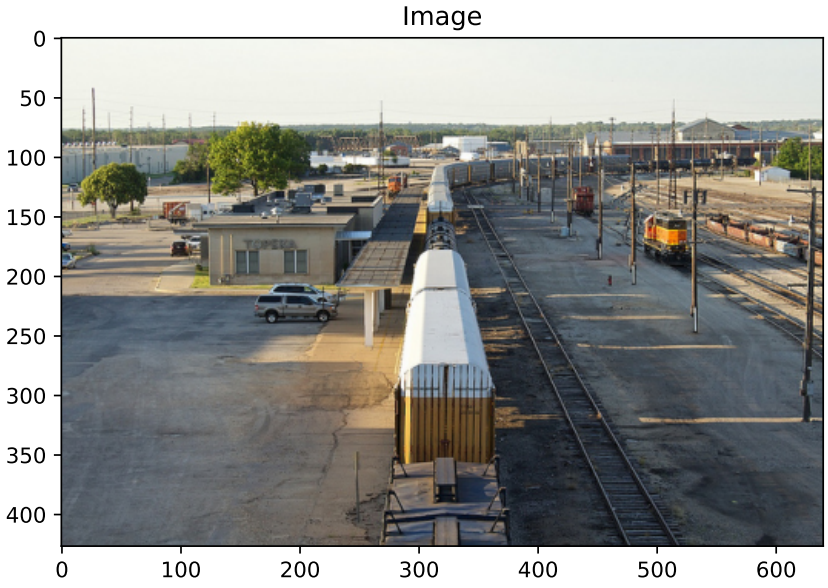
Report of the Mandatory Assignement 2

Lina Saba

29 March 2022, 19th April 2022



A long train traveling through a train yard.

Preview of the data

Contents

1 Introduction

This is a detailed report of the steps that were followed during the second mandatory exercises.

I will put the changed code with its explanation each time I changed the code.

2 Explanation of the Tasks :

RNN = Recurrent neural network uses the gradient descent method to update the weights between its neurons. Its weights are updated from right to left. As we move to the left, The weights of the first layers of neurons are almost unchanged. Thus, these layers learn absolutely nothing. We say that RNN suffers from the gradient dissipation problem.

Towards a better architecture :

1. LSTM = Long Short-Term Memory is composed of 3 gates; calculation areas that regulate the flow of information. Store or delete information in its memory.
2. GRU = Gated Recurrent Unit , appearance of a new operation which consists in inverting a vector with respect to 1.

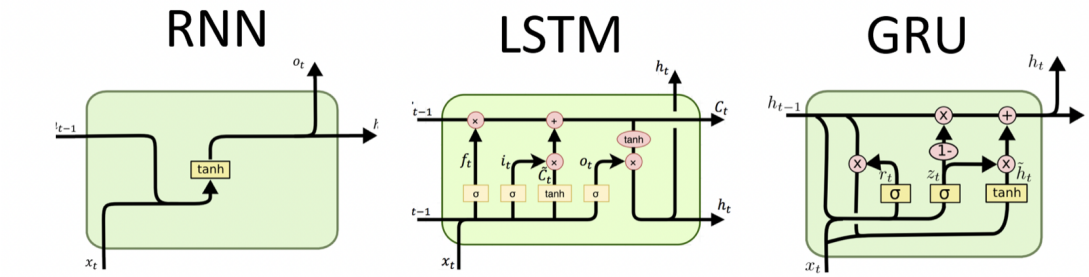


Figure 1: RNN vs LSTM vs GRU

3 Task 1 :

The main goal of this Task was to complete the classes *ImageCaptionModel* and *RNNOneLayerSimplified* codes'.

I added this to the *ImageCaptionModel* _init_() code :

```
self.nn_map_size = 512

# Create the network layers
self.embedding_layer = nn.Embedding(self.vocabulary_size, self.embedding_size)
self.output_layer = nn.Linear(self.hidden_state_sizes, self.vocabulary_size)
self.input_layer = nn.Sequential(
```

```
                        nn.Dropout(p=0.25),
                        nn.Linear(2048, self.nn_map_size),
                        nn.ReLU()
                        )

        self.simplified_rnn = True
```

For the _forward_() part :

```
        if current_hidden_state is None:
            device = cnn_features.get_device()
            initial_hidden_state = torch.zeros(self.num_rnn_layers,
                                               x_tokens.shape[0],
                                               self.hidden_state_sizes,
                                               device=device)
        else:
            initial_hidden_state = current_hidden_state
```

We needed the initial_hidden_state shape to be [num_rnn_layers, batch_size, hidden_state_sizes], the num_rnn_layers and hidden_state_size were already defined in the _init_(), batch_size is the 1st shape of x_tokens, so we define it as x_tokens.shape[0]

For the *RNNOneLayerSimplified* class, only the _forward_() part was modified :

```
        current_time_step_embeddings = all_embeddings[:, 0, :]
        for i in range(sequence_length):
            input_for_the_first_layer = torch.cat((current_time_step_embeddings,
                                               processed_cnn_features),dim=1)
            current_hidden_state = self.cells[0](input_for_the_first_layer,
                                               current_hidden_state[0, :])
            current_hidden_state = torch.unsqueeze(current_hidden_state,0)
```

As a result model_0 was created and a model_0.png as shown as following :

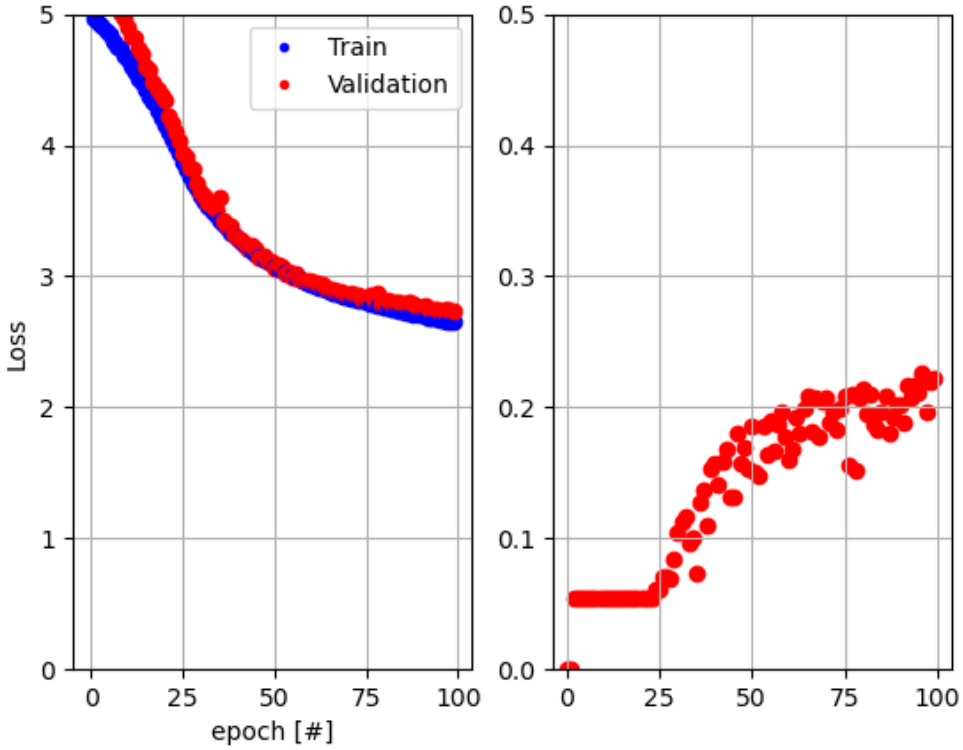


Figure 2: Train and Validation per epochs

This represents the train-validation loss curve and the METEOR validation score per epoch.

We can see that the train and validation decrease in the same shape to the approximate value of 2.7 after 100 epoch. For the Meteor value, it's stable around 0.5 till epoch 25 and increase exponentially to the value of 0.22 around 100 epoch.

Achieved scores : For the BLEU-4 : 0.23852555241162596 and METEOR score : 0.21554031954358588

4 Task 2 :

A new python file **cocoSource2.py** was created from **cocoSource1.py** and implemented this way:

Two main goals were demanded in this task :

1. Replacing vanilla RNN with a GRU :

   In the class *ImageCaptionModel*, the self.simplified_rnn was changed to False :

   ```
   self.simplified_rnn = False
   ```

2. Implementing a 2-layer GRU

   For the class *GRUCell*, we initialize the weights this way :

   ```
   is_hss = input_size + hidden_state_size

   self.weight_u = nn.Parameter(torch.randn(is_hss, hidden_state_size) / np.sqrt(is_hss))
   self.bias_u = nn.Parameter(torch.zeros(1, hidden_state_size))

   self.weight_r = nn.Parameter(torch.randn(is_hss, hidden_state_size) / np.sqrt(is_hss))
   self.bias_r = nn.Parameter(torch.zeros(1, hidden_state_size))

   self.weight = nn.Parameter(torch.randn(is_hss, hidden_state_size) / np.sqrt(is_hss))
   self.bias = nn.Parameter(torch.zeros(1, hidden_state_size
   ```

   is_hss is the variable that will stock the addition of the input_size and the hidden_state_size.

   The _forward_() of this class becomes :

   ```
   sig_u = torch.sigmoid(torch.cat((x, hidden_state), 1).mm(self.weight_u) + self.bias_u)
   sig_r = torch.sigmoid(torch.cat((x, hidden_state), 1).mm(self.weight_r) + self.bias_r)

   tanh = torch.tanh(torch.cat((x, (sig_r * hidden_state)), 1).mm(self.weight + self.bias))

   state_new = sig_u * hidden_state + (1 - sig_u) * tanh
   ```

   In the _init_() of the class *RNN*

   ```
   input_size_list = [GRUCell(self.hidden_state_size, self.input_size)]
   hidden_state_size_origine = self.hidden_state_size
   input_size_list.extend([GRUCell(self.hidden_state_size, hidden_state_size_origine) for i in ra
   ```

   This creates a list of type "nn.ModuleList" input_size_list and populate it with cells of type "self.cell_type" using the GRUCell using the variables previously defined.

   In the _forward_() of the class *RNN*

```
for i in range(sequence_length):
    list_layer_state = torch.zeros_like(current_hidden_state)
    input = torch.cat((current_time_step_embeddings, processed_cnn_features), dim=1)

    for layer in range(self.num_rnn_layers):
        new_input = self.cells[layer](input, current_hidden_state[layer, :])
        list_layer_state[layer, :] = new_input
        input = new_input

    current_hidden_state = list_layer_state
    logits_i = output_layer(input)
    logits_sequence.append(logits_i)
    predictions = torch.argmax(logits_i, dim=1)
```

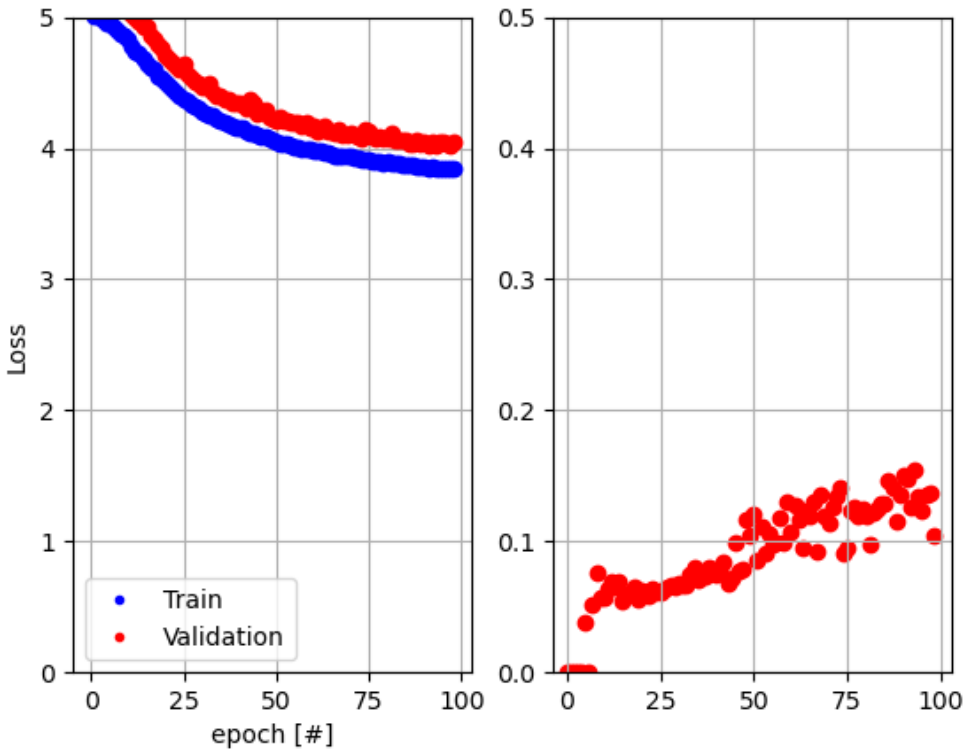


Figure 3: Train and Validation per epochs

We can see that the train and validation decrease in the same shape to the approximate value of 3.7 after 100 epoch. For the Meteor value, it increases till 0.15 around 100 epoch.

Achieved scores : For the BLEU-4 : 00.07843417094266235 and METEOR score : 0.15256612642399275

Unfortunately Meteor doesn't attain the suggested score over 100 epoch.

5 Task 3 :

1. Implementing a 2-layer LSTM For the class LSTMCell, we initialize the weights this way :

```
is_hss = input_size + hidden_state_size
self.weight_f = nn.Parameter(torch.randn(is_hss, hidden_state_size) / np.sqrt(is_hss))
self.bias_f = nn.Parameter(torch.zeros(1, hidden_state_size))
# Input gate parameters
self.weight_i = nn.Parameter(torch.randn(is_hss, hidden_state_size) / np.sqrt(is_hss))
self.bias_i = nn.Parameter(torch.zeros(1, hidden_state_size))
# Output gate parameters
self.weight_o = nn.Parameter(torch.randn(is_hss, hidden_state_size) / np.sqrt(is_hss))
self.bias_o = nn.Parameter(torch.zeros(1, hidden_state_size))
# Memory cell parameters
self.weight = nn.Parameter(torch.randn(is_hss, hidden_state_size) / np.sqrt(is_hss))
self.bias = nn.Parameter(torch.zeros(1, hidden_state_size))
```

   The _forward_() of this class becomes :

```
h_in, C_in = torch.split(tensor=hidden_state,
                         split_size_or_sections=self.hidden_state_size, dim=1)
sig_i = torch.sigmoid(torch.cat((x, h_in), 1).mm(self.weight_i) + self.bias_i)
sig_o = torch.sigmoid(torch.cat((x, h_in), 1).mm(self.weight_o) + self.bias_o)
sig_f = torch.sigmoid(torch.cat((x, h_in), 1).mm(self.weight_f) + self.bias_f)
th = torch.tanh(torch.cat((x, h_in), 1).mm(self.weight) + self.bias)

C = sig_f * C_in + sig_i * th
h = sig_o * torch.tanh(C)

new_hidden_state = torch.cat((h, C), dim=1)
```

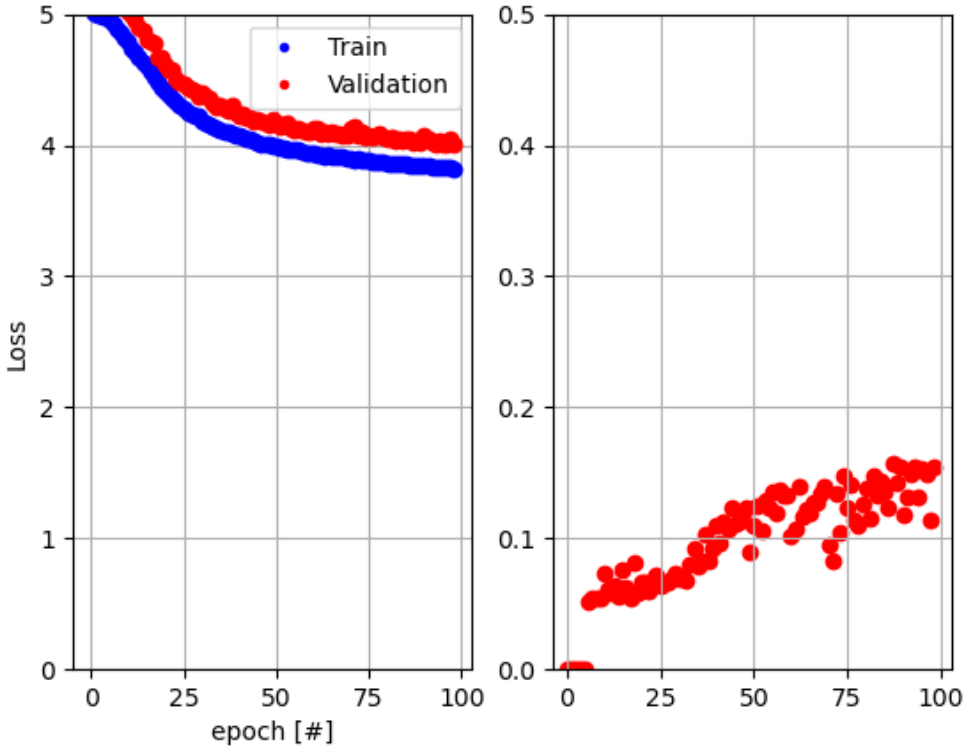


Figure 4: Train and Validation per epochs

We can see that the train and validation decrease in the same shape to the approximate value of 4 after 100 epoch. However, the validation curve isn't following the exact same path as the train curve. For the Meteor value, it increases till 0.17 during the 100 epochs.

Achieved scores : For the Bleu-4 : 0.07494567831668181 and METEOR score : 0.15327521194188137
Same as the task 2 : Unfortunately Meteor doesn't attain the suggested score over 100 epoch.

6 Task 4 :

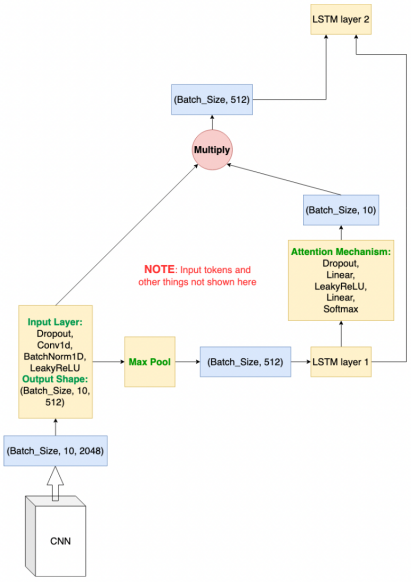


Figure 5: Steps for Task 4 :

These codes were completed following this diagram :

The _init_() of *ImageCaptionModel* becomes :

```
self.input_layer = nn.Sequential(nn.Dropout(p=0.25),
nn.Conv1d(self.number_of_cnn_features ,self.nn_map_size, kernel_size=1),
nn.BatchNorm1d(self.nn_map_size), nn.LeakyReLU() )

self.attention = nn.Sequential(nn.Dropout(p=0.25),
nn.Linear(self.nn_map_size*2, 50), nn.LeakyReLU(), nn.Linear(50, 10),
nn.Softmax(dim=1))

self.simplified_rnn = False
...
self.rnn = RNN(input_size=self.embedding_size + self.nn_map_size,
                           hidden_state_size=self.hidden_state_sizes,
                           num_rnn_layers=self.num_rnn_layers,
                           last_layer_state_size=2 * self.hidden_state_sizes,
                           cell_type=self.cell_type)
```

The _forward_() part :

```
cnn_features = cnn_features.unsqueeze(2)
processed_cnn_features = self.input_layer(cnn_features)
cnn_features = torch.transpose(cnn_features, dim0=1, dim1=2)
if current_hidden_state is None:
```

```
    device = cnn_features.get_device()
    initial_hidden_state = torch.zeros((self.num_rnn_layers, cnn_features.shape[0], 2 * self.hidden
else:
    initial_hidden_state = current_hidden_state

logits, hidden_state = self.rnn(x_tokens, processed_cnn_features, initial_hidden_state,
self.output_layer, self.attention, self.embedding_layer, is_train)
```

For the RNN class; _init_() :

```
cell_1 = LSTMCell(self.hidden_state_size, self.input_size)
cell_2 = LSTMCell(self.hidden_state_size, last_layer_state_size)
```

The _forward_() part :

```
max_pool_1d = nn.MaxPool1d(kernel_size=processed_cnn_features.shape[2])
x = torch.squeeze(max_pool_1d(processed_cnn_features))

embeddings = embedding_layer(input=tokens)  # Should have shape (batch_size, sequence_length, embed
logits_sequence = []
current_hidden_state = initial_hidden_state
#print(current_hidden_state.shape)
current_time_step_embeddings = embeddings[:,0,:]

for i in range(sequence_length):
    list_layer_state = torch.zeros_like(current_hidden_state)
    input1 = torch.cat((current_time_step_embeddings, x), dim=1)

    layer1 = self.cells[0](input1, current_hidden_state[0, :])
    out_attention = attention_lay(layer1)
    out_attention = torch.unsqueeze(out_attention, dim=1)
    param_multi = processed_cnn_features * out_attention
    param_multi = torch.sum(param_multi, dim=2)

    list_layer_state[0, :] = layer1

    input1, _ = torch.split(layer1, self.hidden_state_size, dim=1)

    input2 = torch.cat((input1, param_multi), dim=1)
    layer2 = self.cells[1](input2, current_hidden_state[1, :])

    list_layer_state[1, :] = layer2

    current_hidden_state = list_layer_state
    input_for_logit, _ = torch.split(current_hidden_state[1, :], self.hidden_state_size, dim=1)
    logits_i = output_layer(input_for_logit)

    logits_sequence.append(logits_i)

    predictions = torch.argmax(logits_i, dim=1)
```

We can see that the train and validation decrease in the same shape to the approximate value of 3.7 after 100 epoch. For the Meteor value, it increases till 0.15 around 100 epoch.

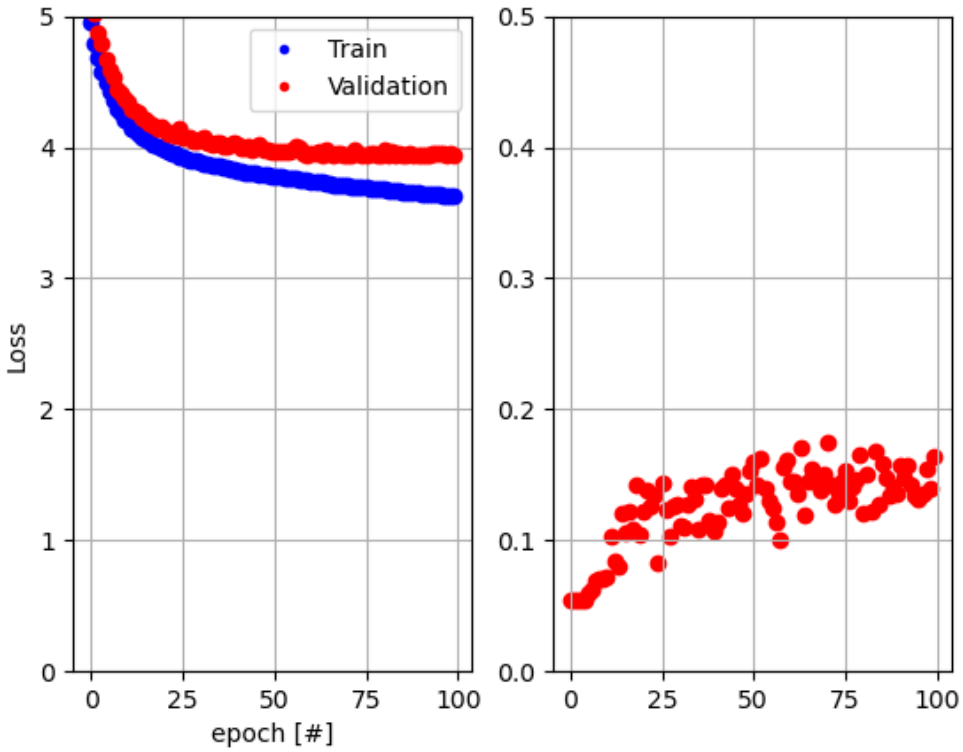


Figure 6: Train and Validation per epochs

Achieved scores : For the BLEU-4 : 00.10098888967293955 and METEOR score : 0.1635375615491447
Unfortunately Meteor doesn't attain the suggested score over 100 epoch.

7 Task4 with a loop :

For the RNN class; _init_() :

```
for i in range(self.num_rnn_layers):
    if i != self.num_rnn_layers-1 :
        list.append(input_size_layer[1])
    else :
        list.append(input_size_layer[2])

lstm_layer = nn.ModuleList([LSTMCell(input_size_layer[0], input_size_layer[1])])
lstm_layer.extend([LSTMCell(input_size_layer[0], list] for i in range(input_size_layer[3]-1))
self.cells = nn.ModuleList(lstm_layer)
```

The _forward_() part :

```
max_pool_1d = nn.MaxPool1d(kernel_size=processed_cnn_features.shape[2])
x = torch.squeeze(max_pool_1d(processed_cnn_features))

embeddings = embedding_layer(input=tokens)  # Should have shape (batch_size, sequence_length, embed
logits_sequence = []
current_hidden_state = initial_hidden_state
#print(current_hidden_state.shape)
current_time_step_embeddings = embeddings[:,0,:]

for i in range(sequence_length):
    list_layer_state = torch.zeros_like(current_hidden_state)
    input1 = torch.cat((current_time_step_embeddings, x), dim=1)

    for j in range(self.num_rnn_layers -1):
        layer1 = self.cells[j](input1, current_hidden_state[j, :])
        out_attention = attention_lay(layer1)
        out_attention = torch.unsqueeze(out_attention, dim=1)
        param_multi = processed_cnn_features * out_attention
        param_multi = torch.sum(param_multi, dim=2)

        list_layer_state[j, :] = layer1
        input1, _ = torch.split(layer1, self.hidden_state_size, dim=1)
        if j == self.num_rnn_layers -1 :
        input2 = torch.cat((input1, param_multi), dim=1)
        layer2 = self.cells[j](input2, current_hidden_state[j, :])

        list_layer_state[j, :] = layer2

        current_hidden_state = list_layer_state
        input_for_logit, _ = torch.split(current_hidden_state[j, :], self.hidden_state_size, dim=1)
    logits_i = output_layer(input_for_logit)
```

```
    logits_sequence.append(logits_i)

    predictions = torch.argmax(logits_i, dim=1)
```

8 Result of the predictions using the first model :

5 example images with predicted captions which you obtained using your code, by uncommenting this line :

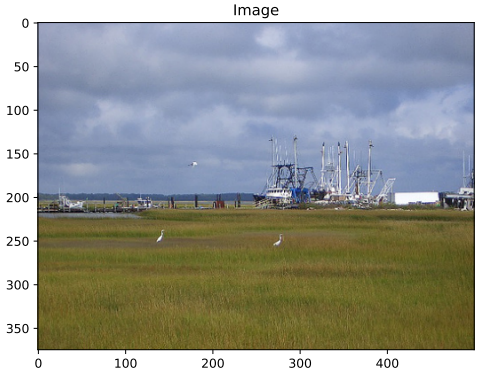
```
plotImagesAndCaptions(model, modelParam, config, dataLoader)
```

and add this to the end of validate.py in utils folder :

```
plt.title("Image")
plt.savefig("Image.pdf")
plt.close()
```

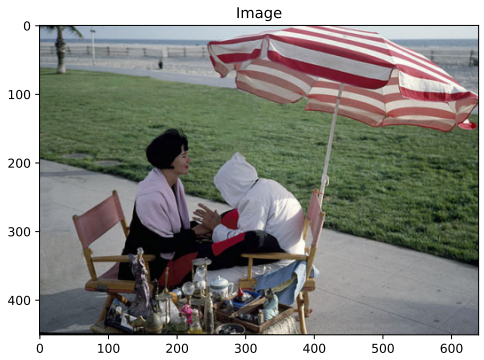
Generated caption a flock of birds flying over a

Original captions: A marsh area with egrets and shrimp boats An overcast day at dock with sea birds resting in the nearby marsh. A couple of birds that are walking in a field. ships in the distance behind a grassy marsh with egrets Several ships in the distance behind a grassy marsh with egrets A couple of birds walk through a field of tall grass near a sail boat harbor.



Generated caption a woman sitting on a bench with a

Original captions: A couple of people sitting in chairs under an umbrella. Two people sit closely together on chairs under an umbrella. Two people sit facing each other under an umbrella. Two women sitting outside,one holding the others hand. A couple of women are sitting under an umbrella

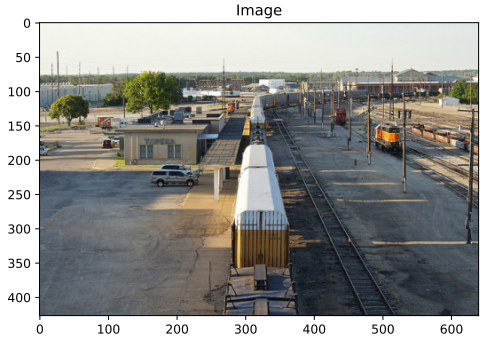


Generated caption a toothbrush and a toothbrush in a

Original captions: A toothbrush in a cup sitting next to a sink. A toothbrush is sitting in a colorful cup. Fancy colored toothbrush sitting in decorative mug, in a bathroom. A bathroom sink with two toothbrush holders on it. A toothbrush is sitting inside a cup next to a sink.

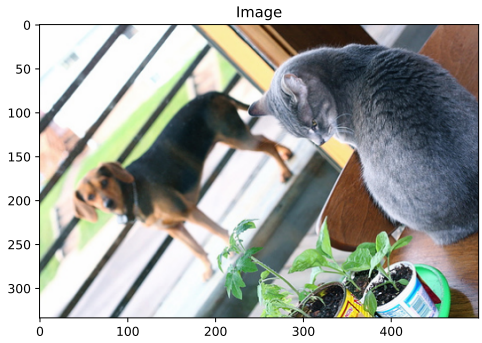
Generated caption a train traveling down a street next to a

Original captions: A long train traveling through a train yard. a train on a train track next to a parking lot An outdoor train yard area with several sets of train tracks and train cars extending down one set of tracks. A long train pulling several cars along the tracks A landscape view of a railroad yard and train



Generated caption a dog is sitting on a

Original captions: A cats sits on the table with a dog outside. A dog and a cat are watching the plants on the table.. A dog looks through a window at a cat sitting on a table. A cat sitting on a table next to plants and a dog outside of a sliding glass door, looking in. Dog looking in a window at a cat next to plants.



9 Training parameters :

```
modelParam = {
        'batch_size': 128,  # Training batch size
        'cuda': {'use_cuda': True,  # Use_cuda=True: use GPU
                 'device_idx': 0},  # Select gpu index: 0,1,2,3
        'numbOfCPUThreadsUsed': 10,  # Number of cpu threads use in the dataloader
        'numbOfEpochs': 99,  # Number of epochs
        'data_dir': data_dir,  # data directory
        'img_dir': 'loss_images_test/',
        'modelsDir': 'storedModels_test/',
        'modelName': 'model_0/',  # name of your trained model
        'restoreModelLast': 0,
        'restoreModelBest': 0,
        'modeSetups': [['train', True], ['val', True]],
        'inNotebook': False,  # If running script in jupyter notebook
        'inference': False
    }

config = {
        'optimizer': 'adamW',  # 'SGD' | 'adam' | 'RMSprop' | 'adamW'
        'learningRate': {'lr': 0.001},  # learning rate to the optimizer
        'weight_decay': 0.00001,  # weight_decay value
        'number_of_cnn_features': 2048,  # Fixed, do not change
        'embedding_size': 300,  # word embedding_layer size
        'vocabulary_size': 10000,  # number of different words
        'truncated_backprop_length': 25,
        'hidden_state_sizes': 512,  #
        'num_rnn_layers': 1,  # number of stacked rnn's
        'scheduler_milestones': [75,90], #45,70 end at 80? or 60, 80
        'scheduler_factor': 0.2, #+0.25 dropout
        #'featurepathstub': 'detectron2vg_features' ,
        #'featurepathstub': 'detectron2m_features' ,
        #'featurepathstub': 'detectron2cocov3_tenmfeatures' ,
        'featurepathstub': 'detectron2_lim10maxfeatures' ,
        'cellType':  'RNN' #'GRU'  # RNN or GRU or GRU??
    }
```

10 Details of obtaining the results :

1. starting a named screen session

   ```
   screen -S session_name
   ```

2. Import the coco file with the edited code :

   ```
   scp -r /Users/lina/Documents/Mandatory2/0version/cocoSource_xcnnfused.py
   linasa@ml6.hpc.uio.no:./Mandatory2
   ```

3. Run the code :

```
ssh linasa@ml6.hpc.uio.no
module load PyTorch-bundle/1.10.0-MKL-bundle-pre-optimised
nvidia-smi
CUDA_VISIBLE_DEVICES=2 python3 Mandatory2/Exercise_Train_an_image_captioning_network_test.py
```

10.1 The curves and Model :

Start a new terminal page, and run to get the curves and the saved model locally :

```
scp linasa@ml6.hpc.uio.no:./Mandatory2/loss_images/model_0.png /Users/lina/Documents
scp -r linasa@ml6.hpc.uio.no:./Mandatory2/storedModels_test /Users/lina/Documents
```

10.2 The best values of Meteor and Bleu-4

To get the best values of Meteor and Bleu-4, the value of *inference* == *True* and re-run the code

```
vim Exercise_Train_an_image_captioning_network_test.py
inference == True
CUDA_VISIBLE_DEVICES=2 python3 Mandatory2/Exercise_Train_an_image_captioning_network_test.py
```

References

[1] https://phoenixnap.com/kb/how-to-use-linux-screen-with-commands

[2] https://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/

[3] https://github.com/henriklg/image-captioning-network

[4] https://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/#Comment_fonctionne_un_RNN

[5] http://dprogrammer.org/rnn-lstm-gru