Assignment 2

This assignment has 2 parts. The evaluation of this assignment will be performed as an overall review of the hand-in, by your TA. To pass the assignment, the TA must be convinced that you have understood all portions of the curriculum which are covered by the assignment.

We expect you to hand in individual assignments, but you are allowed to discuss the questions with each other. **This means that you need to write your own answers, and your own, personal code, for all questions.**

Besides getting the correct answers, solutions will also be judged based on clarity, efficiency and brevity.

The assignment is due on Thursday 9th of November, at 11 pm. We expect each student to upload a pdf of their answers (written in whatever language your TA can read). Same expectations apply regarding formatting, as did for assignment 1.

When preparing your notebook, you may assume that the data files accompanying this assignment are placed in the same folder as the script.

Problem 1

a: NLP
Please discuss the recent trend of rapidly increasing sizes of NLP architectures.

b: Transfer Learning
Classify the following example of transfer learning. More exactly, what are the domains and tasks, and which are being changed?

Source: Using a step counter to monitor exercise in healthy people.

Target: Using a step counter to indicate recovery progression in a patient

c: Attention

Assume dotproduct attention, and that the hidden states of the encoder layer are [0,1,4],[-1,1,2],[1,1,1],[2,1,1]. If the activation for the previous decoder is [0.1,1,-2], what is the attention-context vector?

d: Transformers

Explain the 'positional encoding' step for transformers. Why is it done, how is it done?

e: Bounding box detection:
Given a dataset with two classes; cats and dogs, and the following detections:

TP = True positive
FP = False positive

cat_det = [TP, FP, TP, FP, TP]
pred_scores_cat = [0.7, 0.3, 0.5, 0.6, 0.55]

dog_det = [FP, TP, TP, FP, TP, TP]
pred_scores_dog = [0.4, 0.35, 0.95, 0.5, 0.6, 0.7]

There are in total 3 cats and 4 dogs in the images.

Calculate the mean average precision (mAP)

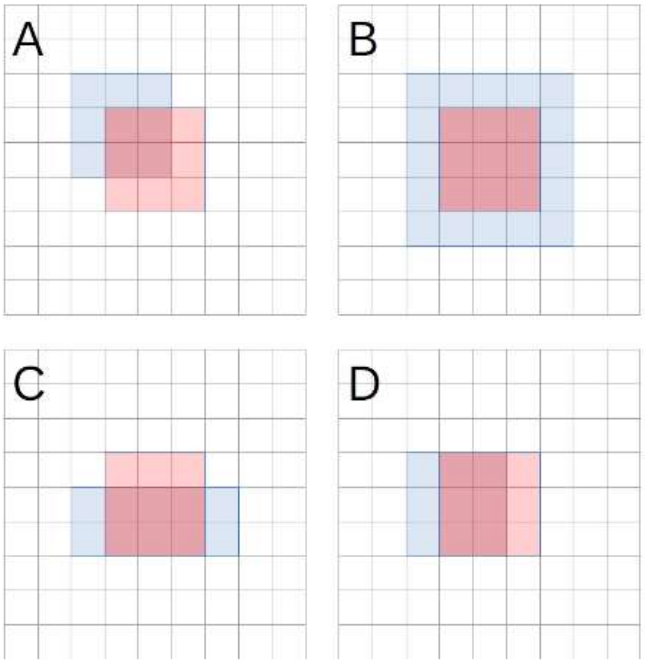
f: Semantic segmentation - FCN 1:
Given an image sized 1024x768x3 (width x height x channels), with 7 classes. What is the size of the target image if targets are one-hot encoded?

g: Semantic segmentation - FCN 2:
What is a fully-convolutional network? When is it useful?

h:Residual Networks:
Explain residual layers and their advantage.

i: Intersection-Over-Union

Calculate the intersection over union in for these four bounding-boxes and target bounding boxes:



j: Variational autoencoders:

What are the strengths of a variational autoencoder (VAE) compared to an autoencoder (AE)?

Answers:

a: NLP

NLP has been a very popular topic for a long time, however, until recently it has never performed well enough to be sought after my regular people. That is until the introduction of ChatGPT-3, which attracted alot of attention because of its great capability to answer all sorts of questions people might have, and also fairly accurately. This improvement came because, OpenAI trained this very large NLP architecture, which took thousands of 'peta-flop days' to perform a single training of GPT-3. This requires an huge computer to run and costs million US dollars in electricity. Nevertheless, perople have been on the trend of building larger NLP architectures, such as GPT-4 and so on, which perform even better at all sort of tasks. However, amidst the chase for bigger models, there's a quiet trend towards crafting smaller, efficient modes, where both size and efficiency matter.

b: Transfer Learning

In the given instance of transfer learning, we're taking a model that was originally used in one setting (source domain) and tweaking it to be applied in another (target domain). The domains and tasks for both the original and new settings can be expressed as:

$$
\begin{aligned}
D_S &= \{\mathcal{X}_S, P(\mathcal{X})\} \\
T_S &= \{y, f(\mathcal{X})\} \\
D_T &= \{\mathcal{X}_S, P(\mathcal{X})\} \\
T_T &= \{y, f(\mathcal{X})\}
\end{aligned}
$$

Here, $\mathcal{X}_S$ denotes the feature space, $P(\mathcal{X})$ is the marginal distribution, y represents the possible labels, and $f\mathcal{X})$ is the target function we aim to learn.

For this particular example, we define:

Source Domain and Task (D_S, T_S):

- Feature space ($\mathcal{X}_S$): Data captured by a step counter.
- Marginal distribution ($P(\mathcal{X})$): Patterns of step counts in healthy individuals.
- Labels (y): Categories of physical activity intensity.
- Prediction function ($f(\mathcal{X})$): Used to track physical activity.

Target Domain and Task (D_T, T_T):

- Feature space ($\mathcal{X}_S$): Remains as step counter data.
- Marginal distribution ($P(\mathcal{X})$): Adjusted to reflect recovery patterns in patients.
- Labels (y): Different recovery phases.
- Prediction function ($f(\mathcal{X})$): Repurposed to assess progress in recovery.

In this scenario of transfer learning, we keep the feature space unchanged while we recalibrate the marginal distributions and tasks to cater to the target domain's requirements of patient recovery monitoring.

c: Attention

Dot-product attention computes the attention scores by taking the dot product of the query (from the decoder) with each of the keys (from the encoder). The attention context vector is then computed as a weighted sum of the values (also from the encoder), where the weights are the softmax-normalized attention scores.

Here, the query q from the decoder is $[0.1, 1, -2]$, and the keys K from the encoder are $[0, 1, 4]$, $[-1, 1, 2]$, $[1, 1, 1]$, $[2, 1, 1]$. The values V are the same as the keys in this context since we're talking about a simple encoder-decoder without separate value vectors.

To compute the attention-context vector, we need to:

1. Calculate the attention scores by performing the dot product of the query with each key.
2. Apply a softmax function to the attention scores to get the weights.
3. Compute the context vector as the weighted sum of the encoder's hidden states.

Let's calculate this step by step:

1. **Attention Scores**:

$$
\begin{aligned}
e_1 &= q \cdot K_1 = [0.1, 1, -2] \cdot [0, 1, 4] = 0 \times 0.1 + 1 \times 1 + (-2) \times 4 = -7 \\
e_2 &= q \cdot K_2 = [0.1, 1, -2] \cdot [-1, 1, 2] = (-0.1) \times 1 + 1 \times 1 + (-2) \times 2 = -3.1 \\
e_3 &= q \cdot K_3 = [0.1, 1, -2] \cdot [1, 1, 1] = 0.1 \times 1 + 1 \times 1 + (-2) \times 1 = -0.9 \\
e_4 &= q \cdot K_4 = [0.1, 1, -2] \cdot [2, 1, 1] = 0.2 \times 1 + 1 \times 1 + (-2) \times 1 = -0.8
\end{aligned}
$$

1. **Softmax-normalized Weights**:

$$
\alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^{4} \exp(e_j)}
$$

1. **Context Vector**:

$$
c = \sum_{i=1}^{4} \alpha_i \cdot K_i
$$

Let's go ahead and compute the actual values.

In [ ]:
```
import torch
import torch.nn.functional as F

# Encoder hidden states and the previous decoder activation
encoder_hidden_states = torch.tensor([[0, 1, 4], [-1, 1, 2], [1, 1, 1],
                                      [2, 1, 1]], dtype=torch.float32)
decoder_activation = torch.tensor([0.1, 1, -2], dtype=torch.float32)

# Calculate the attention scores
attention_scores = torch.matmul(encoder_hidden_states, decoder_activation)

# Apply softmax to get the weights
attention_weights = F.softmax(attention_scores, dim=0)

# Calculate the context vector as the weighted sum of the encoder hidden states
context_vector = torch.sum(attention_weights.unsqueeze(1) * encoder_hidden_states,
                           dim=0)

attention_scores, attention_weights, context_vector
```

Out[ ]:
```
(tensor([-7.0000, -3.1000, -0.9000, -0.8000]),
 tensor([0.0010, 0.0500, 0.4508, 0.4982]),
 tensor([1.3973, 1.0000, 1.0530]))
```

The attention scores, softmax-normalized weights, and the resulting attention-context vector are as follows:

1. Attention Scores: $[-7.0, -3.1, -0.9, -0.8]$
2. Softmax-normalized Weights: $[0.0010, 0.0500, 0.4508, 0.4982]$
3. Attention-context Vector: $[1.3973, 1.0000, 1.0530]$

So, the attention-context vector that would be used as part of the input to the next layer or as part of the decoder's prediction for the current timestep is $[1.3973, 1.0000, 1.0530]$.

d: Transformers

Positional Encoding in Transformers:

Transformers implement positional encoding to retain the sequence information of input data. This step is critical because the transformer architecture doesn't have any recurrent or convolutional components, which typically handle order information in other types of neural networks.

Why it is done: The goal of positional encoding is to give the model insight into the order of the words or tokens, which is fundamental for understanding the meanings and grammatical relationships within sentences.

How it is done: Positional encodings use sine and cosine functions of different frequencies to encode the positions. From slides in week 8, we have that, for a given position pos (runs along the sequence) and dimension i (runs along input dimension), the positional encoding is defined as follows:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

where d_{model} is the dimension of the embeddings and the output vectors of the model, pos is the position of the token in the sequence, and i is the dimension. Each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. This allows the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

The positional encoding is added to the token embeddings before being fed into the transformer model, ensuring that the position information is always a part of the representation.

e: Bounding box detection:

Given the detection outcomes and prediction scores for a dataset with two classes, we follow the steps, given in slides of week 5, to calculate the mean Average Precision (mAP):

1. **Sorting by Confidence Score:** We organize the detection results and their corresponding confidence scores, sorting them in descending order based on the confidence score.
2. **Calculating Precision and Recall:** At each confidence score threshold, we calculate precision and recall. Precision is the number of true positives (TP) divided by the number of true positives plus the number of false positives (FP), while recall is the number of true positives divided by the total number of actual positives (total true cases) in the dataset.

   The formulas for precision (P) and recall (R) are:

   $$P = \frac{TP}{TP + FP}$$

   $$R = \frac{TP}{\text{Total True Positives}}$$
3. **Interpolating Precision:** We interpolate the precision for each recall level to ensure it is the maximum precision achieved for any recall higher than the current level.
4. **Calculating Average Precision (AP):** The AP is the area under the precision-recall curve, calculated using numerical integration:

   $$AP = \text{AUC} = \int_0^1 p(r)\, dr$$

   For practical computation, this integral can be approximated as:

$$AP \approx \sum (\text{Recall}_{i+1} - \text{Recall}_i) \times \text{Precision}_{\text{interpolated},i}$$

5. **Mean Average Precision (mAP):** The mAP is the average of the AP values for all classes. With two classes (cats and dogs), it's the average of their respective APs:

$$mAP = \frac{AP_{\text{cats}} + AP_{\text{dogs}}}{2}$$

Results:

Here are the organized detections with their respective precision and recall values for cats and dogs:

Cats:

#	Confidence	TP/FP	Precision	Recall
1	0.70	TP	$1/1$	$1/3$
2	0.60	FP	$1/2$	$1/3$
3	0.55	TP	$2/3$	$2/3$
4	0.50	TP	$3/4$	$3/3$
5	0.30	FP	$3/5$	$3/3$

Where we get that $AP_{\text{cats}} = 5/6$

Dogs:

#	Confidence	TP/FP	Precision	Recall
1	0.95	TP	$1/1$	$1/4$
2	0.70	TP	$2/2$	$2/4$
3	0.60	TP	$3/3$	$3/4$
4	0.50	FP	$3/4$	$3/4$
5	0.40	FP	$3/5$	$3/4$
6	0.35	TP	$4/6$	$3/4$

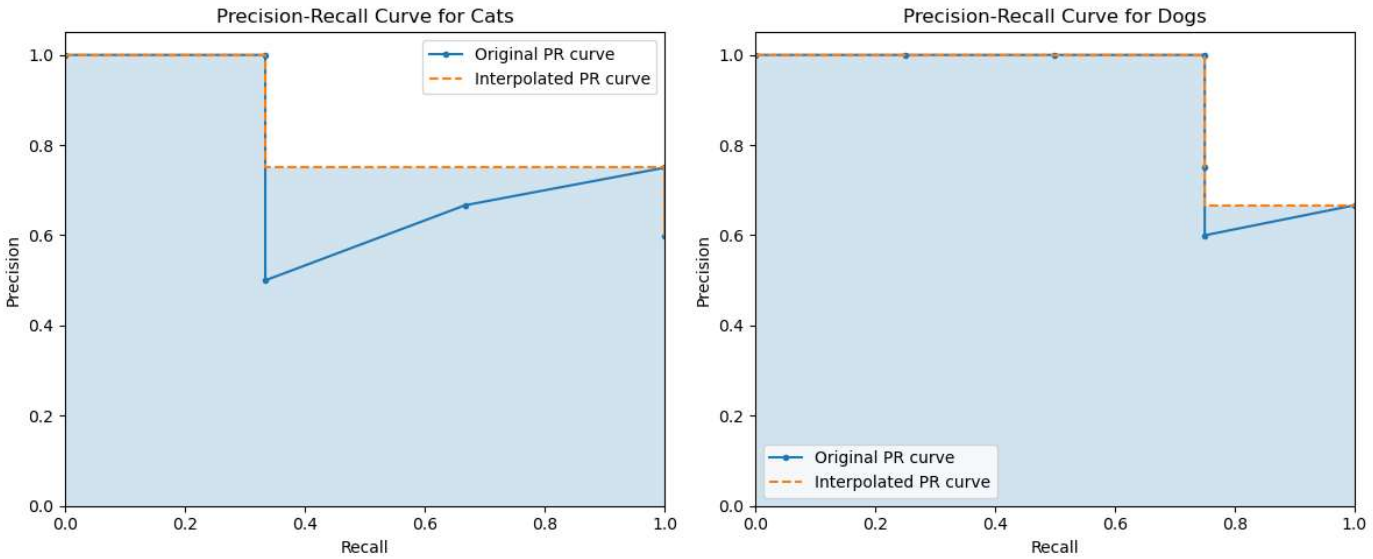
Where we get that $AP_{\text{dogs}} = 11/12$

Thus giving us $mAP = (5/6 + 11/12)/2 = 7/8 = 0.875

To verify this, we will plot the Precision-Recall curves below, and use the sklearn AUC function to verify the results.

In [ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import auc as sklearn_auc

# Correct the complete code for handling everything
def calculate_precision_recall(detections, scores, total_true):
    data = pd.DataFrame({'TP/FP': detections, 'Confidence': scores})
    data = data.sort_values('Confidence', ascending=False).reset_index(drop=True)
    tp = fp = 0
    precision = []
    recall = []
    for index, row in data.iterrows():
        if row['TP/FP'] == 'TP':
            tp += 1
        else:
            fp += 1
        precision.append(tp / (tp + fp))
        recall.append(tp / total_true)
    precision.insert(0, 1.0)  # Precision starts at 1.0 with recall at 0
    recall.insert(0, 0.0)     # Recall starts at 0
    return data, precision, recall

def interpolate_precision_recall(precision):
    # Interpolate precision by taking the maximum precision to the right for each recall level
    for i in range(len(precision)-2, -1, -1):
        precision[i] = max(precision[i], precision[i+1])
    return precision

def plot_precision_recall_curve(recall, precision, interpolated_precision, title):
    plt.plot(recall, precision, label='Original PR curve', linestyle='-', marker='.')
    plt.plot(recall, interpolated_precision, label='Interpolated PR curve', linestyle='--')
    plt.fill_between(recall, 0, interpolated_precision, alpha=0.2)
```

```
    plt.title(title)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.legend()

def compute_map(cat_detections, cat_scores, cat_total, dog_detections, dog_scores, dog_total):
    # Calculate precision and recall for cats and dogs
    cat_data, cat_precision, cat_recall = calculate_precision_recall(cat_detections, cat_scores, cat_total)
    dog_data, dog_precision, dog_recall = calculate_precision_recall(dog_detections, dog_scores, dog_total)

    # Interpolate precision for recall
    cat_interpolated_precision = interpolate_precision_recall(cat_precision.copy())
    dog_interpolated_precision = interpolate_precision_recall(dog_precision.copy())

    # Calculate AUC for interpolated precision-recall
    cat_auc = sklearn_auc(cat_recall, cat_interpolated_precision)
    dog_auc = sklearn_auc(dog_recall, dog_interpolated_precision)

    # Plotting
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plot_precision_recall_curve(cat_recall, cat_precision, cat_interpolated_precision,
                                'Precision-Recall Curve for Cats')

    plt.subplot(1, 2, 2)
    plot_precision_recall_curve(dog_recall, dog_precision, dog_interpolated_precision,
                                'Precision-Recall Curve for Dogs')

    plt.tight_layout()
    plt.show()

    # Calculate mAP
    mAP = (cat_auc + dog_auc) / 2
    return mAP, cat_auc, dog_auc

# Input data
cat_detections = ['TP', 'FP', 'TP', 'FP', 'TP']
cat_scores = [0.7, 0.3, 0.5, 0.6, 0.55]
cat_total = 3

dog_detections = ['FP', 'TP', 'TP', 'FP', 'TP', 'TP']
dog_scores = [0.4, 0.35, 0.95, 0.5, 0.6, 0.7]
dog_total = 4

# Compute mAP
mAP, cat_auc, dog_auc = compute_map(cat_detections, cat_scores, cat_total, dog_detections,
                                    dog_scores, dog_total)

# Output results
print(f"Average Precision (AP) for Cats: {cat_auc:.3f}")
print(f"Average Precision (AP) for Dogs: {dog_auc:.3f}")
print(f"Mean Average Precision (mAP): {mAP:.3f}")
```




```
Mean Average Precision (mAP): 0.875
Average Precision (AP) for Cats: 0.833
Average Precision (AP) for Dogs: 0.917
```

f: Semantic segmentation - FCN 1:

In the context of semantic segmentation using one-hot encoding, the size of the target image is determined by the spatial dimensions of the input image and the number of classes. Here's a general formula to calculate the size of the one-hot encoded target image:

Given:

- Input image dimensions: $W \times H \times C$, where W is width, H is height, and C is the number of color channels (typically 3 for RGB images).
- Number of classes for segmentation: N.

The one-hot encoded target image dimensions will be: $W \times H \times N$.

In this case:

- Input image dimensions: $1024 \times 768 \times 3$.
- Number of classes: 7.

Applying the formula, the target image dimensions will be: $1024 \times 768 \times 7$. The color channels C are not used in the target image size calculation since one-hot encoding replaces the channel depth with the number of classes.

g: Semantic segmentation - FCN 2:

A Fully Convolutional Network (FCN) is a neural network architecture that is composed entirely of convolutional layers and is typically used for semantic segmentation, which involves assigning a class to each pixel of the image. FCNs can handle inputs of any size and produce output maps that match the input dimensions in resolution, making them ideal for tasks where detailed spatial understanding and pixel-level accuracy are crucial, such as in medical imaging, autonomous driving, and land cover classification in satellite imagery.

h:Residual Networks:

Residual layers are a key innovation in Residual Networks (ResNets), where they enable the training of very deep networks by adding shortcut connections that skip one or more layers. These shortcuts, or "residual connections," allow the network to learn the residual of the target function (the difference between the input and the output), rather than having to learn the full mapping.

The primary advantage of residual layers is that they provide a path for gradients to flow through during the training process, which helps to address the vanishing gradient problem that often plagues deep neural networks. This is because the gradients can be propagated back through the shortcuts, maintaining their magnitude and preventing them from becoming too small.

Additionally, residual layers allow a network to learn an identity mapping easily, meaning if the optimal function is close to the identity, the network can push the weights towards zero to approximate it. This ensures that deeper layers can at least perform as well as shallower ones, preventing the performance degradation that can occur when additional layers are added to a network.

By leveraging residual layers, ResNets can efficiently train networks with a much larger number of layers, leading to improved learning of complex patterns and ultimately better performance on a wide array of tasks in computer vision and beyond.

i: Intersection-Over-Union

In order to calculate the Intersection-Over-Union (IoU) of the bounding-boxes on the 4 grids above, we will be using the formula provided in the slides of week 5:

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Thus giving us:

- A

$$IoU = \frac{4}{14} = 0.286$$

- B

$$IoU = \frac{9}{25} = 0.36$$

- C

$$IoU = \frac{6}{13} = 0.462$$

- D

$$IoU = \frac{6}{12} = 0.5$$

j: Variational autoencoders:

Variational Autoencoders (VAEs) offer a generative approach to modeling data, surpassing traditional Autoencoders (AEs) by mapping inputs to a probabilistic latent space. This allows VAEs not only to encode but also to generate new, similar data by sampling from this space. Their built-in regularization helps prevent overfitting, promoting better generalization to new data. VAEs are particularly advantageous for tasks that require the generation of new content, as they can smoothly interpolate between data points and capture complex underlying data distributions.

Problem 2

Below is attached a script for generating a data set to learn translation of dates between multiple human readable and a machine readable format (ISO 8601).

Task: Using an encoder-decoder setup, perform translation from human readable to machine readable formats. Please express the performance of your trained network in terms of average accuracy of the translated output (so, accuracy on a per-character basis).

Restriction: we specifically demand that your presented solution does not include an attention layer.

Despite this restriction, the task can be solved in numerous different ways. Here are some examples of solutions of similar problems, for inspiration:

https://jscriptcoder.github.io/date-translator/Machine%20Translation%20with%20Attention%20model.html

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Script for generating data set:

date_trans

In [ ]:
```
# Required libraries
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import time
import math
import random
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from torch.utils.data import DataLoader
from functions.dateTrans import datesDataset  # Ensure this import works
%matplotlib inline

# Set the device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [ ]:
```
# Prepare the dataset
dataset = datesDataset()
human_vocab = dataset.humanVocab
machine_vocab = dataset.machineVocab
max_length = dataset.outputLength
```

In [ ]:
```
# Assuming the <SOS> and <EOS> symbols are part of your vocabularies
SOS_token = human_vocab['<SOS>']
EOS_token = human_vocab['<EOS>']
```

In [ ]:
```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size, dropout_p=0.1):
        super(EncoderRNN, self).__init__()
```

```
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, input):
        embedded = self.dropout(self.embedding(input))
        output, hidden = self.gru(embedded)
        return output, hidden
```

In [ ]:

```
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None):
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, device=device).fill_(SOS_token)
        decoder_hidden = encoder_hidden
        decoder_outputs = []

        for i in range(max_length):
            decoder_output, decoder_hidden = self.forward_step(decoder_input, decoder_hidden)
            decoder_outputs.append(decoder_output)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teacher forcing
            else:
                # Without teacher forcing: use its own predictions as the next input
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from history as input

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        return decoder_outputs, decoder_hidden, None # We return `None` for consistency in the training loop

    def forward_step(self, input, hidden):
        output = self.embedding(input)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.out(output)
        return output, hidden
```

In [ ]:

```
def train_epoch(dataloader, encoder, decoder, encoder_optimizer,
          decoder_optimizer, criterion):

    total_loss = 0
    for data in dataloader:
        _, _, input_tensor, target_tensor = data

        # Move tensors to the correct device
        input_tensor = input_tensor.to(device)
        target_tensor = target_tensor.to(device)

        encoder_optimizer.zero_grad()
        decoder_optimizer.zero_grad()

        encoder_outputs, encoder_hidden = encoder(input_tensor)
        decoder_outputs, _, _ = decoder(encoder_outputs, encoder_hidden, target_tensor)

        loss = criterion(
            decoder_outputs.view(-1, decoder_outputs.size(-1)),
            target_tensor.view(-1)
        )
        loss.backward()

        encoder_optimizer.step()
        decoder_optimizer.step()

        total_loss += loss.item()

    return total_loss / len(dataloader)
```

In [ ]:

```
def asMinutes(s):
    m = math.floor(s / 60)
    s -= m * 60
```

```
    return '%dm %ds' % (m, s)

def timeSince(since, percent):
    now = time.time()
    s = now - since
    es = s / (percent)
    rs = es - s
    return '%s (- %s)' % (asMinutes(s), asMinutes(rs))


# Plotting results
def showPlot(points):
    plt.figure()
    fig, ax = plt.subplots()
    # this locator puts ticks at regular intervals
    loc = ticker.MultipleLocator(base=0.2)
    ax.yaxis.set_major_locator(loc)
    plt.plot(points)
```

In [ ]:

```
def train(train_dataloader, encoder, decoder, n_epochs, learning_rate=0.001,
                print_every=100, plot_every=100):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.Adam(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.Adam(decoder.parameters(), lr=learning_rate)
    criterion = nn.NLLLoss()

    for epoch in range(1, n_epochs + 1):
        loss = train_epoch(train_dataloader, encoder, decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss

        if epoch % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            print('%s (%d %d%%) %.4f' % (timeSince(start, epoch / n_epochs),
                                         epoch, epoch / n_epochs * 100, print_loss_avg))

        if epoch % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_losses.append(plot_loss_avg)
            plot_loss_total = 0

    showPlot(plot_losses)
```

In [ ]:

```
def evaluate(encoder, decoder, human_readable, human_vocab, max_length):
    with torch.no_grad():
        # Prepare the input tensor
        input_tensor = torch.LongTensor(dataset.string_to_int(human_readable,
                                                              max_length, human_vocab)).unsqueeze(0).to(device)

        # Forward pass through encoder
        encoder_outputs, encoder_hidden = encoder(input_tensor)

        # Forward pass through decoder
        decoder_outputs, decoder_hidden, _ = decoder(encoder_outputs, encoder_hidden)

        # Take the top one word from the output
        _, topi = decoder_outputs.topk(1)
        decoded_ids = topi.squeeze().detach().cpu().numpy()

        # Convert the indices to the machine readable date
        decoded_words = [dataset.invMachineVocab[idx] for idx in decoded_ids if idx not in
                         (dataset.humanVocab['<SOS>'], dataset.humanVocab['<EOS>'], dataset.humanVocab['<pad>'])]

    return ''.join(decoded_words)
```

In [ ]:

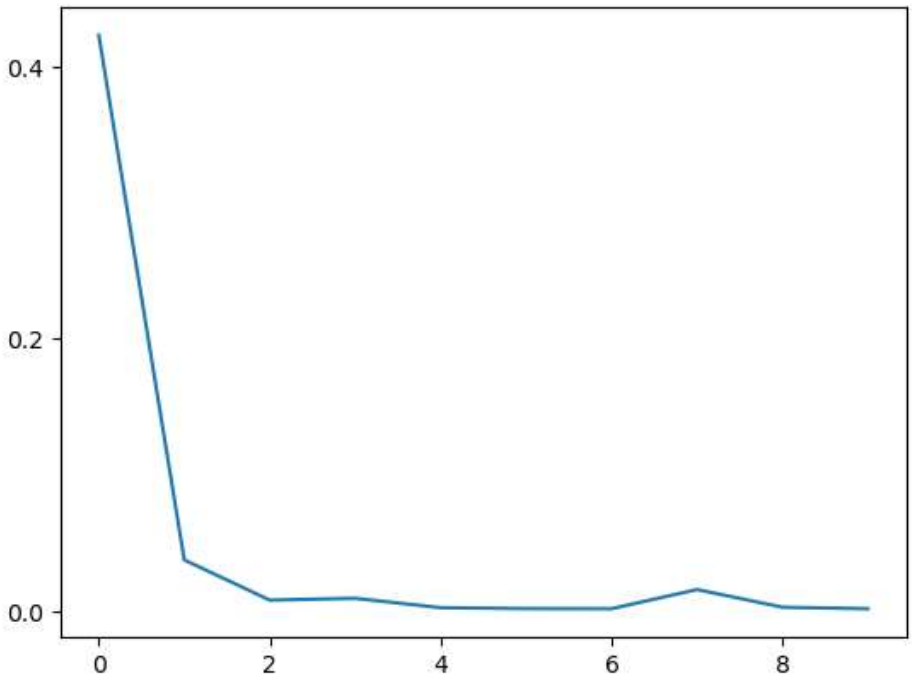
```
def evaluateRandomly(encoder, decoder, dataset, n=10):
    for i in range(n):
        sample = random.choice([dataset[i] for i in range(len(dataset))])
        human_readable, machine_readable, _, _ = sample

        print('>', human_readable)
        print('=', machine_readable)

        output_sentence = evaluate(encoder, decoder, human_readable, dataset.humanVocab, dataset.inputLength)
```

```
        print('<', output_sentence)
        print('')
```

In [ ]:
```
def compute_accuracy(encoder, decoder, dataset):
    total_chars = 0
    correct_chars = 0
    for i in range(len(dataset)):
        human_readable, machine_readable, _, _ = dataset[i]
        prediction = evaluate(encoder, decoder, human_readable, dataset.humanVocab, dataset.inputLength)

        # Ensure that the lengths of prediction and machine_readable are the same for character-wise comparison
        min_length = min(len(prediction), len(machine_readable))
        total_chars += min_length
        correct_chars += sum(1 for j in range(min_length) if prediction[j] == machine_readable[j])

        # If you also want to count the missing or extra characters as incorrect
        total_chars += abs(len(prediction) - len(machine_readable))

    accuracy = correct_chars / total_chars if total_chars > 0 else 0
    return accuracy
```

In [ ]:
```
hidden_size = 1024
batch_size_use = 64
nsim = 2000

from faker import Faker
fake = Faker()

Faker.seed(101)
random.seed(101)

# DataLoader to handle batching
train_dataloader = DataLoader(dataset, batch_size=batch_size_use, shuffle=True)

encoder = EncoderRNN(len(human_vocab), hidden_size).to(device)
decoder = DecoderRNN(hidden_size, len(machine_vocab)).to(device)

train(train_dataloader, encoder, decoder, n_epochs=nsim, print_every=(nsim/10), plot_every=(nsim/10))
```

```
0m 16s (- 2m 31s) (200 10%) 0.4229
0m 33s (- 2m 13s) (400 20%) 0.0377
0m 49s (- 1m 55s) (600 30%) 0.0082
1m 5s (- 1m 38s) (800 40%) 0.0095
1m 21s (- 1m 21s) (1000 50%) 0.0026
1m 38s (- 1m 5s) (1200 60%) 0.0020
1m 54s (- 0m 49s) (1400 70%) 0.0019
2m 11s (- 0m 32s) (1600 80%) 0.0159
2m 27s (- 0m 16s) (1800 90%) 0.0030
2m 44s (- 0m 0s) (2000 100%) 0.0018
<Figure size 640x480 with 0 Axes>
```



In [ ]:
```
encoder.eval()
decoder.eval()
evaluateRandomly(encoder, decoder, dataset, n=7)
```

```
> mandag den 3. januar 1972
= 1972-01-03
< 1972-01-03

> 18/05/2003
= 2003-05-18
< 2003-05-18

> onsdag 12 juni 2019
= 2019-06-12
< 2019-06-12

> 03/07/1994
= 1994-07-03
< 1994-07-03

> 23/12/2014
= 2014-12-23
< 2014-12-23

> 09/08/1984
= 1984-08-09
< 1984-08-09

> 04.03.1970
= 1970-03-04
< 1970-03-04
```

In [ ]:

```python
Faker.seed(101)
random.seed(101)

accuracy = compute_accuracy(encoder, decoder, dataset) * 100
print(f"Model accuracy: {accuracy:.2f}%")
```

```
Model accuracy: 100.00%
```