

I. Char-Level Model

0. Configuration

```
# Configuration
max_len_en = 50
max_len_fr = 50
hidden_size = 256
num_layers = 5
learning_rate = 0.0001
epochs = 10000
batch_size = 8
teacher_forcing_ratio = 0.5
```

Je prends 100 lignes du fichier de données.

1. Encoder

```
class EncoderRNN(nn.Module): 2 usages
    def __init__(self, input_size, hidden_size, num_layers=5):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, num_layers=num_layers, batch_first=True)

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

    def init_hidden(self, batch_size): 10 usages (10 dynamic)
        return torch.zeros(self.num_layers, batch_size, self.hidden_size, device=device)
```

L'encodeur prend une séquence d'indices de caractères en entrée et la passe à travers une couche d'embedding, puis dans un GRU pour obtenir l'état caché.

2. Decoder

```
# Decoder
class DecoderRNN(nn.Module): 2 usages
    def __init__(self, hidden_size, output_size, num_layers=5):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, num_layers=num_layers, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input)
        output, hidden = self.gru(embedded, hidden)
        output = output.squeeze(1)
        output = self.out(output)

        return output, hidden
```

Le décodeur prend en entrée un token (caractère) et un état caché, puis génère la probabilité de chaque token possible en sortie. Il utilise aussi un GRU pour traiter les données séquentielles.

3. Traduction

```
def translate(encoder, decoder, input_tensor, char_to_index_fr, index_to_char_fr): 1 usage

    with torch.no_grad(): # Disable gradient computation
        input_tensor = input_tensor.to(device)
        batch_size = input_tensor.size(0)
        encoder_hidden = encoder.init_hidden(batch_size)

        encoder_outputs, encoder_hidden = encoder(input_tensor, encoder_hidden)

        decoder_input = torch.tensor([SOS_IDX], device=device).unsqueeze(1)
        decoder_hidden = encoder_hidden

        translated_seq = []
        for _ in range(max_len_fr):
            decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
            top1 = decoder_output.argmax(1).item()
            if top1 == EOS_IDX:
                break
            translated_seq.append(index_to_char_fr[top1])
            decoder_input = torch.tensor([top1], device=device)

        return ''.join(translated_seq)
```

Cette fonction prend un tensor d'entrée et traduit cette séquence vers le français en générant un caractère à la fois jusqu'à ce que le token EOS soit généré.

4. Résultats

```
10/13 [00:00<00:00, 13.05it/s]Epoch 1504, Step 10, Loss: 0.0002, Top-1 Accuracy: 0.8975, Top-5 Accuracy: 0.8975
> (EN): <bos>observatoriesdr.jamestoldervyoffredericton
ice (FR): <bos>obseratoiresesdl'obseratoiresduddr..jamesttolderv
13/13 [00:00<00:00, 13.36it/s]
Age Loss: 0.0003, Average Top-1 Accuracy: 0.8904, Average Top-5 Accuracy: 0.8904
10/13 [00:00<00:00, 13.04it/s]Epoch 1505, Step 10, Loss: 0.0003, Top-1 Accuracy: 0.9000, Top-5 Accuracy: 0.9000
> (EN): <bos>links<eos><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
ice (FR): <bos>liens<eos><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
13/13 [00:00<00:00, 13.32it/s]
0/13 [00:00<?, ?it/s]Epoch 1505, Average Loss: 0.0003, Average Top-1 Accuracy: 0.8898, Average Top-5 Accuracy: 0.8898
10/13 [00:00<00:00, 13.05it/s]Epoch 1506, Step 10, Loss: 0.0003, Top-1 Accuracy: 0.8975, Top-5 Accuracy: 0.8975
> (EN): <bos>astronomesjohnstanleyplaskettisborn.<eos><pad><pad><pad><pad><pad><pad><pad>
ice (FR): <bos>astronomesanaissanceddesjohtntstanleypplaskett.<eos><pad><pad>
13/13 [00:00<00:00, 13.34it/s]
Age Loss: 0.0003, Average Top-1 Accuracy: 0.8900, Average Top-5 Accuracy: 0.8900
```

On peut voir que le modèle n'évolue plus et à des résultats convenables. Les petits mots sont bien traduits et les phrases aussi. On obtient une accuracy de 90% et une loss de 0.0003 et ces valeurs ne descendent pas plus.

II. RNN avec SentencePiece

0. Configuration


```
# Configuration
max_len_en = 10
max_len_fr = 10
hidden_size = 512
num_layers = 10
learning_rate = 0.0001
epochs = 10000
batch_size = 256
teacher_forcing_ratio = 0.5
```

Je prends également 1000 lignes du fichier de données.

1. Entrainement tokenizers

```
import sentencepiece as spm

def train_tokenizer(file_path, vocab_size, name): 3 usages
    spm.SentencePieceTrainer.Train(
        input=file_path,
        model_prefix=name,
        vocab_size=vocab_size,
        character_coverage=1.0,
        user_defined_symbols=["<s>", "<pad>", "</s>"],
        model_type='bpe'
    )

def load_tokenizer(model_name): 12 usages
    tokenizer = spm.SentencePieceProcessor()
    tokenizer.Load(model_name)
     return tokenizer
```

J'ai d'abord créé 2 fichiers .txt de 100 000 lignes chacun, l'un avec les phrases en anglais l'autre en français. Ensuite, j'entraîne 2 tokenizers, l'un avec le fichier anglais l'autre avec le français. Cela me donne comme résultats 2 fichiers .model et 2 fichiers .vocab, puis j'utilise la fonction load_tokenizer pour charger les modèles et récupérer mon objet tokenizer.

2. Résultats obtenus

```
Epoch 2365, Average Loss: 0.0812, Average Accuracy: 0.9978
100%|██████████| 4/4 [00:00<00:00, 13.69it/s]
Epoch 2365, Step 4, Loss: 0.0812, Accuracy: 0.9978
Original Sentence (EN): _racine_specializes_in_the_study_of_g
Translated Sentence (FR): racine se spécialise dans l'étude
Epoch 2365, Average Loss: 0.0831, Average Accuracy: 0.9960
100%|██████████| 4/4 [00:00<00:00, 13.71it/s]
Epoch 2366, Step 4, Loss: 0.0765, Accuracy: 0.9983
Original Sentence (EN): _the_installation_of_the_telescope_was_largely_the
Translated Sentence (FR): la conception du télescope est en grande partie l'
Epoch 2366, Average Loss: 0.0838, Average Accuracy: 0.9951
100%|██████████| 4/4 [00:00<00:00, 13.70it/s]
Epoch 2367, Step 4, Loss: 0.0867, Accuracy: 0.9953
Original Sentence (EN): _alexander_rothney_cross_signed_on_to
Translated Sentence (FR): alexander rothney cross se
Epoch 2367, Average Loss: 0.0829, Average Accuracy: 0.9959
Epoch 2367, Step 4, Loss: 0.0829, Accuracy: 0.9959
```

On peut voir que les résultats obtenus sont meilleurs qu'avec le tokenizer classique, l'accuracy est presque de 100% et les phrases sont bien traduites.

III. RNN from scratch

0. Configuration

```
# Configuration
max_len_en = 10
max_len_fr = 10
hidden_size = 512
num_layers = 10
learning_rate = 0.0001
epochs = 10000
batch_size = 256
teacher_forcing_ratio = 0.5
```

Je prends 1000 lignes du jeu de données.

1. Encoder avec RNNCell

```
class EncoderSimpleRNN(nn.Module): 2 usages
    def __init__(self, input_size, hidden_size, num_layers=5):
        super(EncoderSimpleRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.rnn_cell = nn.RNNCell(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input)
        outputs = []
        for t in range(input.size(1)):
            hidden = self.rnn_cell(embedded[:, t, :], hidden)
            outputs.append(hidden)

        return torch.stack(outputs, dim=1), hidden

    def init_hidden(self, batch_size): 10 usages (10 dynamic)
        return torch.zeros(batch_size, self.hidden_size, device=device)
```

L'encodeur prend une séquence d'entrées, la transforme en une série d'états cachés, et la résume dans l'état caché final pour l'utiliser dans le décodeur.

2. Decoder avec RNNCell

```
class DecoderSimpleRNN(nn.Module): 2 usages
    def __init__(self, hidden_size, output_size, num_layers=5):
        super(DecoderSimpleRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.rnn_cell = nn.RNNCell(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).squeeze(1)

        hidden = self.rnn_cell(embedded, hidden)

        output = self.out(hidden)
        return output, hidden
```

Le décodeur génère des prédictions séquentielles à partir de l'état caché fourni, générant chaque élément de la séquence de sortie.

3. Résultats obtenus

```
Epoch 8888, Step 4, Loss: 0.0007, Accuracy: 1.0000
Original Sentence (EN): she received numerous awards for her work in astronomy
Translated Sentence (FR): elle a reçu de nombreux prix pour son travail en
Epoch 8888, Average Loss: 0.0017, Average Accuracy: 0.9992
100%|██████████| 4/4 [00:00<00:00, 47.18it/s]
Epoch 8889, Step 4, Loss: 0.0025, Accuracy: 0.9983
Original Sentence (EN): in addition to the numerous scientific articles that he w
Translated Sentence (FR): en plus des nombreux articles scientifiques qu'il ré
Epoch 8889, Average Loss: 0.0025, Average Accuracy: 0.9990
100%|██████████| 4/4 [00:00<00:00, 47.32it/s]
Epoch 8890, Step 4, Loss: 0.0018, Accuracy: 0.9987
Original Sentence (EN): it proved to be one of the first ver
Translated Sentence (FR): il s'agit de l'une des toutes
Epoch 8890, Average Loss: 0.0021, Average Accuracy: 0.9990
```

Les résultats sont similaires, on a une accuracy autour de 99,9% et une loss très faible et une traduction cohérente. Le modèle est donc efficace

4. Implémentation RNNCell

```
class MyRNNCell(nn.Module): 3 usages
    def __init__(self, input_size, hidden_size):
        super(MyRNNCell, self).__init__()
        self.hidden_size = hidden_size

        self.Wx = nn.Linear(input_size, hidden_size)
        self.Wh = nn.Linear(hidden_size, hidden_size)

    def forward(self, input, hidden):
        new_hidden = torch.tanh(self.Wx(input) + self.Wh(hidden))
        return new_hidden
```

Elle calcule le nouvel état caché en appliquant une fonction d'activation tanh à la somme des sorties des couches linéaires.

5. Encoder

```
class EncoderCustomRNN(nn.Module): 2 usages
    def __init__(self, input_size, hidden_size, num_layers=1):
        super(EncoderCustomRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(input_size, hidden_size)

        self.rnn_cells = nn.ModuleList([MyRNNCell(hidden_size, hidden_size) for _ in range(num_layers)])

    def forward(self, input, hidden):
        embedded = self.embedding(input) # (batch_size, seq_len, hidden_size)
        outputs = []

        for t in range(input.size(1)): # Process each timestep
            x_t = embedded[:, t, :] # Take one token at a time
            for layer in range(self.num_layers):
                hidden[layer] = self.rnn_cells[layer](x_t, hidden[layer]) # Pass through MyRNNCell
                x_t = hidden[layer] # Update input for next layer
            outputs.append(x_t)

        return torch.stack(outputs, dim=1), hidden
```

Chaque élément de la séquence d'entrée est d'abord transformé en un vecteur d'embedding. Ensuite, la séquence est traitée élément par élément. Pour chaque élément de la séquence, la sortie de chaque couche est calculée en passant l'entrée et l'état caché à travers MyRNNCell. Les états cachés de chaque couche sont mis à jour à chaque étape et stockés dans une liste outputs.

6. Decoder

```
class DecoderCustomRNN(nn.Module): 2 usages
    def __init__(self, hidden_size, output_size, num_layers=1):
        super(DecoderCustomRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(output_size, hidden_size)

        # Initialize MyRNNCell for each layer
        self.rnn_cells = nn.ModuleList([MyRNNCell(hidden_size, hidden_size) for _ in range(num_layers)])

        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).squeeze(1) # (batch_size, hidden_size)

        x_t = embedded
        for layer in range(self.num_layers):
            hidden[layer] = self.rnn_cells[layer](x_t, hidden[layer]) # Pass through MyRNNCell
            x_t = hidden[layer] # Update the input for the next layer

        output = self.out(x_t) # (batch_size, output_size)
        return output, hidden
```

L'entrée est d'abord transformée en un vecteur d'embedding. Comme pour l'encodeur, chaque couche du décodeur prend l'entrée et l'état caché, puis met à jour l'état caché. À la fin, l'état caché final est transformé en une prédiction par la couche linéaire self.out.

7. Initialisation des poids

```
def init_weights(m): 2 usages
    if isinstance(m, nn.Linear) or isinstance(m, nn.Embedding):
        nn.init.xavier_uniform_(m.weight)
        if hasattr(m, 'bias') and m.bias is not None:
            nn.init.zeros_(m.bias)
```

Initialisation uniforme des poids selon la méthode Xavier. Les biais sont initialisés à zéro si présents.

8. Résultats obtenus

```
Original Sentence (EN): _belonging_to_the_university_of_calg
Translated Sentence (FR): propriété de l'université de calgary
Epoch 9985, Average Loss: 0.0027, Average Accuracy: 0.9987
100%|██████████| 4/4 [00:00<00:00, 21.43it/s]
Epoch 9986, Step 4, Loss: 0.0014, Accuracy: 0.9996
Original Sentence (EN): _at_that_point,_he_moved_to_ottawa
Translated Sentence (FR): la même année, il déménage à
Epoch 9986, Average Loss: 0.0028, Average Accuracy: 0.9987
100%|██████████| 4/4 [00:00<00:00, 21.07it/s]
Epoch 9987, Step 4, Loss: 0.0022, Accuracy: 0.9983
Original Sentence (EN): _in_1948,_underhill_completed_postd
Translated Sentence (FR): en 1948, underhill effective
Epoch 9987, Average Loss: 0.0020, Average Accuracy: 0.9989
100%|██████████| 4/4 [00:00<00:00, 21.39it/s]
Epoch 9988, Step 4, Loss: 0.0044, Accuracy: 0.9978
Original Sentence (EN): _the_year_1997_was_a_particularly_significant_year_for_m
Translated Sentence (FR): l'année 1997 est particulièrement importante pour lui
Epoch 9988, Average Loss: 0.0019, Average Accuracy: 0.9990
100%|██████████| 4/4 [00:00<00:00, 21.36it/s]
```


Les résultats obtenus sont similaires aux autres et montrent que le modèle traduit bien les phrases.

IV. Top-p sampling

1. Fonction de sampling

```
def top_p_sampling(logits, p=0.5): 1 usage

    sorted_logits, sorted_indices = torch.sort(logits, descending=True, dim=-1)

    # Compute the cumulative probabilities
    cumulative_probs = torch.cumsum(F.softmax(sorted_logits, dim=-1), dim=-1)

    # Create a mask for top-p filtering (set values below the threshold to zero)
    sorted_indices_to_remove = cumulative_probs > p
    sorted_indices_to_remove[..., 1:] = sorted_indices_to_remove[..., :-1].clone() # Fix: clone the tensor
    sorted_indices_to_remove[..., 0] = 0 # Keep the first token
    indices_to_remove = sorted_indices_to_remove.scatter(1, sorted_indices, sorted_indices_to_remove)

    # Set the probabilities of the tokens we don't want to consider to zero
    logits[indices_to_remove] = -float('Inf')

    # Sample from the remaining logits
    probabilities = F.softmax(logits, dim=-1)
    return torch.multinomial(probabilities, num_samples=1)
```

La fonction effectue un échantillonnage basé sur un seuil ppp de probabilité cumulée. Seuls les tokens dont la probabilité cumulée est inférieure ou égale à ppp sont considérés pour l'échantillonnage, ce qui permet de limiter l'espace de recherche aux tokens les plus probables tout en excluant ceux qui ont une probabilité cumulée trop faible.

2. Résultats obtenus

```
Epoch 4884, Step 4, Loss: 0.1500, Accuracy: 0.9927
Original Sentence (EN): _the_idea_to_construct_the_two_observatories_ar
Translated Sentence (FR): le projet de construire les observatoires naît
Epoch 4884, Average Loss: 0.1375, Average Accuracy: 0.9912
100%|██████████| 4/4 [00:00<00:00, 32.23it/s]
Epoch 4885, Step 4, Loss: 0.1310, Accuracy: 0.9931
Original Sentence (EN): _the_federal_and_provincial_governments_approved_the_project_in_19
Translated Sentence (FR): le projet est approuvé par les gouvernements fédéral et
Epoch 4885, Average Loss: 0.1362, Average Accuracy: 0.9911
100%|██████████| 4/4 [00:00<00:00, 32.00it/s]
Epoch 4886, Step 4, Loss: 0.1464, Accuracy: 0.9866
Original Sentence (EN): _the_installation_of_the_telescope_was_largely_the
Translated Sentence (FR): la conception du télescope est en grande partie l'
Epoch 4886, Average Loss: 0.1367, Average Accuracy: 0.9909
100%|██████████| 4/4 [00:00<00:00, 32.16it/s]
Epoch 4887, Step 4, Loss: 0.1284, Accuracy: 0.9931
Original Sentence (EN): _andrew_mckellar_one_of_the
Translated Sentence (FR): andrew mckellar, qui est
Epoch 4887, Average Loss: 0.1340, Average Accuracy: 0.9916
```

On obtient également de bons résultats avec la méthode de top-p sampling, une accuracy de 99% et une loss de 0,1 environ et le modèle traduit correctement.