Lab CNN:

- Carlos Jarrin
- Fausto Yugcha

```
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
```

```
!pip install nltk
import nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2024.5.15)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.5)
```

```
from nltk.stem.snowball import SnowballStemmer
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
```

```
from collections import Counter
```

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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Load and Data pre process

```
text = open('/content/drive/MyDrive/NLP/datos/gabriel_garcia_marquez_cien_annos_soledad.txt', 'r').read().lower()
```

```
nltk.download('stopwords')
ss = SnowballStemmer('spanish')
stpw = stopwords.words('spanish')
tokenizer = RegexpTokenizer(r'\w+')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def split_tokens(text):
    tokens = tokenizer.tokenize(text)
    tokens = [w for w in tokens if w not in stpw]
    return tokens
```

```
tokens = split_tokens(text)
print('len tokens:', len(tokens))
print(tokens[:5])
```

```
len tokens: 70255
['gabriel', 'garcía', 'márquez', 'cien', 'años']
```

```
counts = Counter(tokens)
print(counts.most_common(5))
```

```
[('aureliano', 794), ('úrsula', 514), ('arcadio', 480), ('casa', 463), ('josé', 424)]
```

```
counts_more_than_1 = {k:v for k,v in counts.items() if v > 1}
vocab = list(counts_more_than_1.keys())
itot = dict(enumerate(vocab))
ttoi = {v:k for k,v in itot.items()}
```

```
# Token mayores a 1
tokens_more_than_1 = [w for w in tokens if w in vocab]
print('len tokens mayores a 1:', len(tokens_more_than_1))
```

```
len tokens mayores a 1: 61989
```

```
def window(tokens, win=2):
    output = []
    for i, w in enumerate(tokens):
        target = ttoi[w]
        window = [tokens[i+j] for j in range(-win, win+1,1)
                  if (i+j >= 0) & (i+j < len(tokens)) & (j != 0)]

        output += [(target, ttoi[j]) for j in window]
    return output
```

```
class text_dataset(Dataset):
    def __init__(self, data_windowed, vocab_size):
        self.data = data_windowed
        self.vocab_size = vocab_size

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        x0, y = self.data[idx]
        x =  F.one_hot(torch.tensor(x0), num_classes=self.vocab_size)
        return x, y
```

```
class Word2Vec_CNN(nn.Module):
    def __init__(self, vocab_size, embedding_size, num_filters, filter_sizes):
        super(Word2Vec_CNN, self).__init__()
        self.embed = nn.Embedding(vocab_size, embedding_size)

        # Convolutional layers with padding
        self.convs = nn.ModuleList([
            nn.Conv1d(in_channels=embedding_size,
                      out_channels=num_filters,
                      kernel_size=fs,
                      padding=fs // 2)
            for fs in filter_sizes
        ])

        # Fully connected layer
        self.fc = nn.Linear(num_filters * len(filter_sizes), vocab_size, bias=False)

    def forward(self, input):
        # Embedding layer
        embedded = self.embed(input.to(torch.int64))

        if embedded.dim() == 2:
            embedded = embedded.unsqueeze(1)

        # Reshape for Conv1d
        embedded = embedded.permute(0, 2, 1)

        # Convolution + ReLU + Max Pooling
        conv_outputs = [F.relu(conv(embedded)) for conv in self.convs]

        # Global max pooling
        pooled_outputs = [F.max_pool1d(conv_out, conv_out.shape[2]).squeeze(2) for conv_out in conv_outputs]
        concat_output = torch.cat(pooled_outputs, dim=1)

        logits = self.fc(concat_output)

        return logits
```

```
vocab_size = len(vocab)
embedding_size = 200
num_filters = 1000
filter_sizes = [2, 3, 4, 5]
```

```
model = Word2Vec_CNN(vocab_size, embedding_size, num_filters, filter_sizes)
model.to(device)
model = model.to(device)
```

```
data_windowed = window(tokens_more_than_1, win=4)
```

```
dataset = text_dataset(data_windowed, len(vocab))
```

```
Learning_rate = 3e-3
EPOCHS = 25
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=Learning_rate)

dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
running_loss = []
running_acc = []

for epoch in range(EPOCHS):
    epoch_loss = 0
    train_accuracy = 0
    total_samples = 0
    for center, context in dataloader:
        context = context.to(torch.float).to(device)
        center = center.to(torch.float).to(device)
        optimizer.zero_grad()
        logits = model(input=context)
        loss = loss_fn(logits, center)
        epoch_loss += loss.item()
        loss.backward()
        optimizer.step()
        _, predicted = torch.max(logits.data, 1)
        train_accuracy += (predicted==torch.max(center, 1)[1]).sum().item()
        total_samples += center.size(0)

    train_accuracy /= total_samples
    epoch_loss /= len(dataloader)
    print(f"Epoch:  {epoch+1}/{EPOCHS},  Loss: {epoch_loss},  Accuracy: {train_accuracy}")
    running_loss.append(epoch_loss)
    running_acc.append(train_accuracy)
```

```
Epoch:  1/25,  Loss: 8.496338590650748,  Accuracy: 0.012172005194679487
Epoch:  2/25,  Loss: 7.996065663436041,  Accuracy: 0.013337581570180604
Epoch:  3/25,  Loss: 7.954239573905139,  Accuracy: 0.013287167367087994
Epoch:  4/25,  Loss: 7.9452719009681,  Accuracy: 0.013115759076573124
Epoch:  5/25,  Loss: 7.940053597006094,  Accuracy: 0.013414211158881369
Epoch:  6/25,  Loss: 7.932851924667329,  Accuracy: 0.01398893307413711
Epoch:  7/25,  Loss: 7.932833326725917,  Accuracy: 0.013920369757931162
Epoch:  8/25,  Loss: 7.927501786768952,  Accuracy: 0.014065562662837877
Epoch:  9/25,  Loss: 7.925936873981546,  Accuracy: 0.014162357932775685
Epoch:  10/25,  Loss: 7.922731836929154,  Accuracy: 0.014154291660280868
Epoch:  11/25,  Loss: 7.924673063460589,  Accuracy: 0.014069595799085285
Epoch:  12/25,  Loss: 7.9237882029796385,  Accuracy: 0.014152275092157163
Epoch:  13/25,  Loss: 7.9242433098027405,  Accuracy: 0.014519290490671357
Epoch:  14/25,  Loss: 7.921711288247359,  Accuracy: 0.01456768812564026
Epoch:  15/25,  Loss: 7.9140255042915575,  Accuracy: 0.014761278665515878
Epoch:  16/25,  Loss: 7.912088041998429,  Accuracy: 0.014690698781186225
Epoch:  17/25,  Loss: 7.913314327668768,  Accuracy: 0.014741112984278834
Epoch:  18/25,  Loss: 7.909860413679693,  Accuracy: 0.014676582804320295
Epoch:  19/25,  Loss: 7.910724188654111,  Accuracy: 0.014722963871165496
Epoch:  20/25,  Loss: 7.908342268285728,  Accuracy: 0.014779427778629218
Epoch:  21/25,  Loss: 7.91364628866513,  Accuracy: 0.014860090503577391
Epoch:  22/25,  Loss: 7.91019680318809,  Accuracy: 0.014777411210505513
Epoch:  23/25,  Loss: 7.905723135430423,  Accuracy: 0.014866140207948506
Epoch:  24/25,  Loss: 7.906069486583336,  Accuracy: 0.014906471570422592
Epoch:  25/25,  Loss: 7.905446002399987,  Accuracy: 0.014809676300484784
```

```
def predict_next_word(model, input_sequence, ttoi, itot, top_k=1, device='cpu'):
    model.eval()

    input_indices = torch.tensor([ttoi[word] for word in input_sequence], dtype=torch.long).unsqueeze(0)

    input_indices = input_indices.to(device)

    with torch.no_grad():
        logits = model(input_indices)

    # Logits to probabilities
    probabilities = F.softmax(logits, dim=-1)

    top_probabilities, top_indices = torch.topk(probabilities, top_k, dim=-1)

    # Convert top_indices to words
    predicted_words = [itot[idx.item()] for idx in top_indices[0]]

    return predicted_words
```

```
text
```

```
'gabriel garcía márquez \n\n\n\ncien años de soledad \n\n\n\neditado por "ediciones la cueva" \n\n\n\npara j omi garcía ascot \ny m
aría luisa elio \n\n\n\ncien años de soledad \n\n\n\ngabriel garcía márquez \n\n\n\nmuchos años después, frente al pelotón de fusil
amiento, el coronel aureliano buendía había de \nrecordar aquella tarde remota en que su padre lo llevó a conocer el hielo. macondo
era entonces \nuna aldea de veinte casas de barro y cañabrava construidas a la orilla de un río de aguas diáfanas \nque se precipit
aban por un lecho de piedras pulidas, blancas y enormes como huevos \nprehistóricos. el mundo era tan reciente, que muchas cosas ca
recían de nombre, y para \nmencionarlas había que señalarlas con el dedo. todos los años, por el mes de marzo, una familia \nde git
```

```
intersting_words = ['macondo','enormes', 'gitanos','aldea','barba']

for word in intersting_words:
  n_pred_mx = 5
  full_pred = word
  for i in range(n_pred_mx):
      word2 = predict_next_word(model, [word], ttoi, itot, top_k=1, device=device)[0]
      full_pred = full_pred + ' ' + word2
      word = word2
  print(full_pred)
```

```
macondo aureliano aureliano aureliano aureliano aureliano
enormes aureliano aureliano aureliano aureliano aureliano
gitanos aureliano aureliano aureliano aureliano aureliano
aldea aureliano aureliano aureliano aureliano aureliano
barba aureliano aureliano aureliano aureliano aureliano
```

Conclusion

Se entrenó por 25 epocas una red CNN con 1000 filtros por capa. Word2Vec (word to vector) se utiliza para representar las relaciones entre diferentes palabras en forma de gráfico. De esta manera se captura significado, similaridad semantica y relaciones con el texto cercano. Durante el entrenamiento la pérdida se estabiliza en alrededor de 7.9. Las oraciones generadas caen en una repetición.