Neural Language Model

It's a language Model based on RNN/LSTM/Transformers.
Language Modeling (LM) is one of the most important parts of modern Natural Language Processing (NLP). There are many sorts of applications for Language Modeling, like: Machine Translation, Spell Correction Speech Recognition.

Why we need a Language Model

- Generate next word given a sequence of words
- Calculate the probability of some word given a sentence of words P("is"| "Hi", "my", "name")
- Calculate the probability of a whole sentence
  - P("The apple and help salad) = 0.3
  - P("The apple and pear salad) = 0.99

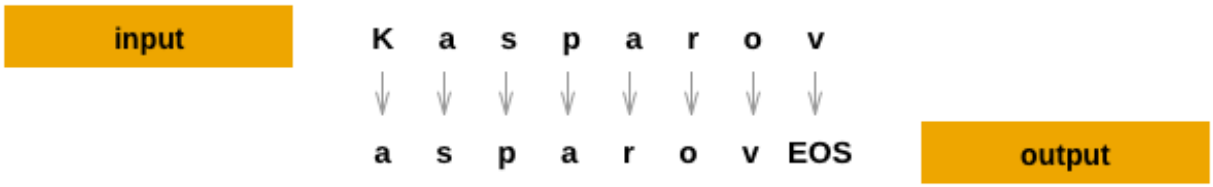
The language model will be used to refine the output of NLP systems.

Types of Language Model

- Character Level language Model
- Word level language Model

Training Character Language Model

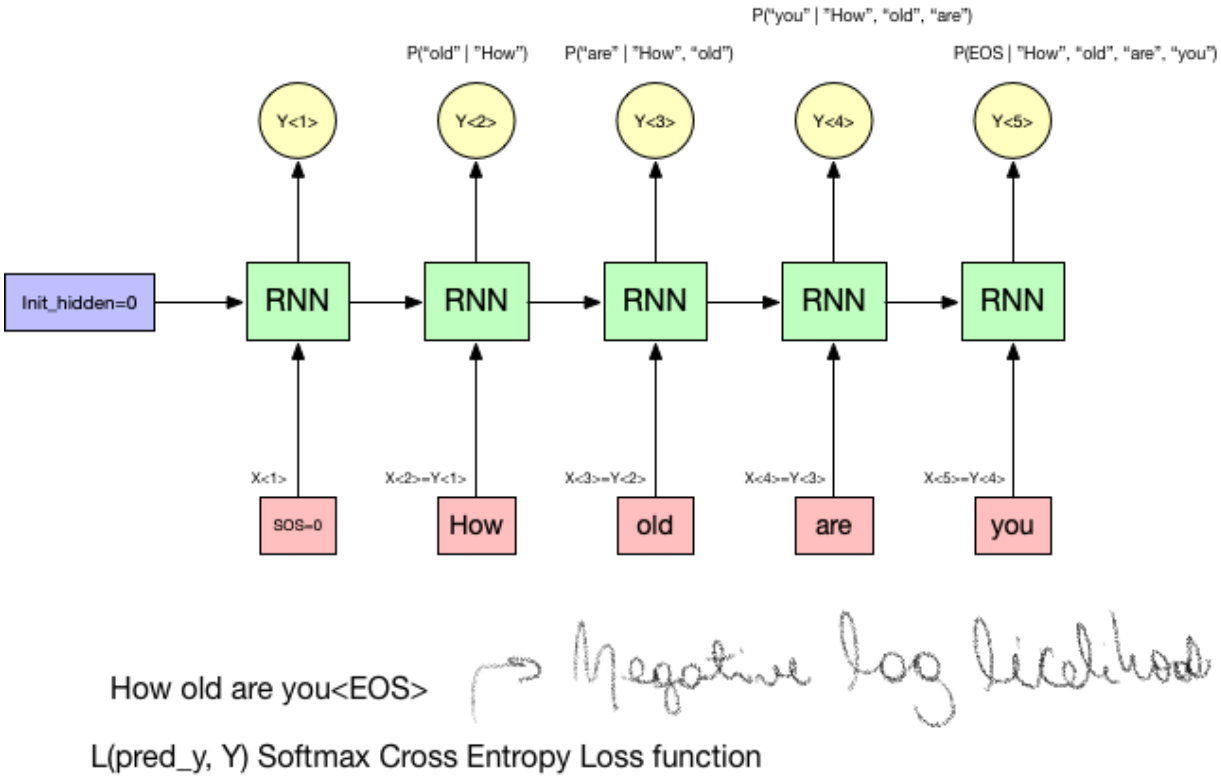
For each timestep (that is, for each letter in a training word) the inputs of the network will be (current letter) and the outputs will be (next letter). So for each time-step, we'll need a set of input letters, and a set of output/target letters.
So for example for "Kasparov" (8 time steps)



1. X: 'K' Y: 'a'
2. X: 'a' Y: 's'
3. X: 's' Y: 'p'
4. X: 'p' Y: 'a'
5. X: 'a' Y: 'r'
6. X: 'r' Y: 'o'
7. X: 'o' Y: 'v'
8. X:'v' Y: <EOS>

Calculate the Probability of a word given some sequence

Now we want to give to this RNN a sequence of characters and know the probability of this sequence.



Every time you push some data to the Language Model it will return the probability vector of the next character/word for the whole vocabulary given the things you pushed in so you need to select the next character/word from this probability vector and store into a list.

$$P(w_t|\text{context})\,\forall t \in V.$$

The final conditional probability will be the product of the probabilities from this list to illustrate this process consider the string ‘Hello’

```
: prob_sentence('Hello', char_lm, device, codemap)
```

```
sentence: Hello
chars_class: [23, 46, 53, 53, 56]
num_chars: 5
0) Push to RNN:[23 --> H]
        P('e' | [H])=0.361420
1) Push to RNN:[46 --> e]
        P('l' | [H,e])=0.044643
2) Push to RNN:[53 --> l]
        P('l' | [H,e,l])=0.086002
3) Push to RNN:[53 --> l]
        P('o' | [H,e,l,l])=0.007452
4) Push to RNN:[56 --> o]
probabilities vector: [0.36142000555992126, 0.0446433499455452, 0.08600206673145294, 0.007451722398400307]
```

The code of the function called

```python
def prob_sentence(word, model, device, codemap):
    print('sentence:', word)
    # Convert each character on the word into it's class id
    chars_class = [utils_char_dataset.class_id_from_char(char, codemap) for char in word]
    print('chars_class:', chars_class)
    num_chars = len(chars_class)
    print('num_chars:', num_chars)
    curr_batch_size = 1
    model.eval()
    scores_lst = []
    prev_chars = []
    with torch.no_grad():
        # Initialize model on the beginning of the sequence
        hidden_state = models.initHidden(curr_batch_size, False, model.hidden_size, model.num_layers, device)
        # Iterate on all charactres from word ie: Hello --> [23, 46, 53, 53, 56]
        for idx in range(num_chars):
            char_curr = utils_char_dataset.char_from_class_id(chars_class[idx], codemap)
            print('%d) Push to RNN:[%d --> %s]' % (idx, chars_class[idx], char_curr))
            # Convert class word index to a tensor
            input = torch.tensor(chars_class[idx]).type(torch.LongTensor).unsqueeze(0).unsqueeze(0).to(device)
            # Push input(character) to the model
            # Probabilities shape [1 x 1 x num_classes]
            probabilities, hidden_state = model(input, hidden_state, torch.tensor(1).unsqueeze(0))

            prev_chars.append(char_curr)

            # Select all characters but exclude the last (Hell), exclude(o)
            if idx < num_chars - 1:
                chars_class_next = torch.tensor(chars_class[idx+1]).type(torch.LongTensor).unsqueeze(0).unsqueeze(0).to(device).item()
                probabilities = probabilities.squeeze(0).squeeze(0)
                # Select the probability of the character that we will push next
                prob_next = probabilities[chars_class_next].item()
                scores_lst.append(prob_next)
                char_next = utils_char_dataset.char_from_class_id(chars_class_next, codemap)
                print('\tP(\'%s\' | [%s])=%f' % (char_next, ','.join(prev_chars), prob_next))

    # Return the product of the probabilities
    # The first element is the probability of 'e' given 'H' P(e|H)
    print('probabilities vector:',scores_lst)
    return np.prod(scores_lst)
```

Language Model Metrics

Depending of the Language model (Character/Word) level we might use different metrics

- Perplexity: For Word level language Model (that can be calculated by doing the exp of the cross entropy loss)
- Character Error Rate, or BPC (Bits per character)

References

- https://mchromiak.github.io/articles/2017/Nov/30/Explaining-Neural-Language-Modeling/#.XQvQPS-ZPRY
- https://www.youtube.com/watch?v=1rOCxV0fSyM
- https://www.youtube.com/watch?v=CKrxdgqBheY

- https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html
- https://www.wikiwand.com/en/Language_model
- https://medium.com/coinmonks/character-to-character-rnn-with-pytorchs-lstmcell-cd923a6d0e72
- https://medium.com/the-artificial-impostor/notes-neural-language-model-with-pytorch-a8369ba80a5c
- https://towardsdatascience.com/writing-like-shakespeare-with-machine-learning-in-pytorch-d77f851d910c
- https://github.com/cedricdeboom/character-level-rnn-datasets
- https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/
- https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf
- https://github.com/joosthub/PyTorchNLPBook
- https://books.google.co.uk/books?id=Gh-EDwAAQBAJ&pg=PR1&source=gbs_selected_pages&cad=2#v=onepage&q&f=false