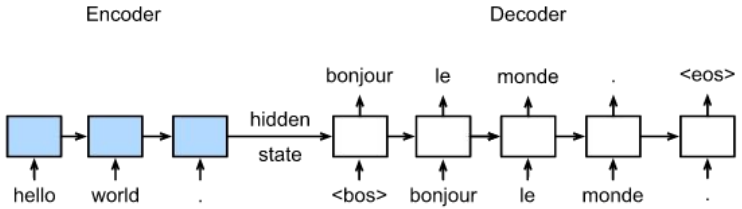
NATURAL LANGUAGE PROCESSING

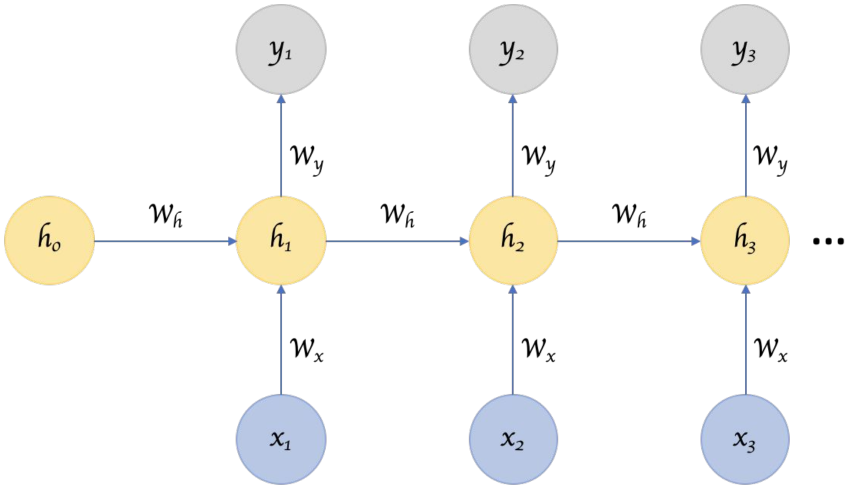
CONTENTS

- RNN review
- Natural Language Processing
- Character RNN
    - Tokenization
    - Embeddings
    - Generating text one character at a time
- Sentiment Analysis
    - Torchtext
    - Bi-directional RNNs
    - Transfer learning
    - Text Classification
- Seq2Seq models
    - Encoder-Decoder architecture
    - Machine translation
- Attention Mechanisms
    - Transformers
    - BERT for text classification



RNN Review

RNN



$$y_t = \mathbf{W}_y \mathbf{h}_t = \mathbf{W}_y f(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1})$$

```python
class RNN(torch.nn.Module):
  def __init__(self, n_in=50, n_out=1):
    super().__init__()
    self.rnn = torch.nn.RNN(input_size=1, hidden_size=20, num_layers=2, batch_first=True)
    self.fc = torch.nn.Linear(20, 1)

  def forward(self, x):
    x, h = self.rnn(x)
    # get the last output
    x = self.fc(x[:,-1])
    return x
```

💡 Use LSTM or GRU for better results.

Natural Language Processing

Natural Language Processing

- Language comprehension (virtual assistants such as Siri, Alexa, ...)
- Machine translation (Google Translate, ...)
- Text generation (language modeling, summarization, question answering, ...)
- Text classification(sentiment analysis, identify hate speech on social media, ...)
- Text-to-speech (generate audio from text) and Speech-to-text
- Image captioning and OCR

⚡ NLP is a very active field at the moment, new huge architectures (Transformers) and training techniques (language modeling on big unsupervised datasets) are providing excellent results improving SOTA by large margin on almost every task. See for example: https://openai.com/blog/openai-api/

Character RNN

CharRNN

- Generate text, one character at a time (https://github.com/karpathy/char-rnn)

AUTOLYCUS:
This is a merry ballad, but a very pretty one.

MOPSA:
Let's have some merry ones.

AUTOLYCUS:
Why, this is a passing merry one and goes to the tune of 'Two maids wooing a man:' there's scarce a maid westward but she sings it; 'tis in request, I can tell you.

MOPSA:
We can both sing it: if thou'lt bear a part, thou shalt hear; 'tis in three parts.

DORCAS:
We had the tune on't a month ago.

AUTOLYCUS:
I can bear my part; you must know 'tis my occupation; have at it with you.

Tokenization

We need to transform each character to a number.

```
import string

class Tokenizer():
  def __init__(self):
    self.all_characters = string.printable
    self.n_characters = len(self.all_characters)
  def text_to_seq(self, string):
    seq = []
    for c in range(len(string)):
        seq.append(self.all_characters.index(string[c]))
    return seq
  def seq_to_text(self, seq):
    text = ''
    for c in range(len(seq)):
        text += self.all_characters[seq[c]]
    return text
```

```
0123456789abcdefghijklmnopqrstuvwxy
zABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'
()*+,-./:;<=>?@[\\]^_`{|}~
\t\n\r\x0b\x0c'
```

```
tokenizer.text_to_seq('abcDEF')
> [10, 11, 12, 39, 40, 41]

tokenizer.seq_to_text([10, 11, 12])
> 'abc'
```

Creating text windows

```python
def windows(text, window_size = 100):
    start_index = 0
    end_index = len(text) - window_size
    text_windows = []
    while start_index < end_index:
      text_windows.append(text[start_index:start_index+window_size+1])
      start_index += 1
    return text_windows
```

```
> ['First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst
Citizen:\nYou ',

 'irst Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst
Citizen:\nYou a',

 'rst Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst
Citizen:\nYou ar']
```

Dataset

```python
class CharRNNDataset(torch.utils.data.Dataset):
  def __init__(self, text_encoded_windows, train=True):
    self.text = text_encoded_windows
    self.train = train

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

  def __getitem__(self, ix):
    if self.train:
      return torch.tensor(self.text[ix][:-1]), torch.tensor(self.text[ix][-1])
    return torch.tensor(self.text[ix])
```

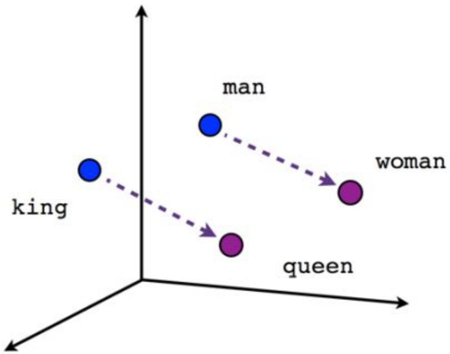
Embeddings

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

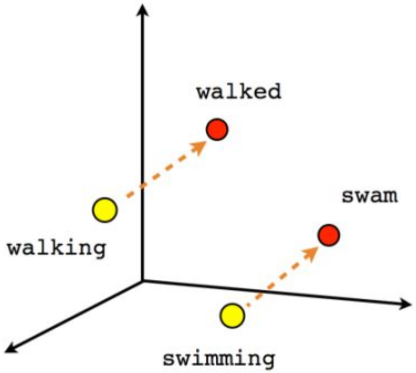
- The tokenizer converts each character into a number (0-99).
- We need to transform each number (class) to either:
  - Embeddings
  - One-hot encoding

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

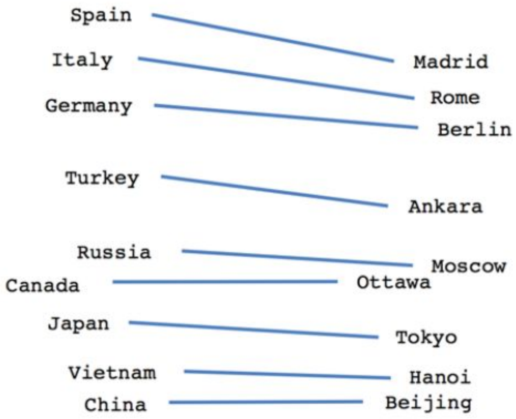
	Femininity	Youth	Royalty
Man	0	0	0
Woman	1	0	0
Boy	0	1	0
Girl	1	1	0
Prince	0	1	1
Princess	1	1	1
Queen	1	0	1
King	0	0	1
Monarch	0.5	0.5	1



Male-Female



Verb tense



Country-Capital

```python
class CharRNN(torch.nn.Module):
  def __init__(self, n_out=100, dropout=0.2, input_size=100, embedding_size=100, 
hidden_size=128):
    super().__init__()
    self.encoder = torch.nn.Embedding(input_size, embedding_size)
    self.rnn = torch.nn.GRU(input_size=embedding_size, hidden_size=hidden_size, num_layers=2, 
dropout=dropout, batch_first=True)
    self.fc = torch.nn.Linear(hidden_size, n_out)

  def forward(self, x):
    x = self.encoder(x)
    x, h = self.rnn(x)
    y = self.fc(x[:,-1,:])
    return y
```

Generating text

```
X_new = "With eyes wide open; standing, speaking, movin"

for i in range(1000):
  X_new_encoded = tokenizer.text_to_seq(X_new[-100:])
  y_pred = model.predict(X_new_encoded)
  y_pred = torch.argmax(y_pred, axis=1)[0].item()
  X_new += tokenizer.seq_to_text([y_pred])
```

```
> 'With eyes wide open; standing, speaking, movingment\nWhere should should should should should
should should should should should should should should should she shall she\nshall should should
should should should should should should should should should should should should she shall
she\nshall should should should should should should should should should should should should
should should she shall she\nshall should should should should should should should should should
should should should should should she shall she\nshall should should should should should should
should should should should should should should should she shall she\nshall should should should
should should should should should should should should should should she shall she\nshall
should should should should should should should should should should should should should should
she shall she\nshall should should should should should should should should should should should
should should should she shall she\nshall should should should should should should sh'
```

```
X_new = "With eyes wide open; standing, speaking, movin"

temp= 0.8
for i in range(1000):
  X_new_encoded = tokenizer.text_to_seq(X_new[-100:])
  y_pred = model.predict(X_new_encoded)
  y_pred = y_pred.view(-1).div(temp).exp()
  top_i = torch.multinomial(y_pred, 1)[0]
  predicted_char = tokenizer.all_characters[top_i]
  X_new += predicted_char
```

With eyes wide open; standing, speaking, moving,
Do the word, gain is the own spoke quick, and my heart
Leates of his face.
Come and all from me reday:
Now not seem by sight?

Third Citizen:
Now bring the life of aid the deprisonant contrees;
For where to your repongine enough here stright
Should sure them with lupken Cite best we weed should
should not leave:
What can here is thee he is my child and there is thee,
Not son tears of me believe the best see purgal they she
Twict in the noble with his words of stross
Of that you cut make you to be confess,
His stricks, but myself free so popurine of yourself.

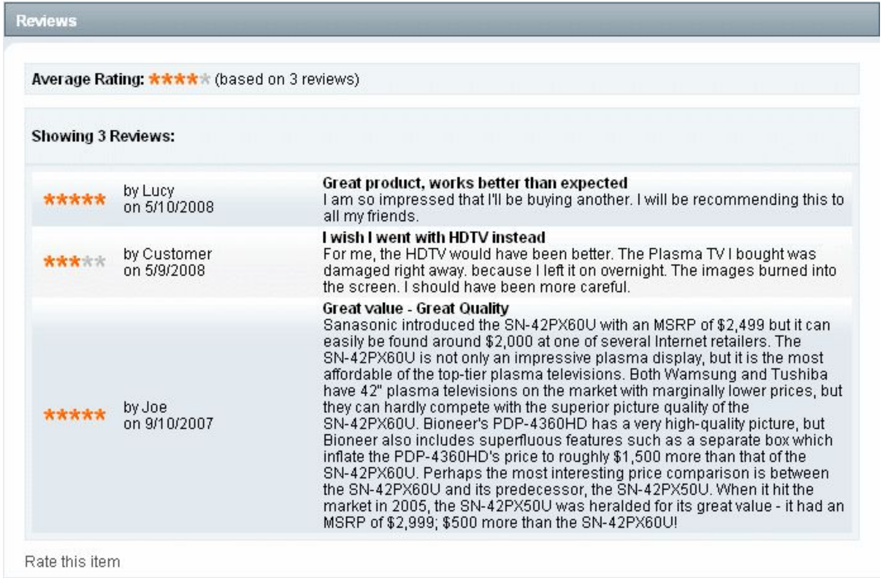
LUCIO:
Now, for her hand? that sing for the prince is speak
Ot shall desire death, serve for that what we die me;
Send you are passion vapeen and move princes me them,
Which you that knows that from from your airful truester
My lord, for when I see, speak, and i' the conspiny,
Luth as promised.

DUCHESS OF YORK:
Now, with this heart, tyrant me shall the refore that you
emstre:
Gaunter, known her stept that is grace is the sheep

Sentiment Analysis

Sentiment Analysis

- Sentiment analysis is a case of text classification, where we want to assign a particular label to a piece of text (detect hate speech in social networks, user satisfaction on product reviews, ...)



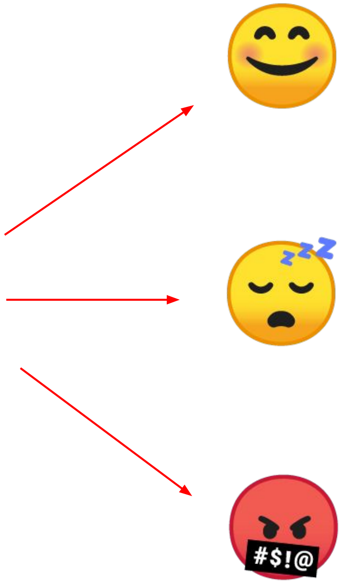
Reviews

Average Rating: ★★★★☆ (based on 3 reviews)

Showing 3 Reviews:

★★★★★	by Lucy on 5/10/2008	**Great product, works better than expected** I am so impressed that I'll be buying another. I will be recommending this to all my friends.
★★★☆☆	by Customer on 5/9/2008	**I wish I went with HDTV instead** For me, the HDTV would have been better. The Plasma TV I bought was damaged right away. because I left it on overnight. The images burned into the screen. I should have been more careful.
★★★★★	by Joe on 9/10/2007	**Great value - Great Quality** Sanasonic introduced the SN-42PX60U with an MSRP of $2,499 but it can easily be found around $2,000 at one of several Internet retailers. The SN-42PX60U is not only an impressive plasma display, but it is the most affordable of the top-tier plasma televisions. Both Wamsung and Tushiba have 42" plasma televisions on the market with marginally lower prices, but they can hardly compete with the superior picture quality of the SN-42PX60U. Bioneer's PDP-4360HD has a very high-quality picture, but Bioneer also includes superfluous features such as a separate box which inflate the PDP-4360HD's price to roughly $1,500 more than that of the SN-42PX60U. Perhaps the most interesting price comparison is between the SN-42PX60U and its predecessor, the SN-42PX50U. When it hit the market in 2005, the SN-42PX50U was heralded for its great value - it had an MSRP of $2,999; $500 more than the SN-42PX60U!

Rate this item



Torchtext

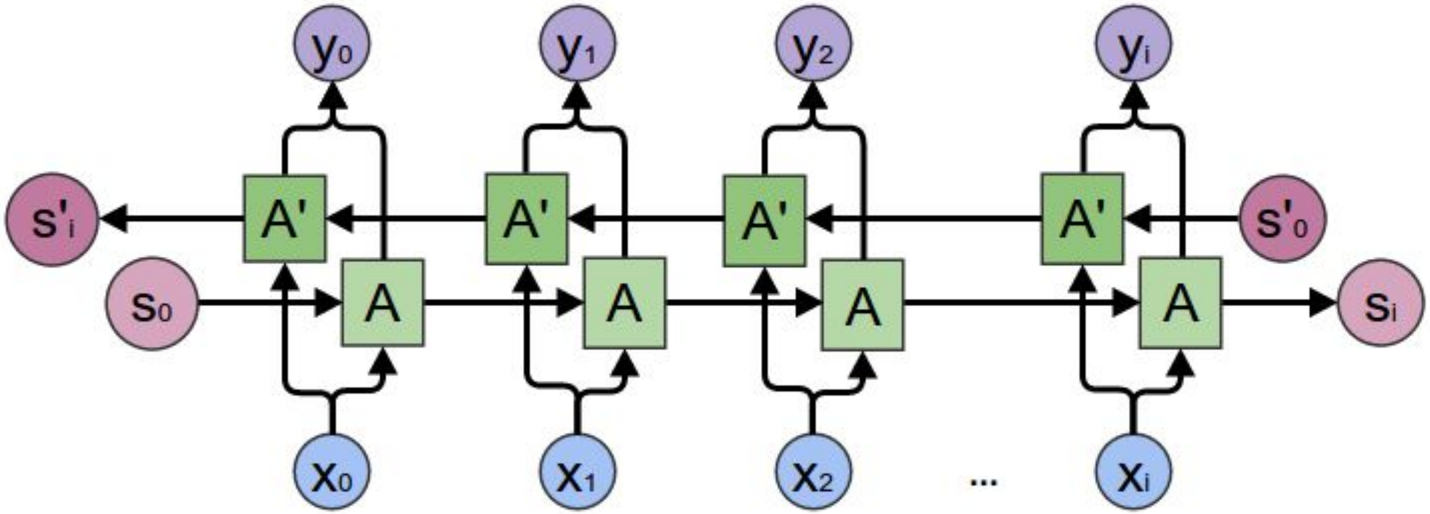
https://pytorch.org/text/

- Abstract and automate text data preprocessing (tokenization, data splitting, batching, ...)
- Includes some popular datasets, here we use IMDB movie reviews dataset.
- Customize to your needs.

```
import torch
import torchtext

TEXT = torchtext.data.Field(tokenize = 'spacy')
LABEL = torchtext.data.LabelField(dtype = torch.float)

train_data, test_data = torchtext.datasets.IMDB.splits(TEXT, LABEL)

MAX_VOCAB_SIZE = 10000
TEXT.build_vocab(train_data, max_size = MAX_VOCAB_SIZE)
LABEL.build_vocab(train_data)
```

```python
class RNN(torch.nn.Module):
    def __init__(self, input_dim, embedding_dim=128, hidden_dim=128, output_dim=1, dropout=0.2):
        super().__init__()
        self.embedding = torch.nn.Embedding(input_dim, embedding_dim)
        self.rnn = torch.nn.GRU(input_size=embedding_dim, hidden_size=hidden_dim, num_layers=2,
dropout=dropout)
        self.fc = torch.nn.Linear(hidden_dim, output_dim)

    def forward(self, text):
        #text = [sent len, batch size]
        embedded = self.embedding(text)
        #embedded = [sent len, batch size, emb dim]
        output, hidden = self.rnn(embedded)
        #output = [sent len, batch size, hid dim]
        y = self.fc(output[-1,:,:].squeeze(0)).squeeze(1)
        return y
```

Bidirectional RNNs



```python
class BidirectionalRNN(RNN):
    def __init__(self, input_dim, embedding_dim=128, hidden_dim=128, output_dim=1, dropout=0.2,
pad_idx=0, bidirectional=True):
        super().__init__(input_dim, embedding_dim, hidden_dim, output_dim, dropout)
        self.rnn = torch.nn.GRU(input_size=embedding_dim,
                                hidden_size=hidden_dim,
                                num_layers=2,
                                dropout=dropout,
                                bidirectional=bidirectional)
        if bidirectional:
          self.fc = torch.nn.Linear(2*hidden_dim, output_dim)
```

Transfer Learning

- Use pre-trained embeddings

```
TEXT.build_vocab(train_data,
                 max_size = MAX_VOCAB_SIZE,
                 vectors = "glove.6B.100d",
                 unk_init = torch.Tensor.normal_)
LABEL.build_vocab(train_data)

pretrained_embeddings = TEXT.vocab.vectors

net.embedding.weight.data.copy_(pretrained_embeddings)

net.embedding.weight.data[TEXT.vocab.stoi[TEXT.unk_token]] = torch.zeros(100)
net.embedding.weight.data[TEXT.vocab.stoi[TEXT.pad_token]] = torch.zeros(100)
```

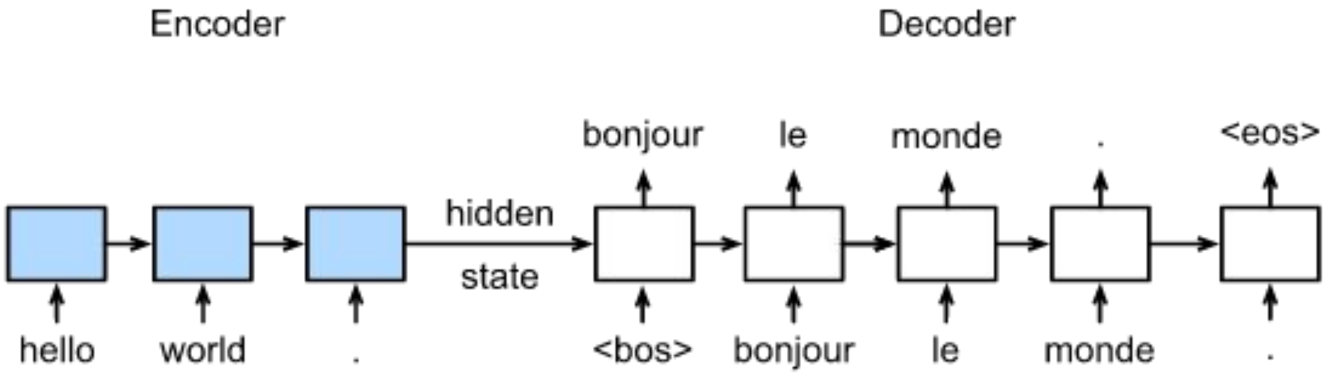
Text Classification

```
import spacy
nlp = spacy.load('en')

sentences = ["this film is terrible", "this film is great", "this film is good", "a waste of time"]
tokenized = [[tok.text for tok in nlp.tokenizer(sentence)] for sentence in sentences]
indexed = [[TEXT.vocab.stoi[_t] for _t in t] for t in tokenized]
tensor = torch.tensor(indexed).to(device).permute(1,0)
net.eval()
prediction = torch.sigmoid(net(tensor))

> [0.0732, 0.9613, 0.8762, 0.0119]
```

Sequence to sequence models

Sequence to sequence models

- Encoder-Decoder architecture.
- The encoder creates the initial state of the decoder.
- The decoder generates text one word at a time, using the output at each step as input for the next until termination.
- Used for machine translation, text summarization, ...
- If the encoder is a CNN, it can be used for image captioning, OCR, ...



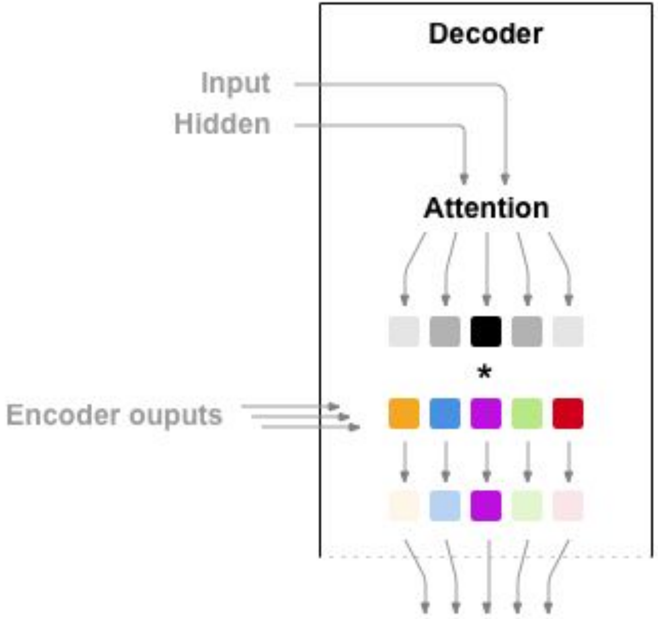
EXERCISE !

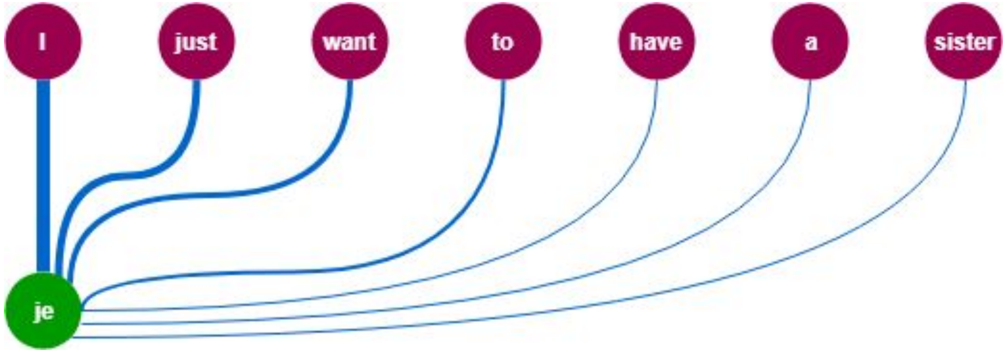
https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Attention Mechanisms

seq2seq with attention

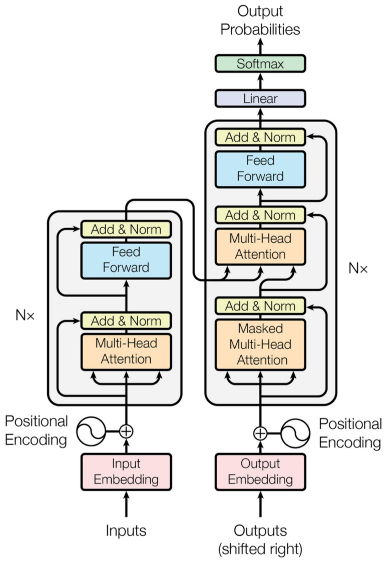
- Attention mechanisms allows the network to focus on specific parts of its inputs.

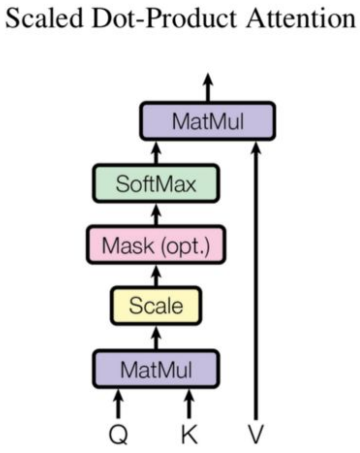


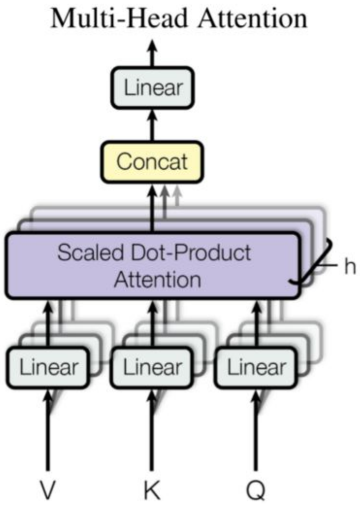


Transformers

- A *Transformer* is a Neural Network architecture based on *Attention Mechanisms* https://arxiv.org/pdf/1706.03762.pdf
- They are the SOTA today in NLP.







BERT for text classification

```
from transformers import BertTokenizer
```

```
pip install transformers
```

```
tokenizer = BertTokenizer.from_pretrained ('bert-base-uncased' )

max_input_length = tokenizer.max_model_input_sizes ['bert-base-uncased' ]

def tokenize_and_cut (sentence):
    tokens = tokenizer.tokenize (sentence)
    tokens = tokens [:max_input_length -2]
    return tokens
```

```
TEXT = torchtext.data.Field (batch_first = True,
                  use_vocab = False,
                  tokenize = tokenize_and_cut ,
                  preprocessing = tokenizer.convert_tokens_to_ids ,
                  init_token = tokenizer.cls_token_id ,
                  eos_token = tokenizer.sep_token_id ,
                  pad_token = tokenizer.pad_token_id ,
                  unk_token = tokenizer.unk_token_id )

LABEL = torchtext.data.LabelField (dtype = torch. float)
```

```python
from transformers import BertModel
bert = BertModel.from_pretrained ('bert-base-uncased' )

class BERTGRUSentiment (nn.Module):
    def __init__ (self, bert, hidden_dim=256, output_dim=1, n_layers=2, bidirectional =True, dropout=0.2):
        super () .__init__ ()
        self.bert = bert
        embedding_dim = bert.config.to_dict  ()['hidden_size' ]
        self.rnn = nn.GRU (embedding_dim , hidden_dim, num_layers = n_layers , bidirectional = bidirectional ,
batch_first = True, dropout = 0 if n_layers < 2 else dropout)

        self.out = nn.Linear (hidden_dim * 2 if bidirectional  else hidden_dim, output_dim)
        self.dropout = nn.Dropout (dropout)

    def forward(self, text):
        with torch.no_grad ():
            embedded =  self.bert (text) [0]
        _, hidden =  self.rnn (embedded)
        if self.rnn.bidirectional :
            hidden =  self.dropout (torch.cat ((hidden [-2,:,:], hidden [-1,:,:]), dim = 1))
        else:
            hidden =  self.dropout (hidden [-1,:,:])
        output =  self.out (hidden)
        return output.squeeze (1)
```

```
def predict(sentence):
    tokenized = [tok[:max_input_length-2] for tok in tokenizer.tokenize(sentence)]
    indexed = [tokenizer.cls_token_id] + tokenizer.convert_tokens_to_ids(tokenized) +
tokenizer.sep_token_id]
    tensor = torch.tensor([indexed]).to(device)
    model.net.eval()
    return torch.sigmoid(model.net(tensor)).item()

sentences = ["Best film ever !", "this movie is terrible"]
preds = [predict(s) for s in sentences]
```

> [0.984, 0.012]

https://github.com/sensioai/dl/tree/master/nlp

Resources

Learn

- https://www.youtube.com/watch?v=8rXD5-xhemo&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z
- https://www.fast.ai/2019/07/08/fastai-nlp/
- https://www.youtube.com/watch?v=4jROlXH9Nvc

Practice

- https://pytorch.org/tutorials/index.html
- https://github.com/bentrevett/pytorch-sentiment-analysis

NATURAL LANGUAGE PROCESSING