Building text classifiers using state-of-the-art Deep Learning frameworks

Inbal Horev - Senior Data Scientist, Gong.io

Gong.io at a glance

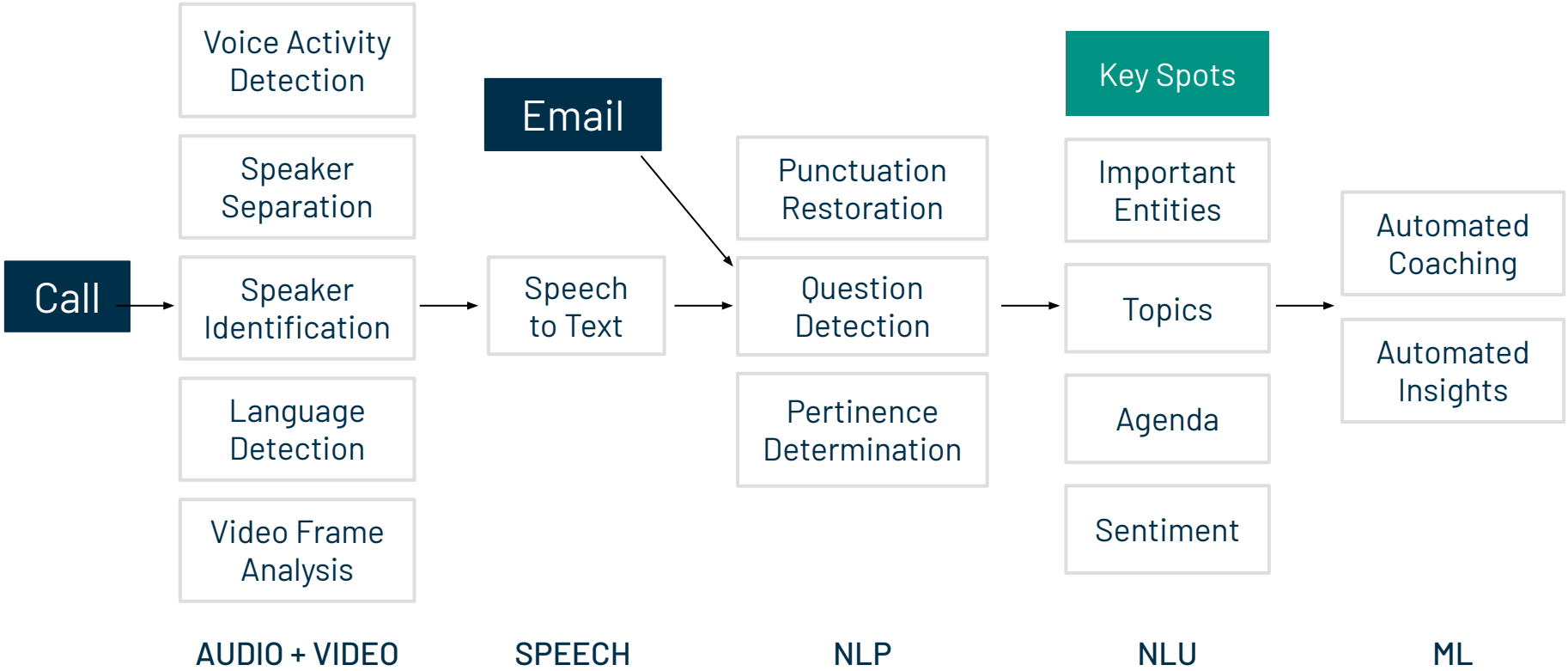
RECORD

TRANSCRIBE

ANALYZE

Provide Insights: “Top sales representatives talk ***more*** about ***the competition***, but ***later*** in the call, and *after* they’ve discussed the ***value*** of the solution ***for X minutes***”

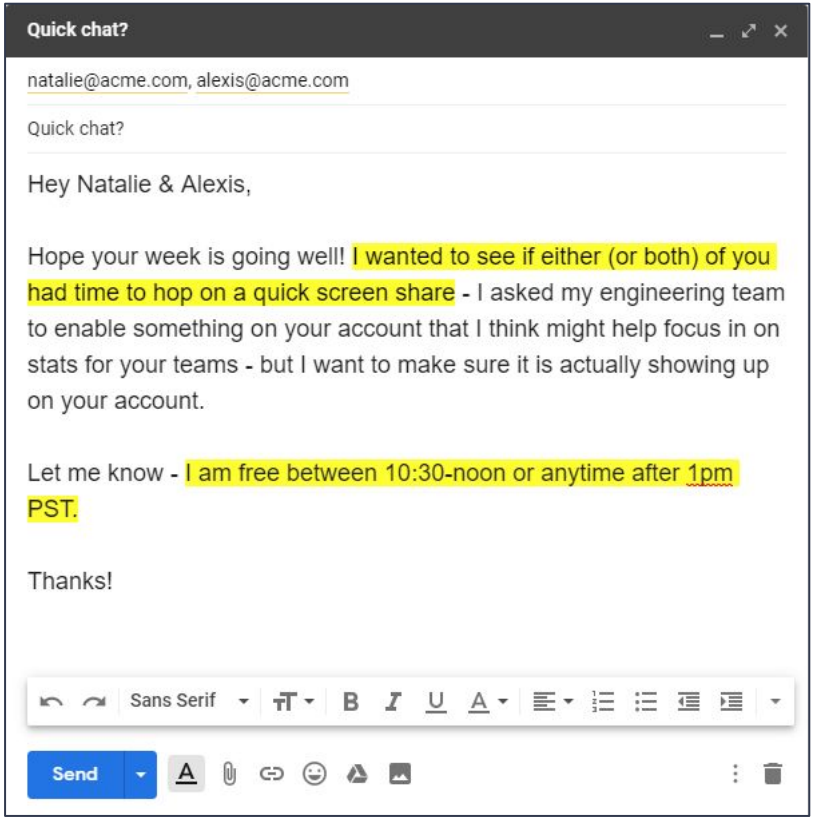
High Level ML models



Case study: Email tagging

Key Spots

- Action items
- Objections
- Requests
- Scheduling



Bag of words (< 2013)

Count the number of appearances of each word

“Hope your week is going well!”

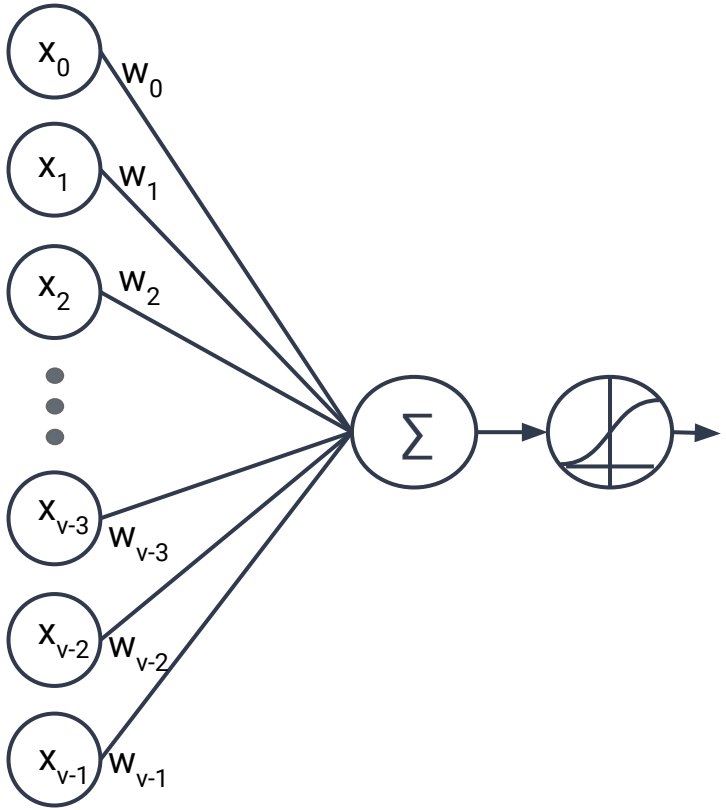
0	account
0	both
0	business
0	do
1	going
⋮	
1	hope
1	week
1	well
0	why
1	your
0	yourself

`sklearn.feature_extraction.text.TfidfVectorizer`

A very simple model

BOW + Logistic regression
Weighted sum of word counts

We can think of this as a single layer NN



`sklearn.linear_model.LogisticRegression`

Results so far

method	f1-score
BOW + LR	**0.88**

The trouble with BOW

- Vectors are very large - the size of the vocabulary
- Out-of-vocabulary (OOV) words are ignored
- Does not encode the similarity between words

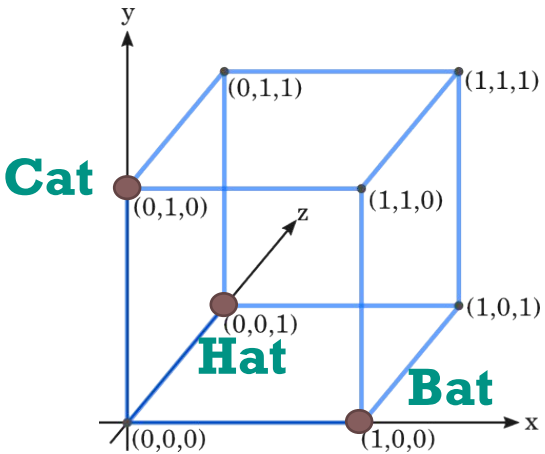
$$\|x_{cat} - x_{kitten}\| = \|x_{cat} - x_{teapot}\|$$

- Order of words within sentence is disregarded

"The movie was boring and not funny."

↕

"The movie was funny and not boring."



Word embeddings (2013 -)

Objectives

- Represent words with dense, fixed size vectors
- Similar words get similar representations
- Implicitly encode linguistic features

Word embeddings (2013 -)

*"**You shall know a word by the company it keeps**"*
John Rupert Firth

Word embeddings are learned from co-occurrences

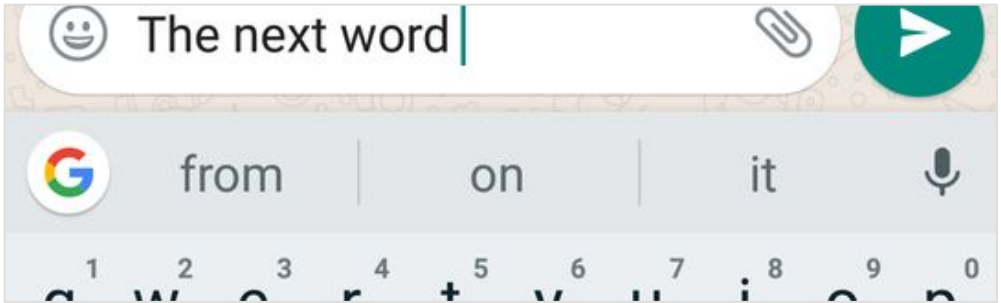
"The capital of England is London."

"The capital of France is Paris."

Word embeddings (2013 -)

Predict a word given its context,
a.k.a learn a **Language Model**

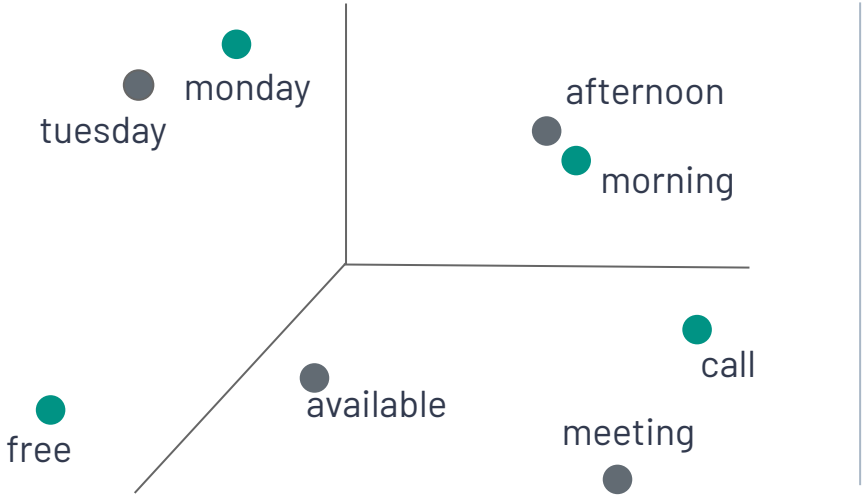
Intermediate computations
are used as the word vectors



Word embeddings (2013 -)

"Would you be available for a meeting on Tuesday afternoon?"

"Are you free for a call this Monday morning?"

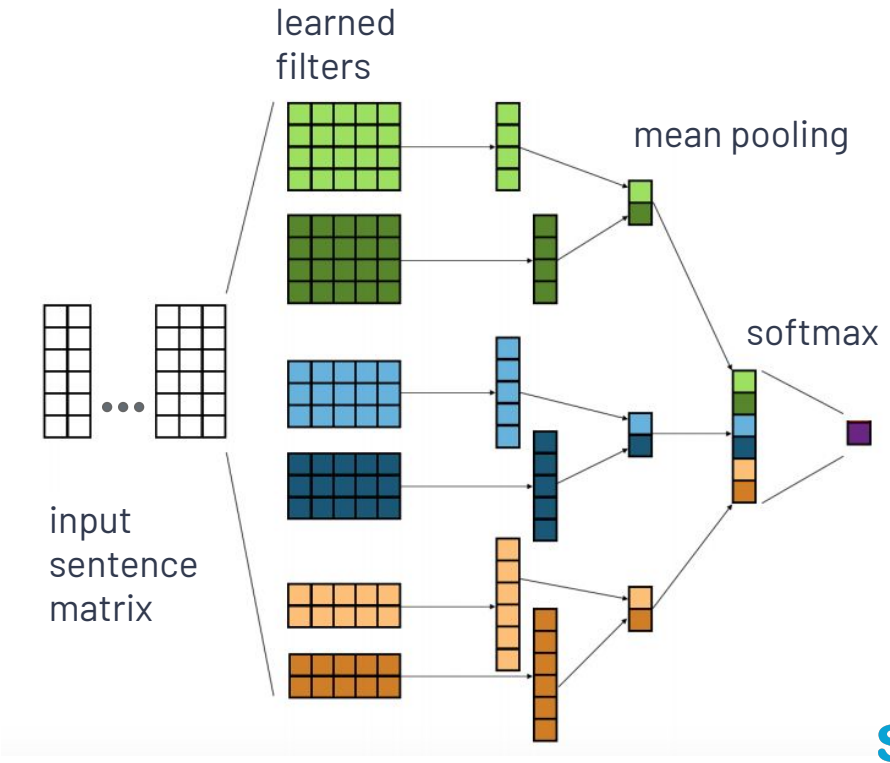


Similar words are close in embedding space

A more complex model

Word2Vec
Trained on our own email corpus

\+ **Convolutional NN**
Filters capture relation between words and linguistic features



spaCy

Training your own word embeddings

```
from gensim.models import Word2Vec
w2v_model = Word2Vec(sentence_generator, size=d,
                     workers=workers,
min_count=min_cnt)
w2v_model.wv.save_word2vec_format("word2vec_model.txt")
```

```
python -m spacy init-model en email_word2vec
--vectors-loc word2vec.txt

custom_vec_nlp = spacy.load("word2vec_model.txt")
```

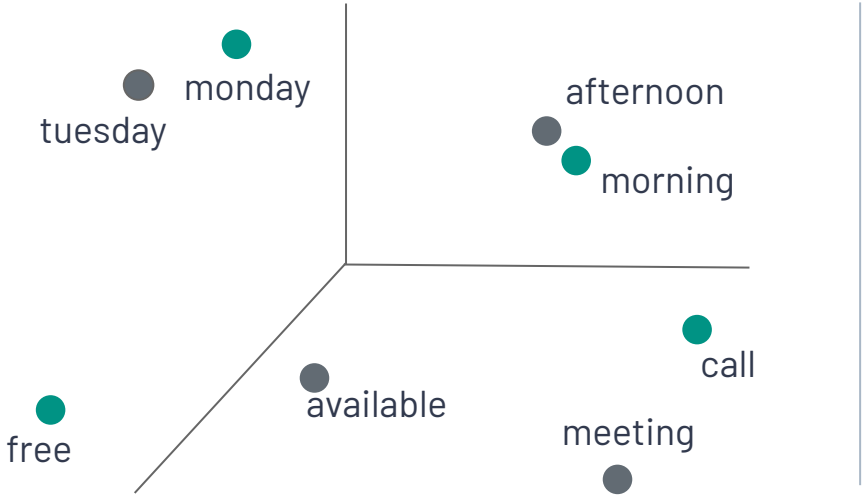
Results so far

method	f1-score
BOW + LR	0.88
W2V + CNN	**0.89**

Word embeddings (2013 -)

"Would you be available for a meeting on Tuesday afternoon?"

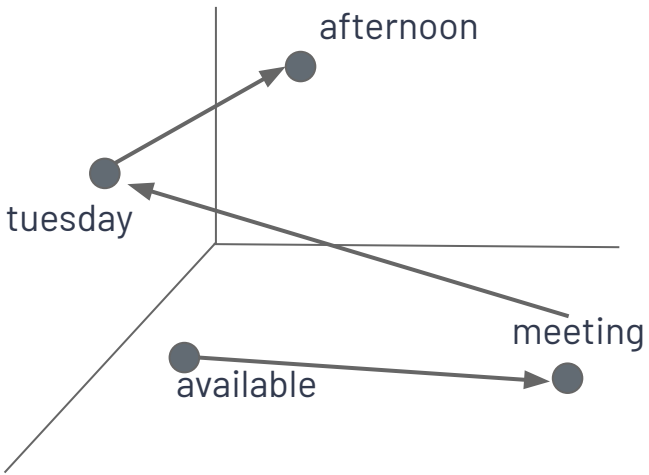
"Are you free for a call this Monday morning?"



Similar words are close in embedding space

Contextual word embeddings (2017-)

"Would you be available for a meeting on Tuesday afternoon?"

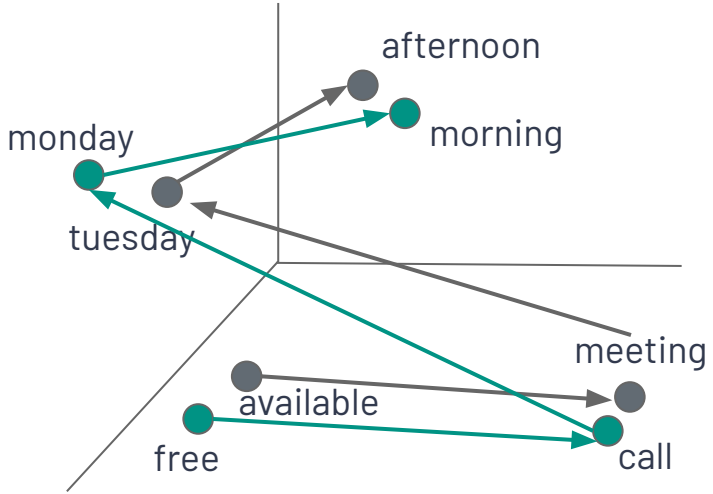


Word embeddings are dependent on other words in sentence

Contextual word embeddings (2017-)

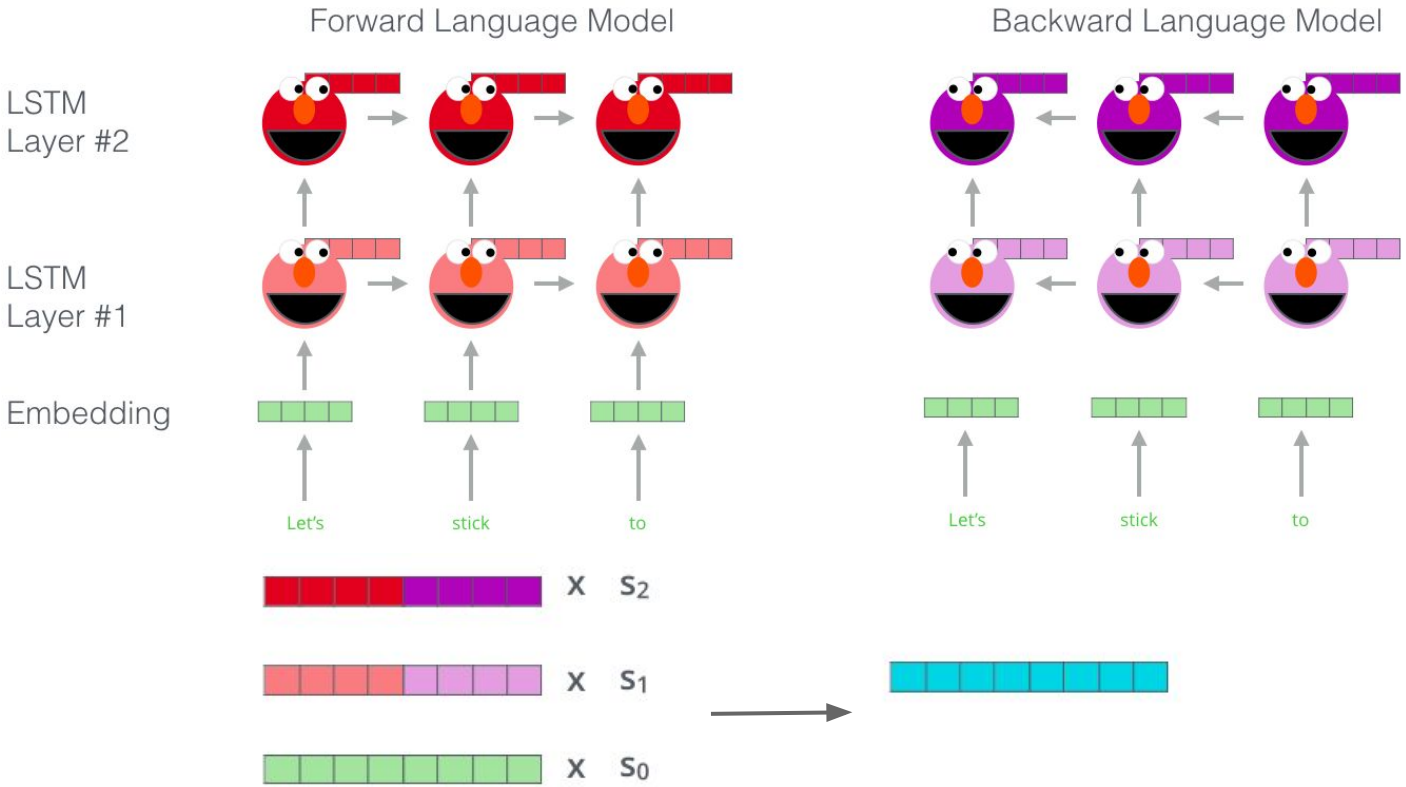
"Would you be available for a meeting on Tuesday afternoon?"

"Are you free for a call this Monday morning?"



In this context, "free" and "available" are similar

ELMO



http://jalammar.github.io/illustrated-bert/

Results so far

method	f1-score
BOW + LR	0.88
W2V + CNN	0.89
ELMo	**0.91**

Transfer learning

Responsible for *giant leap forward* in CV and NLP

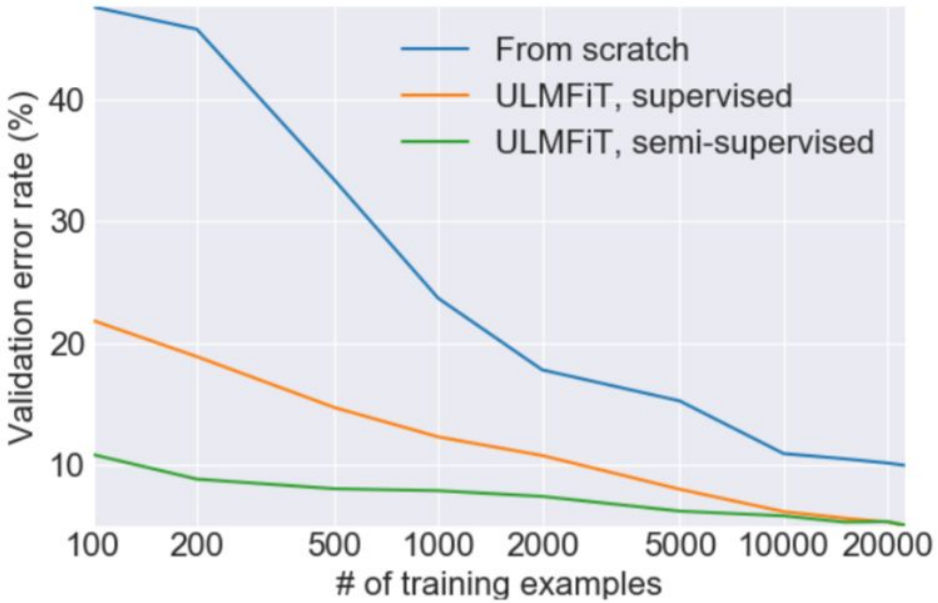
1. Use a **pre-trained language model** trained on huge corpora
   Encodes world knowledge

2. **Fine tune** the language model to own domain
   Use large unlabeled corpus

3. Fine tune classifier
   Requires only a **small labeled data set**

ULMFiT (2018)

- Trained on Wikipedia103 (100M tokens)
- Runs fast, even without a GPU
- Can be used with only a small set of labeled documents ("supervised"), but improves with the addition of a large dataset of unlabeled data




```
from fastai import *
```

ULMFiT (2018)

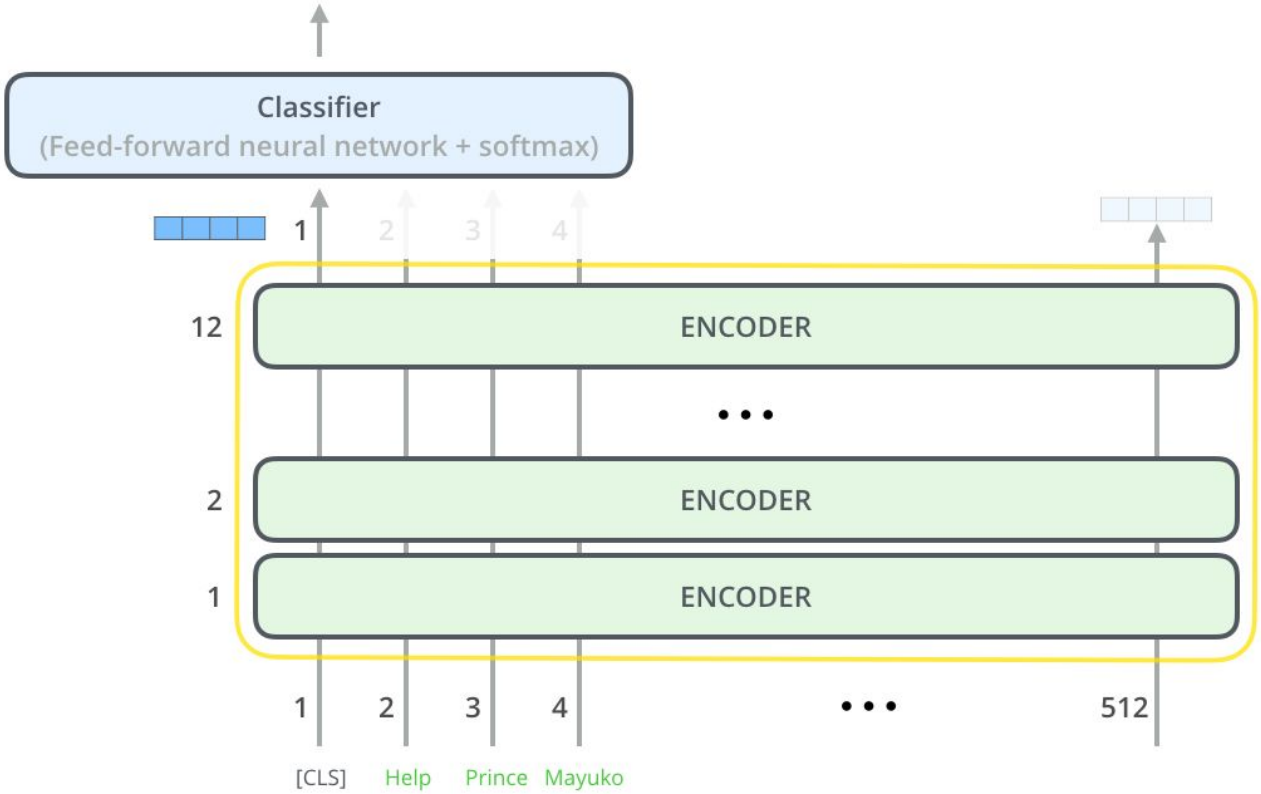
```
data_lm = TextLMDataBunch.from_df(train_df=df_trn,
valid_df=df_trn, path="", label_cols='label',
text_cols='text')
learn = language_model_learner(data_lm, drop_mult=0.1,
arch=AWD_LSTM)
learn.fit_one_cycle(1, 1e-2)
learn.save_encoder('ft_enc')

data_clas = TextClasDataBunch.from_df(path="",
train_df=df_trn, valid_df=df_val, label_cols='label',
text_cols='text', vocab=data_lm.train_ds.vocab, bs=16)
learn = text_classifier_learner(data_clas,drop_mult=0.1,
arch=AWD_LSTM)
learn.load_encoder('ft_enc')
learn.fit_one_cycle(1, 5e-2)
```

Results so far

method	f1-score
BOW + LR	0.88
W2V + CNN	0.89
ELMo	0.91
ULMFiT	**0.9**

Finally, BERT



http://jalammar.github.io/illustrated-bert/

Final results

method	f1-score
BOW + LR	0.88
W2V + CNN	0.89
ELMo	0.91
ULMFiT	0.9
BERT	**0.94**

BERT (2018)

Practical considerations:

- Huge model
  - Data: Wikipedia (2.5B words) + BookCorpus (800M words)
  - Trained on TPU for 4 days
- Fine tuning requires GPU
- Possibly too heavy for production
  - processes 4 sentences per second
- Limited to 512 input tokens
- Throws out OOV tokens

```
import pytorch_pretrained_bert
```

Conclusions

- In recent years there was a big jump in the performance of NLP algorithms
- They work pretty well out-of-the-box - even without extensive hyperparameter optimization
- Well documented code is available for all - you can be up and running within hours or days
- State-of-the-art models require a GPU
- ULMFiT strikes a good balance between classification performance and computational requirements

Questions?