Text Classification – Zero to Hero

The 1st NLP Day, November 2019

Dr. Omri Allouche, omri.allouche@gmail.com

Omri Allouche

Head of Research, Gong.io

Senior Teacher,

Bar Ilan University

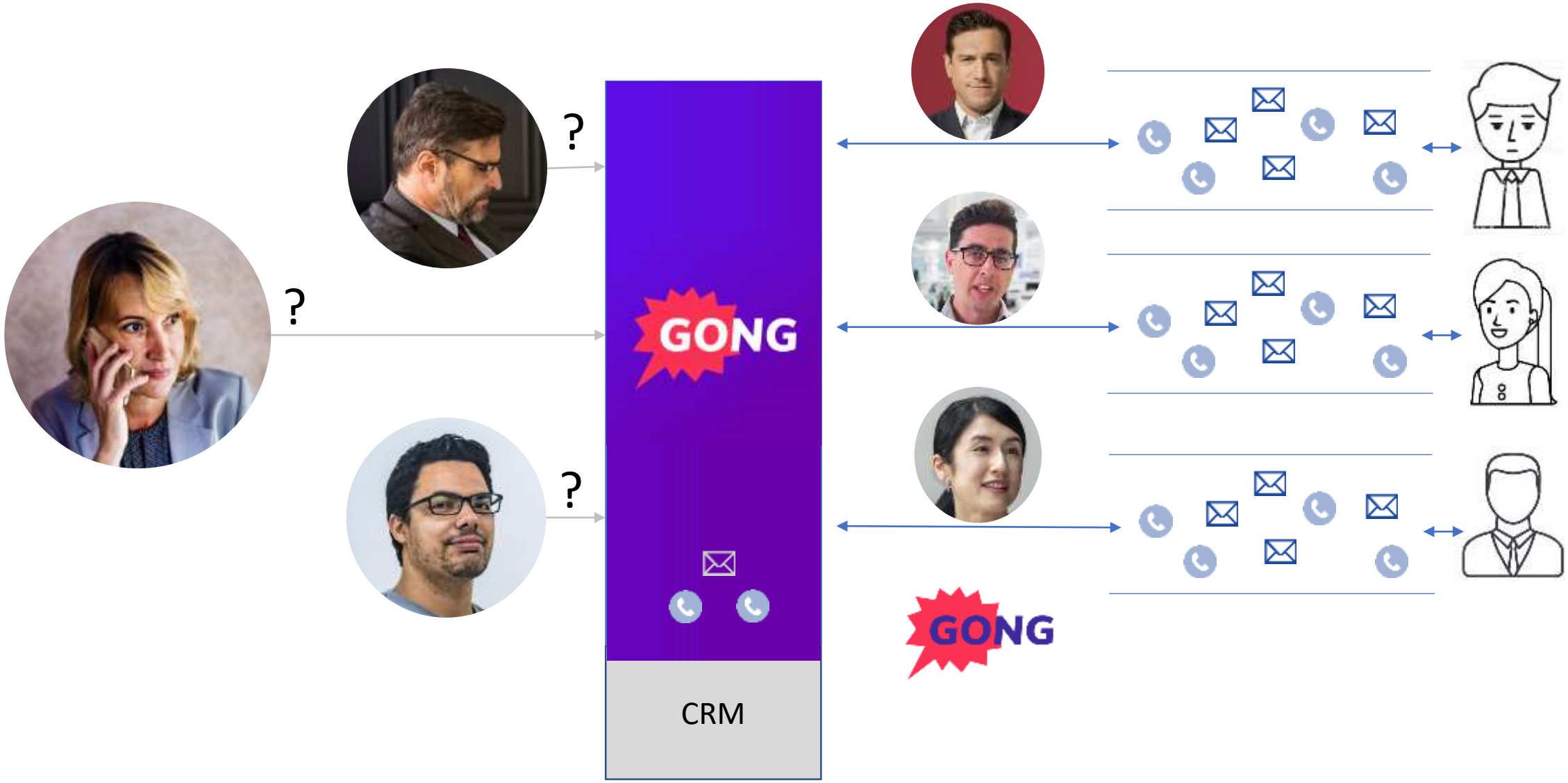
The Yandex Y-Data program

Founder of Otonomic (acquired by Algomizer)

PhD in Computational Ecology from HUJI

Happy to connect on LinkedIn & facebook

The Gong Product



Understanding Spoken Language

50-minute phone conversation

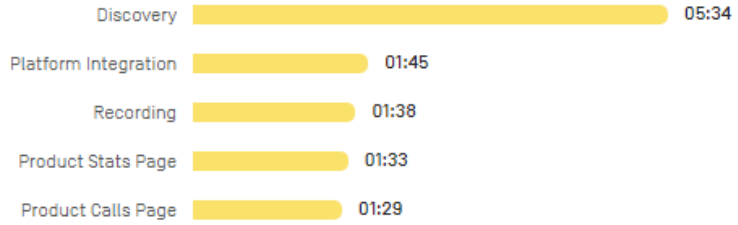
Gong.io - Understanding Spoken Language

50-minute phone conversation

How did the conversation progress?

Who spoke when?

What was discussed?



What was said?

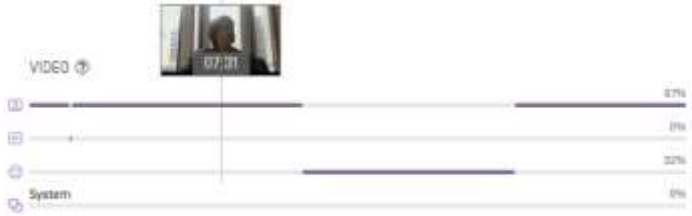
JAMESON

Glad we got connected this time. I'm sorry about last time.

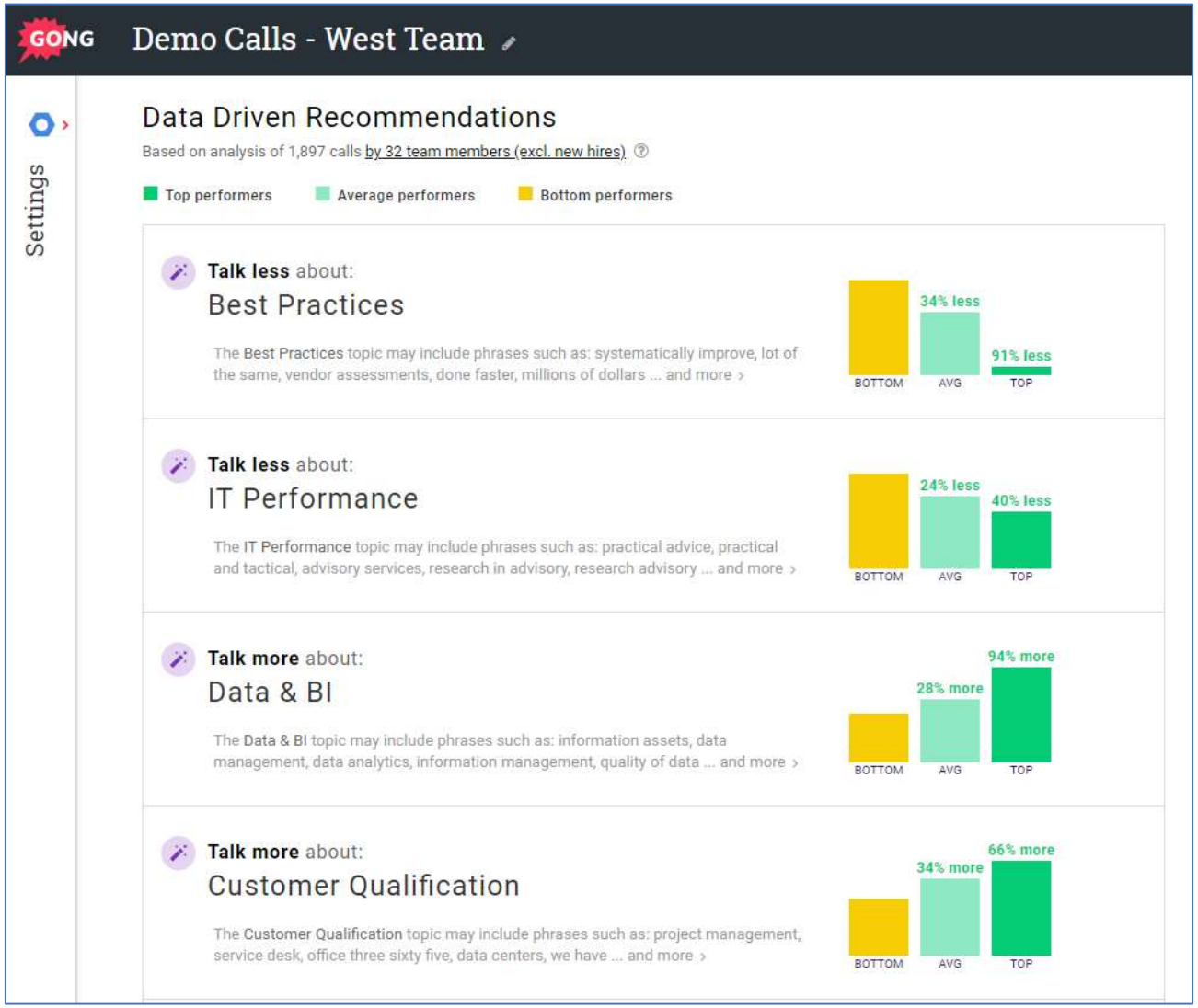
MICHAEL

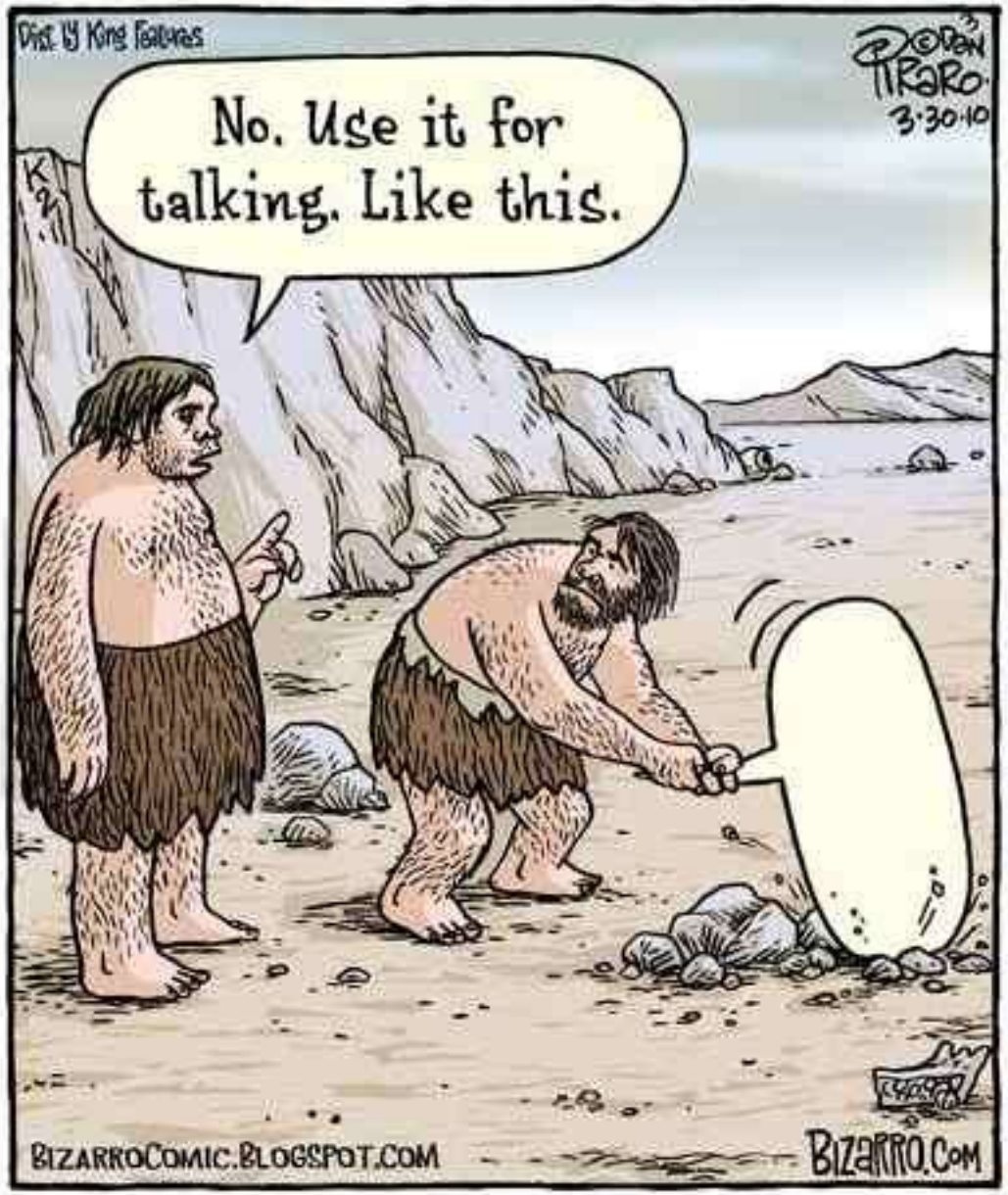
Oh, that's okay. No problem. I apologize that we just moved to a new house. I have bunch of contractors at the house. So I apologize for the background. Noise.

What was shown?



Gong.io - Sample Business Insight




Dist. by King Features
No. Use it for talking. Like this.
DAN PIRARO
3-30-10
BIZARROCOMIC.BLOGSPOT.COM
BIZARRO.COM

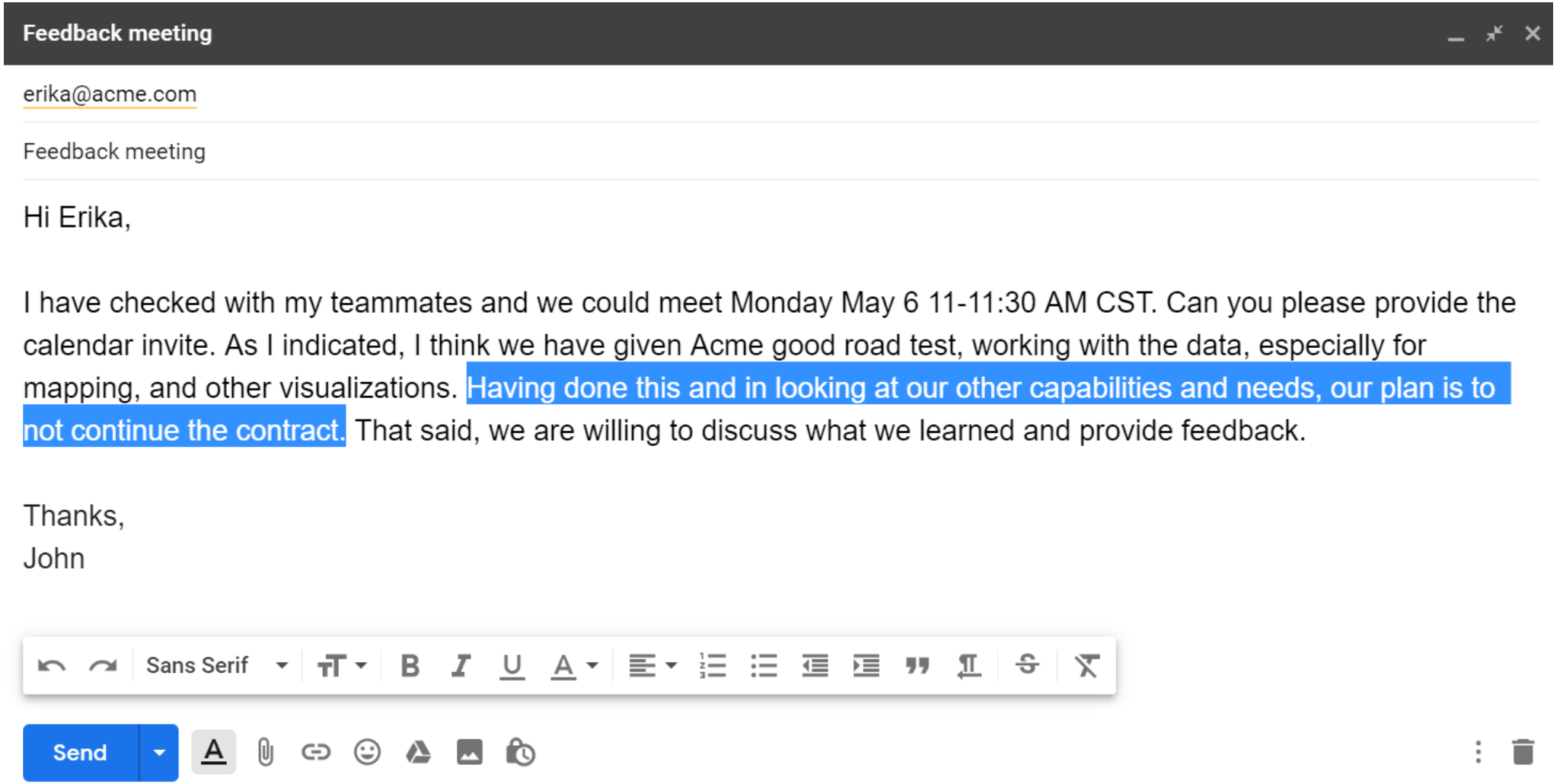

THE ORIGIN OF LANGUAGE
CLUB
CUPCAKE
CLUB!
CUPCAKE!
BONK
CLUB
CLUB
© 2010 GONZALEZ


DID YOU KNOW THAT SUBURBAN WHITE MALES HAVE OVER 100 WORDS FOR "LAWN"?
7-29
SPEEDBUMP.COM

I'm sorry, but shouldn't there be an apostrophe in that?


NIGHT SCHOOL ENROLMENT
LEARN FRENCH
LEARN GERMAN
LEARN TEXT MESSAGE
John Hill
CARTOONSTOCK .com
Search ID: jhi0082

ML Example - Email Tagging



Emails - Suggested Tags

1. Scheduling
    a. Request for meeting (also includes "let's set up a demo", "let's set up a call")
    b. Request for rescheduling
2. Nudge (reminder)
3. Introduction of a new person
4. Objections/concerns
5. Request for information about the product by the client
6. Product features information
7. Pricing request by the client
8. Pricing info by the seller
9. Proposal / commercial discussion
10. Request for client signature to close the deal
11. Technical issues

Today

1. We'll start with the basics…
2. … and quickly move to State-of-the-Art models
3. We'll have one lecture covering the material, followed by hands-on competition
4. We have a lot to cover – we'll cover the most important parts

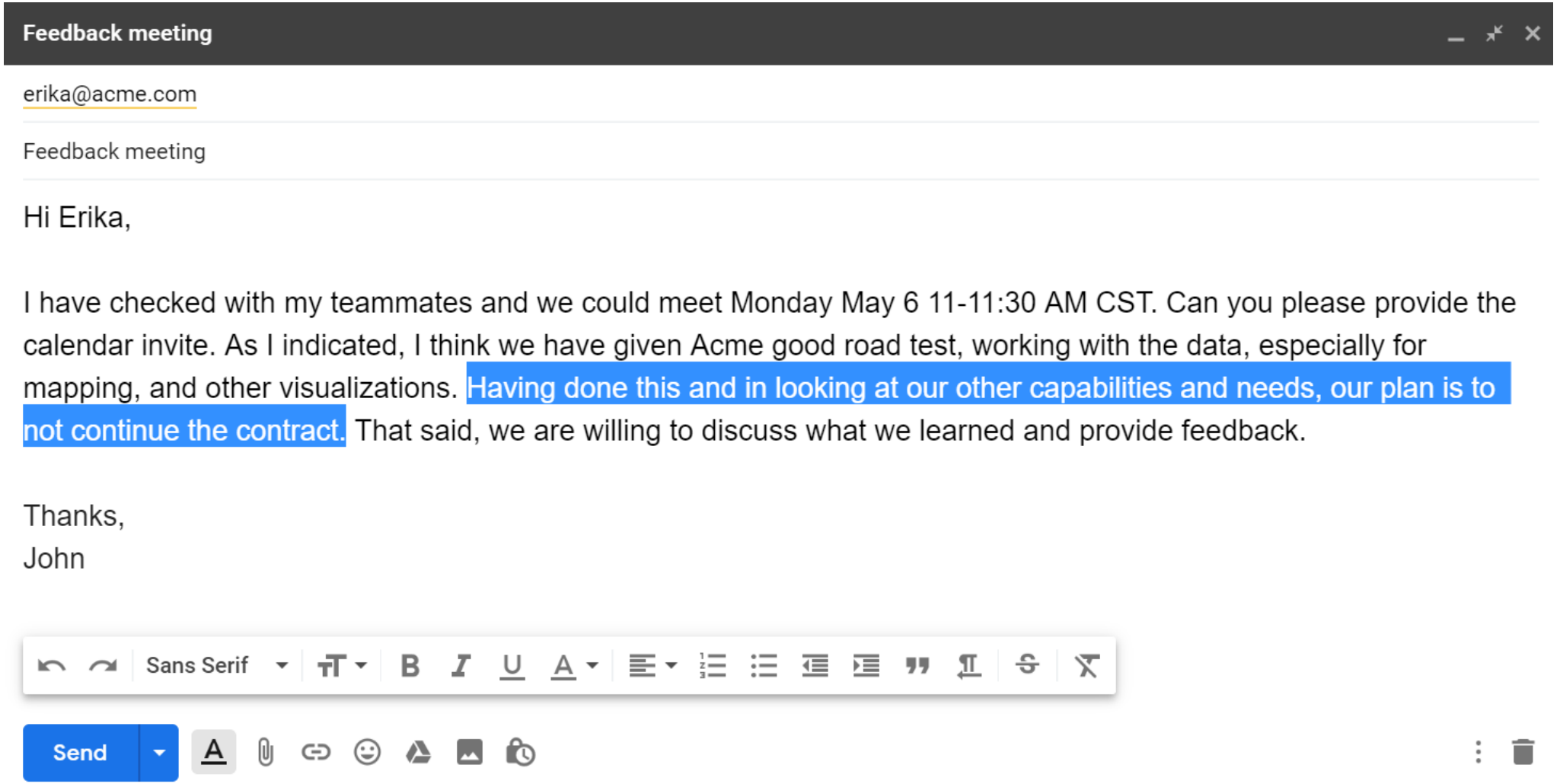
The Notebooks

1. Bag of Words and Tf-Idf
2. Word embeddings
   2a. Optional: Train word embeddings
   2b. Optional: Advanced sentence embedding methods in Flair
3. Contextual embeddings with ELMo
   3a. Optional: Contextual word vectors with BERT and stacking embeddings
4. (Optional) Fine tuning a Language Model with ULMFiT
5. State-of-the-art Transformer with BERT

Our Dataset

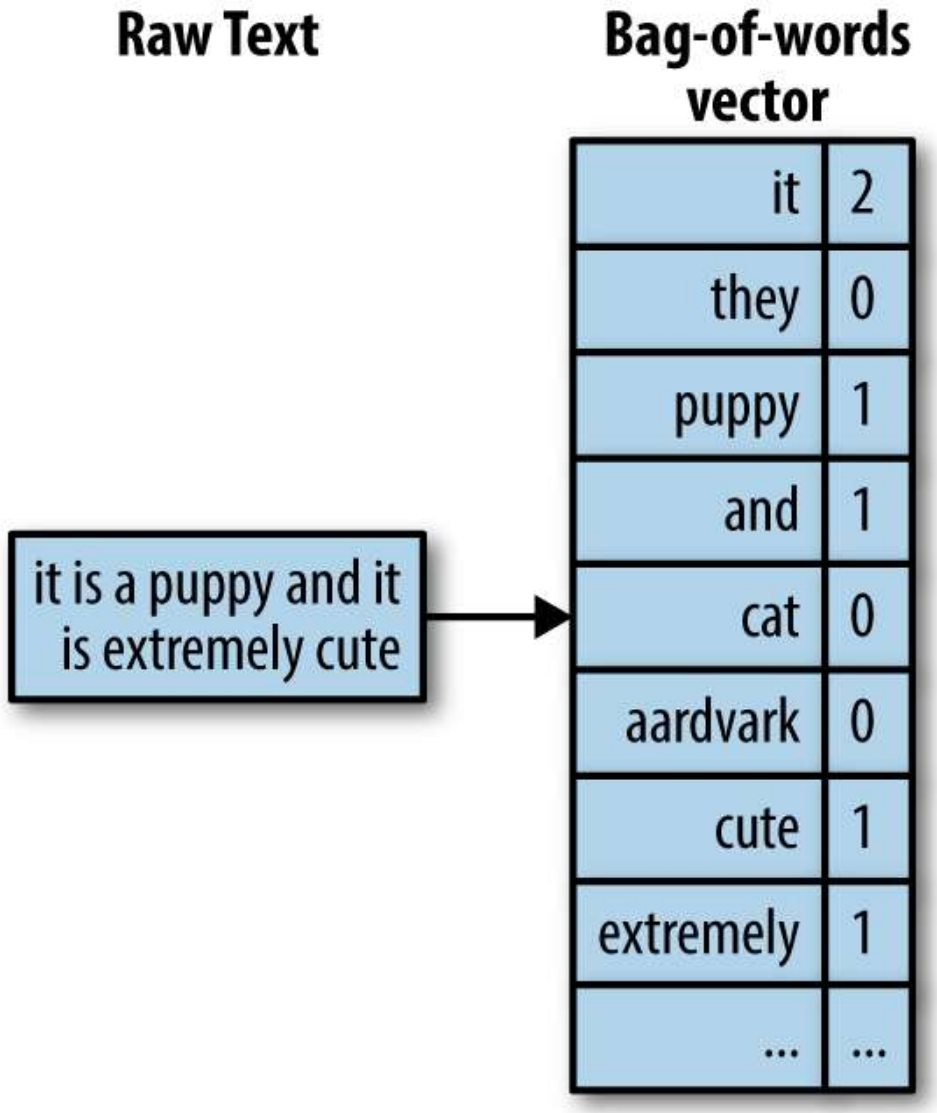
- Posts from 4 different newsgroups
  - rec.sport.baseball
  - rec.sport.hockey
  - talk.politics.guns
  - talk.politics.Mideast
- Cut to length of 200-400 characters
- Metadata was removed
- This is a subset of a larger dataset of 20 newsgroups

ML Example - Email Tagging



Text Classification – Bag of Words

- Count the number of times each word appears in the text
- Disregard word order
- Represent the text as a vector of length V (size of the vocabulary)
  - Vocabulary size can be millions. We often use only the most common words (e.g. top 30,000 words)
  - Some words might be less informative (e.g. "a", "the") – they are called *stopwords* and are often removed (though be cautious, as they might be critical for some tasks!)
- Use Logistic Regression, Naïve Bayes or any other ML algorithm to fit the data



TF-IDF

TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t)$$

Term frequency

Number of times term t appears in a doc, d

Inverse document frequency

$$\log \frac{1 + n}{1 + \text{df}(d, t)} + 1$$

n ← # of documents

df(d, t) ↖ Document frequency of the term t

Common Tf-Idf Formulas

Variants of term frequency (tf) weight

weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \lvert\{d \in D : t \in d\}\rvert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(\frac{N}{1 + n_t}\right) + 1$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

Code Sample

- Code is in the notebook *"bow_tfidf"*
- We'll use sklearn. The Vectorizer conforms to the fit+transform API of sklearn
- The notebook includes code for analyzing model performance

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english')

df_train = pd.read_csv('data/train.csv')
vectors = vectorizer.fit_transform(df_train['text'])
vectors.shape
```

Classification Report

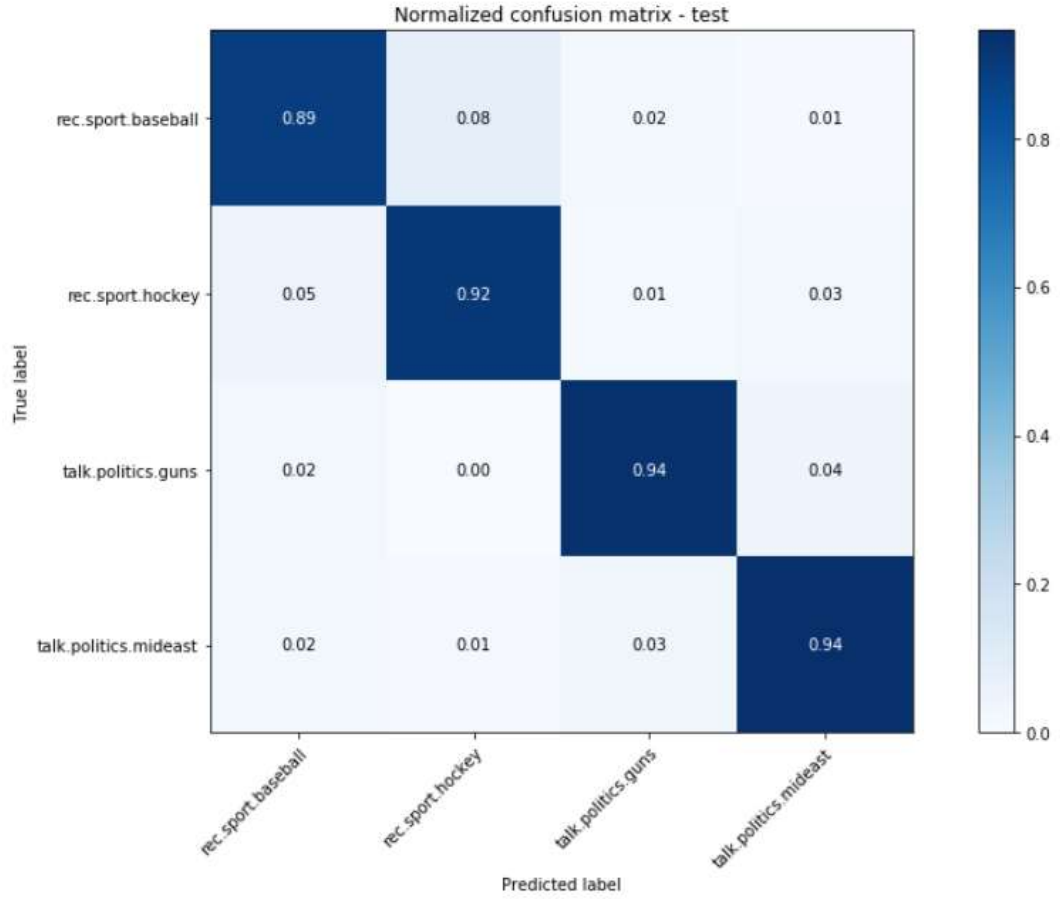
```
from sklearn import metrics
print(metrics.classification_report(y_truth, y_predict))
```

	precision	recall	f1-score	support
rec.sport.baseball	0.92	0.89	0.90	155
rec.sport.hockey	0.91	0.92	0.91	153
talk.politics.guns	0.93	0.94	0.93	125
talk.politics.mideast	0.92	0.94	0.93	127
micro avg	0.92	0.92	0.92	560
macro avg	0.92	0.92	0.92	560
weighted avg	0.92	0.92	0.92	560

Confusion Matrix

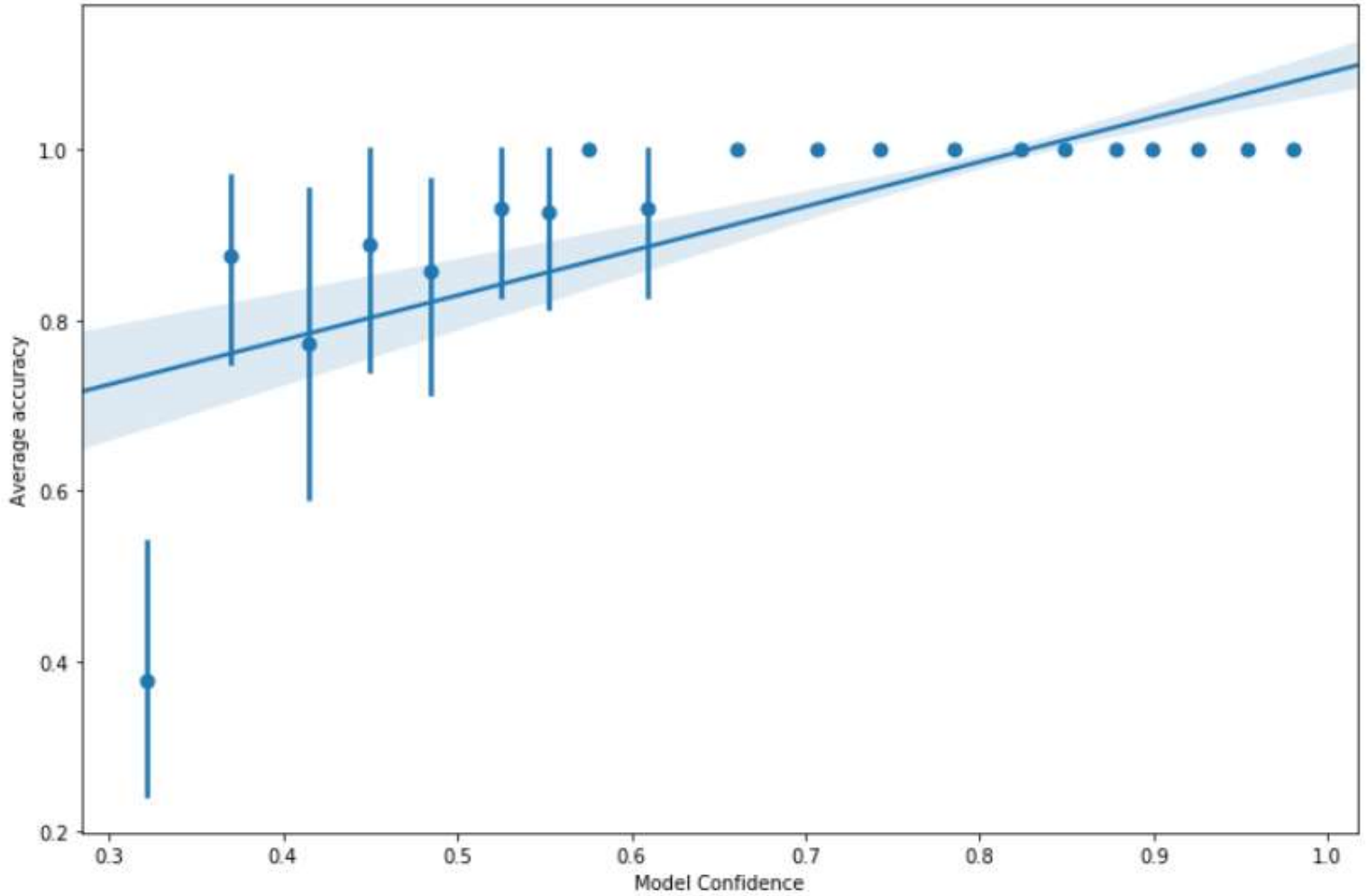
```
plot_confusion_matrix(y_truth, y_predict, normalize=True, title='Normalized confusion matrix - test');
```

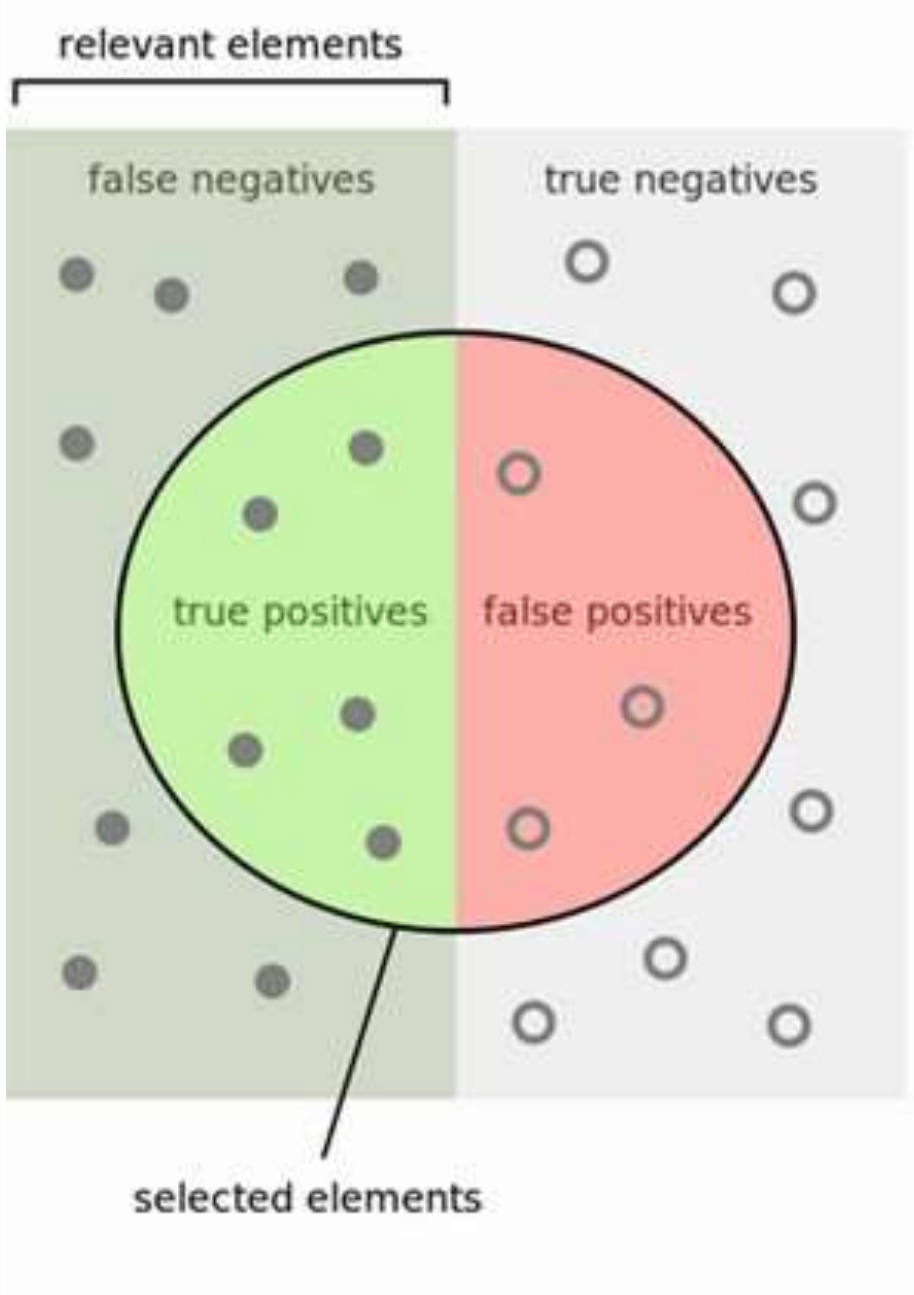
```
Normalized confusion matrix
[[0.89032258 0.08387097 0.01935484 0.00645161]
 [0.04575163 0.91503268 0.0130719  0.02614379]
 [0.024      0.         0.936      0.04      ]
 [0.01574803 0.00787402 0.03149606 0.94488189]]
```



Confidence-Accuracy Graph

```
plot_confidence_performance(y_predict, y_predict_proba, y_truth, num_bins=20);
```





How many selected items are relevant?

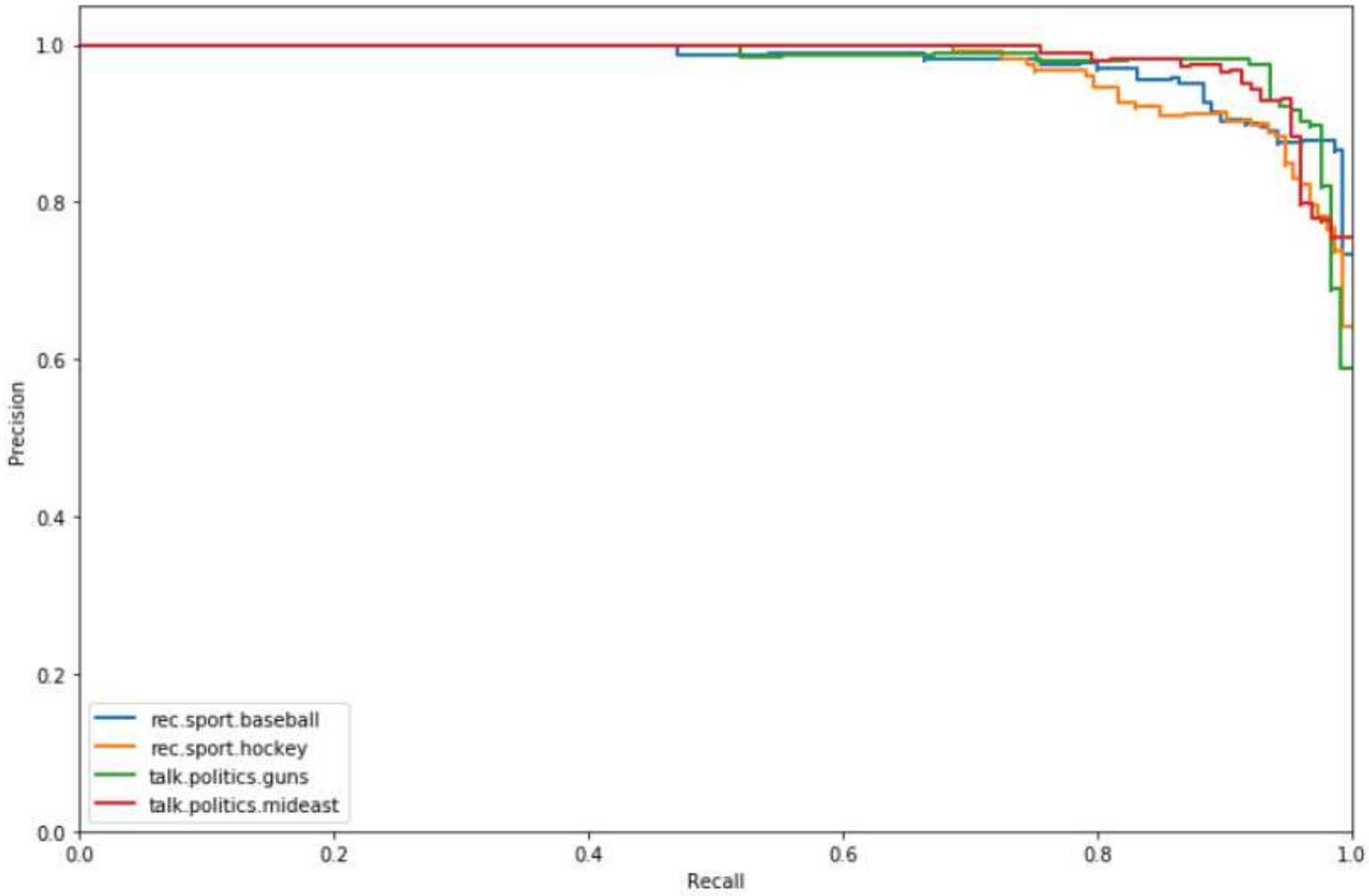
$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Precision-Recall Curve

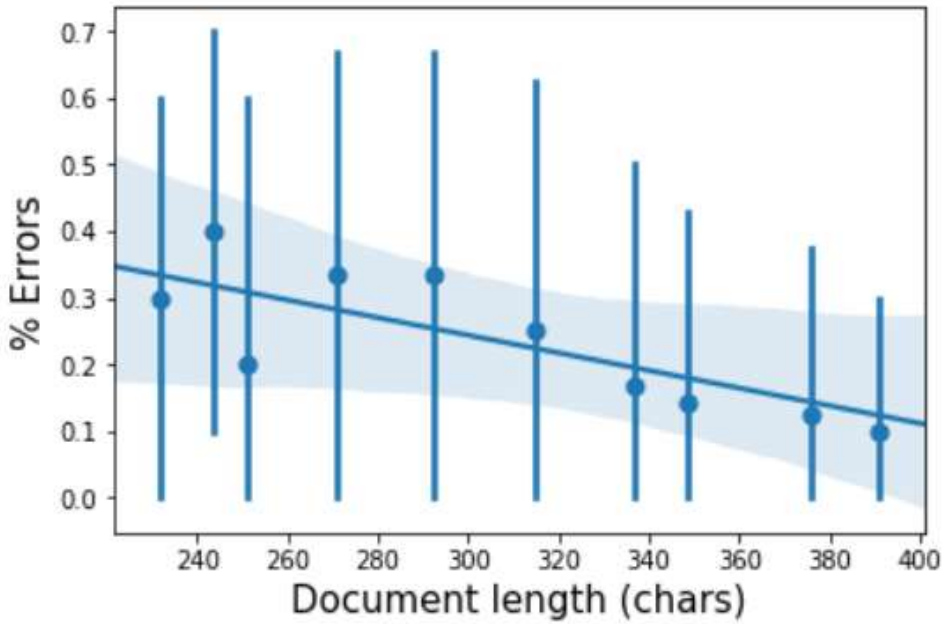
```
plot_precision_recall_curve(y_predict_proba, y_truth);
```



Error Analysis

- It's very useful to perform an error analysis for our basic model
- Running on the entire dataset of all document lengths, we see that we mostly make errors for short documents

```
ax = sns.regplot(data=df_test, x='num_chars', y=df_test['is_error'], x_bins=10)
plt.xlabel('Document length (chars)');
plt.ylabel('% Errors');
```



Reviewing Specific Documents

```
from ipywidgets import interact

@interact(i=(0,len(df_test)))
def explore_doc(i):
    print_document(df_test.iloc[i])
```


```
Document #31.
True label: talk.politics.guns.
Predicted: talk.politics.mideast (confidence: 28.2%)

. then it also supports basing such regulations on ignorance. miller had
disappeared, and nobody bothered to present his side to the supreme court
in particular, that sawedoff shotguns were used in the world war i trench
es, and in other tight spots ever since guns had been invented. would you
turn one down if you had to clean an alley in e. st. louis?
```

Bag of Words - Problems

- We disregard the order of words
- Words we haven't seen before will be treated as a special token (UNK)
- Very rare word (e.g. a word that only appears once in the training set) will also be treated as UNK
- *Cat ≠ Cats ≠ Kitten*

Bag of Words – One-Hot Encoding

Problem:

- We use *one-hot encoding* of words:

  x_{cat}: [0 0 0 0 1 0 0 0 0]

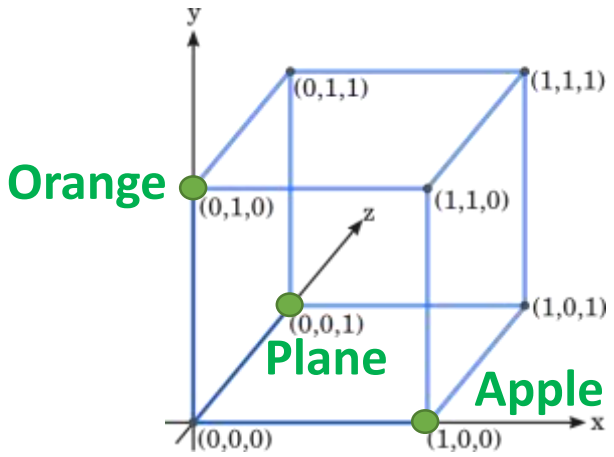
  x_{kitten}: [0 0 0 0 0 0 1 0 0]

  x_{teapot}: [0 1 0 0 0 0 0 0 0]

  and so treat words separately

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

One-Hot vectors



One Hot Encoding - Alternatives

Solution:

- Lemmatization / Stemming
- Use a dictionary of synonyms (e.g. WordNet)
- Encode words as fixed-size vectors
  - Words with the same meaning can replace each other
  - What if we can map similar words to similar low-dimension vectors?

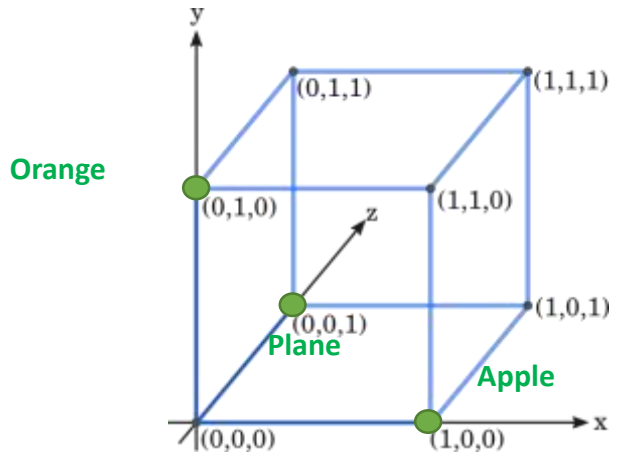
The Distributional Hypothesis (D.H.) (Harris, 1954):

"words are characterized by the company that they keep"

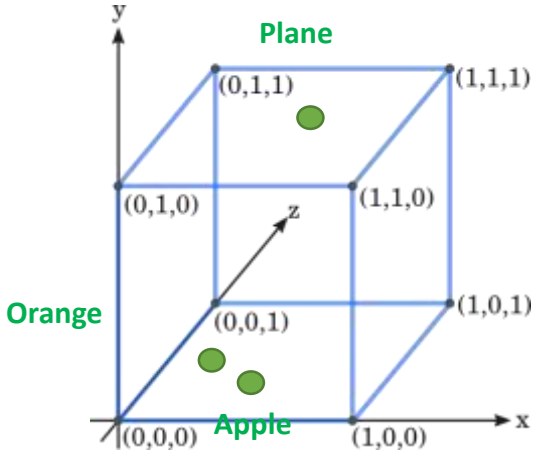
Words used in a similar context should have similar vectors

"She swoomped him with her frying pan" (G. Hinton)

One-Hot vectors



Wanted behavior



Word Embeddings

Co-occurrences

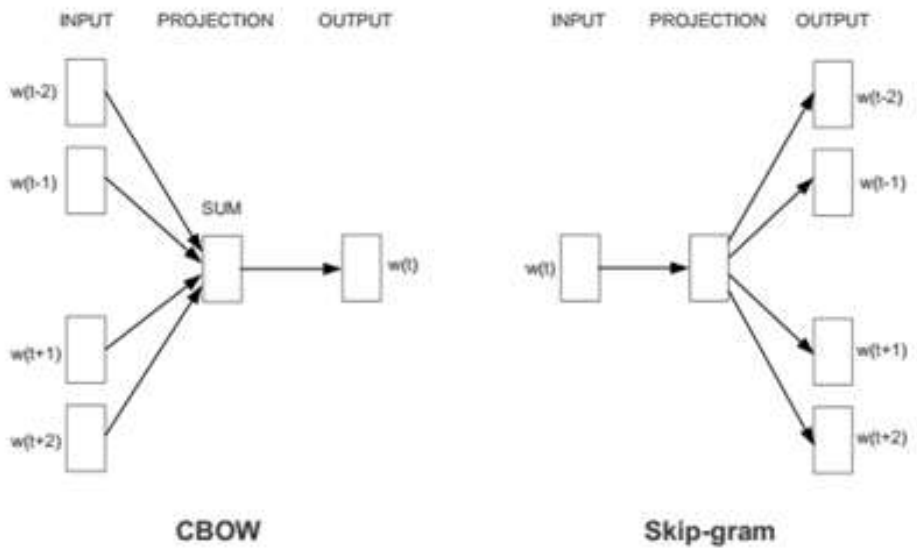
- We wish to find words that appear **frequently together**, and **infrequently** in other contexts.

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j)}{\text{count}(w_i) \times \text{count}(w_j)}$$

- Think: What type of similarity does co-occurrence imply?

Distributed Word Representations

- Continuous Bag–of–Words (CBOW):
  - As in N – Gram, predicts a word given its context (bidirectional).
- Skip-gram:
  - Opposite from N – Gram, predicts the context given a word (bidirectional).

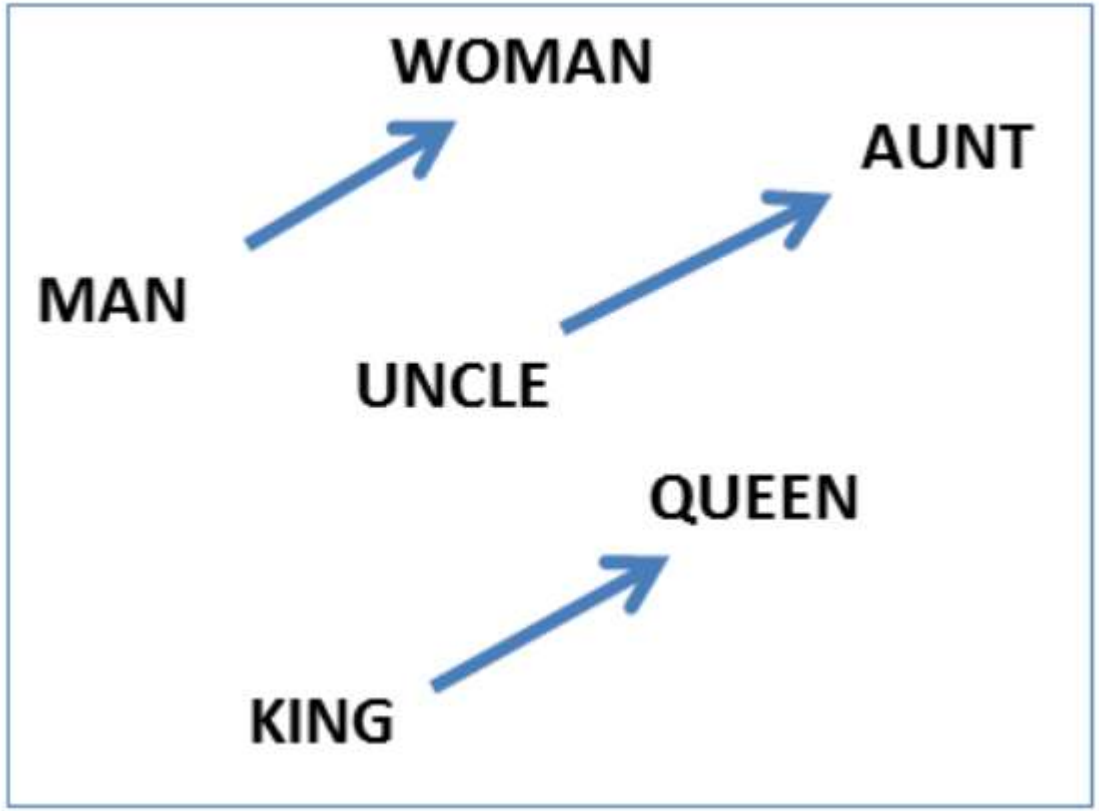


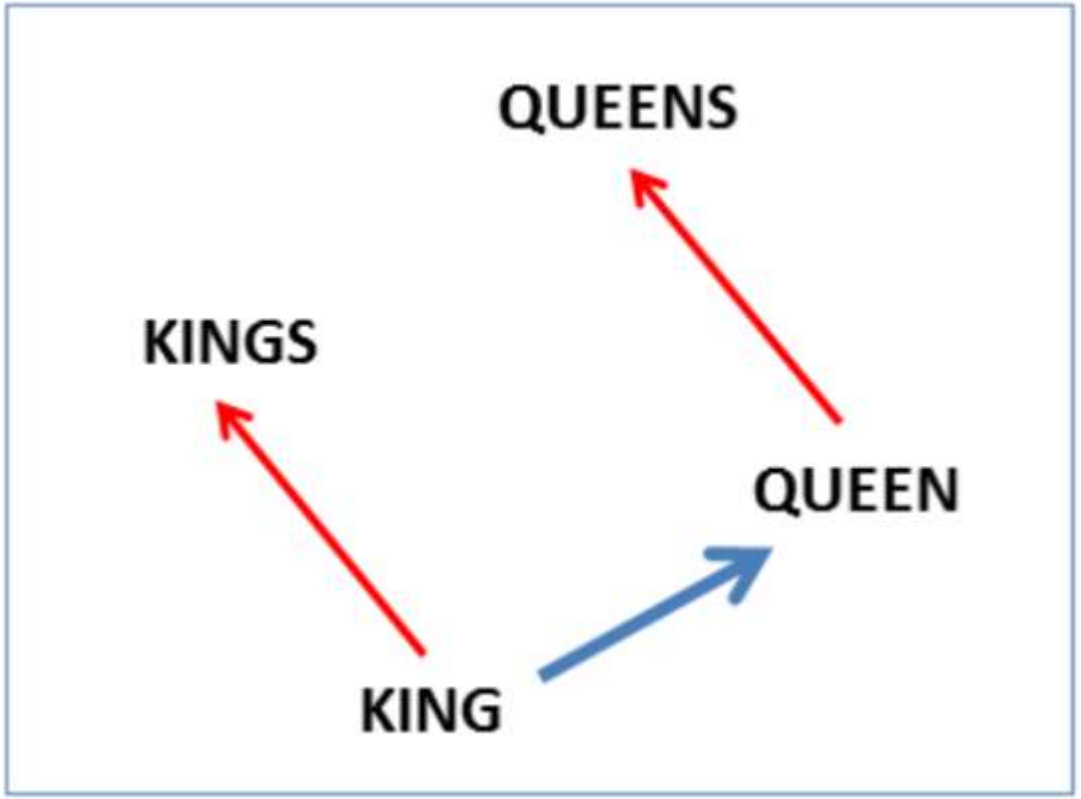
Mikolov at al., 2013

Benefits of Word Vectors

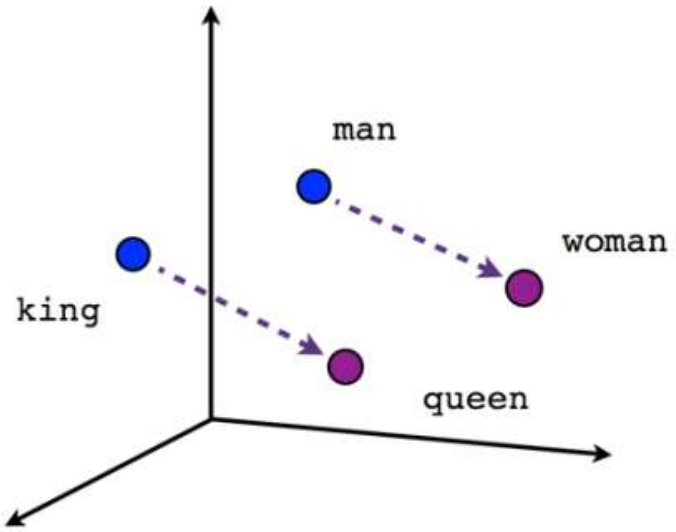
- Much smaller in size than One-Hot encoding
- Better similarity measures
- Distributed representation. Might represent meaningful features like Part-of-Speech, singular/plural, gender etc.
- Probably better generalization

Word Vector Algebra

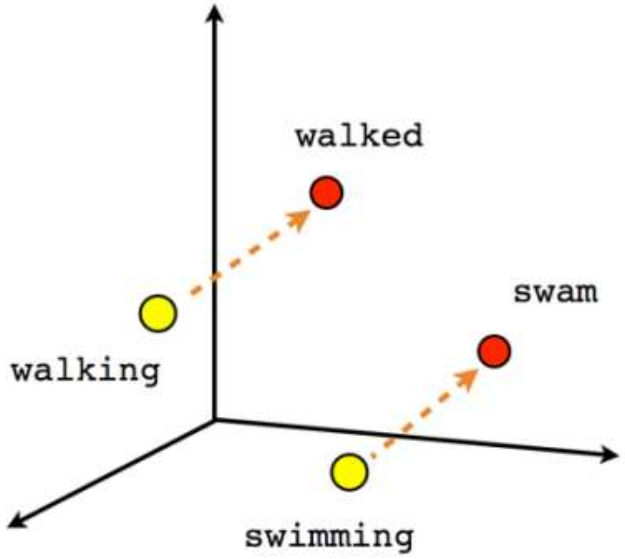




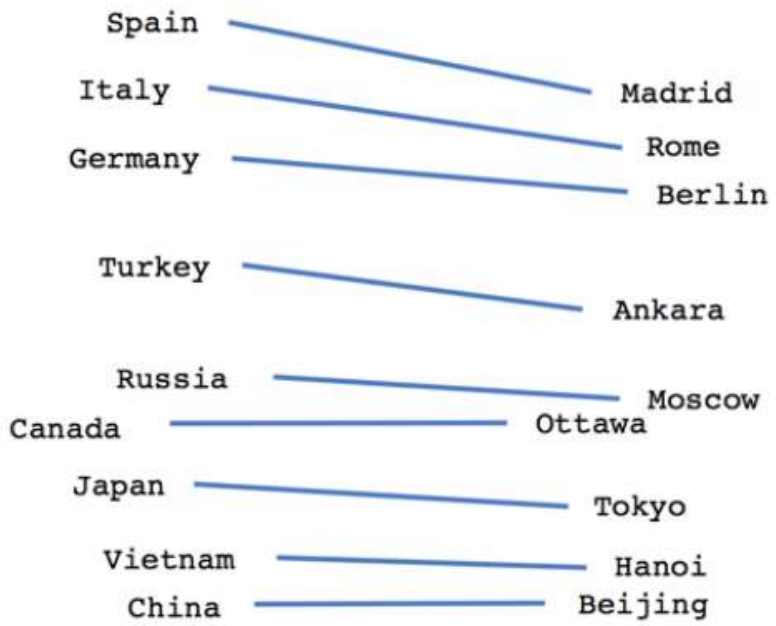
Word Vector Algebra



Male-Female



Verb tense

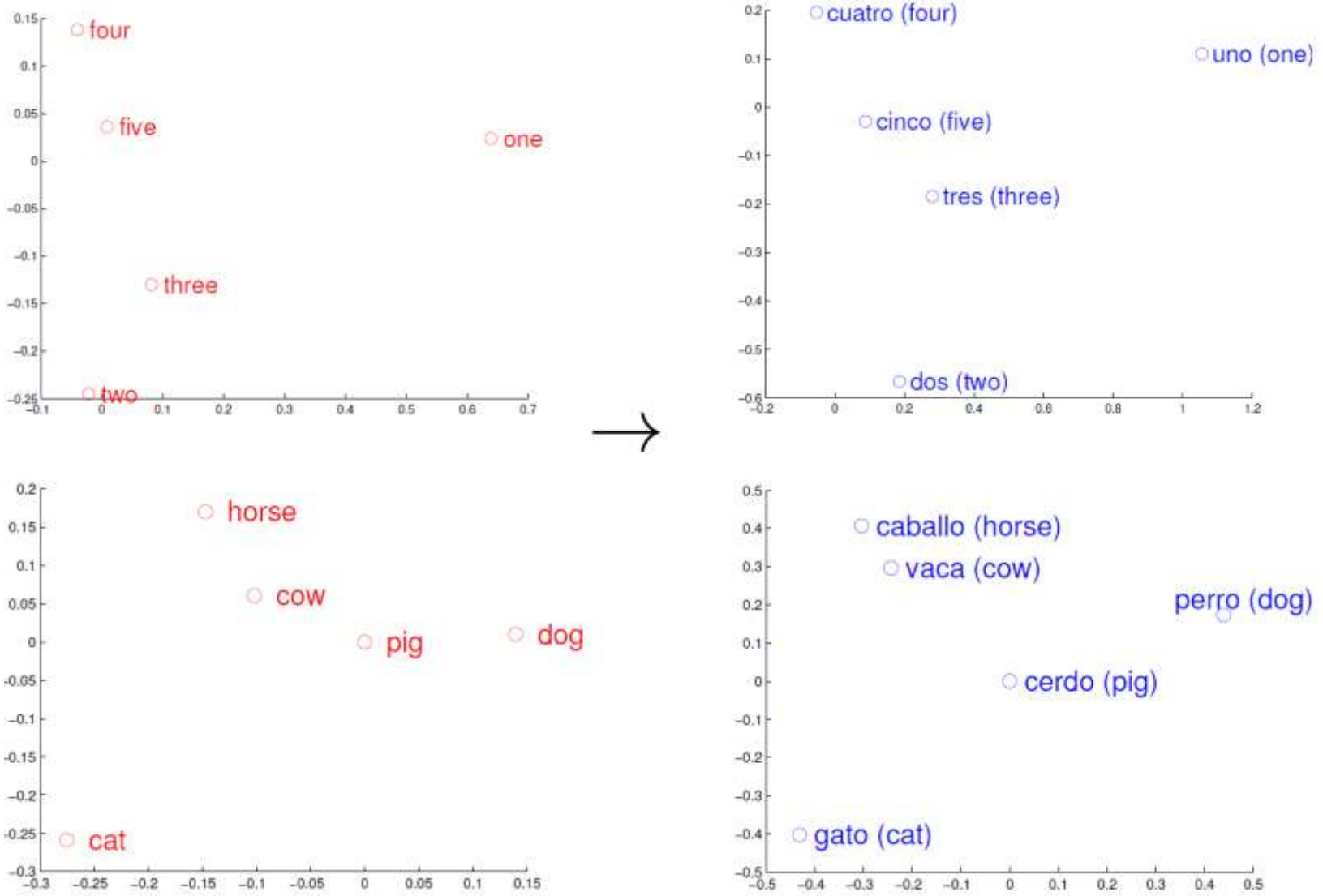


Country-Capital

vector[Queen] = vector[King] - vector[Man] + vector[Woman]

Similar relations across languages

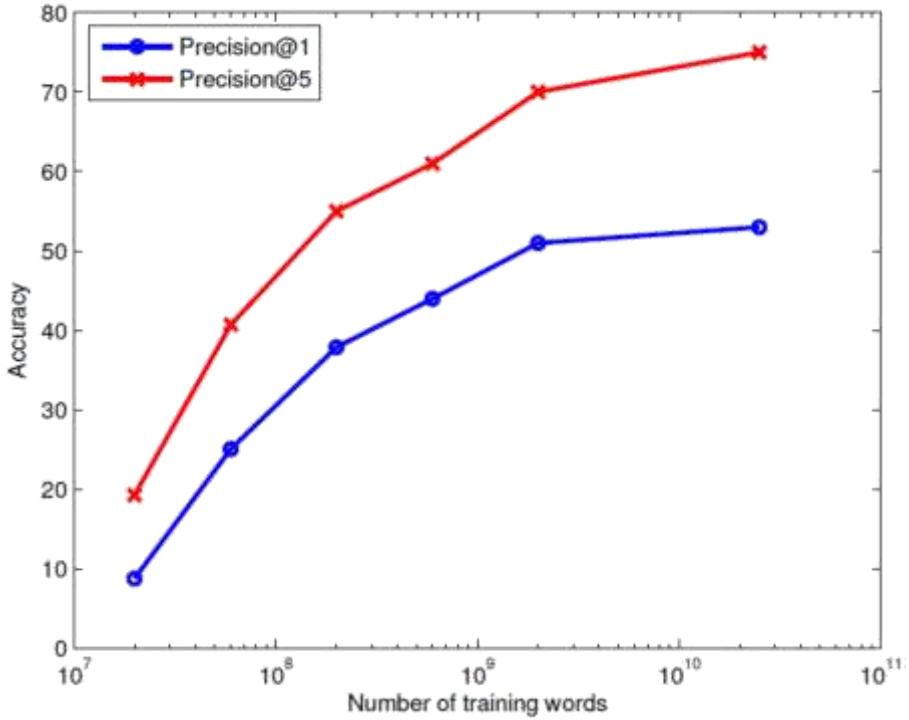
- The dimensionality of the embeddings was reduced to 2 using PCA
- The vectors were then rotated for maximal similarity



Mikolov et al. 2013

Cross-lingual

- Train on 5K most frequent



Influence of word2vec training set

Spanish word	Computed English Translations	Dictionary Entry
emociones	emotions emotion feelings	emotions
protegida	wetland undevelopable protected	protected
imperio	dictatorship imperialism tyranny	empire
determinante	crucial key important	determinant
preparada	prepared ready prepare	prepared
millas	kilometers kilometres miles	miles
hablamos	talking talked talk	talk
destacaron	highlighted emphasized emphasised	highlighted

Mikolov et al. 2013

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Canadiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon

Table 2: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples). The goal is to compute the fourth phrase using the first three. Our best model achieved an accuracy of 72% on this dataset.

Code: Word Embeddings Using Flair

```
from flair.embeddings import Sentence, WordEmbeddings
glove_embedding = WordEmbeddings('glove')

sentence = Sentence('The grass is green .')

glove_embedding.embed(sentence)
for token in sentence:
    print(token)
    print(token.embedding)
```

- A full list of word embeddings available in Flair is here.

Code: Training Our Own Word Embeddings

- The notebook *"train_word_embeddings.ipynb"* demonstrates training word vectors on our own corpus.

```
import gensim
w2v = gensim.models.Word2Vec([s.split() for s in df['text'].values,
                              iter=50,
                              sg=1,
                              min_count=5,
                              size=100,
                              window=3,
                              workers=7)
w2v.init_sims(replace=True) # frees memory of word vectors but prevents further training
```

Practical Advice

- It's simple to train your own vectors
  - e.g. using genism in python - https://radimrehurek.com/gensim/models/word2vec.html
- Pretrained vectors are a good starting point in many cases, if your data isn't very domain-specific
  - Can be frozen, or used as initial weights for a learned embedding matrix
- It serves as a good tool to get to know your data
  - Which words are closest to a certain term?
- Often very useful as a tool for exploration and guided search
  - E.g. "smart search" – suggest other search terms
  - Find competitors of a company
  - Cluster words
  - …

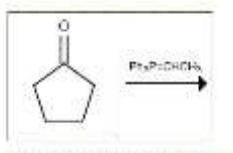
Fitting the problem domain

- A word embedding model trained on a medical domain will be **quite different** for a model trained on a e-commerce domain.
- One should build a **dedicated** vector representation usually by starting from a **pre-trained general-purpose** embedding model.

Comparison of Results Related words to "*ring*"

Wikipedia

Word
bell
tone
token
five-membered
cage
signet
wrestling
tag
wedding
inside



Pinterest

Word
engagement
wedding
promise
sterling
rings
knuckle
silver
band
stacking
pillow

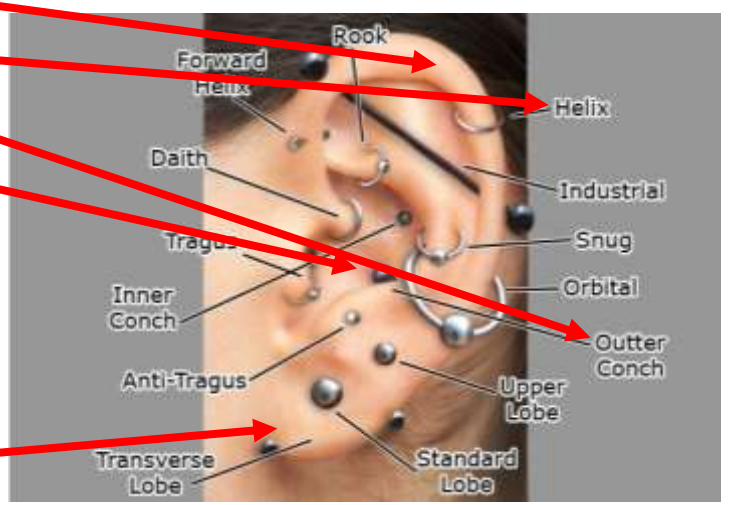
Comparison of Results - piercing

Wikipedia

Word
piercings
pierce
penetrating
sucking
pierced
armour
ear
armor
forehead
tattooing

Pinterest

Word
cartilage
helix
conch
tragus
16g
earring
navel
bellyring
lobe
barbell



Sentence Embedding

Sentence Embedding

- Generally problematic, as the same text can have different emphasis in different scenarios
- A few baselines to consider:
  - Average word vectors
  - Average word vectors, inversely weighted by prevalence (so common words have less influence)
  - Learn a vector for each sentence during training (aka doc2vec, paragraph2vec)
  - Use the final hidden state of a RNNLM as an embedding of the entire document

Sentence Embedding Using Flair & Numpy

```
import numpy as np
def get_sentence_embedding(sentence):
    sentence = Sentence(sentence)
    glove_embedding.embed(sentence)
    sentence_embedding = np.mean( [np.array(token.embedding) for token in sentence], axis=0)
    return sentence_embedding
```

Contextualized Word Embeddings with ELMo

Contextualized Word Representations

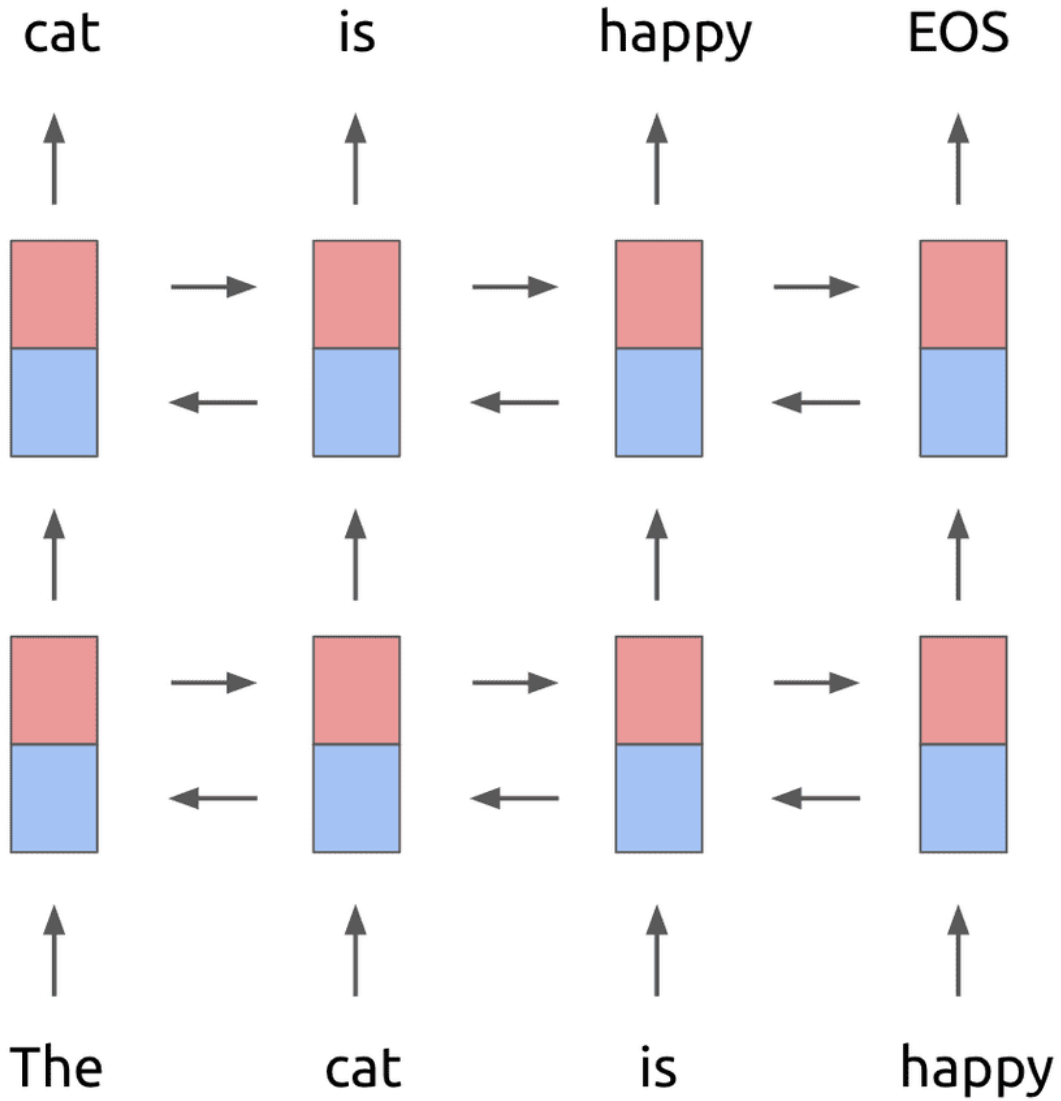
- The meaning of a word is **context-dependent;** their embeddings should also take context into account
  - "The bank on the other end of the street was robbed"
  - "We had a picnic on the bank of the river"
- Embeddings from Language Model (ELMo) uses **language models** to obtain embeddings for individual words while taking the entire sentence or paragraph into account

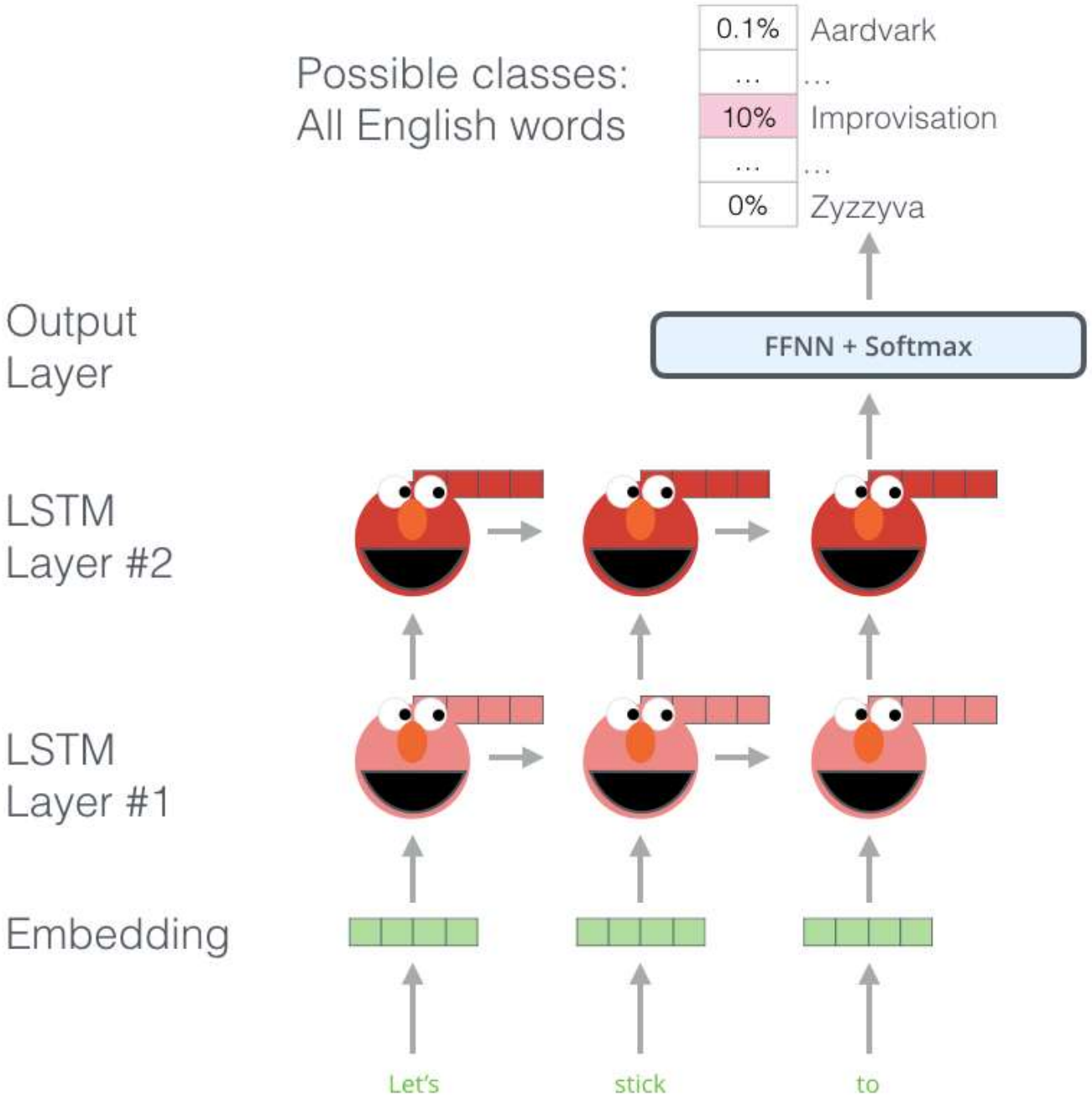
Contextualized Word Representations

- Concretely, ELMos use a pre-trained, multi-layer, bi-directional, LSTM-based language model and extract the **hidden state** of each layer for the input sequence of words
- ELMo improves the performance of models across a wide range of tasks, spanning from question answering and sentiment analysis to named entity recognition

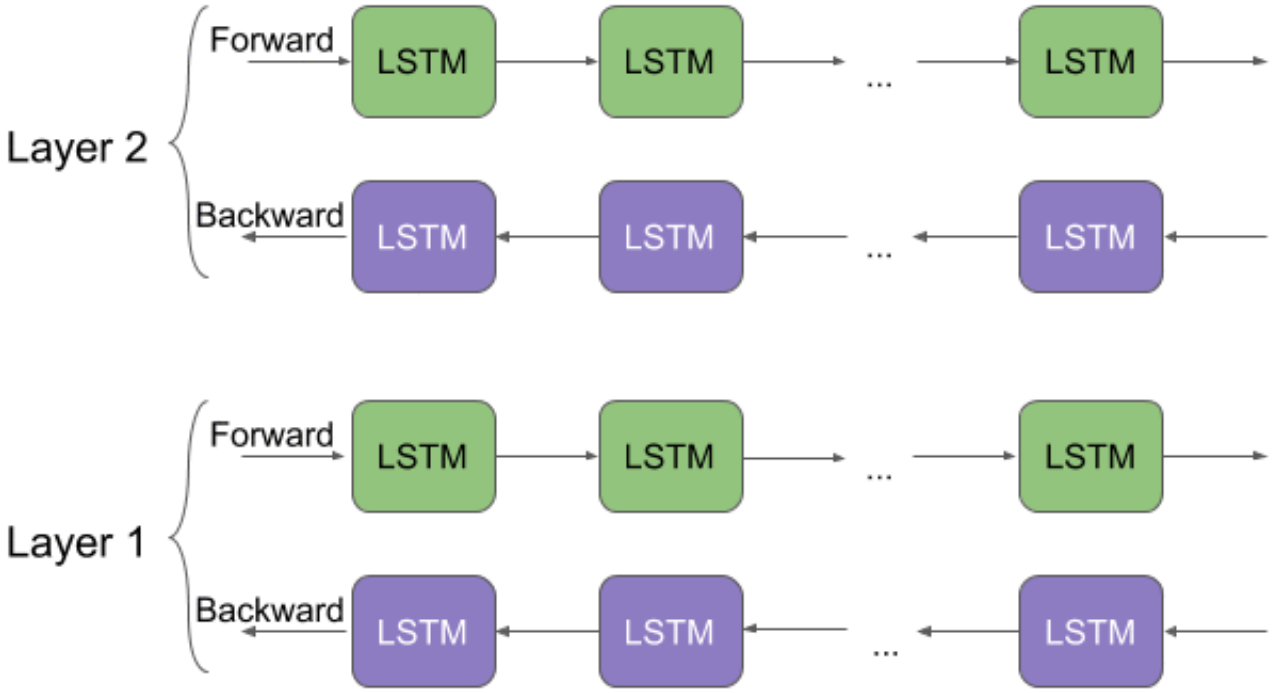
ELMo

- Train a Language Model on a large corpus (1B Word Benchmark)
- Train using a 2-layer LSTM

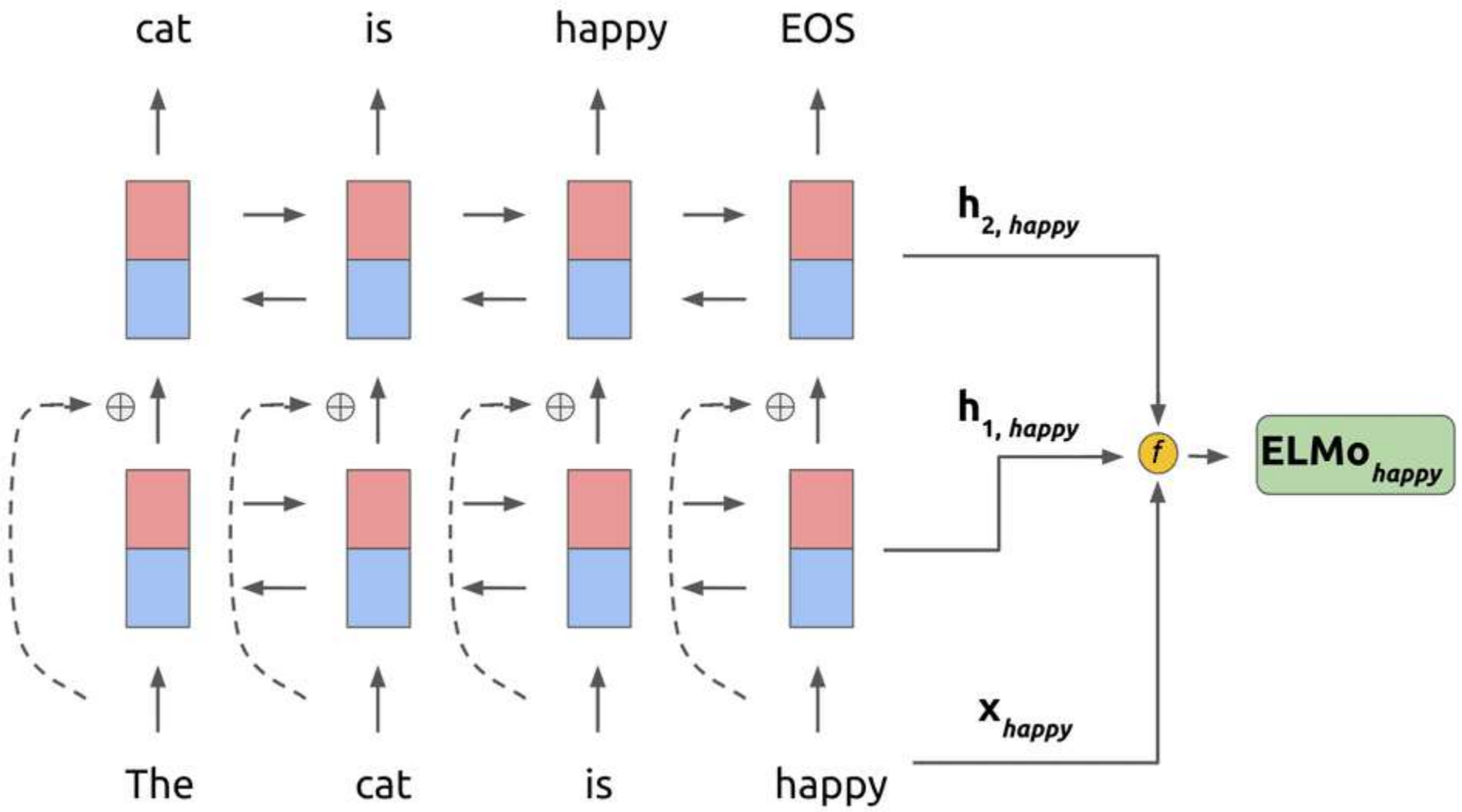



Possible classes:
All English words
0.1%
Aardvark
...
...
10%
Improvisation
...
...
0%
Zyzzyva
Output Layer
FFNN + Softmax
LSTM Layer #2
LSTM Layer #1
Embedding
Let's
stick
to

ELMo embeddings are comprised of multiple layers

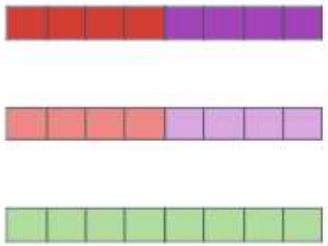


ELMo embeddings are comprised of multiple layers

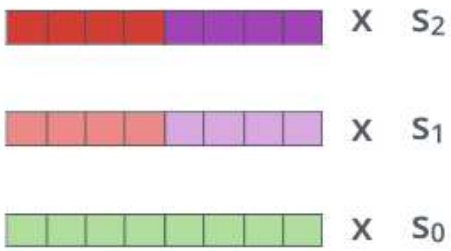


Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers

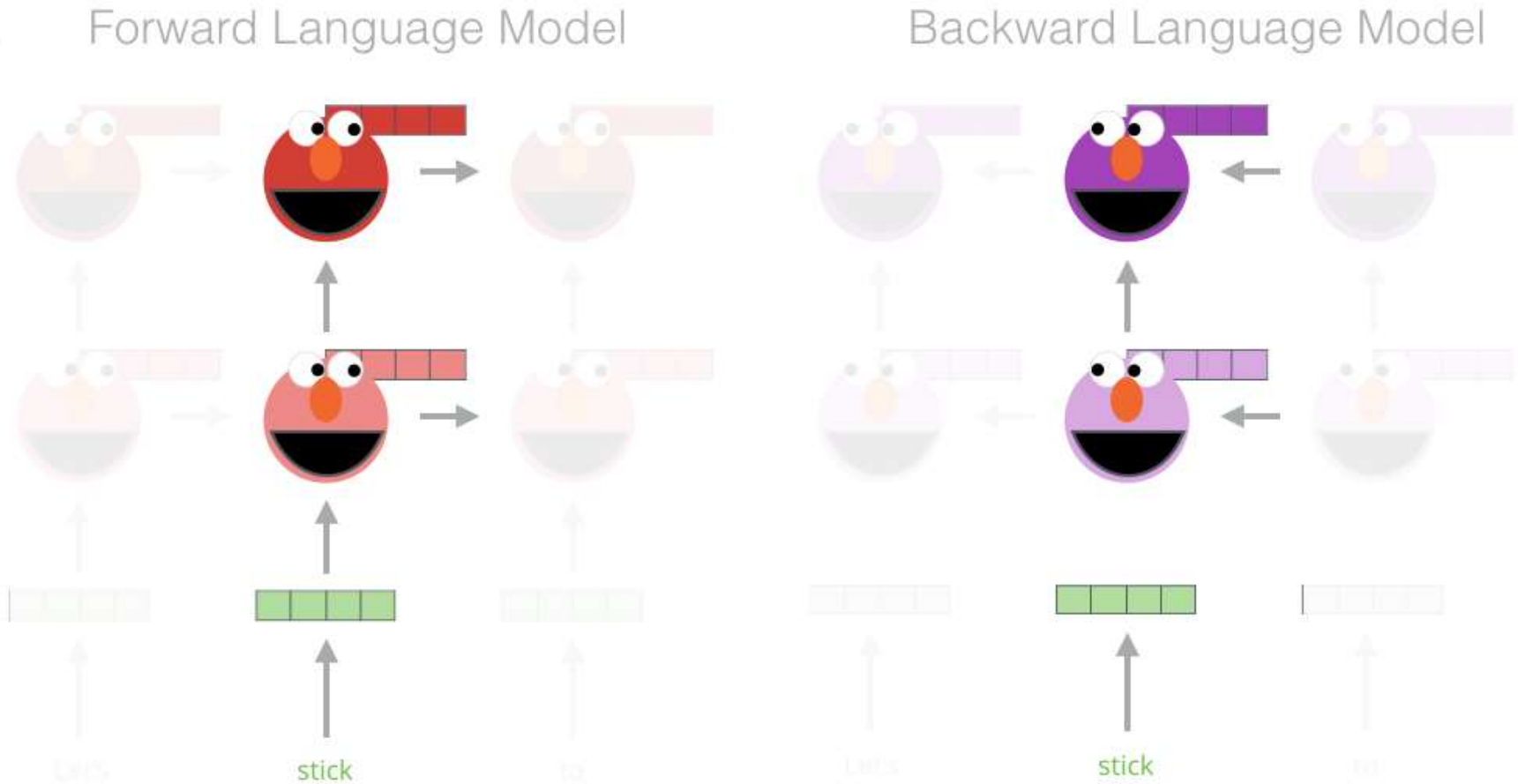


2- Multiply each vector by a weight based on the task



3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context



Concatenating ELMo vectors

- Adding ELMo vectors leads to SOTA performance

Task	Previous SOTA	ELMo Results
SQuAD (question/answering)	84.4	85.8
SNLI (textual entailment)	88.6	88.7
Semantic Role Labelling	81.7	84.6
Coref Resolution	67.2	70.4
NER	91.93	92.22
SST-5 (sentiment analysis)	53.7	54.7

ELMo – Recommended Use

- Concatenate ELMos with context-independent word embeddings instead of replacing them

```
from flair.embeddings import Sentence
sentence = Sentence('The grass is green .')
```

We also init a class of the desired embedding method. The `embed` method of this class gets a Sentence and adds to its tokens the relevant embedding.

```
from flair.embeddings import ELMoEmbeddings

# init embedding
elmo_embedding = ELMoEmbeddings()

elmo_embedding.embed(sentence)
for token in sentence:
    print(token)
    print(token.embedding.shape)
    print(token.embedding)
```

Transfer Learning

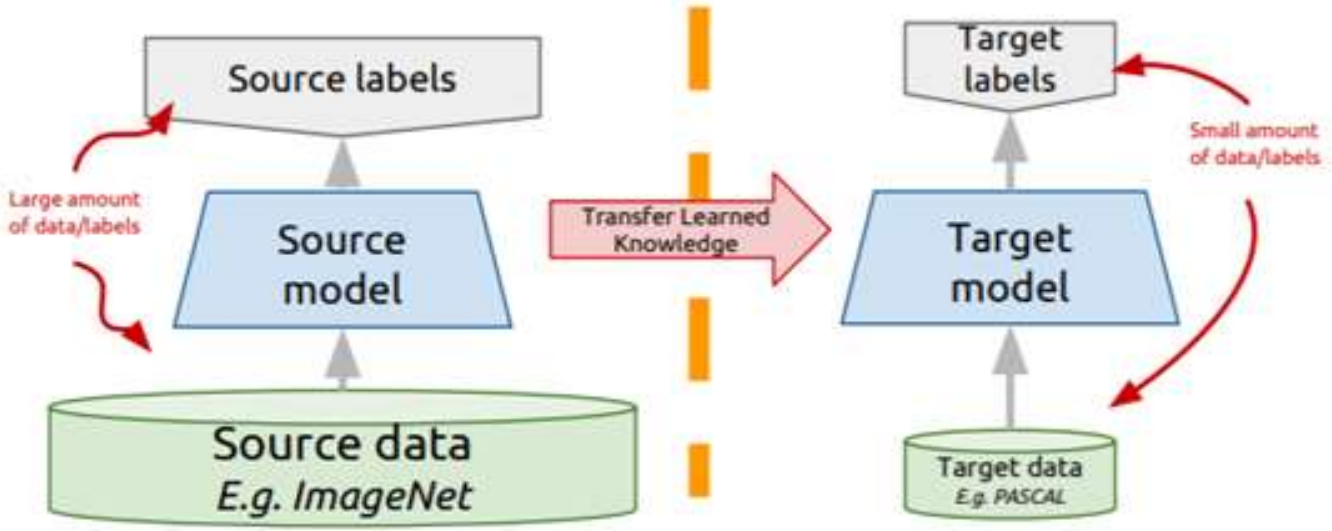
Transfer learning: idea

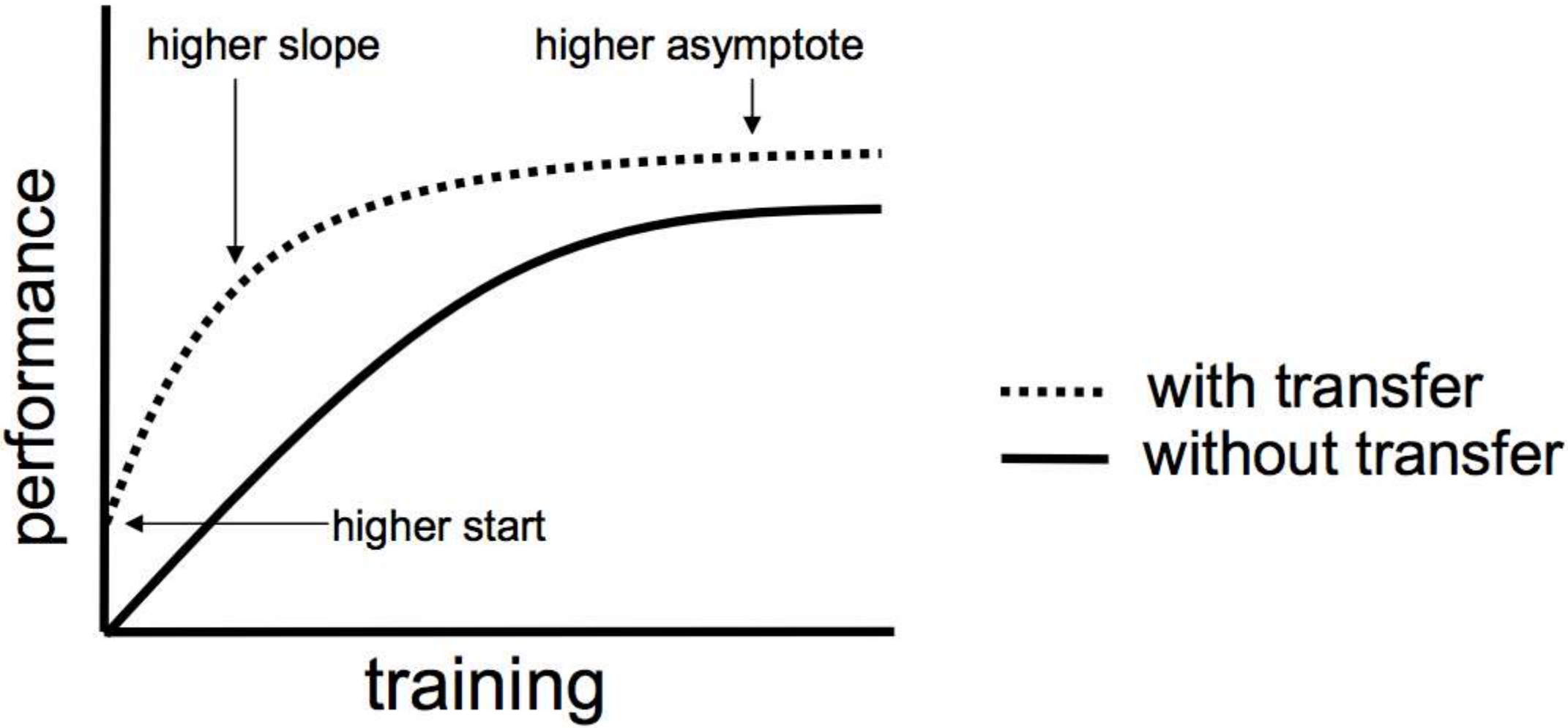
Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

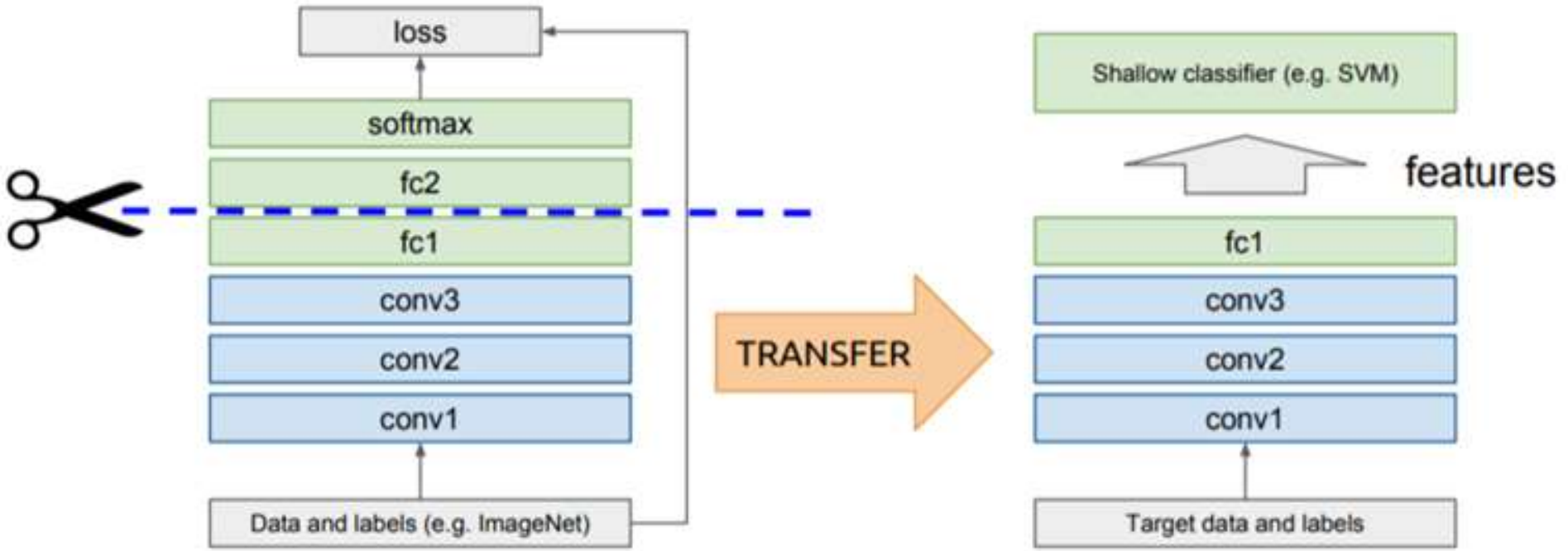
- Same domain, different task
- Different domain, same task




higher slope
higher asymptote
performance
higher start
training
with transfer
without transfer

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$



Freeze or fine-tune?

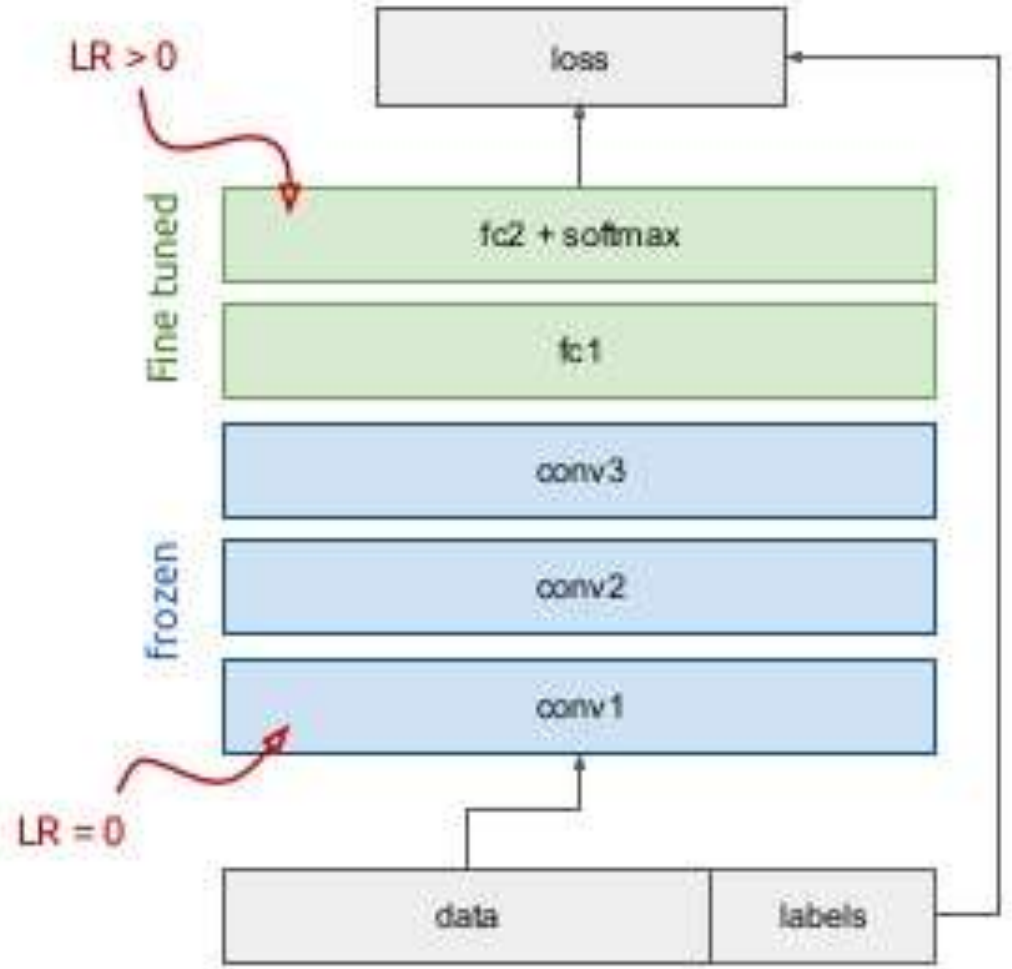
Bottom *n* layers can be frozen or fine tuned:

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

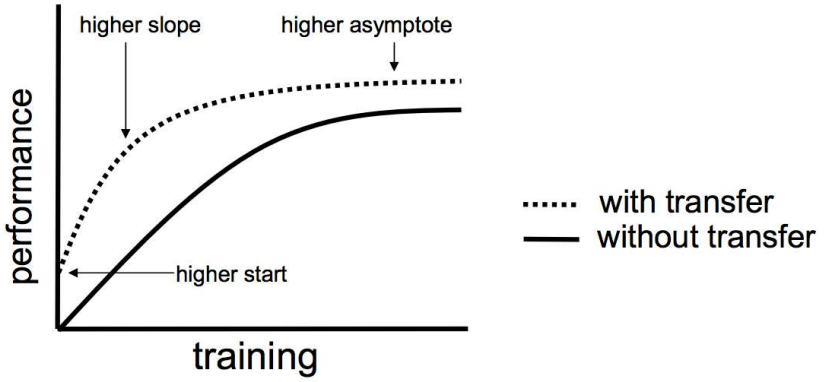
- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning

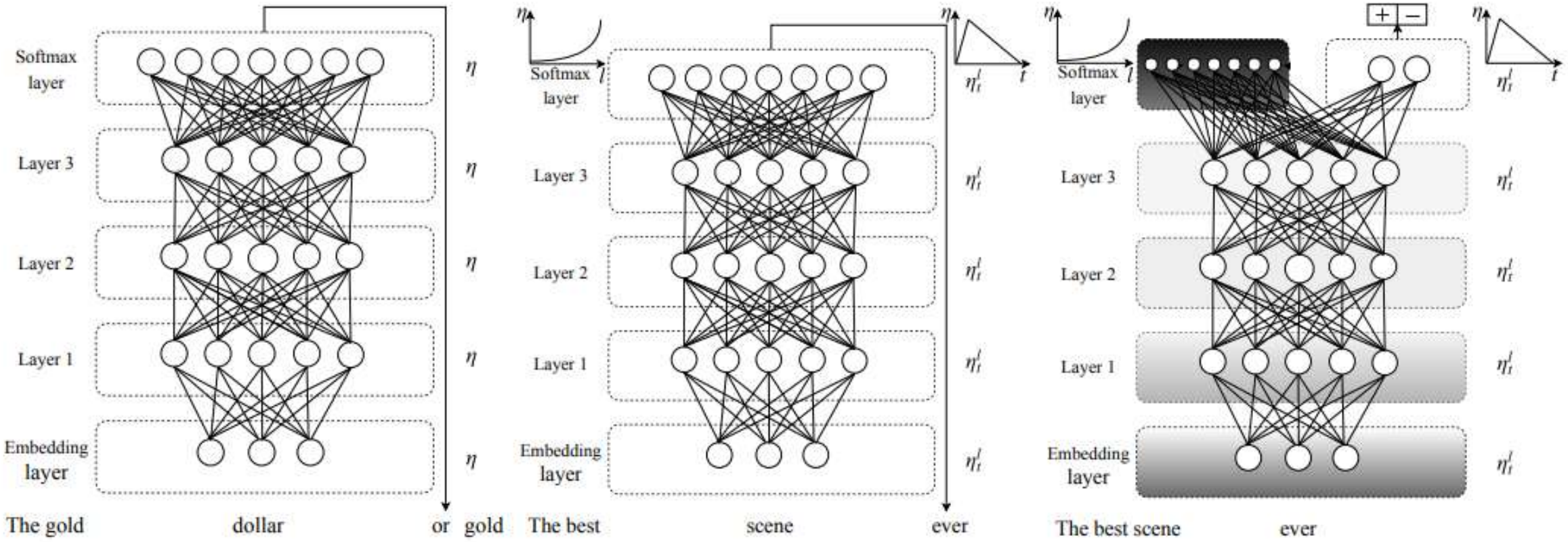


Transfer Learning using a Language Model

- Use a Language Model to gain information about the language, learning both:
  - Low-level features (word representations)
  - High-level features (semantic meaning)
- This is a form of *Self-Supervised Learning*
- Fine-tune the model to the specific task
  - Faster
  - Achieves better performance with little data
  - Achieves better performance even with lots of data



Transfer Learning using ULMFiT



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

ULMFiT: LM Fine-tuning

- LM fine-tuning can use only labeled data, or all task data

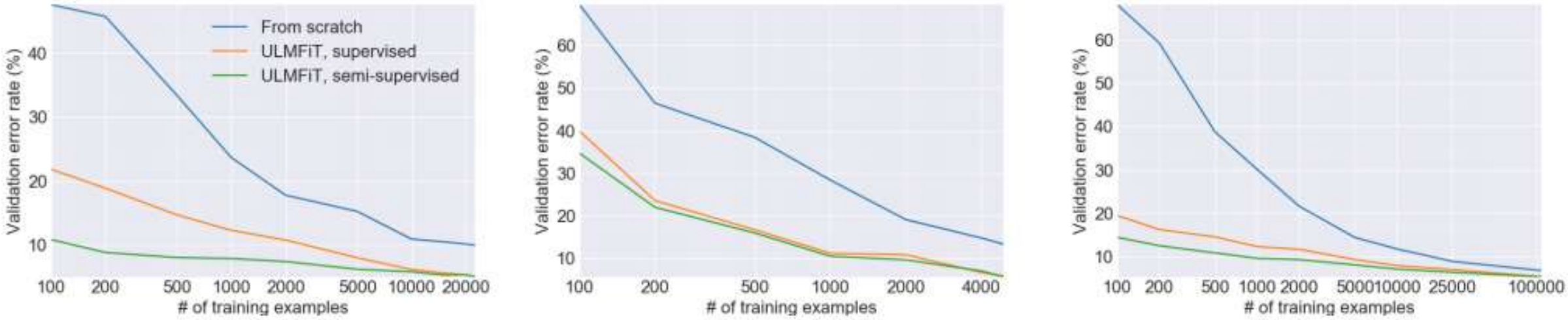


Figure 3: Validation error rates for supervised and semi-supervised ULMFiT vs. training from scratch with different numbers of training examples on IMDb, TREC-6, and AG (from left to right).

BERT - **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

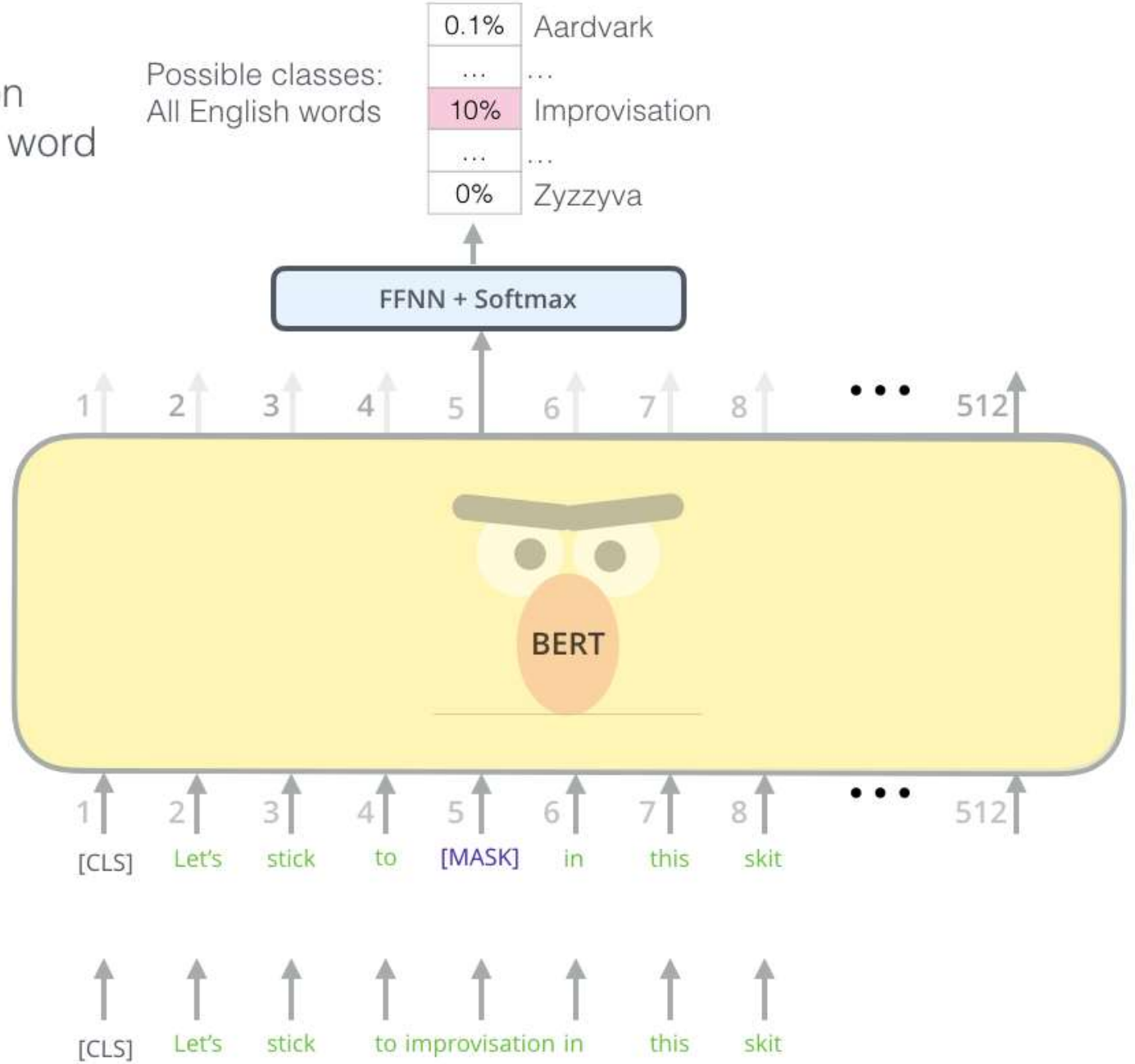
- Trained on masked words + “is next sentence” prediction
- To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Input: The man went to the $[MASK]_1$. He bought a $[MASK]_2$ of milk .
Labels: $[MASK]_1$ = store; $[MASK]_2$ = gallon

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

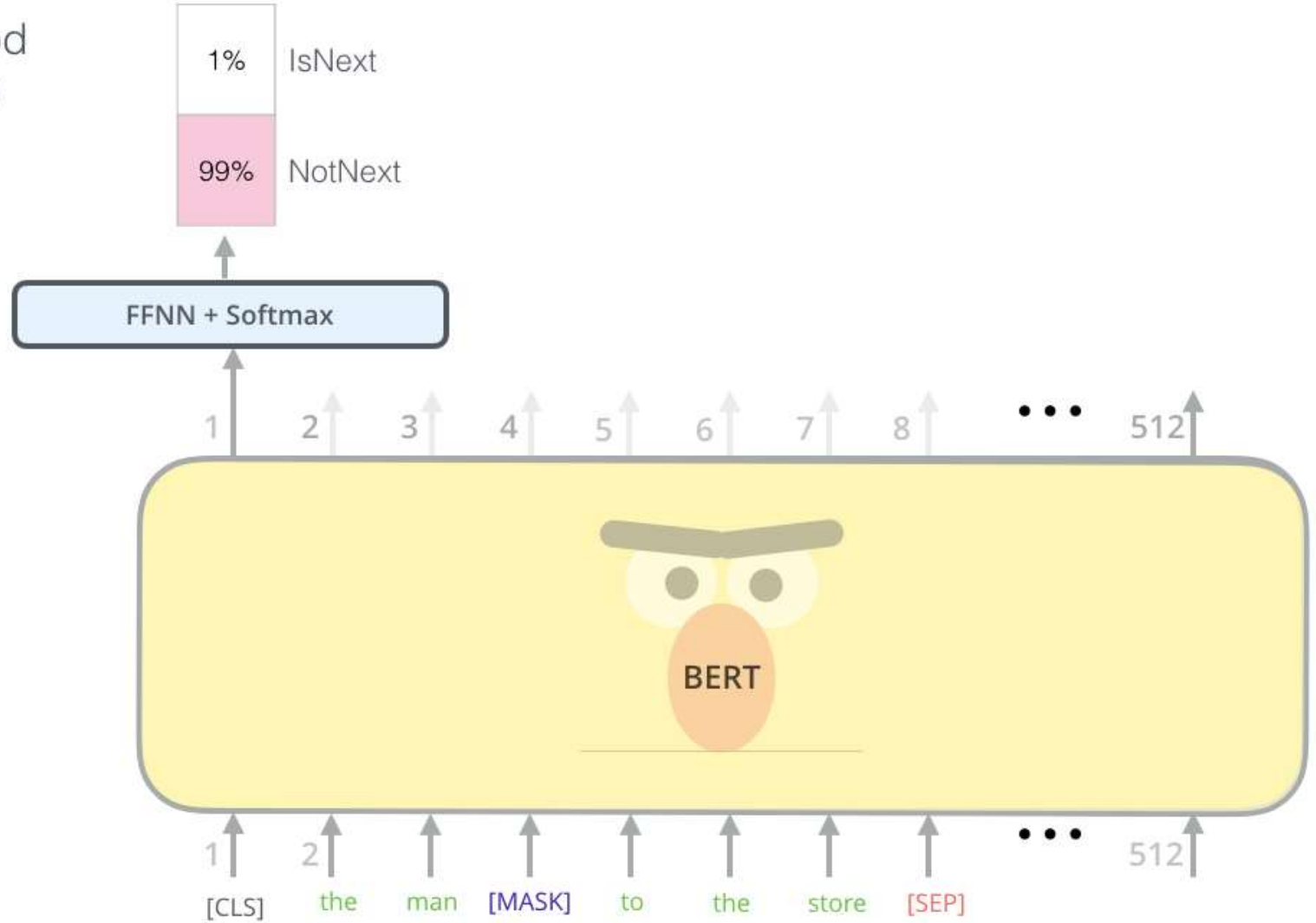
Use the output of the masked word's position to predict the masked word



Randomly mask 15% of tokens

Input

Predict likelihood that sentence B belongs after sentence A



Tokenized Input

Input

[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Sentence A

Sentence B

Model Architecture

- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning

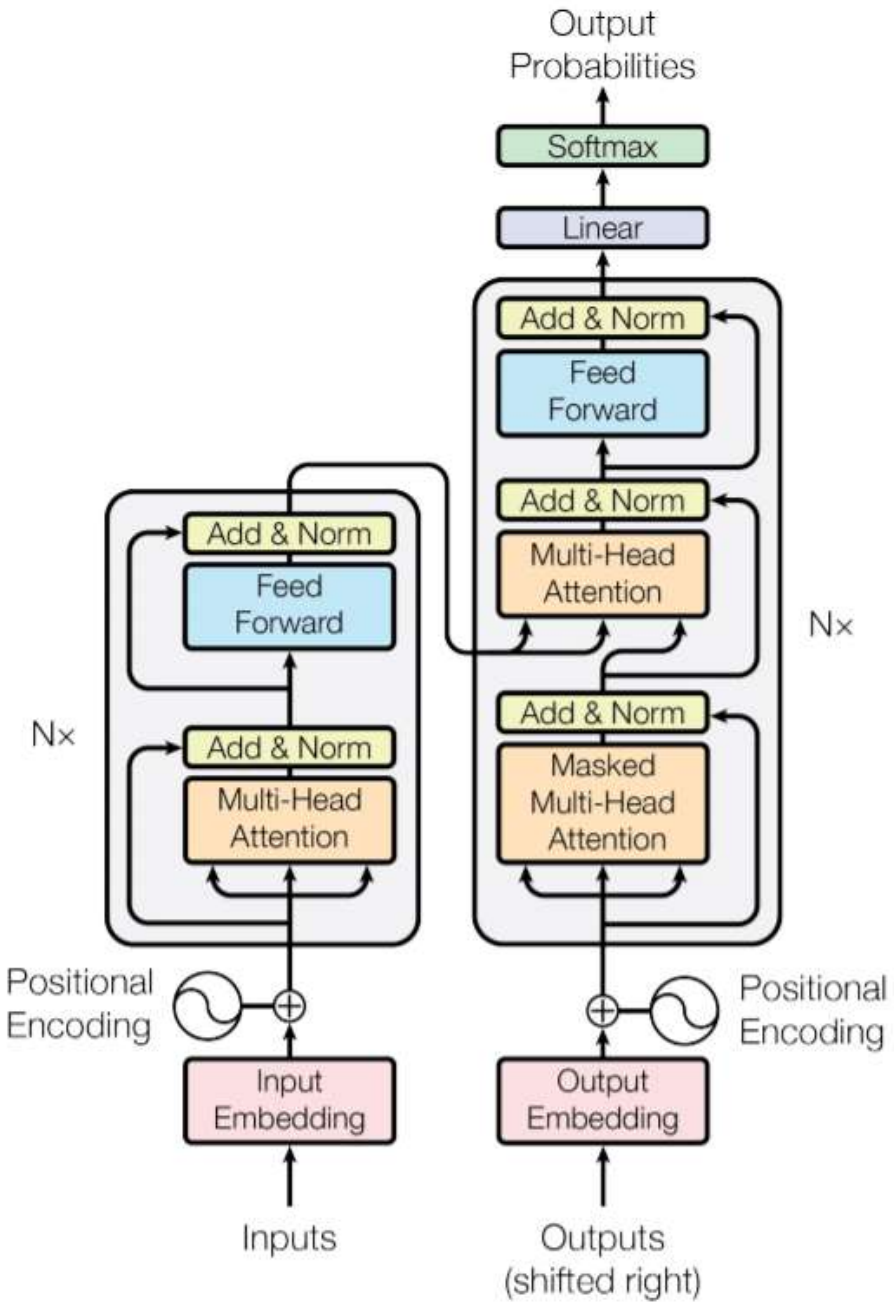
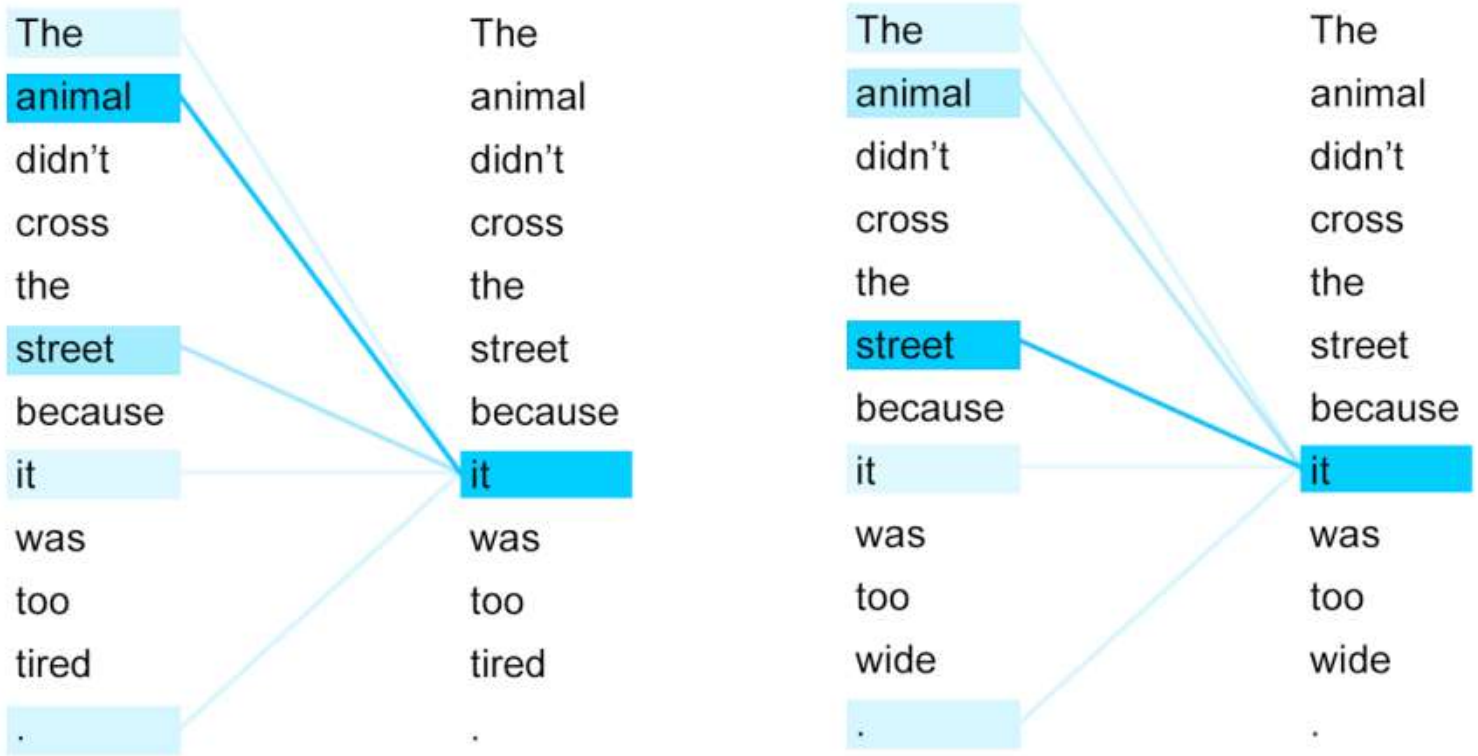
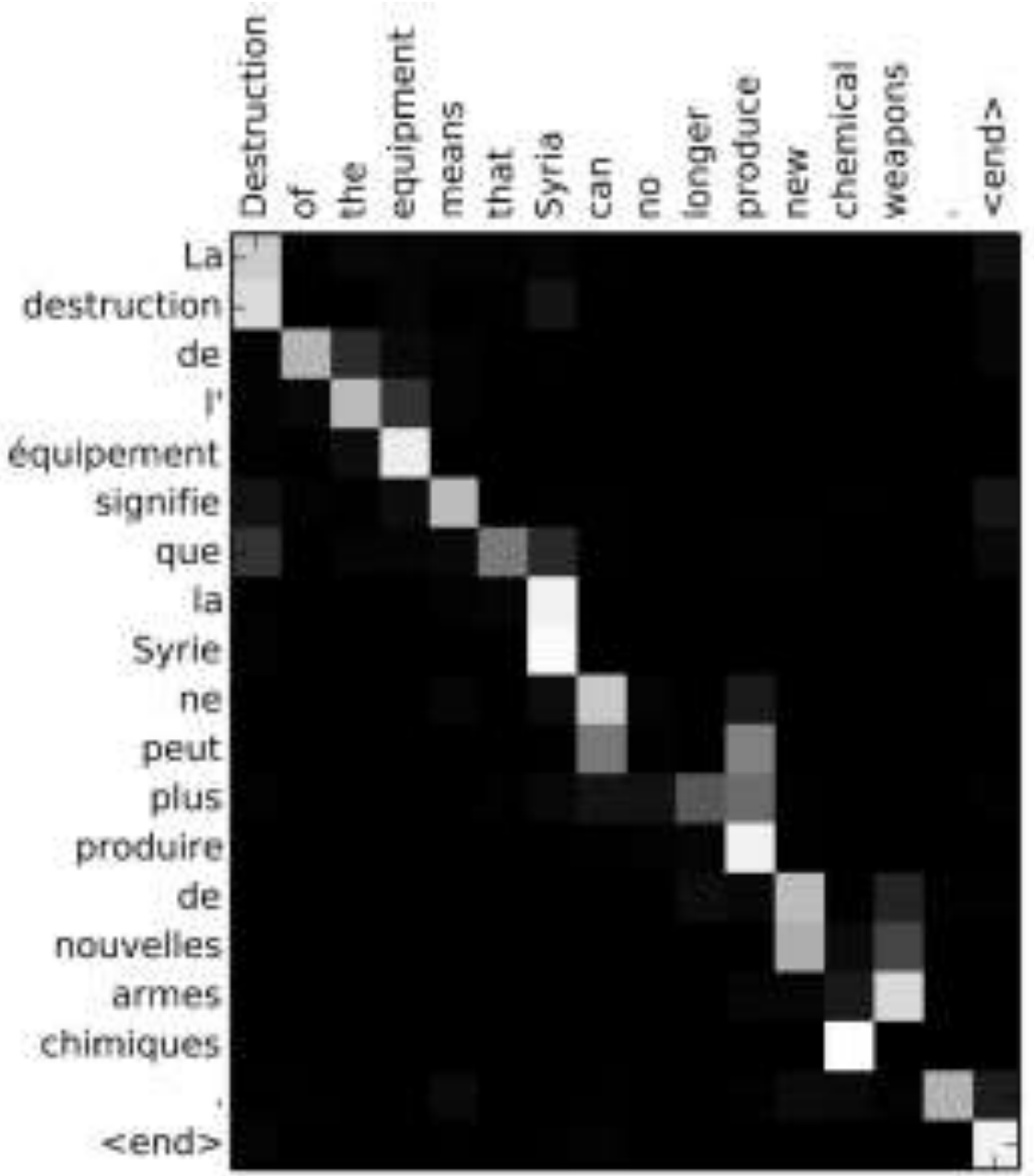


Figure 1: The Transformer - model architecture.

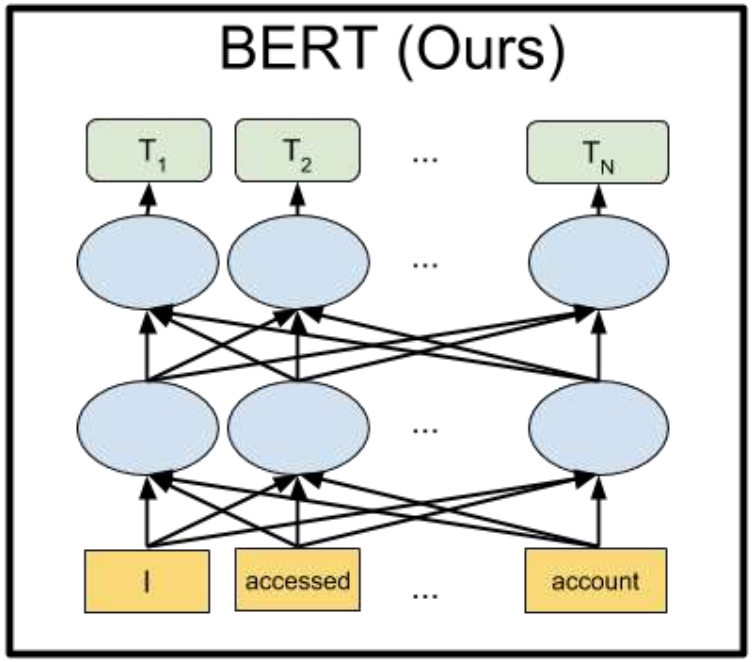
“Attention is all you need”

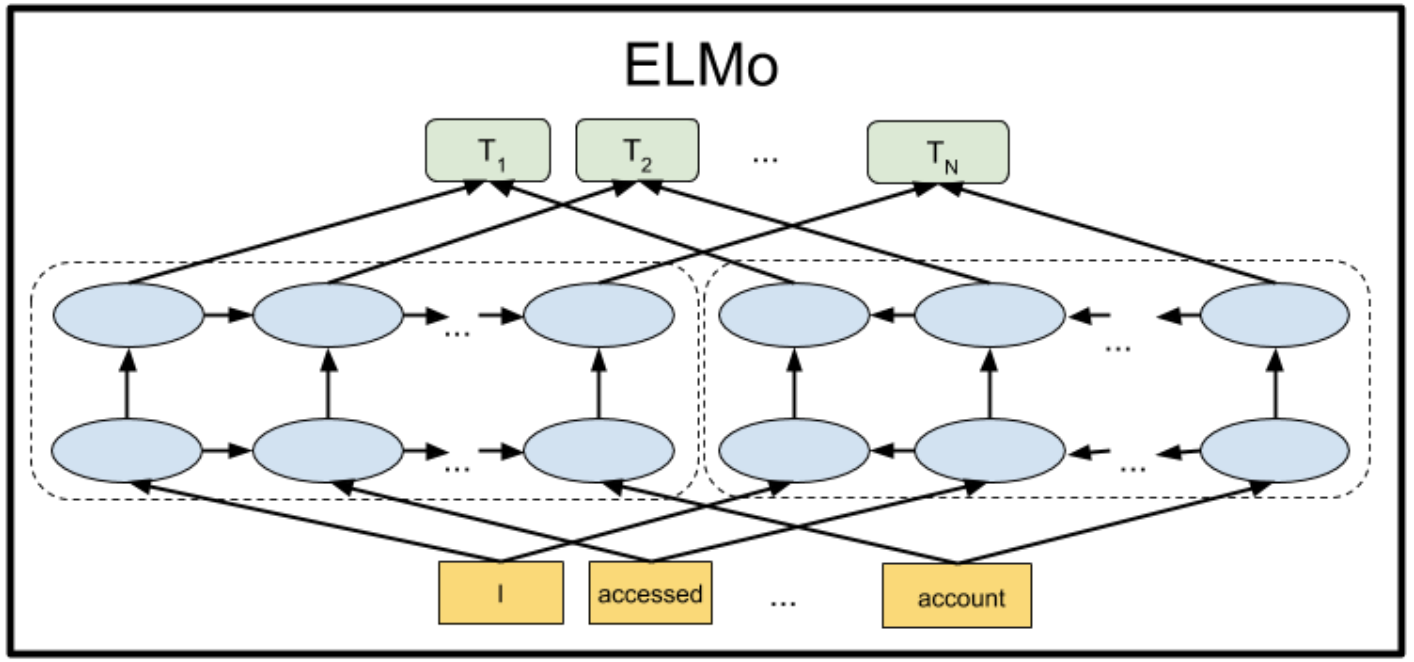
- Allow attending to multiple words in the sentence
- Similar to the way humans read text




Destruction
of
the
equipment
means
that
Syria
can
no
longer
produce
new
chemical
weapons
.
<end>
La
destruction
de
l'
équipement
signifie
que
la
Syrie
ne
peut
plus
produire
de
nouvelles
armes
chimiques
.
<end>

BERT vs. ELMo


BERT (Ours)
T1
T2
...
TN
I
accessed
account


ELMo
T1
T2
...
TN
I
accessed
account

Model Architecture

- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning

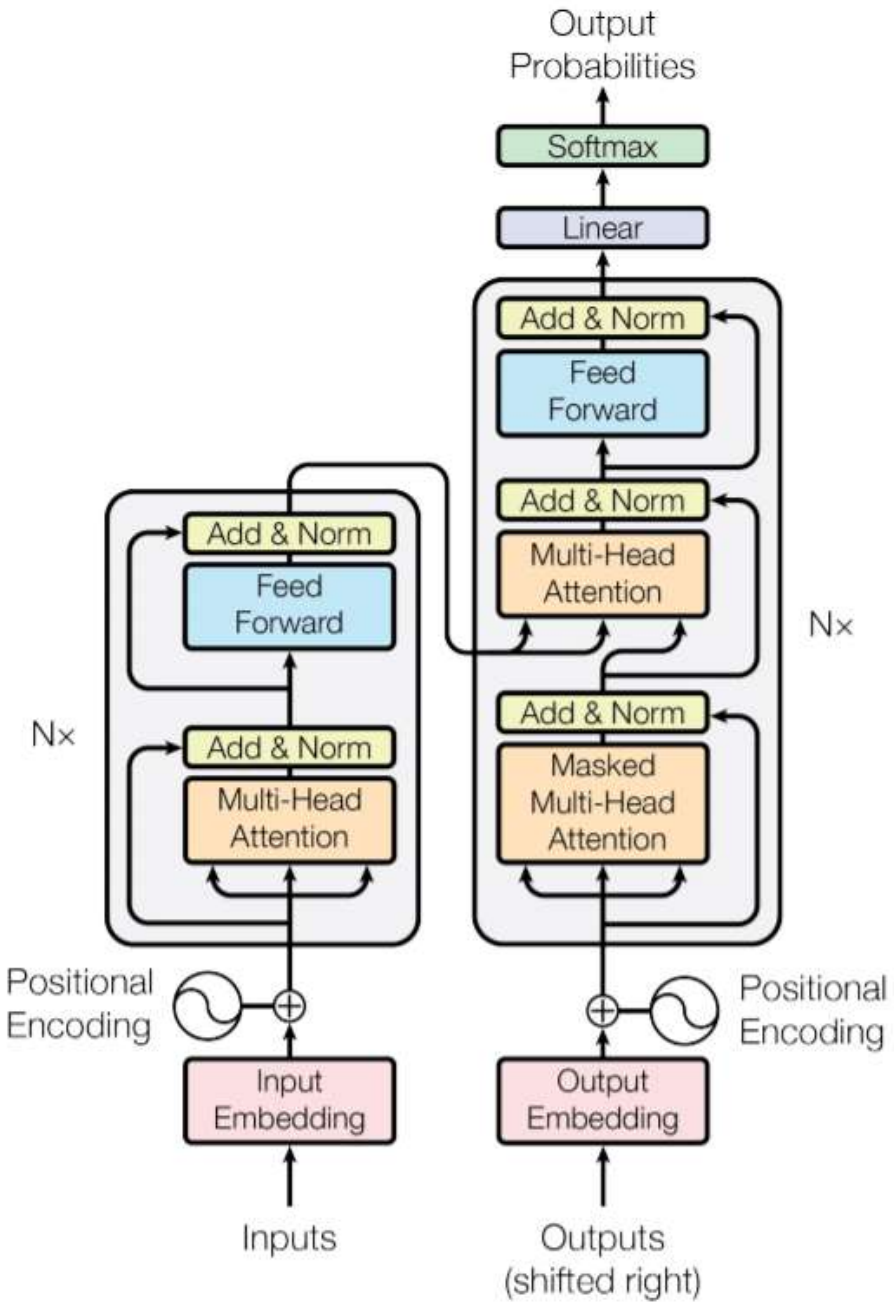
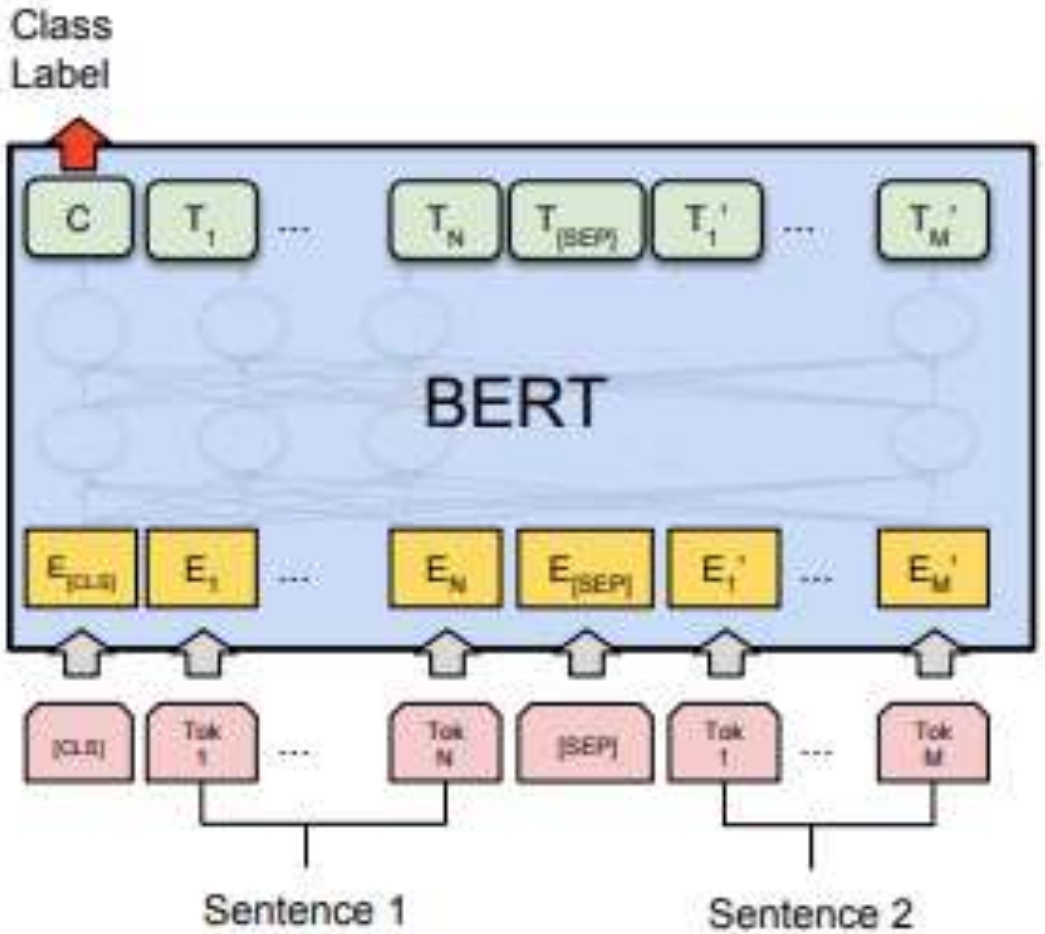


Figure 1: The Transformer - model architecture.

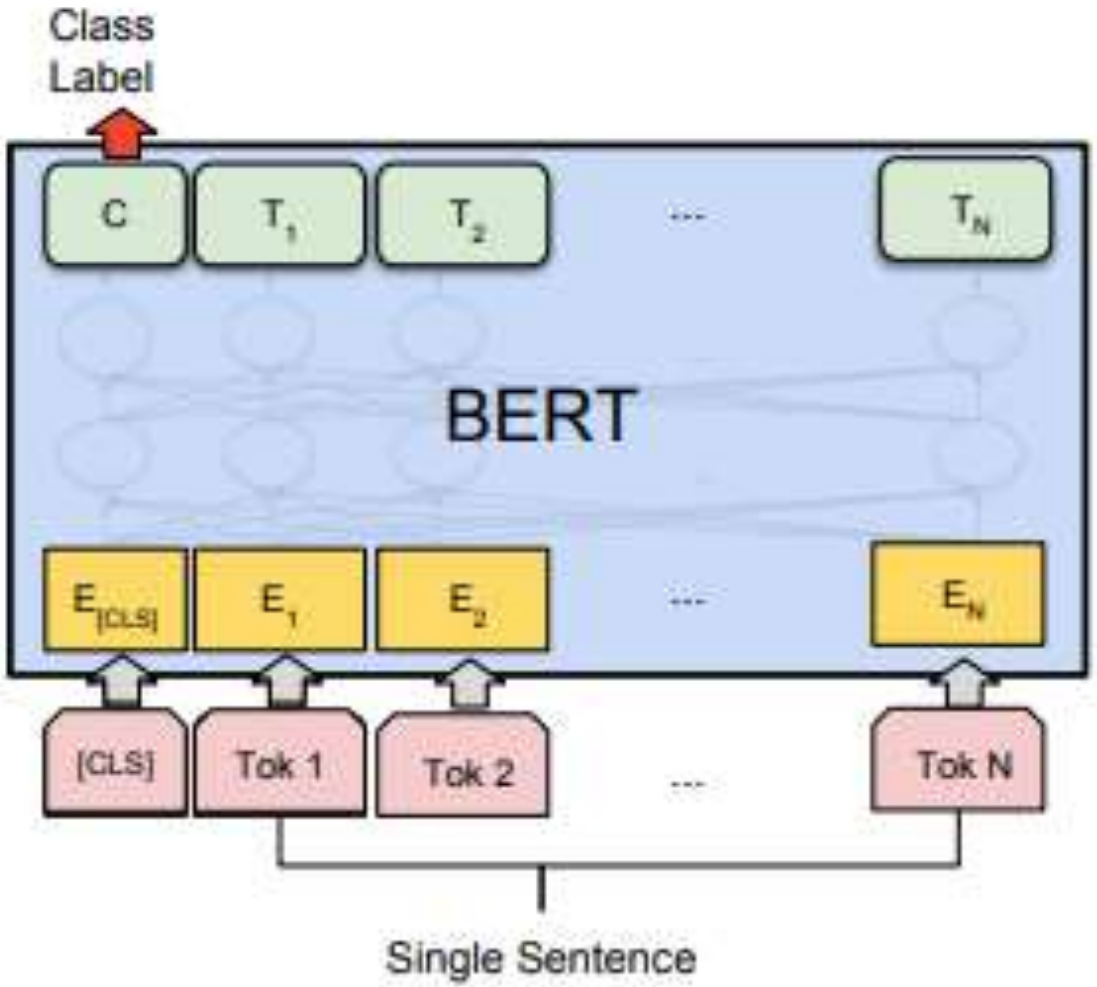
BERT GLUE Performance

- BERT smashed SOTA by a large margin
- Since then, it lost the lead to similar architectures, with more parameters, trained on larger corpuses

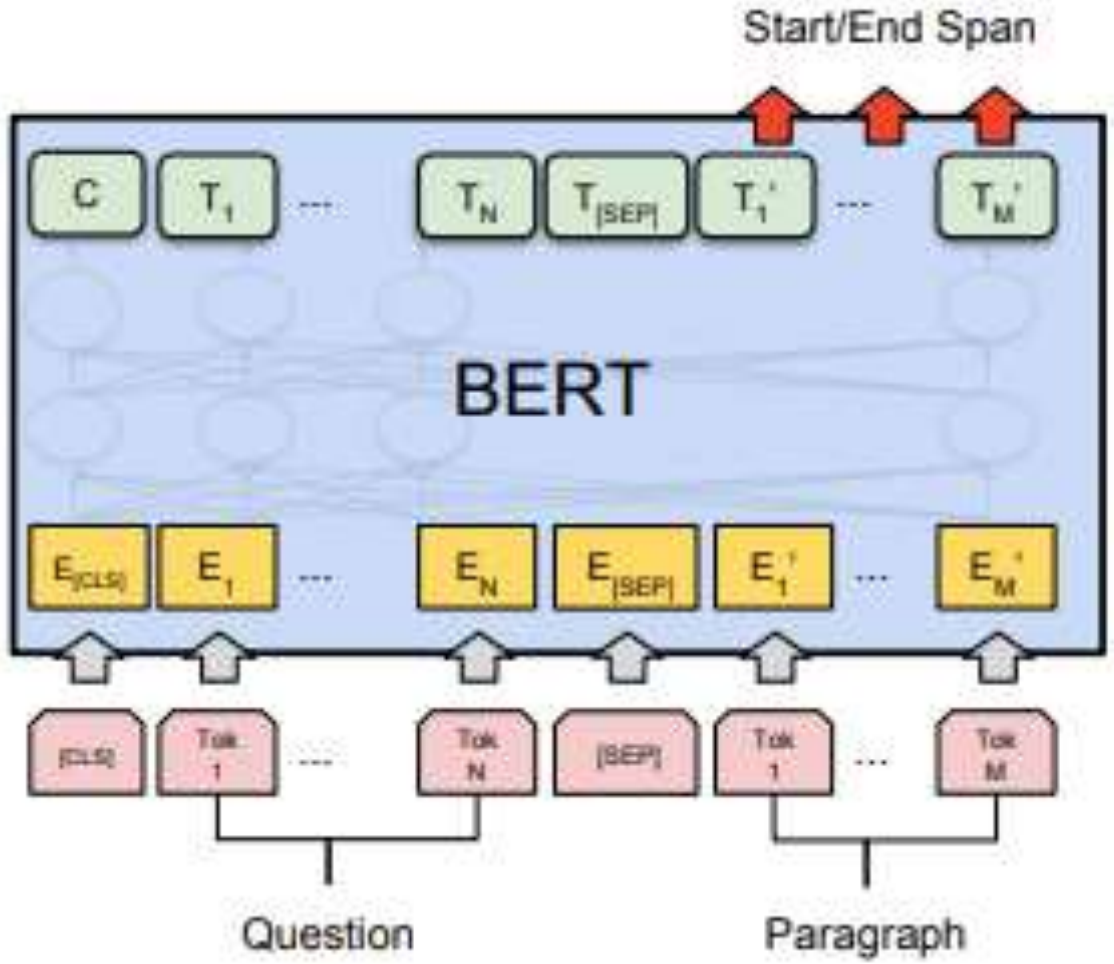
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**



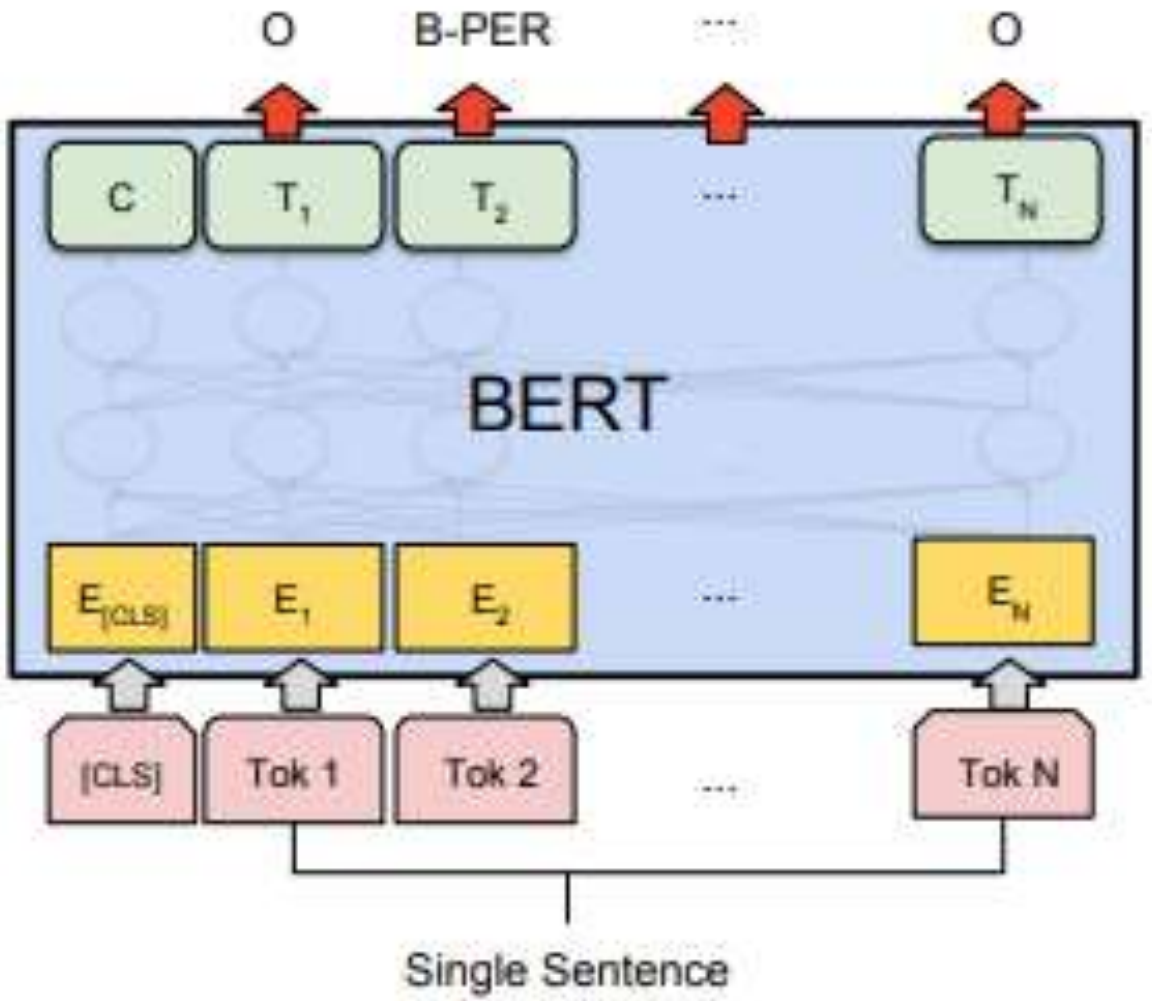
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



(b) Single Sentence Classification Tasks:
SST-2, CoLA



(c) Question Answering Tasks:
SQuAD v1.1



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

Fine-tuning BERT

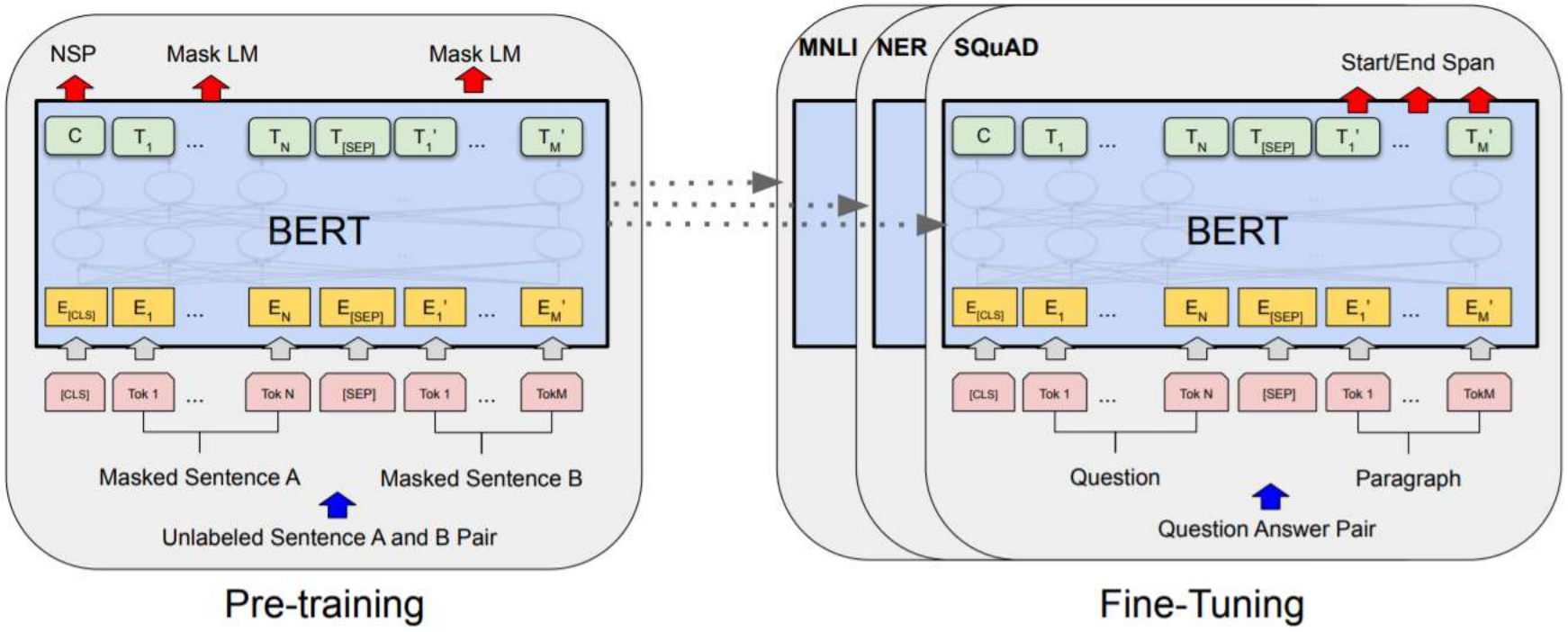
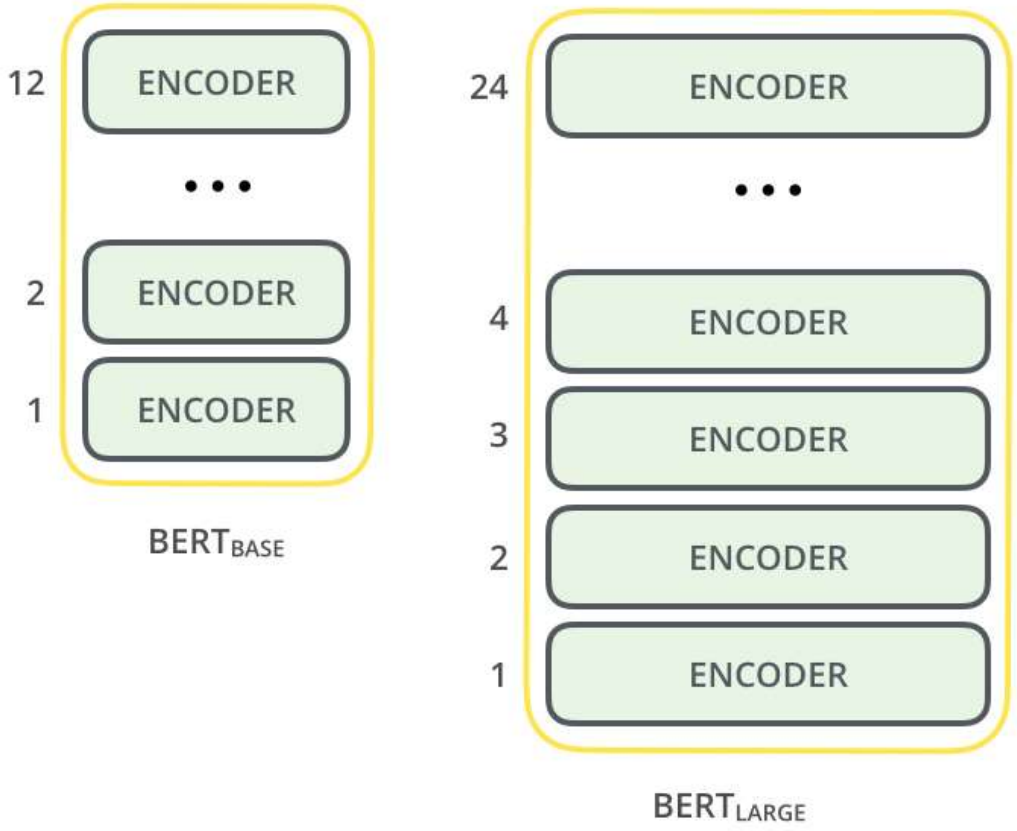


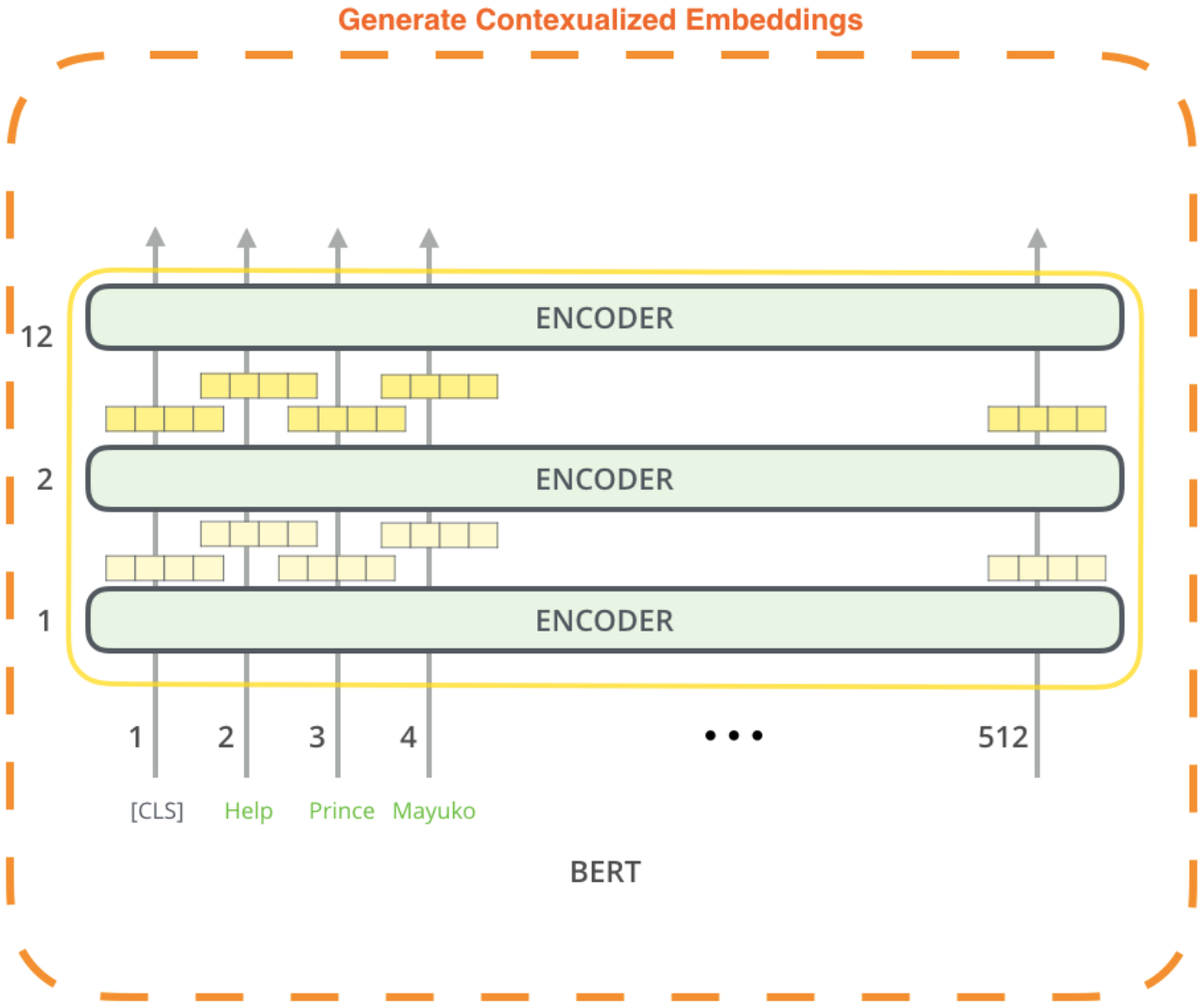
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

Main BERT architectures

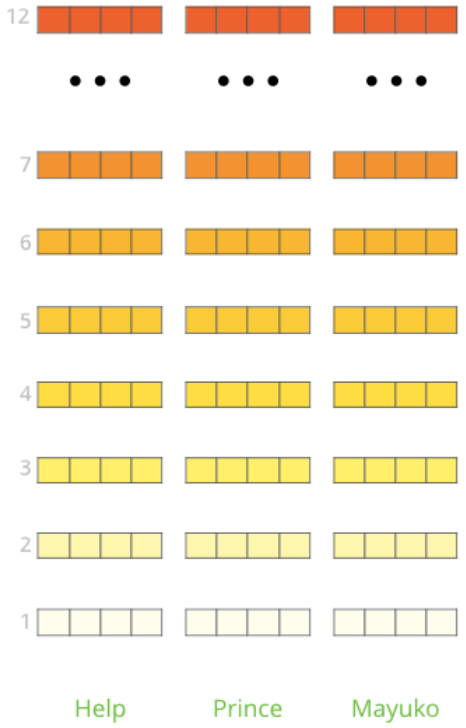


- `BERT-Large, Uncased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Large, Cased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Uncased` : 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Large, Uncased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Cased` : 12-layer, 768-hidden, 12-heads , 110M parameters
- `BERT-Large, Cased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Multilingual Cased (New, recommended)` : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Multilingual Uncased (Orig, not recommended)` (Not recommended, use `Multilingual Cased` instead): 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Chinese` : Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

Which embedding should we use?



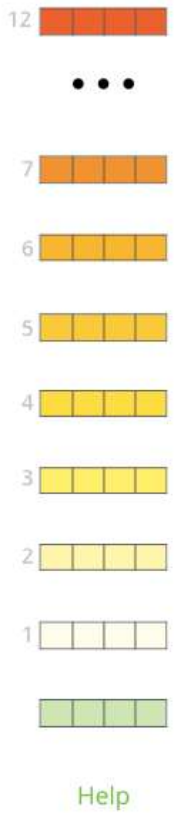
The output of each encoder layer along each token's path can be used as a feature representing that token.

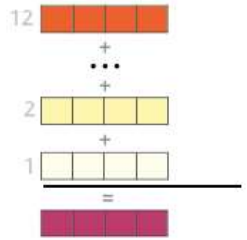
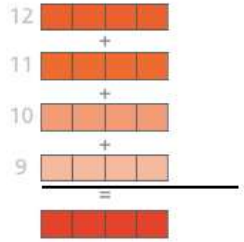
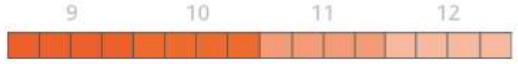


But which one should we use?

Which embedding should we use?

What is the best contextualized embedding for "Help" in that context?
For named-entity recognition task CoNLL-2003 NER



		Dev F1 Score
First Layer		91.0
Last Hidden Layer		94.9
Sum All 12 Layers		95.5
Second-to-Last Hidden Layer		95.6
Sum Last Four Hidden		95.9
Concat Last Four Hidden		96.1

BERT Practical Considerations

- BERT handles OOV tokens rather well, with sub-words tokenization
- BERT is a huge model and is very slow
- GPU is required for fine-tuning
- Inference is also very slow (~10 sentences/second)

Play Time!

Prizes courtesy of Gong.io

Prizes courtesy of Gong.io

Our Dataset

- Posts from 4 different newsgroups
  - rec.sport.baseball
  - rec.sport.hockey
  - talk.politics.guns
  - talk.politics.Mideast
- Cut to length of 200-400 characters
- Metadata was removed
- This is a subset of a larger dataset of 20 newsgroups

Can you be #1? - Ideas

1. Use ELMo/BERT to derive word/sentence embeddings
2. Use larger SOTA models – BERT, XLNet, RoBERTa…
3. Stack word embeddings from multiple models
4. *<INSERT YOUR WINNING SOLUTION HERE>*

Advanced Ideas

1. Use character-based methods for word/sentence embeddings
2. Ensemble the predictions of different models
3. Train your own language model
4. Use semi-supervised methods
5. *<INSERT YOUR WINNING SOLUTION HERE>*

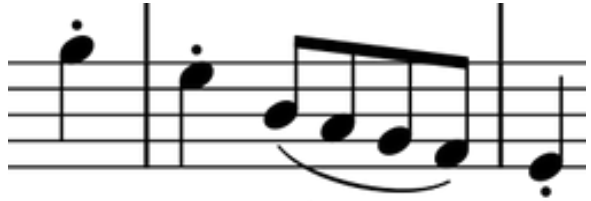
Please note

1. Code in Jupyter Notebooks should give you a head-start
2. Recent models like BERT are computationally-heavy and require a GPU –
    1. Google Colab provides free GPU – enable it in your notebook
3. We haven't performed any hyper-parameters optimization
4. The notebooks contain additional content, aimed to help you exploring things on your own at home

The Slides Graveyard

Sequences and Recurrent Neural Networks (RNNs)

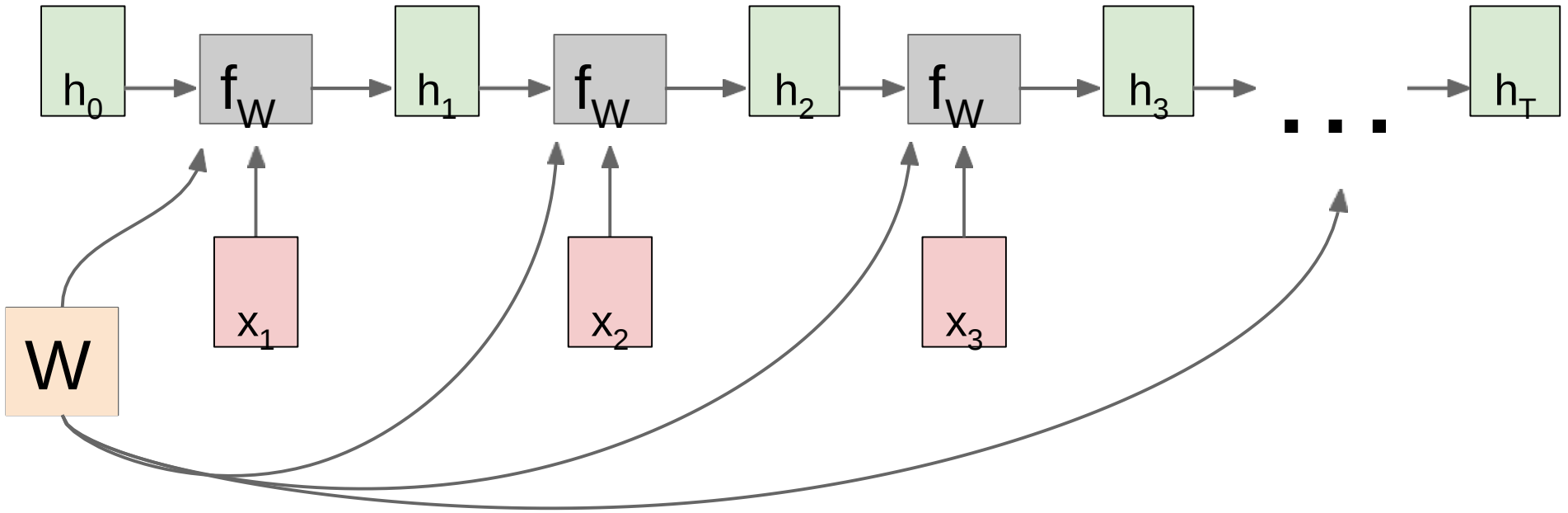
Examples of sequence data

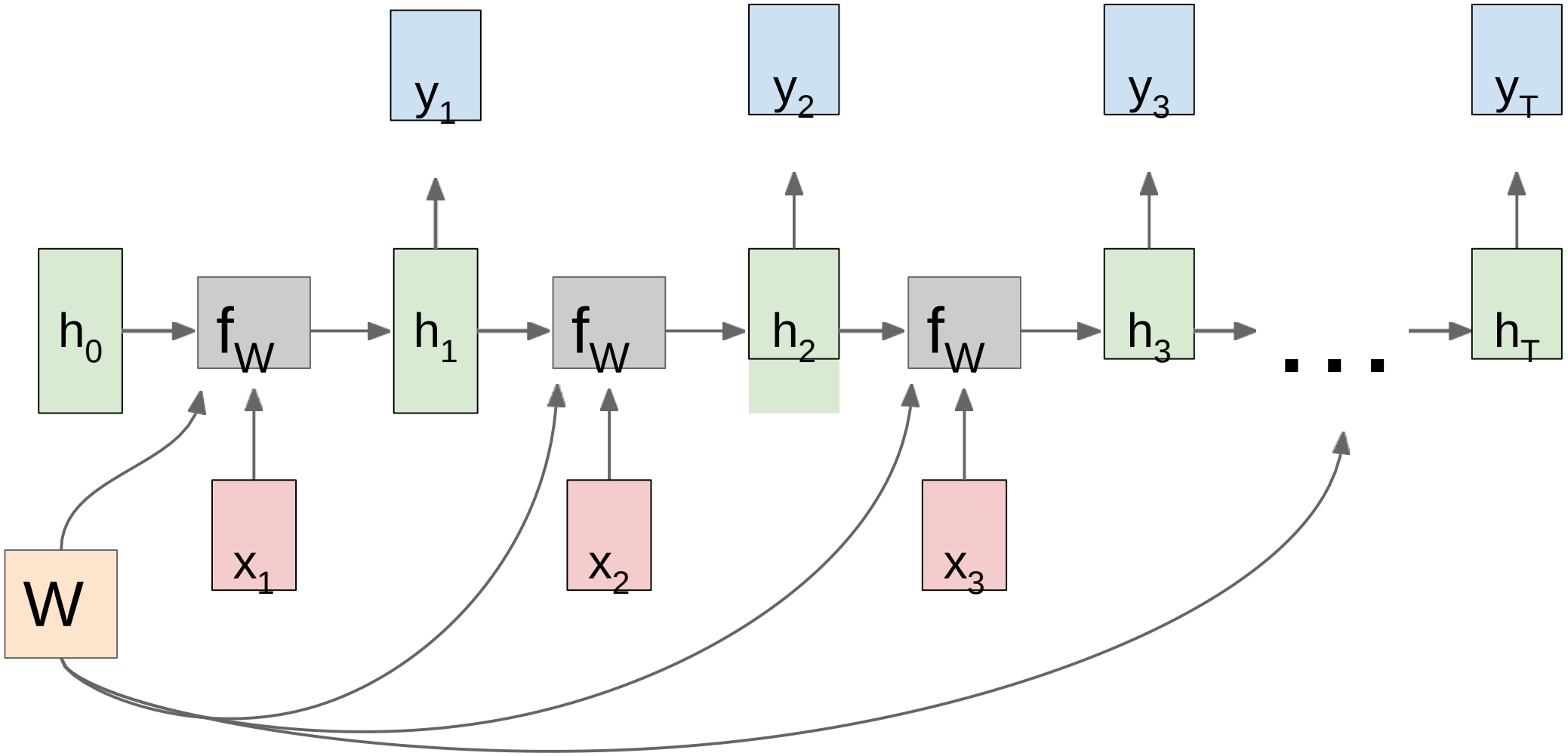
Speech recognition		→	“The quick brown fox jumped over the lazy dog.”
Music generation	∅	→	
Sentiment classification	“There is nothing to like in this movie.”	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

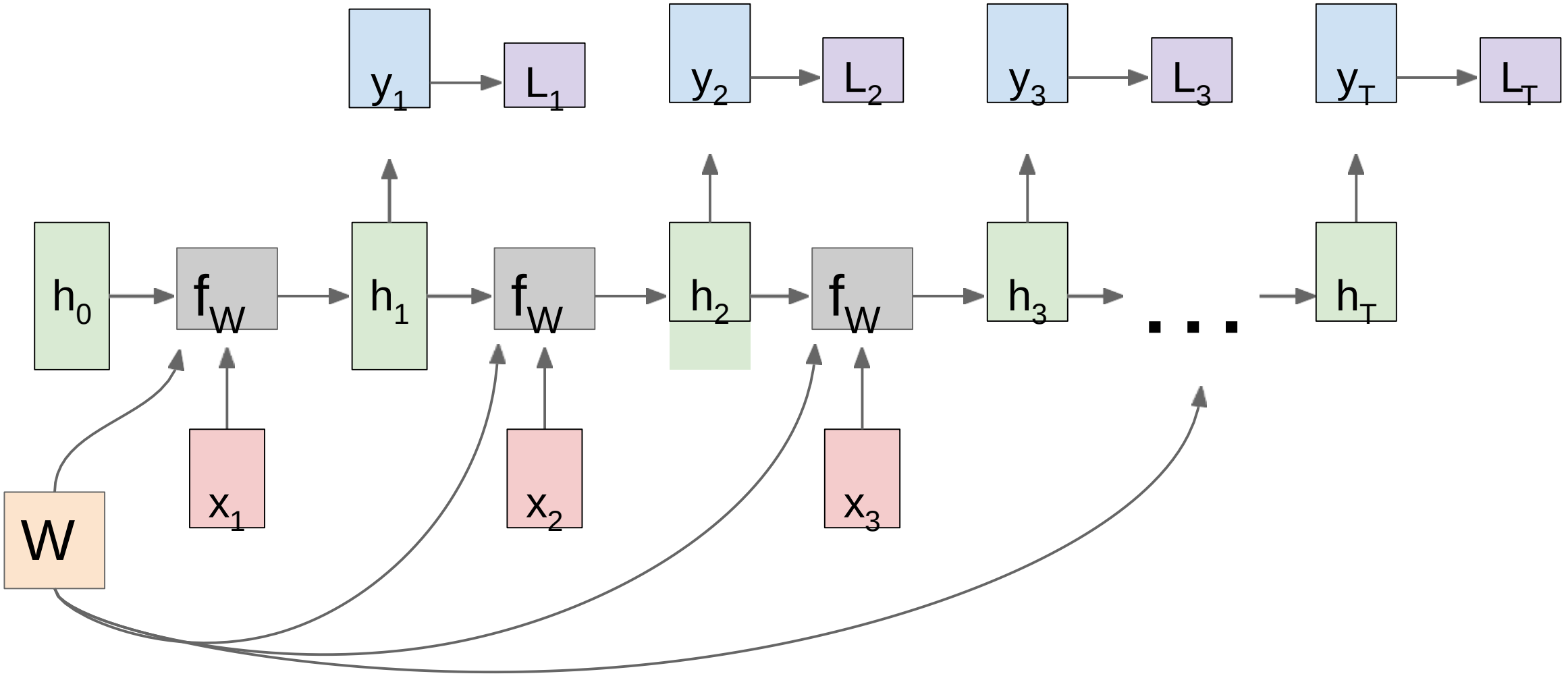
Sequence Modeling using a Neural Network

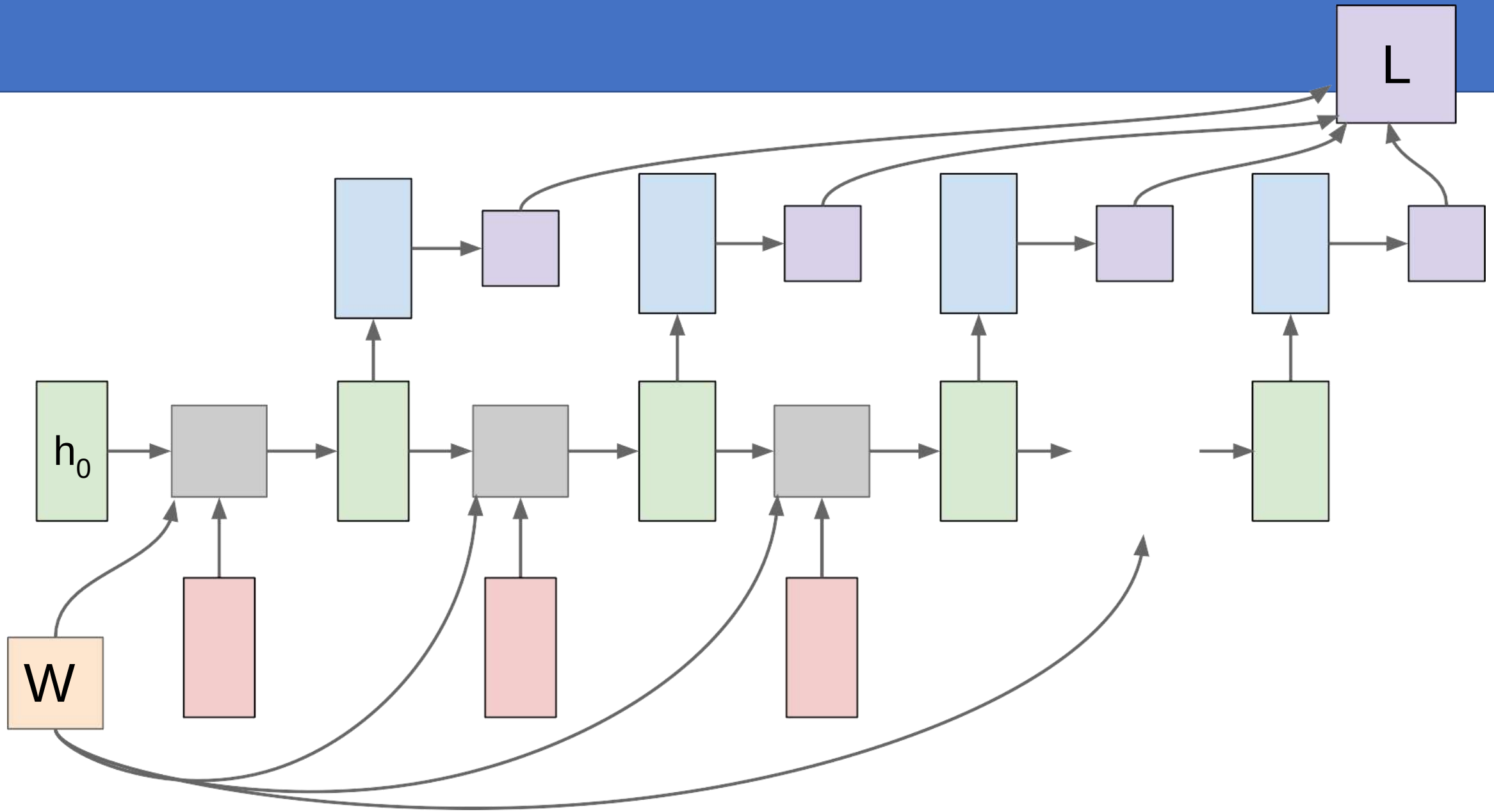
Main differences:

- The length of sequences changes
- We want to share the weights

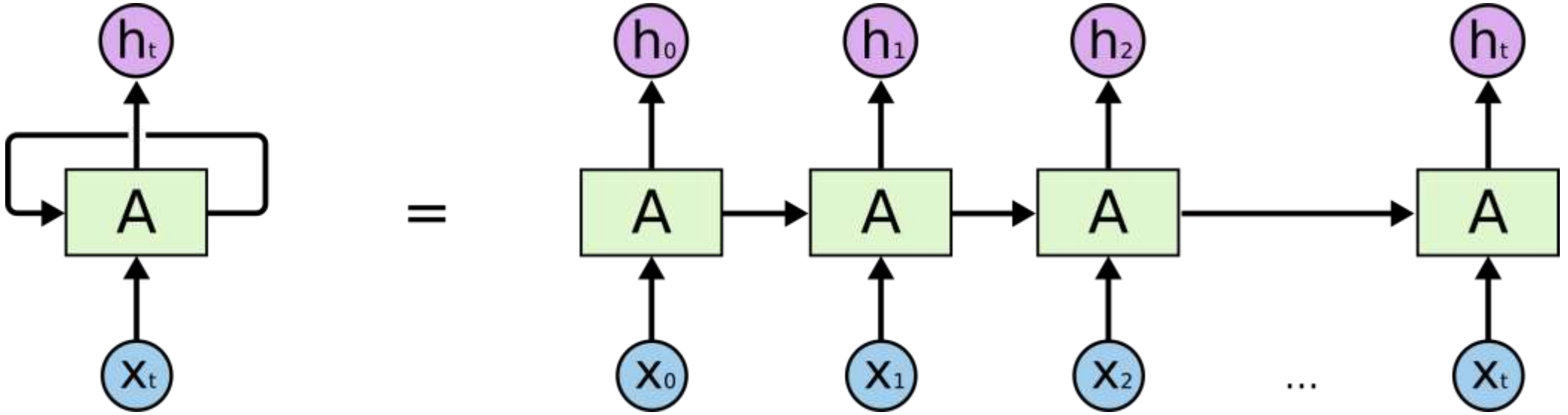



y1
y2
y3
yT
h0
fW
h1
fW
h2
fW
h3
. . .
hT
x1
x2
x3
W


y1
L1
y2
L2
y3
L3
yT
LT
h0
fW
h1
fW
h2
fW
h3
. . .
hT
x1
x2
x3
W


L
h_0
W

Unrolling an RNN



RNN – Recurrent Neural Network

- We can process a sequence of vectors x by applying a recurrence formula at every time step:

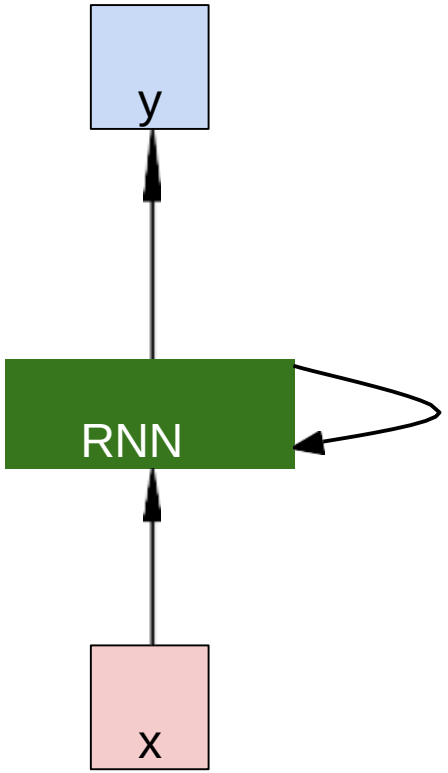
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

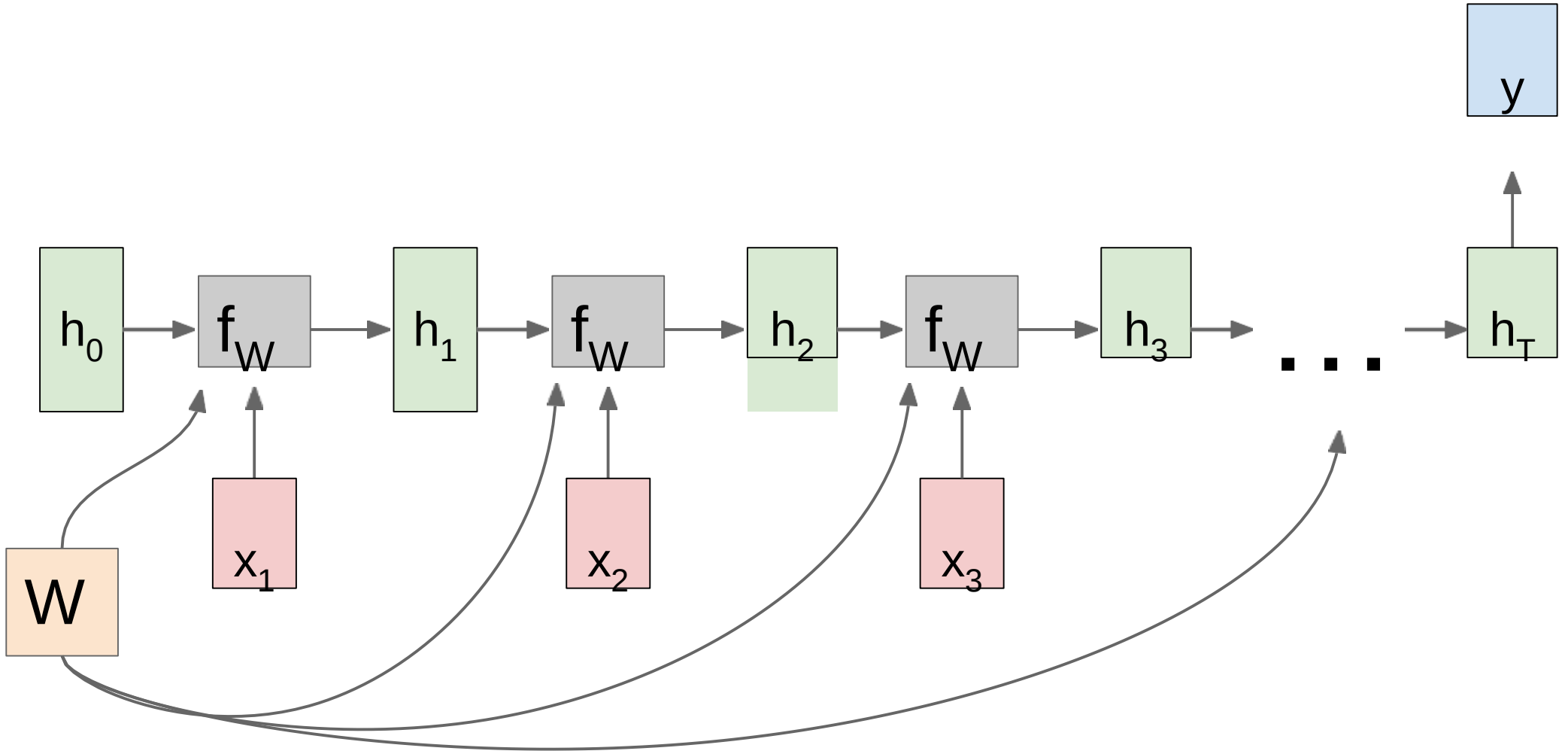
old state

input vector at some time step

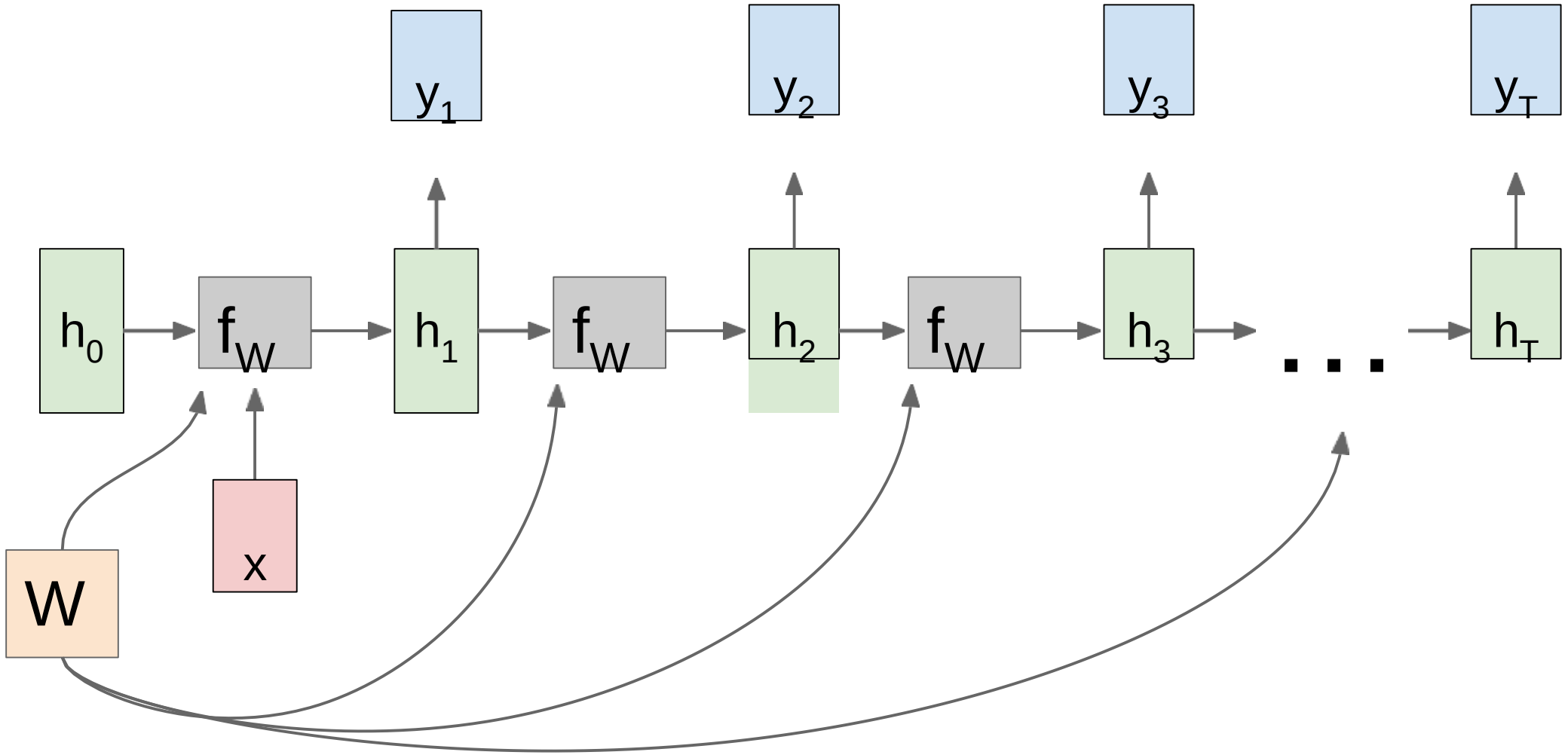


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$y_t = W_{hy}h_t$$

Many to one



One to many

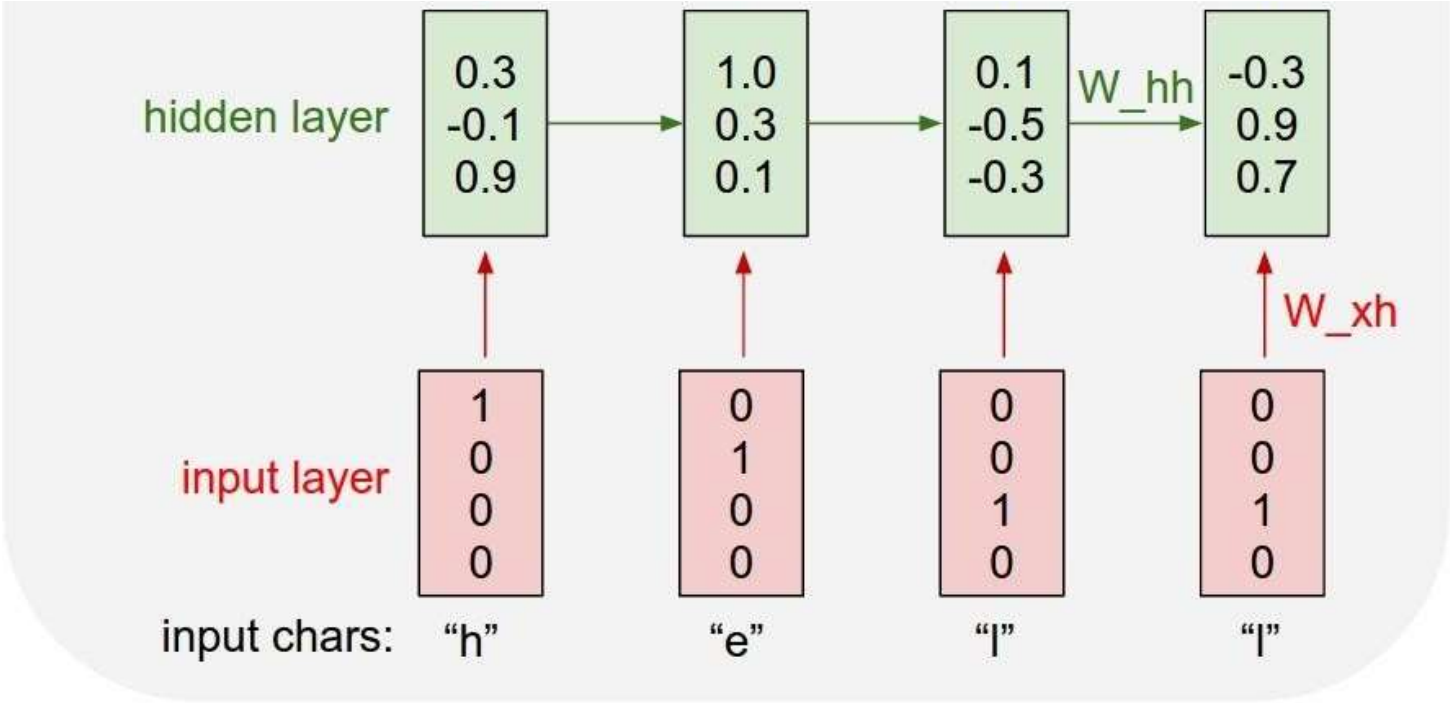


Example: Character-level Language Model

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

Vocabulary:
[h,e,l,o]

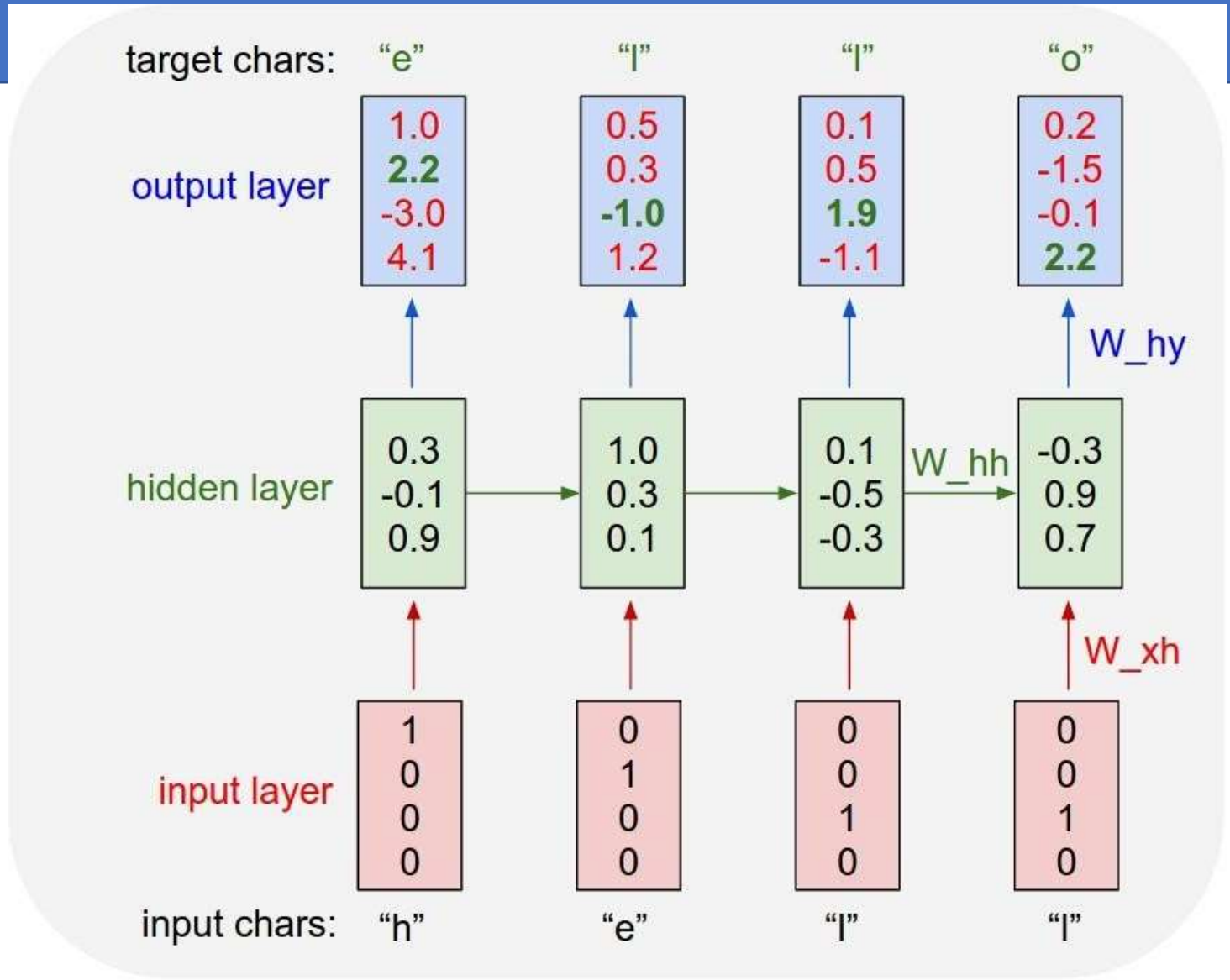
Example training sequence: **"hello"**



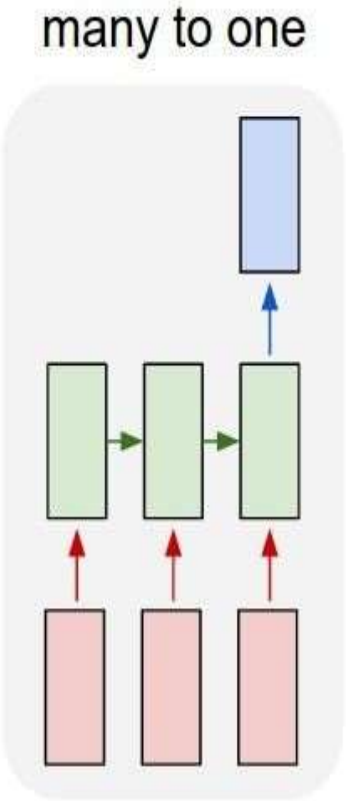
Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence: **"hello"**

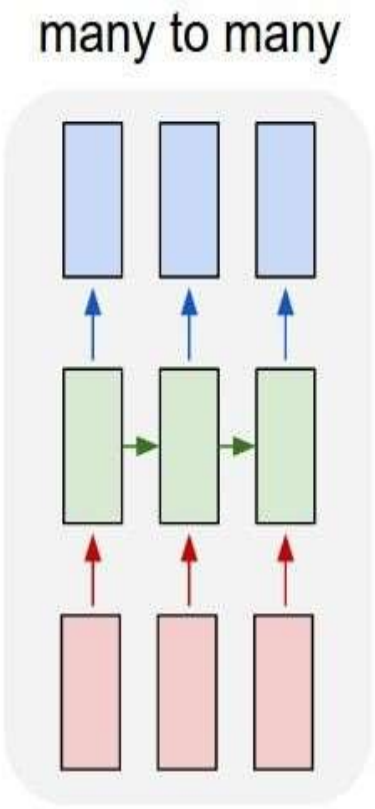


Types of Sequence Labels



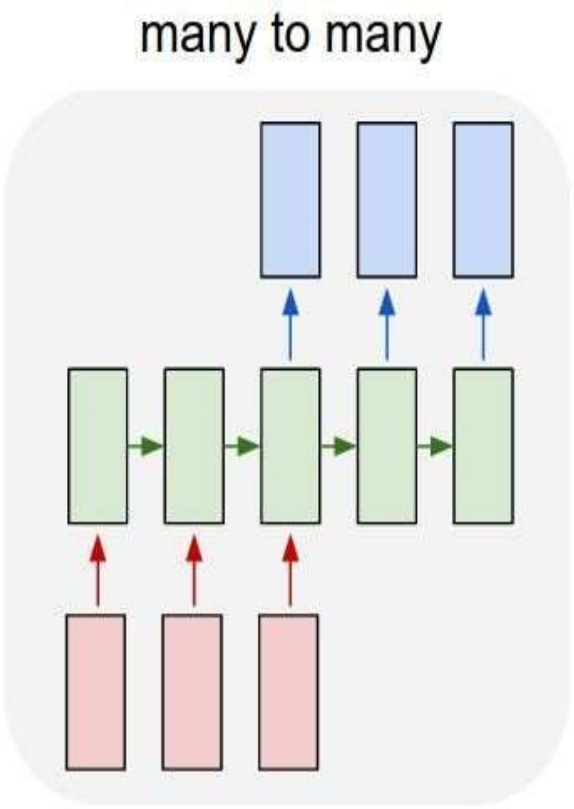
Text Classification

Types of Sequence Labels



Language Model
Part-of-Speech Tags

Types of Sequence Labels



Machine Translation

Types of Sequence Labels

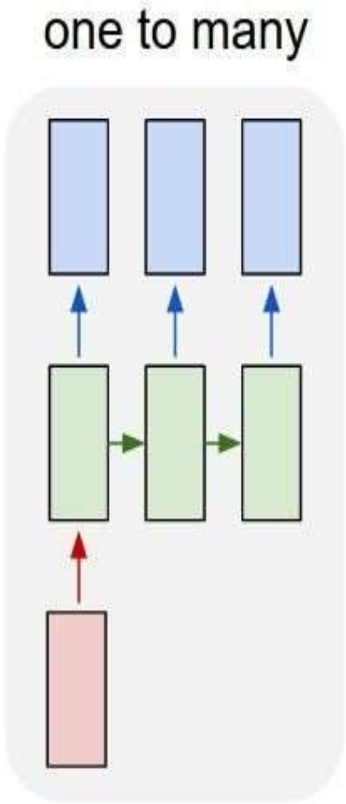


Image Captioning

Types of Sequence Labels

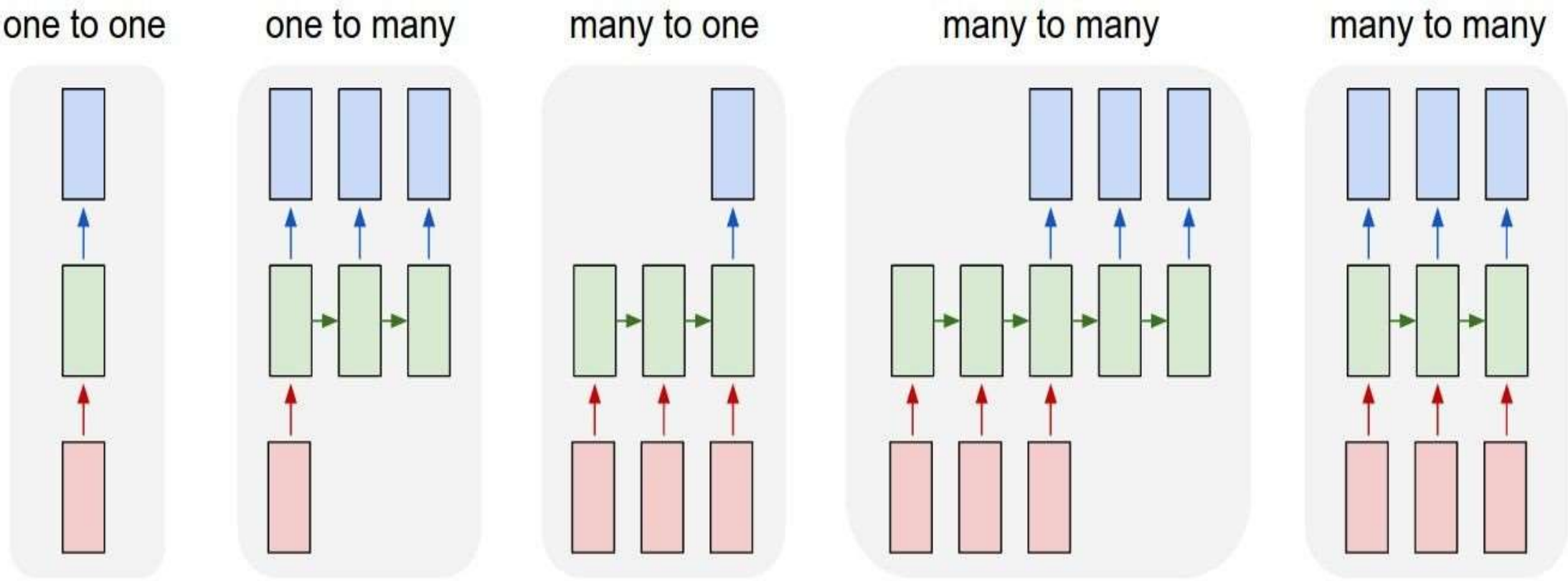


Image Captioning

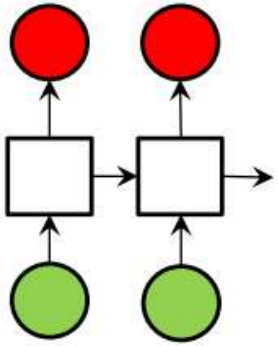
Text Classification

Machine Translation

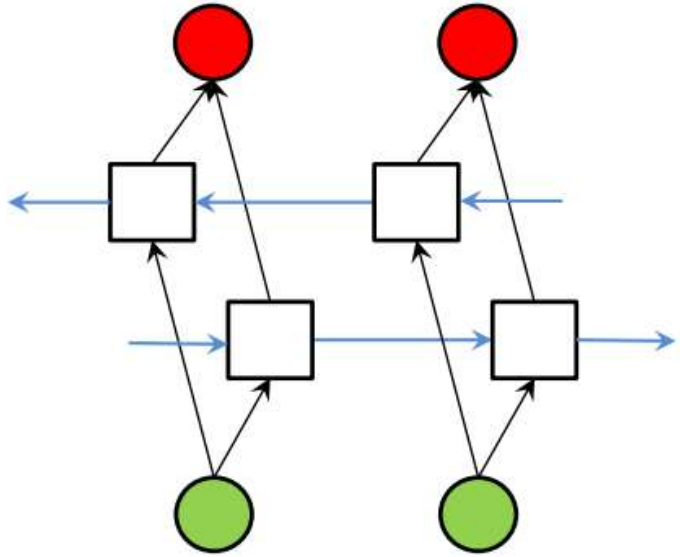
Language Model
Part-of-Speech Tags

Can you see the future?

- Predictions for current observations might be influenced by future observation
  - E.g. “It’s just that, you know, I don’t like that”.
- We can use bi-directional RNNs



RNN

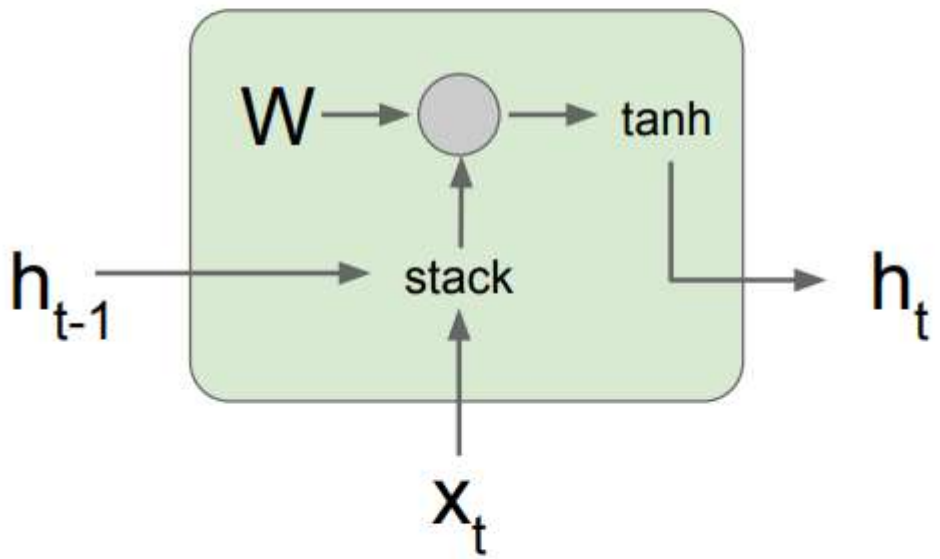


Bidirectional RNN

The Problem with RNNs

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

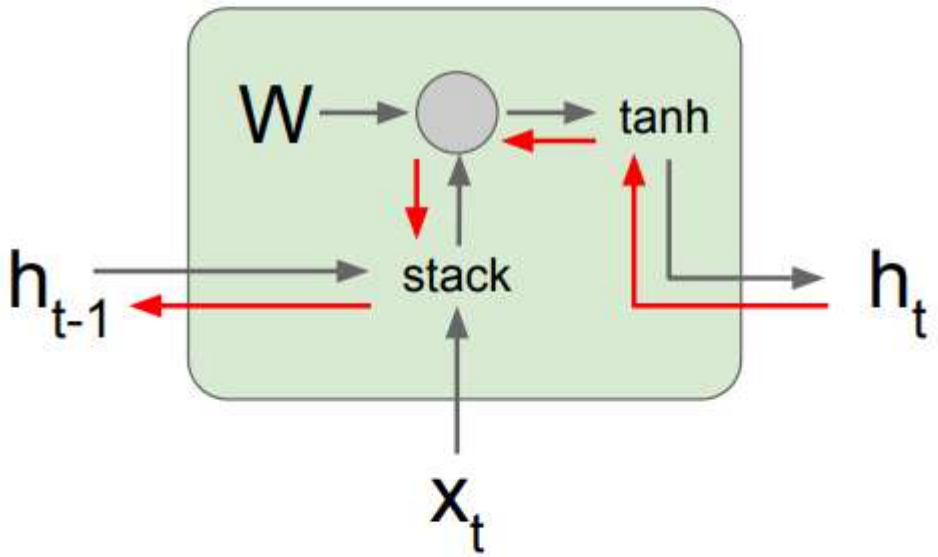
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



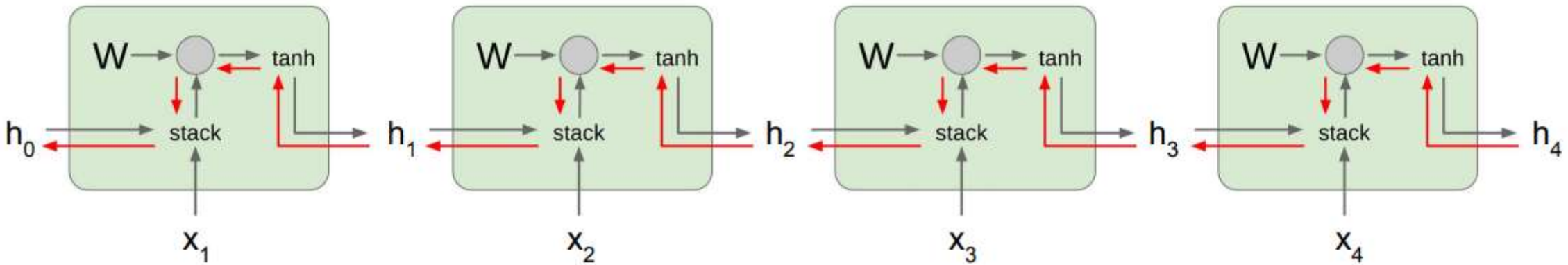
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

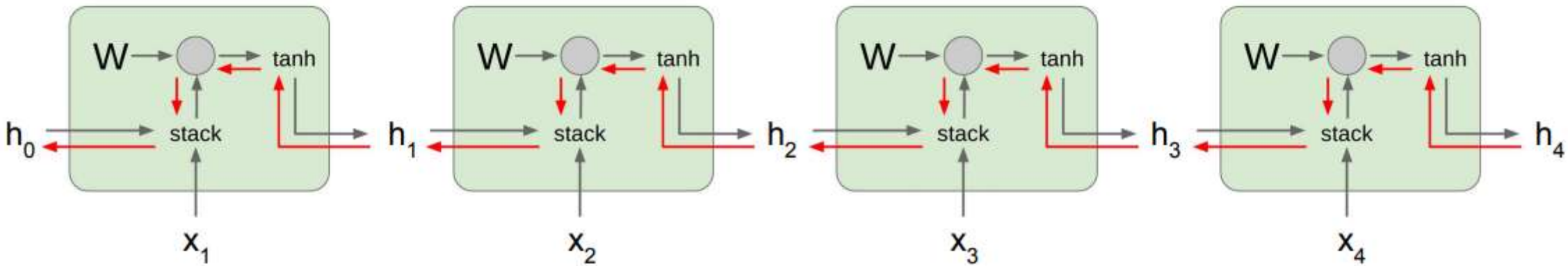
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



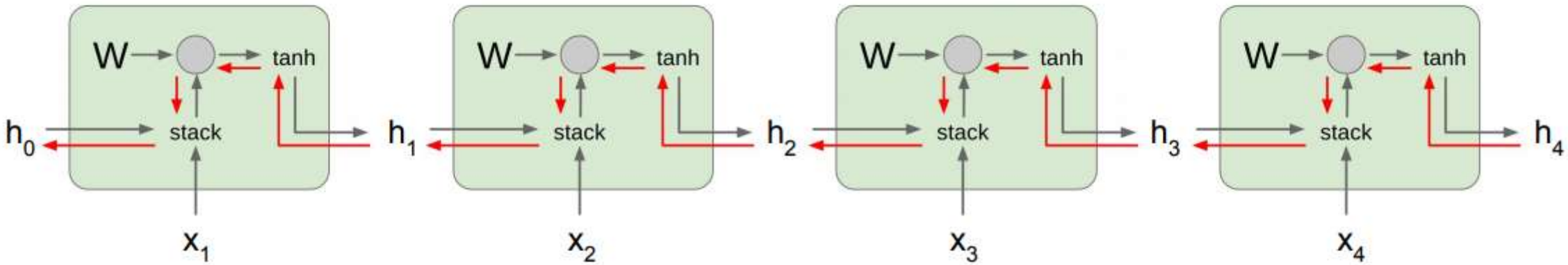
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

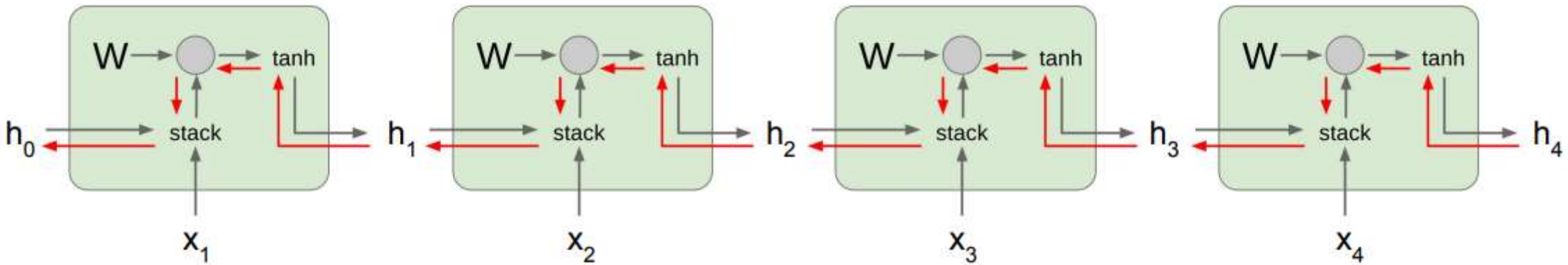
→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

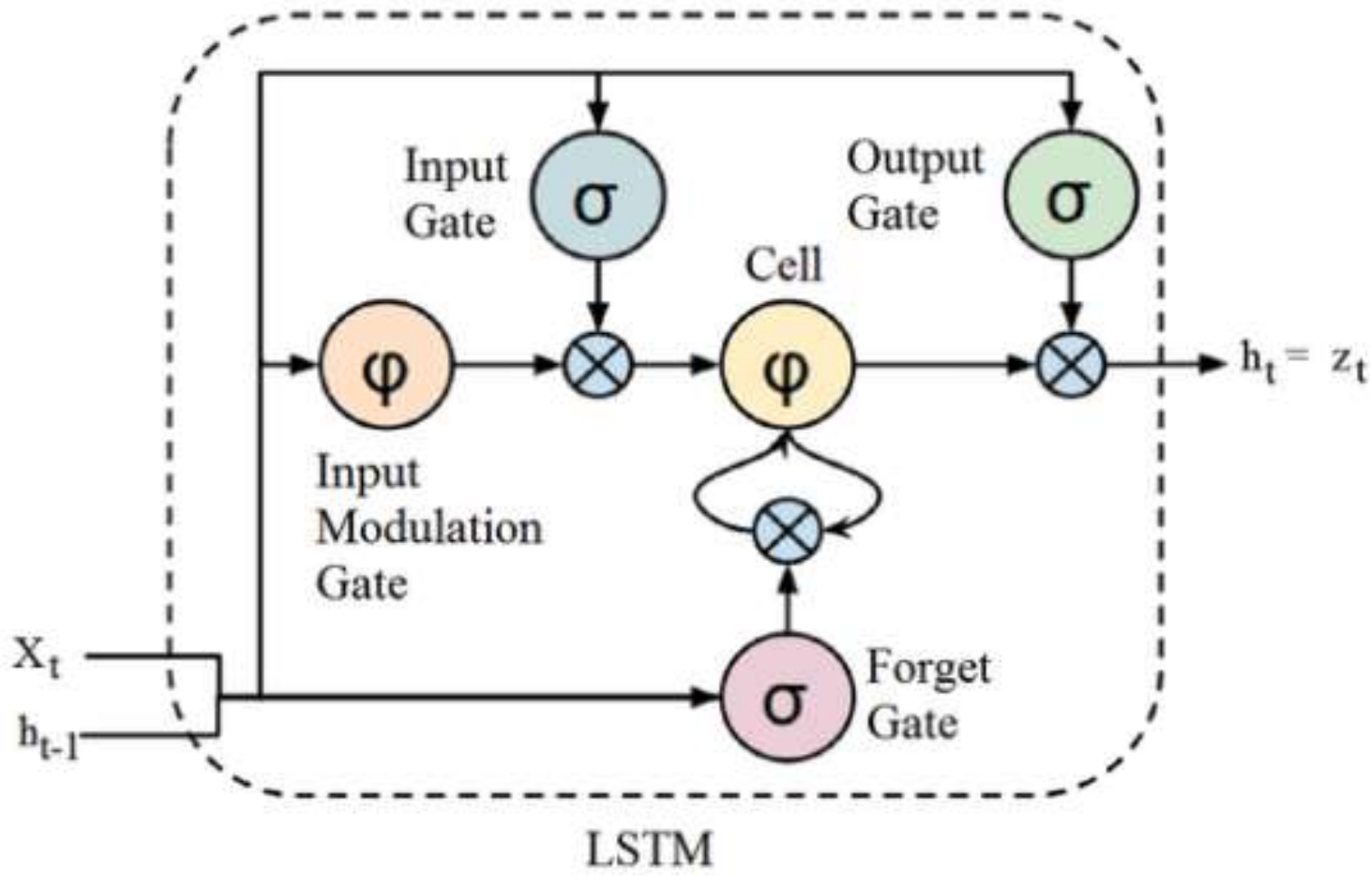
Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

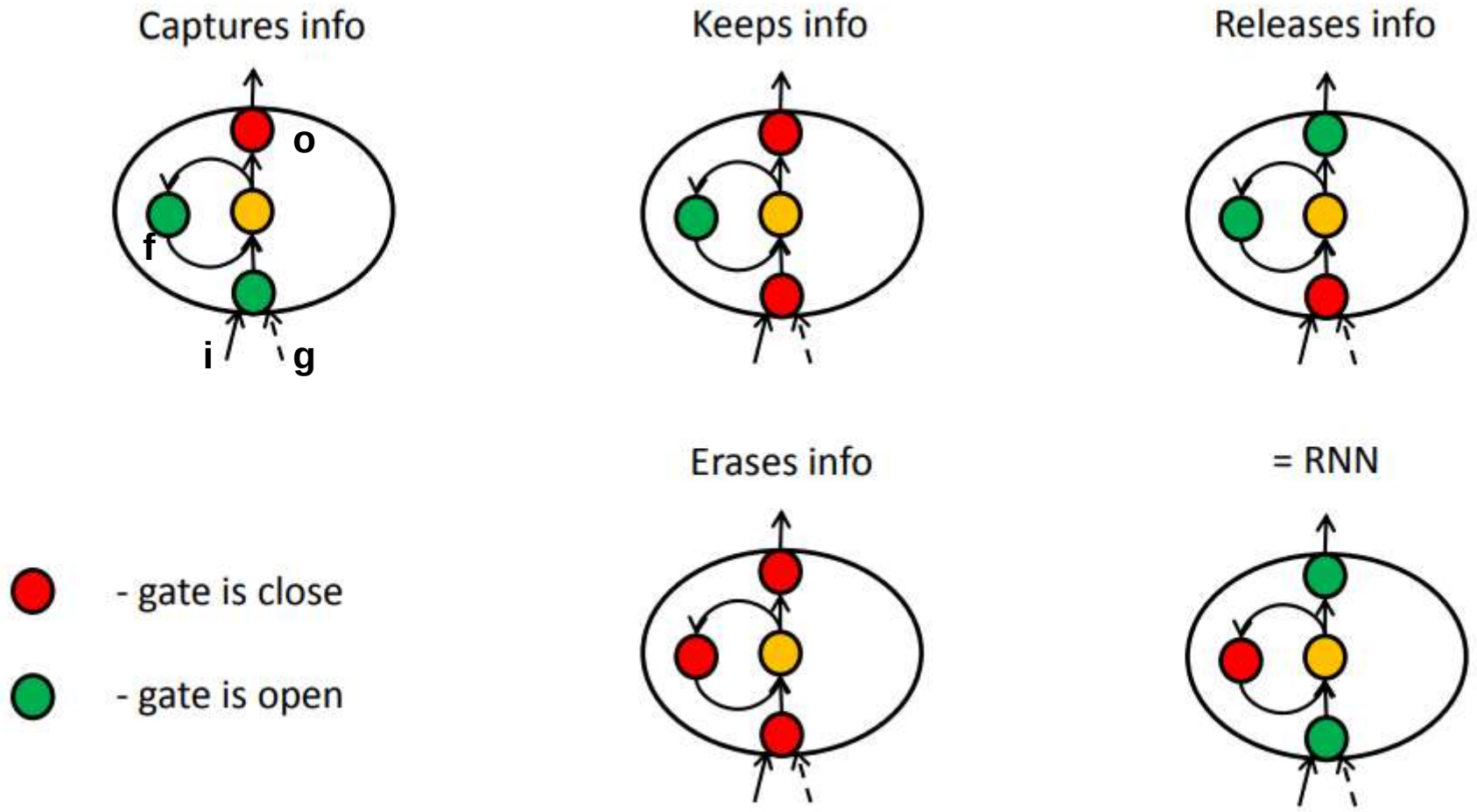
→ Change RNN architecture

LSTM – Long Short Term Memory

- 4 gates:
  - Input
  - Output
  - Input modulation
  - Forget



LSTM States



[Pictures: E Lobacheva, D Vetrov]

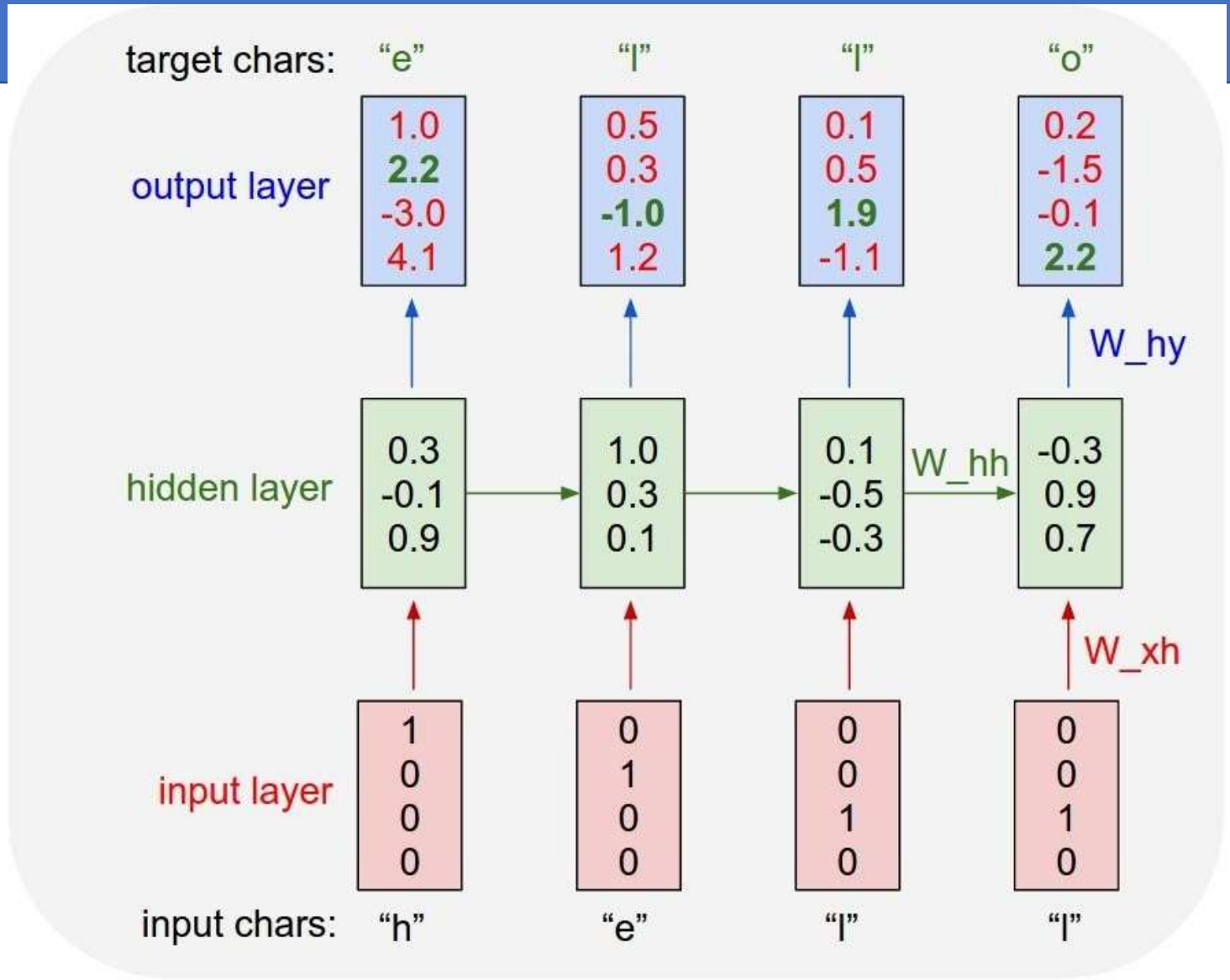
Generating Sequences with RNNs

- Start with some hidden state and input
- Compute the most probable output using the RNN
- Feed the output to the network as input, and continue
- Think: How do we know that the sequence has ended?
- Think: During training, should you use the real output, or the network's?
- Think: Can we do better than this greedy method of selecting the most probable output each time?

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence: **"hello"**



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{étale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

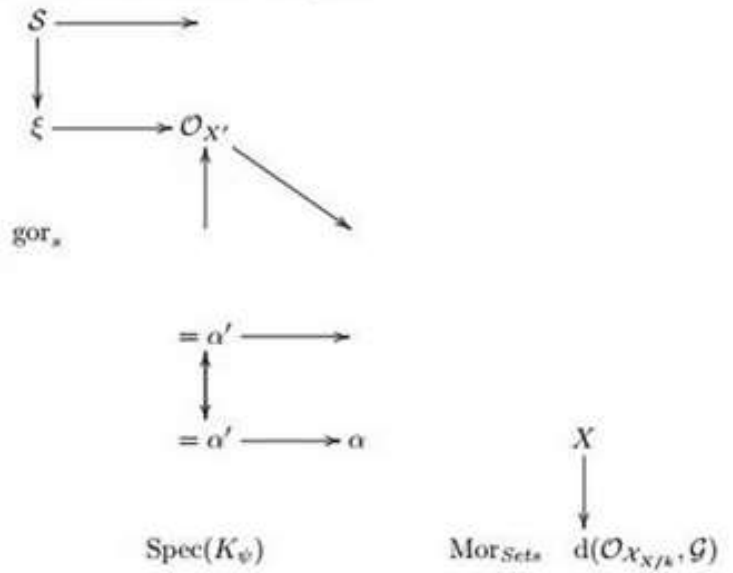
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f_\bullet$. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{étale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\nabla})$$

is an isomorphism of covering of $\mathcal{O}_{X_\ell}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

Image Captioning Dataset

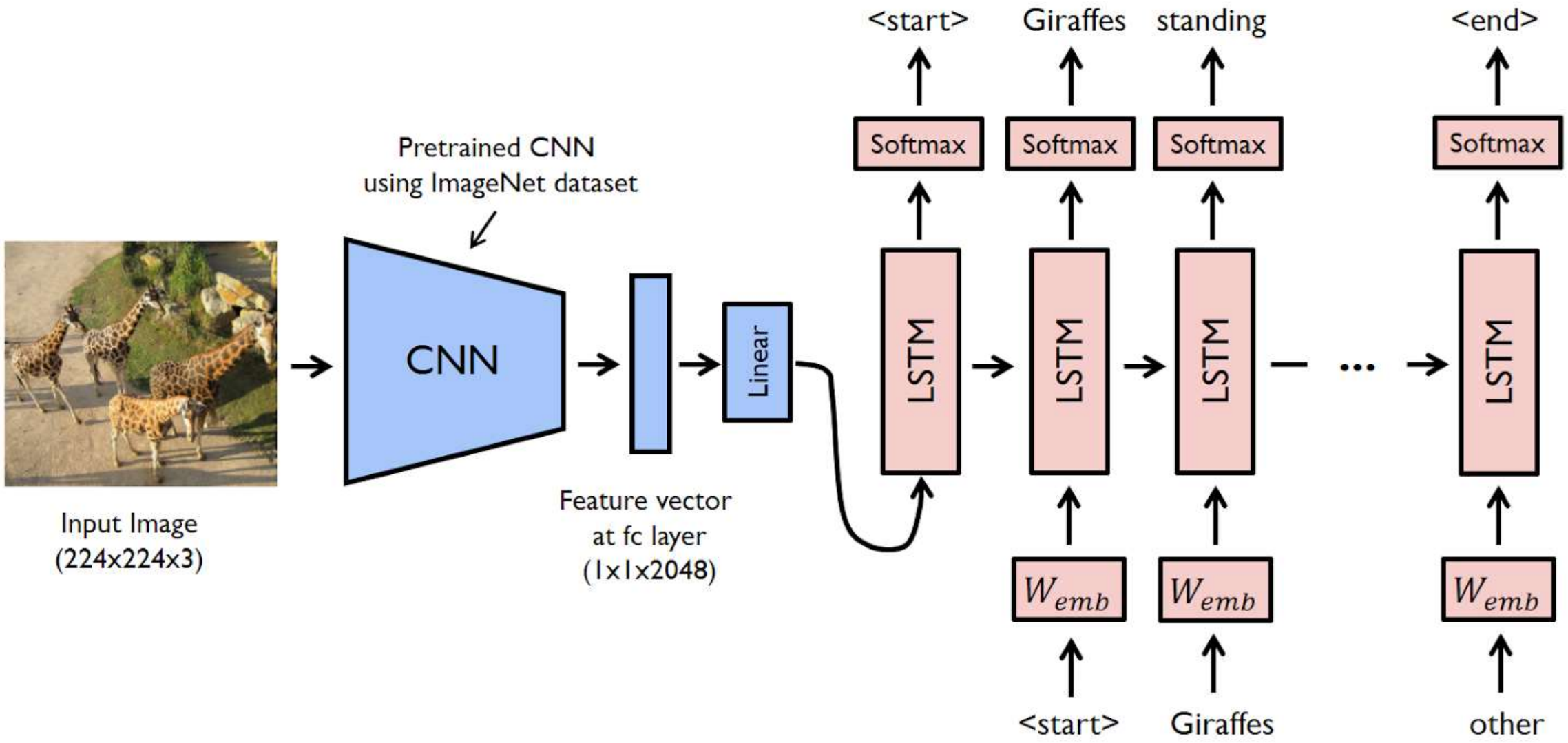
a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.

Microsoft COCO

[Tsung-Yi Lin et al. 2014]

mscoco.org

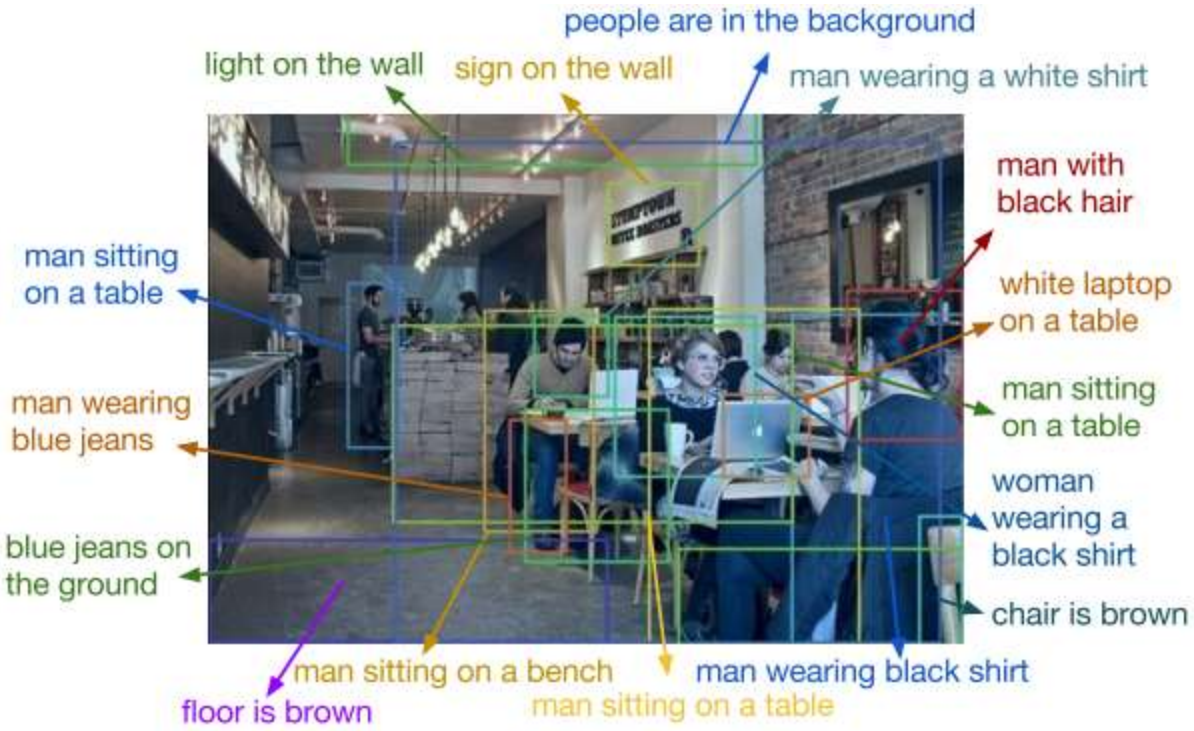
currently:
~120K images
~5 sentences each

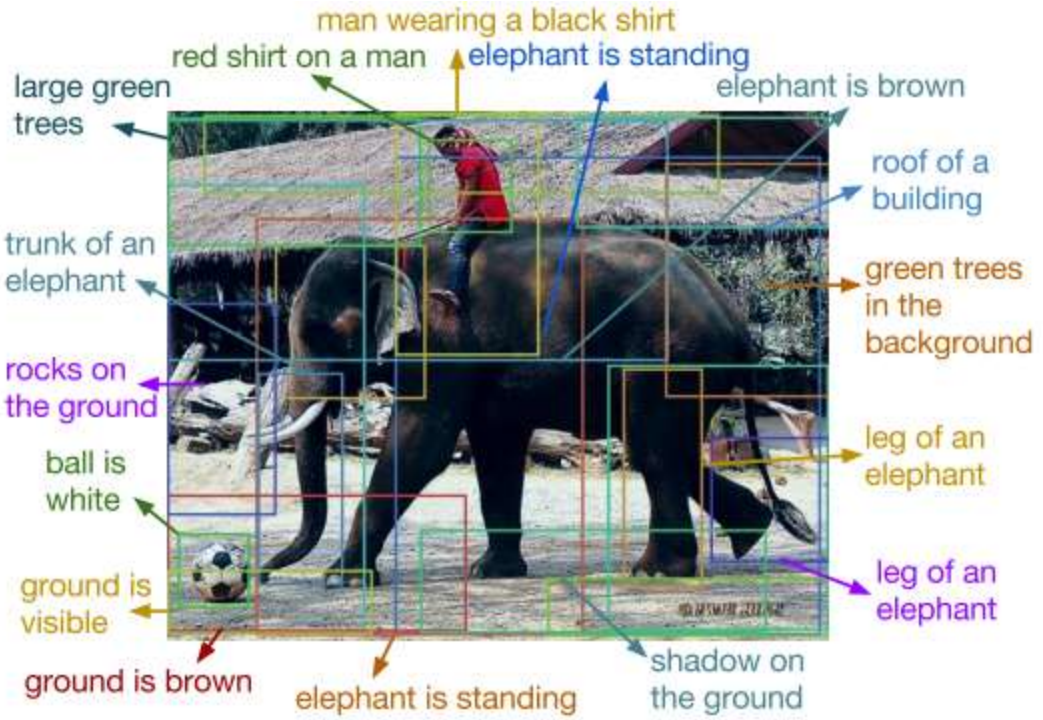

Pretrained CNN
using ImageNet dataset
CNN
Input Image
(224x224x3)
Feature vector
at fc layer
(1x1x2048)
Linear
LSTM
LSTM
LSTM
LSTM
Softmax
Softmax
Softmax
Softmax
<start>
Giraffes
standing
<end>
W_{emb}
W_{emb}
W_{emb}
<start>
Giraffes
other

Can we take this further?

- We know how to generate a caption for an image
- In Object detection, we encode a region using a CNN and used it to detect objects in it
- We can encode a region using CNN and create a label for it
- We can use a Region Proposal Network
- Can be used for image retrieval - "find me an image with a cat riding on a skateboard"

Describing images





Common NLP Tasks