Machine Learning

Representation learning

Dan Goldwasser

dgoldwas@purdue.edu

Some slides based on M. Gormley
NN Example slides

Multi Layer NN: forward computation

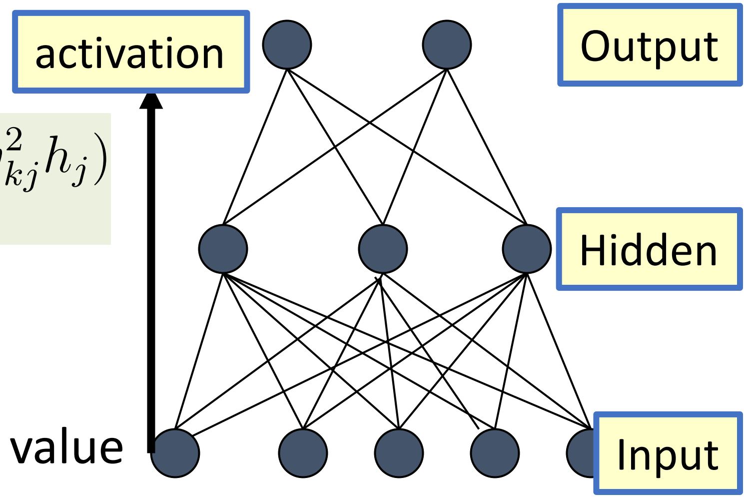
- Observe an input vector x
- *Push x through the network:*
  - For each **hidden unit** compute the activation value

$$h_j = \sigma(t_j) = \sigma(\sum_i w^1_{ji}\; x_i)$$

  - For each **output value**, compute the activation value coming from the hidden units

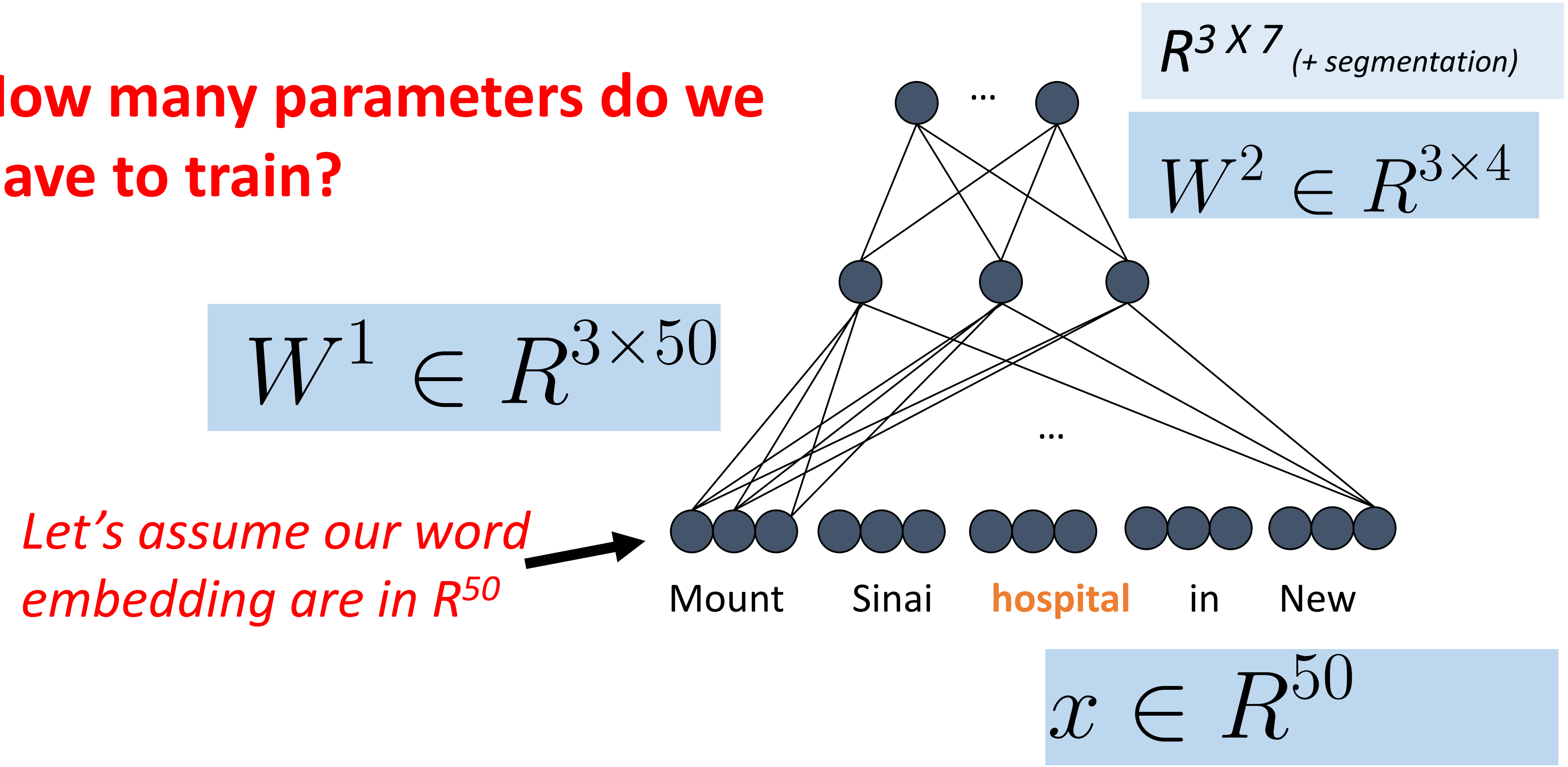
$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w^2_{kj} h_j)$$

  - **Prediction:**
    - **Categories**: winner take all
    - **Vector**: take all output values
    - **Binary outputs**: Round to nearest 0-1 value



Let's recall our NER problem

How many parameters do we have to train?



<u>Question</u>: *How would this architecture change if we wanted to add NE segmentation as well?*

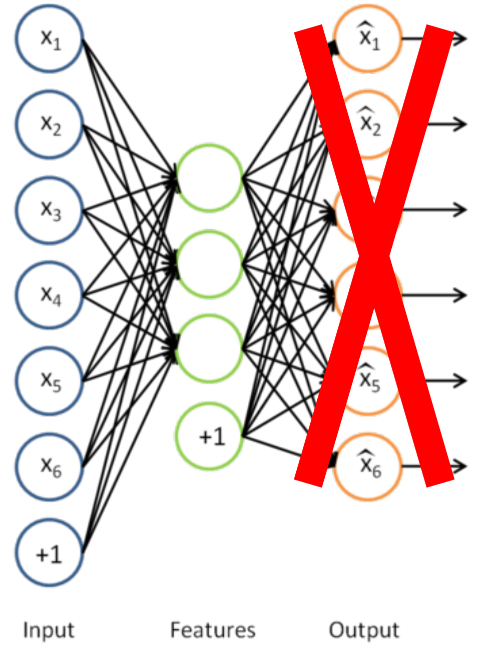
Learning Hidden Layer Representation

- **NN can be seen as a way to learn a feature representation**
  - Weight-tuning sets weights that define hidden units representation most effective at minimizing the error
- Backpropagation can define new hidden layer features that are not explicit in the input representation, but which capture properties of the input instances that are most relevant to learning the target function.
- *Trained hidden units can be seen as newly constructed features that re-represent the examples so that they are linearly separable*

Sparse Auto-Encoder

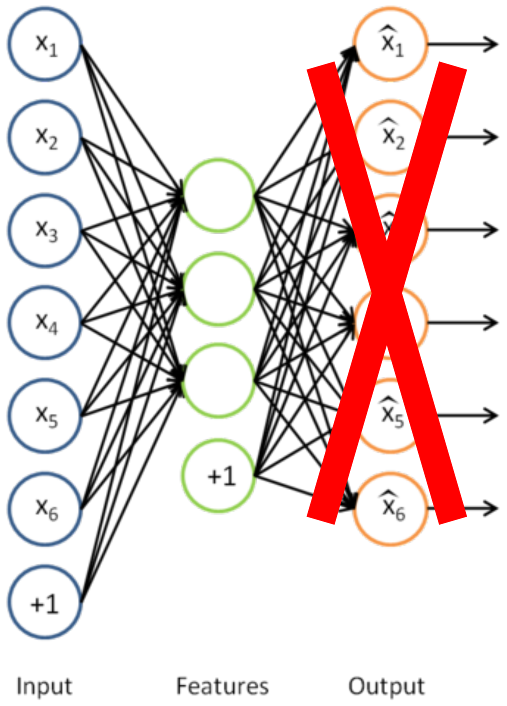
Goal: perfect reconstruction of the input vector x, by the output x’

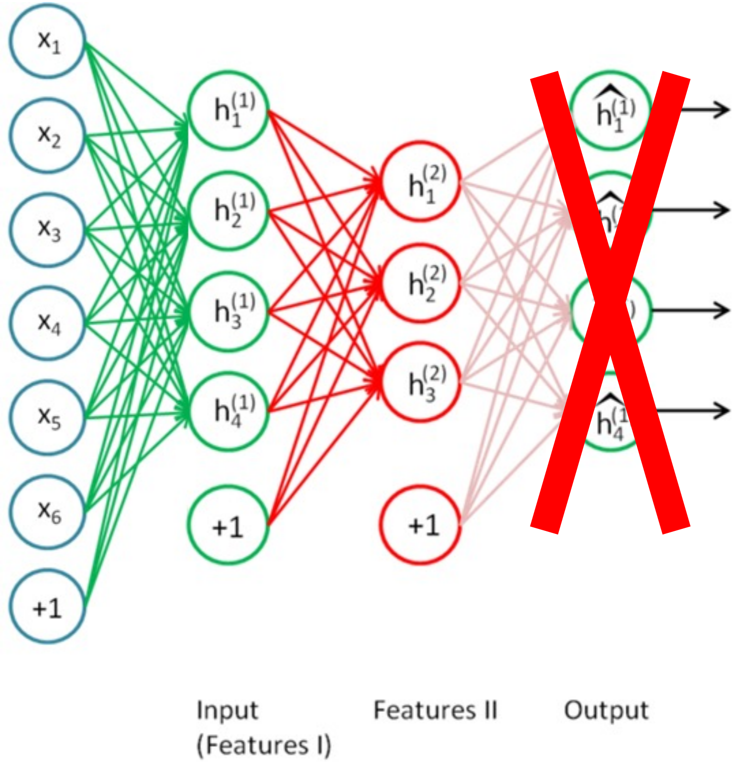
- **Simple approach**:
  - Minimize the error function l(h(x),x)
  - After optimization:
    - Drop the reconstruction layer



Stacking Auto Encoder

- Add a new layer, and a reconstruction layer for it.
- Repeat.





Representation Learning

- Learning Deep Structured Semantic Models for Web Search using Clickthrough Data. Huang et-al. CIKM'13
- **IR goal:** *find relevant documents given a short query*
  - **Challenge:** *how can a short query capture the intent and information need of the user?*
- **Settings**: *Massive clickthrough log data, consisting of documents and search queries.*

Paper Review

- Our goal is to learn a representation for queries such that they match documents selected by users.
- **What is the simplest way to do that given the data?**
- Given a document and query, are the two a good match or not?
- How should we encode each element?

Learning Deep Structured Semantic Models for Web Search using Clickthrough Data. Huang et-al. CIKM'13

Word Hashing

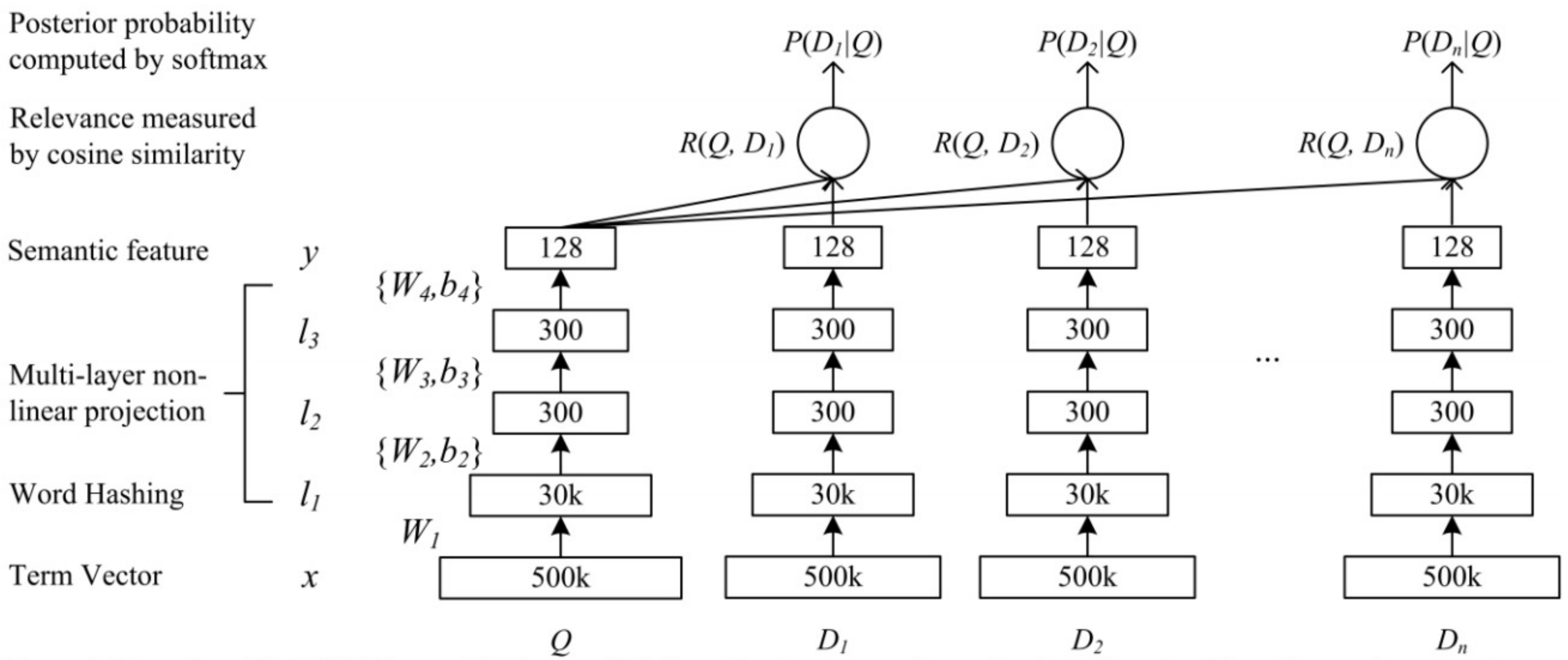
- One of the key challenges: map long documents and queries to the same space.
- Key idea: break words into ”bag of ngrams”
  - e.g. *#good#* ➔ *{ #go, goo, ood, od# }*

	Letter-Bigram		**Letter-Trigram**	
Word Size	**Token Size**	**Collision**	**Token Size**	**Collision**
40k	1107	18	10306	2
500k	1607	1192	30621	22

Table 1: Word hashing token size and collision numbers as a function of the vocabulary size and the type of letter ngrams.

Objective Function

- Define the relationship between two document vectors: $R(Q,D) = \text{cosine}(y_Q, y_D) = \frac{{y_Q}^T y_D}{\|y_Q\|\|y_D\|}$
- Objective: maximize the prob. of relevant docs, give query: $P(D|Q) = \frac{\exp(\gamma R(Q,D))}{\sum_{D' \in \mathbf{D}} \exp(\gamma R(Q,D'))}$
  - In practice, sample a few negative examples to represent **D**



Paper Review

#	Models	NDCG@1	NDCG@3	NDCG@10
1	TF-IDF	0.319	0.382	0.462
2	BM25	0.308	0.373	0.455
3	WTM	0.332	0.400	0.478
4	LSA	0.298	0.372	0.455
5	PLSA	0.295	0.371	0.456
6	DAE	0.310	0.377	0.459
7	BLTM-PR	0.337	0.403	0.480
8	DPM	0.329	0.401	0.479
9	DNN	0.342	0.410	0.486
10	L-WH linear	0.357	0.422	0.495
11	L-WH non-linear	0.357	0.421	0.494
12	**L-WH DNN**	**0.362**	**0.425**	**0.498**

Table 2: Comparative results with the previous state of the art approaches and various settings of DSSM.

10,000 feet view

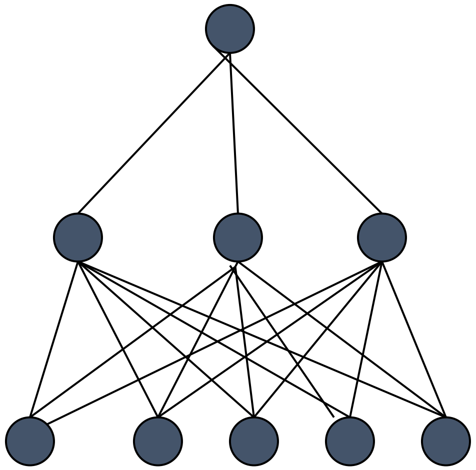
- Neural networks are an extremely flexible way to define complex prediction models.
  - **Simple update rule:** propagate the error on the architecture of the network (essentially DAG).
  - All deep learning models share this property, **just different DAGs**!
- **Key issues**:
  - Preventing over fitting
  - Representation learning, pre-training with minimal supervision

10,000 feet view

- So far, we looked at simple classification problems.
  - Assume a word window, that provides fixed sized inputs.
    - In case you “run out of input” – zero padding.
- What can you do if the size of the input is not fixed?
  - Some notion of compositionality is needed
  - Simplest approach: sum up word vectors
    - Document structure is lost.. **Can we do better?**

Open Question

Recall the NER problem and the architecture associated with it.



Mount Sinai ***hospital*** in New York

Is the center word a Named Entity?

How can we define a NN for other NLP problems?
E.g., Sentiment Analysis defined over complete sentences?
(**hint**: how do you deal with different sized inputs?)

Distributional Similarity

A bottle of tesgüino is on the table
Everybody likes tesgüino
Tesgüino makes you drunk
We make tesgüino out of corn.

Question:

What is tesgüino?

Tesgüino = (Bottle = 123, Table =54, drunk = 141, Corn = 91, …)

Bourbon = (Bottle = 231, Table =41, drunk =231, corn =121, …)

Vodka = (Bottle = 311, Table =82, drunk =321, corn =0, …)

Distributional Similarity

Given the vector based representation of words
we can compute their similarity easily -

$$\cos(v, w) = \frac{v \cdot w}{|v|\ |w|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Using Positive PMI, ensure that Cosine similarity
will have non-negative values

A machine learning perspective

- So far we looked at words as **discrete objects**.
  - For example, when building a sentiment classifier, each word was represented as a different coordinate

- "Great" = [0,0,0,0,0,0,0,1,0,0,... 0]
- "Awesome" = [0,0,0,1,0,0,...,0]
  - *This is known as "**one hot**" representation.*

A machine learning perspective

- Using “one-hot” representation, the connections between words are lost.

- We typically designed **complex feature functions** to get over that:
  - Maintain a dictionary of related words according to
    - Meaning (identify synonyms)
    - Word group (slang, function words, positive, negative,..)

A machine learning perspective

- **Can we use vector based methods to represent words other decision tasks?**
  - We can potentially overcome lexical sparseness problems
  - Reduce the dimensionality of the learning problem
- How should we evaluate the learned semantic representation?

Word Embedding

- Basic idea: represent words in a continuous vector space.
  - Similar idea as using PMI
- **Key difference:**
  - Find low dimensional **dense** representation
  - **Instead of counting co-occurrence, use discriminative learning methods**
    - Predict surrounding words

Word2Vec

"AI fields such as NLP, machine learning, vision, have increased in popularity in recent years"

- For each word, predict other words in window C
- **Training Objective**: maximize the probability of context word, given the current word.

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j<c}\log p(w_{t+j}|w_t)$$

Word2Vec

- We want to evaluate $p(w_o|w_i)$
- For each word maintain **two** vectors
  - **Inside** word and **outside** word (context)
  - V represent inside, V' outside.

$$p(w_o|w_i) = \frac{\exp(v'_{w_o} v_{w_i})}{\sum_{w=1}^{W} \exp(v'_w v_{w_i})}$$

Efficient Implementation

- For non-trivial vocabulary, the normalization factor is too costly to compute accurately.

$$p(w_o|w_i) = \frac{\exp(v'_{w_o} v_{w_i})}{\sum_{w=1}^{W} \exp(v'_w v_{w_i})}$$

- **Skip-gram with negative sampling**
  - Binary logistic regression for a small subset:
    - True pair, small subset of negative examples.

Skip-gram with Negative Sampling

- New objective function:

$$\log \sigma(v'_{w_o} v_{w_I}) + \sum_{i=1}^{k} \mathop{\mathbb{E}}_{w_i \sim P_n(w)} \left[\log \sigma(-v'_{w_i}{}^{T} v_{w_I})\right]$$

Maximize the probability of center + context words

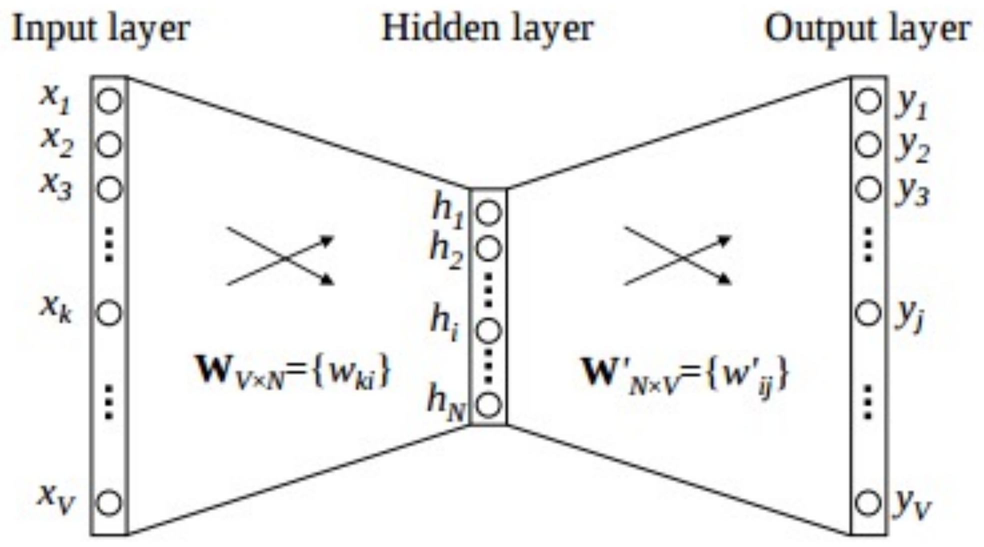
Minimize the probability of random words

Note:

- Only pick a **small subset** of negative examples
- samples are drawn from a distribution: $P_n(w)$
- $P_n(w)$ captures unigram statistics, modified to increase the probability of sampling low frequency words.

Word Embedding vs. AE

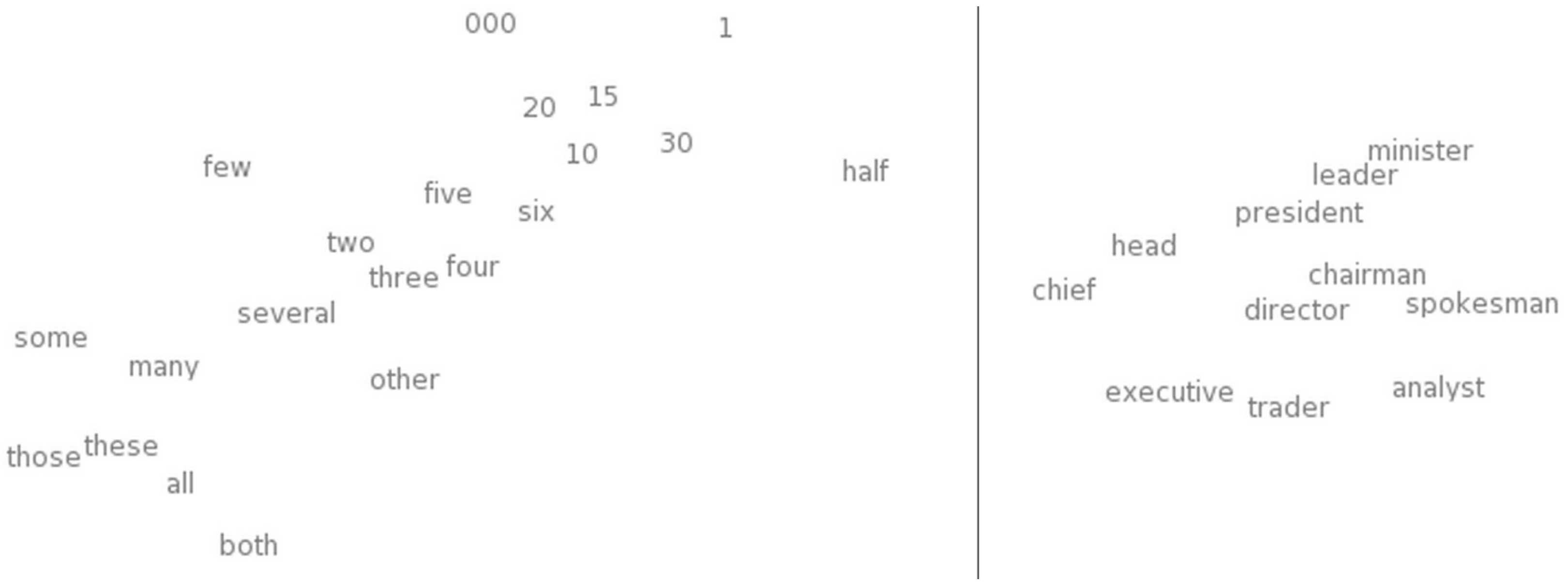
- Is this the same as an Auto-Encoder?



word2vec model architecture

Word Embedding

- **Word embedding**: *move to a low dimensional, real valued dense representation of the input*
  - Key idea: similar words should have similar vectors



Turian et-al 2010

Word2Vec

```
Enter word or sentence (EXIT to break): Chinese river

                 Word       Cosine distance
-------------------------------------------
        Yangtze_River              0.667376
              Yangtze              0.644091
       Qiantang_River              0.632979
    Yangtze_tributary              0.623527
     Xiangjiang_River              0.615482
        Huangpu_River              0.604726
       Hanjiang_River              0.598110
        Yangtze_river              0.597621
          Hongze_Lake              0.594108
              Yangtse              0.593442
```

Mikolov et-al 2013

Reminder: *Language Models*

- A Language model defines a probability distribution over a sequence of words:

$$P(w_1, ..., w_n)$$

- Simple, yet very useful idea!
  - Estimate using a large collection of text (no supervision!)
  - **P("I like NLP") > P("me like NLP")**

- Key assumption: **Markov model**

Reminder: *Language Models*

- Markov model: *Probability of word$_i$ is conditioned on previous **n** words*

$$P(w_1, ..., w_n) = \prod_{i=1}^{n} P(w_i | w_1, .., w_{i-1}) = \prod_{i=1}^{n} P(w_i | w_{i-n-1}, .., w_{i-1})$$

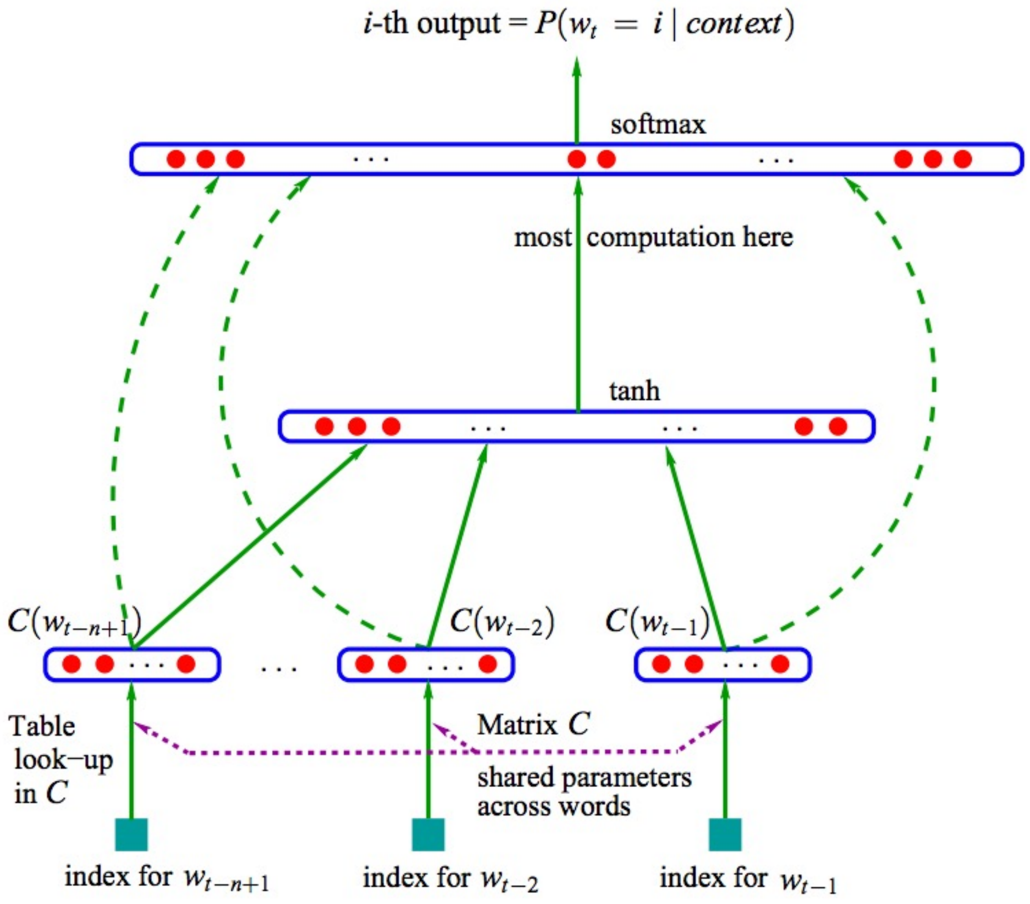
  - **Why is it needed?**
  - What will happen if we increase/decrease n?

Reminder: *Language Models*

- Language models will become **more accurate**, yet **harder to estimate** as n grows.
  - Even for a small size of n, the number of parameters is too large!

- **Question**: *do we need to condition on more than 2-3 words?*

Neural Language Model – Take 1

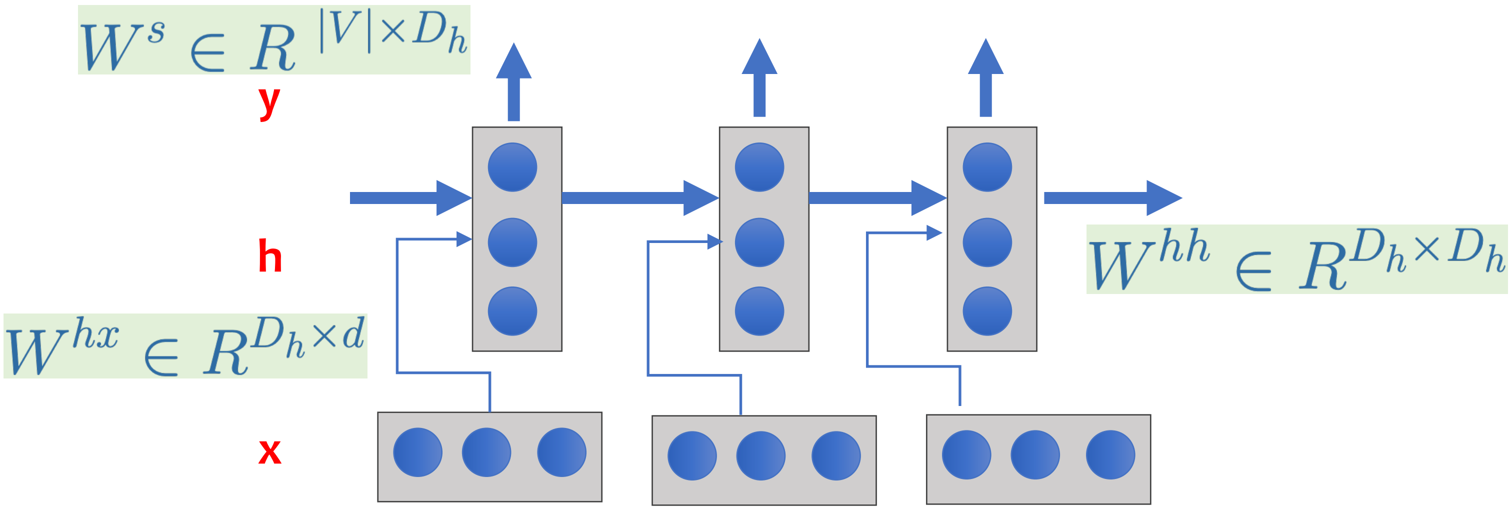


A Neural Probabilistic Language Model. Bengio et-al 2003

Recurrent Neural Networks

- A NN version of a language model.
  - More broadly: deal with **data over time.**
- Unlike N-gram models, an RNN conditions the current word on all previous words.
- **Efficient**, both in time and space

Recurrent Neural Networks



Input is a word (vectors) sequence: $x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n$

At any given time step *i* :

$$h_i = \sigma\left(W^{hh} h_{i-1} + W^{hx} x_i\right)$$

$$\hat{y} = \mathrm{softmax}(W^s\ h_i)$$

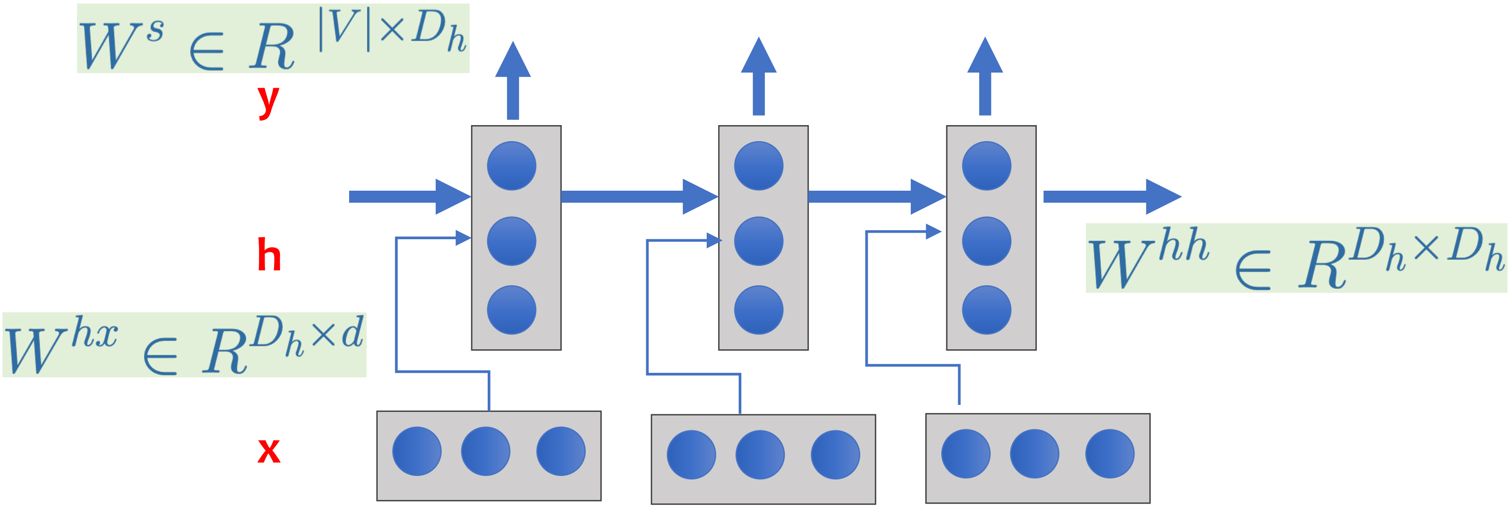
$$P(x_{i+1} = v_j | x_i, \ldots, x_1) = \hat{y}_{i,j}$$

Recurrent Neural Networks

- Similar to traditional LM, we define a probability distribution, estimated using an RNN
  - I.e., $\hat{y} \in R^{|V|}$ is a probability distribution over V
  - We can define a loss function (cross-entropy):

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

RNN: Forward Propagation



The cat sat on the mat. Where has the cat sat?

The cat sat on the hat. Where has the cat sat?

RNN-LM Results

Model	Perplexity
Kneser-Ney 5-gram	141
Random forest [Xu 2005]	132
Structured LM [Filimonov 2009]	125
Feedforward NN LM	116
Syntactic NN LM [Emami 2004]	110
RNN trained by BP	113
RNN trained by BPTT	106
4x RNN trained by BPTT	98

Extensions of recurrent neural network language model by Mikolov et al'11

Beyond Language Models

- **Consider the homework problem – *text classification.***
  - How did we approach it so far using NN?
  - Can RNNs change that?

- **Now, consider the NER problem we discussed.**
  - How did we approach it so far using NN?
  - Can RNNs change that?

Dealing with Structures. *New ideas?*

- Structured prediction – dealing with multiple decisions at the same time. Modeling the interactions between decisions is the key challenge.

Conceptually two considerations:

- Entity **boundary (Begin, Inside ,Outside)**
- Entity **type (PER,ORG,LOC)**

Begin_p Inside_p

Barak Obama visited Mount Sinai hospital in New York

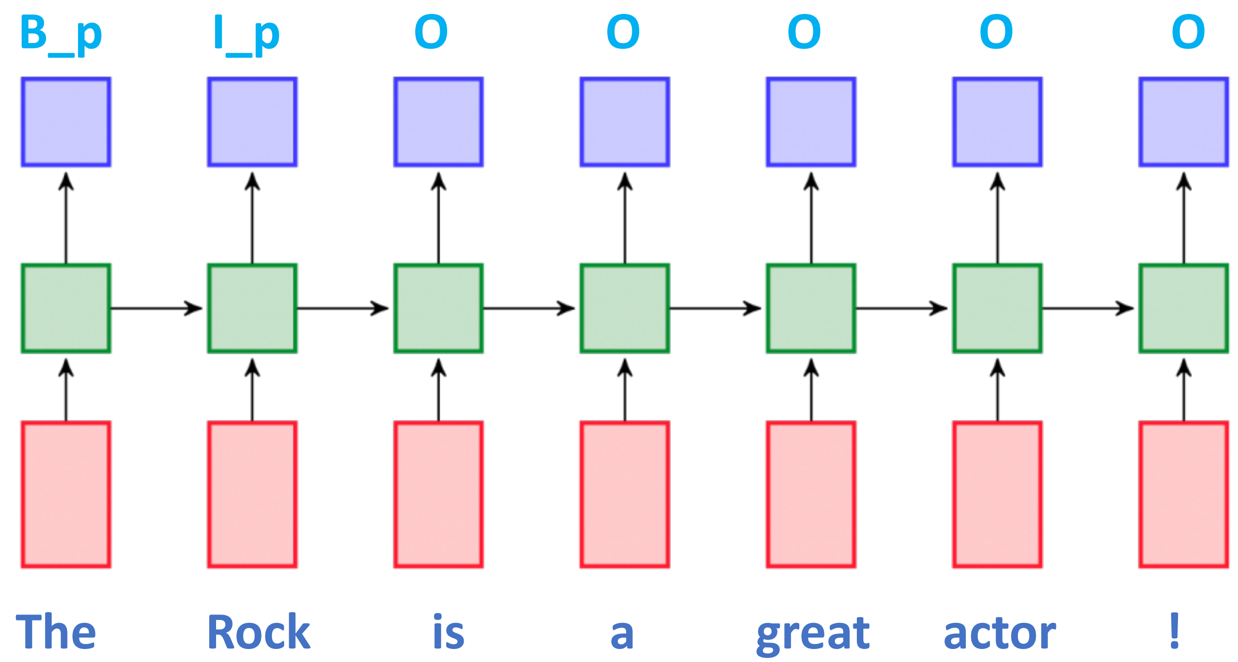
PER ORG LOC

Many of the predictions are context dependent: e.g., Mount Sinai is a LOC, while Mount Sinai hospital is an ORG.

How can you capture it using the machinery we currently have? At what cost?

Beyond Language Models

- Recurrent architectures are extremely flexible.
  - They can be used as **text encoders**,
  - or as **sequence taggers**.



Positive Sentiment

RNN Extensions

- **Key issue**: *long range dependencies between inputs.*
  - “how can we know which word is important to keep around, when predicting the i+1 word?”
- **Solution idea**: *complex hidden units that implement a “memory”*
  - Maintain ”old memories” representing relevant long range dependencies
  - Error updates can be back-propagated at different strengths.

Gated Recurrent Units (GRU)

- Until now, we assumed a simple hidden layer:
  - representing the previous steps and input word

$$h_i = \sigma\left(W^{hh} h_{i-1} + W^{hx} x_i\right)$$

- In GRU's the picture is more complex, it adds **gates,** that control how the hidden state is computed
- *Essentially, more layers that can be learned from data*
  - **Update Gate**
  - **Reset Gate**

Gated Recurrent Units (GRU)

$$h_i = \sigma\left(W^{hh}h_{i-1} + W^{hx}x_i\right)$$ ⬅ **Original RNN**

GRU:

Update Gate: $$z_i = \sigma\left(W^z x_i + U^z h_{i-1}\right)$$

Reset Gate: $$r_i = \sigma\left(W^r x_i + U^r h_{i-1}\right)$$

New memory $$\tilde{h}_i = \tanh\left(W x_i + r_i \circ U h_{i-1}\right)$$

Final memory (aka *hidden Layer*) $$h_i = z_t \circ h_{i-1} + (1 - z_i) \circ \tilde{h}_i$$

Why it works

- Learn a set of parameters for each one of the gates
  - **Recall**: *gates output a probability*
  - If **reset** gate is ~0: **ignore previous hidden state**
    - "forget" irrelevant information
    - **Short term dependencies**
  - If **update** gate ~1: **copy past information**
    - *"remember" past state*
    - **long term dependencies**

$$z_i = \sigma\left(W^z x_i + U^z h_{i-1}\right)$$

$$r_i = \sigma\left(W^r x_i + U^r h_{i-1}\right)$$

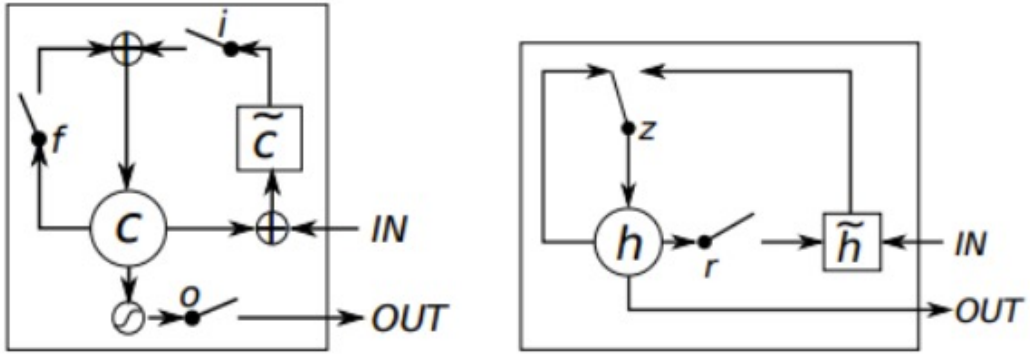
$$\tilde{h}_i = \tanh\left(W x_i + r_i \circ U h_{i-1}\right)$$

$$h_i = z_t \circ h_{i-1} + (1 - z_i) \circ \tilde{h}_i$$

Long-Short-Term-Memories (LSTM)

- Similar (and older!) idea, though more complex
- Input gate $$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
- Forget gate $$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
- Output $$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
- New memory $$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
- Final Memory $$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
- Final hidden state $$h_t = o_t \circ \tanh(c_t)$$

RNN vs. LSTM vs. GRU

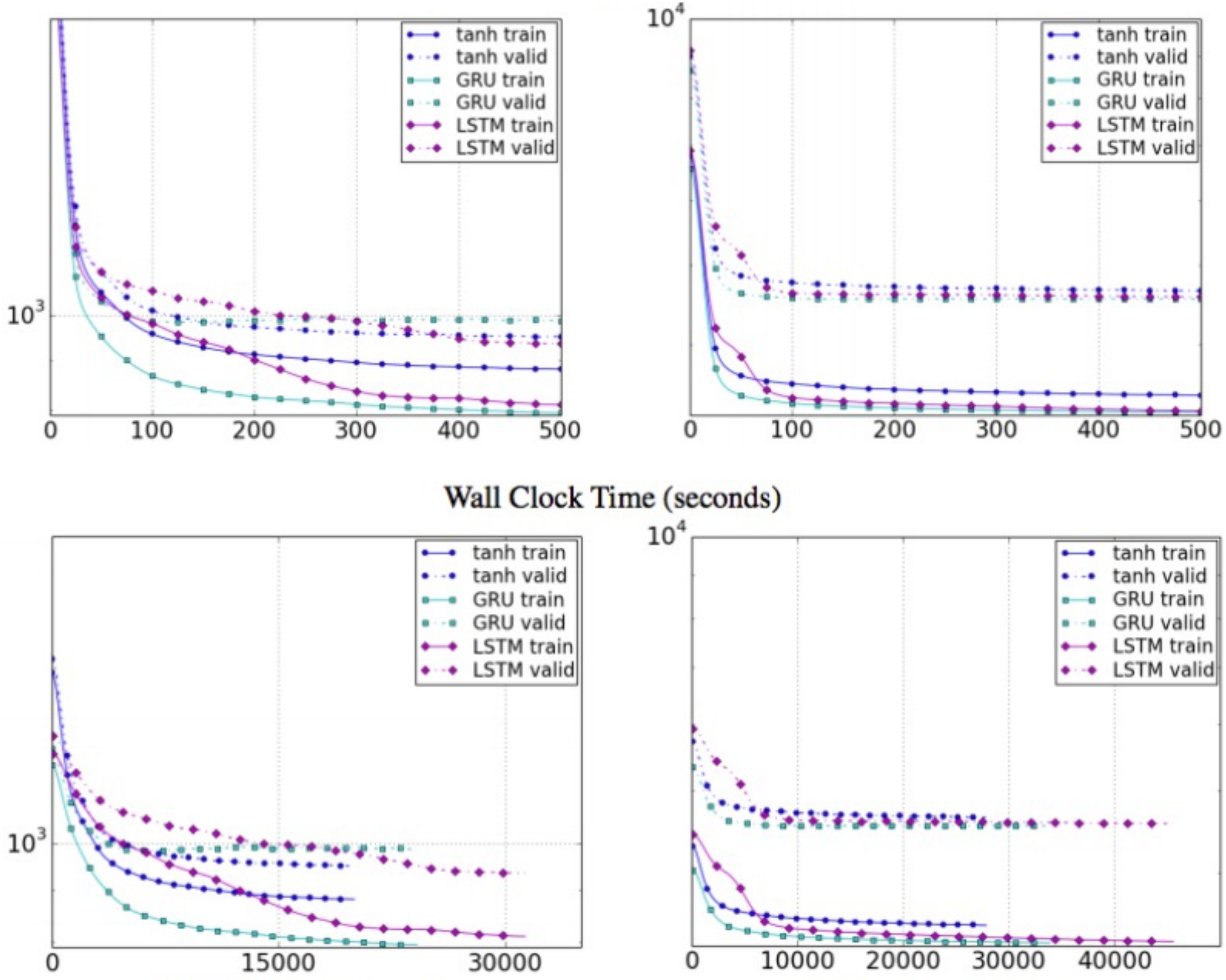


(a) Long Short-Term Memory (b) Gated Recurrent Unit

			tanh	GRU	LSTM
Music Datasets	Nottingham	train	3.22	2.79	3.08
		test	**3.13**	3.23	3.20
	JSB Chorales	train	8.82	6.94	8.15
		test	9.10	**8.54**	8.67
	MuseData	train	5.64	5.06	5.18
		test	6.23	**5.99**	6.23
	Piano-midi	train	5.64	4.93	6.49
		test	9.03	**8.82**	9.03
Ubisoft Datasets	Ubisoft dataset A	train	6.29	2.31	1.44
		test	6.44	3.59	**2.70**
	Ubisoft dataset B	train	7.61	0.38	0.80
		test	7.62	**0.88**	1.26

Table 2: The average negative log-probabilities of the training and test sets.

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. Chung et-al 2014



(a) Nottingham Dataset (b) MuseData Dataset

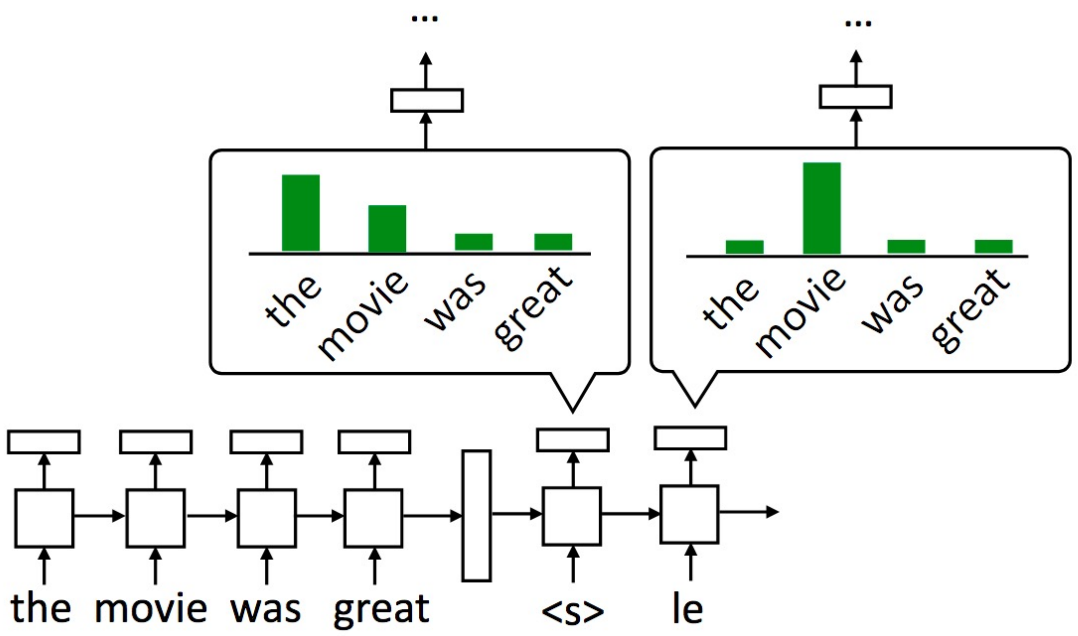
Attention based "reading"

- Many of the challenges introduced by machine comprehension can be addressed using a general solution based on **attention**

- A general tool, currently used in **all** NLP tasks
    - Essentially, learn meaningful associations between inputs and outputs which can represent structural dependencies.

Attention

- **Attention**: at each decoder state computes a **distribution over the source inputs** based on the current decoder state

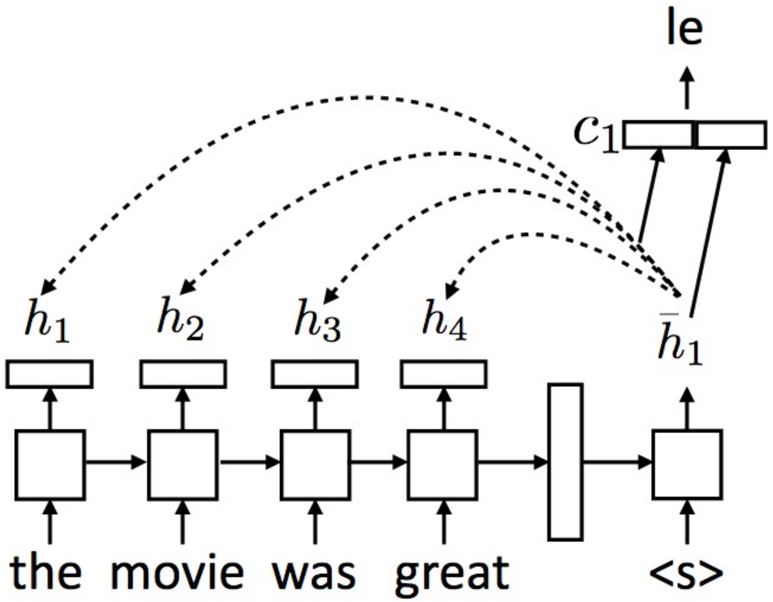


Attention

- For each decoder state compute the weighted sum of input states

Decision at step i:

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W[c_i; \bar{h}_i])$$



$$c_i = \sum_j \alpha_{ij} h_j$$

- Weighted sum of input hidden states (vector)

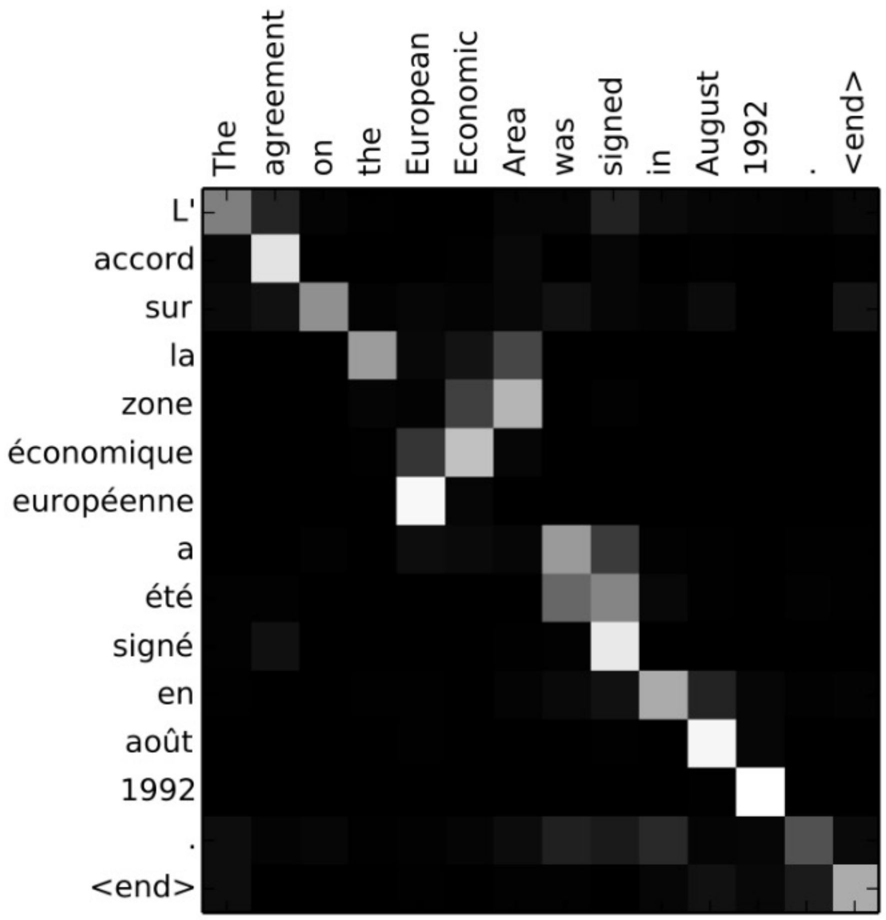
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

- Unnormalized scalar weight

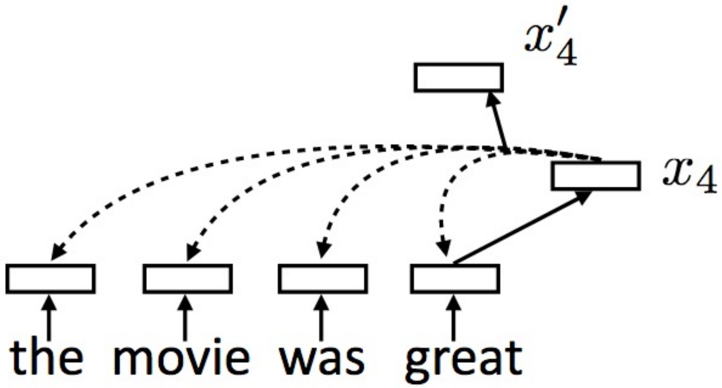
Attention

We can identify the source word context for output predictions



Self-attention

- **A new way to represent structure**
  - *Each word forms a query which computes attention over each word*



$$\alpha_{i,j} = \mathrm{softmax}(x_i^\top x_j)$$ scalar

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j$$ vector = sum of scalar * vector

The representation of each word is a function of its neighbors. Does that sound familiar?

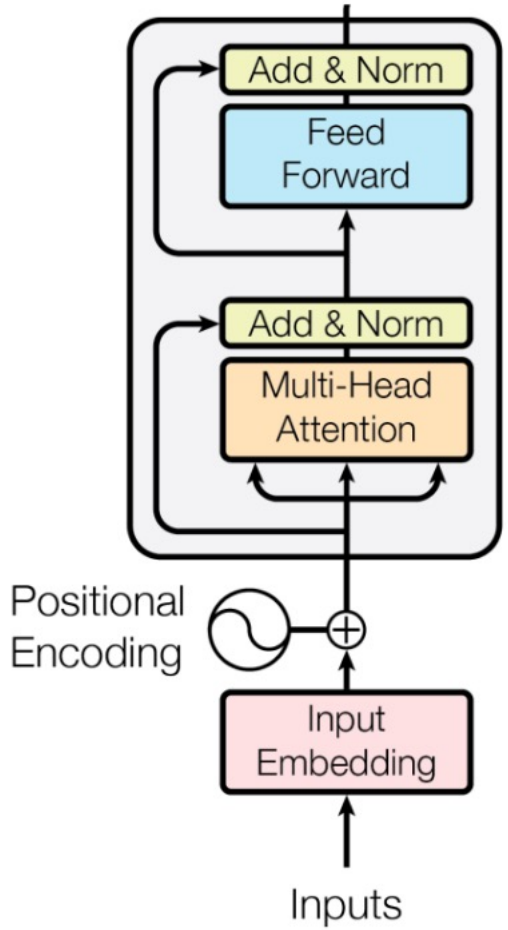
Vaswani et al. (2017)

Transformers

- The idea of self attention was extremely influential in NLP
  - **No fixed position representation as in LSTM instead structure is represented the attention assignments.**

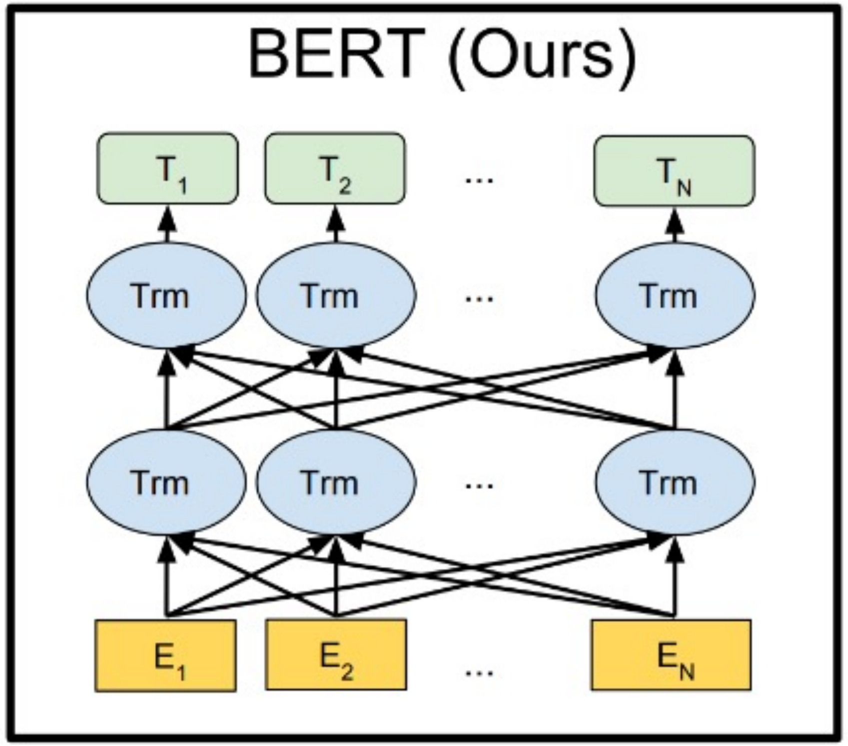
I like bananas but not carrots.
Vs.
I like carrots but not bananas

- In reality, position information is needed, but it is used differently compared to an LSTM, by encoding it as part of the input



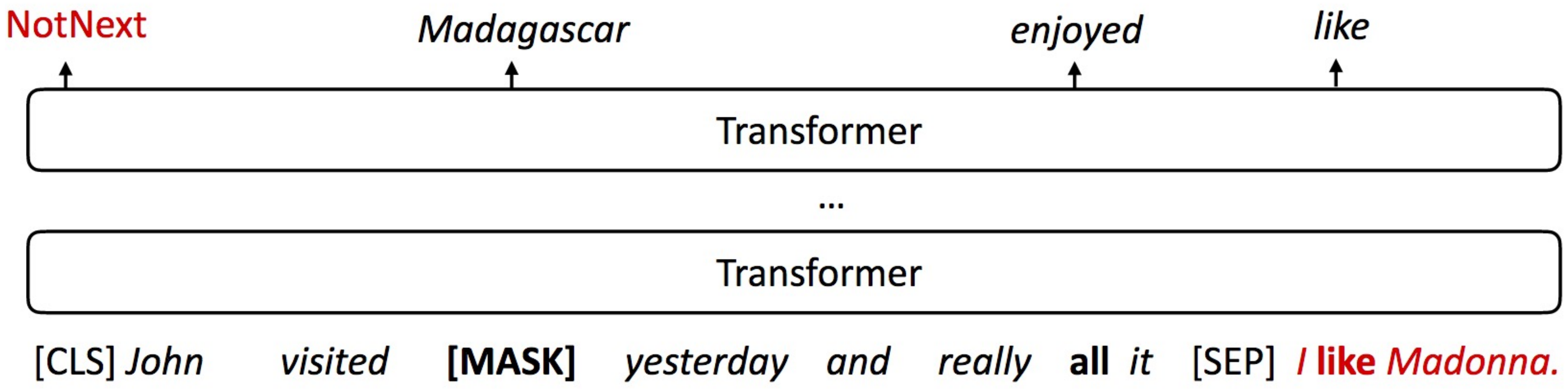
BERT

- **Transformer-based approach** *instead of an LSTM-based like ELMo*.
  - Transformer vs. LSTM
  - Masked language objective instead of usual LM
  - Fine-tuned at test time



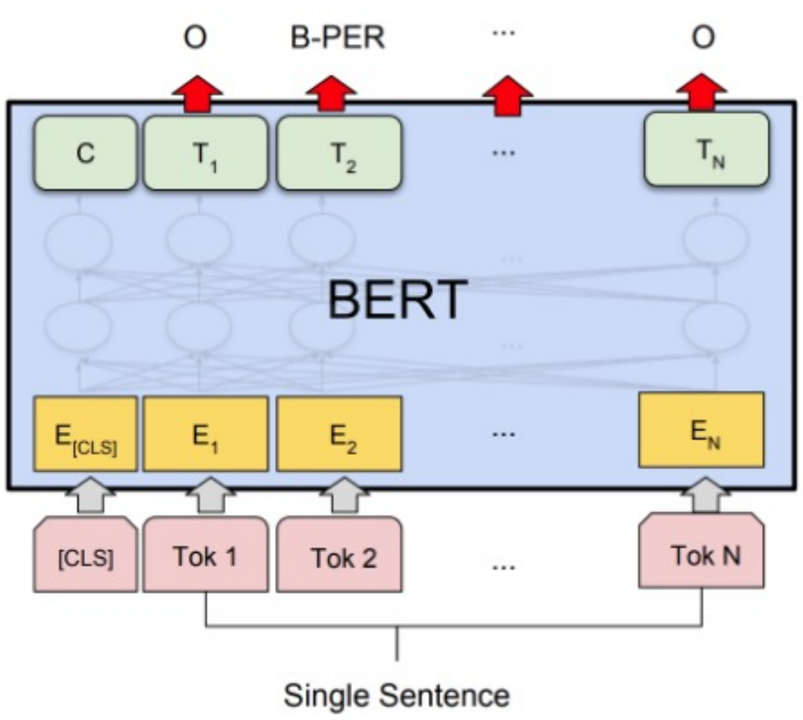
Next sentence

- **BERT objective**: masked LM + next sentence

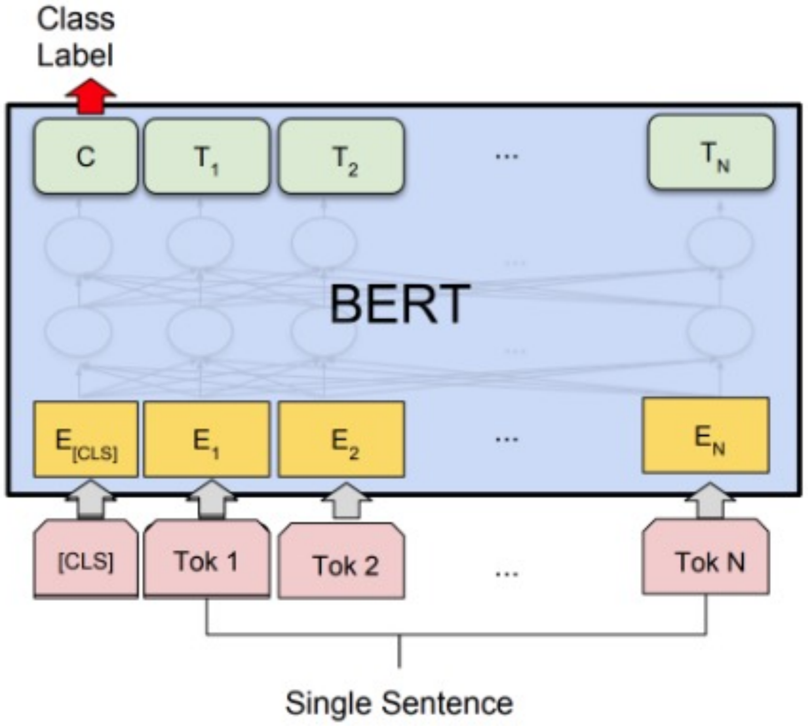


BERT in practice

Very flexible, can be used for NLI, classification, tagging, etc.

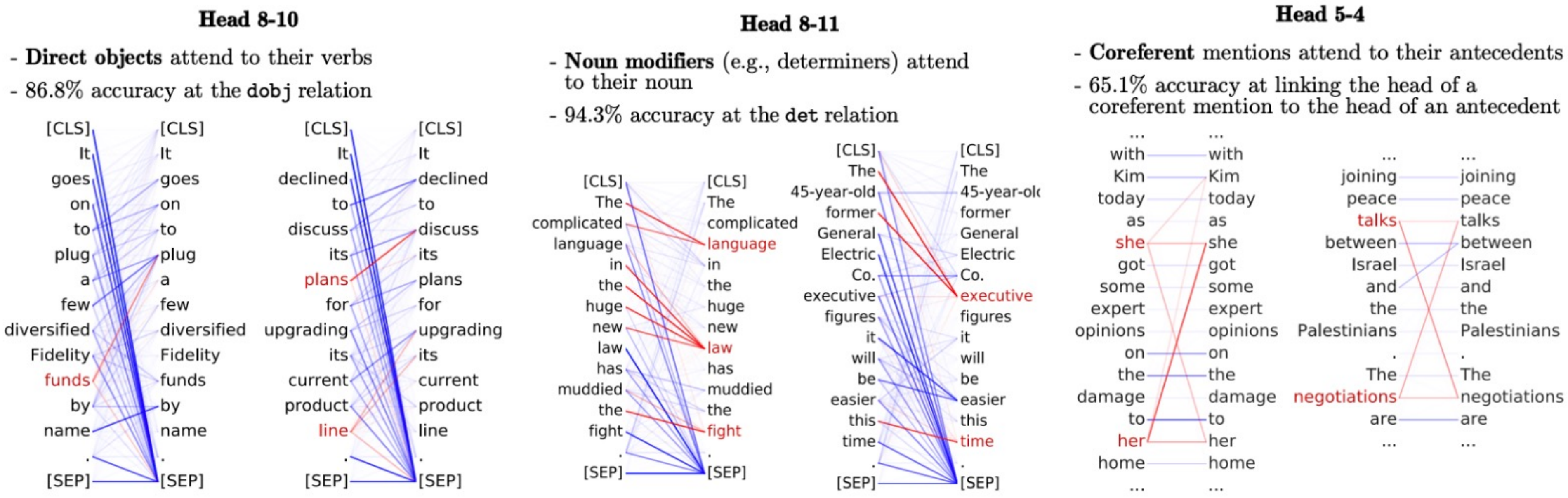


(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER



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

What does BERT learn?



- Still way worse than what supervised systems can do, but interesting that this is learned organically

Discussion

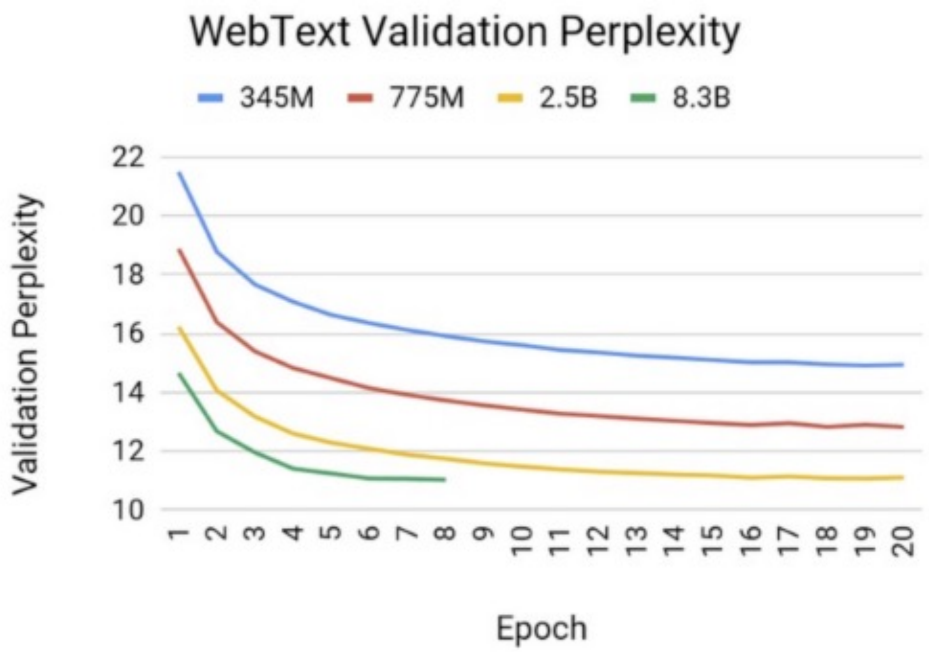
- **Current NLP trend**: *train a **very complex** neural language model using **massive** amounts of data*
- The learned representation should capture ”language understanding capability”
  - Word meaning
  - Linguistic structure
  - World knowledge
  - Bad stuff expressed through language.

Language Modeling

- *Impossible, **but**.. Bigger models do better and better*
  - Limit not yet reached

    - NVIDIA: trained 8.3B parameter GPT model (5.6x the size of GPT-2)

    - Arguable these models are still underfit: larger models still get better held-out perplexities



NVIDIA blog (Narasimhan, August 2019)

Open Questions

- Language modeling as a pre-training, and transformer-based architectures are now SoTA for all major NLP task.
  - GPT-3 uses 175 Billion parameters.

  That's a lot!

  Is this a sustainable direction? Currently all models developed in industry

  Building these models in academia would require a collective effort, might require changing how research is done.

Open Questions

- **What is really learned by a language model and how should we think about them?**
  - It’s clear that our original definition was too simplistic (capture word-patterns for tasks like context sensitive spelling correction)
  - Do we still need symbolic knowledge graphs? Can everything ”just” be learned from raw text using a language model?

Evaluating Learned Representations

Q: What is human life expectancy in the United States?
A: Human life expectancy in the United States is 78 years.

Q: Who was president of the United States in 1955?
A: Dwight D. Eisenhower was president of the United States in 1955.

Q: What party did he belong to?
A: He belonged to the Republican Party.

GPT 3 Generated Answers

Q: Which is heavier, a mouse or an elephant?
A: An elephant is heavier than a mouse.

Q: Which is heavier, a mouse or a toaster?
A: A toaster is heavier than a mouse.

Q: Which is heavier, a toaster or a pencil?
A: A pencil is heavier than a toaster.

https://lacker.io/ai/2020/07/06/giving-gpt-3-a-turing-test.html

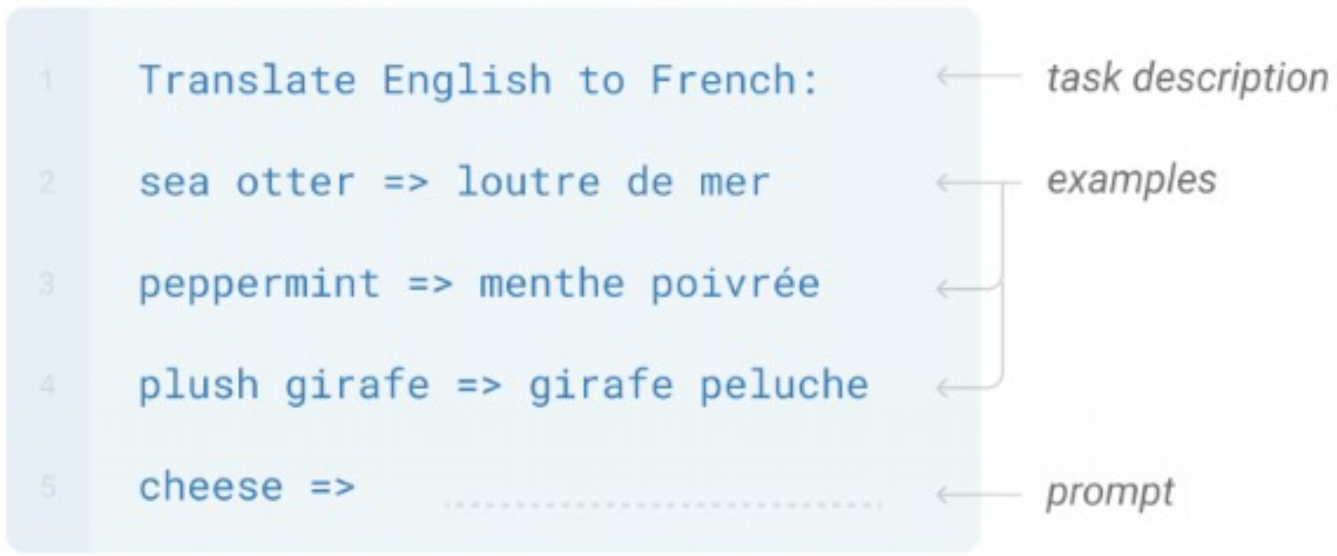
Evaluating Machine Comprehension

- The models we discussed are very flexible.
  - Highly expressive models, trained over massive amounts of text.
  - **What do they actually learn?**
  - Memorize facts, capture common-sense judgements, world knowledge, etc.
- Many tasks can be formulated as LM
  - Question answering, discourse relation, etc.

GPT3: new directions

Few-shot

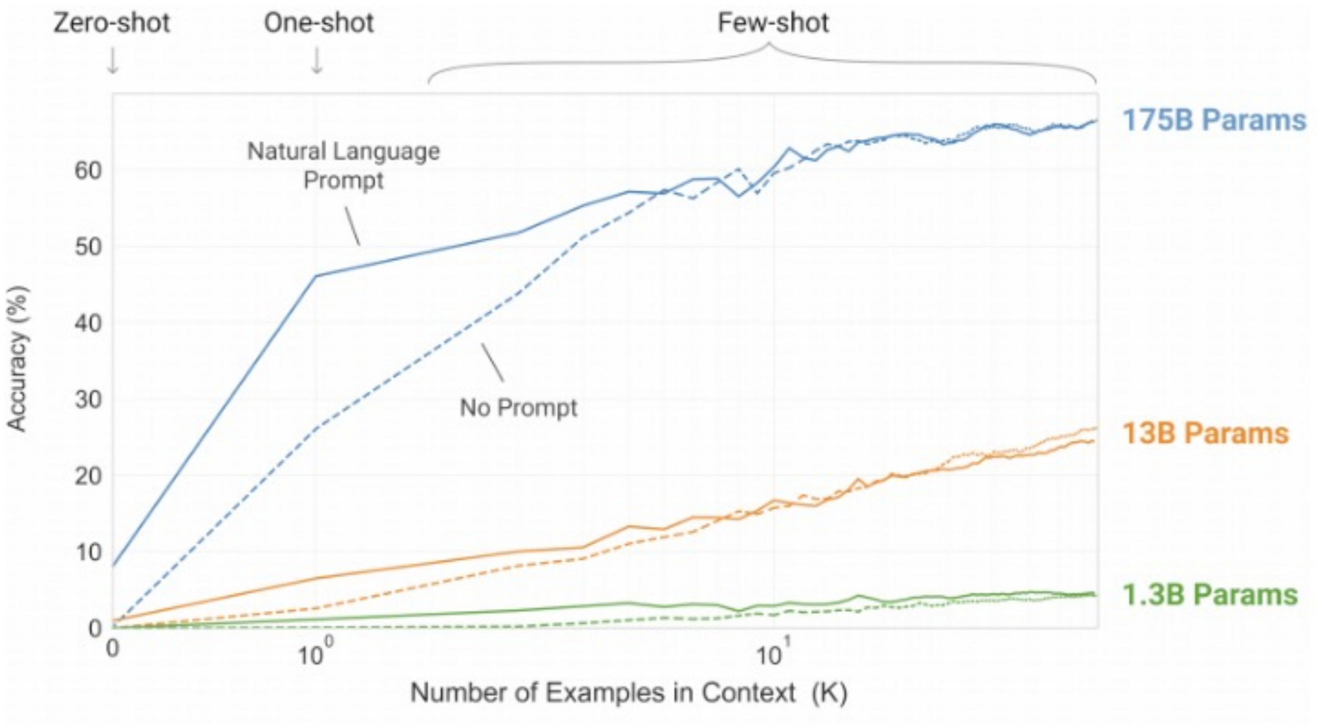
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Brown et al. (2020)

GPT3: new directions

- **Key observation:** few-shot learning only works with the very largest models!



Brown et al. (2020)