LING 334 - Introduction to Computational Linguistics

Week 9

Neural Nets, RNNs, and Transformers

Coming Up

2/22 - More NNs, RNNs, and Transformers

2/27 - Research in NLP, final project brainstorming

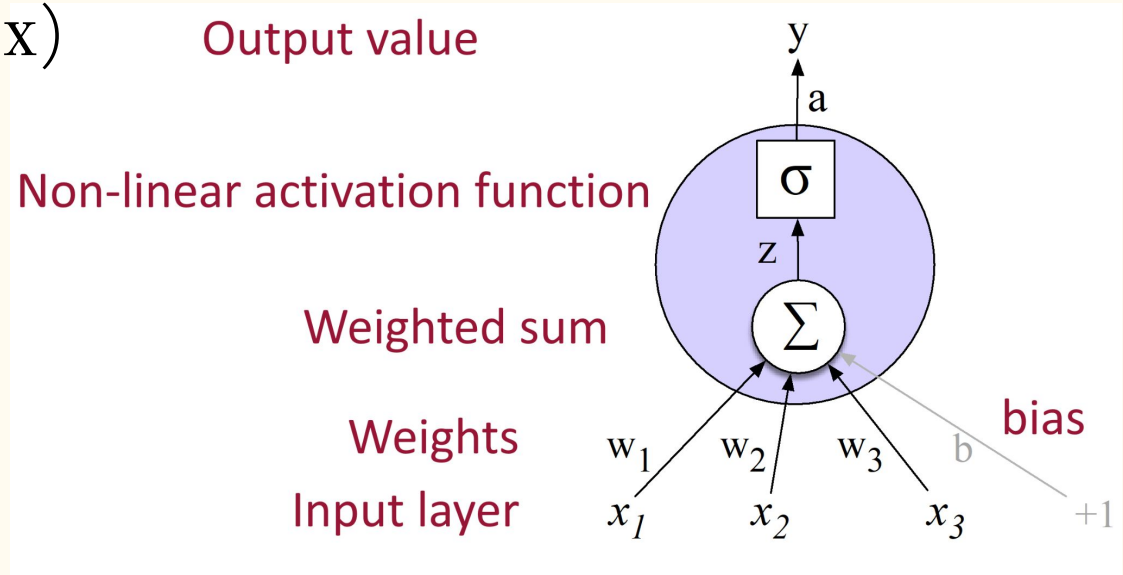
2/29 - Topic Models

3/5 - Extra TA OH (here) for final projects/assignments

3/7 - State of the art and contemporary issues

Reminder - One Neuron ($\approx$ Logistic Regression)

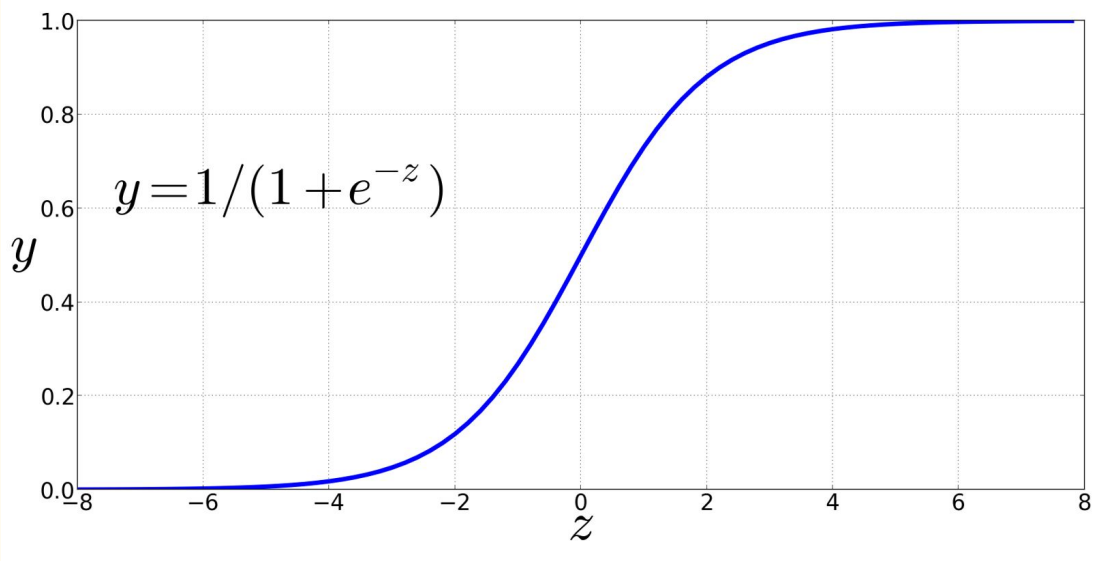
Biologically inspired
(but way less complex)



this and future figures from SLP Ch. 7, 9, 10, and 11 unless noted

Non-Linearities - Sigmoid

Transforms any value to be between 0 and 1, pseudo-probability



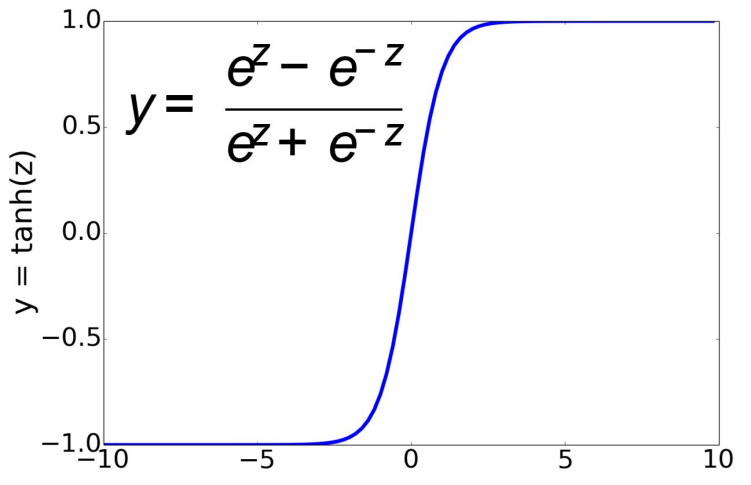
x axis = sum of weights times inputs

y axis = output value of neuron

Non-Linearities - tanh and ReLU

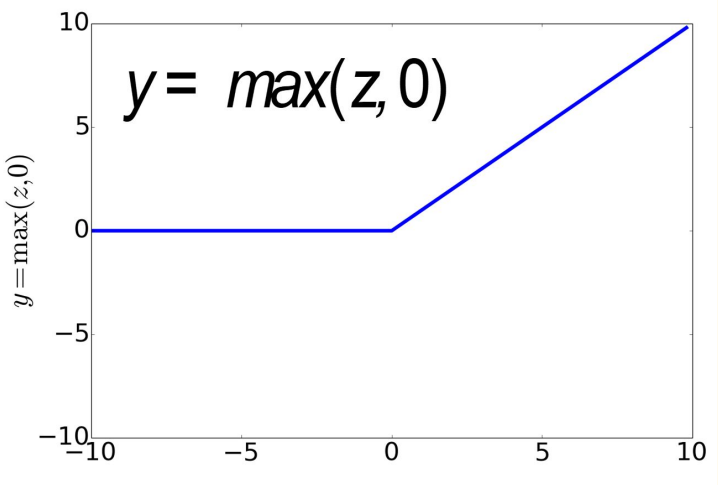
tanh

(like sigmoid, works better)



ReLU

(most common)



credit J+M, SLP slides

Why Non-Linearities?

Naive Bayes is a linear classifier

Decision boundary from $\sum w \cdot x$

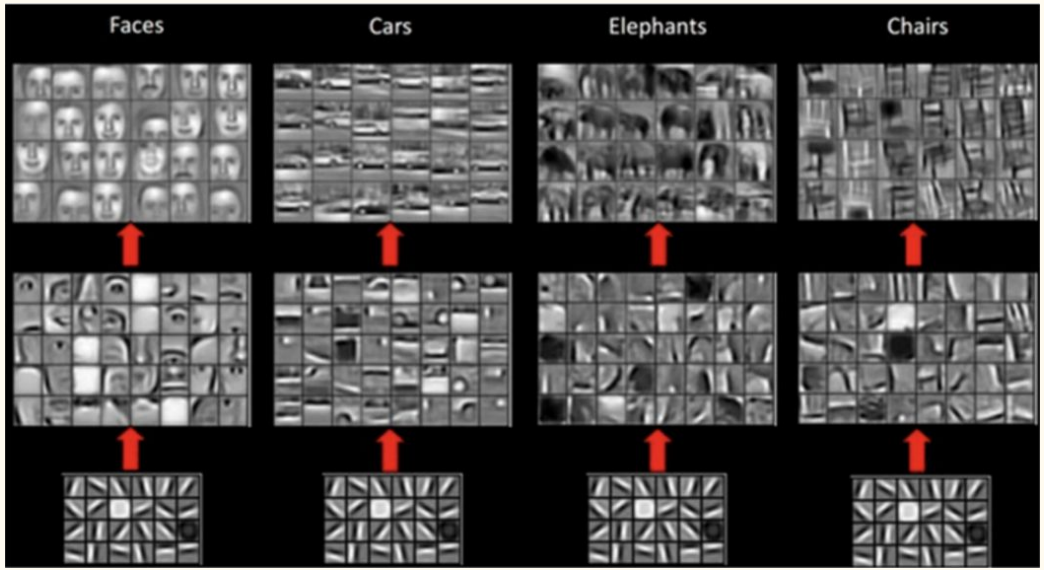
For NNs, key idea is representing the input in increasingly abstract non-linear transformations

"Hidden Layers"

Until the final decision can be made linearly

Example from Computer Vision

Each layer in a convolutional neural network is activated by increasingly abstract stimuli

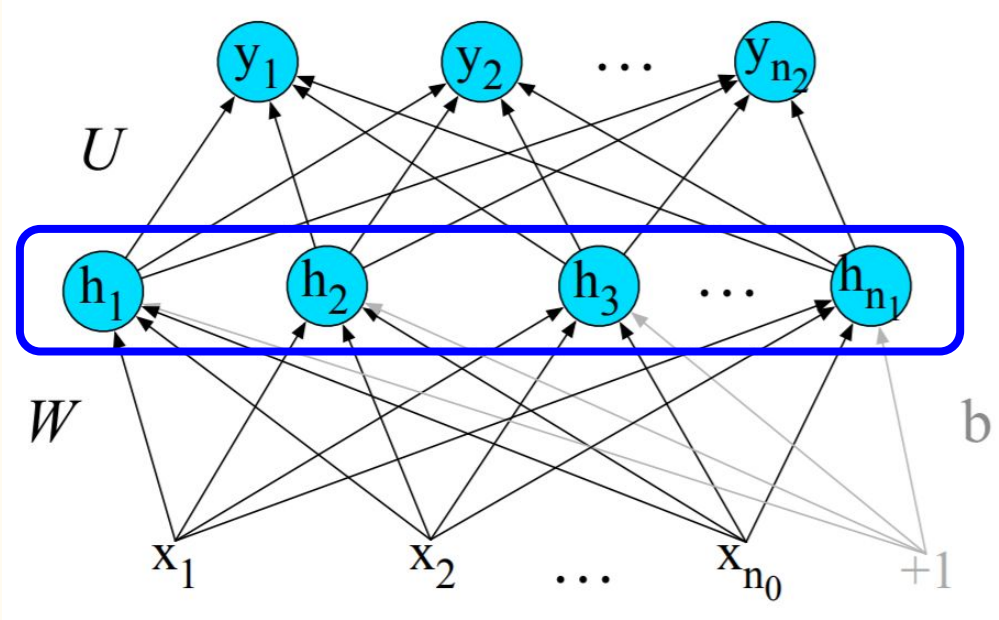


Simple Feed-forward Neural Net

Each arrow represents multiplication of value by a weight

Summed at each node, non-linear transform

Like multiple logistic regressions running concurrently on the same inputs



Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

State of the art systems use more powerful neural architectures, but simple models are useful to consider!

Classification: Sentiment Analysis

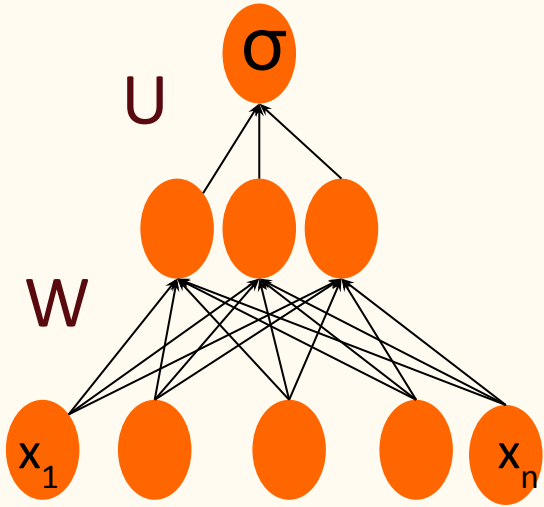
We could do exactly what we did with Naive Bayes or Perceptron

Input layer is binary features as before

e.g. bag of words

Output layer is 0 or 1

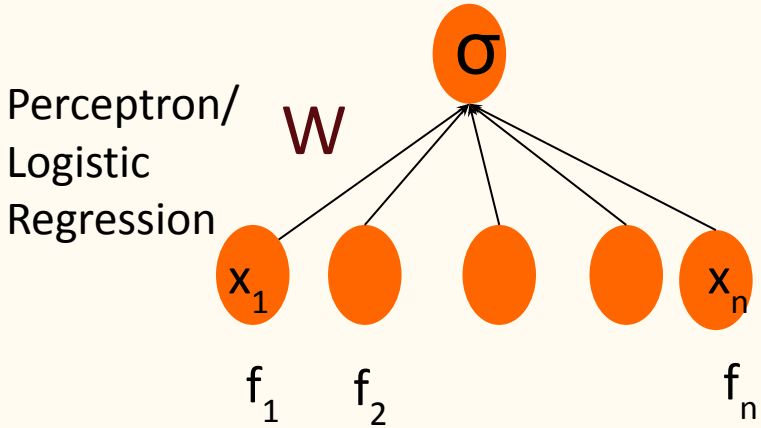
Learn weights from labeled examples

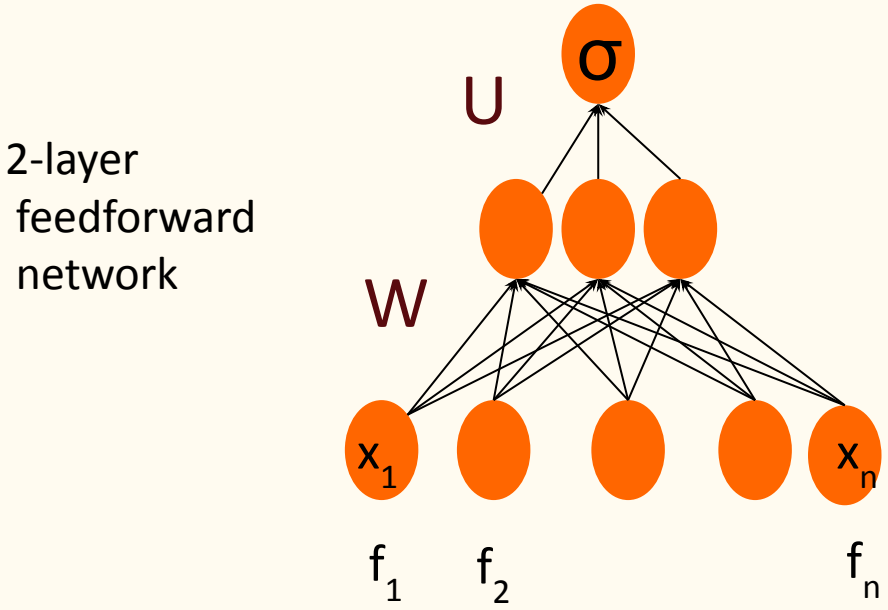


Sentiment Features

Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if ``no'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Feedforward nets for simple classification





Just adding a hidden layer to logistic regression

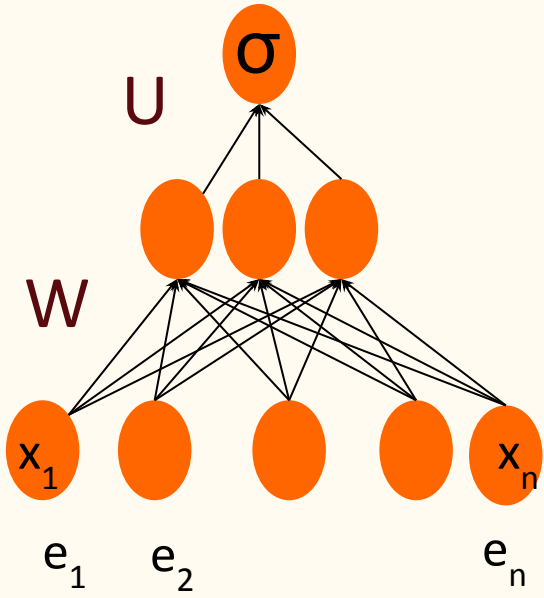
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

Even better: representation learning

The real power of deep learning comes from the ability to **learn** features from the data

Instead of using hand-built human-engineered features for classification

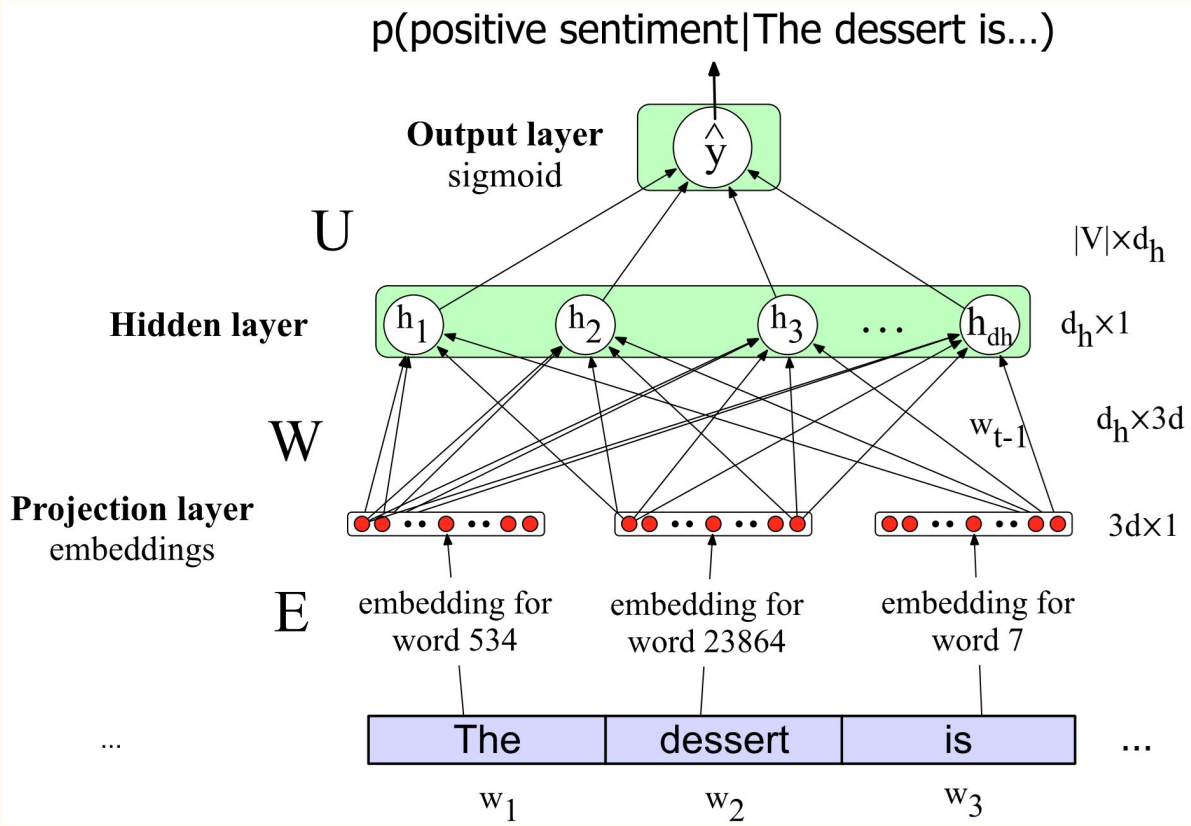
Use learned representations like embeddings!



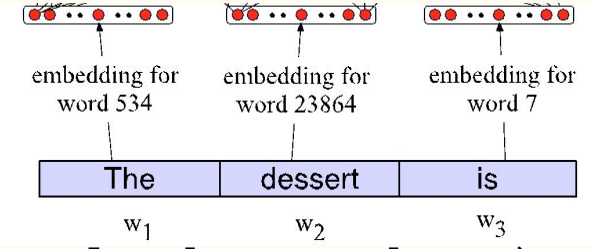
Simple NN - Another View

Large input layer!

Many weights!



Issue: texts come in different sizes



This assumes a fixed size length (3)

Kind of unrealistic.

Some simple solutions (more sophisticated solutions later)

1. Make the input the length of the longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
   - Take the element-wise max of all the word embeddings
     - For each dimension, pick the max value from all words

Neural Language Models (LMs)

Language Modeling: Calculating the probability of the next word in a sequence given some history.

- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models

State-of-the-art neural LMs are based on more powerful neural network technology like Transformers

But **simple feedforward LMs** can do almost as well!

Simple feedforward Neural Language Models

Task: predict next word w_t

given prior words w_{t-1}, w_{t-2}, w_{t-3}, ...

Problem: Now we're dealing with sequences of arbitrary length.

Solution: Sliding windows (of fixed length)

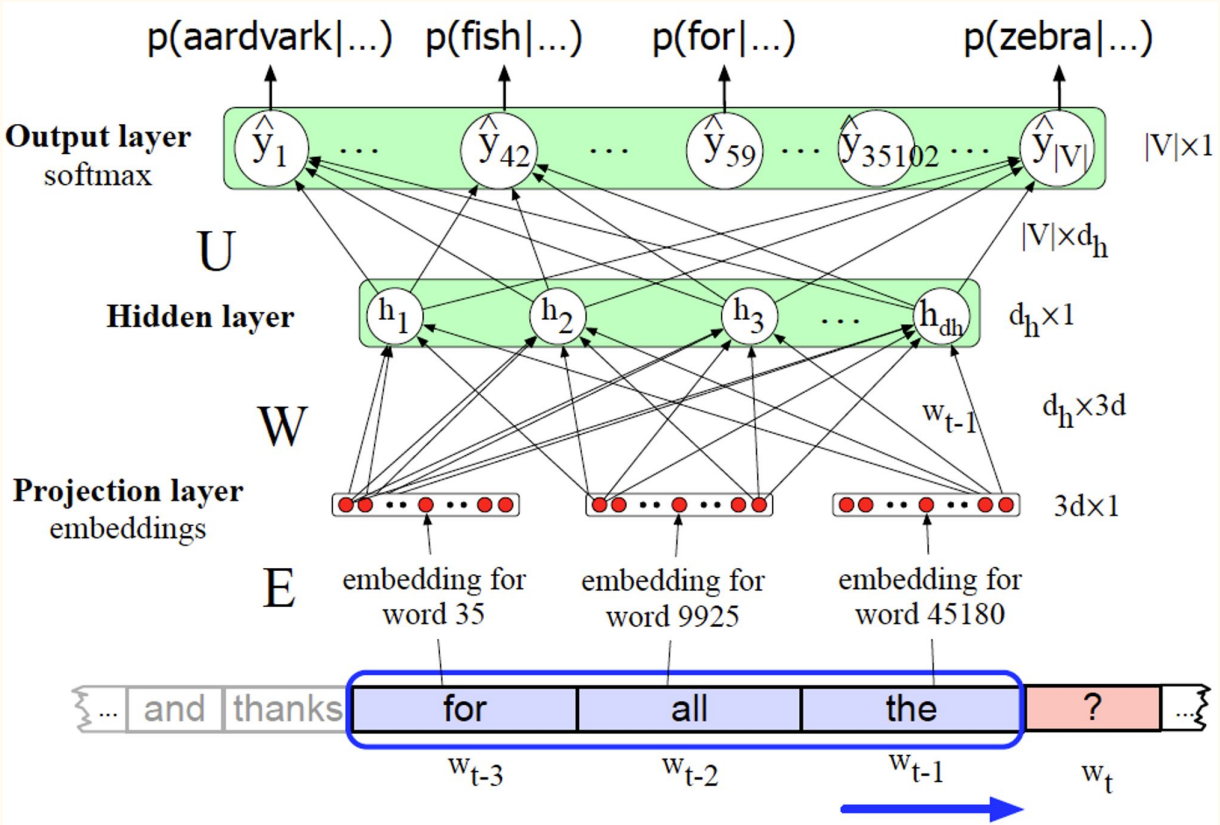
$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Network Language Model

Sliding window over words

Large output layer of all words in V

Notice the hidden layer is itself a vector!



Why Neural LMs work better than N-gram LMs

Training data:

We've seen: I have to make sure that the cat gets fed.

Never seen: dog gets fed

Test data:

I forgot to make sure that the dog gets ___

N-gram LM can't predict "fed"!

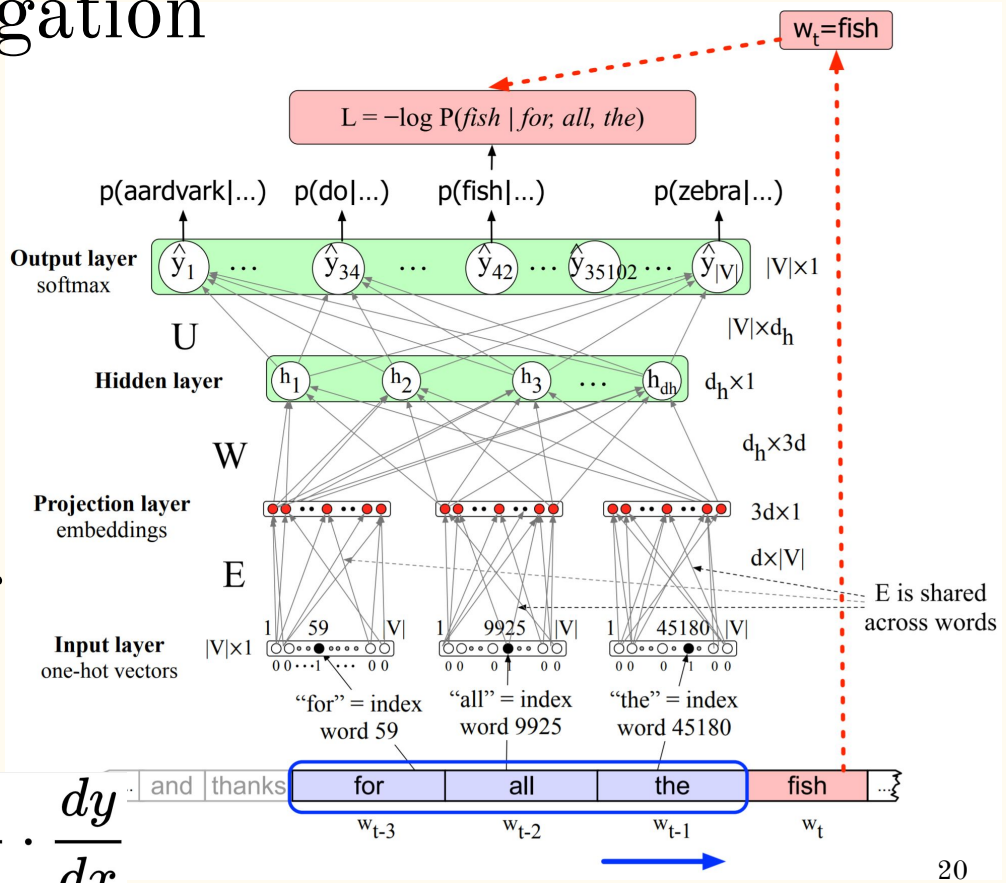
Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict “fed” after dog

Training via Backpropagation

Loss = function saying, how wrong are we?

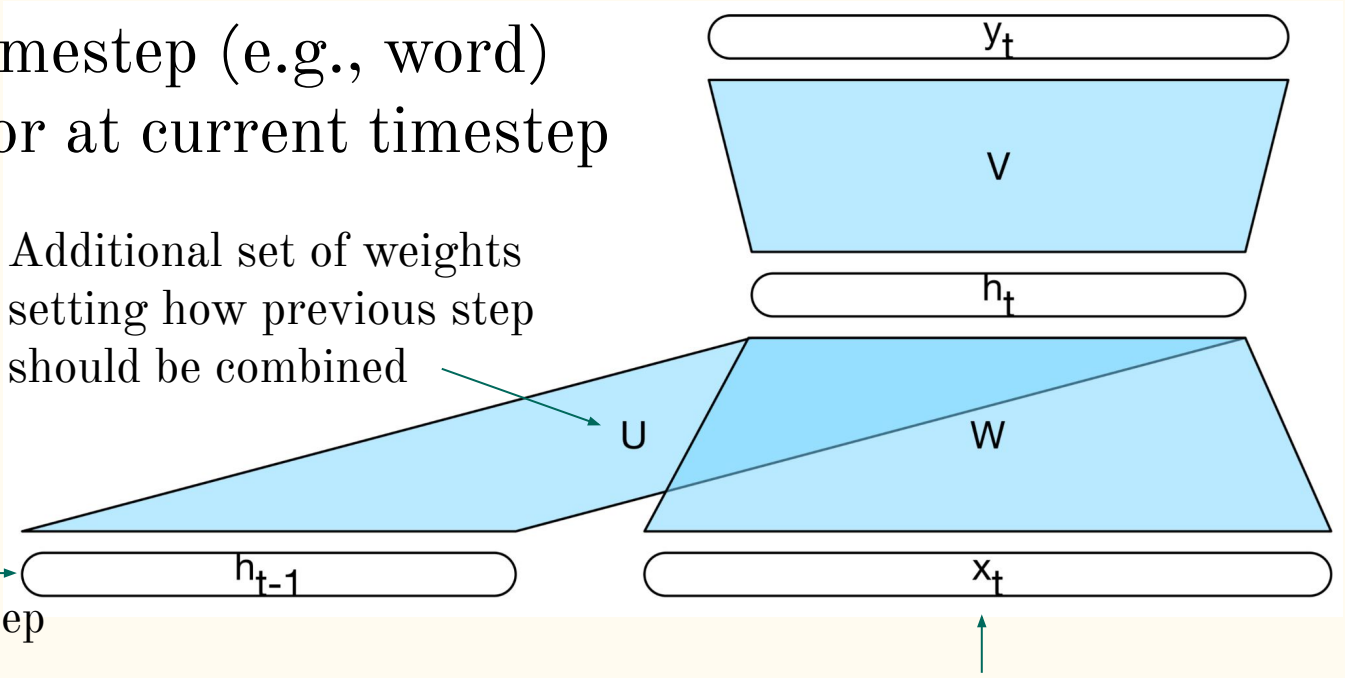
Derivative of this function at any point tells us which way to go to be less wrong

Chain rule allows us to go back arbitrarily far $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$



Recurrent Neural Networks

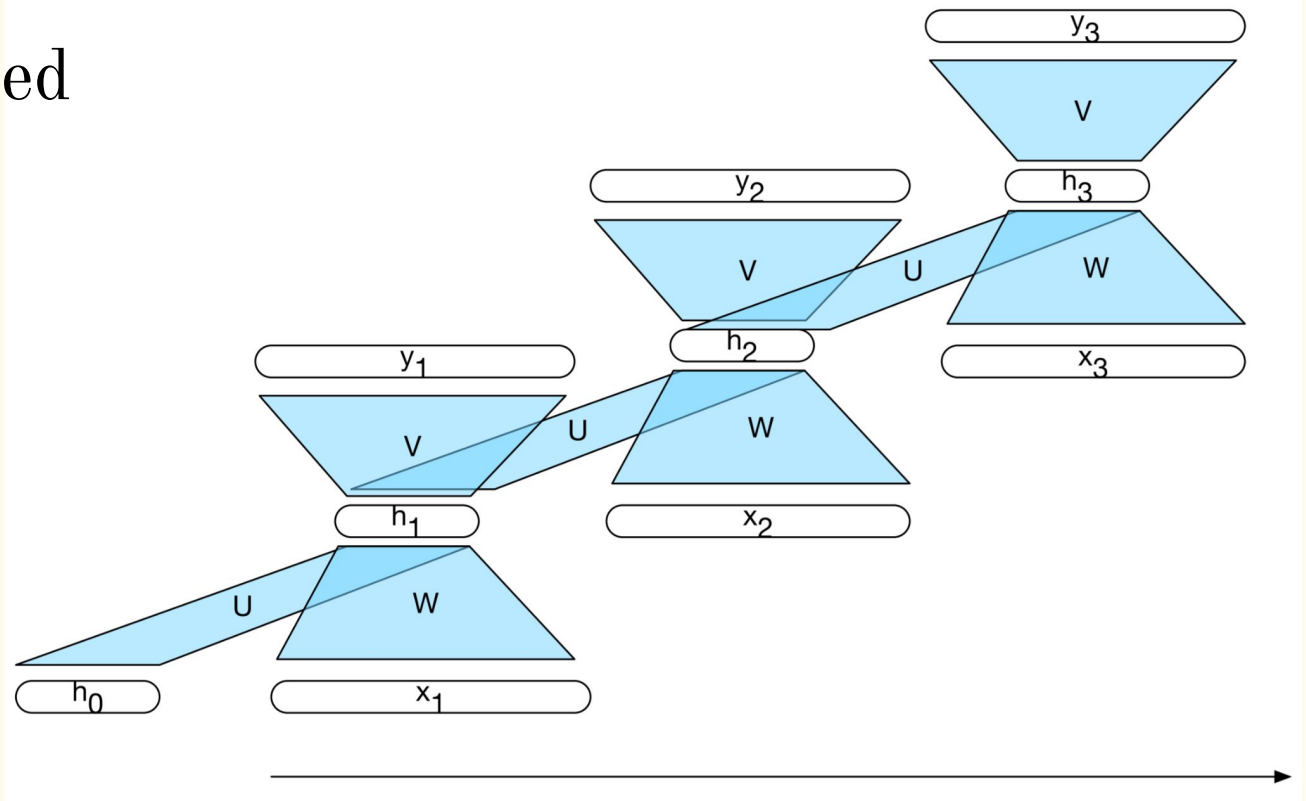
Core idea: combine hidden state vector from previous timestep (e.g., word) With input vector at current timestep



Recurrent Neural Networks - unrolled view

Weights are shared across timesteps

E.g., the same U, V, W are applied at each timestep

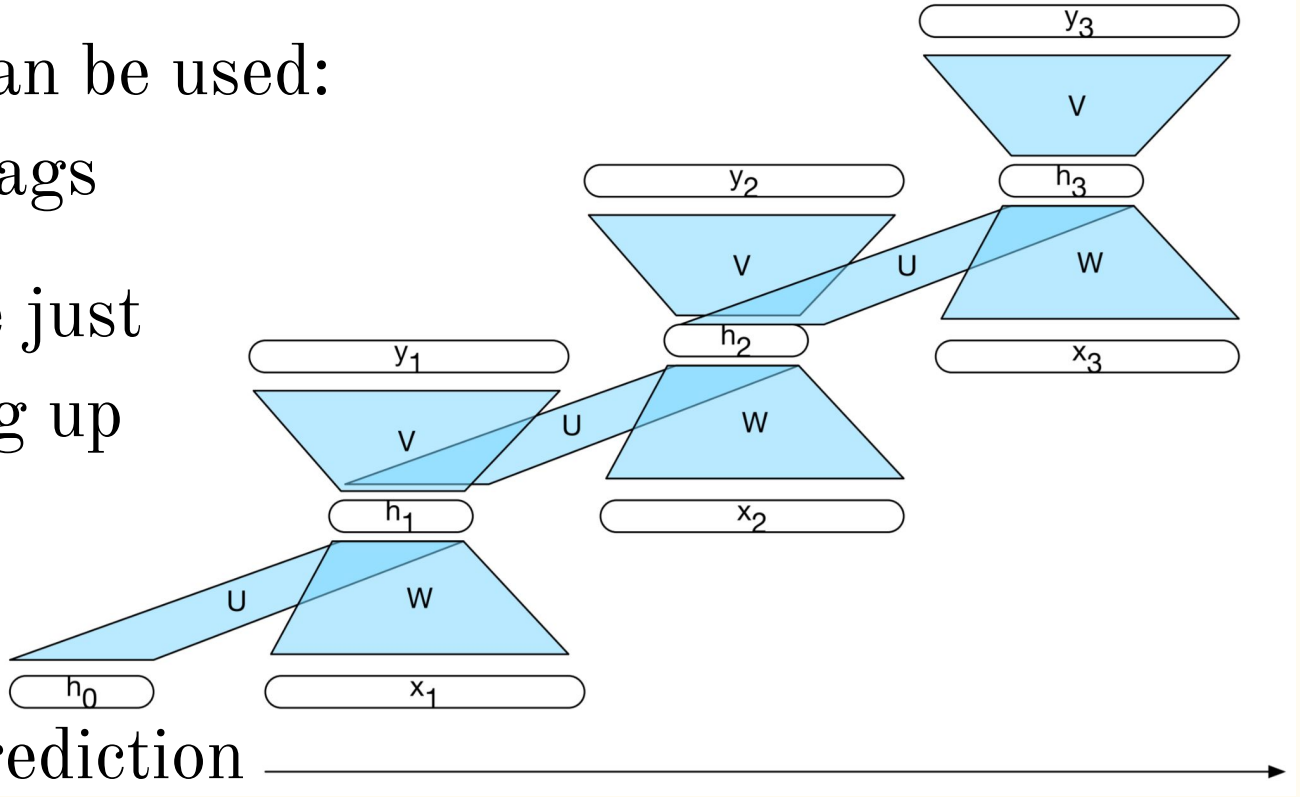


Recurrent Neural Networks - unrolled view

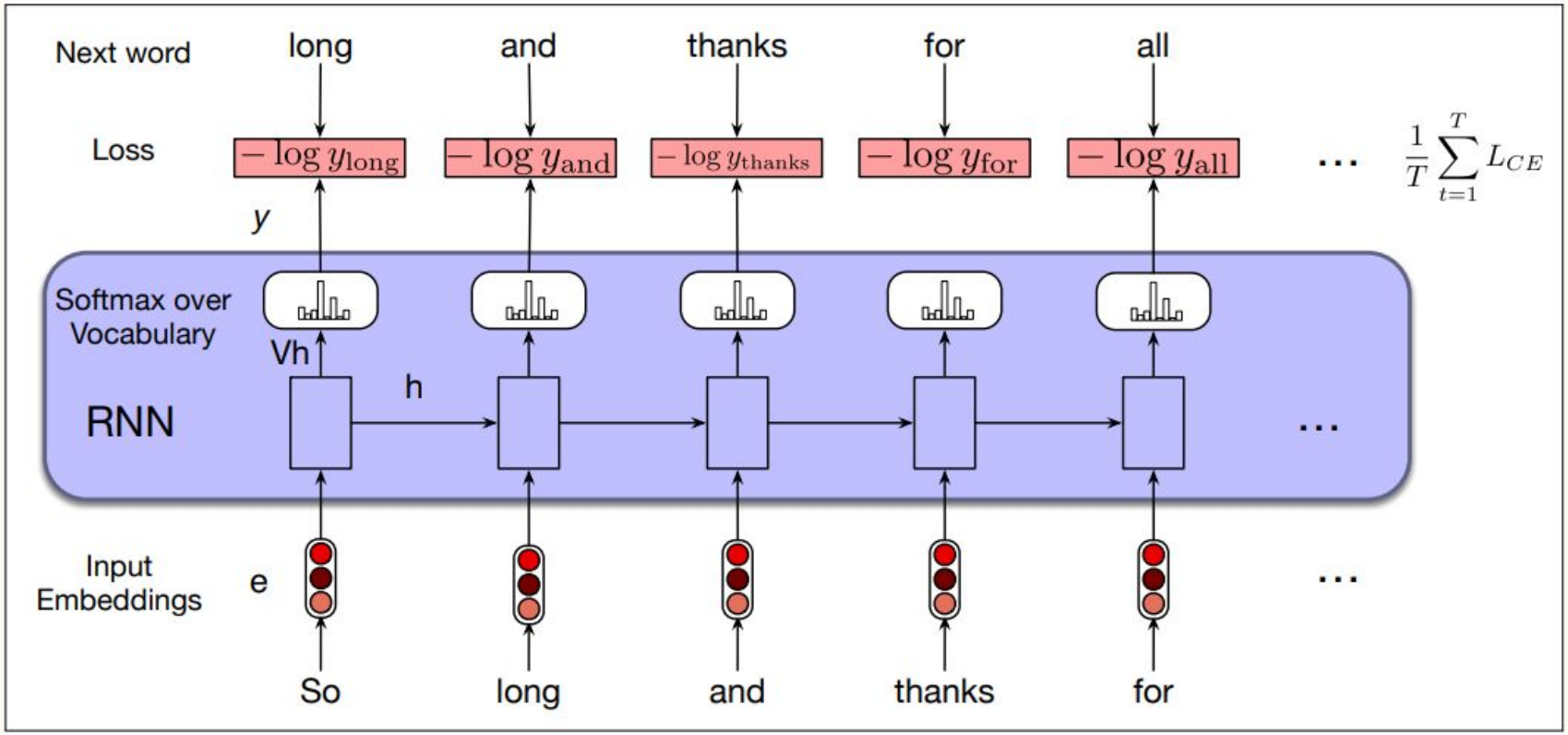
Output layer (y) can be used:
e.g. predict POS tags

or discarded, if we just care about building up the hidden state

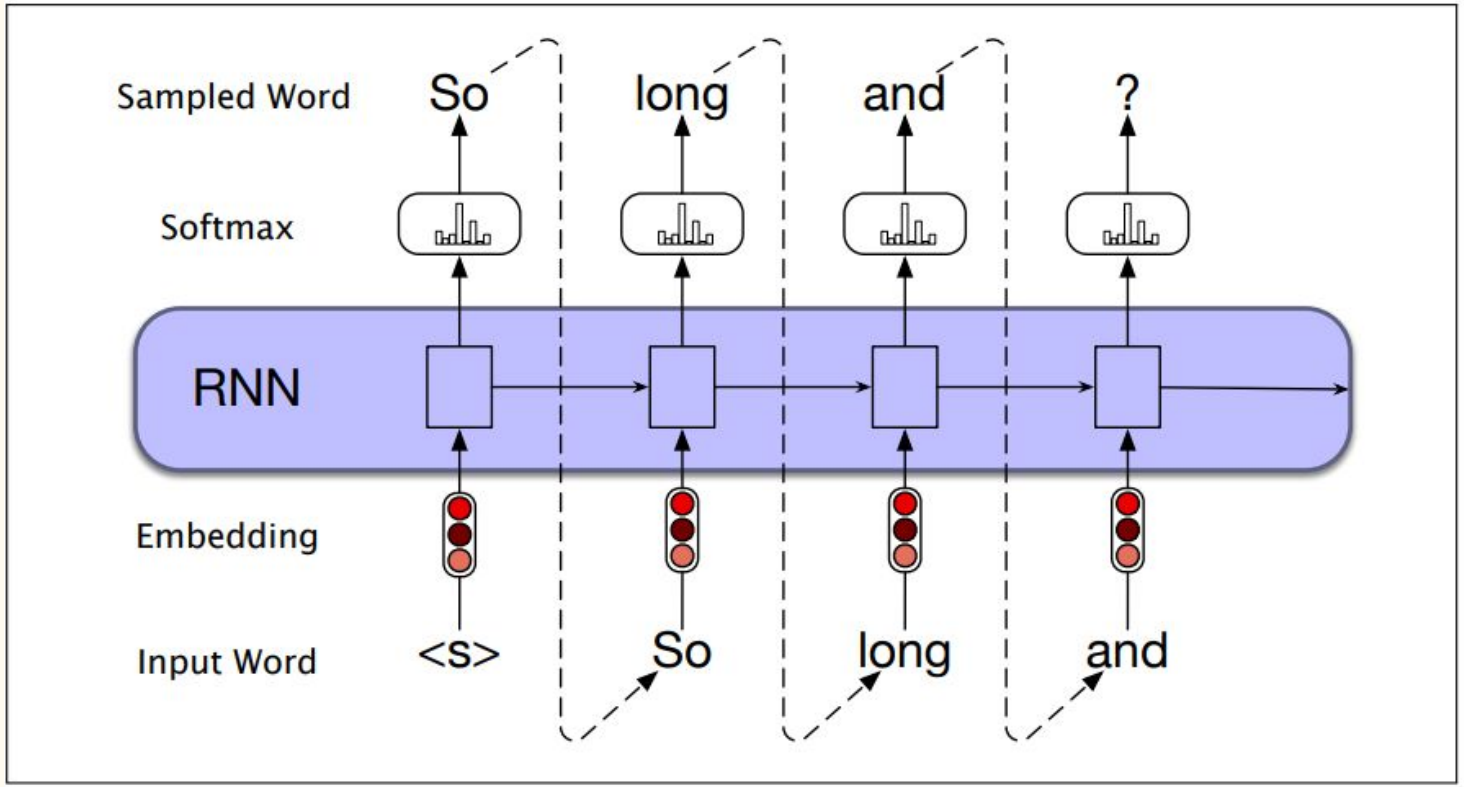
Can just use final output layer for prediction



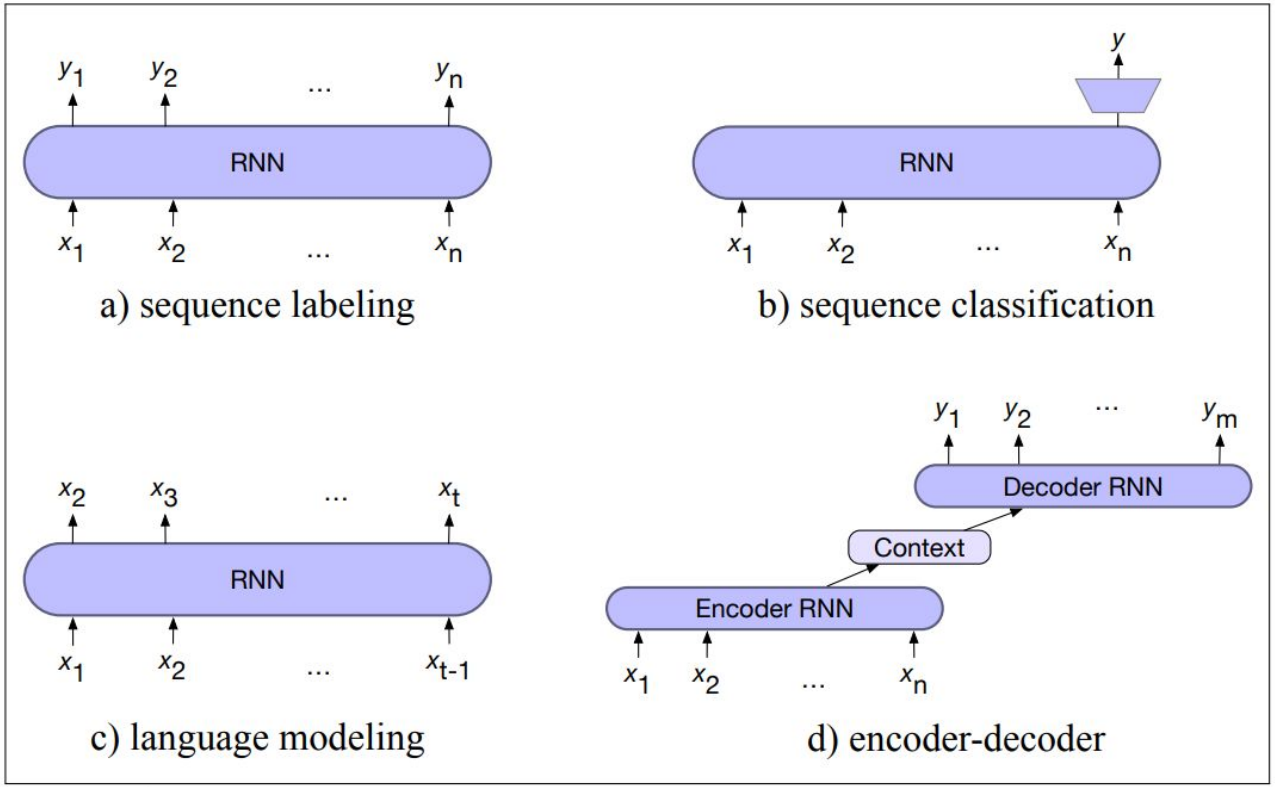
RNNs as a language model



RNNs as a language model - generation



Recurrent Neural Networks - a flexible mechanism



Sequence-to-Sequence Models

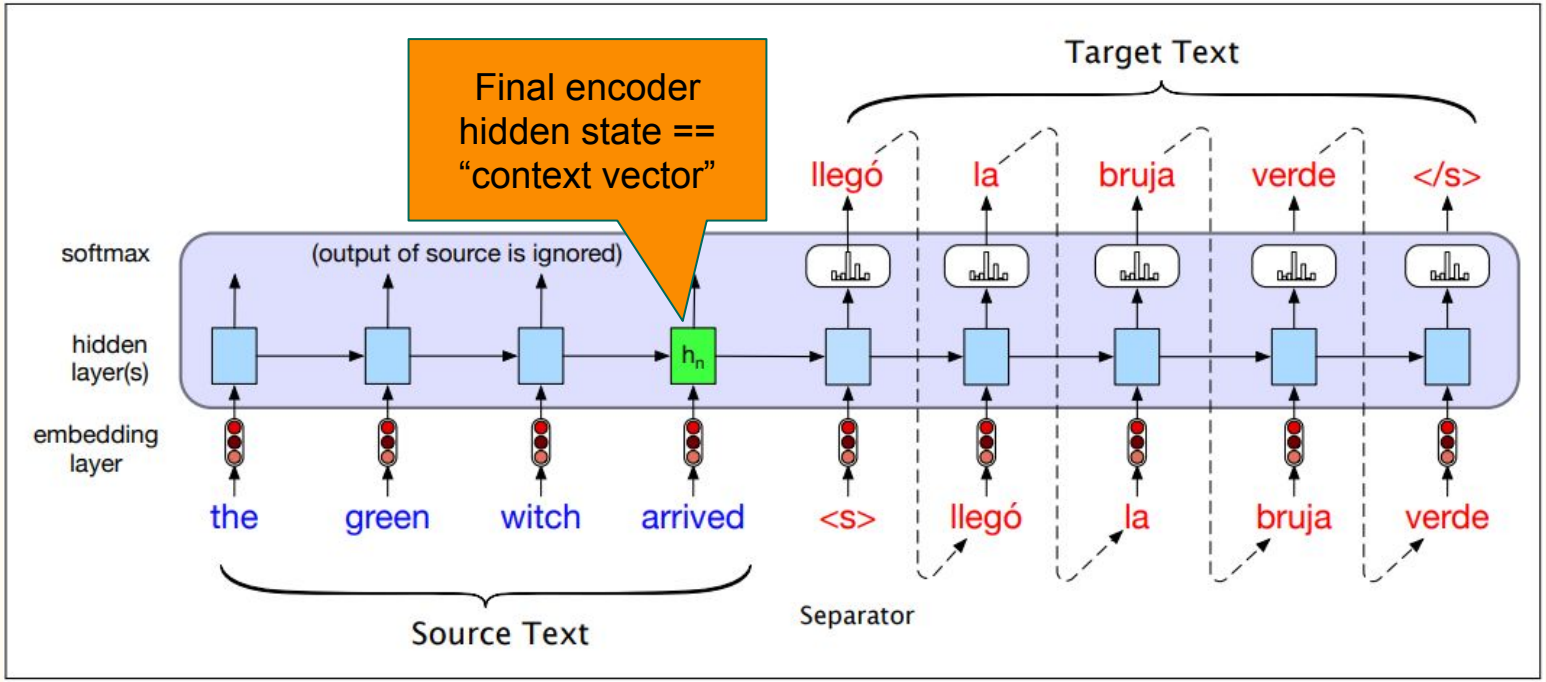
Encode a sequence word by word, building the hidden state

Pass final hidden state to another RNN to “decode”

Common use case: machine translation

Seq2seq Models - another view

Remember encoder and decoder are separate RNNs

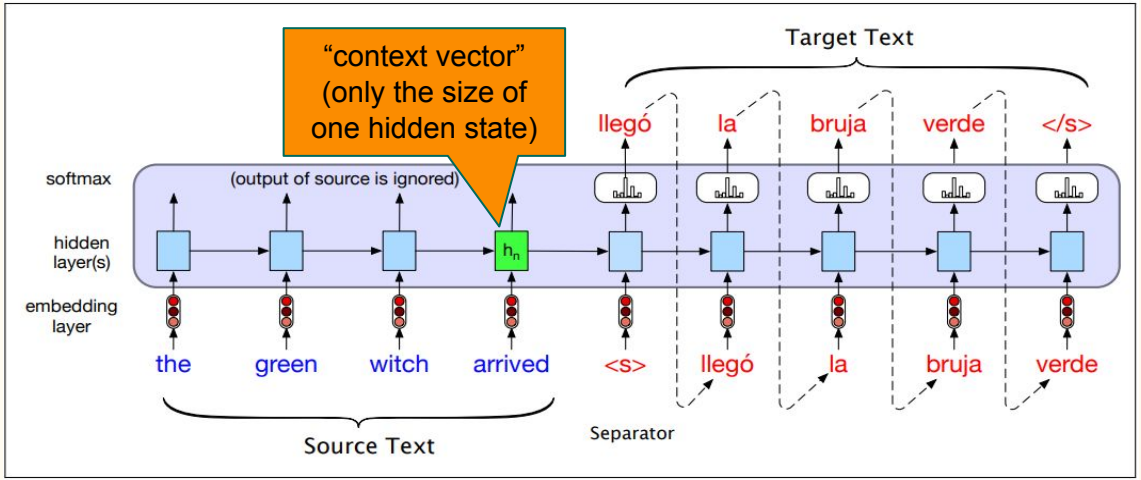


Bottleneck Problem

This final hidden state is a *bottleneck* - this one vector is being asked to encode *everything* about the input

How can we let the model look back?

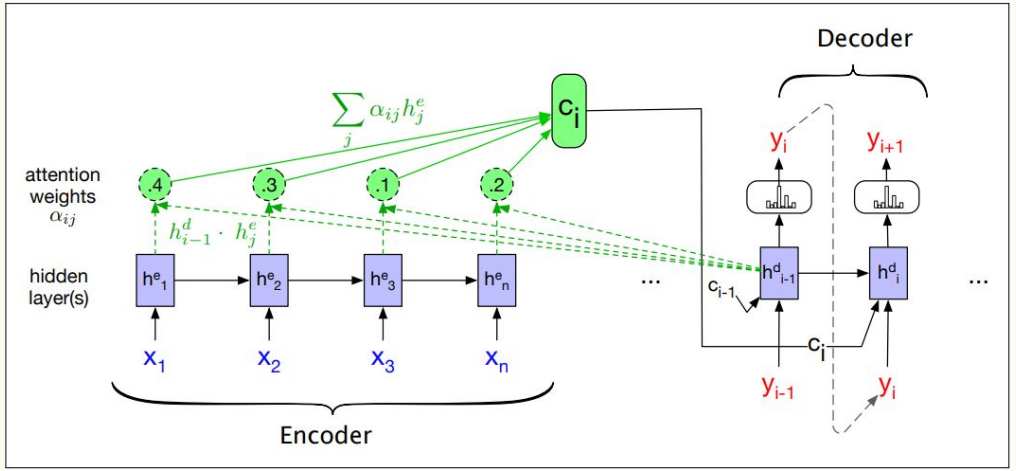
Answer:
Attention!



Attention

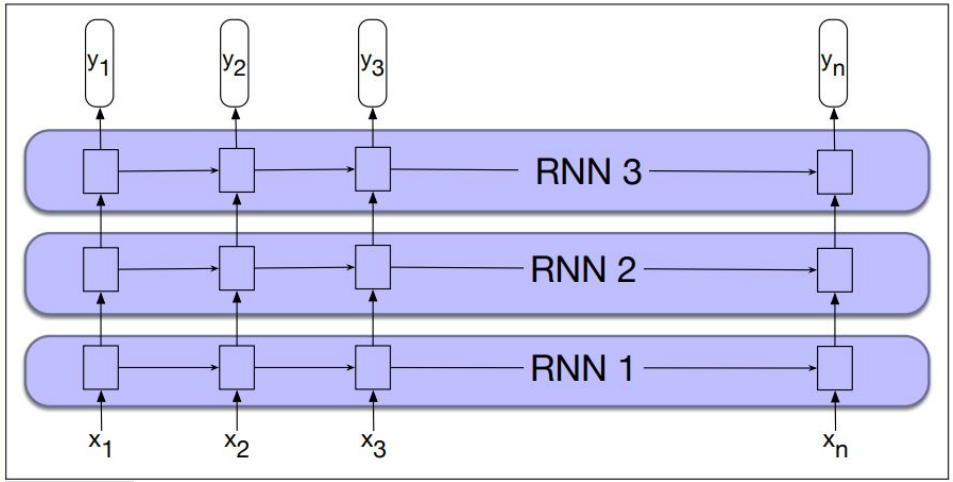
Incorporate an additional “context vector” at each step:

- Weighted sum of the encoder hidden states
- Simplest: dot product similarity of current decoder hidden state and each encoder state
- Many methods!



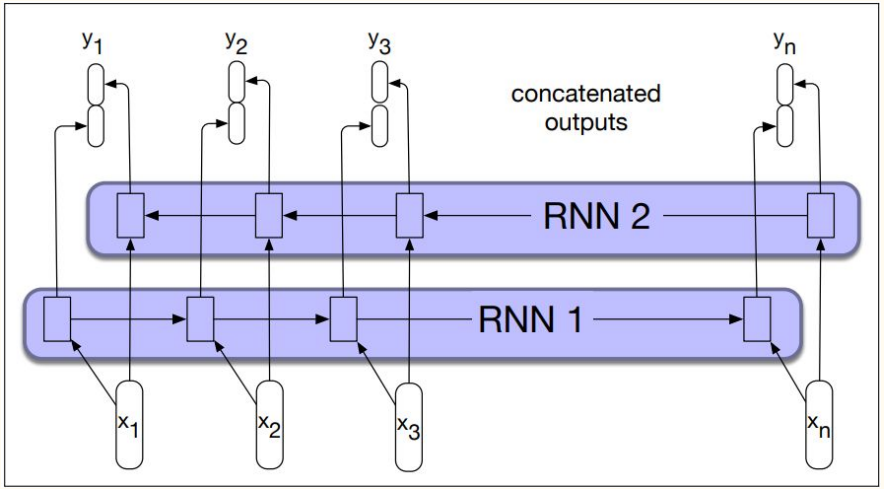
Other Key Concepts: Stacked and Bidirectional

Stacked RNN



Hidden state from previous layer becomes input for next layer, like feedforward

Bidirectional RNN



Two separate RNNs processing the input in opposite directions to “see” both sides of any particular token

Aside: Notes on Training

Architecture often about setting up a structure that "could work":

- Reasonable-seeming information flow
- Differentiable loss function that says how bad guesses are
- Training data to train it on

Calculus tells how to "wiggle the weights" to get it to work, learning from training data.

Often surprising it does! Classic article:
The Unreasonable Effectiveness of Recurrent Neural Networks

Aside: Notes on Tokenization

Contemporary NNs use subword tokenization

Like the Byte Pair Encoding algorithm introduced at the very beginning of the course, and variants

Contextual Embeddings

Problem with word2vec!

Embedding for “sound” is always the same, even in:

- “Does that sound good?”
- “I heard a loud sound.”
- “I’m going boating out on the sound.”
- “That’s sound logic right there!”

Doesn’t seem quite right.

ELMo (Embeddings from a Language Model)

Key insight: don't use static embeddings; instead, use hidden state from an RNN language model (Peters et al. 2018)

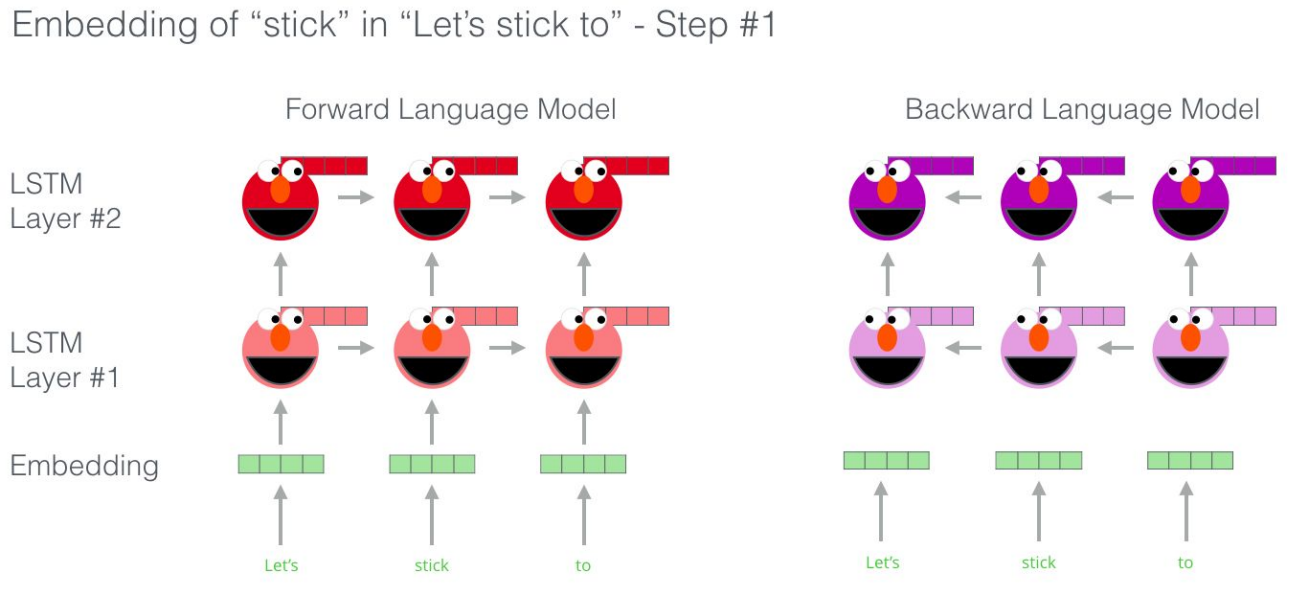


figure from Jay Alammar

ELMo (Embeddings from a Language Model)

Result is “contextual” embeddings

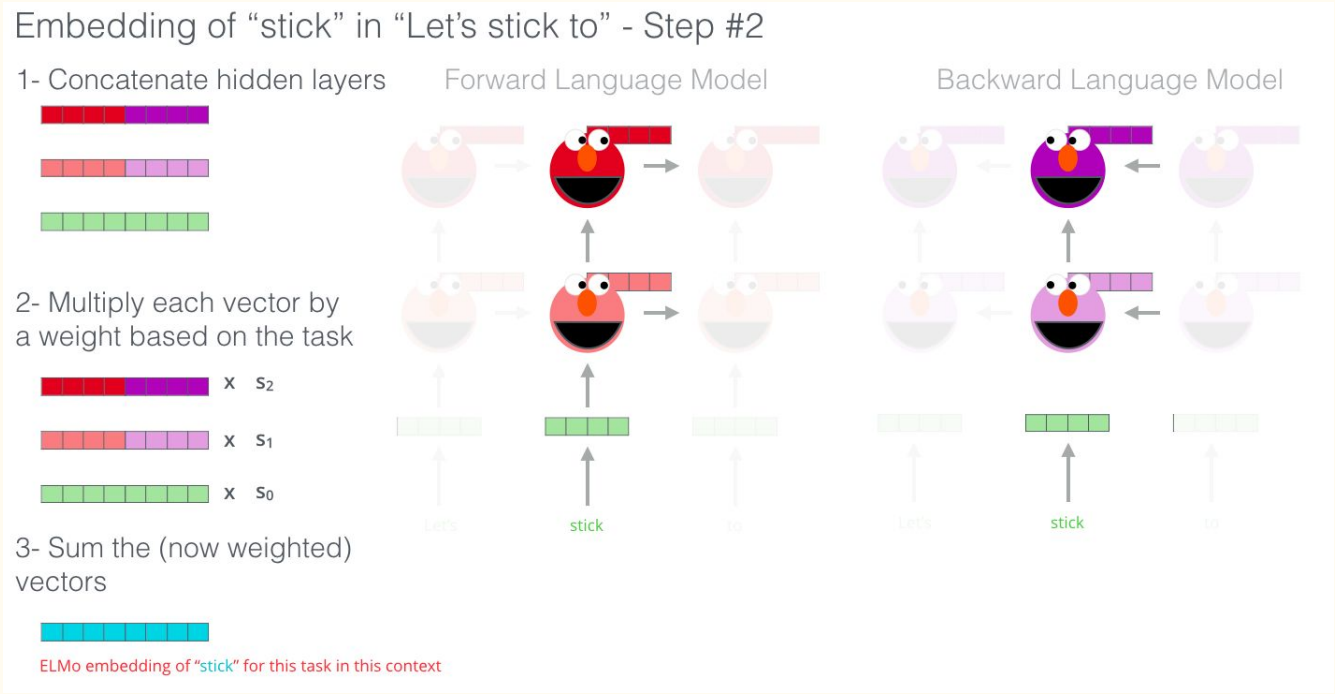
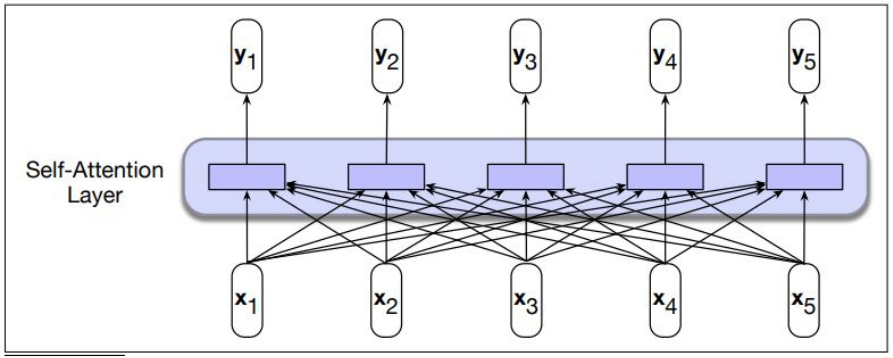


figure from Jay Alammar

The Muppet Parade

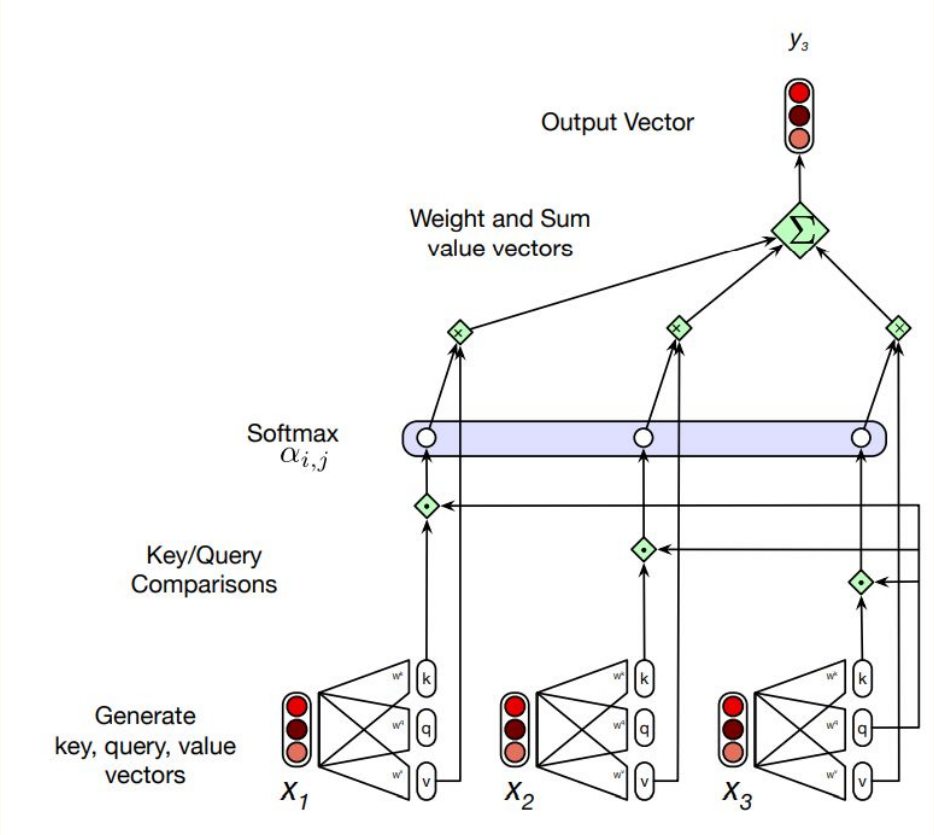
BERT and others follow on this idea with more complex architectures - key idea is self-attention

Many layers!
Details matter a lot!

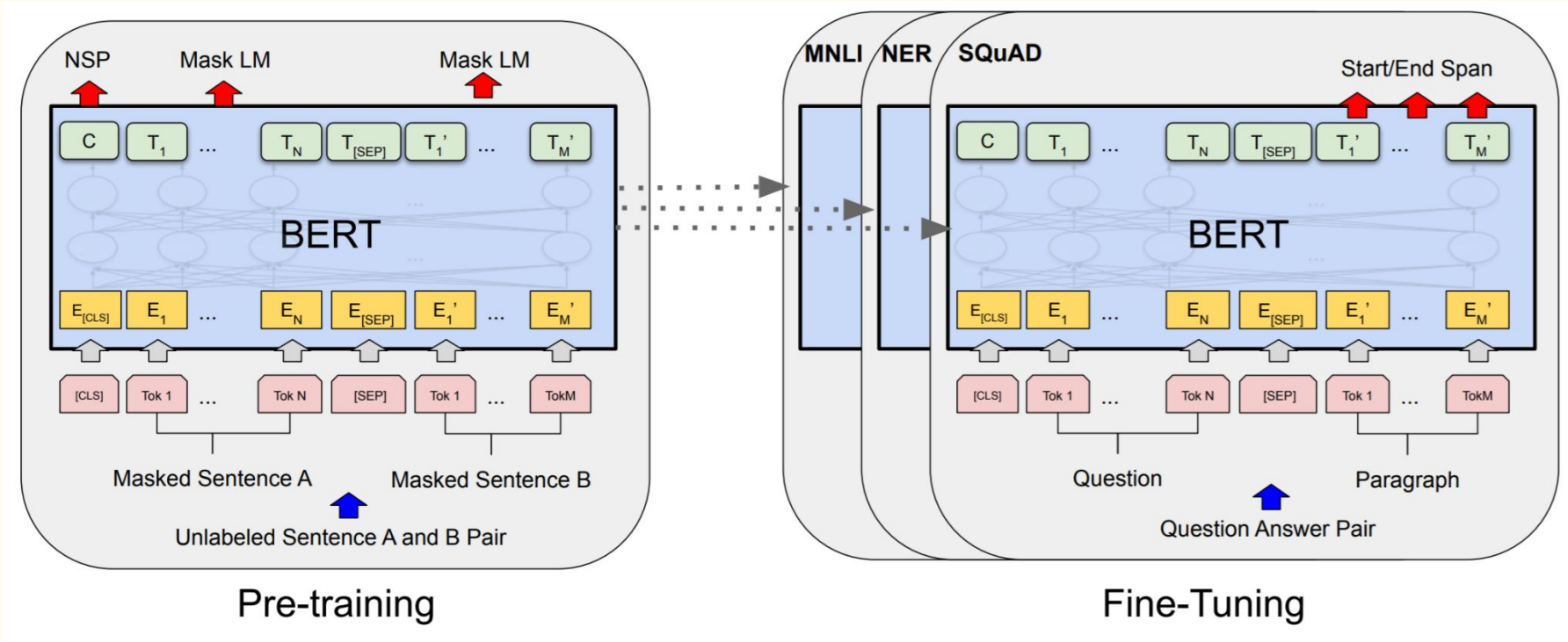


Transformers

Detailed view of self-attention



Pre-Training → Fine-Tuning Paradigm

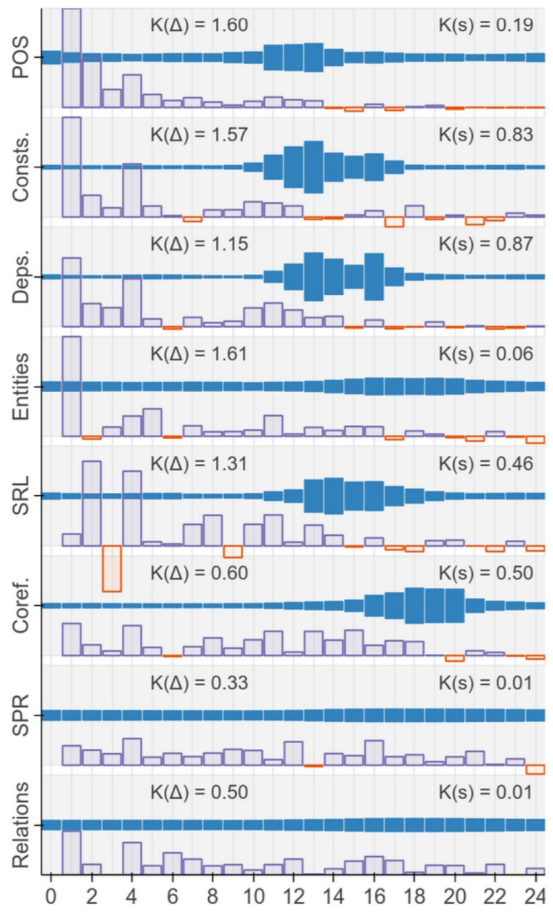


The many layers of BERT

BERT-Large has 24 transformer layers (each of which has a number of further internal layers itself)

Empirical work has shown that BERT encodes increasingly abstract linguistic information in higher layers

(Tenney et al. 2019)

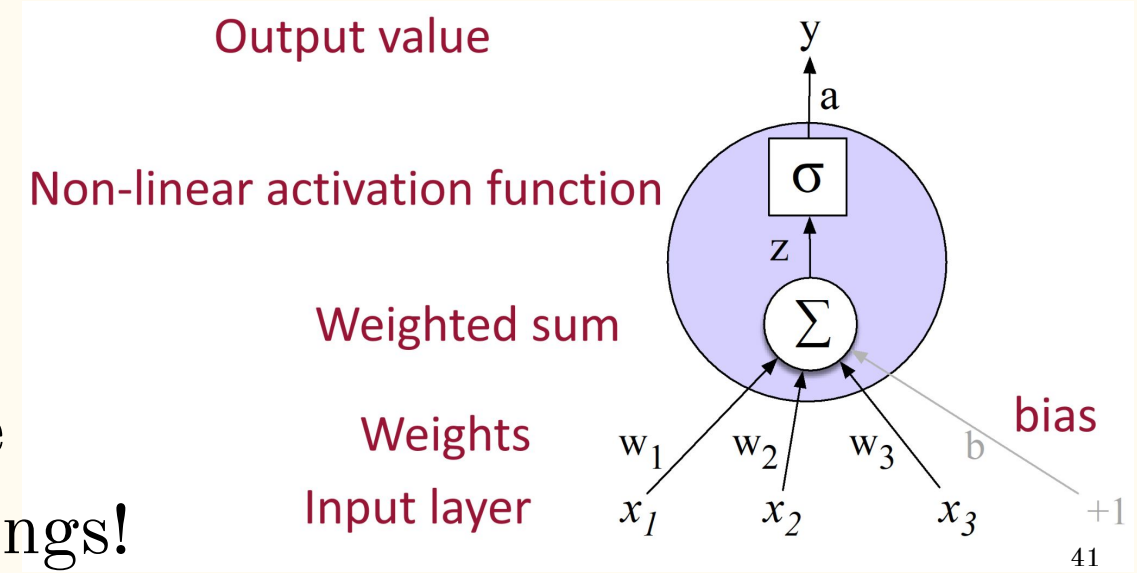


BERT et al. for Classification

BERT in particular provides a [CLS] token, contextual embedding token for classification

Frequently just start the cycle over again...

Train a new classifier where the features are BERT [CLS] embeddings!



Parameter Explosion!

Parameters are any values we have to set - e.g. weights

Naive Bayes

two classes, vocab size of 30k = 60k params

BERT-Large, 300 million params

More recent models in the trillions

Parameter Explosion!

Therefore, these big NNs are very data hungry!

We need many examples (at least 10x params) to train

Training on the internet, basically (Common Crawl)

- Multiple terabytes of text

Costs to train one model up to the millions USD

- not to mention all the failed attempts...

A Tricky Proposition

We got here empirically -

you see many cards have been stacked,

people kept trying stuff until they stayed standing

It all sounds reasonable, but it's also weird that it works

New subfield: BERTology

trying to understand what linguistic things

BERT et al know and can do, and why

What did we gain from doing this?

Better results on concrete tasks, real world applications

Neural Machine Translation for instance - transformative

previously very complex statistical systems,
now trained end-to-end

No feature engineering! (Lots of architecture tinkering.)

Many building blocks for complex models

New Terminology

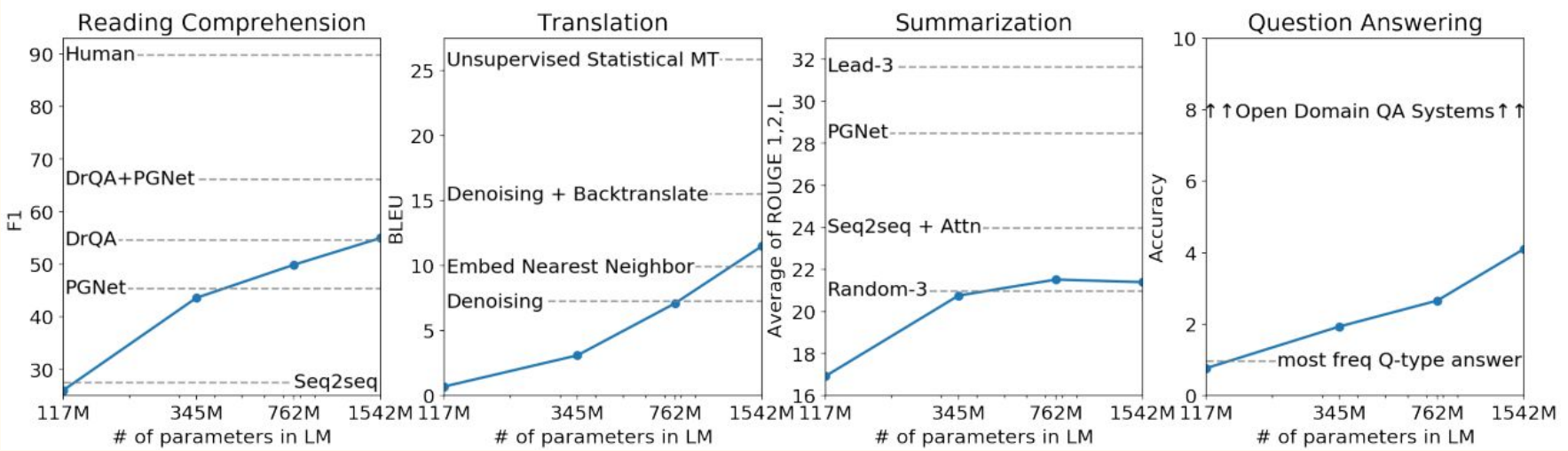
“Large Language Models” (LLMs)

LMs with (very) high parameter counts as adaptable or general-purpose NLP solvers

a.k.a. “foundation models” (FMs)
“pre-trained language models” (PLMs)

Huge Capacity → “Emergent” Properties

LLMs appear to display new abilities with greater size,



Huge Capacity → “Emergent” Properties

LLMs appear to display new abilities with greater size

One of the most striking has been “few-shot learning,” also called “in-context learning” or “prompting”

General paradigm:

- providing correct examples in LM input context
- prompt for generation of structured output

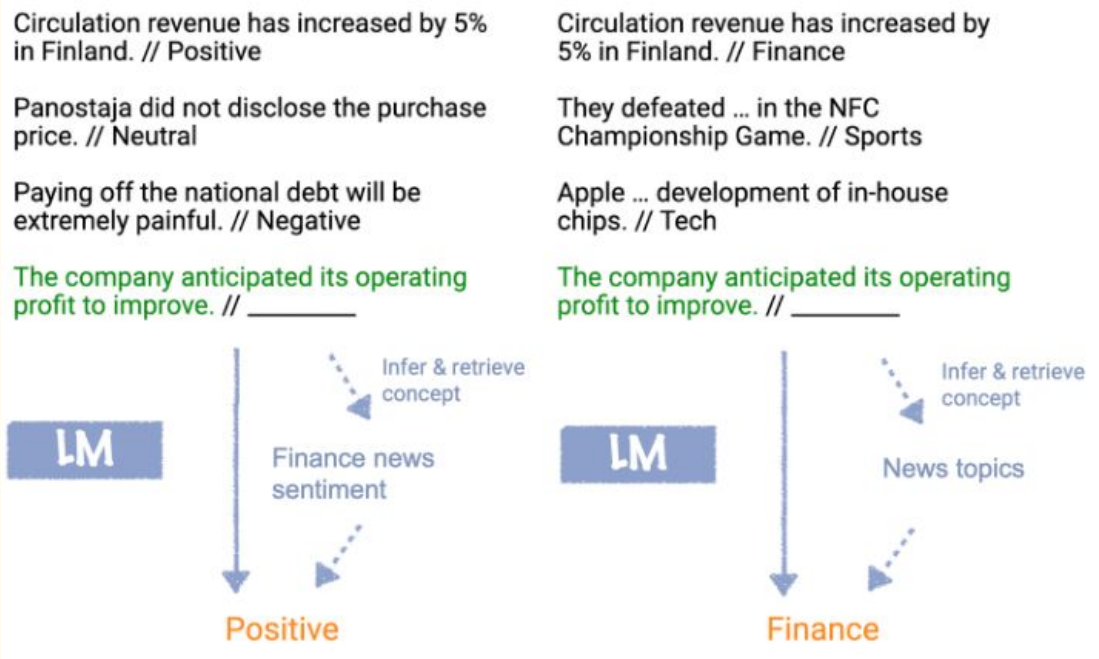
In-Context Learning Paradigm

Gradient as to what the LLM is shown

- Fine-tuning: thousands of examples, model weights are updated (either in a final layer or thoughout)
- Few-shot: provide a small number of examples in the context and ask for an answer, model weights constant
- One-shot: show one example and ask for an answer
- Zero-shot: provide a natural language description of the task and ask for an answer

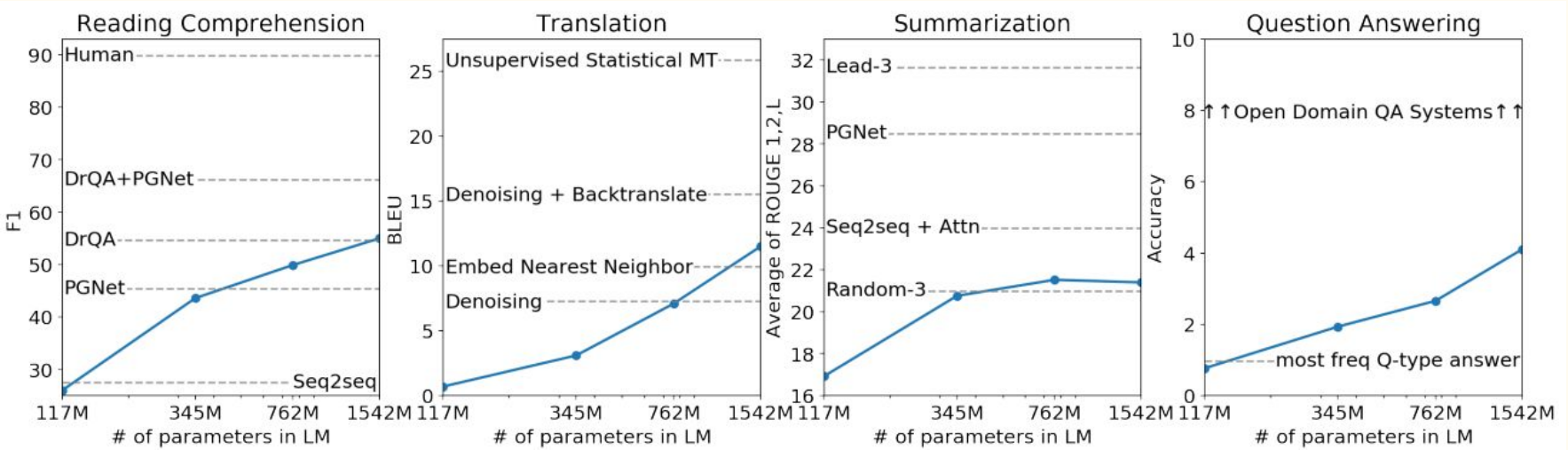
In-Context Learning Examples - Few-shot

From http://ai.stanford.edu/blog/understanding-incontext/



… even works for MT! (somewhat)

Zero-shot performance from GPT-2 (Radford et al. 2019):

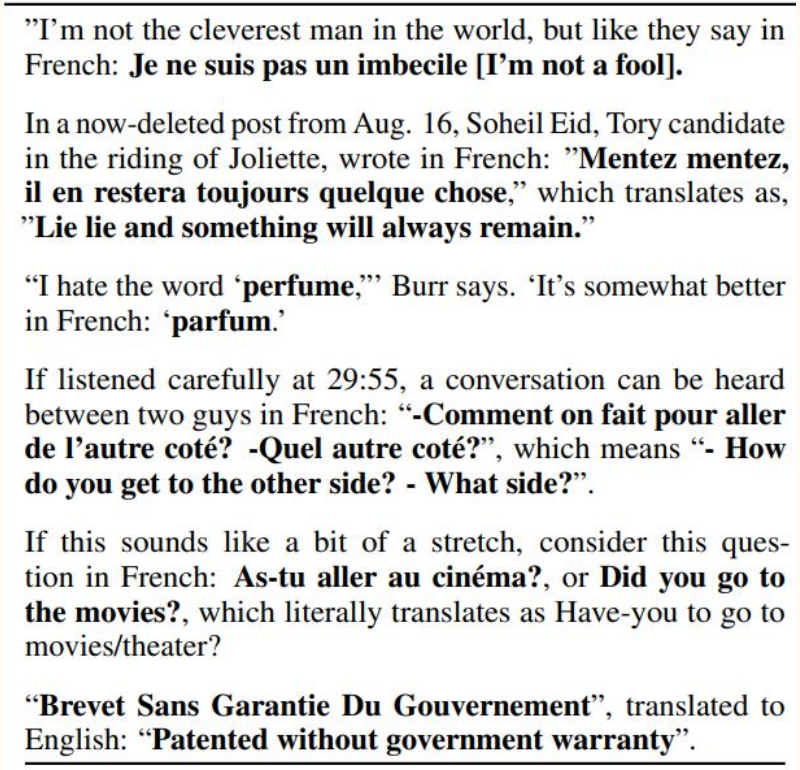


… even works for MT! (somewhat)

- How is that possible?

One possible explanation:

- Natural demonstrations of useful language tasks do appear in the wild!



”I’m not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile [I’m not a fool].**

In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: ”**Mentez mentez, il en restera toujours quelque chose**,” which translates as, ”**Lie lie and something will always remain.**”

“I hate the word ‘**perfume**,”’ Burr says. ‘It’s somewhat better in French: ‘**parfum**.’

If listened carefully at 29:55, a conversation can be heard between two guys in French: “**-Comment on fait pour aller de l’autre coté? -Quel autre coté?**”, which means “**- How do you get to the other side? - What side?**”.

If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as Have-you to go to movies/theater?

“**Brevet Sans Garantie Du Gouvernement**”, translated to English: “**Patented without government warranty**”.

How has this affected the field?

The gap between modern, task-based NLP and "Computational Linguistics" has maybe never been wider

Divergence between properly linguistic/behavioral and simply "increase performance on this task"

Still, earlier non-neural methods are not worthless!

Especially re: interpretability

Great Free Courses on This Neural Stuff

Stanford CS224n:
https://www.youtube.com/playlist?list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z

CMU CS 11-747:
https://www.youtube.com/playlist?list=PL8PYTP1V4I8AkaHEJ7lOOrlex-pcxS-XV

... and obviously others here at NU!

Model Ecosystem

Training these huge models is expensive,
inference (running them on stuff) is relatively cheap.

Community sharing is ideal and happening constantly

HuggingFace is an incredible resource of models and datasets, with a corresponding python library:

https://huggingface.co/models (quick demo)