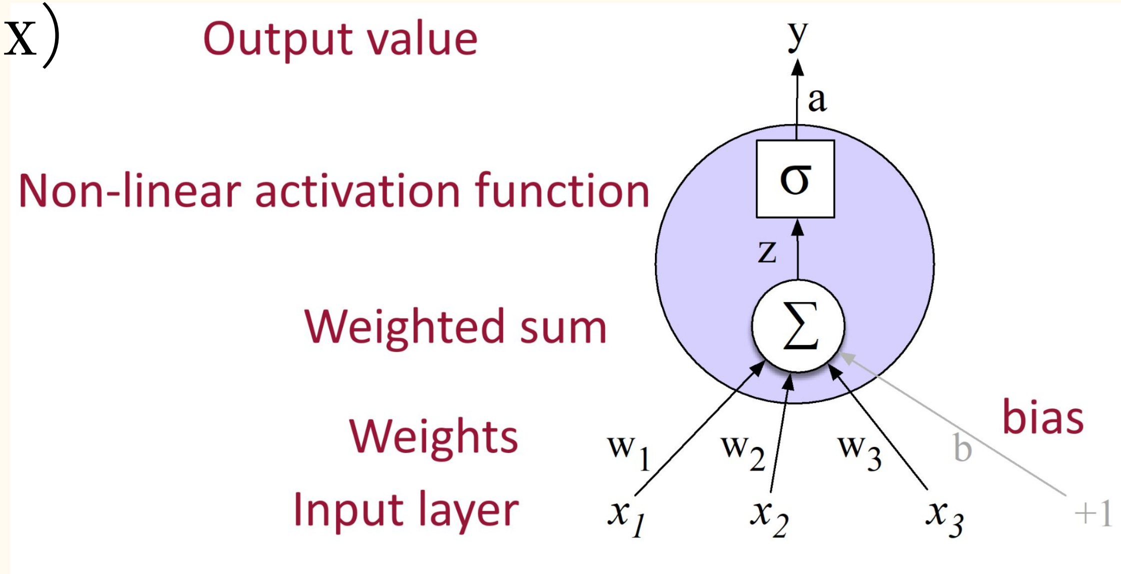
LING 334 - Introduction to Computational Linguistics

Week 9

State of the Art

Reminder - One Neuron (≈ Logistic Regression)

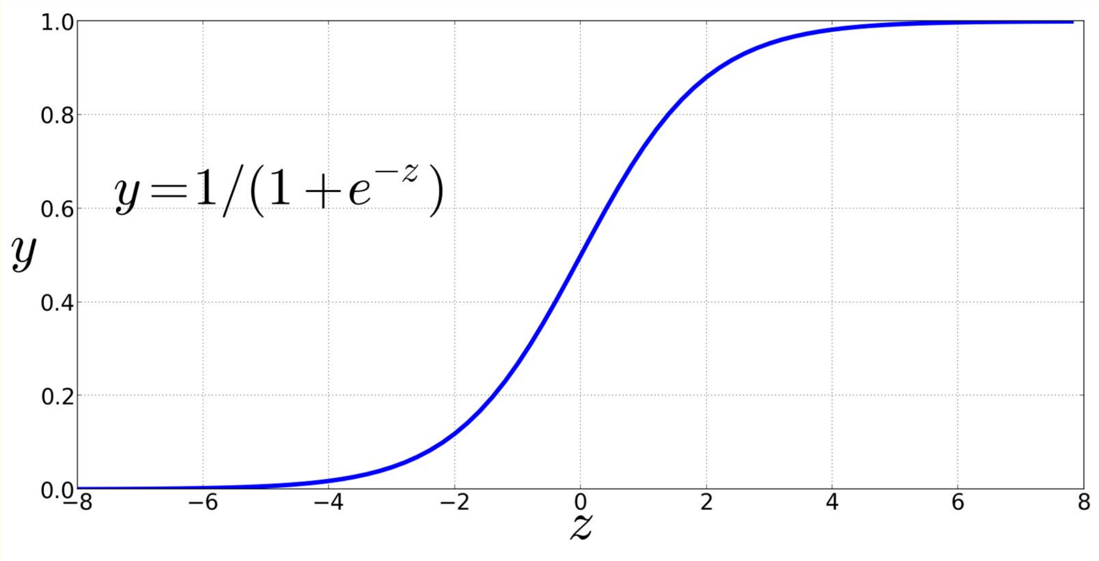
Biologically inspired
(but way less complex)



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



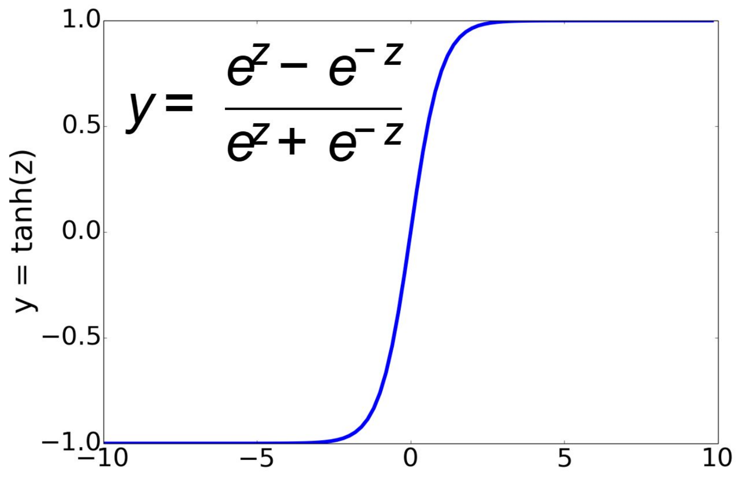
x axis = sum of weights times inputs

y axis = output value of neuron

Non-Linearities - tanh and ReLU

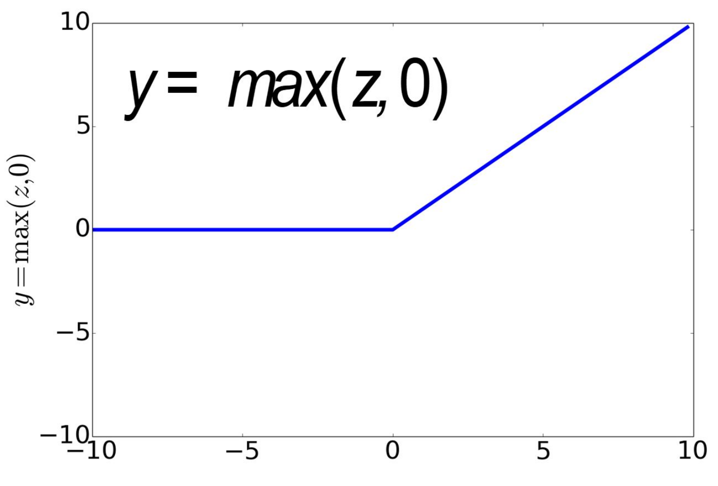
tanh

(like sigmoid, works better)



ReLU

(most common)

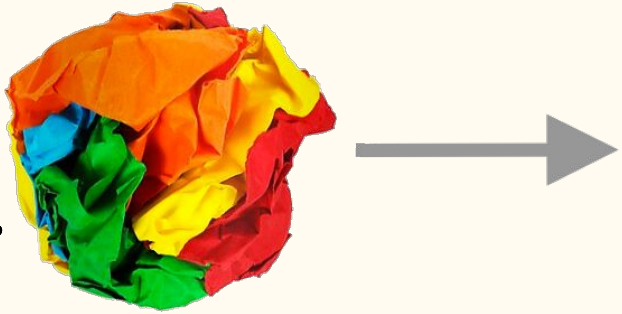


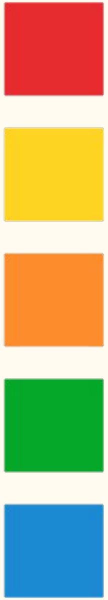
credit J+M, SLP slides

Why Non-Linearities?

Naive Bayes is a linear classifier

Decision boundary from $\sum w \cdot x$





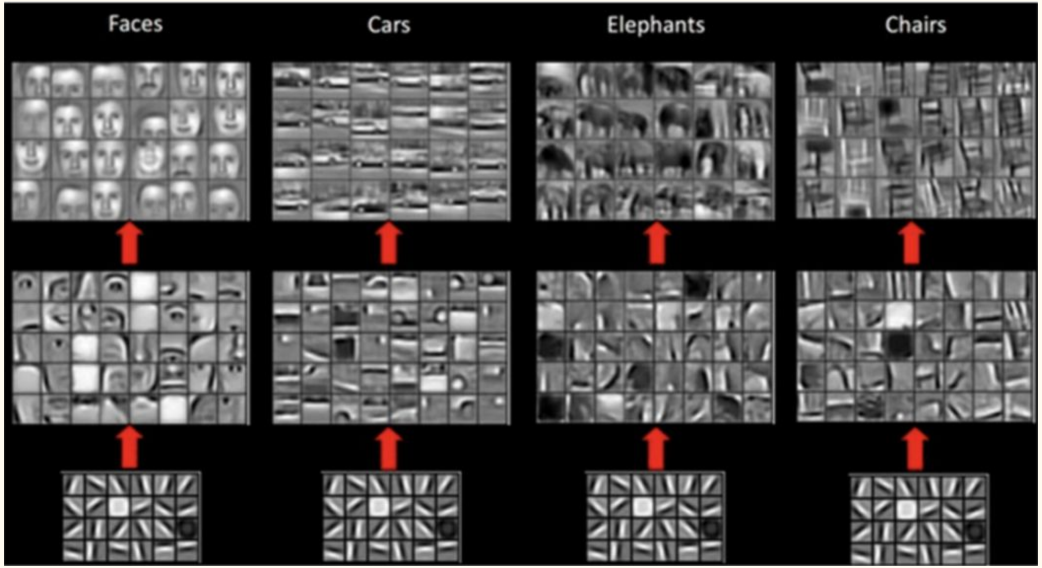
For NNs, key idea is representing the input in increasingly abstract non-linear transformations

“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

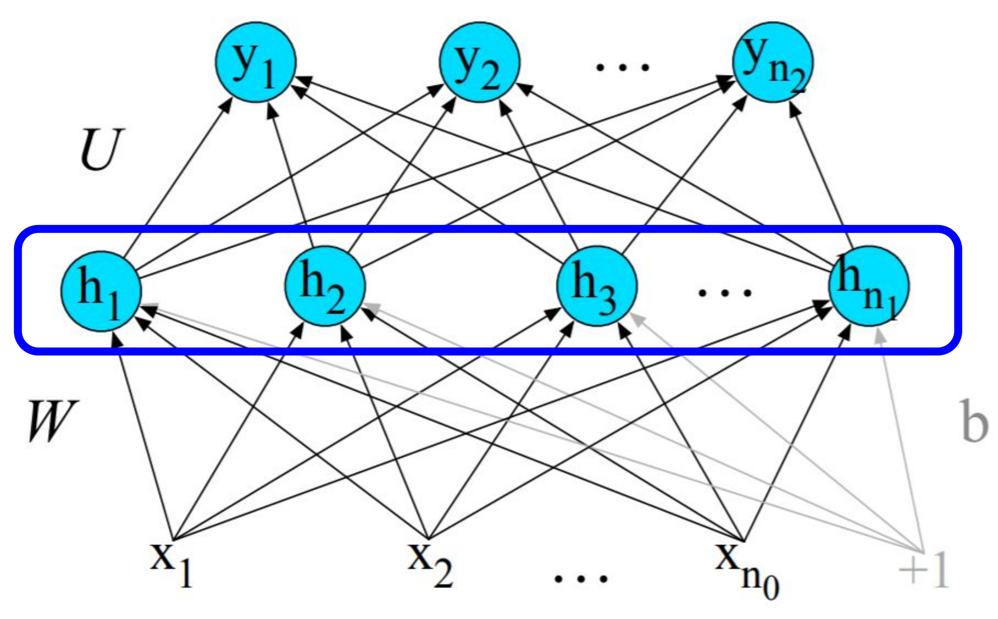


Simple Feed-forward Neural Net

Each arrow represents multiplication of value by a weight

Summed at each node, non-linear transform

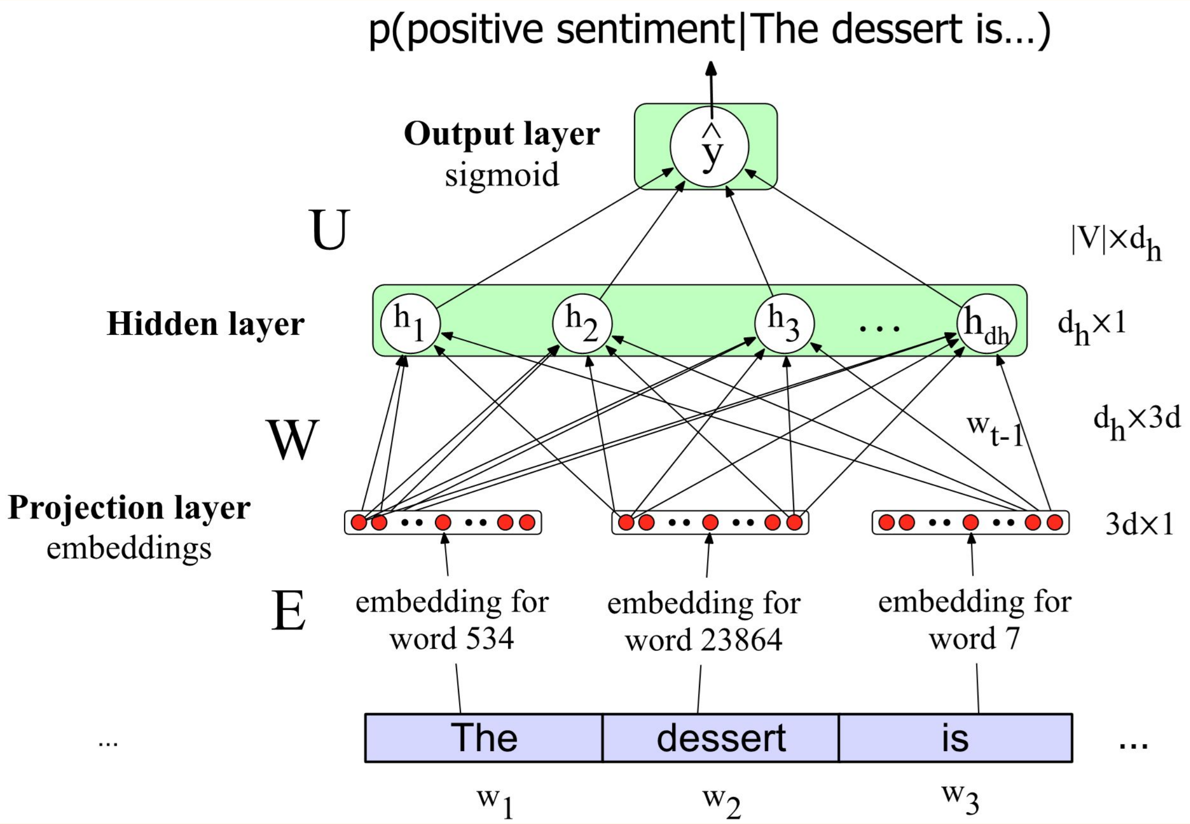
Like multiple logistic regressions running concurrently on the same inputs



Simple NN - Another View

Large input layer!

Many weights!

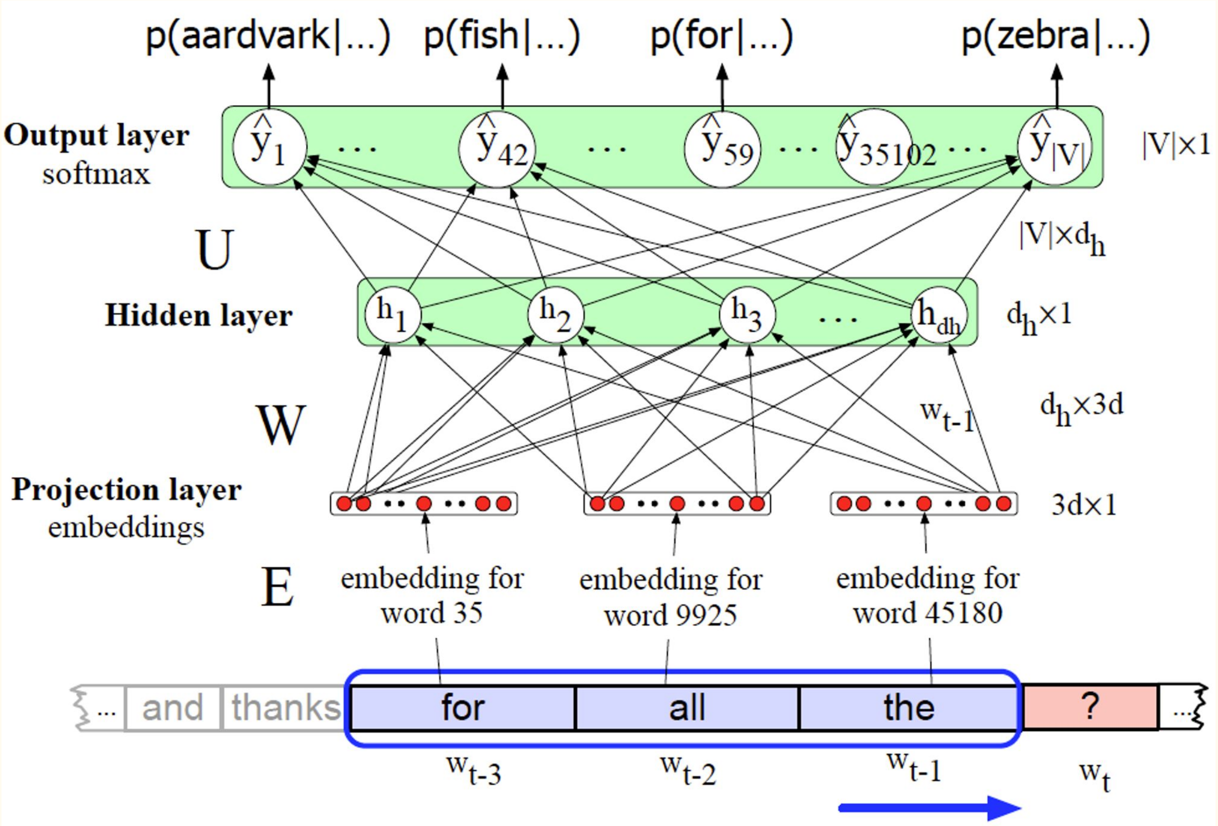


Neural Network Language Model

Sliding window over words

Large output layer of all words in V

Notice the hidden layer is itself a vector!

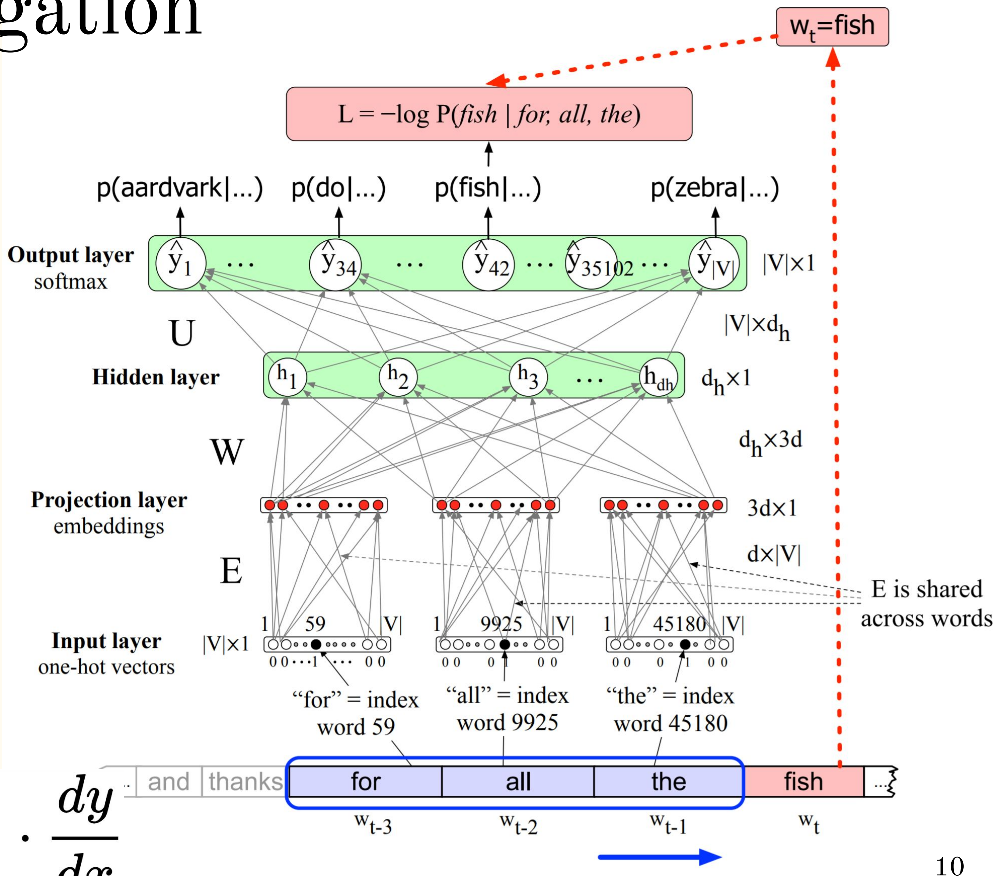


Training via Backpropagation

Loss = function saying, how wrong are we?

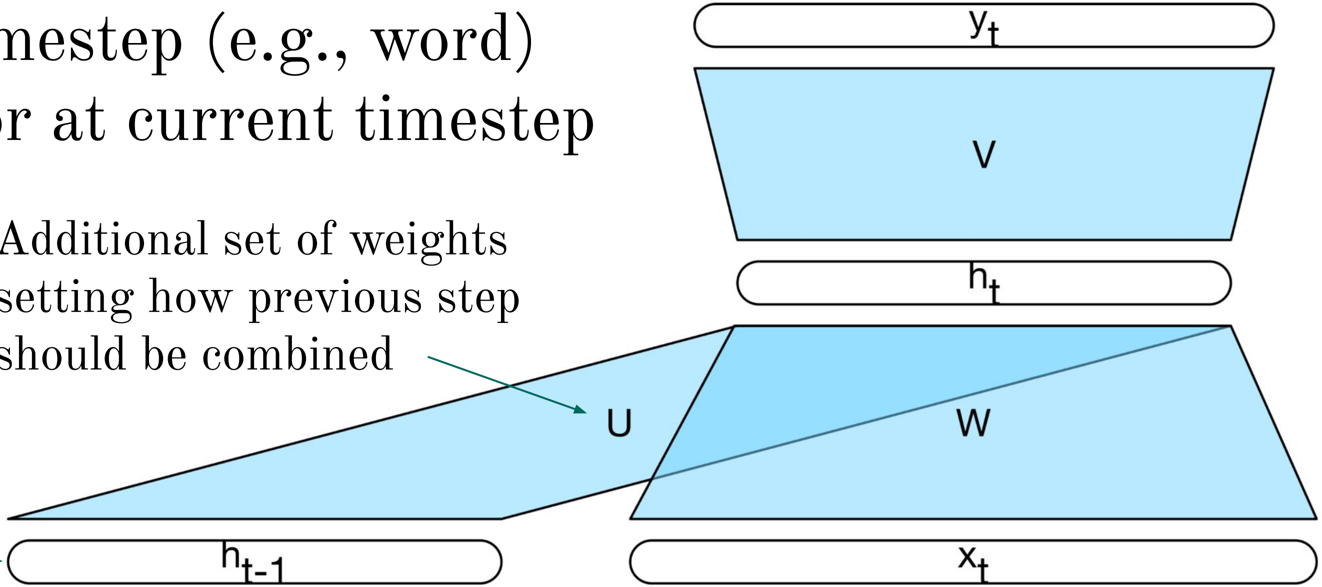
Derivative of this function at any point tells us which way to go to be less wrong

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



Recurrent Neural Networks

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



Additional set of weights setting how previous step should be combined

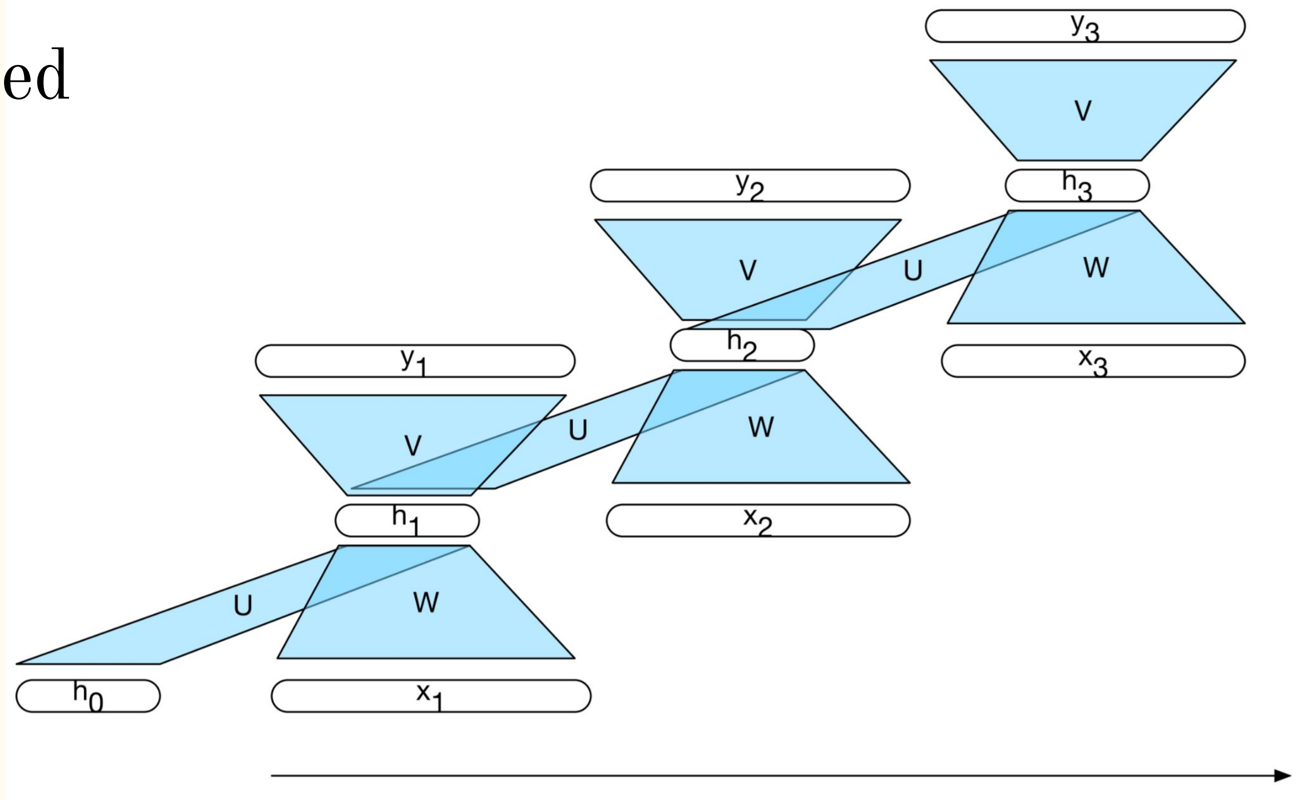
Vector of hidden state from the previous timestep

Inputs at this timestep

Recurrent Neural Networks - unrolled view

Weights are shared across timesteps

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

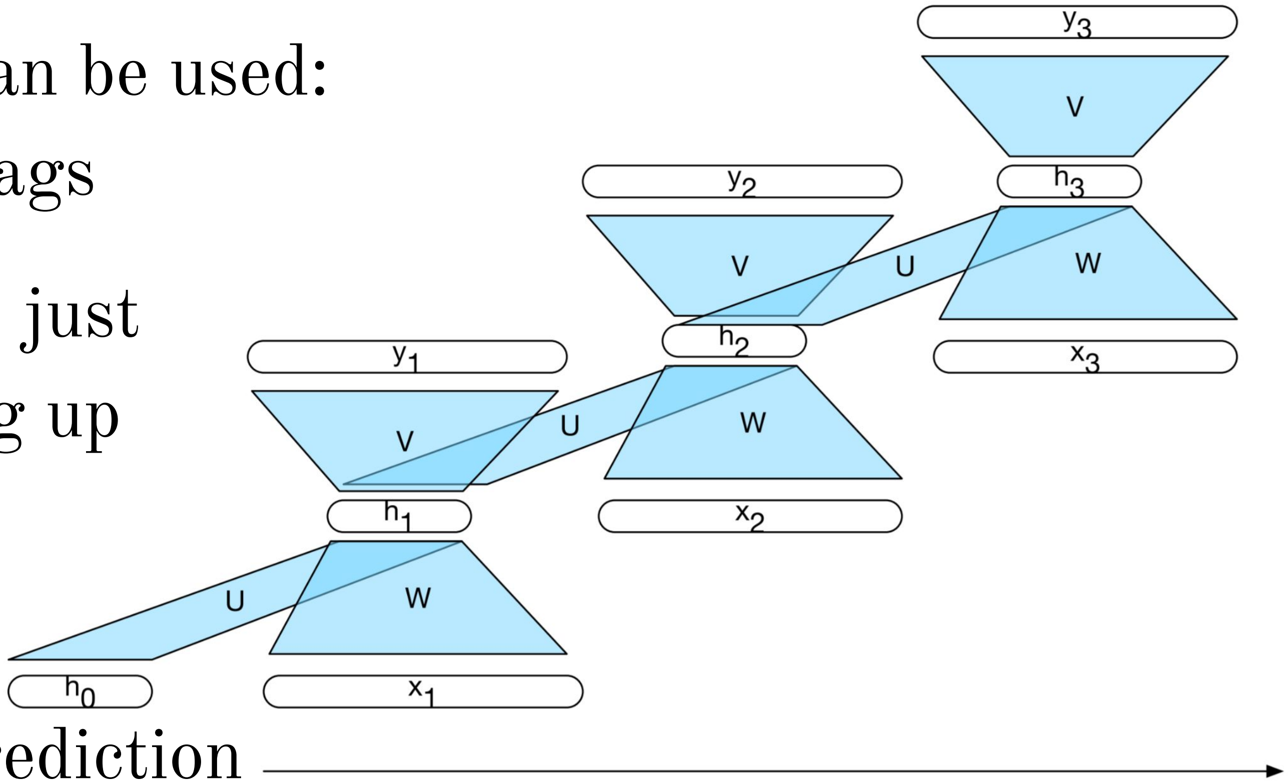


Recurrent Neural Networks - unrolled view

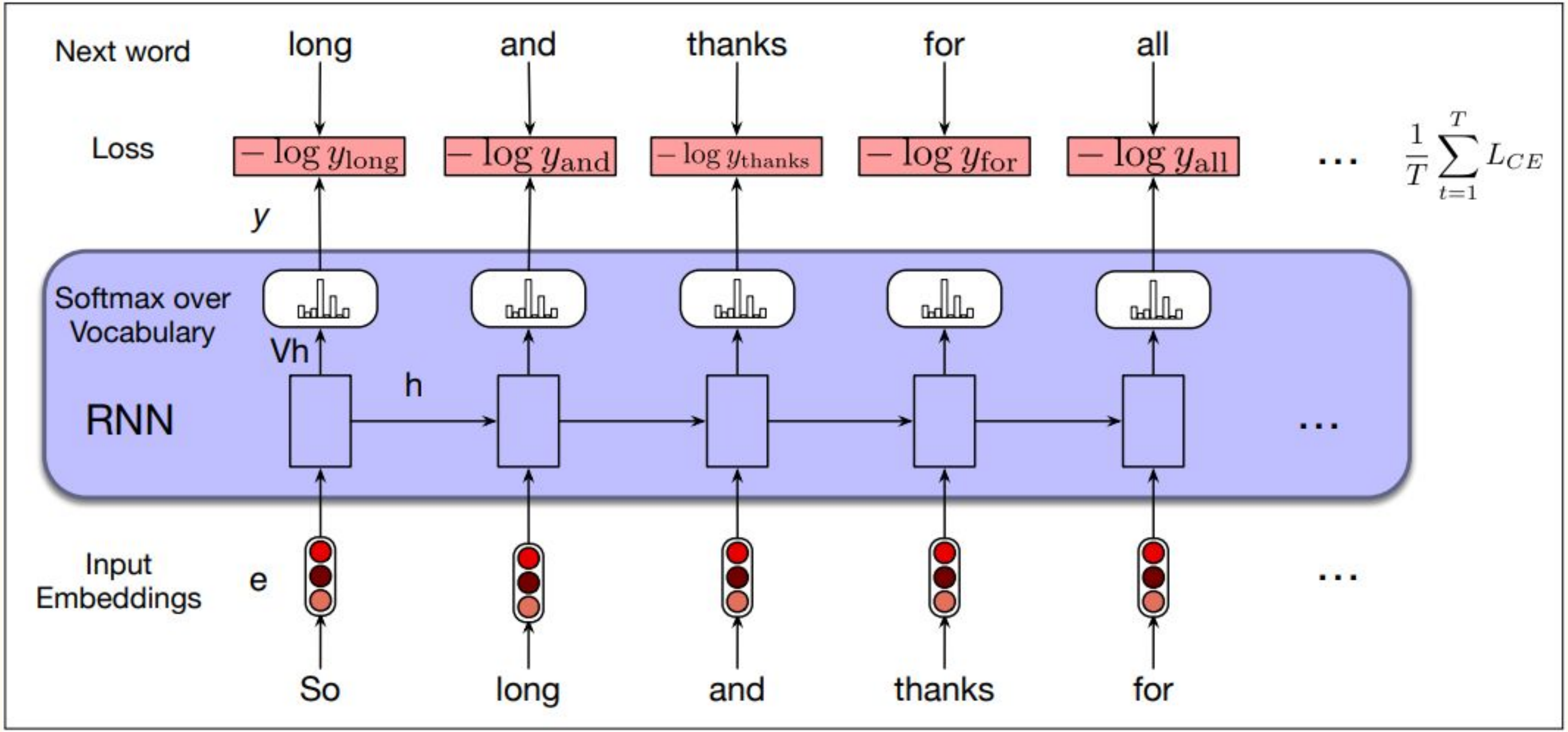
Output layer (y) can be used:
e.g. predict POS tags

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

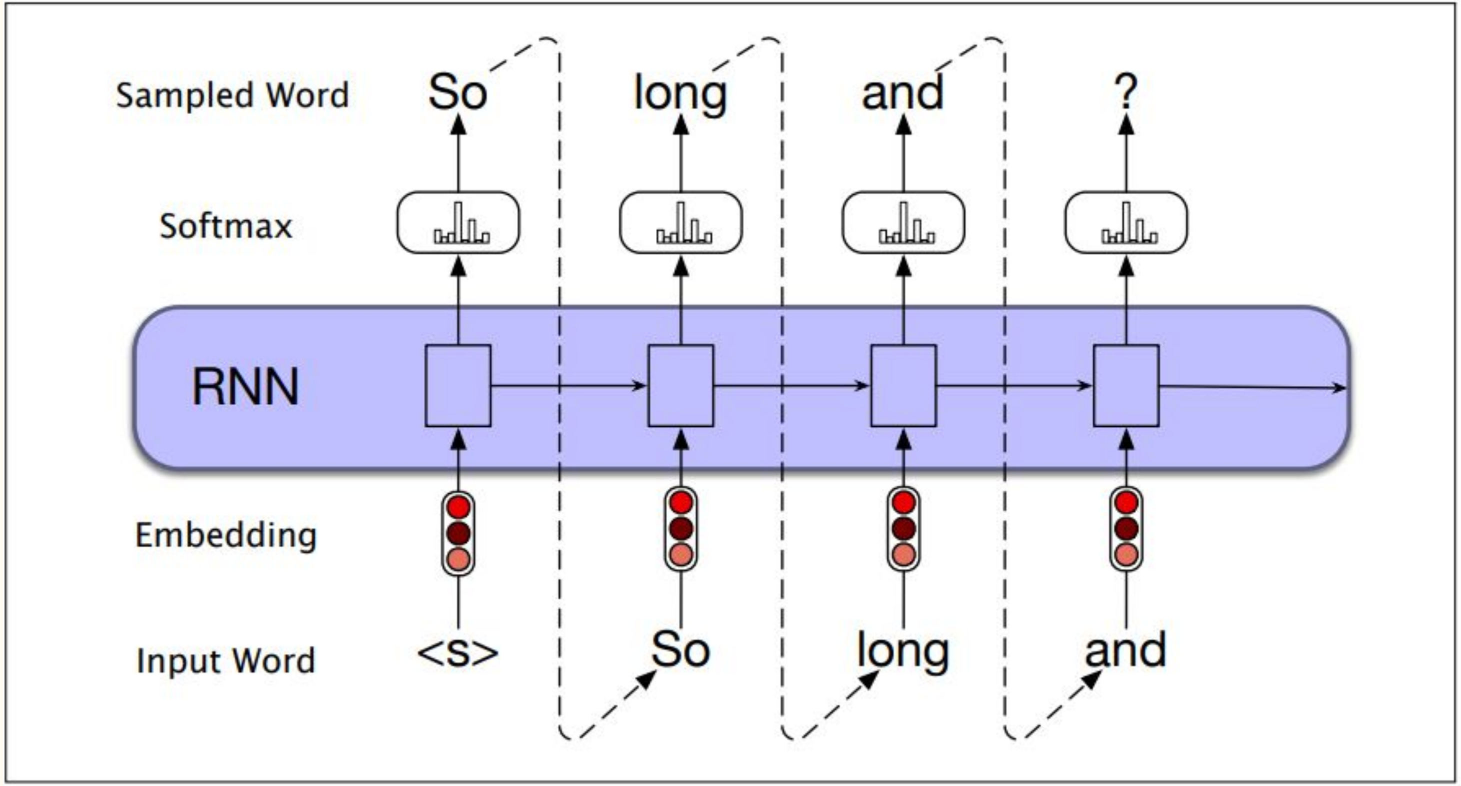
Can just use final output layer for prediction



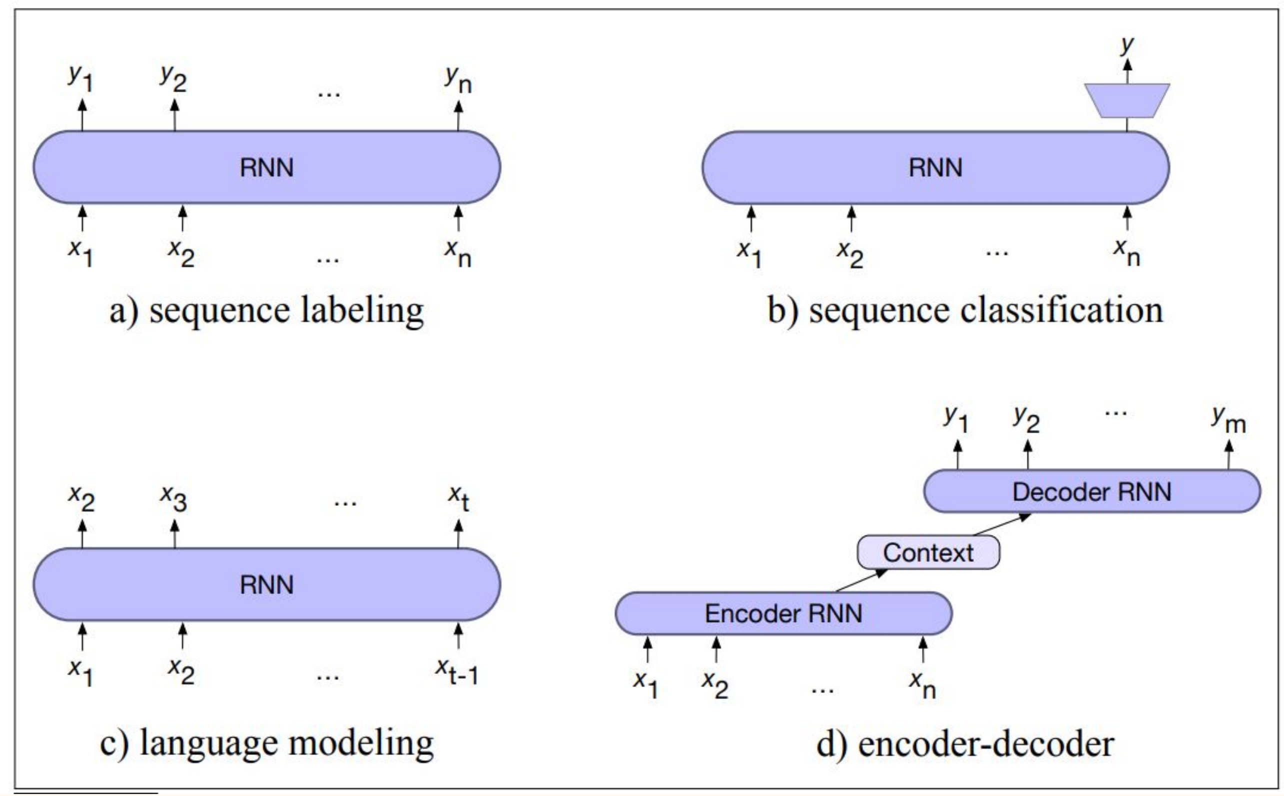
RNNs as a language model



RNNs as a language model - generation



Recurrent Neural Networks - a flexible mechanism

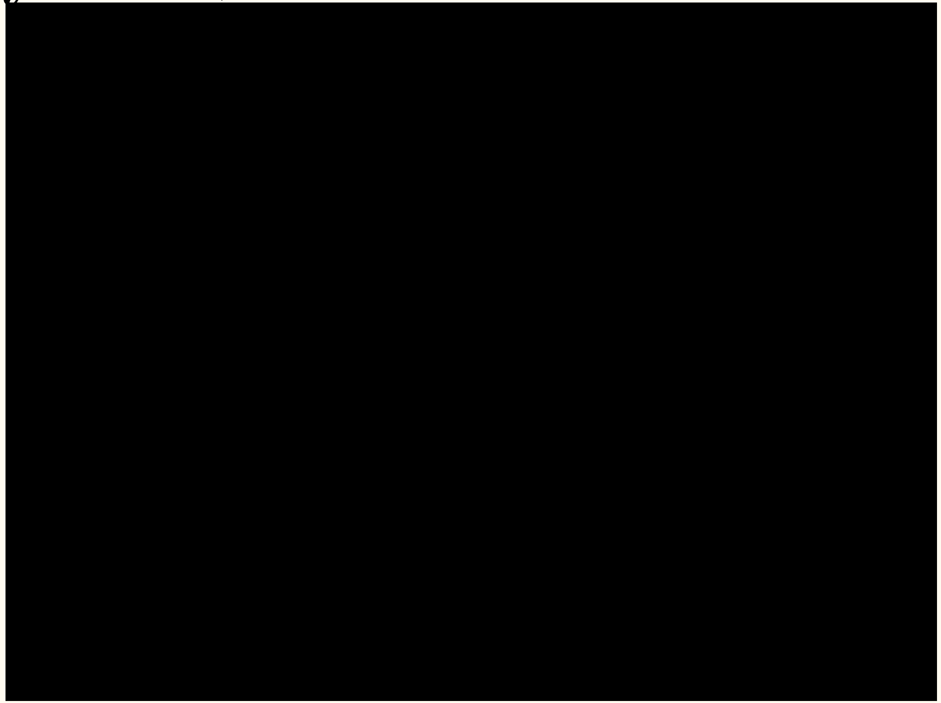


Sequence-to-Sequence Models

Encode a sequence word by word, building the hidden state

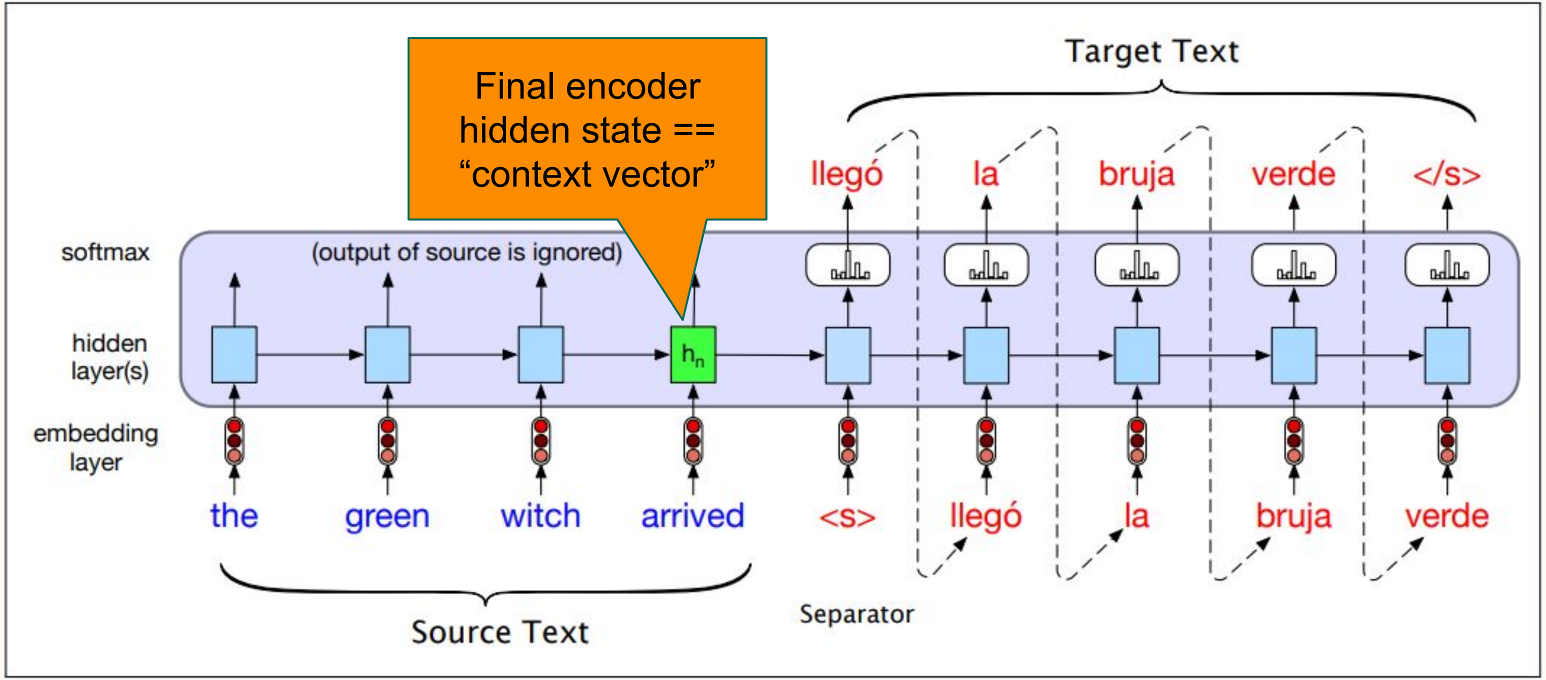
Pass final hidden state to another RNN to “decode”

Common use case: machine translation



Seq2seq Models - another view

Remember encoder and decoder are separate RNNs

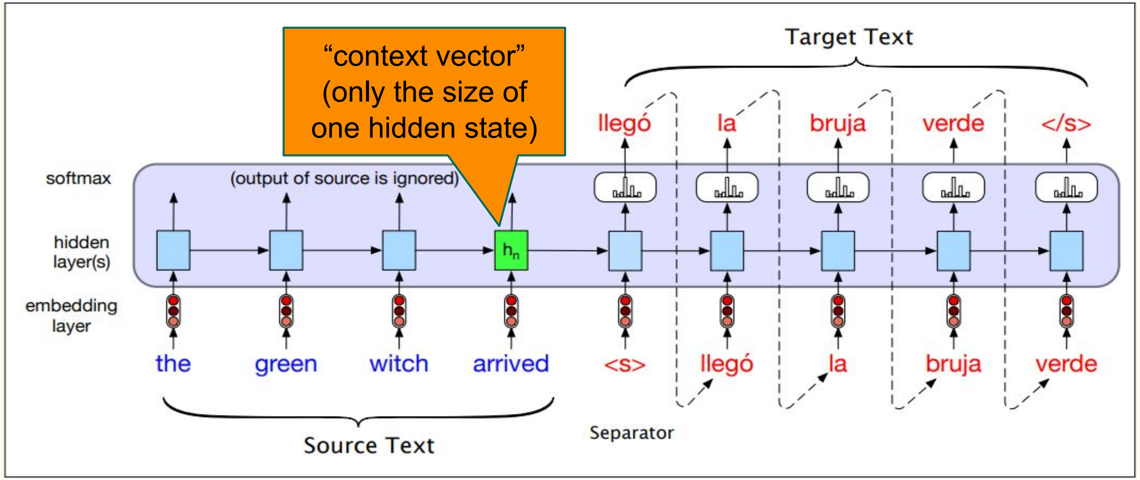


Bottleneck Problem

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?

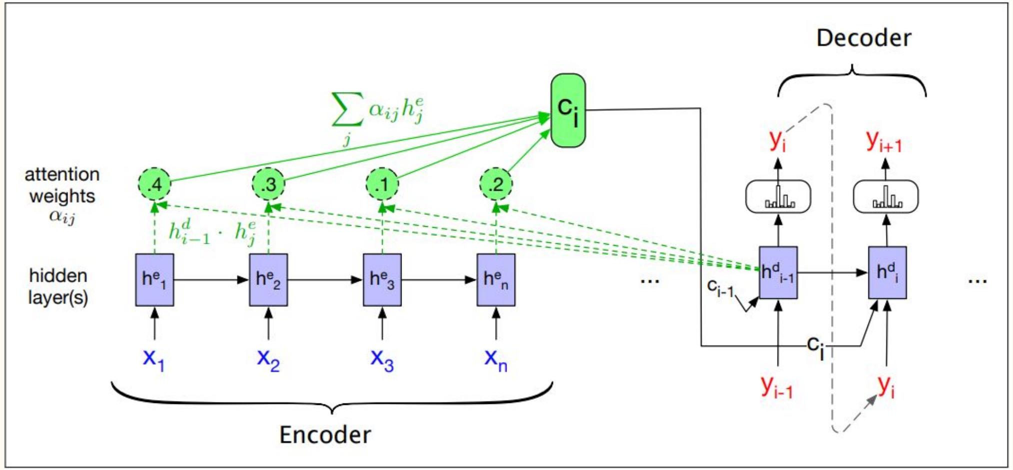
Answer: Attention!



Attention

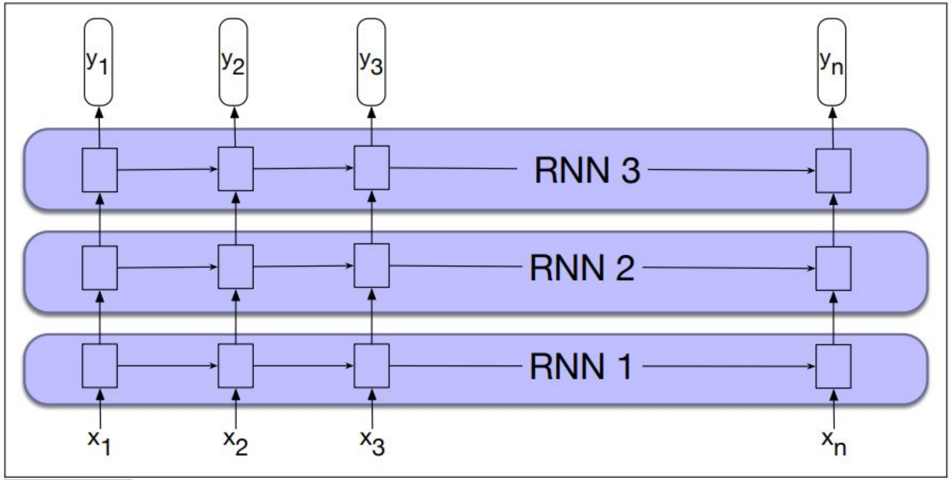
Incorporate an additional "context vector" at each step:

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



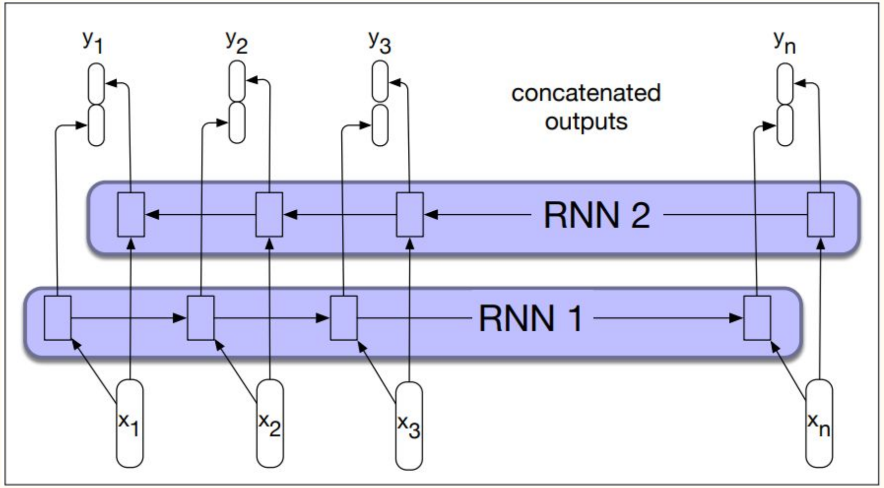
Other Key Concepts: Stacked and Bidirectional

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.

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)

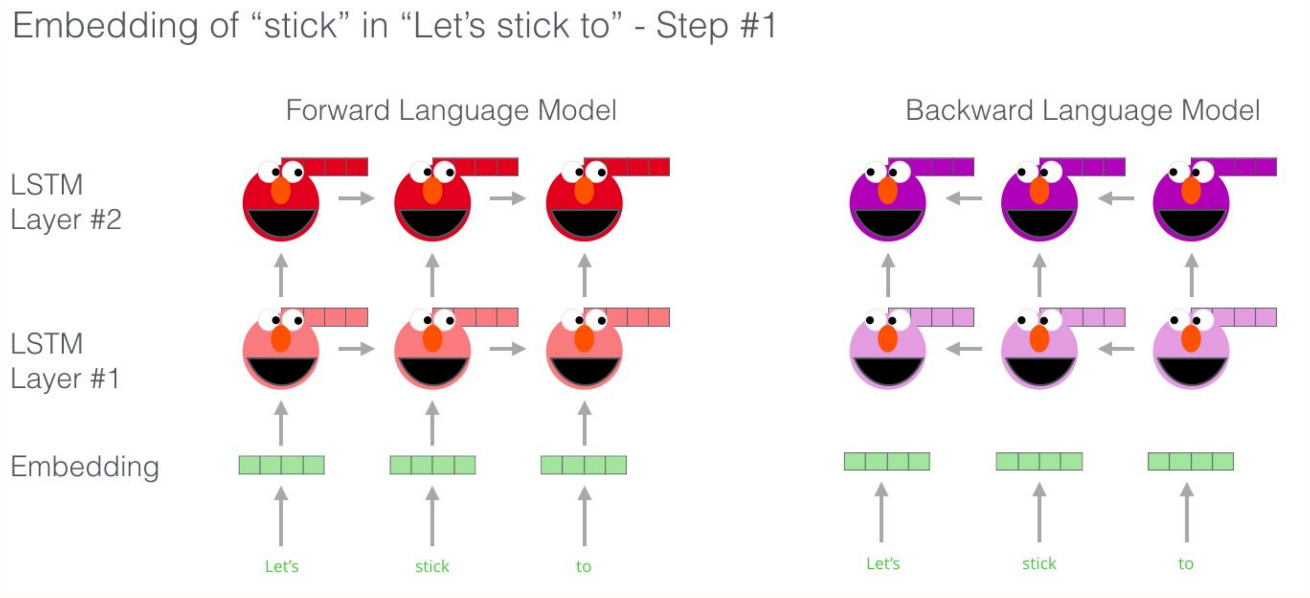


figure from Jay Alammar

ELMo (Embeddings from a Language Model)

Result is “contextual” embeddings

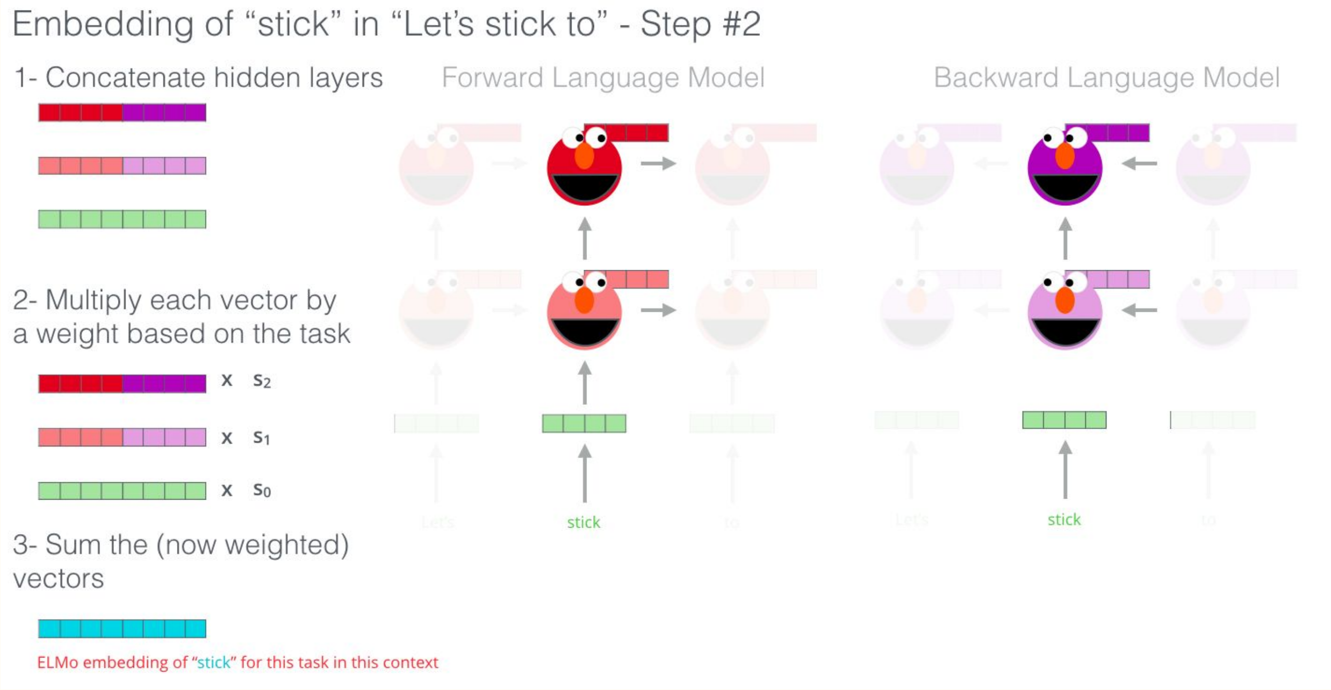
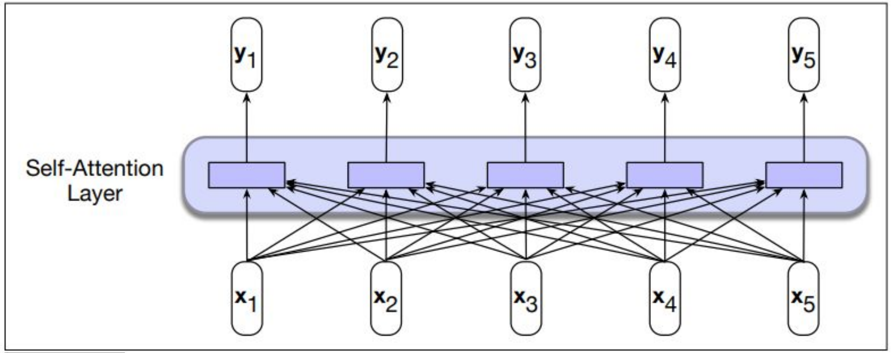


figure from Jay Alammar

The Muppet Parade

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

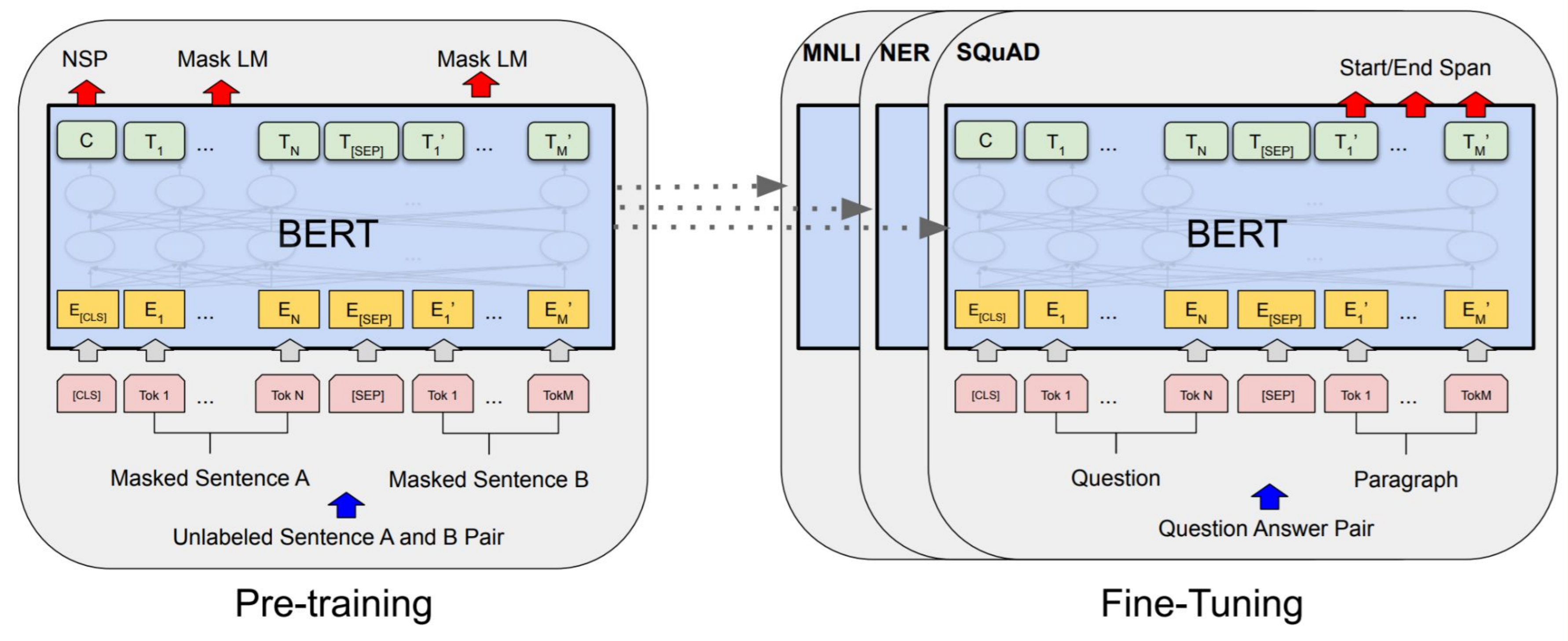


Many layers!
Details matter a lot!

Very common paradigm:

- “Fine-tune” BERT-like model for a specific task
- e.g., train it a little bit extra on some relevant data

Pre-Training → Fine-Tuning Paradigm

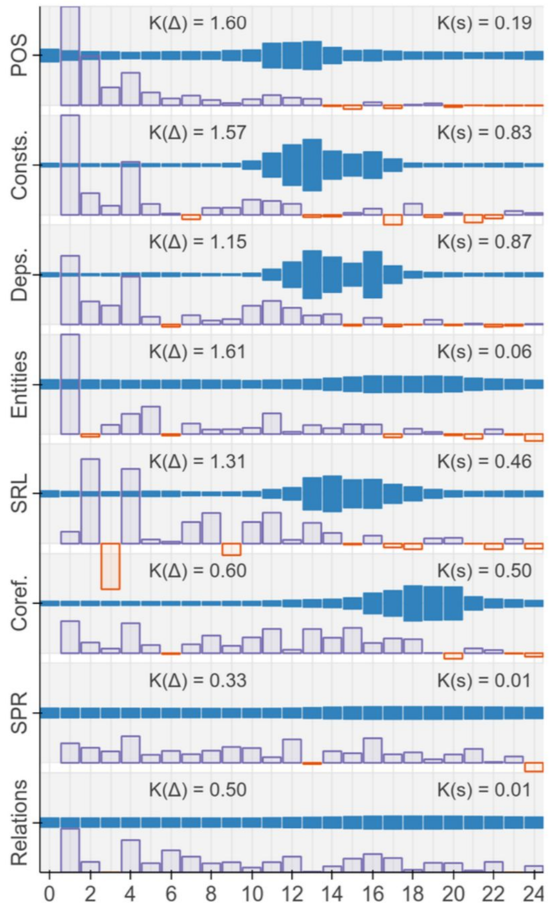


The many layers of BERT

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)

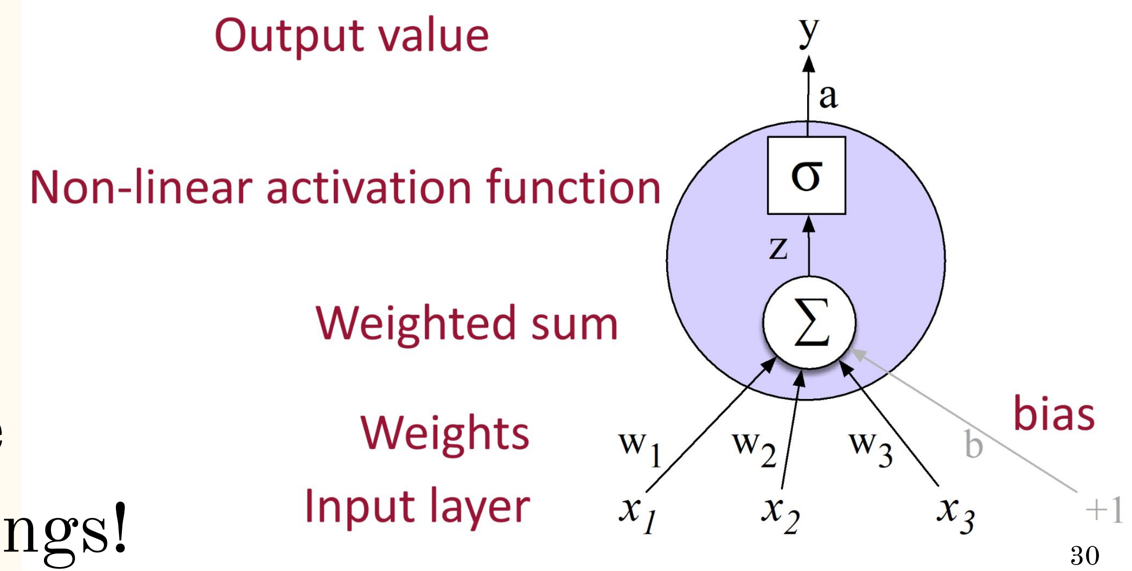


BERT for Classification

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

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)

Coming Up

Thursday this week:

Final project brainstorming activities

Next Week Tuesday:

Topic Models (unsupervised learning)

Next Week Thursday:

Larger discussion on contemporary issues