LING 334 - Introduction to Computational Linguistics

Week 9

State of the Art

Plan for Today

Neural nets crash course!

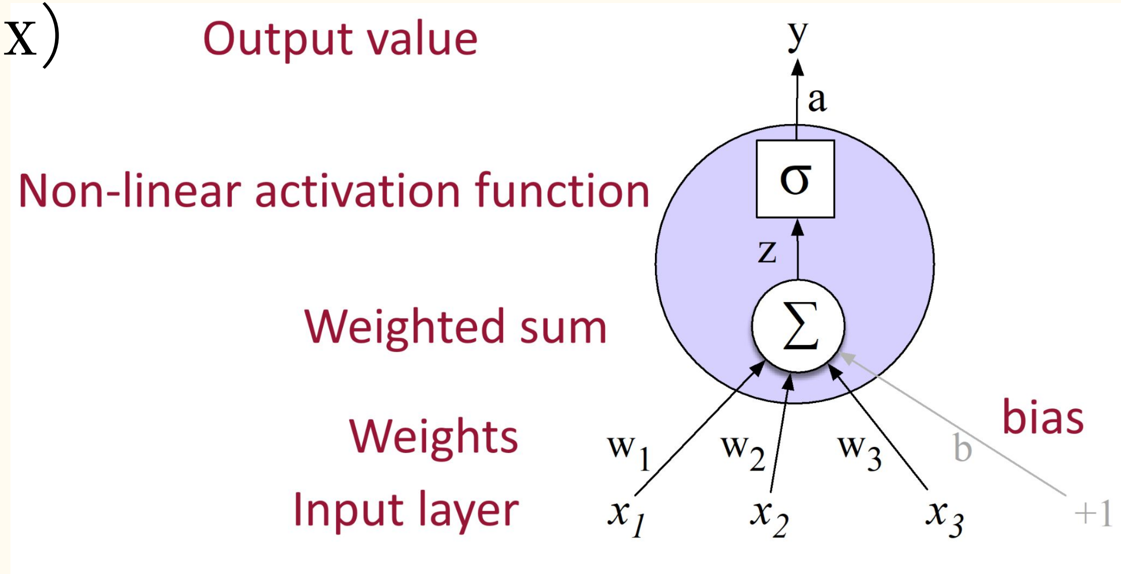
Conceptual understanding -
up to and including BERT-style (“transformers”)

Contextualizing NNs in the field

Where to go from here

One Neuron ($\approx$ Logistic Regression)

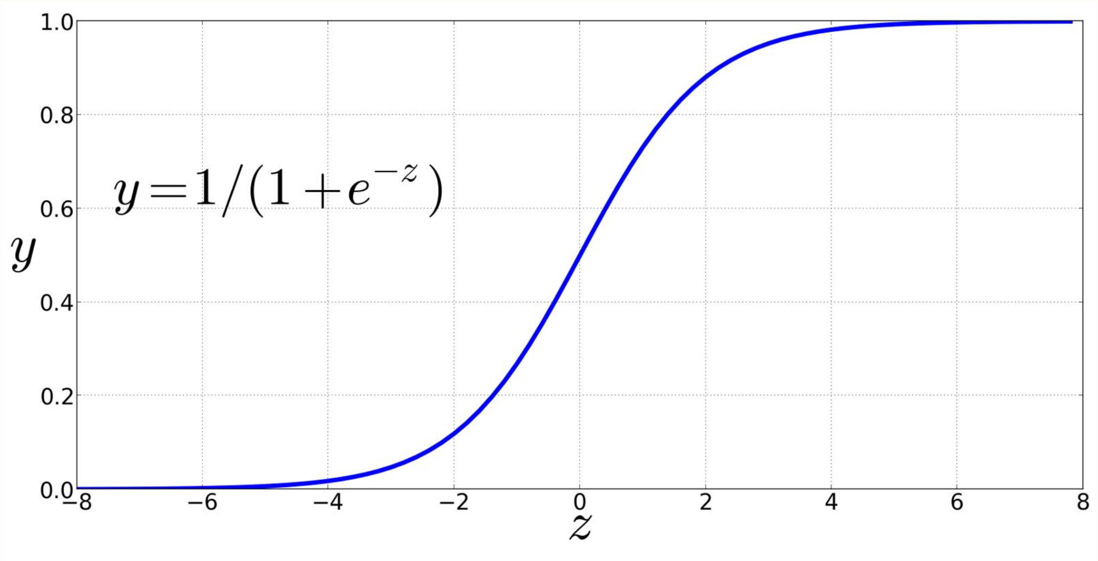
Biologically inspired
(but way less complex)



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

Non-Linearities - Sigmoid

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



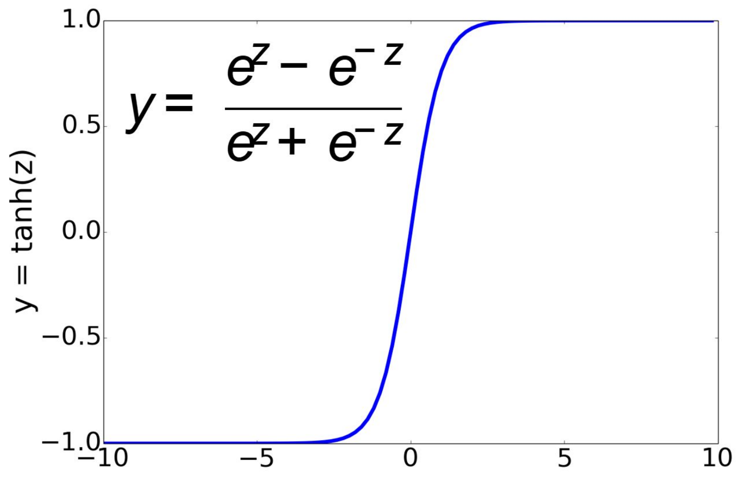
x axis = sum of weights times inputs

y axis = output value of neuron

Non-Linearities - tanh and ReLU

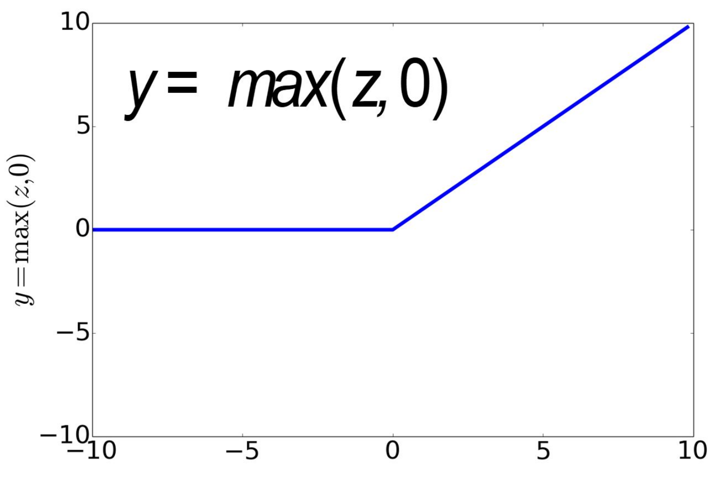
tanh

(like sigmoid, works better)



ReLU

(most common)



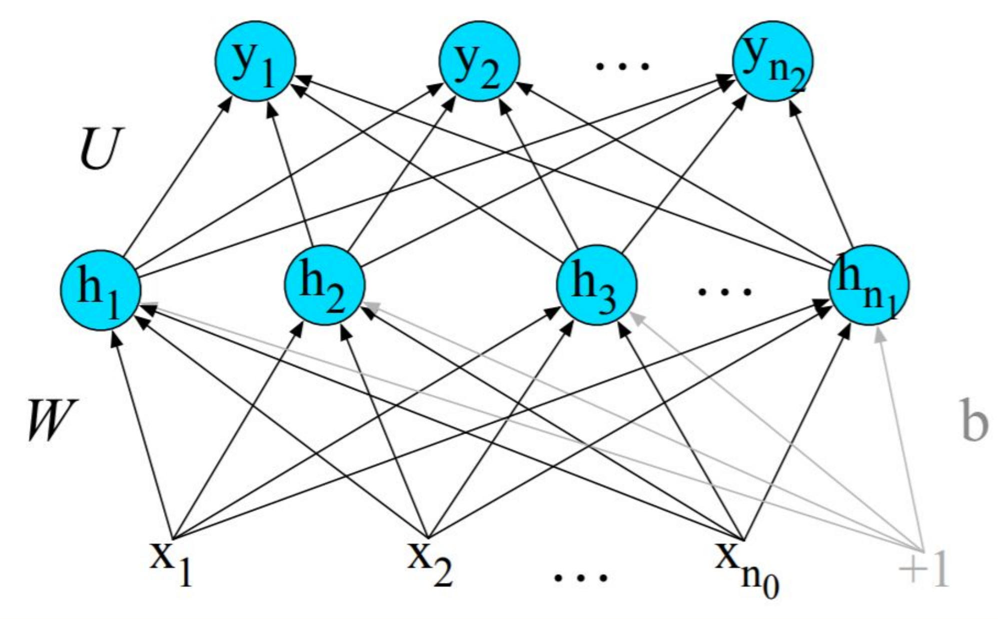
credit J+M, SLP slides

Simple Feed-forward Neural Net

Each arrow represents multiplication of value by a weight

Summed at each node, non-linear transform

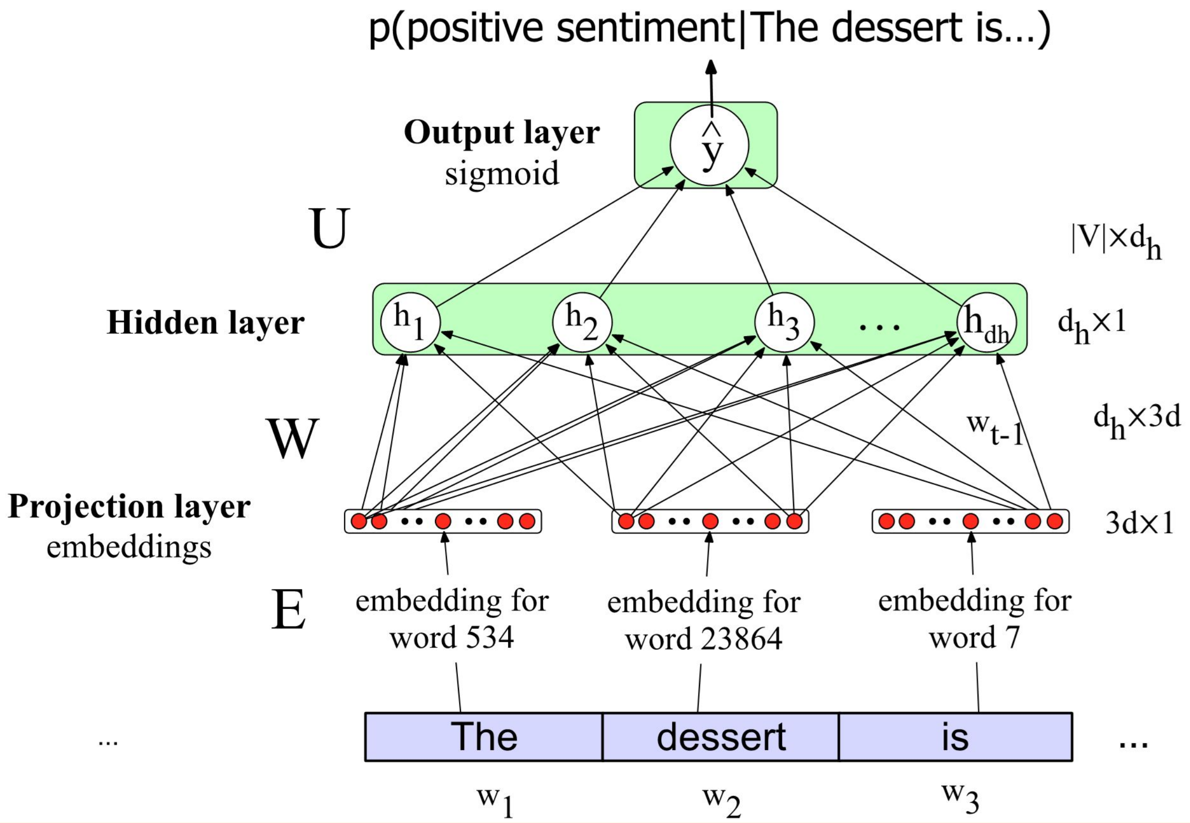
Like multiple logistic regressions running concurrently on the same inputs



Simple NN - Another View

Large input layer!

Many weights!

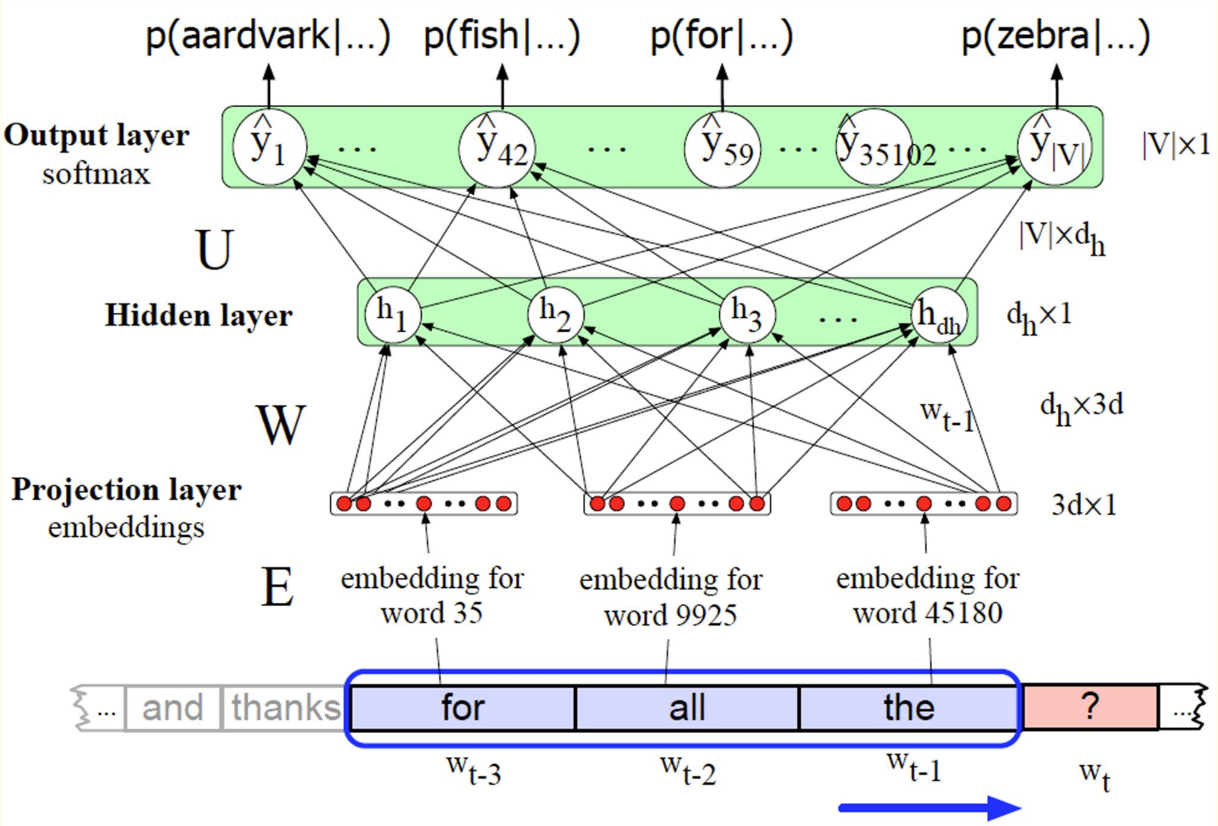


Neural Network Language Model

Sliding window over words

Large output layer of all words in V

Notice the hidden layer is itself a vector!

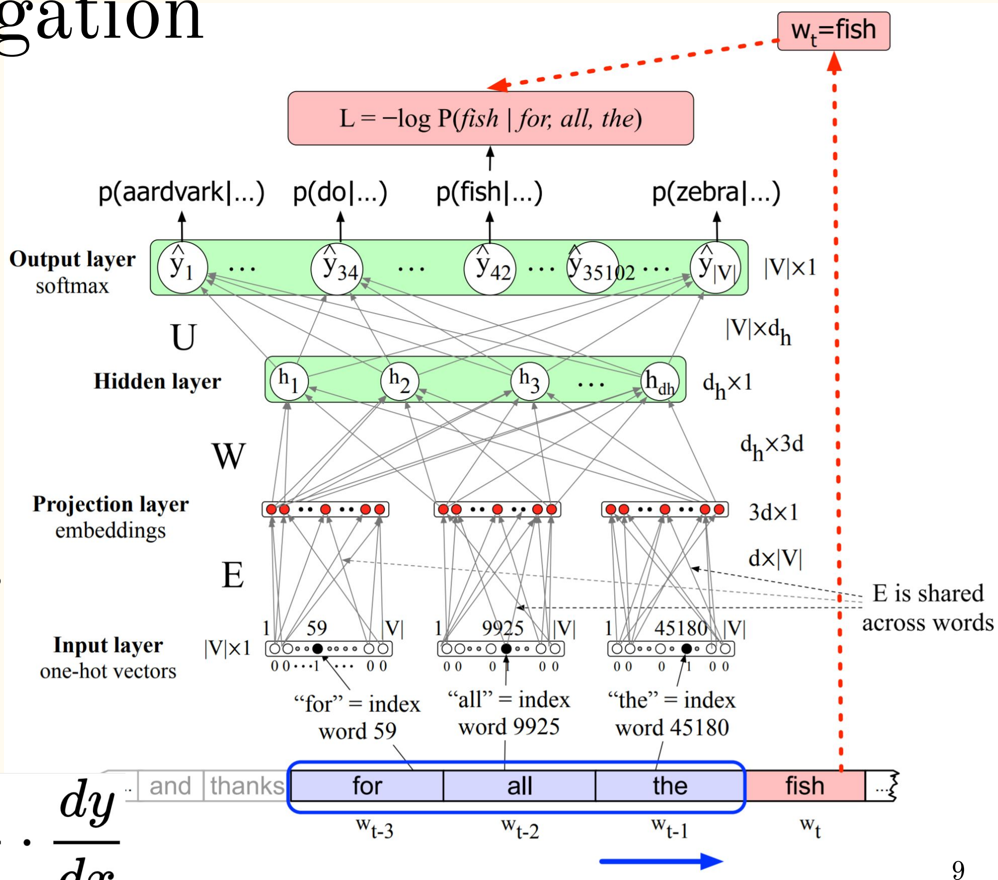


Training via Backpropagation

Loss = function saying, how wrong are we?

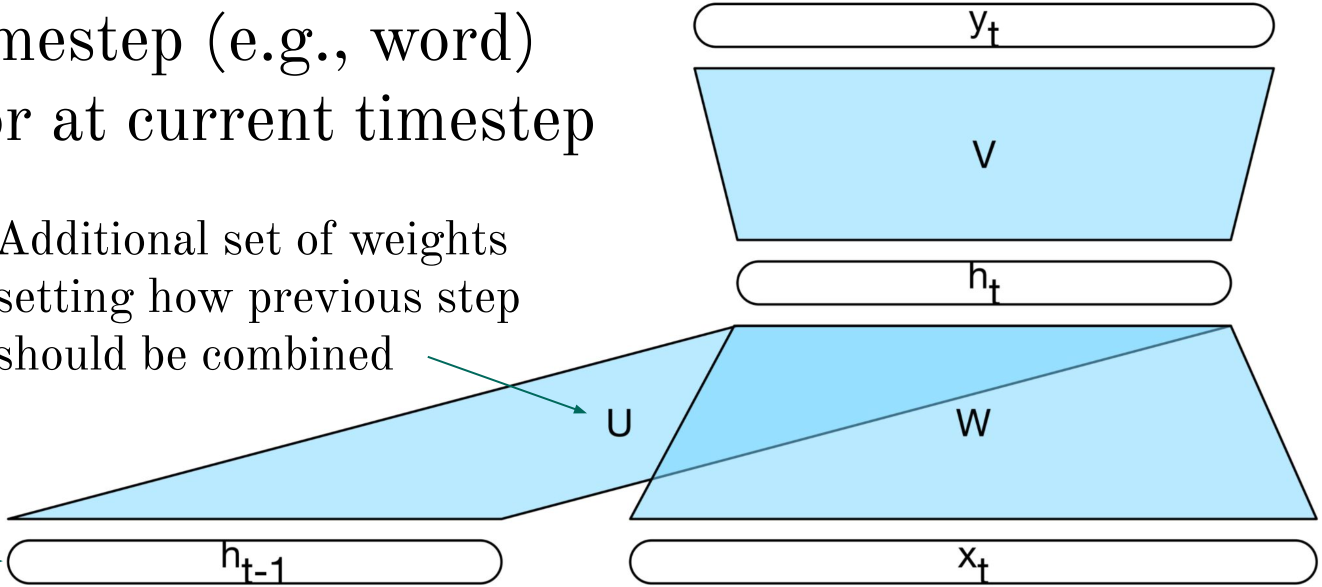
Derivative of this function at any point tells us which way to go to be less wrong

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

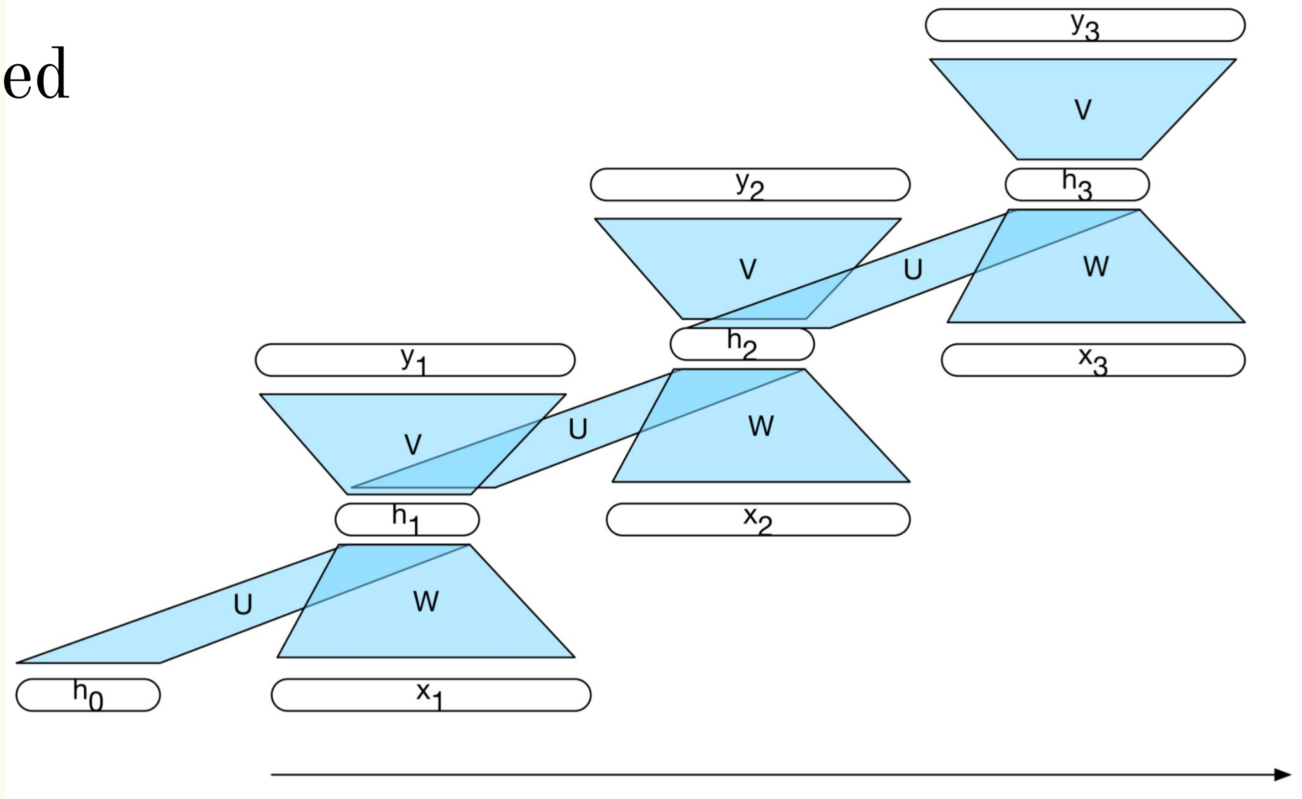
Vector of hidden state from the previous timestep

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

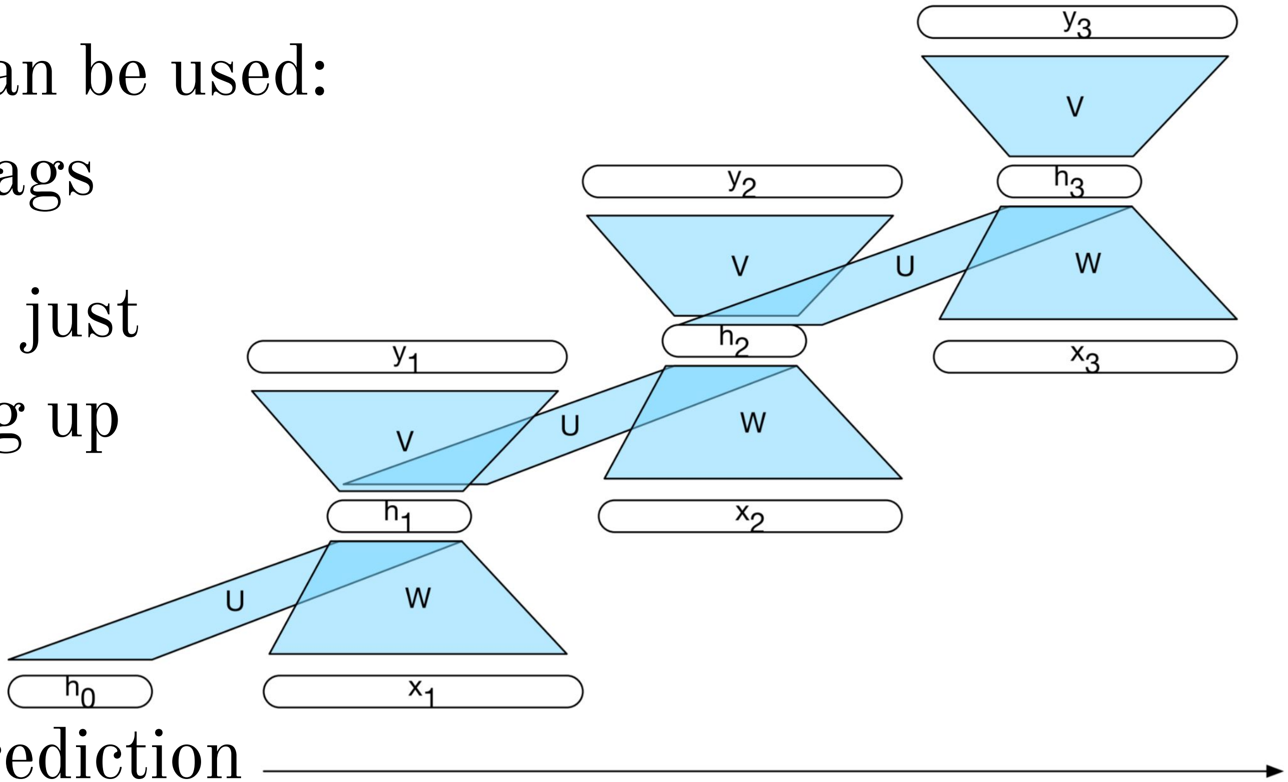


Recurrent Neural Networks - unrolled view

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



Seq2seq 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

[redacted]

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)

Result is “contextual” embeddings

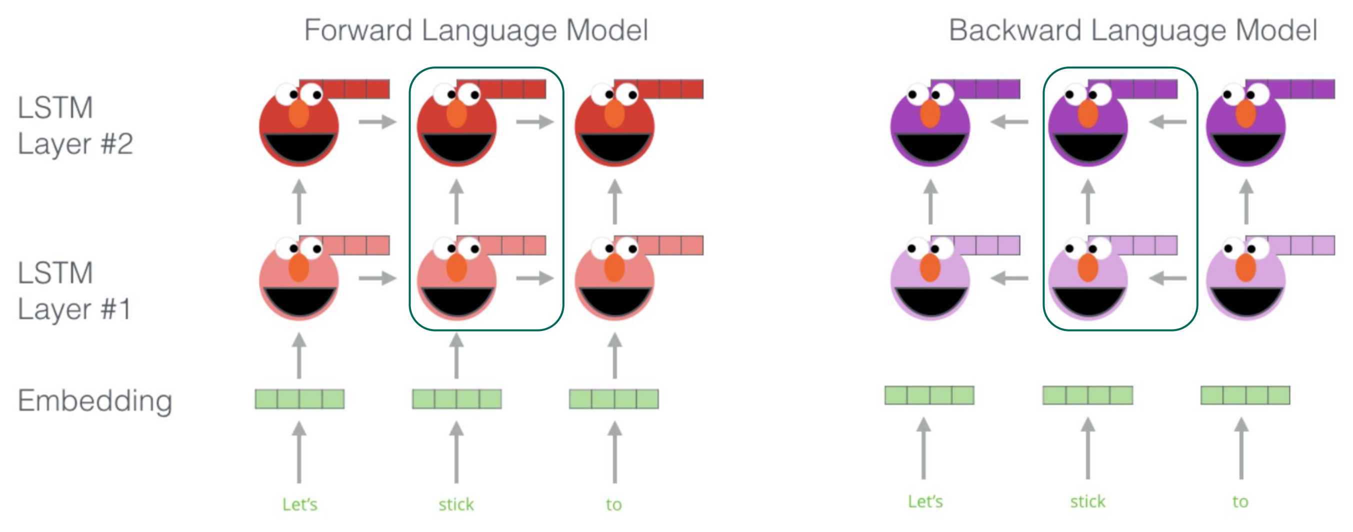


figure from Jay Alammar

The Muppet Parade

BERT and others follow on this idea
with more complex architectures

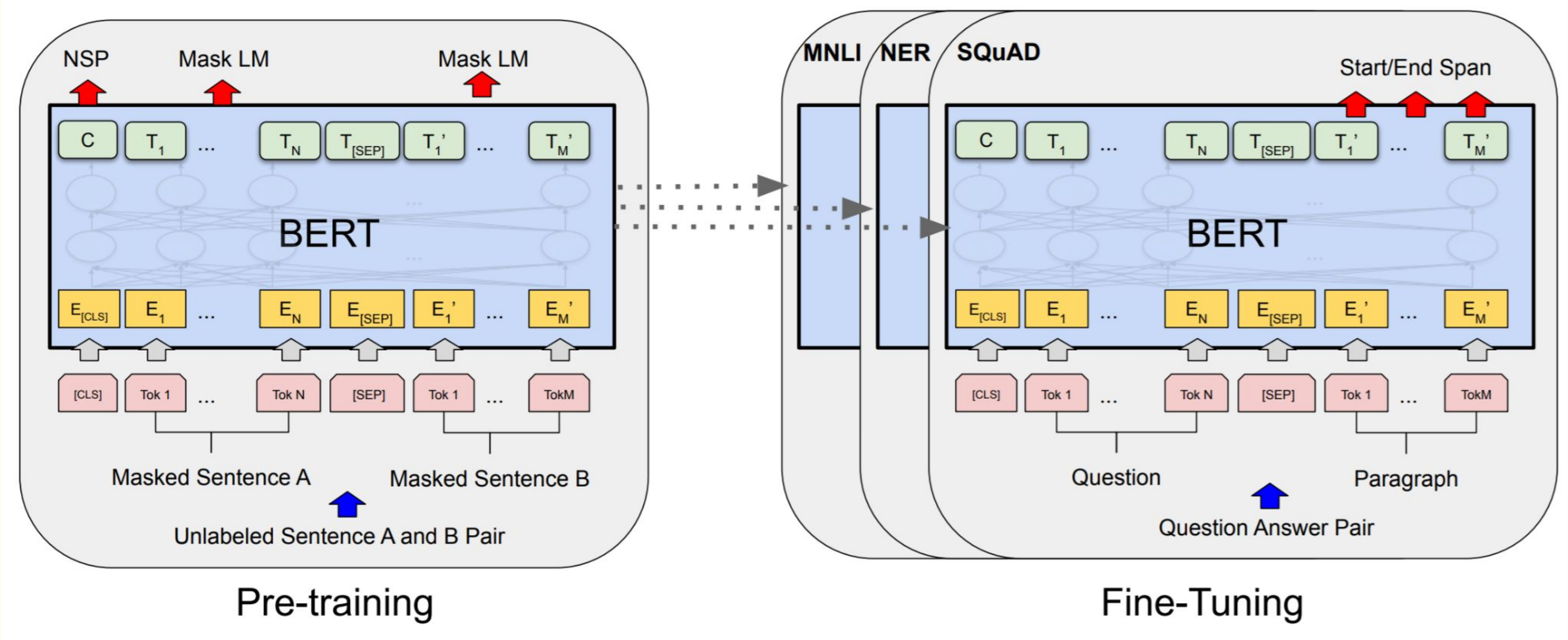
Many layers, complex flow of information

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

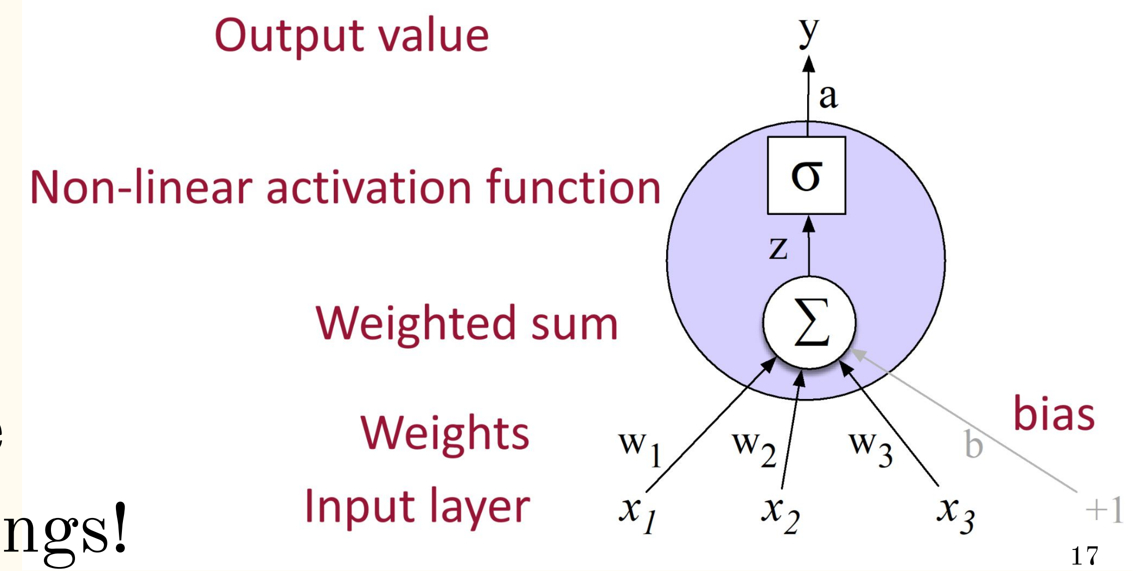


BERT for Classification

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!

Interesting time to be a computational linguist!

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

Key software packages to look into

Applied statistical MT: scikit-learn

Neural Networks: pytorch

Pre-trained BERT-like Models: huggingface

Thank you!!!