Deep Learning for NLP: Feedforward Networks

COMP90042

Natural Language Processing

Lecture 7

Semester 1 2021 Week 4
Jey Han Lau

THE UNIVERSITY OF MELBOURNE

Outline

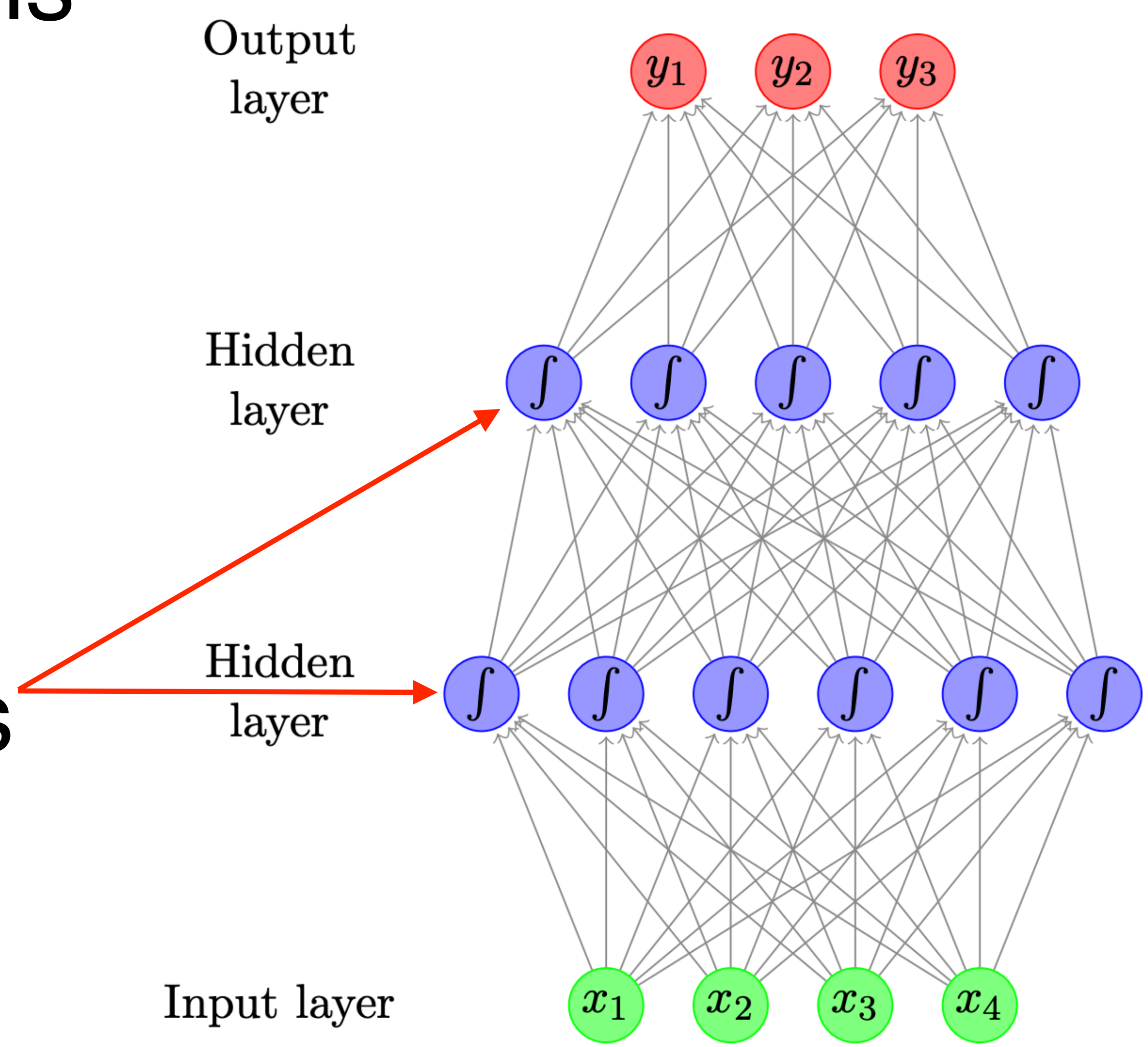
- Feedforward Neural Networks Basics
- Applications in NLP
- Convolutional Networks

Deep Learning

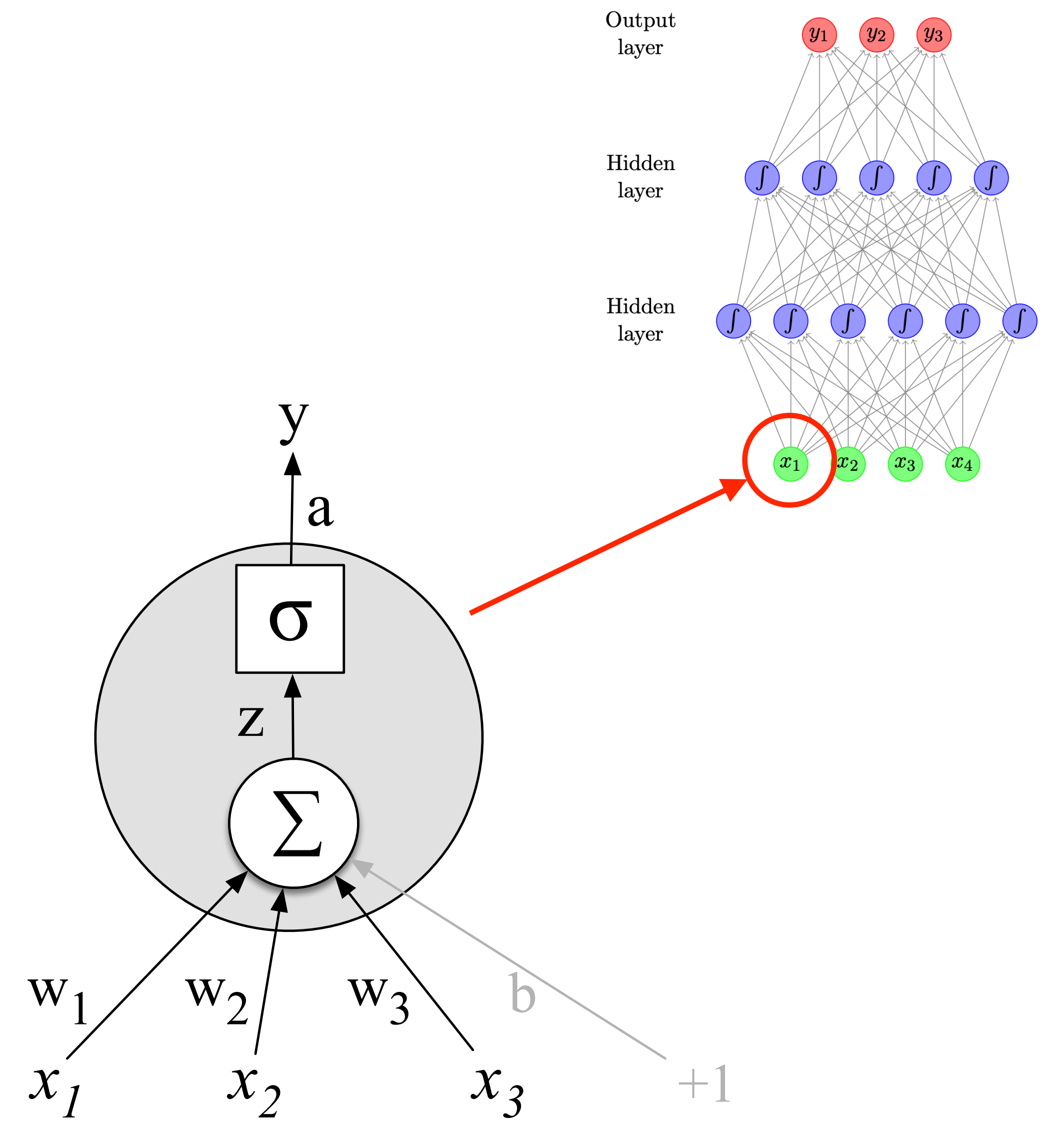
- A branch of machine learning
- Re-branded name for neural networks
- Why deep? Many layers are chained together in modern deep learning models
- Neural networks: historically inspired by the way computation works in the brain
  - Consists of computation units called neurons

Feed-forward NN

- Aka multilayer perceptrons
- Each arrow carries a weight, reflecting its importance
- Certain layers have non-linear activation functions



Neuron



- Each neuron is a function
  - given input x, computes real-value (scalar) h

$$h = \tanh\left(\sum_j w_j x_j + b\right)$$

  - scales input (with weights, w) and adds offset (bias, b)
  - applies non-linear function, such as logistic sigmoid, hyperbolic sigmoid (tanh), or rectified linear unit
  - w and b are **parameters** of the model

Matrix Vector Notation

- Typically have several hidden units, i.e.

$$h_i = \tanh\left(\sum_j w_{ij} x_j + b_i\right)$$

- Each with its own weights (w_i) and bias term (b_i)
- Can be expressed using matrix and vector operators

$$\vec{h} = \tanh\left(W\vec{x} + \vec{b}\right)$$

- Where W is a matrix comprising the weight vectors, and $\vec{b}$ is a vector of all bias terms
- Non-linear function applied element-wise

Output Layer

- Binary classification problem
  - e.g. classify whether a tweet is + or - in sentiment
  - sigmoid activation function
- Multi-class classification problem
  - e.g. native language identification
  - softmax ensures probabilities > 0 and sum to 1

$$\left[\frac{\exp(v_1)}{\sum_i \exp(v_i)}, \frac{\exp(v_2)}{\sum_i \exp(v_i)}, \ldots, \frac{\exp(v_m)}{\sum_i \exp(v_i)}\right]$$

Learning from Data

- How to learn the parameters from data?
- Consider how well the model "fits" the training data, in terms of the probability it assigns to the correct output

$$L = \prod_{i=0}^{m} P(y_i \mid x_i)$$

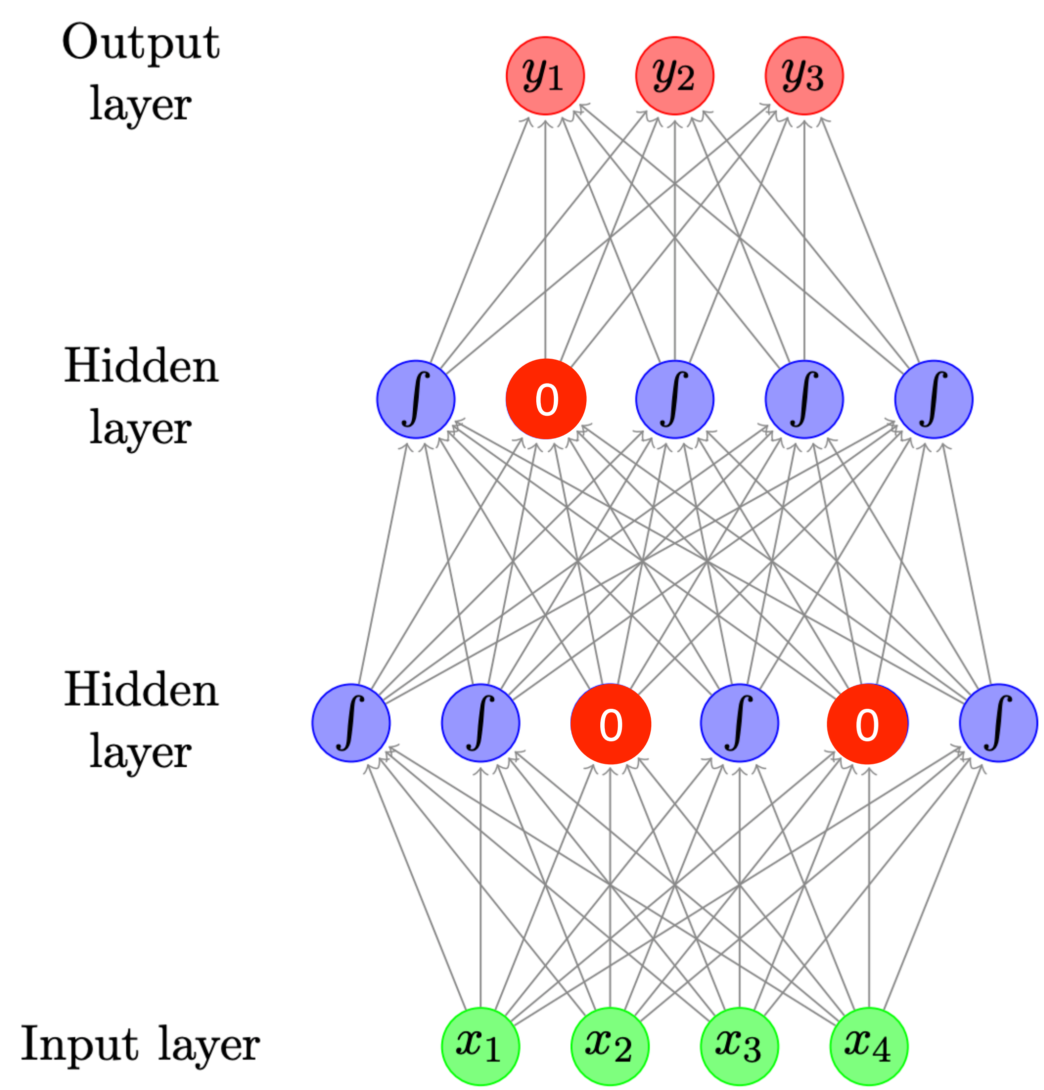
  - want to *maximise* total probability, L
  - equivalently *minimise* -log L with respect to parameters
- Trained using gradient descent
  - tools like *tensorflow, pytorch, dynet* use autodiff to compute gradients automatically

Regularisation

- Have many parameters, overfits easily
- Low bias, high variance
- Regularisation is very very important in NNs
- L1-norm: sum of absolute values of all parameters (W, b, etc)
- L2-norm: sum of squares
- Dropout: randomly zero-out some neurons of a layer

Dropout

- If dropout rate = 0.1, a random 10% of neurons now have 0 values
- Can apply dropout to any layer, but in practice, mostly to the hidden layers

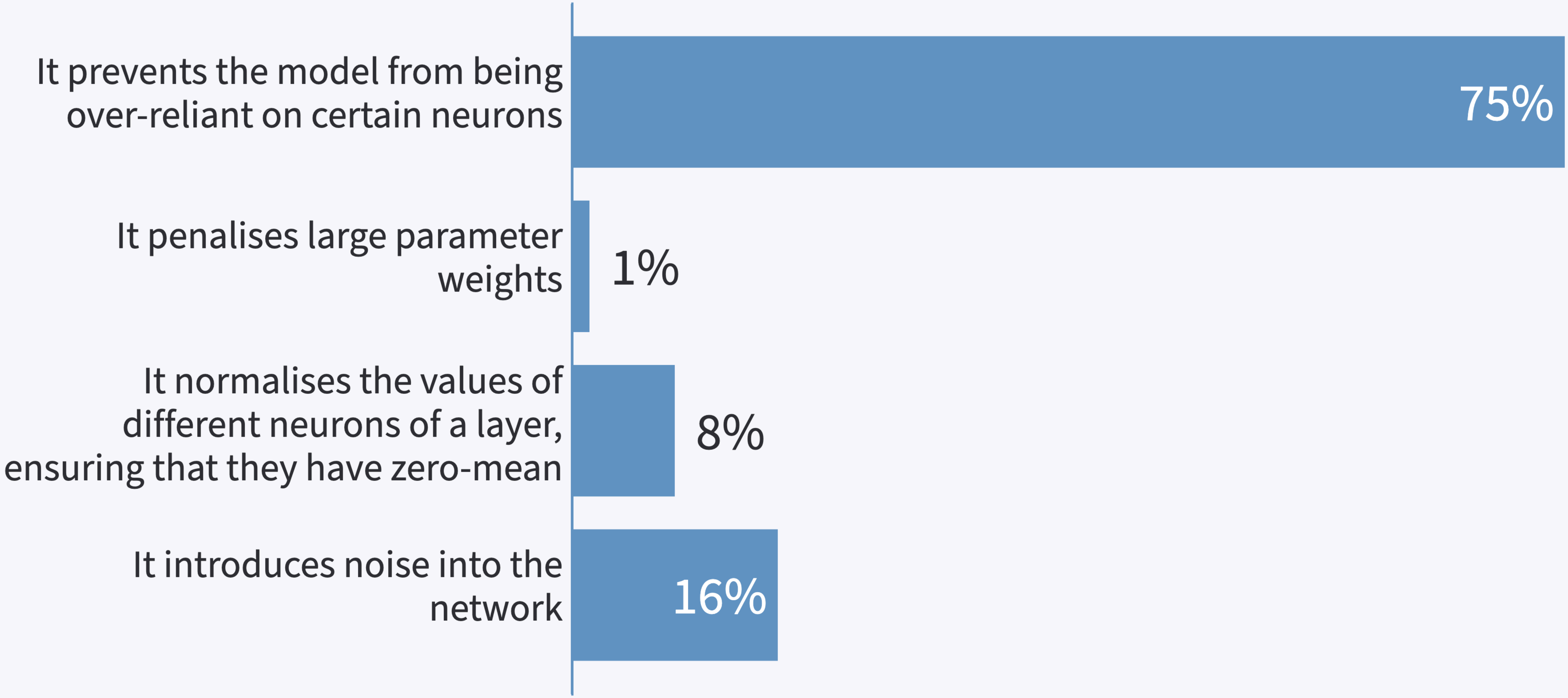


Why Does Dropout Work?

- It prevents the model from being over-reliant on certain neurons
- It penalises large parameter weights
- It normalises the values of different neurons of a layer, ensuring that they have zero-mean
- It introduces noise into the network

PollEv.com/jeyhanlau569

Why Does Dropout Work?



Applications in NLP

Topic Classification

- Given a document, classify it into a predefined set of topics (e.g. economy, politics, sports)
- Input: bag-of-words

	love	cat	dog	doctor
doc 1	0	2	3	0
doc 2	2	0	2	0
doc 3	0	0	0	4
doc 4	3	0	0	2

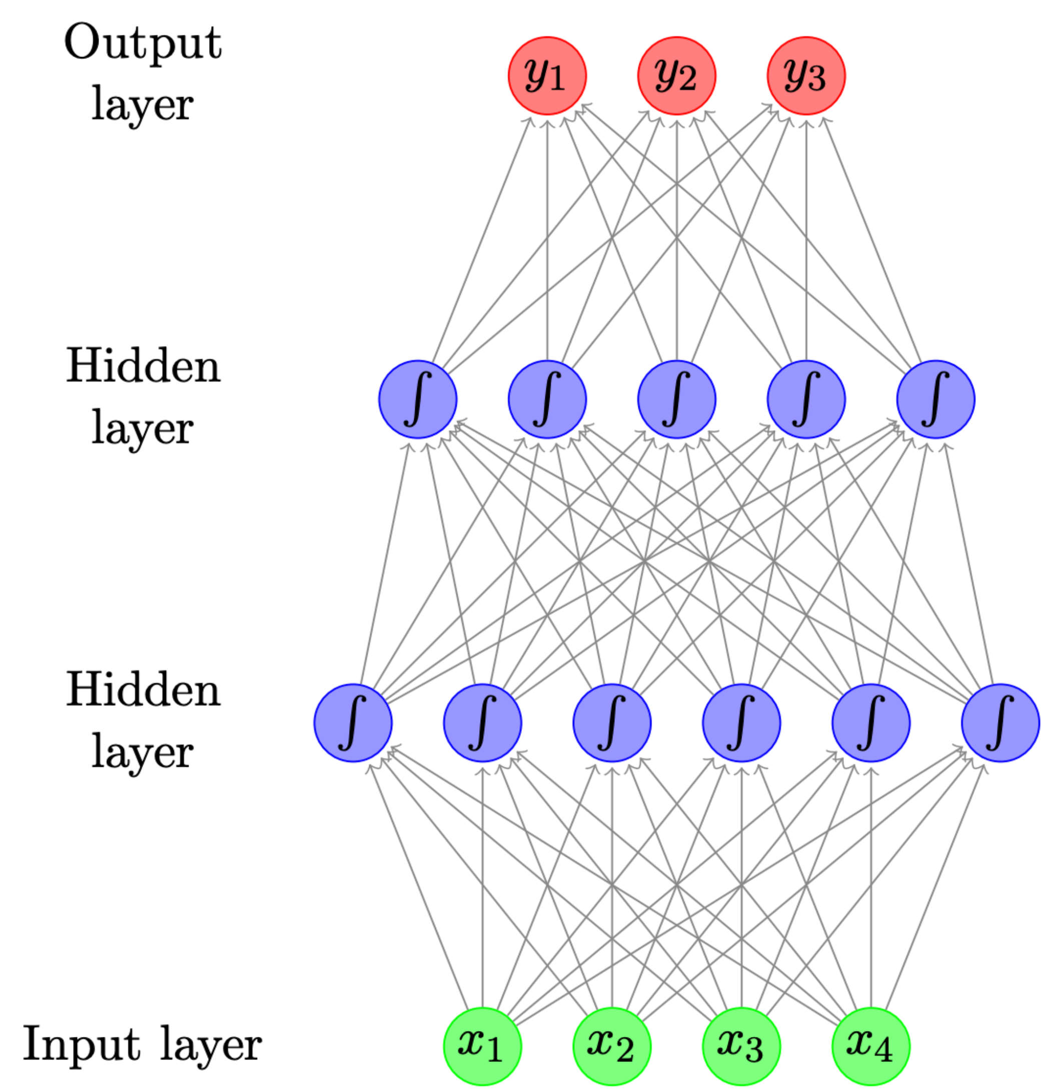
Topic Classification - Training

$$\overrightarrow{h_1} = \tanh\left(W_1 \overrightarrow{x} + \overrightarrow{b_1}\right)$$

$$\overrightarrow{h_2} = \tanh\left(W_2 \overrightarrow{h_1} + \overrightarrow{b_2}\right)$$

$$\overrightarrow{y} = \text{softmax}\left(W_3 \overrightarrow{h_2}\right)$$

- Randomly initialise W and b
- $\overrightarrow{x}$ = [0, 2, 3, 0]
- $\overrightarrow{y}$ = [0.1, 0.6, 0.3]: probability distribution over C_1, C_2, C_3
- $L = -log(0.1)$ if true label is C_1



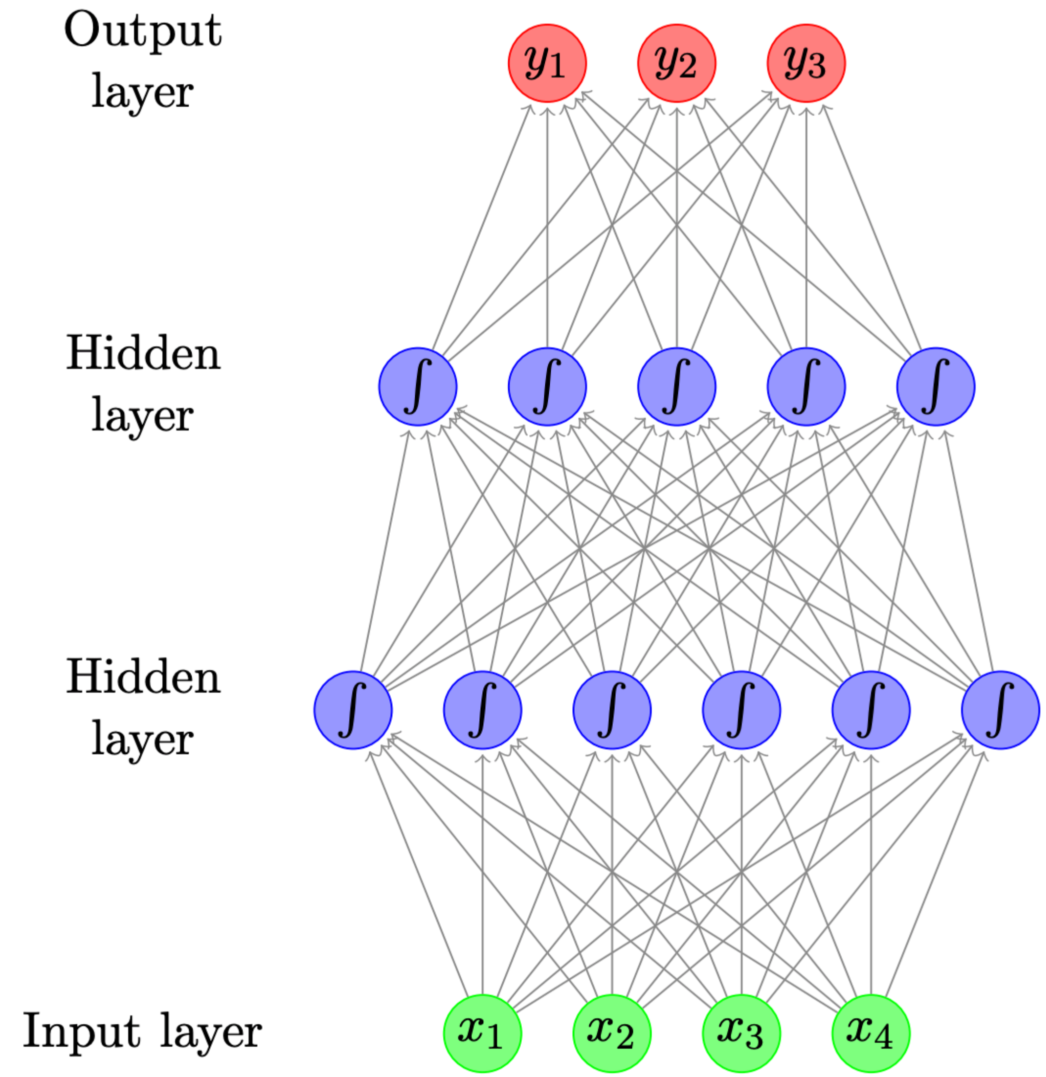
Topic Classification - Prediction

$$\vec{h}_1 = \tanh\left(W_1\vec{x} + \vec{b}_1\right)$$

$$\vec{h}_2 = \tanh\left(W_2\vec{h}_1 + \vec{b}_2\right)$$

$$\vec{y} = \text{softmax}\left(W_3\vec{h}_2\right)$$

- $\vec{x}$ = [1, 3, 5, 0] (test document)
- $\vec{y}$ = [0.2, 0.1, 0.7]
- Predicted class = C_3



Topic Classification - Improvements

- + Bag of bigrams as input
- Preprocess text to lemmatise words and remove stopwords
- Instead of raw counts, we can weight words using TF-IDF or indicators (0 or 1 depending on presence of words)

Language Model Revisited

- Assign a probability to a sequence of words
- Framed as "sliding a window" over the sentence, predicting each word from finite context
  E.g., $n = 3$, a trigram model

$$P(w_1, w_2, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-2}, w_{i-1})$$

- Training involves collecting frequency counts
  - Difficulty with rare events → smoothing

Language Models as Classifiers

LMs can be considered simple classifiers, e.g. for a trigram model:

$$P(w_i \mid w_{i-2} = \text{salt}, w_{i-1} = \text{and})$$

classifies the likely next word in a sequence, given "salt" and "and".

Feed-forward NN Language Model

- Use neural network as a classifier to model

$$P(w_i \,|\, w_{i-2} = \text{salt}, w_{i-1} = \text{and})$$

- Input features = the previous two words
- Output class = the next word
- How to represent words? **Embeddings**

0.1	-1.5	2.3	0.9	-3.2	2.5	1.1

Word Embeddings

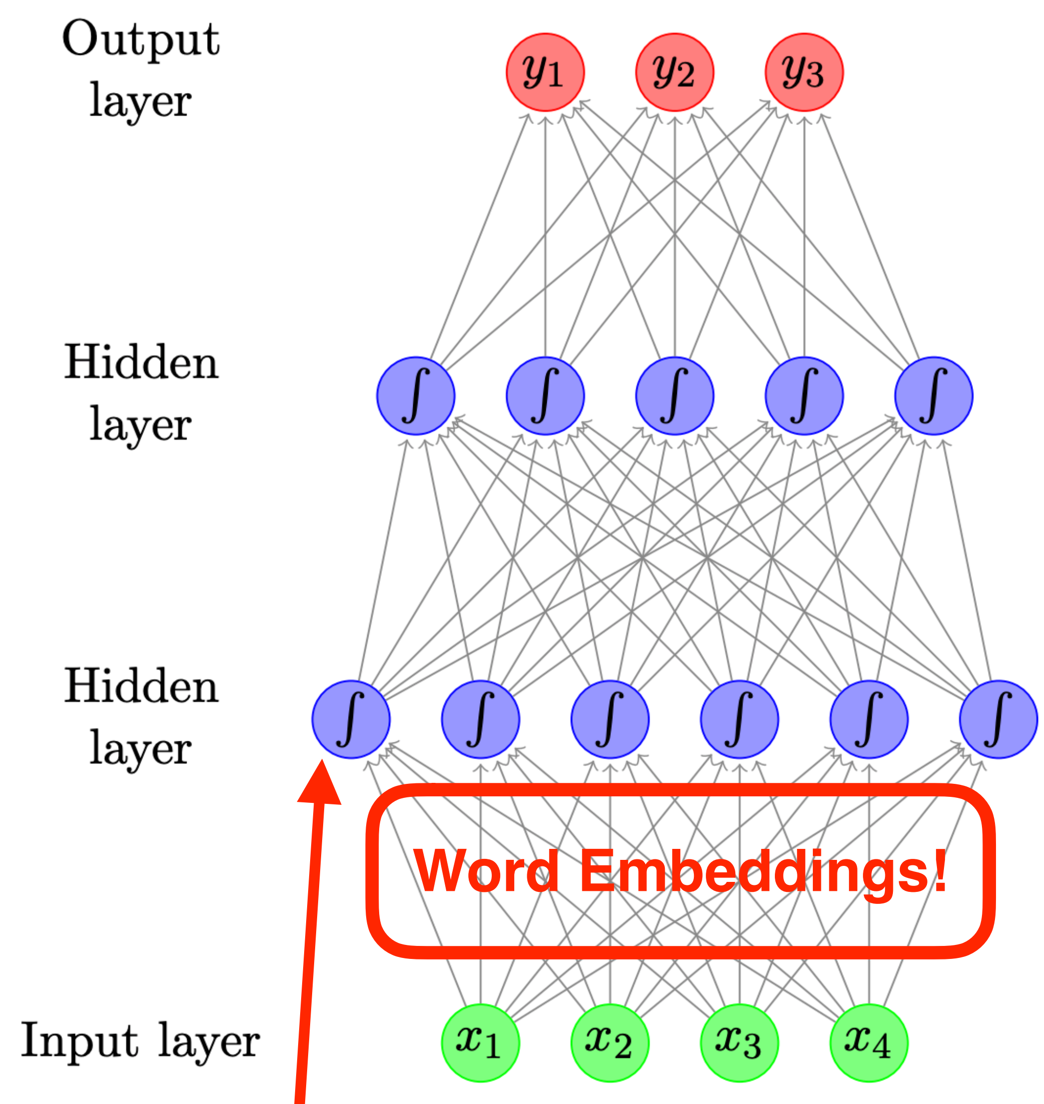
- Maps discrete word symbols to continuous vectors in a relatively low dimensional space
- Word embeddings allow the model to capture similarity between words
  - *dog* vs. *cat*
  - *walking* vs. *running*

Topic Classification

$$\overrightarrow{h_1} = \tanh\left(W_1\vec{x} + \vec{b_1}\right)$$

$$\overrightarrow{h_2} = \tanh\left(W_2\overrightarrow{h_1} + \vec{b_2}\right)$$

$$\vec{y} = \mathrm{softmax}\left(W_3\overrightarrow{h_2}\right)$$



First layer = sum of input word embeddings

Training a FFNN LM

- $P(w_i = \text{grass} \mid w_{i-3} = \text{a}, w_{i-2} = \text{cow}, w_{i-1} = \text{eats})$
- Lookup word embeddings (W_1) for *a*, *cow* and *eats*

a	grass	eats	hunts	cow
0.9	0.2	-3.3	-0.1	-0.5
0.2	-2.3	0.6	-1.5	1.2
-0.6	0.8	1.1	0.3	-2.4
1.5	0.8	0.1	2.5	0.4

- Concatenate them and feed it to the network

$$\vec{x} = \vec{v}_a \oplus \vec{v}_{cow} \oplus \vec{v}_{eats}$$

$$\vec{h} = \tanh(W_2 \vec{x} + \vec{b}_1)$$

$$\vec{y} = \text{softmax}(W_3 \vec{h})$$

Training a FFNN LM

- $\overrightarrow{y}$ gives the probability distribution over all words in the vocabulary

0.01	0.80	0.05	0.10	0.04
rabbit	grass	eats	hunts	cow

$$P(w_i = \text{grass} \mid w_{i-3} = \text{a}, w_{i-2} = \text{cow}, w_{i-1} = \text{eats}) = 0.8$$

- $L = -\log(0.8)$
- Most parameters are in the word embeddings W_1 (size = $d_1 \times |V|$) and the output embeddings W_3 (size = $|V| \times d_3$)

Input and Output Word Embeddings

- $P(w_i = \text{grass} \,|\, w_{i-3} = \text{a}, w_{i-2} = \text{cow}, w_{i-1} = \text{eats})$
- Lookup word embeddings (W_1) for *a*, *cow* and *eats*

a	grass	eats	hunts	cow
0.9	0.2	-3.3	-0.1	-0.5
0.2	-2.3	0.6	-1.5	1.2
-0.6	0.8	1.1	0.3	-2.4
1.5	0.8	0.1	2.5	0.4

Word embeddings W_1
$d_1 \times |V|$

- Concatenate them and feed it to the network

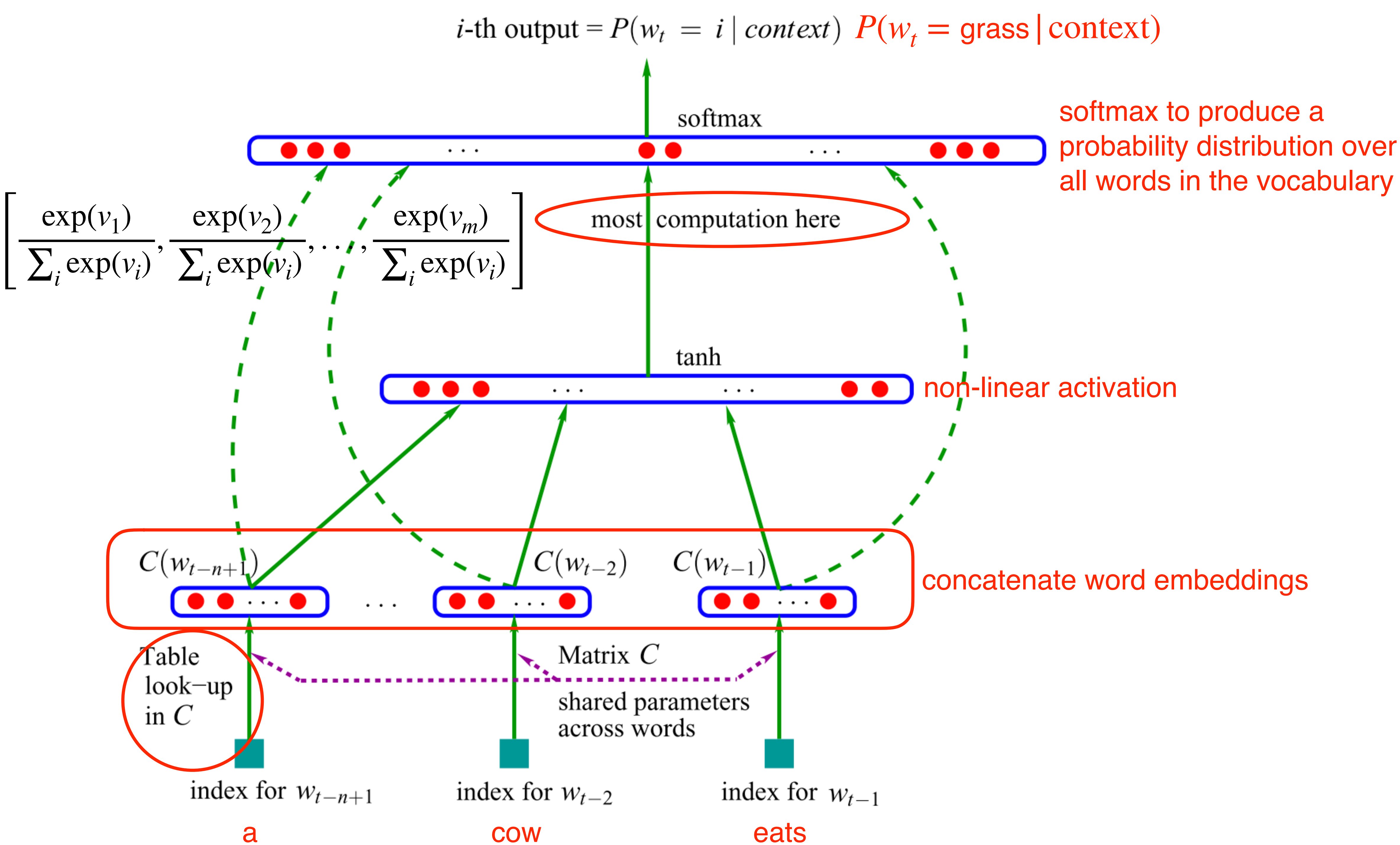
$$\vec{x} = \vec{v}_a \oplus \vec{v}_{cow} \oplus \vec{v}_{eats}$$

$$\vec{h} = \tanh(W_2 \vec{x} + \vec{b}_1)$$

$$\vec{y} = \text{softmax}(W_3 \vec{h})$$

Output word embeddings W_3
$|V| \times d_3$

Language Model: Architecture



Bengio et al, 2003

Advantages of FFNN LM

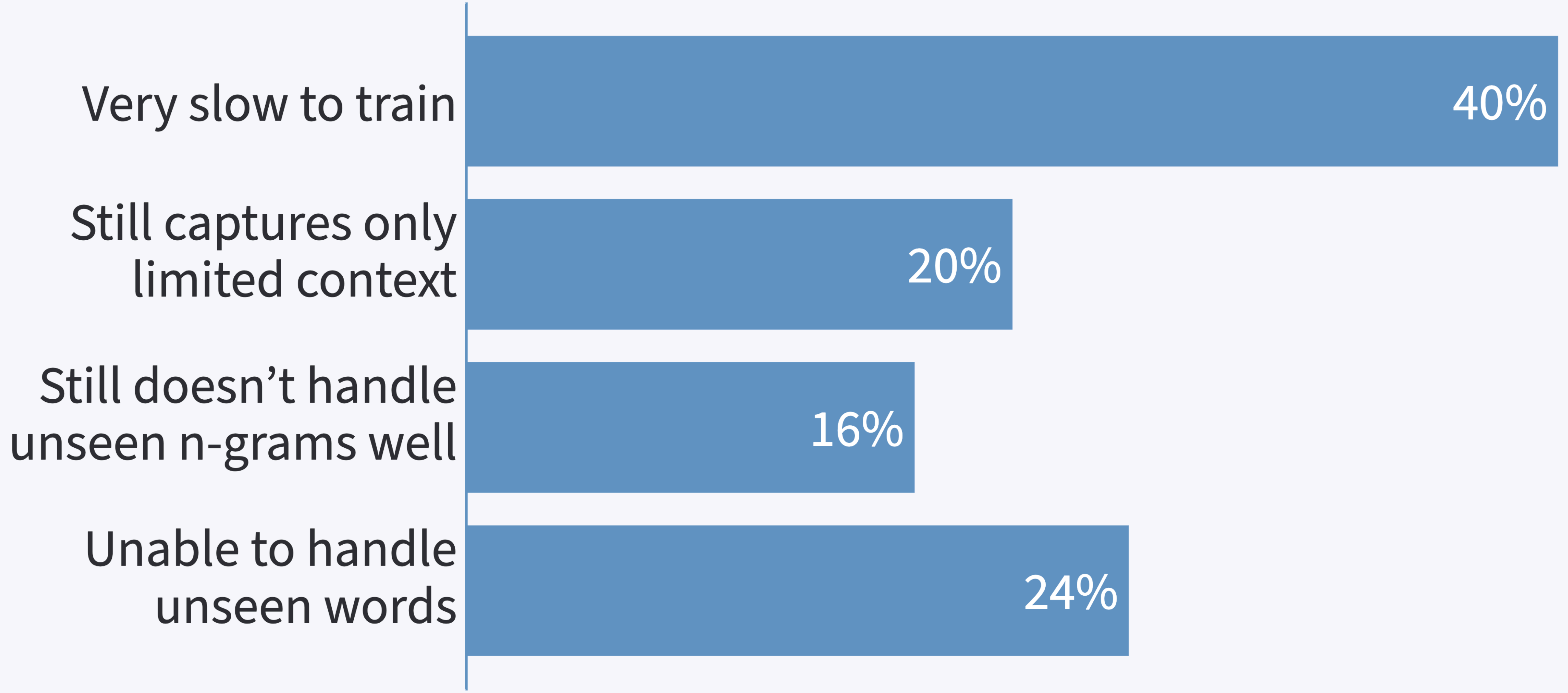
- Count-based *N*-gram models (lecture 3)
  - cheap to train (just collect counts)
  - problems with sparsity and scaling to larger contexts
  - don't adequately capture properties of words (grammatical and semantic similarity), e.g., film vs movie
- FFNN *N*-gram models
  - automatically capture word properties, leading to more robust estimates

What Are The Disadvantages of Feedforward NN Language Model?

- Very slow to train
- Captures only limited context
- Unable to handle unseen *n*-grams
- Unable to handle unseen words

PollEv.com/jeyhanlau569

What Are The Limitations of Feedforward NN Language Model?



POS Tagging

- POS tagging can also be framed as classification:

  $$P(t_i \mid w_{i-1} = \text{cow}, w_i = \text{eats})$$

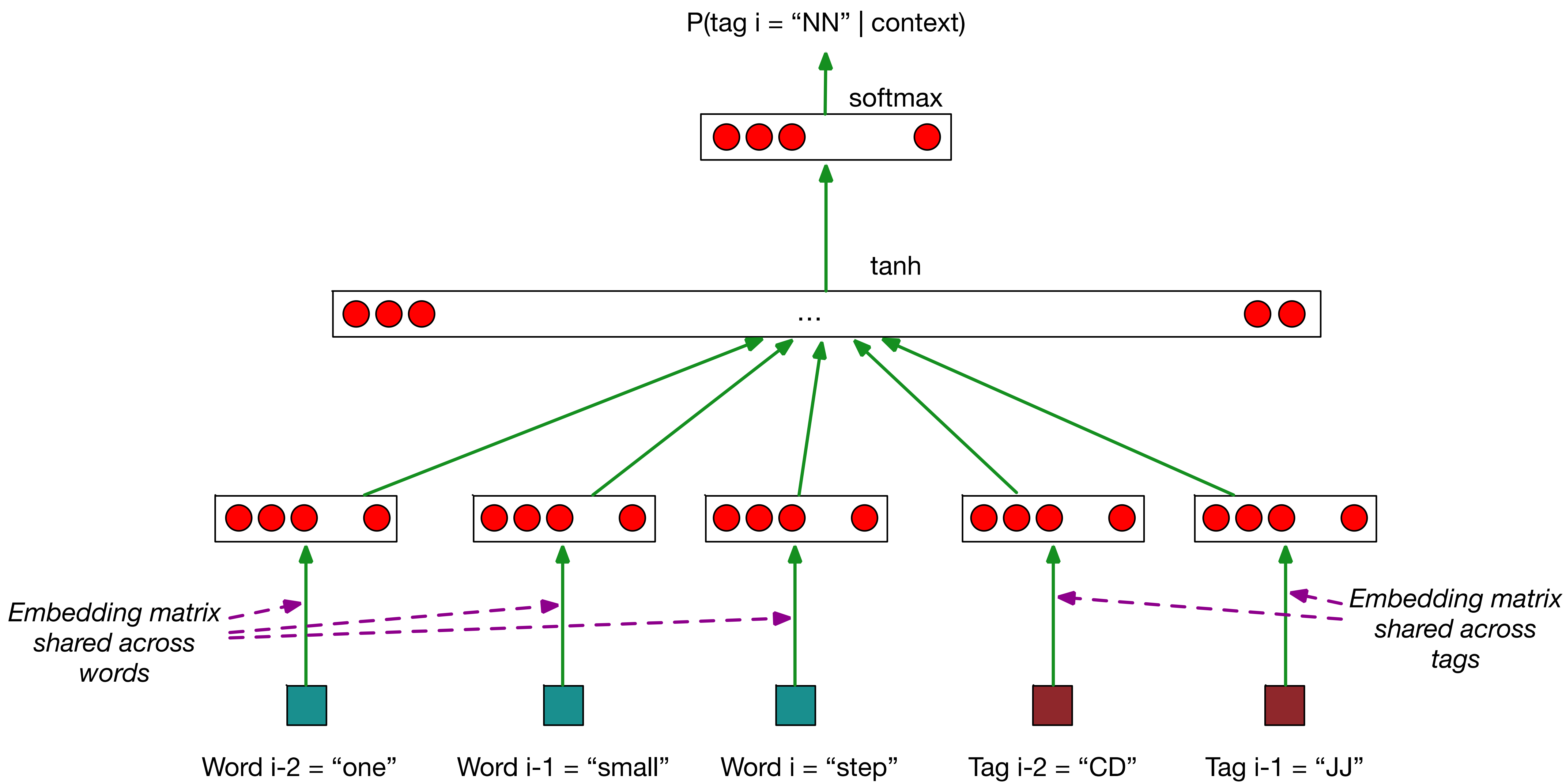
  - classifies the likely POS tag for "eats".
- FFNN LM architecture can be adapted to the task directly

Feed-forward NN for Tagging

- Inputs:
  - recent words w_{i-2}, w_{i-1}, w_i
  - recent tags t_{i-2}, t_{i-1}
- And outputs: current tag t_i
- Frame as neural network with
  - 5 inputs: 3 x word embeddings and 2 x tag embeddings
  - 1 output: vector of size $|T|$, using softmax
- Train to minimise

$$-\sum_i \log P(t_i \mid w_{i-2}, w_{i-1}, w_i, t_{i-2}, t_{i-1})$$

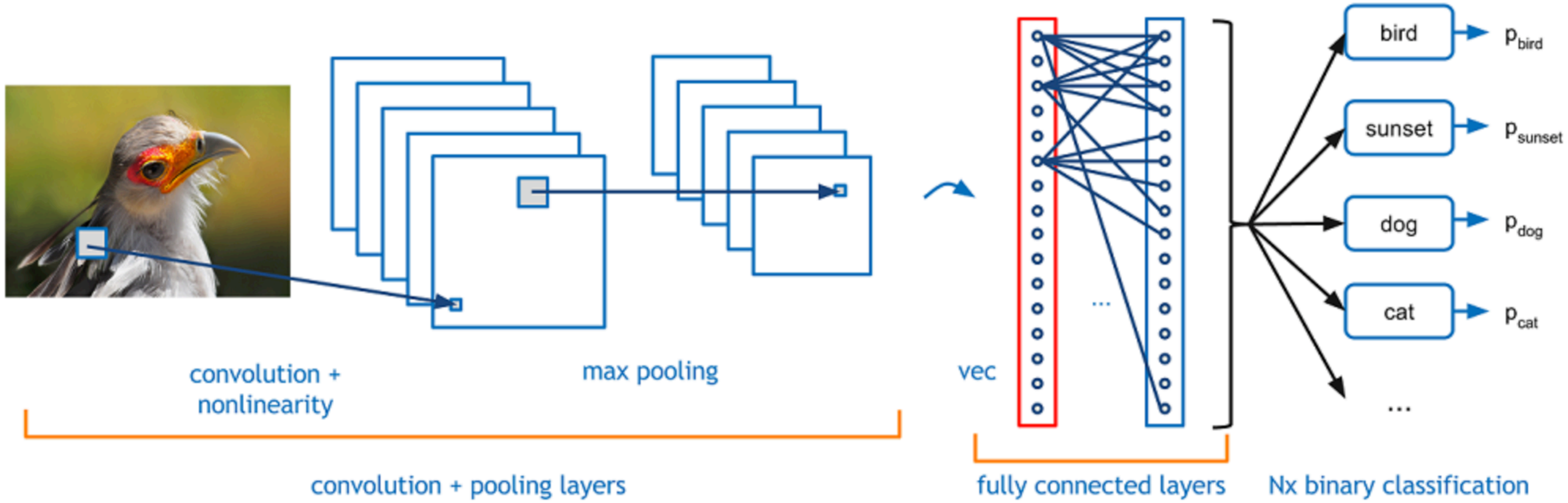
FFNN for Tagging



Convolutional Networks

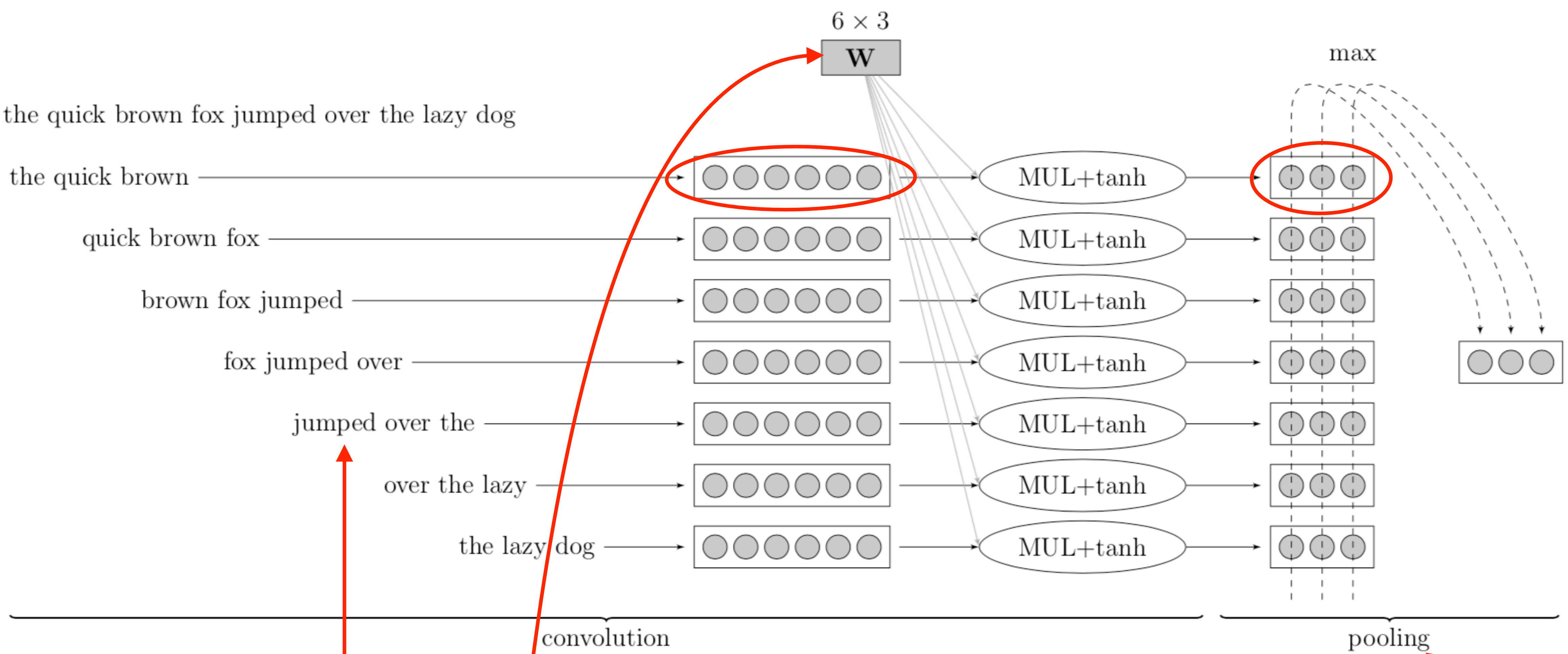
Convolutional Networks

- Commonly used in computer vision
- Identify indicative local predictors
- Combine them to produce a fixed-size representation



https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Networks for NLP



- Sliding window (e.g. 3 words) over sequence
- W = convolution filter (linear transformation+tanh)
- max-pool to produce a fixed-size representation

Final Words

- Pros
  - Excellent performance
  - Less hand-engineering of features
  - Flexible — customised architecture for different tasks
- Cons
  - Much slower than classical ML models… needs GPU
  - Lots of parameters due to vocabulary size
  - Data hungry, not so good on tiny data sets
    - Pre-training on big corpora helps

Readings

- Feed-forward network: G15, section 4; JM Ch. 7.3
- Convolutional network: G15, section 9