Deep Learning for NLP: Feedforward Networks

COMP90042

Natural Language Processing

Lecture 7

THE UNIVERSITY OF MELBOURNE

Corrections on L3: page 21/22

Absolute Discounting

Context = *alleged*
- 5 observed n-grams
- 2 unobserved n-grams

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$		Discounting, $d = 0.1$	
			effective counts	smoothed probability	effective counts	smoothed probability
impropriety	8	0.4	7.826	0.391	7.9	0.395
offense	5	0.25	4.928	0.246	4.9	0.245
damage	4	0.2	3.961	0.198	3.9	0.195
deficiencies	2	0.1	2.029	0.101	1.9	0.095
outbreak	1	0.05	1.063	0.053	0.9	0.045
infirmity	0	0	0.097	0.005	0.25	0.013
cephalopods	0	0	0.097	0.005	0.25	0.013

8 - 0.1

(0.1 x 5) / 2

total amount of discounted probability mass
(0.1 x 5) / 20

2

Corrections on L3: page 21/22

Backoff

- Absolute discounting redistributes the probability mass **equally** for all unseen n-grams

- Katz Backoff: redistributes the mass based on a **lower order** model (e.g. unigram)

$$P_{katz}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1}, w_i) - D}{C(w_{i-1})}, & \text{if } C(w_{i-1}, w_i) > 0 \\ \alpha(w_{i-1}) \times \frac{P(w_i)}{\sum_{w_j : C(w_{i-1}, w_j) = 0} P(w_j)}, & \text{otherwise} \end{cases}$$

unigram probability for w_i

sum unigram probabilities for all words that do not co-occur with context w_{i-1}

the amount of probability mass that has been discounted for context w_{i-1}
((0.1 x 5) / 20 in previous slide)

Deep Learning

- A branch of machine learning

- Re-branded name for neural networks

- Neural networks: historically inspired by the way computation works in the brain

  - Consists of computation units called neurons

- Why deep? Many layers are chained together in modern deep learning models

Feed-forward NN

- Aka multilayer perceptrons

- Each arrow carries a weight, reflecting its importance

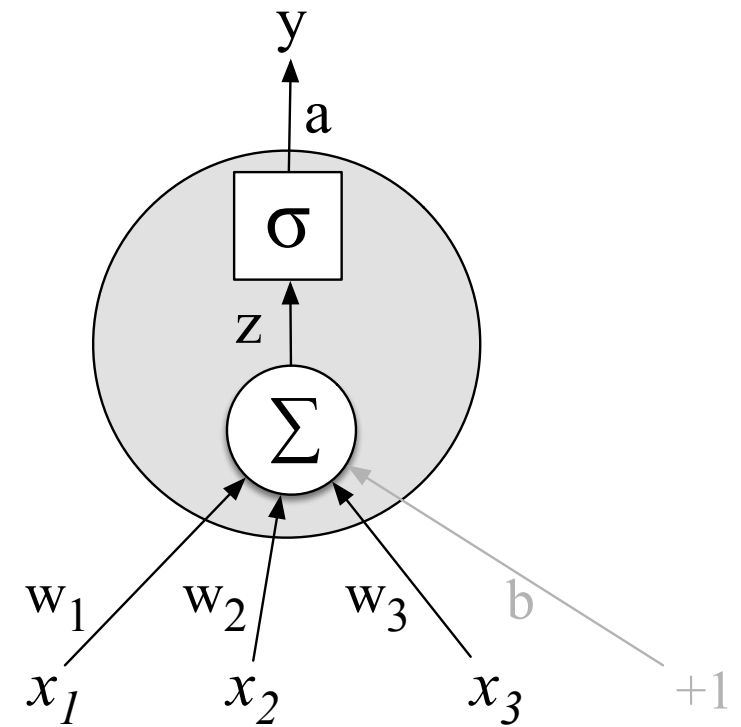- Sigmoid function represents a non-linear function
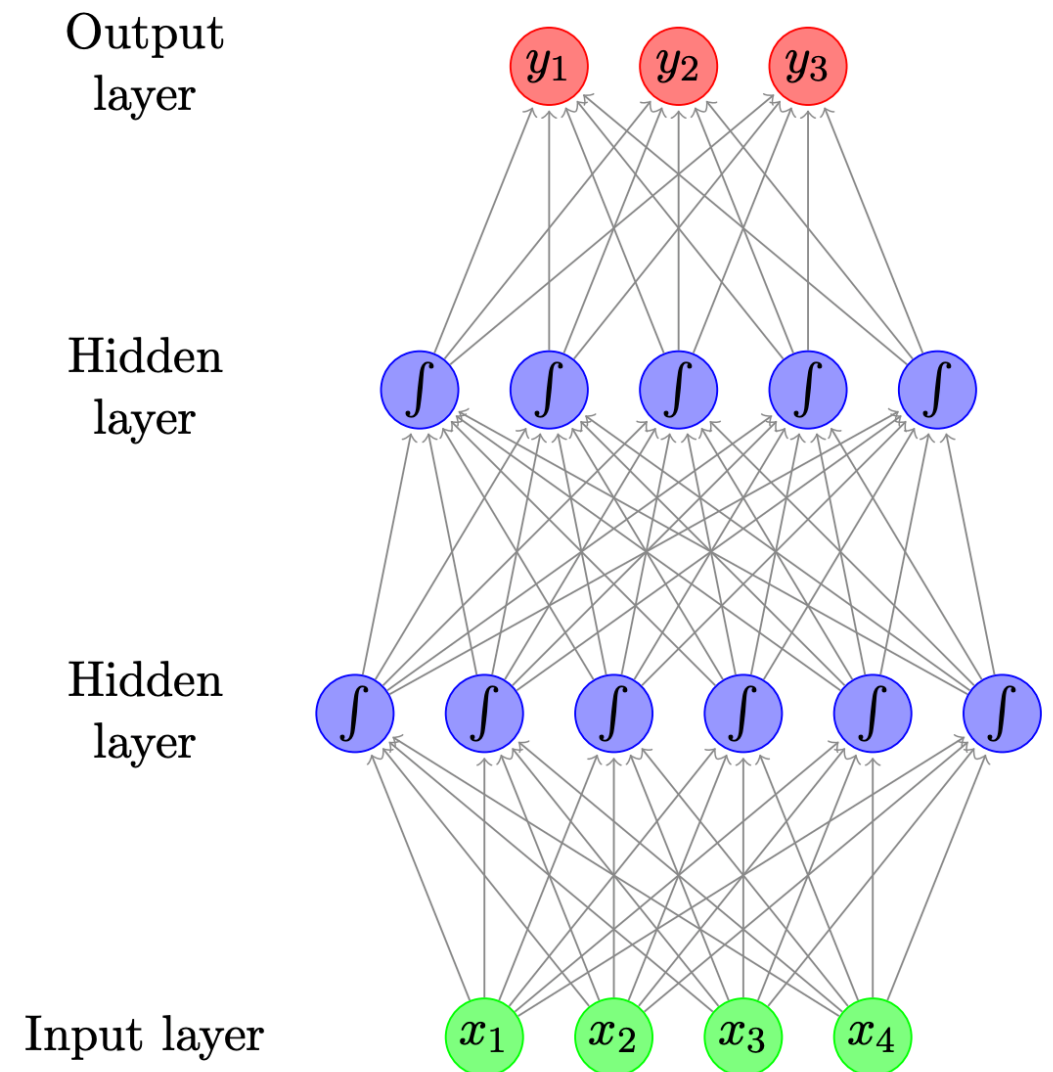
NN Units

- Each "unit" 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

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 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 positive or negative in sentiment):

  ‣ sigmoid activation function (aka logistic function)

- Multi-class classification problem (e.g. classify the topics of a document)

  ‣ softmax ensures probabilities > 0 and sum to 1

$$\left[\frac{\exp(v_1)}{\sum_i \exp(v_i)}, \quad \frac{\exp(v_2)}{\sum_i \exp(v_i)}, \quad \cdots \quad \frac{\exp(v_m)}{\sum_i \exp(v_i)} \right]$$
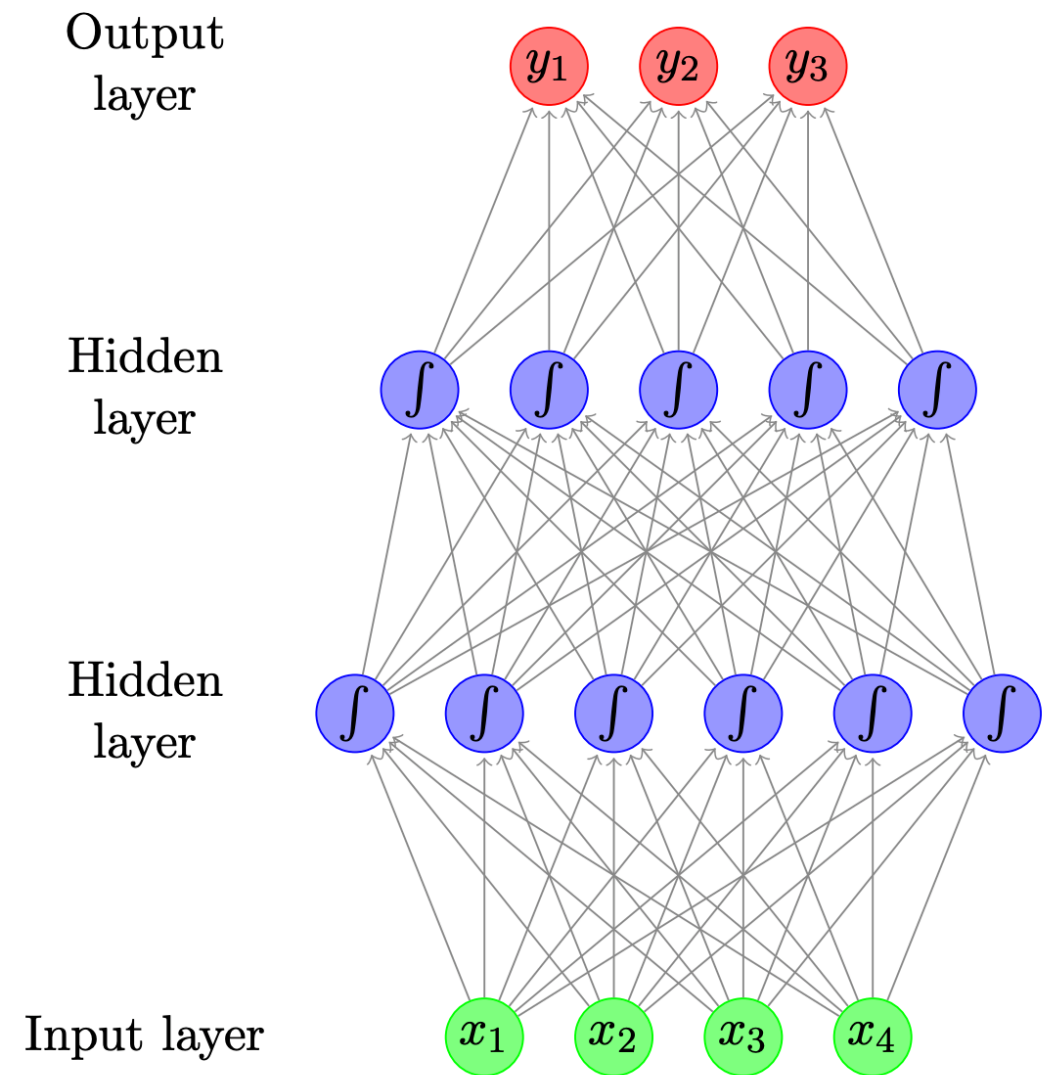
Feed-forward NN

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

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

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

- Matrices and biases = parameters of the model

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

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

   ‣ 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

Topic Classification

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

- Input: bag-of-words

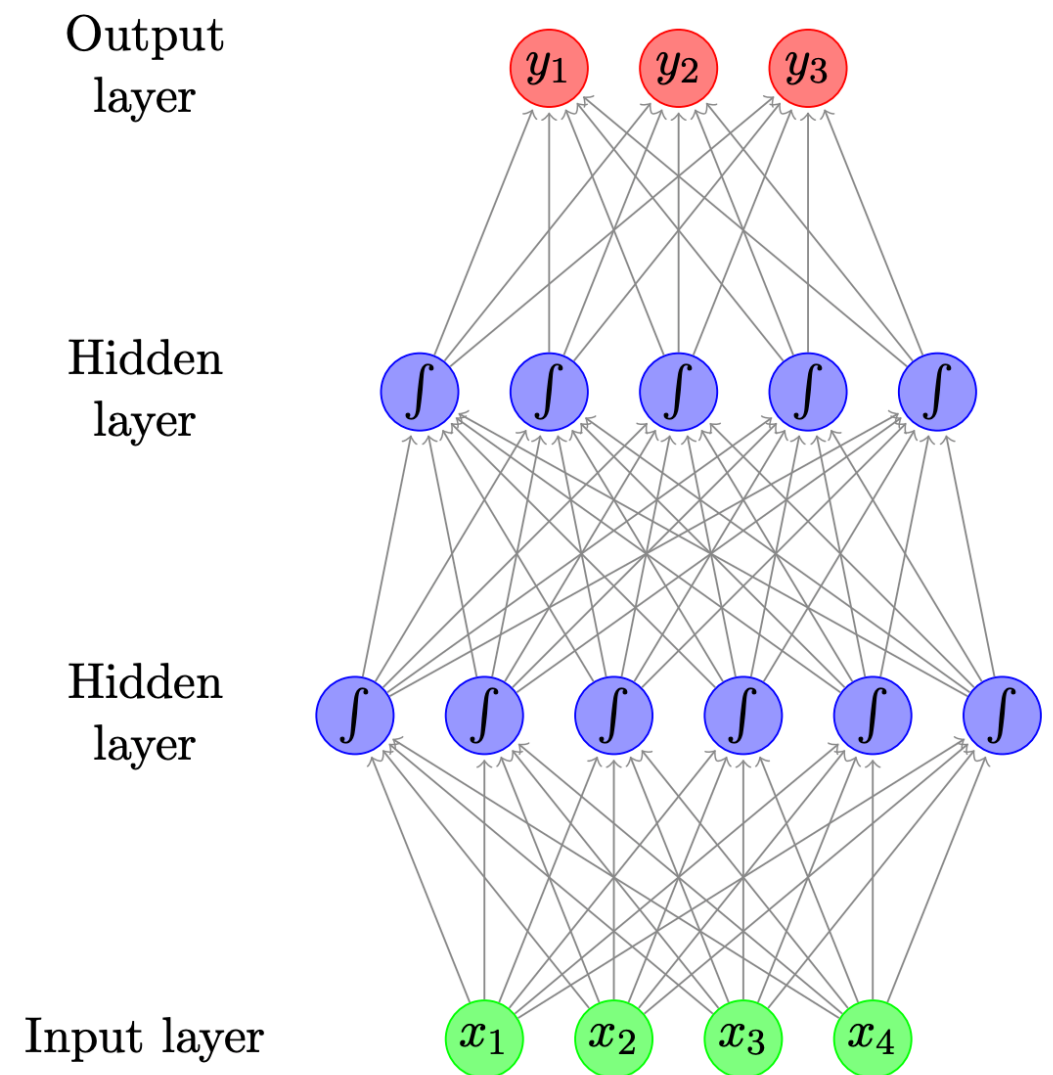|	love	cat	dog	doctor
doc 1	0	2	3	0
doc 2	2	0	2	0
doc 3	0	0	0	4
doc 4	3	0	0	2

Topic Classification

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

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

$$\vec{y} = \text{softmax}(W_3\vec{h_2})$$

- x = [0, 2, 3, 0] for the first document

- y = [0.1, 0.6, 0.3]: probability distribution over the 3 classes

Output layer: y_1 y_2 y_3

Hidden layer

Hidden layer

Input layer: x_1 x_2 x_3 x_4

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)

Authorship Attribution

- Given a document, infer the identity of its author or characteristics of the author (e.g. gender, age, native language)

- Stylistic properties of text are more important than content words in this task
  - ‣ POS tags and function words (e.g. *on*, *of*, *the*, *and*)

- Good approximation of function words: top-300 most frequent words in a large corpus

- Input: bag of function words, bag of POS tags, bag of POS bigrams, trigrams

- Word weighting: density (e.g. ratio between no. of function words and content words in a window of text)

- Other features: distribution of distances between consecutive function words

Language model (Recap)

- 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

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

- Training (estimation) from frequency counts
  ‣ Difficulty with rare events → smoothing

Language Models as Classifiers

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

$$P\left(w_i \middle| w_{i-2} = \text{``}cow\text{''},\ \text{w}_{i-1} = \text{``}eats\text{''}\right)$$

classifies the likely next word in a sequence.

Feed-forward NN Language Model

- Use neural network as a classifier to model

$$P\left(w_i \,\middle|\, w_{i-2} = \text{``}cow\text{''}, \mathrm{w}_{i-1} = \text{``}eats\text{''}\right)$$

  ‣ input features = the previous two words

  ‣ output class = the next word

- How to represent words? **Embeddings**

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*
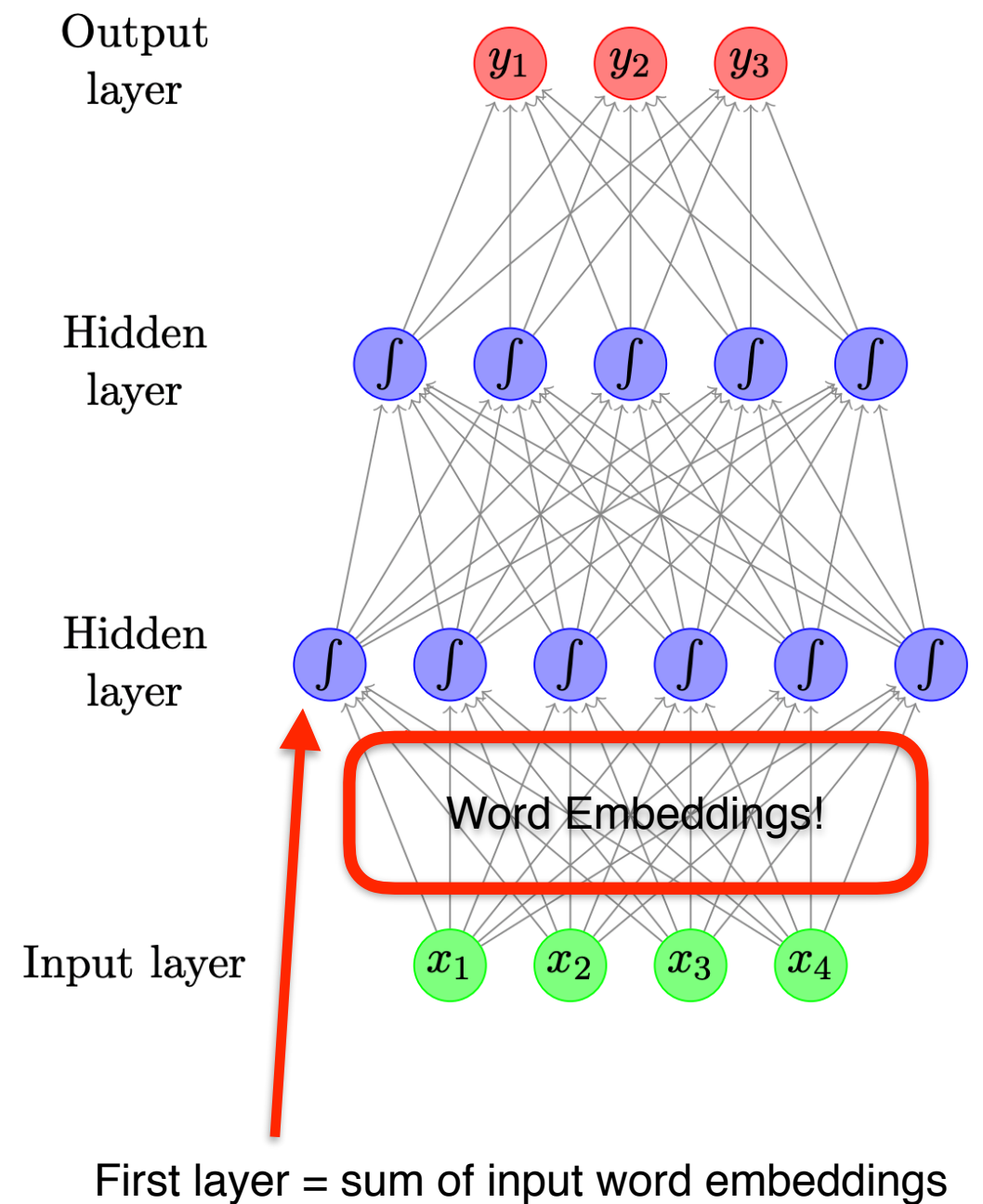
- Alleviates data-sparsity problems

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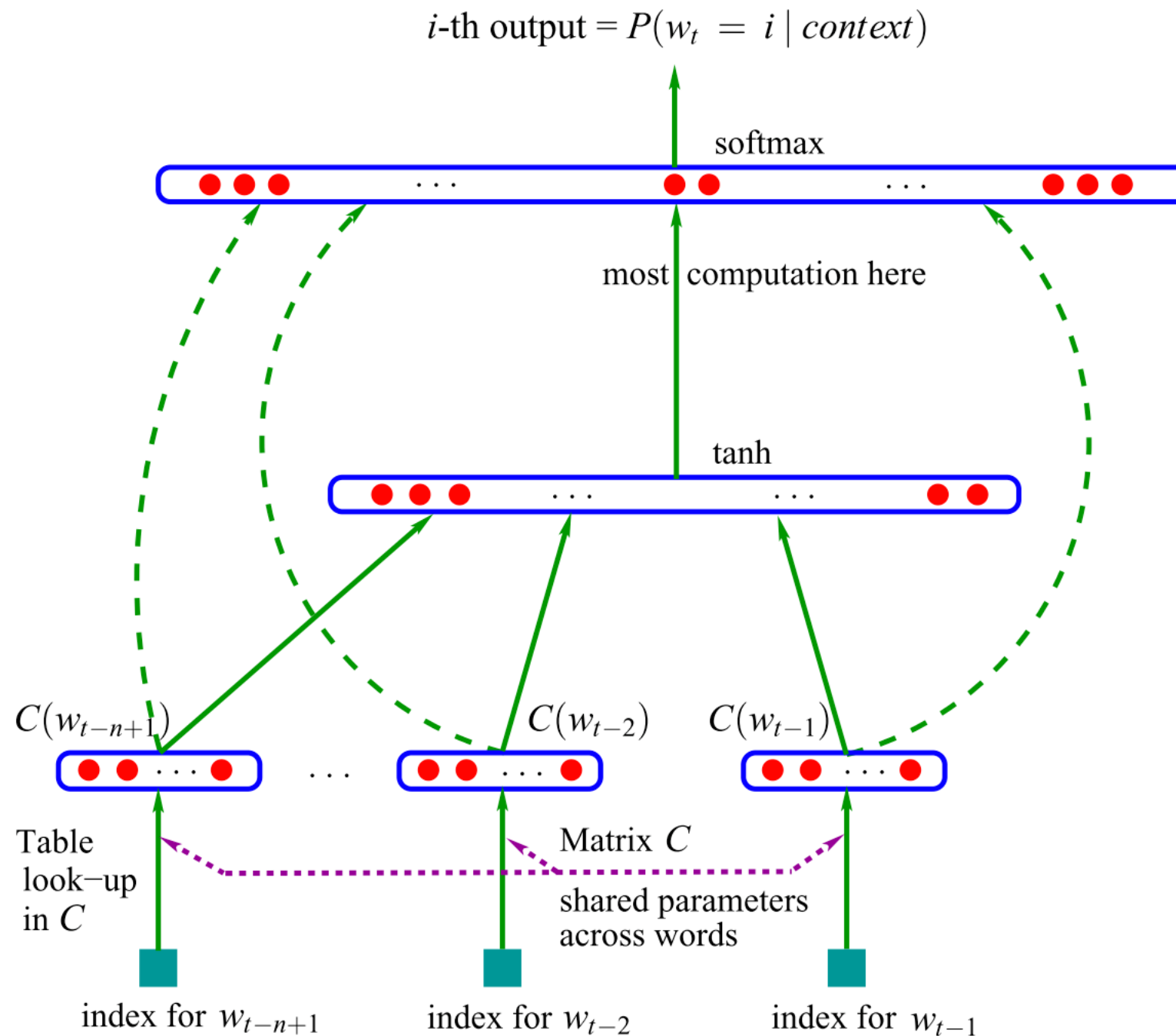
$$\vec{h_1} = \tanh\left(W_1\vec{x} + \vec{b_1}\right)$$

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

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

- x = [0, 2, 3, 0] for the first document

- y = [0.1, 0.6, 0.3]: probability distribution over the 3 classes

Output layer

Hidden layer

Hidden layer

Word Embeddings!

Input layer

First layer = sum of input word embeddings

Language Model: Architecture

Bengio et al, 2003

Example

$$P(w_i = \text{``}grass\text{''} | w_{i-2} = \text{``}cow\text{''}, w_{i-1} = \text{``}eats\text{''})$$

- Lookup word embeddings for *cow* and *eats*

rabbit	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

$$\vec{x} = \vec{v_{\text{cow}}} \oplus \vec{v_{\text{eats}}}$$

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

$$\vec{y} = \text{softmax}(W_2\vec{h_1})$$

21

Output

- *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

$$P(w_i = \text{``}grass\text{''}|w_{i-2} = \text{``}cow\text{''}, w_{i-1} = \text{``}eats\text{''}) = 0.8$$

- Most parameters are in the word embeddings (size = d x |V|) and the output embeddings (size = |V| x d)

Example

$$P(w_i = \text{``}grass\text{''} | w_{i-2} = \text{``}cow\text{''}, w_{i-1} = \text{``}eats\text{''})$$

- Lookup word embeddings for *cow* and *eats*

rabbit	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
d x IVI

- Concatenate them and feed it to the network

$$\vec{x} = \vec{v_{\text{cow}}} \oplus \vec{v_{\text{eats}}}$$

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

$$\vec{y} = \text{softmax}(W_2 \vec{h_1})$$

Output word embeddings
IVI x d

Why Bother?

- ## Ngram LMs

  ‣ cheap to train (just compute 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

- ## NNLMs more robust

  ‣ force words through low-dimensional embeddings

  ‣ automatically capture word properties, leading to more robust estimates

  ‣ flexible: minor change to adapt to other tasks (tagging)

POS Tagging

- POS tagging can also be framed as classification:

$$P\!\left(t_i \,\middle|\, w_{i-1} = \text{"}cow\text{"},\ w_i = \text{"}eats\text{"}\right)$$
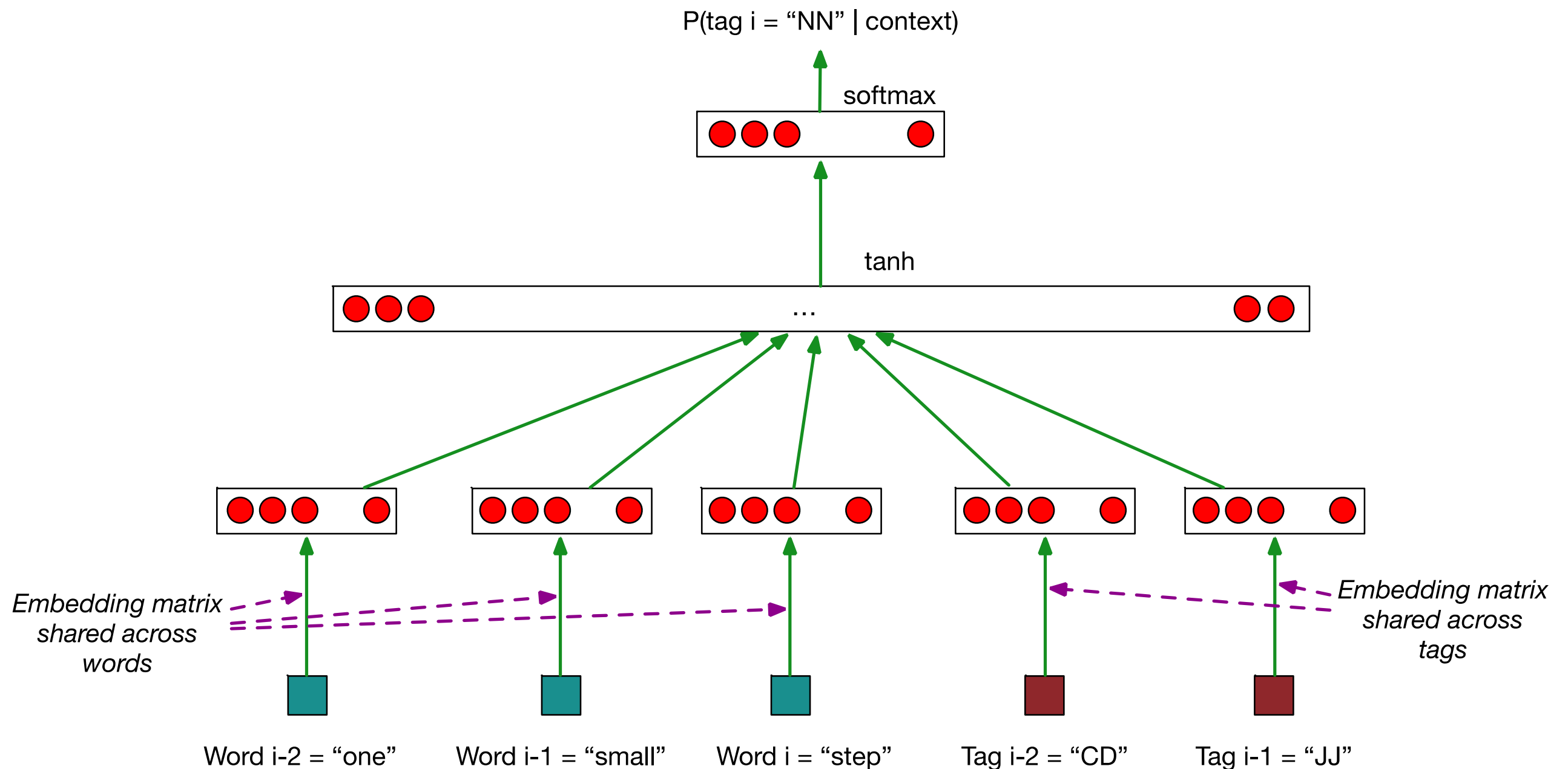
classifies the likely POS tag for "eats".

- Why not use a fancier classifier? (Neural net)

- NNLM architecture can be adapted to the task directly

Feed-forward NN for Tagging

- MEMM tagger takes as input:
  - ‣ 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 ITI, using softmax

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

FF-NN for Tagging

P(tag i = "NN" | context)

softmax

tanh

...

Embedding matrix shared across words

Embedding matrix shared across tags

Word i-2 = "one" Word i-1 = "small" Word i = "step" Tag i-2 = "CD" Tag i-1 = "JJ"
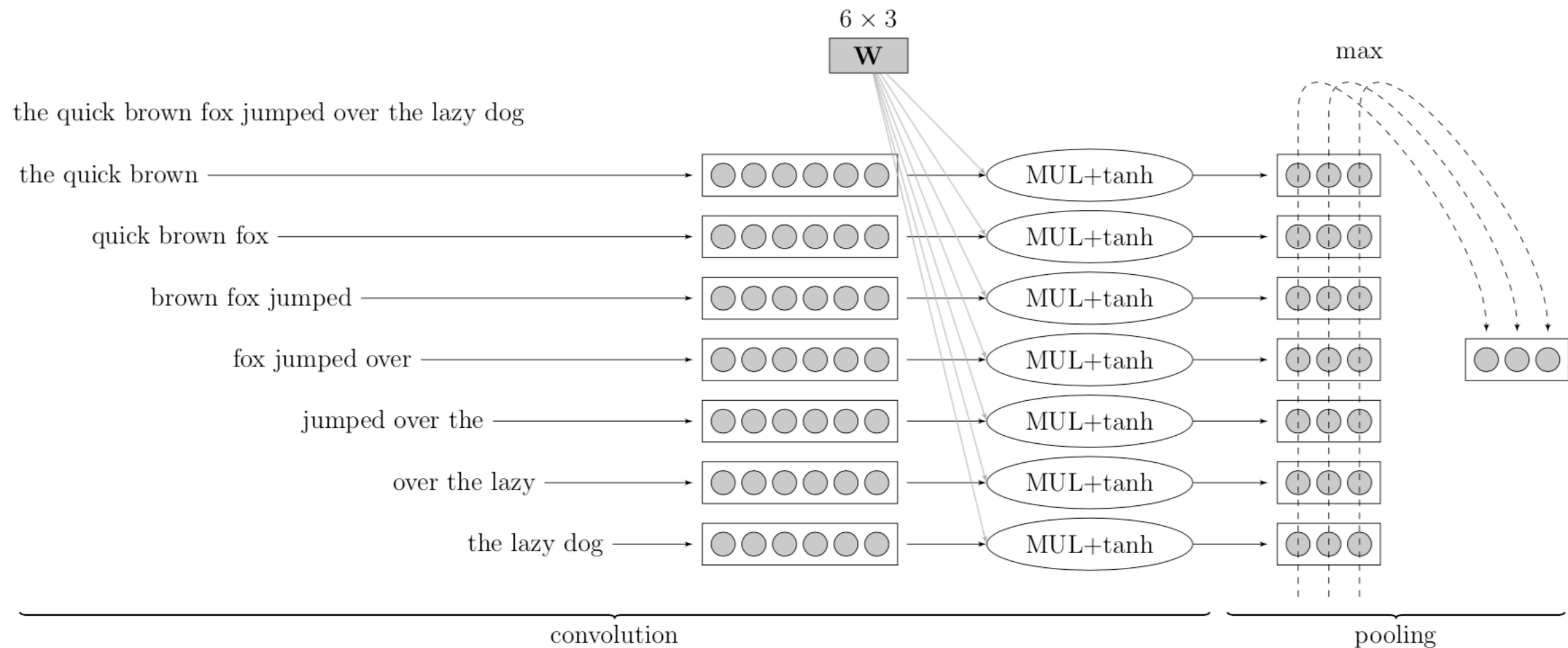
Convolutional Networks

- Commonly used in computer vision

- Identify indicative local predictors

- Combine them to produce a fixed-size representation
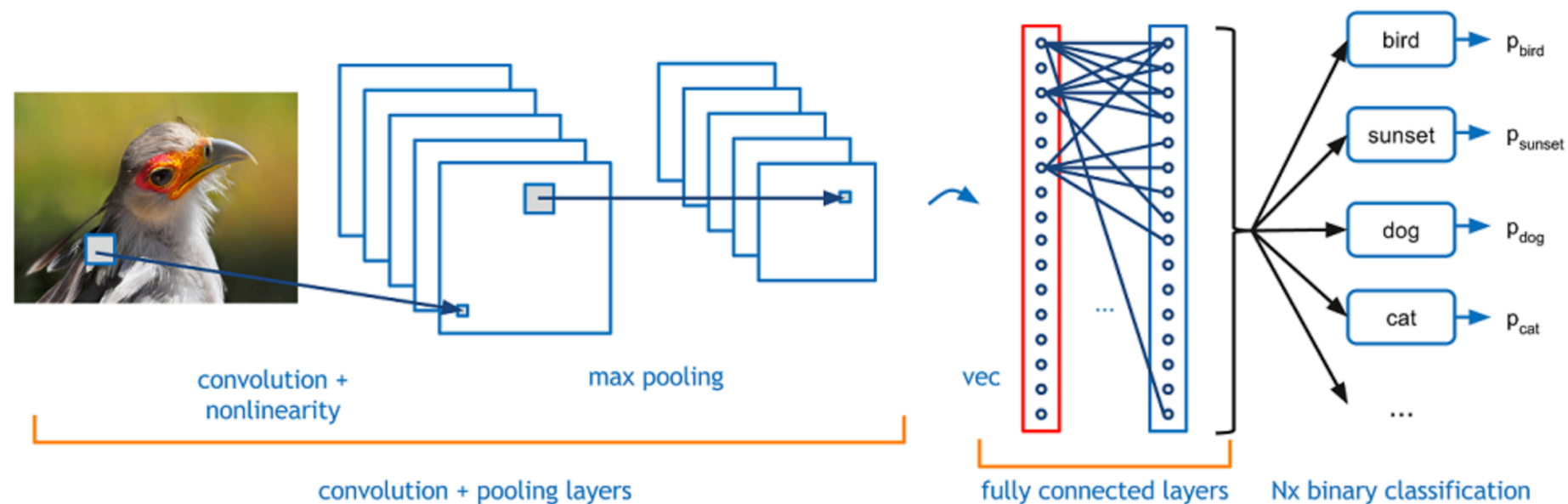
Convolutional Networks for NLP

- Sliding window (e.g. 3 words) over sequence

- W = convolution filter (linear transformation+tanh)

- max-pool to produce a fixed-size representation

Final Words

- ## Neural networks

  ▸ Robust to word variation, typos, etc

  ▸ Excellent generalization

  ▸ Flexible — customised architecture for different tasks

- ## Cons

  ▸ Much slower than classical ML models… but GPU acceleration

  ▸ 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

- Convolutional network: G15, section 9