COMP90042 LECTURE 14

NEURAL LANGUAGE MODELS

LANGUAGE MODELS

- Assign a probability to a sequence of words
- Framed as "sliding a window" over the sentence, predicting each word from finite context to left

  E.g., $n = 3$, a trigram model

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

- Training (estimation) from frequency counts
  - Difficulty with rare events → smoothing
- But are there better solutions?
  Neural networks (deep learning) a popular alternative.

OUTLINE

- Neural network fundamentals
- "Feed-forward" neural language models
- Recurrent neural language models

LMS AS CLASSIFIERS

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

$$P(w_m | w_{m-2} = \text{“}cow\text{”}, \mathrm{w}_{\mathrm{m-1}} = \text{“}eats\text{”})$$

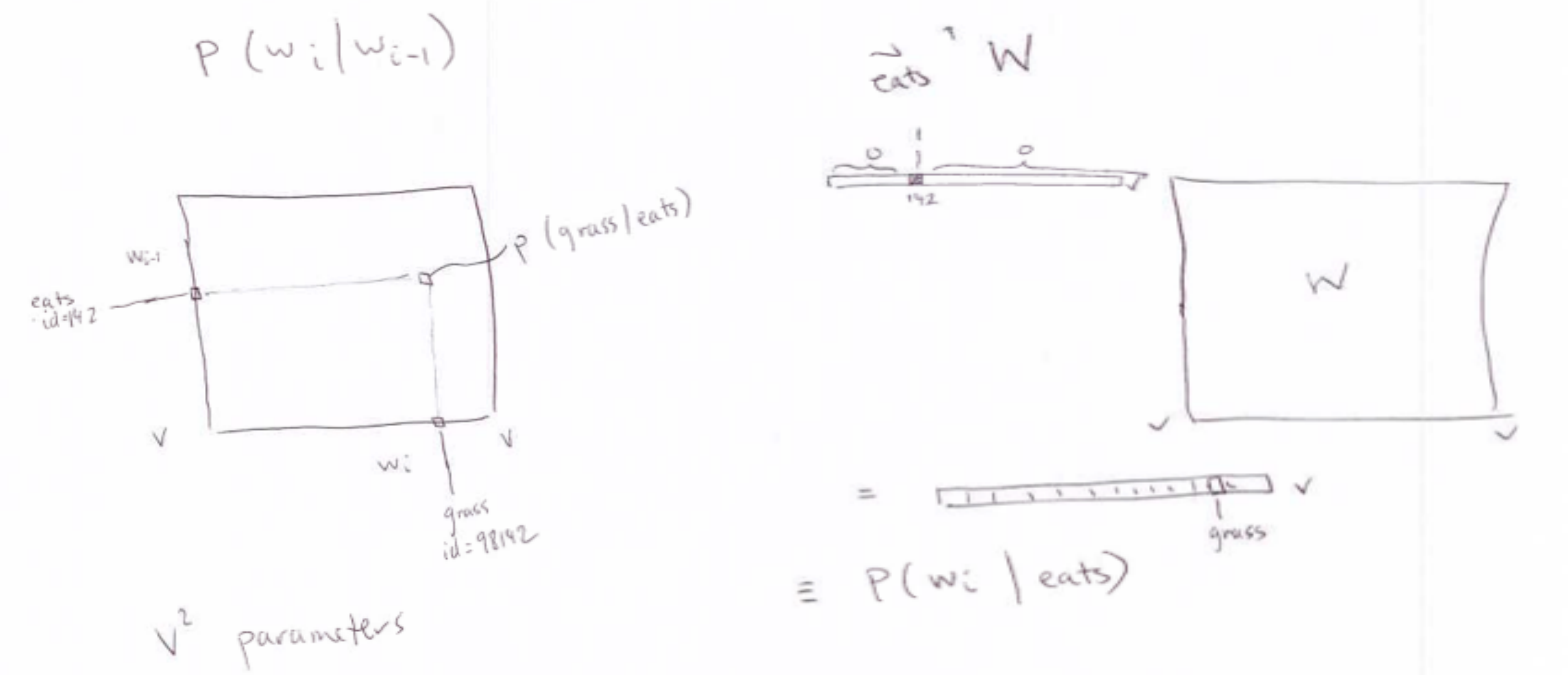
classifies the likely next word in a sequence.

Has a parameter for every combination of

$$w_{m-2}, \mathrm{w}_{\mathrm{m-1}}, w_m$$

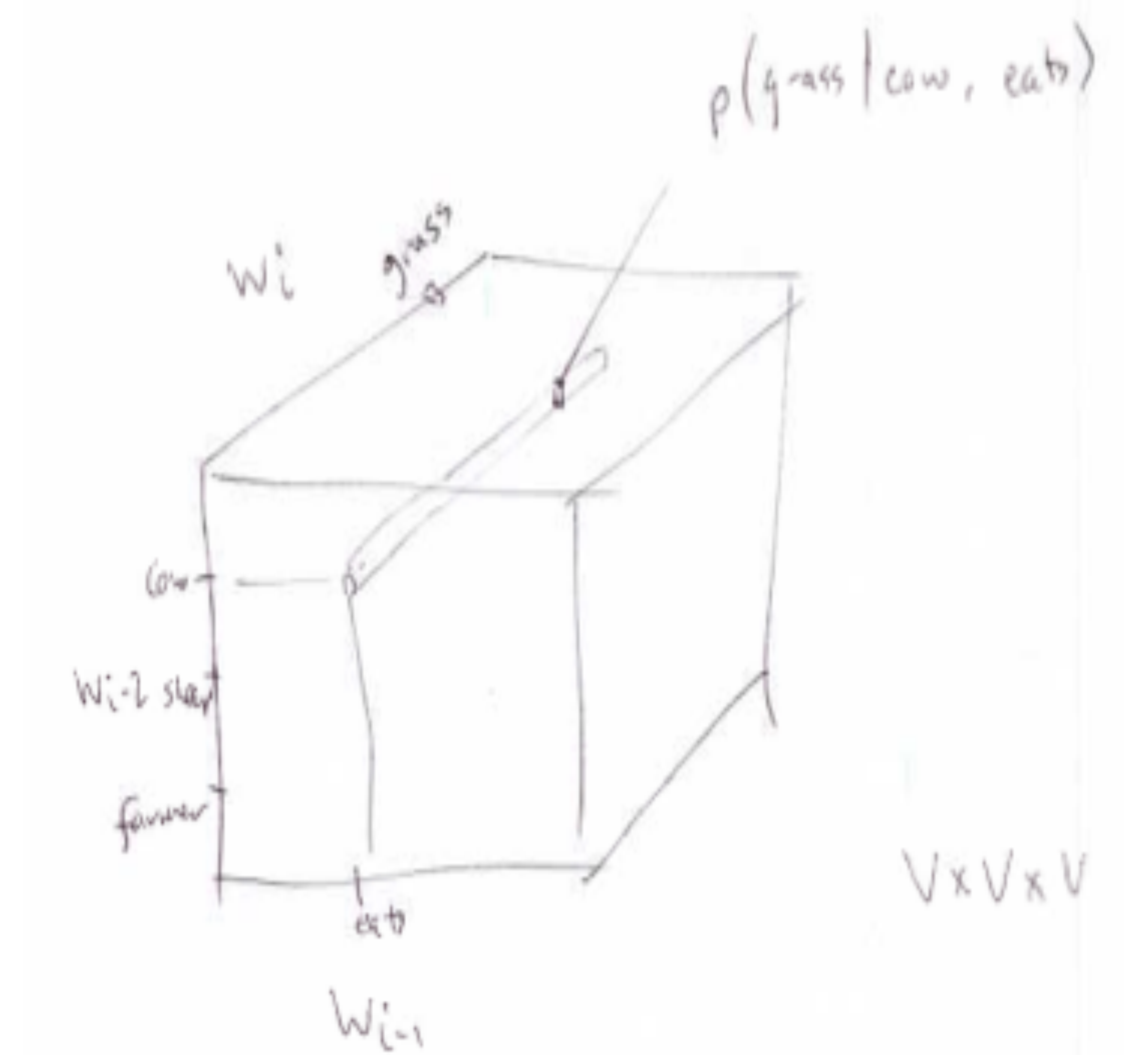
Can think of this as a specific type of classifier — one with a very simple parameterisation.

BIGRAM LM IN PICTURES



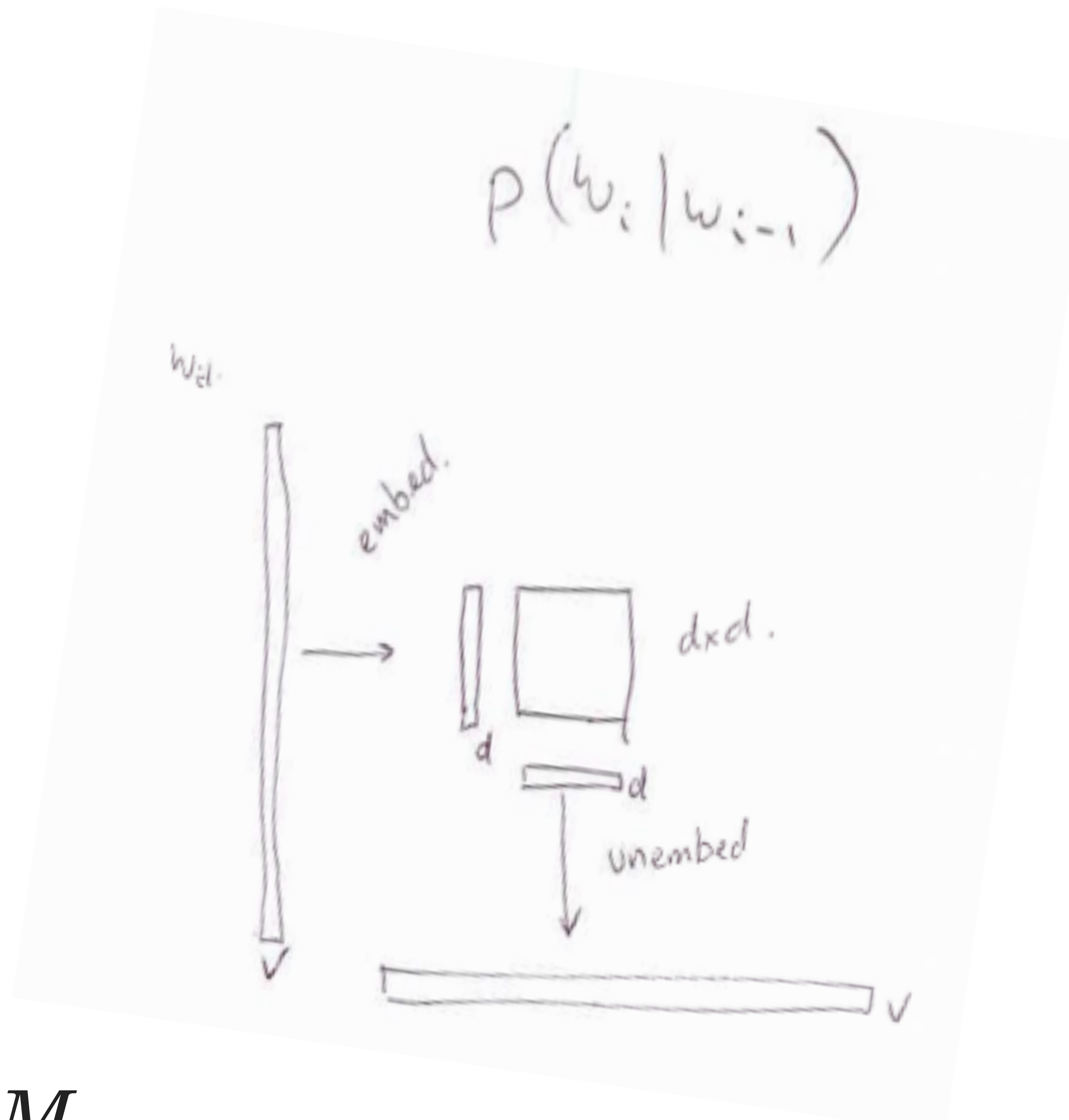
FROM BIGRAMS TO TRIGRAMS

- Problems
  - too many parameters (V^3) to be learned from data (growing with LM order)
  - parameters completely separate for different trigrams
    - even when sharing words, e.g., (cow, eats) vs (farmer, eats) *[motivating smoothing]*
    - doesn't recognise similar words e.g., cow vs sheep, eats vs chews



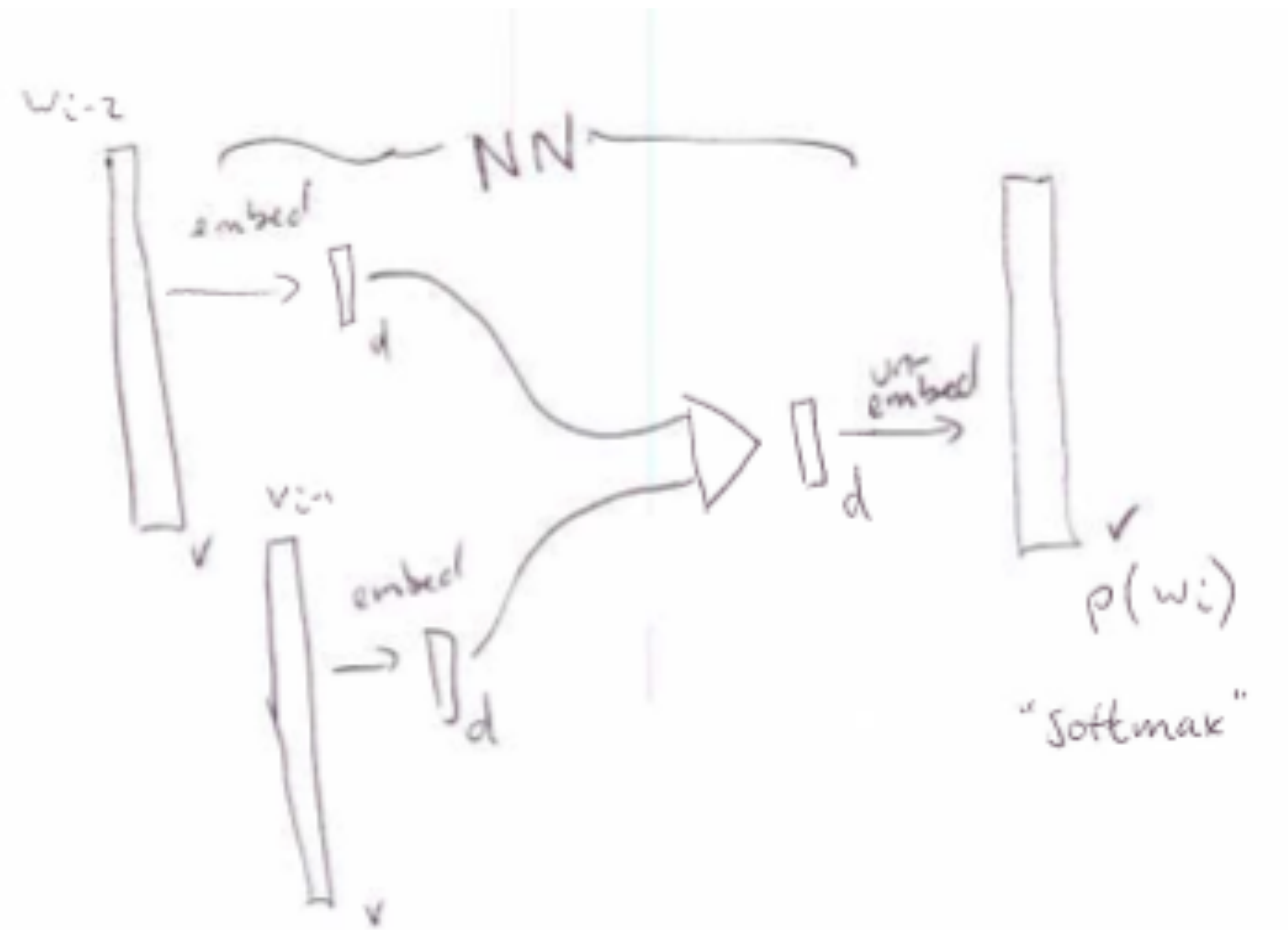
CAN WE USE WORD EMBEDDINGS?

- How about embedding t the words?
- Fewer parameters:
  - embedding matrix (or two matrices), of size V x d
  - weights now d x d
- d << V is a parameter, typically in [100, 1000]
- Model is the *log-bilinear LM* (closely related to *word2vec*)



FEED FORWARD NEURAL NET LMS

- Neural networks more general approach, based on same principle
  - embed input context words
  - transform in “hidden” space
  - un-embed to prob over full vocab
- Neural network used to define transformations
  - e.g., feed forward LM (FFLM)



INTERIM SUMMARY

- Ngram LMs
  - cheap to train (just compute counts)
  - but too many parameters, 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

NEURAL NETWORKS

"Deep" neural networks provide mechanism for learning richer models.

Based on **vector** *embeddings* and compositional functions over these vectors.

- *Word embeddings capture* grammatical and semantic similarity "cows" ~ "sheep", "eats" ~ "chews" etc.
- Vector composition can allow for combinations of features to be learned (e.g., humans consume meat)
- Limit size of vector representation to keep model capacity under control.

COMPONENTS OF NN CLASSIFIER

- NN = Neural Network
  - a.k.a. artificial NN, deep learning, multilayer perceptron
- Composed of simple functions of vector-valued inputs

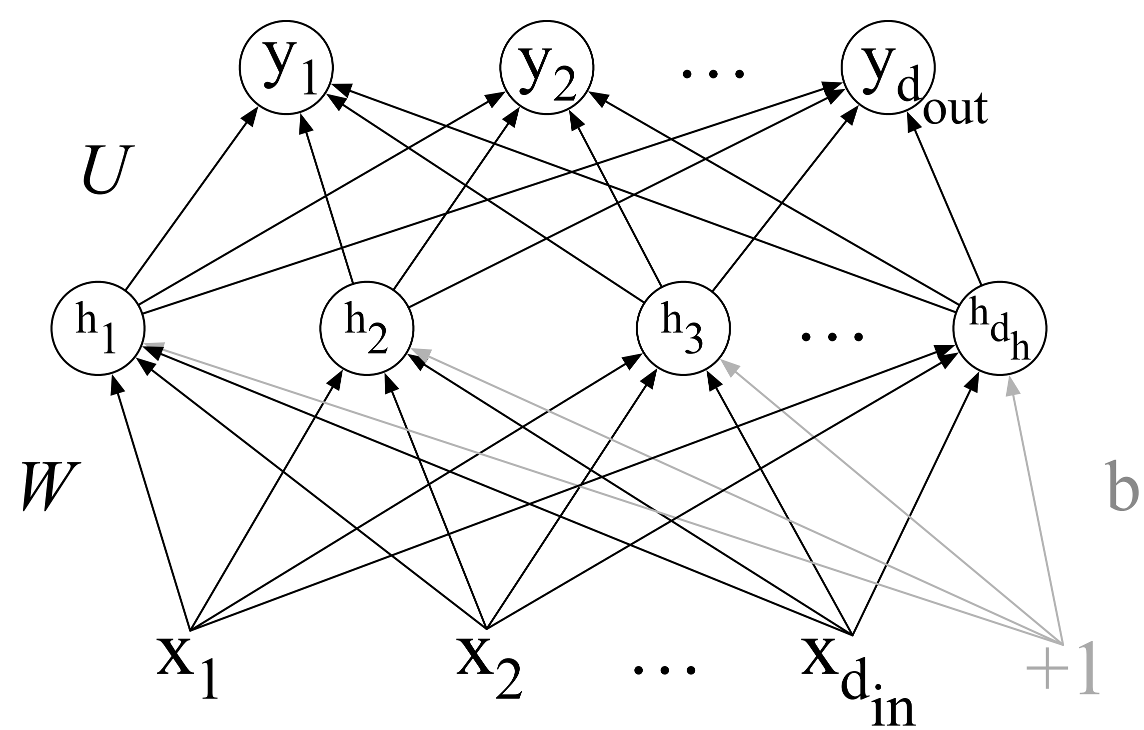


Fig JM3 Ch 8

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

NEURAL NETWORK COMPONENTS

- Typically have several hidden units, i.e.,

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

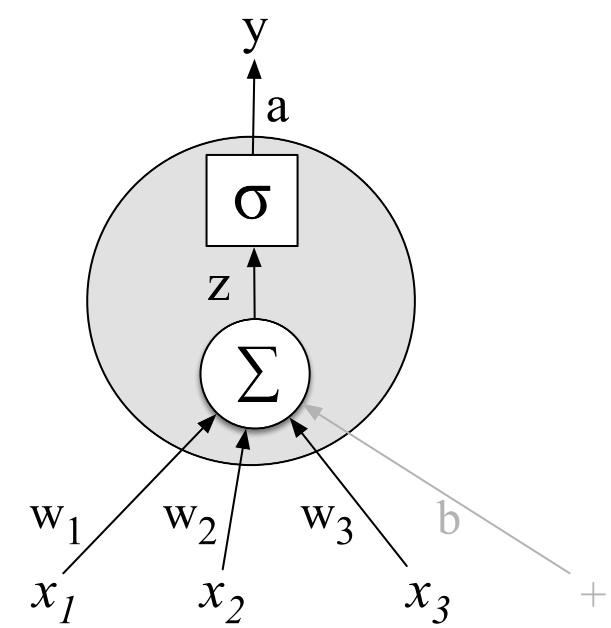
  - each with own weights (w_i) and bias term (b_i)
  - can be expressed using matrix & vector operators

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

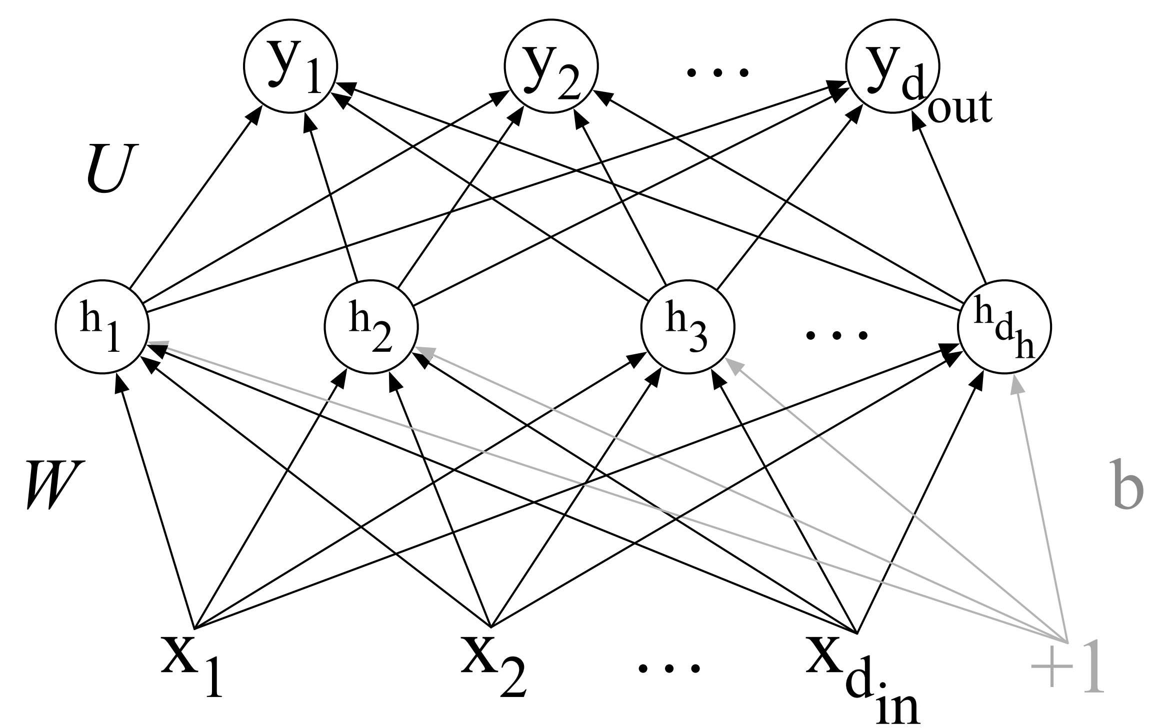
  - where W is a matrix comprising the unit weight vectors, and b is a vector of all the bias terms
  - and *tanh* applied element-wise to a vector
- Very efficient and simple implementation

ANN IN PICTURES

- Pictorial representation of a single unit, for computing y from x



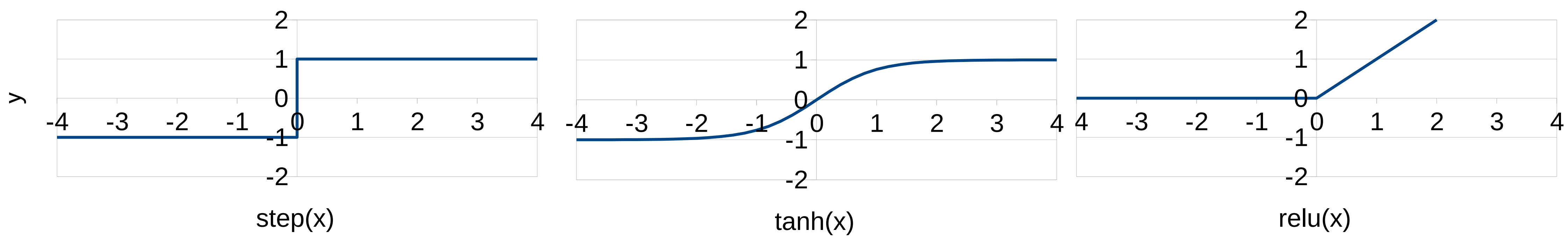
- Typical networks have several units, and additional layers
- E.g., output layer, for classification target



Figs JM3 Ch 8

BEHIND THE NON-LINEARITY

- Non-linearity lends expressive power beyond logistic regression (linear ANN is otherwise equivalent)

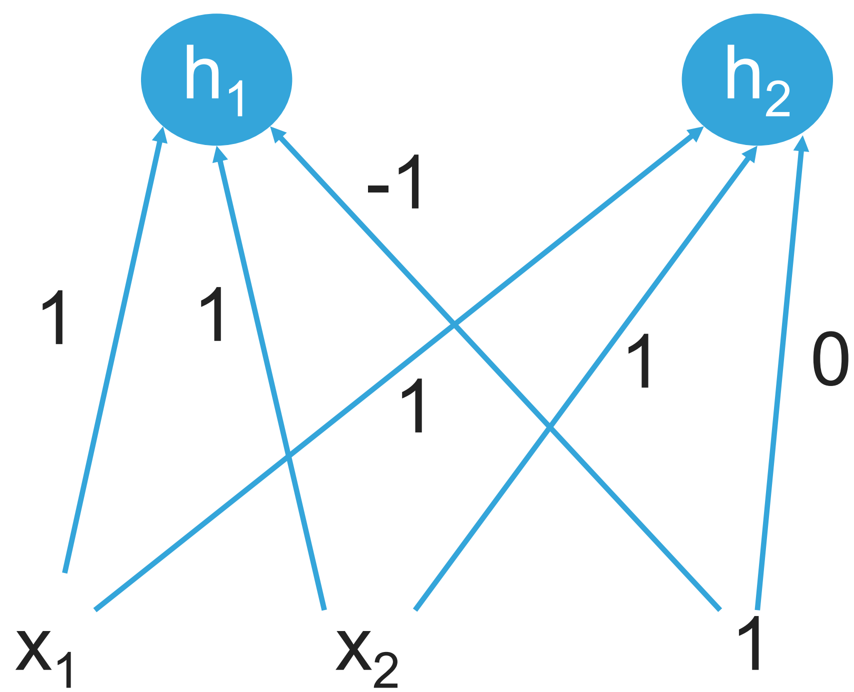


- Non-linearity allows for complex functions to be learned
  - e.g., logical operations: AND, OR, NOT etc.
  - single hidden layer ANN is *universal approximator*: can represent any function with sufficiently large hidden state

Fig from Neubig (2017)

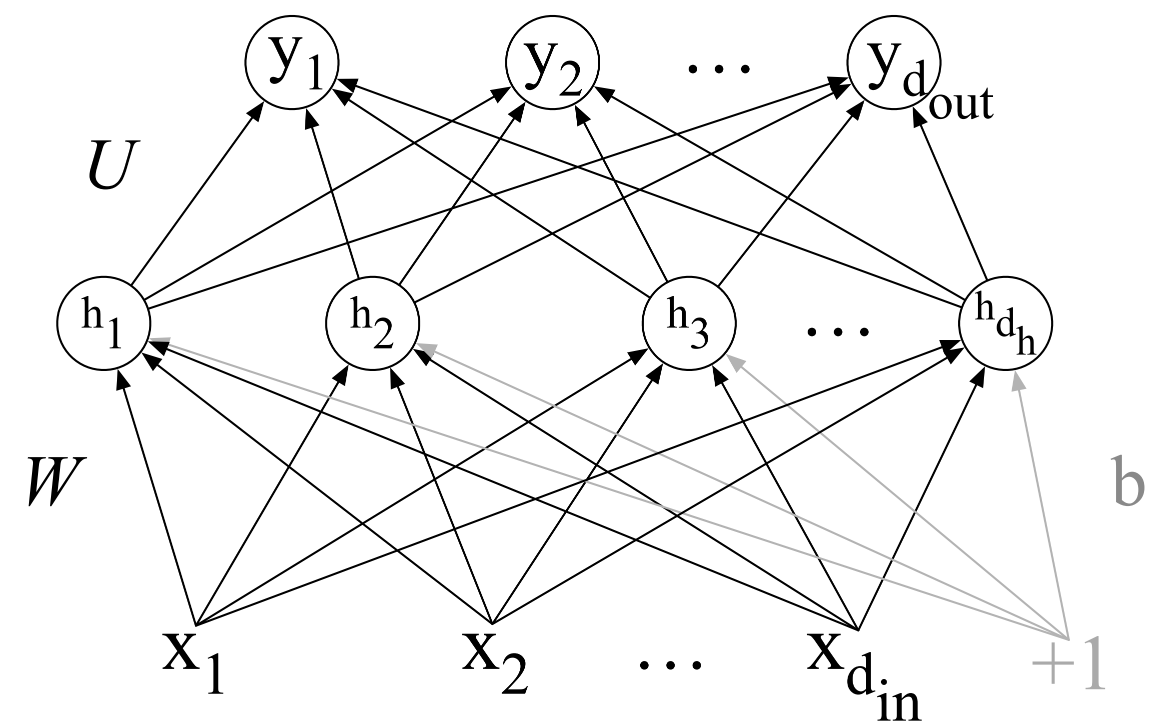
EXAMPLE NETWORKS

- What do these units do?
  - using the "step" activation function
  - consider binary values for x_1 and x_2



COUPLING THE OUTPUT LAYER

- To make this into a classifier, need to produce a classification output
  - e.g., vector of probabilities for the next word
- Add another layer, which takes h as input, and maps to classification target (e.g., V)
  - to ensure probabilities, apply "softmax" transform
  - softmax applies exponential, and normalises, i.e., applied to vector $\boldsymbol{v}$



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

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

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

DEEP STRUCTURES

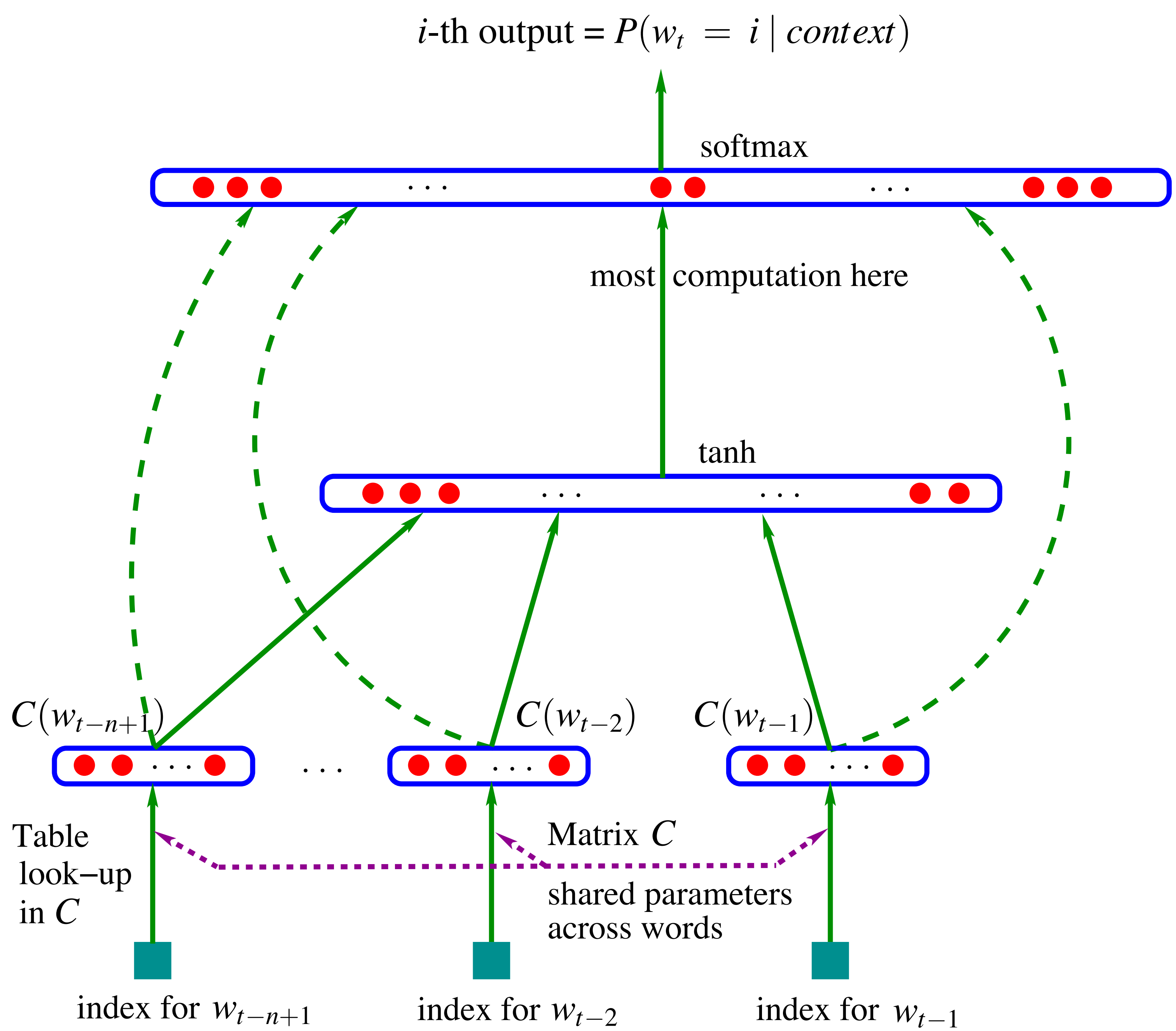
- Can stack several hidden layers; e.g.,
  1. map from 1-hot words, w, to word embeddings, $\boldsymbol{e}$ (lookup)
  2. transform $\boldsymbol{e}$ to hidden state $\boldsymbol{h}_1$ (with non-linearity)
  3. transform $\boldsymbol{h}_1$ to hidden state $\boldsymbol{h}_2$ (with non-linearity)
  4. … *repeat* …
  5. transform $\boldsymbol{h}_n$, to output classification space $\boldsymbol{y}$ (with softmax)
- Each layer typically fully-connected to next lower layer, i.e., each unit is connected to all input elements
- Depth allows for more complex functions, e.g., longer logical expressions

LEARNING WITH SGD

- How to learn the parameters from data?
  - parameters = sets of weights, bias, embeddings
- Consider how well the model "fits" the training data, in terms of the probability it assigns to the correct output
  - *e.g.*, $\prod_{i=1}^{m} P(w_i|w_{i-2}w_{i-1})$ for sequence of m words
  - want to *maximise* this probability, equivalently *minimise* its negative log
  - -log P is known as the "loss" function
- Trained using gradient based methods, much like logistic regression (tools like *tensorflow*, *theano*, *torch* etc use autodiff to compute gradients automatically)

FFNNLM

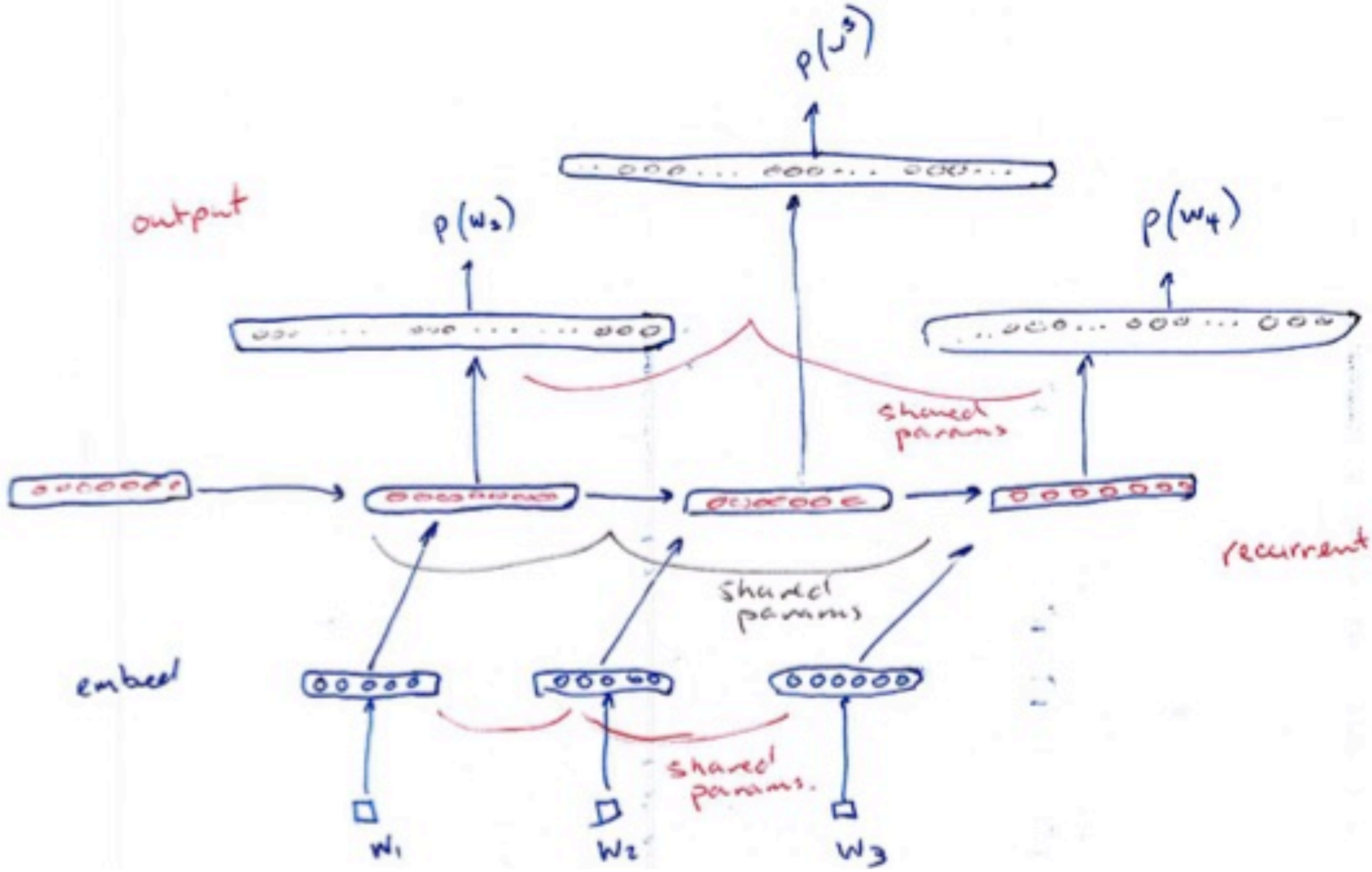
- Application to language modelling



Bengio et al, 2003

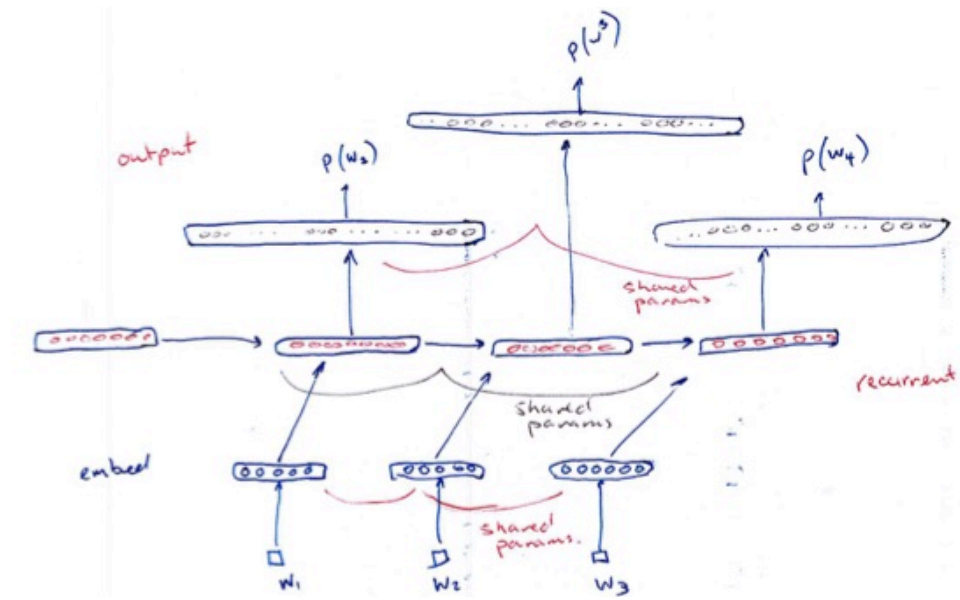
RECURRENT NNLMS

- What if we structure the network differently, e.g., according to sequence with Recurrent Neural Networks (RNNs)



RECURRENT NNLMS

- Start with
  - initial hidden state $\boldsymbol{h}_0$
- For each word, w_i, in order $i=1..m$
  - embed word to produce vector, $\boldsymbol{e}_i$
  - compute hidden $\boldsymbol{h}_i = \tanh(W\,\boldsymbol{e}_i + V\,\boldsymbol{h}_{i-1} + \boldsymbol{b})$
  - compute output $\mathrm{P}(\boldsymbol{w}_{i+1}) = \mathrm{softmax}(U\,\boldsymbol{h}_i + \boldsymbol{c})$
- Train such to minimise $\sum_i -\log \mathrm{P}(\boldsymbol{w}_i)$
  - to learn parameters W, V, U, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{h}_0$



RNNS

- Can results in very “deep” networks, difficult to train due to gradient explosion or vanishing
  - variant RNNs designed to behave better, e.g., GRU, LSTM
- RNNs used widely as sentence encodings
  - RNN processes sentence, word at a time, use final state as fixed dimensional representation of sentence (of any length)
  - Can also run another RNN over reversed sentence, and concatenate both final representations
  - Used in translation, summarisation, generation, text classification, and more

FINAL WORDS

- NNet models
  - Robust to word variation, typos, etc
  - Excellent generalization, especially RNNs
  - Flexible — forms the basis for many other models (translation, summarization, generation, tagging, etc)
- Cons
  - Much slower than counts… but hardware acceleration
  - Need to limit vocabulary, not so good with rare words
  - Not as good at memorizing fixed sequences
  - Data hungry, not so good on tiny data sets

REQUIRED READING

- J&M3 Ch. 8
- Neubig 2017, "Neural Machine Translation and Sequence-to-sequence Models: A Tutorial", Sections 5 & 6