CS440/ECE448 Artificial Intelligence

Lecture 25: Natural Language Processing with Neural Nets

Julia Hockenmaier

April 2019

Today's lecture

- A very quick intro to natural language processing (NLP)
  - What is NLP? Why is NLP hard?
- How are neural networks ("deep learning") being used in NLP
  - And why do they work so well?

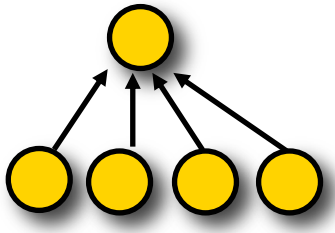
Recap:
Neural Nets/Deep Learning

What is “deep learning”?

- Neural networks, typically with several hidden layers
  - (depth = # of hidden layers)
  - Single-layer neural nets are linear classifiers
  - Multi-layer neural nets are more expressive
- Very impressive performance gains in computer vision (ImageNet) and speech recognition over the last several years.
- Neural nets have been around for decades.
- Why have they suddenly made a comeback?
  - Fast computers (GPUs!) and (very) large datasets have made it possible to train these very complex models.

Single-layer feedforward nets

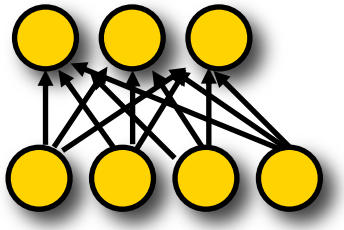
For binary classification tasks:
Single output unit
Return 1 if y > 0.5
Return 0 otherwise



Output unit: scalar *y*

Input layer: vector **x**

For multiclass classification tasks:
K output units (a vector)
Each output unit y_i corresponds to a class i
Return $\text{argmax}_i(y_i)$ where
$y_i = P(i) = \text{softmax}(z_i)$
$= \exp(z_i) / \sum_{k=0..K} \exp(z_k)$

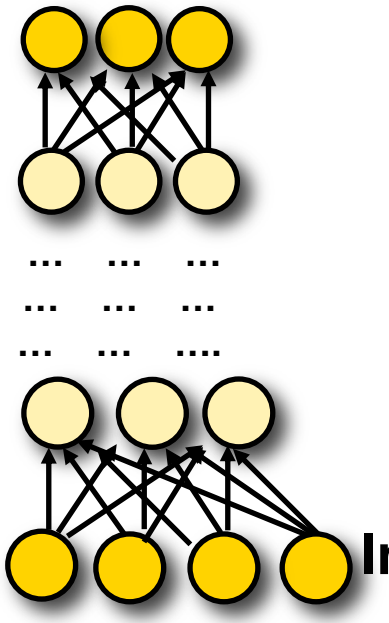


Output layer: vector **y**

Input layer: vector **x**

Multi-layer feedforward networks

We can generalize this to **multi-layer** feedforward nets



Output layer: vector **y**

Hidden layer: vector $\mathbf{h}_n$

Hidden layer: vector $\mathbf{h}_1$

Input layer: vector **x**

Multiclass models: softmax(y_i)

Multiclass classification = predict one of K classes.

Return the class i with the highest score: $\text{argmax}_i(y_i)$

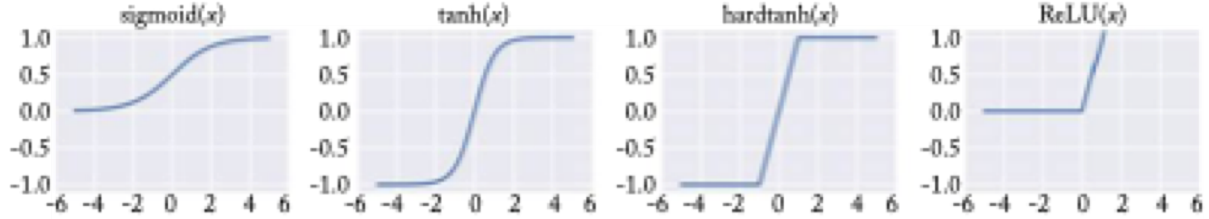
In neural networks, this is typically done by using the **softmax** function, which maps real-valued vectors in R^K to distributions over the K outputs

Given a vector $\mathbf{z} = (z_0 \ldots z_K)$ of activations z_i for each K classes

Probability of class i: $P(i) = \text{softmax}(z_i) = \exp(z_i) / \sum_{k=0..K} \exp(z_k)$

(NB: This is just logistic regression)

Nonlinear activation functions



Sigmoid (logistic function): $\sigma(x) = 1/(1 + e^{-x})$

Useful for output units (probabilities) [0,1] range

Hyperbolic tangent: $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

Useful for internal units: [-1,1] range

Hard tanh (approximates tanh)

htanh(x) = $^{-}1$ for $x < {}^{-}1$, 1 for $x > 1$, x otherwise

Rectified Linear Unit: $\mathrm{ReLU}(x) = \max(0, x)$

Useful for internal units

What is Natural Language Processing?

… and why is it challenging?

What is Natural Language?

- Any **human language:** English, Chinese, Arabic, Inuktitut,...

  NLP typically assumes **written** language (this could be transcripts of spoken language).

  Speech understanding and generation requires additional tools (signal processing etc.)

- Consists of a **vocabulary** (set of words) and a **grammar** to form **phrases** and **sentences** from these words.

  NLP (and modern linguistics) is largely not concerned with ”prescriptive” grammar (which is what you may have learned in school), but with formal (computational) models of grammar, and with how people *actually* use language

- Used by people to **communicate**
  - **Texts written by a single person**: articles, books, tweets, etc.
  - **Dialogues**: communications between two or more people

What is Natural Language Processing?

Any processing of (written) natural languages by computers:

- **Natural Language Understanding (NLU)**
  - Translate from text to a semantic **meaning representation**
  - May (should?) require **reasoning** over semantic representations
- **Natural Language Generation (NLG)**
  - **Produce text** (e.g. from a semantic representation)
  - Decode ***what*** **to say** as well as ***how*** **to say it**.
- **Dialogue Systems:**
  - Require both **NLU and NLG**
  - Often **task-driven** (e.g. to book a flight, get customer service, etc.)
- **Machine Translation:**
  - Translate **from one human language to another**
  - Typically done without intermediate semantic representations

What do we mean by “meaning”?

Lexical semantics: the (literal) meaning of words

Nouns (mostly) describe *entities*, verbs *actions, events, states*, adjectives and adverbs *properties*, prepositions *relations*, etc.

Compositional semantics: the (literal) meaning of sentences

Principle of compositionality:
The meaning of a phrase or sentence depends on the meanings of its parts and on how these parts are put together.

Declarative sentences *describe* events, entities or facts,
questions *request information* from the listener,
commands *request actions* from the listener, etc.

Pragmatics studies how (non-literal) meaning depends on context, speaker intent, etc.

How do we represent “meaning”?

A) **Symbolic meaning representation languages**:

Often based on (**predicate) logic** (or inspired by it)

May focus on different aspects of meaning, depending on the application

Have to be **explicitly defined and specified**

Can be verified by humans (useful for development/explainability)

NLU: How do we get to that “meaning”?

A) The **traditional NLP pipeline** assumes a sequence of **intermediate symbolic representations**, produced by models whose output can be reused by any system

Map raw text to part-of-speech tags,
then map POS-tagged text to syntactic parse trees,
then map syntactically parsed text to semantic parses, etc.

Components of the NLP pipeline

All steps (except tokenization) return a **symbolic representation**

- **Tokenization**: Identify word and sentence boundaries
- **POS tagging**: Label each word as noun, verb, etc.
- **Named Entity Recognition (NER):** Identify all named mentions of people, places, organizations, dates etc. as such
- **Coreference Resolution (Coref):** Identify which mentions in a document refer to the same entity
- **(Syntactic) Parsing:** Identify the grammatical structure of each sentence
- **Semantic Parsing:** Identify the meaning of each sentence
- **Discourse Parsing:** Identify the (rhetorical) relations between sentences/phrases

Why is NLU difficult?

- Natural languages are **infinite…**
  … because their **vocabularies** have a power law distribution (Zipf's Law)
  … and because their **grammars** allow recursive structures
- Natural languages are **highly ambiguous…**
  … because many words have **multiple senses**
  … and because there is a **combinatorial explosion** of sentence meanings
- Much of the meaning **is not expressed explicitly…**
  … because listeners/readers have **commonsense/world knowledge**
  … and because they can **draw inferences from what is and isn't said**.

Why is NLU difficult?

- Natural languages are **infinite…**
  … so any input will contain **new/unknown words/constructions**

- Natural languages are **highly ambiguous…**
  … so recovering **the correct structure/meaning** is often very difficult

- Much of the meaning **is not expressed explicitly…**
  … so a **symbolic meaning representation** of the explicit meaning may not be sufficient.

Why are NLG and MT difficult?

- The generated text (or translation) has to be **fluent**
  Sentences should be **grammatical**. Texts need to be **coherent/cohesive**.
  This requires capturing **non-local dependencies**
  between words that are far apart in the string.

- The text (or translation) has to convey the **intended meaning**.
  Translations have to be **faithful** to the original.
  Generated text should **not be misunderstood** by the human reader
  But there are **many different ways to express the same information**

- NLG and MT are difficult to **evaluate automatically**
  Automated metrics exist, but correlate poorly with **human judgments**

NLP research questions redux…
…and answers from traditional NLP

- How do you **represent (or predict) words?**
  - Each word is its own **atomic symbol**.
    All unknown words are mapped to the same **UNK token**.
  - We capture lexical semantics through an **ontology** (WordNet) or **sparse vectors**
- How do you **represent (or predict) word sequences?**
  - Through an n-gram language model (with fixed n=3,4,5,…), or a grammar
- How do you **represent (or predict) structures?**
  - Representations are symbolic
  - Predictions are made by statistical models/classifiers

Neural Approaches to NLP

Challenges in using NNs for NLP

NLP input (and output) consists of **variable length sequences of discrete symbols** (sentences, documents, …)

But the input to neural nets typically consists of **fixed-length continuous vectors**

Solutions

1) Learn a mapping (**embedding**) from discrete symbols (words) to dense continuous vectors that can be used as input to NNs
2) Use **recurrent neural nets** to handle variable length inputs and outputs

Added benefits of these solutions

Benefits of **word embeddings**:

- Words that are similar have similar word vectors
- We have a much better handle on lexical semantics
- Because we can train these embeddings on massive amounts of raw text, we now have a much better way to handle and generalize to rare and unseen words.

Benefits of **recurrent nets**:

- We do not need to learn and store explicit n-gram models
- RNNs are much better at capturing non-local dependencies
- RNNs need far fewer parameters than n-gram models with large n.

How does NLP use NNs?

- **Word embeddings** (word2vec, Glove, etc.)
  - Train a NN to predict a word from its context (or the context from a word).
  - This gives a dense vector representation of each word
- **Neural language models:**
  - Use recurrent neural networks (RNNs) to predict word sequences
    More advanced: use LSTMs (special case of RNNs)
- **Sequence-to-sequence (seq2seq) models:**
  - From machine translation: use one RNN to encode source string, and another RNN to decode this into a target string.
  - Also used for automatic image captioning, etc.
- **Convolutional neural networks (CNNs)**
  - Used for text classification

How do we represent “meaning”?

A) **Symbolic meaning representation languages**:

Often based on (**predicate) logic** (or inspired by it)

May focus on different aspects of meaning, depending on the application

Have to be **explicitly defined and specified**

Can be verified by humans (useful for development/explainability)

B) **Continuous (vector-based) meaning representations**:

Non-neural approaches: sparse vectors with a very large number of dimensions (10K+) each of which has an explicit interpretation

Neural approaches: dense vectors with far fewer dimensions (~300) without explicit interpretation

Are **automatically learned** from data.

Can typically not be verified by humans.

NLU: How do we get to that "meaning"?

A) The **traditional NLP pipeline** assumes a sequence of **intermediate symbolic representations**, produced by models whose output can be reused by any system

- Map raw text to part-of-speech tags,
- then map POS-tagged text to syntactic parse trees,
- then map syntactically parsed text to semantic parses, etc.

B) Many current **neural models** map *directly* from text to the output required for the task.

- Map each word in a text to a vector representation
- Train the neural model to perform the task directly from these vectors
- Intermediate representations (activations of hidden layers) may be used by other neural models, but are difficult to interpret by humans

NLP research questions redux …
…and answers from neural NLP

- How do you **represent (or predict) words?**
  - Each **word** is a **dense vector (embedding)** that captures a lot of syntactic and semantic information implicitly.
  - **Character embeddings** allow us us handle unseen words more robustly
- How do you **represent (or predict) word sequences?**
  - Through a **recurrent neural net** that does not need to truncate history to a fixed length
- How do you **represent (or predict) structures?**
  - Input is typically assumed to be raw text (mapped to embeddings)
  - Output representations may still be symbolic
  - Internal representations are dense vectors (activations), without explicit interpretation

Language models

Traditional Language Models

A language model defines a distribution $P(\mathbf{w})$ over the strings
$\mathbf{w} = w_1 w_2 .. w_i ...$ in a language
Typically we factor $P(\mathbf{w})$ such that we compute the probability
word by word: $P(\mathbf{w}) = P(w_1)\, P(w_2 \mid w_1) ... P(w_i \mid w_1 ... w_{i-1})$

Standard **n-gram models** make the Markov assumption that w_i depends (only) on the preceding n−1 words: $P(w_i \mid w_1 ... w_{i-1}) := P(w_i \mid w_{i-n+1} ... w_{i-1})$

We know that this independence assumption is invalid
(there are many long-range dependencies), but it is computationally and statistically necessary (we can't store or estimate larger models)

Neural Language Models

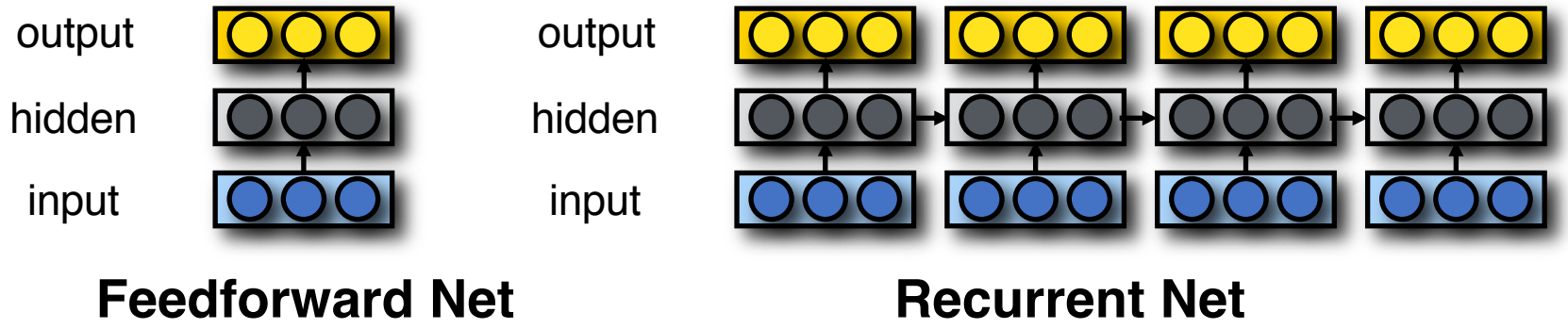
A neural LM defines a distribution over the V words in the vocabulary, conditioned on the preceding words.

- **Output layer:** V units (one per word in the vocabulary) with softmax to get a distribution
- **Input:** Represent each preceding word by its d-dimensional embedding.
  - **Fixed-length history** (n-gram): use preceding n−1 words
  - **Variable-length history**: use **a recurrent neural net**

Recurrent neural networks (RNNs)

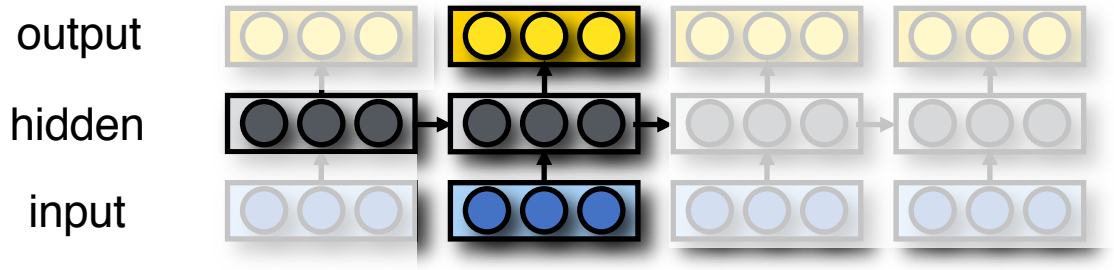
Basic RNN: Modify the standard feedforward architecture (which predicts a string $w_0 \ldots w_n$ one word at a time) such that the output of the current step (w_i) is given as additional input to the next time step (when predicting the output for w_{i+1}).

- "Output" — typically (the last) hidden layer.



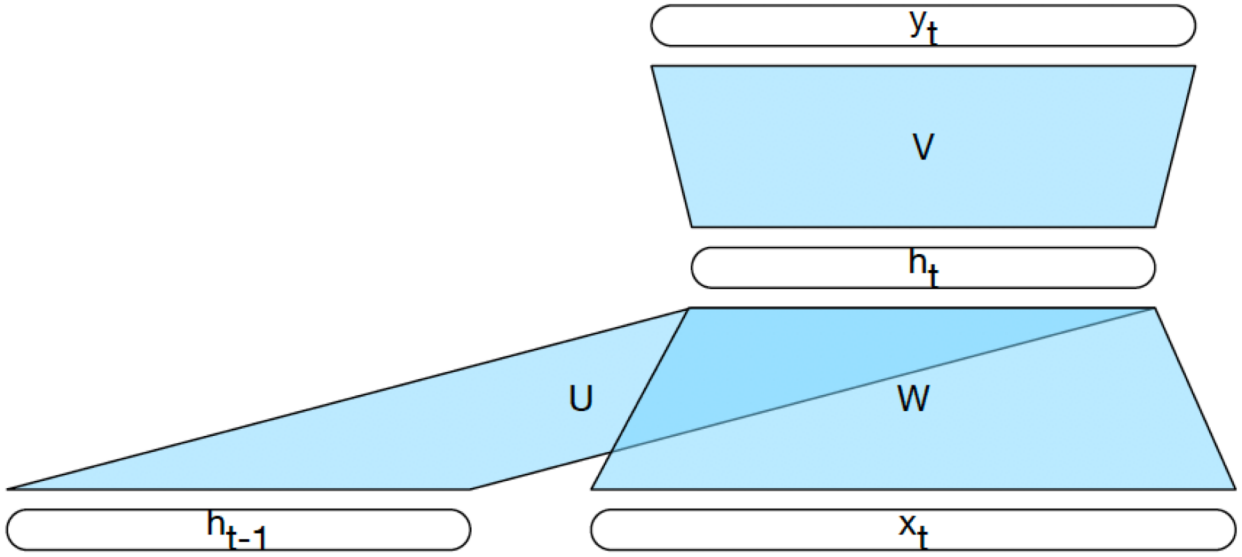
Basic RNNs

Each time step corresponds to a feedforward net where the hidden layer gets its input not just from the layer below but also from the activations of the hidden layer at the previous time step

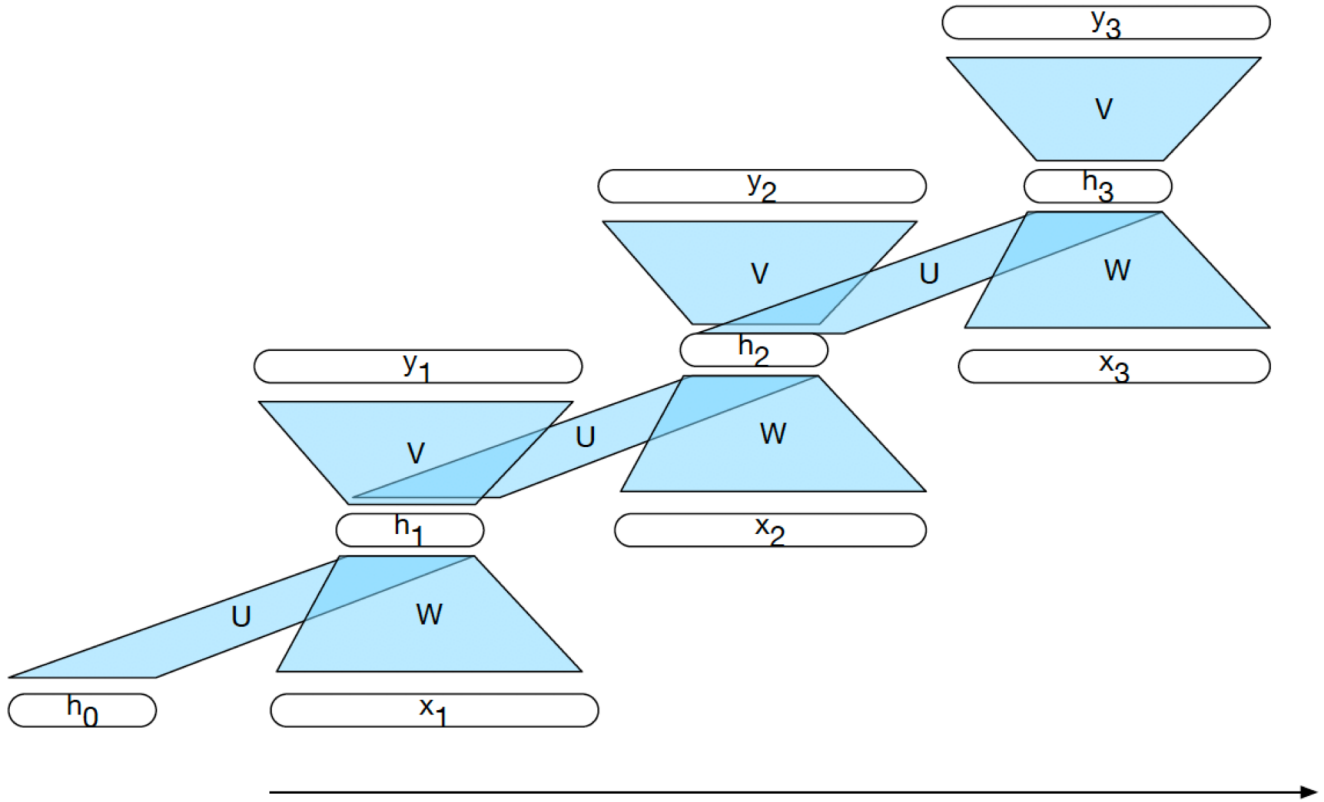


Basic RNNs

Each time step corresponds to a feedforward net where the hidden layer gets its input not just from the layer below but also from the activations of the hidden layer at the previous time step



A basic RNN unrolled in time

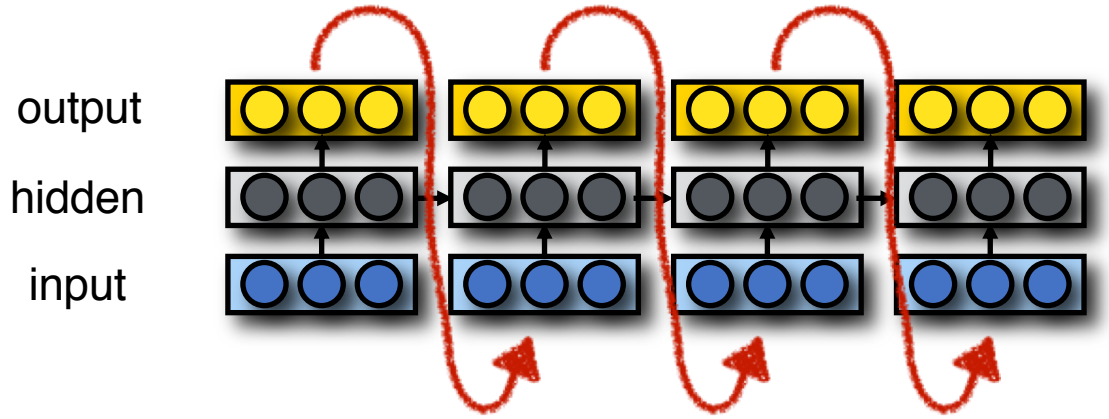


RNNs for generation

To generate a string $w_0 w_1 \ldots w_n\ w_{n+1}$, give w_0 as first input, and then pick the next word according to the computed probability

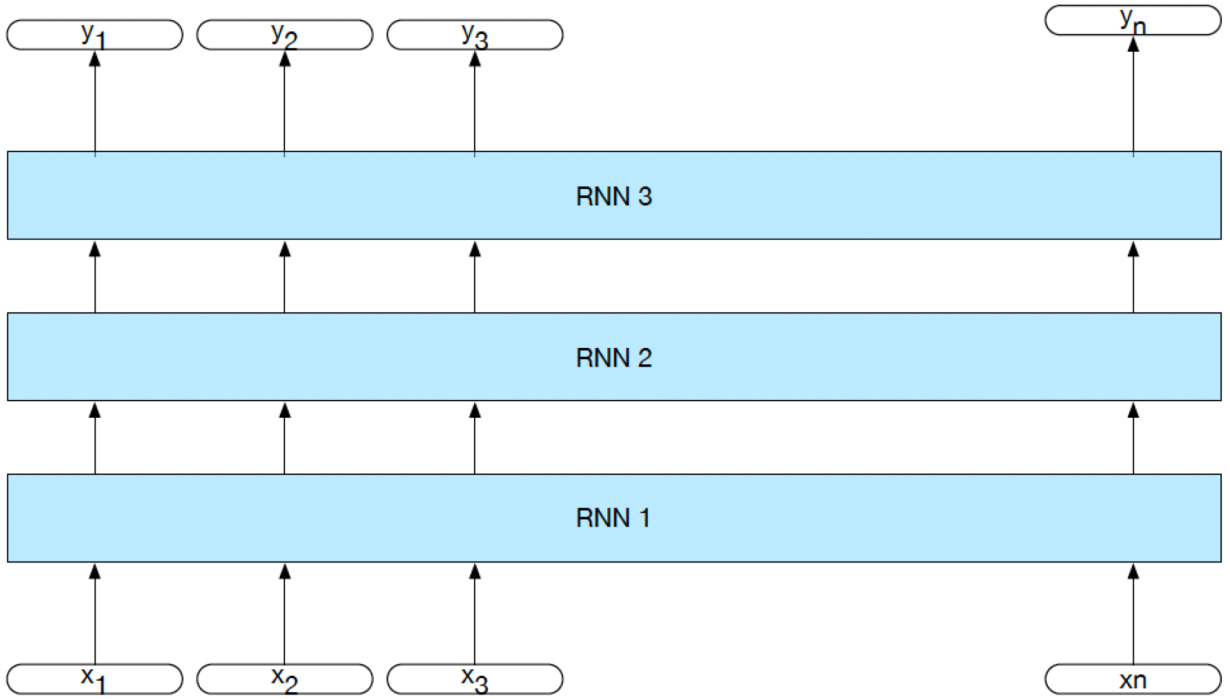
$$P(w_i | w_0 \ldots w_{i-1})$$

Feed this word in as input into the next layer.



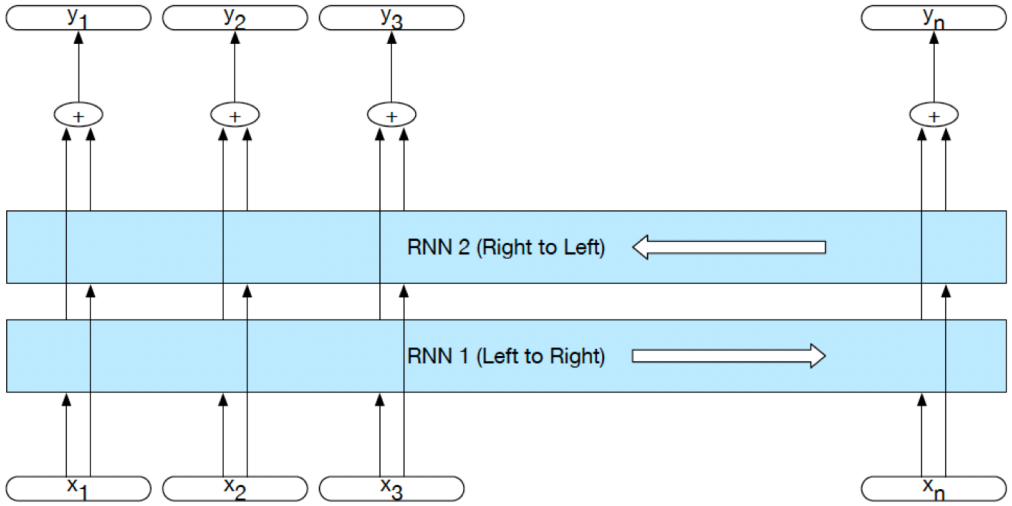
Stacked RNNs

We can create an RNN that has “vertical” depth (at each time step) by stacking multiple RNNs



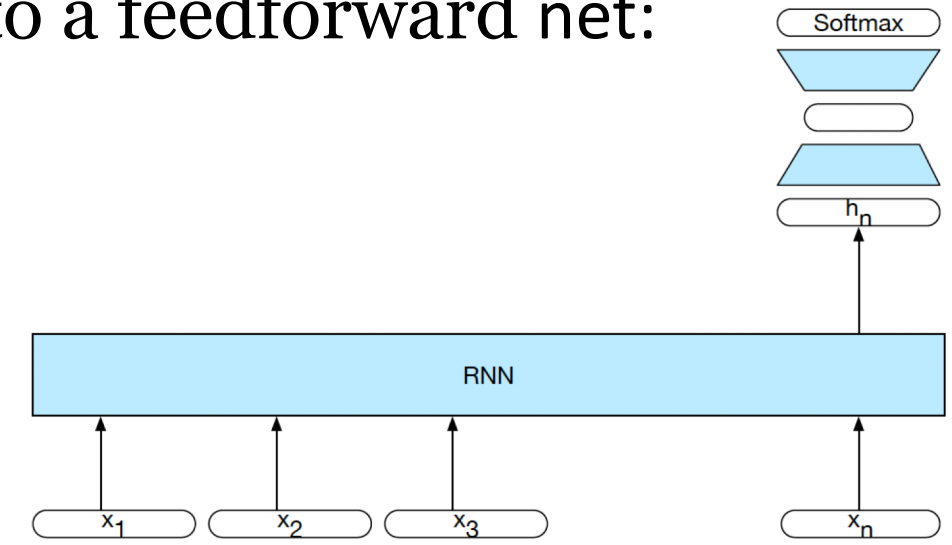
Bidirectional RNNs

- Unless we need to generate a sequence,
  we can run two RNNs over the input sequence
  — one going forward, and one going backward.
- Their hidden states will capture different context information.



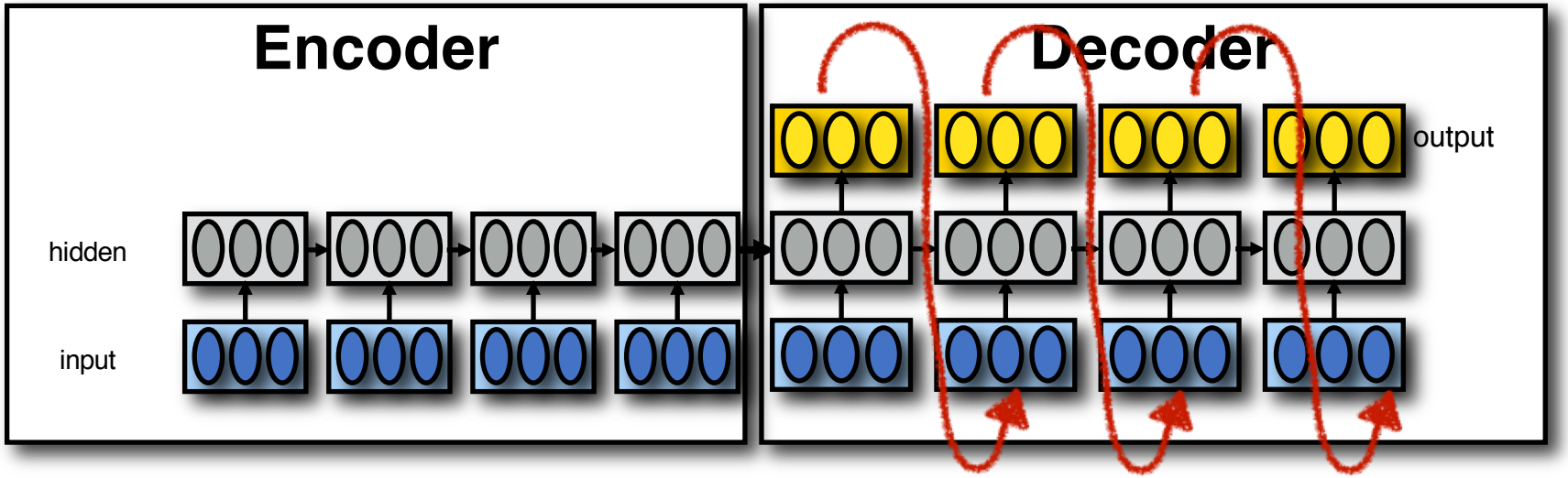
RNNs for sequence classification

- If we just want to **assign one label to the sequence**, we don't need to produce output at each time step, so we can use a simpler architecture.
- We can use **the hidden state of the last word** in the sequence as input to a feedforward net:



Encoder-Decoder (seq2seq) model

- Task: Read an input sequence and return an output sequence
  - Machine translation: translate source into target language
  - Dialog system/chatbot: generate a response
- Reading the input sequence: RNN Encoder
- Generating the output sequence: RNN Decoder



Neural Word Embeddings

Word Embeddings (e.g. word2vec)

Main idea:

If you use a feedforward network to predict the probability of words that appear near an input word, the hidden layer of that network provides a dense vector representation of the input word.

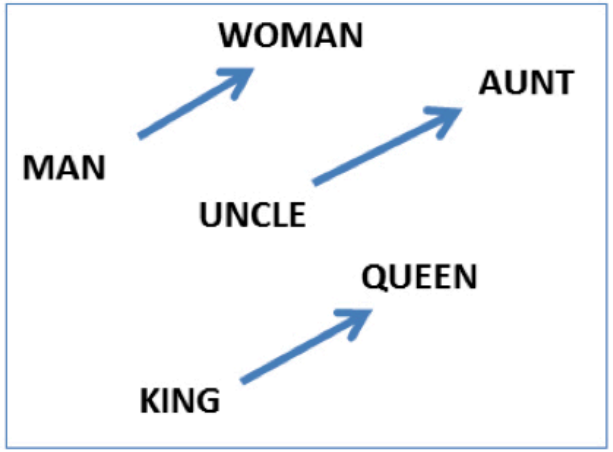
Words that appear in similar contexts (that have high distributional similarity) will have very similar vector representations.

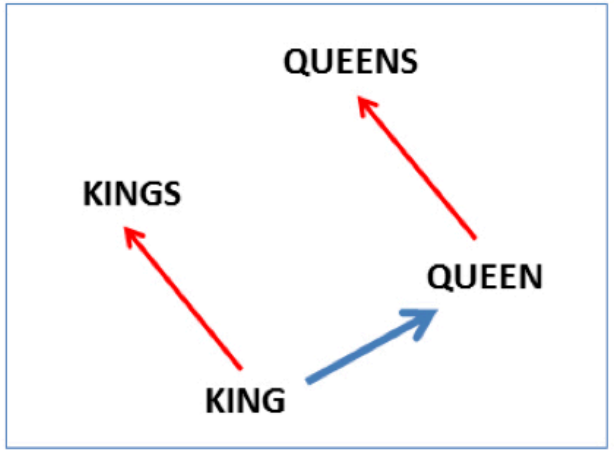
These models can be trained on large amounts of raw text (and pretrained embeddings can be downloaded)

Analogy: Embeddings capture relational meaning!

vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) = vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) = vector(‘*Rome*’)





Embeddings you can use

Static embeddings:

Word2vec (Mikolov et al.)
https://code.google.com/archive/p/word2vec/

Fasttext http://www.fasttext.cc/

Glove (Pennington et al.)
http://nlp.stanford.edu/projects/glove/

More recent developments:

RNN-based embeddings that depend on current word context

BERT (Devlin et al.)
https://github.com/google-research/bert

ELMO https://allennlp.org/elmo (Peters et al.)

Summary

Today's lecture (I)

NLP is difficult because…

- … natural languages have very large (infinite) vocabularies
- … natural language sentences/documents have variable length
- … natural language is highly ambiguous

The traditional NLP (NLU) pipeline consists of a sequence of models that predict symbolic features for the next model

- … but that is quite brittle: mistakes get propagated through the pipeline

Traditional statistical NLG relies on fixed order n-gram models

- … but these are very large, and don't capture long-range dependencies

Today's lecture (II)

To use neural nets for NLP requires…

- … the use of word embeddings that map words to dense vectors
- … more complex architectures (e.g. RNNs, but also CNNs)

Word embeddings help us handle the long tail of rare and unknown words in the input

- Other people have trained them for us on massive amounts of text

RNNs help us capture long-range dependencies between words that are far apart in the sentence.

- No need to make fixed-order Markov assumptions

Word representations as by-product of neural LMs

- **Output embeddings:** the weights that connect the last hidden layer h to the i-th ouput is a dim(**h**)-dimensional vector that is associated with the i-th vocabulary item w ∈ V

h is a dense (non-linear) representation of the context Words that are similar appear in similar contexts.

- Hence their columns in $\mathbf{W}^2$ should be similar.
- **Input embeddings:** each row in the embedding matrix is a representation of a word.