Reminders

Practice midterm is due now. Answers will be released later today.

HW9 is due tonight before midnight.

Midterm 3 will be in-class on Thursday. Topics covered:
- Naive Bayes Classification and Perceptron
- N-gram Language Models
- Vector Space Models
- Logistic Regression
- Neural Networks

You can bring a one page "cheat sheet" with handwritten notes.

Please bring your R2D2s to the exam to return.

The makeup date for midterm 3 is Friday from 9:30am-11am in 3401 Walnut room 401B. Please fill out the request form by tonight.

Extra credit 4 is due on Monday (Dec 9) before 11:59PM. Late days OK!

How hard should you study?

If you'd like to estimate your grade in this course so far, here's how:

1. Compute your average score on the homework. If you're enrolled in CIS 421, you can drop your lowest scoring HW and divide by 8 HWs.

2. Look up your scores on midterm 1 and midterm 2.

3. Multiply them by the grade distribution, which is: 55% HW, and 15% for each midterm.

If you have a 93% avg on the HW, and your midterm 1 grade was 78% and your midterm 2 grade was 89%, then you have:
(93*.55 + 78*.15 + 89*.15)/.85 = 89.65%, which would round up to 90% which is an A-. The class isn't curved. A is 90 and above, B is 80-90, etc.

Each EC assignment is worth up to 1% towards your overall grade. If you did 4 of 4 EC assignments, and you originally got a 89% then you could raise your score from an 89% (B+) to a 93% (A).

Neural Networks part 3

JURAFSKY AND MARTIN CHAPTERS 7 AND 9

Language Modeling

Goal: Learn a **function** that returns the joint probability

Primary difficulty:

1. There are too many parameters to accurately estimate.
2. In n-gram-based models we fail to generalize to related words / word sequences that we have observed.

Curse of dimensionality / sparse statistics

Suppose we want a joint distribution over 10 words.
Suppose we have a vocabulary of size 100,000.

$$100{,}000^{10} = 10^{50} \text{ parameters}$$

This is too high to estimate from data.

Chain rule

In LMs we use the chain rule to get the conditional probability of the next word in the sequence given all of the previous words:

$$P(w_1 w_2 w_3 \ldots w_t) = \prod_{t=1}^{T} P(w_t | w_1 \ldots w_{t-1})$$

What assumption do we make in n-gram LMs to simplify this?

The probability of the next word only depends on the previous *n*-1 words.

A small *n* makes it easier for us to get an estimate of the probability from data.

N-gram Language Models

Estimate the probability of the next word in a sequence, given the entire prior context $P(w_t | w_1^{t-1})$. We use the Markov assumption approximate the probability based on the n-1 previous words.

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

For a 4-gram model, we use MLE estimate the probability a large corpus.

$$P(w_t|w_{t-3}, w_{t-2}, w_{t-1}) = \frac{count(w_{t-3}\, w_{t-2} w_{t-1}\, w_t)}{count(w_{t-3}\, w_{t-2} w_{t-1})}$$

Probability tables

We construct tables to look up the probability of seeing a word given a history.

curse of	$P(w_t \mid w_{t-n} \ldots w_{t-1})$
dimensionality	
azure	
knowledge	
oak	

The tables only store observed sequences.

What happens when we have a new (unseen) combination of n words?

Unseen sequences

What happens when we have a new (unseen) combination of n words?

1. Back-off
2. Smoothing / interpolation

We are basically just stitching together short sequences of observed words.

Alternate idea

Let's try **generalizing**.

Intuition: Take a sentence like

The cat is walking in the bedroom

And use it when we assign probabilities to similar sentences like

The dog is running around the room

Similarity of words / contexts

Use word embeddings!

sim (cat , dog)

Vector for *cat* Vector for *dog*

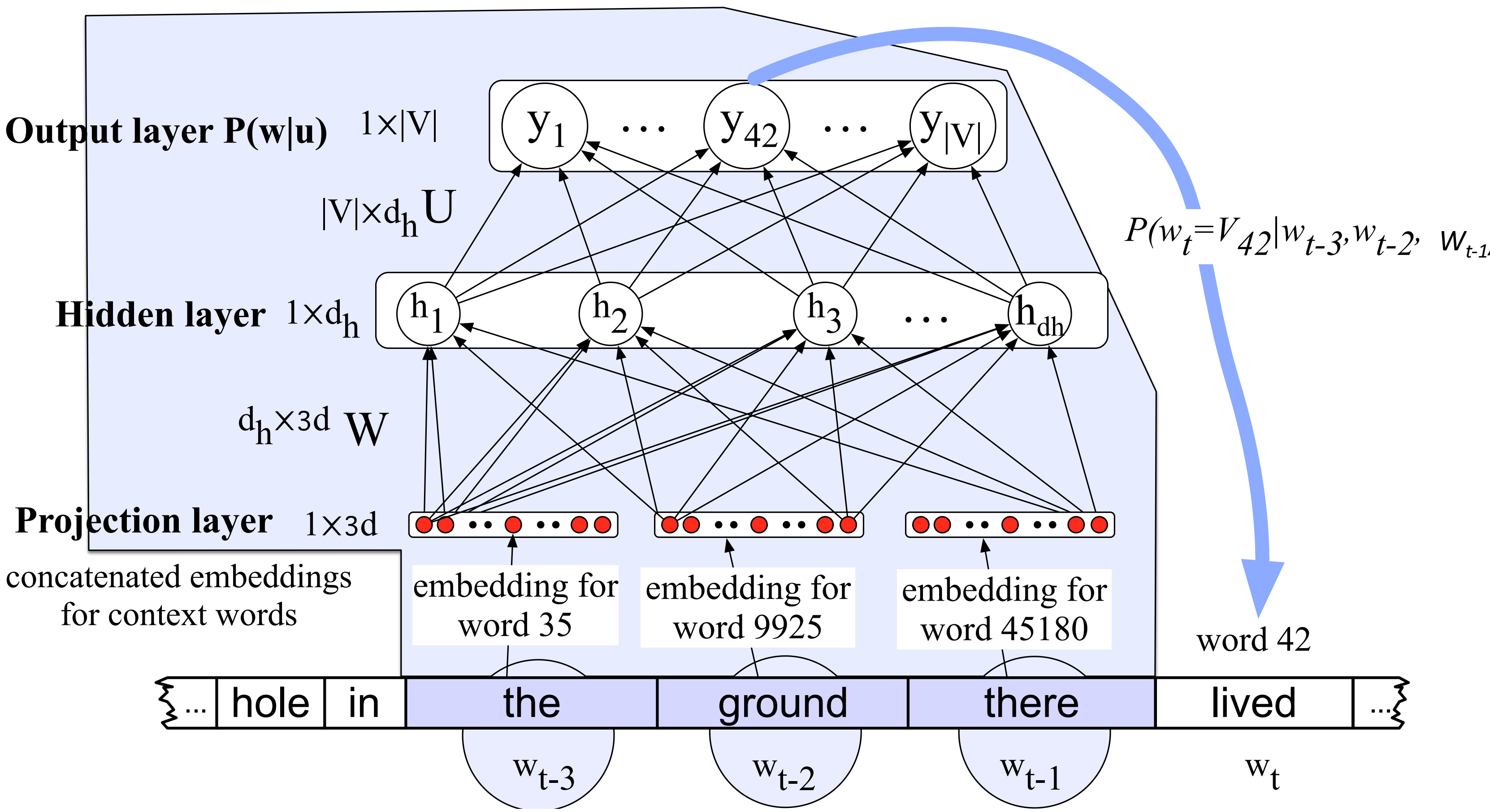
How can we use embeddings to estimate language model probabilities?

p(cat | please feed the)

Concatenate these 3 vectors together, use that as input vector to a feed forward neural network

Compute the probability of all words in the vocabulary with a softmax on the output layer

Neural network with embeddings as input



A Neural Probabilistic LM

In NIPS 2003, Yoshua Begio and his colleagues introduced a neural probabilistic language model

1. They used a vector space model where the words are vectors with real values $\mathbb{R}^m$. m=30, 60, 100. This gave a way to compute word similarity.
2. They defined a function that returns a joint probability of words in a sequence based on a sequence of these vectors.
3. Their model simultaneously learned the word representations **and** the probability function from data.

Seeing one of the cat/dog sentences allows them to increase the probability for that sentence **and** its combinatorial # of **"neighbor" sentences** in vector space.

A Neural Probabilistic LM

Given:

A training set $w_1 \ldots w_t$ where $w_t \in V$

Learn:

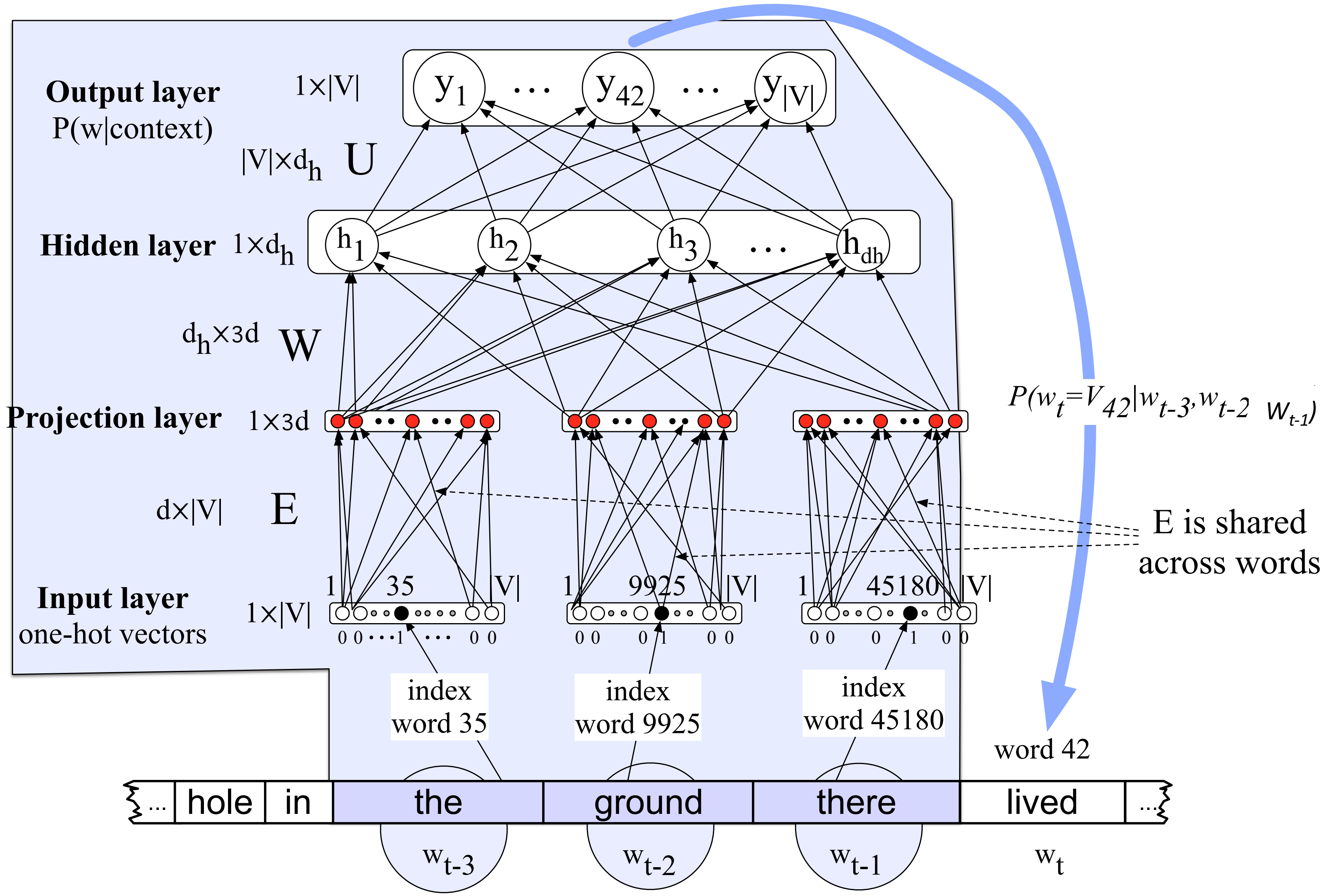
$f(w_1 \ldots w_t) = P(w_t | w_1 \ldots w_{t-1})$

Subject to giving a high probability to an unseen text/dev set (e.g. minimizing the perplexity)

Constraint:

Create a proper probability distribution (e.g. sums to 1) so that we can take the product of conditional probabilities to get the joint probability of a sentence

Neural net that learns embeddings



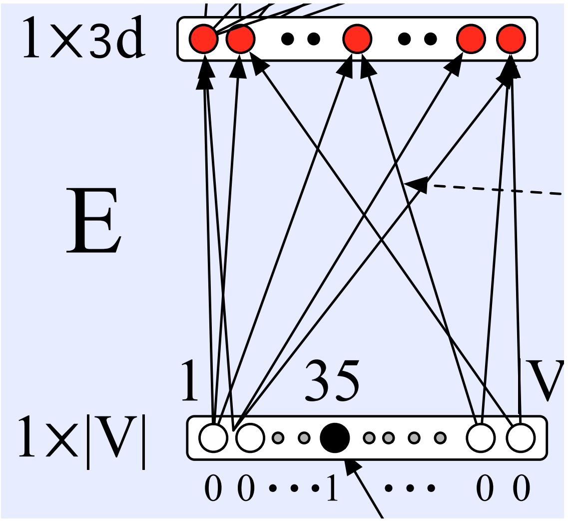
One-hot vectors

To learn the embeddings, we added an extra layer to the network. Instead of pre-trained embeddings as the input layer, we instead use **one-hot vectors.**

```
[0 0 0 0 1 0 0 ... 0 0 0 0]
 1 2 3 4 5 6 7 ...   ... |V|
```

These are then used to look up a **row vector** in the embedding matrix **E**, which is of size d by |V|.

With this small change, we now can learn the emebddings of words.



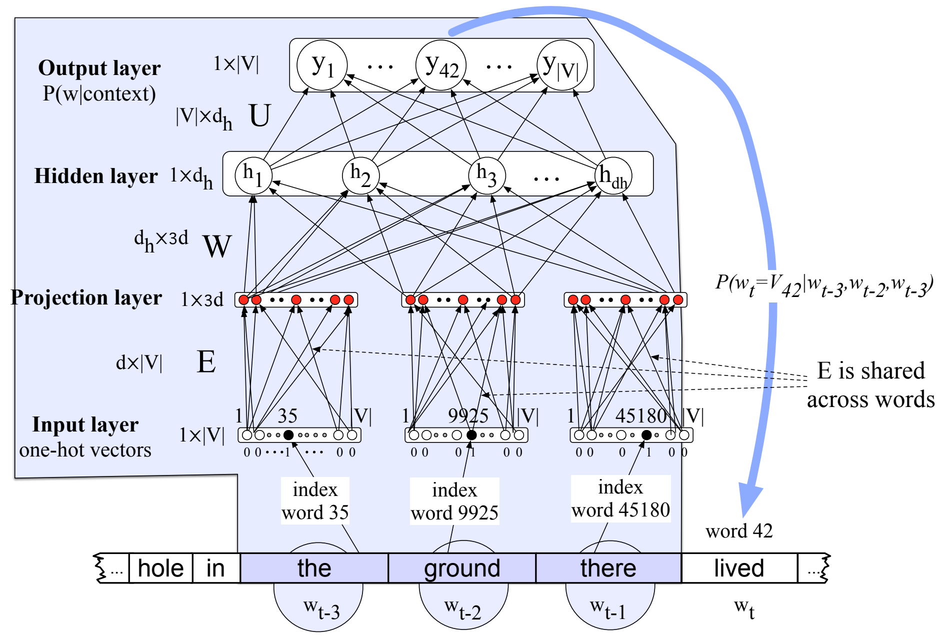
Forward pass

1. Select embeddings from **E** for the three context words (*the ground there*) and concatenate them together

2. Multiply by **W** and add **b** (not shown), and pass it through an activation function (sigmoid, ReLU, etc) to get the hidden layer **h**.

3. Multiply by **U** (the weight matrix for the hidden layer) to get the output layer, which is of size **1 by |V|**.

4. Apply **softmax** to get the probability. Each node *i* in the output layer estimates the probability $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$



$$
\begin{aligned}
e &= (Ex_1, Ex_2, ..., Ex) \\
h &= \sigma(We + b) \\
z &= Uh \\
y &= \text{softmax}(z)
\end{aligned}
$$

Training the neural LM

To train the models we need to find good settings for all of the parameters θ = ***E*,*W*,*U*,*b***.

How do we do it?

Gradient descent using error backpropagation on the computation graph to compute the gradient.

Using a ***cross-entropy loss function.***

Training

The training examples are simply word k-grams from the corpus

The identities of the first k-1 words are used as features, and the last word is used as the target label for the classification.

Conceptually, the model is trained using cross-entropy loss.

Training the neural LM

Use a large text to train. Start with random weights Iteratively moving through the text predicting each word w_t .

At each word w_t, the cross-entropy (negative log likelihood) loss is:

$$L = -\log p(w_t | w_{t-1}, ..., w_{t-n+1})$$

The gradient for the loss is:

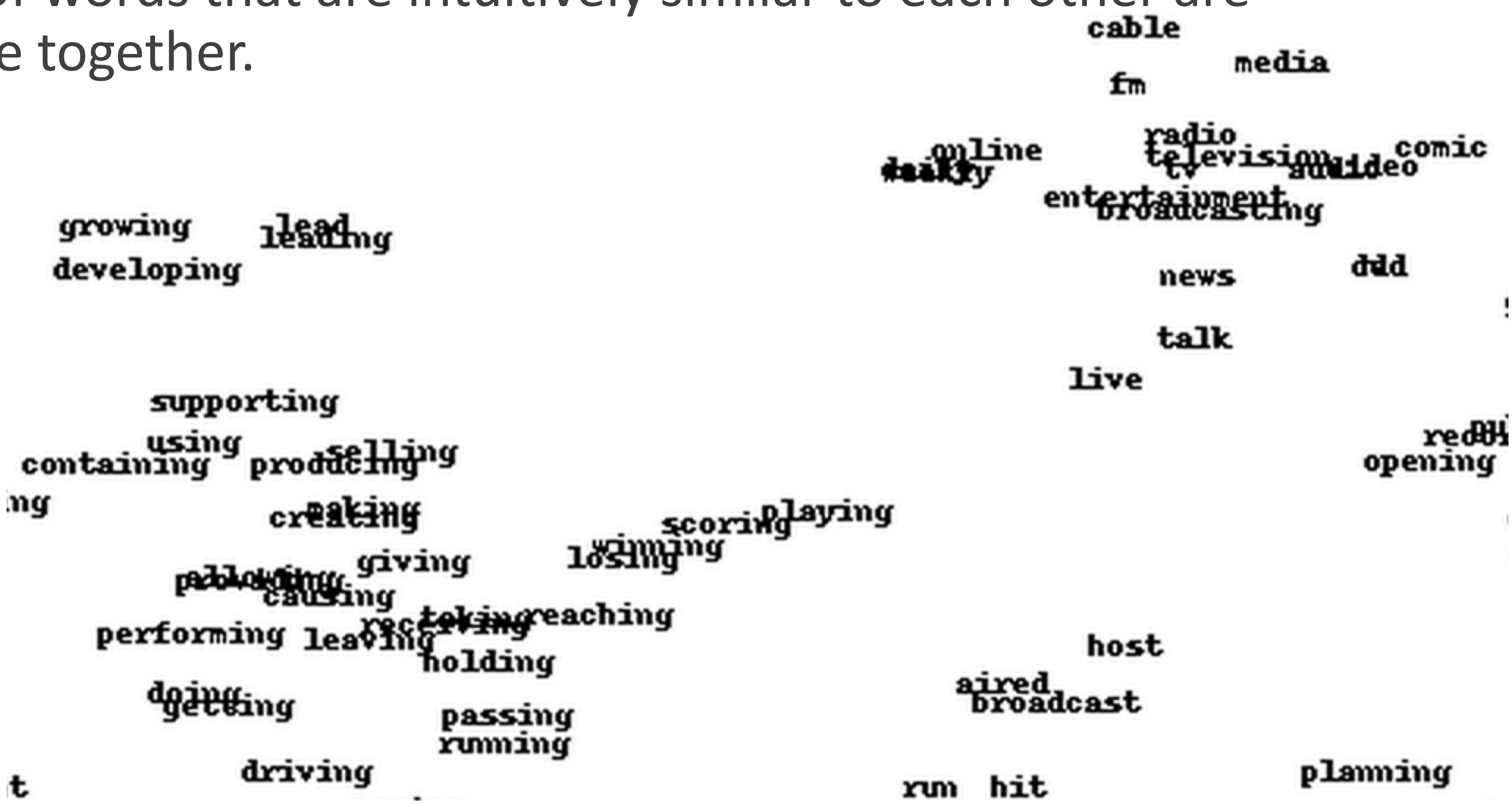
$$\theta_{t+1} = \theta_t - \eta \frac{\partial - \log p(w_t | w_{t-1}, ..., w_{t-n+1})}{\partial \theta}$$

The gradient can be computed in any standard neural network framework which will then backpropagate through ***U* , *W* , *b*, *E***.

The model learns both a function to predict the probability of the next word, **and it learns word emeddings** too!

Learned embeddings

When the ~50 dimensional vectors that result from training a neural LM are projected down to 2-dimensions, we see a lot of words that are intuitively similar to each other are close together.



Advantages of NN LMs

Better results. They achieve better preplexity scores than SOTA n-gram LMs.

Larger N. NN LMs can scale to much larger orders of n. This is achievable because parameters are associated only with individual words, and not with n-grams.

They generalize across contexts. For example, by observing that the words *blue, green, red, black*, etc. appear in similar contexts, the model will be able to assign a reasonable score to the *green car* even though it never observed in training, because it did observe *blue car* and *red car*.

A by-product of training are word embeddings!

Disadvantage of FF-NN

One problem with the Bengio (2003) neural network LM is that it operates only on **fixed size inputs**.

For sequences longer than that size, it slides a window over the input, and makes predictions as it goes.

The decision for one window **has no impact** on the later decisions.

This shares the **weakness of Markov** approaches, because it limits the context to the window size.

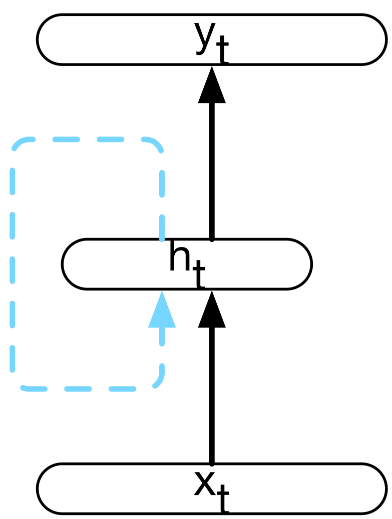
To fix this, we're going to look at **recurrent neural networks**.

Recurrent Neural Networks

A recurrent neural network (RNN) is any network that contains a cycle within its network.

In such networks the value of a unit can be dependent on earlier outputs as an input.

RNNs have proven extremely effective when applied to NLP.



Memory

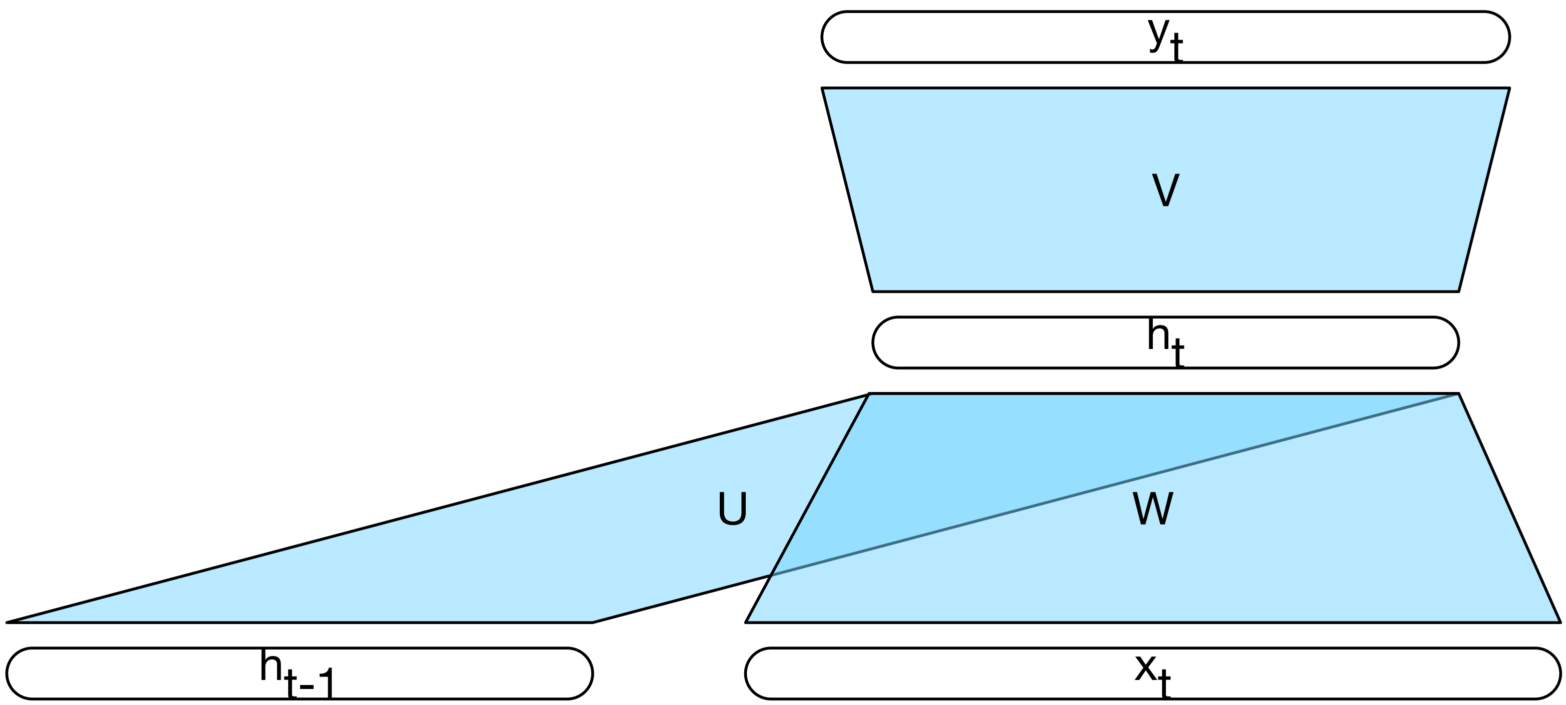
We use a hidden layer from a **preceding point in time** to augment the input layer.

This hidden layer from the preceding point in time provides a form of **memory** or context.

This architecture **does not impose a fixed-length limit** on its prior context.

As a result, information can come from all the way back at the beginning of the input sequence.   Thus we get away from the Markov assumption.

RNN as a feedforward network



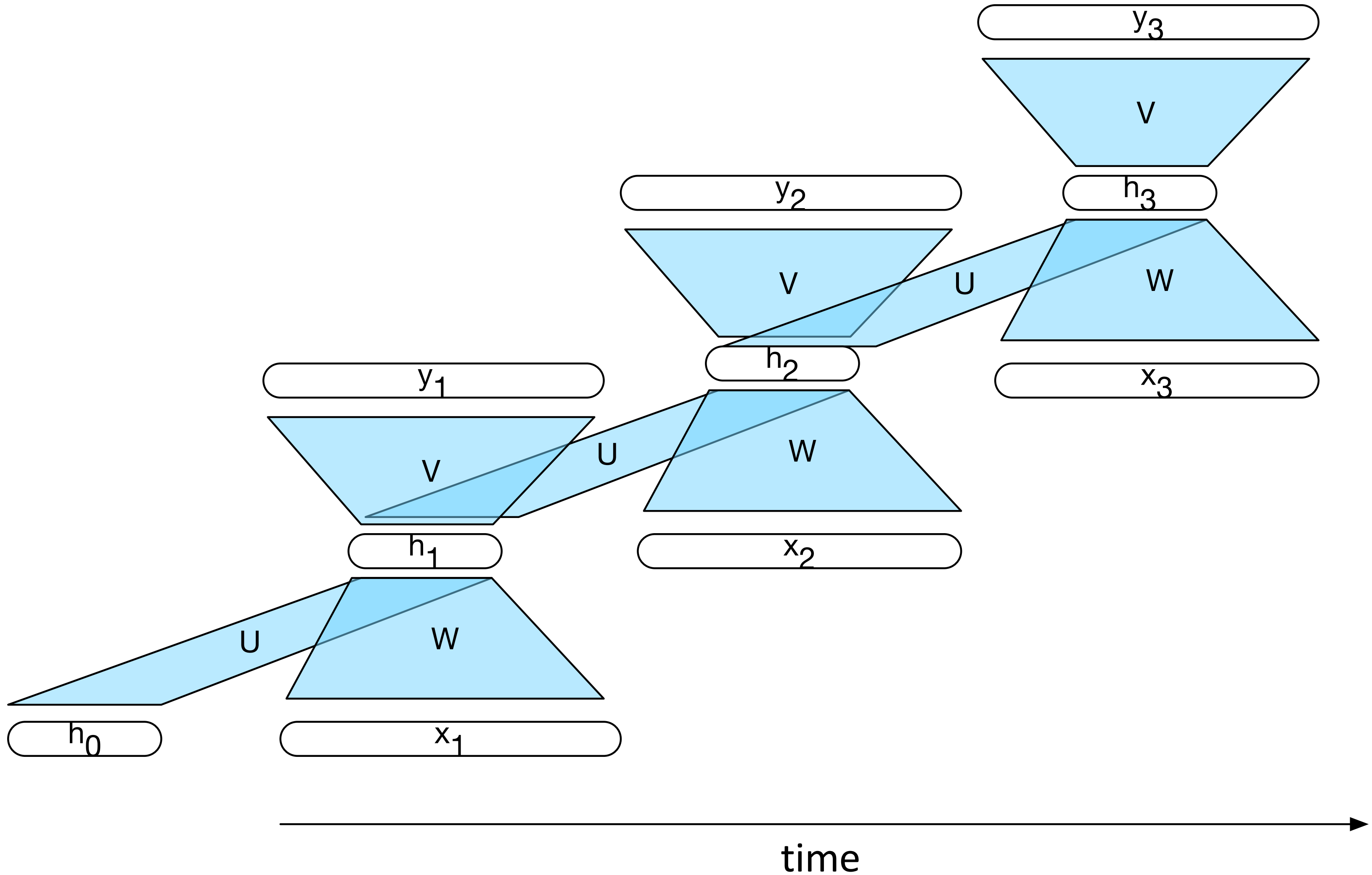
Forward inference

function FORWARDRNN(x, $network$) **returns** output sequence y

$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
  $h_i \leftarrow g(U\ h_{i-1}\ +\ W\ x_i)$
  $y_i \leftarrow f(V\ \ h_i)$
return y

This allows us to have an output sequence equal in length to the input sequece.

Unrolled RNN



Training RNNs

Just like with feedforward networks, we'll use a training set, a loss function, and back-propagation to get the gradients needed to adjust the weights in an RNN.

The weights we need to update are:

W – the weights from the input layer to the hidden layer
U – the weights from the previous hidden layer to the current hidden layer
V – the weights from the hidden layer to the output layer

Training RNNs

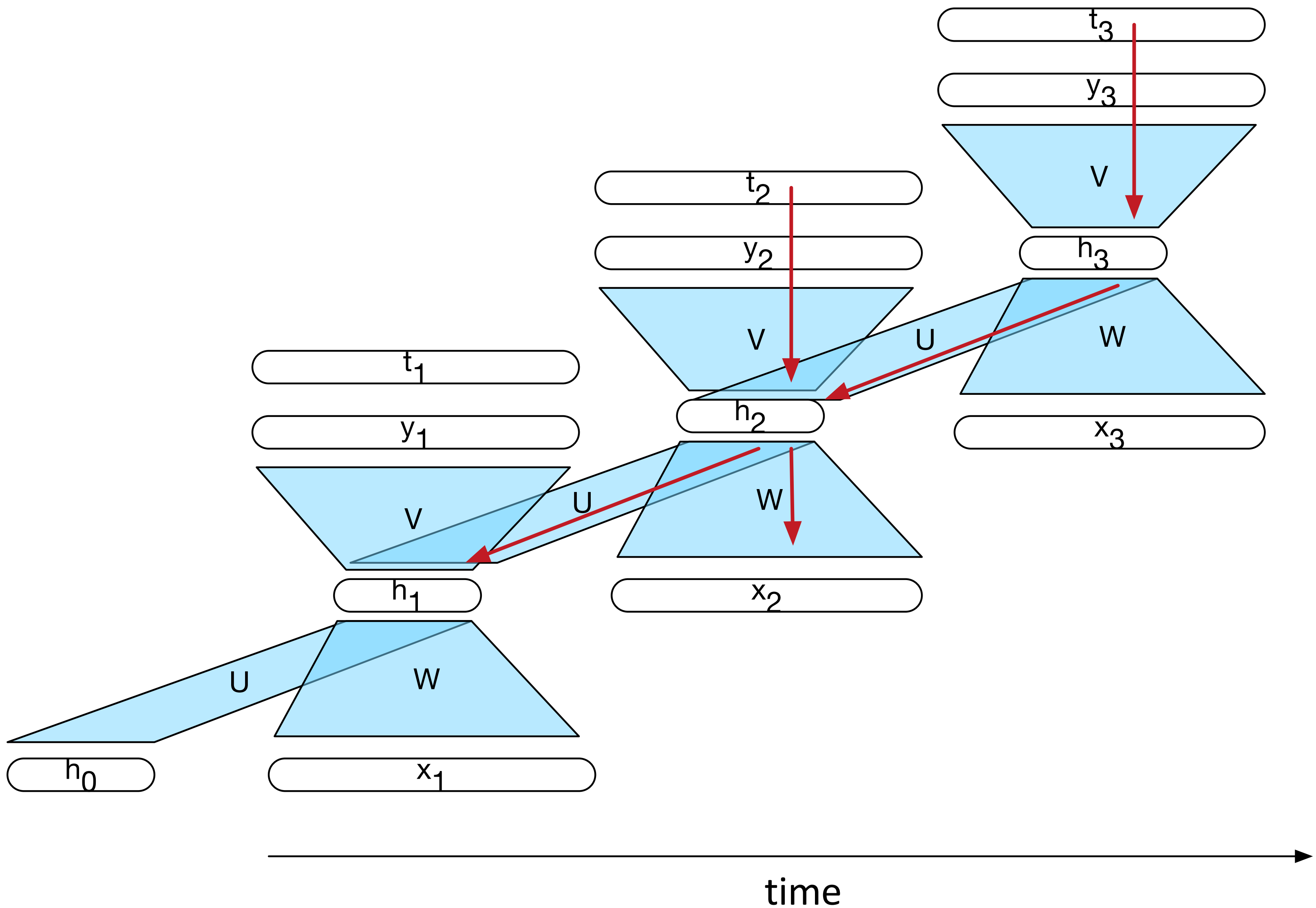
New considerations:

1. to compute the loss function for the output at time t we need the hidden layer from time $t - 1$.

2. The hidden layer at time t influences both the output at time t and the hidden layer at time $t + 1$ (and hence the output and loss at t+1)

To assess the error accruing to h_t , we'll need to know its influence on both the current output *as well as the ones that follow*.

Backpropagation of errors



Recurrent Neural Language Models

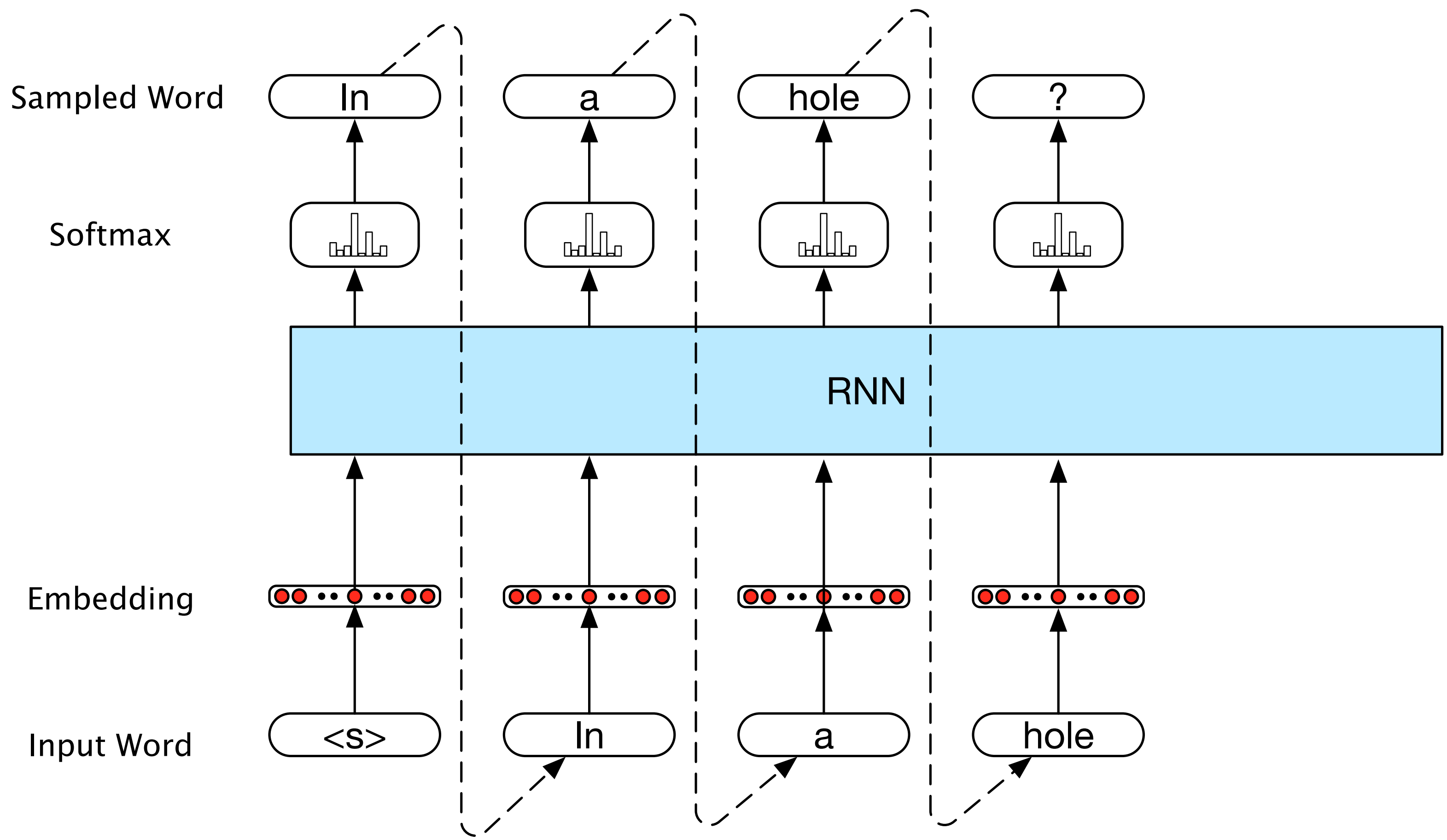
Unlike n-gram LMs and feedforward networks with sliding windows, RNN LMs don’t use a fixed size context window.

They predict the next word in a sequence by using the current word and the previous hidden state as input.

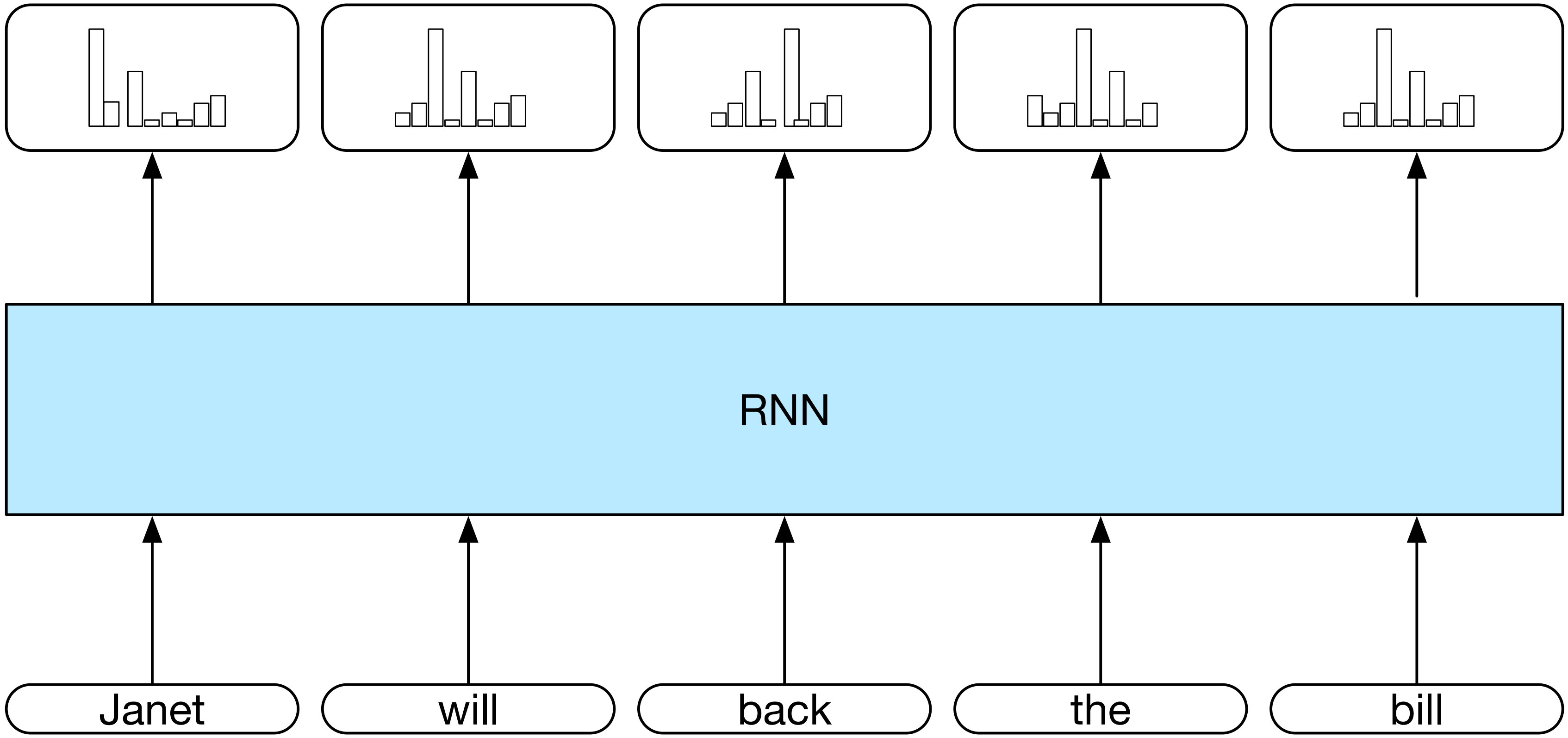
The hidden state embodies information about all of the preceding words all the way back to the beginning of the sequence.

Thus they can potentially take more context into account than n-gram LMs and NN LMs that use a sliding window.

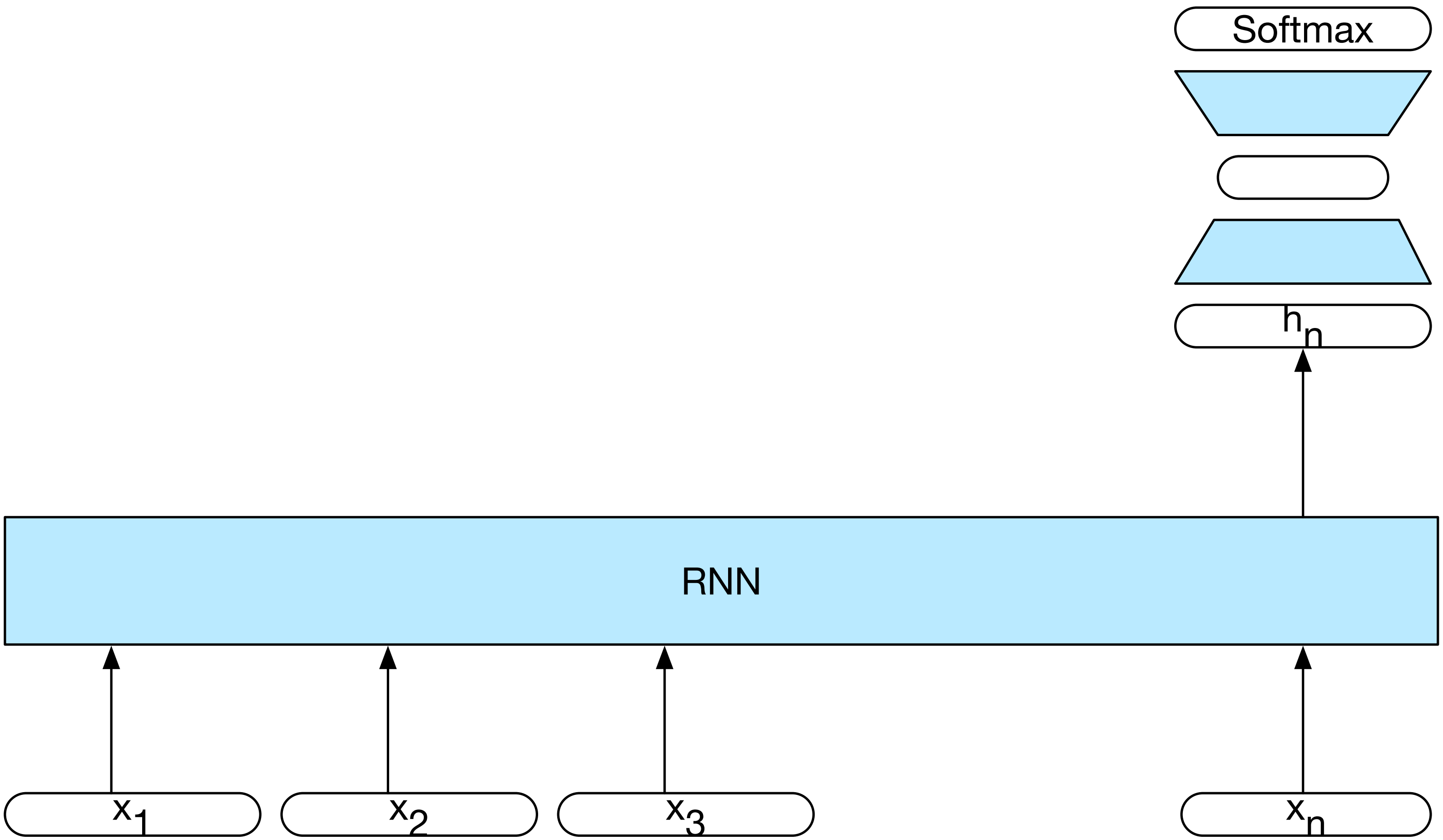
Autoregressive generation with an RNN LM



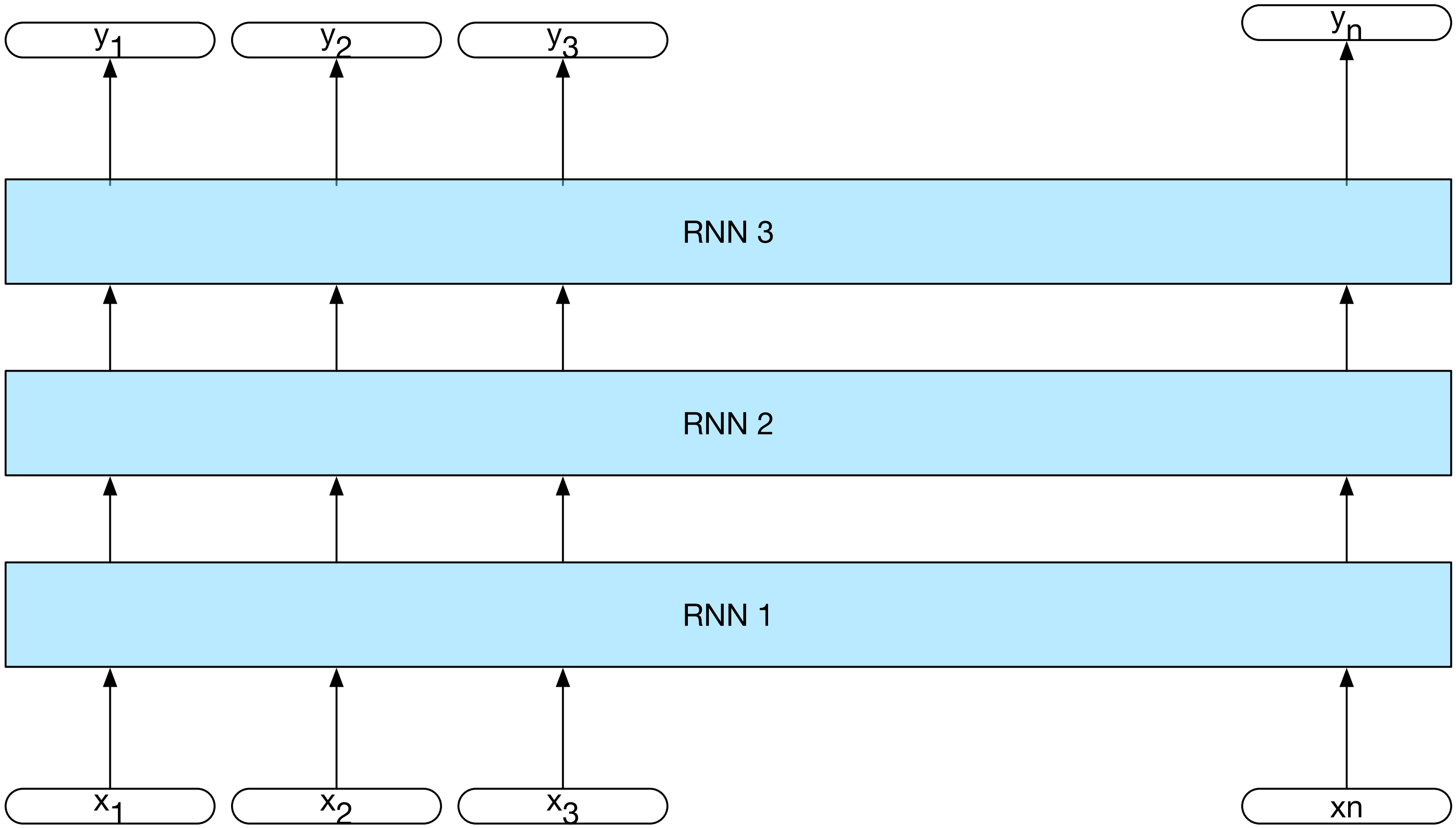
Tag Sequences



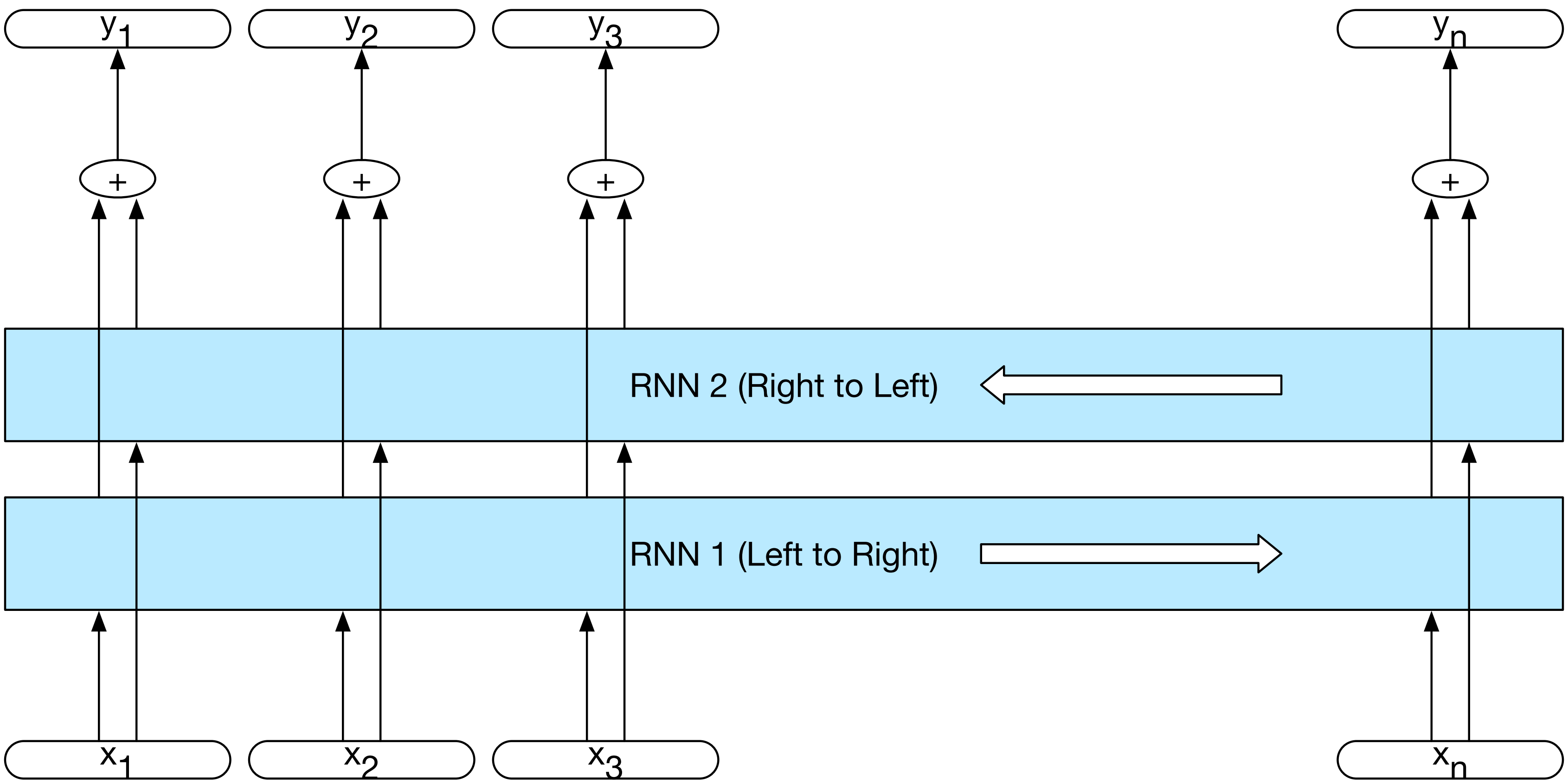
Sequence Classifiers



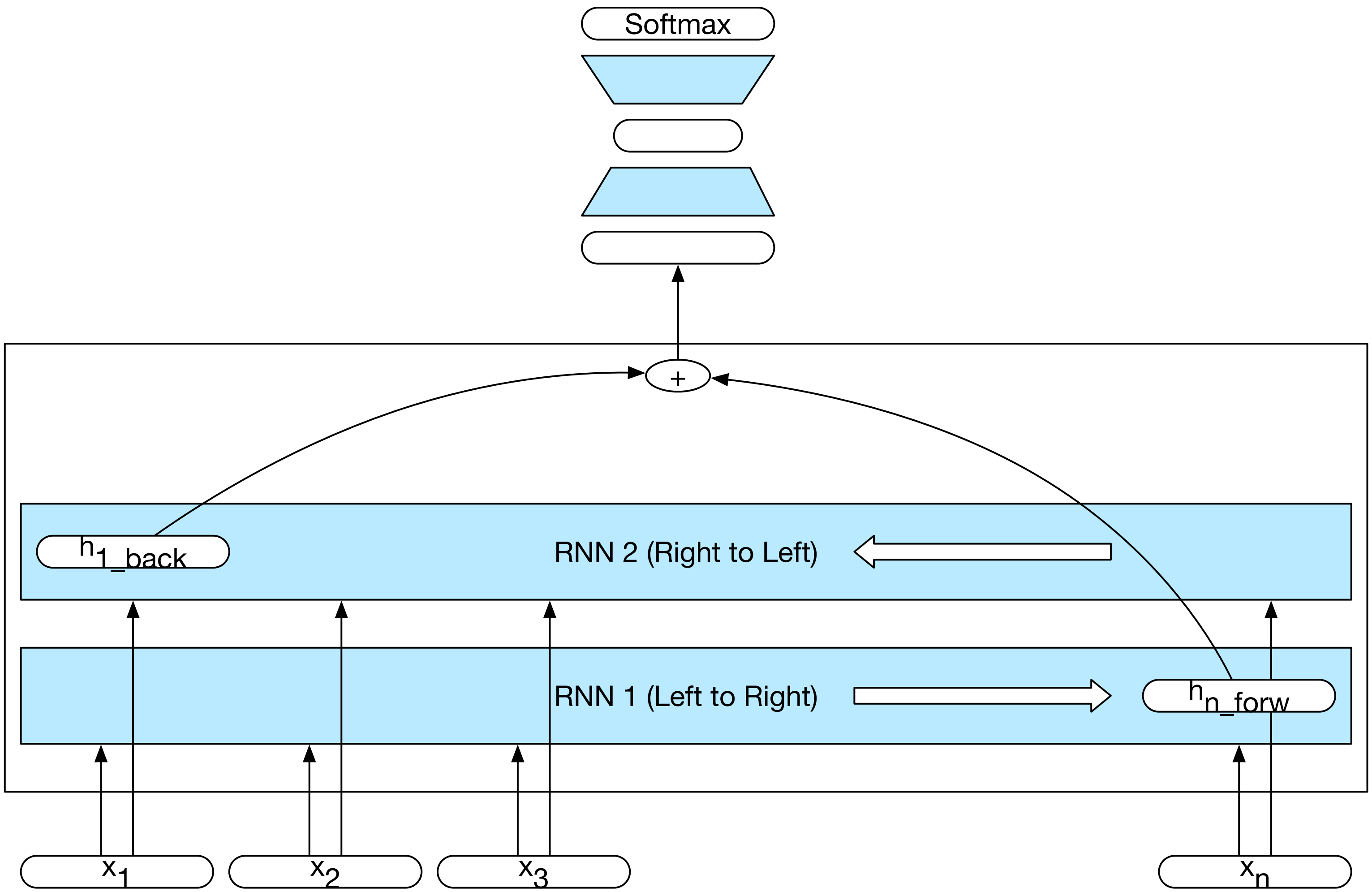
Stacked RNNs



Bidirectional RNNs



Bidirectional RNNs for sequence classification



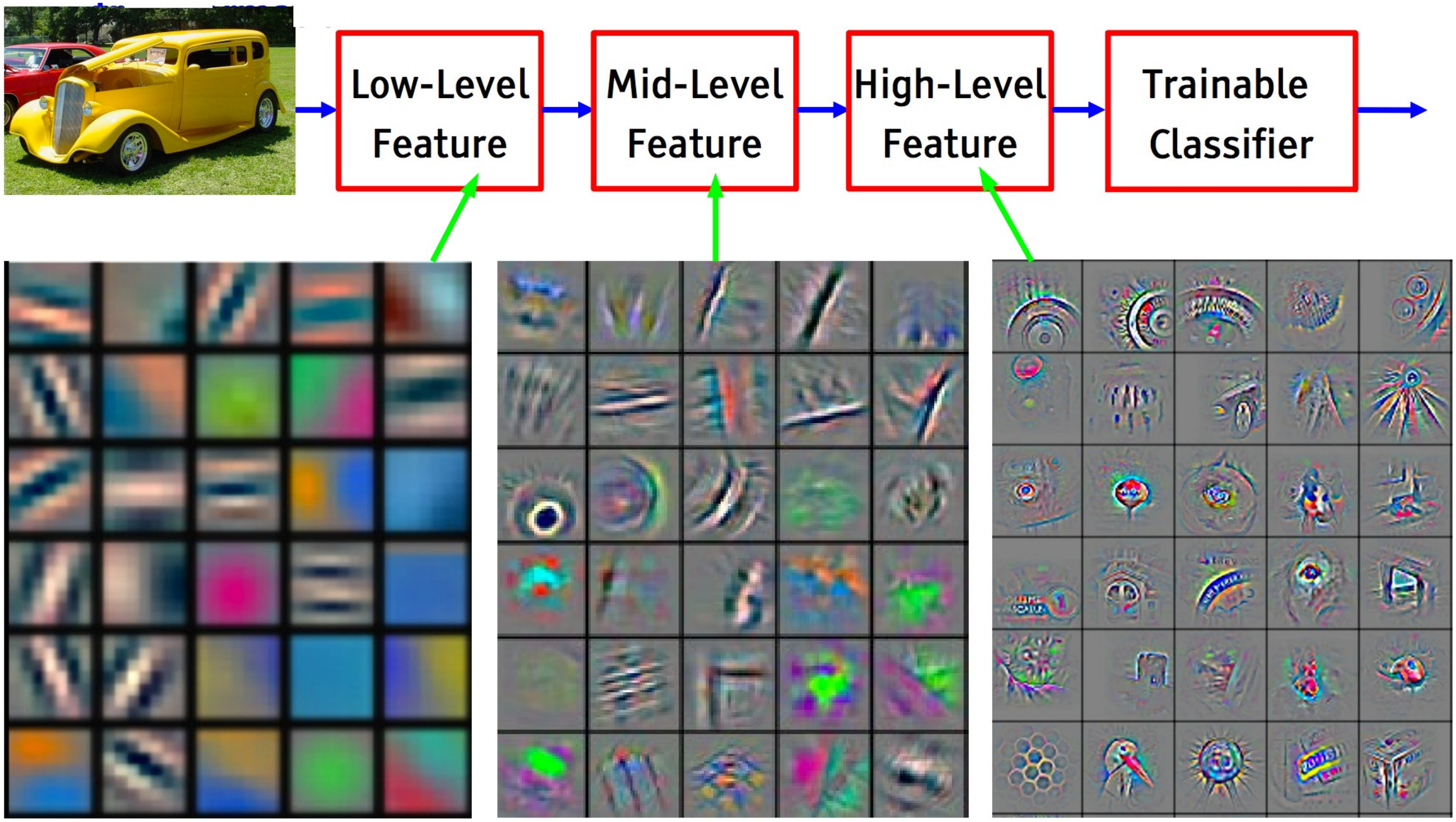
Current state of the art neural LMs

[ELMo]

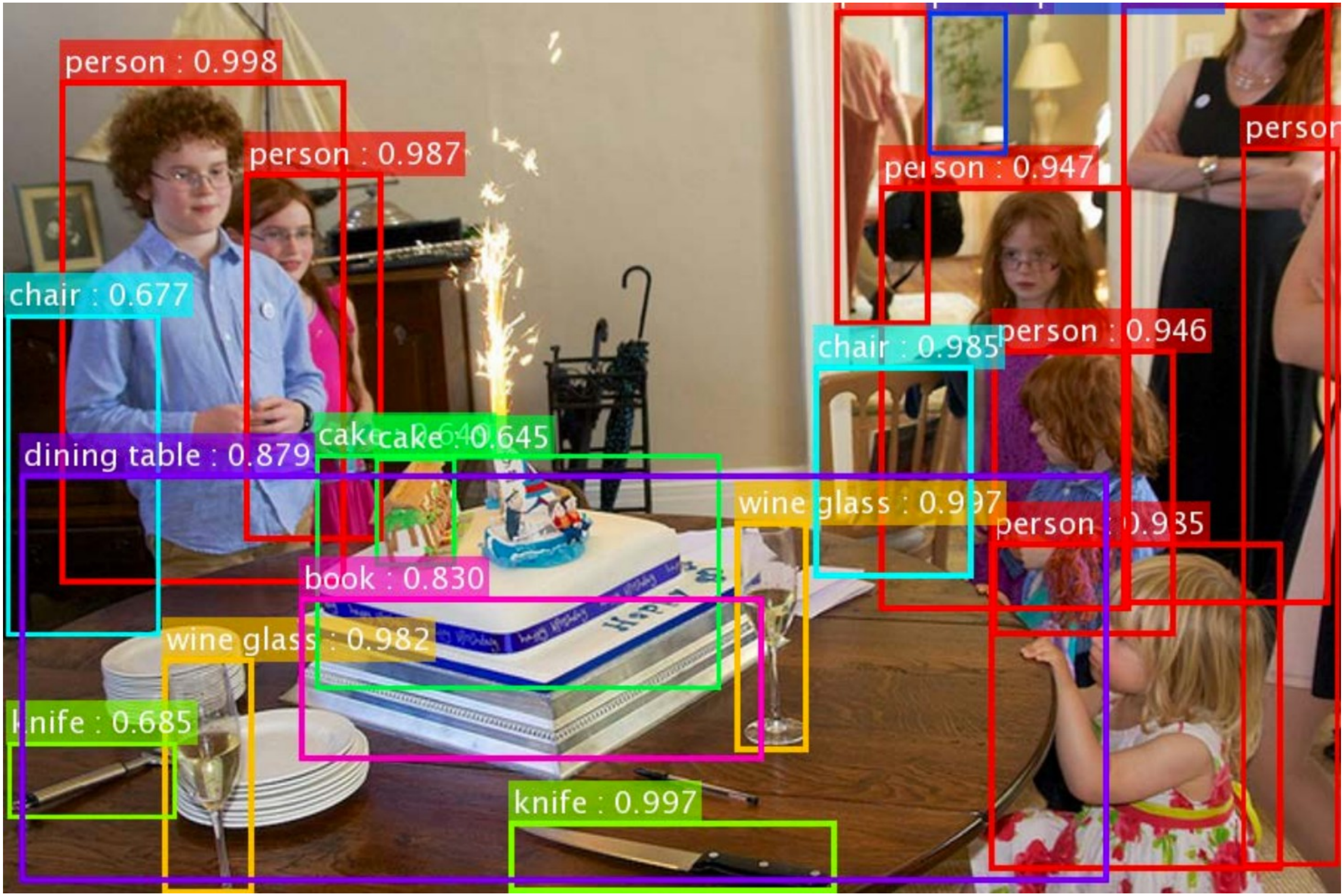
[GPT]

[BERT]

[GPT-2]



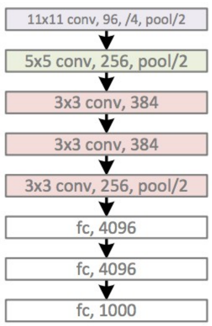
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]


person : 0.998
person : 0.987
chair : 0.677
dining table : 0.879
cake : 0.645
book : 0.830
wine glass : 0.982
knife : 0.997
chair : 0.985
person : 0.946

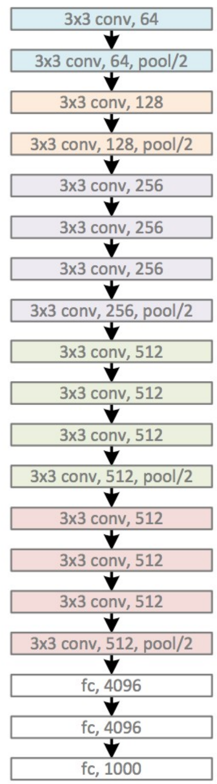
Microsoft Research

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

ICCV15 International Conference on Computer Vision

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Oren Etzioni
@etzioni

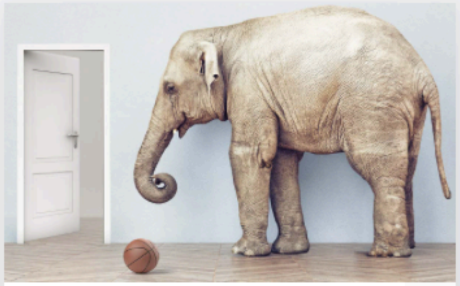
Following

The winner of the 2014 ImageNet competition had 4 million parameters, while the winner of the 2017 challenge had 145.8 million parameters - a 36X increase in three years. source: @jackclarkSF Shall we increase parameters by another 36X, or solve more interesting problems?1

9:21 PM - 27 Nov 2018

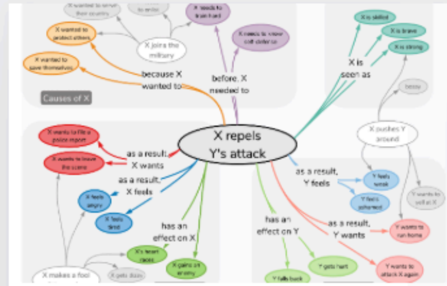
63 Retweets **222** Likes

12 63 222



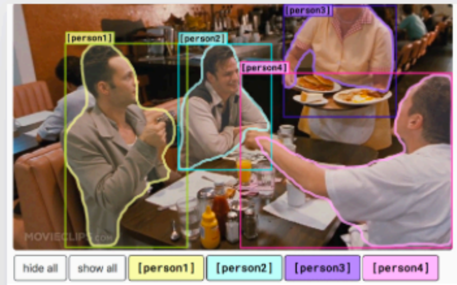
Mosaic Commonsense Benchmarks

To measure progress in machine common sense, we're developing a suite of benchmark datasets.



Commonsense Knowledge Graphs

Knowledge graphs provide useful semi-structured representations of commonsense. Currently, we're developing and exploring how to better use such graphs with current models.

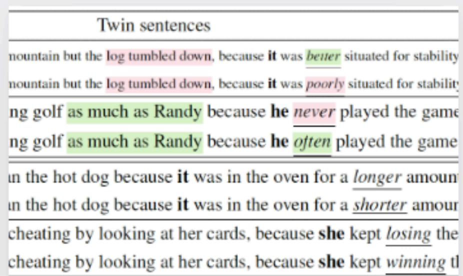


Visual Commonsense Reasoning

Visual Commonsense Reasoning (VCR) is a new task and large-scale dataset for cognition-level visual understanding.

SWAG: Situations with Adversarial Generations

SWAG (Situations With Adversarial Generations) is a large-scale dataset for the task of grounded commonsense inference, unifying natural language inference and physically grounded reasoning.



WinoGrande: Adversarial Winograd Schema Challenge at Scale

Winogrande is a large-scale Winograd Schema Challenge dataset, adjusted to improve both the scale and the hardness of the dataset.

What you can do next at Penn

Machine Learning – CIS 419 and CIS 520

Deep Learning – CIS 522

Computational Linguistics – CIS 530

Special Topics next semester:

* CIS 700-008 – Interactive Fiction and Text Generation

* CIS 700-003 – Explainable AI

* CIS 700-001 – Reasoning for Natural Language Understanding

Thank you!

Good luck with your exams!