

Lecture 10: Recurrent Neural Networks

Instructor: Jackie CK Cheung
COMP-550

Primer by Yoav Goldberg:

<https://arxiv.org/abs/1510.00726>

Eisenstein, Section 7.6

J&M Chapter 9 (3rd ed)

Reminders

- RA1 coming out this Thursday, due Oct 16
- Friday Oct 6 to Wednesday Oct 11: Fall reading break
- Tutorials on probability and information theory!
 - Check Ed for schedule

Outline

Review of neural networks and deep learning

Recurrent neural networks

Long short-term memory networks

LSTM-CRFs

(Artificial) Neural Networks

A kind of learning model which automatically learns non-linear functions from input to output

Biologically inspired metaphor:

- Network of computational units called neurons
- Each neuron takes scalar inputs, and produces a scalar output, very much like a logistic regression model

$$\text{Neuron}(\vec{x}) = g(a_1x_1 + a_2x_2 + \dots + a_nx_n + b)$$

As a whole, the network can theoretically compute any computable function, given enough neurons. (These notions can be formalized.)

Feedforward Neural Networks

All connections flow forward (no loops); each layer of hidden units is fully connected to the next.

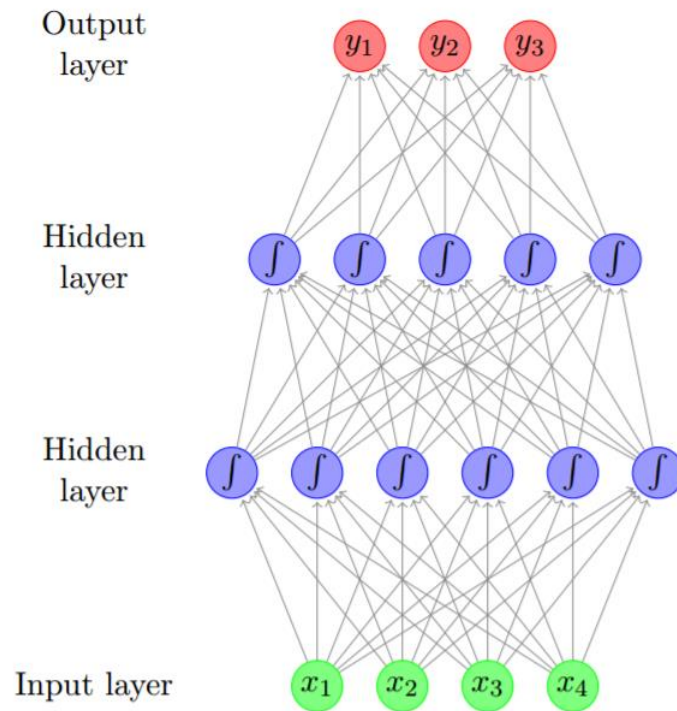


Figure 2: Feed-forward neural network with two hidden layers.

Figure from Goldberg (2015)

Inference in a FF Neural Network

Perform computations forwards through the graph:

$$\mathbf{h}^1 = g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)$$
$$\mathbf{h}^2 = g^2(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2)$$
$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3$$

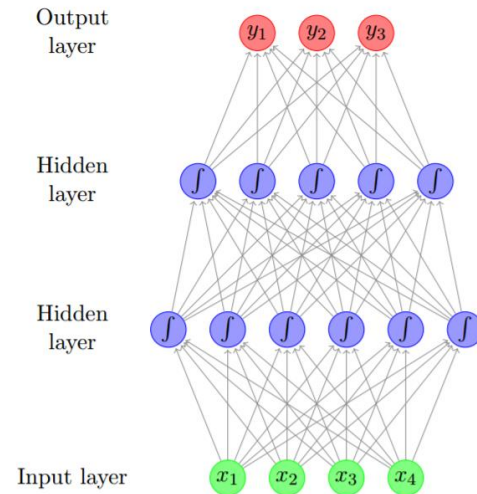


Figure 2: Feed-forward neural network with two hidden layers.

Note that we are now representing each layer as a vector; combining all of the weights in a layer across the units into a weight matrix

Training Neural Networks

Typically done by **stochastic gradient descent**

- For one training example, find gradient of loss function wrt parameters of the network (i.e., the weights of each layer); “travel along in that direction”.

Network has very many parameters!

Efficient algorithm to compute the gradient with respect to all parameters: **backpropagation** (Rumelhart et al., 1986)

- Boils down to an efficient way to use the chain rule of derivatives to propagate the error signal from the loss function backwards through the network back to the inputs

SGD Overview

Inputs:

- Function computed by neural network, $f(\mathbf{x}; \theta)$
- Training samples $\{\mathbf{x}^k, \mathbf{y}^k\}$
- Loss function L

Repeat for a while:

Sample a training case, $\mathbf{x}^k, \mathbf{y}^k$

Compute loss $L(f(\mathbf{x}^k; \theta), \mathbf{y}^k)$

Forward pass

Compute gradient $\nabla L(\mathbf{x}^k)$ wrt the parameters θ

Update $\theta \leftarrow \theta - \eta \nabla L(\mathbf{x}^k)$

In neural networks,
by backpropagation

Return θ

Example: Forward Pass

$$\mathbf{h}^1 = g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)$$

$$\mathbf{h}^2 = g^2(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2)$$

$$f(\mathbf{x}) = \mathbf{y} = g^3(\mathbf{h}^2) = \mathbf{h}^2\mathbf{W}^3$$

Loss function: $L(\mathbf{y}, \mathbf{y}^{gold})$

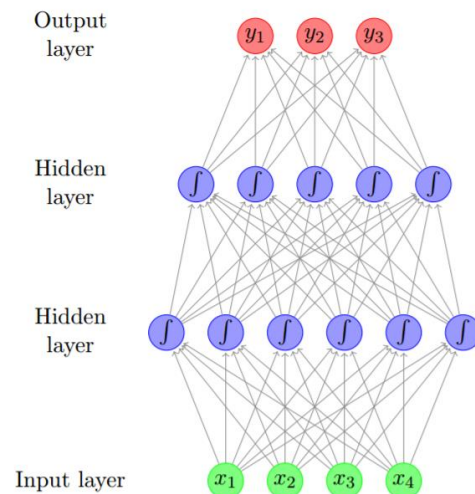


Figure 2: Feed-forward neural network with two hidden layers.

Save the values for $\mathbf{h}^1, \mathbf{h}^2, \mathbf{y}$ too!

Example: Time Delay Neural Network

Let's draw a neural network architecture for POS tagging using a feedforward neural network.

We'll construct a context window around each word, and predict the POS tag of that word as the output.

Limitations of this approach?

Recurrent Neural Networks

A neural network **sequence model**:

$$RNN(s_0, \mathbf{x}_{1:n}) = \mathbf{s}_{1:n}, \mathbf{y}_{1:n}$$

$$\mathbf{s}_i = R(\mathbf{s}_{i-1}, \mathbf{x}_i) \quad \# \mathbf{s}_i : \text{state vector}$$

$$\mathbf{y}_i = O(\mathbf{s}_i) \quad \# \mathbf{y}_i : \text{output vector}$$

R and O are parts of the neural network that compute the next state vector and the output vector

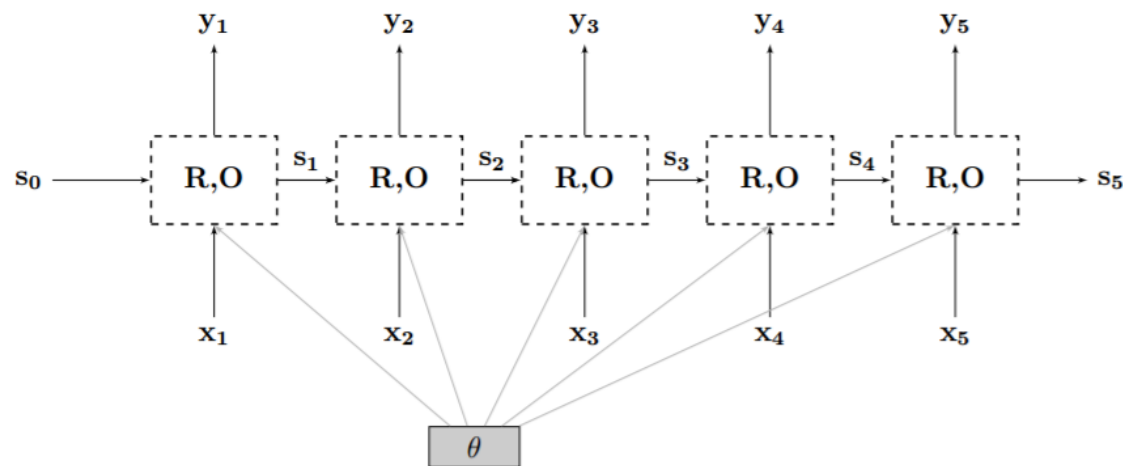


Figure 6: Graphical representation of an RNN (unrolled).

Long-Term Dependencies in Language

There can be dependencies between words that are *arbitrarily* far apart.

- *I will look the word that you have described that doesn't make sense to her up.*
- Can you think of some other examples in English of long-range dependencies?

Cannot easily model with HMMs or even LC-CRFs, but can with RNNs

Comparing LC-CRFs and RNNs

	LC-CRFs	RNNs
At each timestep	Linear model (linear between feature values and weights) similar to logistic regression	A neural network (more expressive model, may need more data to train)
Feature engineering		
Inference properties		

Comparing LC-CRFs and RNNs

	LC-CRFs	RNNs
At each timestep	Linear model (linear between feature values and weights) similar to logistic regression	A neural network (more expressive model, may need more data to train)
Feature engineering	Necessary for performance	Less critical; neural network can learn useful features given enough labelled data
Inference properties		

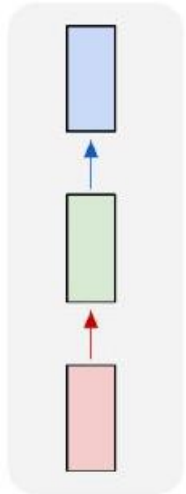
Comparing LC-CRFs and RNNs

	LC-CRFs	RNNs
At each timestep	Linear model (linear between feature values and weights) similar to logistic regression	A neural network (more expressive model, may need more data to train)
Feature engineering	Necessary for performance	Less critical; neural network can learn useful features given enough labelled data
Inference properties	Exact polynomial-time inference due to strong local independence assumptions	Approximate inference only (greedy or beam search)

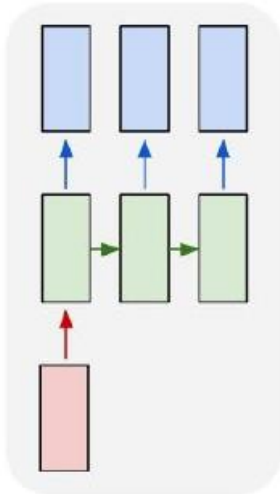
Different RNN Architectures

Different architectures for different use cases

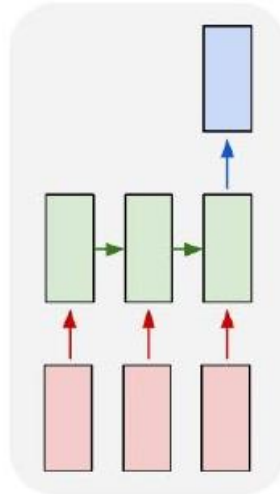
one to one



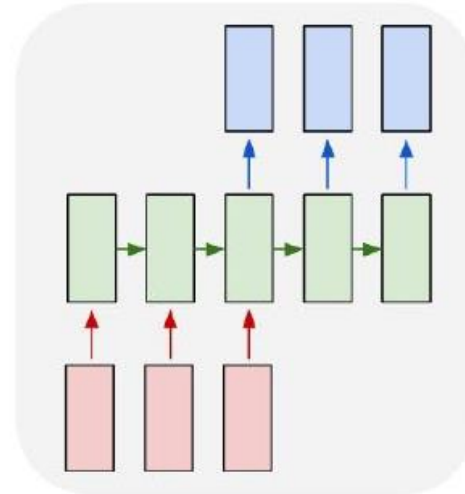
one to many



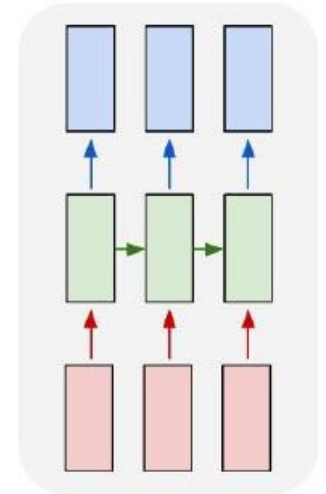
many to one



many to many



many to many



Which architecture would you use for:

- document classification?
- language modelling?
- POS tagging?

Long Short-Term Memory Networks

Popular RNN architecture for NLP

(Hochreiter and Schmidhuber, 1997)

Model includes a “memory” cell

- A vector of weights in a hidden layer
- Learned operations to:
 - Forget old information
 - Extract new information
 - Integrate relevant information into memory
 - Predict output at current timestep

Basic Operations / Components

Masking

- Multiply a vector by numbers between 0 and 1 to model forgetting vs. retaining information
- **Ingredients:** Sigmoid function, component-wise multiplication

Integrating information

- Done via component-wise addition and concatenation

Non-linearities:

- Sigmoid function (output between 0 and 1)
- Tanh function (output between -1 and 1)

Visual step-by-step explanation

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Vanishing and Exploding Gradients

If R and O are simple fully connected layers, we have a problem. In the unrolled network, the gradient signal can get lost on its way back to the words far in the past:

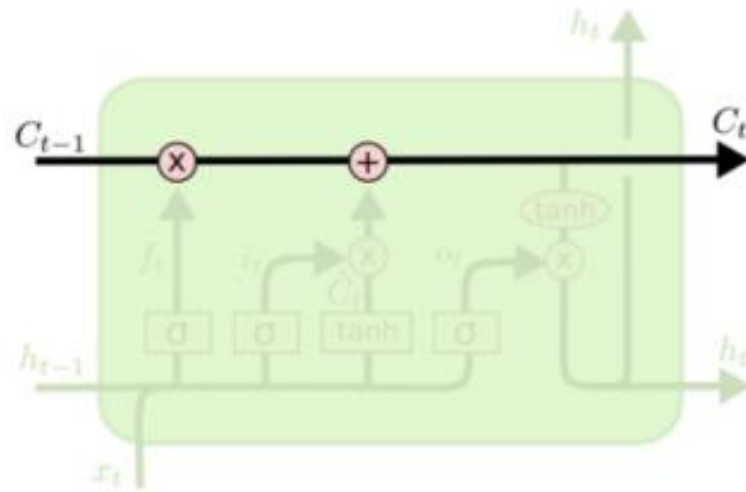
- Suppose it is \mathbf{W}^1 that we want to modify, and there are N layers between that and the loss function.

$$\frac{\partial L}{\partial \mathbf{W}^1} = \frac{\partial L}{\partial g^N} \frac{\partial g^N}{\partial g^{N-1}} \cdots \frac{\partial g^2}{\partial g^1} \frac{\partial g^1}{\partial \mathbf{W}^1}$$

- If the gradient norms are small (<1), the gradient will vanish to near-zero (or explode to near-infinity if >1)
- This happens especially because we have repeated applications of the same weight matrices in the recurrence

Fix for Vanishing Gradients

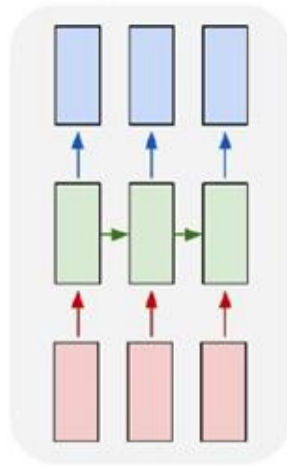
In LSTMs, we can propagate a cell state directly, fixing the vanishing gradient problem:



There is no repeated weight application between the internal states across time!

Bidirectional LSTMs – Motivation

Standard LSTMs go forward in time

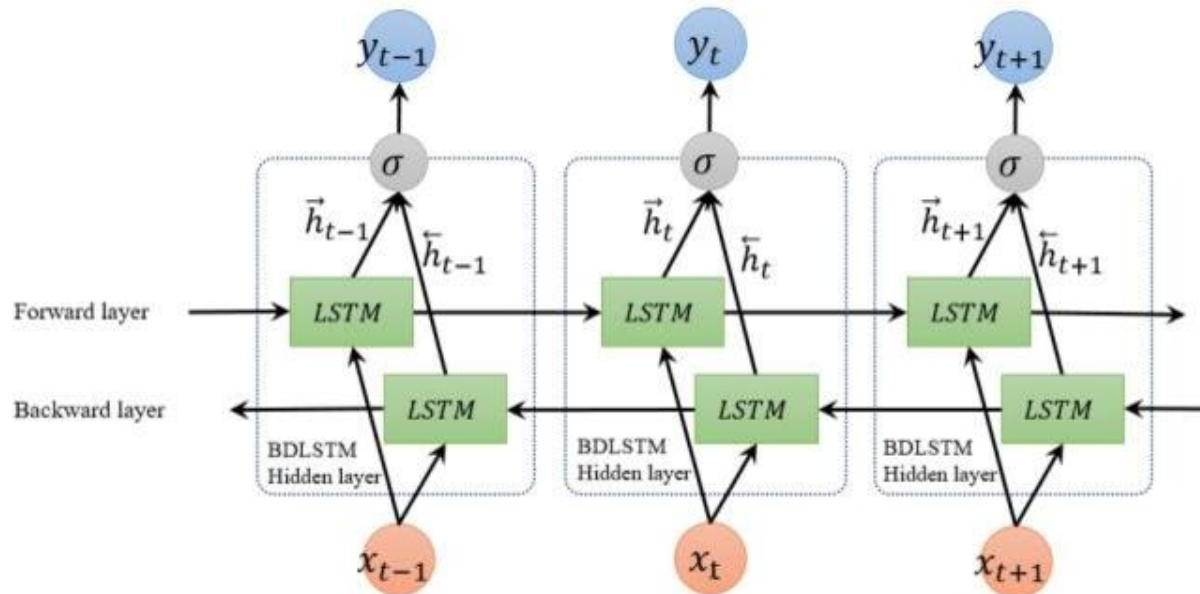


For some applications, this is necessary

For others, we would like to use both *past* context and *future* context

BiLSTMs

Have two LSTM layers, forward and backward in time



Concatenate their outputs to make final prediction

Combining LSTMs and CRFs

LSTMs allow learning of complex relationships between input and output

(LC-)CRFs allow learning of relationships between output labels in a sequence

Can we combine advantages of both?

LSTM-CRFs

Add a linear-chain CRF layer on top of a (Bi)LSTM
(Huang et al., 2015; Lample et al., 2016)!

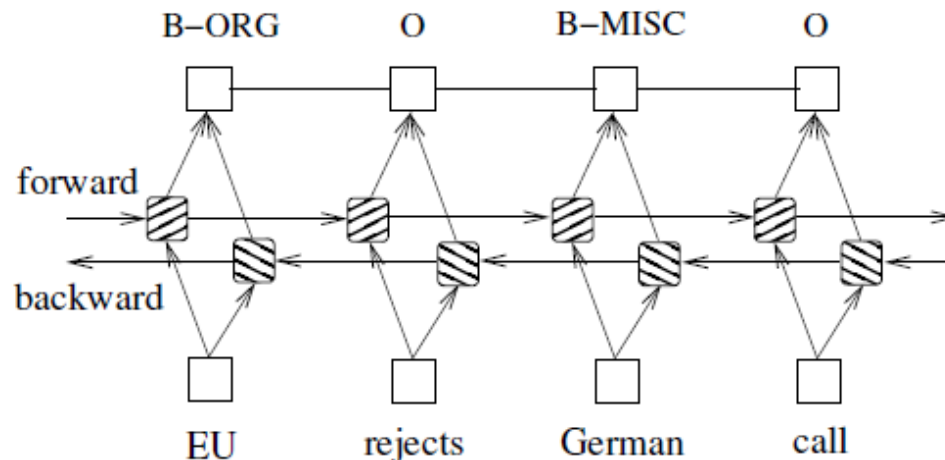


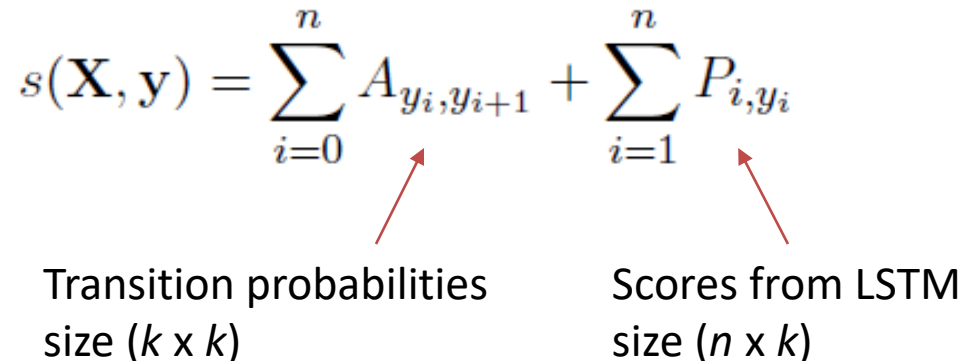
Figure 7: A BI-LSTM-CRF model.

LSTM-CRF

Features of the CRF:

- The output scores of the LSTM
- Transition probabilities between tags (need to be learned)

Score of a sequence in LSTM-CRF:

$$s(\mathbf{X}, \mathbf{y}) = \sum_{i=0}^n A_{y_i, y_{i+1}} + \sum_{i=1}^n P_{i, y_i}$$


Transition probabilities
size $(k \times k)$

Scores from LSTM
size $(n \times k)$

sequence of length n , with k possible tags

LSTM-CRF: Inference and Training

The LSTM-CRF is trained to minimize negative log likelihood on the training corpus (same as regular CRF):

Algorithm 1 Bidirectional LSTM CRF model training procedure

```
1: for each epoch do
2:   for each batch do
3:     1) bidirectional LSTM-CRF model forward pass:
4:       forward pass for forward state LSTM
5:       forward pass for backward state LSTM
6:     2) CRF layer forward and backward pass
7:     3) bidirectional LSTM-CRF model backward pass:

8:       backward pass for forward state LSTM
9:       backward pass for backward state LSTM
10:    4) update parameters
11:   end for
12: end for
```

Summary

LSTMs are the backbone of many modern NLP models!

LSTM-CRFs are very high performing on many basic sequence labelling tasks:

- Named entity recognition
- POS tagging
- Chunking
- Competitive with Transformers without language modelling pre-training.

Next, we will start looking at models of hierarchical structure in language.