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)

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

$$\text{Neuron}(\vec{x}) = g(a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + b)$$

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.

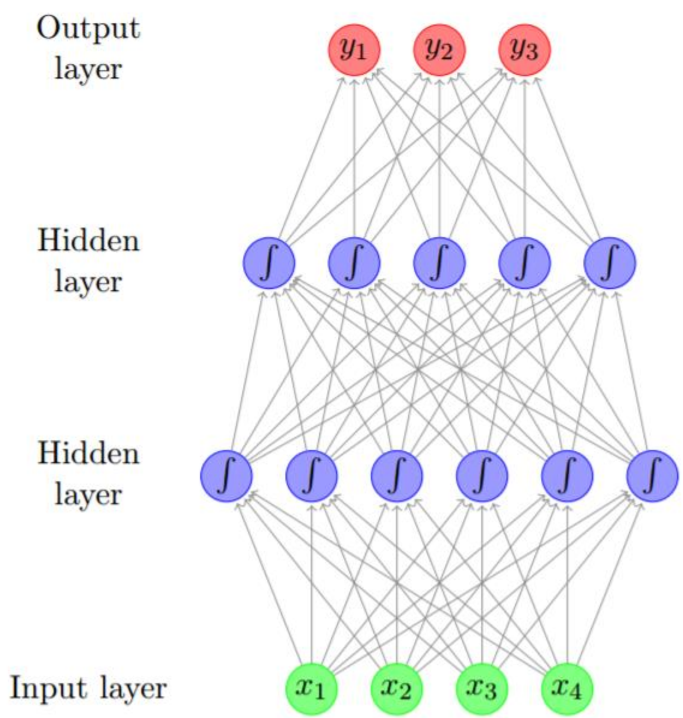


Figure 2: Feed-forward neural network with two hidden layers.

Figure from Goldberg (2015)

Inference in a FF Neural Network

Perform computations forwads through the graph:

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

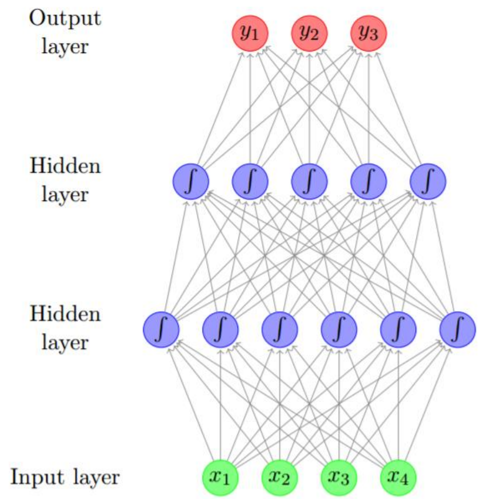


Figure 2: Feed-forward neural network with two hidden layers.

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 $\{\mathbf{x}^{\mathbf{k}}, \mathbf{y}^{\mathbf{k}}\}$
- Loss function L

Repeat for a while:

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

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

Forward pass

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

In neural networks, by backpropagation

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

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

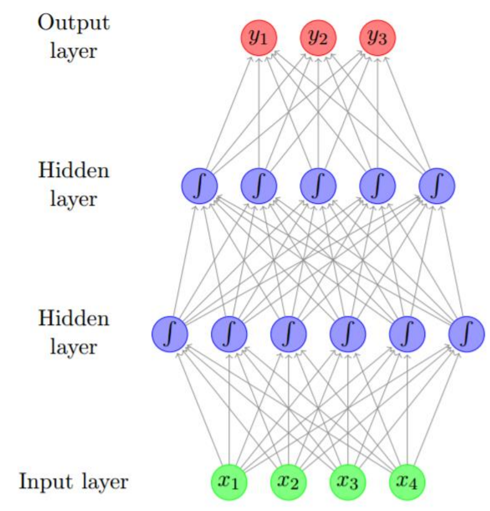


Figure 2: Feed-forward neural network with two hidden layers.

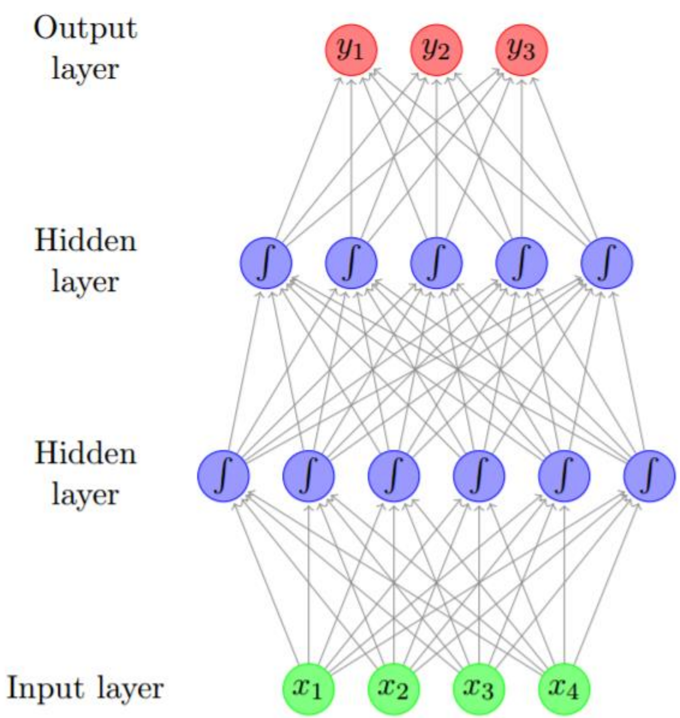
Save the values for $\mathbf{h^1}$, $\mathbf{h^2}$, $\mathbf{y}$ too!

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.

TDNN for POS Tagging



Probabilities of output POS at time t

Words at times t-2 to t+2

Limitations?

- Need to decide on input feature representations – fixed horizon
- Interactions between steps in sequence learned indirectly

Recurrent Neural Networks

A neural network **sequence model**:

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_{1:n}}, \mathbf{y_{1:n}}$$

$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$ # $\mathbf{s_i}$: state vector

$$\mathbf{y_i} = O(\mathbf{s_i})$$ # $\mathbf{y_i}$: output vector

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

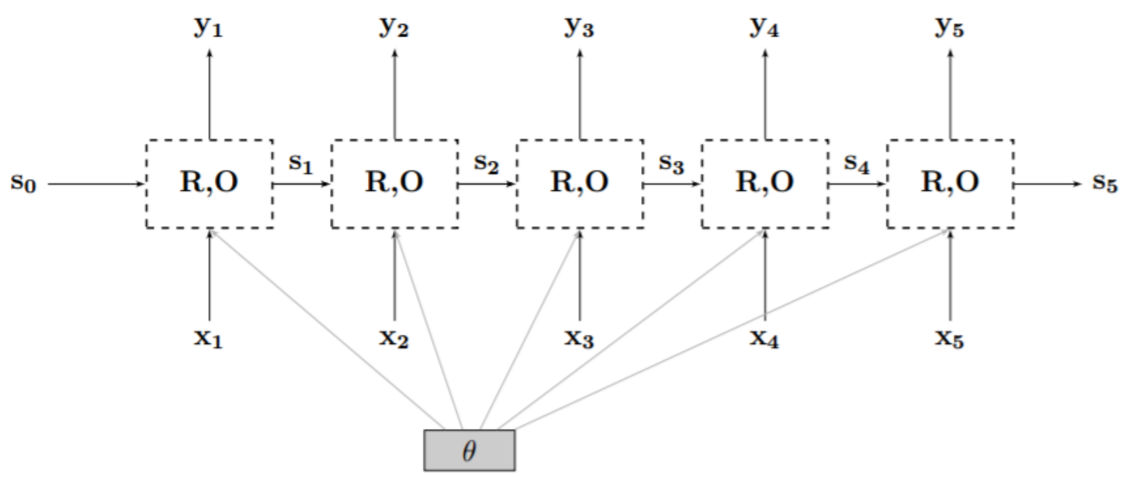


Figure 6: Graphical representation of an RNN (unrolled).

Long-Range Dependencies in Language

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

- e.g., *I will <u>look</u> the word that you have described that doesn't make sense to her <u>up</u>.*
- Can you think of some other examples in English of long-range dependencies? (Not just for POS tagging, but in general)

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

Comparison between LC-CRFs, RNNs

1. At each step of the sequence, LC-CRFs define a linear model (linear between feature values and weights) similar to logistic regression

   > RNNs replace this with a neural network (more expressive model, may need more data to train)

2. LC-CRFs depend on a good selection of input features; requires more feature engineering

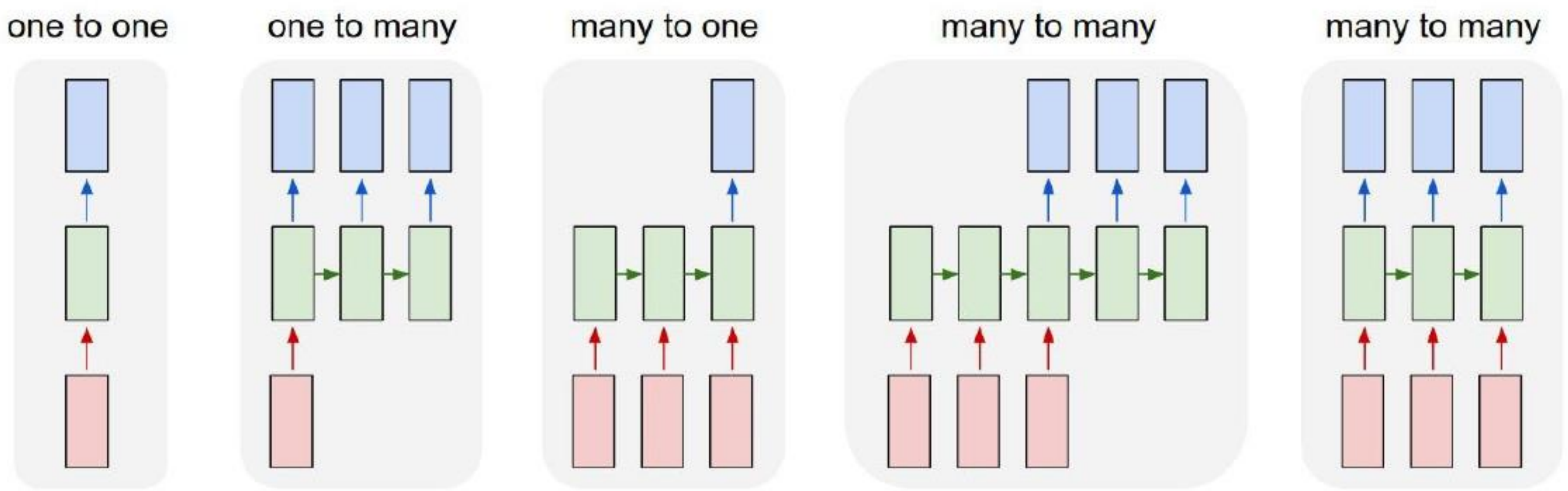
   > RNNs can also make use of features, or can try to learn useful ones

3. LC-CRFs make strong local independence assumptions – exact polynomial-time inference

   > RNNS may require approximate inference algorithm, depending on the specific task and application

Different RNN Architectures

Different architectures for different use cases



Which architecture would you use for:

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

Long Short-Term Memory Networks

A popular RNN architecture for NLP (Hochreiter and Schmidhuber, 1997)

- Explicitly models a "memory" cell (i.e., a hidden-layer vector), and how it is updated as a response to the current input and the previous state of the memory cell.

Memory helps us keep track of long-range dependencies

STORE

- *I will <u>look</u> the word that you have described that doesn't make sense to her <u>up</u>.*

UPDATE

Main Operations in LSTM

C_t: memory cell at time t

h_t: hidden state at time t

x_t: input at time t

y_t: output at time t

forget decide what to keep or forget from C_{t-1}

input decide what info from x_t to add into C_t

output compute h_t to predict y_t, and for time t+1

Implement by using component-wise addition, multiplication, and non-linearities.

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:

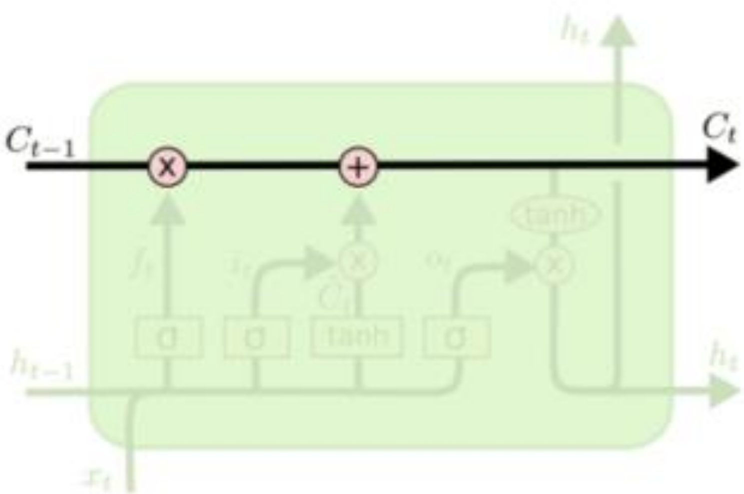
- Suppose it is $\mathbf{W}^1$ that we want to modify, and there are N layers between that and the loss function. Let f^lrepresent the computations at layer l.

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

- 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

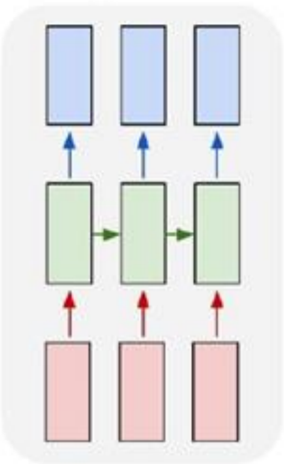
It is the fact that in the LSTM, we can propagate a cell state directly that fixed the vanishing gradient problem:



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

Bidirectional LSTMs – Motivation

Standard LSTMs go forward in time

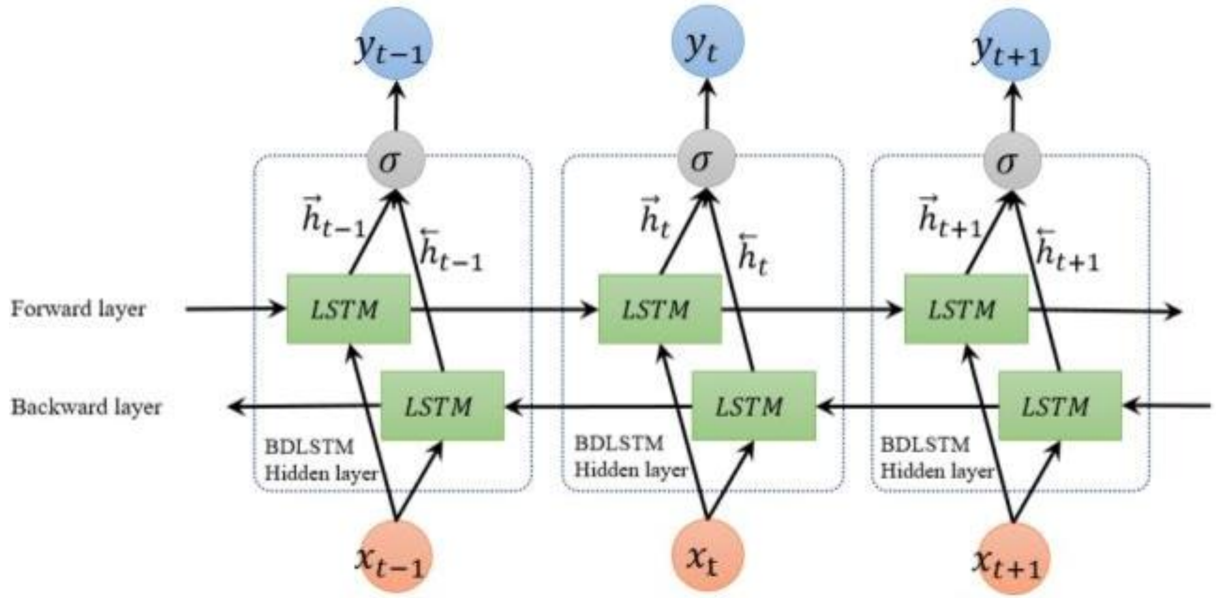


For some applications, this is necessary

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)!

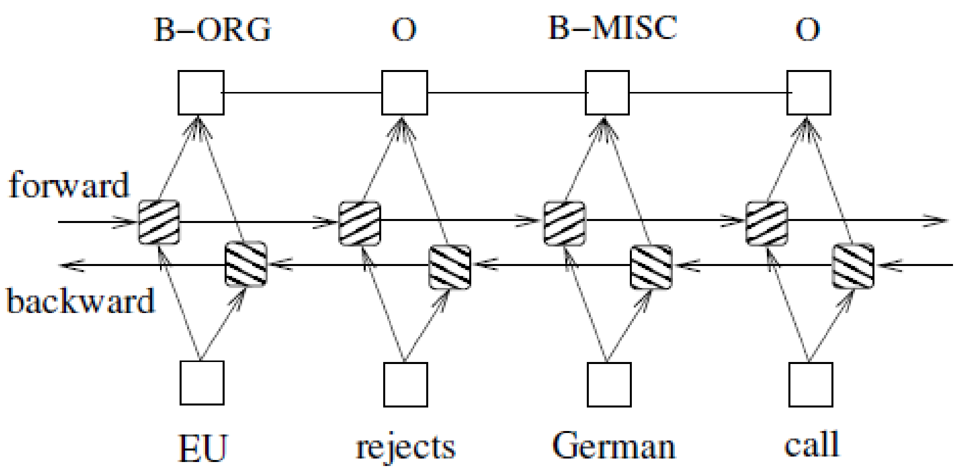


Figure 7: A BI-LSTM-CRF model.

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* x *k*)

Scores from LSTM
size (*n* x *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

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

            backward pass for forward state LSTM
            backward pass for backward state LSTM
        4) update parameters
    end for
end for
```

Summary

LSTMs are the backbone of many modern NLP models!

LSTM-CRFs are the state of the art on many basic sequence labelling tasks:

- Named entity recognition
- POS tagging
- Chunking

Next, we will look at developments in modelling sequences in the last two years.