Recurrent Neural Networks

COMP-550
Instructor: Jackie CK Cheung
COMP-550
Fall 2018
Primer by Yoav Goldberg:
https://arxiv.org/abs/1510.00726

Outline

Introduction to neural networks and deep learning

Feedforward neural networks

Recurrent neural networks

Classification Review

$$y = f(\vec{x})$$

output label

classifier

input

Represent input $\vec{x}$ as a list of features

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

$\rightarrow$

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8 ...
1.0,	0.0,	1.0,	1.0,	0.0,	0.0,	0.0,	1.0 ...

Logistic Regression

Linear regression:

$$y = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + b$$

Intuition: Linear regression gives as continuous values in [-∞, ∞] —let's squish the values to be in [0, 1]!

Function that does this: logit function

$$P(y|\vec{x}) = \frac{1}{Z} e^{a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + b}$$

This Z is a normalizing constant to ensure this is a probability distribution.

(a.k.a., maximum entropy or MaxEnt classifier)

N.B.: Don't be confused by name—this method is most often used to solve classification problems.

Linear Model

Logistic regression, support vector machines, etc. are examples of **linear models**.

$$P(y|\vec{x}) = \frac{1}{Z} e^{\underbrace{a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + b}_{\text{Linear combination of feature weights and values}}}$$

Cannot learn complex, non-linear functions from input features to output labels (without adding features)

e.g., Starts with a capital AND not at beginning of sentence -> proper noun

(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.)

Responsible For:

AlphaGo (Google) (2015)

- Beat Go champion Lee Sedol in a series of 5 matches, 4-1

Atari game-playing bot (Google) (2015)

Above results use NNs in conjunction with **reinforcement learning**

State of the art in:

- Speech recognition
- Machine translation
- Object detection
- Other NLP tasks

Feedforward Neural Networks

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

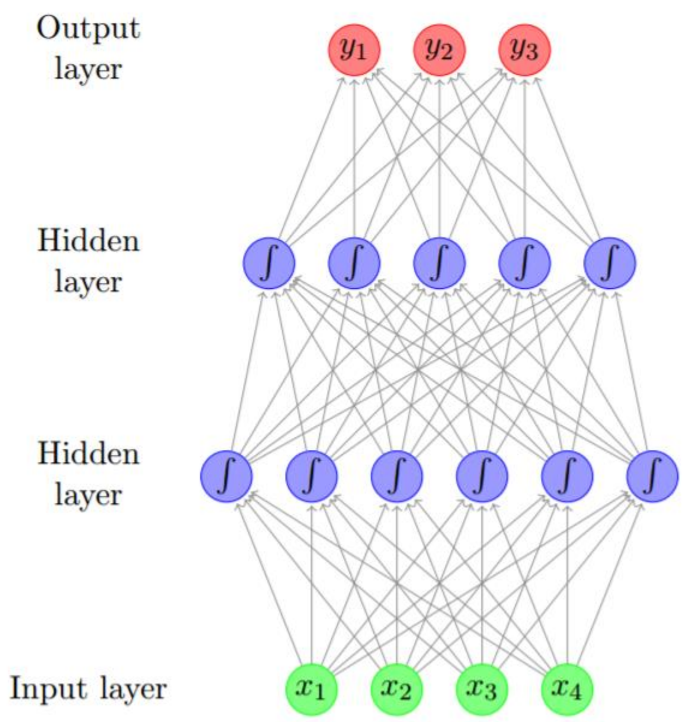


Figure 2: Feed-forward neural network with two hidden layers.

Figure from Goldberg (2015)

Inference in a FF Neural Network

Perform computations forwads through the graph:

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

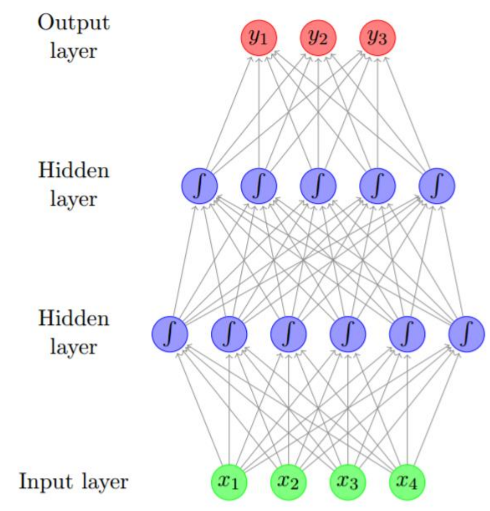


Figure 2: Feed-forward neural network with two hidden layers.

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

Activation Function

In one unit:

Linear combination of inputs and weight values → non-linearity

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

Popular choices:

Sigmoid function (just like logistic regression!)

tanh function

Rectifier/ramp function: $g(x) = \max(0, x)$

Why do we need the non-linearity?

Softmax Layer

In NLP, we often care about discrete outcomes

- e.g., words, POS tags, topic label

Output layer can be constructed such that the output values sum to one:

$$\text{Let } \mathbf{x} = x_1 \ldots x_k$$

$$softmax(x_i) = \frac{\exp(x_i)}{\sum_j^k \exp(x_j)}$$

Interpretation: unit x_i represents probability that outcome is i.

Essentially, the last layer is like a multinomial logistic regression

Loss Function

A neural network is optimized with respect to a **loss function**, which measures how much error it is making on predictions:

$\mathbf{y}$: correct, gold-standard distribution over class labels

$\hat{\mathbf{y}}$: system predicted distribution over class labels

$L(\mathbf{y}, \hat{\mathbf{y}})$: loss function between the two

Popular choice for classification (usually with a softmax output layer) – **cross entropy**:

$$L_{ce}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log(\widehat{y_i})$$

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

Gradient Descent Summary

Descent vs ascent

Convention: think about the problem as a minimization problem

Minimize the *loss function*

- $\theta \leftarrow \theta - \gamma(\nabla L(\theta))$

Initialize $\theta = \{\theta_1, \theta_2, \dots, \theta_k\}$ randomly

Do for a while:

Compute $\nabla L(\theta)$, [e.g., forward algorithm with LC-CRFs]
$\theta \leftarrow \theta - \gamma \nabla L(\theta)$

Stochastic Gradient Descent (SGD)

In the standard version of the algorithm, the gradient is computed over the entire training corpus.

- Weight update only once per iteration through training corpus.

Alternative: calculate gradient over a small mini-batch of the training corpus and update weights

SGD is when mini-batch size is one.

- Many weight updates per iteration through training corpus
- Usually results in much faster convergence to final solution, without loss in performance

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

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

In neural networks, by backpropagation

Return θ

Example: Forward Pass

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

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

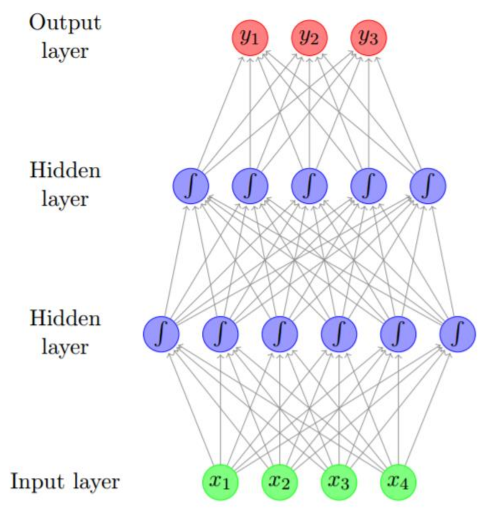


Figure 2: Feed-forward neural network with two hidden layers.

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

Example Cont'd: Backpropagation

$$f(\mathbf{x}) = g^3(g^2(g^1(\mathbf{x})))$$

Need to compute: $\frac{\partial L}{\partial \mathbf{W}^3}, \frac{\partial L}{\partial \mathbf{W}^2}, \frac{\partial L}{\partial \mathbf{W}^1}$

By calculus and chain rule:

- $\frac{\partial L}{\partial \mathbf{W}^3} = \frac{\partial L}{\partial g^3} \frac{\partial g^3}{\partial \mathbf{W}^3}$
- $\frac{\partial L}{\partial \mathbf{W}^2} = \frac{\partial L}{\partial g^3} \frac{\partial g^3}{\partial g^2} \frac{\partial g^2}{\partial \mathbf{W}^2}$
- $\frac{\partial L}{\partial \mathbf{W}^1} = \frac{\partial L}{\partial g^3} \frac{\partial g^3}{\partial g^2} \frac{\partial g^2}{\partial g^1} \frac{\partial g^1}{\partial \mathbf{W}^1}$

Notice the overlapping computations? Be sure to do this in a smart order to avoid redundant computations!

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**:

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

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

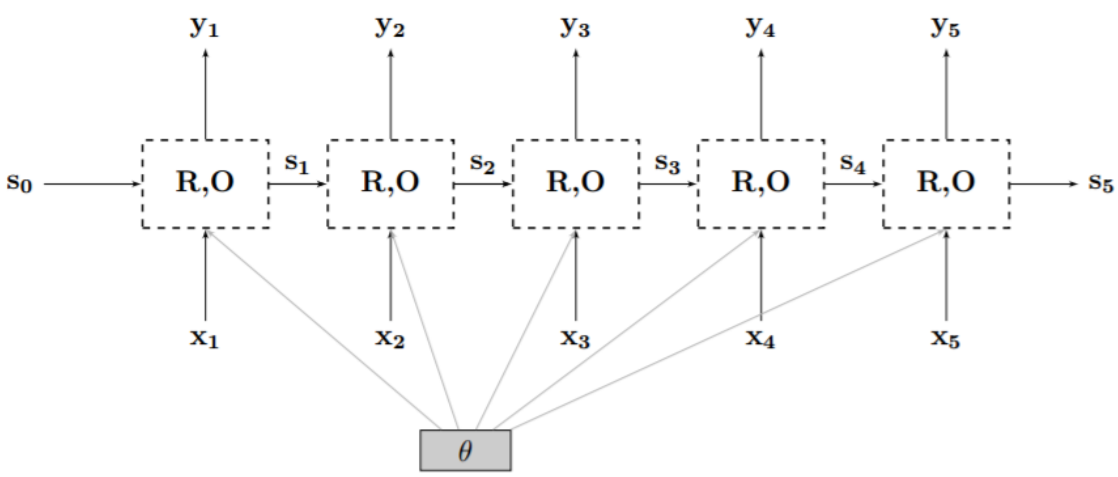


Figure 6: Graphical representation of an RNN (unrolled).

Long-Term Dependencies in Language

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

- *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?

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

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

Long Short-Term Memory Networks

Currently one of the most popular RNN architectures for NLP (Hochreiter and Schmidhuber, 1997)

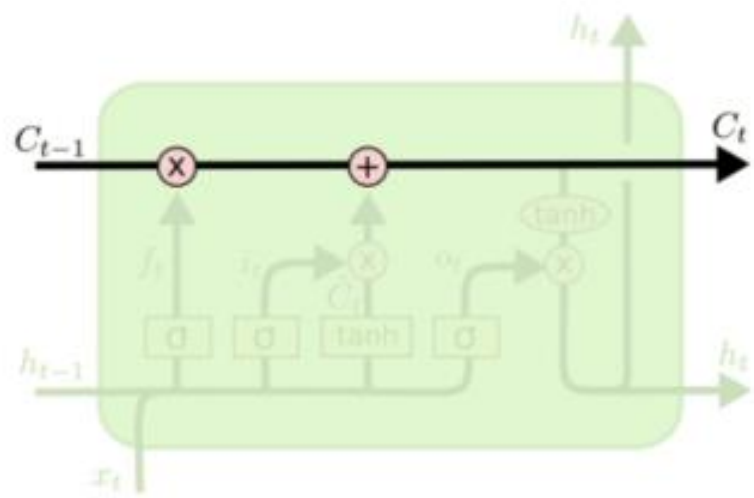
- Explicitly models a "memory" cell (i.e., a hidden-layer vector), and how it is updated as a response to the current input and the previous state of the memory cell.

Visual step-by-step explanation:

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

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!

Hardware for NNs

Common operations in inference and learning:

- Matrix multiplication
- Component-wise operations (e.g., activation functions)

This operation is **highly parallelizable**!

Graphical processing units (**GPUs**) are specifically designed to perform this type of computation efficiently

Packages for Implementing NNs

TensorFlow	https://www.tensorflow.org/
PyTorch	http://pytorch.org/
Caffe	http://caffe.berkeleyvision.org/
Theano	http://deeplearning.net/software/theano/

- These packages support GPU and CPU computations
- Write interface code in high-level programming language, like Python

Summary: Advantages of NNs

Learn relationships between inputs and outputs:

- Complex features and dependencies between inputs and states over long ranges with no fixed horizon assumption (i.e., **non-Markovian**)
- Reduces need for feature engineering
- More efficient use of input data via weight sharing

Highly flexible, generic architecture

- **Multi-task learning**: jointly train model that solves multiple tasks simultaneously
- **Transfer learning**: Take part of a neural network used for an initial task, use that as an initialization for a second, related task

Summary: Challenges of NNs

Complex models may need a lot of training data

Many fiddly hyperparameters to tune, little guidance on how to do so, except empirically or through experience:

- Learning rate, number of hidden units, number of hidden layers, how to connect units, non-linearity, loss function, how to sample data, training procedure, etc.

Can be difficult to interpret the output of a system

- *Why* did the model predict a certain label? Have to examine weights in the network.
- Important to convince people to act on the outputs of the model!

NNs for NLP

Neural networks have “taken over” mainstream NLP since 2014; most empirical work at recent conferences use them in some way

Lots of interesting open research questions:

- How to use linguistic structure (e.g., word senses, parses, other resources) with NNs, either as input or output?
- When is linguistic feature engineering a good idea, rather than just throwing more data with a simple representation for the NN to learn the features?
- Multitask and transfer learning for NLP
- Defining and solving new, challenging NLP tasks