

# Neural Networks for NLP

COMP-599

Nov 30, 2016

# Outline

---

Neural networks and deep learning: introduction

Feedforward neural networks

word2vec

Complex neural network architectures

Convolutional neural networks

Recurrent and recursive neural networks

# Reference

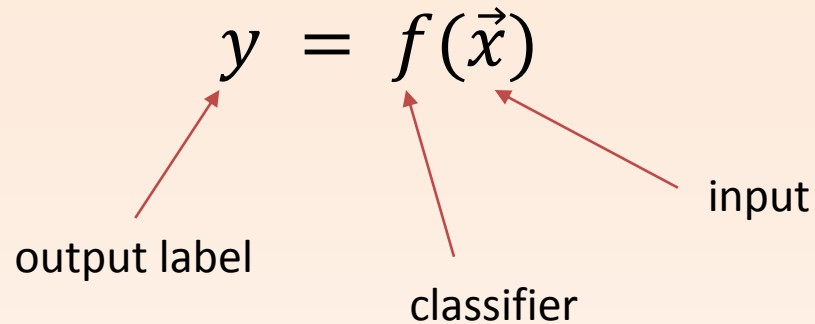
---

Much of the figures and explanations are drawn from this reference:

A Primer on Neural Network Models for Natural Language Processing. Yoav Goldberg. 2015.

<http://u.cs.biu.ac.il/~yogo/nnlp.pdf>

# Classification Review



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.

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

# Logistic Regression

---


Linear regression:

$$y = a_1x_1 + a_2x_2 + \dots + a_nx_n + b$$

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

Function that does this: logit function

$$P(y|\vec{x}) = \frac{1}{Z} e^{a_1x_1 + a_2x_2 + \dots + a_nx_n + b}$$

 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**.

$$P(y|\vec{x}) = \frac{1}{Z} e^{\underbrace{a_1x_1 + a_2x_2 + \dots + a_nx_n + b}_{\text{Linear combination of feature weights and values}}}$$

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

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

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

# Responsible For:

---

AlphaGo (Google) (2015)

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

Atari game-playing bot (Google) (2015)

State of the art in:

- Speech recognition
- Machine translation
- Object detection
- Many other NLP tasks



# Feedforward Neural Networks

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

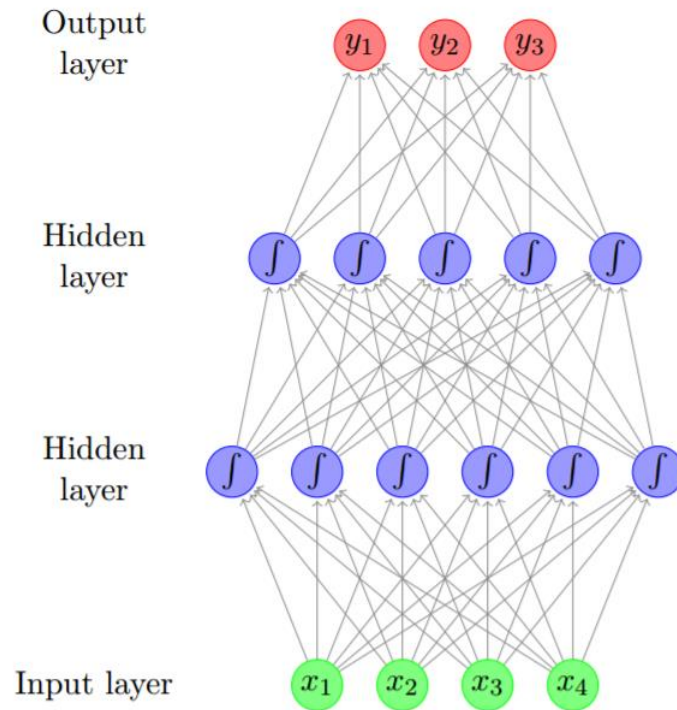


Figure 2: Feed-forward neural network with two hidden layers.

Figure from Goldberg (2015)

# Inference in a FF Neural Network

Perform computations forwards through the graph:

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

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

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3$$

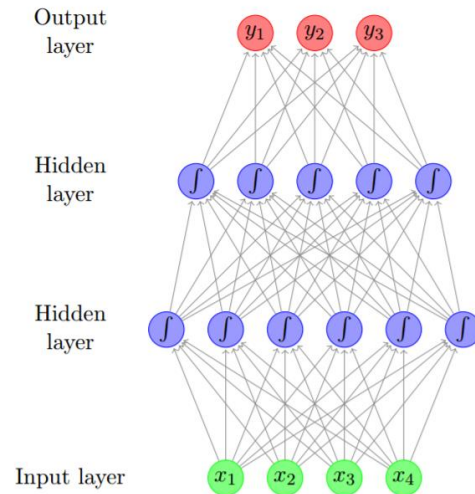


Figure 2: Feed-forward neural network with two hidden layers.


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

# Activation Function

---

In one unit:

Linear combination of inputs and weight values → non-linearity

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


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:

$$\text{Let } \mathbf{x} = x_1 \dots x_k$$
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j^k \exp(x_j)}$$

**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**:

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

# 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

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

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

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

By backpropagation

Return  $\theta$

# 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.



# Unsupervised Learning

---

Neural networks can be used for unsupervised learning

Usual approach: the outputs to be predicted are derived from the input

Popular task: **representation learning**

- Learn an **embedding** or **representation** of some vocabulary of objects using a neural network
- Usually, these embeddings are vectors which are taken from a hidden layer of a neural network
- These embeddings can then be used to compute similarity or relatedness, or in downstream tasks (remember **distributional semantics**)

# word2vec (Mikolov et al., 2013)

---

Learn vector representations of words

Actually two models:

- Continuous bag of words (CBOW) – use context words to predict a target word
- Skip-gram – use target word to predict context words

In both cases, the representation that is associated with the target word is the embedding that is learned.

# word2vec Architectures

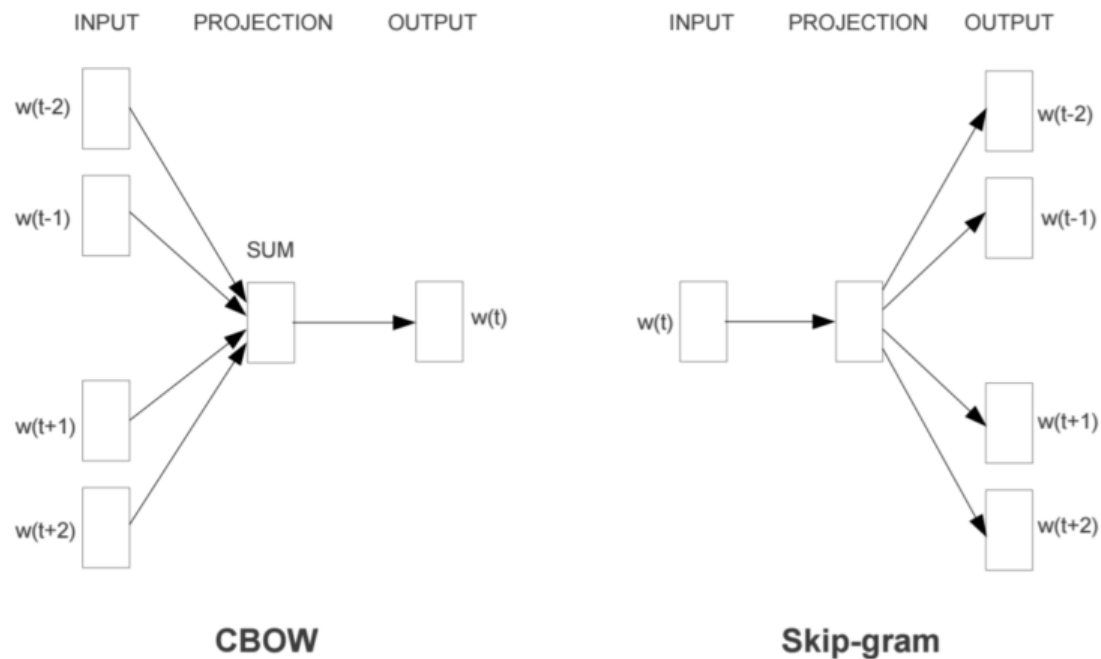


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

(Mikolov et al., 2013)

# Validation

---

## Word similarity

- **Analogical reasoning task**

$a : b :: c : d$  – what is  $d$ ?

New York : New York Times :: Baltimore : ?

(Answer: Baltimore Sun)

man : king :: woman : ?

(Answer: queen)

Solve by ranking vocabulary items according to assumption:

$$v_a - v_b = v_c - v_d$$

$$\text{or } v_d = v_b - v_a + v_c$$

$$(? = \text{king} - \text{man} + \text{queen})$$

# Impact of Hyperparameters

---

There are many hyperparameters that are important to the performance of word2vec:

- Weighting relative importance of context words
- Sampling procedure during training of target and context words

These have a large impact on performance (Levy et al., 2015)

- Applying analogous changes to previous approaches such as Singular Value Decomposition results in similar performance on word similarity tasks

# Use of Word Embeddings in NLP

---

Can download pretrained word2vec embeddings from the web

Use word vectors as features in other NLP tasks

- Now very widespread

Advantages:

- Vectors are trained from a large amount of general-domain text: proxy of background knowledge!
- Cheap and easy to try

Disadvantages:

- Doesn't always work (especially if you have a lot of task-specific data)

# Other Neural Network Architectures

---

Convolutional neural networks

Recurrent neural networks

Recursive neural networks

# Convolutional Neural Networks (CNN)

Apply a neural network to a window around elements in a sequence, then **pool** the results into a single vector

- **Convolution layer:** much like the TDNN
- **Pooling layer:** combine output vectors of convolution layer

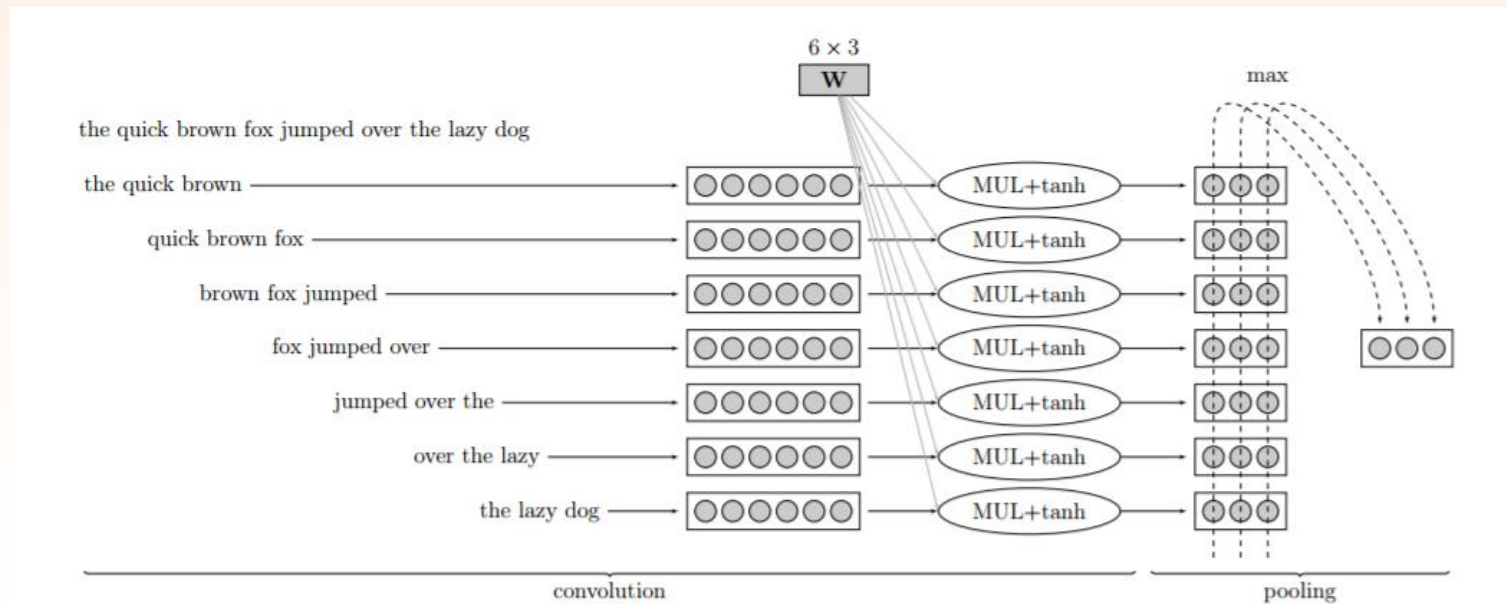


Figure from Goldberg, (2015)



# Uses of CNNs

---

Widely used in image processing for object detection

**Intuition:** the convolution+pooling is a kind of search through the data to find features that make up some object

In NLP, effective at similar tasks, which require searching for a part of an indicative part of a sentence.

e.g., Kalchbrenner et al. (2014) apply CNNs to:

- Sentiment analysis
- Question type classification

# Recurrent Neural Networks

A neural network sequence model:

$$RNN(\mathbf{s}_0, \mathbf{x}_{1:n}) = \mathbf{s}_{1:n}, \mathbf{y}_{1:n}$$

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

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

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

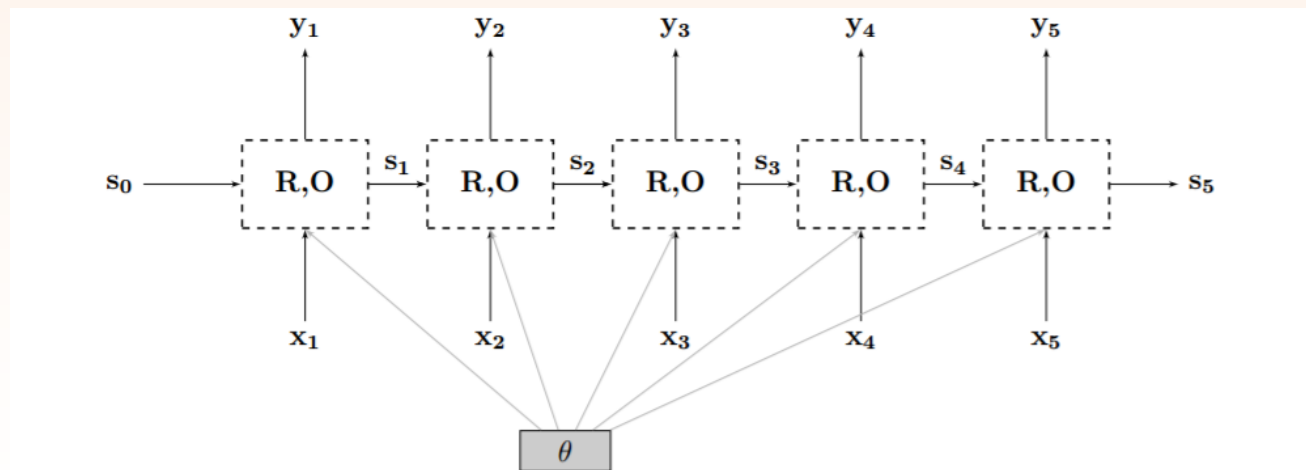


Figure 6: Graphical representation of an RNN (unrolled).

# Long 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/>

# Recursive Neural Networks

Similar to recurrent neural networks, but with a (static) tree structure

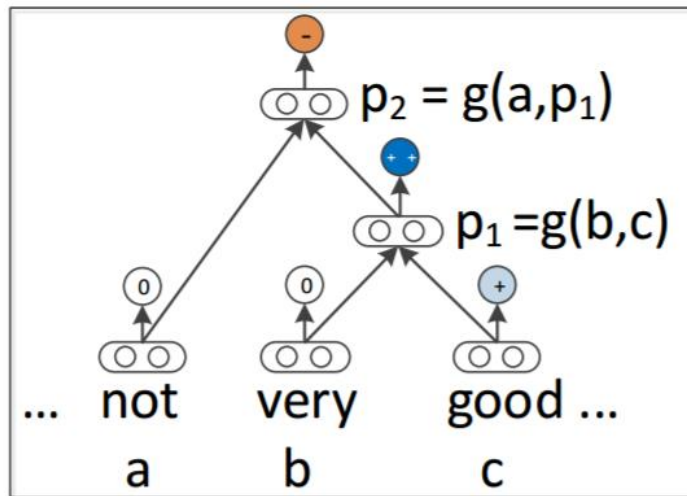


Figure 4: Approach of Recursive Neural Network models for sentiment: Compute parent vectors in a bottom up fashion using a compositionality function  $g$  and use node vectors as features for a classifier at that node. This function varies for the different models.

What makes it recursive is that the composition function at each step is the same (the weights are tied)

(Socher et al., 2013)

# Advantages of Neural Networks

---

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

# Challenges of Neural Networks in NLP

---

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

# References

---

- Mikolov, Sutskever, Chen, Corrado, and Dean. Distributed Representations of Words and Phrases and Their Compositionality. *NIPS 2013*.
- Hochreiter and Schmidhuber. Long Short-Term Memory. *Neural Computation*. 1997.
- Kalchbrenner, Grefenstette, and Blunsom. A Convolutional Neural Network for Modelling Sentences. <https://arxiv.org/pdf/1404.2188v1.pdf>
- Levy, Goldberg, and Dagan. Improving distributional similarity with lessons learned from word embeddings. *TACL 2015*.
- Socher, Perelygin, Wu, Chuang, Manning, Ng, and Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. *EMNLP 2013*.