Neural Networks

Awni Hannun

CS221: Artificial Intelligence (Autumn 2013)
Outline

1. Overview: Neural Networks
2. Feed forward calculation
3. Training : Backpropagation
4. Applications and Extensions
Outline

1. Overview: Neural Networks
2. Feed forward calculation
3. Training: Backpropagation
4. Applications and Extensions
What is a Neural Network?

Logistic regression as a “neuron”

\[
\sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b) = \sigma(W x + b)
\]

\[W \in \mathbb{R}^{1 \times 3}\]
What is a Neural Network?

Stack many logistic units to create a Neural Network

Layer 1 / Input

Layer 2 / hidden layer

Layer 3 / output

CS221: Artificial Intelligence (Autumn 2013)
Why Neural Networks?

Too many reasons, here are a few –

1. Highly expressive (universal approximators)

2. Deep learning, hierarchical representations

3. Supervised learning
   - Binary classification
   - Multiclass Classification
   - Regression

4. Unsupervised Learning
   - Feature learning
   - Dimensionality reduction
   - Generative models
Notation

\( l = 1, ..., L \) - \( l \)-th layer

\( W^{(l)} \in \mathbb{R}^{m \times n} \) - weights for layer \( l \)

\( b^{(l)} \in \mathbb{R}^m \) - bias for layer \( l \)

\( \sigma(z) \) - activation function (for the following \( \sigma \) is the sigmoid function)

\( z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)} \) - input to \((l + 1)\)-st layer

\( a^{(l+1)} = \sigma(z^{(l+1)}) \) - activation of \((l + 1)\)-st layer, let \( a^{(1)} = x \)
Outline

1. Overview: Neural Networks
2. Feed forward calculation
3. Training: Backpropagation
4. Applications and Extensions
Forward Propagation

\[ a^{(2)} = \sigma(W^{(1)}x + b^{(1)}) \]
Forward Propagation

\[ a^{(3)} = \sigma(W^{(2)}a^{(2)} + b^{(2)}) \]

Layer 1 / Input
Layer 2 / hidden layer
Layer 3 / output

CS221: Artificial Intelligence (Autumn 2013)
Forward Propagation

\[ \text{Loss}(x, y; W, z) = \frac{1}{2} \| a^{(3)} - y \|_2^2 \]

Layer 1 / Input
Layer 2 / hidden layer
Layer 3 / output

CS221: Artificial Intelligence (Autumn 2013)
Forward Propagation

\[ z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)} \]

\[ a^{(l+1)} = \sigma(z^{(l+1)}) \]
Forward Propagation

Summary: Feed forward pass is just function composition + cost calculation

\[ f(x) = \sigma \left( W^{(2)} \sigma \left( W^{(1)}x + b^{(1)} \right) + b^{(2)} \right) \]

\[ \text{Loss}(x, y; W, z) = \frac{1}{2} \| f(x) - y \|_2^2 \]
Outline

1. Overview: Neural Networks
2. Feed forward calculation
3. Training: Backpropagation
4. Applications and Extensions
Training

Use gradient based updates to learn parameters for Network

\[ \text{Loss}(x, y; W, z) = \frac{1}{2} \| a^{(L)} - y \|_2^2 \]

\[ W \leftarrow W - \eta \nabla_W \text{Loss}(x, y; W, b) \]

\[ b \leftarrow b - \eta \nabla_b \text{Loss}(x, y; W, b) \]
Training: Backpropagation

Backpropagation
Algorithm to compute the derivative of the Loss function with respect to the parameters of the Network

\[ \nabla_W \text{Loss}(x, y; W, b) \]

\[ \nabla_b \text{Loss}(x, y; W, b) \]
Chain Rule

\[(f \circ g)(x) = f(g(x))\]

\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}
\]
Chain Rule

\[ f(g_1(x) + g_2(x)) \]

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial g_1} \frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2} \frac{\partial g_2}{\partial x} \]
Chain Rule

\[ f(\sum_{i}^{n} g_{i}(x)) \]

\[ \frac{\partial f}{\partial x} = \sum_{i}^{n} \frac{\partial f}{\partial g_{i}} \frac{\partial g_{i}}{\partial x} \]
Backpropagation

Idea: apply chain rule recursively

\[ f(x) = f_3(w_3 f_2(w_2 f_1(w_1 x))) \]

\[ \frac{df}{dx} = f'_3(w_3 f_2(w_2 f_1(w_1 x))) \frac{df}{dx}(w_3 f_2(w_2 f_1(w_1 x))) \]

\[ \frac{df}{dx} = w_3 f'_3(w_3 f_2(w_2 f_1(w_1 x))) f'_2(w_2 f_1(w_1 x)) \frac{df}{dx}(w_2 f_1(w_1 x)) \]
Backpropagation

\[ \delta^{(3)} \]
Backpropagation

Derivative of sigmoid

\[
\sigma(z) = \frac{1}{1 + e^{-z}}
\]

\[
\sigma'(z) = \frac{e^{-z}}{(1 + e^{-z})^2}
\]

\[
= \sigma(z)(1 - \sigma(z))
\]
Backpropagation

\[
\frac{\partial \text{Loss}}{\partial z^{(3)}} = \frac{\partial}{\partial z^{(3)}} \left( \frac{1}{2} \|a^{(3)} - y\|^2 \right) \\
= \frac{\partial}{\partial a^{(3)}} \left( \frac{1}{2} \|a^{(3)} - y\|^2 \right) \frac{\partial a^{(3)}}{\partial z^{(3)}} \\
= \left( a^{(3)} - y \right) \left( a^{(3)} (1 - a^{(3)}) \right) \\
= \delta^{(3)}
\]
Backpropagation

\[
\frac{\partial \text{Loss}}{\partial z^{(l)}} = \delta^{(l)}
\]

CS221: Artificial Intelligence (Autumn 2013)
Backpropagation

Recursively compute delta at each hidden layer

\[ \delta_i^{(l)} = \frac{\partial \text{Loss}}{\partial z_i^{(l)}} = \frac{\partial \text{Loss}}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \]

\[ \frac{\partial \text{Loss}}{\partial a_i^{(l)}} = \sum_{j=1}^{m} \frac{\partial \text{Loss}}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial a_i^{(l)}} \]

\[ = \sum_{j=1}^{m} \delta_j^{(l+1)} w_{ji}^{(l)} \]

\[ = (w_i^{(l)})^T \delta_j^{(l+1)} \]

\[ \delta_i^{(l)} = (w_i^{(l)})^T \delta_j^{(l+1)} (a_i^{(l)}) (1 - a_i^{(l)}) \]

\[ \delta^{(l)} = (w^{(l)})^T \delta^{(l+1)} \circ (a^{(l)}) (1 - a^{(l)}) \]
Backpropagation

Compute gradient of Loss w.r.t. weights

$$\frac{\partial \text{Loss}}{\partial w_{ij}^{(l)}} = \frac{\partial \text{Loss}}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}}$$

$$= \frac{\partial \text{Loss}}{\partial z_i^{(l+1)}} \frac{\partial (W^{(l)}a^{(l)} + b^{(l)})_i}{\partial w_{ij}^{(l)}}$$

$$= \delta_i^{(l+1)} a_j^{(l)}$$

$$\nabla_{W^{(l)}} \text{Loss} = \delta^{(l+1)} (a^{(l)})^T$$

$$\nabla_{b^{(l)}} \text{Loss} = \delta^{(l)}$$
Backpropagation

Backpropagation Algorithm

1. Feed forward input \((x, y)\), computing activation for layers \(l = 2, \ldots, L\)

2. For the output layer, \(L\) set:

\[
\delta^{(L)} = a^{(L)}(1 - a^{(L)})(a^{(L)} - y)
\]

3. For layers \(l = 2, \ldots, L - 1\) set:

\[
\delta^{(l)} = (W^{(l)})^T \delta^{(l+1)} \circ a^{(l)}(1 - a^{(l)})
\]

4. Compute gradient with respect to parameters \(W^{(l)}, b^{(l)}\) as:

\[
\nabla_{W^{(l)}} \text{Loss}(x, y; W, b) = \delta^{(l+1)}(a^{(l)})^T
\]

\[
\nabla_{b^{(l)}} \text{Loss}(x, y; W, b) = \delta^{(l+1)}
\]
Training: Backpropagation

**SGD Algorithm**
run SGD as usual using backpropagation to compute derivative of Loss w.r.t. params

For each input \((x, y)\) in the training set

\[
\nabla_{w,b} \text{Loss}(x, y; W, b) = \text{Backpropagate}(x, y)
\]

\[
W \leftarrow W - \eta \nabla_W \text{Loss}(x, y; W, b)
\]

\[
b \leftarrow b - \eta \nabla_b \text{Loss}(x, y; W, b)
\]
Outline

1. Overview: Neural Networks
2. Feed forward calculation
3. Training: Backpropagation
4. Applications and Extensions
Model: Deep NN

$x \xrightarrow{W^{(1)},b^{(1)}} \ldots \xrightarrow{W^{(l)},b^{(l)}} \ h_{W,b}(x)$

hidden layers
Applications: Deep NN

Speech Recognition

Acoustic Model

Language Model

Decoder (HMM)

W = "The fat cat"
Applications: Deep NN

Speech Recognition

Convert output of NN to observation probabilities using Bayes Thm

\[ p(o|s) = \frac{p(s|o)p(o)}{p(s)} \]
Model: Convolutional NN

Ideas:

- small receptive fields
- tie weights, re-use features
- pooling gives translation invariance

- convolution demo
- pooling demo
Model: Convolutional NN

CS221: Artificial Intelligence (Autumn 2013)
Applications: Convolutional NN

Computer Vision

ImageNet object recognition of 22k classes (state-of-the-art) -
~37% error rate

- beer bottle
- water bottle
- soda bottle
- Egyptian cat
- Tabby cat
Applications: Convolutional NN

Computer Vision – feature learning

Applications: Google Brain

Major differences from Conv Net:

- Unsupervised training on 10 Million images from YouTube
- Untied weights, i.e. locally connected (1+ Billion parameters)

Model Type: Auto-encoder
Applications: Auto-encoder

Dimensionality reduction

PCA: First two principle components

AE: 784-1000-500-250-2

Applications: Auto-encoder

Learn features with sparse auto-encoder
Model: Restricted Boltzmann Machine (RBM)

Bipartite Markov Random Field

Edges are undirected.

We’ll learn about MRFs later!
Model: Deep Boltzmann Machine (DBN)

Stack of “directed” RBMs with RBM on top.
Applications: RBM / DBN

Most common: Pre-training for a DNN

Learning Features

Generating Samples using MCMC
(We’ll learn about that later!)

Images: deeplearning.net
Model: Recurrent NN

\[ h_{W,b}(x) \]

\[ W^{(6)}, b^{(6)} \]

\[ W^{(5)}, b^{(5)} \]

\[ W^{(4)}, b^{(4)} \]

\[ W^{(3)}, b^{(3)} \]

\[ \ldots \]

\[ W^{(1)}, b^{(1)} \]

\[ x_0 \]

\[ x_1 \]

\[ \ldots \]

\[ x_{n-1} \]

\[ x_n \]
Applications: Recurrent NN

Any Sequence-based problem:

1. Speech recognition
2. Handwriting recognition (state-of-the-art)
3. Stock-market prediction
4. Vision

A. Graves. Generating Sequences With Recurrent Neural Networks
http://www.cs.toronto.edu/~graves/handwriting.html
Where to from here?

Online Tutorials:

Stanford Deep Learning Tutorial –

http://deeplearning.net/

Reading:

Chris Bishop - *Neural Networks for Pattern Recognition*

Raul Rojas – *Neural Networks* (online)

Much much more online...
Appendix
Backpropagation

Presentation following *Neural Networks* - Rojas
Backpropagation

Composition

$x$  forward  $f(g(x))$

$\frac{d}{dx} f(g(x)) g'(x)$  backward  1

CS221: Artificial Intelligence (Autumn 2013)
Backpropagation

Addition

\[ x \overset{\text{forward}}{\longrightarrow} f(x) + g(x) \]

\[ f'(x) + g'(x) \overset{\text{backward}}{\longleftarrow} 1 \]

CS221: Artificial Intelligence (Autumn 2013)
Backpropagation

Scaling

\[ w \]

forward

\[ x \rightarrow wx \]

backward

\[ w \leftarrow 1 \]
Backpropagation
Backpropagation

\[
\delta^{(L)} = a^{(L)}(1 - a^{(L)})(a^{(L)} - y) \quad \text{backward}
\]

\[
a^{(L)} = \sigma(z^{(L)})
\]

\[
a^{(L)} - y \quad \frac{1}{2}(a^{(L)} - y)^2
\]
Backpropagation

\[
\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_{j=1}^{n} w_{ji}^{(l)} \delta_j^{(l+1)} \\
= a_i^{(l)}(1 - a_i^{(l)}) (W_{i}^{(l)})^T \delta^{(l+1)}
\]

\[
\delta^{(l)} = (W^{(l)})^T \delta^{(l+1)} \circ a^{(l)}(1 - a^{(l)})
\]
Backpropagation

\[
\frac{\partial \text{Loss}}{\partial w_{ij}^{(l)}} = \frac{\partial \text{Loss}}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}} = \frac{\partial \text{Loss}}{\partial z_i^{(l+1)}} \frac{\partial (W^{(l)} a^{(i)} + b^{(l)})_i}{\partial w_{ij}^{(l)}} = \delta_i^{(l+1)} a_j^{(l)}
\]

\[
\nabla_{W^{(l)}} \text{Loss} = \delta^{(l+1)} (a^{(l)})^T
\]