Neural networks

Chapter 7 in Martin/Jurafsky

• Artificial neural networks: computational models inspired by the brain
• Properties:
  ✓ Highly interconnected
  ✓ Distributed computation/memory
  ✓ Robust to noise, failures

http://dragonoverwashington.blogspot.com/2013/03/the-blue-brain-project-making-human.html
A simple neural network

- Logistic regression as a neural network

\[ y = \sigma(w \cdot x + b) \]

Limitation of logistic regression

\[ P(y = 1|x) = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} \]
Linear models are limited

Cannot solve problems that look like this:

This data is not linearly separable

The XOR problem

While AND / OR are linearly separable, the XOR problem is not
A neural network that solves the XOR problem

Activation functions

\[ y = \frac{e^z - e^{-z}}{e^z + e^{-z}} \]

\[ y = \max(x, 0) \]

A neural network that solves the XOR problem

a) The original \( x \) space

b) The new \( h \) space
Feed forward neural networks

Let's introduce some constants to represent the dimensionalities of these vectors. Let's call the hidden layer layer 1 and the output layer layer 2. The hidden layer has dimensionality \( n_g \) and the output layer dimensionality \( n \), so

\[
\text{norm} = \frac{\|x\|}{\|y\|}.
\]

\( \text{bias} \) is the bias scalar. We represent the parameters for the entire hidden layer by combining the weight matrix (let's call it \( W \)) and one input layer (the input layer is usually not counted when enumerating layers).

If we are doing a binary task like sentiment classification, we might have a single output node, and its value \( z_i \) would represent the weight of the connection from the \( i \)-th input unit. The role of the output layer is to take the \( z \) vector, sum to one. The output layer thus gives a probability distribution across the output classes.

If we are doing multinomial classification, such as assigning a part-of-speech tag, we might have one output node for each potential part-of-speech, whose output value \( y \) would be a vector with \( y_i \) representing the probability of the \( i \)-th part-of-speech.

Thus for example given a vector \( x \) \((\text{since each hidden unit can take a different bias value). And the weight matrix \( W \) has dimensionality \( n \times n \)).

We'll call this network a 2-layer network (we traditionally don't count the input layer when numbering layers, but do count the output layer). So by this terminology we can think of a neural network classifier with one hidden layer and one output layer.
Feed forward neural networks

- A three layer network:

\[
\begin{align*}
    z^{[1]} &= W^{[1]}a^{[0]} + b^{[1]} \\
    a^{[1]} &= g^{[1]}(z^{[1]}) \\
    z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
    a^{[2]} &= g^{[2]}(z^{[2]}) \\
    \hat{y} &= a^{[2]}
\end{align*}
\]

Application: Neural language models

Similarly to regular language models: predict the next word given several preceding words

The embedding vectors can be precomputed or learned as part of the training process.
Neural language models

Learning the embedding as part of network training:

How many hidden layers?

"Theorem": A neural network with a single hidden layer can approximate any function arbitrarily well.

- But more layers can potentially do so more efficiently!
Loss function

- For classification tasks we use the cross-entropy loss:

- Binary: \( L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y\log \hat{y} + (1 - y)\log(1 - \hat{y})] \)

- Multi-class: \( L_{CE}(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log \hat{y}_i \)

- Overall loss is summed over all training examples.

Optimizing the loss function

- Want to use some variant of gradient descent

- But: the output is a complicated function of the parameters

\[
\nabla_{\theta} L(f(x; \theta), y) = \begin{bmatrix}
\frac{\partial}{\partial w_1} L(f(x; \theta), y) \\
\frac{\partial}{\partial w_2} L(f(x; \theta), y) \\
\vdots \\
\frac{\partial}{\partial w_n} L(f(x; \theta), y)
\end{bmatrix}
\]
Computation graphs

• As an example let's consider the computation of the following function:
  \[ L(a, b, c) = c(a + 2b) \]
• Want to compute: \( \frac{\partial L}{\partial a} \), \( \frac{\partial L}{\partial b} \), and \( \frac{\partial L}{\partial c} \).
• The computation can be expressed as:
  \[ d = 2b \]
  \[ e = a + d \]
  \[ L = c \cdot e \]

Computation graphs

• As an example let's consider the computation of the following function:
  \[ L(a, b, c) = c(a + 2b) \].
• It can be expressed as:
  \[ d = 2b \]
  \[ e = a + d \]
  \[ L = c \cdot e \]
• The computation graph for \( L \):
The chain rule

- Suppose we have a composite function \( f(x) = u(v(x)) \)
- The chain rule: \( \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx} \)
- Example: \( f(x) = \log(x^2 - 1) \)
  \[
  \frac{df}{dx} = \frac{1}{x^2 - 1} \cdot 2x
  \]

The chain rule over computation graphs

\[ L(a, b, c) = c(a + 2b) \quad d = 2b \quad e = a + d \quad L = c e \]
\[
\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a} \\
\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}
\]
The chain rule over computation graphs

\[ L(a, b, c) = c(a + 2b) \]

\[ d = 2b \]

\[ e = a + d \]

\[ L = c \cdot e \]

\[ \frac{\partial L}{\partial c} = e \]

\[ \frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a} \]

\[ \frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b} \]

Computation graph for a simple neural network

Computation graph for a neural network with two layers for two dimensional data:
Training using backpropagation

The method was proposed independently several times:


Comments about gradient-based training

- The neural network cost function can have many *local minima*. Therefore gradient-based training doesn’t necessarily find the global minimum.
- Is this a problem? Not necessarily.
- Compare to SVM which have a globally optimal solution, and typically faster training times.
Comments about gradient-based training

- Another issue – plateaus
- Especially problematic when you have many layers
- Arise because of saturation of activation function

Another issue – plateaus
- Especially problematic when you have many layers
- Arise because of saturation of activation function
- Solutions: non-saturating activation functions

Comments about gradient-based training

- Another issue – plateaus
- Especially problematic when you have many layers
- Arise because of saturation of activation function
- Solutions: non-saturating activation functions
Issues with neural networks

• Take a long time to train.
• Local minima of cost function.
• Lots of parameters to tweak and decisions to make: how many layers, how many neurons in each layer, activation function, learning rate.


Deep learning

Some of the most iconic companies are investing heavily in deep learning

https://www.tensorflow.org/

https://deepmind.com

http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html
Why this interest in deep learning?

- The 2012 ImageNet visual recognition challenge:
  - 1000 classes, 1,431,167 images
- Geoff Hinton’s group: 16% error using convolutional neural networks
- Closest competitor: 26%
- Current level of error: around 5%

Deep learning

Deep learning: neural architectures with lots of layers.
- An umbrella name for lots of different network architectures
- A neural network with a single hidden layer can approximate any function.
- However, a network with multiple layers can represent the target function more efficiently.
The human visual system

- The human visual system performs image processing at increasing levels of abstraction

Image from https://grey.colorado.edu/CompCogNeuro/index.php/CCNBook/Perception

NN architectures inspired by the brain

For image classification researchers have been exploring architectures that are motivated by the working of the visual system.

The architecture of the Neocognitron (Fukushima, 1980)

http://www.scholarpedia.org/article/Neocognitron
convolutional networks

The LeNet-5 network: http://yann.lecun.com/exdb/lenet/

Important ideas:

• Local features that are useful in one region are likely to be useful elsewhere (weight sharing)

• Extract local features (local receptive fields) and combine them to create global features at a more abstract level


From GPUs to TPUs

Google has recently unveiled a chip that is specifically designed for the tensor operations that are used in machine learning.

Idea: let’s sacrifice precision for speed!

Claims to be an order of magnitude faster for ml applications.

https://cloud.google.com/tpu/
deep learning software

TensorFlow (google)

PyTorch (facebook)

Keras

Should I use a (deep) neural network for my project?

- First, try something simpler (Naive Bayes, Random Forests, logistic regression).
- If Performance is good enough – you are done.
- If not, ask yourself the following questions:
  - Is the problem appropriate?
  - Do you have enough data?
  - Do you have the time and resources to train them?