Neural Networks Workshop: Training and Stochastic Gradient Descent

William Guss
wguss@berkeley.edu

Phillip Kuznetsov
philkuz@berkeley.edu

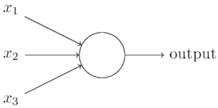
University of California, Berkeley
Robotics @ Berkeley

December 1, 2015

Overview

Today we use and train Feed-Forward Artificial Neural Networks

Perceptron Review

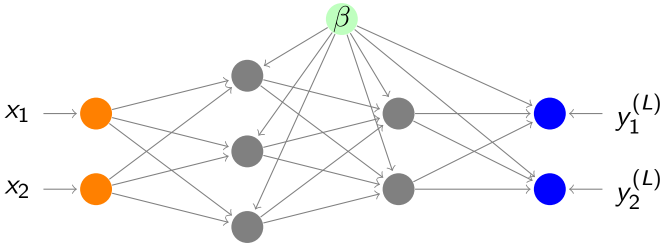


- Perceptrons are neural computation units which make *weighted* decisions:

$$p(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \sum w_i x_i + b \geq 0 \\ 0 \text{ otherwise} \end{cases}$$
$$= \text{step}\left(\sum w_i x_i + b\right)$$

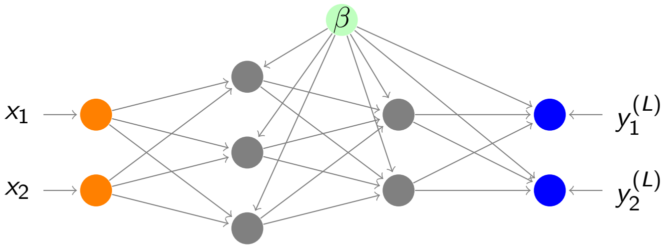
- Single perceptrons are not powerful enough, as seen last time with XOR.
- What if we want real valued output for tasks like predicting the temparature or stock prices?

Feedforward Neural Networks



- Feedforward Artifical Neural Networks (ANNs) are the *continuous* extensions of perceptrons.
- ANNs can have many layers and different nodes which are *fully connected.*
- The intuition behind this model is that each neuron in the network makes a weighted decision like the perceptron. Many *stacked* decisions allows for extremely complex logic.

A Bit of Notation



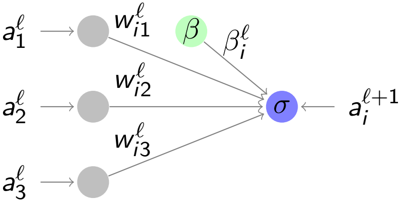
Definition

A **weight** on the ℓth layer between the jth neuron on that layer and the ith neuron on the next layer is denoted $w_{ij}^{\ell} \in \mathbb{R}$.

Definition

The **input** to the neural network is a vector $\boldsymbol{x} \in \mathbb{R}^n$ and the **output** is a vector $\boldsymbol{y} \in \mathbb{R}^m$.

A Bit of Notation



Definition

The **output** of the ith neuron on the $(\ell+1)$th layer is given by the weighted sum of its inputs

$$a_i^{\ell+1} = \sigma\left(\sum_{j\in A_i} a_j^\ell w_{ij}^\ell + \beta_i^\ell\right)$$

where A_i is the set of anterior neurons.

The Sigmoid Activation Function

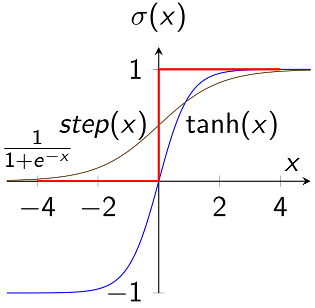
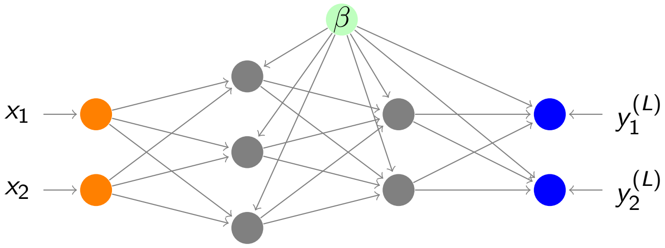


Figure: Two sigmoid functions and the perceptron $step(x)$.

Definition

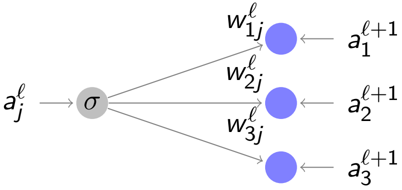
We say that $\sigma : \mathbb{R} \to \mathbb{R}$ is a **sigmoid activation function** if $\sigma(x) \to 1$ as $x \to \infty$ and $\sigma(x) \to -1$ or 0 as $x \to -\infty$

The Feed-Forward Algorithm



- The feed-forward algorithm propagates input through the neural network layer by layer.
- On every layer each neuron accumulates input from previous layers and then *activates* through the sigmoid activation function.

The Feed-Forward Algorithm



Algorithm 1 An Intuitive Version

1: **for** layer $\ell = 1$ to $L - 1$ **do**
2:     **for** neuron a_j on layer ℓ **do**
3:         $a_j.activate()$
4:         **for** neuron a_i on layer $\ell + 1$ **do**
5:             $a_i.feed(w_{ij}a_j)$
6:         **end for**
7:     **end for**
8: **end for**

Our Network Implementation

Network

- List of neurons and List of connections
- Feedforward Method
- Backpropagation Method

Neuron

- List of connections anterior and posterior connections
- Feed Method
- Activation Method
- Error update

Our Network Implementation

Connection

General weight information

- Reference to anterior and posterior neuron
- Weight value
- Feedforward

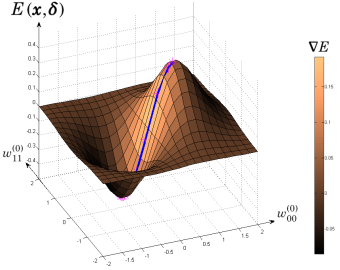
Training

- The goal of machine learning is to adjust learning parameters to better approximate a function.
- For neural networks, these learning parameters are the weights and the bias values.
- We train our network by trying to minimize a loss function:

$$E = \sum (\delta_i - y_i)^2$$

  where δ is the expected output and y is the actual output.
- Minimizing the loss function is a nonconvex problem - we need to develop heuristics to properly train the network
- One such heuristic is called gradient descent

Gradient Descent and Error Backpropagation (Calculus Heavy)



- The goal is to travel down the gradient (slope at a point) towards a minima of the error function.
- Error backpropagation is an algorithm that calculates the gradient and then updates the weights according to a *learning rate.*

Definition

We call ∇E the **gradient** of E if

$$\nabla E = \left(\frac{\partial E}{\partial w_{00}^{(0)}}, \frac{\partial E}{\partial w_{01}^{(0)}}, \dots, \frac{\partial E}{\partial w_{ij}^{(L)}} \right).$$

Algorithm 2 The Weight Update Rule

1: **for** every weight w_{ij}^{ℓ} **do**
2: calculate $\Delta w_{ij}^{\ell} = -\alpha \frac{\partial E}{\partial w_{ij}^{\ell}}$
3: $w_{ij}^{\ell} + \Delta w_{ij}^{\ell} \to w_{ij}^{\ell}$
4: **end for**

Program Reference

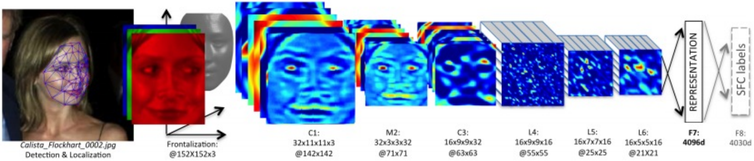
Example (Creating a network, loading a dataset, and training the net)

```
n = Network(<list of layer sizes>)
t = Trainer(n)
t.load_data(<string of the filename>)
t.interactive_step(<learning_rate> [, optional verbose flag])
```

Example (A dataset file)

```
# This is a sample data file
# You can comment with ####
# This dataset has two inputs and one desired output
{0,1} -> {1};
{1,0} -> {1};
{0,0} -> {0.3};
```

Deep Learning



- A more powerful variation of the artificial neural network that uses deep architecture, where there are many hidden layers in a network
- Represnts problems as a heirarchy of concepts and representations
- Each concept is defined in relation to simpler concepts.

Deep Learning

- Neural networks saw a number of falls from popularity
- Two main events occurred that brought neural networks back
  - The availability of better hardware and more extensive dataset (think big data)
  - The creation of more simplified training methods for deep network architectures
- With the availability of better hardware and large datasets (think big data

References

Yaniv Taigman et. al. (2014)
DeepFace: Closing the Gap to Human-Level Performance in Face Verification
Facebook AI Research

Michael Nielsen (2014)
Neural Networks and Deep Learning
http://neuralnetworksanddeeplearning.com/

The End