CS 6347

Lecture 23

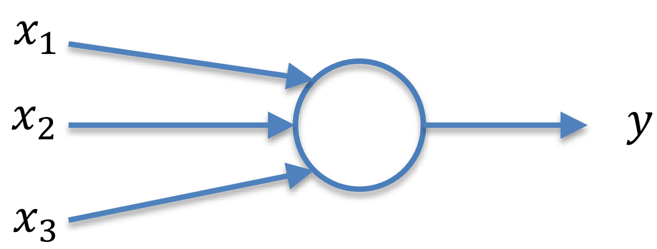
Neural Networks

Backpropagation

Restricted Boltzmann Machines

The Sigmoid Neuron

- A sigmoid neuron is an artificial neuron that takes a collection of inputs in the interval $[0,1]$ and produces an output in the interval $[0,1]$
  - The output is determined by summing up the weighted inputs plus the bias and applying the sigmoid function to the result



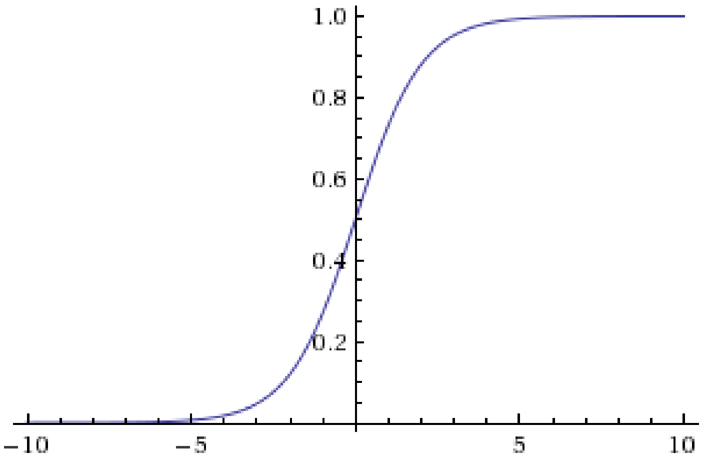
$$y = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$$

where σ is the sigmoid function

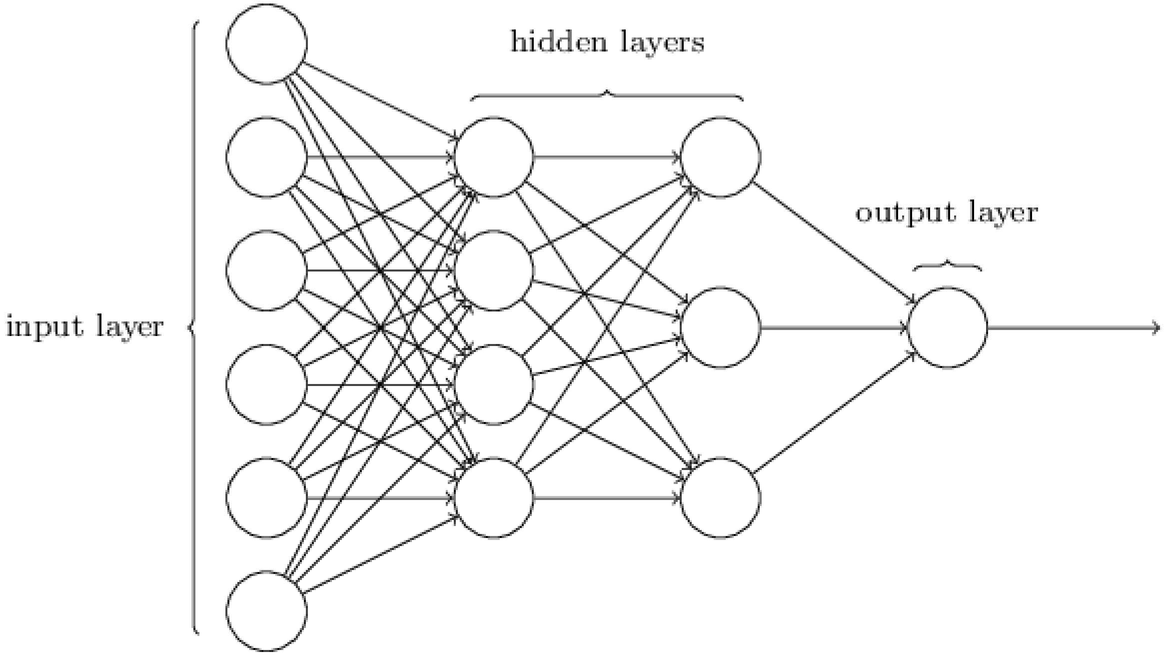
The Sigmoid Function

- The sigmoid function is a continuous function that approximates a step function

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



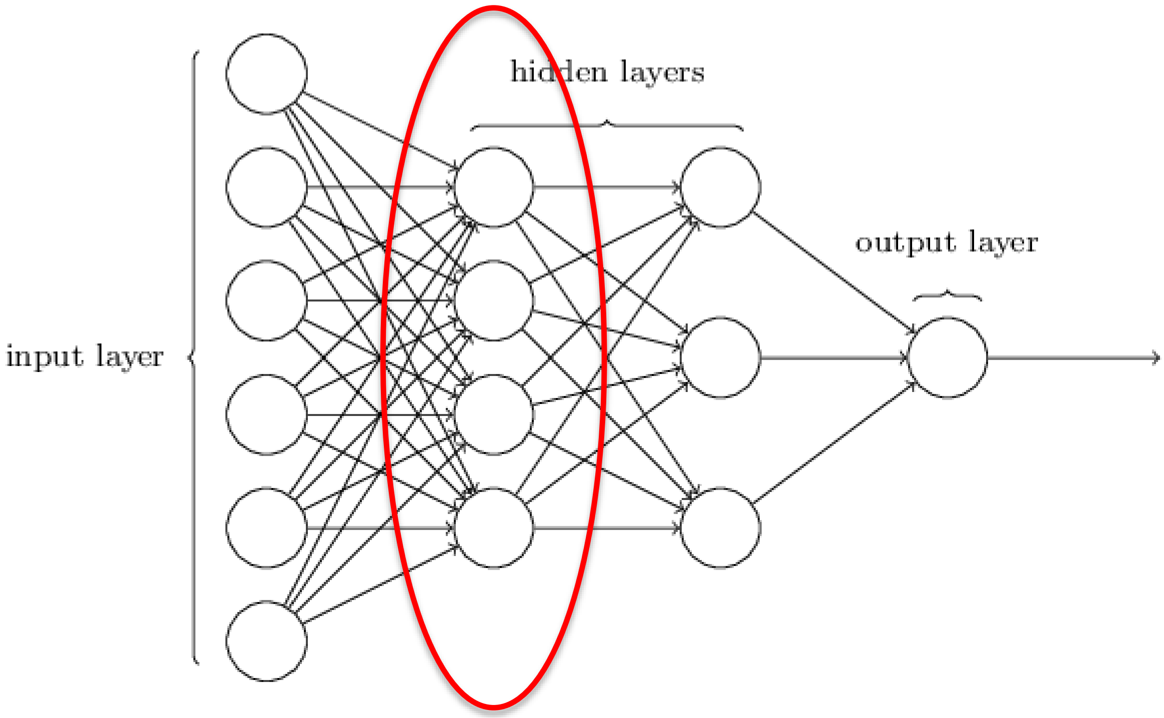
Multilayer Neural Networks



from Neural Networks and Deep Learning by Michael Nielson

Multilayer Neural Networks

NO intralayer connections



from Neural Networks and Deep Learning by Michael Nielson

Training Neural Networks

- To do the learning, we first need to define a cost function to minimize

$$C(w,b) = \frac{1}{2M}\sum_{m} \|y^m - a(x^m, w, b)\|^2$$

- The data consists of input output pairs $(x^1, y^1), \dots, (x^M, y^M)$
- $a(x, w, b)$ is the output of the neural network for the m^{th} sample
- w and b are the weights an biases

Gradient of the Cost Function

- The derivative of the cost function is relatively straightforward to calculate

$$\frac{\partial C(w,b)}{\partial w_k} = \frac{1}{M}\sum_m \left[y^m - \frac{\partial a(x^m, w, b)}{\partial w_k} \right]$$

  - To compute the derivative of a, use the chain rule and the derivative of the sigmoid function

$$\frac{d\sigma(z)}{dz} = \sigma(z) \cdot (1 - \sigma(z))$$

  - This gets complicated quickly with lots of layers of neurons

Stochastic Gradient Descent

- To make the training more practical, stochastic gradient descent is used instead of standard gradient descent

- The idea of stochastic gradient descent is to approximate the gradient of a sum by sampling a few indices uniformly at random and averaging

$$\nabla_x \sum_{i=1}^{n} f_i(x) \approx \frac{1}{K} \sum_{k=1}^{K} \nabla_x f_{i^k}(x)$$

here, each i^k is sampled uniformly at random from $\{1, \ldots, n\}$

Computing the Gradient

- We'll compute the gradient for a single sample

$$C(w, b) = \|y - a(x, w, b)\|^2$$

- Some definitions:
  - L is the number of layers
  - a_j^l is the output of the j^{th} neuron on the l^{th} layer
  - z_j^l is the input of the j^{th} neuron on the l^{th} layer

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

  - δ_j^l is defined to be $\frac{\partial C}{\partial z_j^l}$

Computing the Gradient

For the output layer, we have the following partial derivative

$$\frac{\partial C}{\partial z_j^L} = -\left(y_j - a_j^L\right)\frac{\partial a_j^L}{\partial z_j^L}$$

$$= -\left(y_j - a_j^L\right)\frac{\partial \sigma\left(z_j^L\right)}{\partial z_j^L}$$

$$= -\left(y_j - a_j^L\right)\sigma\left(z_j^L\right)\left(1 - \sigma\left(z_j^L\right)\right)$$

- For simplicity, we will denote the vector of all such partials for each node in the l^{th} layer as δ^l

Computing the Gradient

For the $L-1$ layer, we have the following partial derivative

$$
\begin{aligned}
\frac{\partial C}{\partial z_k^{L-1}} &= \sum_j \left(a_j^L - y_j\right) \frac{\partial a_j^L}{\partial z_k^{L-1}} \\
&= \sum_j \left(a_j^L - y_j\right) \frac{\partial \sigma\left(z_j^L\right)}{\partial z_k^{L-1}} \\
&= \sum_j \left(a_j^L - y_j\right) \sigma\left(z_j^L\right) \left(1 - \sigma\left(z_j^L\right)\right) \frac{\partial z_j^L}{\partial z_k^{L-1}} \\
&= \sum_j \left(a_j^L - y_j\right) \sigma\left(z_j^L\right) \left(1 - \sigma\left(z_j^L\right)\right) \frac{\partial \sum_{k'} w_{jk'}^L a_{k'}^{L-1} + b_j^L}{\partial z_k^{L-1}} \\
&= \sum_j \left(a_j^L - y_j\right) \sigma\left(z_j^L\right) \left(1 - \sigma\left(z_j^L\right)\right) \sigma\left(z_k^{L-1}\right) \left(1 - \sigma\left(z_k^{L-1}\right)\right) w_{jk}^L \\
&= \left((\delta^L)^T w_{*k}^L\right) \left(1 - \sigma\left(z_k^{L-1}\right)\right) \sigma\left(z_k^{L-1}\right)
\end{aligned}
$$

Computing the Gradient

- We can think of w^l as a matrix

- This allows us to write

$$\delta^{L-1} = \left((\delta^L)^T w^L\right)\left(1 - \sigma(z^{L-1})\right)\sigma(z^{L-1})$$

where $\sigma(z^{L-1})$ is the vector whose k^{th} component is $\sigma\left(z_k^{L-1}\right)$

- Applying the same strategy, for $l < L$

$$\delta^l = \left((\delta^{l+1})^T w^{l+1}\right)\left(1 - \sigma\left(z^l\right)\right)\sigma\left(z^l\right)$$

Computing the Gradient

- Now, for the partial derivatives that we care about

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$$

- We can compute these derivatives one layer at a time!

Backpropagation

- Compute the inputs/outputs for each layer by starting at the input layer and applying the sigmoid functions
- Compute δ^L and the output layer
- Starting from the output layer and working backwards, compute $\delta^{L-1}, \delta^{L-2}, \ldots$
- Compute the gradient of the objective function

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$$

Backpropagation

- Many ways to improve this approach
  - Use a regularizer!   (better generalization)
  - Try other cost functions
  - Initialize the weights of the network more cleverly
    - Random initializations are likely to be far from optimal
  - etc.
- The algorithm can have numerical difficulties if there are a large number of layers

Restricted Boltzmann Machines

- A special kind of undirected graphical model
  - Potentials are parameterized as they are in the Ising model
  - Consists of a bipartite graph in which one side of the partition is observed and the other side of the partition is unobserved
    - A single “hidden layer” with no edges between hidden variables
  - Can be made to perform well for digit classification

Restricted Boltzmann Machines

- Because of the properties of MRFs, the hidden variables are conditionally independent given the observed variables
  - We can do learning in these networks by exploiting this fact
  - Instead of the EM algorithm, we can employ sampling based techniques to approximate the gradient of the log-likelihoof
    - A special type of approximate sampling where we only draw a few samples is referred to as contrastive divergence
      - Start the Gibbs sampler with one of the observed samples x^m, use it to sample the hidden variables, then use the hidden variables to generate a new x