Neural Networks

Siddharth Ramakrishnan and Bryce Harlan

Works Cited

- "Artificial Neural Networks." *Wikipedia*. Wikimedia Foundation, n.d. Web. 18 Nov. 2015. <https://en.wikipedia.org/wiki/Artificial_neural_network#History>.
- Castro, R. M. "Maximum Likelihood Estimation and Complexity Regularization." N.p., n.d. Web. <http://www.win.tue.nl/~rmcastro/AppStat2011/files/MLE_partI.pdf>.
- "Derivation of Backpropagation." Swarthmore University, n.d. Web. <http://web.cs.swarthmore.edu/~meeden/cs81/f15/BackPropDeriv.pdf>.
- "Jensen's Inequality." *Wikipedia*. Wikimedia Foundation, n.d. Web. 18 Nov. 2015. <https://en.wikipedia.org/wiki/Jensen%27s_inequality>.
- "Keras: Theano-based Deep Learning Library." *Keras Documentation*. N.p., n.d. Web. 18 Nov. 2015. <http://keras.io/>.
- Ng, Andrew. "CS294A Lecture Notes." *Stanford Computer Science*. Stanford University, n.d. Web. <https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf>.
- "UCI Machine Learning Repository: Wine Data Set." *UCI Machine Learning Repository: Wine Data Set*. N.p., n.d. Web. 18 Nov. 2015. <http://archive.ics.uci.edu/ml/datasets/Wine>.
- "Unsupervised Feature Learning and Deep Learning Tutorial." *Unsupervised Feature Learning and Deep Learning Tutorial*. N.p., n.d. Web. 18 Nov. 2015. <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>.
- LeCun, Yann. "Theoretical Framework for Backpropagation." (n.d.): n. pag. University of Toronto. Web. <http://yann.lecun.com/exdb/publis/pdf/lecun-88.pdf>.

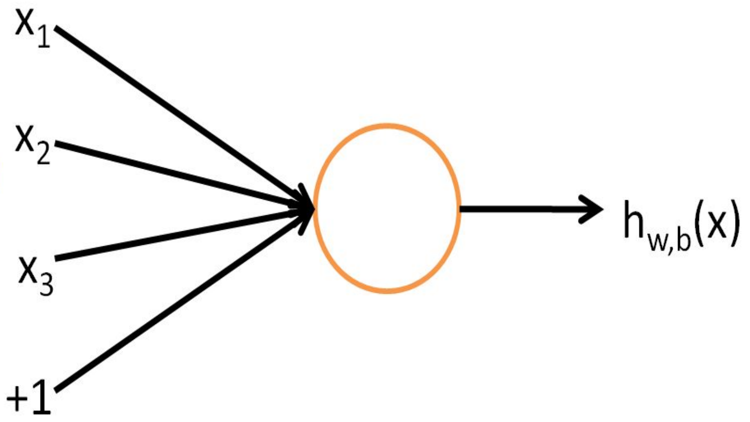
Why We're Interested

History

- Early versions were used to model neurons in the brain, hence the name
  - 1940s
- Rosenblatt (1958)
  - Perceptron and XOR
- Minsky and Papert (1969)
  - Discovered that a single layer network could not compute the XOR
  - Computers were not powerful enough to make significant progress in research
- Backpropagation algorithm was proposed by Paul Werbos in 1975
  - Formalized by Yann Lecun in 1985
- Deep Learning
  - Applications to machine learning such as classification and regression
  - Stacking of multiple highly-nonlinear layers

Simplest Neural Net: Perceptron

$$h_{W,b}(x) = f(W^T X) = f(\sum_{i=1}^{n} W_i x_i + b)$$



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

$$f'(z) = f(z)(1 - f(z))$$

Source: http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

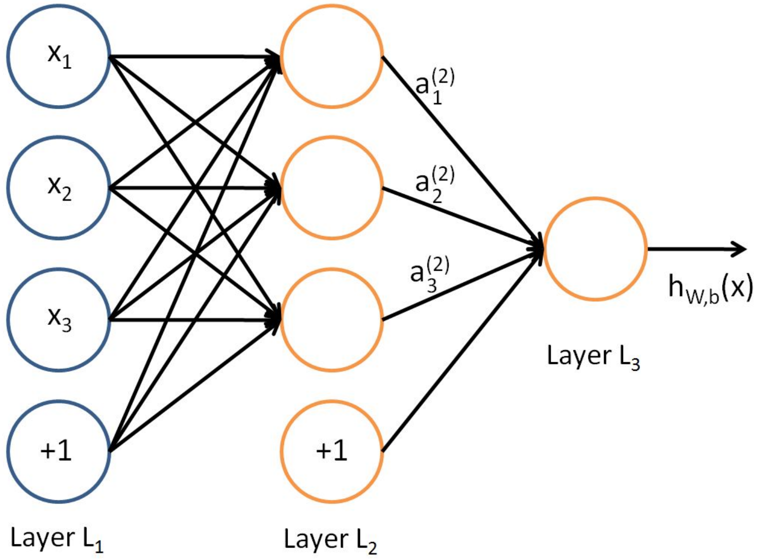
Adding a “hidden” layer

L_1 is called the *input layer*

L_2 is called a *hidden layer*

L_3 is called an *output layer*

FORWARD PROPAGATION



$$a_i^{(2)} = f(W_{i1}^{(1)}x_1 + W_{i2}^{(1)}x_2 + \ldots + W_{in}^{(1)}x_n) + b_i^{(1)}$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{12}^{(2)}a_3^{(2)} + b_1^{(2)})$$

Source: http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

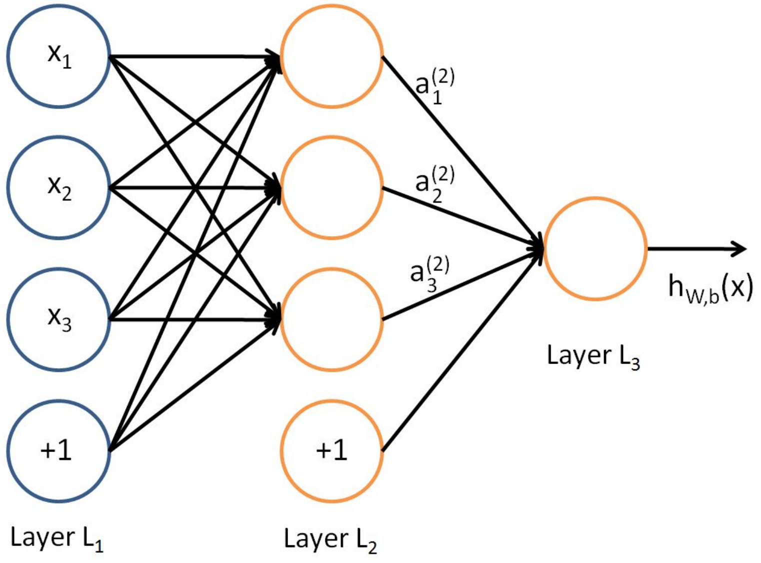
A Cleaner Notation

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)}$$

$$a_i^{(l)} = f(z_i^{(l)})$$

Forward propagation with new notation:

$$z^{(2)} = W^T X + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)} a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$



Backpropagation Algorithm

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)}$$

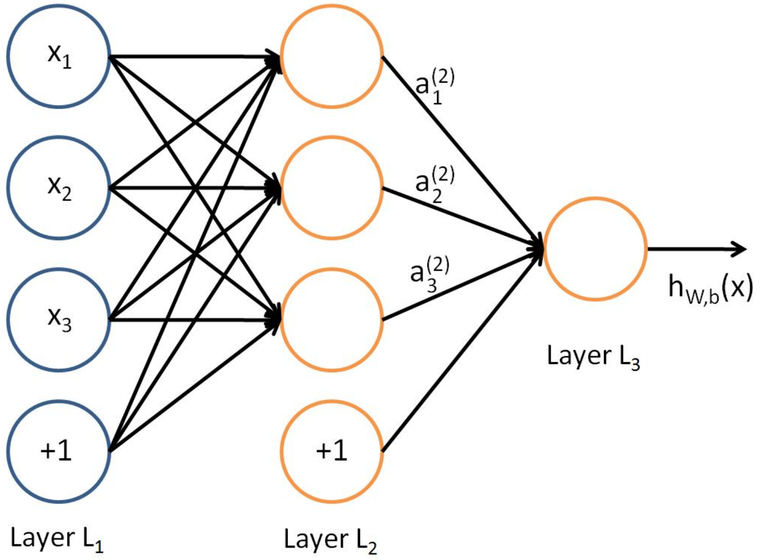
$$a_i^{(l)} = f(z_i^{(l)})$$

Error for a single training example (x, y):

$$E(W, b; x, y) = \frac{1}{2}||h_{W,b}(x) - y||^2$$

We want to change W to reduce E_{tot}

$$\Delta W \propto -\frac{\partial E_{tot}}{\partial W}$$



Backpropagation Algorithm

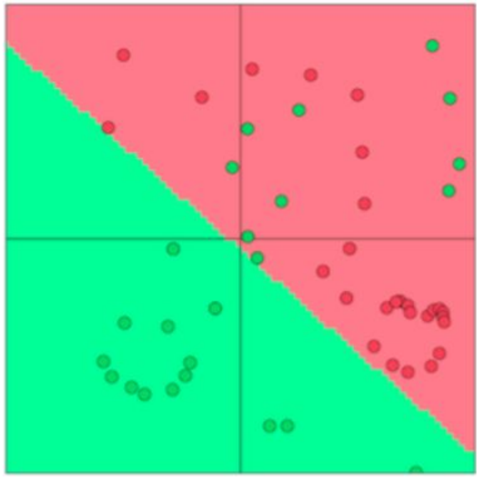
Phase 1: Propagation

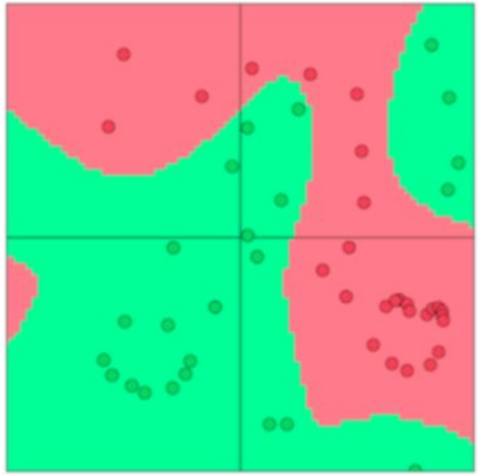
1) Perform a feed-forward pass to compute activations
2) Find δ for each neuron in the network (as shown)

Phase 2: Weight Update

1) Multiply the output activation by the input δ
2) Subtract a fixed percentage of this gradient

Classification


Linear Decision Boundary


Non-Linear Decision Boundary

Source: http://cs224d.stanford.edu/lecture_notes/LectureNotes2.pdf

Code

Classification Demo!

Classifying types of wine based off 13 measured properties including hue and magnesium content

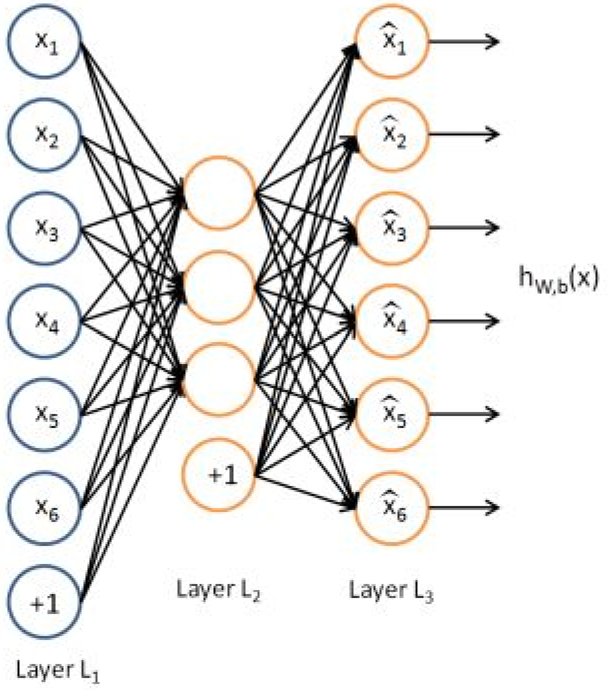
3 wines in the data set (classes), 178 data points

From the UCI data set collection

http://keras.io/

Autoencoders

- Learns a new encoding of the identity function
- Used for dimensionality reduction
  - like principal component analysis (PCA)
- Sparsity constraint
  - Forces the network to find an efficient encoding



Autoencoders

- Sparsity parameter (ρ)
  - Desired average activation of the hidden layer
  - We want it to be small (say ρ=0.05) because that means less neurons are being used and a more efficient encoding is being found
  - We want: $\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m} a_j^{(2)}(x_i)$, our actual parameter to be close to ρ
    - Where $a_j(x_i)$ is the average activation of node j in layer 2 across all data points
    - So we make the new loss function: $J_{sparse}(W,b) = J(W,b) + \beta\sum_{j=1}^{J} KL(\rho||\hat{\rho})$
      - Where J is the total number of neurons in L_2

Kullback-Leibler Divergence

- KL(P||Q)
  - Measures the information lost when distribution Q is used to approximate distribution P
  - Non-symmetric difference measure between two distributions

$$KL(P||Q) = \sum_i P(i) log \frac{P(i)}{Q(i)}$$

- We can see that if P=Q, the KL(P||Q) = 0
- Our sparsity parameters can be seen as a bernoulli distribution with mean ρ

Future of the Field

- Recurrent Neural Networks
  - LSTM
  - Useful for time series data and NLP
- Convolutional Neural Networks
  - Image recognition
- Feature Learning
  - Step past autoencoders
- Deep Learning

Thanks for listening!