Introduction to Neural Networks and Theano

Kushal Arora
Karora@cise.ufl.edu

Outline

1. Motivation
2. Logistic Regression to Neural Networks
3. Deep Networks and Issues
4. Autoencoders and Stacked Autoencoders
5. Why Deep Learning Works
6. Theano Overview
7. Code Hands On

#1 Learning Representation

- Handcrafting features is inefficient and time consuming
- Must be done again for each task and domain
- The features are often incomplete and highly correlated

#2 Distributed Representation

- Need to move beyond one-hot representations such as from clustering algorithms, k-nearest neighbors etc
- O(n) parameters/examples for O(N) input regions but we can do better O(k) parameters/inputs for $O(2^k)$ input regions
- Similar to multiclustering, where multiple clustering algorithms are applied in parallel or same clustering algorithm is applied to different input region

#3 Unsupervised Feature Learning

- Most of the current ML systems require labeled training data
- Classification is as good as features
- A good model of observed data can really simplify the classification model.

#4 Learning multiple level of representation

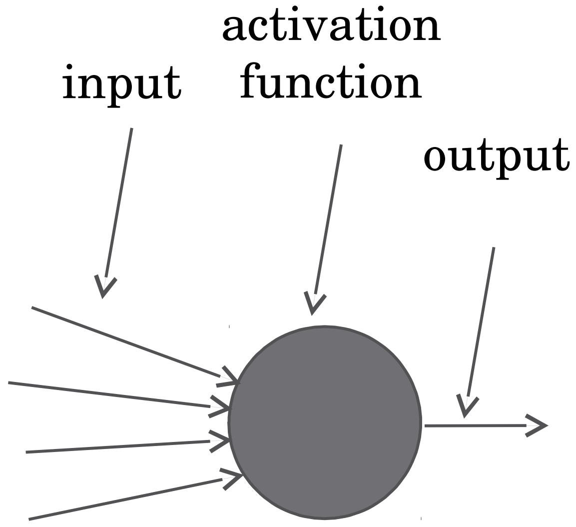
- Deep architectures promote reuse of features
- Deep architecture can potentially lead to progressively more abstract features at higher layer
- Good intermediate representation can be shared across tasks
- Insufficient depth can be exponentially inefficient

The Basics

- Building block of the network is a neuron.
- The basic terminologies
  - Input
  - Activation layer
  - Output
- Activation function is usually of the form

$$h_{(w,b)} = f\left(w^T . x + b\right)$$

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



Logistic Regression

- Logistic regression is a probabilistic, linear classifier
- Parameterized by W and b
- Each output is probability of input belonging to class y_i
- Prob of x being member of class Y_i is calculated as

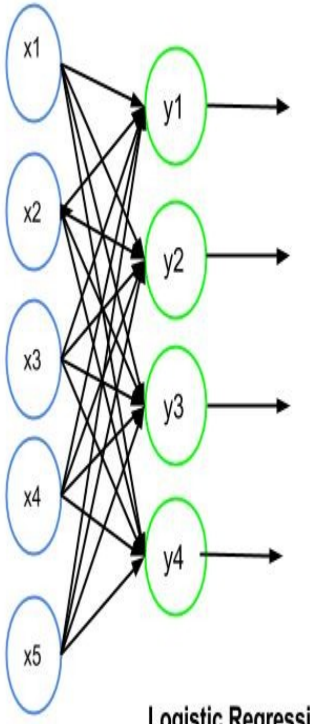
$$P(Y=Y_i|x)=softmax_i(Wx+b)$$

where

$$softmax_i(Wx+b)=\frac{e^{W_i^{Tx}+b_i}}{\sum_j e^{W_j^{Tx}+b_i}}$$

and

$$y_{pred}=argmax_{Y_i} P(Y=Y_i|x)$$



Logistic Regression

Logistic Regression – Training

- The loss function for logistic regression is negative log likelihood, defined as

$$\mathcal{L}(\theta = \{W, b\}, \mathcal{D}) = \sum_{i=0}^{|\mathcal{D}|} \log(P(Y = y^{(i)} | x^{(i)}, W, b))$$
$$\ell(\theta = \{W, b\}, \mathcal{D}) = -\mathcal{L}(\theta = \{W, b\}, \mathcal{D})$$

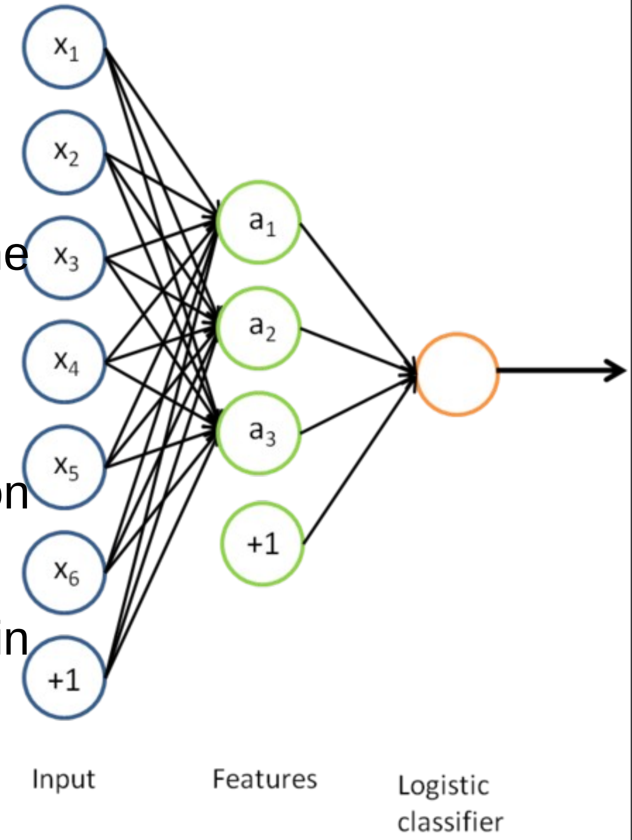
- The loss function minimization is done using stochastic gradient descent or mini batch stochastic gradient descent.

$$\theta^{k+1} = \theta^k - \epsilon_k \frac{\partial L(\theta^k, z)}{\partial \theta^k}$$

- The function is convex and will always reach global minima which is not true for other architectures we will discuss.

Multilayer preceptron

- An MLP can be viewed as a logistic regression classifier where the input is first transformed using a learned non-linear transformation.
- The non linear transformation can be a sigmoid or a tanh function.
- Due to addition of non linear hidden layer, the loss function now is non convex
- Though there is no sure way to avoid minima, some empirical in initializing the Weight matrix help.



Multilayer preceptron – Training

- Let D be the size of input vector and there be L output labels. The output vector (of size L) will be given as

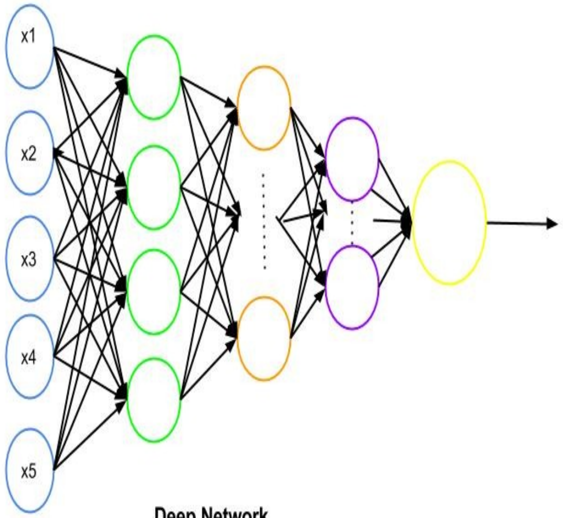
$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))),$$

where G and s are activation functions.

- The hard part of training is calculating the gradient for stochastic gradient descent and of course avoid the local minima.
- Back-propagation used as an optimization technique to avoid calculation gradient at each step. (It is like Dynamic programming for derivative chain rule recursion)

Deep Neural Nets and issues

- Generally networks with more than 2-3 hidden layers are called deep nets.
- Pre 2006 they performed worse than shallow networks.
- Though hard to analyze the exact cause the experimental results suggested that gradient-based training for deep nets got stuck in local mininma
- It was difficult to get good generalization.
- Random initialization which worked for shallow network couldn't be used deep networks.
- The issue was solved using a unsupervised pre-training of hidden layers followed by a fine tuning or supervised training.



Deep Network

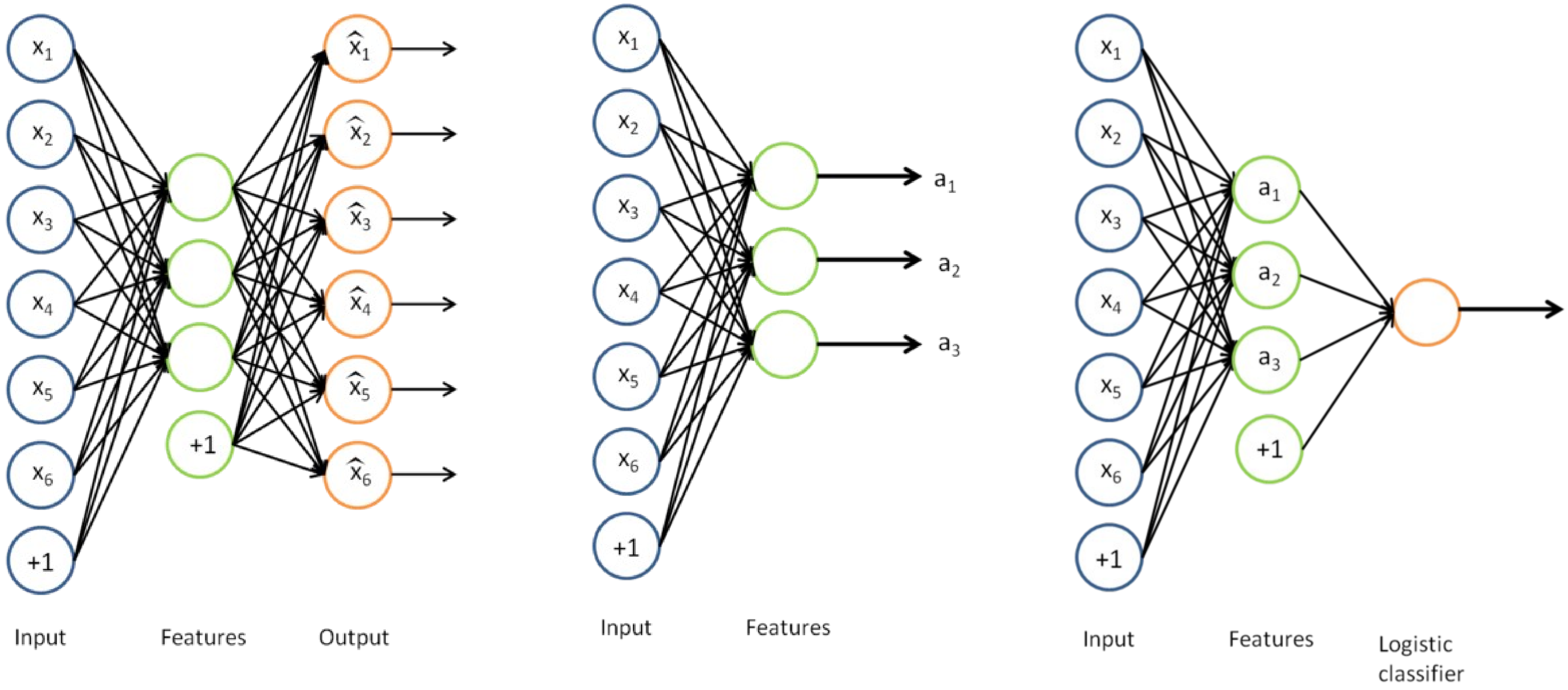
Auto-Encoders

- Multilayer neural nets with target output = input.
- Reconstruction = decoder(encoder(input))

$$a = \tanh(Wx + b)$$
$$x' = \tanh(W^{Tx} + c)$$
$$L = \|x' - x\|_2$$

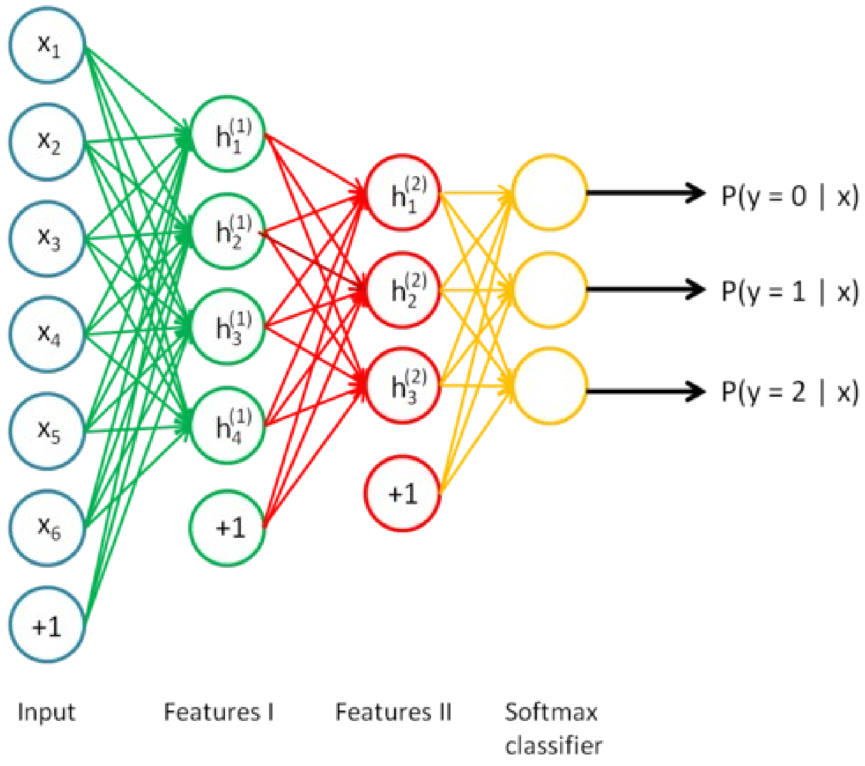
- Objective is to minimize the reconstruction error.
- PCA can be seen as a auto-encoder with $a = Wx$ and $x' = W^T a$
- So autoencoder could be seen as an non-linear PCA which tries of learn latent representation of input.

Auto-Encoders - Training

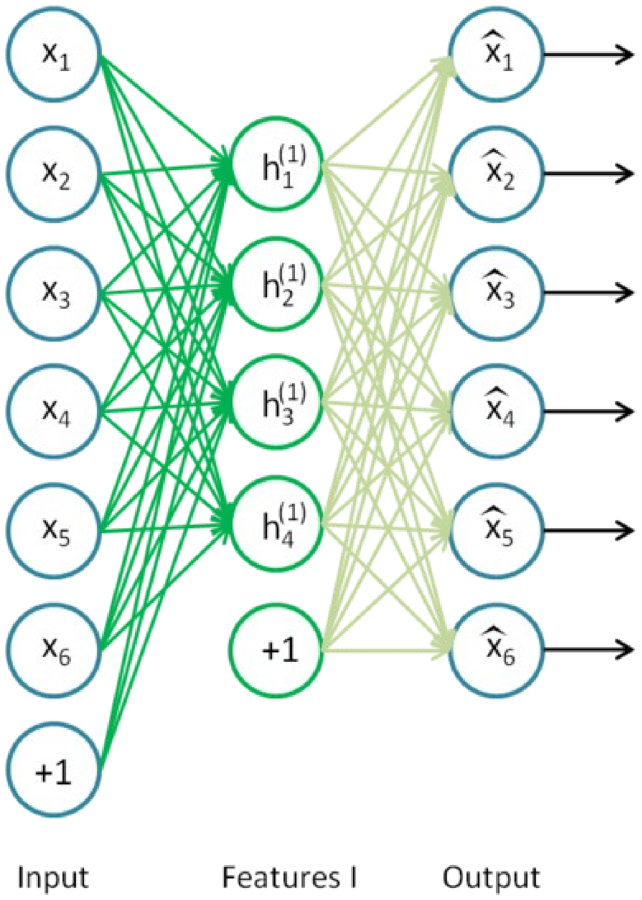


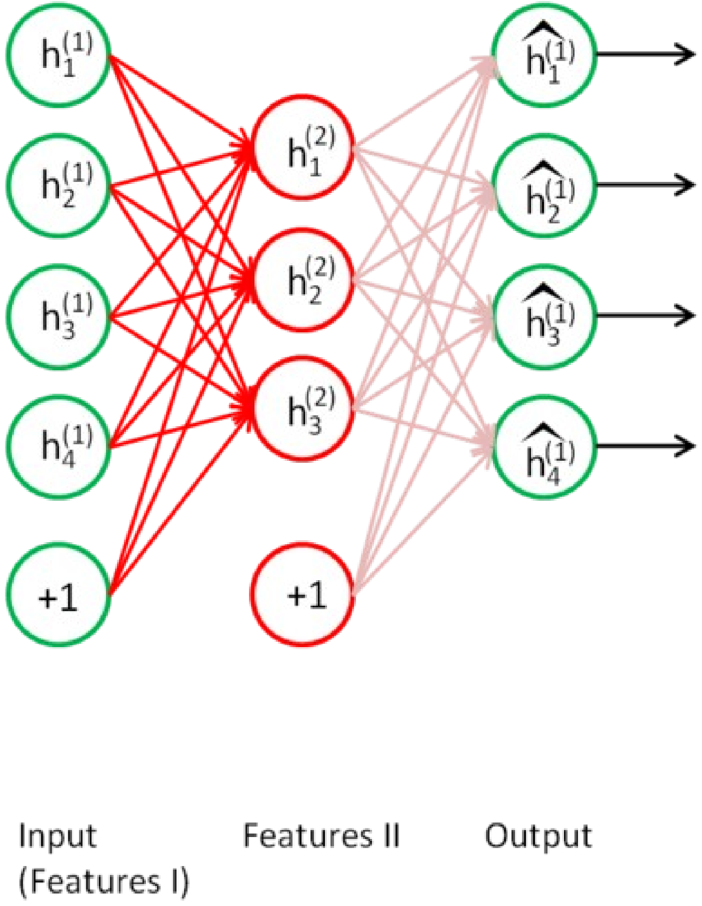
Stacked Auto-Encoders

- One way of creating deep networks.
- Other is Deep Belief Networks that is made of stacked RBMS trained greedily.
- Training is divided into two steps
  - Pre Training
  - Fine Tuning
- In Pre Training auto-encoders are recursively trained one at a time.
- In second phase, i.e. fine tuning, the whole network is trained using to minimize negative log likelihood of predictions.
- Pre Training is unsupervised but post training is supervised



Pre-Training





Why Pre-Training Works

- Hard to know exactly as deep nets are hard to analyze
- Regularization Hypothesis
  - Pre-Training acts as adding regularization term leading to better generalization.
  - It can be seen as good representation for P(X) leads to good representation of P(Y|X)
- Optimization Hypothesis
  - Pre-Training leads to weight initialization that restricts the parameter space near better local minima
  - These minimas are not achievable via random initialization

Other Type of Neural Networks

- Recurrent Neural Nets
- Recursive Neural Networks
- Long Short Term Memory Architecture
- Convolutional Neural Networks

Libraries to use

- Theano/Pylearn2 (U Motreal, Python)
- Caffe (UCB, C++, Fast, CUDA based)
- Torch7 (NYU, Lua, supported by Facebook)

Introduction to Theano

What is Theano?

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.

Why should it be used?

- Tighter integration with numpy
- Transparent use of GPU
- Efficient symbolic representation
- Does lots of heavy lifting like gradient calculation for you
- Uses simple abstractions for tensors and matrices so you don't have to deal with it

Theano Basics (Tutorial)

- How To:
  - Declare Expression
  - Compute Gradient

- How expression is evaluated. (link)
- Debugging

Implementations

- Logistic Regression
- Multilayer preceptron
- Auto-Encoders
- Stacked Auto-Encoders

Thank You!