Introduction to Statistical Learning
with applications in Python

Based on "Introduction to Statistical Learning, with applications in R" by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Neural Networks

Basic Concepts, Single Neurons, Multiple Neurons, Deep Networks, Practical Considerations

Kurt Rinnert

Physics Without Frontiers

The Abdus Salam International Centre for Theoretical Physics

University of Liverpool

Abstract

We now introduce an extremely flexible learning approach: artificial neural networks.
Neural network, in particular deep neural networks, have become very popular in machine learning. The concept is old, but recent advancements in computing have made things feasible not too long ago.
We will present and explain the basic concepts but not go beyond *fully connected feed-forward networks* (FCNN), also known as *multilayer perceptrons*.

Overview

Feedforward Neural Networks

Training Neural Networks

Deep Neural Networks

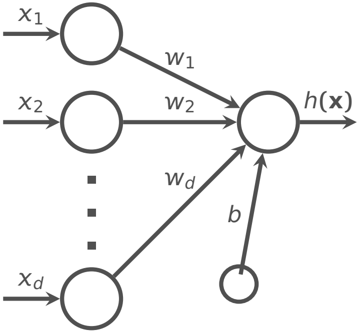
Practical Considerations

Further Reading

We assume a classification setting and can only cover some basics here.

Artificial Neuron

- Neuron pre-ativation (or input activation)
  $a(x) = b + \sum_i w_i x_i = b + \mathbf{w}^T \mathbf{x}$
- Neuron (output) activation
  $h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$
- $\mathbf{w}$ are the connection weights
- b is the neuron bias
- $g()$ is the activation function

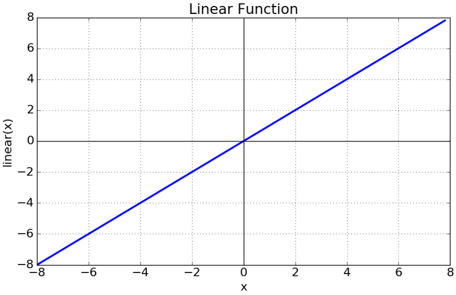


This is the basic building block of all that follows.

Activation Functions

Linear Function: $g(a) = a$

- Range of g same as domain
- Not very interesting

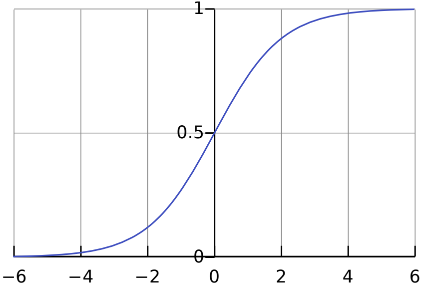


Only linear transformations can be modeled.

Activation Functions

Sigmoid Function: $g(a) = \mathrm{sigm}(a) = \frac{1}{1+\exp(-a)}$

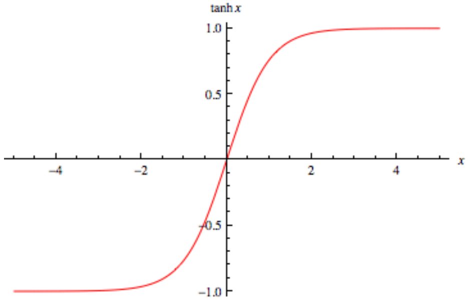
- Maps the pre-activation a to $[0, 1]$
- Always positive
- Bounded
- Strictly increasing



Non-linear models possible.

Activation Functions

tanh Function: $g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a}{\exp(a) + \exp(-a)}$



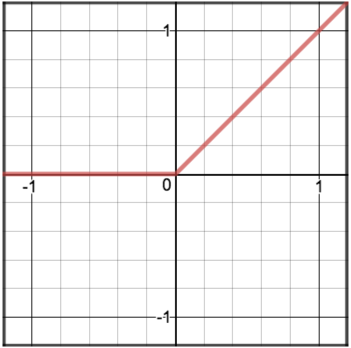
- Maps the pre-activation a to $[-1, 1]$
- Positive and negative
- Bounded
- Strictly increasing

Non-linear models possible.

Activation Functions

Rectified Linear Function (Unit): $g(a) = \text{reclin}(a) = \text{relu}(a) = \max(0, a)$

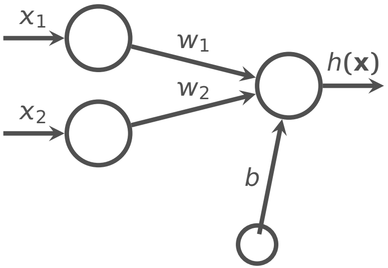
- Bound below by 0
- No upper bound
- Monotonically increasing
- Tends to create "sparse" neurons



A very popular choice.

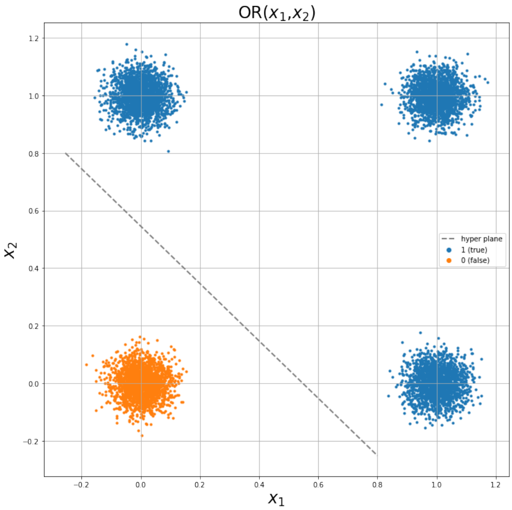
Capacity of a Single Neuron

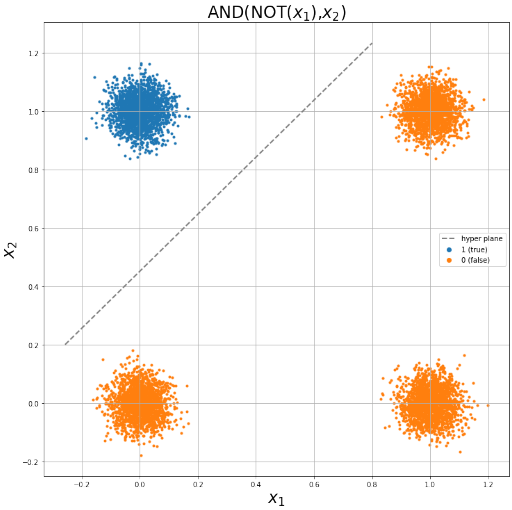
- Can separate two classes…
- …if separation is linear (hyperplane)
- Sigmoid activation allows for probability interpretation
- Cut at 0.5 for classification



A single neuron can act as a binary classifier.

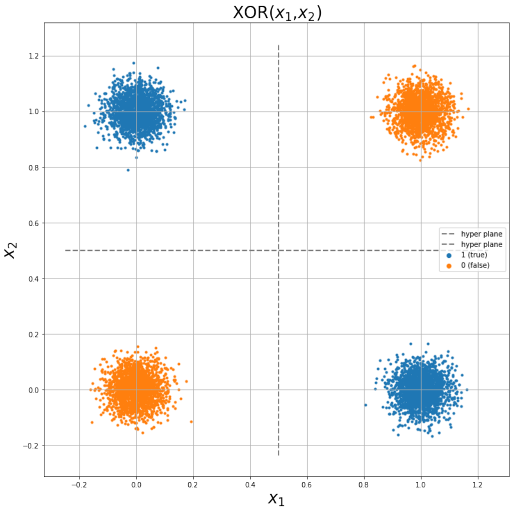
Linear Classification Examples

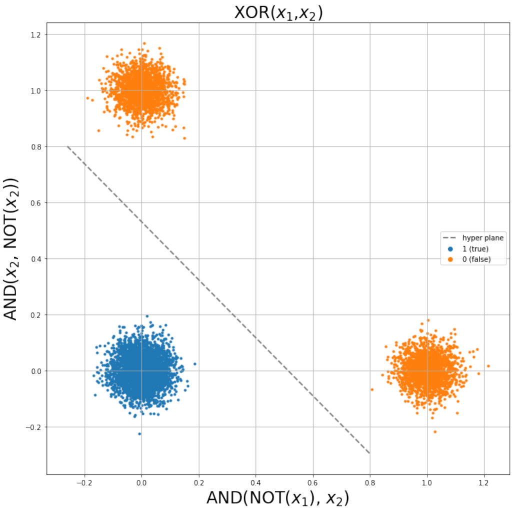




Can be separated by a single neuron.

Non-Linear Example

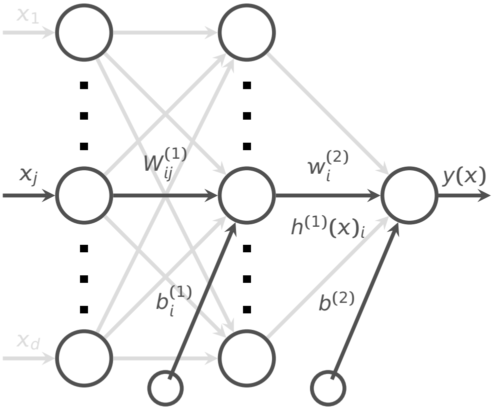




Additional neurons can encode the transformation!

One Hidden Layer

- Hidden layer pre-activation:
  $\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$
- Hidden layer activation:
  $\mathbf{h}^{(1)}(\mathbf{x}) = \mathbf{h}^{(1)}(\mathbf{a}(\mathbf{x}))$
- Output Layer:
  $\mathbf{y}(\mathbf{x}) = \mathbf{o}(\mathbf{b}^{(2)} + \mathbf{w}^{(2)T}\mathbf{h}^{(1)}\mathbf{x})$



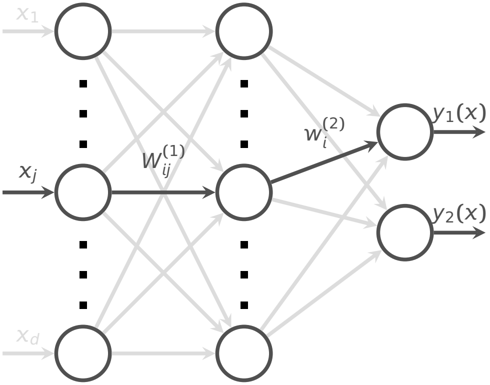
The function $o()$ is the output layer activation.

Multiple Classes

- Softmax as output activation:

  $$y_j(\mathbf{x}) = o(\boldsymbol{\alpha})_j = \frac{e^{a_j}}{\sum_{k=1}^{K} e^{a_k}}$$

  for $j = 1, \ldots, K$
- Strictly positive
- Sums to one



Softmax provides normalized probabilities.

Empirical Risk Minimization

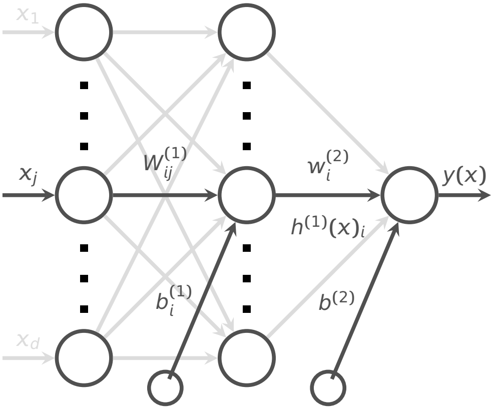
- Framework to design learning algorithms
  $\arg\min \frac{1}{T}\sum_t l(y(\mathbf{x}^{(t)};\boldsymbol{\theta}), y^{(t)}) + \lambda\Omega(\boldsymbol{\theta})$
- $\boldsymbol{\theta}$ is the set of all parameters
- $l(y(\mathbf{x}^{(t)};\boldsymbol{\theta}), y^{(t)})$ is the loss function
- $\Omega(\boldsymbol{\theta})$ is a regularizer (penalizes certain values of $\boldsymbol{\theta}$)
- the loss function is an upper bound on the classification error

Learning is cast as optimization.

Stochastic Gradient Descent (SDG)

Algorithm for update after each seen example:

- initialize $\boldsymbol{\theta}$ (all parameters)
- Then, for N iterations (epochs):
- For each training example $(\mathbf{x}^{(t)}, \mathbf{x}^{(t)})$:
- $\Delta = -\Delta_\theta l(f(x^{(t)}, \theta), y^{(t)}) - \lambda \Delta_\theta \Omega(\theta)$
- $\theta \leftarrow \theta + \alpha\Delta$

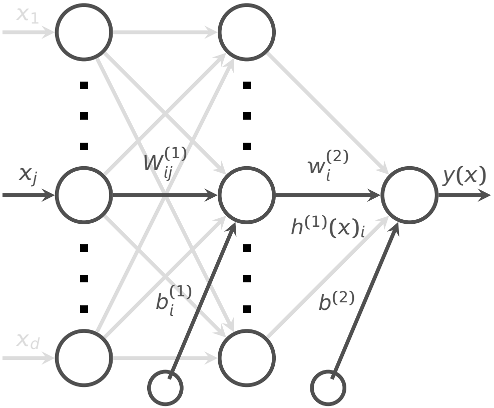


Meta parameters like α are not optimized!

Ingredients for SDG

To apply the algorithm wee need:

- The loss function $l(f(x^{(t)}, \theta), y^{(t)})$
- The parameter gradients, $\Delta_\theta l(f(x^{(t)}, \theta), y^{(t)})$ etc.
- The regularizer Ω and its gradiend $\Delta_\theta \Omega$
- An initialization method
- A method to compute the gradients in practice



Gradient computation is done by back-propagation.

Regularization

L2 Regularization

$$\Omega(\theta) = \sum_k \sum_i \sum_j (W_{i,j}^{(k)})^2$$

- Only applied to weights, not biases
- Causes weights to decay

Can be interpreted as a Gaussian prior.

Regularization

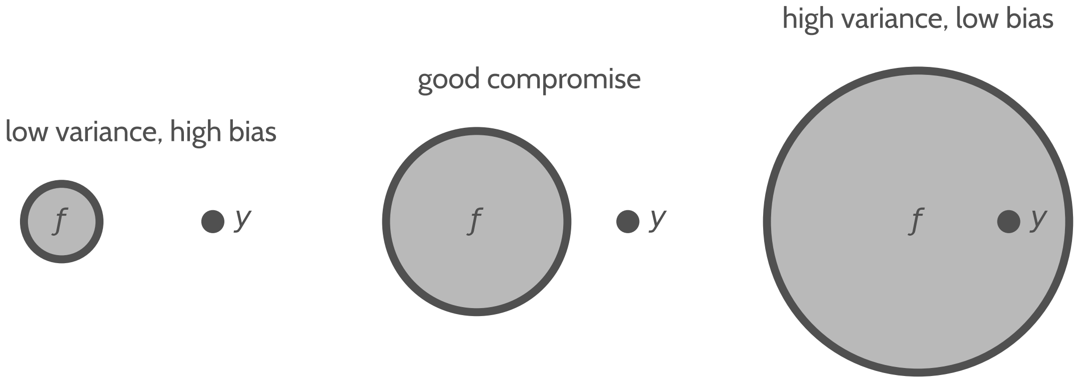
L1 Regularization

$$\Omega(\theta) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

- Only applied to weights, not biases
- Will push some weiths to exactly zero

Can be interpreted as a Laplacian prior.

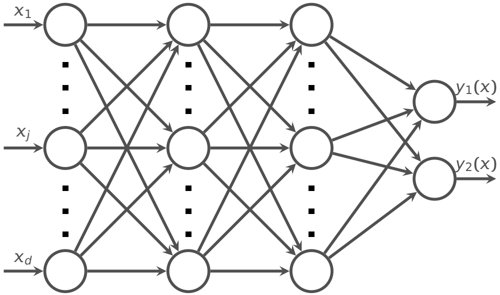
Variance vs. Bias



This intuitively motivates regularization.

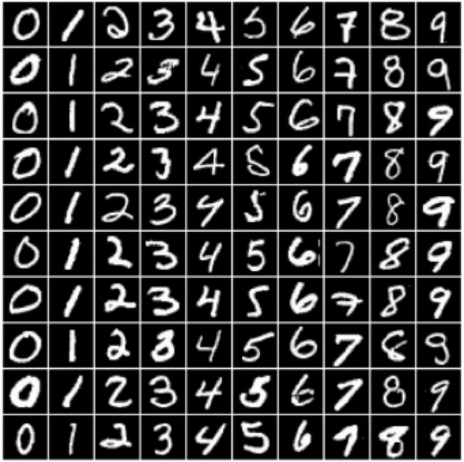
Deep Neural Networks

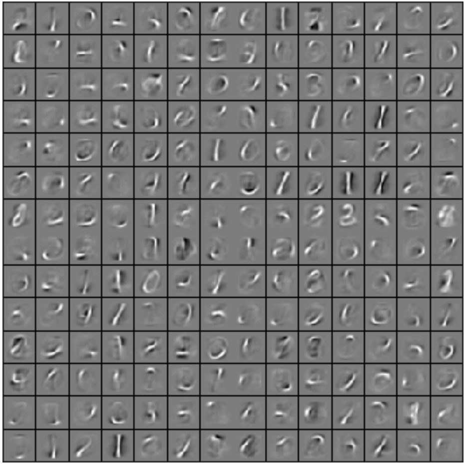
- Instance of multilayer representation
- Each layer corresponds to "distributed" representation
- There motivations from biology (visual cortex)
- Feature extraction
- Grouping of features
- Recognition of classes



More compact representation than single layer.

Example: MNIST, Handwritten Digits





Multiple classes. Feature extraction.

Training Difficulties

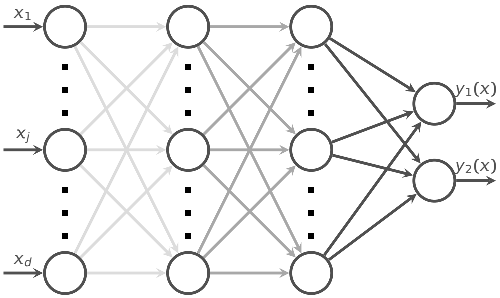
Harder optimization problem

- → vanishing gradient problem
- Underfitting
- Saturated units block propagation
- Can be mitigated by pre-training followed by refining

High variance / low bias situation

- Many parameters
- Complex function space
- Overfitting

Pre-training can be unsupervised!



Practical Considerations

- There many frameworks avaialable that do most of the tedious work for you:
  - Tensorflow/Keras
  - Theano/Keras
  - SciKit Learn
  - PyTorch
  - …
- With various levels of abstraction
- And programming styles
- Most are GPU enabled
- I prefer PyTorch (for now)

We'll demostrate the practicalities in the lab and learn more in the exercises.

Further Reading

- A very accessible series of lectures:
  youtube video series
- Books:
  "Introduction to Statistical Learning"
  "The Elements of Statistical Learning"
  "Bayesian Reasoning and Machine Learning"

There is a lot more than we can cover in this course.