Machine Learning
Lecture 13

Lecture by Dr. Shai Fine
Typeset by Steven Karas

2018-01-21
Last edited 21:07:16 2018-01-21

Disclaimer These lecture notes are based on the lecture for the course Machine Learning, taught by Dr. Shai Fine at IDC Herzliyah in the fall semester of 2017/2018. Sections may be based on the lecture slides written by Dr. Shai Fine.

Agenda

- Neural Networks
- Deep Learning

1 History of ML

First commercial project was recognizing zip codes for the USPS back in the 1960's. Historically, the Perceptron[7] and Widrow-Hoff (ADALINE)[10] were the first practical learning functions. In 1969, Minsky and Papert[6] proved that a simple perceptron cannot learn non-linear functions such as XOR. Backpropagation was invented in 1986 by Rumelhart, Hinton, and Williams[8], which reopened AI/ML research. In 1995, Vapnik and Cortes presented the SVM[1]. In 2006, Hinton and Salakhutdinov presented deep learning[3], and proved that it is practical to pretrain such networks.

2 Perceptron

Perceptron provides a linear classification rule:

$$out = \begin{cases} 1 & \text{if } \sum_{i=0}^{d} w_i x_i > 0 \\ 0 & \text{else} \end{cases} = \frac{1}{2}\left[sign\left(\sum_{i=1}^{d} w_i x_i + w_0 \right) + 1 \right]$$

The goal is to minimize the classification error. Find the weight vector $w = [w_0, \ldots, w_d]^\intercal$ that minimizes the criterion:

$$E^{perc}(w) = -\sum_{i \in D_{miss}} w^\intercal (x^i y^i)$$

Rosenblatt's Learning Algorithm is *Stochastic Gradient Descent* with η learning rate and $\hat{y^i}$ output of the i-th instance:

$$w_j^{new} = w_j + \eta \Delta w_j = w_j + \eta(\hat{y^i} - y^i) x_j^i$$

2.1 Sigmoid Perceptron

Instead of using the classification error, we can replace the step function with the sigmoid function:

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

Which gives a probability as an output:

$$P(y_i \mid x_i, w) = \begin{cases} \pi_{i0} = 1 - \frac{1}{1+e^{-w^\intercal x}} & \text{if } y_i = 0 \\ \pi_{i1} = \frac{1}{1+e^{-w^\intercal x}} & \text{if } y_i = 1 \end{cases}$$

2.2 K-Class output

$$P(y = k \mid x, W) = \begin{cases} \frac{\exp(w_{k0} + w_k^\intercal x)}{1 + \sum_{l=0}^{K-1} \exp(w_{l0} + w_l^\intercal x)} & k = 1, \ldots, K-1 \\ \frac{1}{1 + \sum_{l=0}^{K-1} \exp(w_{l0} + w_l^\intercal x)} & k = K \end{cases}$$

Derived from the log-odds rule (also called logit) for $k = 1, \ldots, K-1$:

$$\log \frac{\Pr(y = k \mid x)}{\Pr(y = K \mid x)} = w_{k0} + w_k^\intercal x$$

With the classification rule:

$$\arg\max_k P(y = k \mid x, W)$$

2.3 ReLU Perceptron

TODO - effectively equivalent to the SVM

2.4 Limitations

Minsky and Papert showed in 1969[6] that Perceptrons cannot represent nonlinear functions.

3 Multi-Layer Perceptron (MLP)

Stack together perceptrons into multiple hidden layers, where the output of each layer is the input of the next.

The intuition is that the hidden layers can perform nonlinear transformations on the feature space, and then draw a linear decision boundary in the transformed space.

Given an adequate number of hidden units, many-layer networks can learn arbitrary decision boundaries. Those boundaries need not create linear or even convex regions.

A network has two modes: feedforward and learning. Feedforward mode simply gathers the outputs of the network given an input. Learning does both feedforward and then propagates the error backwards

Practical Advice MLPs allow parallel processing, and can have distributed computation and memory. They tend to be robust to noise and failures, but are sensitive to initialization and the scaling of the feature values. In practice, there isn't much difference between normalization and simple range scaling.

MLPs are useful for high-dimensional data such as raw sensor data. The output is very flexible, and can be discrete or continuous. However, the output is not necessarily humanly comprehensible.

3.1 Autoencoders

Recall that PCA[1] takes a subset of eigenvectors as the basis of a subspace to project the data onto, to minimize the reconstruction error. We proved the equivalence of the eigenvalues to the variance.

We can consider PCA in terms of neural networks, as a linear transformation. In this case, the weights should converge on the eigenvectors. This concept of encoding the feature space into itself is called *autoencoding*.

Using nonlinear transfer functions for PCA is called Kernel PCA, which is included in scikits. If we add nonlinear transforms between the layers, we can learn efficient nonlinear representations. This approach is called *deep autoencoding*.

3.2 Limits

Kolmogorov showed in 1957 that any continuous function on the unit n-dimensional hypercube can be represented by a neural network.

Hava Siegelmann and Eduardo Sontag showed in 1991[9] that neural networks are reducible to Turing Machines.

[1] That this is an unsupervised algorithm that does not take labels will appear on at least one of the exams

4 Deep Learning

Deep learning is not covered within this course, and will not appear on the exam. There are courses that cover deep learning specifically. Deep learning got it's biggest wins from image recognition and computer vision.

Generally speaking, deep learning is any neural network that has more than 1 hidden layer.

4.1 Convolutional Neural Networks (CNN)

First came out of NeoCognitron[2] in 1982. First modern form was LeNet[5] in 1998. Biggest jump came from ImageNet[4] in 2012.

A convolutional neural network is comprised of multiple layers of convolutions, pooling, and non-linearities. Recall that a convolution considers a local set of features, and reduces the dimensionality by the width of the kernel. Generally constructed from convolutional layers that detect local conjunctions of upper layer features and pooling layers that merge semantically related features together. The extracted features are invariant to location, rotation, or scaling.

Spatial Convolution Convolutions are local functions that consider a sliding window of adjacent pixels. There is an example of this on slide 35.

Traditional image recognition would handcraft such kernels as preprocessing steps. Deep learning is so powerful in part due to learning such kernels as part of the training process.

Practically, the initialization is less important, but the minima in the hypothesis space tend to be more or less the same, but different. Why this is so is an open problem, and indicates that we don't understand the computational model/power from a mathematical standpoint.

Pooling To describe a large image, we can summarize the features detected in the convolutional layers by taking either the average or the max. This is generally done for dimensionality reduction, to encourage local invariance to translation/transformation, and reduces our tendency to overfit on specific images.

An example of a CNN can be found on slides 37-38.

AlexNet AlexNet was constructed for the ImageNet competition in 2012, which was significantly larger than any other previous network: 60M parameters, 8 trainable layers.

Current Trends There is a trend towards smaller kernels, but much deeper networks, and less pooling. NVidia has pushed the field forwards much faster due to the computational capabilities of GPUs.

4.1.1 Transfer learning

Rather than train a full network from scratch, simply start from a pretrained model as the initialization, even if unrelated to the target domain.

Generic feature extractor Take the pretrained network, remove the last layer, and then treat the rest as a fixed feature extractor for another learning algorithm.

Fine-tuning Replace the last layer, and continue training for the new target domain.

Style Transfer There's an example on slide 46, but he did not describe how to do this in detail.

5 Course recap

The lectures covered theory and practice of machine learning.

We are expected to already know linear algebra and probability theory.

The taxonomy of statistical learning is very important, but Bayes formula is hands down the most important thing we learned.

- Known class conditional densities
  - Bayes Decision Theorem
- Unknown class conditional densities
  - Supervised Learning
    - Generative
    - Discriminative
  - Unsupervised Learning
    - Mixture Models
    - Cluster Analysis

The homeworks taught us how to practically apply machine learning to a toy dataset. Generally speaking, we followed the Cross Industry Standard Protocol for Data Mining (CRISP-DM) model.

1. Business understanding
2. Data understanding
3. Data preparation
4. Modeling
5. Evaluation
6. Deployment

Grading 50% homeworks. Each HW is worth 10%. The exam must be passed with a passing grade to pass the course.

Exam Example questions will be published. We will not be asked to prove anything we didn't prove in class, but we will have to prove some particularly important things.

References

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[2] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[3] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, UA Muller, Eduard Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

[6] Marvin Minsky and Seymour Papert. Perceptrons. 1969.

[7] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[8] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[9] Hava T Siegelmann and Eduardo D Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[10] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, STANFORD UNIV CA STANFORD ELECTRONICS LABS, 1960.