Università degli studi di Trento

Deep Learning

Prof. Ricci

Academic Year 2024/2025

Contents

Chapter 1

Brief History of Neural Networks

Electronic Brain (1943)

Try to compute a model that imitates our brain. Two major characteristics of this artificial neuron are: binary inputs and no concept of weights values (all inputs are equal important). It generates a binary output. They demonstrate that with this model is possible to model simple boolean functions such as OR, AND or NOT.

Perceptron (1958)

The model is able to learn the weights. Bias is added (θ_0 as extra weight and x_0 equal to 1). Perceptron is used as a machine for linear classification.
Perceptron training rule
Given a training set $T = \{(x_1, y_1), ..., (x_n, y_n)\}, \quad y_i \in \{0, 1\}$ and a learning rate $0 < \eta < 1$.
LOOP:

- Initialize weights θ_i and a bias θ_0 to small random values.
- Iterate until the entire training set is classified correctly:
  - Select random sample (x_i, y_i) from T.
  - Compute $\hat{y}_i = h(\theta^T x_i + \theta_0)$
  - Update weights and bias:
    $\theta^{t+1} = \theta^t + \eta(\hat{y}_i - y_i)x_i$
    $\theta_0^{t+1} = \theta_0^t + \eta(\hat{y}_i - y_i)$

Today an artificial neuron, i.e. a perceptron, is a non-linear parameterized function with restricted output range.

XOR Problem (1969)

In 1969 a problem emerged: perceptron cannot solve complex problems like XOR. In particular a perceptron cannot solve non linearly separable problems.

Densely connect artificial neurons to realize compositions of nonlinear functions. It can be used both for classification and regression tasks. The information is propagated from the inputs to the outputs. No cycles between outputs and inputs (Directed Acyclic Graph, DAG).

Computes one or more non-linear functions. Computation is carried out by composition of som number of algebraic functions implemented by the connections, weights and biases of the hidden and output layers. The model become:

$$\hat{y} = h(\sum_i \theta_i x_i + \theta_0) = h(\theta^T x + \theta_0)$$

$$z = \sum_i \theta_i x_i + \theta_0$$

In this case the perceptron algorithm is not applicable because it expects to know the desired target. For hidden layers we cannot know the desired target.

MLP Backpropagation (1986)

Backpropagation revitalized the field, showing that learning MLP for complicated functions is possible. Efficient algorithm to process "large" training sets. It allowed for complicated NN models to be trained. Today still at the core of NNs training.

1. Forward propagation: sum inputs, produce activations, feed-forward.
2. Error estimation: compare labels with predictions.
3. Back propagate: the error signals and used it to update weights.

In neural network training we have training samples $T = \{(x_1, y_1), ..., (x_n, y_n)\}$. Adjust all the weights of the network θ such that a cost function is minimized $min_\theta \sum_i L(y_i, f(x_i; \Theta))$. Use the BackPropagation of the error signal.

In 1997 were introduced Recurrent Long-Short Term Memory Networks. They allow us to deal with structured data like sequences.

In 1998 were introduced Convolutional Neural Networks and OCR was solved with LeNet.

In 9989 G.Cybenko publishes a paper with an important theorem, the Universal approximation theorem: "2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy."
In practice the more the number of hidden layers, the better. However, deeper NNs need a lot of labeled data and cannot be trained easily.

NN cannot exploit many layers, this lead to overfitting. Vanishing gradient (at training several small numbers are multiplied: weights do not get updated). Reasons are no GPUs and no large annotated datasets.

SVM

SVMs have similar accuracies than NNs but much fewer heuristics and parameters. Also, they have nice proofs on generalization.

Deep Neural Networks (2006)

Weights initialization: train each layer one by one with unsupervised training (using contrastive divergence). Much better than random values. Fine-tune weights with supervised learning.

Availability of large datasets and fast GPU implementations have made backpropagation from scratch almost unbeatable.

2012 AlexNet: CNN similar to LeNet. Trained on ImageNet with two GPUs. Technical improvements to avoid overfitting (ReLU, dropout, data augmentation).

Summary

- 1943: McCulloch and Pitts showed that neurons can be combined to construct a Turing machine (using ANDs, ORs and NOTs)
- 1958: Rosenblatt shows that perceptrons will converge if what they are trying to learn can be represented
- 1969: Minsky and Papert showed the limitations of perceptrons, killing research for a decade
- 1986: The backpropagation algorithm revitalizes the field
- 1998: Kernel methods become the dominant learning paradigm
- 2006: The Hinton lab “solves” the training problem for deep networks
- 2012: AlexNet and the deep learning revolution.

Chapter 2

Machine Learning Recap

In machine learning we combine data and results, and after a computation we have a program.

ML allows computers to acquire knowledge. This knowledge is acquired through algorithms by learning from the data. Knowledge is represented by a model.

ML is useful in tasks for which we have no human expertise or we cannot explain our expertise.

Models need to be personalized and need a huge amount of data.
We can distinguish between 3 types of learning:

- Supervised Learning: learn a function mapping to a categorical space given a training set: $f : \mathbb{R}^d \mapsto \{1, 2, ..., k\}$. The output from model can be a binary or a multiclass classification. In the case when the problem is a regression we have a training set and we want to learn a function to a continuous space: $f : \mathbb{R}^d \mapsto \mathbb{R}$. This give us a real value output.
  Structured Output Prediction Problem: the output is a vector (or other data structure containing multiple values) with important relationships between the elements. There can be a very complex relationship between input and output. Sometimes there may be many possible valid answers. But the output obeys rules (for example, language obeys grammatical rules).
  NNs are very powerful for SO prediction. Applications can be image captioning, machine translation, semantic segmentation.

- Semi-supervised Learning: is given a training data plus a few desired outputs (labels).

- Unsupervised Learning: are given training data without desired outputs. In clustering, given the training set, the output is the hidden structure behind data (e.g. clusters). Deep Clustering methods aare recently developed. In Density Estimation the objectiv is to learn the underlying data distribution Δ. Clustering is useful for anomaly detection, identifying an outlier of a cluster.
  Generative models and Probabilistic generative models are part of Unsupervised Learning models.
  In Unsupervised Learning is important the concept of dimensionality reduc-

tion: given a training set i want to map data x in a lower dimensional space $\mathbb{R}^k$ with $k << d$. It is done by Autoencoders.

- Reinforcement Learning: a set of states, actions and rewards. The Goal is take actions to change the state so that you receive rewards. You don't receive any data but you have to explore the environment yourself to gather data as you go.
  The environment is composed by from states, actions and rewards (positive and/or negative)

How to design a learning system

1. Choose the training experience
2. Choose the model (or target function)
3. Choose how to represent the target function
4. Choose a learning algorithm to infer the target function from the experience
5. Define an evaluation metric

There are thousands of machine learning algorithms. They all have 3 components:

- Function representation: first option are instance-based functions like K-nearest neighbors, kernel regression and local regression. Second option are numerical function like linear regression, neural networks, support vector machines. Or there can be parametric functions like naive Bayes and Bayesian networks, Hidden-Markov models, conditional random fields (CRFs). There are also symbolic functions like decision trees, rules in propositional logic or in first-order logic.
  Discriminative models focus on the decision boundary, are more powerful than generative with lots of examples. They are not designed for incorporating unsupervised data.
  Generative models are probabilistic models that compute decision boundary. They use unsupervised data.

- Optimization: the success of a machine learning system also depends on the search and optimization algorithms. These algorithms are crucial to find the optimal model.

- Evaluation: difference in the supervised and unsupervised setting. Not always easy to decide which metric. Same algorithms may be used for different metrics. Ideally loss function and evaluation metric should be the same.

Assumptions in Machine Learning

The challenge is that the algorithm must perform well on new, previously unseen inputs. The ability to perform well on previously unobserved inputs is called generalization.

When training a model (with optimization) we reduce the training error. What distinguishes ML from optimization is that we want the generalization error (the test error) to be low as well.
Statistical Learning Theory: we assume that training and test examples are independently drawn from the same data distribution. Example and label pairs (x, y) are drawn from an unknown distribution $p(x, y)$. Data are i.i.d: Same (unknown) distribution for all (x, y) in both training and test data. Empirical loss is measured on training set: $L(\theta, X, y) = \frac{1}{N}\sum_{i=1}^{N} L(f(x_i;\theta), y_i)$. The expected loss is also known as risk: $R(\theta) = \mathbb{E}_{(x_0,y_0)\sim p(x,y)}[L(f(x_0;\theta), y_0)]$.
If the training set is a representative of the underlying (unknown) distribution $p(x, y)$ the empirical loss is a proxy for the risk.

The factors determining how well a machine learning algorithm will perform are its ability to make the training error small and make the gap between training and test error small.
In case of underfitting the model is not able to obtain a sufficiently low error value on the training set.
In case of overfitting the gap between the training error and test error is too large.
To control overfitting we can done:

- Reduce model capacity: capacity is the ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that are not useful on the test set. We control the capacity by choosing the hypothesis space, the set of functions that the learning algorithm is allowed to select as being the solution.

- No Free Lunch Theorem: our model is a simplification of the reality. Simplifications are based on assumptions (model bias). Assumptions fail in some situations. There are no universally good ML algorithms. If we make assumptions about the kinds of probability distributions we encounter, then we can design learning algorithms that perform well on these distributions.

- Regularization: any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. Many regularizer are possible. Typically a function on the model's parameters. NFL for regularization.

How to choose the algorithm

Most machine learning algorithms have hyperparameters, settings that we can use to control the algorithm's behavior (e.g. for controlling regularization). Need a validation set (held-out) from training data.

In Cross-Validation the strategy is to partitioning a dataset into subsets such that the training is initially performed on some subsets, while the remaining subset is retained for validation.