MTAT.03.227 Machine Learning

Feed-forward neural networks

Sven Laur
University of Tartu
Ways to view neural networks

Neural network as a tool to model brain activity
▷ Various spike models. Large scale simulations. Blue Brain project

Neural network as a way to repackage and sell old statistical methods
▷ Large enough neural networks solve every problem!

Neural network as a nature inspired tool for solving machine learning tasks
▷ Sometimes brain is statistically sensible ...
Feed-forward networks

Feed-forward network have no information propagation cycles

- Information flows from one layer to the other layer
- For a fixed input they always give the same output
- Internal configuration of neurons completely determine the output
Recurrent networks

Recurrent network has many information propagation cycles.

- Its state is iteratively or stochastically updated
- For a fixed input there can be many different outputs
- The initial state plays large role besides the network configuration
Description of single neuron

Let $\mathbf{x} = (x_0, \ldots, x_k)$ be the inputs and $y$ be the output of the neuron.

- First all inputs are linearly aggregated: $a = w_0 x_0 + w_1 x_1 + \cdots + w_k x_k$.
- Then the output $y$ is obtained by computing: $y = f(a)$. 
Implementation details and terminology

Input $x_0$ is always set to 1
The weight $w_0$ is referred as a bias term.
Vector $w = (w_0, \ldots, w_k)$ is referred as a weight vector.
The function $f$ is commonly known as an activation function.
Example

Compute the output value of the neural network if

- The neuron \( \mathcal{N}_1 \) has a weight vector \( \mathbf{w}_1 = (0, 1, 3) \) and \( f_1(a) = \tanh(a) \)
- The neuron \( \mathcal{N}_2 \) has a weight vector \( \mathbf{w}_2 = (2, 3, 1) \) and \( f_2(a) = \tanh(2a) \)
- The neuron \( \mathcal{N}_3 \) has a weight vector \( \mathbf{w}_3 = (1, 1, 1) \) and \( f_3(a) = a \)
How to train a feed-forward neural network?
Standard problem statement

Network configuration

- The set of neurons $\mathcal{N}_i$ together with activation functions $f_i$.
- The set of wires between the neurons.

Standard training task

- Find the weight vectors $w_1, \ldots, w_\ell$ for each neuron such that the empirical risk on the training set (training error) is minimal.
Neural networks with a single neuron
Regression with a single linear neuron

If the activation function is linear then we can simplify the mapping

\[ y = w_0 + w_1 x_1 + \ldots + w_k x_k \]

If we use standard mean square error as an optimisation target, we get

\[ \sum_{i=1}^{N} (w^t x_i - y_i)^2 \rightarrow \min \]

which is equivalent to the standard linear regression task.
 Classification with a single neuron

A linear neuron combined with mean square error does not lead to a good classification algorithm as potential error is unbounded

Quick fix. Lets clip the linear function into the range \([0, 1]\) by defining

\[
    f(a) = \begin{cases} 
    0, & \text{if } a < 0, \\
    a, & \text{if } 0 \leq a \leq 1, \\
    1, & \text{if } a > 1
    \end{cases}
\]

Problem. Clipping introduces two angle points – problems with gradients!
Sigmoid function provides a smooth clipping with bounded derivative:

$$f(a) = \frac{1}{1 + \exp(-a)}$$

The function is a natural choice as it also models common saturation effects.
Minimal relative entropy as a target

Mean square error is \textit{ad hoc} optimisation target for the classification task. Relative entropy

\[
RE = -\frac{1}{N} \sum_{i=1}^{N} \ln\left( f(x_i)^{y_i}(1 - f(x_i))^{1-y_i} \right)
\]

is theoretically much more justified. This cost function combined with a sigmoid activation function is known as \textit{logistic regression}.

\textbf{Properties}

- Logistic regression has no closed form solution
- Stochastic gradient decent is commonly used to solve this problem.
Limitations of single neuron networks
A decision border of a single sigmoid neuron is a straight line.

No straight line can separate red dots from blue dots.
A neuron with a sinusoidal activation function \( y = \sin(wx) \) can track any signal in finite number of training points.

- It is impossible to train such a neuron
- Predictions outside the training set are arbitrary
Neural networks with a single hidden layer
Polynomial regression and classification

Polynomial regression can be represented as a simple neural network with dedicated neurons that compute polynomial powers.

- For regression the features are combined with a linear neuron.
- For classification the features are combined with a sigmoid neuron.
Stepwise approximation of functions

By using a single neuron with activation function $f(a) = \text{sign}(a)$ in the hidden layer, we can approximate the original function with two steps.

- Bias term in the hidden neuron determines the location of the jump.
- Weights in the output layer determine the heights of the steps.
Stepwise approximation of functions

By adding the number to threshold neurone we can get arbitrary precision.

- Good approximation requires many neurons in the hidden layer.
- Approximation has jumps and is equivalent to decision tree learning.
Smooth approximation with sigmoid units

The usage of sigmoid neurons in the hidden layer smooths the signal.
▶ It is possible to track both linear segments and rough jumps.
▶ Approximation is still built from stepwise jumps.
Smooth approximation with radial basis functions

\[ F(x) = \exp \left( -\frac{\|x - \mu\|^2}{2\sigma} \right) \]

Functions can be approximated with radial basis units

- The location of the peak is determined by the *centre* \( \mu \)
- The shape of the peak is determined by the *width* \( \sigma \)
Numerical instability and regularisation
Linear regression in the concept space

A small error in a point with big leverage can make linear regression function arbitrary large, which can lead to large test errors.

▷ In many cases we know that the final output must be in fixed range.
Two common sanity checks

If the bound \( f(x) \in [-1, 1] \) holds for rectangular area \([-1, 1] \times \ldots \times [-1, 1]\) then we should solve the following task instead:

\[
\frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2 \rightarrow \min
\]

s.t. \( |w_1| + \cdots + |w_k| \leq 1 \)

If the bound holds for unit circle \( x_1^2 + \cdots + x_k^2 \leq q \) then we should solve the following task instead:

\[
\frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2 \rightarrow \min
\]

s.t. \( w_1^2 + \cdots + w_k^2 \leq 1 \)
Lagrange’ trick

If we want to minimise \( f(x) \) such that \( g(x) \leq c \) for a non-negative function \( g(\cdot) \), then there exists \( \lambda \geq 0 \) such that the solution of the original problem is a minimum for a modified function

\[
f_*(x) = f(x) + \lambda g(x)
\]

Consequences

▷ We can use a penalty term \( \lambda \| w \|_1 \) for rectangular area
▷ We can use a penalty term \( \lambda \| w \|_2^2 \) for circular area