AI Questions

Massimo Perego

Contents

1 Neural Networks

1.1 Multi Layer Perceptrons MLP

1.1.1 Definition

An r-layered perceptron is a feed-forward neural network with a strictly layered structure and an acyclic graph.

Each layer receives input from the preceding one and passes its output to the subsequent layer, jumps between non-consecutive layers are not allowed. Usually, each neuron is fully connected to the neurons in the preceding layer.

They are "feed-forward", information flows in one direction only, there can't be cycles. They can have significant processing capabilities, since they can be increased by increasing the number of neurons and/or layers.

There are different types of layers (and thus neurons):

- Input: receive inputs from the external world
- Hidden: one or more layers that perform transformations on the inputs
- Output: produces the final outputs of the network

Each neuron receives multiple inputs, either from the previous layer or from the external world, each one with an associated weight. The network input function combines these input, usually with a weighted sum, and thus determines the global solicitation received by the neuron.

The activation function determines the neuron's activation status based on its inputs. It is a so-called sigmoid function, a monotonically non-decreasing function with a range [0,1] (or [-1,1] for bipolar sigmoid functions). It (usually) introduces non-linearity, giving the neuron its processing capabilities and allowing the network to learn complex patterns (if they were linear we could merge all functions into one).

The output function produces the final output of the neuron based on its activation status and the output is then passed to the next layer. It is often the identity function. It might be useful to scale the output to a desired range (linear function).

1.1.2 Function Approximation

Any Riemann-integrable function can be approximated with arbitrary accuracy by a four-layer multi-layer perceptron. We approximate the function into a step function and construct a neural network that computes said function.

The input is taken by a single input neuron, and the first hidden layer is composed of a neuron for each of the step borders of our approximated function. Each neuron determines on which side of the step border an input lies.
In the second hidden layer there's a neuron for each step, that receives input from the two neurons that refer to the values marking the border of the step. The weights and threshold are chosen is such a way that neurons in the second layer are active only if the function value is inside the step, i.e., only one of the neurons in the preceding layers is active. Only a single neuron in the second layer can be active at a time.
The output layer has the identity function as activation and receives only the value of the step as input.

The accuracy can be increased arbitrarily by increasing the number of steps.

We can remove a layer and simplify the approach by considering the variation of the function value between each step. There is a single hidden layer and outputs from this layer are weighted with the relative difference between steps.

1.1.3 Regression

Training a NN is closely related to regression, the statistical technique for finding a function that best approximates the relationship in a data set; both regression and MLP training involve minimizing an error function, usually the mean square error. We need to adapt weights and parameters of the activation function to minimize said error.

Types of regression:

- Linear: When a linear relationship between quantities is expected;
- Polynomial Regression: Extends linear regression to polynomial functions of arbitrary order;
- Multi-linear Regression: Used to fit functions with multiple arguments;
- Logistic Regression: Particularly relevant to ANNs because many such networks use a logistic function as their activation function. If we can transform a function to a linear/polynomial case we can determine weights and thresholds for the system, for a logistic function this can be done by

way of the Logit transformation, allowing a single neuron to compute the logistic regression function.

1.1.4 Training

With the term "training", for an MLP, we're talking about minimizing the error function on the data set given for the training.

Gradient Descent: We can derive from the error function a direction in which to change the weights and thresholds to minimize the error.
The gradient is a vector in the direction of the steepest increase of the function. We make small steps (size determined by a learning rate) in the direction indicated by the gradient on the error function until convergence, i.e., a local optima of the error function is found.
This requires having a differentiable activation and output function.
The algorithm will essentially be:

- Compute the gradient
- Small step in the opposite direction of the gradient
- Repeat until convergence

Some variants have been developed to address the challenge of learning rate selection and overcoming local optima:

- Random restart: train the network multiple times, with different starting points
- Momentum: Adds a fraction of the previous weight change to the current step
- Manhattan Training: Uses only the sign of the gradient to determine the direction of the step, simplifying computation
- Adaptive Learning Rates: Adapt the learning rate for each parameter based on the history of gradients

Error Backpropagation: Only the output neurons are connected to the error, but we need to train the whole network.
The error values of any (hidden) layer of a multi layer perceptron can be computed from the error values of is successor layer. The error is computed at the end of the network and then backpropagated through the whole network.
General structure of the algorithm:

1. Setting and forward propagation of the input
2. Calculate the error and adapt the weights for the last layer

3. Error backpropagation, the "new" error factor is computed starting from the error of the subsequent layer, and this is done layer by layer

This allows to calculate how much each neuron "contributes" towards the final error.

The weight adaptation depends on a learning rate, which has to be initialized properly in order to not "jump" over the minimum without ever converging (i.e., it's too high).

The error can't completely vanish due to the properties of the logistic function, there will always be some residual errors due to the computation of the various parameters.

1.1.5 Deep Learning