# Understanding Long Short-Term Memory Architecture

Compiled by D. Gueorguiev 2/20/2024

## Abbreviations

NN – Neural network

BP – Error Back-propagation algorithm

BPTT – Back-propagation through time

LSTM – Long Short-Term Memory

## Notation

– learning rate of a network

– time unit

- initial time of an epoch

– final time of an epoch

– the set of units in the network

– (generic) units in the network;

- the set of input units, – input unit;

- the set of output units, – output unit;

– the set of non-input units, – generic or non-input unit;

- the output of unit ; the output of a unit is scalar

– the set of units with connections to a unit ; i.e. its predecessors

– the set of units with connections from a unit ; i.e. its successors

- the weight that connects the unit to unit

- the input from a unit coming from a unit

– the weighted input of the unit

- the bias of the unit

- the state of the unit

– the squashing function of unit

-the error of unit

-the error signal of unit

– the output sensitivity of the unit with respect to the weight

## Perceptron and the Delta Learning Rule

**Definition** Perceptron

Given the input vector and trained weights , the perceptron outputs , which is computed as

We refer to as the *weighted input* and to as the perceptron’s *state*. For perceptron to fire, its state must exceed the value of the threshold.

In cases of misclassification the Perceptron modifies weights accordingly. The perceptron will converge to reproduce the correct behavior provided that the training examples are linearly separable. Convergence is not assured if the training data is not linearly separable (Note: the proof is in the Marvin Minsky’s book on Perceptrons).

A variety of training algorithms for the Perceptron exist of which the most common ones are the *Perceptron learning rule* and the *Delta learning rule*. Both start with random weights and both guarantee convergence to an acceptable hypothesis.

Using the *Perceptron learning rule* algorithm the Perceptron learns from a set of samples where a sample is a pair where is the input and is its label. For the sample , given the input , the old weight vector is updated to the new vector using the rule

with ,

where is the output calculated using the input and the weights and is the *learning rate*. The *learning rate* is a constant that controls the degree to which the weights are changed. The initial weight vector has random values. The algorithm will only converge towards an optimum if the training data is linearly separable, and the learning rate is sufficiently small. The perceptron rule fails if the training examples are not linearly separable.

The modification of weights is achieved by using the gradient optimization descent algorithm, which alters them in the direction that produces the steepest descent along the error surface toward the global minimum error.

**Definition** The Sigmoid Threshold Unit

The output is computed by with

where is the bias and is positive constant that determines the steepness of the sigmoid function. The function is also known as the squashing function. The advantage of the neural networks using sigmoid units is that they are capable of representing non-linear functions.

## Feed-forward Neural Networks and Backpropagation

**Definition** *Feed-forward neural networks* (FFNN)

In FFNNs sets of neurons are organized in layers, where each neuron computes a weighted sum of its inputs. Input neurons take signals from the environment, and output neurons present signals to the environment. Neurons that are not directly connected to the environment, but which are connected to other layers are called *hidden neurons*.

Feed-forward neural networks are loop-free and fully connected. Feed-forward neural networks are loop-free and fully connected. This means that each neuron provides an input to each neuron in the following layer, and that none of the weights give an input to a neuron in a previous layer.

A diagram of a network

Description automatically generatedFigure: A multi-layer feed-forward neural network with one input layer, two hidden layers and an output layer. Using neurons with sigmoid threshold functions, these neural networks are able to express non-linear decision surfaces.

### The Error Backpropagation Algorithm (BP)

BP is a NN learning technique which uses gradient descent to learn the weights in multi-layer NNs. It works in

Small iterative steps, starting backwards from the output layer towards the input layer. A requirement is that the activation function of the neuron is differentiable. The weights of the FFNN are initialized in certain way. Then error backpropagation applies the training samples to the network and computes the input and output of each unit for all layers (*input*, *hidden* and *output*).

The set of units in the network is where denotes the disjoint union and are the sets of input, hidden and output units respectively. We denote the input units by , hidden units by , and output units by . We define the set of non-input units . For a non-input unit , the input to is denoted by , its bias by and its output by . Given units , the weight that connects with is denoted with .

To model the external input that the neural network receives, we use the *external input vector* . Clearly, we have .

For the non-input unit , the output of , denoted with , is defined using the sigmoid activation function as:

(1)

where is the state of , defined as

(2)

where is the bias of , and is the weighted input of , defined in turn by

, with

(3)

where is the information that passes as input to , and is the set of units that precede ; the set are input and hidden units that feed their outputs to the unit .

Starting from the input layer, the inputs are propagated forwards through the network until the output units are reached at the output layer. Thus the network output is constructed and for , its output corresponds to the component of .

Next, backpropagation takes place – the error is propagated backwards and the weights and biases are updated in such way that the error with respect to the current training sample has been reduced. Starting from the output layer, the algorithm compares the network output with the corresponding desired target output . The error is and the overall network error is given with

To update the weight , we will use the formula

where is the learning rate. We use the factors and to calculate the weight update by deriving the error with respect to the activation, and the activation in terms of the state, and in turn the derivative of the state with respect to the weight:

The derivative of the error with respect to the activation for output units is

Now , the derivative of the activation with respect to the state for output units is

**\***

**\*** Clearly, with one can easily show that . This is important relation because in the backpropagation step when we compute we do not need to compute anything additionally to what we have computed on the forward pass.

The derivative of the state with respect to the weight the connects hidden unit to an output unit is:

For an output unit o we define the error signal of o by:

(4)

Thus for output units we have:

(5)

Thus the weight between the hidden unit and the output unit becomes

Similar error analysis can be done for the hidden unit : the notion of error in the hidden unit is related to how much it contributed to the production of a faulty output. We backpropagate the error from the output units that sends signals to. Recall, for an input unit we have derived – we expand it to:

with .

where is the set of units that . From the last expression for we obtain

If we define the error signal of the hidden unit by

with ,

then we have a uniform expression for weight change; that is,

With the successive training iterations the error of the network given with will continuously decrease and at certain point a termination condition will be reached.

## Recurrent Neural Networks

### RNN Architectures

## Literature

[Understanding Long Short-Term Memory Recurrent Neural Networks – a tutorial-like introduction, Ralf C. Staudemeyer, Eric R. Morris, 2019](https://github.com/dimitarpg13/transformers_intro/blob/main/articles_and_books/TutorialOnLongShortTermMemory2019.pdf)