Introduction to Transformers

D. Gueorguiev 6/24/22

Table of Contents

[Preliminaries 1](#_Toc107168028)

[Language Modeling 1](#_Toc107168029)

[Sequential Modeling 1](#_Toc107168030)

[Notes on RNN (Recurrent Neural Networks) 2](#_Toc107168031)

[Notes on LSTM (Long Short Term Memory) 4](#_Toc107168032)

[Notes on Gated Recurrent Neural Nets 5](#_Toc107168033)

[Latent Semantic Analysis (LSA) 5](#_Toc107168034)

[The Attention Mechanism 5](#_Toc107168035)

[Works Cited 5](#_Toc107168036)

# Preliminaries

*Note 0*: Knowledge of feed-forward neural networks and the back-propagation algorithm is assumed throughout this document.

It is important to understand how the *Transformer architecture* fits historically as a modeling tool. For the purpose first we will look into other architectures and models which evolved earlier compared to the *Attention mechanism* and the *Transformer architecture*.

## Language Modeling

Tools for Language Modeling:

* Sequential Models
* Probabilistic Models e.g. *Latent Semantic Analysis* (LSA)

Besides used in Language Modeling the *Sequential models* are widely used in the modeling and prediction of Time-Series, Images, and Voice. Thus, we will start with a discussion on various architectures for Sequential Models which precede the *Transformer architecture*.

## Sequential Modeling

In general, we build ML models where we have data points which are usually uncorrelated, and no order relation can be imposed on them.

*Sequential models*: In many cases such as language, voice, and time-series data a data point is dependent on a set of other data points which have already been processed. We call such stream of data points a stream of *sequential data*. Machine learning models which accept input or create an output sequence of data points are known as *sequential models*.

Previous tools for sequence modeling:

* Recurrent Neural Networks
* Long Short Term Memory
* Gated Recurrent Neural Nets

### Notes on RNN (Recurrent Neural Networks)

What is RNN?

RNN is a Neural network with at least one cycle. If the network contains cycle the computation is not uniquely defined by the interconnection pattern and the *temporal dimension* must be considered. When the output of one neural node is fed back to the same node we are dealing with recursive computation. We must define what we expect from the network – is the fixed point of the recursive evaluation the desired result or one of the intermediate computations? We can assume that every computation takes a fixed amount of time and can be expressed as a certain number of time units. If the inputs of a neural node have been sent at time , its output will be available at time . A recursive computation can be stopped after predetermined number of steps and the last output will be considered the result of the recursive computation.

Figure 1: Node with a loop

Continuous vs Epochwise mode

Two approaches to operate and train an RNN:

* *Epochwise operation*

The network is run from a given start time until a given stopping time is reached. After reaching the stopping time the network is reset in its *initial state* for the next epoch. It is not essential that all of *the state* at the beginning of each epoch is the same. The important aspect of the *Epochwise operation* is the starting state of a new Epoch is not causally related to the ending state of the previous epoch.

Thus, every epoch serves as a boundary through which learning credit cannot pass. The purpose of the epoch boundaries is to make sure that activity from one epoch is causally related to activity in another later epoch. Note that the notion of epoch is defined in a loose sense indicating only that the boundaries are present between an interval(s) and an interval of past activity is separated by boundary from the activities after the boundary.

This allows us to introduce the notion of *batch training* distinguished from the notion of *incremental training*. The difference between the batch training vs incremental training is in when the network weights are updated. With the batch training approach weights are updated only after presenting a complete set of training examples. With the incremental approach the weights are updated after presenting each training example.

* *Continuous operation*

No manual state resets nor any barriers against the flow of training credit are imposed in the network.

Continuous operation makes sense when online learning is required.

Backpropagation through time (BPTT)

This paragraph follows the notation introduced by [Rojas](https://github.com/dimitarpg13/transformers_intro/blob/main/articles_and_books/neuron_Rojas.pdf) and borrows some of the diagrams introduced in his work.

For simplicity, let us consider a finite number of iterations only. Assume that a network of computing nodes is fully connected and that is the weight associated with the edge from node to node . We can unfold the network at times , , , transforming the original RNN into a feed-forward network with stages of computation. At each discrete time an external input is fed into the network and the outputs of all computing nodes are recorded. We will denote vector of all network outputs at time with . We assume that i.e. all network outputs at the initial moment are zeros. The unfolded network is depicted below:

. . .

. . .

. . .

. . .

. . .

. . .

. . .

. . .

. . .

. . .

. . .

Figure 2: Backpropagation through time

This unfolding strategy which converts the RNN into a feed-forward network in order to apply the classic back-propagation algorithm is called *Back-Propagation Through Time* (BPTT).

Let us denote by the matrix with the network weights . Let us denote by the matrix of interconnections between input sites and units. The feed-forward step is computed in the usual manner with a feed-forward network. At time we feed the transformed network with the dimensional external input . At each discrete time there are given the network state (-dimensional row vector) and the vector of derivatives of the activation function at each node:

(1)

In (1) we have:

(2)

where is the input to the activation function of the -th node at time .

Recall, we need the derivative of the activation function at each node in order to compute the back-propagated error when we are doing the back-propagation step. Refer to the figures below:

*- activation function*

*+*

backpropagation

Figure 3: Result of the backpropagation step

. . .

. . .

hidden node

input site

Backpropagated error to the -th hidden node

backpropagated

error

backpropagation to input site

Figure 4: Backpropagation from extended output layer through a hidden node to input site

The error of the network depicted on Figure 2 can be measured after each discrete time moment if a sequence of values is to be produced or just after the final moment of time if only the final output is of importance. The error between the -dimensional target and the output of the network is given with :

(3)

which is -dimensional column vector.

Thing to consider: each weight of the network is present at each stage of the unfolded network (Figure 2).

**Theorem**: Any network with repeated weights can be transformed into a network with unique weights.

Let us consider an unfolded feed-forward network with structure as the one shown in Figure 5.

forward step

Figure 5: A duplicated weight in a network

\*

\*

backprop step

Figure 6: Transformation guaranteeing unique weights

### Notes on LSTM (Long Short Term Memory)

### Notes on Gated Recurrent Neural Nets

## Latent Semantic Analysis (LSA)

LSA belongs to the family of *Probabilistic Methods* for analyzing and modeling the structure of the language.

# The Attention Mechanism

# Works Cited

Rojas, R. (1996). *Neural Networks Systematic Introduction.* Berlin: Springer Verlag.