Forget hidden states?

Kacper Gibas, Sam Kierans and Billy Yan

2020

Contents

Chapter 1

Why hide?

A basic RNN [1] achieves memory by receiving 2 sets of inputs;new information input and previous hidden state(context units) and returning 2 sets of outputs the output and the new hidden state.
LSTM [2] perhaps the most successful type of RNN, uses hidden states and gates to retain information for arbitrary timesteps. This allows LSTMs ,when combined with other techniques such as transformers [3] become state of the art in practically all areas of a.i. research where it is applicable. Is a hidden state essential for retaining information for many timesteps?

Biological neural networks (brains) don't seem to use one; they probably don't send of its information to a separate entity in order to wipe or reset itself, I think we can all agree that the brain doesn't get wiped for new input. If the brain is able to work by keeping all the information in the brain, why don't we try do the same thing with artificial neural networks?

What we aim to do is retain information for many timesteps without relying on hidden states.

We choose to use the simplest method for getting the information from past timesteps: don't use a blank version of the neural network every timestep and just do all the calculations on top of the value the neurons already has. Information from the previous timestep for free! i.e. for a particular neuron
$n_{at} = f(n_{at-1} + \sum_{i=1}^{n} inputs_i \odot weights_{ai} + bias_a)$
$f(x)$ is the activation function,
n_{at-1} is the value of the neuron n_a from the previous timestep, n_{at} is the new value of n_a for this timestep,

$\sum_{i=1}^{n} inputs_i \odot weights_{ai}i$ is the sum of the elements of the Hadamard product of input to n_a: $inputs_i$ and weights connected to n_a: $weights_{ai}$,
and $bias_a$ is the bias of the neuron n_a.

The only difference between this and "plain vanilla" simple neural network is that we add a n_{at-1} to the input of the activation function. This shouldn't be too much extra work in practice, the value of n_{at-1} would be in the tensor/array/matrix/vector/list of neurons if you don't clear/deconstruct/delete it

(which might even save some time).

Can a network that works this way hold information for arbitrary timesteps? let's assume e = $\pi = 3$ (insert other ridiculous and covenient approximation/assumption of choice), and let the speculating/"gedankenexperiment" begin.

One way to try and prove that it is possible for this type of network to hold information for arbitrary timesteps is to show for a neuron n_a, n_{at} can be equal to n_{at-1} . If n_{at} can be equal to n_{at-1} for some possible criteria for any timestep; it is possible for arbitrarily many timesteps to fit that criteria. Therefore information can theoretically stay in the network indefinitely i.e. holding information for arbitrary timesteps.

First we ignore $inputs_i$, $weights_{ai}$ and $bias_a$ considering just the activation function,we need to at least show: $n_{at} = f(n_{at-1})$ is possible,
sadly we are forced to use non-linear activation functions in most neural networks, or else the entire network becomes a linear regression model; a useful tool but usually not what most recurrent neural networks are trying to model.

A non-linear activation function would in theory change the input at least bit by bit through every timestep becuase of its non-linearity until it becomes useless as a memory. So back to the drawing board?

No, we are saved by the saviour of deep learning: reLU [4] !!!, for $n_{at} = \text{reLU}(n_{at-1})$ to be true we just need $n_{at-1} \geq 0$ becuase reLU is linear and has a gradient of 1 as long as $n_{at-1} \geq 0$.

Next, we need the conditions for $\sum_{i=1}^{n} inputs_i \odot weights_{ai} + bias_a = 0$,
because $n_{at} = f(n_{at-1} + \sum_{i=1}^{n} inputs_i \odot weights_{ai} + bias_a)$, would be then equal to $n_{at} = f(n_{at-1})$ which we know can be true.

So lets assume $bias_a = 0$, then we need the conditions for $\sum_{i=1}^{n} inputs_i \odot weights_{ai}$ to equal to zero, that can be achieved if either $inputs_i$ or $weights_{ai}$ are filled with 0, if all elements of the weight vector/tensor/matrix are 0 then that neuron recieves no useful input.

If the input vector/tensor is filled with zeros instead for a network made of fully-connected layers that would mean no new information gets introduced to any neurons from that layer onwards;

We can get around this by assuming only some input elements are 0, and the weight elements that correspond to the non 0 input elements equal to zero
e.g. if $inputs = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ and $weights_a = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$
the sum of the Hadamard product of the 2 vectors/tensors is 0 even though not all elements are 0, because the weights that correspond to the non-zero inputs are 0. In other words we need sparse connections.

Going over the conditions needed, we need reLU, $bias_a$, sparse connections and elements of input vector/tensor equal to 0, or more relaxed criteria of reLU/some variation of reLU, $bias_a$ close to 0, sparse connections and some elements of input vector/tensor close to 0 for senarios where the information isn't needed indefinitely.
Looking at these conditions storing information for many timesteps may not be impossible for this type of neural network.

Alas we are still dreaming about models of "spherical cows in a vacuum". There is still much more to do.

Firstly, will the network learn to keep the inputs to a neuron at 0 if it needs to that information some time later? (probably?, but we don't know how to prove it with any level of rigour)

Secondly, will the network learn to keep bias close to 0 to preserve information? (probably?, but we don't know how to prove it with any level of rigour)

Thirdly, will the neural network learn to work the way we want it to in the first place? (absolutely no idea, but "wE lEaVe It AS An ExErcISe fOR THe ReaDeR")

Chapter 2

"fastlane"

So we decided to try and design a new variant of RNN, that can fit the criteria and has a few other features. We are going to refer to these as "fastlane networks". We ditched a layered structure, instead we have an order for the neurons to "fire" in and every neuron can have connections with any other neuron(no connections to itself, no point in doing so).

to be continued...

Chapter 3

citations/references

[1] Elman, Jeffrey L. (1990). "Finding Structure in Time". https://crl.ucsd.edu/~elman/Papers/fsit.pdf
[2] Hochreiter, Sepp; Schmidhuber, Jürgen (1997-11-01). "Long Short-Term Memory". Neural Computation. https://www.bioinf.jku.at/publications/older/2604.pdf
[3] Polosukhin, Illia; Kaiser, Lukasz; Gomez, Aidan N.; Jones, Llion; Uszkoreit, Jakob; Parmar, Niki; Shazeer, Noam; Vaswani, Ashish (2017-06-12). "Attention Is All You Need" https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
[4] Xavier Glorot, Antoine Bordes and Yoshua Bengio (2011). Deep sparse rectifier neural networks http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf