LSTM (Long short-term memory)

1 Why we need LSTM

1.1 Recurrent Neural Network

Recurrent Neural Networks (RNN) were used mainly due to their ability to remember past states. RNNs are popular because of their ability to repeatedly use the same weights. RNN takes the sequence of inputs instead of just one input. RNN has the ability to process the inputs of any length. There is a symmetry in how inputs are being processed using RNN because of the usage of same wights at each time steps. RNN also uses the same activation function at each time step.

The basic working of RNN is shown in figure 1 [2]. In the forward pass, input is given at each timestep to the hidden state. Hidden state processes the inputs and predicts the estimated output $\hat{y}$. The loss function is calculated at each time step time by comparing the estimated output and the actual output. The total loss J(θ) is calculated by taking the average of loss calculated at each time step. RNN networks are then trained using back propagation algorithm to minimize the loss. Training of RNN is not straight forward task. RNN are trained by backpropagating through each layer and time step.

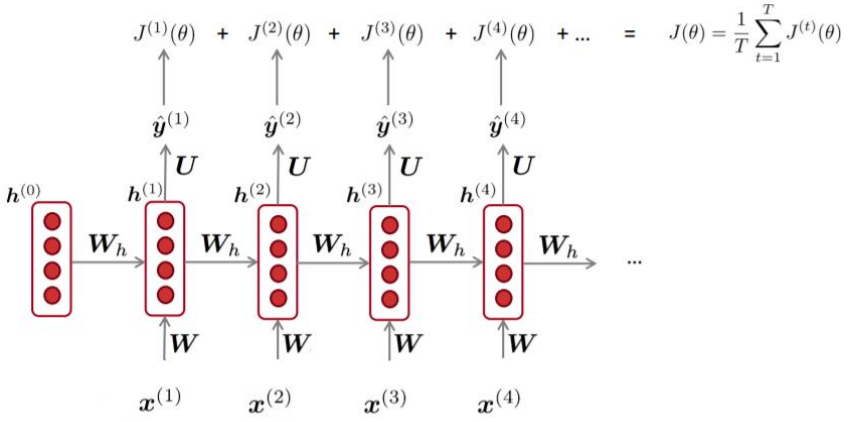

Fig 1: Block Diagram of RNN

In general Loss function $J^{(t)}(\theta)$ with respect to weight matrix W_h at time step (t) can be calculated by adding all the loss functions calculated at each time step using the formula in the equation (1). This is method is called backpropagation over time.

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h} \quad at\ each\ time\ step\ (i) \tag{1}$$

The output of hidden state is calculated using the formula given in the equation (2)

$$h_t = f_h\left(Wx^t + W_h h^{(t-1)} + b_h\right) \tag{2}$$

Similarly, the estimated output at each time step is calculated using the formula given in equation 3.

$$\hat{y} = f_y(Uh_t + b_y) \tag{3}$$

Where f_h, f_y are the activation function at the hidden state and output state.

Problem with RNN: RNN has the problem of vanishing gradient and exploding gradient as discussed in [1].

1.2 Vanishing Gradient and Exploding Gradient Problem in Recurrent Neural Networks

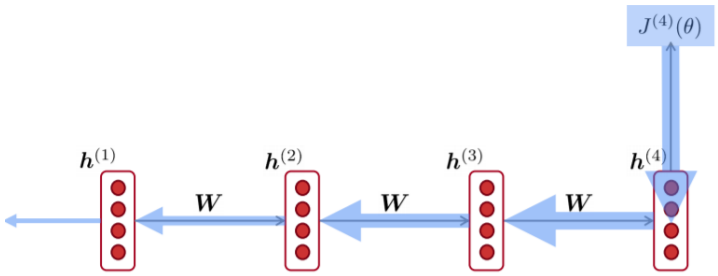

Fig 2: Gradient Calculation with respect to hidden state

The gradient of loss function $J^{(4)}(\theta)$ with respect to the hidden state $h^{(1)}$ can be calculated using the formula give in equation (4):

$$\frac{\partial J^{(4)}}{h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}} \tag{4}$$

Whereas gradient of hidden state at time step (t) with respect to its previous state (t-1) is given using the formula:

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \text{diag}(\sigma'(Wx^t + W_h h^{(t-1)} + b_h))\, W_h \qquad (5)$$

Now if $\frac{\partial h^{(2)}}{\partial h^{(1)}}, \frac{\partial h^{(3)}}{\partial h^{(2)}}, \frac{\partial h^{(4)}}{\partial h^{(3)}}$ are small, then we will have smaller and smaller gradient signal as we backpropagate further in the network.

Mathematically the gradient of loss $J^{(i)}(\theta)$ on the step i with respect to hidden state $h^{(j)}$ on some previous state can be presented using the formula

$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^{i-j} \prod_{j<t\leq 1} \text{diag}(\sigma'(Wx^t + W_h h^{(t-1)} + b_h))\, W_h \qquad (6)$$

Now if W_h^{i-j} becomes smaller and smaller if the distance between i and j increases. If the largest eigenvalue of W_h is less than 1, then $\left\|\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}}\right\|$ will shrink at exponential rate [3]. Similarly, if $W_h$is greater than 1, then we will have the problem of exploding gradient.

1.3 Effect of vanishing gradient and exploding gradient on training

Due to the vanishing gradient problem, gradient signal from faraway distance will be lost. This is due to the reason that gradient signal from a nearby will be much larger than the gradient signal that is received from the far way distance. Now the model will only update due to the near effect. We will no longer see the long-term effect in the network. If the gradient becomes very small over large distance, then we will not be able to tell that there is no dependency between time step t and t+ n data. We will not be able to tell that the parameters that are being used to the capture the true dependency at time t and t + n.

If the gradient becomes too big then stochastic gradient descent update becomes too big.

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta) \qquad (7)$$

Where α is the learning rate and $\nabla_\theta J(\theta)$ is the gradient of loss function. If the gradient is too big, then θ^{new} will be big also. We will have large loss during training due to this.

The hidden states were constantly rewritten in the hidden state of RNN. The problem of vanishing gradient was solved using the Long short-term memory (LSTM) networks proposed by Hochreiter and Schmidhuber as presented in [4].

2 LSTM

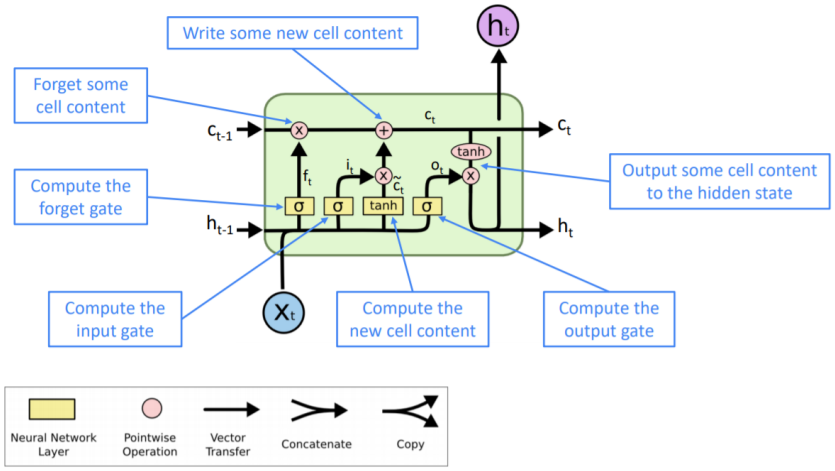

Figure 1. An illustration of the LSTM block [2,5]

Figure 1 shows the architecture for LSTM memory cell. LSTM has cell state $c^{(t)}$ in addition to the hidden state $h^{(t)}$ at each time step. LSTM has 3 gates namely input gate, forget gate and output gate. The cell state is used for storing the long-term memory information in LSTM. We can read, write and erase the information from these cells in LSTM by controlling the gates. Gates consists of sigmoid layer. Sigmoid layer gives the output between zero and one. At each time the value of gates has output values between zero and one. If gate has the output value of zero, this means gate is closed and we are not allowing any information to pass through these gates. If gate has the output value of one, this means gate is open and we are allowing all of the information to pass through these gates.

2.1 Step by Step Explanation of LSTM Algorithm.

Step 1

LSTM has sequence of inputs. We have to calculate the sequence of hidden states and cell states in LSTM at each time step. The first step in LSTM is to decide how much information we want to retain or discard from the previous cell state using the forget gate f_t. Forget f_t has the sigmoid layer which will give the output between 0 and 1 by looking at the previous hidden state $h^{(t-1)}$ and input $x^{(t)}$ at each time step using the equation 8. Again, value of 1 means we want to retain information and completely and the output value of 0 means we want to discard the entire information.

$$f^t = \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f) \qquad (8)$$

Step 2

In the next step we update the cell state $c^{(t)}$. In this step we decide whether we want to forget the some of content from the last cell state or we want to write new content to the cell. This step is divided into two parts. In the first part we have input gate $i^{(t)}$ consisting of sigmoid layer. The input gate controls what part of the new cell content should be written to the cell state. Input gate takes the previous hidden state $h^{(t-1)}$ and current input $x^{(t)}$ as input and then produces output between zero and one using sigmoid layer.

$$i^{(t)} = \sigma(W_f h^{(t-1)} + U_i x^{(t)} + b_i) \qquad (10)$$

Next step is to calculate new vector of values $\tilde{c}^t$ that we need to be write to the cell state. $\tilde{c}^t$ takes the previous hidden state and current input state as input and produces the output between -1 to 1 using tanh layer. This helps in regulating the network.

$$\tilde{c}^t = tanh(W_c h^{(t-1)} + U_c x^{(t)} + b_c) \quad (11)$$

In the part two we update the cell state. The output of input gate $i^{(t)}$ and the new cell content $\tilde{c}^t$ and the output of forget gate f^t is multiplied and then added to create the update for the cell state $c^{(t)}$. Gates are applied using the dot product.

$$c^{(t)} = f^t . c^{(t-1)} + i^t . \tilde{c}^t \quad (12)$$

Step 3

In the last step we decide what should be our next hidden state. The hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we will pass the previous hidden state $h^{(t-1)}$ and the current input $x^{(t)}$ through the output gate. Output gate is used for controlling the part of the cells that are given as input to the hidden state using the sigmoid layer.

$$o^{(t)} = \sigma(W_o h^{(t-1)} + U_o x^{(t)} + b_o) \quad (13)$$

The newly updated cell state $c^{(t)}$ is the passed through tanh function. This will push the values of $c^{(t)}$ between -1 and 1. Finally the hidden state $h^{(t)}$ is calculated by multiplying $o^{(t)}$ with the output of tanh function.

$$h^{(t)} = o^t . \tanh c^{(t)} \quad (14)$$

2.2 Training of LSTM

LSTM is trained using backpropagation with time. LSTM learns the weight of input, output and forget gate during the training process. LSTM also learns the weight of input tan hyperbolic layer during the training.

2.3 How LSTM solved the problem of vanishing gradient

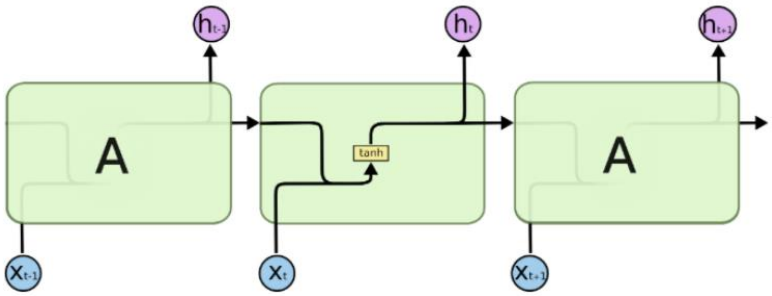

Fig 3a: Repeating module of RNN [5]

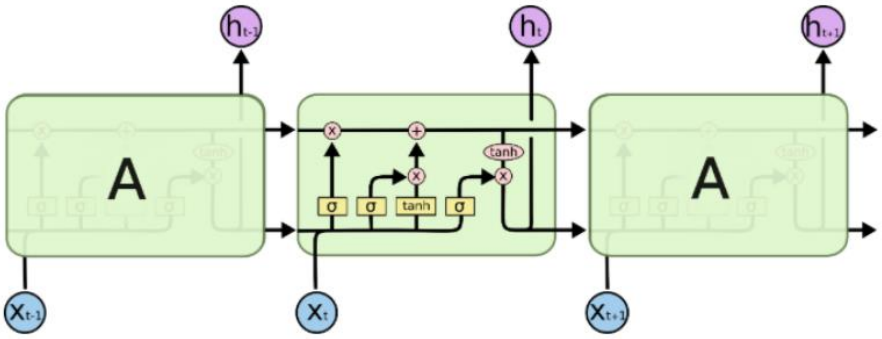

Fig 3b: Repeating module of LSTM [5]

RNN and LSTM forms a chain like structure by repeating the module multiple times. The main difference between LSTM and RNN is in the structure of repeating module. Structure of repeating module of RNN is very simple. As shown in figure 3a, it only consists of one tanh layer while repeating LSTM has four layers that interacts in a special way.

Long short-term memory (LSTM) solves the vanishing gradient in vanilla recurrent neural networks by the addition of cells, input, and output gates. Instinctively, the problem of vanishing gradients is resolved by the adding input and forget gate activations resulting in better control over the gradient flow. Information in the cell can be preserved indefinitely by setting the forget gate remember everything on every timestep.

The addition of these gates will help gradients to flow throughout the network. LSTM solves the problem of long-term dependencies by increasing the number of repeating layers.

2.4 Limitations of LSTM

LSTM requires two additional inputs (input gate and output gate) as compared to the RNN. Increase in the number of inputs increases the total number of weights that we need to train in LSTM by the factor of 9 [4]. The hidden layers in RNN are replaced by maximum of 3 units in LSTM. So, we will 3^2 more parameters in the network.

LSTM are computationally expensive than RNN. They require more memory [6]. On average number of parameters in LSTM are 4 times higher than the number of parameters in RNN [7]. The number of parameters in LSTM and RNN are presented using the formulas given in the equation 15 and 16.

$$\text{LSTM} = 4 \times [\,(h + i) \times h + h] \qquad (15)$$

$$\text{RNN} = 1 \times [\,(h + i) \times h + h] \qquad (16)$$

Where h is the number of hidden states and i is the number of inputs. LSTM has 4 fully connected neural networks (FNN) while RNN has only one fully connected neural network.

Similar to feed forward neural networks, LSTM faces the problem of not seeing all the inputs strings at the same time. This is due to the constant flow of errors through constant error carrousel (CECs) in the memory cell [4].

2.5 Advantages of LSTM

LSTM is robust to vanishing gradient problems as compared to the RNN. LSTM has the ability of modeling long time dependencies as compared to RNN due to the constant backpropagation of errors in memory cells [4]. There is no need of fine tuning of parameters in LSTM [4].

3 Variants of LSTM

3.1 LSTM with a Peephole Connection

In traditional LSTM gates are connected with inputs and outputs of each cell. They lack the direct connection of cell states to the gates. The gates do not have any information about the cell states. This lack of information can affect the performance of the network [8]. Gers and Schmidhuber solved this problem by proposing LSTM with peephole connections in [8].

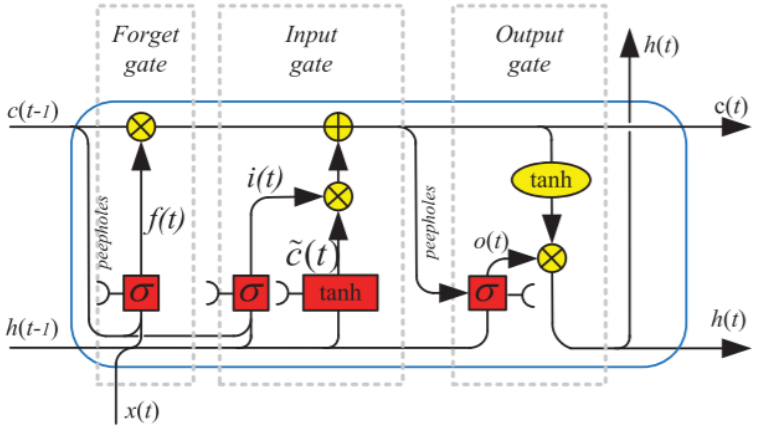

Fig 4: LSTM with peephole connections

As seen in the above figure now there is a direct connection between cell state and gates through these peephole connections. This direct connection allows gates to inspect cell state directly. Modified equations of input gate, forget gate and output gate after adding peephole connections are given as:

$$f^t = \sigma(W_f h^{(t-1)} + U_f x^{(t)} + P_f . c^{(t-1)} + b_f)$$

$$i^{(t)} = \sigma(W_f h^{(t-1)} + U_i x^{(t)} + P_i . c^{(t-1)} + b_i)$$

$$o^{(t)} = \sigma(W_o h^{(t-1)} + U_o x^{(t)} + P_o . c^{(t-1)} + b_o) \quad (17)$$

Whereas P_f, P_i, P_o are the weights of peephole connections. The addition of peephole connections allows LSTM to learn precise and stable learning algorithms [8].

3.2 LSTM with a coupled input and forget gate

Greff et all in [9] compared performance of the eight variants. Eights variant were LSTM without input gate, output gate, forget gate, input activation function, output activation function, peepholes connection, recurrent units and LSTM with coupled forget and coupled input gates.

Results showed that LSTM with coupled forget gate and input gate reduces the computational cost and number of parameters without the significant decrease in the performance of network. LSTM is becoming very popular in deep learning due to this property. In contrast to the simple LSTM decision to forget or add new information are taken together with coupled input and forget gates. LSTM with coupled gates only forgets when it needs to write something and it only writes when it wants to forget something. New cell state using this model can be given using the equation (18).

$$c^{(t)} = f^t . c^{(t-1)} + (1 - f^t) . \tilde{c}^t \qquad (18)$$

3.3 Gated Recurrent Units (GRU)

Despite tackling the problem of vanishing gradient and exploding gradient, LSTM has high computational complexity as compared to RNN. Memory requirement for the LSTM is also higher than RNN. To overcome these problems Cho et all proposed GRU in [10].

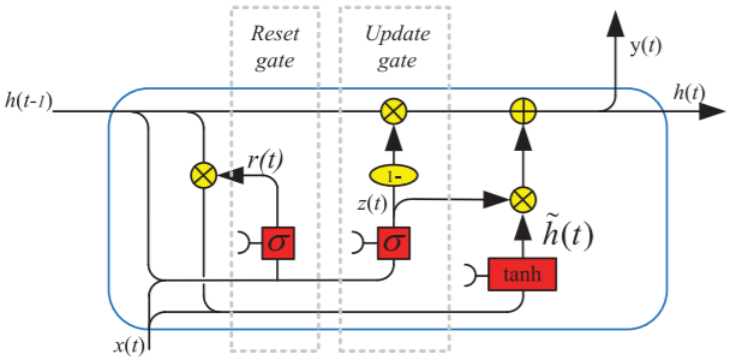

Fig 5: GRU Architecture

GRUs do not have cell states in them. At each time step GRU has input state and hidden state. GRU has only two gates. Update gate and reset gate. GRU combines input gate and forget into update gate. Hidden states can be updated or preserved by controlling the update gate. Update gate controls which parts updating or preserving the part of hidden states. Reset gate. Reset gates are used for controlling the part of previous hidden states that are required for

$$u^{(t)} = \sigma(W_u h^{(t-1)} + U_u x^{(t)} + b_u)$$

$$r^{(t)} = \sigma(W_r h^{(t-1)} + U_r x^{(t)} + b_r) \qquad (19)$$

Useful parts of previous hidden states are selected using the reset gate. GRU uses the useful part selected by the reset gate and the current input to compute the new hidden state $\tilde{h}^t$. As shown in equation 20.

$$\tilde{h}^t = tanh(W_h(r_h . h^{(t-1)}) + U_h x^{(t)} + b_h) \qquad (20)$$

Hidden state in the GRU is computed using the update gate. Update gate simultaneously control which part of the previous hidden state needs to be kept and which part of the previous hidden state is updated to the new hidden state $h^{(t)}$.

$$h^{(t)} = \left(1 - u^{(t)}\right) . h^{(t-1)} + u^{(t)} . \tilde{h}^t$$

Because GRU has a smaller number of gates, the number of parameters associated with GRU will also be less. GRU is faster than LSTM. The GRU cannot be taught to count or to solve context-free language [11].

3.4 Bidirectional LSTM

Bidirectional LSTM was proposed by Graves, A. and Schmidhuber in [12]. Bidirectional LSTM can share information in two ways. Bidirectional LSTM network can be trained with forward and reverse direction. The LSTM and bidirectional LSTM networks are linked to the similar output layer and this kind of design improves the performance for language modeling

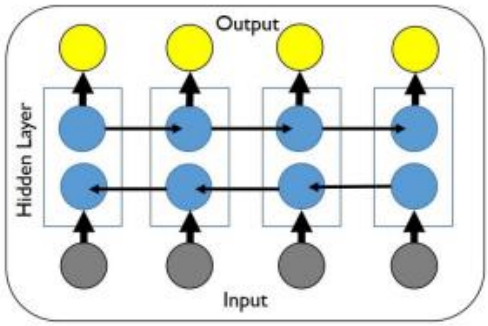

Fig 6: Bidirectional LSTM

4 Applications of LSTM

Different methods have been developed along with deep bidirectional, long short-term memory models to implement speech recognition method i.e. graph method.

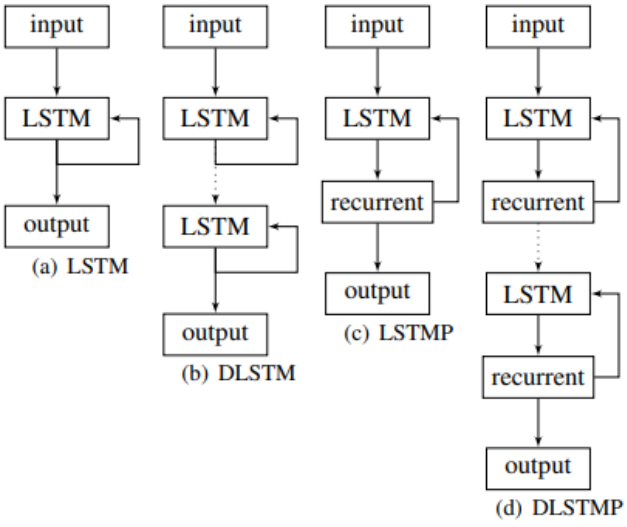

Figure 7 : LSTM RNN architectures

In deep neural networks (DNNs) along with deeper layers, deep LSTM recurrent neural networks (RNNs) have been effectivity used speech recognition tasks. Multiple LSTMs layers can be stacked to constructed

deep LSTM recurrent neural networks (RNNs). Notice that LSTM recurrent neural networks (RNNs) are also deep models in the context that they are being found a feed-forward neural network unpacked at a time interval when each layer in the network contains the same variables of the model. In the deep LSTM model, the input is given to model through multiple non-linear layers as in deep neural networks, but the attributes from the over the certain time period are only handled by only one non-linear layer before participating the output for that time period. Thus, the number of layers (depth) in the deep LSTM recurrent neural network has a supplementary sense. The input data over specific time interval passes through numerous LSTM layers along with backpropagation through time and LSTM network layers. It has been claimed that deep layers in the recurrent neural networks (RNNs) enables the model to learn from input data over distinct time intervals. Deep LSTM recurrent neural networks (RNNs) provide an additional advantage over traditional LSTM recurrent neural networks (RNNs): They can produce efficient use of variables by scattering them across numerous levels over space. For example, instead by growing the size of the memory by a multiplier of 2, one must have exactly similar number of parameters. This leads to input data moving through non-linear processors every time period. Figure 7 shows the LSTM RNN architecture.

References:

[1] Bengio, Y., Simard, P. and Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), pp.157-166.

[2] https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture06-rnnlm.pdf

[3] Pascanu, R., Mikolov, T. and Bengio, Y., 2013, February. On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310-1318).

[4] Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, *9*(8), pp.1735-1780.

[5] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] Salehinejad, H., Sankar, S., Barfett, J., Colak, E. and Valaee, S., 2017. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*.

[7] https://towardsdatascience.com/counting-no-of-parameters-in-deep-learning-models-by-hand-8f1716241889#192e

[8] Gers, F.A. and Schmidhuber, J., 2000, July. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* (Vol. 3, pp. 189-194). IEEE.

[9] Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R. and Schmidhuber, J., 2016. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), pp.2222-2232.

[10] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[11] Weiss, G., Goldberg, Y. and Yahav, E., 2018. On the practical computational power of finite precision RNNs for language recognition. *arXiv preprint arXiv:1805.04908*.

[12] Graves, A. and Schmidhuber, J., 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks*, *18*(5-6), pp.602-610.