Abstract

In these recent years significant progress has been made in successfully training RNNs using stochastic gradient descent methods such as back-propagation through time (BPTT) (Rumelhart et al., 1986) have been riddled with difficulty. Early attempts suffered from the so-called vanishing gradient or exploding gradient problems resulting in difficulties to learn longrange temporal dependencies (Hochreiter et al., 2001). Several methods have been proposed to overcome the difficulty in training RNNs. These methods broadly fall into three categories: (i) Algorithmic improvements involving sophisticated optimization techniques, like the Hessian-Free (HF) optimization method. (ii) Network design, the most successful technique to date under this category is the Long Short Term Memory (LSTM) RNN (Hochreiter & Schmidhuber, 1997). The LSTM is a standard RNN in which the monotonic nonlinearity such as the tanh or sigmoid is replaced with a memory unit that can efficiently store continuous values and transmit information over a long temporal range. Lastly, (iii) Weight initialization, it has emerged as a critical area of focus.

This paper aims to explore and study how various weight initialization methods impact the performance of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) models. We will examine a range of initialization techniques, including IRNN (initialized to Identity matrix), iRNN (0.01 times the Identity matrix), nRNN (matrix with all eigenvalues except the highest less than 1), np-RNN (normalized positive definite matrix with all eigenvalues except the highest less than 1), oRNN (orthogonal random matrix), gRNN (random Gaussian matrix), and well-known methods like Xavier and Kaiming initialization.

For the assessment of LSTM performance, we will utilize a "polymorphic music modeling" approach, while RNN performance will be examined through a "Sentiment Analysis" model. These specific applications will allow us to directly observe the effects of different weight initialization methods on the capabilities and efficiencies of LSTMs and RNNs in handling complex tasks, ranging from music generation to interpreting and analysing sentiments expressed in text.

The RNN consists of an input layer, a hidden layer with recurrent connections and an output layer. Given an input sequence $X = \{x_0, x_1, x_2 \cdots x_T\}$, the RNN will take the input $x_t \in R^N$ and generate the prediction $y_t \in R^C$ for the output sequence $Z = \{z_0, z_1, z_2 \cdots z_T\}$, where zt $\in R^C$. In between the input and the output there is a hidden layer with M units, which store information on the previous values (t 0 < t) of the input sequence. More precisely, the RNN takes X as input and generates an estimate Y of the output Z by iterating the equations.

$$s_t = W_{hx} x_t + W_{hh} h_t{-}1 + b_h \quad (1)$$

$$h_t = f(s_t) \quad (2)$$

$$o_t = W_{yh} h_t + b_y \quad (3)$$

$$y_t = g(o_t) \quad (4)$$

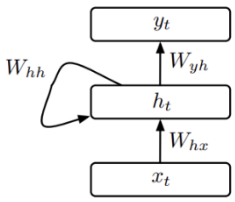


Figure 1: Schematic diagram of a simple RNN network

where W_{hx}, W_{hh} and W_{yh} are the weight matrices, b_h and by are the biases, $s_t \in R^M$ and $o_t \in R^C$ are inputs to the hidden layer and the output layer respectively and f and g are functions.

Initialization type	**Description**
Identity	Weight matrix initialized to Identity matrix
Identity * 0.01	Weight matrix initialized to 0.01 times Identity matrix
Zero	Weight matrix initialized to zero values
Non-zero constant	Weight matrix initialized to non-zero constant
Random Uniform	Random weight initialized with a uniform distribution
Random Normal	Random weight initialized with a normal distribution
Truncated	Random weight initialized with Truncated method
Xavier	Weight matrix initialized with Xavier method
Normalized Xavier	Weight matrix initialized with Normalized Xavier method
Kaiming	Weight matrix initialized with Kaiming method
Orthogonal	Weight matrix initialized to orthogonal random matrix

Different method of RNN initialization

1. Identity matrix initialization and variants

This initialization initializes the weights with identity matrix and the bias terms are set to zero.

$$W = I_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \qquad 0.01 \times I_n = 0.01 \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

2. Constant initialization

Initializing all the weights of the RNN with any non-zero constant initialization the model will perform poorly, assuming the model initialize all the bias to 0 and the weights withs some constant α. If the forward propagates an input (x_1, x_2, ..., x_n), the output of both hidden units will be *Relu*($\alpha x_1 + \alpha x_2 + ... + \alpha x_n$). The hidden units will have identical influence on the cost, which will lead to identical gradients. So, both neurons will change in the same way during training, and this may produce a convergence issue, to how weights are updated during the learning process, it will have a lack of initial diversity in the weights can limit the exploration of the solution space.

But, sometimes, this initialization may be helpful in order to train the same final set of network weights given a training dataset in the case where a model is being used in a specific production environment.

$$W = \begin{pmatrix} \alpha & \alpha & \cdots & \alpha \\ \alpha & \alpha & \cdots & \alpha \\ \vdots & \vdots & \ddots & \vdots \\ \alpha & \alpha & \cdots & \alpha \end{pmatrix}$$

Figure 2: Non-zero constant weights initialization

In fact, if the constant initialization weights the constant value is zero, the weight update during training depends on the gradient value, if all weights start and stay at zero, the gradients will be zero, and therefore, there will be no weight update. The network will not be able to learn anything.

$$W = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

Figure 3: Zero weights initialization

3. Radom initialization

This initialization uses randomness in order to find a good diversity set of weights for the specific mapping function from inputs to outputs in your data that is being learned. It means that your specific network on your specific training data will fit a different network with a different model skill each time the training algorithm is run, getting a good convergence of the model.

Usually, especially with stochastic gradient descent, the weights are initialized to small random values between 0 and 0,1.

https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/

Random normal - generates the random weights with a normal distribution, the arguments for the distribution are the mean and the standard deviation vales for the normal distribution.

$$W \sim \mathcal{N}(\mu, \sigma^2)$$

Figure 4: Normal distribution

Random uniform - generates the random weights with a uniform distribution, the arguments for the distribution are the lower and upper bound of the range of values of the uniform distribution.

$$W \sim \mathcal{U}(\text{minval}, \text{maxval})$$

Figure 5: Uniform distribution, bounds [minval, maxval]

4. Truncated normal initialization

Initializer that generates a truncated normal distribution.

The values of the weights generated are similar to values from a Random normal initializer, except that values more than two standard deviations from the mean are discarded and re-drawn.

$$W \sim \text{Trunc}\mathcal{N}(\mu, \sigma^2)$$

Figure 6: Truncated normal distribution

5. Xavier/Glorot and Normalized Xavier/Glorot initialization

The Xavier initialization was described by Xavier Glorot, currently a research AI scientist at Google DeepMind, in the 2010 paper "Understanding The Difficulties Of Training Deep Feedforward Neural Networks".

Xavier initialization calculates the weights as a random number from a uniform distribution (*U*), where *n* is the number of inputs to the node:

$$W \sim \mathcal{U}\left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right)$$

Figure 7: Xavier/Glorot initialization

Normalized Xavier initialization calculates the weights as a random number from a uniform distribution(*U*), where *n* us the number of inputs to the node and *m* is the number of outputs from the layer.

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}}\right)$$

Figure 8: Normalized Xavier/Glorot initialization

The "*xavier*" weight initialization was found to have problems when used to initialize networks that use the rectified linear (ReLU) activation function.

The current standard approach for initialization of the weights of neural network layers and nodes that use the rectified linear (ReLU) activation function is called "kaiming/he" initialization.

6. Kaiming/He initialization

The Kaiming initialization was described by Kaiming He, currently a research AI scientist at Facebook, in the 2015 paper, winner of 2015 ImageNet challenge: "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification".

Kaiming initialization calculates the weights as a random number with a Gaussian distribution (G) with a mean of 0.0 and a standard deviation of sqrt(2/n), where n is the number of inputs to the node.

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right)$$

Figure 9: Kaiming/He initialization

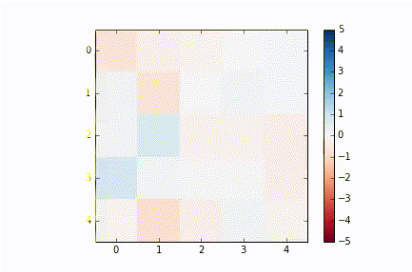
Kaiming initialization is very similar to Xavier initialization the only difference is that the Kaiming paper takes into account the activation function, whereas Xavier does not (or rather, Xavier approximates the derivative at 0 of the activation function by 1).

7. Orthogonal initialization

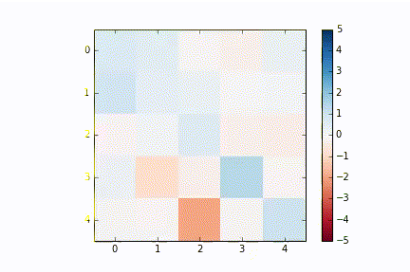
Orthogonal initialization is based on the algebraic principle of Orthogonal matrix, it has many interesting properties but the most important for us is that all the eigenvalues of an orthogonal matrix have absolute value 1. This means that, no matter how many times we perform repeated matrix multiplication, the resulting matrix doesn't explode or vanish.

It's interesting to note what the constraint that an eigenvalue must have absolute value 1 means. If we are only using real numbers, that means the eigenvalues must be either +1 or -1, resulting in very boring matrices. We can extend to complex eigenvalues however, allowing for far more interesting results when repeatedly multiplied.

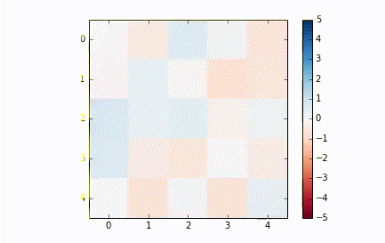
Vanishing matrix due to small eigenvalues



Exploding matrix due to an eigenvalue > 1



Orthogonal matrix



Prior related work:

Talathi, S. S., & Vartak, A. (2016). Improving Performance of Recurrent Neural Network with ReLU Nonlinearity. Qualcomm Research. San Diego, USA.

https://openreview.net/pdf/wVqq536NJiG0qV7mtBNp.pdf

This document tackles the RNN difficulties in training, comparing different types of initialization methods. This paper serves as a starting reference for what our different matrices could look like. For example, it uses IRNN and iRRN and compares their performance under the same conditions.

Glorot, X., & Bengio, Y. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. DIRO, Université de Montréal, Montréal, Québec, Canada.

https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

This research highlights the inefficiencies of random initializations in deep neural network training, such as the saturation of the top hidden layer when using logistic sigmoid activations. Xavier initialization maintaining the variances of activations and back-propagated gradients as consistent as possible across the layers of the network. This approach aims to address the issue of vanishing or exploding gradients by ensuring a balanced flow of gradients and activations throughout the network.

Ouannes, P. (2019, March 22). How to initialize deep neural networks? Xavier and Kaiming initialization [Blog post]. Retrieved in April 2024. https://pouannes.github.io/blog/initialization/

The text shows the mathematical implications for forward and backward propagation in neural networks. Also, this publication introduces another initialization method known as Kaiming/He initialization. The difference between Xavier and Kaiming initializations is that Kaiming considers the activation function, whereas Xavier does not (or rather, Xavier approximates the derivative at 0 of the activation functions by 1).

Stephen Merity (June 27, 2016). Explaining and illustrating orthogonal initialization for recurring neural networks.

https://smerity.com/articles/2016/orthogonal_init.html

Orthogonal initialization stands out as a superior approach for using the principle of matrix eigendecomposition, which breaks down a matrix into its eigenvalues and eigenvectors. This method ensures that the multiplication of a matrix by itself, even repeatedly, results in stable outcomes. Orthogonal matrix, characterized by eigenvalues with absolute values of 1, guarantee that repeated matrix multiplication maintains the norm of the result, avoiding the common problems of exploding or vanishing gradients, especially in recurrent neural networks (RNNs). This approach allows for the stable training of RNNs by ensuring that the weight matrix, when initialized orthogonally, preserves gradient magnitude over time.

Different initialization with PyTorch: https://pytorch.org/docs/stable/nn.init.html