Finding Time Series Breakpoints with Fully Connected Neural Networks

Amy Pitts[1] **and Pablo Rivas**[2]
[1]Department of Mathematics, [2]Department of Computer Science
School of Computer Science and Mathematics, Marist College, New York

Abstract— *Breakpoint analysis is a technique that helps individuals better understand and model time series. This paper describes how a fully connected neural network can predict breakpoints in time series. Using a sigmoidal activation, an MSE loss function, an Adam optimizer and drop out rate this function is able to estimate the location of breakpoints. This model is training on simulated time series with clear breaks created by mean changes and multiple trials were run to identify the optimal hyper-parameters. Then, the model is applied to real world pelican population data. Comparing results to other breakpoint analysis approaches this neural network model identifies the general location where a known breakpoint occurs in this pelican data.*

Keywords: Neural Networks, Breakpoints, Time Series

1. Introduction

When modeling time series data, it can be necessary to identify places or points in time where significant change occurs in the behavior of the data. By identifying these breakpoints or change points, different parts of the data can be fitted with separate, more appropriate models, allowing for noteworthy changes in the data to be better represented in the combined model. In 2018, in collaboration with Lafayette College Research Experience for Undergraduates (REU) program, we developed a Bayesian procedure that identifies the number and location of breakpoints in times series. Such research produced a Bayesian Adaptive Auto-Regression (BAAR) model which is a new procedure for finding the distribution of the number and location of breakpoints in time series [cite under review]. While such approach is statistical model, we now address the same problem from a neural network perspective. This research proposes using a fully connected dense neural network approach to locate breakpoints in time series. The algorithm learns from simulated data where the breakpoint set is known, and then it is tested on unseen non-simulated data. Both the neural model and BAAR are compared and finding that the neural model is also able to find breaking points in highly nosy data.

2. Background and History

Since breakpoints are found in numerous types of time series, there has been ample interest in developing techniques to detect them in recent decades across a wide range of fields. Existing techniques have been applied to everything from the United States Treasury bill rates [3] to hydrology [5] and climate records [4].

There have been various techniques in detecting change-point locations. The simplest technique relies on expert opinion. This is the process of experts approximating where the breakpoint location will occur based on historical knowledge. However, there is a lot of human error introduced with that technique which has sparked the development of more computational methods. One widely used technique is the Bai-Perron Test [1] which has an accessible R package "strucchange" [7]. The test returns a single optimal breakpoint set but requires the user to specify a maximum number of breakpoints and minimum segment size. Another technique is Bayesian Adaptive Regression Splines (BARS), a Bayesian curve fitting technique developed by DiMatteo et al. [2] and implemented by Wallstrom, Liebner, and Kass [6]. These two methods inspired the Bayesian Adaptive Auto-Regression (BAAR) procedure mentioned earlier [cite under review]; however, no machine learning approach has been investigated until now. This paper is the first attempt on using fully connected neural networks for the detection of breakpoints.

A fully connected neural network consists of a collection of fully connected layers from one domain $m \in \mathbb{R}$ to $n \in \mathbb{R}$ [10]. The input nodes affect the output nodes. Historically, neural networks were first introduced by Frank Rosenblatt with the Perceptron in the 1950s [10]. The input data is used to find weights, and then the weights and inputs are multiplied and summed together. Using the total sum, a step function was applied to obtain an output [10]. However, the original Perceptron model was fundamentally limited when encountered with data that cannot be linearly separated into two groups [10]. This problem can be overcome with a multilayered Perceptron. Each node is referred to as a neuron. This name was inspired by a paper published by Warren McCulloch and Walter Pitts in the 1940s about the mathematical model of a brain, specifically how neurons are connected and interact [10]. However, it should be noted that fully connected networks do not fully capture how neurons behave. Today, fully connected networks are more robust with several alternatives for backpropagation, choices or activation functions, and ability to train multiple layers

more efficiently.

3. Method

Fully connected networks are known to be "structure agnostic", meaning that there are no special assumptions needed to be made about the input [10]. The network starts with an initial number of neurons. Let m represent the input of the first layer and let y_i be the output of the layers. The y_i is given by [10]:

$$y_i = \sigma(w_1 x_1 + \cdots + w_m x_m). \tag{1}$$

Since σ is a nonlinear function and w_i are the parameters learned, the full output y is then [10]

$$\begin{aligned} \mathbf{y} = &[\sigma(w_{1,1} x_1 + \cdots + w_{1,m} x_m), \\ &\ldots, \sigma(w_{n,1} x_n + \cdots + w_{n,m} x_m)]\,. \end{aligned} \tag{2}$$

As an example, Fig. 1 shows a network with 64 neurons that fully connect to 5 other neurons in a separate layer.

In our dataset, we produce vectors that contain breaking points at random locations. The data is split into chunks on five and passed into the network. The chunks of five points are necessary so the network can detect if there is a single jump (breakpoint) in the data. If the chunk was a larger number than five then the section of data might encompass more then one breakpoint. This would make it difficult for the neural network to select the placement of both breakpoints. Therefore, by limiting the size of the chunks we limit the chances of getting two breakpoints or more in each chunk.

After choosing a specific number of neurons for the dense layer, it follows to choose a proper activation. The sigmoidal function is used for this activation in this project. The sigmoid also known as a logistic function, and is given by [9]:

$$f(s_i) = \frac{1}{1 + e^{-s_i}}. \tag{3}$$

A softmax activation can be also attempted; however, our experimental results showed that the sigmoid converges to more accurate results in terms of our loss function. Our choice of loss function is the mean squared error (MSE). This is given by:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2 \tag{4}$$

where $\tilde{y}_i$ is our predicted value and y_i is the actual value.

To control the gradient descent the 'adam' optimizer is used. Adam is "an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments" [8]. This gradient descent algorithm does backpropagation based on the loss function.

In the proposed model we also applied a drop out of strategy of 25%. This form of regularization randomly drops some of the nodes during the gradient descent [10]. Therefore, less nodes go into the next fully connected layer. This plays an important role in avoiding a single neuron gaining too much power, thus controlling the model. A neuron with too much power will make other neurons too dependent on it, which is a phenomenon known as co-adaption [10]. With drop out, all neurons will have to learn and be independent on their own.

4. Implementation in Python

4.1 Simulated Data

For the fully connected Neural Network to generalize properly, it required a large amount of data. Specifically, it needs several time series data. Time series data tends to be messy, hard to obtain, and it is hard to definitely know where the exact breakpoints are. Therefore, we created simulated time series and trained the neural network. Simulated time series is ideal because one can control and record the exact location as well as how big the drastic change of the data at the breakpoint location is.

We randomly generated a specified amount of random values between two numbers. We alternating between randomly choosing values in the interval (0, 50) and (50, 100). The interval lengths are selected by randomly generating values which represent the breakpoints. For each iteration, 10 different data sets are produced. The first starting with one break, then two breaks, all the way up to 10 breakpoints. Every even interval has values between (0, 50) and every odd interval has values between (50, 100). Fig. 2 depicts an example of a randomly generated time series with one breaking point, while Fig. 3 depicts one with multiple breaking points.

4.2 Fully Connected Neural Network

The methodology described above is implemented in Python and available here:

`github.com/amypitts01/data440.git`

With the code in the repository, the reader has the ability to define the number of data points in each data set and the length of the time series, with a default value of 500. The reader can also define the number of time series produced, with a default value of 100. In regards to the simulated data, the user can also define the sequence length, *i.e.*, the chunk of the time series that will be evaluated one at a time by the network, with a default value of 10. After the data is created, split into sequences and stored in a matrix $\mathbf{X}$, where $x_i \in \mathbb{R}^{10}$. The reader can also define the number of neurons in the hidden layer, with a default value of 128. The reader can also define the number of epochs, with the default value of 100. The default values are set after a large number of experiments collected over multiple trials, all of which are address in the next section.

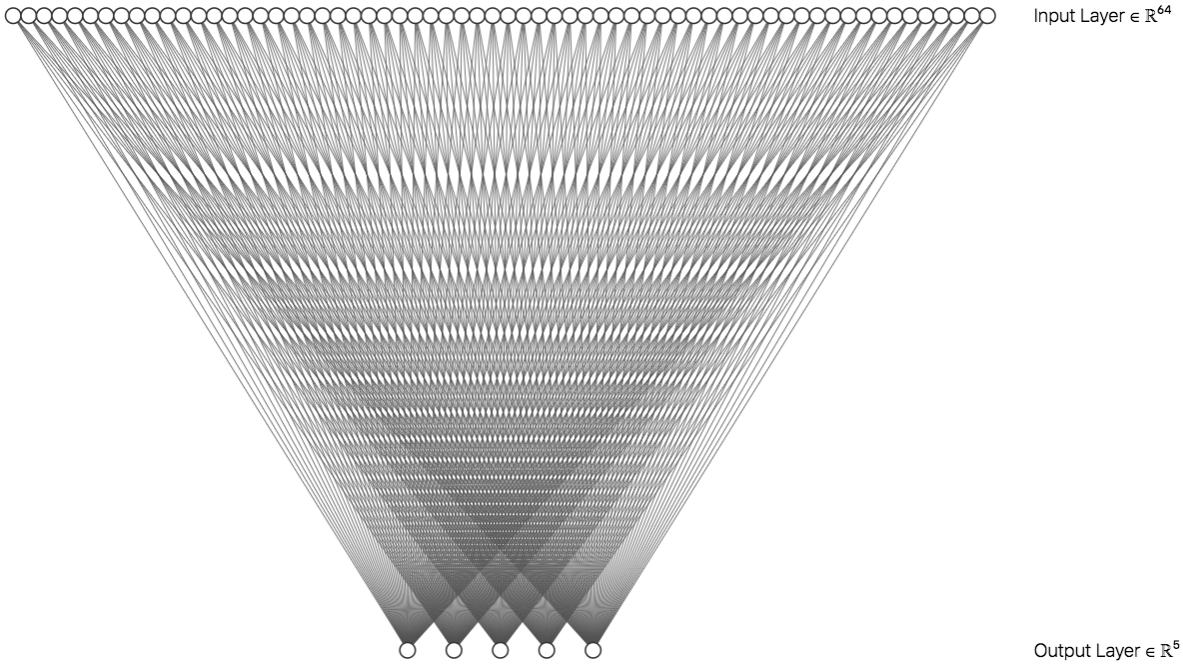

Fig. 1: A dense fully connected network architecture. The input layer takes on an input vector $\mathbf{x} \in \mathbb{R}^{64}$ and, using non-linear activation functions, it maps the input into an output $\mathbf{y} \in \mathbb{R}^{5}$.

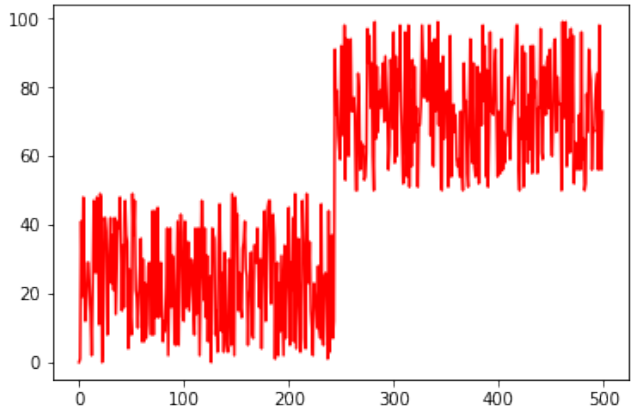

Fig. 2: Randomly generated time series with one Breaking-point.

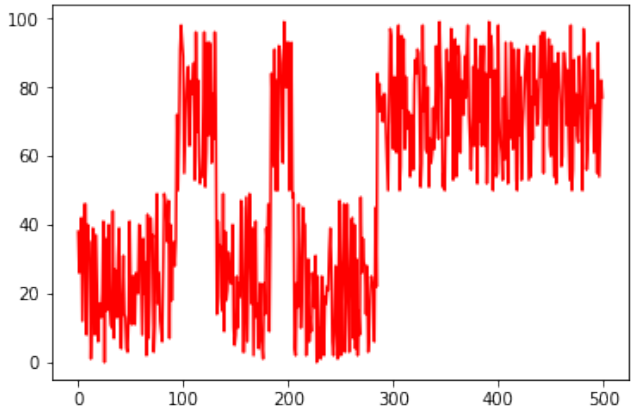

Fig. 3: Randomly generated time series with five Breaking-points.

5. Results

To find the best set model architecture we ran multiple different trails aiming to find the optimal number of neurons, drop out rate, and chunk size of data. Each trial was run for 100 epochs with 500 data points. Both the neurons value and drop out rate were tested against the sequence length. While changing the number of neurons, the drop out rate stayed at 0.1. When testing the number for drop out rate the number of neurons stayed at a constant rate of 128 neurons. The results are displayed in Table 1. Each box displays the average validation binary accuracy in 5-fold cross-validation.

The highest average validation binary in Table 1 is 0.9113 which corresponds to a sequence length of 10 and 128 neurons. The sequence length of 10 obtains a higher accuracy compared to all the other length options. Other attempts were made with 15 and 20 neurons; however, it was overfitting on predicting zero breaking points. Since the goal is to predict

Table 1: 5-fold cross validated accuracy for finding the best number of neurons for a sequence length

	Sequence Length:		
Number of Neurons:	5	7	10
64	0.8567	0.8860	0.9076
128	0.8616	0.8914	**0.9113**
256	0.8605	0.9005	0.9034
512	0.8578	0.8960	0.9061

Table 2: 5-fold cross validated accuracy for drop out rate against sequence length fixing the number of neurons to 128

	Sequence Length:		
Drop Out Rate	5	7	10
0.1	0.8616	0.8914	0.9113
0.25	0.8805	0.8970	**0.9177**
0.5	0.8794	0.8899	0.8982

breakpoints, overfitting of this kind is very detrimental. With this information, we eliminated the option of having 15 and 20 Neurons. Using 10 neurons obtains a higher accuracy and it also has an average validation loss of about 0.0875. This value is smaller than anything produced from the other sequence lengths of 5 and 7. Using this information we can now test different drop out rate. We keep the number of neurons to a consistent 128 and change the sequence lengths and drop out rate. The results are displayed in Table 2.

The highest validation accuracy value is 0.9177 which is obtained by using a sequence length of 10 and a drop out rate of 25%. Using a sequence length of 10 obtains higher accuracy compared to using 5 and 7 as the sequence length. This result is also shown in the data above. Therefore, the optimal combination is using a sequence length of 10, 128 number of neurons and a drop out rate of 25%.

The next test is to see if running the model for more epochs improves the overall validation binary accuracy of the model. In the trials above only 100 epochs were used, now the table below shows the validations after 100, 200, 500, and 1000 epochs. The model was trained on the optimal 10 sequence length, 128 neurons, and a drop out rate of 25%.

Table 3 shows that increasing the number of epochs does not necessarily increase the validation accuracy of the model therefore, 100 epochs is the ideal which will save a lot of training time.

Using these optimal values found I obtain predictions

Table 3: 5-fold cross validated accuracy for finding the best number of epochs

	Number of Epochs:			
	100	200	500	1000
Accuracy	**0.9177**	0.9099	0.9089	0.9083

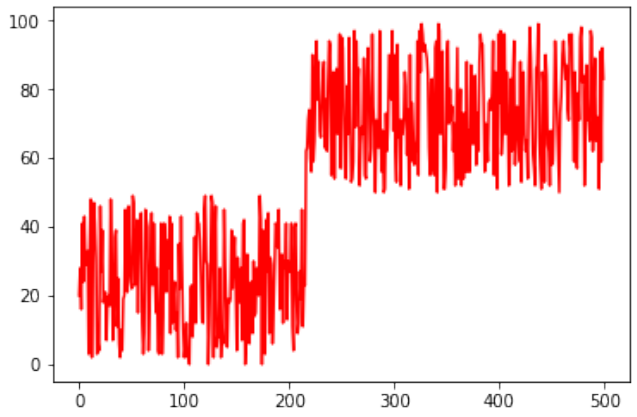

Fig. 4: Data set with one breakpoint shown around time 210.

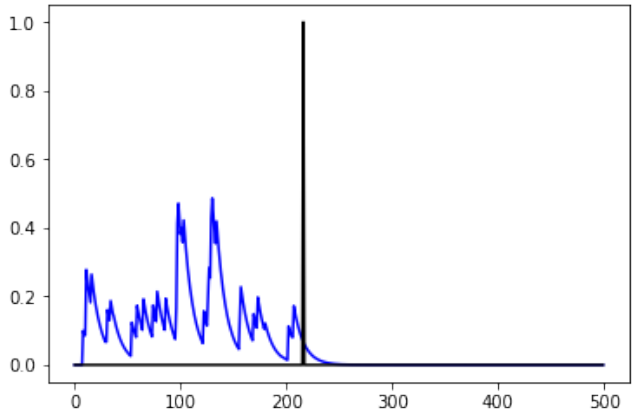

Fig. 5: Black line represents actual breakpoint set, Blue line represents predicted breakpoint set.

which can be visualized in Fig. 4 and 5 with a breaking point at time 210; and in Fig. 6 and 7.

Fig. 5 shows that the model overpredicts the data on the left of the actual breakpoint, displayed in black, and correctly identifies no breakpoints left of the actual breakpoint. The two taller peaks that occur around time 100, and 125 are in fact not breakpoints which demonstrate the limitation of this process. If the model was predicting correctly the predicted breakpoint graph would display black spikes showing were the breakpoints actually are with blue spikes directly over the black.

Fig. 6 shows the data with the predicted breakpoints on Fig. 7. The predicted breakpoint set shown in blue shows small spikes that are close to the actual breakpoints displayed in black. Although the blue spikes are not as tall as the black spikes they are close to the true breakpoints showing that the model does a better job identifying the seven breaks compared to Fig. 5.

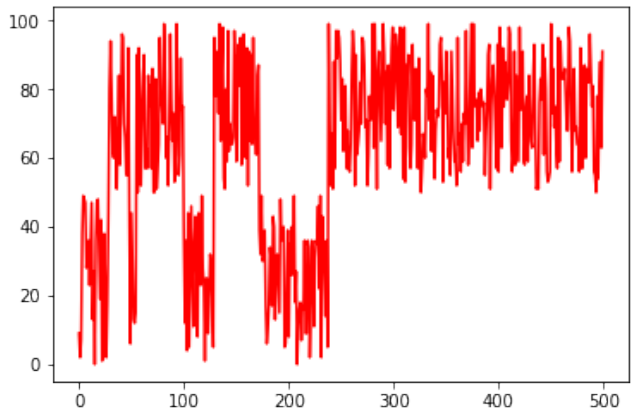

Fig. 6: Data set with seven breakpoints distributed through the graph.

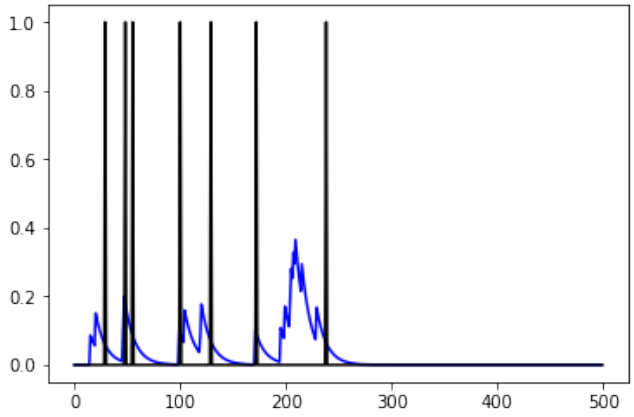

Fig. 7: Black line represents actual breakpoint set, and the blue line represents predicted breakpoint set.

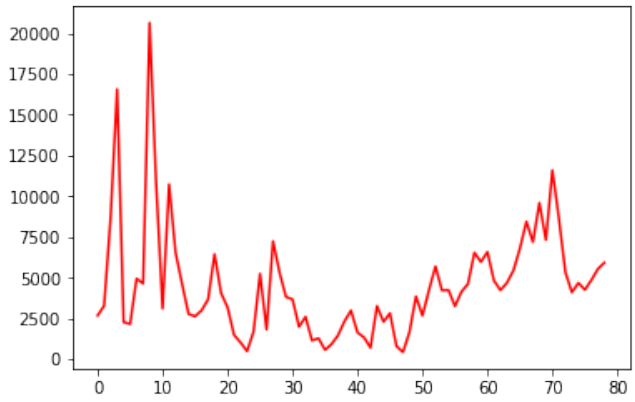

Fig. 8: Original Pelican data

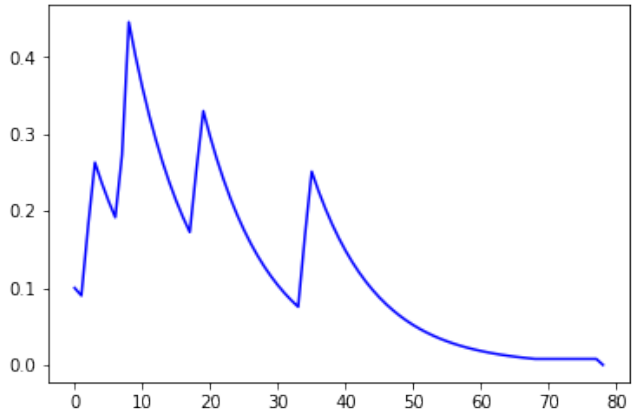

Fig. 9: predicted breakpoints

6. Discussion

Using the best set of neurons, *i.e.*, 128, a dropout rate of 25%, and data partitioning into sequences of length 10 for 100 epochs we can now test the model on new unseen data. The dataset we used is the *Pelican Population* data. The Pacific brown Pelican population data is generated by the Christmas Bird count from 1938 to 2016. This can be seen in Fig. 8, where 1938 is index 0, and 2016 is index 78. The model created by the dense neural network is then used to analyze the pelican data and Fig. 9 is produced. The line represents the predicted breakpoints. Note that there is a spike around time 10 which corresponds to around the 1940s. In fact, using the Bayesian procedure, BAAR, to predict breakpoints, we found a breakpoint in the time frame 1949 and 1952 which is very close to where the method in this paper predicted the breakpoint. This breakpoint corresponds to the introduction of pesticide DDT (dichloro-diphenyl-trichloroethane) for public use. The link between DDT and the decline of brown pelicans is well-established making it an excellent case study for testing the efficacy with real data. This result is exciting because it shows that the breakpoint prediction corresponds to a breakpoint that has been accepted in the academic community.

7. Conclusion

Using a fully connected neural network consisting of a collection of fully connected layers to predict breakpoints although it produces a high accuracy is only semi-reliable. Analyzing the simulated data model using the ideal hyper-parameters produces results shows that the model over-predicts on low mean areas, predicts the break, and then accurately predicts no breaks in the high mean areas. Even with the overfitting, the method does typically identify the correct breakpoint, there are just more false positives. Using the knowledge that there are over predictions when applying the method to pelican data the reader can know be wary excessive results on low mean data. Therefore, using this technique on unseen data can give you general vicinity on where breakpoints occur. An argument can be made about using first derivative filters; however, these are known to

be sequence length-dependent and this causes an unwanted dependency. The model presented here is not depending on the length of the time sequence to give a prediction, but rather, it uses the length to look at chunks of data at a time, but is able to detect changes at smaller time scales. Further research will include an analysis of the weights that the network is learning; it is likely that the network is learning first order filters at different scales and these are activated in different segments of the neural network. This is a conjecture that remains to be supported by further evidence.

References

[1] J. Bai, and P. Perron, "Computation and analysis of multiple structural change models." *Journal of applied econometrics*, vol.18(1), pp.1-22. 2003.

[2] I. DiMatteo, C.R. Genovese, R.E. and Kass "Bayesian curve-fitting with free-knot splines." *Biometrika*, vol.88(4), pp.1055-1071. 2001

[3] M.H. Pesaran, D. Pettenuzzo, and A. Timmermann, Forecasting time series subject to multiple structural breaks. *The Review of Economic Studies*, vol.73(4), pp.1057-1084. 2006

[4] E. Ruggieri, A Bayesian approach to detecting change points in climatic records. *International Journal of Climatology*, 33(2), pp.520-528. (2013).

[5] O. Seidou, and T.B. Ouarda, "Recursion-based multiple changepoint detection in multiple linear regression and application to river streamflows". *Water Resources Research*, vol.43(7). 2007

[6] G. Wallstrom, J. Liebner, and R.E. Kass, "An implementation of Bayesian adaptive regression splines (BARS) in C with S and R wrappers". *Journal of Statistical Software*, vol.26(1), p.1. (2008).

[7] A. Zeileis, F. Leisch, B. Hansen, K. Hornik, C. Kleiber, and M.A. Zeileis. *The strucchange Package*. R manual. breakpoint in the strucchange package (2007)

[8] D. Kingma, and J. Ba "Adam: A method for stochastic optimization" *arXiv preprint arXiv:1412.6980* 2014.

[9] R. Gomez, "Understanding Encoder-Decoder Sequence to Sequence Model" *Raul Gomez blog* Available at: `gombru.github.io/2018/05/23/cross_entropy_loss/` (2018).

[10] R. Zadeh, B. Ramsundar, "Chapter 4. Fully Connected Deep Networks" *TensorFlow for Deep Learning* Available at: `oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html`