HOW POSITION VS DIRECTION AFFECTS LEARNING TO PASS A BALL

Dr. Girard
cdgira@ship.edu
Shippensburg University

ABSTRACT
Spatial information can be difficult to represent to neural networks. This is because as location values increase in value this triggers a stronger response from the neural network, such as 5,5 going to 10,10. However, in many cases this is not the desired effect. This research builds on previous work to see how the way spatial information is presented affects the accuracy of the neural network. This research looks at using an abbreviated model of representing (x,y) positions verses using direction combined with distance. The problem environment is the passing of an object to a moving target. The overall accuracy of each model will be tested and compared to one another.

KEY WORDS
Neural Network, Spatial Location

1 Introduction

Neural networks learn by updating weights such that they can correctly predict the output from a given input. They try to find patterns, so they perform best where inputs do not generate conflicting outputs [1,3,7,8,12]. The design of the neural network has three key aspects: how many nodes in each layer, how the nodes are connected, and how values are computed. can vary greatly depending on the connection of the input to the output and one input to the next [1,3,7,8]. This paper will explore designs to represent spatial information to a neural network.

2 Background

Because the value being computed has no bearing on previous inputs, the work here uses a standard feed forward neural network. A simple feed forward network can be seen in Figure 1. In Figure 1 there are 8 nodes: i1, i2, h1, h2, o1, o2, b1, and b2. The i1 and i2 nodes are the input nodes for the neural network and form what is called the input layer. The input layer will take in numerical values, usually within a range of 0 to 1, that represent the situation being evaluated [1,3,6,7,8,9].

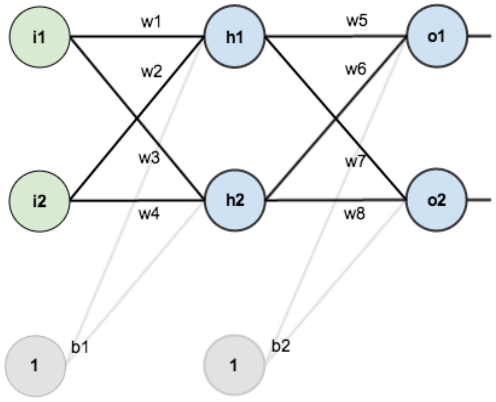

Figure 1: Connected Neural Network [11]

The h1 and h2 nodes are the hidden nodes for this neural network and form what is called the hidden layer. Each hidden node in figure 1 gets input from both input nodes. As such the input layer and hidden layer are considered fully connected. This is not required, but is common in many feed forward neural networks. Additionally, h1 and h2 are connected to one of the bias nodes, b1. While the models in this paper only used one hidden layer, it is possible to have additional hidden layers [1,3,6,7,8,9]. The o1 and o2 nodes are the output nodes for this neural network and form what is called the output layer. Each output node in figure 1 gets input from both the hidden nodes. Additionally, the output nodes are connected to the other bias node, b2 [1,3,6,7,8,9].

The values for the bias nodes, b1 and b2, are set at the start and do not change. These values are normally used to help the neural network overcome an expected threshold. For example, if the input nodes create a large combined positive value, the bias node could be set to a negative value to adjust this value back down [3,5,9,11]. The models used in this study did not have any bias nodes.

2.1 Computing Output Value

The process of providing input values and computing the output values in a neural network is called forward propagation. This starts by first setting the values for the input nodes. [3,6]. From there, each node in the first hidden layer will sum the value of each input node times the

weight of the connection. For example, in Figure 1 the node h1 would sum the value of i1*w1 + i2*w3. If there are any bias nodes that value is included in the sum as well. So, for node h1 it would also add the value from node b1 to its summation. After the summation step a node may apply an activation function [3,6,9,10,11].

The role of the activation function is to constrain the values produced by a node. Sigmoid, Tanh, and ReLU are commonly used for this role [3,6,9,11,17]. Both Sigmoid and Tanh were tested with the model, but found both to be less reliable than a version of ReLU. As such only ReLU is focused on in this paper.

$$(1)\quad ReLU(x) = \begin{cases} x \; if \; x > 0 \\ 0 \; if \; x \leq 0 \end{cases}$$

$$(2)\quad Leaky\; ReLU(x) = \begin{cases} x \; if \; x > 0 \\ 0.1 * x \; if \; x \leq 0 \end{cases}$$

$$(3)\; ACT(x) = \begin{cases} \max(2, 1 + (x - 1) * 0.01 \text{ if } x > 1 \\ x \; if \; -1 \leq x \leq 1 \\ \min(-2, -1 + (x + 1) * 0.01 \text{ if } x < 1 \end{cases}$$

The basic ReLU activation function, equation 1, converts any value less than 0 to 0, and leaves unchanged any value greater than 0. However, having a hard cutoff can have unintended effects. This is where Leaky ReLU, see equation 2, comes in. This version of Leaky ReLU allows for slow growth in values when the input is less than 0 [6,18]. This paper uses a specialized version of Leaky ReLU, see equation 3. This allowed for both positive and negative values, no hard cutoff, but also no growth to infinity.

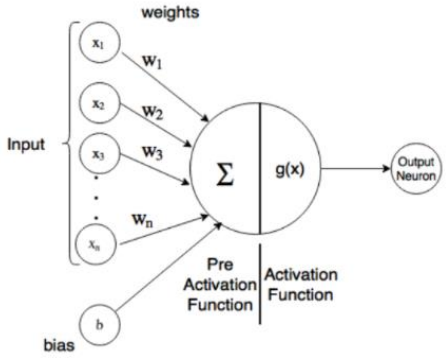

Figure 2 - An Artificial Neuron [9]

The full forward propagation process for a node is summarized in Figure 2. The process continues until it reaches the output layer. When the output layer's neurons have been evaluated, a decision will be made based on the result. This process happens every time a new input is given to the network, and everything is computed again [3,6,9,10,11].

2.2 Learning Process

The weights of a neural network are usually set to random values between -1 and 1. Because of this, the initial output of a neural network usually does not match the expected output. For this to occur the weights are adjusted using a process called back propagation. This process starts at the output layer and works its way backward through the layers as it self-evaluates. Back propagation starts by first determining the amount of error, see equation 4, with each output node [4,11,14,15].

$$(4)\; E = output - target \; [11]$$

The connected weights to that output neuron are then adjusted, see equation 6, based on the activation function used, the learning rate, and the amount it contributed to the output value. For the specialized version of Leaky ReLU equation 5 computes the rate of change that is occurring based on the value of output. Additionally, equation 6 assumes no use of a bias value.

$$(5)\; adjOut = \begin{cases} 0 \; if \; output \leq -2 \; or \; output \geq 2 \\ 1 \; if \; -1 \leq output \leq 1 \\ 0.1 \; all \; other \; conditions \end{cases} \quad [11]$$

$$(6)\; w_i = w_i + \eta * in_j * adjOut * E \; [11]$$

In back propagation, for equation 5, *output* is initially the result produced by an output node. Later it will be the value for a hidden node. For equation 6 it is updating the weight, w_i, for the connection to a node in the previous layer. The value in_j is the output value from the node in the previous layer. The value η is the learning rate value. The learning rate dictates how much of an impact each learning iteration will have [4,11,14,15]. In the first phase of back propagation this process is applied to all connections between all hidden layer nodes and output layer nodes. In Figure 1 this would mean updating w5, w6, w7, and w8.

$$(7)\; E_{total} = \sum_m^1 (output_n - target_n) \; [11]$$

The total error is used when updating the weights between the hidden layer and the input layer or between hidden layers. In equation 7 the total error, E_{total}, is computed by summing the error for each output neuron. Then, the weights are adjusted that connect these layers the same as with the weights between the final hidden layer and the output layer by substituting E_{total} for E in equation 6 [4,11,14,15]. For example, E_{total} would be used to update w1, w2, w3, and w4 in Figure 1.

With back propagation, it can be applied to every cycle of forward propagation, or it can be applied after multiple forward propagations. In either case, this process is effectively repeated until the desired rate of error or improvement rate has been reached within the network [4,11,14,15].

2.2 Representing Location

Representing location in a neural network is a challenge. Due to how neural networks respond to values changing it is not possible to simply encode the coordinate locations directly [17]. There are a few approaches already in use: sinusoidal, clustering, grid, and abbreviated [1,2,12,13,17].

The sinusoidal encoder is based on work related to encoding the location on a rotating shaft using sin and cos. This approach allows for the location to be represented by two values that range between -1 and 1. It scales well with any dimension size and how the values change work well within the scope of a neural network [2,12,13]. However, it comes with the cost of converting to and from their sin and cos representations and as such will not be used here.

In the case of clustering, a fixed number of groups of related locations are created. An approach to encoding this data is to use one-hot encoding, where each group/location is represented by a single node. Overall, this approach works well when the total number of groups stays the same and the neural network does not need to intuit spatial information between two groups of points [2,16,19]. Because of the number of possible locations and need to intuit spatial information this approach will not be used.

Grid takes the approach of providing a node for every possible location, while also preserving spatial information between nodes. By representing all possible locations in the domain, it solves the spatial problem found in the cluster approach [1,17]. Unfortunately, unlike the sinusoidal approach, using a grid encoding does not scale well and as such will not be used here.

An abbreviated approach makes use of one-hot encoding to try and turn coordinate information into something that is better understood by the neural network. It takes a numerical value and then encodes the value in terms of ones, tends, hundreds, etc. The neural network would have a group of 10 nodes for the ones, each one representing a specific digit (0-9) and the same for the tens place, etc [17]. This approach provides the same information as grid, but with better scaling. Additionally, it doesn't require the conversion expense of sinusoidal. Because of these properties it will be one of the approaches used.

2.3 Direction Based Approach

All the approaches above focused on representing spatial information by raw location values. This paper also tries a directional approach to representing the information. Instead of representing the source and target as locations, everything will be represented using relative direction and distance. Two approaches will be used to represent direction: compass and vector.

In the case of the compass approach, there will be 9 total nodes used. Eight of the nodes are used to represent a direction on a compass: N, NE, E, SE, S, SW, W, and NW. If the object is between compass locations (e.g. NNW), then two nodes (e.g. N and NW for NNW) are used to represent direction. The ninth node is used to represent distance to the object.

For the vector approach, there will be three total nodes used. Two of the nodes will hold the x and y values for the unit vector representing the direction to object. The third node will then represent the distance to the object. For both the compass and unit vector approach all distances will be normalized so they fall within the range of 0 to 1.

3 Experiment Design

The neural network will learn how to pass a ball to a moving person called the receiver in a 10x10 and 50x50 sized grid. This is a similar testing approach to that taken in [17]. Two different sized areas are used to see if field size affects the performance of each approach.

The input to the neural network is broken up into three approaches: Location using abbreviated, Direction using compass, and Direction using vector. The speed of the ball and the receiver will be fixed and so are not provided as input. The neural network will then output the direction the ball should be kicked.

3.1 Hypotheses

Because passing a ball relies on computing a direction usually relative to the receiver it is assumed that providing the direction to the receiver reduces the required calculations. So, the first hypothesis is:

> Direction based (Compass and Vector) approaches will perform better than location only (Abbreviated) based approaches.

Additionally, from the work in [17] it appears that more data is not always better from the results of grid vs abbreviated. As such the second hypothesis is:

> Vector encoding approach will perform better than Compass encoding approach.

During testing of the different neural network designs the drawback of using total error to update the weights between the hidden and input layers came to light. For the output to hidden layer the amount of error for each weight is directly tied to a specific output node. However, for the input to hidden layer it is the combined error, which creates the chance the individual errors could be in conflict (one needs weights increased and the other weights decreased). As such, a Vector model is tested that only computes the x output of the unit vector and one that only computes the y output of the unit vector. The idea is this approach will be able to tune the neural network more accurately, creating one more hypothesis:

Vector Separated (one for X and one for Y) will perform better than Vector combined.

3.2 Setup Details

The constants in the system are the speed of the ball, speed of the receiver, and the position of the sending player, called the sender. The receiver's speed is always set to 1. The ball's speed is set to 2, so it is always faster than the receiver. The sender is always placed in the middle of the grid, at position 5,5 for the 10x10 grid and 25, 25 for the 50x50 grid. The ball is assumed kicked at time 0.

For the Abbreviated method there are a total of 42 input nodes for the 10x10: 20 for location of receiver, 20 for location of sender, and 2 for direction receiver is moving. For the 50x50 Abbreviated needs 66 input nodes: 32 for location of receiver, 32 for location of sender, and 2 for the direction receiver is moving. Because the options for tens place is just 0 to 5, we only need 6 nodes for the 10's encoding for the 50x50. The direction of the receiver only needs 2 nodes as it is represented using a unit vector, following the design from [17].

The Compass method has a total of 17 input nodes: 8 for direction to receiver, 8 for direction receiver is moving, and 1 for distance to receiver. Vector has 5 input nodes: 2 for direction to receiver, 2 for direction receiver is moving, and 1 for distance to receiver. For all methods, but Vector Separated, there are two output neurons that represent the unit vector direction to kick the ball. For Vector Separated there is just one output neuron that represents either the x or y value for the unit vector.

After some test runs the number of hidden nodes for all methods was set to 20, except for Vector Separated where each neural network only received 10 hidden nodes.

3.3 Examples

Figures 3 and 4 show examples of possible neural network responses. Both figures show the same scenarios with different results, figure 3 is the correct output and figure 4 is an incorrect output. In Figure 3 the ball is moving at a speed of 1, while in Figure 4 the ball is moving at a speed of 1.41. In the actual experiments the ball will always be moving at a speed of 2.

In Figures 3 and 4 the red dot represents the receiver, the green dot represents the sender, and the blue dot the ball. In Figure 3 the receiver is moving in the direction of 0,-1 (or north) from location 0,10 at a speed of 1. The ball is moving from location 5,5 in the direction of -1,0 (or west) at a speed of 1. The ball and sender meet at location 0,5 for a successful pass.

In Figure 4 the receiver is again moving in the direction of 0,-1 from location 0,10 at a speed of 1. The ball, however, is moving at a speed of 1.41 from location 5,5 in the direction of -0.707,-0.707 (or north-west). The ball ends up at location 0,0 while the sender is at location 0,5 and as such a failed pass.

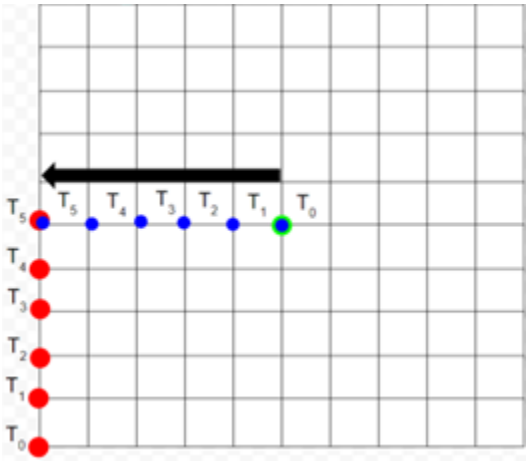

Figure 3 Correct example [17]

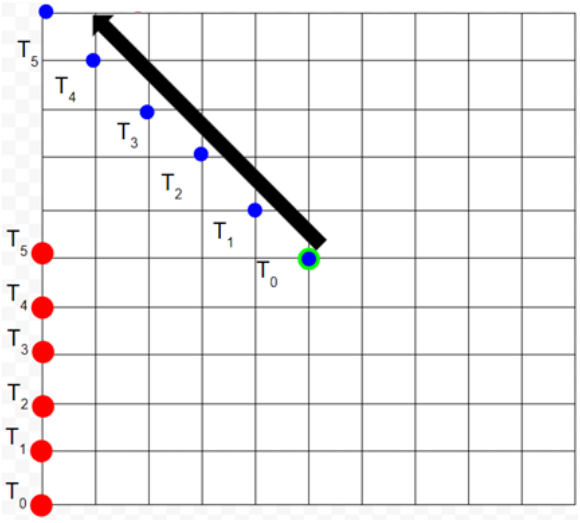

Figure 4 Incorrect example [17]

3.4 Training Data

Ten sets of training scenarios were generated for training and testing purposes. Each scenario would select a subset of all possible grid locations for the receiver to start from. No locations are selected when within 2.5 units of the sender. For the 10x10 area this equated to roughly half the total grid locations being selected. For the 50x50 area this equated to roughly 1/25th the total grid locations being selected.

Each subset of locations was selected at random in the following way. For the 50x50 it would loop over the x and y by steps of 5, but then randomly select a grid location near that spot. For example, for location 5,5 the location would be selected from 0,0 to 10,10. A similar approach was done for the 10x10, just with a smaller step rate. For the 10x10 there are roughly 38 locations in a subset, rather

than 94 if all valid grid locations had been used. For the 50x50 there are roughly 94 locations per subset, rather than the nearly 2495 if all valid grid locations were used.

For each grid location selected, 8 random directions are generated for the receiver to be moving. The 8 random directions are generated by selecting each compass direction (N, NE, E, SE, S, SW, W, or NW) and adjust it randomly by +/- 0.125 in the x and y direction. For example, if the receiver is moving north (0,-1) then it might get randomly changed to (0.1,-1.1) and then rescaled to a unit vector as (0.0905,-0.996). This works out to roughly 304 situations per training scenario for 10x10 and 752 situations per training scenario for 50x50.

4 Solution Description

The Python programming language was utilized to implement a neural network with one hidden layer. It did not utilize a bias and used the specialized ReLU discussed in section 2.1 as the activation function.

During testing it was found which random weights a neural network started with could occasionally affect its ability to learn successfully. As such, for each treatment a neural network was generated with random weights and then trained on 9 scenario sets and tested on the remaining set. Additionally, this was repeated 5 times for each treatment to ensure less chance of any treatment getting unlucky on the randomly generated weights. Then the best result of those 5 times was used as a data value for that treatment. This generated 10 data values for each representation method.

	10x10	50x50
Abbrev	X	X
Compass	X	X
Vec (C)	X	X
Vec (S)	X	X

Table 1: Block Design

In Table 1, Abbrev stands for the Abbreviated approach, Compass for the compass approach, Vec (C) for the Vector approach with two output nodes, and Vec (S) for the Vector Separated approach. This designation is used for all future tables as well.

$$(5)\ Err_X = \sum(|(X_n - X_{epc})|)$$
$$(6)\ Err_Y = \sum(|(Y_n - Y_{epc})|)$$
$$(7)\ Err_{total} = \sum(|(X_n - X_{epc})| + |(Y_n - Y_{epc})|)$$
$$(8)\ Acc = Err_{type}/m$$

The accuracy of the network is measured based on how far off the output unit vector was from the correct value using equations 5-8. X_n is the result for the X unit vector for test run n, same for Y_n with respect to Y. X_{epc} and Y_{epc} are the correct values for run n. In equation 8 the total error is then divided by the total number of examples tested, represented as m, to give the overall accuracy. The Err_{type} in equation 8 is either the result for Err_x, Err_y, or $Err_{totatal}$ depending on the accuracy being computed.

5 Results

Approach (10x10)	Avg Error Rise (Y)	Std Dev Rise (Y)	Avg Error Run (X)	Std Dev Run (X)
Abbrev	0.0270	0.00237	0.0265	0.00246
Compass	0.0142	0.000607	0.0150	0.00126
Vec (C)	0.00467	0.000573	0.00479	0.000473
Vec (S)	0.00285	0.000585	0.00280	0.000582

Table 2: Accuracy Results – 10x10

Approach A	Approach B	Rise (Y)	Run (X)
Abbrev	Compass	15.7	12.5
Abbrev	Vec (C)	27.5	26.0
Abbrev	Vec (S)	29.7	28.1
Compass	Vec (C)	34.2	22.8
Compass	Vec (S)	40.4	26.4
Vec (C)	Vec (S)	6.67	7.96

Table 3: T-test Results – 10x10

Based on the T-scores we can say with confidence that Vec (S) performed the best, then Vec (C), followed by Compass, and lastly Abbrev. Just providing the directional information allowed Compass to reduce the error of Abbrev by roughly 50%. Additionally, simplifying the direction information to just a unit vector allowed Vec (C) to reduce the error of Compass by roughly 66%. Lastly, splitting the neural network allowed Vec (S) to reduce the error of Vec (C) by roughly 40%.

Approach (50x50)	Avg Error Rise (Y)	Std Dev Rise (Y)	Avg Error Run (X)	Std Dev Run (X)
Abbrev	0.0268	0.00180	0.0268	0.00157
Compass	0.0116	0.000632	0.0117	0.000686
Vec (C)	0.00444	0.000224	0.00449	0.000285
Vec (S)	0.00284	0.000549	0.00315	0.000796

Table 4: Accuracy Results – 50x50

Approach A	Approach B	Rise (Y)	Run (X)
Abbrev	Compass	23.9	26.5
Abbrev	Vec (C)	37.1	42.1
Abbrev	Vec (S)	38.3	40.4
Compass	Vec (C)	32.1	29.2
Compass	Vec (S)	31.4	24.5
Vec (C)	Vec (S)	8.10	4.76

Table 5: T-test Results – 50x50

For the 50x50 test area we see the same ordering as from the 10x10. Compass appears to have improved in relative accuracy to Abbrev, Vec (C), and Vec (S) in the larger area. There was little to no change in relative accuracy for any of the other approaches with the increase in test area size.

6 Conclusion

First, what would be considered a successful pass. Since the error is for a unit vector our worst error was 2.7% and our best result was 0.28%. Distance to the receiver does matter here, so if the receiver was 100 yards away and the sender was off by 2.7% then the ball could end up about 3.8 yards off. At 0.28% error the ball would be less than ½ a yard away from the receiver. In the case of 2.7% error the receiver could adjust their speed to make sure the pass was successful. As such, it would be reasonable to assume any of the approaches would be functional in settings where the receiver can adjust for small amounts of error.

Based on the results it can be concluded that all hypotheses held. This builds on the work from [17] that the fewer nodes needed to represent the information the better. It also opens up the question of is it better to have one neural network with multiple output nodes or multiple neural networks each with a single output node. Lastly, while the ReLU function used here worked well, it still has the 0 gradient problem found in the basic ReLU function. The standard Leaky ReLU was tested for this project, but not used here because growing to infinity did impact learning during testing. This is not an issue for Sigmoid and Tanh. It would be interesting to create a version of ReLU that tried to mimic Sigmoid and Tanh more closely. For example on the edges it could still have a very tiny gradient, but no actual change in max or min value.

References:

[1] H. Bergkvist, P. Davidsson, and P. Exner, Positioning with Map Matching using Deep Neural Networks, *MobiQuitous 2020 - 17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2020.

[2] Klemmer, Konstantin, Nathan Safir, Daniel B. Neill, Positional Encoder Graph Neural Networks for Geographic Data, *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, 2023.

[3] Perry, Steven J., Create an artificial neural network using the Neuroph Java framework, *IBM Developer*. Jan 8, 2018. [Online]. Available: https://developer.ibm.com/tutorials/cc-artificial-neural-networks-neuroph-machine-learning

[4] J. Deus, Implementing an Artificial Neural Network in Pure Java (No external dependencies)., *Medium*, 01-Sep-2020. [Online]. Available: https://medium.com/coinmonks/implementing-an-artificial-neural-network-in-pure-java-no-external-dependencies-975749a3811

[5] Lukasz Gebel, Why We Need Bias in Neural Networks, Medium, 23-Jan-2022. [Online]. Available: https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98

[6] V. Gupta, Understanding Feedforward Neural Networks, *LearnOpenCV*, 20-Apr-2021. [Online]. Available: https://learnopencv.com/understanding-feedforward-neural-networks/

[7] M. Ibrahim, M. Louie, C. Modarres, and J. Paisley, Global Explanations of Neural Networks, *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 2019.

[8] D. M. F. Izidio, A. P. D. A. Ferreira, and E. N. D. S. Barros, Towards better generalization in WLAN positioning systems with genetic algorithms and neural networks, *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019.

[9] A. Malhotra, Tutorial on Feedforward Neural Network - Part 1, *Medium*, 02-Feb-2018. [Online]. Available: https://medium.com/@akankshamalhotra24/tutorial-on-feedforward-neural-network-part-1-659eeff574c3

[10] S. Mall and S. Chakraverty, Multi Layer Versus Functional Link Single Layer Neural Network for Solving Nonlinear Singular Initial Value Problems, *Proceedings of the Third International Symposium on Women in Computing and Informatics - WCI '15*, 2015.

[11] Mazur, A Step by Step Backpropagation Example, Matt Mazur, 15-Feb-2022. [Online]. Available: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[12] Mai, Gengchen, Krzysztof Janowicz, Bo Yan, Rui Zhu, Ling Cai, and Ni Lao, Multi-Scale Representation Learning for Spatial Feature Distributions Using Grid Cells, *ICLR 2020*, 2020.

[13] Burke, J., J. F. Moynihan, K. Unterkofler, Extraction of High Resolution Position Information From Sinusoidal Encoders, [Online]. Available:

https://www.analog.com/media/en/technical-documentation/technical-articles/452913422000_sin_encoder.pdf

[14] V. Parmar, Blog: Building a simple neural net in Java, *SmallData*. [Online]. Available: https://smalldata.tech/blog/2016/05/03/building-a-simple-neural-net-in-java

[15] M. J. Piovoso and A. J. Owens, Neural network process control, *Proceedings of the conference on Analysis of neural network applications - ANNA '91*, 1991.

[16] Tokuyama, Yusuke, Ryo Miki, Yukinobu Fukushima, Yuya Tarutani, and Tokumi Yokohira, Performance Evaluation of Feature Encoding Methods in Network Traffic Prediction Using Recurrent Neural Networks, *ICIET*, 2020.

[17] Lewis, Joshua and C. Dudley Girard, Position Information with Neural Networks, *38th Annual Conference of PACISE*, 2023.

[18] P. Baheti, "Activation Functions in Neural Networks [12 Types & Use Cases]," v7 Labs, Feb. 02, 2023. https://www.v7labs.com/blog/neural-networks-activation-functions

[19] Shaikh, Rahil, Choosing the right Encoding method-Label vs OneHot Encoder, *Towards Data Science*. 2018.