Positional Neuron Layers

Jonah Shader

The fully connected layer is one of the most common layers found in neural networks today. In this layer, a neuron is not discretely represented, but is simply the name given to a collection of weights, a bias, and an activation function. This is often visually depicted as ordered columns of neurons which are fully connected to the previous layer; however, their graphical location bares no significance on the behavior of the layer. In this paper, position is introduced to neurons so that they exist in Euclidian space in such a way that impacts the forward pass, giving significance to their position. Position can be used to encourage strong weights between close neurons and discourage strong weights between distant neurons, leading to neural networks that can be spatially interpretable. In addition, it was shown experimentally that these models perform better than the non-positional equivalent when pruning is applied.

I. Introduction

In biological neural networks, neurons must exist in physical space and form connections with each other over a physical medium. If these networks formed connections without respect to location, we would have brains that appear structureless since every neuron would have arbitrary connections. However, we do not observe this but instead find structure within biological neural networks. When looking at one of the most common artificial neural network layer types, the fully connected layer, one cannot determine meaningful structure from their graphical depictions alone. This is because their location bares no significance on the behavior of the layer but is merely a location chosen to minimize visual clutter. Any neuron can form connections with any neurons from the previous layer. This paper explores the idea of re-introducing this spatial coherence that was lost when artificial neural networks were created.

With this model, we introduce a vector representing position to each neuron. This vector can be of any size, but in this paper, we only explore vectors representing two dimensions for the sake of visualization. With these position vectors, we can incorporate the distance between neurons to encourage strong weights between close neurons and encourage weak weights between distant neurons. The hope is that this leads to cluster-like structures forming within networks that collectively perform a function that is relatively isolated from the rest of the model. In addition, input and output layers can be fixed to represent the spatial properties of the

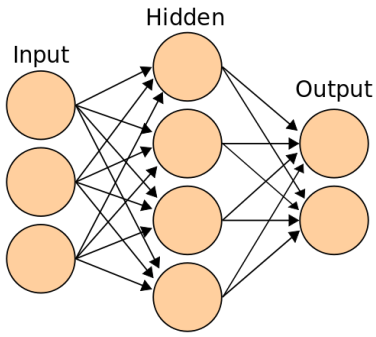

Figure 1: A standard graphical depiction of a densely connected neural network

dataset. For example, the input layer for an image classification model could be initialized and frozen to a grid pattern that matches the resolution of the images. This encourages neurons in subsequent layers to make strong connections to regions of the image closest to the neuron while minimizing connections to other regions of the image. One potential benefit of this is the ability for a neuron to observe a different region of the image only by changing its position. In the non-positional feedforward equivalent, the weights connecting the neuron to the current region would have to shrink while weights connecting to the new region would have to grow. The movement of position is a higher-level abstraction on this behavior. This movement of position could occur at any layer in the model, which may encourage neurons near each other to perform similar tasks. One hope is that this leads to a model that can generalize better. However, experimental results show that this model does not generalize as well as the non-positional equivalent.

Lastly, this layer type seems to be fit for pruning. During training, some neurons end up far from any other cluster of neurons, and thus are encouraged to have small weights. In comparison, the neurons in non-positional fully connected layers could have more evenly distributed weights within one neuron. It is less common to have neurons that have exclusively small weights. When pruning networks, it is desirable to remove entire neurons instead of individual weights because removing neurons directly reduces

the size of the weight matrices, whereas removing a single weight does not. Experimental results show that this model outperforms the non-positional equivalent when pruning is applied.

II. Related Work

There does not appear to be any directly related work, but positional neuron layers can behave similarly to convolutional layers. Mainly, a kernel has a fixed size window that is applied to many regions of an image. These kernels are typically much smaller than the input size, so the region it sees is small. Similarly, a single neuron in a positional neuron layer only sees nearby neurons, so in a similar sense the region it sees is small. However, unlike the discrete size of a kernel, the region boundary for positional neurons is continuous.

III. Data

For the evaluation of this model, the MNIST handwritten dataset was used. This dataset consists of 60,000 training images and 10,000 test images. There are ten classes, one for each digit. An image is grayscale and 28x28 resolution [1]. MNIST was chosen for its familiarity and simplicity.

IV. Methods

Two ways of producing the desired behavior was explored. For both methods, we store the original weights but calculate effective weights for the forward pass. Effective weights are scaled by neuron distance applied to a falloff or bell curve function. The purpose of this function is to approach one as neurons get closer and to approach zero as they get further. The function we used is $b(x) = \frac{1}{1+x^2}$ where x is neuron distance, but other similar functions should also work. With our falloff function and neuron distance, we can compute the effective weight as $w_f(x, w) = b(x)w$ where w_f is the effective weight, w is the weight between two neurons, x is the distance between two neurons, and b is the bell curve/falloff function. This is how weights are affected by distance in the forward pass.

If we freeze the neuron positions, this effectively becomes a non-positional layer with different weight initialization because $b(x)$ never changes. It simply becomes a constant in $w_f(x, w) = b(x)w$ where w is the only trainable parameter. For the optimizer to obtain a strong effective weight between two neurons, it can either reduce the distance or increase the original weight. Fixing distance just leaves weight, reducing this to a non-positional layer. So, this alone does not produce the desired behavior. One solution to force distance to be trained is to use regularization on the original weights. Now, since the original weights are penalized for being large, the encouraged way of obtaining a large effective weight is by moving the neurons closer together.

An alternative way to achieve the desired behavior is by weighting the regularization by distance. This simply looks like $L_2(w, x) = wx^2$ where w is the original weight and x is the neuron distance. Using this as a penalty term (times a small constant) achieves a similar result and is combined with unweighted regularization in the experiments. These experiments were implemented using Julia [2] and the source code is available on GitHub [3].

V. Experiments

The metrics we are measuring are ability to generalize, ability to be pruned, and interpretability. To do this, we will compare a positional neuron model to the equivalent densely connected model. Both have layers of sizes [784, 112, 56, 50, 50, 50, 10]. They use the Adam optimizer [4] and the swish activation function [5]. For the positional model, the input layer's position is fixed as a grid proportional to the position of the image pixels. The output is a fixed radial spread of 10 outputs representing the 10 classes of the MNIST dataset (see figure 4). These need to be fixed in place because the current methods only discourage distance, therefore the model would reduce the distance of all neurons to zero without these constraints.

To explore the models' abilities to generalize, we trained the models on subsets of the training data and compared their accuracy on the test set. Ten different proportion levels were used from 0.015 to 0.15, with four trials per level averaged.

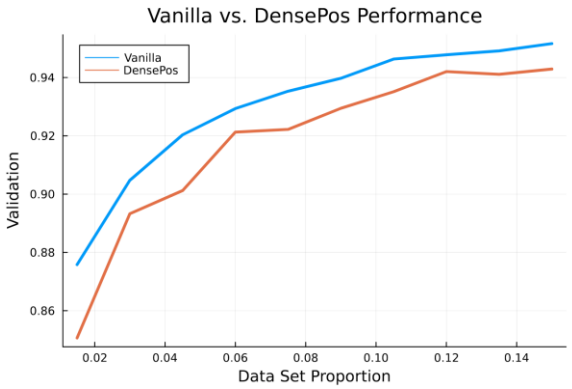

Figure 2: Vanilla (non-positional) vs DensePos (positional) performance over data set proportion indicates poor ability to generalize

Unfortunately, the positional model underperforms the non-positional equivalent in this experiment. Other experiments need to be performed with different hyperparameters to better understand this difference in performance.

Next, to explore the models' abilities to be pruned, we fixed the data set proportion to 0.25 and explored ten different proportion levels for pruning between 0.05 and 0.5. Neurons were pruned by sorting by mean squared weight and removing the smallest ones until the desired pruned proportion was achieved.

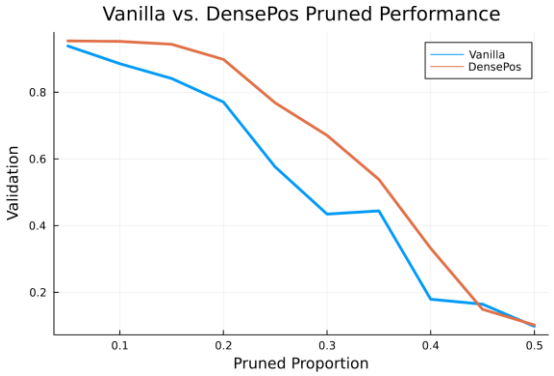

Figure 3: Vanilla (non-positional) vs DensePos (positional) validation over pruned proportion indicates good pruning performance

We found that the positional model outperformed the non-positional equivalent in nearly all evaluated proportions, with the largest difference in performance at 0.3 pruned proportion.

Lastly, we will look at interpretability. Several animations were produced displaying the position of the model's neurons during training, and we can see interesting behavior within the training period. Firstly, we see the first hidden layer moving towards the center of the input layer, roughly matching the distribution of activity seen in the images. Secondly, neurons in the last hidden layer can be seen approaching the 10 output neurons, demonstrating their relative influence on the outputs. Animations can be found in the GitHub repository.

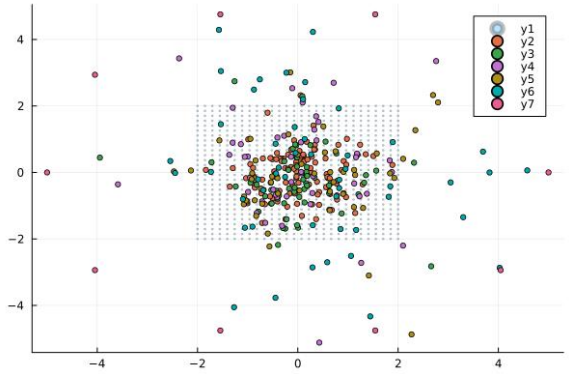

Figure 4: A screenshot of an animation of a positional model MNIST classifier being trained

VI. Conclusion

With the limited time spent exploring this concept, there appears to be a glimpse of potential. Ability to generalize was relatively poor, but the pruning performance was relatively strong. Pruning performance needs to be compared to the non-positional equivalent with regularization. This paper compared it to the equivalent *without* regularization, which may not be the correct comparison. Lots of other hyperparameters need to be explored in more detail, like model size, epochs, position vector dimensionality, etc. It is possible that this model could still generalize better under certain conditions.

One interesting idea to pursue is performing the opposite of pruning. Perhaps one could insert neurons into the model in random positions until they gain large weights and merge with existing clusters or form new clusters.

Another interesting idea is to somehow combine positional neuron layers with convolutional layers. Maybe a positional neuron layer could replace the kernel in a convolutional layer, so that the positional neuron layer is repeatedly applied as a moving window.

Works Cited

[Figure 1] https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg
[1] https://en.wikipedia.org/wiki/MNIST_database
[2] https://julialang.org/
[3] https://github.com/jonahshader/position_neurons_julia
[4] https://arxiv.org/abs/1412.6980
[5] https://arxiv.org/abs/1710.05941