A Re-Implementation of Weight Agnostic Neural Networks

Martin Pömsl, Julia Kaltenborn

Osnabrück University

A Report for the Course
Implementing Artificial Neural Networks with TensorFlow
by Luke Effenberger

1 Introduction

Artificial Neural Networks (ANNs) have proven successful for numerous tasks from the domains Computer Vision, Reinforcement Learning and Natural Language Processing. One of the broader aims of ANNs is to eventually perform well on tasks that are currently exclusively solvable by the natural neural networks of humans and other animals. ANNs still work very differently from natural neural networks. One major difference is that natural neural networks evolve over time by adjusting their connection strengths as well as their neural architecture. ANNs can adjust connections strengths via weights, but they cannot change their architecture. Another difference is that so-called presocial behaviour exists in nature, meaning that animals immediately after birth already have certain abilities such as swimming or escaping. Transferred to ANNs, that means that their neural architectures on their own encode inductive biases that enable them to perform well on certain tasks even without supervised training. An ANN that operates on such an neural architecture combined with gradient-based weight learning methods could lead to great advances in many fields.

The goal of Weight Agnostic Neural Networks (WANNs) is to find neural architectures that perform well for a given task independently from their connection weights. In the following we introduce the work of Gaier and Ha (2019), who search in a architecture space inspired by the neuroevolution algorithm NEAT, and present a re-implementation of their model. While Gaier and Ha (2019) focused mainly on the performance of WANNs in reinforcement learning tasks, we evaluate and tailor our WANN model to perform well on classification and regression tasks. At first, we explain the background of WANNs, then the basic algorithm of WANNs and our partly simplified version. Finally, we compare our results to an ANN implementation that has been exposed to a comparable number of samples. The code for all our experiments can be found at `https://github.com/mpoemsl/wann`.

2 Background

The main goals of WANNs are to [A] perform a neural architecture search for a certain task, [B] shrink the importance of weights in the training step and [C] find a minimal architecture that performs well in the given task.

[A] The idea of looking for an architecture rather than the optimal weights is not new at all. It is noteworthy that the general concept is even present in conventional ANNs, since certain blocks of an ANN are deployed for very specific tasks, such as convolutional layers for image processing (He et al., 2016; Ulyanov et al., 2018) or LSTM-layers for time series processing Hochreiter and Schmidhuber (1997). However, there exist many algorithms that search not only certain layers but a complete network topology (Maniezzo, 1994; Opitz and Shavlik, 1997; Pujol and Poli, 1998). Genetic algorithms such as NEAT (Stanley and Miikkulainen, 2002) have proven to be very helpful for navigating through such an architecture search space. While NEAT lookes for both architectures and weights for the ANN at the same time, WANNs focus only on the architecture search. Gaier and Ha (2019) proposed that a weight search still can take place after the networks architecture has been determined.

[B] Weights parameters do not need to be fixed, but can be sampled from distributions as Bayesian Neural Networks (BNN) (Krueger et al., 2017) and variance networks (Neklyudov et al., 2018) show. The aim for this networks is to learn the distributions parameters instead of the weight parameters themselves. Similarly, WANNs allow the use of random sampled weights, however, do not aim to learn the distribution's parameters. In a more extreme version WANNs also use single shared weight values, meaning that all weights of the complete network are equal.

[C] The goal of obtaining a minimal network structure that performs well for a given task is in line with the concept of Minimal Description Length (MDL) (Grünwald and Grunwald, 2007; Rissanen, 1978) in Algorithmic Information Theory (AIT). Essentially, MDL means that a smaller and shorter model is always preferable to a more complex model if the more complex model does not enhance the performance significantly. It has also been shown that network pruning approaches (Frankle and Carbin, 2018; Lee et al., 2018) can simplify conventional ANNs by removing connections with small weights from the network. WANNs do not need to prune their network, because WANNs start with minimal complexity and only add complexity to the model if the more complex model outperforms the earlier simpler model. Thereby WANNs guard against missing minimal well-performing architectures.

It is clear that the concepts behind WANNs are not entirely new, but in conjunction they result in a model that can match the performance of conventional gradient-based learning while still leaving the option to improve the result even further with gradient-based methods.

3 Model Overview

In the following we present our re-implementation of WANNs as outlined by Gaier and Ha (2019). Conceptually, our code follows the algorithm of the original WANN. However, for the sake of parsimony we simplified some portions of the model. These modification as well as their benefits and drawbacks are also discussed.

3.1 Representation

We represent a single neural architecture (an `Individuum`) as an object with a set of node layers, a set of connection tables connecting all node layers and a set of activation functions, one for each node in each node layer. Specifically, we chose to represent layers as *Python dictionaries*, connection tables as *NumPy arrays* and all activation functions in one *NumPy array*.

The following shows the state of an `Individuum` directly after its creation.

```python
class Individuum():

    def __init__(self, ratio_enabled=0.4, n_inputs=784, n_outputs=10):
        """ Initializes node layers, connection tables and activations."""

        input_layer = {
            "id": 0,
            "table_ix": 0,
            "size": n_inputs,
            "src": None,
            "dst": 1,
            "pos": 0
        }

        output_layer = {
            "id": 1,
            "table_ix": None,
            "size": n_outputs,
            "src": 0,
            "dst": None,
            "pos": 1
        }

        main_connection_table = (np.random.random((input_layer["size"],
            output_layer["size"])) < ratio_enabled).astype(bool)

        self.layers = [input_layer, output_layer]
        self.connection_tables = [main_connection_table]
        self.activations = np.ones(output_layer["size"])
```

As can be seen, an `Individuum` starts its life with an input layer, an output layer and a single connection table that contains a boolean weight for each potential connections between each node in the input layer and each node in the output layer. A boolean value of `true` is taken to imply that the connection is enabled, while `false` indicates that it is not yet enabled or disabled.

3.2 Training

During training, the space of possible architectures is searched. A population of `Individuum`s is created, each of which represents a different WANN architecture. In each generation the population is evaluated and ranked based on a objective function that takes into account performance on the task as well as architecture complexity. Through repeated use of mutation and crossbreeding operators, the population is continuously modified and evaluated, thereby simulating an evolution process in which only the best performing architectures survive.

```python
def main(n_gen=5, dataset_name="mnist", **hyper):
    """ Main loop for the WANNs construction.
    Creates a population of WANNs and evolves
    it over several generations.

    Parameters:
    n_gen        -  (int) number of generations or
                    how often the evolution should take place
    dataset_name -  (string) name of the dataset for
                    which the WANN is constructed
    """
    # load data set
    X, y = load_dataset(dataset_name)
    hyper["n_inputs"], hyper["n_outputs"] = X.shape[1], y.shape[1]

    # initialize population
    population = init_population(**hyper)

    for gen in tqdm(range(n_gen)):

        # get data samples
        inputs, targets = sample_data(X, y, **hyper)

        # evaluate the performance of population
        eval_scores = evaluate_population(population, inputs, targets)

        # create new population based on evaluation
        population = evolve_population(population, eval_scores)

        # [...] parts of the code were excluded for readability
```

3.2.1 Evolution

In each generation, a new population is created based on the fitness ranking of the old population by breeding. Breeding means in this case that two `Individuums`, each chosen by tournament selection, contribute their architectures for the creation of a new individual. The unfittest of the population are not allowed to take part in the breeding (*cull*), whereas the fittest pass on to the new population immediately and unchanged (*elite*).

During breeding, the hyper-parameter `prob_crossover` determines if a crossover of both parents architecture takes place or if the new `Individuum` merely mimics the architecture of the better parent. In the case of a crossover, the new WANN architecture adopts the activation functions and nodes of the fitter parent, but the connections are a mixture of both parents. In any case the resulting `Individuum` subsequently mutates in one of three ways:

1. A connection is added
2. A node is added
3. An activation function is changed

The new population is a collection of these new `Individuums` who build up a new and hopefully better performing generation.

3.2.2 Evaluation

Evaluating the population is the very basis for finding good architectures during the evolutionary part of the algorithm.

During evaluation, the performance of each `Individuum` of the current population is measured by feeding it with data input samples and comparing its prediction with the target samples. The network's performance is measured for different weights: If the parameter `weight_type` is set to `random`, each connection gets a random weight value, if it is set to `shared` all connections get a single shared weight value, retrieved from a list of weight values ($[2, 1, 0.5, +0.5, +1, +2]$). This specific list was also used and suggested by Gaier and Ha (2019).

However, not only the performance of a network architecture matters, but also its complexity. For this reason the fitness ranking of the different architectures is not only done according to their prediction performance, but also according to their complexity. The prediction's performance is measured via the mean loss and the min loss, whereas the architecture's complexity is measured via its number of connections. In 80% of the cases the mean loss and the number of connections are used to evaluate a population. The mean loss and the minimum loss are used in the remaining cases, as Gaier and Ha (2019) proposed to do.

An essential difference between our evaluation and the original evaluation by Gaier and Ha (2019) is that we use a naive ranking method to balance between mean loss and number of connection, respectively mean loss and minimum loss. The original implementation used Pareto dominance ranking based on Deb et al.

(2002). For the sake of simplicity we used the following equations, where z_i is the resulting score, con_i the number of connections and $mean\_loss_i$ the loss averaged over all weight values for the `Individuum` i:

$$z_i = \tau \cdot \frac{con_i}{max(con)} + (1-\tau) \cdot \frac{mean\_loss_i}{max(mean\_loss)}$$

Respectively for the mean loss and the minimum loss of all weight value per `Individuum`:

$$z_i = \phi \cdot \frac{min\_loss_i}{max(min\_loss)} + (1-\phi) \cdot \frac{mean\_loss_i}{max(mean\_loss)}$$

Since in an evolutionary algorithm the objective function is maximized, we used $1 - z_i$ such that 0 is the worst value and 1 the best value an individuum can reach.

An additional divergence from the original WANN evaluation by Gaier and Ha (2019) is that we do not use several rollouts to determine an architectures performance for each weight. These rollouts are important for reinforcement learning tasks where different environments can lead to different performance values of the WANN. In the case of classification and regression tasks the results of the WANNs prediction are deterministic for a fixed weight and a set of inputs. Therefore we do only one rollout for classification and regression tasks, as Gaier and Ha (2019) also did for MNIST.

```
def evaluate_population(population, inputs, targets, prob_rank_n_cons=0.8,
tau=0.5, phi=0.5, weight_type="shared", loss_name="cce", **hyper):
    """ Evaluates a complete population via mean loss and either
    number of connections XOR minimum loss performance.

    Returns:
    (np.array) evaluation scores and statistics about the population.
    The higher the value, the fitter the individuum.
    """

    # gather weight values, losses and complexities
    # [...] code excluded here for readability

    mean_losses = losses.mean(axis=1)
    min_losses = losses.min(axis=1)

    # normalization
    normed_mean_losses = mean_losses / mean_losses.max()

    # scoring
    # in 0.8 of the cases ranking is done via
    # mean performance and number of connections
```

```
if np.random.random() <= prob_rank_n_cons:
    # 0 - best score / 1 - worst score (for tau == 0.5)
    normed_n_cons = n_cons / n_cons.max()
    scores = tau * normed_n_cons + (1 - tau) * normed_mean_losses
else:
# in 0.2 of the cases ranking is done via
# mean performance and max performance
    # 0 - best score / 1 - worst score (for phi == 0.5)
    normed_min_losses = min_losses / min_losses.max()
    scores = phi * normed_min_losses + (1 - phi) * normed_mean_losses

# invert the scores to get fitness scores:
# the higher the better! (maximization task)
return 1 - scores, stats
```

3.3 Testing

In the testing phase of the WANN algorithm, the best individuum of each generation is retrieved and its performance is measured. The best architecture is used for predicting on the test input samples and its output is compared with the test target samples. The WANN carries this prediction out for several weight values - either it uses different random weights or the single shared weight values that were defined earlier. The predictions for the different weights are averaged and returned as testing evaluation score for this individuum and stored for further visualization.

4 Experiments

We perform experiments on two models: The WANN re-implementation that we discussed in the previous section and a fully-connected ANN with one hidden layer.

4.1 Metrics

To ensure comparability, we make sure to expose both WANN and ANN to exactly the same number of observed samples (including duplicates). We also evaluate both on the same set of metrics:

- Train Loss
- Test Evaluation Score
- Number of Enabled Connections
- Number of Hidden Layers

4.2 Data

We evaluate WANN and ANN on two canonical tasks: The image classification task MNIST Handwritten Digits (LeCun et al., 2010) and the regression task Forest Fires (Cortez and Morais, 2007).

The goal of MNIST is to classify the intended digits based on downsampled 28 x 28 grayscale images of handwritten digits. In accordance with Gaier and Ha (2019), we preprocess the images by downsampling the images even further to 16 x 16, deskewing them and flattening the images. We evaluate this task in classification accuracy on an unseen test set and minimize the categorical crossentropy loss of the softmaxed predictions during training.

The goal of Forest Fires is to predict the size of burned areas of forest fires in Portugal based on meteorological and other data. Similar to Cortez and Morais (2007), we use only four variables as the basis for our prediction: Temperature, Relative Humidity, Wind and Rain. We normalize these values as well as the affected area values to be between 0 and 1 for training. Like Cortez and Morais (2007), we also evaluate this task in Mean Absolute Deviation or Mean Absolute Error (MAE) on the test set. On the training set we minimize the Mean Squared Error (MSE) loss.

4.3 Hyperparameters

While WANNs have many hyperparameters, there are three in particular that we chose to vary because of their effect on runtime, memory and performance.

- `pop_size`: The size of the population controls the degree of exploration, but is also directly correlated with runtime and memory consumption. We chose this based on the dataset size and memory we had at our disposal.
- `n_gen`: The number of generations controls the degree of optimization, but is also directly correlated with runtime. We chose this based on the runtime we were willing to allocate.
- `weight_type`: The type of weight (*random* weights or a single *shared* weight) has influence on a number of factors, not least robustness and performance. We tried both options in every setting.

5 Results and Discussion

5.1 WANN

5.1.1 Forest Fires

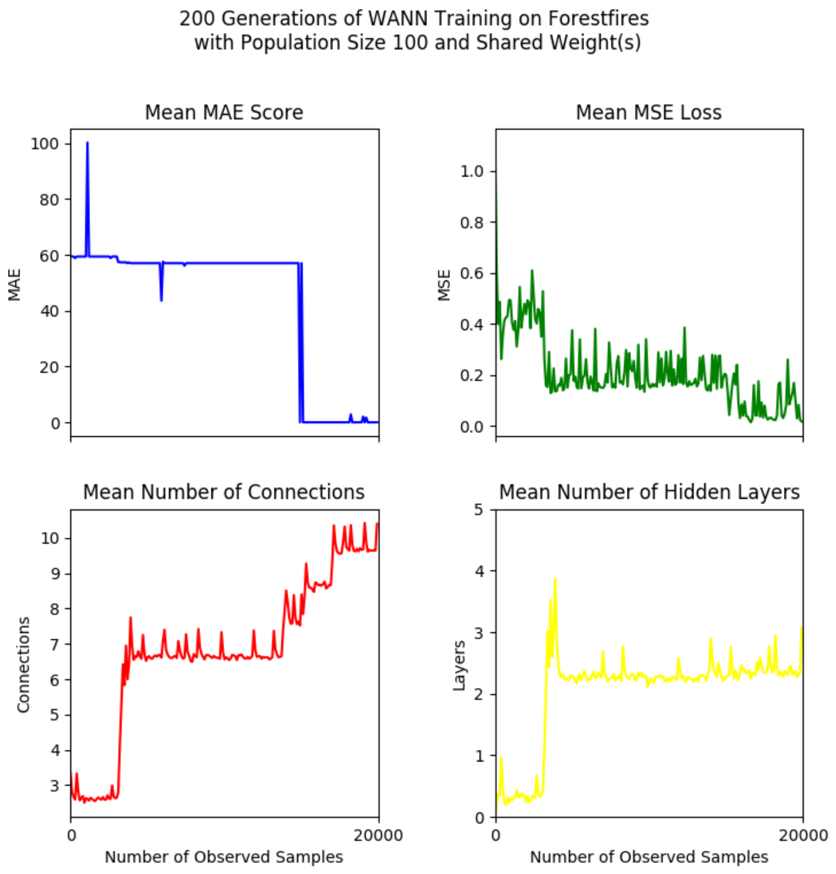

Figure 1: WANN on Forest Fires with a Shared Weight

As Figure 1 shows, our WANN implementation works well in the sense that the metrics exhibit the desired and expected trends. As the loss on the train dataset decreases (Mean MSE Loss), so does the evaluation score on the test dataset (Mean MAE Score). The fact that it reaches an error of 0.0 should not be given too much weight, since the dataset has many repeating values and we do not employ cross-validation. However, the general trend is definitely commendable. It is also noteworthy that the number of connections only increases when it is accompanied by an increase in performance, which is a desired property.

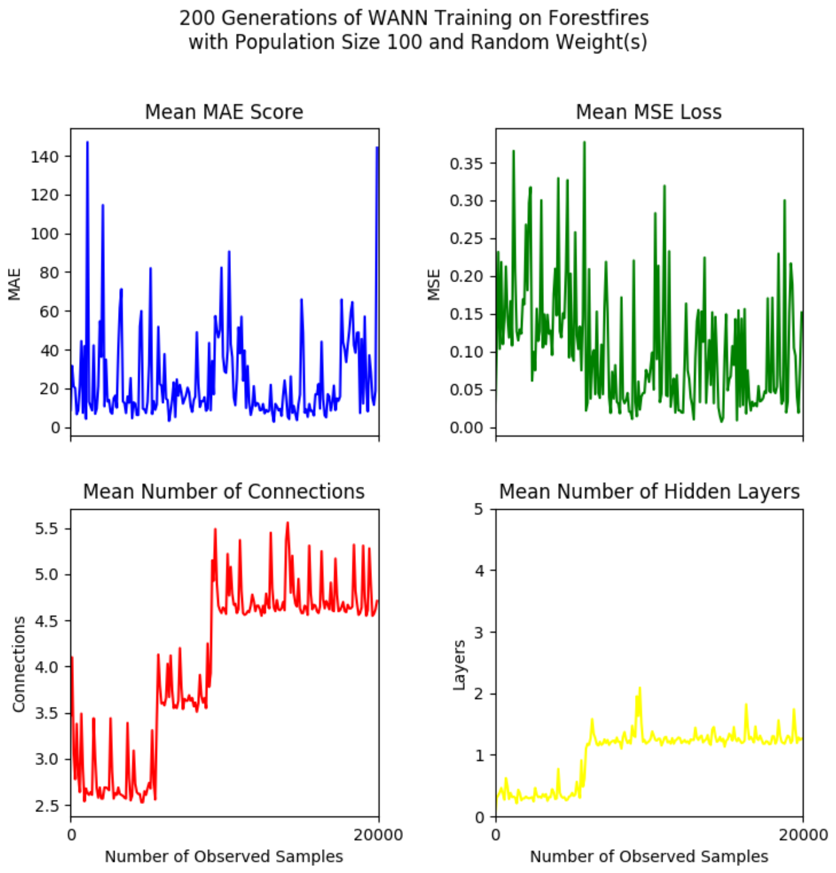

Figure 2: WANN on Forest Fires with Random Weights

As Figure 2 shows, the same experiment with random weights instead of a single shared weight is much less robust. The learning effect is also much less distinct.

5.1.2 MNIST

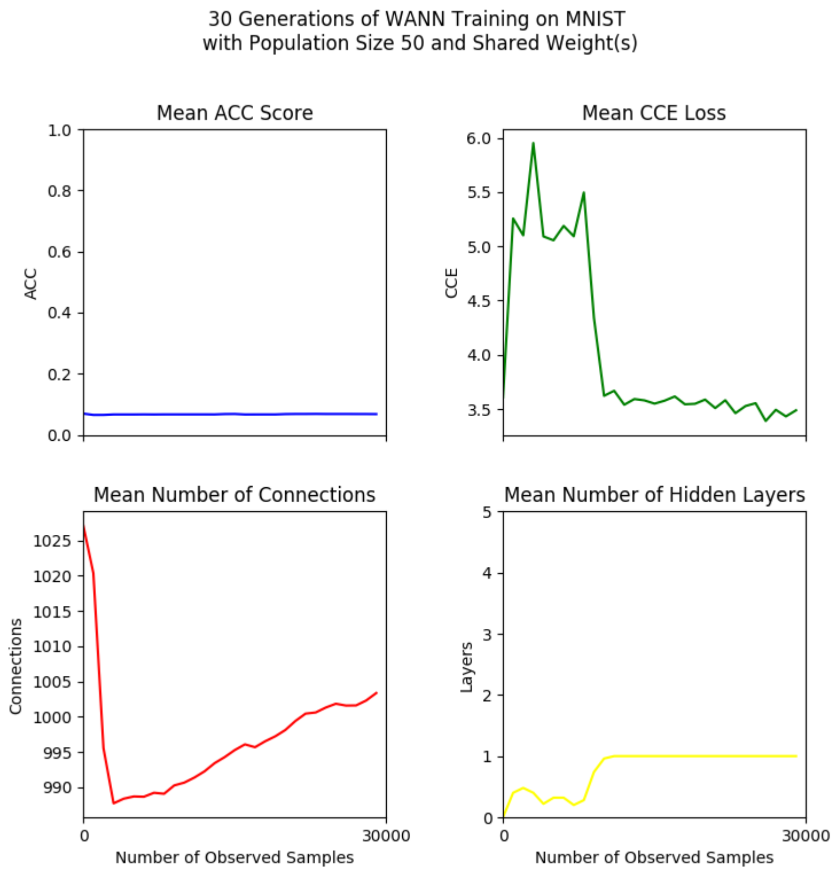

Figure 3: WANN on MNIST with a Shared Weight

On MNIST, we had some troubles due to the nature of our implementation and the limits of our computational resources. While our choice to represent all potential connections between two node layers with a NumPy array provided quick inference and easy handling, it turned out that pre-allocating memory for a huge number of potential connections puts a strain on RAM usage. Because of this, we had to restrict the number of layers for the each `Individuum` for the MNIST dataset. This might explain the medicore results observable in Figure 3 as well as the ceiling in the number of hidden layers. It should also be noted that runtime constraints prevented us from running the training for more than 30 000 observed samples, which is only half the MNIST dataset. In light of this, it is not surprising that this experiment achieved no significant accuracy.

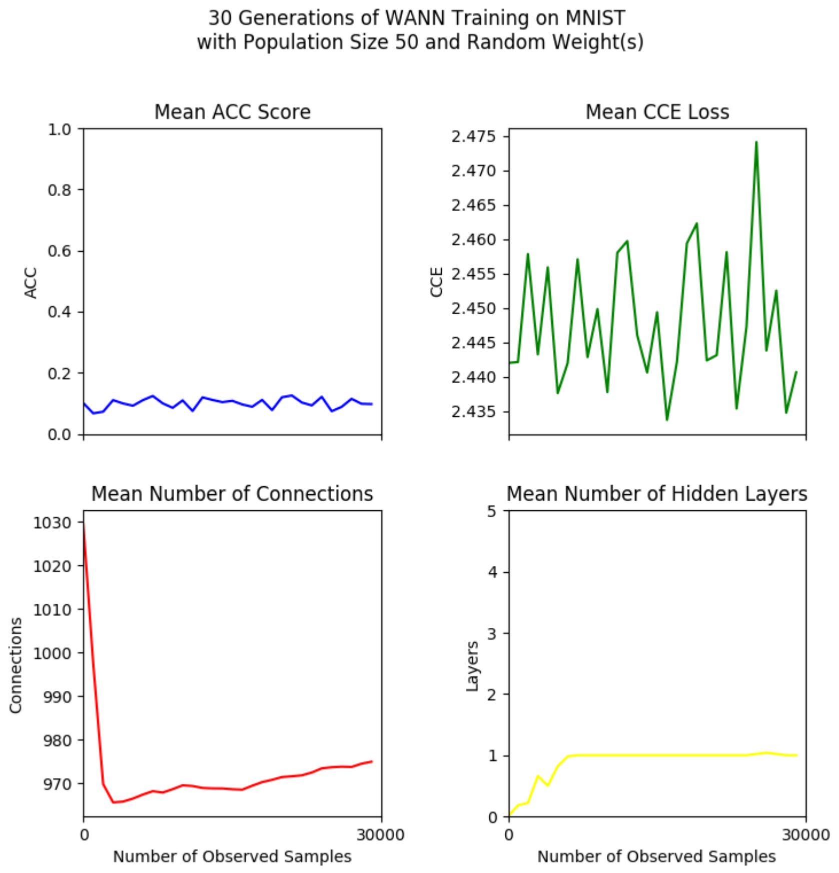

Figure 4: WANN on MNIST with Random Weights

Figure 4 shows that once again the random weights exhibit mostly the same trends as the shared weights, only in a much less robust fashion.

5.2 ANN

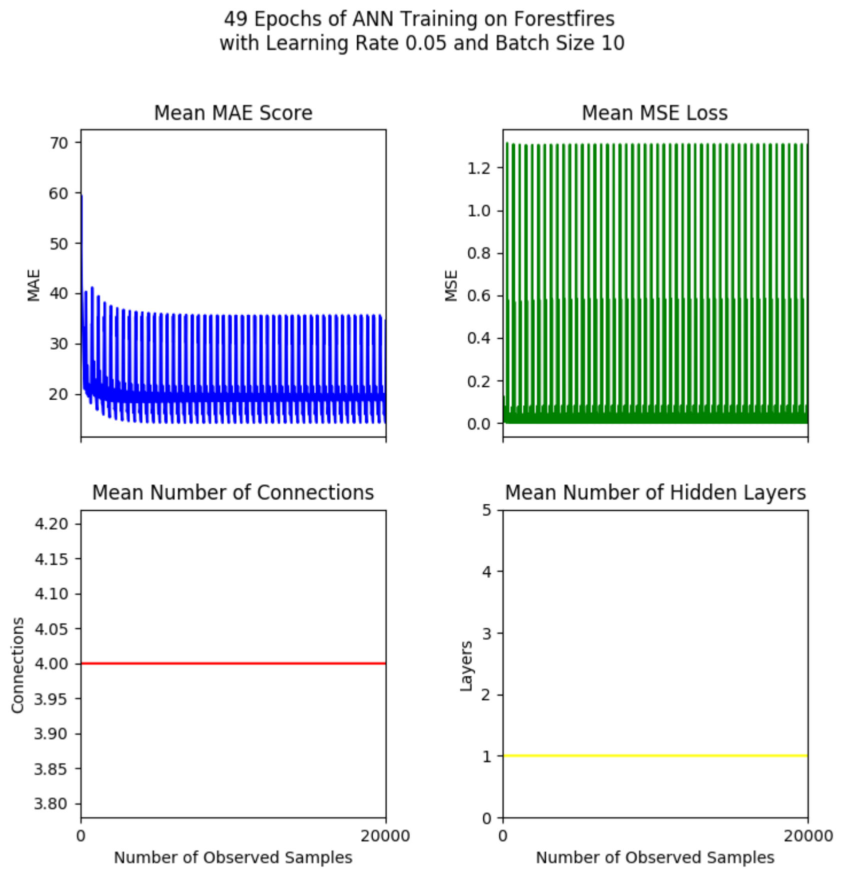

Figure 5: ANN on Forest Fires

On Forest Fires in Figure 5, the ANN exhibits the success that we are used to from conventional ANNs. The evaluation score on the test dataset (Mean MAE Score) is minimized successfully. The high degree of variance in test evaluation score and train loss may be attributed to the high learning rate and the small batch size, which were necessary to ensure that the model learns to predict well despite the low number of available samples. The number of connections and number of hidden layers stay constant, as usual for conventional ANNs.

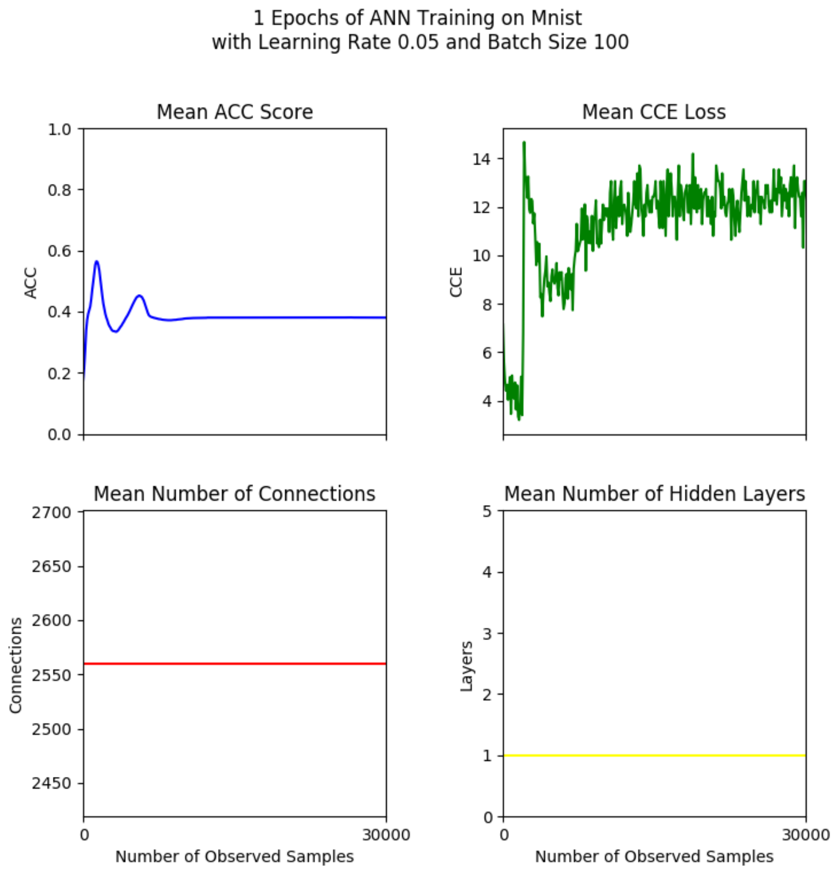

Figure 6: ANN on MNIST

On MNIST, we can see an unusual trend in Figure 6. The train loss appears to be rising and the test evaluation score (Mean ACC Score) is stagnating. This can be explained by the fact that for reasons of comparison, we show only metrics for only as many observed samples as our limited WANN was able to observe, which is only half the train dataset. It is also worth pointing out that even the value at which the accuracy seems to be stagnating is better than random in a evenly balanced digit classification task.

5.3 Comparison

It is clear that our WANN does not come close to the results reported by Gaier and Ha (2019) on MNIST, which is mostly due to the mentioned memory issues. However, on the task of regression learning, a WANN with a single shared weight is able to successfully minimize both train loss and test evaluation score and even comes into the ballpark of reported evaluation scores of other systems (Mean Absolute Deviation of 12.71 with a Support Vector Machine as reported by Cortez and Morais (2007)). However, the same goes for the ANN. Confirming the results reported by Gaier and Ha (2019), we can report that a single shared weight is more robust and achieves better performance than using random weights for connections.

6 Conclusion

In sum, our re-implementation of WANNs does not seem to be particularly cost-efficient in terms of computational resources, especially in comparison to conventional ANNs and other established machine learning methods. However, it is also worth pointing out that a lot of effort has gone into optimising these techniques, while WANNs are still a relatively new development.

Despite these drawbacks, WANNs can already achieve competitive results for tasks such as Forest Fires with low-dimensional data at the same number of observed samples. However, there are a lot of parameters and trade-offs to be explored, some of which we implemented in our model but were not able to investigate due to computational costs. Once this is done, maybe WANNs can be used systematically to improve the performance of neural networks in all domains.

References

Cortez, P. and A. d. J. R. Morais (2007). A data mining approach to predict forest fires using meteorological data.

Deb, K., A. Pratap, S. Agarwal, and T. Meyarivan (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation 6*(2), 182–197.

Frankle, J. and M. Carbin (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Gaier, A. and D. Ha (2019). Weight agnostic neural networks.

Grünwald, P. D. and A. Grunwald (2007). *The minimum description length principle.* MIT press.

He, K., Y. Wang, and J. Hopcroft (2016). A powerful generative model using random weights for the deep image representation. In *Advances in Neural Information Processing Systems*, pp. 631–639.

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

Krueger, D., C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville (2017). Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*.

LeCun, Y., C. Cortes, and C. Burges (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist 2*.

Lee, N., T. Ajanthan, and P. H. Torr (2018). Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*.

Maniezzo, V. (1994). Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on neural networks 5*(1), 39–53.

Neklyudov, K., D. Molchanov, A. Ashukha, and D. Vetrov (2018). Variance networks: When expectation does not meet your expectations. *arXiv preprint arXiv:1803.03764*.

Opitz, D. W. and J. W. Shavlik (1997). Connectionist theory refinement: Genetically searching the space of network topologies. *Journal of Artificial Intelligence Research 6*, 177–209.

Pujol, J. C. F. and R. Poli (1998). Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence 8*(1), 73–84.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica 14*(5), 465–471.

Stanley, K. O. and R. Miikkulainen (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation 10*(2), 99–127.

Ulyanov, D., A. Vedaldi, and V. Lempitsky (2018). Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9446–9454.