Training Artificial Neural Networks

Romerico Habla David Jr.
Towson University
Maryland, U.S.
rdavid6@students.towson.edu

Keywords—*Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNN), and Long Short Term Memory (LSTM).*

I. Background

There is active research in developing and improving intelligent systems capable of emulating human cognition and intelligence. One prominent area of study is the Artificial Neural Networks (ANN), a machine learning algorithm that mimics a neuron's biological processes, enabling computers to learn without being explicitly programmed. My research seeks to gain a theoretical understanding of ANNs with a particular emphasis on classification. I aim to understand theoretically and experimentally three applications of classification algorithms used in ANNs: forward pass and backpropagation (the foundational basis of all neural network types), convolutional, and recurrent neural networks. This paper focuses on understanding the mathematical concepts of these architectures and determining how to best tune the parameters within the experiments to improve the performance of each neural network.

Neurons, specialized biological cells responsible for processing information, are constructed of a cell body, dendrites, and an axon. The dendrites receive signals from other neurons while the cell body processes the signal and transmits it along the axon which branches into strands and sub-strands. At the terminals of these branching strands are the synapses. When a signal reaches the terminal of a synapse, neurotransmitters are released. Depending on the type of synapse, the neurotransmitter can either enhance or inhibit the neuron's tendency to emit electrical impulses. The effectiveness of the synapse adjusts according to the signals passing through them enabling the neuron to learn and adapt [1].

In this research, the ANN models will operate under supervised learning wherein the model is trained using a labeled data set where the target value is known. They consist of densely interconnected node layers that include an input layer, one or more hidden layers, and an output layer. The weight and bias parameters in the nural network are initialized to random values between 0 and 1 that effectively represent some arbitrary points within a weight-bias-loss function plane. During the forward pass, each node possesses an associated weight and bias that combine to determine the input for the next layer and so on. These weights can be either excitatory (positive value) or inhibitory (negative value) consequently increasing or decreasing the activation of a node/neuron [2]. Activation functions like Sigmoid, Rectified Linear Unit (ReLU), and the hyperbolic tangent functions that embed the data to specific ranges are necessary due to the model's imprecision when inaccurately emphasizing weights based on the range of the features it analyses thus assigning more importance to certain weights over others [3]. Similar to finding the line of best fit, the model adjusts these parameters iteratively to minimize the overall loss function, i.e., the Mean Squared Error, which is the sum of the differences between the true value and predicted value until it converges toward an optimal minimum [4]. This optimization strategy is known as gradient descent. The gradient represents the direction of the steepest ascent where the function is minimized as quickly as possible. In gradient descent, the negative value of the gradient is taken in order to minimize the function and find the steepest descent. The model iteratively propagates a data set backward through the network, calculating the gradients for each weight and bias, and updating them accordingly. While the gradients represent the direction of change needed to minimize the loss, the weights, and biases are scaled based on a predetermined learning rate. This process is referred to as backpropagation and is exactly how neural networks learn [5]. The ANN architecture finds extensive applicability across various fields because of its versatile applications within classification, categorization, regression, prediction, and optimization problems.

II. ANN Model and Results

An ANN model using the torch module in Jupyter Notebook is implemented using the digits datasets from the sklearn dataset package. This dataset contains 1,797 8 by 8 images of hand-written numbers between 0 to 9 which is split into 80% training data and 20% testing data. The activation function chosen for the model is the Sigmoid function. Lastly, the Mean Squared Error is used to calculate the overall loss function. The hyperparameters, learning rate, iterations, and hidden layer size are initialized to 0.1, 5000, and 32 respectively. The result shows 91% accuracy. These values are chosen as the benchmark for comparison with the subsequent experiments.

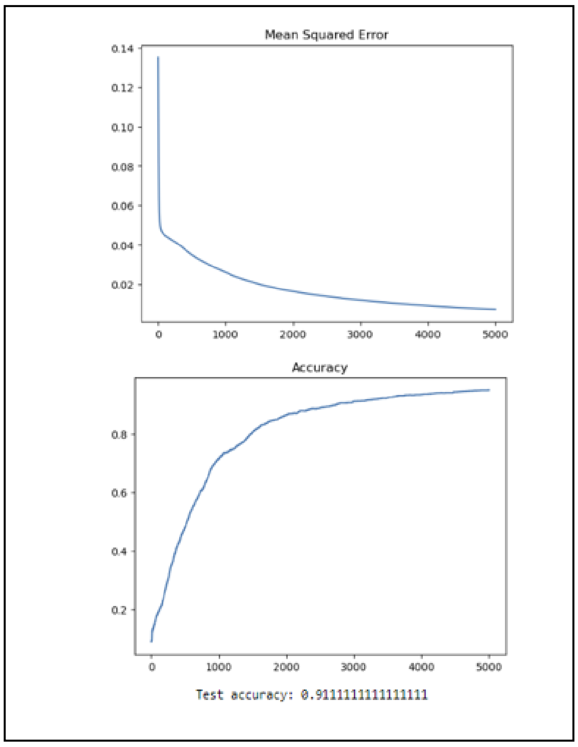


Fig. 1. Benchmark for subsequent experiments.

Afterward, the learning rate is set to 0.2 and keeps the other two hyperparameters the same. This increases the accuracy to 93%. I reverted the iterations and learning rate back then increased the iterations to 10000 and keeps the other two parameters constant. This increases the accuracy to 93%. Lastly, I reverted the hyperparameters and increased the learning size to 45. This decreased the accuracy to 85.9%. Conversely, I conducted experiments by decreasing these hyperparameters. The learning rate and iterations showed a linear relationship by decreasing the accuracy. However, decreasing the hidden size layer resulted in a decrease in accuracy. I observed that there was a positive correlation between the learning rate and iterations and the accuracy but the relationship between the hidden size layer and accuracy was inconclusive.

We conducted several more experiments by simultaneously adjusting two hyperparameters and all three. As expected, the learning rate and iterations would either increase or decrease the accuracy depending on whether they were increased or decreased but the observations became inconclusive when adjusting the hidden size layer.

However, I was alerted of the potential possibility of overfitting. Overfitting occurs when a model is excessively trained on a dataset which could lead to inaccurate predictions on new data. It is characterized by high training data accuracy but low testing data accuracy. Although the model did not show any signs of overfitting, increasing the learning rate increases the likelihood of overfitting. The learning rate should be kept at a small value to ensure a smoother convergence to the optimal minimum. Furthermore, when the dataset is incredibly large, increasing the iterations can pose challenges because the model becomes more computationally expensive. It is advised to run the data in batches for every epoch to help mitigate this issue.

III. CNN Architecture

Convolutional Neural Networks (CNNs) are composed of three main layers: the convolutional layer, the pooling layer, and the fully connected layer. The input passed to a CNN is an image represented as a matrix with a specific channel size, 1 for grayscale or 3 for RGB images.

Particularly, in the convolutional layer, a kernel or filter, which is a matrix of weights that captures a specific feature, slides in a window-like manner over the input image. At each stop, the kernel performs the dot product between its weights and the corresponding pixels of the input image. The dot product involves multiplying the weights with the pixel values, summing them up, dividing them by the total number of pixels in the feature, and then outputting the value to a new matrix called a feature map. The feature map represents the locations within the image where the features are most strongly matched. Positive values represent a match while negative values suggest no match. The magnitudes of the values represent the intensity of the detected feature. To ensure stability during training data, normalization techniques are applied. An activation function, often, ReLU, is applied where it changes all values that are negative to zero. This prevents the data from becoming unmanageably large. Convolution is the most important process in CNNs where a feature is taken and applied across every possible patch of a whole image. The next layer is the pooling layer, which is responsible for down sampling the feature maps and reducing their spatial dimensionality. The most common pooling method is max pooling where a 2 by 2 filter with stride of 2 slides across the feature map and returns the maximum value. The pooling layer reduces learning time, computation, the number of parameters, and the likelihood of overfitting. Pooling layers enable less amounts of data while maintaining the most essential features of the image [6].

These layers are usually stacked multiple times. The convolutional layers filter through a number of features, the ReLU function ensures everything is non-negative, and the pooling layers shrink the dimensionality of the channels. As the network progresses through these layers, the features are combined to form larger and more abstract features. When the channels reach the fully connected layer, they will be a highly abstract representation of the original image [7].

The fully connected layer follows a very similar structure to ANNs but with distinct features. The feature maps are concatenated and flattened before being passed as a 1D vector to the fully connected layer. This involves converting each pixel of the combined channels and converting them into a list of numbers that represent input vectors. Each input vector contains a receptive field, which is a crucial element in CNNs. Receptive fields are the portion of the input image that makes up the feature at a specific layer. They maximize the value of the input vector as high as possible. These maximized input vectors are passed to an input neuron through the forward pass process. Each input vector is multiplied by an associated weight and the results are summed up. An activation function is applied to this weighted sum. This process is similarly applied to other neurons of subsequent layers [7].

Each time a neuron connects to a neuron in a different layer, by some positive or negative weight, the receptive fields become increasingly complex due to the combination of features and formation of patterns. The final hidden layer connects to an output layer which is a list of the possible values expected to be generated by the network as a result of the receptive fields. Each of the output neurons has a vote that determines the classification of the image [7].

The model learns by using the backpropagation algorithm to optimize the overall loss function. The type of learning algorithm employed highly depends on the situation. The algorithm can range from an exhaustive exploration, a robust but computationally expensive algorithm, to gradient descent, a sensitive algorithm but highly efficient. During this process, a data set is propagated from the output layer of the fully connected layer to the first convolutional layer and utilizes its respective learning algorithm and some predetermined learning rate to adjust and update the parameters at each iteration. Initially, the neurons within the network are all fully connected to each other. After several thousands of iterations, the network learns which weights are most important for neurons. As a result, certain weights to neurons become negligible and are assigned to lower values or even zero thus effectively reducing the impact of these weights. This creates a network that is much more efficient and produces outputs closer to the target classification [7].

IV. CNN Model and Results

I implemented a CNN model using the torch module in Google Colab using the CIFAR10 dataset. The CIFAR10 dataset consists of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. There is a total of 60,000 32 by 32 images within the dataset that were split into five training batches and one test batch, each with 10,000 images.

The model utilizes four convolutional layers that were arbitrarily given 48 kernels for the first two layers and 96 kernels for the last two layers. The model implements a dropout function to replace the pooling layer. In each iteration, an element has a 50% probability of randomly being dropped out to prevent the model from becoming excessively large and the risk of overfitting. Following every two convolution layers and a dropout layer, a ReLU function is applied to make all the values within the channels nonnegative. Additionally, batch normalization is applied to each training batch to improve the model's computational productivity. Lastly, the model utilizes two fully connected layers to classify the input images. The model calculates the overall losses using the Cross-Entropy Loss function which measures the difference between the predicted probability distribution and true probability distribution of the target classes. During backpropagation, the model optimized the overall loss function using the Adaptive Moment Estimation (ADAM) learning algorithm, a variant of gradient descent with a learning rate of 0.001 [8].

In the first run, we trained the model for 10 epochs. The highest accuracy of 59.2% occurred at epoch 9 and the lowest accuracy of 42.4% occurred at epoch 1. This aligned with our expectations as we anticipated earlier epochs would yield lower accuracies while later epochs would show improvement.

To improve the performance of the model, we employed two primary techniques: image augmentation and grid search. The images were randomly transformed such as rotating them a few degrees, flipping them horizontally/vertically, cropping, brightness/hue shifts, zooming, and more. This new artificial data set would allow the model to generalize the training data resulting in more accurate predictions. This artificial dataset is used in a grid search algorithm that explores and evaluates various combinations of the number of kernels and nodes in the fully connected layers and stores the value of the hyperparameters that produced the most accurate results [8]. The hyperparameter values that achieved the best performance were 48 kernels for the first two convolutional layers, 96 for the last two convolutional layers, and 512 nodes for the fully connected layers.

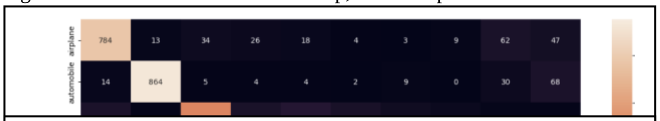
Finally, the model was configured to the values determined by the grid search algorithm, the learning rate was decreased to 0.0008 and the epochs were increased to 50. This resulted in an average testing accuracy of 76.3% and an average loss of 0.69 across all 10 classes. The following table shows the accuracy of each individual class:

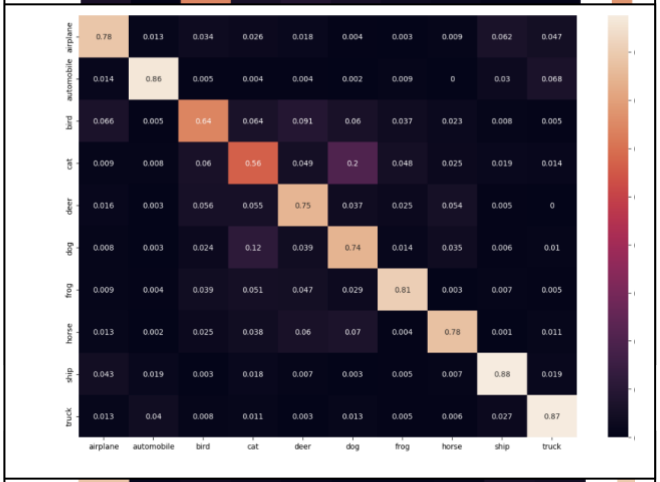
TABLE I. ACCURACY OF EACH 10 CLASSES

	Accuracy
Airplane	0.882
Automobile	0.828
Bird	0.548
Cat	0.458
Deer	0.654
Dog	0.750
Frog	0.579
Horse	0.889
Ship	0.733
Truck	0.833

Furthermore, I evaluated the model using three performance evaluation metrics, Confusion Matrix, Precision, and Recall. A Confusion Matrix is a performance measurement that has four components: true positive, true negative, false positive, and false negative. True positive (TP) is when the model predicts positive, and the target value is positive. True negative (TN) is when the model predicts negative, and the target value is negative. False positive (FP) is when the model predicts positive, but the target value is negative. False negative (FN) is when the model predicts negative, but the target value is positive. These components are used for measuring Precision and Recall. [9]. The following figures show the Confusion Matrix, Precision, and Recall heat maps:

Fig. 2. Confusion Matrix: 2D heat map, actual vs predicted values.





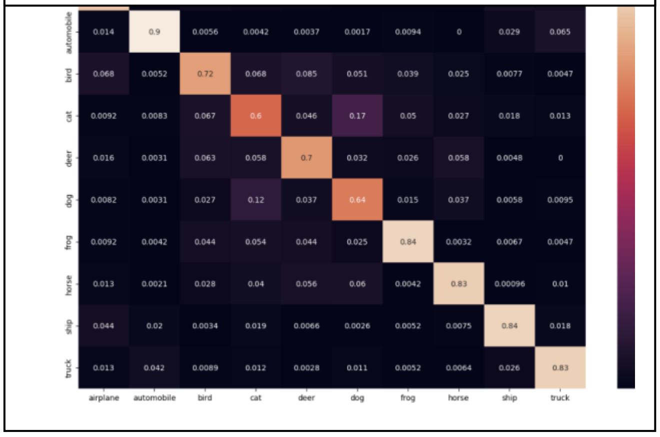


Fig. 3. Precision: Seeks to determine the proportion of identified positives that were correct. It is calculated by TP/TP+FP [10].

Fig. 4. Recall: Seeks to determine the proportion of all positives that were correct. It is calculated by TP/TP+FN [10].

Accuracy is the measure of how often the model predicts correctly. It is calculated by dividing the number of correct predictions by the total number of predictions. Accuracy can be misleading when the datasets are imbalanced, but, each class in CIFAR10 dataset contains 6000 images. However, there are outliers within the results that indicate an issue with the model. The Confusion Matrix provides a comparison between the "actual" and "predicted" values for each class. Cats and dogs were commonly mistaken for each other. Transportation vehicles were primarily confused with one another but would still yield high identifications and accuracies. Notably, deer and frogs were identified correctly more often than not but produced low accuracies. Precision provides information about the fraction of correctly identified positive cases. High precision means the model is good at avoiding FP but it does not provide information on FN. Recall measures the fraction of actual positive cases that are correctly identified. High recall indicates the model is good at identifying all real positive cases but does not address FP. Precision and Recall are inversely related meaning increasing one leads to a decrease in the other [10]. The values of the Recall and Precision heat maps are relative to each other indicating a good balance between the two metrics. Both metrics show similar correlations with the conclusions made from the Confusion Matrix heat map. The cats and dogs were mistaken for each other, and the transportation vehicles were confused with one another but produced high identifications and accuracies. However, the frog and deer produced lower accuracies but exhibit considerably higher Precision and Recall values.

V. Recurrent Neural Network and Long Short Term Memory (LSTM)

To learn how time sequential data can be classified, we study the recurrent neural networks and long short-term memory (LSTM).

VI. Conclusions

…

References

[1] A. K. Jain, Jianchang Mao and K. M. Mohiuddin, "Artificial neural networks: a tutorial," in Computer, vol. 29, no. 3, pp. 31-44, March 1996, doi: 10.1109/2.485891.

[2] Introduction to Artificial Neutral Networks | Set 1. (2018, January 21). GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-artificial-neutral-networks/

[3] Roy, R. (2022, November 17). Neural Networks: Forward pass and Backpropagation. Medium. https://towardsdatascience.com/neural-networks-forward-pass-and-backpropagation-be3b75a1cfcc

[4] Han, J., Kamber, M., & Pei, J. (2012). Data mining: Concepts and techniques, third edition (3rd ed.). Morgan Kaufmann Publishers.

[5] *What is Gradient Descent? | IBM*. (n.d.). Retrieved June 30, 2023, from https://www.ibm.com/topics/gradient-descent

[6] Kamali, K. (2023, January 24). Deep Learning (Part 3) - Convolutional neural networks (CNN) [Text]. Galaxy Training Network. https://training.galaxyproject.org/training-material/topics/statistics/tutorials/CNN/tutorial.html

[7] Brandon Rohrer. (2018, October 17). How convolutional neural networks work, in depth. https://www.youtube.com/watch?v=JB8T_zN7ZC0

[8] Flores, T. (2023, February 14). Intro to PyTorch 2: Convolutional Neural Networks. Medium. https://towardsdatascience.com/intro-to-pytorch-2-convolutional-neural-networks-487d8a35139a

[9] Narkhede, S. (2021, June 15). Understanding Confusion Matrix. Medium. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

[10] Classification: Precision and Recall | Machine Learning. (n.d.). Google for Developers. Retrieved July 12, 2023, from https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall