Handwriting digit recognition use ANN

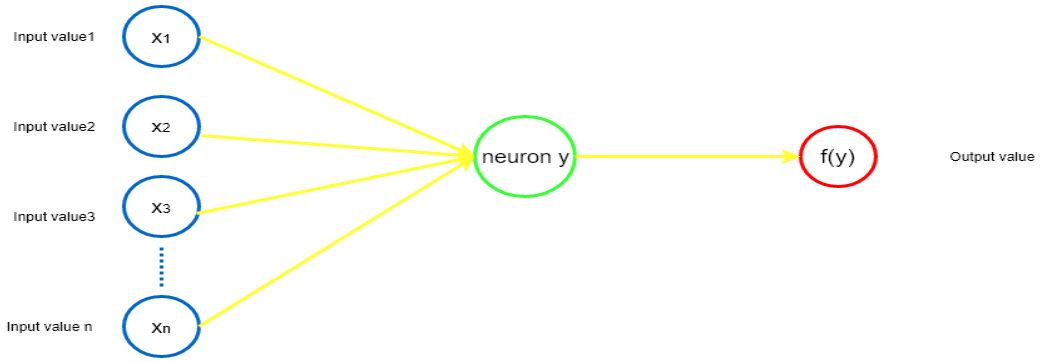
Introduction

Artificial intelligence is a branch of computer science that aims to create intelligent machines that can perform complex tasks that humans do. It has become an essential part of the technology industry. Some of the activities computers with artificial intelligence are designed for include, speech recognition, handwrite recognition, planning, problem-solving, self-driving cars. There are two types of machine learning, are unsupervised and supervised. The supervised machine learning it is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process, we know the correct answers, and the algorithm iteratively makes predictions on the training data and is corrected by the teacher. The learning process stops when the algorithm achieves an acceptable level of performance. [1] It's taken two decades for computer scientists to train and develop machines that can "see" the world around them—another example of an everyday skill humans take for granted yet one that is quite challenging to train a machine to do. Image recognition is considered important for computer technologies that can recognize people, animals, objects or other targeted subjects through the use of algorithms and machine learning The term "image recognition" is connected to "computer vision," which is an overarching label for the process of training computers to "see" like humans, and "image processing," the concepts Image recognition is done in many different ways, k-Nearest Neighbor (k-NN) or Artificial Neural Network (ANN). But many of the top techniques involve the use of artificial neural networks to filter images through a series of artificial neuron layers. The artificial neural network was specifically set up for image recognition and similar image processing. Artificial neural filters work on images to help machine learning programs get better at identifying the subject of the picture. Handwritten Digit Recognition is one of the most fundamental problems in designing practical recognition system. Immediate applications of the digit recognition techniques include postal mail sorting, automatically address reading and mail routing, bank check processing, etc. In this report will write a computer program implementing a neural network that learns to recognize handwritten digits. We're focusing on handwriting digit recognition because it's a simple prototype problem for learning about neural networks in general and it's also the first step to entering to a machine learning. In our work, we will use open source library the MINST training dataset which contains tens of thousands of scanned

images of handwritten digits, together with their correct classifications. The images are greyscale and 28 by 28 pixels in size. The MNIST data comes in two parts. The first part contains 60,000 images to be used as training data. The second part of the MNIST data set is 10,000 images to be used as test data. We'll use the test data to evaluate how well our neural network has learned to recognize digits. And TensorFlow is an open-source Python library developed by the Google Brain labs for deep learning research, you will take hand-drawn images of the numbers 0-9 and build and train a neural network to recognize and predict the correct label for the digit displayed

1-Artificial Neural Networks:

Artificial neural networks are one of the main algorithm used in machine learning. As the "neural" part of their name suggests, the inspired by biological neural networks which are intended to replicate the way that humans learn. Neural networks consist of input and output layers, as well as a hidden layer consisting of units that transform the input into something that the output layer can use. They are excellent algorithms for finding patterns which are far too complex or numerous for a human programmer to extract and teach the machine to recognize. Where neural networks have become a major part of artificial intelligence. This is due to the arrival of a technique called "backpropagation" which allows networks to adjust their hidden layers of neurons in situations where the outcome doesn't close to the truth value. Another important advance has been the arrival of deep learning neural networks, in which different layers of a multilayer network extract different features until it can recognize what it is looking for. The basic component of the neural network is a neuron, where the input signals are passed in from the input layer and the prediction comes out from the output layer. Each input position has a specific weight in the neuron, based on each weight, the neural network decides what information is important, and what isn't. When the neuron active it adding all the input multiplied by its corresponding connection weight but the neurons have always extra input the bias, this makes sure that even when all the inputs are none (all 0s) there's gonna be activation in the neuron. After computing its state, the neuron passes it through its activation function, which normalizes the result (normally between 0-1).



Backpropagation:

Backpropagation algorithms are a family of methods used to efficiently train artificial neural networks (ANNs) following a gradient descent approach that exploits the chain rule. The main feature of backpropagation is its iterative, recursive and efficient method for calculating the weights updates to improve the network until it is able to perform the task for which it is being trained. Backpropagation requires the derivatives of activation functions to be known at network design time. Automatic differentiation is a technique that can automatically and analytically provide the derivatives to the training algorithm. In the context of learning, backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function; backpropagation computes the gradient(s), whereas (stochastic) gradient descent uses the gradients for training the model (via optimization)[2].

2-The architecture of the neural network:

The architecture of the neural network is a set of connected neurons organized in layers: input layer and hidden layers and output layer. The input layer is for inputting values and the output layer is for the output values, hidden layer is used for all of the layers in between the input and output layers in our work we use three hidden layers. The network architecture code show in the picture below:

```
n_input = 784  # input layer (28x28 pixels)
n_hidden1 = 512  # 1st hidden layer
n_hidden2 = 256  # 2nd hidden layer
n_hidden3 = 128  # 3rd hidden layer
n_output = 10  # output layer (0-9 digits)
```

3-Build our network:

To build our network, we will set up the network as a computational graph for TensorFlow to execute. The core concept of TensorFlow is the tensor, a data structure similar to an array or list. Initialized, manipulated as they are passed through the graph, and updated through the learning process. The parameters that the network will update in the training process are the weight and bias values, so for these we need to set an initial value rather than an empty placeholder. These values are essentially

where the network does its learning, as they are used in the activation functions of the neurons, representing the strength of the connections between units.

We'll use random values from a truncated normal distribution for the weights. We want them to be close to zero

```
weights = {
    'w1': tf.Variable(tf.truncated_normal([n_input, n_hidden1], stddev=0.1)),
    'w2': tf.Variable(tf.truncated_normal([n_hidden1, n_hidden2], stddev=0.1)),
    'w3': tf.Variable(tf.truncated_normal([n_hidden2, n_hidden3], stddev=0.1)),
    'out': tf.Variable(tf.truncated_normal([n_hidden3, n_output], stddev=0.1)),
}
```

For the bias, we use a small constant value to ensure that the tensors activate in the initial stages and therefore contribute to the propagation.

```
biases = {
    'b1': tf.Variable(tf.constant(0.1, shape=[n_hidden1])),
    'b2': tf.Variable(tf.constant(0.1, shape=[n_hidden2])),
    'b3': tf.Variable(tf.constant(0.1, shape=[n_hidden3])),
    'out': tf.Variable(tf.constant(0.1, shape=[n_output]))
}
```

4-Forward propagation:

Forward propagation is the process of feeding the neural network with a set of inputs to get their dot product with their weights then feeding the latter to an activation function and comparing its numerical value to the actual output called “the ground truth”.

Now, to set up the layers of the network by defining the operations that will manipulate the tensors. Like in the picture below

```
# Feed forward
layer_1 = tf.add(tf.matmul(X, weights['w1']), biases['b1'])
layer_2 = tf.add(tf.matmul(layer_1, weights['w2']), biases['b2'])
layer_3 = tf.add(tf.matmul(layer_2, weights['w3']), biases['b3'])
output_layer = tf.matmul(layer_3, weights['out']) + biases['out']
```

Each hidden layer will execute matrix multiplication on the previous layer's outputs and the current layer's weights, and add the bias to these values.

5-Cross entropy error :

The final step in building the graph is to define the loss function that we want to optimize. A popular choice of loss function in TensorFlow programs is cross-entropy, also known as log-loss, which quantifies the difference between two probability distributions (the predictions and the labels). A perfect classification would result in a cross-entropy of 0, with the loss completely minimized. And it defines by relation below

$$- \frac{1}{m} \sum_{i=1}^{m} (1-y_i)\ln(1-\sigma(Wx^{(i)}+b) + y_i \ln(\sigma(Wx^{(i)}+b))$$

We also need to choose the optimization algorithm which will be used to minimize the loss function. A process named gradient descent optimization is a common method for finding the minimum of a function by taking iterative steps along the gradient in a negative (descending) direction. The gradient descent optimization algorithms already implemented in TensorFlow as:

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=Y, logits=output_layer))

train_step = tf.train.GradientDescentOptimizer(1e-4).minimize(cross_entropy)
```

6-Neural Network learning:

The essence of the training process in deep learning is to optimize the loss function, by a way to minimize the difference between the predicted labels of the images, and the true labels of the images, this will be changed (update) in weights .The process involves four steps which are repeated for a set number of iterations:

-Propagate values forward through the network

-Compute the loss

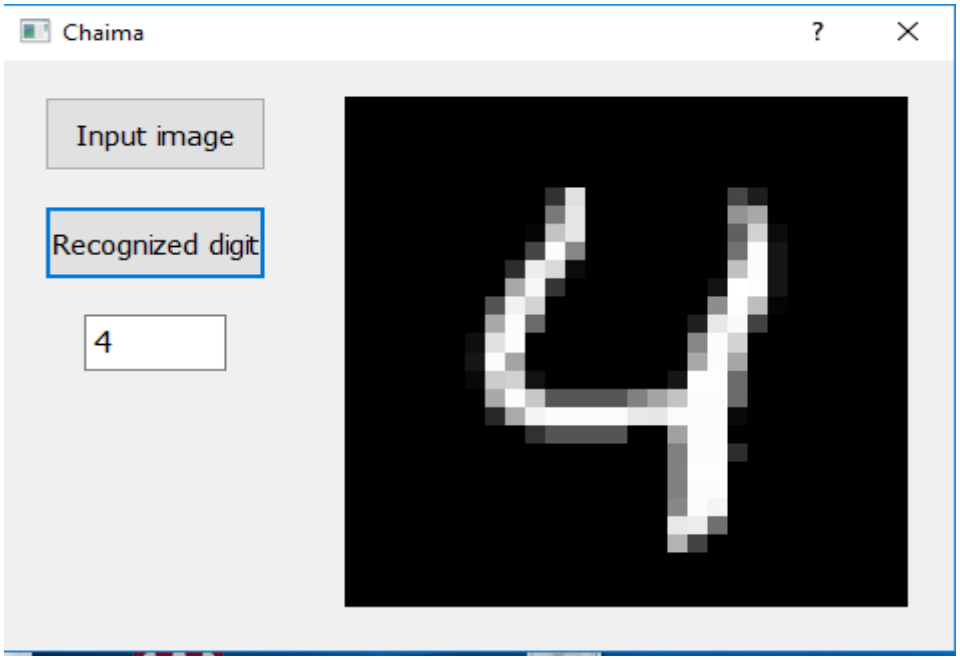
-Propagate values backwards through the network

-Update the parameters

At each training step, the parameters are adjusted to try and reduce the loss for the next step. As the learning progresses, we should see a reduction in loss, and eventually, we can stop training and use the network as a model for testing our new data. The picture below show how to implement the process:

```
# train on mini batches
for i in range(n_iterations):
    batch_x, batch_y = mnist.train.next_batch(batch_size)
    sess.run(train_step, feed_dict={
        X: batch_x, Y: batch_y, keep_prob: dropout
    })
```

It’s now time to run our program and see how accurately our neural network can recognize these handwritten digits.



8-Conclusion :

In this report, we successfully trained a neural network to classify the MNIST dataset with help TensorFlow who is a very important tool to simplify work and tested it on an image of our .and if you want to check my code visit my Github (https://github.com/chaimaomar/handwritten-digit-recognition) And this is the first step to entre to deep learning and machine learning to solve the problems complicated

References:

[1] https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/