Programming Assignment 5

Artificial Neural Network with Backpropagation

Patrick Humphries (7097-1087-72, pvhumphr@usc.edu)
INF552 Machine Learning for Data Science (32458)
University of Southern California
Viterbi School of Engineering
Spring 2020

Abstract This is a three-part assignment intended to inculcate the concepts of artificial neural networks. As the first part, a simple artificial neural network was implemented to identify hands gesturing with the thumb pointing down among a collection hand gestures. The accuracy rate achieved is 0.77. The same accuracy was achieved using the software packages *keras* and *tensorflow* in the second part. The third part identifies some interesting applications of artificial neural networks including *augmented cognition* and *human-robot collaboration.*

Contents

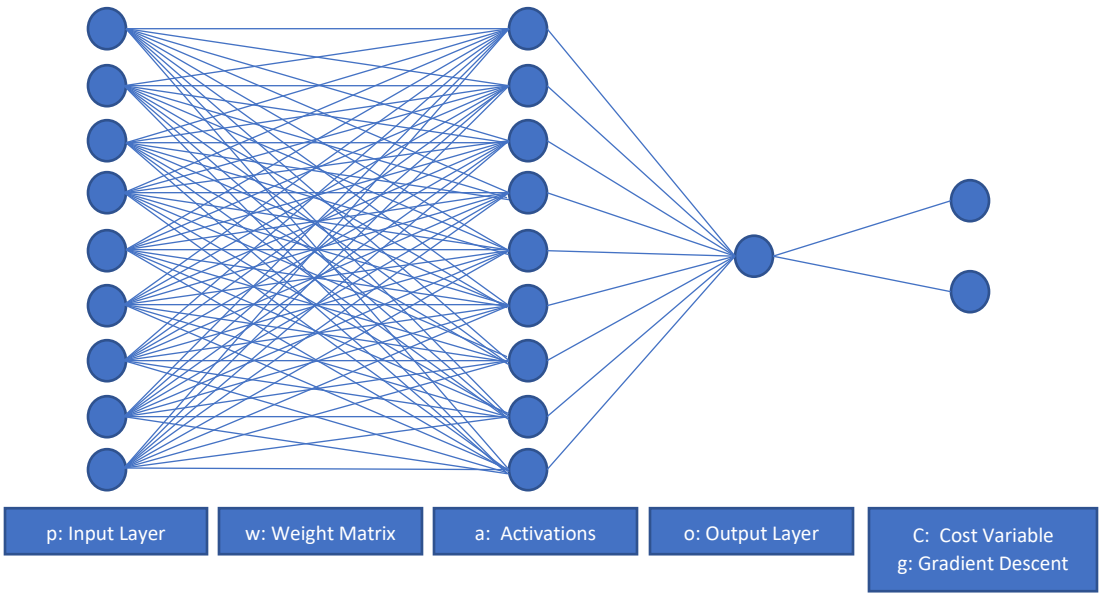
Part 1: Implementation

Data

The data consists of 30x32 images of grey-scaled samples. The grey scale ranges from 0 to 255. Most images have a normal contrast of a light background and a dark background. Some samples have the pixel values reversed because the background is darker than the foreground. The following visualization is of 25 sample images. Note that images 22 through 25 had a darker background.

These files have their pixel values reversed. This yields consistent data for training and testing.

Network Structure

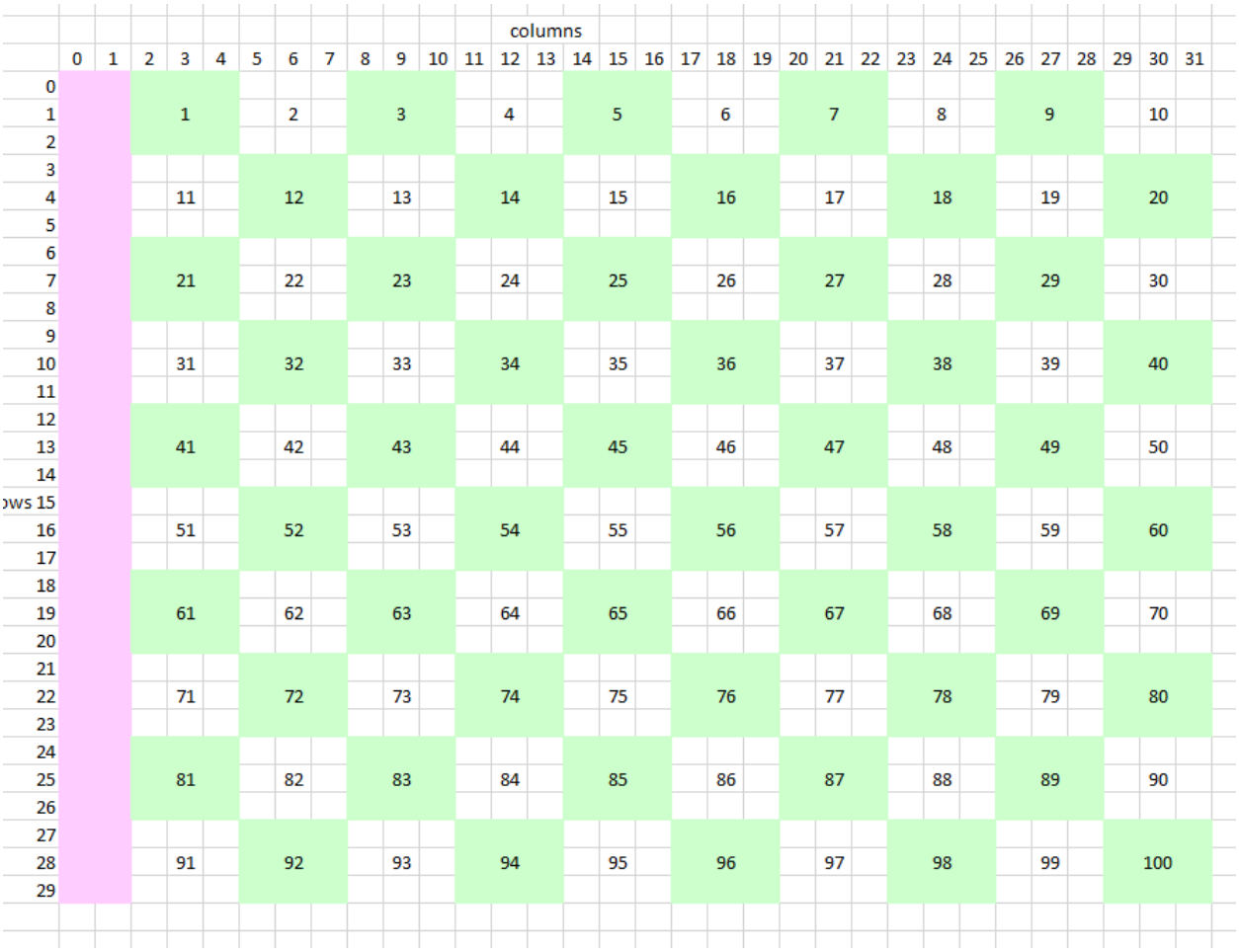


Input Layer

Each pixel will be addressed by p_{ik} where i is the index of the sample and k is the index of the pixel in the sample. Training data contains m samples with 960 pixels per sample. Each column of the matrix will be the input layer for the given sample.

$$\begin{bmatrix} p_{0.0} & \cdots & p_{m.o} \\ \vdots & \ddots & \vdots \\ p_{0.959} & \cdots & p_{m.959} \end{bmatrix}$$

The following figure is the layout of the input layer. The image is organized by 30 rows and 32 columns. The first two columns contain no information, so they are ignored. The number in each 3x3 square of pixels is the *j* index of the corresponding weight and activation. Each activation is responsible for a single 3x3 square of pixels.



Weights

The weight for each activation-pixel combination is identified by $\omega_{j.k}$ where j is the index of the activation and k is the index of the pixel $p_{i.k}$ where i is the index of the sample. The dot product of a row of this matrix and a column in the pixel matrix will yield a vector of values, each corresponding to an activation.

$$\begin{bmatrix} \omega_{0.0} & \cdots & \omega_{0.10} \\ \vdots & \ddots & \vdots \\ \omega_{99.0} & \cdots & \omega_{99.10} \end{bmatrix}$$

Since each activations is only responsible for 9 pixels, the size of the weights for any activation is only 9, not the full 960. Weights were limited to a range of -0.01 to 0.01.

Bias

A bias is identified by b_j where j is index of activations. The bias participates in the calculation of a probability yielded by the corresponding activation. It is initialized to be a vector of zeroes.

$$\begin{bmatrix} b_0 \\ \vdots \\ b_{99} \end{bmatrix}$$

Hidden Layer

The hidden layer will contain 100 activations, each identified by a_j where j is the index of the activation in the hidden layer. Activations a_0 through a_{99} are responsible for individual 3x3 squares of pixels. There are 100 of these squares in the input layer. Using a sigmoid function with the dot product of weights vector and pixels vector, a probability is yielded. It is further modified by the corresponding bias.

$$\begin{bmatrix} a_0 \\ \vdots \\ a_{99} \end{bmatrix}$$

Note that 100 3x3 squares covers only 900 pixels. The input has 960 pixels. The data was inspected an there was no information included in the first two columns. Hence, these columns are ignored.

Activation

An activation calculates the probability based on pixels, weights, and biases:

$$a_j = \left(\sum_{k=0}^{9} \omega_{jk} p_k \right) + b_j$$

An activation performs the following steps.

1. Create a nine-pixel vector from the input layer based on the index of the activation.
2. Create a nine-weight vector from the weight matrix based on the index of the activation.
3. Perform the dot product of the pixel and weight vectors.
4. Calculate the probability using the sigmoid function.
5. Add the corresponding bias.

An activation is indexed by j and is responsible for a 3x3 square. The input layer (sample) is organized by 30x32. That is 30 rows and 32 columns. The first two columns are ignored because they contain no information. The activation uses its index to determine for which pixels it is responsible. The row of the top left corner of the nine-pixel square is

$$row = \left(\left(math.ceil\left(\frac{(j+1)}{10} \right) \right) * 3 \right) - 3$$

The column of the top left corner is

$$column = ((j+1) * 3 - 1) - (row * 10)$$

Once the upper left corner is known, the other eight are calculated by incrementing the row and column indices until the 3x3 square is complete.

Output Layer

The output layer consists of a single perceptron identified by o. It averages the probabilities from the hidden layer and then subtracts 0.5. The subtraction is necessary because of the sigmoid that is required. If all sigmoids of the hidden layer could not decide on an output, each would return 0.5, resulting in an average of 0.5. However, 0.5 as input to the output sigmoid would indicate a 1 output, which would be false. Subtracting 0.5 from the average would result in a zero as input to the output sigmoid. This would yield a probability of 0.5 which is more accurate. The calculation for o is:

$$a = \left(\frac{1}{100} \sum_{j=0}^{99} a_j \right) - 0.5$$

$$o = \sigma(a) = \frac{1}{1 + e^{-a}} = \frac{e^a}{e^a + 1}$$

Cost Variable

The cost variable for each sample is calculated as shown below. The index i identifies the sample and y is the label for the instance of the sample.

$$C = y_i - o$$

$$C_i = C^2$$

The total cost for an iteration is:

$$C_h = \sum_{i=0}^{m} C_i$$

The total cost for all iterations is:

$$C = \sum_{h=0}^{999} C_h$$

The index m is the total number of samples in this iteration and h is the iteration number. There can be up to 1,000 iterations. However, if C meets a threshold, then the process would be stopped early.

Gradient Descent

The gradient descent for a single input from a into o is defined as $\frac{\partial C}{\partial \omega} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial \omega}$. It used for identifying the a_j and corresponding ω_{jk} to be used for backpropagation. The decision criteria is the largest negative value. The negative value indicates the change in direction of the cost. The magnitude indicates the most influential weights. The "@" symbol is the dot product operator.

$$\frac{\partial C}{\partial a} = 2(y - a)$$

$$\frac{\partial a}{\partial \omega} = \sigma(a)(1 - \sigma(a))$$

$$\frac{\partial C}{\partial \omega} = 2(y - a)\sigma(a)\big(1 - \sigma(a)\big)$$

$$\sigma(a) = \frac{1}{1 + e^{-a}} = \frac{e^{a}}{e^{a} + 1}$$

$$a = w@p_i$$

Gradient descent was not used in this assignment. There were only 900 weights needed to be adjusted per sample per iteration. For this simple model, it was deemed that this complexity was not needed.

Backpropagation

The amount to adjust each weight is as follows:

$$\omega_{jk} = \omega_{jk} + \alpha C p_k$$

The learning rate is α and it is specified to be 0.1. The index j is the index of the activations. Index k is the index into the weights and pixels for the activation.

Model Execution

The following output is from the files specified in the requirements. All images are individual files in the *gestures* directory. File *downgesture_train.list.txt* contained the relative path for each of the images to be used in training. The relative paths for the training images are found in *downgesture_test.list.txt.*

```
Starting main.

Building model.
        Building samples, weights, biases, and labels.
        Building samples and labels.
        Getting file paths.
        Reversing images.
        Building weights.

Training model.
        Fitting weights and biases.
        iteration: 0 count: 184 correct: 154 rate: 0.84 cost: 30.0
        iteration: 100 count: 184 correct: 156 rate: 0.85 cost: 28.0
        iteration: 200 count: 184 correct: 156 rate: 0.85 cost: 28.0
        iteration: 300 count: 184 correct: 156 rate: 0.85 cost: 28.0
        iteration: 400 count: 184 correct: 156 rate: 0.85 cost: 28.0
        iteration: 500 count: 184 correct: 156 rate: 0.85 cost: 28.0
        iteration: 600 count: 184 correct: 156 rate: 0.85 cost: 28.0
        iteration: 700 count: 184 correct: 156 rate: 0.85 cost: 28.0
        iteration: 800 count: 184 correct: 156 rate: 0.85 cost: 28.0
        iteration: 900 count: 184 correct: 156 rate: 0.85 cost: 28.0

Scoring model.
        Building samples and labels.
        Getting file paths.
        Reversing images.
        count: 83 correct: 64 rate: 0.77

Done!
```

During the learning of the model, the predicted label and the training label were compared to gain an estimate of the accuracy. Only an accuracy rate 0.85 was obtained. During testing the model only scored a rate of 0.77.

Part 2: Software Familiarization

keras and tensorflow

The artificial neural network package *keras* appears to be popular. It is an API that uses a variety of "backend" infrastructures: *Theano, tensorflow,* and *CNTK.* The default backing is *tensorflow*.

keras is parameter driven. What are specified are input layer, hidden layers, and output layer. Also specified are methods to be used for decision making, weight adjustment, and cost calculation. With just a few lines of code to the API, a model can be created, configured, trained, validated, and tested.

This is the output from the model creation:

```
 Executing keras with tensorflow.
        If this step should fail, check if keras and tensorflow are installed.

Using TensorFlow backend.

         Building samples and labels.
         Getting file paths.
         Building samples and labels.
         Getting file paths.

 Model summary.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               96100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 22
=================================================================
Total params: 97,132
Trainable params: 97,132
Non-trainable params: 0
_________________________________________________________________
```

These "Dense" specifications correspond to the layers of the implementation. The first "Dense" is the input layer and it yields the hidden layer. The second "Dense" is the hidden layer and it yields the activations. The third "Dense" is the output layer that yields either "0" or "1".

This is the output from the model training:

```
 Training model.
WARNING:tensorflow:From C:\Users\Patrick\Anaconda3\lib\site-packages\keras\backend\tensorflow_
l_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Epoch 1/8
184/184 [==============================] - 1s 5ms/step - loss: 10.3467 - accuracy: 0.5652
Epoch 2/8
184/184 [==============================] - 0s 196us/step - loss: 0.5896 - accuracy: 0.7337
Epoch 3/8
184/184 [==============================] - 0s 774us/step - loss: 0.5833 - accuracy: 0.7337
Epoch 4/8
184/184 [==============================] - 0s 549us/step - loss: 0.5810 - accuracy: 0.7337
Epoch 5/8
184/184 [==============================] - 0s 728us/step - loss: 0.5817 - accuracy: 0.7337
Epoch 6/8
184/184 [==============================] - 0s 902us/step - loss: 0.5817 - accuracy: 0.7337
Epoch 7/8
184/184 [==============================] - 0s 562us/step - loss: 0.5817 - accuracy: 0.7337
Epoch 8/8
184/184 [==============================] - 0s 701us/step - loss: 0.5819 - accuracy: 0.7337
```

This the output from the model testing:

```
 Testing model.
count: 83 correct: 64 rate: 0.77

 Done!
```

The accuracy rate is the same as the implementation. However, with experimentation with *keras* parameters, it is likely that a better rating could be achieved.

Code Improvement

Loops are slow. Using *numpy* vectors should reduce execution time.

For more accuracy, consider the following:

- Have a hidden layer of 960 activations, one per pixel.
- Replace the averaging function of the output layer with weights.
- Try shuffling the data per iteration instead of the same sequence of samples for every iteration.

Part 3: Applications

Games There are the headlines about machine learning winning at *Go* and *Astragon.* However, there has been a plethora of games such as *dinosaur jump* and a car finding a parking place. As simple as these games are, they are allowing the democratizing of artificial neural networks.

Medial Diagnostics The accuracy of esophageal cancer diagnosis has reached ninety percent. This was possible because of abundance of medical history feeding artificial neural networks.

Cognitive Augmentation Passive devices such as smartphones and voice command devices are common today. With artificial neural networks, applications are becoming generative. Given data, goals, and constraints, an artificial neural network would yield a design. What is happening is the network is searching the entire solution space to find a best design. Cognitive augmentation will allow humans to design solutions that are beyond the human capabilities.

Human-Robot Collaboration Humans are good at awareness, perception, and decision making. Robots are good at precision and repetition. The artificial neural network can design and track the implementation of a solution. It would tell what tasks the humans to do and what motions the robot needs to take. A demonstration of this synergy was *The HIVE Project* (https://www.youtube.com/watch?v=LZ2PKnJa-Ew). After designing a structure composed of string and bamboo elements, the artificial neural network instructed the robots where to position the bamboo elements for the humans and instructed the humans how the connect bamboo elements with string. Humans can handle string, robots cannot.