# Programming Assignment 4

Varun Nagaraj

19th April, 2019

## 1 Problem 1.a

Demonstrating that a neural network to maximize the log likelihood of observing the training data is one that has softmax output nodes and minimizes the criterion function of the negative log probability of training data set.

Desired output is a posterior probability distribution over the ten digit classes.

$$\text{Jo(w)} = \ \log p(\text{(xn,tn)} : n = 1, 2..9;\text{w}) = \log \Pi_n \Pi_m^9 p(tn = m|xn, w)$$

w - synapses weights;
10 softmax output nodes generating log p(t = m—x;w), where m = 0, 1, ..., 9;
tn - label of xn image;
n - index of a pattern in the training data set.

We have 10 classes m = 0, 1, 2, ...9 Prior probability that a label(t) = m:

$$\pi_m = Pr[t = m] \tag{1}$$

Conditional distribution - probability that an image belongs to class m:

$$Pm(x) = P(x|y = m) \tag{2}$$

Thus, an overall distribution:

$$P(x) = \Sigma_{m=0}^{9} \pi_m P_m(x) \tag{3}$$

In our case the prior probability is equal for each classes, as we consider the same constant number of images (100) per each label. So we can omit constant m and the posterior probability is:

$$P(x) = \Sigma_{m=0}^{9} P_m(x) = 1 \tag{4}$$

Suppose that we are given a data set of pairs D = (xi, yi) consisting of pairs of inputs $x_i$ and corresponding outputs $y_i$ for i = 1, 2, , N. The log likelihood function for neural network is going to be P(y—x; ), where  is the set of adjustable parameters of the model i.e., the weights and biases of the network. The log likelihood function for a dataset of D is:

$$F(O) = \Sigma_{m=0}^{9} logp(y_m|x; \omega) \tag{5}$$

The logistic output function can only be used for the classification between two target classes t=1 and t=0. We have a multiclass categorical probability distribution, for this case we can use the softmax function, that returns values between 0 and 1, which allows us to interpret the outputs as probabilities. This function is a normalized exponential and is defined as:

$$P(t = m|x; \omega) = \frac{e^{x^T.w_m}}{\Sigma_{m=1}^{M} e^{x^T.w_m}} \tag{6}$$

where denominator acts as regularizer to assure, that $\Sigma_{m=0}^{9} y_m = 1$.
We apply criterion function over the output from softmax. Softmax in this case play a key role that helps to omit logarithms with exponent while doing a backpropagation, thus it is easier to perform interpretation and optimization.

$$Jo = \Sigma_{m=0}^{9} y_m log p_m \tag{7}$$

Where $p_m$ is the output from the softmax. In this case we use negative sign, as pm is a number within the range [0, 1] and the logarithm of it is negative, in order to keep the loss function positive number we add a negative sign to it. $p_m$ is one-hot encoded labels, where one label = 1 and all the rest is 0. $p_m$ is one-hot encoded labels, where one label = 1 and all the rest is 0.

$$\Sigma_{m=0}^{9} y_m = 1 \tag{8}$$

$$J_0 = -\Sigma_{m=0}^{9} log p_m \tag{9}$$

We have a large distribution, thus we need to take a sum over all samples (n):

$$J_0(\omega) = -\Sigma_n \Sigma_{m=0}^{9} log p(t_n = m|x_n; \omega) \tag{10}$$

Product rule of logarithm defines as follows:

$$log_b(xy) = log_b x + log_b y \tag{11}$$

Applying product rule to our equation(10), we get:

$$Jo() = log \Pi_n \Pi_{m=0}^{9} p(t_n = m|x_n; \omega) \tag{12}$$

Therefore it minimizes the criterion function of the negative log probability of training data set.

# 2 Problem 1.b

Demonstrate that a neural network to maximize the a posterior likelihood of observing the training data given a Gaussian prior of the weight distribution is one that minimizes the criterion function with L2 regularization.

$$max(p(w; \alpha) = N(0, \alpha I)) \equiv min(J(w) = J_0(w) - log p(w; \alpha^{-1})) \tag{13}$$

Let us imagine that you want to infer some parameter $\beta$ from some of the observed input-output pairs $(x_1, y_1)..(x_N, y_N)$. Let us assume that the outputs are linearly related to the inputs via $\beta$ and that the data are corrupted by some noise $\epsilon$

$$y_n = \beta * x_n + \epsilon \tag{14}$$

where $\epsilon$ is Gaussian noise with mean 0 and variance $\sigma^2$. This gives rise to a Gaussian likelihood as follows:

$$\Pi_{n=1}^{N} N(y_n|\beta * x_n, \sigma^2) \tag{15}$$

Let us regularise parameter $\beta$ by imposing the Gaussian prior $N(\beta|0, \lambda^{-1})$, where $\lambda$ is strictly positive. Hence, combining the likelihood and the prior we simply have:

$$\Pi_{n=1}^{N} N(y_n|\beta * x_n, \sigma^2) N(\beta|0, \lambda^{-1}) \tag{16}$$

Let us take the logarithm of the above expression. Dropping some constants we get:

$$-\Sigma_{n=1}^{N} - \frac{1}{\sigma^2}(y_n - \beta x_n)^2 - \lambda\beta^2 + const. \tag{17}$$

If we maximise the above expression with respect to $\beta$ we get the so called maximum a-posteriori estimate for $\beta$. In this expression it becomes apparent why the Gaussian prior can be interpreted as a L2 regularisation term.

# 3 Code for part 2

```
# Created by Varun at 17/04/19
import gzip
import os
import pickle

import matplotlib.pyplot as plt
import pandas as pd
from keras import optimizers
from keras.layers import MaxPooling2D, Dropout, Flatten, Dense,
BatchNormalization
from keras.utils import np_utils

os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

filename = 'mnist.pkl.gz'
f = gzip.open(filename, 'rb')
minst_training_data, minst_validation_data, minst_test_data = pickle.load(f)
f.close()
def neural_net_implementation_part1(dataset, minst_testing):
    """
    Neural Network Implementation part 1
    :param dataset: Dataset
```

```
    :param minst_testing: MNIST Test data
    :param USPS_mat: USPS Feature Matrix
    :param USPS_target: USPS Test Matrix
    """
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.callbacks import EarlyStopping
    from keras.utils import to_categorical, plot_model
    from keras import backend

    def rmse(y_true, y_pred):
        return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axi

    image_vector_size = 28 * 28
    x_train = dataset[0].reshape(dataset[0].shape[0], image_vector_size)
    x_test = minst_testing[0].reshape(minst_testing[0].shape[0], image_vecto
    y_train = to_categorical(dataset[1], 10)
    y_test = to_categorical(minst_testing[1], 10)
    x_train = x_train[:1000, :]
    x_test = x_test[:1000, :]
    y_train = y_train[:1000]
    y_test = y_test[:1000]
    print(y_train.shape)
    print(x_train.shape)
    model = Sequential()
    model.add(Dense(30, input_shape=(784,), activation='sigmoid'))

    model.add(Dense(10, activation='softmax'))
    sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='mse', optimizer=sgd, metrics=['accuracy', rmse])
    plot_model(model, 'model.png')
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, v
    history = model.fit(x_train, y_train, batch_size=10, validation_split=0.2
                        epochs=30)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('error.png')
    plt.clf()
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
```

```
    plt.ylabel('acc')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('accuracy.png')
    plt.clf()
    plt.plot(history.history['rmse'])
    plt.plot(history.history['val_rmse'])
    plt.title('criterion')
    plt.ylabel('training')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('criterion.png')

    print("MNIST")
    score, acc, rmserror = model.evaluate(x_test, y_test)
    print("Score (RMSE) : {}".format(score))
    print("accuracy is :{}".format(acc))


def neural_net_implementation_part2ab(dataset, minst_testing):
    """
    Neural Network Implementation part 2 a and b (Uncomment lines)
    :param dataset: Dataset
    :param minst_testing: MNIST Test data
    :param USPS_mat: USPS Feature Matrix
    :param USPS_target: USPS Test Matrix
    """
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.callbacks import EarlyStopping
    from keras.utils import to_categorical, plot_model
    from keras import backend

    def rmse(y_true, y_pred):
        return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axi

    image_vector_size = 28 * 28
    x_train = dataset[0].reshape(dataset[0].shape[0], image_vector_size)
    x_test = minst_testing[0].reshape(minst_testing[0].shape[0], image_vecto
    y_train = to_categorical(dataset[1], 10)
    y_test = to_categorical(minst_testing[1], 10)
    x_train = x_train[:1000, :]
    x_test = x_test[:1000, :]
    y_train = y_train[:1000]
    y_test = y_test[:1000]
```

```
    print(y_train.shape)
    print(x_train.shape)
    model = Sequential()
    model.add(Dense(30, input_shape=(784,), activation='sigmoid'))
    model.add(Dense(30, activation='sigmoid'))
    # model.add(Dense(30, activation='sigmoid', activity_regularizer=l2(5)))
    model.add(Dense(10, activation='softmax'))
    sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accurac
    plot_model(model, 'model2a2.png')
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, v
    history = model.fit(x_train, y_train, batch_size=10, validation_split=0.2
                        epochs=30)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('error2a2.png')
    plt.clf()
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('acc')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('accuracy2a2.png')
    plt.clf()
    plt.plot(history.history['rmse'])
    plt.plot(history.history['val_rmse'])
    plt.title('criterion')
    plt.ylabel('training')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('criterion2a2.png')

    print("MNIST")
    score, acc, rmserror = model.evaluate(x_test, y_test)
    print("Score (RMSE) : {}".format(score))
    print("accuracy is :{}".format(acc))

def neural_net_implementation_part3ab(dataset, minst_testing):
    """
    Neural Network Implementation part 3 a and b (Uncomment lines)
```

```
    :param dataset: Dataset
    :param minst_testing: MNIST Test data
    :param USPS_mat: USPS Feature Matrix
    :param USPS_target: USPS Test Matrix
    """
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.callbacks import EarlyStopping
    from keras.utils import to_categorical, plot_model
    from keras import backend

    def rmse(y_true, y_pred):
        return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axi

    image_vector_size = 28 * 28
    x_train = dataset[0].reshape(dataset[0].shape[0], image_vector_size)
    x_test = minst_testing[0].reshape(minst_testing[0].shape[0], image_vecto
    y_train = to_categorical(dataset[1], 10)
    y_test = to_categorical(minst_testing[1], 10)
    x_train = x_train[:1000, :]
    x_test = x_test[:1000, :]
    y_train = y_train[:1000]
    y_test = y_test[:1000]
    print(y_train.shape)
    print(x_train.shape)
    model = Sequential()
    model.add(Dense(30, input_shape=(784,), activation='sigmoid'))
    model.add(Dense(30, activation='sigmoid'))
    model.add(Dense(30, activation='sigmoid'))
    # model.add(Dense(30, activation='sigmoid', activity_regularizer=l2(5)))
    # model.add(Dense(30, activation='sigmoid', activity_regularizer=l2(5)))
    model.add(Dense(10, activation='softmax'))
    sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accurac
    plot_model(model, 'model3a.png')
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, v
    history = model.fit(x_train, y_train, batch_size=10, validation_split=0.2
                        epochs=30)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('error3a.png')
```

```
    plt.clf()
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('acc')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('accuracy3a.png')
    plt.clf()
    plt.plot(history.history['rmse'])
    plt.plot(history.history['val_rmse'])
    plt.title('criterion')
    plt.ylabel('training')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('criterion3a.png')

    print("MNIST")
    score, acc, rmserror = model.evaluate(x_test, y_test)
    print("Score (RMSE) : {}".format(score))
    print("accuracy is :{}".format(acc))


def augment_and_train():
    from keras.datasets import mnist
    from keras.preprocessing.image import ImageDataGenerator
    from keras.models import Sequential
    from keras.layers import Conv2D
    from keras import backend as K
    K.set_image_dim_ordering('th')
    (X_train, y_train), (X_test, y_test) = mnist.load_data()
    X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
    X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')
    X_train /= 255
    X_test /= 255

    Y_train = np_utils.to_categorical(y_train, 10)
    Y_test = np_utils.to_categorical(y_test, 10)

    X_train = X_train[:1000, :]
    X_test = X_test[:1000, :]
    Y_train = Y_train[:1000]
    Y_test = Y_test[:1000]
```

```
    model = Sequential()

    model.add(Conv2D(32, (3, 3), input_shape=(1, 28, 28), activation='relu'))
    BatchNormalization(axis=-1)
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    BatchNormalization(axis=-1)
    model.add(Conv2D(64, (3, 3), activation='relu'))
    BatchNormalization(axis=-1)
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    BatchNormalization()
    model.add(Dense(512, activation='relu'))
    BatchNormalization()
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=

    gen = ImageDataGenerator(rotation_range=3, width_shift_range=0.1, height_
    test_gen = ImageDataGenerator()
    train_generator = gen.flow(X_train, Y_train, batch_size=64)
    test_generator = test_gen.flow(X_test, Y_test, batch_size=64)

    model.fit_generator(train_generator, steps_per_epoch=1000 // 64, epochs=5
                        validation_data=test_generator, validation_steps=1000

    score = model.evaluate(X_test, Y_test)
    print('Test accuracy: ', score[1])
    predictions = model.predict_classes(X_test)

    predictions = list(predictions)
    actuals = list(Y_test)

    sub = pd.DataFrame({'Actual': actuals, 'Predictions': predictions})
    sub.to_csv('./output_cnn.csv', index=False)


neural_net_implementation_part1(minst_training_data, minst_test_data)
neural_net_implementation_part2ab(minst_training_data, minst_test_data)
neural_net_implementation_part3ab(minst_training_data, minst_test_data)
augment_and_train()
```

# 4 References


https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/
https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/
https://datascience.stackexchange.com/questions/26264/how-to-train-neural-network-using-maximum-likelihood-principle
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols#Relation_operators
https://medium.com/datadriveninvestor/image-processing-for-mnist-using-keras-f9a1021f6ef0
https://datascience.stackexchange.com/questions/9302/the-cross-entropy-error-function-in-neural-networks
https://stats.stackexchange.com/questions/198038/cross-entropy-or-log-likelihood-in-output-layer/198047198047
https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932
https://yashk2810.github.io/Applying-Convolutional-Neural-Network-on-the-MNIST-dataset/
https://stats.stackexchange.com/questions/163388/l2-regularization-is-equivalent-to-gaussian-prior