

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.

$$J(w) = -\log p((x_n, t_n) : n = 1, 2, \dots, N; w) = -\log \prod_n \prod_m p(t_n = m | x_n, w)$$

w - synapses weights;

10 softmax output nodes generating $p(t = m | x; w)$, where $m = 0, 1, \dots, 9$;

t_n - label of x_n image;

n - index of a pattern in the training data set.

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

$$\pi_m = Pr[t = m] \quad (1)$$

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

$$P_m(x) = P(x | y = m) \quad (2)$$

Thus, an overall distribution:

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

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 π_m and the posterior probability is:

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

Suppose that we are given a data set of pairs $D = (x_i, y_i)$ consisting of pairs of inputs x_i and corresponding outputs y_i for $i = 1, 2, \dots, N$. The log likelihood function for neural network is going to be $P(y | x; \omega)$, where ω 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:

$$F(O) = \sum_{m=0}^9 \log p(y_m | x; \omega) \quad (5)$$

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:

$$P(t = m|x; \omega) = \frac{e^{x^T \cdot w_m}}{\sum_{m=1}^M e^{x^T \cdot w_m}} \quad (6)$$

where denominator acts as regularizer to assure, that $\sum_{m=0}^9 y_m = 1$.

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.

$$Jo = \sum_{m=0}^9 y_m \log p_m \quad (7)$$

Where p_m is the output from the softmax. In this case we use negative sign, as p_m 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.

$$\sum_{m=0}^9 y_m = 1 \quad (8)$$

$$J_0 = -\sum_{m=0}^9 \log p_m \quad (9)$$

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

$$J_0(\omega) = -\sum_n \sum_{m=0}^9 \log p(t_n = m|x_n; \omega) \quad (10)$$

Product rule of logarithm defines as follows:

$$\log_b(xy) = \log_b x + \log_b y \quad (11)$$

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

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

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.

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

Let us imagine that you want to infer some parameter β from some of the observed input-output pairs $(x_1, y_1) \dots (x_N, y_N)$. Let us assume that the outputs are linearly related to the

inputs via β and that the data are corrupted by some noise ϵ

$$y_n = \beta * x_n + \epsilon \quad (14)$$

where ϵ is Gaussian noise with mean 0 and variance σ^2 . This gives rise to a Gaussian likelihood as follows:

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

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

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

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

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

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

3 Code for part 2

```

1 # Created by Varun at 17/04/19
2 import gzip
3 import os
4 import pickle
5
6 import matplotlib.pyplot as plt
7 import pandas as pd
8 from keras import optimizers
9 from keras.layers import MaxPooling2D, Dropout, Flatten, Dense,
10 BatchNormalization
11 from keras.utils import np_utils
12
13 os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
14
15 filename = 'mnist.pkl.gz'
16 f = gzip.open(filename, 'rb')
17 minst_training_data, minst_validation_data, minst_test_data = pickle.load(f)
18 f.close()
19 def neural_net_implementation_part1(dataset, minst_testing):
20     """
21     Neural Network Implementation part 1
22     :param dataset: Dataset

```

```

23      :param minst_testing: MNIST Test data
24      :param USPS_mat: USPS Feature Matrix
25      :param USPS_target: USPS Test Matrix
26      """
27      from keras.models import Sequential
28      from keras.layers import Dense
29      from keras.callbacks import EarlyStopping
30      from keras.utils import to_categorical, plot_model
31      from keras import backend
32
33      def rmse(y_true, y_pred):
34          return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axis=-1))
35
36      image_vector_size = 28 * 28
37      x_train = dataset[0].reshape(dataset[0].shape[0], image_vector_size)
38      x_test = minst_testing[0].reshape(minst_testing[0].shape[0], image_vector_size)
39      y_train = to_categorical(dataset[1], 10)
40      y_test = to_categorical(minst_testing[1], 10)
41      x_train = x_train[:1000, :]
42      x_test = x_test[:1000, :]
43      y_train = y_train[:1000]
44      y_test = y_test[:1000]
45      print(y_train.shape)
46      print(x_train.shape)
47      model = Sequential()
48      model.add(Dense(30, input_shape=(784,), activation='sigmoid'))
49
50      model.add(Dense(10, activation='softmax'))
51      sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
52      model.compile(loss='mse', optimizer=sgd, metrics=['accuracy', rmse])
53      plot_model(model, 'model.png')
54      monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1)
55      history = model.fit(x_train, y_train, batch_size=10, validation_split=0.2,
56                          callbacks=[monitor], epochs=30)
57      plt.plot(history.history['loss'])
58      plt.plot(history.history['val_loss'])
59      plt.title('model loss')
60      plt.ylabel('loss')
61      plt.xlabel('epoch')
62      plt.legend(['train', 'test'], loc='upper left')
63      plt.savefig('error.png')
64      plt.clf()
65      plt.plot(history.history['acc'])
66      plt.plot(history.history['val_acc'])
67      plt.title('model accuracy')

```

```

68     plt.ylabel('acc')
69     plt.xlabel('epoch')
70     plt.legend(['train', 'test'], loc='upper left')
71     plt.savefig('accuracy.png')
72     plt.clf()
73     plt.plot(history.history['rmse'])
74     plt.plot(history.history['val_rmse'])
75     plt.title('criterion')
76     plt.ylabel('training')
77     plt.xlabel('epoch')
78     plt.legend(['train', 'test'], loc='upper left')
79     plt.savefig('criterion.png')
80
81     print("MNIST")
82     score, acc, rmse = model.evaluate(x_test, y_test)
83     print("Score (RMSE) : {}".format(score))
84     print("accuracy is : {}".format(acc))
85
86
87 def neural_net_implementation_part2ab(dataset, minst_testing):
88     """
89     Neural Network Implementation part 2 a and b (Uncomment lines)
90     :param dataset: Dataset
91     :param minst_testing: MNIST Test data
92     :param USPS_mat: USPS Feature Matrix
93     :param USPS_target: USPS Test Matrix
94     """
95     from keras.models import Sequential
96     from keras.layers import Dense
97     from keras.callbacks import EarlyStopping
98     from keras.utils import to_categorical, plot_model
99     from keras import backend
100
101     def rmse(y_true, y_pred):
102         return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axis=-1))
103
104     image_vector_size = 28 * 28
105     x_train = dataset[0].reshape(dataset[0].shape[0], image_vector_size)
106     x_test = minst_testing[0].reshape(minst_testing[0].shape[0], image_vector_size)
107     y_train = to_categorical(dataset[1], 10)
108     y_test = to_categorical(minst_testing[1], 10)
109     x_train = x_train[:1000, :]
110     x_test = x_test[:1000, :]
111     y_train = y_train[:1000]
112     y_test = y_test[:1000]

```

```

113     print(y_train.shape)
114     print(x_train.shape)
115     model = Sequential()
116     model.add(Dense(30, input_shape=(784,), activation='sigmoid'))
117     model.add(Dense(30, activation='sigmoid'))
118     # model.add(Dense(30, activation='sigmoid', activity_regularizer=l2(5)))
119     model.add(Dense(10, activation='softmax'))
120     sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
121     model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accuracy'])
122     plot_model(model, 'model2a2.png')
123     monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1)
124     history = model.fit(x_train, y_train, batch_size=10, validation_split=0.2,
125                        epochs=30)
126     plt.plot(history.history['loss'])
127     plt.plot(history.history['val_loss'])
128     plt.title('model loss')
129     plt.ylabel('loss')
130     plt.xlabel('epoch')
131     plt.legend(['train', 'test'], loc='upper left')
132     plt.savefig('error2a2.png')
133     plt.clf()
134     plt.plot(history.history['acc'])
135     plt.plot(history.history['val_acc'])
136     plt.title('model accuracy')
137     plt.ylabel('acc')
138     plt.xlabel('epoch')
139     plt.legend(['train', 'test'], loc='upper left')
140     plt.savefig('accuracy2a2.png')
141     plt.clf()
142     plt.plot(history.history['rmse'])
143     plt.plot(history.history['val_rmse'])
144     plt.title('criterion')
145     plt.ylabel('training')
146     plt.xlabel('epoch')
147     plt.legend(['train', 'test'], loc='upper left')
148     plt.savefig('criterion2a2.png')
149
150     print("MNIST")
151     score, acc, rmse = model.evaluate(x_test, y_test)
152     print("Score (RMSE) : {}".format(score))
153     print("accuracy is :{}".format(acc))
154
155 def neural_net_implementation_part3ab(dataset, minst_testing):
156     """
157     Neural Network Implementation part 3 a and b (Uncomment lines)

```

```

158 :param dataset: Dataset
159 :param minst_testing: MNIST Test data
160 :param USPS_mat: USPS Feature Matrix
161 :param USPS_target: USPS Test Matrix
162 """
163 from keras.models import Sequential
164 from keras.layers import Dense
165 from keras.callbacks import EarlyStopping
166 from keras.utils import to_categorical, plot_model
167 from keras import backend
168
169 def rmse(y_true, y_pred):
170     return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axis=-1))
171
172 image_vector_size = 28 * 28
173 x_train = dataset[0].reshape(dataset[0].shape[0], image_vector_size)
174 x_test = minst_testing[0].reshape(minst_testing[0].shape[0], image_vector_size)
175 y_train = to_categorical(dataset[1], 10)
176 y_test = to_categorical(minst_testing[1], 10)
177 x_train = x_train[:1000, :]
178 x_test = x_test[:1000, :]
179 y_train = y_train[:1000]
180 y_test = y_test[:1000]
181 print(y_train.shape)
182 print(x_train.shape)
183 model = Sequential()
184 model.add(Dense(30, input_shape=(784,), activation='sigmoid'))
185 model.add(Dense(30, activation='sigmoid'))
186 model.add(Dense(30, activation='sigmoid'))
187 # model.add(Dense(30, activation='sigmoid', activity_regularizer=l2(5)))
188 # model.add(Dense(30, activation='sigmoid', activity_regularizer=l2(5)))
189 model.add(Dense(10, activation='softmax'))
190 sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
191 model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accuracy'])
192 plot_model(model, 'model3a.png')
193 monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1)
194 history = model.fit(x_train, y_train, batch_size=10, validation_split=0.2,
195                    epochs=30)
196 plt.plot(history.history['loss'])
197 plt.plot(history.history['val_loss'])
198 plt.title('model loss')
199 plt.ylabel('loss')
200 plt.xlabel('epoch')
201 plt.legend(['train', 'test'], loc='upper left')
202 plt.savefig('error3a.png')

```

```

203     plt.clf()
204     plt.plot(history.history['acc'])
205     plt.plot(history.history['val_acc'])
206     plt.title('model accuracy')
207     plt.ylabel('acc')
208     plt.xlabel('epoch')
209     plt.legend(['train', 'test'], loc='upper left')
210     plt.savefig('accuracy3a.png')
211     plt.clf()
212     plt.plot(history.history['rmse'])
213     plt.plot(history.history['val_rmse'])
214     plt.title('criterion')
215     plt.ylabel('training')
216     plt.xlabel('epoch')
217     plt.legend(['train', 'test'], loc='upper left')
218     plt.savefig('criterion3a.png')
219
220     print("MNIST")
221     score, acc, rmse = model.evaluate(x_test, y_test)
222     print("Score (RMSE) : {}".format(score))
223     print("accuracy is :{}".format(acc))
224
225
226 def augment_and_train():
227     from keras.datasets import mnist
228     from keras.preprocessing.image import ImageDataGenerator
229     from keras.models import Sequential
230     from keras.layers import Conv2D
231     from keras import backend as K
232     K.set_image_dim_ordering('th')
233     (X_train, y_train), (X_test, y_test) = mnist.load_data()
234     X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
235     X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
236     X_train = X_train.astype('float32')
237     X_test = X_test.astype('float32')
238     X_train /= 255
239     X_test /= 255
240
241     Y_train = np_utils.to_categorical(y_train, 10)
242     Y_test = np_utils.to_categorical(y_test, 10)
243
244     X_train = X_train[:1000, :]
245     X_test = X_test[:1000, :]
246     Y_train = Y_train[:1000]
247     Y_test = Y_test[:1000]

```



```

248
249     model = Sequential()
250
251     model.add(Conv2D(32, (3, 3), input_shape=(1, 28, 28), activation='relu'))
252     BatchNormalization(axis=-1)
253     model.add(Conv2D(32, (3, 3), activation='relu'))
254     model.add(MaxPooling2D(pool_size=(2, 2)))
255     BatchNormalization(axis=-1)
256     model.add(Conv2D(64, (3, 3), activation='relu'))
257     BatchNormalization(axis=-1)
258     model.add(Conv2D(64, (3, 3), activation='relu'))
259     model.add(MaxPooling2D(pool_size=(2, 2)))
260     model.add(Flatten())
261     BatchNormalization()
262     model.add(Dense(512, activation='relu'))
263     BatchNormalization()
264     model.add(Dropout(0.2))
265     model.add(Dense(10, activation='softmax'))
266     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
267
268     gen = ImageDataGenerator(rotation_range=3, width_shift_range=0.1, height_
269     test_gen = ImageDataGenerator()
270     train_generator = gen.flow(X_train, Y_train, batch_size=64)
271     test_generator = test_gen.flow(X_test, Y_test, batch_size=64)
272
273     model.fit_generator(train_generator, steps_per_epoch=1000 // 64, epochs=5
274                        validation_data=test_generator, validation_steps=1000
275
276     score = model.evaluate(X_test, Y_test)
277     print('Test accuracy: ', score[1])
278     predictions = model.predict_classes(X_test)
279
280     predictions = list(predictions)
281     actuals = list(Y_test)
282
283     sub = pd.DataFrame({'Actual': actuals, 'Predictions': predictions})
284     sub.to_csv('./output_cnn.csv', index=False)
285
286
287     neural_net_implementation_part1(minst_training_data, minst_test_data)
288     neural_net_implementation_part2ab(minst_training_data, minst_test_data)
289     neural_net_implementation_part3ab(minst_training_data, minst_test_data)
290     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>