Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем
Кафедра Информатики

ЛАБОРАТОРНАЯ РАБОТА №2
«Реализация глубокой нейронной сети»

Магистрант:
гр. 956241
Шуба И.А.

Проверил:
Заливако С. С.

Минск, 2020

ХОД РАБОТЫ

Задание.

Данные: В работе предлагается использовать набор данных notMNIST, который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A … J, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке:

- https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz (большой набор данных);
- https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz (маленький набор данных);

Описание данных на английском языке доступно по ссылке: http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html

Задание 1.

Реализуйте полносвязную нейронную сеть с помощью библиотеки Tensor Flow. В качестве алгоритма оптимизации можно использовать, например, стохастический градиент (Stochastic Gradient Descent, SGD). Определите количество скрытых слоев от 1 до 5, количество нейронов в каждом из слоев до нескольких сотен, а также их функции активации (кусочно-линейная, сигмоидная, гиперболический тангенс и т.д.).

Задание 2.

Как улучшилась точность классификатора по сравнению с логистической регрессией?

Задание 3.

Используйте регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Как улучшилось качество классификации?

Задание 4.

Воспользуйтесь динамически изменяемой скоростью обучения (learning rate). Наилучшая точность, достигнутая с помощью данной модели составляет 97.1%. Какую точность демонстрирует Ваша реализованная модель?

Результат выполнения:

Задание 1. Реализуйте полносвязную нейронную сеть с помощью библиотеки TensorFlow. В качестве алгоритма оптимизации можно использовать, например, стохастический градиент (Stochastic Gradient Descent, SGD). Определите количество скрытых слоев от 1 до 5, количество нейронов в каждом из слоев до нескольких сотен, а также их функции активации (кусочно-линейная, сигмоидная, гиперболический тангенс и т.д.).

Загрузим датасет, как в предыдущей лабораторной работе.

```
X_features = pickle.load(open("X_features.pickle","rb"))
Y_label = pickle.load(open("Y_labels.pickle","rb"))
```

```
X_features = np.array(X_features) / 255 # normalization of data for easy to calculation
s.
Y_label = np.array(Y_label)
```

```
resolution = 28
classes = 10
```

```
X_features = X_features.reshape((-1, resolution, resolution, 1))
Y_labels = np_utils.to_categorical(Y_label, 10)
```

```
TRAIN = 200000
VAL = 10000
TEST = 19000
```

```
x_train, y_train = X_features[:TRAIN], Y_labels[:TRAIN]
x_val, y_val = X_features[TRAIN:TRAIN+VAL], Y_labels[TRAIN:TRAIN+VAL]
x_test, y_test = X_features[TRAIN+VAL:TRAIN+VAL+TEST], Y_labels[TRAIN+VAL:TRAIN+VAL+TES
T]
```

Рисунок 1 – Загрузка данных

Создадим нейронную сеть, состоящую из 5 полносвязных слоев. Функции активации для каждого слоя – relu, для последнего слоя – softmax. В качестве алгоритма оптимизации выберем стохастический градиент.

```
model = Sequential()

model.add(Flatten(input_shape=(resolution, resolution, 1)))
model.add(Dense(128, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(classes, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
keras2ascii(model)
```

Рисунок 2 – Реализация полносвязной нейронной сети

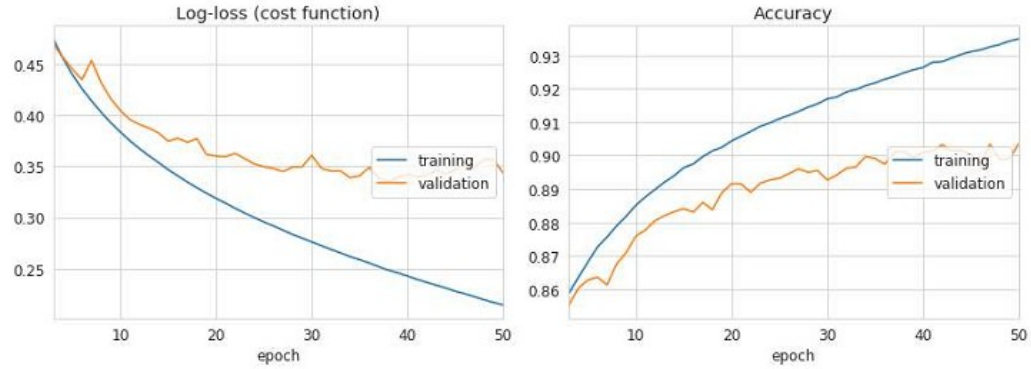
```
OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

    Input   #####     28   28    1
  Flatten   ||||| -------------------        0     0.0%
            #####          784
    Dense   XXXXX -------------------   100480    57.3%
     relu   #####          128
    Dense   XXXXX -------------------    33024    18.8%
     relu   #####          256
    Dense   XXXXX -------------------    32896    18.8%
     relu   #####          128
    Dense   XXXXX -------------------     8256     4.7%
     relu   #####           64
    Dense   XXXXX -------------------      650     0.4%
  softmax   #####           10
```

Рисунок 3 – Схема полносвязной нейронной сети

Обучение будет проводиться с размером батча 128 и с 50 эпохами.

```
model.fit(x_train, y_train,
          epochs=50,
          batch_size=128,
          validation_data=(x_val, y_val),
          callbacks=[PlotLossesKeras()],
          verbose=0)
```




```
Log-loss (cost function):
training   (min:    0.215, max:    0.528, cur:    0.215)
validation (min:    0.335, max:    0.510, cur:    0.343)

Accuracy:
training   (min:    0.846, max:    0.935, cur:    0.935)
validation (min:    0.846, max:    0.903, cur:    0.903)
```

Рисунок 4 – Результаты обучения

Как видно из графиков, точность на валидационных данных немного ниже, чем на обученных, но не критично. Максимальная точность на валидационных данных составила 0,903, а минимальный log-loss – 0,335.

Проверим полученную модель на контрольной выборке.

```
y_pred = model.predict_classes(x_test)
```

```
accuracy_score(y_test, y_pred)
```

```
Out[42]:

0.9006842105263158
```

```
f1_score(y_test, y_pred, average='weighted')
```

```
Out[43]:

0.9007160279592916
```

```
confusion_matrix(y_test, y_pred)
```

```
Out[44]:

array([[1643,   11,    9,   25,    8,   13,   19,   41,   17,    7],
       [  26, 1632,   12,   73,   34,   13,   43,   45,   28,   11],
       [  10,   10, 1813,   11,   33,   10,   38,   16,   15,    6],
       [  24,   17,    7, 1712,   12,    6,   13,   19,   19,   24],
       [  25,   18,   50,    8, 1736,   17,   39,   10,   23,    9],
       [  16,    7,   16,   22,   24, 1760,   26,   15,   31,   13],
       [  21,   15,   52,   20,   18,   14, 1685,   22,   21,   21],
       [  31,   23,    5,   26,   10,    7,   26, 1658,   30,   10],
       [  34,    8,   29,   31,   23,    8,   27,   22, 1716,   56],
       [  31,    4,   10,   25,    6,   16,   23,   28,   40, 1758]])
```

Рисунок 5 – Результаты модели на контрольной выборке

Задание 2. Как улучшилась точность классификатора по сравнению с логистической регрессией?

По сравнению с логистической регрессией увеличилась точность модели и если точно на контрольной выборке составляла ~0,83, то сейчас ~ 0.9. Это касается и метрики F1. Также стоит обратить внимание на confusion matrix. По сравнению с логистической регрессией, полученная модель стала меньше путаться в определении числа 8 и 9. Да и в общем, более-менее равномерно распределены фолсдетекты, нет явных выбросов, что говорит о том, что модель является адекватной и запомнила основные особенности каждого класса.

Задание 3. Используйте регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Как улучшилось качество классификации?

На основе предыдущей модели, дополним ее на каждом слое регуляризацией l2, а также после каждого слоя применим метод сброса нейронов.

```
model = Sequential()

model.add(Flatten(input_shape=(resolution, resolution, 1)))
model.add(Dense(128, activation = 'relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.1))
model.add(Dense(256, activation = 'relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.1))
model.add(Dense(128,activation = 'relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.1))
model.add(Dense(64, activation = 'relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(classes, activation='softmax'))
epochs = 50
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
keras2ascii(model)
```

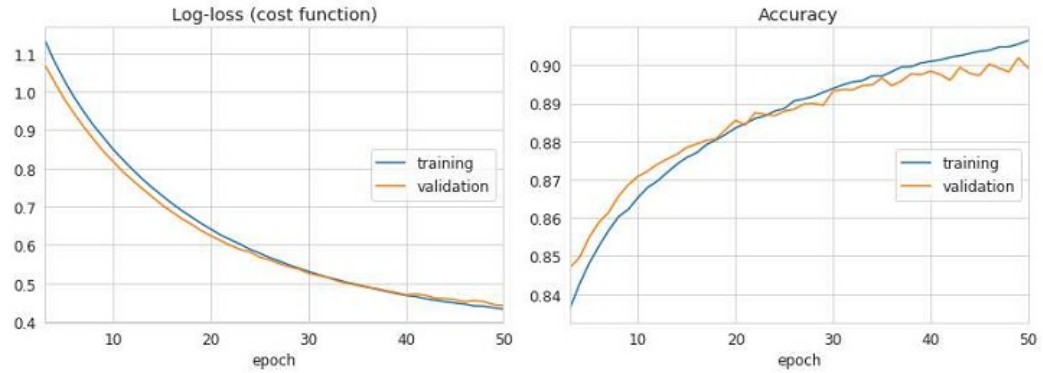
Рисунок 6 – Реализация полносвязной нейронной сети с добавлением регуляризации и сброса нейронов

```
OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

    Input   #####     28   28    1
  Flatten   |||||  -------------------        0     0.0%
            #####          784
    Dense   XXXXX  -------------------   100480    57.3%
     relu   #####          128
  Dropout    | ||  -------------------        0     0.0%
            #####          128
    Dense   XXXXX  -------------------    33024    18.8%
     relu   #####          256
  Dropout    | ||  -------------------        0     0.0%
            #####          256
    Dense   XXXXX  -------------------    32896    18.8%
     relu   #####          128
  Dropout    | ||  -------------------        0     0.0%
            #####          128
    Dense   XXXXX  -------------------     8256     4.7%
     relu   #####           64
    Dense   XXXXX  -------------------      650     0.4%
  softmax   #####           10
```

Рисунок 7 – Схема полносвязной нейронной сети с добавлением регуляризации и сброса нейронов

```
model.fit(x_train, y_train,
          epochs=epochs,
          batch_size=128,
          validation_data=(x_val, y_val),
          callbacks=[PlotLossesKeras()],
          verbose=0)
```




```
Log-loss (cost function):
training   (min:    0.433, max:    1.575, cur:    0.433)
validation (min:    0.442, max:    1.214, cur:    0.442)

Accuracy:
training   (min:    0.724, max:    0.906, cur:    0.906)
validation (min:    0.828, max:    0.902, cur:    0.899)
```

Рисунок 8 – Результаты обучения

Как видно из графиков, точность на валидационных данных сходится с точностью на обученных данных. Тоже самое произошло и с log-loss. Регуляризация и метод сброса нейронов применяется для предотвращения переобучения, причем могут применятся раздельно или вместе, как реализовано. Максимальная точность на валидационных данных составила 0,902, а минимальный log-loss – 0,442.

```
accuracy_score(y_test, y_pred)
```

```
Out[62]:

0.8993684210526316
```

```
f1_score(y_test, y_pred, average='weighted')
```

```
Out[63]:

0.8994492180075866
```

```
confusion_matrix(y_test, y_pred)
```

```
Out[64]:

array([[1655,   20,    8,   17,    4,    7,   15,   34,   22,   11],
       [  30, 1694,    9,   39,   18,   18,   27,   47,   30,    5],
       [  11,   19, 1788,   11,   32,   13,   54,   12,   15,    7],
       [  32,   43,    2, 1698,    3,   11,   12,   11,   21,   20],
       [  21,   38,   44,   18, 1694,   23,   54,   13,   27,    3],
       [  15,   15,   18,   15,   21, 1740,   32,   19,   32,   23],
       [  26,   32,   32,   19,   10,   12, 1698,   15,   24,   21],
       [  31,   31,    2,   13,    6,   12,   17, 1683,   25,    6],
       [  30,   35,   19,   32,   11,   23,   30,   23, 1694,   57],
       [  29,    8,    8,   31,    8,   15,   22,   21,   55, 1744]])
```

Рисунок 9 – Результаты модели на контрольной выборке

В данном случае метод сброса нейронов и регуляризация не повлияли на увеличение точности модели, однако при должной настройке параметров, может быть, скор бы и увеличился. Однако факт того, что кривые при обучении практически повторяли форму друг друга, говорит о том, что модель явно не переобучена и ей можно доверять.

Задание 4. Воспользуйтесь динамически изменяемой скоростью обучения (learning rate). Наилучшая точность, достигнутая с помощью данной модели составляет 97.1%. Какую точность демонстрирует Ваша реализованная модель?

```
model = Sequential()

model.add(Flatten(input_shape=(resolution, resolution, 1)))
model.add(Dense(128, activation = 'relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.1))
model.add(Dense(256, activation = 'relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.1))
model.add(Dense(128,activation = 'relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.1))
model.add(Dense(64, activation = 'relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(classes, activation='softmax'))
epochs = 50
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.9
sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rate, nesterov=False)
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
keras2ascii(model)
```

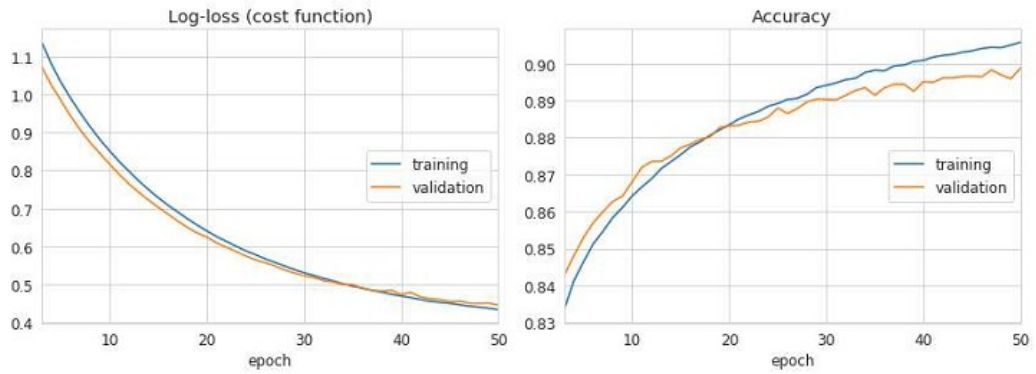
Рисунок 10 – Реализация полносвязной нейронной сети с добавлением динамически изменяемой скоростью обучения

```
OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

    Input   #####     28   28    1
  Flatten   ||||| -------------------        0     0.0%
            #####          784
    Dense   XXXXX -------------------   100480    57.3%
     relu   #####          128
  Dropout    | || -------------------        0     0.0%
            #####          128
    Dense   XXXXX -------------------    33024    18.8%
     relu   #####          256
  Dropout    | || -------------------        0     0.0%
            #####          256
    Dense   XXXXX -------------------    32896    18.8%
     relu   #####          128
  Dropout    | || -------------------        0     0.0%
            #####          128
    Dense   XXXXX -------------------     8256     4.7%
     relu   #####           64
    Dense   XXXXX -------------------      650     0.4%
  softmax   #####           10
```

Рисунок 11 – Схема полносвязной нейронной сети с добавлением регуляризации и сброса нейронов

```
model.fit(x_train, y_train,
          epochs=epochs,
          batch_size=128,
          validation_data=(x_val, y_val),
          callbacks=[PlotLossesKeras()],
          verbose=0)
```




```
Log-loss (cost function):
training   (min:    0.434, max:    1.605, cur:    0.434)
validation (min:    0.446, max:    1.221, cur:    0.446)

Accuracy:
training   (min:    0.712, max:    0.906, cur:    0.906)
validation (min:    0.819, max:    0.899, cur:    0.899)
```

Рисунок 12 – Реализация полносвязной нейронной сети с добавлением динамически изменяемой скоростью обучения

```
accuracy_score(y_test, y_pred)
```

```
Out[56]:
```

```
0.902578947368421
```

```
f1_score(y_test, y_pred, average='weighted')
```

```
Out[57]:
```

```
0.9028796868350255
```

```
confusion_matrix(y_test, y_pred)
```

```
Out[58]:
```

```
array([[1649,   23,    5,   11,    6,    6,   18,   23,   45,    7],
       [  23, 1711,    5,   31,   29,   16,   22,   28,   48,    4],
       [  11,   17, 1778,    7,   48,   12,   42,   12,   29,    6],
       [  28,   46,    0, 1685,    6,   10,   10,   10,   40,   18],
       [  24,   30,   34,   12, 1738,   25,   25,    9,   35,    3],
       [  15,   10,   10,   19,   28, 1755,   11,   17,   48,   17],
       [  22,   45,   38,   14,   22,   18, 1654,   20,   42,   14],
       [  21,   35,    1,   17,   10,   12,   15, 1674,   34,    7],
       [  23,   21,   13,   20,   23,   22,   15,   16, 1746,   55],
       [  22,    7,    5,   22,   10,   17,    9,   20,   70, 1759]])
```

Рисунок 13 – Результаты модели на контрольной выборке

Точность реализованной модели составляет ~0,9. Применение динамически изменяемой скорости обучения не увеличило точность модели, однако видно, что графики обучения на тренировочной и валидационной выборке достаточно хорошо сходятся, что говорит о адекватности полученной модели.

Вывод:

В ходе выполнения лабораторной работы были изучены полносвязные слои. Был построена простейшая нейронная сеть с 5 слоями. С помощью методов подавляемых переобучение: сброс нейронов, регуляризация, а также динамически изменяемая скорость обучения (шаг при оптимизации), были подобраны параметры для реализации модели, при этом скор на валидационной и контрольной выборке составил ~0,9, а метрика F1 ~0,9 в обеих выборках, что говорит о том, что модель не переобучена на тренировочных данных.