Лабораторная работа №1 Логистическая регрессия в качестве нейронной сети

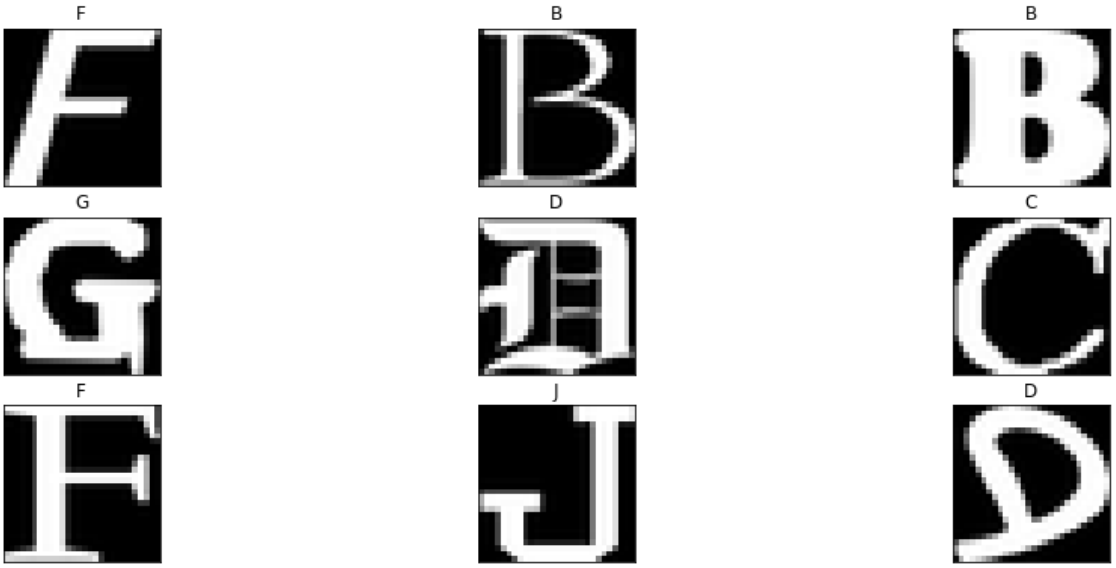
Цель

Построить классификатор с помощью логистической регрессии для набора данных notMNIST

Ход выполнения работы

Набор данных notMNIST состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A … J, соответственно). Обучающая выборка состоит из около 500 тыс. Изображений, а тестовая из примерно 19 тыс.

Загрузим набор данных и оторбазим несколько изображений:



Проверим, являются ли классы сбалансированными. Количество данных, принадлежащих кажлому классу, должно быть примерно одинаковым. Это необходимо для успешной классификации.

Для обучающей выборки: `{'I': 52912, 'G': 52912, 'A': 52909, 'F': 52912, 'H': 52912, 'J': 52911, 'C': 52912, 'D': 52911, 'E': 52912, 'B': 52911}`

Для тренировочной: `{'I': 1872, 'G': 1872, 'A': 1872, 'F': 1872, 'H': 1872, 'J': 1872, 'C': 1873, 'D': 1873, 'E': 1873, 'B': 1873}`

Изображений для каждой буквы примерно одинаково в обоих выборках.

Для корректного обучения модели необходимо, чтобы данные из тестовой и тренировочной выборок не пересекались. Поэтому такие дубликаты удалим. 12213 дублирующихся данных было удалено.

Преобразуем каждое изображение в вектор размером 28x28 = 784 и сформируем вектор ответов, содержащий информацию, какому классу соответствует изображение.

Разделим данные на выборки. Обучающая – 200 000 изображений, валидационная – 10 000. Тестовая выборка размером 19 000 изображений уже сформирована.

```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, tra
in_size=200000, test_size=10000)
```

Разделим данные на выборки. Обучающая – 200 000 изображений, валидационная – 10 000. Тестовая выборка размером 19 000 изображений уже сформирована. И удалим дубликаты для тестовой и валидационной выборок. 1355 дубликатов удалено.

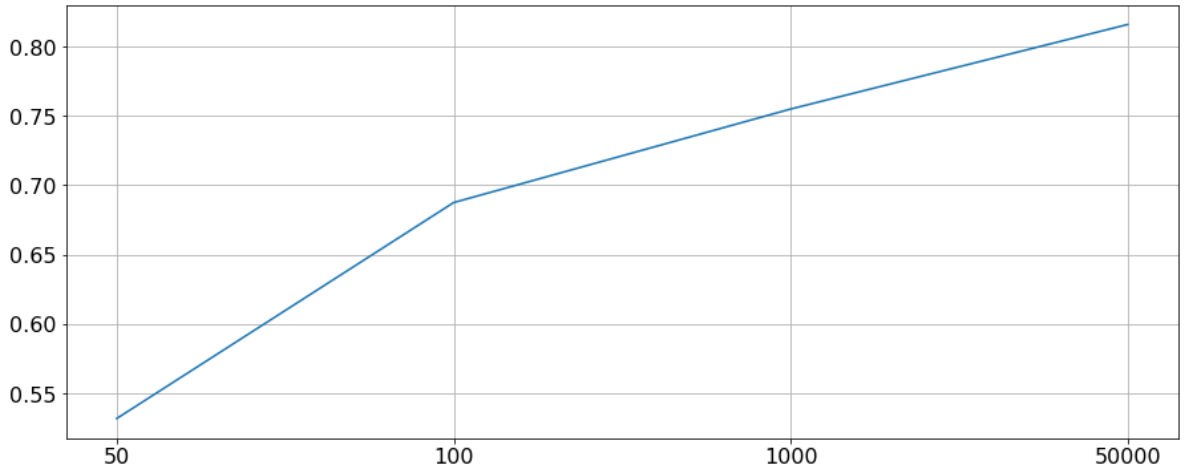
Теперь, когда данные подготовлены, можно преступать к построению классификатора. Обучение проведем на разных по размеру выборках: 50, 100, 1000 и 50000.

```
model = LogisticRegression(max_iter = 10000)
model.fit(x_train[count], y_train[count])
```

Результаты обучения на выборках выгледят следующим образом:

```
n = 50, accuracy = 0.59280, iterations = [70]
n = 100, accuracy = 0.70590, iterations = [91]
n = 1000, accuracy = 0.73810, iterations = [581]
n = 50000, accuracy = 0.81260, iterations = [7324]
```

Для визуализации отобразим на графике зависимость точности классификатора от размера обучающей выборки:



В работе построена модель для классификации изображений. В качестве классификатора использована логистическая регрессия, а в качетве данных использовался набор изображений notMnist. Проанализирована работа логистической регрессии на на выборках разного размера. Из представленного графика можно сделать вывод, что точность классификации растет с ростом обучающей выборки.

Лабораторная работа №2 Реализация глубокой нейронной сети

Цель

Построить полносвязную нейронную сеть для набора данных notMNIST. Использовать регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Использовать динамически изменяемую скорость обучения (learning rate).

Ход выполнения работы

Каждое изображение было преобразовано в векторное представление 28x28 и сформирован вектор ответов, содержащий информацию, какому классу соответствует изображение.

Построим полносвязную сеть с двумя скрытыми слоями и функцией активации relu. Входной слой использует Flatten для преобразования входных данных в вектор. Каждый скрытый слой содержит 512 нейронов. Выходной слой содержит 10 нейронов и использует функцию активации softmax.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

В качестве алгоритма оптимизации используем стохастический градиентный спуск с коэффициентом обучения LR = 0.002.

```
opt = tf.optimizers.SGD(lr=learning_rate)
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Выделим из обучающей выборки валидационную в размере 10% от исходных данных. Обучающую выборку будем разбирать на батчи по 100 элементов. Таким образом на каждой эпохе итерация обучения будет проходить на каждом батче.

```
model.fit(x=img_train, y=labels_train, epochs=50,
          validation_split=0.1, batch_size=batch_size)
```

Обучение происходило на 10 эпохах. После 10 эпох точность классификации на тестовой выборке составила 0.8078.

Для борьбы с переобучением используем регуляризацию. Для каждого скрытого слоя добавим регуляризацию L2 с регуляризационным фактором 0,001. Точность классификации на тестовой выборке составила 0.8149.

Следующим шагом к имеющейся модели добавим метод сброса нейронов. После каждого скрытого слоя добавим dropout-слой с процентом сброса 20.

```
model3 = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512,
                          activation='relu',
                          kernel_regularizer=regL2,
                          bias_regularizer=regL2,
                          bias_initializer=initializer,
                          kernel_initializer=initializer),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(512,
                          activation='relu',
                          kernel_regularizer=regL2,
                          bias_regularizer=regL2,
                          bias_initializer=initializer,
                          kernel_initializer=initializer),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Точность классификации на тестовой выборке составила 0.8767, что заметно выше предыдущих полученных результатов.

Используем другой метод оптимизации, который включает в себя динамически изменяемую скорости обучения. Используем оптимизатор Adagrad. Точность классификации на тестовой выборке составила 0.8254.

В ходе выполнения лабораторной работы были построены 4 нейронные сети для классификации изображений notMNIST. Каждая из которых состоит из двух скрытых слоев и имеет одинаковое количество нейронов. Во второй была добавлена регуляризация, в третьей метод сброса

нейронов, а четвертая отличается от третьей выбором оптимизатора с динамически изменяемой скоростью обучения.

Наиболее высокая точность была достинеута при использовании модели с регуляризацией и методом сброса нейронов. Точность составила 0.8767. Наименьшая точность у первой модели, что объясняется эффектом переобучения.

Лабораторная работа №3. Реализация сверточной нейронной сети

Цель

Построить нейронную сеть с двумя сверточными слоями, и одним полносвязным с нейронами с кусочно-линейной функцией активации для набора данных notMNIST. Заменить один из сверточных слоев на слой, реализующий операцию пулинга (Pooling). Реализовать классическую архитектуру сверточных сетей LeNet-5.

Ход выполнения работы

Каждое изображение было преобразовано в векторное представление 28x28 и сформирован вектор ответов, содержащий информацию, какому классу соответствует изображение.

Построим нейронную сеть с двумя сверточными слоями каждый из которых содержит 16 выходных фильтров, рамер ядра 5x5 и кусочно линейную функцию активации. Добавим полносвязный слой с 100 нейронами. Выходной слой содержит 10 нейронов и использует функцию активации softmax.

```
model = Sequential([
            layers.Conv2D(16, 5, activation='relu', input_shape=(28, 28
, 1)),
            layers.Conv2D(16, 5, activation='relu'),
            layers.Flatten(),
            layers.Dense(100, activation='relu'),
            layers.Dense(10, activation='softmax')
        ])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Точность классификации на тестовой выборке составила 0.9297.

Заменим второй сверточный слой на слой, реализующий операцию пулинга с ядром 2x2.

```
model2 = Sequential([
            layers.Conv2D(16, 5, activation='relu', input_shape=(28,28,
1), padding='same'),
            layers.MaxPool2D(pool_size=(2, 2), padding='same'),
            layers.Flatten(),
```

```
            layers.Dense(100, activation='relu'),
            layers.Dense(10, activation='softmax')
        ])
```

Точность классификации на тестовой выборке составила 0.9115.

Реализуем архитектуру сверточных сетей LeNet-5.

```
model3 = Sequential([
            layers.Conv2D(6, 5, activation='tanh', input_shape=(28,28,1
)),
            layers.AvgPool2D(pool_size=(2, 2), strides=(2, 2)),
            layers.Conv2D(16, 5, activation='tanh'),
            layers.AvgPool2D(pool_size=(2, 2), strides=(2, 2)),
            layers.Conv2D(120, 4, activation='tanh'),
            layers.Flatten(),
            layers.Dense(84, activation='tanh'),
            layers.Dense(10, activation='softmax')
        ])
```

Точность классификации на тестовой выборке составила 0.9424.

По результатам трех лабораторных работ можно сделать вывод, что наименьшую точность показывают логистическая регрессия и нейросетевые модели. Наибольшую точность показывают сверточные сети, что показывает применимость данного типа сетей для работы с изображениями.

Лабораторная работа №4. Реализация приложения по распознаванию номеров домов

Цель

Построить глубокую нейронную сеть для распознавания цифр в номерах домов.

Ход выполнения работы

В качестве синтетических данных используем архив MNIST. Для соответствия размеров изображений из архива MNIST и SVHN, изображения MNIST были приведены к разрешению 32x32, а изображения в SVHN были приведены к черно-белому виду.

В качестве модели для классификации построим глубокую сверточную сеть, состоящую из трех сверточных слоев и слоев, реализующий операцию пулинга, после каждого из них. За сверточными слоями следует полносвязный слой. Функция всех скрытых слоев – кусочно-линейная Выходной слой также полносвязный с функцией активации softmax.

```
model = keras.Sequential([
            keras.layers.Conv2D(16, 5, activation='relu', input_shape=x
_train.shape[1:], padding='same'),
            keras.layers.MaxPool2D(pool_size=(2, 2), padding = 'same'),
            keras.layers.Conv2D(32, 5, activation='relu', padding='same
'),
            keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
            keras.layers.Conv2D(64, 5, activation='relu', padding='same
'),
            keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
            keras.layers.Flatten(),
            keras.layers.Dropout(rate=0.1),
            keras.layers.Dense(100, activation='relu'),
            keras.layers.Dropout(rate=0.1),
            keras.layers.Dense(y_train.shape[1], activation='softmax')
        ])
```

Из тренировочной выборки выделим валидационную выборку в 10% исходной. Обучение нейронной сети произведем с помощью алгоритма

оптимизации Adam. Размер батча выбран 100. Количество эпох без улучшения, при которых сеть заканчивает обучаться равно 10.

В результате точность на тестовой выборке MNIST равна 0.9879, а на тестовой выборке SVHN – 0.18074.

Проведем обучение модели на данных из архива SVHN. Точность классификации этой модели на тестовых данных SVHN составила 0.89121.

Для улучшения точности модели она была дообучена на дополнительных данных из архива. Точность классификации модели обученной на дополнительной выборке составила 0.92959.

Лабораторная работа №5. Применение сверточных нейронных сетей (бинарная классификация)

Цель

Построить глубокую нейронную сеть с как минимум тремя сверточными слоями для набора данных DogsVsCats. Применить дополнение данных (data augmentation). Применить готовую нейронную сеть (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение.

Ход выполнения работы

Разделим данные на тернировочную, валидационную и тестовую выборки.

Построим глубокую нейронную сеть, состоящую из трех сверточных слоев и слоев, реализующих операцию пулинга, после каждого из них. За сверточными слоями следует полносвязный слой и dropout. Функция всех скрытых слоев – кусочно-линейная. Выходной слой полносвязный с функцией активации softmax.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=X.shape[1:]
))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1, activation='softmax'))
```

В результате точность на тестовой выборке составила 0.85.

Применим дополнение данных (data augmentation). Модель используем такую же, как описано выше. Точность на тестовой выборке составила 0.56.

Используем предобученную сеть MobileNet с весами imagenet. Добавим полносвязные слои и запустим обучение. Точность на тестовой выборке составила 0.92.

Лабораторная работа №6. Применение сверточных нейронных сетей (многоклассовая классификация)

Цель

Построить глубокую нейронную сеть со сверточными слоями для набора данных для распознования жестов. Применить дополнение данных (data augmentation). Применить готовую нейронную сеть (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение.

Ход выполнения работы

Разделим данные на тернировочную, валидационную и тестовую выборки.

Построим глубокую нейронную сеть, состоящую из четырех сверточных слоев и слоев, реализующих операцию пулинга, после них. За сверточными слоями следует полносвязный слой. Функция всех скрытых слоев – кусочно-линейная. Выходной слой полносвязный с функцией активации softmax.

```
model = Sequential([
    layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=(28
,28,1)),
    layers.Conv2D(32, kernel_size=5, activation='relu'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(500, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(25, activation='softmax')
])
```

В результате точность на тестовой выборке составила 0.9582.

Применим дополнение данных (data augmentation). Модель используем такую же, как описано выше. Точность на тестовой выборке составила 0.0591.

Используем предобученную сеть MobileNet с весами imagenet. Добавим полносвязные слои и запустим обучение. Точность не оказалась высокой и составила 0.0461. Вероятно это связано с тем, что модель обчучена на изображениях повседневных предметов и не умеет распозновать жесты.

Лабораторная работа №7. Рекуррентные нейронные сети для анализа текста

Цель

Построить и обучите двунаправленную рекуррентную сеть (LSTM или GRU) для набора данных с сайта imdb. Использовать индексы слов и их различное внутреннее представление (word2vec, glove). Поэкспериментировать со структурой сети (добавьте больше рекуррентных, полносвязных или сверточных слоев).

Ход выполнения работы

Набор данных состоит из тренировочных и тестовых. Преобразуем данные, чтобы обозначить соотвествие отзыва с его характеристикой (положительный или отрицательный). В результате получим тренировочный набор данных и тестовый. Преобразуем отзывы в наборы чисел. Для этого получим набор из 10000 наиболее используемых слов. И далее в каждом отзыве соответствующее слово заменим на число. Установим длинну отзыва равную 500, отзывы длиннее 500 обрежем до требуемой длинны, отзывы меньше 500 дополним до требуемой длинны.

Построим двунаправленную сеть LSTM.

```
model = tf.keras.Sequential([
  tf.keras.layers.Embedding(NUM_WORDS, EMBEDDING_SIZE),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)),
  tf.keras.layers.Dense(1, activation='sigmoid')])
```

Точность на тестовой выборке составила 0.845.

Используем glove для представления данных. Точность на тестовой выборке составила 0.813.

Добавим к модели еще два полносвязных слоя с 256 нейронами.

```
model1 = tf.keras.Sequential([
  tf.keras.layers.Embedding(NUM_WORDS, EMBEDDING_SIZE),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')])
```

Точность на тестовой выборке для данной модели составила 0.8524.

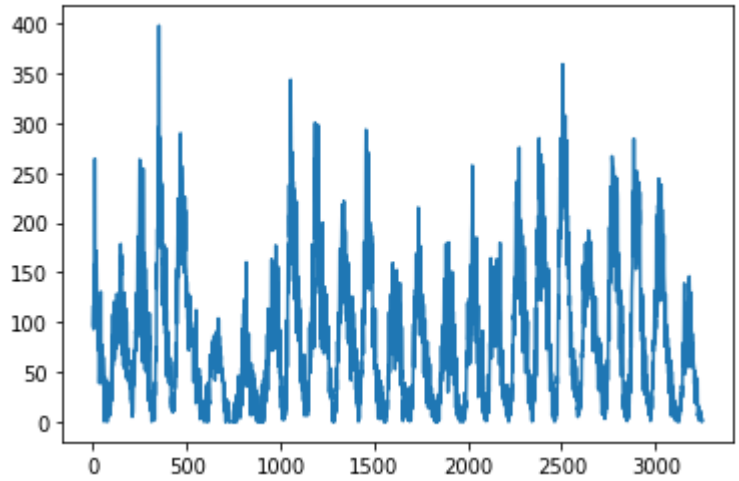
Лабораторная работа №8. Рекуррентные нейронные сети для анализа временных рядов

Цель

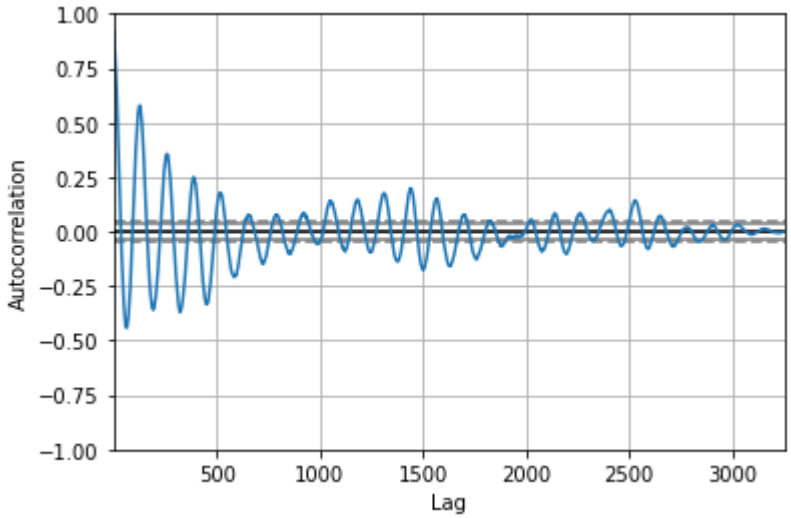
Применить модель ARIMA для прогнозирования значений временного ряда. Реализовать рекуррентную нейронную сеть.

Ход выполнения работы

Загрузим данные и изобразим ряд в виде графика.



Построим график автокорреляции



Разделим набор данных на обучающую и валижационную выборки.

Применим модель ARIMA. Средняя квадратичная ошибка при использовании данной модели составляет 622.374.

Построим рекуррентную нейронную сеть с двумя слоями LSTM.

```
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back), return_sequences=True))
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=5, batch_size=1, verbose=2)
```

Средняя квадратичная ошибка на тестовом наборе составляет 26.66.