Ознакомьтесь с набором данных MNIST.

Первым этапом работы с данными является их предобработка. Этот шаг заключается в разделении выборки на train-validation-test. Необходимо разделить данные на три части. Обучающая выборка. Как правило, большее количество обучающих данных поможет вашей нейронной сети лучше понять распределение ваших данных. Чем больше данных, тем лучше будет ваша обученная сеть. Всегда отдавайте приоритет этой части данных. Далее идут проверочные данные. Это часть данных, которые будут оцениваться в процессе обучения. Эти данные используются для оценки ошибки прогнозирования. Наконец, тестовая выборка. Это данные, используемые для оценки модели нейронной сети.

При разделении данных есть практическое правило. Если данных не так много, может быть, тысячи или десятки тысяч, тогда используйте следующее процентное соотношение 70–10–20: 70% данных для обучения, 10% - на проверку и 20% - на набор тестов. Однако, если у вас есть миллионы данных, то 90–5–5 - лучшая стратегия разделения. Или, если данных больше, возможно, вы можете использовать 98–1–1 в качестве стратегии разделения. Данные, предоставленные Keras, уже поделены между обучающими и тестовыми наборами: 60 тысяч для обучения и 10 тысяч для тестирования. Для проверки давайте возьмем 10% данных обучения. Таким образом, это будет 54K изображений для обучения, 6K изображений для проверки и 10K изображений для тестирования.

Импорт необходимых библиотек:

```
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
```

Загружаем данные:

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

Начинайте с простого. Используйте однослойный персептрон и оцените результат. Если предыдущий шаг недостаточно хорош, попробуйте расширить свою сеть и / или углубить. Добавьте несколько нейронов в ваш однослойный персептрон. Или добавьте еще один слой в существующую сеть. Оцените и, если это хорошо, приступайте к развертыванию. Если нет, то добавьте больше нейронов или слоев. Когда после добавления еще нескольких слоев в вашу сеть результаты все еще не так хороши, тогда нужно изменить архитектуру сети. Можно использовать сверточную нейронную сеть (CNN) для изображений или рекуррентную нейронную сеть для временных рядов и текстов.

Изображения имеют размер 28x28 и, следовательно, является двухмерными. Поскольку наш персептрон способен считывать только одномерные данные, преобразуем их.

```
x_train = x_train.reshape(x_train.shape[0], -1) / 255.0
x_test = x_test.reshape(x_test.shape[0], -1) / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Опишем архитектуру сети:

```
model = Sequential()
model.add(Dense(10, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Начнем обучение:

```
model.fit(x_train, y_train, epochs=10, validation_split=0.1)
```

Проанализируйте результат на проверочных и тестовых данных.

```
_, test_acc = model.evaluate(x_test, y_test) print(test_acc)
```

Опишем архитектуру сети 2:

model2 = Sequential()

```
model2.add(Dense( 50, input_dim=784, activation='relu'))
model2.add(Dense(10, activation='softmax'))
model2.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model2.fit(x_train, y_train, epochs=10, validation_split=0.1)
```

Проанализируйте результат на проверочных и тестовых данных.

Опишем архитектуру сети 3:

```
model3 = Sequential()
model3.add(Dense(50, input_dim=784, activation='relu'))
model3.add(Dense(50, activation='relu'))
model3.add(Dense(10, activation='softmax'))
model3.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model3.fit(x_train, y_train, epochs=10, validation_split=0.1)
```

Проанализируйте результат на проверочных и тестовых данных.

Сверточная нейронная сеть (CNN) - это нейронная сеть, которая может «видеть» подмножество наших данных. С помощью нее можно обнаружить образ на изображениях лучше, чем при работе с персептроном. Импортируем необходимые библиотеки чтобы создать сверточную нейронную сеть.

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten
import numpy as np
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train[:,:,:,np.newaxis] / 255.0
x_test = x_test[:,:,:,np.newaxis] / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

CNN читает наши изображения как есть. Если вы проверите x_train, у вас будет 60 000 x 28 x 28 x 1 данных. Почему x 1? Данные, которые нужно

прочитать CNN, должны выглядеть следующим образом: объем_данных x ширина x высота x количество_каналов. Высота и ширина говорят сами за себя. Каналы похожи на красный, зеленый или синий в изображениях RGB. В RGB имеется три канала, необходимо использовать данные x 3. Но поскольку мы работаем с изображениями в градациях серого, каждое значение на красном, зеленом или синем канале одинаково, и мы уменьшаем до одного канала.

Опишем архитектуру сети 4:

```
model4 = Sequential()
model4.add(Conv2D(filters=64, kernel_size=2, padding='same',
activation='relu', input_shape=(28,28, 1)))
model4.add(MaxPooling2D(pool_size=2))
model4.add(Flatten())
model4.add(Dense(10, activation='softmax'))
model4.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model4.fit(x_train, y_train, epochs=10, validation_split=0.1)
```

Проанализируйте результат на проверочных и тестовых данных.

Проведите сравнение работы четырех архитектур. Какая из архитектур показывает лучший результат и почему? Как можно улучшить результаты каждой из архитектур?

По вариантам (в соответствии с номером в списке) спроектируйте архитектуру, реализуйте алгоритм корректировки синаптических весов с помощью алгоритма обратного распространения ошибки, используя в качестве функции активации логистический сигмоид $f(net) = \frac{1}{1+\exp(-net)}$

Варианты индивидуальных заданий

№	Архитектура	Скорость обучения	Входной вектор	Матрицы синапсов 1 и 2 слоя	Эталонный выход
1	2-3-2	0.25	X = {0.2;-0.3}	W_1={0.5 0.2 -0.6; 0.7 -0.1 0.4; 0.3 -0.6 0.8} W_2={-0.8 0.2; 0.6 0.4; 0.5 -0.2; 0.1 -0.1}	Y = {0.4; 0.6}
2	3-3-2	0.25	X = {-0.2; 0.5; 0.7}	Начальные значения весов взять произвольным образом из интервала [-0.3 0.3]	Y = {0.2; -0.3}
3	2-3-3	0.25	X = {0.4; -0.4}	Начальные значения весов взять произвольным образом из интервала [-0.3 0.3]	Y = {0.3; -0.5; 0.8}
4	3-4-3	0.4	X={0.4; -0.7; 1.3}	Начальные значения весов взять произвольным образом из интервала [-0.3 0.3]	Y = {0.3; -0.5; 0.8}
5	2-2-1	0.5	X={0.4; -0.7}	W_1={0.9 0.5; 1.2 -0.7; -0.5 0.6} W_2={-0.7;-0.2;0.4}	Y={0.5}
6	3-2-1	0.2	X = {0.4;-0.7;1.3}	W_1={0.4 -0.7; 1.2 0.6; 0.1 0.5; -1.4 0.5} W_2={-0.8;0.3;0.5}	Y=0.7
7	3-4-2	0.25	X={0.1;-0.4;1.3}	W_1={0.1 0.3 -0.5 0.4; 0.4 -0.2 0.3 -0.3; -0.6 0.5 0.2 -0.1; -0.3 0.4 0.5 0.3}	Y={0.5;0.3}

				Остальные значения весов взять произвольным образом из интервала [-0.3 0.3]	
8	3-3-2	0.25	X={0.4;-0.8;0.2}	Начальные значения весов для первого слоя взять произвольным образом из интервала [-0.3 0.3] W_2={0.4 -0.7; 1.2 0.6; 0.1 0.5; -1.4 0.5}	Y={0.5;0.3}
9	2-4-2	0.35	X={0.1 -0.1}	Начальные значения весов взять произвольным образом из интервала [-0.3 0.3]	Y= {0.5;-0.5}
10	3-4-2	0.35	X={0.1;-0.4;-0,2}	Начальные значения весов взять произвольным образом из интервала [-0.3 0.3]	Y={0.4}

Контрольные вопросы:

1. Понятие нейронной сети, архитектуры
2. Обучение нейронной сети
3. Основные определения: скорость обучения, эпоха, нейрон, обучающая выборка, тестовая выборка, вариационная выборка, функция активации.
4. Алгоритм обратного распространения ошибки
5. Типы функций активации

6. Алгоритмы обучения