Задача классификации: MNIST

Линдеманн Никита

29 апреля 2019 г.

Машинное обучение

Обучение

Говорят, что компьютерная программа обучается решению задачи T на основе опыта E, если качество решения по метрике P растет по мере накопления опыта E (Mitchell «Machine learning», 1997).

Основные задачи машинного обучения

Самые популярные задачи

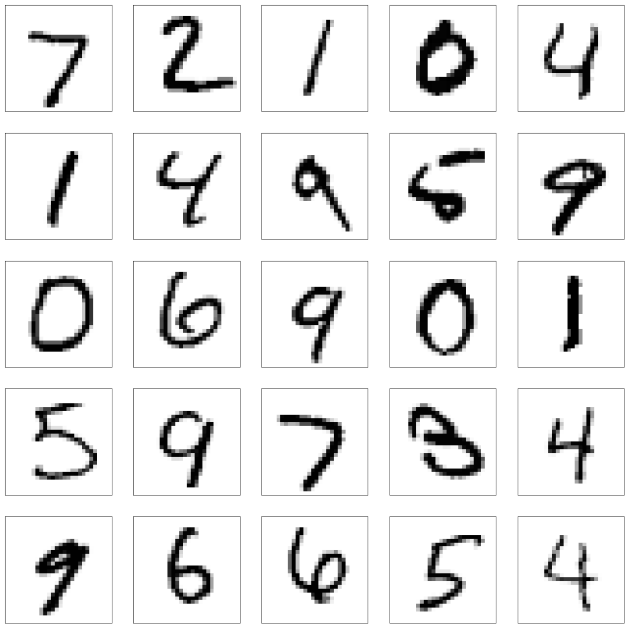
Среди задач T чаще всего выделяют:

1. Регрессию - предсказание непрерывной переменной (например, цены на недвижимость)
2. Классификацию - предсказание класса объекта (например, марки авто по фотографии)
3. Кластеризация - группировка данных (например, разбиение потока новостей на сюжеты)
4. Обучение с подкреплением - обучение игре на основе опыта (например, шахматы или го)

Постановка задачи

Дано

Набор MNIST, состоящий из обучающей (60 000) и тестовой (10 000) выборок фотографий рукописных чисел от 0 до 9.
Формат фотографий – 28×28 пикселей.



Первый шаг: учимся отличать 5 от 6

Подготовка данных

1. Отбираем из выборки только объекты с метками 5 и 6
2. Делим выборку на тестовую и обучающую в размере 33% на 66%
3. «Разворачиваем» каждое изображение обучающей выборки в вектор размера 1×784 (вектор признаков)
4. Нормируем векторы, деля их на 225 (яркость пикселя в RGB кодируется числом от 0 до 255)

Notebook

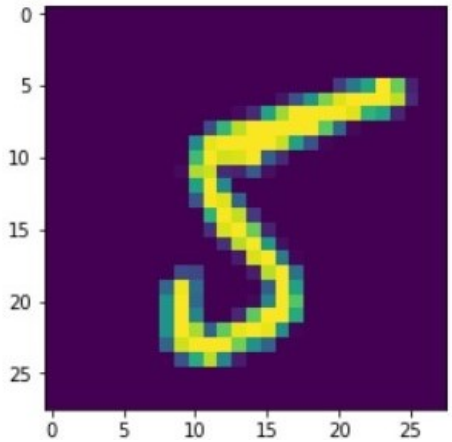
Загружаем датасет (только 5 и 6)

In [17]:
```
with open('train.pickle', 'rb') as f:
    dataset = pickle.load(f)

print(dataset['data'].shape)
print(dataset)
```

```
(10000, 784)
{'data': array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8), 'labels': array([5., 5., 6., ..., 6., 5., 6.])}
```

In [18]:
```
# Пример, как можно визуализировать данные
plt.imshow(dataset['data'][0].reshape(28,28))
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x1eb88a37b70>
```



Классификация

Задача бинарной классификации

В задаче бинарной классификации каждому объекту из обучающей выборки присваивается метка $+1$ или -1 в зависимости от класса. При этом мы хотим научиться оценивать вероятность $P_+(\vec{x})$ принадлежности объекта к классу 1.

Шанс

Шансом называют

$$OR_+ = \frac{P_+(\vec{x})}{P_-(\vec{x})} = \frac{P_+(\vec{x})}{1 - P_+(\vec{x})}.$$

Линейная модель

Логарифм шанса

В то время как значение вероятности лежит в диапазоне $[0, 1]$, значение шанса лежит в диапазоне $[0, \infty]$. А логарифм шанса лежит в диапазоне $[-\infty, +\infty]$ и может быть приближен линейной моделью:

$$\ln\left(OR_+(\vec{x})\right) = \vec{w}^T\vec{x}.$$

Таким образом, мы свели задачу классификации к задаче регрессии.

Сигмойда

Теперь осталось совместить два определения и получить модель для $P_+(\vec{x})$:

$$P_+(\vec{x}) = \frac{OR_+(\vec{x})}{1 + OR_+(\vec{x})} = \frac{\exp^{\vec{w}^T\vec{x}}}{1 + \exp^{\vec{w}^T\vec{x}}} = \frac{1}{1 + \exp^{-\vec{w}^T\vec{x}}} = \sigma(\vec{w}^T\vec{x}).$$

Задачу регрессии можно решать аналитически

МНК

Пусть у нас есть целевая переменная y, которую мы хотим научиться предсказывать, и k параметров $(x_1, x_2, \dots x_k)$, от которых y зависит линейно: $y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik} + \varepsilon_i$. Пусть так же нам удалось собрать n штук наблюдений – как ведет себя y в зависимости от параметров. Мы хотим предсказать $\tilde{y}$ так, чтобы сумма квадратов расстояний от y до $\tilde{y}$ была минимальной (по теореме Гаусса-Маркова найденное таким образом решение оптимально в классе несмещенных состоятельных оценок):

$$S(\beta_0, \dots \beta_k) = \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \dots - \beta_k x_{ik})^2 \to \min .$$

В терминах матриц и векторов

Обозначения

Введем обозначения:

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nk} \end{pmatrix}, \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad W = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix}.$$

Тогда задача минимизации функции S превращается в задачу нахождения минимума квадрата длины вектора $Y - XW$:

$$S = |Y - XW|^2 = (Y - XW)^T (Y - XW) \to \min.$$

Используем матанализ

Необходимое условие экстремума

Посчитаем градиент функции S и приравняем его к нулю:

$$\nabla S = \begin{pmatrix} -2\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_k x_{ik}) \\ -2\sum_{i=1}^{n} x_{i1}(y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_k x_{ik}) \\ \vdots \\ -2\sum_{i=1}^{n} x_{ik}(y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_k x_{ik}) \end{pmatrix} = -2X^T(Y - XW)$$

Решение

Аналитическое решение

Так как функция S не имеет максимума, то приравняв градиент к нулю, получим решение:

$$-2X^T(Y - XW) = 0,$$

$$X^T XW = X^T Y,$$

$$W = (X^T X)^{-1} X^T Y.$$

Почему так не делают

Очевидно большая вычислительная сложность: обращение матрицы – $O(n^3)$, умножение матриц – $O(n^{2.3})$. К тому же никто не гарантирует, что $X^T X$ не вырождена.

Метрика

Likelihood

Правдоподобием называют вероятность того, что данная выборка была семплирована из данного распределения. Если объекты независимы и одинаково распределены, то правдоподобие вычисляется как:

$$\mathcal{L}_\theta = \prod_{i=1}^{N} P_\theta(\vec{x}_i).$$

Logloss

Функция

$$L(\vec{x}, \vec{y}, \vec{w}) = \log(\mathcal{L}_\theta) = \sum_{i=1}^{N} \log(1 + \exp^{-y_i \vec{w}^T \vec{x}_i})$$

называется логистичской функцией потерь или сокращенно logloss.

Стохастичский градиентный спуск

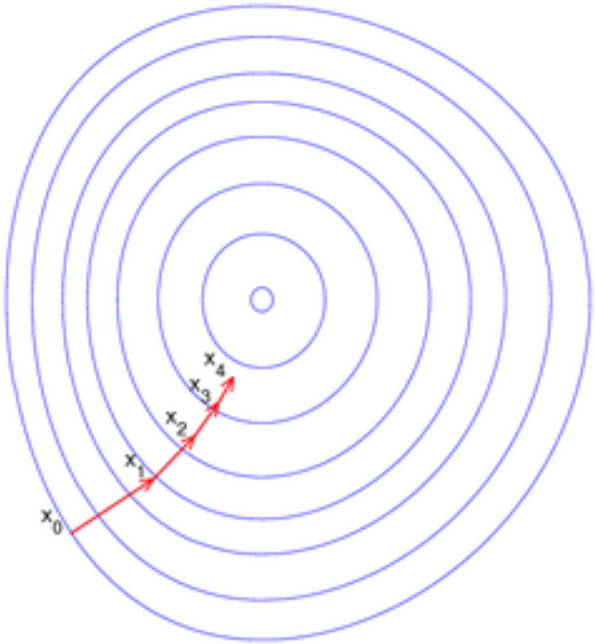
Обозначения

1. X - обучающее множество
2. Y - метки классов для обучающего множества
3. η - скорость обучения

Алгоритм

1. Инициализируем вектор весов $\vec{w}_0$ небольшими случайными значениями (инициализация Ксавье)
2. Выбираем равномерно случайный объект $\vec{x_i}$ с меткой y_i
3. Вычисляем градиент, используя регуляризацию, чтобы избежать переобучения: $\operatorname{grad} L = \nabla L(\vec{x_i}, y_i, \vec{w_t}) + \lambda \vec{w_t}$
4. Обновляем веса, используя константный learning rate $\vec{w}_{t+1} = \vec{w}_t - \eta \cdot \operatorname{grad} L$
5. Если условие остановки не выполнено, возвращаемся к шагу 2

Стохастичский градиентный спуск



Условие остановки алгоритма

На практике, в качестве условия остановки, часто используют изменение функции потерь на тестовой выборке.

Результат обучения модели

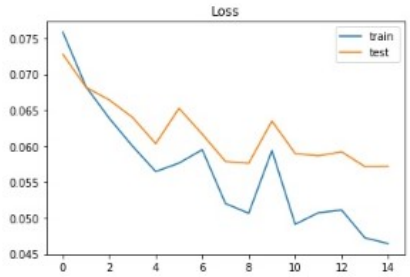
Итог

Обученная таким образом модель дает точность предсказания почти 98% на отложенной (тестовой) выборке.

Оцениваем результат

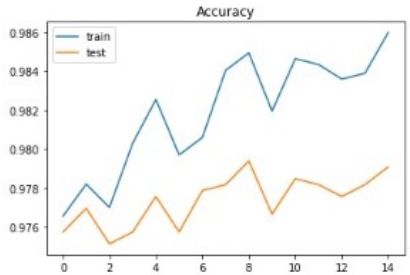
In [30]:
```
plt.title('Loss')
plt.plot(logloss_tr, label = 'train')
plt.plot(logloss_test, label = 'test')
plt.legend()
```

Out[30]: <matplotlib.legend.Legend at 0x1eb888c7a90>



In [31]:
```
plt.title('Accuracy')
plt.plot(acc_tr, label = 'train')
plt.plot(acc_test, label = 'test')
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x1eb8893b6a0>



Шаг 2: Учимся классифицировать все цифры

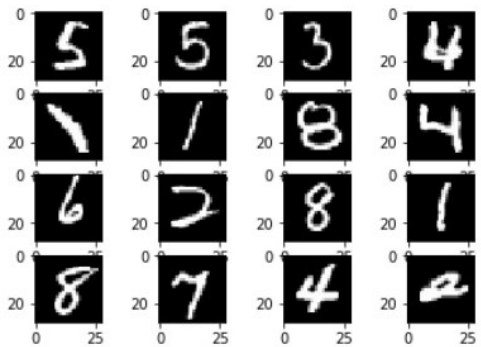
Подготовка данных

Поступаем аналогично первому шагу.

Загружаем датасет

In [52]:
```
encoder = OneHotEncoder()

with open('data_train.pickle','rb') as f:
    train_data = pickle.load(f)

X_train = train_data['data']
X_train = X_train.astype('float')
Y_train = train_data['target']
Y_train_oh = np.array(list(map(lambda x : encoder.transform(x), Y_train)))

print(type(train_data))
print(train_data['data'].shape)

n = 4
for i in range(n * n):
    plt.subplot(n, n, i + 1)
    I = train_data['data'][np.random.randint(0, X_train.shape[0]), :]
    I = I.reshape((28, 28))
    plt.imshow(I, cmap = 'gray')
```

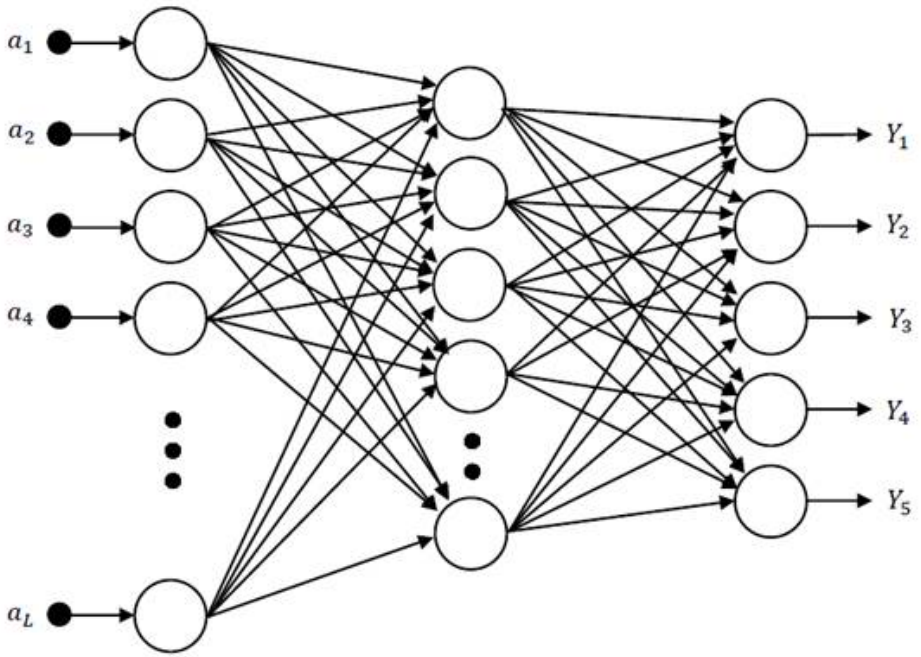
```
<class 'dict'>
(52500, 784)
```



Концепция

Процесс обучения

Нормированную обучающую выборку подаем на вход нейросети. Получаем вектор ответов, подаём его и реальные ответы в функцию ошибки. Получаем градиент функции ошибки и обновляем веса сети, используя метод обратного распространения ошибки.



Нейросеть

Устройство сети

1. Используем трехслойную полносвязную сеть
2. В классе Dense хранятся веса сети
3. Функция активации на обучаемых слоях – ReLU (ее легко считать и она не выходит на плато)
4. Функция активации на последнем слое – softmax (она переводит output в вектор вероятностей)
5. Функция ошибки – кросс-энтропия (аналог logloss в задаче бинарной классификации)
6. Все объединяется в класс нейронной сети с двумя методами forward и backward и в цикле обучается, применяя dropout

Notebook

In [50]:

```
class MnistNet:
    def __init__(self):
        self.d1_layer = Dense(784, 256)
        self.a1_layer = ReLu()
        self.drop1_layer = Dropout(0.5)

        self.d2_layer = Dense(256, 256)
        self.a2_layer = ReLu()
        self.drop2_layer = Dropout(0.5)

        self.d3_layer = Dense(256, 10)
        self.a3_layer = Softmax()

    def forward(self,x, train = True):
        net = self.d1_layer.forward(x)
        net = self.a1_layer.forward(net)
        net = self.drop1_layer.forward(net, train)

        net = self.d2_layer.forward(net)
        net = self.a2_layer.forward(net)
        net = self.drop2_layer.forward(net, train)

        net = self.d3_layer.forward(net)
        net = self.a3_layer.forward(net)

        return(net)

    def backward(self, dz, lr = 0.1):
        dz = self.a3_layer.backward(dz)
        dz = self.d3_layer.backward(dz, lr)

        dz = self.drop2_layer.backward(dz)
        dz = self.a2_layer.backward(dz)
        dz = self.d2_layer.backward(dz, lr)

        dz = self.drop1_layer.backward(dz)
        dz = self.a1_layer.backward(dz)
        dz = self.d1_layer.backward(dz, lr)

        return dz
```

Основные функции

Softmax

Это аналог сигмойды в задаче бинарной классификации. Так как мы говорим про задачу многоклассовой классификации, то на выходе сети мы ожидаем набор вероятностей p_i, такие что $\sum p_i = 1$:

$$\text{softmax}_i = p_i = \frac{e^{a_i}}{\sum_j e^{a_j}}.$$

Loss функция

$$H(p, q) = -\sum_x p(x) \log q(x).$$

Нелинейная функция активации ReLU

$$\text{ReLU} = \max(0, x).$$

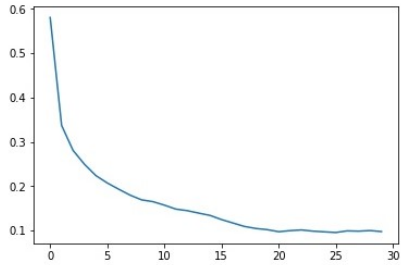
Результат обучения

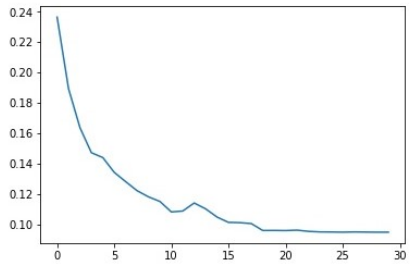
Спустя 3 часа

Точность на тестовом множестве 95.6%.

Оцениваем результат

In [57]:
```
plt.plot(L_train, label = "train")
plt.show()
plt.plot(L_test, label = "test")
plt.show()
```





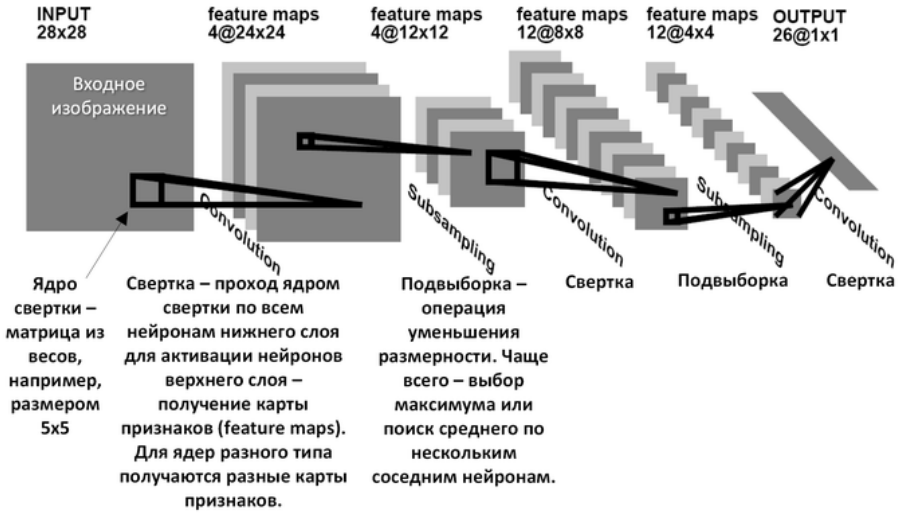
In [58]:
```
L_e_acc = 0.
for i in range(test_x.shape[0]):
    x = test_x[i]
    y = test_y_oh[i]
    y_h = net.forward(x, train = False)
    L = loss.forward(y, y_h)
    L_e_acc += L
L_e_acc /= test_y_oh.shape[0]
acc = compute_acc(test_x, test_y, net)

print('accuracy:', acc, 'loss:', L_e_acc)
```

```
accuracy: 0.9563809523809523 loss: 0.09473860835281571
```

Решение задачи с помощью библиотек глубокого обучения

Отличия

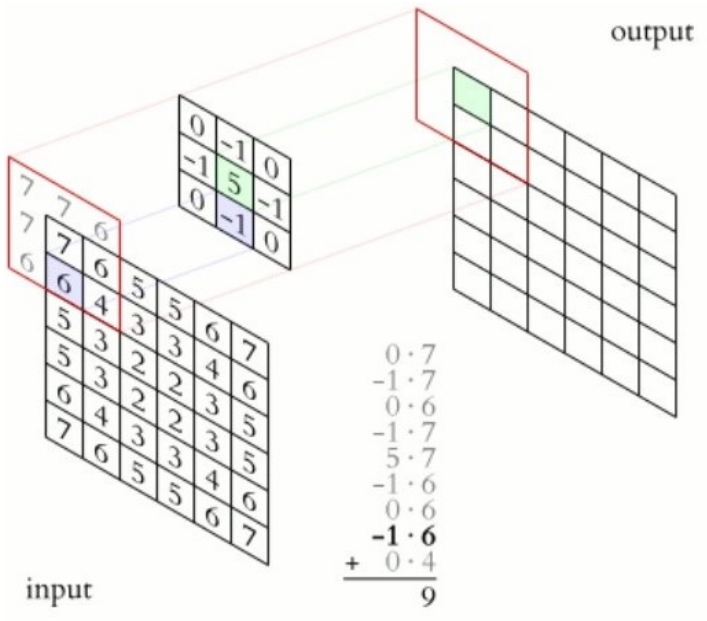
До этого мы теряли информацию о топологии изображения, хотя это существенная информация. Так же мы не использовали никакие библиотеки для построения и работы с нейронными сетями (типа Sklearn, TensorFlow, Keras).



Сверточные нейронные сети

Концепция глубокого обучения

Для обучения сверточных нейронных сетей используют глубокое обучение. Эта техника (использующая некоторые особенности зрительной коры) очень успешно применяется для распознования образов. Свое название архитектура сети получала из-за спецефической операции свертки с ядром (матрицы небольшого размера).



Результаты обучения

TensorFlow

Два сверточных слоя с 32 и 64 фильтрами соответственно и полносвязный слой дают точность 99.1%.

TensorFlow

Результаты обучения

Keras

Аналогичная архитектура дает точность 99.15 %

Спасибо за внимание

Использованаая литература

1. А.И. Кибзун, Е.Р. Горяинова, А.В. Наумов, А.Н. Сиротин – Теория вероятностей и математическая статистика
2. С. Николенко, А. Кадурин, Е. Архангельская – Глубокое обучение. Погружение в мир нейронных сетей
3. Джоэл Грас – Data Science. Наука о данных с нуля