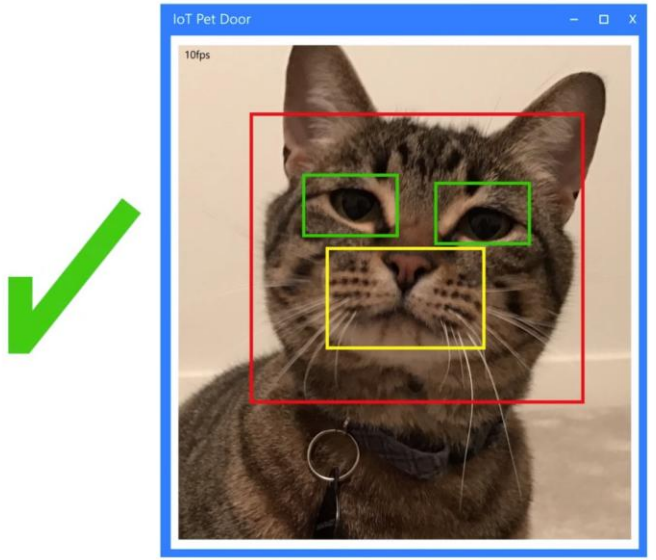
Глубокая нейронная сеть
Обратное распространение

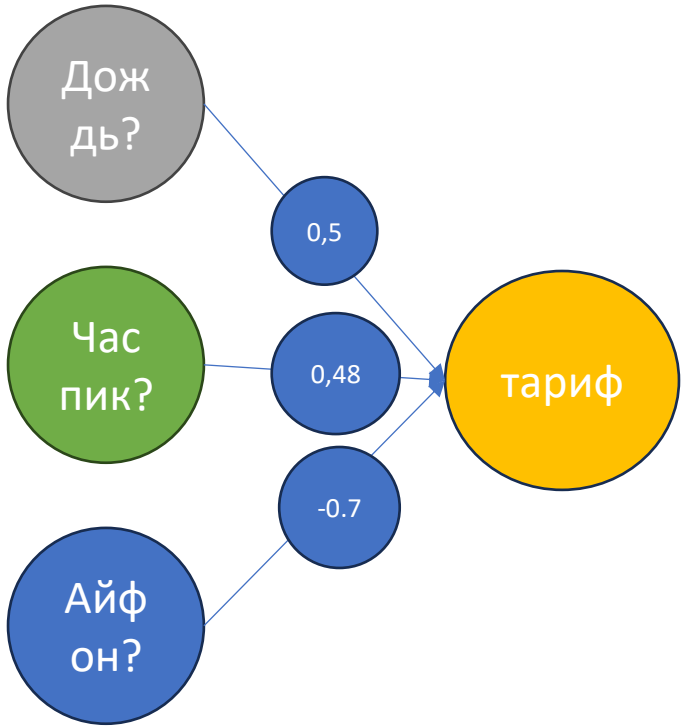
Лекция 3



Задача

В другой стране выяснить как образуется повышенный тариф на такси

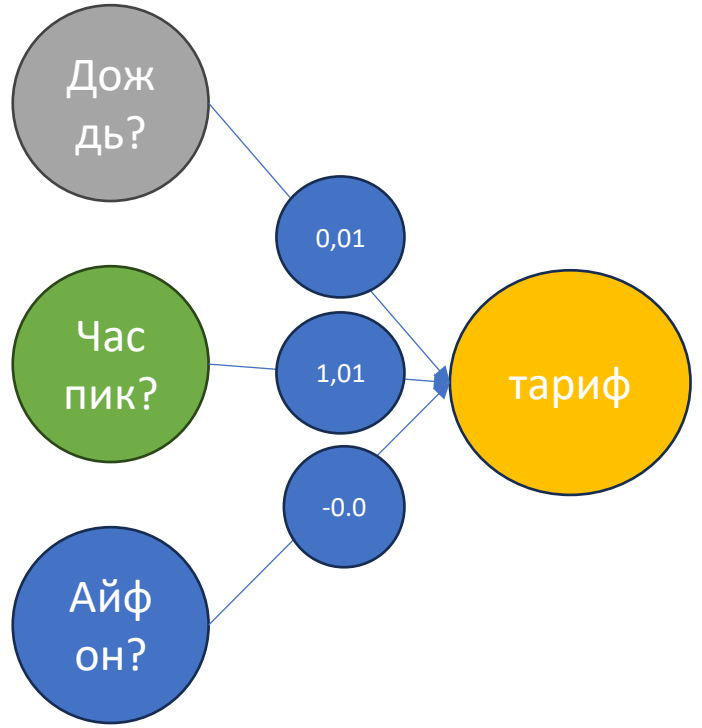
Дождь?	Час пик?	Айфон?	Тариф
+	-	+	стандартный
-	+	+	повышенный
-	-	+	стандартный
+	+	+	повышенный
-	+	+	повышенный
+	-	+	стандартный



Обучение однослойной нейронной сети на полном наборе данных

```python
import numpy as np
weights = np.array([0.5, 0.48, -0.7])
alpha = 0.1

factors = np.array([[1, 0, 1],
                    [0, 1, 1],
                    [0, 0, 1],
                    [1, 1, 1],
                    [0, 1, 1],
                    [1, 0, 1]])
higt_or_low_rate = np.array([0, 1, 0, 1, 1, 0])

input = factors[0]
true = higt_or_low_rate[0]

for i in range(40):
    error_rates = 0
    for row in range(len(higt_or_low_rate)):
        input = factors[row]
        true = higt_or_low_rate[row]
        prediction = input.dot(weights)
        error = (true - prediction)**2
        error_rates += error
        delta = prediction - true
        weights = weights - (alpha * (input*delta))
        print("Prediction: " + str(prediction))
    print("Error: " + str(error_rates) + "\n")
```

```
Prediction: -0.19999999999999996
Weights: [ 0.52  0.48 -0.68]
Prediction: -0.19999999999999996
Weights: [ 0.52  0.6  -0.56]
Prediction: -0.5599999999999999
Weights: [ 0.52   0.6   -0.504]
Prediction: 0.616
Weights: [ 0.5584  0.6384 -0.4656]
Prediction: 0.17279999999999995
Weights: [ 0.5584   0.72112 -0.38288]
Prediction: 0.17552
Weights: [ 0.540848  0.72112  -0.400432]
Error: 2.6561231104


Prediction: -0.0022410273814405194
Weights: [ 0.01514099  1.01480835 -0.01693381]
Prediction: 0.9978745386023716
Weights: [ 0.01514099  1.0150209  -0.01672126]
Prediction: -0.016721264429884933
Weights: [ 0.01514099  1.0150209  -0.01504914]
Prediction: 1.0151127459893812
Weights: [ 0.01362971  1.01350962 -0.01656041]
Prediction: 0.9969492081270097
Weights: [ 0.01362971  1.0138147  -0.01625533]
Prediction: -0.00262561933297829
Weights: [ 0.01389228  1.0138147  -0.01599277]
Error: 0.0005337367732848793
```



Корреляция на основе понижающего и повышающего давления

1	0	1		0
0	1	1		1
0	0	1		0
1	1	1		1
0	1	1		1
1	0	1		0

-	0	-		0
0	+	+		1
0	0	-		0
+	+	+		1
0	+	+		1
-	0	-		0

Прогноз – это взвешенная сумма входных значений. Обучение придает дополнительную значимость входам, которые **коррелируют с выходами**, оказывая **повышающее** давление на эти веса, и уменьшает значимость входов, **не коррелирующих с выходами**, оказывая **понижающее** давление.

Т.Е. ПРОЦЕСС ОБУЧЕНИЯ ОТЫСКИВАЕТ КОРРЕЛЯЦИЮ МЕЖДУ ДВУМЯ НАБОРАМИ ДАННЫХ

Пограничные случаи: переобучение и конфликт давлений

```
weights = np.array([0.5, 0.48, -0.5])
alpha = 0.1

factors = np.array([[1, 0, 1],
                    [0, 1, 1],
                    [0, 0, 1],
                    [1, 1, 1],
                    [0, 1, 1],
                    [1, 0, 1]])
higt_or_low_rate = np.array([0, 1, 0, 1, 1, 0])

input = factors[0]
true = higt_or_low_rate[0]
```

Ошибка является ОБЩЕЙ для всех весов. Если какая-то комбинация весов *случайно* создаст идеальную корреляцию, то обучение остановится.

Нам нужно, чтобы нейронная сеть не просто **запомнила** данные, а попыталась их **обобщить**.

-	0	-		0
0	+	+		1
0	0	-		0
+	+	+		1
0	+	+		1
-	0	-		0

В ходе обучения другие узлы поглощают часть ошибки и, соответственно, часть корреляции, т.е. заставляют сеть генерировать прогноз с **умеренной** степенью корреляции. Это уменьшает ошибку. Другие веса будут пытаться скорректировать свое значение, чтобы правильно предсказать то, что осталось.

Косвенная корреляция

1	0	1	1
0	1	1	1
0	0	1	0
1	1	1	0

+	0	+	1
0	+	+	1
0	0	-	0
-	-	-	0

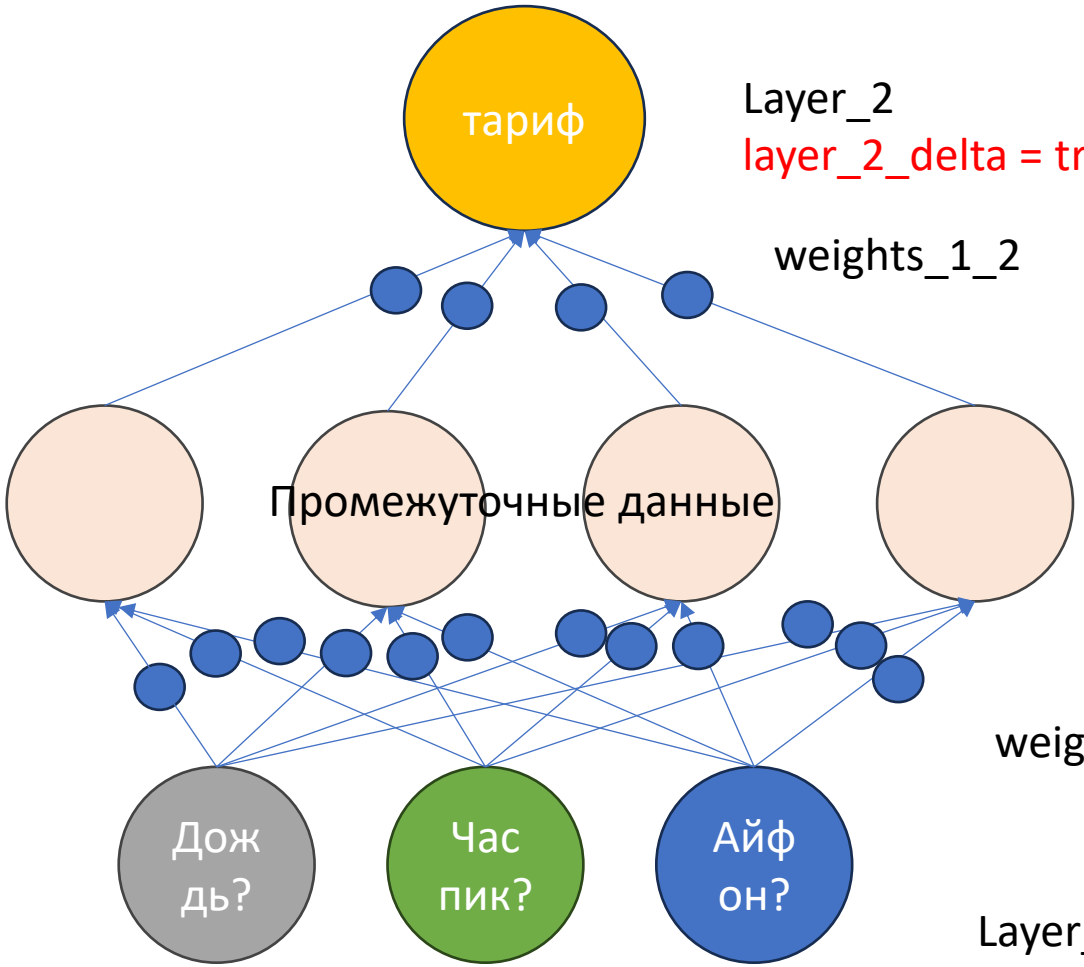
ПРОБЛЕМА ДЛЯ НАШЕЙ НЕЙРОННОЙ СЕТИ!!!

УТОЧНЕНИЕ К РАНЕЕ СКАЗАННОМУ: В действительности нейросеть ищет корреляцию не между входными данными и выходом, а **между своими входными и выходными *слоями***.

Нет явной корреляции? Создадим косвенную корреляцию, используя две сети: первая создаст промежуточный Набор данных, имеющий ограниченную корреляцию с выходными данными, а вторая используют эту ограниченную корреляцию для правильного прогнозирования результата.

Такая вот ПОДТАСОВКА ☺

Создание корреляции: объединение нейронных сетей в стек



Layer_2
layer_2_delta = true - prediction

weights_1_2

Layer_1
layer_1_delta = layer_2_delta.dot(weights_1_2)

weights_0_1

Обратное
распространение
ошибки сигнала delta

Layer_0

Линейность и нелинейность

5*10*2 = 100 1*0,25*0,9 = 0,225

5*20 = 100 1*0,225 = 0,225

Для любой трехслойной сети существует двухслойная сеть с идентичным поведением.
ТАКАЯ ТРЕХСЛОЙНАЯ СЕТЬ НЕ СОЙДЕТСЯ.

А нам нужно, чтобы узлы в среднем слое *иногда могли зависеть* от узлов во входном слое, а *иногда нет*.

Эпизодическая (условная) корреляция: ReLU.

Отключение узла, когда значение оказывается ниже нуля задается простой функцией ReLU, что придает трехслойной нейронной сети – нелинейность.

Обратное распространение в коде

```
weights = np.array([0.5, 0.48, -0.5])
alpha = 0.1

factors = np.array([[1, 0, 1],
                    [0, 1, 1],
                    [0, 0, 1],
                    [1, 1, 1],
                    [0, 1, 1],
                    [1, 0, 1]])
higt_or_low_rate = np.array([0, 1, 0, 1, 1, 0])

input = factors[0]
true = higt_or_low_rate[0]
```

Возвращает x, если x>0, иначе 0

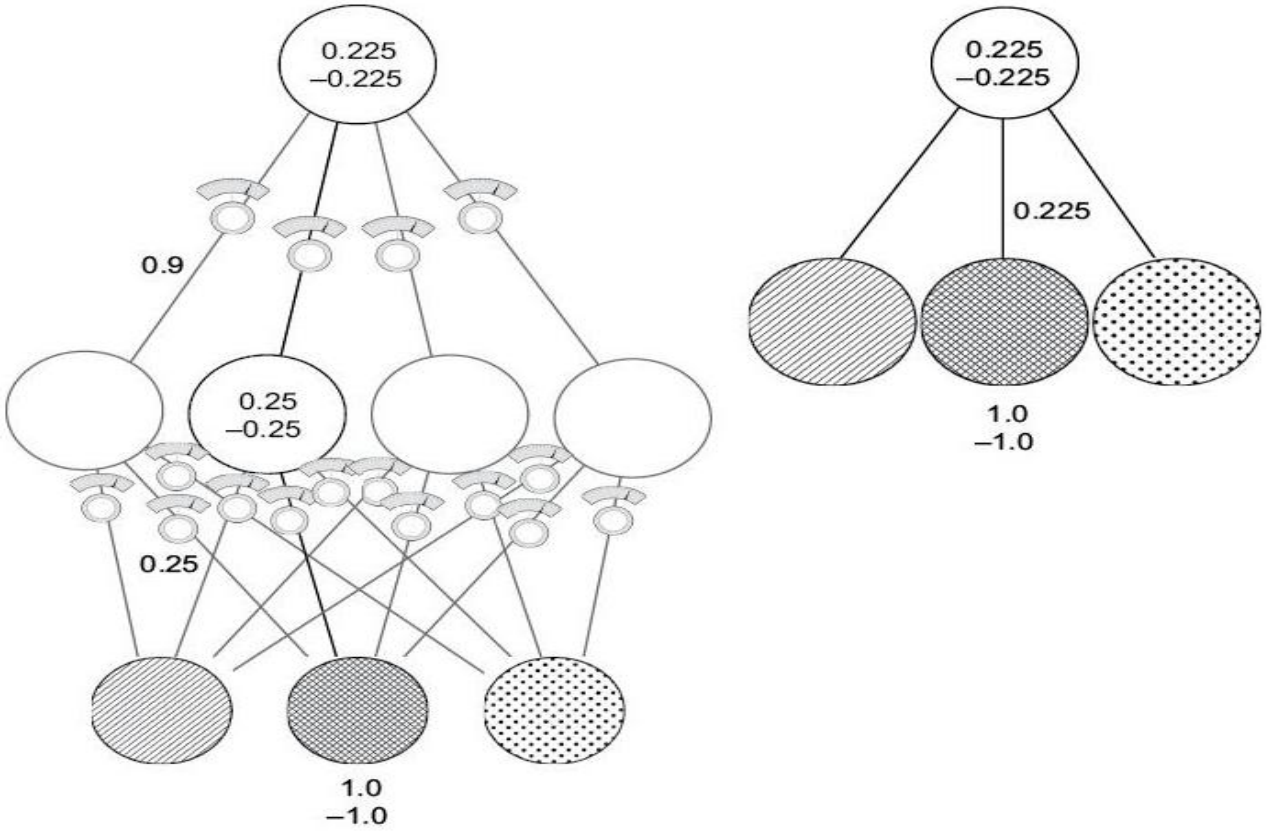
Возвращает 1, если output >0, иначе 0

$l_1 = relu(l_0 W_0)$

$l_2 = l_1 W_1$ $l_2 = relu(l_0 W_0) W_1$

Регуляризация и группировка: усиление сигнала и фильтрация шума

Обучение на примере трехслойной нейронной сети для классификации набора данных MNIST



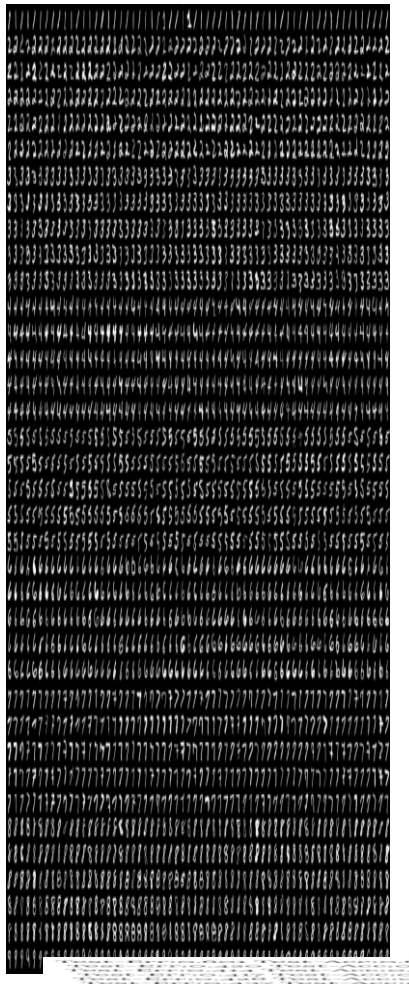
1. Возьмем 1000 примеров
2. Обучим нашу трехслойную сеть
3. На 350 итерации полученная точность составит 100% с ошибкой СКО 0,108.
4. Теперь возьмем другую тысячу примеров и посмотрим на предсказательную силу нашей обученной сети.
5. СКО составила 0,653, а точность всего 70,73% - *точность проверки*.

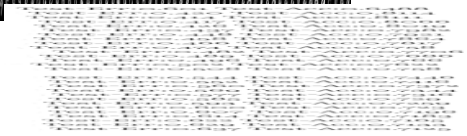
Запоминание и обобщение

Обучающая выборка

```
import numpy as np

np.random.seed(1)
def relu(x):
    return (x > 0) * x

def relu2deriv(output):
    return output>0

alpha = 0.2
hidden_size = 4

weights_0_1 = 2*np.random.random((2, hidden_size))-1
weights_1_2 = 2*np.random.random((hidden_size, 1))-1

for iteratin in range(60):
    layer_2_error = 0
    for i in range(len(factors)):
        layer_0 = factors[i:i+1]
        layer_1 = relu(np.dot(layer_0, weights_1_2)
        layer_2 = np.dot(layer_1, weights_1_2)

        layer_2_error += np.sum((layer_2 - higt_or_low_rate[i:i+1])**2)

        layer_2_delta = (higt_or_low_rate[i:i+1] - layer_2)
        layer_1_delta = layer_2_delta.dot(weights_1_2.T)*relu2deriv(layer_1)

        weights_1_2 += alpha*layer_1.T.dot(layer_2_delta)
        weights_0_1 += alpha*layer_0.T.dot(layer_1_delta)
```

Тестовая

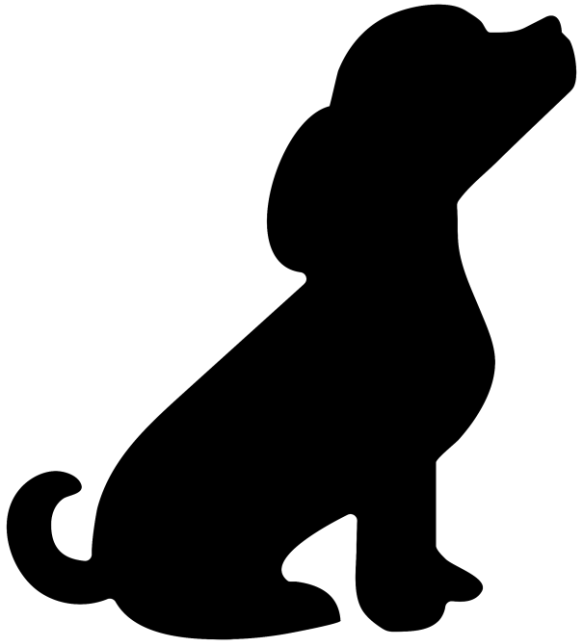




Запомнить 1000 изображений проще, чем обобщить все изображения

Переобучение нейронных сетей

Если нейронную сеть обучать слишком долго, точность прогноза может ухудшиться!



Что на изображении полезный сигнал, а что шум?

Регуляризация

Это подмножество методов, помогающих обобщить модель для распознавания новых данных (вместо простого запоминания обучающих примеров).

Простые типы регуляризации:

- Ранняя остановка с использованием проверочного набора данных
- Прореживание (дропаут)
- Пакетный градиентный спуск

Регуляризация: прореживание

Суть: выключение (установка в ноль) случайных нейронов в процессе обучения.

Прореживание заставляет большую сеть действовать подобно маленькой сети, обучая случайно выбираемые подразделы, а маленькие сети не подвержены переобучению, т.к. не обладают необходимой выразительной силой.

Прореживание – это форма обучения множества сетей и их усреднения.

ИДЕЯ 1: Даже при том, что большие, нерегуляризированные нейронные сети изучают шум, весьма маловероятно, что это будет один и тот же шум.

ИДЕЯ 2: Нейронные сети, даже при том, что имею случайное начальное состояние, все еще начинают изучение данных с самых общих закономерностей, и только потом начинают изучать шум.

```
dropout_mask = np.random.randint(2, size=layer_1.shape)
layer_1 *= dropout_mask*2
```

Прореживание в коде

dropout_mask = np.random.randint(2, size=layer_1.shape) 50%-распределение Бернулли
layer_1 *= dropout_mask*2

I:0 Test-Err:0.641 Test-Acc:0.6333 Train-Err:0.891 Train-Acc:0.413
I:10 Test-Err:0.458 Test-Acc:0.787 Train-Err:0.472 Train-Acc:0.764
I:20 Test-Err:0.415 Test-Acc:0.8133 Train-Err:0.430 Train-Acc:0.809
I:30 Test-Err:0.421 Test-Acc:0.8114 Train-Err:0.415 Train-Acc:0.811
I:40 Test-Err:0.419 Test-Acc:0.8112 Train-Err:0.413 Train-Acc:0.827
I:50 Test-Err:0.409 Test-Acc:0.8133 Train-Err:0.392 Train-Acc:0.836
I:60 Test-Err:0.412 Test-Acc:0.8236 Train-Err:0.402 Train-Acc:0.836
I:70 Test-Err:0.412 Test-Acc:0.8033 Train-Err:0.383 Train-Acc:0.857
I:80 Test-Err:0.410 Test-Acc:0.8054 Train-Err:0.386 Train-Acc:0.854
I:90 Test-Err:0.411 Test-Acc:0.8144 Train-Err:0.376 Train-Acc:0.868
I:100 Test-Err:0.411 Test-Acc:0.7903 Train-Err:0.369 Train-Acc:0.864
I:110 Test-Err:0.411 Test-Acc:0.8003 Train-Err:0.371 Train-Acc:0.868
I:120 Test-Err:0.402 Test-Acc:0.8046 Train-Err:0.353 Train-Acc:0.857
I:130 Test-Err:0.408 Test-Acc:0.8091 Train-Err:0.352 Train-Acc:0.867
I:140 Test-Err:0.405 Test-Acc:0.8083 Train-Err:0.355 Train-Acc:0.885
I:150 Test-Err:0.404 Test-Acc:0.8107 Train-Err:0.342 Train-Acc:0.883
I:160 Test-Err:0.399 Test-Acc:0.8146 Train-Err:0.361 Train-Acc:0.876
I:170 Test-Err:0.404 Test-Acc:0.8074 Train-Err:0.344 Train-Acc:0.889
I:180 Test-Err:0.399 Test-Acc:0.807 Train-Err:0.333 Train-Acc:0.892
I:190 Test-Err:0.407 Test-Acc:0.8066 Train-Err:0.335 Train-Acc:0.898

I:200 Test-Err:0.405 Test-Acc:0.8036 Train-Err:0.347 Train-Acc:0.893
I:210 Test-Err:0.405 Test-Acc:0.8034 Train-Err:0.336 Train-Acc:0.894
I:220 Test-Err:0.402 Test-Acc:0.8067 Train-Err:0.325 Train-Acc:0.896
I:230 Test-Err:0.404 Test-Acc:0.8091 Train-Err:0.321 Train-Acc:0.894
I:240 Test-Err:0.415 Test-Acc:0.8091 Train-Err:0.332 Train-Acc:0.898
I:250 Test-Err:0.395 Test-Acc:0.8182 Train-Err:0.320 Train-Acc:0.899
I:260 Test-Err:0.390 Test-Acc:0.8204 Train-Err:0.321 Train-Acc:0.899
I:270 Test-Err:0.382 Test-Acc:0.8194 Train-Err:0.312 Train-Acc:0.906
I:280 Test-Err:0.396 Test-Acc:0.8208 Train-Err:0.317 Train-Acc:0.9
I:290 Test-Err:0.399 Test-Acc:0.8181 Train-Err:0.301 Train-Acc:0.908

Прореживание тормозит обучение и отсеивает шум!

Полный, пакетный и стохастический градиентный спуск

Характеристика	Полный (Batch) GD	Стохастический (SGD)	Мини-пакетный (Mini-batch) GD
Размер данных на шаг	Весь датасет (N)	Один пример (1)	Небольшая группа (k, напр. 32)
Скорость на шаг	Медленно	Очень быстро	Быстро (оптимизировано для GPU)
Стабильность сходимости	Высокая, плавная	Низкая, шумная	Умеренная, небольшие колебания
Векторизация	Полная	Нет	Полная (эффективна)
Память	Требует много	Требует мало	Требует умеренно
Застревание в минимумах	Высокий риск	Низкий риск	Умеренный риск

Пакетный градиентный спуск

Этот метод увеличивает скорость обучения и улучшает сходимость

I:200 Test-Err:0.405 Test-Acc:0.8036 Train-Err:0.347 Train-Acc:0.893
I:210 Test-Err:0.405 Test-Acc:0.8034 Train-Err:0.336 Train-Acc:0.894
I:220 Test-Err:0.402 Test-Acc:0.8067 Train-Err:0.325 Train-Acc:0.896
I:230 Test-Err:0.404 Test-Acc:0.8091 Train-Err:0.321 Train-Acc:0.894
I:240 Test-Err:0.415 Test-Acc:0.8091 Train-Err:0.332 Train-Acc:0.898
I:250 Test-Err:0.395 Test-Acc:0.8182 Train-Err:0.320 Train-Acc:0.899
I:260 Test-Err:0.390 Test-Acc:0.8204 Train-Err:0.321 Train-Acc:0.899
I:270 Test-Err:0.382 Test-Acc:0.8194 Train-Err:0.312 Train-Acc:0.906
I:280 Test-Err:0.396 Test-Acc:0.8208 Train-Err:0.317 Train-Acc:0.9
I:290 Test-Err:0.399 Test-Acc:0.8181 Train-Err:0.301 Train-Acc:0.908

I:250 Test-Err:0.429 Test-Acc:0.8028 Train-Err:0.386 Train-Acc:0.843
I:260 Test-Err:0.431 Test-Acc:0.8038 Train-Err:0.394 Train-Acc:0.843
I:270 Test-Err:0.428 Test-Acc:0.8014 Train-Err:0.384 Train-Acc:0.845
I:280 Test-Err:0.430 Test-Acc:0.8067 Train-Err:0.401 Train-Acc:0.846
I:290 Test-Err:0.428 Test-Acc:0.7975 Train-Err:0.383 Train-Acc:0.851

Сети передаются N-примеров (в нашем случае 100), усредняя Корректирующие значения для весов по всем 100 примерам.

- Точность на контрольных данных изменяется более плавно за счет усреднения (меньше шума)
- Код выполняется намного быстрее, т.к. скалярное произведение пакетами выполняется быстрее.
- Можно коэффициент alpha увеличить в несколько раз.