Искусственные нейронные сети: основы практического применения

Лекция 3

Крощенко А.А.

Брестский государственный технический университет

30.05.2017

Задачи, решаемые машинным обучением

Пример небинарной классификации

Часто задача классификации представляет собой т.н. **мультиклассовый случай**.
Мультиклассовость предполагает наличие более 2-х классов, которым могут принадлежать распознаваемые образы.
Типичный пример мультиклассовой выборки – **MNIST (выборка рукописных цифр)**. Она содержит 70.000 образов (формат 28X28 пикселей), принадлежащих 10 классам (0-9). Традиционно 60.000 используются для обучения, 10.000 – для тестирования.

Figure 1: Пример образов из выборки MNIST

(https://www.npmjs.com/package/mnist)

Исходные данные

Итак, необходимо обучить нейронную сеть классификации образов из выборки MNIST. В качестве модели использовалась нейронная сеть вида:

784-800{Logistic}-800{Logictic}-10{Logistic}

Скорость обучения – 0.1, моментный параметр – 0.5... 0.9, размер мини-батча – 100, максимальное количество эпох обучения – 100, without weight-loss.

Выбор целевой минимизируемой функции

- MSE (Mean Squared Error)

$$\frac{1}{2N}\sum_{i=1}^{N}(\tilde{y}_i - y_i)^2$$

- CE (Cross-entropy Error)

$$-\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\tilde{y}_i) + (1 - y_i)\log(1 - \tilde{y}_i))$$

Оказывается, выбор минимизируемой функции оказывает ощутимое влияние на результат. Для обучения автоэнкодера рекомендуется использовать MSE, для обучения классификатора – CE.

Показатель эффективности

Классификация производилась следующим образом. Вначале определялся k-тый нейрон, выходное значение которого было максимальным для заданного образа s:

$$k_s = \arg\max_j y_j^s$$

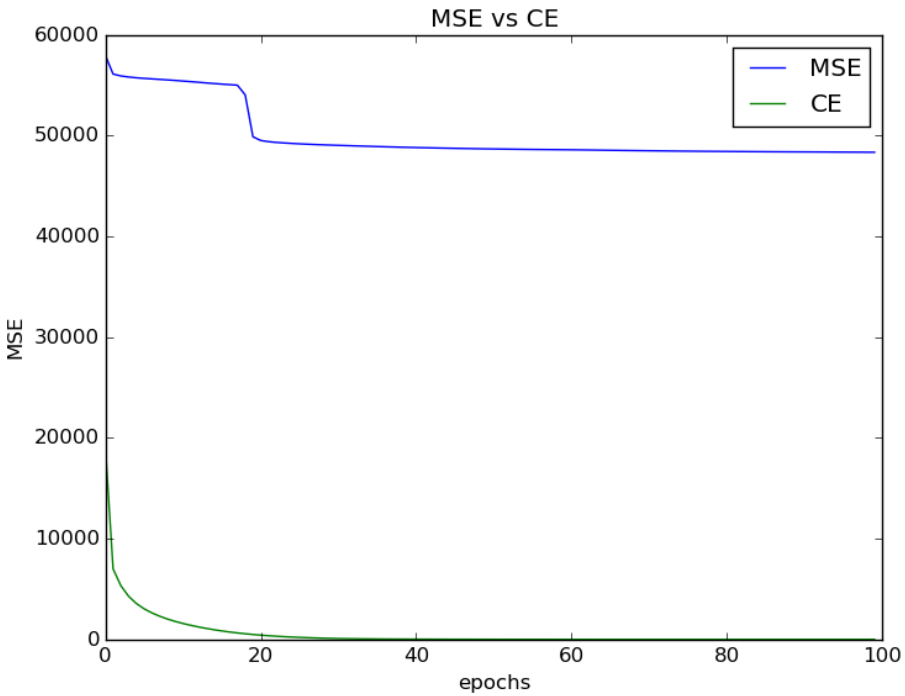
Затем получившееся значение сравнивалось с эталонными значениями и количество совпадений суммировалось для всех образов:

$$S = \sum_{s=1}^{L} I[k_s = e_s]$$

где $I[]$ – индикаторная функция, L – общее количество образов из оцениваемого множества, e_s – эталонное значение, соответствующее s-тому образу (может быть получено по формуле $e_s = \arg\max_j t_j^s$). Далее, произведя необходимые нормировки, получим следующий показатель эффективности метода:

$$\text{Efficiency} = \frac{S}{L} * 100\%$$

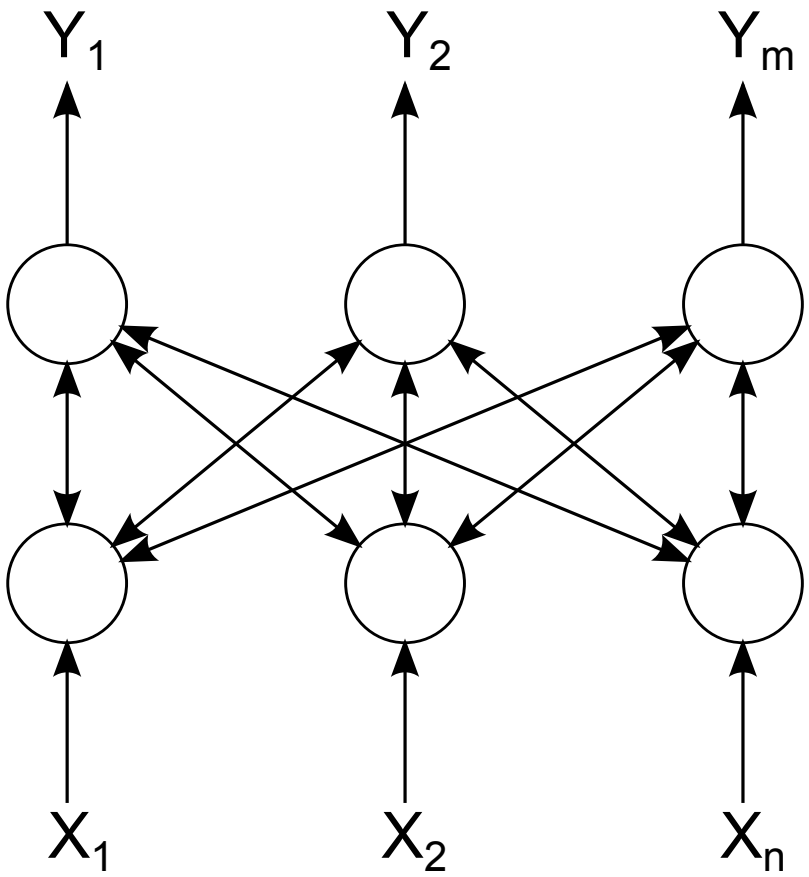
Результат: MSE vs CE



	CE, %	MSE_{bad}, %	MSE_{good}, %
Tr	100	20.615	79.41
Te	98.44	20.77	78.63

Таблица 1: Тестирование

Предобучение: RBM

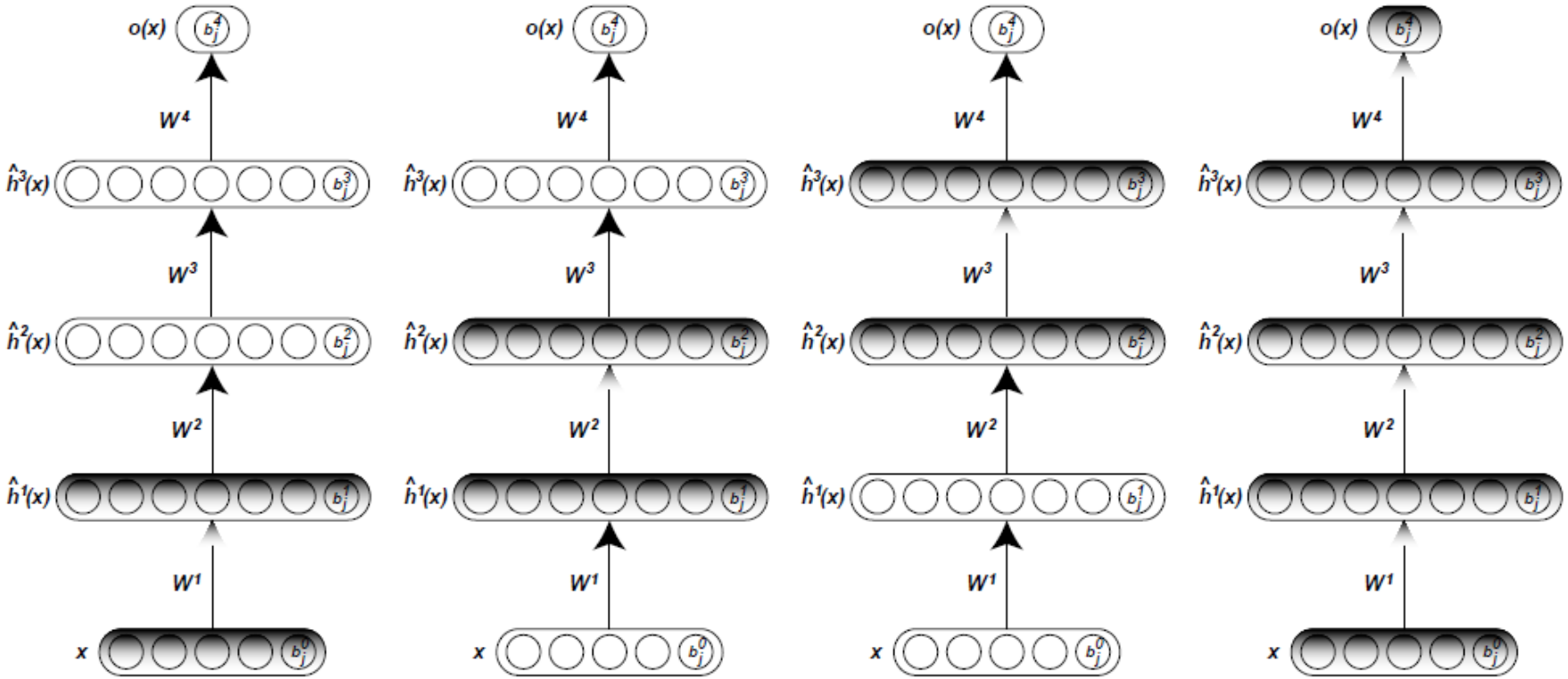


$$w_{ij}(t+1) = w_{ij}(t) + \alpha(x_i(0)y_j(0) - x_i(1)y_j(1))$$
$$T_i(t+1) = T_i(t) + \alpha(x_i(0) - x_i(1))$$
$$T_j(t+1) = T_j(t) + \alpha(y_j(0) - y_j(1)).$$

Вход: $x_i(0)$ – образ из обучающей выборки, α - скорость обучения
Результат: матрица весовых коэффициентов W, вектор порогов видимых элементов b, вектор порогов скрытых нейронов c

foreach *скрытого нейрона* j **do**
  Вычислить $P(y_j(0) = 1|x_i(0))$ (для биномиальных нейронов $\text{sigm}(\sum_i w_{ij}x_i(0) + T_j)$)
  Генерировать $y_j(0) \in \{0, 1\}$ из $P(y_j(0)|x_i(0))$
end
foreach *видимого нейрона* i **do**
  Вычислить $P(x_i(1) = 1|y_j(0))$ (для биномиальных нейронов $\text{sigm}(\sum_j w_{ij}y_j(0) + T_i)$)
  Генерировать $x_i(1) \in \{0, 1\}$ из $P(x_i(1)|y_j(0)$
end
foreach *скрытых нейронов* j **do**
  Вычислить $P(y_j(1) = 1|x_i(1))$ (для биномиальных нейронов $\text{sigm}(\sum_i w_{ij}x_i(1) + T_j)$)
end
$W \leftarrow W + \alpha(x(0)y(0)' - x(1)P(y(1) = 1|x(1))')$
$T_i \leftarrow T_i + \alpha(x(0) - x(1))$
$T_j \leftarrow T_j + \alpha(y(0) - P(y(1) = 1|x(1)))$

Предобучение: визуализация алгоритма



(a) First hidden layer pre-training

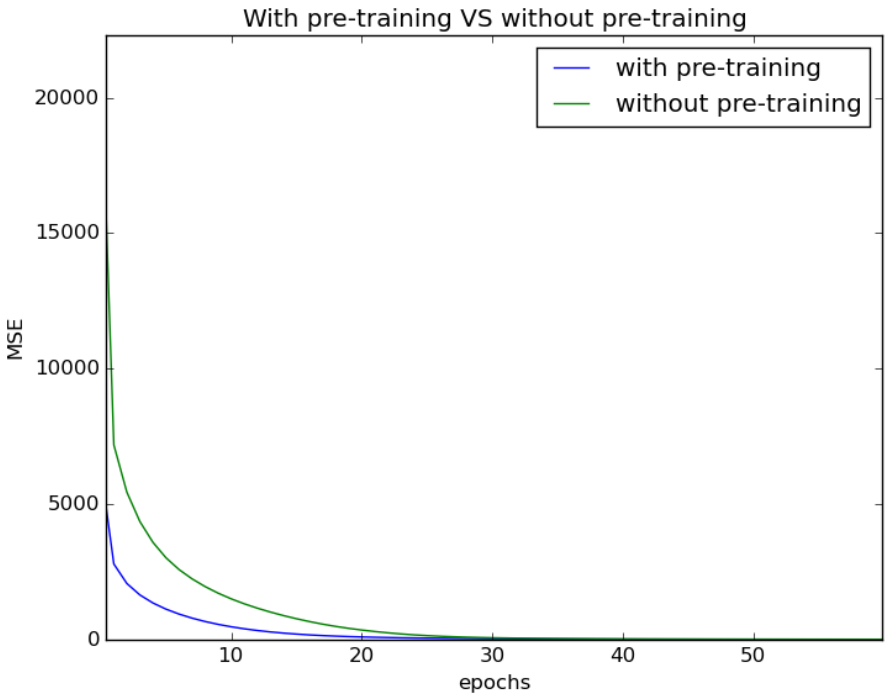
(b) Second hidden layer pre-training

(c) Third hidden layer pre-training

(d) Fine-tuning of whole network

Взято из Hinton G. Greedy layer-wise algorithm (Journal of Machine Learning Research), 2009

Результат: Pretrain vs Non-pretrain (CE)



	Pretrain, %	Non-pretrain, %
Tr	100	100
Te	**98.81**	98.44

Таблица 2: Тестирование

*MSE с предобучением $\approx$ 97%

Решение: параметры предобучения

```
class Pretrain_params:
  def __init__(self, max_epochs, min_error, batch_size, smomentum,
      fmomentum, rates, weight_loss):
    self.max_epochs = max_epochs
    self.batch_size = batch_size
    self.start_momentum = smomentum
    self.final_momentum = fmomentum
    self.rates = rates
    self.weight_loss = weight_loss
    self.min_error = min_error
```

Решение: предобучение глубокой сети

```
def pretrain_deep_network(data, params, architecture, activ_funcs):
    output = data
    i = 0
    inputs = output.shape[1]
    rbm_stack = []
    rbm_train_method = RBM_pretrain(params)
    while i < len(architecture) - 1:
        outputs = architecture[i]
        act_func_vis = activ_funcs[i]
        act_func_hid = activ_funcs[i + 1]
        print 'Pretrain layer ' + str(inputs) + '-' + str(outputs)
        rbm, output = rbm_train_method.train(output, (inputs, outputs),
            (act_func_vis, act_func_hid), params.rates[i])
        rbm_stack.append(rbm)
        inputs = outputs
        i += 1
    outputs = architecture[i]
    act_func_vis = activ_funcs[i]
    act_func_hid = activ_funcs[i + 1]
    last_rbm = network.RBM((inputs, outputs), (act_func_vis,
        act_func_hid))
    rbm_stack.append(last_rbm)
    return rbm_stack
```

Решение: обучение RBM

```
class RBM_pretrain:
  def __init__(self, params):
    self.params = params

  def train(self, input, shape, activ_funcs, rate, rbm=None):
    params = self.params
    batch_size = params.batch_size
    num_batches = input.shape[0] / batch_size
    rbm = network.RBM(shape, activ_funcs)
    #initialize weight updates
    wu_vh = np.zeros(shape)
    bu_v = np.zeros(shape[0])
    bu_h = np.zeros(shape[1])
    #initialize output array
    output = np.zeros((input.shape[0], shape[1]))
    i = 0
    isFinish = False
```

Решение: обучение RBM

```
while not isFinish:
  err = 0
  momentum = params.start_momentum if (i < 5) else params.
      final_momentum
  for batch in range(num_batches):
    v1 = np.array(input[batch*batch_size:(batch+1)*batch_size])
    h1 = rbm.hid_activate(v1)
    if i == params.max_epochs-1:
      output[batch*batch_size:(batch+1)*batch_size] = h1
    if isinstance(activ_funcs[1], act.Logistic):
      h_sampled = h1 > np.random.random(h1.shape)
    else:
      h_sampled = h1
    v2 = rbm.vis_activate(h_sampled)
    h2 = rbm.hid_activate(v2)
    wu_vh = wu_vh * momentum + rate * ((np.dot(v1.T, h1) - np.dot(
        v2.T, h2)) / batch_size)
    bu_v = bu_v * momentum + rate / batch_size * (v1.sum(0) - v2.
        sum(0))
    bu_h = bu_h * momentum + rate / batch_size * (h1.sum(0) - h2.
        sum(0))
```

Решение: обучение RBM

```
    rbm.weights += wu_vh
    rbm.vis_biases += bu_v
    rbm.hid_biases += bu_h
    err += ((v2-v1)**2).sum()
  i += 1
  print str(i) + ' epoch is complete... Error is ' + str(err)
  isFinish = i >= params.max_epochs or err < params.min_error
return [rbm, output]
```

Реализация класса RBM

```
class RBM:
  def __init__(self, shape, act_func):
    self.weights = 0.1 * np.random.randn(shape[0], shape[1])
    self.vis_biases = np.zeros((1, shape[0]))
    self.hid_biases = -1 * np.ones((1, shape[1]))
    self.act_func = act_func

  def hid_activate(self, data):
    weighted_sum = np.dot(data, self.weights) + self.hid_biases
    return self.act_func[1].apply(weighted_sum)

  def vis_activate(self, data):
    weighted_sum = np.dot(data, self.weights.T) + self.vis_biases
    return self.act_func[0].apply(weighted_sum)
```

Оценка качества построенной модели

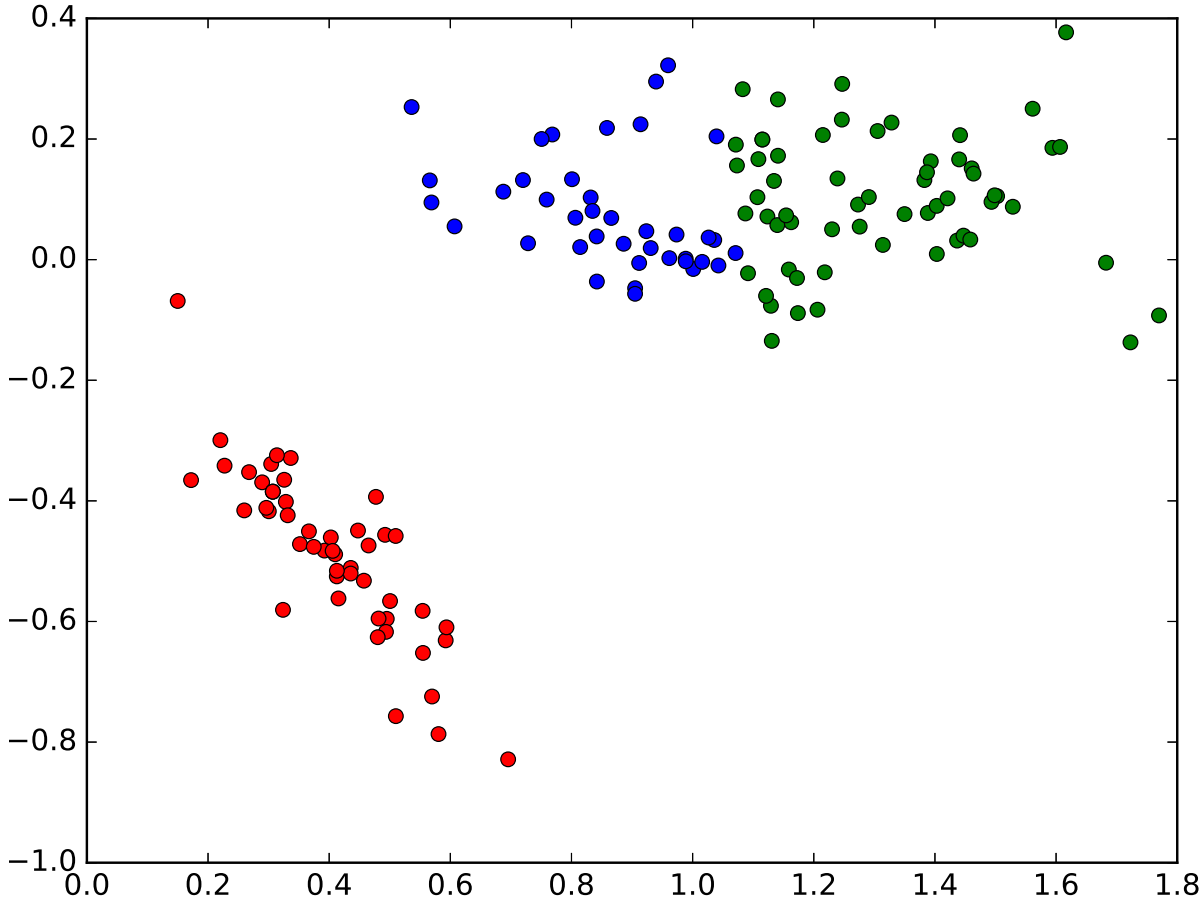
Для оценки качества построенной модели может применяться введенные раннее показатели ROC-анализа, которые могут быть обобщены на случай многих классов, применяя стратегию «один против всех». Для обученного классификатора может быть построена следующая таблица, называемая confusion matrix.

Действительные значения									
973	0	2	0	0	2	3	1	2	2
1	1129	1	0	0	0	2	2	0	1
0	2	1019	1	1	0	1	5	2	0
0	1	1	1000	1	4	1	0	1	4
0	0	1	0	970	0	1	0	2	9
0	0	0	5	0	882	1	0	1	2
1	0	0	0	4	1	948	0	1	1
1	1	5	2	0	1	0	1015	2	3
2	2	3	2	1	2	1	1	960	1
2	0	0	0	5	0	0	4	3	986

Таблица 3: Confusion matrix

Предобучение для выборки IRIS

Применяя классические подходы кластеризации (например, метод k-средних), можно заметить неприемлемое качество распознавания на границах двух линейно неразделимых классов



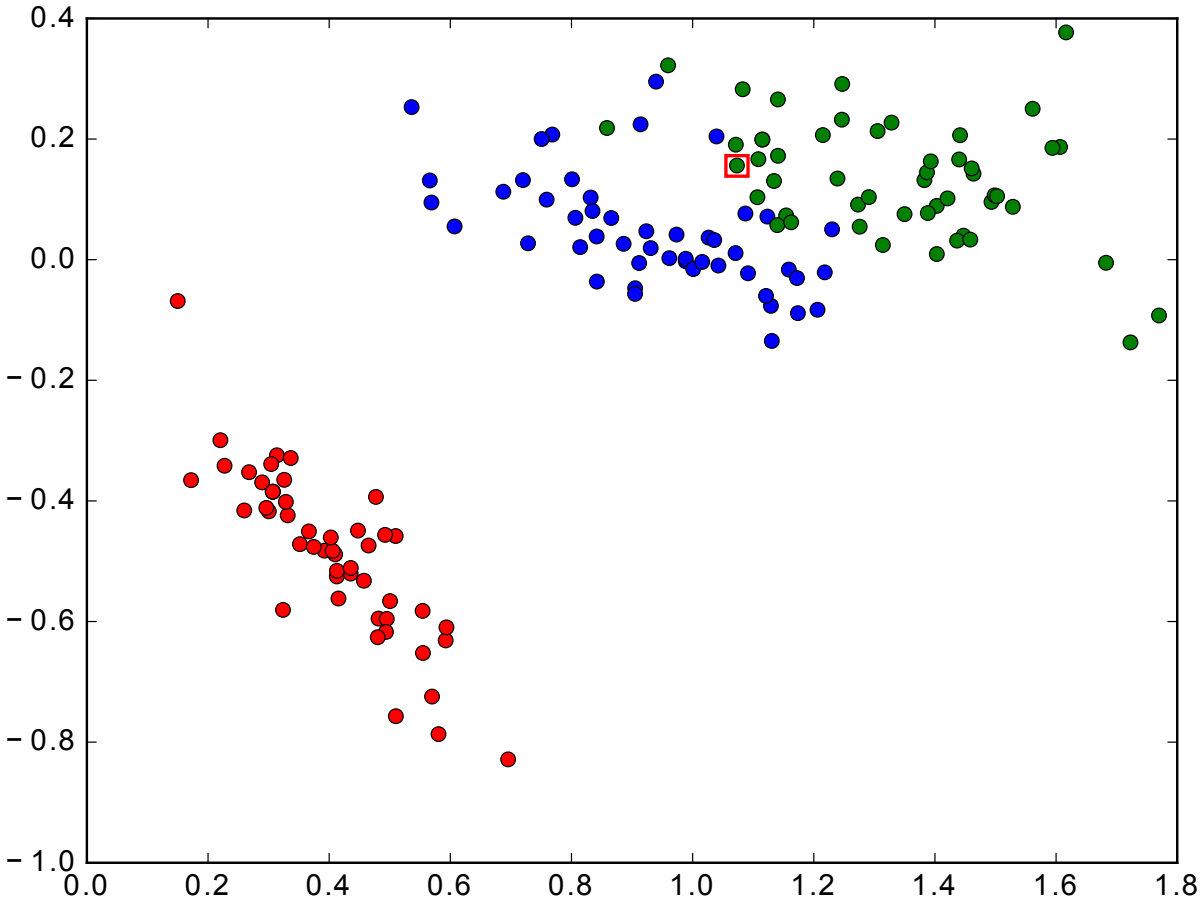
Параметры сети

Для решения этой задачи воспользуемся сетью с архитектурой **4-32-16-8-3** с сигмоидными функциями активации на каждом обрабатывающем слое.
Другие параметры:

- Фаза предобучения: скорость – 0.1, моментный параметр – переменный (от 0.5 до 0.9), размер мини-батча - 5, количество эпох обучения каждого слоя – 50.
- Фаза обучения: скорость – 0.05 (с редукцией, коэффициент 0.99), моментный параметр – 0.9, размер мини-батча – 5, количество эпох обучения – 2000, параметр L2-регуляризации (weight decay) – 0.00001.

Результат обучения

После обучения нейронной сети сетью была достигнута совокупная ошибка распознавания 99,33%. Таким образом, неправильно распознанным остался только один образ из всей выборки (см. рис. 3).



Показательной в данном случае является эволюция среднеквадратичной ошибки после предобучения и без него. Таким образом, предобучение позволяет получить хорошую начальную инициализацию весов и порогов нейронной сети, что делает последующий этап «тонкой настройки» методом обратного распространения ошибки значительно более продуктивным.

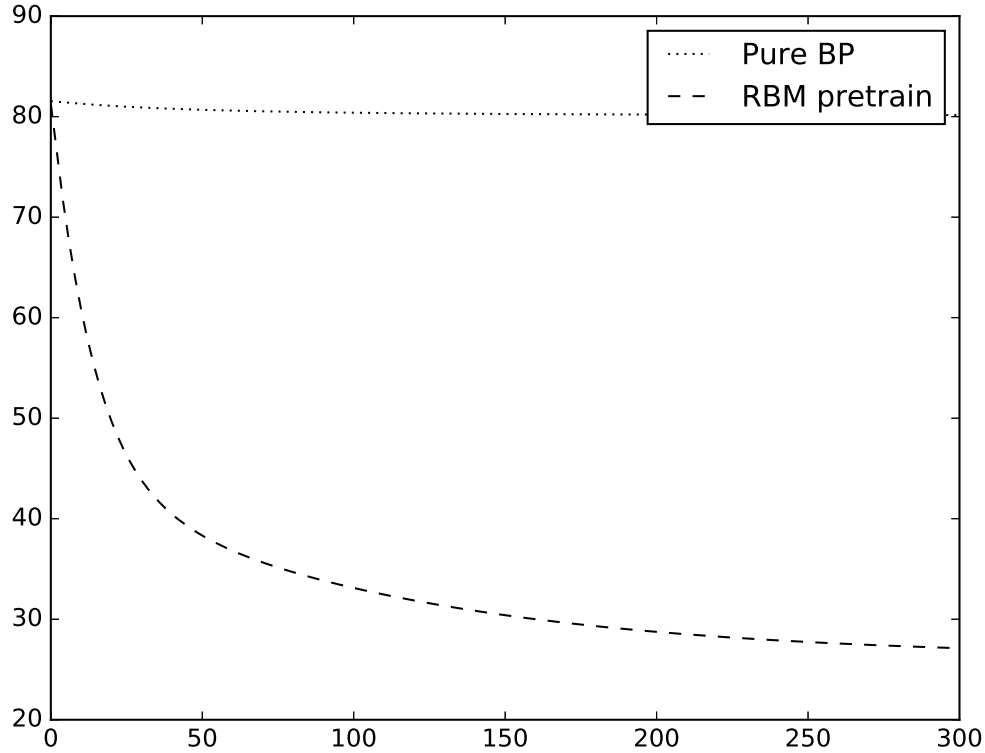


Figure 4: Эволюция ошибок для метода с предобучением и без

Пример решения регрессионной задачи

Оценочной функцией называется отображение $\tilde{f} : X \to \mathbb{R}$.
Проблема обучения регрессии заключается в построении оценочной функции по примерам $(x_i, f(x_i))$, где $f(x)$ – неизвестная функция.
Рассмотрим пример регрессионной задачи Housing Data Set(`https://archive.ics.uci.edu/ml/datasets/Housing`).
Этот датасет состоит из описаний условий жизни различных районов Бостона (уровень преступности, удаленность от крупнейших торговых центров, концентрация оксидов азота и т.д.). Необходимо на основе имеющихся данных построить модель, способную выдавать среднюю цену на жилье в этом районе. Размерность: 506 образов по 13 компонент каждая.

0.00632	18.00	2.310	0	0.5380	6.5750	65.20	4.0900	1	296.0	15.30	396.90	4.98	24.00
0.02731	0.00	7.070	0	0.4690	6.4210	78.90	4.9671	2	242.0	17.80	396.90	9.14	21.60
0.02729	0.00	7.070	0	0.4690	7.1850	61.10	4.9671	2	242.0	17.80	392.83	4.03	34.70
0.03237	0.00	2.180	0	0.4580	6.9980	45.80	6.0622	3	222.0	18.70	394.63	2.94	33.40
0.06905	0.00	2.180	0	0.4580	7.1470	54.20	6.0622	3	222.0	18.70	396.90	5.33	36.20
0.02985	0.00	2.180	0	0.4580	6.4300	58.70	6.0622	3	222.0	18.70	394.12	5.21	28.70
0.08829	12.50	7.870	0	0.5240	6.0120	66.60	5.5605	5	311.0	15.20	395.60	12.43	22.90
0.14455	12.50	7.870	0	0.5240	6.1720	96.10	5.9505	5	311.0	15.20	396.90	19.15	27.10
0.21124	12.50	7.870	0	0.5240	5.6310	100.00	6.0821	5	311.0	15.20	386.63	29.93	16.50
0.17004	12.50	7.870	0	0.5240	6.0040	85.90	6.5921	5	311.0	15.20	386.71	17.10	18.90

Figure 5: Фрагмент датасета

Решение

В качестве модели мы используем конфигурацию нейронной сети с 2 слоями обрабатывающих элементов вида **13-40{Logistic}-1{Linear}**. Задаем уровень минимальной ошибки 1e-5, максимальное количество эпох обучения – 2000, количество образов в минибатче – 10, моментный параметр – 0.9 и скорость 0.01. **Запускаем...**

В чем проблема? Почему сеть не обучается?

```
1987 epoch is complete... error is 39601.4038181
1988 epoch is complete... error is 39601.4038181
1989 epoch is complete... error is 39601.4038181
1990 epoch is complete... error is 39601.4038181
1991 epoch is complete... error is 39601.4038181
1992 epoch is complete... error is 39601.4038181
1993 epoch is complete... error is 39601.4038181
1994 epoch is complete... error is 39601.4038181
1995 epoch is complete... error is 39601.4038181
1996 epoch is complete... error is 39601.4038181
1997 epoch is complete... error is 39601.4038181
1998 epoch is complete... error is 39601.4038181
1999 epoch is complete... error is 39601.4038181
2000 epoch is complete... error is 39601.4038181
```

Figure 6: Эволюция ошибки

Забыли о предобработке?

Линейное преобразование (нормирование):

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

Стандартизация:

$$x_{new} = \frac{x_{old} - x_{mean}}{\sigma}, x_{mean} = \frac{1}{N}\sum_{i=1}^{N} x_i, \sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - x_{mean})^2}$$

Оценка качества построенной модели

Качество построенной модели можно оценить по следующей формуле (Mean Absolute Percentage Error):

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} \frac{|\tilde{y}_i - y_i|}{y_i} \times 100\%$$

где N – количество образов в проверяемой выборке.
Эту формулу удобно применять, когда нет нулевых значений y_i.

Решение: загрузка и подготовка данных

```
def load_data(path):
  f = open(path)
  data = []
  for _str in f:
    components = _str.rstrip('\n').split()
    data.append(components)
    data = np.array(data).astype('float')
    targets = data[:, len(data[0])-1]
    data = data[:, 0:len(data[0]) - 1]
    data = (data - data.min(axis=0)) / (data.max(axis=0) - data.min
        (axis=0))
  return train_test_split(data, targets, test_size=0.2,
      random_state=RANDOM_SEED)
```

Решение: вычисление MAE

```
def MAE(net, data, targets):
    output = net.activate(data)
    _targets = targets.reshape((len(targets), 1))
    mae_error = (np.abs(output - _targets) / _targets).sum()/len(
        _targets) * 100
    return mae_error
```

Решение: основная программа

```
if __name__ == "__main__":
    data = load_data("Datasets/housing.data.txt")
    net = network.Network()
    layer_1 = layer.FullyConnectedLayer(activate_functions.Logistic()
        , 13, 40)
    layer_2 = layer.FullyConnectedLayer(activate_functions.Linear(),
        40, 1)
    net.append_layer(layer_1)
    net.append_layer(layer_2)
    params = backpropagation.Backprop_params(2000, 1e-5, 10, 0.9,
        False, [0.01, 0.01], 0.00001)
    method = backpropagation.Backpropagation(params, net)
    train_data = data[0]
    test_data = data[1]
    train_labels = data[2]
    test_labels = data[3]
    print len(train_data)
    print len(test_data)

    error_curve = method.train(train_data, train_labels)
    print MAE(net, test_data, test_labels)
    print MAE(net, train_data, train_labels)
```

Результаты

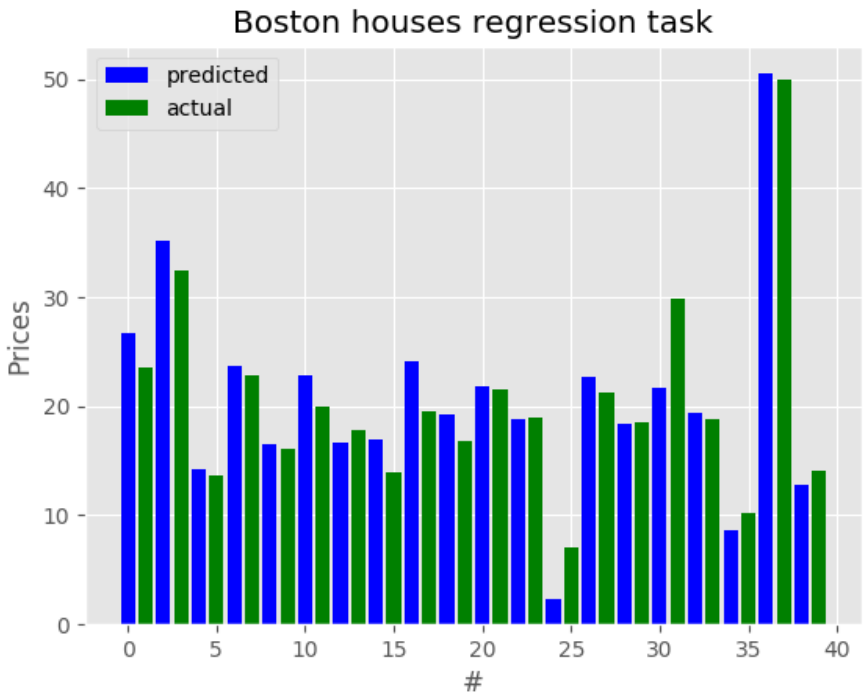


Figure 7: Test, MAPE = 11.93%

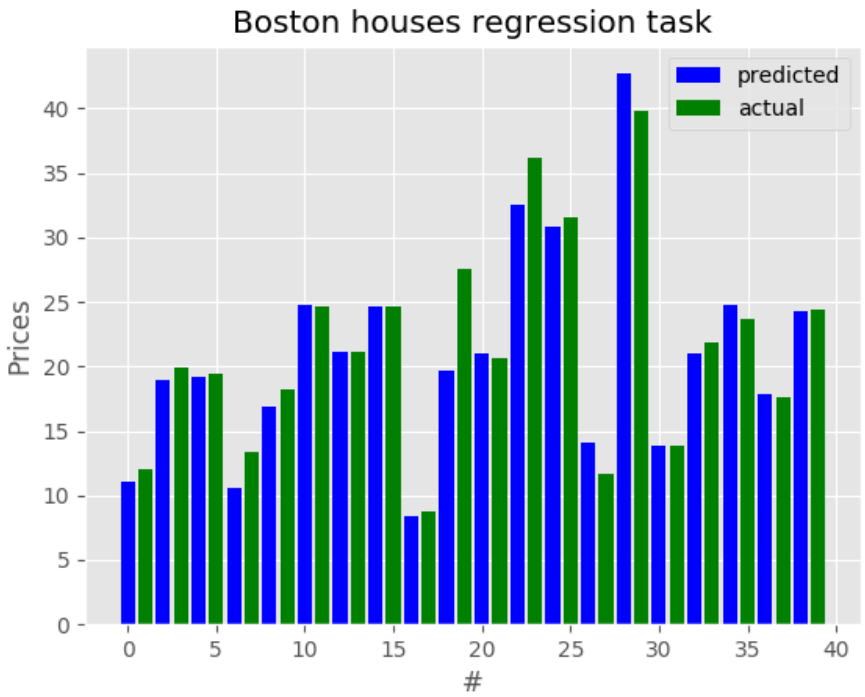


Figure 8: Train, MAPE = 6.38%

Две проблемы обучения ИНС: классификационная задача – bias-variance tradeoff

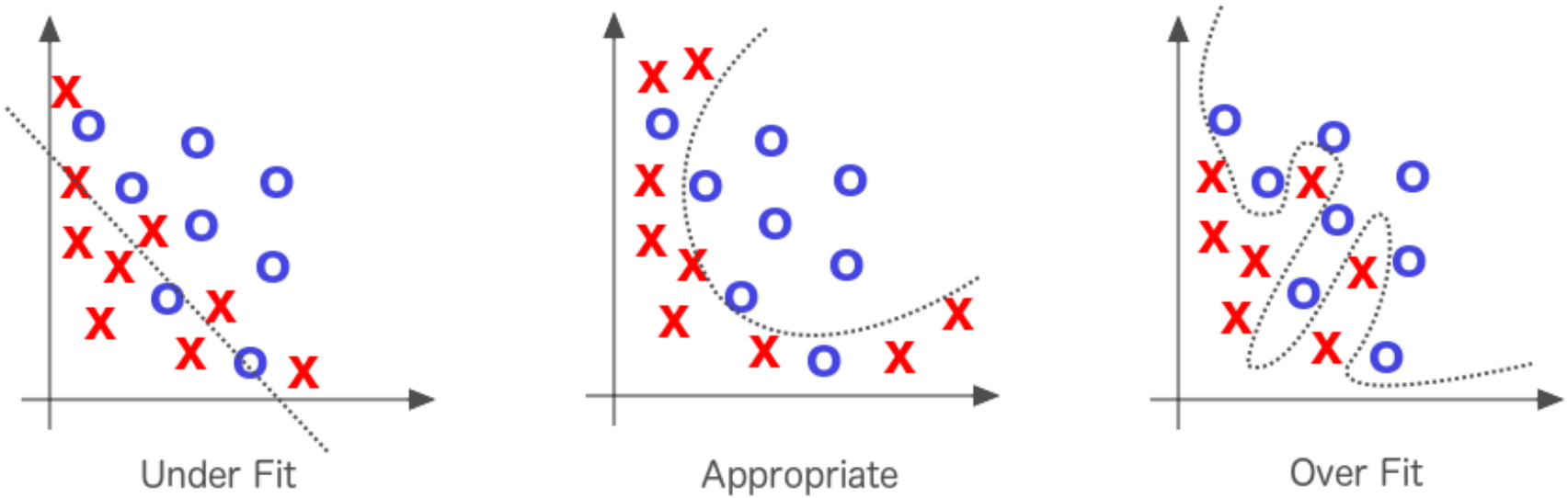


Figure 9: Демонстрация на примере классификационной задачи

https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/ch01.html

Две проблемы обучения ИНС: регрессионная задача – bias-variance tradeoff

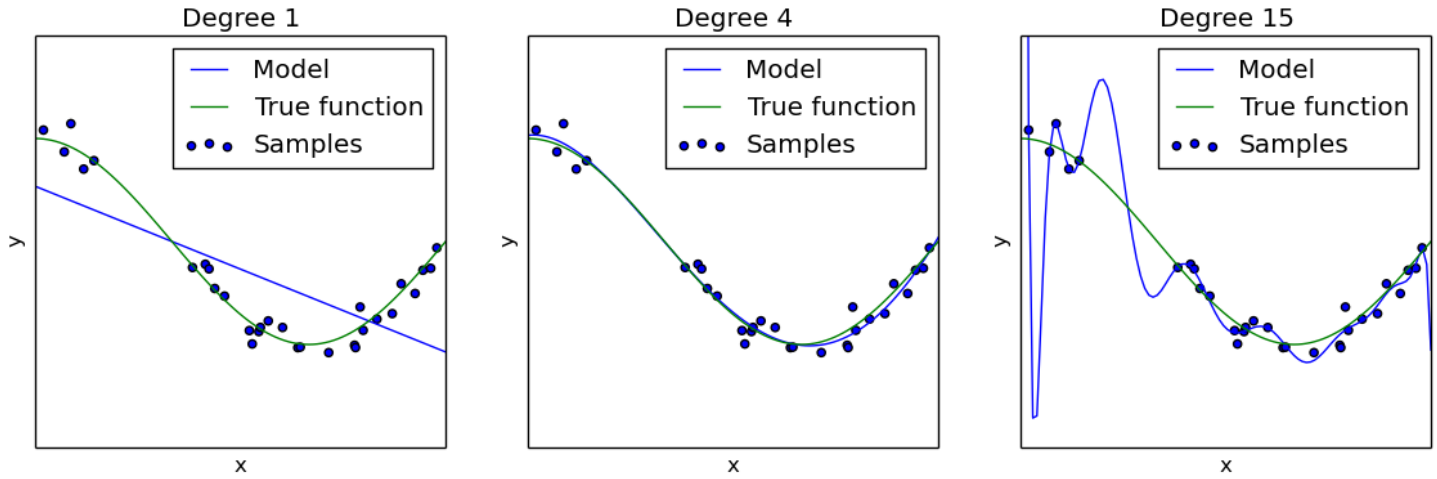


Figure 10: Демонстрация на примере регрессионной задачи

`http://scikit-learn.org/0.15/auto_examples/plot_underfitting_overfitting.html`

Решением проблемы переобучения является проведение регуляризации, которая снижает дисперсию модели (variance) за счет увеличения смещения (bias). Также может рассматриваться уменьшение количества нейронов в скрытом слое и проведение кросс-валидации.

Пути повышения эффективности обучения

- Увеличение обобщающей способности нейронной сети
- Увеличение скорости выполнения обучения

Данные характеристики тесно взаимосвязаны, зачастую бывает сложно сказать, каким образом повлияет тот или иной подход. Многое зависит от задачи.

Способы улучшения обобщающей способности

- регуляризация (dropout, L1 и L2 (weight-decay)-регуляризации, искусственное расширение обучающей выборки - elastic distortions)

$$E = \frac{1}{N}\sum_{k=1}^{N}(\tilde{y}_k - y_k)^2 + \lambda \sum |w_{ij}|$$

$$E = \frac{1}{N}\sum_{k=1}^{N}(\tilde{y}_k - y_k)^2 + \lambda \sum (w_{ij})^2$$

  Использование L1-регуляризации может иметь хороший побочный эффект в виде обнуления части весовых коэффициентов. Это одна из форм feature selection. L2-регуляризация ограничивает весовые коэффициенты, но не обнуляет их. Однако, L2-регуляризация может использоваться с любыми обучающими алгоритмами, а L1-регуляризация – нет (алгоритмы, вычисляющие градиент).
- правильный выбор целевой минимизируемой функции mean-squared error (MSE), cross-entropy error (CE).

Способы ускорения обучения

- Программные (применение моментного параметра, обучение мини-батчами)
- Аппаратные (ускорение обучения за счет использования массивно-параллельных вычислений (вычисления на видеокартах nVidia – технология CUDA)).

Применение массивно-параллельных вычислений на видеокарте с поддержкой технологии CUDA позволили ускорить обучение нейронной сети в 4,5 раза.
Факторы, влияющие на стабильность обучения: степень однородности выборки, подходящие величины шага и моментного параметра.

Способы улучшения качества обучения глубоких нейронных сетей

- **Применение преобучения** (используя обучение ограниченных машин Больцмана, формируемых из слоев исходной сети или автоэнкодерный подход).
- **Использование специальных активационных функций** (например, ReLU, Maxout, noisy activation function), позволяющих устранить проблему исчезающего градиента, тем самым обеспечивая корректную настройку весовых коэффициентов для первых слоев нейронной сети.