

Методы машинного обучения

Шорохов С.Г.

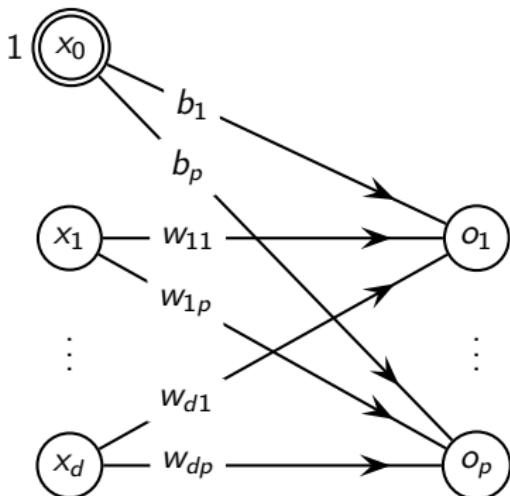
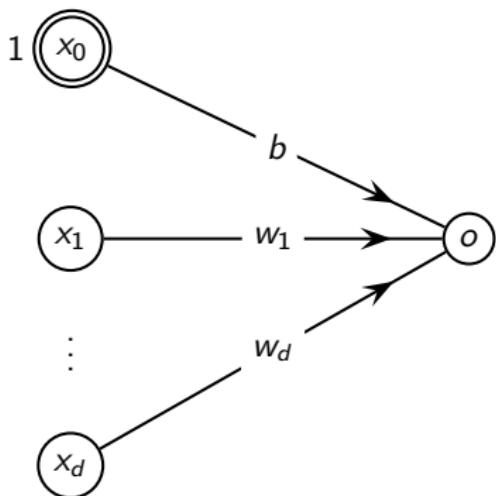
кафедра математического моделирования и искусственного интеллекта

Лекция 4. Нейронные сети MLP



Регрессия с ANN

Искусственные нейронные сети способны аппроксимировать произвольно сложные скалярные и векторные функции для решения задачи регрессии (множественной и многомерной).





Рассмотрим множественную регрессию длины чашелистика (sepal length) и длины лепестка (petal length) для зависимого признака ширины лепестка (petal width) для набора данных «Ирисы» с оптимальным решением:

$$\hat{y} = -0.014 - 0.082 x_1 + 0.45 x_2$$

Квадратичная ошибка для этого оптимального решения составляет 6.179 на обучающих данных.

Нейронная сеть с линейной активацией и минимизацией функции SSE посредством градиентного спуска приводит к результату

$$o = 0.0096 - 0.087 x_1 + 0.452 x_2$$

с показателем SSE, равным 6.18, что очень близко к оптимальному решению.

Пример многомерной линейной регрессии



Используем длину чашелистика и ширину чашелистика в качестве независимых признаков, а длину и ширину лепестка в качестве многомерного отклика (зависимых признаков).

Следовательно, каждый входной вектор \mathbf{x}_i является двумерным, а отклик \mathbf{y}_i также является двумерным вектором. Минимизация функции потерь SSE с помощью градиентного спуска дает следующие веса и смещения нейронной сети:

$$\begin{pmatrix} b_1 & b_2 \\ w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \begin{pmatrix} -1.83 & -1.47 \\ 1.72 & 0.72 \\ -1.46 & -0.50 \end{pmatrix}, \begin{pmatrix} o_1 \\ o_2 \end{pmatrix} = \begin{pmatrix} -1.83 + 1.72x_1 - 1.46x_2 \\ -1.47 + 0.72x_1 - 0.50x_2 \end{pmatrix}$$

Ошибка SSE на тренировочном наборе составляет 84.9. Оптимальная многомерная регрессия дает SSE 84.16:

$$\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \end{pmatrix} = \begin{pmatrix} -2.56 + 1.78x_1 - 1.34x_2 \\ -1.59 + 0.73x_1 - 0.48x_2 \end{pmatrix}$$



Классификация при помощи нейронных сетей

Искусственные нейронные сети также могут быть обучены, чтобы классифицировать входные данные. Простое изменение в нейронной сети позволяет решать задачу бинарной классификации при помощи логистической регрессии. Все, что нам нужно сделать, это использовать сигмоидную функцию активации на выходном нейроне o и использовать бинарную кросс-энтропию вместо квадратичной ошибки:

$$\mathcal{E}_{\mathbf{x}} = - (y \ln o + (1 - y) \ln (1 - o))$$

Выход нейронной сети o для входных данных \mathbf{x} равен

$$\begin{aligned} o &= f(\text{net}_o) = \text{sigmoid}(b + \mathbf{w}^T \mathbf{x}) = \\ &= \frac{1}{1 + \exp(-(b + \mathbf{w}^T \mathbf{x}))} = \pi(\mathbf{x}) \end{aligned}$$



ANN для классификации

Решая задачу бинарной классификации, используем нейронную сеть с одним выходом с логистической активацией на выходном нейроне и бинарной кросс-энтропией в качестве функции ошибки к главным компонентам набора данных Ирисы.

Выходные данные представляют собой бинарный отклик, указывающий на тип Iris-virginica ($Y = 1$) или один из других типов ирисов ($Y = 0$). Как и ожидалось, в результате обучения нейронной сети получаем тот же набор весов и смещений, что и для модели логистической регрессии, а именно:

$$o = -6.79 - 5.07x_1 - 3.29x_2$$

Решая задачу бинарной классификации, используем функцию активации softmax и кросс-энтропийную функцию ошибки для главных компонентов набора данных Ирисы с тремя классами: Iris-setosa ($Y = 1$), Iris-versicolor ($Y = 2$) и Iris-virginica ($Y = 3$). Зафиксируем веса и смещения для выходного нейрона o_3 равными нулю, тогда:

$$\begin{aligned} o_1 &= -3.49 + 3.61x_1 + 2.65x_2 \\ o_2 &= -6.95 - 5.18x_1 - 3.40x_2 \\ o_3 &= 0 + 0x_1 + 0x_2 \end{aligned}$$



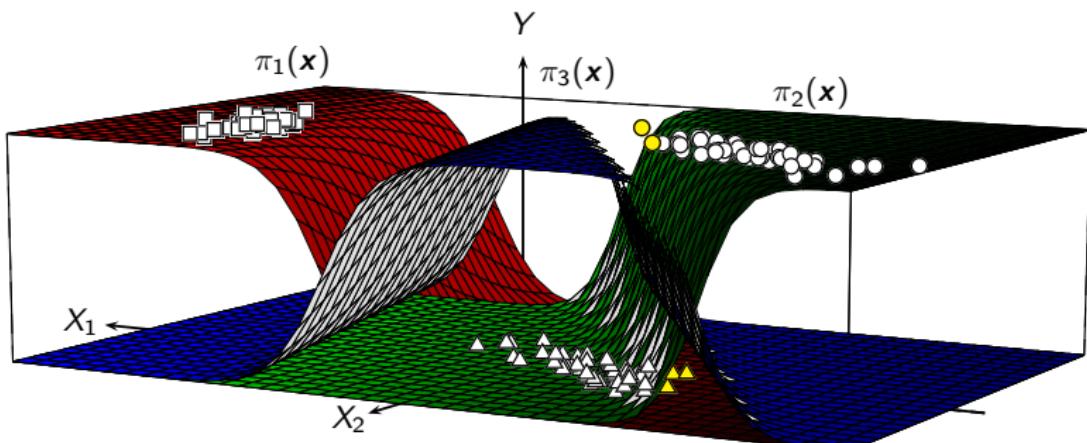
Если не ограничивать веса и смещение для нейрона o_3 , то получим следующую модель:

$$o_1 = -0.89 + 4.54x_1 + 1.96x_2$$

$$o_2 = -3.38 - 5.11x_1 - 2.88x_2$$

$$o_3 = 4.24 + 0.52x_1 + 0.92x_2$$

Неправильно классифицированные точки показаны желтым цветом. Точки в классе c_1 и c_2 показаны смещенными относительно базового класса c_3 только для иллюстрации.





Многослойный персепtron

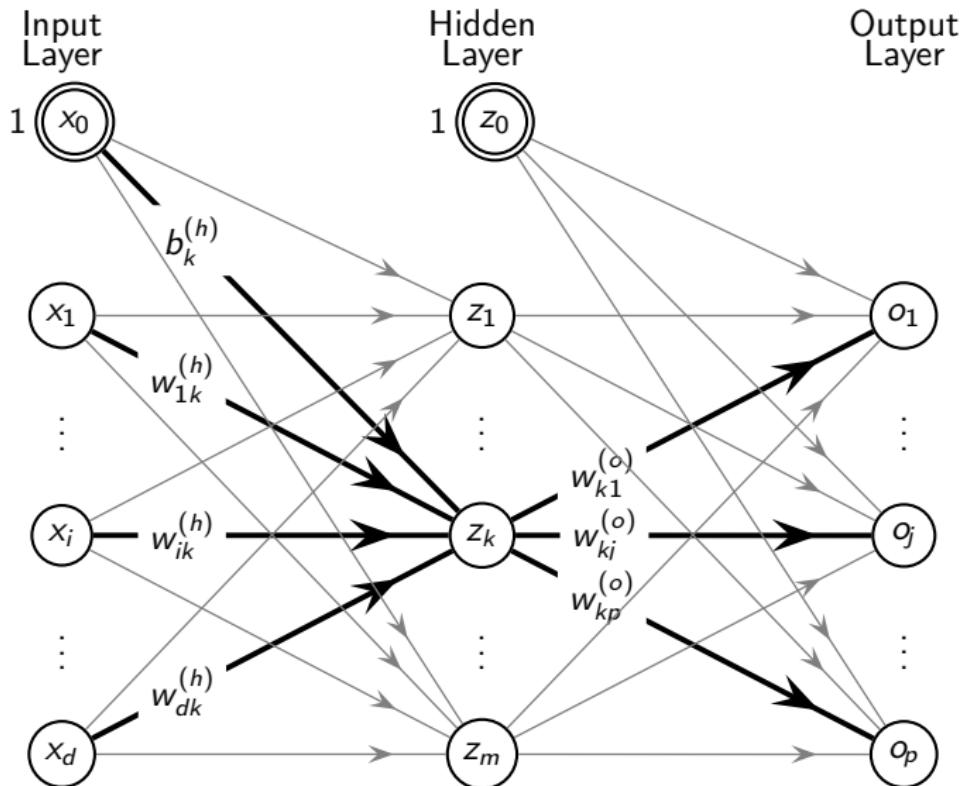
В нейронных сетях **прямого распространения** связи (feed-forward network) данные передаются в прямом направлении от одного слоя к следующему слою.

Многослойный персепtron (MLP) — это класс искусственных нейронных сетей прямого распространения, состоящих как минимум из трех слоёв: входного, скрытого и выходного. Входные данные нейронной сети составляют входной слой (input layer), а конечные выходные данные MLP составляют выходной слой (output layer). Любой промежуточный слой называется скрытым слоем (hidden layer), и MLP может иметь один или несколько скрытых слоев. Нейроны в скрытых слоях используют нелинейную функцию активации.

Сети со многими скрытыми слоями называются **глубокими** нейронными сетями (deep neural networks, DNN).

При обучении MLP используется алгоритм обратного распространения ошибки.

MLP с одним скрытым слоем





Выход нейронов MLP с одним скрытым слоем

Пусть даны значения на входных нейронах x_i , тогда выход для каждого скрытого нейрона z_k равен:

$$z_k = f \left(\text{net}_k^{(h)} \right) = f \left(b_k^{(h)} + \sum_{i=1}^d w_{ik}^{(h)} x_i \right),$$

где f — некоторая функция активации, а $w_{ik}^{(h)}$ обозначает вес между входным нейроном x_i и скрытым нейроном z_k , $b_k^{(h)}$ — смещение. Далее, учитывая значения выхода скрытых нейронов, получим, что значение для каждого выходного нейрона o_j равно:

$$o_j = f \left(\text{net}_j^{(o)} \right) = f \left(b_j^{(o)} + \sum_{i=1}^m w_{ij}^{(o)} z_i \right),$$

где $w_{ij}^{(o)}$ обозначает вес между скрытым нейроном z_i и выходным нейроном o_j , $b_j^{(o)}$ — смещение.



Выход MLP с одним скрытым слоем

Тогда выходные векторы слоев $\mathbf{z} = (z_1, z_2, \dots, z_m)^T$ и $\mathbf{o} = (o_1, o_2, \dots, o_p)^T$ MLP можно вычислить по следующим матричным формулам:

$$\text{net}_h = \mathbf{b}_h + \mathbf{W}_h^T \mathbf{x}, \quad \mathbf{W}_h = \begin{pmatrix} w_{ij}^{(h)} \end{pmatrix}, \quad \mathbf{b}_h = \begin{pmatrix} b_j^{(h)} \end{pmatrix},$$

$$\mathbf{z} = f(\text{net}_h) = f(\mathbf{b}_h + \mathbf{W}_h^T \mathbf{x}),$$

$$\text{net}_o = \mathbf{b}_o + \mathbf{W}_o^T \mathbf{z}, \quad \mathbf{W}_o = \begin{pmatrix} w_{ij}^{(o)} \end{pmatrix}, \quad \mathbf{b}_o = \begin{pmatrix} b_j^{(o)} \end{pmatrix},$$

$$\mathbf{o} = f(\text{net}_o) = f(\mathbf{b}_o + \mathbf{W}_o^T \mathbf{z})$$

Подводя итог, для данного входа $\mathbf{x} \in \mathbf{D}$ с желаемым откликом \mathbf{y} MLP с одним скрытым слоем вычисляет прогнозируемый выходной вектор \mathbf{o} посредством процесса прямого прохода по нейронной сети следующим образом:

$$\mathbf{o} = f(\mathbf{b}_o + \mathbf{W}_o^T \mathbf{z}) = f(\mathbf{b}_o + \mathbf{W}_o^T f(\mathbf{b}_h + \mathbf{W}_h^T \mathbf{x}))$$



Обратное распространение ошибки в MLP

Обратное распространение (backpropagation) ошибки – это алгоритм, используемый для обучения весов последовательных слоев MLP. Название происходит от способа, которым градиент ошибки распространяется обратно от выходного слоя к входному через скрытые слои.

Для заданной входной пары (\mathbf{x}, \mathbf{y}) в обучающих данных MLP сначала вычисляет выходной вектор \mathbf{o} с помощью прямого прохода по MLP. Затем MLP вычисляет ошибку в прогнозируемом выходе по сравнению с истинным откликом \mathbf{y} , используя, например, функцию квадратичной ошибки

$$\mathcal{E}_{\mathbf{x}} = \frac{1}{2} \|\mathbf{y} - \mathbf{o}\|^2 = \frac{1}{2} \sum_{j=1}^p (y_j - o_j)^2$$

Основная идея состоит в том, чтобы проверить, в какой степени выходной нейрон, скажем, o_j , отклоняется от соответствующей целевой реакции y_j , и изменить веса w_{ij} между каждым скрытым нейроном z_i и o_j как некоторую функцию ошибки.



Обновление весов и смещений MLP

Обновление весов выполняется с помощью метода градиентного спуска, чтобы минимизировать ошибку.

Пусть $\nabla_{w_{ij}}$ – это градиент функции ошибки относительно весов w_{ij} (или градиент весов). Если предыдущая оценка весов равна w_{ij} , то новые веса вычисляются путем совершения малого шага η в направлении, противоположном градиенту весов в точке w_{ij} :

$$w_{ij} = w_{ij} - \eta \nabla_{w_{ij}}$$

Аналогичным образом смещения b_j также обновляются с помощью градиентного спуска:

$$b_j = b_j - \eta \nabla_{b_j},$$

где ∇_{b_j} – градиент функции ошибок по отношению к смещению b_j (градиент смещения в точке b_j).



Параметры между скрытым и выходным слоями

Рассмотрим обновление параметров между скрытым и выходным слоями.

Пусть $w_{ij}^{(o)}$ – вес между скрытым нейроном z_i и выходным нейроном o_j и $b_j^{(o)}$ – смещение между нейроном z_0 и o_j . Вычислим градиент весов в точке $w_{ij}^{(o)}$ и градиент смещения в точке $b_j^{(o)}$ следующим образом:

$$\nabla_{w_{ij}^{(o)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial w_{ij}^{(o)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^{(o)}} \frac{\partial \text{net}_j^{(o)}}{\partial w_{ij}^{(o)}} = \delta_j^{(o)} z_i,$$

$$\nabla_{b_j^{(o)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial b_j^{(o)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^{(o)}} \frac{\partial \text{net}_j^{(o)}}{\partial b_j^{(o)}} = \delta_j^{(o)},$$

где символ $\delta_j^{(o)}$ используется для обозначения частной производной ошибки относительно чистого входного сигнала в o_j (чистого градиента в o_j). Тогда чистый градиент равен

$$\delta_j^{(o)} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^{(o)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial f(\text{net}_j^{(o)})} \frac{\partial f(\text{net}_j^{(o)})}{\partial \text{net}_j^{(o)}}$$



Градиент между скрытым и выходным слоями

Так как $f(\text{net}_j^{(o)}) = o_j$, для квадратичной функции потерь получим:

$$\frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial f(\text{net}_j^{(o)})} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial o_j} = \frac{\partial}{\partial o_j} \left\{ \frac{1}{2} \sum_{j=1}^p (y_j - o_j)^2 \right\} = o_j - y_j,$$

где учтено, что все o_k для $k \neq j$ не зависят от o_j . Если в качестве функции активации используется сигмоида, то имеем

$$\frac{\partial f(\text{net}_j^{(o)})}{\partial \text{net}_j^{(o)}} = o_j (1 - o_j)$$

Итак, для квадратичной функции потерь и функции активации сигмоида получаем чистый градиент в виде

$$\delta_j^{(o)} = (o_j - y_j) o_j (1 - o_j)$$



Параметры между входным и скрытым слоями

Рассмотрим обновление параметров между входным и скрытым слоями.

Пусть $w_{ij}^{(h)}$ – вес между входным нейроном x_i и скрытым нейроном z_j и $b_j^{(h)}$ – смещение между нейроном x_0 и z_j , тогда:

$$\nabla_{w_{ij}^{(h)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial w_{ij}^{(h)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^{(h)}} \frac{\partial \text{net}_j^{(h)}}{\partial w_{ij}^{(h)}} = \delta_j^{(h)} x_i,$$

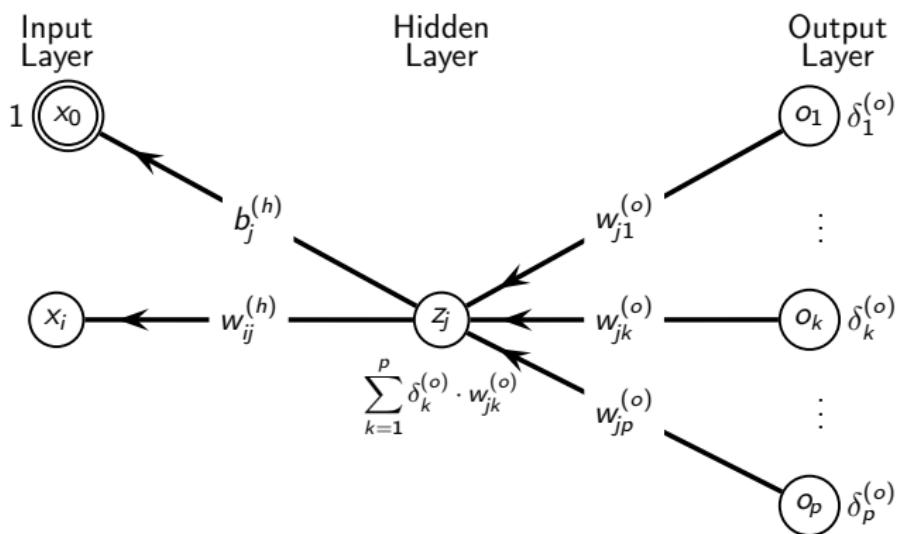
$$\nabla_{b_j^{(h)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial b_j^{(h)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^{(h)}} \frac{\partial \text{net}_j^{(h)}}{\partial b_j^{(h)}} = \delta_j^{(h)}$$

Величины $\delta_j^{(h)}$ в z_j должны учитывать градиенты ошибок, которые возвращаются от всех выходных нейронов к z_j :

$$\begin{aligned}\delta_j^{(h)} &= \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^{(h)}} = \sum_{k=1}^p \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_k^{(o)}} \frac{\partial \text{net}_k^{(o)}}{\partial z_j} \frac{\partial z_j}{\partial \text{net}_j^{(h)}} = \frac{\partial z_j}{\partial \text{net}_j^{(h)}} \sum_{k=1}^p \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_k^{(o)}} \frac{\partial \text{net}_k^{(o)}}{\partial z_j} = \\ &= \left[\frac{\partial z_j}{\partial \text{net}_j^{(h)}} = z_j (1 - z_j) \right] = z_j (1 - z_j) \sum_{k=1}^p \delta_k^{(o)} w_{jk}^{(o)}\end{aligned}$$

Обратное распространение градиентов

Чтобы найти чистый градиент в z_j , необходимо учитывать чистые градиенты на каждом из выходных нейронов $\delta_k^{(o)}$, взвешенные по силе связи $w_{jk}^{(o)}$ между z_j и o_k . То есть вычисляется взвешенная сумма градиентов $\sum_{k=1}^p \delta_k^{(o)} w_{jk}^{(o)}$, которая используется для вычисления чистого градиента $\delta_j^{(h)}$ в скрытом нейроне z_j .





Обучение MLP: алгоритм SGD

Обучение MLP требует множественных итераций по входным точкам. Для каждого входа \mathbf{x}_i MLP вычисляет выходной вектор \mathbf{o}_i при помощи прямого прохода (feed forward). На этапе обратного распространения (backpropagation) вычисляется вектор чистых градиентов ошибки δ_o по отношению к параметрам (весам и смещениям) сети на выходных нейронах, за которым следует вычисление вектора чистых градиентов δ_h для нейронов скрытого слоя. На этапе стохастического градиентного спуска вычисляются градиенты ошибок относительно весов и смещений, которые используются для обновления весовых матриц и векторов смещения.

```
MLP-Training ( $D, m, \eta, \text{maxiter}$ ):  
    // Initialize bias vectors  
    1  $\mathbf{b}_h \leftarrow$  random  $m$ -dimensional vector with small values  
    2  $\mathbf{b}_o \leftarrow$  random  $p$ -dimensional vector with small values  
        // Initialize weight matrices  
    3  $\mathbf{W}_h \leftarrow$  random  $d \times m$  matrix with small values  
    4  $\mathbf{W}_o \leftarrow$  random  $m \times p$  matrix with small values  
    5  $t \leftarrow 0$  // iteration counter
```



Обучение MLP: алгоритм SGD

```
6 repeat
7   foreach  $(x_i, y_i) \in D$  in random order do
8     // Feed-forward phase
9      $z_i \leftarrow f(\mathbf{b}_h + \mathbf{W}_h^T \mathbf{x}_i)$ 
10     $\mathbf{o}_i \leftarrow f(\mathbf{b}_o + \mathbf{W}_o^T z_i)$ 
11    // Backpropagation phase: net gradients
12     $\delta_o \leftarrow \mathbf{o}_i \odot (1 - \mathbf{o}_i) \odot (\mathbf{o}_i - \mathbf{y}_i)$ 
13     $\delta_h \leftarrow z_i \odot (1 - z_i) \odot (\mathbf{W}_o \cdot \delta_o)$ 
14    // Gradient descent for bias vectors
15     $\nabla_{\mathbf{b}_o} \leftarrow \delta_o; \quad \mathbf{b}_o \leftarrow \mathbf{b}_o - \eta \cdot \nabla_{\mathbf{b}_o}$ 
16     $\nabla_{\mathbf{b}_h} \leftarrow \delta_h; \quad \mathbf{b}_h \leftarrow \mathbf{b}_h - \eta \cdot \nabla_{\mathbf{b}_h}$ 
17    // Gradient descent for weight matrices
18     $\nabla_{\mathbf{W}_o} \leftarrow z_i \cdot \delta_o^T; \quad \mathbf{W}_o \leftarrow \mathbf{W}_o - \eta \cdot \nabla_{\mathbf{W}_o}$ 
19     $\nabla_{\mathbf{W}_h} \leftarrow \mathbf{x}_i \cdot \delta_h^T; \quad \mathbf{W}_h \leftarrow \mathbf{W}_h - \eta \cdot \nabla_{\mathbf{W}_h}$ 
20
21    $t \leftarrow t + 1$ 
22 until  $t \geq \text{maxiter}$ 
```



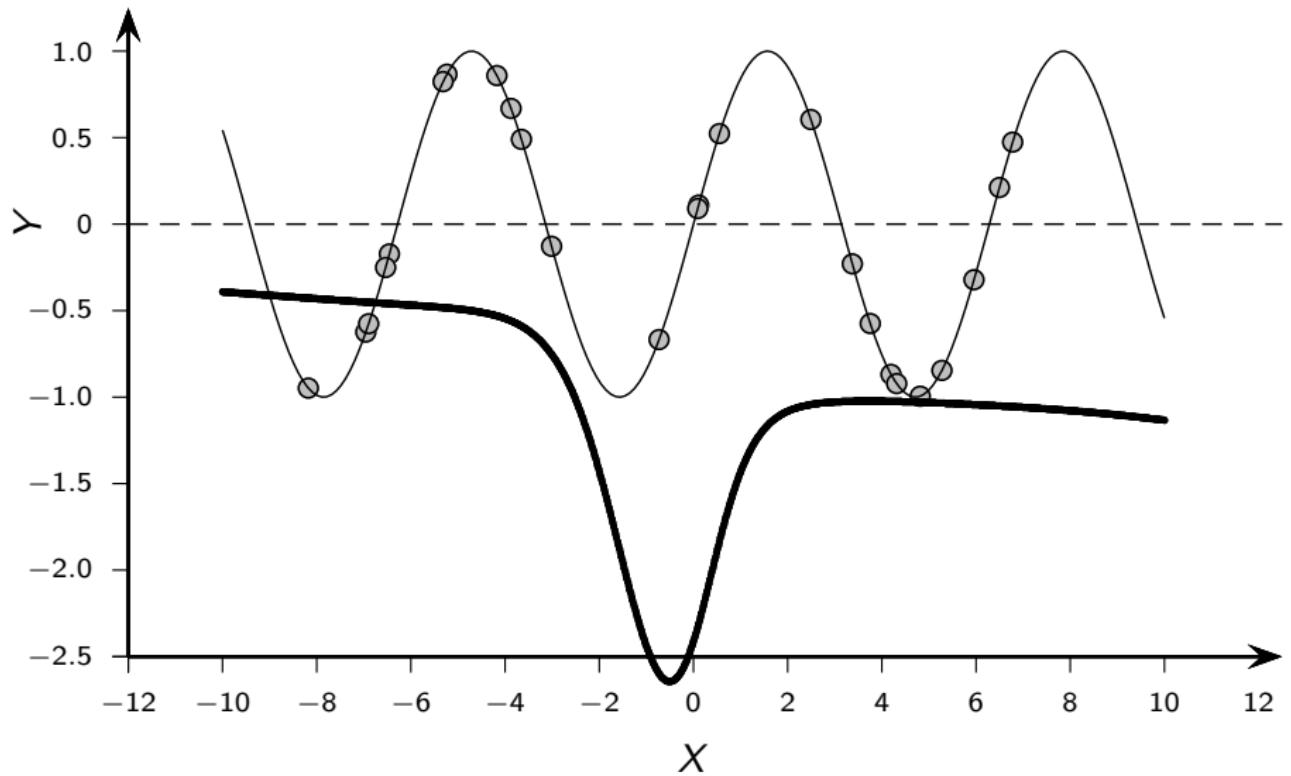
MLP с одним скрытым слоем для синусоиды

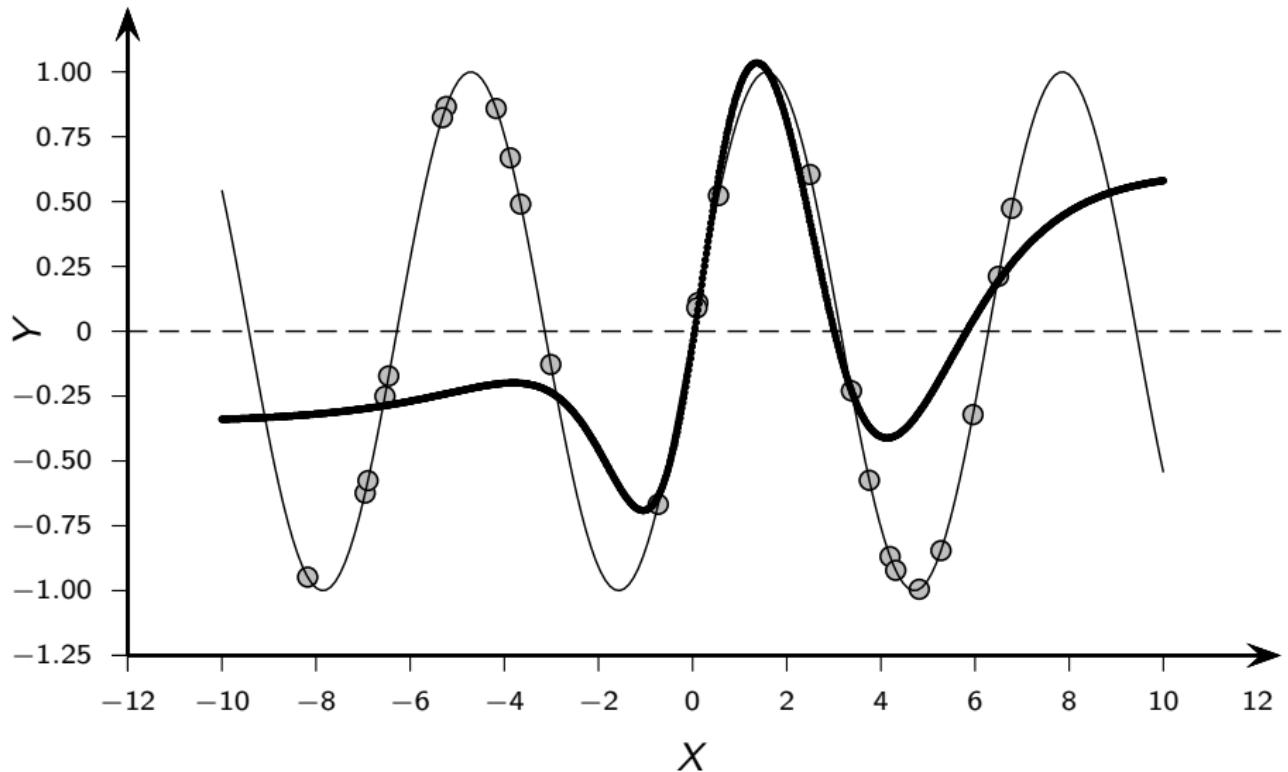
Рассмотрим сеть MLP с одним скрытым слоем, использующую нелинейную функцию активации, чтобы распознавать нелинейную зависимость, задаваемую синусоидой.

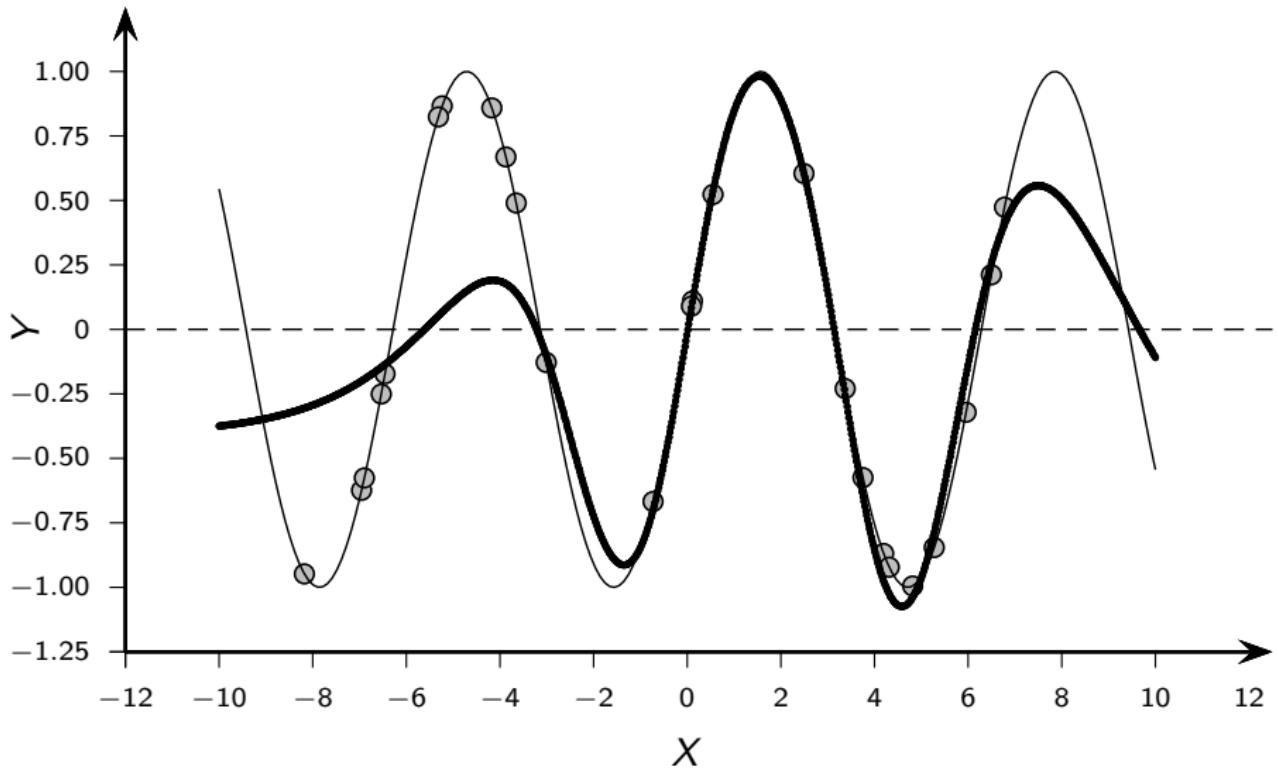
Обучающие данные содержат $n = 25$ точек x_i , выбранных случайным образом в диапазоне $[-10, 10]$, где $y_i = \sin(x_i)$.

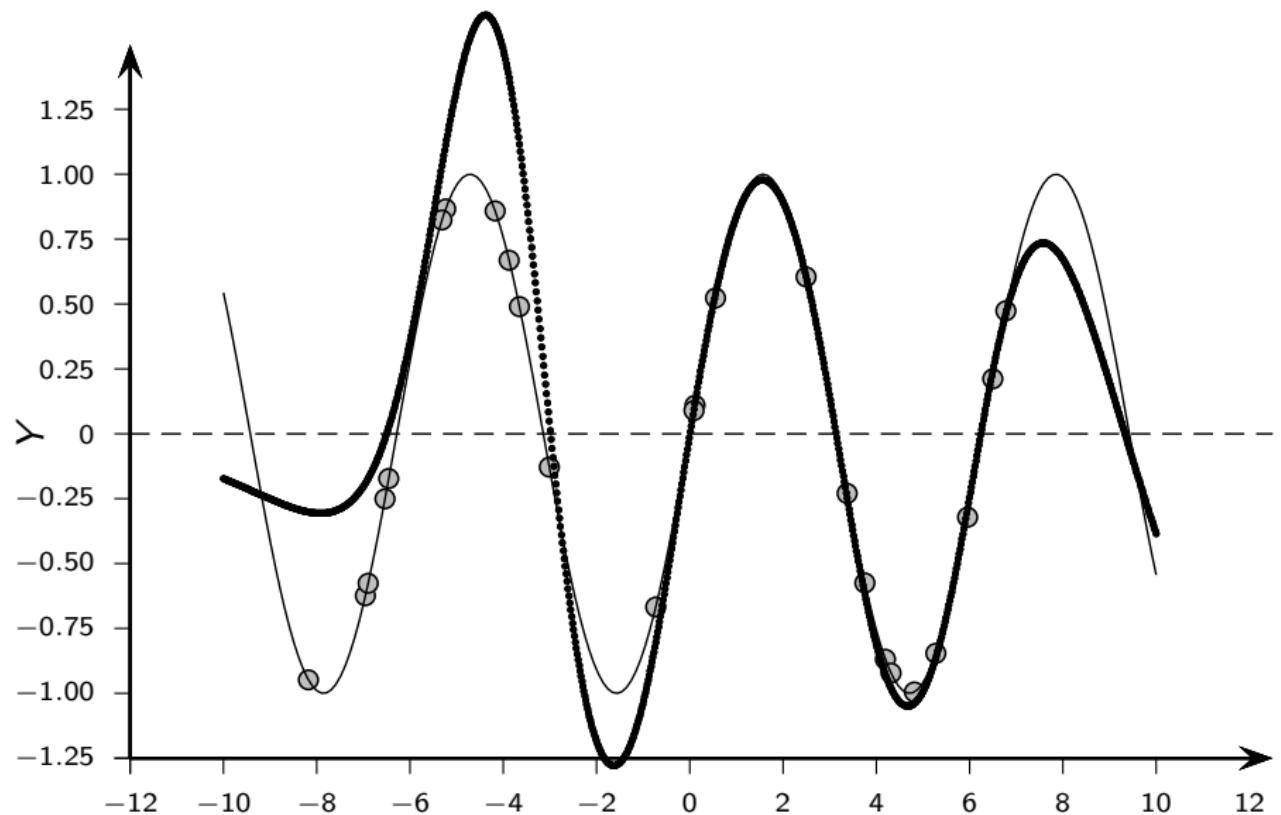
Тестовые данные состоят из 1000 точек, выбранных из равномерного распределения с тем же диапазоном. Желаемая выходная кривая показана на рисунке тонкой линией. Фактическая выходная кривая показана толстой линией.

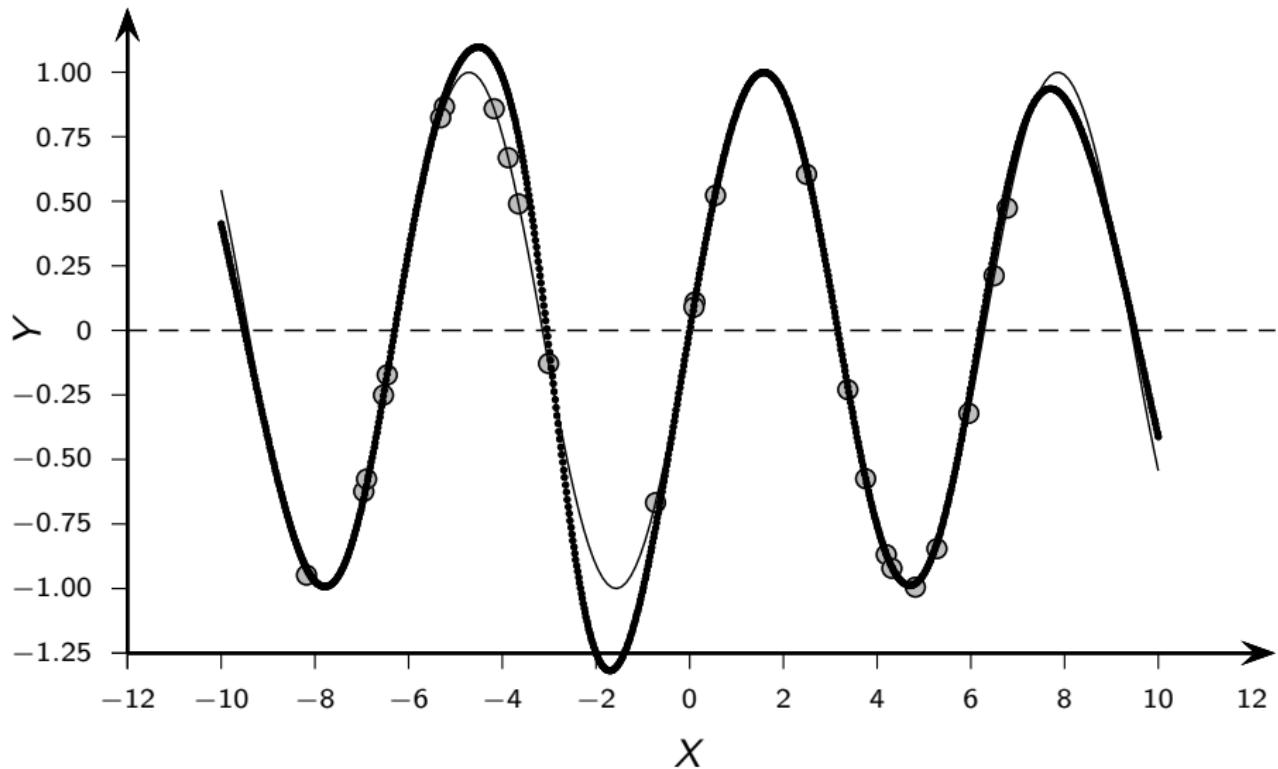
Использована MLP с одним входным нейроном ($d = 1$), десятью скрытыми нейронами ($m = 10$) и одним выходным нейроном ($p = 1$). Скрытые нейроны используют функцию активации гиперболический тангенс (\tanh), тогда как выходной блок использует тождественную функцию активации. Размер шага $\eta = 0.005$. Количество итераций стохастического градиентного спуска обозначается параметром t .

MLP для синусоиды ($t = 1$)

MLP для синусоиды ($t = 1000$)

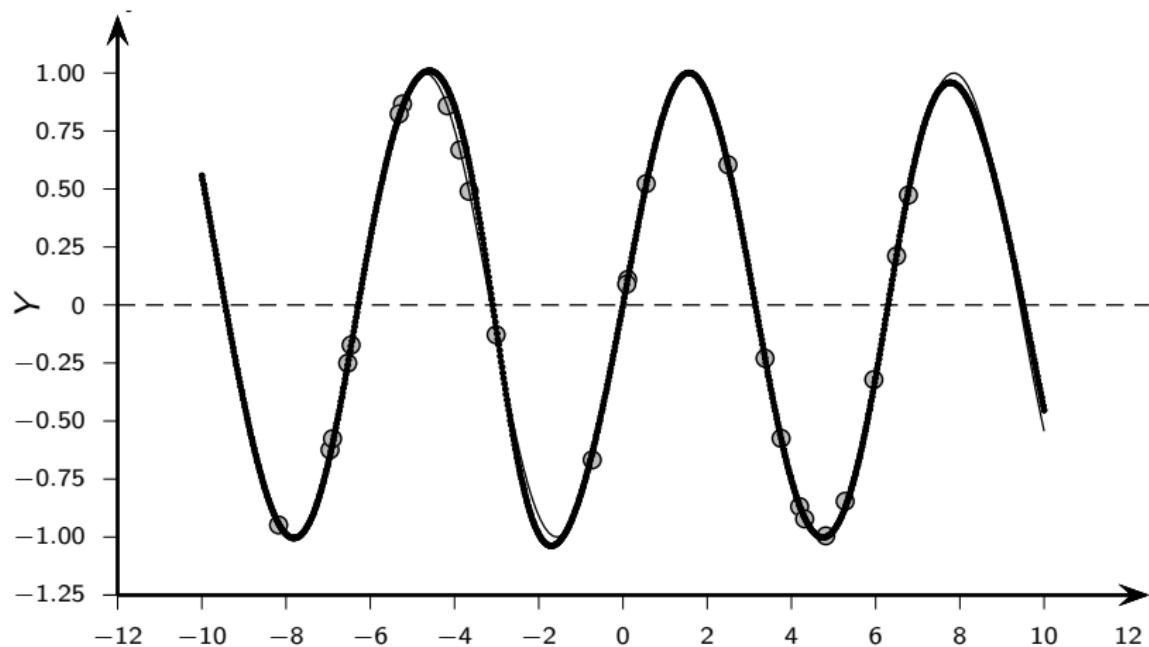
MLP для синусоиды ($t = 5000$)

MLP для синусоиды ($t = 10000$)

MLP для синусоиды ($t = 15000$)

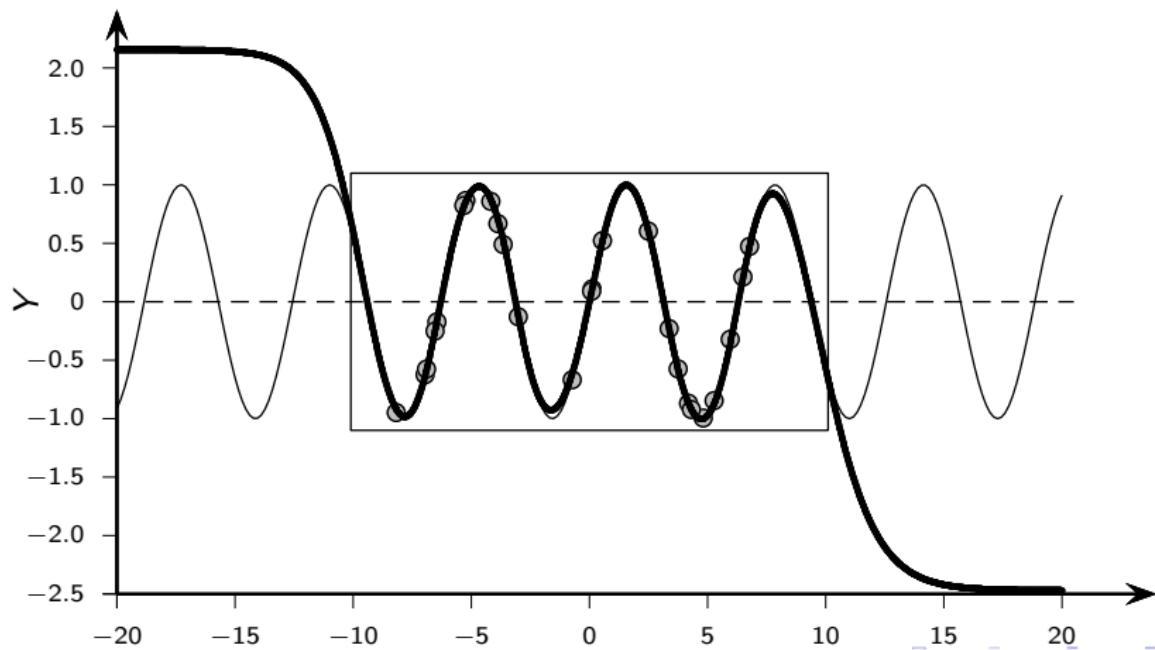
MLP для синусоиды ($t = 30000$)

Заметим, что даже для очень небольшого набора обучающих данных из 25 точек, выбранных случайным образом из синусоидальной кривой, нейронная сеть MLP достаточно хорошо аппроксимировала желаемую функцию.



MLP для синусоиды для диапазона $[-20, 20]$

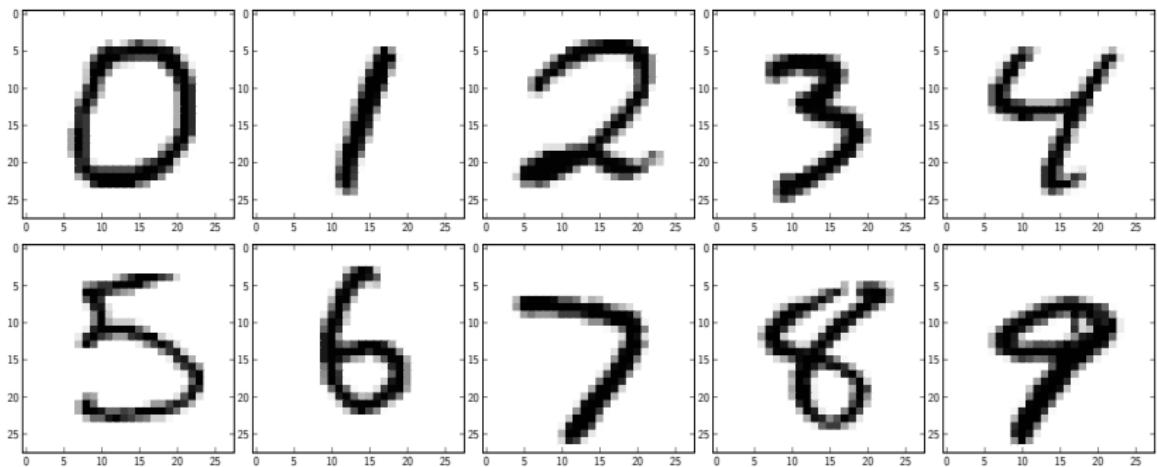
Однако нейронная сеть MLP на самом деле не изучила функцию синуса; скорее, сеть научилась аппроксимировать синус только в указанном диапазоне $[-10, 10]$.



MLP для классификации рукописных цифр

Рассматривается MLP с одним скрытым слоем для задачи предсказания правильной метки для рукописной цифры из базы данных MNIST, которая содержит 60 000 обучающих изображений, охватывающих 10 меток цифр от 0 до 9.

Каждое изображение (в виде оттенков серого) представляет собой матрицу пикселей 28×28 со значениями от 0 до 255. Каждый пиксель преобразуется в значение в интервале [0, 1] путем деления на 255.





MLP для классификации рукописных цифр

Поскольку изображения являются двумерными матрицами, сначала выпрямим их в вектор $\mathbf{x} \in \mathbb{R}^{784}$ с размерностью $d = 28 \times 28 = 784$. Это делается путем простого объединения всех строк изображений для получения одного длинного вектора.

Далее, поскольку выходные метки представляют собой категориальные значения, обозначающие цифры от 0 до 9, нам нужно преобразовать их в двоичные (числовые) векторы, используя однократное кодирование (one-hot encoding).

Таким образом, метка 0 кодируется как $\mathbf{e}_1 = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T \in \mathbb{R}^{10}$, метка 1 как $\mathbf{e}_2 = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)^T \in \mathbb{R}^{10}$ и так далее, и, наконец, метка 9 кодируется как $\mathbf{e}_{10} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1)^T \in \mathbb{R}^{10}$. То есть каждый входной вектор изображения \mathbf{x} имеет соответствующий целевой вектор отклика $\mathbf{y} \in \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{10}\}$.

Таким образом, входной слой для MLP имеет $d = 784$ нейрона, а выходной слой имеет $p = 10$ нейронов.



MLP для классификации рукописных цифр

Для скрытого слоя рассматривается несколько моделей нейронной сети MLP, каждая из которых имеет разное количество скрытых нейронов m . Мы пробуем варианты $m = 0, 7, 49, 98, 196, 392$, чтобы изучить эффект от увеличения количества скрытых нейронов от малого к большему.

Для скрытого слоя используется функция активации ReLU, а для выходного слоя используется функция активации softmax, поскольку целевой вектор ответа имеет только один нейрон со значением 1, а остальные равны 0.

Обратите внимание, что $m = 0$ означает, что скрытого слоя нет — входной слой напрямую связан с выходным слоем, что эквивалентно модели мультиклассовой логистической регрессии. Мы обучаем каждую сеть MLP для $t = 15$ эпох, используя размер шага $\eta = 0.25$.



MLP для классификации рукописных цифр

Окончательное количество ошибок на тестовых данных в конце обучения определяется следующей таблицей:

m	0	7	10	49	98	196	392
Кол-во ошибок	1677	901	792	546	495	470	454

Очевидно, что добавление скрытого слоя значительно повышает точность предсказания. Использование даже небольшого количества скрытых нейронов помогает повысить точность предсказания по сравнению с моделью логистической регрессии ($m = 0$). Например, использование $m = 7$ приводит к 901 ошибке (или частоте ошибок 9.01%) по сравнению с использованием $m = 0$, что приводит к 1677 ошибкам (или частоте ошибок 16.77%).

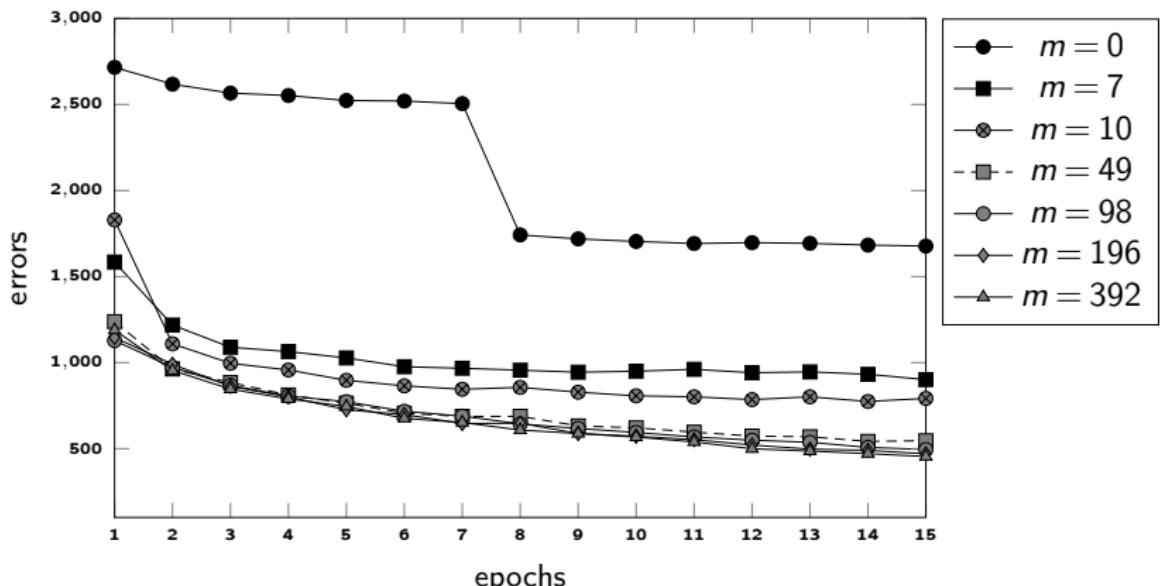
С другой стороны, по мере увеличения числа скрытых нейронов частота ошибок уменьшается, хотя и с уменьшающейся отдачей (темпом). При $m = 196$ частота ошибок составляет 4.7%, но даже после удвоения числа скрытых нейронов ($m = 392$) частота ошибок снижается до 4.54%. Дальнейшее увеличение m не снижает частоту ошибок.

MNIST: ошибка прогноза как функция числа эпох



Во время обучения мы отображаем после каждой эпохи количество неправильно классифицированных изображений для отдельного тестового набора MNIST, содержащего 10 000 изображений.

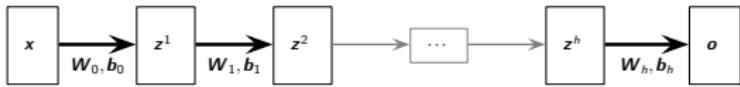
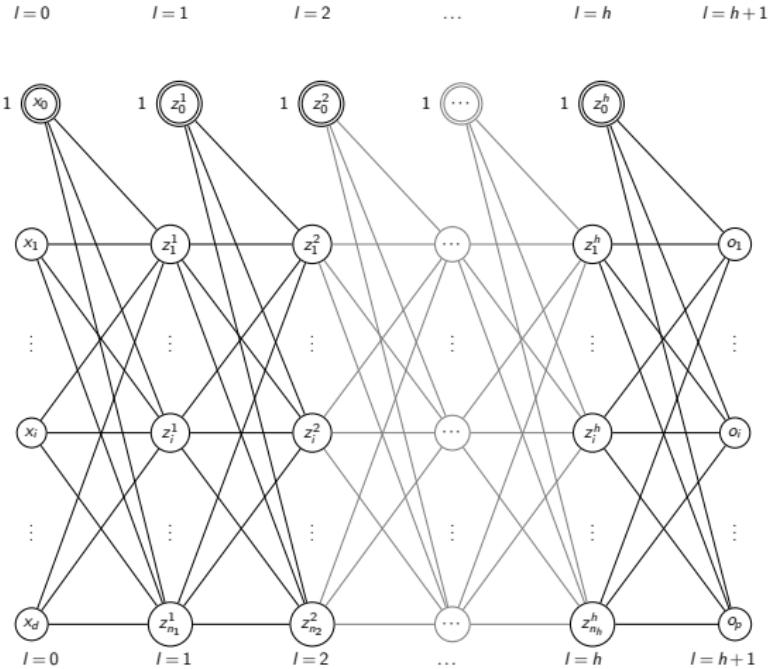
На рисунке показано количество ошибок от каждой из моделей (с разным количеством скрытых нейронов m) после каждой эпохи.



Глубокий многослойный персепtron (MLP)

Теперь обобщим прямое и обратное распространение на случай многих скрытых слоев, а также произвольные функции ошибки и активации нейронов.

Рассмотрим MLP с h скрытыми слоями, n входными точками $\mathbf{x}_i \in \mathbb{R}^d$ ($l = 0$) и вектором отклика $\mathbf{y}_i \in \mathbb{R}^p$ ($l = h + 1$).





Глубокий MLP – прямое распространение

Обычно в глубоком MLP одна и та же функция активации f^l используется для всех нейронов в заданном слое l .

Входной слой всегда использует тождественную активацию, поэтому f^0 – это тождественная функция. Кроме того, все нейроны смещения также используют тождественную функцию с фиксированным значением 1.

Скрытые слои обычно используют функции активации сигмоида, гиперболический тангенс или ReLU.

Выходной слой обычно использует функции активации сигмоида или softmax для задач классификации и тождественную функцию активации для задач регрессии.

Для $(\mathbf{x}, \mathbf{y}) \in \mathbf{D}$ глубокий MLP вычисляет выходной вектор как:

$$\begin{aligned}\mathbf{o} &= f^{h+1}(\mathbf{b}_h + \mathbf{W}_h^T \mathbf{z}^h) = f^{h+1}(\mathbf{b}_h + \mathbf{W}_h^T f^h(\mathbf{b}_{h-1} + \mathbf{W}_{h-1}^T \mathbf{z}^{h-1})) = \dots = \\ &= f^{h+1}(\mathbf{b}_h + \mathbf{W}_h^T f^h(\mathbf{b}_{h-1} + \mathbf{W}_{h-1}^T f^{h-1}(\dots f^2(\mathbf{b}_1 + \mathbf{W}_1^T f^1(\mathbf{b}_0 + \mathbf{W}_0^T \mathbf{x}))))))\end{aligned}$$



Глубокий MLP – обновление параметров

Рассмотрим обновление весов между текущим и следующим слоями, в том числе между входным и скрытым слоями, между двумя скрытыми слоями или между последним скрытым слоем и выходным слоем.

Пусть z_i^l — это нейрон слоя l , а z_j^{l+1} — нейрон следующего слоя $l + 1$. Пусть w_{ij}^l — вес между z_i^l и z_j^{l+1} , а b_j^l — смещение между z_0^l и z_j^{l+1} . Вес и смещение обновляются с использованием градиентного спуска:

$$w_{ij}^l = w_{ij}^l - \eta \nabla_{w_{ij}^l}, \quad b_j^l = b_j^l - \eta \nabla_{b_j^l},$$

где $\nabla_{w_{ij}^l}$ — градиент весов, а $\nabla_{b_j^l}$ — градиент смещения, т. е. частная производная функции ошибки по весам и смещению соответственно.



Глубокий MLP – обратное распространение

Можно использовать цепное правило, чтобы записать градиент весов и смещения следующим образом:

$$\nabla_{w_{ij}^l} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial w_{ij}^l} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^l} \frac{\partial \text{net}_j^l}{\partial w_{ij}^l} = \delta_j^{l+1} z_i^l = z_i^l \delta_j^{l+1},$$

$$\nabla_{b_j^l} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial b_j^l} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^l} \frac{\partial \text{net}_j^l}{\partial b_j^l} = \delta_j^{l+1}$$

Таким образом, обновления весов и смещений равны

$$\mathbf{W}_l = \mathbf{W}_l - \eta \nabla_{\mathbf{W}_l},$$

$$\mathbf{b}_l = \mathbf{b}_l - \eta \nabla_{\mathbf{b}_l},$$

где η — размер шага. Однако для вычисления градиентов весов и смещения для слоя l необходимо вычислить чистые градиенты $\boldsymbol{\delta}^{l+1}$ в слое $l + 1$.

Глубокий MLP – градиенты в выходном слое

Если все выходные нейроны независимы (например, при использовании линейной или сигмоидной активации), чистый градиент получается путем дифференцирования функции ошибки по чистому входу на выходных нейронах. То есть,

$$\delta_j^{h+1} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^{h+1}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial f^{h+1}(\text{net}_j^{h+1})} \frac{\partial f^{h+1}(\text{net}_j^{h+1})}{\partial \text{net}_j^{h+1}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial o_j} \frac{\partial f^{h+1}(\text{net}_j^{h+1})}{\partial \text{net}_j^{h+1}}$$

Если выходные нейроны не являются независимыми (например, при использовании функции активации softmax), то:

$$\delta_j^{h+1} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^{h+1}} = \sum_{i=1}^p \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial f^{h+1}(\text{net}_i^{h+1})} \frac{\partial f^{h+1}(\text{net}_i^{h+1})}{\partial \text{net}_j^{h+1}}$$

Для регрессии обычно используется ошибка SSE с линейной функцией активации, тогда как для логистической регрессии и классификации используется кросс-энтропийная функция ошибки с сигмоидальной активацией для бинарных классов и активацией softmax для задач с несколькими классами.

Глубокий MLP – градиенты в скрытых слоях

Предположим, что уже вычислен чистый градиент на слое $l + 1$, равный δ^{l+1} .

Поскольку нейрон z_j^l в слое l соединен со всеми нейронами в слое $l + 1$ (за исключением нейрона смещения z_0^{l+1}), для вычисления чистого градиента в z_j^l нужно учитывать ошибку каждого нейрона в слое $l + 1$ следующим образом:

$$\begin{aligned}\delta_j^l &= \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_j^l} = \sum_{k=1}^{n_{l+1}} \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial \text{net}_k^{l+1}} \frac{\partial \text{net}_k^{l+1}}{\partial f^l(\text{net}_j^l)} \frac{\partial f^l(\text{net}_j^l)}{\partial \text{net}_j^l} = \\ &= \frac{\partial f^l(\text{net}_j^l)}{\partial \text{net}_j^l} \sum_{k=1}^{n_{l+1}} \delta_k^{l+1} w_{jk}^l,\end{aligned}$$

Таким образом, чистый градиент на z_j^l в слое l зависит от производной функции активации по net_j^l и взвешенной суммы чистых градиентов от всех нейронов z_k^{l+1} на следующем слое $l + 1$.



Глубокий MLP – градиенты в скрытых слоях

Для часто используемых функций активации в скрытом слое мы имеем

$$\partial \mathbf{f}^l = \begin{cases} 1 & \text{линейная активация} \\ z^l (1 - z^l) & \text{сигмоида} \\ 1 - z^l \odot z^l & \text{гиперболический тангенс} \end{cases}$$

Векторы чистых градиентов вычисляются рекурсивно, начиная с выходного слоя $l = h + 1$, затем скрытого слоя $l = h$ и так далее, пока, наконец, не будут вычислены чистые градиенты в первом скрытом слое $l = 1$. То есть,

$$\boldsymbol{\delta}^h = \partial \mathbf{f}^h \odot (\mathbf{W}_h \boldsymbol{\delta}^{h+1})$$

$$\boldsymbol{\delta}^{h-1} = \partial \mathbf{f}^{h-1} \odot (\mathbf{W}_{h-1} \boldsymbol{\delta}^h) = \partial \mathbf{f}^{h-1} \odot \left(\mathbf{W}_{h-1} \left(\partial \mathbf{f}^h \odot (\mathbf{W}_h \boldsymbol{\delta}^{h+1}) \right) \right)$$

⋮

$$\boldsymbol{\delta}^1 = \partial \mathbf{f}^1 \odot \left(\mathbf{W}_1 \left(\partial \mathbf{f}^2 \odot \left(\mathbf{W}_2 \dots \left(\partial \mathbf{f}^h \odot (\mathbf{W}_h \boldsymbol{\delta}^{h+1}) \right) \right) \right) \right)$$



Обучение глубокого MLP: алгоритм SGD

Входными данными для алгоритма обучения глубокого MLP при помощи стохастического градиентного спуска являются:

- обучающий набор данных \mathbf{D} с признаками \mathbf{x} и откликом \mathbf{y}
- количество скрытых слоев h в сети MLP
- шаг обучения η
- максимальное количество итераций maxiter
- количество нейронов в скрытых слоях n_1, n_2, \dots, n_h
- функции активации в скрытых и выходном слое f^1, f^2, \dots, f^{h+1}

Deep-MLP-Training ($D, h, \eta, \text{maxiter}, n_1, n_2, \dots, n_h, f^1, f^2, \dots, f^{h+1}$):

```
1  $n_0 \leftarrow d$  // input layer size
2  $n_{h+1} \leftarrow p$  // output layer size
   // Initialize weight matrices and bias vectors
3 for  $l = 0, 1, 2, \dots, h$  do
4    $b_l \leftarrow$  random  $n_{l+1}$  vector with small values
5    $W_l \leftarrow$  random  $n_l \times n_{l+1}$  matrix with small values
6  $t \leftarrow 0$  // iteration counter
```



Обучение глубокого MLP: алгоритм SGD

```
7 repeat
8     foreach  $(x_i, y_i) \in D$  in random order do
9          $z^0 \leftarrow x_i$  // Feed-Forward Phase
10        for  $l = 0, 1, 2, \dots, h$  do  $z^{l+1} \leftarrow f^{l+1}(b_l + W_l^T \cdot z^l)$ 
11         $o_i \leftarrow z^{h+1}$ 
12        if independent outputs then // Backpropagation phase
13             $\delta^{h+1} \leftarrow \partial f^{h+1} \odot \partial E_{x_i}$  // net gradients at output
14        else
15             $\delta^{h+1} \leftarrow \partial F^{h+1} \cdot \partial E_{x_i}$  // net gradients at output
16        for  $l = h, h-1, \dots, 1$  do  $\delta^l \leftarrow \partial f^l \odot (W_l \cdot \delta^{l+1})$  // net gradients
17        for  $l = 0, 1, \dots, h$  do // Gradient Descent Step
18             $\nabla_{W_l} \leftarrow z^l \cdot (\delta^{l+1})^T$  // weight gradient matrix at layer  $l$ 
19             $\nabla_{b_l} \leftarrow \delta^{l+1}$  // bias gradient vector at layer  $l$ 
20        for  $l = 0, 1, \dots, h$  do
21             $W_l \leftarrow W_l - \eta \cdot \nabla_{W_l}$  // update  $W_l$ 
22             $b_l \leftarrow b_l - \eta \cdot \nabla_{b_l}$  // update  $b_l$ 
23     $t \leftarrow t + 1$ 
24 until  $t \geq \text{maxiter}$ 
```



Проблемы обучения нейронных сетей

Одной из проблем машинного обучения в целом и глубокого обучения нейронных сетей в частности является т.н. переобучение.

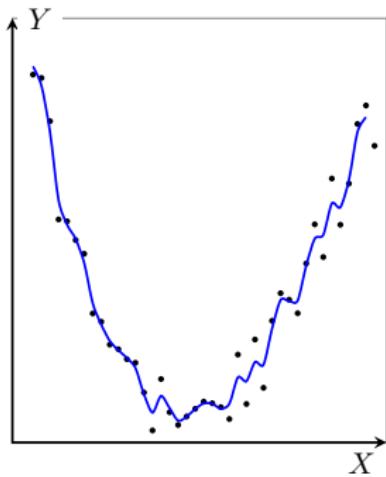
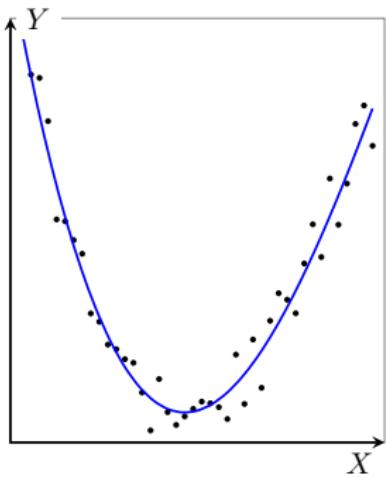
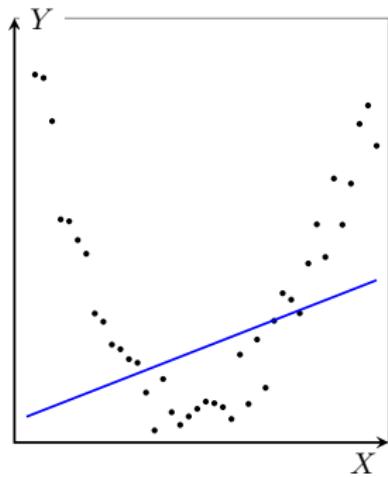
Переобучение (overfitting, overtraining) — это ситуация, когда обучаемая модель для данных обучающего набора прогнозирует отклики, которые близко или даже точно соответствуют откликам в этом наборе, однако для данных, не участвовавших в процессе обучения, модель вырабатывает предсказания низкого качества. Переобучение может возникать

- при использовании слишком сложных моделей
- при слишком долгом процессе обучения
- при неудачной обучающей выборке

Недообучение (underfitting) — это ситуация, когда обучаемая модель не обеспечивает приемлемого качества (достаточно малой величины средней ошибки) даже на обучающем наборе. Недообучение может возникать

- при использовании слишком простых моделей
- при прекращении процесса обучения до достижения состояния с достаточно малой ошибкой
- при неудачной обучающей выборке

Переобучение и недообучение модели





Переобучение нейронных сетей

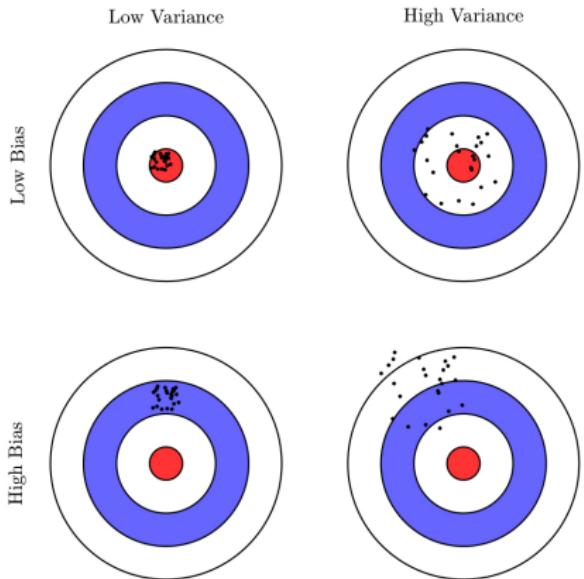
Проблема переобучения в той или иной степени характерна для всех видов моделей машинного обучения, но особенно остро она стоит для нейронных сетей. В процессе обучения производится подгонка весов нейронной сети таким образом, что сеть преобразовывала входные данные к желаемым выходным данным в соответствии с зависимостями, обнаруженными в обучающих данных.

Однако, если нейронную сеть обучать слишком долго или с использованием слишком большого числа параметров (весов), то произойдёт переобучение. Это связано с тем, что с определённого момента сеть начинает «подстраиваться» не под общие зависимости в данных, а под особенности обучающего набора данных, который может содержать аномальные значения, ошибки и т.д.

Как следствие, нейронная сеть начнёт проверять новые, предъявляемые ей входные данные не на соответствие общим зависимостям, а на соответствие отдельным экземплярам данных из обучающего набора. В итоге модель сможет корректно предсказать отклик для новых данных только в том случае, если они совпадут с данными в обучающем наборе.



Показатель ошибки обученной модели машинного обучения может зависеть от используемой для обучения обучающей выборки.



Смещение (bias) отражает ошибку, связанную с неверными допущениями модели машинного обучения. Высокое смещение может привести к недообучению модели.

Дисперсия (variance) отражает ошибку, вызванную большой чувствительностью модели к небольшим отклонениям в обучающем наборе. Высокая дисперсия может привести к переобучению модели.



Ожидаемые потери модели машинного обучения

Идеальный (оптимальный) алгоритм машинного обучения минимизирует функцию потерь. Поскольку истинное значение отклика y для тестовой точки \mathbf{x} неизвестно, целью обучения модели M по набору данных \mathbf{D} является минимизация **ожидаемых потерь** (по всем откликам):

$$\mathbb{E}_y [\mathcal{L}(y, M(\mathbf{x})) \mid \mathbf{x}] = \sum_y \mathcal{L}(y, M(\mathbf{x})) \mathbb{P}[y \mid \mathbf{x}],$$

где $\mathcal{L}(y, \hat{y})$ – функция потерь (ошибки) для истинного отклика y и прогноза \hat{y} , $\mathbb{P}[y \mid \mathbf{x}]$ – это условная вероятность отклика y для заданной тестовой точки \mathbf{x} , и \mathbb{E}_y обозначает, что математическое ожидание вычисляется по различным откликам y . Ожидаемые потери для функции квадратичных потерь $\mathcal{L}(y, \hat{y}) = (y - \hat{y})^2$ для заданной точки \mathbf{x} и набора данных \mathbf{D} можно разложить на **смещение** (bias, систематическую ошибку) и **дисперсию** (variance) следующим образом:

$$\mathbb{E}_y [L(y, M(\mathbf{x}, \mathbf{D})) \mid \mathbf{x}, \mathbf{D}] = \underbrace{\mathbb{E}_y [(y - \mathbb{E}_y [y \mid \mathbf{x}])^2 \mid \mathbf{x}, \mathbf{D}]}_{var(y|\mathbf{x})} + \underbrace{(M(\mathbf{x}, \mathbf{D}) - \mathbb{E}_y [y \mid \mathbf{x}])^2}_{\text{квадратичная ошибка}}$$

Здесь $M(\mathbf{x}, \mathbf{D})$ обозначает прогноз модели, обученной на наборе данных \mathbf{D} , для точки \mathbf{x} .



Ожидаемые потери по всем обучающим выборкам

Для различных обучающих выборок \mathbf{D} обученная модель будет иметь различный уровень ожидаемых потерь. Средняя или ожидаемая квадратичная ошибка для данной тестовой точки \mathbf{x} по всем обучающим выборкам \mathbf{D} тогда задается как

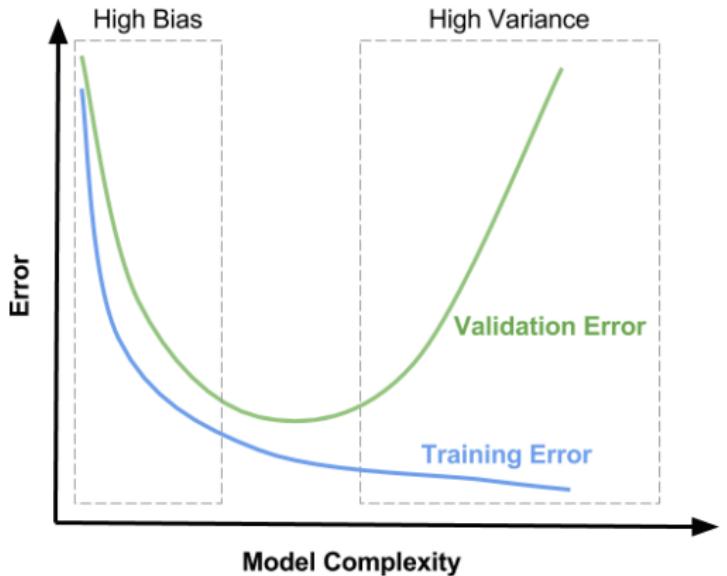
$$\mathbb{E}_{\mathbf{D}} \left[(M(\mathbf{x}, \mathbf{D}) - \mathbb{E}_y [y | \mathbf{x}])^2 \right] = \\ = \underbrace{\mathbb{E}_{\mathbf{D}} \left[(M(\mathbf{x}, \mathbf{D}) - \mathbb{E}_{\mathbf{D}} [M(\mathbf{x}, \mathbf{D})])^2 \right]}_{\text{дисперсия (variance)}} + \left(\underbrace{\mathbb{E}_{\mathbf{D}} [M(\mathbf{x}, \mathbf{D})] - \mathbb{E}_y [y | \mathbf{x}]}_{\text{смещение (bias)}} \right)^2$$

Здесь $\mathbb{E}_{\mathbf{D}}$ обозначает, что математическое ожидание вычисляется по различным обучающим выборкам \mathbf{D} .

Таким образом, высокие ожидаемые потери модели машинного обучения могут быть вызваны высоким смещением и/или высокой дисперсией модели.

Конфликт между смещением и дисперсией

Конфликт между смещением и дисперсией (дилемма bias-variance) — это противоречие при попытке одновременно минимизировать смещение и дисперсию, тогда как уменьшение одного приводит к увеличению другого.



При небольшой сложности модели наблюдается высокое смещение. При усложнении модели смещение уменьшается, но дисперсия увеличивается и приводит к проблеме высокой дисперсии.



Кривая обучения — это графическое представление зависимости показателя качества обучения (по вертикальной оси) от определенной характеристики процесса обучения (по горизонтальной оси). Кривые обучения (learning curves) — это широко используемый диагностический инструмент для таких алгоритмов машинного обучения, как глубокое обучение, которые обучаются постепенно. Во время обучения оценивается производительность модели как на обучающем (train) наборе данных, так и на проверочном (validation) наборе данных, и строится график показателя производительности для каждого шага обучения (каждой эпохи глубокого обучения). Анализ кривых обучения можно использовать для диагностики проблем с обучением, таких как недообучение или переобучение, а также того, являются ли обучающий и проверочные наборы данных достаточно репрезентативными.

Обсудим, как можно использовать кривые обучения, чтобы:

- диагностировать такие свойства модели, как недообучение или переобучение
- диагностировать проблемы, связанные с диспропорциями в представлении данных



Диагностика свойств модели

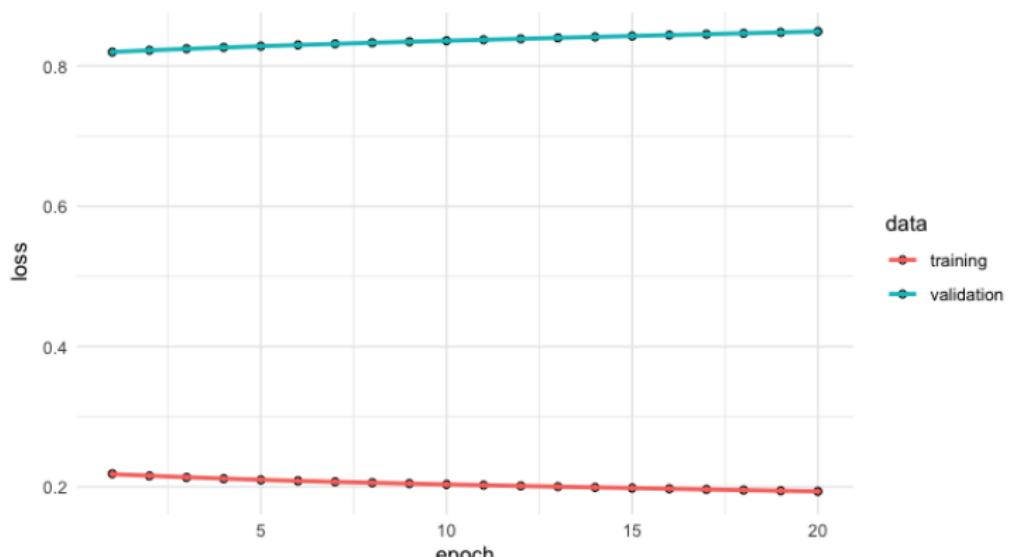
Форму и наклон кривой обучения можно использовать для диагностики свойств модели машинного обучения и, в свою очередь, возможно, подсказать тип изменений конфигурации, которые можно внести для улучшения обучения и/или производительности. На кривых обучения принято выделять три общих случая:

- недообучение (underfit)
- переобучение (overfit)
- оптимальное обучение (optimal fit)

Рассмотрим каждый случай более подробно на примерах. В примерах предполагается, что рассматривается показатель потерь (ошибки) модели, а это означает, что меньшие относительные значения по оси Y указывают на лучшую производительность модели.

Кривые обучения при недообучении модели

Термин «недообучение» относится к модели, которая не обучилась должным образом на обучающих данных, чтобы выйти на достаточно низкое значение ошибки обучения. Есть два распространенных сигнала о недостаточности обучения. Во-первых, кривая обучения может быть плоской линией или иметь запутанные значения с высокими потерями, что указывает на то, что модель вообще не смогла изучить обучающий набор данных.





Решение при недообучении модели

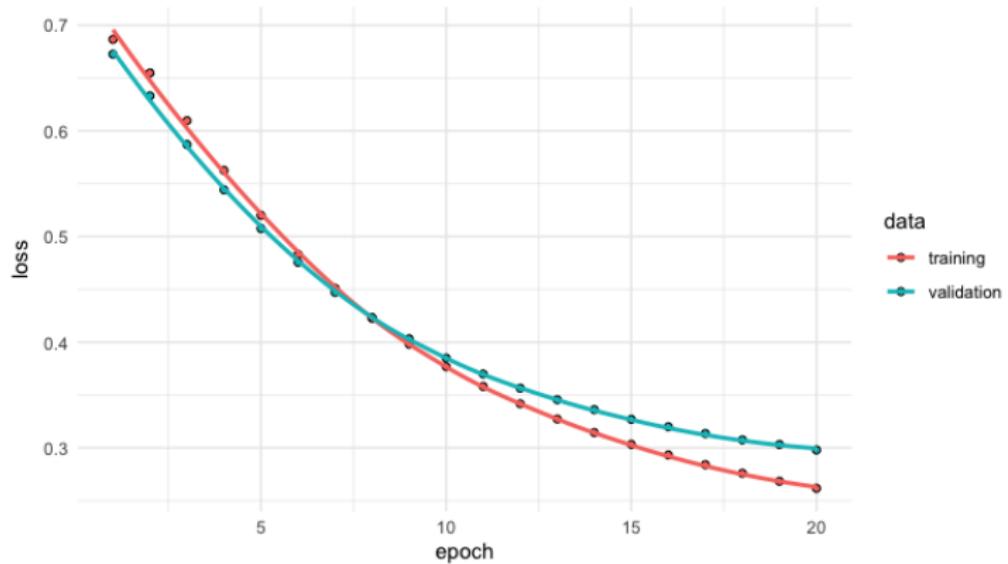
Пример на предыдущем слайде встречается, когда модель не имеет подходящей гибкости (емкости) для сложного набора данных.

Решение:

- Добавьте больше наблюдений (точек данных). Возможно, данных недостаточно, чтобы существующие закономерности в данных стали понятны модели машинного обучения.
- Добавьте больше признаков (features) в набор данных. Иногда модель не может обучиться на том основании, что имеющихся признаков недостаточно.
- Уменьшите любую используемую регуляризацию модели. Если у вас указаны явные параметры регуляризации (например, процент отсева, регуляризация весов), то удалите или уменьшите эти параметры.
- Увеличьте сложность модели. Возможностей модели может быть недостаточно для выявления и изучения существующих в данных закономерностей.

Кривые обучения при недообучении модели

Недообученную модель также можно определить по потерям на обучающих и проверочных данных, которые продолжают уменьшаться в конце графика (в конце периода обучения). Это указывает на то, что модель способна к дальнейшему обучению и что процесс обучения был остановлен преждевременно.





Решение:

- Увеличивайте количество эпох обучения до тех пор, пока кривая для проверочных данных не перестанет улучшаться. Можно добавить в модель обратный вызов (callback) досрочной остановки, чтобы определить, сколько эпох требуется для обучения модели.
- Если достижение минимума кривой обучения на проверочных данных занимает много времени, увеличьте скорость обучения, чтобы ускорить градиентный спуск, а также добавьте обратный вызов (callback) для автоматической регулировки скорости обучения.



Кривые обучения при переобучении модели

Термин «переобучение» относится к модели, которая слишком хорошо изучила набор обучающих данных, включая статистический шум или случайные флюктуации в обучающем наборе данных.

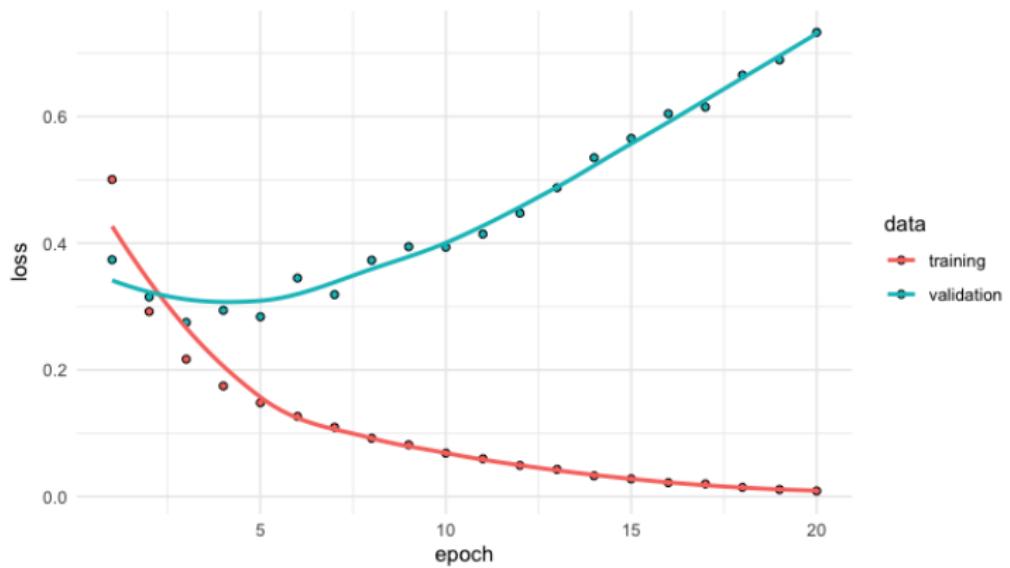
Проблема с переобучением заключается в том, что чем более специализированной модель становится для конкретных обучающих данных, тем хуже она может обобщаться на новые данные, что приводит к увеличению ошибки обобщения. Переобучение становится очевидным, когда:

- потери на обучающих данных продолжают уменьшаться с каждой новой эпохой обучения
- одновременно потери на проверочных (валидационных) данных снижаются до минимума и потом начинают увеличиваться.

Модель, которая переобучена, не обязательно является плохой. Фактически, переобучение сигнализирует о том, что модель извлекла все закономерности в данных, который могла изучить эта конкретная модель. Проблемы, о которых следует беспокоиться при переобучении, — это величина и точка перегиба.

Кривые обучения при переобучении модели

Если модель на ранних стадиях обучения переобучается и имеет острую U-образную форму кривой обучения для проверочных данных, то это часто указывает на избыточную сложность модели и/или слишком высокую скорость ее обучения.





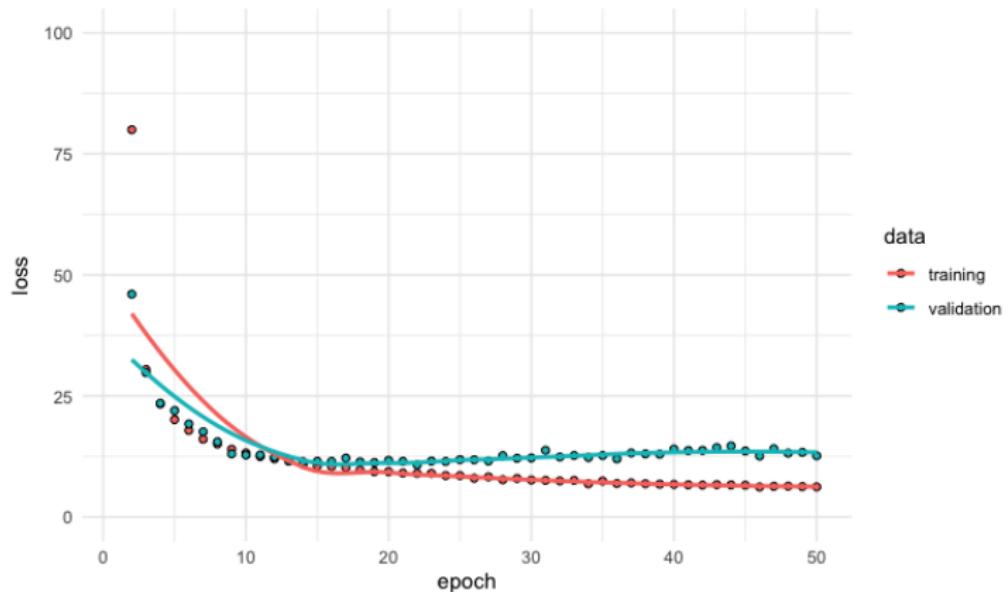
Решение при переобучении модели

Решение:

- Отрегулируйте, как быстро обучается модель, уменьшив скорость обучения. Добавьте обратный вызов (callback), чтобы автоматически снижать скорость обучения по мере того, как потери на проверочном наборе выходят на плато (постоянный уровень).
- Отрегулируйте емкость модели, уменьшив количество и/или размер (количество нейронов) скрытых слоев.
- Выполните регуляризацию весов, чтобы ограничить сложность нейронной сети.
- Выполните регуляризацию случайных закономерностей, добавляя отсев (dropout), чтобы минимизировать вероятность обучения шуму в данных.

Кривые обучения при переобучении модели

Часто переобучение можно свести к минимуму, но редко его можно полностью устраниТЬ и при этом минимизировать потери. Следующий рисунок иллюстрирует случай, в котором переобучение сведено к минимуму, но все же присутствует.





Решение при переобучении модели

Решение:

- Добавьте обратный вызов (callback) досрочной остановки (EarlyStopping), чтобы остановить обучение, как только проверочная кривая (validation curve) перестанет улучшаться.
- Добавьте параметр `restore_best_weights = TRUE` в свой обратный вызов (callback), чтобы ваша окончательная модель использовала веса эпохи с лучшим показателем потерь.



Оптимальные кривые обучения

Целью алгоритма обучения является оптимальное обучение (optimal fit). Потери модели почти всегда будут ниже в обучающем наборе данных по сравнению с проверочным набором данных. Это означает, что должен быть некоторый разрыв между обучающей и проверочной кривыми обучения. Этот разрыв называется разрывом обобщения (generalization gap).

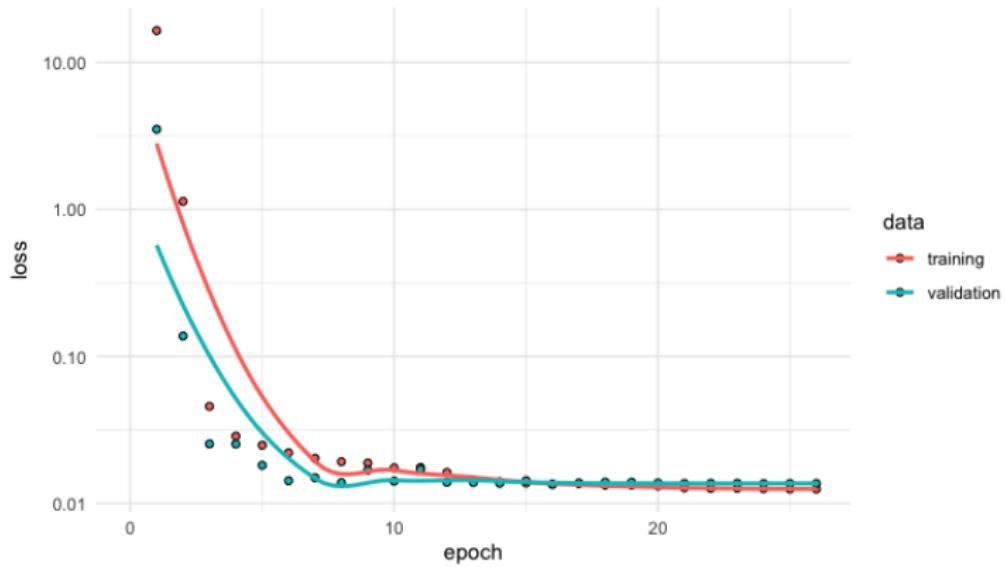
Оптимальное обучение – это такое обучение, при котором:

- График потерь на обучающем наборе данных уменьшается до точки (уровня) стабильности.
- График потерь на проверочном наборе данных уменьшается до точки (уровня) стабильности.
- Разрыв обобщения минимален (почти нулевой в идеальной ситуации).

Продолжение обучения оптимально обученной модели, скорее всего, приведет к ее переобучению.

Оптимальные кривые обучения

Пример графика ниже демонстрирует случай оптимального обучения в предположении, что найден глобальный минимум функции потерь.





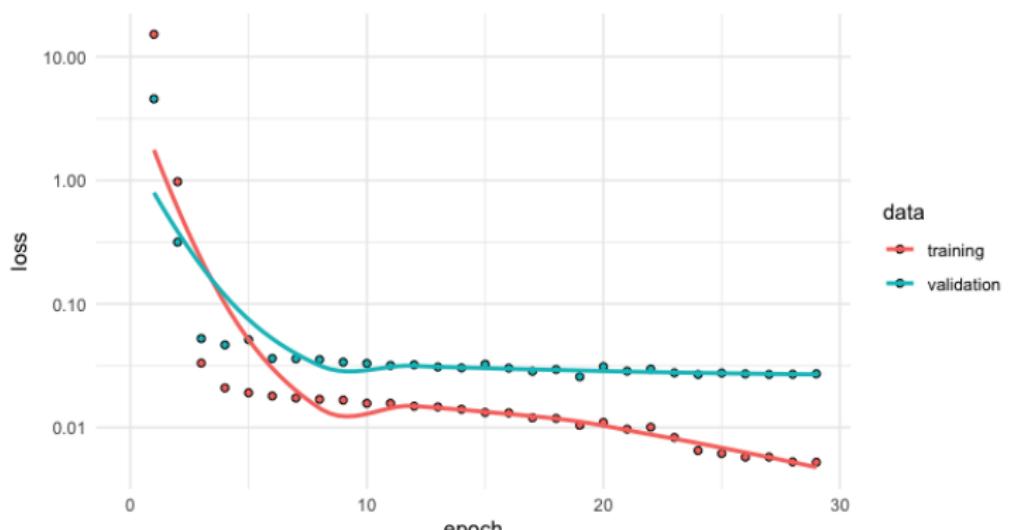
Кривые обучения также можно использовать для диагностики свойств набора данных и определения его относительной репрезентативности. Нерепрезентативный набор данных означает набор данных, который может не отражать статистические характеристики по сравнению с другим набором данных, взятым из той же области, например, между обучающим и проверочным наборами данных. Обычно такое может произойти, если количество точек в наборе данных слишком мало или если определенные характеристики не представлены адекватно по сравнению с другим набором данных.

Можно наблюдать два распространенных случая:

- обучающий набор данных относительно нерепрезентативен
- проверочный набор данных относительно нерепрезентативен

Проблемы с обучающим набором данных

Нерепрезентативность обучающего набора данных означает, что обучающий набор данных не предоставляет достаточной информации для обучения по сравнению с проверочным набором данных, используемым для ее оценки. Эту ситуацию можно определить по кривой обучения для обучающих данных, которая показывает улучшение потерь, и аналогичной кривой обучения для проверочных данных, которая также показывает улучшение потерь, но при этом между обеими кривыми остается большой разрыв.





Решение при проблемах с обучающими данными

Такое может произойти, когда:

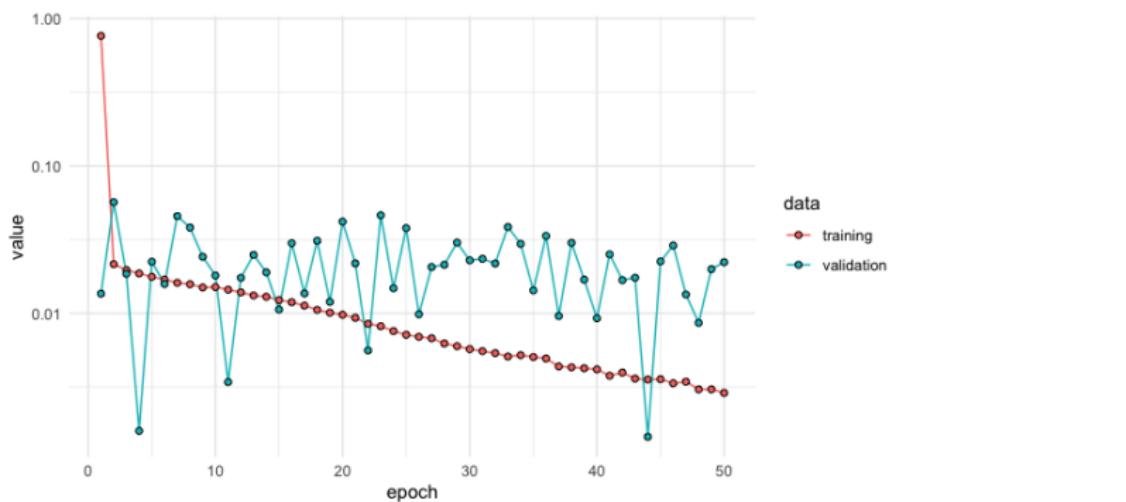
- В обучающем наборе данных слишком мало точек по сравнению с проверочным набором данных.
- Обучающий набор данных содержит признаки с меньшей дисперсией, чем проверочный набор данных.

Решение:

- Добавьте больше записей. Возможно, у вас недостаточно данных для выявления закономерностей, присутствующих как в обучающих данных, так и в проверочных данных.
- При использовании сетей CNN воспользуйтесь аугментацией данных, чтобы увеличить вариативность признаков в обучающих данных.
- Убедитесь, что вы случайным образом выбираете записи для использования в обучающем и проверочном наборах. Если данные упорядочены по какому-либо признаку, то проверочные данные могут содержать значения признаков, не представленные в обучающих данных.
- Выполните перекрестную проверку (кросс-валидацию), чтобы все данные могли быть представлены как в обучающем, так и в проверочном наборе.

Проблемы с проверочным набором данных

Нерепрезентативность проверочного набора данных означает, что набор не предоставляет достаточной информации для оценки способности модели к обобщению. Это может произойти, если в проверочном наборе слишком мало точек по сравнению с обучающим набором. Этот случай можно определить по кривой для обучающих данных, которая демонстрирует снижение потерь, и по кривой для проверочных данных, которая показывает шумные движения и небольшое улучшение потерь или его отсутствие.





Решение при проблеме с проверочными данными

Решение:

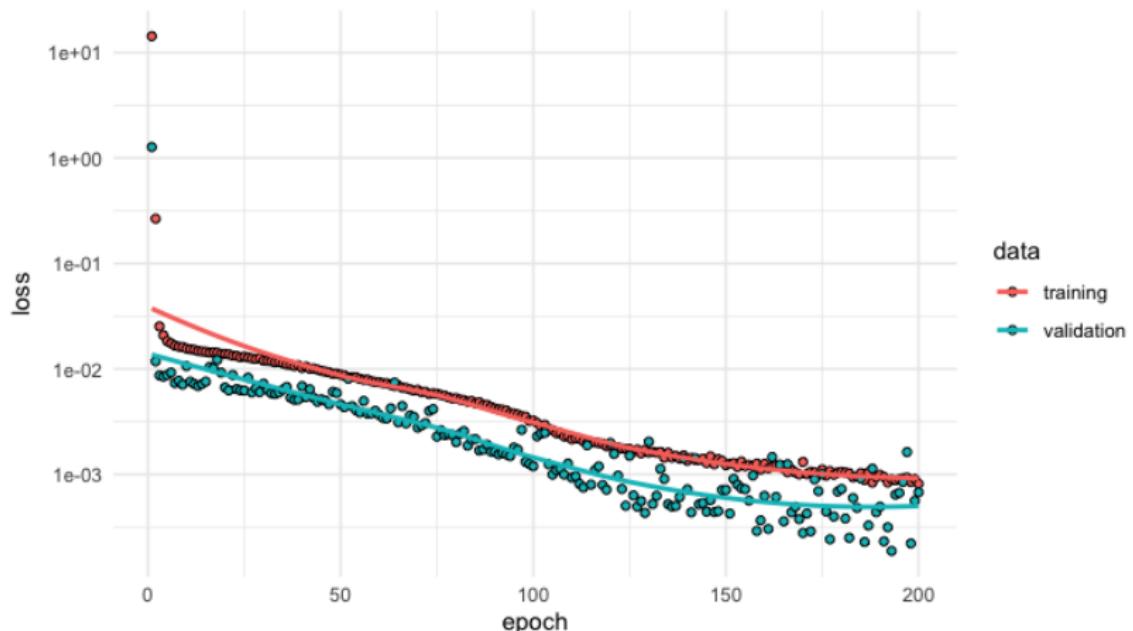
- Добавьте больше записей в проверочный набор данных.
- Если количество записей ограничено, то выполните перекрестную проверку (кросс-валидацию), чтобы все имеющиеся данные имели возможность быть представлены как в обучающем, так и в проверочном наборе.

Также проблемы с проверочным набором данных можно определить по потерям на наборе, которые ниже потерь на обучающем наборе данных, независимо от того, сколько итераций обучения выполняется. Это может произойти по разным причинам, но обычно это связано с:

- утечкой информации, когда признак в обучающих данных напрямую связан с данными и откликами в проверочном наборе
- плохими процедурами формирования случайной выборки, при которых в обучающем и проверочном наборах данных существуют повторяющиеся записи
- проверочный набор данных содержит признаки с меньшей дисперсией, чем обучающий набор данных

Проблемы с проверочным набором данных

В этом случае модели может быть легче прогнозировать отклик на проверочном наборе данных, чем на обучающем наборе данных.





Решение:

- Убедитесь, что в обучающем и проверочном наборах данных нет повторяющихся записей.
- Убедитесь, что нет утечки информации между обучающим и проверочным наборами данных.
- Убедитесь, что записи выбираются случайным образом для использования в обучающем и проверочном наборах так, чтобы дисперсия признаков была примерно одинаковой в обоих наборах.
- Выполните перекрестную проверку (кросс-валидацию), чтобы все данные могли быть представлены как в обучающем, так и в проверочном наборе данных.



- ❶ Применение проверочной (*validation*) выборки. Проверочная выборка формируется из набора данных случайным образом и применяется для оценки ошибки модели, но не влияет на корректировку весов сети. В начале обучения сети ошибка и на обучающей, и на проверочной выборках уменьшаются. Но начиная с какого-то момента ошибка на проверочной выборке начинает расти. Это сигнализирует о начале переобучения и необходимости принудительно остановить процесс обучения.
- ❷ Использование **кросс-валидации** (перекрёстной проверки). Все данные, на которых строится модель, разделяются на k блоков равного размера. При этом обучение производится на $k - 1$ блоках, а проверка — на k -м. Процедура повторяется k раз, при этом для проверки каждый раз выбирается другой блок. В результате все блоки оказываются используемыми и как обучающие, и как проверочные.
- ❸ Выбор **конфигурации** нейронной сети (числа слоёв и нейронов) так, чтобы количество параметров модели (весов) было в 2-3 раза меньше числа точек обучающей выборки. Если число параметров модели и точек данных окажется соизмеримым, то сеть запомнит все комбинации вход-выход в обучающих данных, и будет воспроизводить их, а на новых данных допускать ошибки.