Deep Learning

Mario Dohr

October 25, 2022

1 Basic neural network architectures

The loss function gives the loss for one sample. For example the squared loss is $L(y, \tilde{y}) = (y - \tilde{y})^2$. The risk is the expectation of the loss $R\mathbb{E}_z(L(y, \tilde{y}))$ and the empirical risk is $R_{emp} = \frac{1}{N}\sum_{n=1}^{N} L(y^n, \tilde{y}^n)$. This is the function we try to minimize to find the parameters of the neural network.

1.1 Linear model:Neuron

In this simple model we express y as a linear function of x

$$y = w_1 x + x_0$$

We try to find a line which minimizes the deviation between y and the regression line. The line that optimizes the criterion is the *least square line.*

1.1.1 Model

The linear neural network is

$$g(x; w) = a = w^T x$$

The general form is

$$y = g(x; w) + \epsilon$$

where ϵ is assumed as Gaussian noise with $\epsilon \sim \mathcal{N}(0, \sigma)$. This means, that

$$p(y|x, w, \sigma) = \mathcal{N}(y|(g(x; w), \sigma)$$

This means, we are looking for the mean of this distribution.

$$\mathbb{E}(y|x) = \int yp(y|x)dy = g(x; w)$$

1.1.2 Loss function

To solve the problem we use the maximum likelihood approach.

$$p(y|X, w, \sigma = \prod_{n=1}^{N} p(y^n|x^n, w, \sigma) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(y^n - w^T x^n)^2}{2\sigma^2}$$

But we don't try to maximize the likelihood but the negative log-likelihood

$$\log p(y|X, w, \sigma) = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}S(w, X, y)$$

where

$$S(w, X, y) = \frac{1}{2}\sum_{n=1}^{N}(\underbrace{y^n - w^T x^n}_{\epsilon^n})^2$$

where S is the *sum of squared residuals* and $L(y, g(x, w)) = \frac{1}{2}(y - g(x, q))^2$ is the **squared loss**. To maximize the likelihood we only need to minimize the negative log-likelihood we only need to minimize $S(\tilde{w}, X, y) = S(\tilde{w})$, since the log is monotone and all other terms are constant.

$$S(\tilde{w}) = \frac{1}{2}\sum_{n=1}^{N}(y^n - \tilde{w}^T x^n)^2 = \frac{1}{2}(y - X\tilde{w})^T(y - X\tilde{w})$$

1.1.3 Learning

To solve this, we have to solve

$$\tilde{w} = \arg\min_{\tilde{w}} S(\tilde{w})$$

This can be done be taking the derivative and set this to zero to get a closed form solution.

$$\nabla_w \log p(y|X, w, \sigma) = \frac{1}{\sigma^2}\sum_{n=1}^{N}(w^T x^n - y^n)x^n$$

From that we get the closed form solution

$$\tilde{w} = (X^T X)^{-1} X^T y$$

Another way to solve this is (stochastic) gradient descent.

$$w^{new} = w^{old} - \eta\nabla_w S(w)$$

1.2 Perceptron

The perceptron is used for linear binary classification tasks. The classes are $y \in \{-1, 1\}$

1.2.1 Model

For the perceptron we use the following model

$$g(x; w) = sign(g(x; w)) = a = sign(w^T x)$$

Here we are looking for a hyperplane which separates the labels. The perceptron only works for labels that are linearly separable.

1.2.2 Loss function

$$L(y, g(x, w)) = \begin{cases} 0, & \text{if } ysign(w^T x) \geq 0 \\ -yw^T x & \text{else} \end{cases}$$

The empirical error is

$$R_{emp}(y, X, w) = - \sum_{n=1; y^n a^n = -1} y^n w^T x^n$$

1.2.3 Learning

We use gradient descent to find the solution. First we need the derivative of R_{empg} with respect to w.

$$\nabla_w R_{emp}(y, X, W) = \nabla_w - \sum_{n=1; y^n a^n = -1} y^n w^T x^n = - \sum_{n=1; a^n y^n} = y^n x^n$$

if the data is linearly separable, then this algorithm converges, but there is no unique solution. The solution depends on the initial weights w.

insert convergence theorem

1.3 Logistic regression - sigmoid

The logistic regression is also used for binary classification tasks. The classes are $y \in \{0, 1\}$ To overcome the shortcomings of the perceptron we use a non-linear activation function $\sigma(z) = \frac{1}{1+e^z}$, so it is also possible to classify data that is not linear separable.

1.3.1 Model

We get the following model

$$g(x; w) = a = \sigma(w^T x)$$

The derivative is $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

1.3.2 Loss function

The sigmoid function takes values in $(0, 1)$ and can be interpreted as a probability. So to solve this we use maximum likelihood again. (Bernoulli distribution)

$$p(y = 1 | x, w) = g(x; w)$$

$$p(y = -1 | x, w) = (1 - g(x; w))$$

We can write this in a more compact form

$$p(y | x, w) = g(x; w)^y (1 - g(x; w))^{1-y}$$

Therefore we get the following likelihood

$$p(y | X, w) = \prod_{n=1}^{N} g(x^n; w) y^n (1 - g(x^n; w)^{1-y^n}$$

Because we have a product of numbers between 0 and 1 the product can get very small and this can get numerically unstable. To find a solution we again take the negative log likelihood. The negative log likelihood is

$$-\log p(y|X,w) = -\sum_{n=1}^{N} y^n \log g(x^n;w) + (1-y^n)\log(1-g(x^n;w)$$

where $L(y, g(x;w)) = -(y \log g(x;w) + (1-y)\log(1-g(x;w))$ is called the *binary cross-entropy.*

$$\nabla_w - log(y|X,w) = \nabla_w R_{emp}(y,X,w) = \sum_{n=1}^{N}(\sigma(w^T x^n) - y^n)x^n$$

So the gradient is a weighted sum of the samples.

1.3.3 Learning

For learning we have two options:

- gradient descent with $\nabla_w R_{emp}(y, X, W)$
- use the second derivative (Hessian). Newton algorithm to find a zero.

AMS example - Problem with biased dataset describe algorithm for Hessian optimization intermezzo

1.4 Softmax regression

Is used for multi categories classification. It is a supervised classification task. Examples are image recognition or predicting the next word in a sequence of words (here the classes are all possible words). The classes are usually one-hot encoded.

1.4.1 Model

We use the softmax function:

$$g(x, W) = a = softmax(Wx)$$

- we have a weight matrix instead of a weight vector, because multiple outputs
- softmax(s) = $\left(\frac{e^{s_1}}{\sum_{j=1}^{K} e^{e_j}}, \cdots, \frac{e^{s_K}}{\sum_{j=1}^{K} e^{e_j}}\right)^T$
- the result is a probability vector that sums up to 1.

exam: calculate softmax vector from W and x

1.4.2 Loss function and error

- Likelihood of the label vector: (Multinomial distribution)

$$p(y|a) = \prod_{k=1}^{K} a_k^{y_k}$$

- Categorical cross-entropy loss:

$$-\log p(y|a) = -\sum_{k=1}^{K} y_k \log(a_k)$$

- Empirical error

$$R_{emp}(W, X, Y) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \log(a_{nk}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} (y^n)_k \log((a^n)_k)$$

1.4.3 Learning

ganze Herleitung + Erklärung

$$\frac{\partial}{\partial w_{\xi d}}\Big(-\sum_{k=1}^{K} y_k \log(a_k)\Big) = (a_\xi - y_\xi)x_d$$

The derivative w.r.t the weight matrix is

$$\nabla_W R_{emp}(W, x, y) = (a - y)x^T$$

This is a convex problem, so gradient descent converge to a unique solution. insert comparision linear regression, logistic regression, softmax regression and summary

1.5 Single-layer can't solve XOR

überarbeiten

- 4 points $x_1 = (0, 0)$, $x_2 = (1, 0)$, $x_3 = (0, 1)$, $x_4 = (1, 1)$
- classify XOR, x_1, and x_4 are a class
- $g_2(x; W, w, b) = w^T \max(0, Wx + b)$ solves this. To get this we need to add an additional layer

1.6 Multi-layer perceptron