5 - Deep Learning Overview

5.2 - Training Rules

Marciel Barros

Setembro, 2020

1. Deep Learning Summary
2. **Training Rules**
3. Activation Function
4. Normalizer
5. Optimizer
6. Types of Neural Networks
7. Common Problems

Training in Neural Networks

- Once a network has been structured for a particular application, that network is ready to be **trained**. To start this process the **initial weights are chosen randomly**. Then, the training, or learning, begins.
- Training is the process of update weights according to a cost function.
- The **Gradient Descent** is one of major approaches for neural network training;

Gradient Descent and Loss Function

- The gradient of the multivariate function $o = f(x) = f(x_0, x_1, \dots, x_n)$ at $X' = [x_0', x_1', \dots, x_n']^T$ is shown as follows:

$$\nabla f(x_0', x_1', \dots, x_n') = [\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n}]^T |_{X=X'},$$

  The direction of the gradient vector is the fastest growing direction of the function. As a result, the direction of the negative gradient vector $-\nabla f$ is the fastest descent direction of the function.

- During the training of the deep learning network, target classification errors must be parameterized. **A loss function (error function)** is used, which reflects the error between the target output and actual output of the perceptron. For a single training sample x, the most common error function is the **Quadratic cost function**.

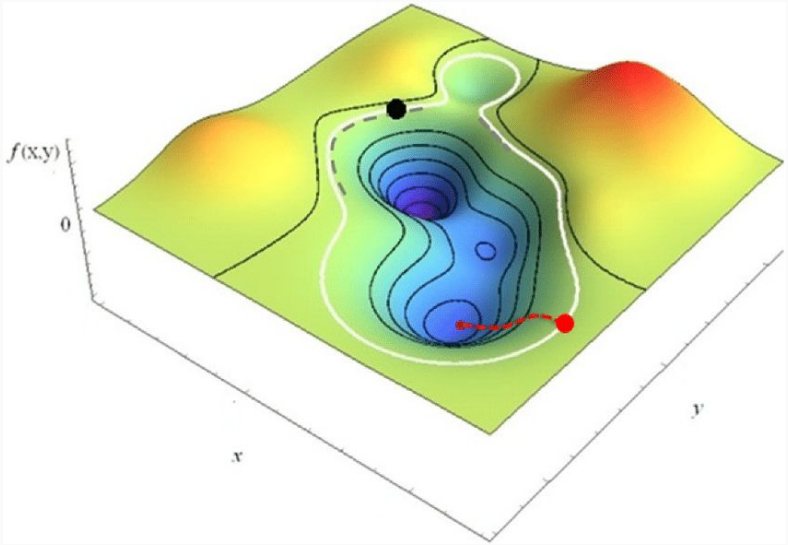
$$E(w) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2,$$

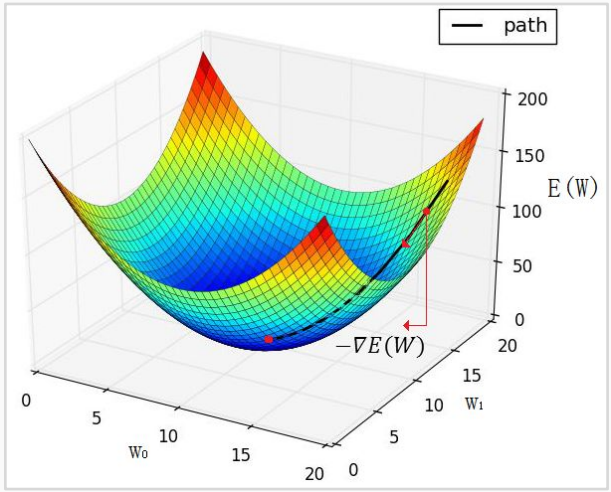
  In the preceding function, d is one neuron in the output layer, D is all the neurons in the output layer, t_d is the target output, and o_d is the actual output.

- The gradient descent method enables the loss function to search along the negative gradient direction and update the parameters iteratively, finally minimizing the loss function.

Extrema of the Loss Function

- **Purpose: The loss function $E(W)$ is defined on the weight space. The objective is to search for the weight vector W that can minimize $E(W)$.**
- **Limitation:** No effective method can solve the extremum in mathematics on the complex high-dimensional surface of $E(W) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$.





Example of gradient descent of binary paraboloid

Common Loss Functions in Deep Learning

- Quadratic cost function:

$$E(W) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

- Cross entropy error function:

$$E(W) = -\frac{1}{n}\sum_{x}\sum_{d \in D}[t_d \ln o_d + (1 - t_d)\ln(1 - o_d)]$$

- The cross entropy error function depicts the distance between two probability distributions, which is a widely used loss function for classification problems.
- Generally, the mean square error function is used to solve the regression problem, while the cross entropy error function is used to solve the classification problem.

Batch Gradient Descent Algorithm (BGD)

- In the training sample set D, each sample is recorded as < X,t >, in which X is the input vector, t the target output, o the actual output, and η the learning rate.
- Initializes each wi to a random value with a smaller absolute value.
- Before the end condition is met: Initializes each Δwi to zero.
- For each < X,t > in D:
    - Input X to this unit and calculate the output o.
    - For each wi in this unit: $\Delta w_i \mathrel{+}= -\eta \frac{1}{n} \sum_x \sum_{d \in D} \frac{\partial C(t_d, o_d)}{\partial w_i}$.
    - For each wi in this unit: $wi \mathrel{+}= \Delta wi$.
- The gradient descent algorithm of this version is not commonly used because: The convergence process is very slow as all training samples need to be calculated every time the weight is updated.

Stochastic Gradient Descent Algorithm (SGD)

- To address the BGD algorithm defect, a common variant called Incremental Gradient Descent algorithm is used, which is also called the Stochastic Gradient Descent (SGD) algorithm. One implementation is called Online Learning, which updates the gradient based on each sample:

$$\Delta w_i = -\eta \frac{1}{n} \sum_x \sum_{d \in D} \frac{\partial C(t_d, o_d)}{\partial w_i} \Rightarrow \Delta w_i = -\eta \sum_{d \in D} \frac{\partial C(t_d, o_d)}{\partial w_i}.$$

ONLINE-GRADIENT-DESCENT

- Initializes each wi to a random value with a smaller absolute value.
- Before the end condition is met, generates a random from D and does the following calculation:
  - Input X to this unit and calculate the output o.
  - For each wi in this unit: $w_i \mathrel{+}= -\eta \sum_{d \in D} \frac{\partial C(t_d, o_d)}{\partial w_i}.$

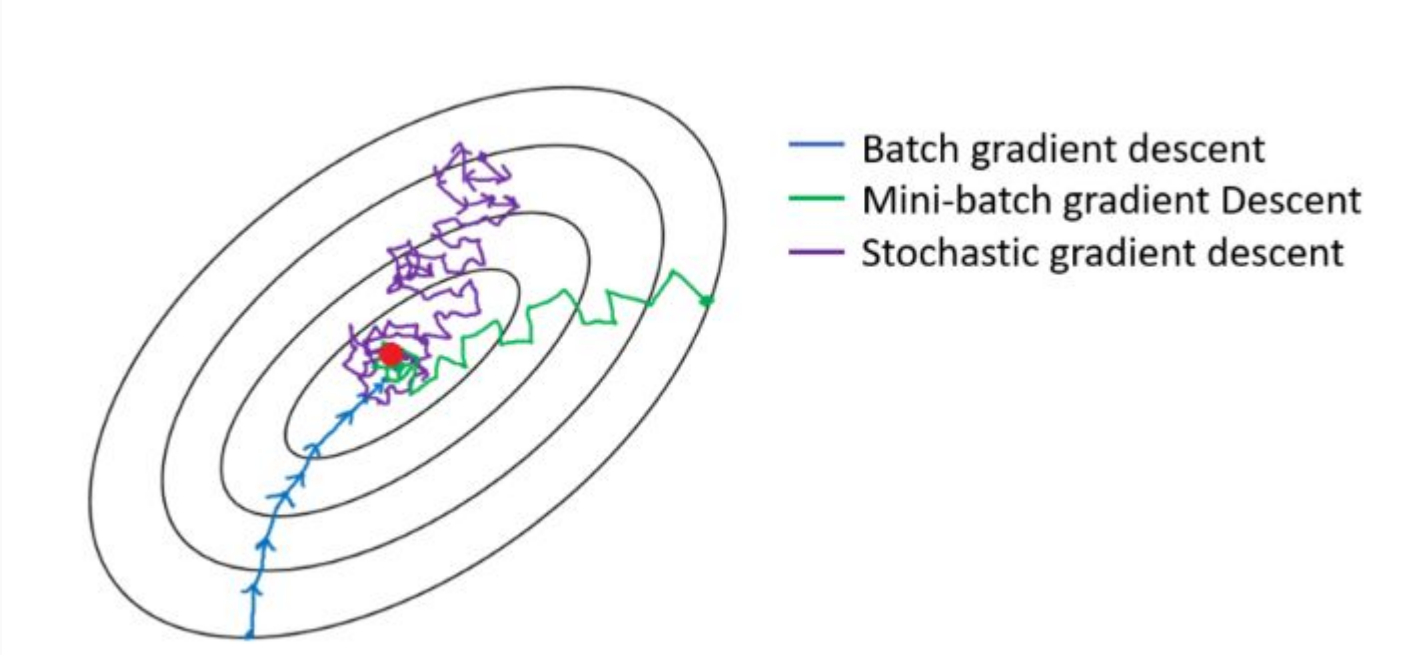
Mini-Batch Gradient Descent Algorithm (MBGD)

To address the defects of the previous two gradient descent algorithms, the Mini-batch Gradient Descent Algorithm (MBGD) was proposed and has been most widely used. A small number of Batch Size (BS) samples are used at a time to calculate Δwi , and then the weight is updated accordingly.

Batch-gradient-descent Initializes each wi to a random value with a smaller absolute value. Before the end condition is met:

- Initializes each Δwi to zero.
- For each < X,t > in the BS samples in the next batch in D:
  - Input X to this unit and calculate the output o.
  - For each wi in this unit: $\Delta w_i \mathrel{+}= -\eta \frac{1}{n} \sum_x \sum_{d \in D} \frac{\partial C(t_d, o_d)}{\partial w_i}$
  - For each wi in this unit: $wi \mathrel{+}= \Delta wi$
- For the last batch, the training samples are mixed up in a random order.

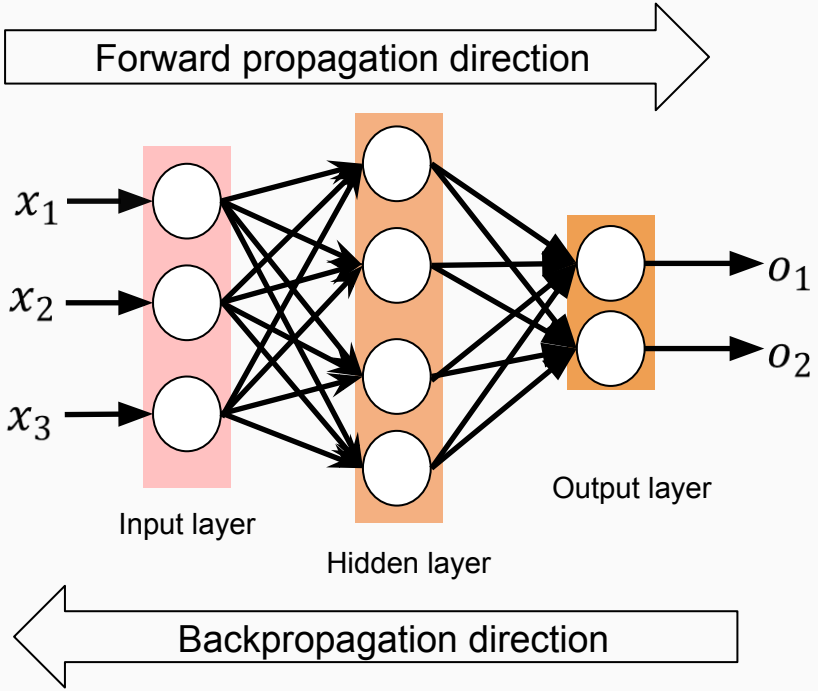
Gradient Descent Methods Comparison



https://medium.com/@xzz201920/gradient-descent-stochastic-vs-mini-batch-vs-batch-vs-adagrad-vs-rmsprop-vs-adam-3aa652318b0d

Backpropagation Algorithm (1)

- Signals are propagated in forward direction, and errors are propagated in backward direction.
- In the training sample set *D*, each sample is recorded as <*X*, *t*>, in which *X* is the input vector, *t* the target output, *o* the actual output, and *w* the weight coefficient.
- Loss function: $E(w)=\frac{1}{2}\sum_{(d\in D)}(t_d-o_d)^2$



Backpropagation Algorithm (2)

- According to the following formulas, errors in the input, hidden, and output layers are accumulated to generate the error in the loss function.
- *wc* is the weight coefficient between the hidden layer and the output layer, while *wb* is the weight coefficient between the input layer and the hidden layer. f is the activation function, D is the output layer set, and C and B are the hidden layer set and input layer set respectively. Assume that the loss function is a quadratic cost function:

- Output layer error: $$E = \frac{1}{2}\sum_{(d \in D)} (t_d - o_d)^2$$
- Expanded hidden layer error: $$E = \frac{1}{2}\sum_{(d \in D)} \left[t_d - f(net_d)\right]^2 = \frac{1}{2}\sum_{(d \in D)} \left[t_d - f(\sum_{(c \in C)} w_c y_c)\right]^2$$
- Expanded input layer error: $$E = \frac{1}{2}\sum_{(d \in D)} \left[t_d - f\left(\sum_{(c \in C)} w_c f(net_c)\right)\right]^2 =$$ $$\frac{1}{2}\sum_{(d \in D)} \left[t_d - f\left(\sum_{(c \in C)} w_c f\left(\sum_{b \in B} w_b x_b\right)\right)\right]^2$$

Backpropagation Algorithm (3)

- To minimize error E, the gradient descent iterative calculation can be used to solve w_c and w_b, that is, calculating w_c and w_b to minimize error E.
- Formula:

$$\Delta w_c = -\eta \frac{\partial E}{\partial w_c}, c \in C$$

$$\Delta w_b = -\eta \frac{\partial E}{\partial w_b}, b \in B$$

- If there are multiple hidden layers, chain rules are used to take a derivative for each layer to obtain the optimized parameters by iteration.

Backpropagation Algorithm (4)

- For a neural network with any number of layers, the arranged formula for training is as follows:

$$\Delta w_{jk}^{l} = -\eta \delta_{k}^{l+1} f_{j}(z_{j}^{l})$$

$$\delta_{j}^{l} = \begin{cases} f_{j}^{'}(z_{j}^{l})(t_{j} - f_{j}(z_{j}^{l})), l \in outputs, (1) \\ \sum_{k} \delta_{k}^{l+1} w_{jk}^{l} f_{j}^{'}(z_{j}^{l}), otherwise, (2) \end{cases}$$

- The BP algorithm is used to train the network as follows:
  - Takes out the next training sample <*X*, *T*>, inputs *X* to the network, and obtains the actual output *o*.
  - Calculates output layer δ according to the output layer error formula (1).
  - Calculates δ of each hidden layer from output to input by iteration according to the hidden layer error propagation formula (2).
  - According to the δ of each layer, the weight values of all the layer are updated

5 - Deep Learning Overview

Next: 5.3 - Activation Function

Marciel Barros

Setembro, 2020