Lecture 3

Learning the whole dataset

So far we train neural networks to model one (single) training example (a data set)

$$(input_0, goal_0).$$

First consider the simple network with one input and one output

In fact the learning process is reduced to memorizing data:

the weight (W_0) is selected to fit the prediction $pred_0$ to the given $goal_0$.

$$weight_0 = goal_0 / input_0,$$

$$\Rightarrow pred_0 := input_0 * weight_0 = goal_0$$

Next time when the network gets input := $input_0$ it returns the correct predict value $goal_0$. Still if this network see new input value inp, its prediction can be not accurate at all.

A real training (learning) is based on the larger (whole) dataset (info we have from experiments)

$input = [input_0, input_1, \dots, input_n]$

$goal = [goal_0, goal_1, \dots, goal_n]$

The weight is obtained by solving the error equation (the loss function)

$$w_0 = \arg\min \text{error}(w) \qquad (*)$$

where

$$\text{error}(w) = \frac{1}{2} \sum_{j=0}^{M-1} (inp_j \times w - goal_j)^2.$$

The optimal weight w_0 is a solution of the equation

$$\frac{\partial \text{error}(w)}{\partial w} = 0$$

i.e.

$$\sum_{j=0}^{M} (inp_j \times w - goal_j) \times inp_j = 0$$

$$\Rightarrow \quad w_0 = \left(\sum_{j=0}^{M} goal_j \times inp_j\right) \Big/ \sum_{j=0}^{M} (inp_j)^2 \qquad (2)$$

In Machine Learning (ML)
an equivalent and more general
relations are defined

$$pred_j = a + W \times inp_j \quad (a\text{- bias})$$

This relation is called
a linear regression model.

In the deep learning method
more general networks are used.
Then instead formula (2) the
optimal value of the weight
is determined by using the
gradient descent algorithm
(and efficient modifications
of it).

Multiple inputs

Let the network has N inputs,

we denote the j-th data set

$$input^j = [inp_0^j, inp_1^j, \cdots, inp_{N-1}^j]$$

$j = 0, 1, \cdots, M-1,$

$$goal = [goal^0, goal^1, \cdots, goal^{M-1}]$$

$$weight = [W_0, W_1, \cdots, W_{N-1}]$$

$$predict = [pred^0, pred^1, \cdots, pred^{M-1}]$$

The $pred^j$ is calculated as

$$pred^j = \sum_{n=0}^{N-1} inp_n^j * W_n.$$

The loss function (error)

$$error = \frac{1}{2} \sum_{j=0}^{M-1} (pred^j - goal^j) ** 2.$$

$$delta^j = pred^j - goal^j$$

If bias coefficient is included into prediction formula

$$pred^j = \sum_{n=0}^{N-1} inp_n^j * W_n + \alpha ,$$

then the loss function depends also on one more parameter

$$parameters = [weight, \alpha]$$

In gradient descent step we use also the derivative with respect to α ($\nabla error$ depends also on $\frac{\partial error}{\partial \alpha}$)

$$\frac{\partial error}{\partial \alpha} = \sum_{j=0}^{M-1} (pred^j - goal^j) \times \frac{\partial pred^j}{\partial \alpha}$$

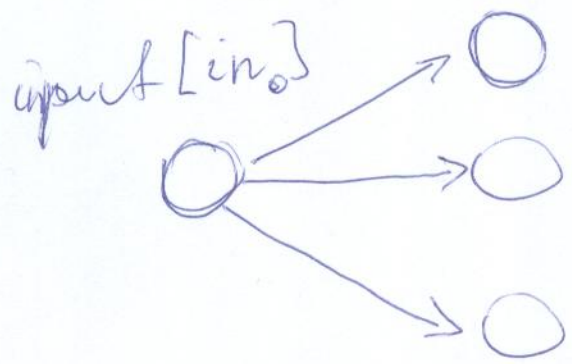
$$= \sum_{j=0}^{M-1} (pred^j - goal^j)$$

$$= \sum_{j=0}^{M-1} delta^j .$$

Multiple outputs

For simplicity we start with a network fore

- one single input data
- multiple data for outputs



Predictions are calculated as

$$\boxed{pred_j = w_j * in_0} \quad (1)$$

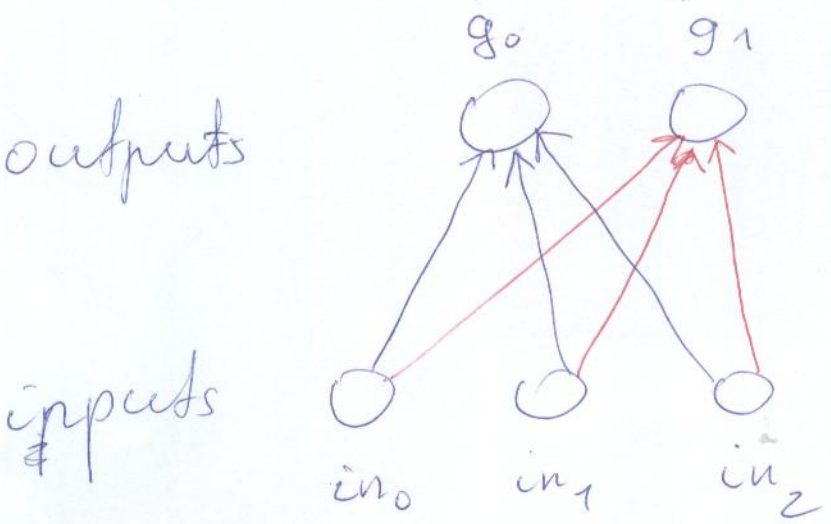
$$delta_j = pred_j - g_j$$

We make two main notes:

- Three networks are independent
- Gradient descent method can be applied in parallel for each network.

Generalization

1. The ANN can be considered for multiple inputs and multiple outputs



goal [$goal_0$, $goal_1$]

weights [
[W_{00}, W_{01}, W_{02}],
[W_{10}, W_{11}, W_{12}],
~~[W_{20}, W_{21}, W_{22}]~~]

$$pred_k = \sum_{n=0}^{N-1} W_{kn} * in_n$$

inputs

$k = 0, 1$ - outputs

A real training is based on the larger datasets.

- Let us assume that we have one set of inputs (dataset)

$$input = [inp_0, inp_1, \cdots, inp_{N-1}]$$

N = # of inputs

- We want to predict K outputs.

The training is done on a dataset of size M = # size of the dataset

$$goals = [goal^0, goal^1, \cdots, goal^{M-1}]$$

$$goal^m = [g_0^m, g_1^m, \cdots, g_{K-1}^m]$$

$$m = 0, \cdots, M-1.$$

Predictions

$$predictions = [predict_0, \cdots, predict_{K-1}]$$

$$predict_k = \sum_{n=0}^{N-1} W_{k,n}\, inp_n, \quad k = 0, \cdots, K-1.$$

Our task is to make a training and to find weights of ANN

K = # outputs

$$W = [W_0, \dots, W_{K-1}]$$

N = # inputs

$$W_k = [w_{k0}, w_{k1}, \dots, w_{k,N-1}]$$

We have K independed ANN for each output and the training of each ANN can be done in parallel.

The loss function $error_k(W_k)$:

$$error_k(W_k) = \frac{1}{2} \sum_{m=0}^{M-1} \left(\sum_{n=0}^{N-1} inp_n^m \times W_{kn} - goal_k^m \right)^2$$

The optimal weights W_k are calculated by using gradient descent algorithm.

If inputs are also different for each m

$$error_k(W_k) = \frac{1}{2} \sum_{m=0}^{M-1} \left(\sum_{n=0}^{N-1} inp_n^m \times W_{kn} - goal_k^m \right)^2$$

Full, batch and stochastic gradient descent

It was noted that the case of multiple outputs can be resolved as K independed trainings of separate ANN.

Thus we restrict to the case of multiple N inputs and one output

The size of the dataset is equal to M:

$$goals = [goal^0, goal^1, \dots, goal^{M-1}]$$

$$inputs = [input^0, input^1, \dots, input^{M-1}]$$

$$input^m = [inp_0^m, inp_1^m, \dots, inp_{N-1}^m]$$

$$m = 0, \dots, M-1.$$

We want to find weights

$$W = [W_0, W_1, \cdots, W_{N-1}].$$

They are used to compute predictions

$$pred^m = \sum_{n=0}^{N-1} inp_n^m * W_n.$$

The loss function error(W) is defined as

$$error(W) = \frac{1}{2} \sum_{m=0}^{M-1} (pred^m - goal^m)^2$$

We have investigated the

Full gradient descent version

An entire dataset is used at each iteration of iteration: the average weight_delta is calculated over the entire dataset and only then the weights are changed

Batch gradient descent

This modification of GDM that takes an approach laying in between of two previous modifications

Instead of updating the weights after just one example (as in stochastic gradient descent) or after the entire dataset (after all M examples of data) we choose a batch size (between 8 and 256) of examples, after which the weights are updated

Weight updates:

$$W^{s+1} = W^{s} - \alpha \Delta W^{s},$$

where weight-delta is calculated as

$$\Delta W^{s} = \sum_{m=0}^{M-1} \left(\sum_{n=0}^{N-1} inp_n^m * W_n^s - goal^m \right) inp_n^m$$

$$= \sum_{m=0}^{M-1} delta^m * inp^m$$

dot product

Stochastic gradient descent

It performs a prediction and weight update for each training example separately (for each $m = 0, \dots M-1$,