Introduction to Optimization

HW 3

Quasi-Newton BFGS method and DNN training

Part 1 – BFGS

Task 1

Implement in Matlab Quasi-Newton BFGS method.

$$x, m = BFGS(fun, x_0)$$

Where x_0 is the starting point.

fun is the target function to minimize, specified as a function handle or function name.

fun is a function that accepts a vector x and returns a real scalar, the objective function evaluated at x and the gradient of the objective function at this point.

BFGS attempts to find a local minimum of the function described in fun.

BFGS will return a vector m with the values of the target function $fun(x_k)$ at iteration number k and a solution, returned as a real vector. The size of x is the same as the size of x_0. x is a local solution to the problem.

In order to compute step size use Armijo inexact line search with the following parameters:

$$Initial\ step\ size: \alpha_0 = s = 1$$
$$\sigma = 0.25$$
$$\beta = 0.5$$
$$\varepsilon = 10^{-5}$$

Before updating the Hessian check that the directional derivative at current step length α is less negative than for $\alpha = 0$ (this ensures positive definiteness of the Hessian model after the update, see the lecture).

If this requirement is not satisfied, just skip the Hessian update and continue to the next step.

Use the Identity matrix (I) as your initial guess to the inverse of the Hessian matrix (B_0).

Use the following stopping criteria: $\|\nabla f\| < \varepsilon$

Task 2 – Rosenbrock function

Test your implementation with Rosenbrock function:

$$f(x) = f(x_1, x_2, ...x_N) = \sum_{i=1}^{N-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2 \right]$$

Use staring point $x_0 = (0, 0, .., 0)$

Plot the function's convergence curve i.e. plot $f(x_k) - p^*$ as a function of the iteration number

k where p^* is the function optimal value.

Use logarithmic scale for the y-axis, i.e. use semilogy for plotting this graph.

Part 2 – Deep Neural Network

In this part we will set the background for an example of the universal approximation theory of DNN.

The universal approximation theorem states that a feed-forward deep network (with one or more hidden layers) containing a finite number of neurons can approximate any continuous function over compact subsets of $\mathbb{R}^n$.

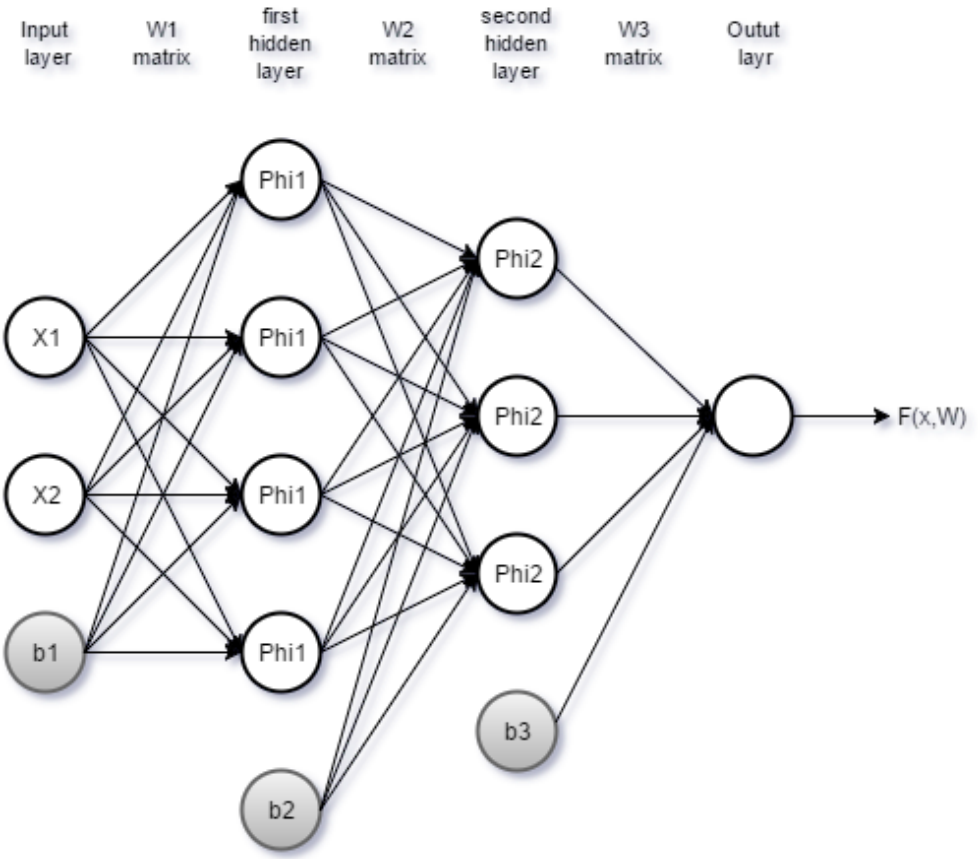
In this exercise the network task will be to approximate the following function:

$f(x_1, x_2) = x_1 \cdot exp(-x_1^2 - x_2^2)$

In order to approximate the above function we will use the following network architecture:

Two inputs.
Four neurons in the first hidden layer.
Three neurons in the second hidden layer.
One linear output.
The network structure is shown in the following diagram:

Introduction to Optimization
HW 3



As seen in the lectures, we can also represent the above feed forward neural network with the following algebraic equation:

$$F(x, W_1, W_2, W_3, b_1, b_2, b_3) = W_3^T \varphi_2(W_2^T \varphi_1(W_1^T x + b_1) + b_2) + b_3$$

With the following dimensions:

$$x \in \mathbb{R}^2$$
$$W_1 \in \mathbb{R}^{2\times 4}$$
$$W_2 \in \mathbb{R}^{4\times 3}$$
$$W_3 \in \mathbb{R}^{3\times 1}$$
$$b_1 \in \mathbb{R}^4$$
$$b_2 \in \mathbb{R}^3$$
$$b_3 \in \mathbb{R}$$

Introduction to Optimization
HW 3

φ_1 and φ_2 are multivariate vector nonlinear functions as seen in lectures, i.e. $\varphi\begin{pmatrix} u_1 \\ u_2 \\ \dots \\ u_n \end{pmatrix} \triangleq \begin{pmatrix} \varphi(u_1) \\ \varphi(u_2) \\ \dots \\ \varphi(u_n) \end{pmatrix}$

(For simplicity we will use the general notation W for all weights and bias terms and get

$$F(x,W) \triangleq F(x,W_1,W_2,W_3,b_1,b_2,b_3))$$

In this exercise we will choose the hyperbolic tangent as the nonlinearity of the hidden layers φ_1 and φ_2 (also referred to as an activation functions).

$$\varphi(x) = tanh(x) = \frac{1-exp(-2x)}{1+exp(-2x)} \text{ , i.e.}$$

$$\varphi_1\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_n \end{pmatrix} \triangleq \begin{pmatrix} \frac{1-exp(-2u_1)}{1+exp(-2u_1)} \\ \frac{1-exp(-2u_2)}{1+exp(-2u_2)} \\ \frac{1-exp(-2u_3)}{1+exp(-2u_3)} \\ \frac{1-exp(-2u_4)}{1+exp(-2u_4)} \end{pmatrix} \quad \varphi_2\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \triangleq \begin{pmatrix} \frac{1-exp(-2u_1)}{1+exp(-2u_1)} \\ \frac{1-exp(-2u_2)}{1+exp(-2u_2)} \\ \frac{1-exp(-2u_3)}{1+exp(-2u_3)} \end{pmatrix}$$

Task 3 – analytical computation and numerical check

For this task we will choose the penalty error function to be the square error function:

$$\psi(r) = r^2$$

Using the notations from the lecture, we will denote the models error on the i^{th} example:

$$r_i \triangleq F(x^i,W) - y_i$$

Where $x^i \triangleq (x_1^{\,i}, x_2^{\,i})$ is the i^{th} training example, $F(x^i,W)$ is the output of the neural network with x^i as input, and y_i is the desired output for corresponding input vector x_i i.e.

$$y_i = f(x_1^{\,i}, x_2^{\,i}) = x_1^{\,i} \cdot exp(-(x_1^{\,i})^2 - (x_2^{\,i})^2)$$

Introduction to Optimization
HW 3

1. Derive an analytical expression for the gradient of the error function $\psi(r_i)$ for a specific example x^i, with respect to each of the weight and bias parameters $(W_1, W_2, W_3, b_1, b_2, b_3)$ as shown in the video. i.e.
$$\nabla_{W_1}\psi, \nabla_{W_2}\psi, \nabla_{W_3}\psi, \nabla_{b_1}\psi, \nabla_{b_2}\psi, \nabla_{b_3}\psi$$
2. For each one of the above gradient vectors, test the correctness of the computed gradient via finite differences, as done in HW1.

Task 4 – training the DNN

We will use the expressions obtained in previous task to train a neural network and see an example of the universal approximation theory of DNN.

1. Generate the training set and test set
   - Plot in Matlab a two dimensional plot of the function $f(x_1, x_2) = x_1 \cdot exp(-x_1^2 - x_2^2)$ in the range: $x_1 \in [-2, 2]$ $x_2 \in [-2, 2]$, you can use the following Matlab code:

     ```
     [X1, X2] = meshgrid(-2:.2:2, -2:.2:2);
     Y = X1 .* exp(-X1.^2 - X2.^2);
     figure; surf(X1, X2, Y)
     ```

   - Generate a training set of 500 uniformly distributed random points in range $[-2, 2]$, you can use the following Matlab code:

     ```
     Ntrain=500;
     X_train= 4*rand(2,Ntrain)-2;
     ```

   - In a similar way generate 200 points for a test set. Name the resulting $\mathbb{R}^{2\times500}$ matrix X_test.

   - Evaluate corresponding function values to vectors Y_train and Y_test of dimensionality $\mathbb{R}^{1\times500}$ and $\mathbb{R}^{1\times200}$ correspondingly.

Introduction to Optimization
HW 3

Train the neural network

We would like to train the neural network (find the optimal weights and bias terms $W_1, W_2, W_3, b_1, b_2, b_3$) by minimizing the error over the generated training set using the BFGS method.

We will optimize the error over all examples. This is called "batch optimization" in contrast to stochastic optimization, which performs a step for small group of examples.
We will use the notation N_{train} to represent the number of elements in the training set (In our case, $N_{train} = 500$).

In order to minimize the error over all examples, we will use the average of the square error functions over each of the examples. Our target function will now be:

$$\psi_{train}(r_1, r_2, ..., r_{N_{train}}) = \frac{1}{N_{train}} \sum_{i \in Trainig\ set} r_i^2 =$$

$$\frac{1}{|Trainig\ set|} \sum_{i=1}^{N_{train}} (F(x^i, W) - y_i)^2$$

The test error can be defined in a similar way.

2. Minimize the training error ψ_{train} using the BFGS method.

The target function should receive as an argument all your optimization variables organized as a column stack of all the weights and bias terms $W_1, W_2, W_3, b_1, b_2, b_3$, and return the function value (the cost) and the gradients of each of the variables stacked in a similar way.

Use the following initialization (starting point of optimization):
Biases b_1, b_2, b_3 are initialized to zero.
Weight matrices of size $m \times n$: W = randn(m,n)/sqrt(n)

Use analytic expressions for evaluating the gradients during minimization.

3. Compute and plot the functions reconstruction for the test set using your trained network.