Dr. Frank Lenzen
Frank.Lenzen@iwr.uni-heidelberg.de

Project 1: Neural Networks II – full net

Presentation scheduled for 11.1.2016.

This project deals with the implementation of a neural net with basic functionality and testing its functionality based on simple examples.

In the following we consider a neural net implementing a function $f(x; W, b)$.

Let us first consider a layer of neurons. The functionality of one layer of neurons can be described by

$$x^k := \varphi(\sum_{i=1}^{n} W_{j,i}^k x_i^{k-1} + b_j),$$

mapping from $\mathbb{R}^{n_{k-1}}$ to $\mathbb{R}^{n_k}$. Here, superscript k is used as the layer index, i is the index for the input and j is the index for the output. Moreover, the activation function φ is applied separately to each element of

$$\tilde{x}^k := \sum_{i=1}^{n} W_{j,i}^k x_i^{k-1} + b_j.$$

When implementing a layer of neurons, we requite routines performing the following tasks:

- initialization: this should set all parameters $W_{j,i}$ to random values and b_j to zero.
- different activation functions: consider a sigmoid and a ReLU as activation functions. The implementation should be in a way that it can be easily switched between the two cases.
- forward propagation: this routines calculates x^k for a given x^{k-1}, using the currently stored parameters W^k and b^k.
- forward propagation with storing the intermediate values: this is similar as before, except that we store x^{k-1} and $\tilde{x}^k := \sum_{i=1}^n W_{j,i}^k x_i^{k-1}$.
- back-propagation: this routine should take a value v (later: the error $f(\hat{x}; W, b) - \hat{y}$), calculate

  $$dW^k = (\varphi'(\tilde{x}^k) \odot v)(x^{k-1})^\top$$
  $$db^k = \varphi'(\tilde{x}^k) \odot v$$

  store these variables and return the value $(W^k)^\top(\varphi'(\tilde{x}^k) \odot v)$.

Dr. Frank Lenzen
Frank.Lenzen@iwr.uni-heidelberg.de

- parameter update:
$$\begin{aligned} W &\leftarrow W^k - \tau dW^k \\ b &\leftarrow b^k - \tau db^k \end{aligned}$$
  with some provided parameter τ.

- parameter update undo: this should just undo the parameter update (in case that it didn't lead to a minimization of the loss function)

- parameter update for regularization: There, we consider the effect, a regularization of parameter has on the update of the parameters. Depending on if we perform either an L2 or an L1 regularization, one has to perform an update
$$dW \leftarrow dW + \lambda W, \qquad dW^k_{i,j} \leftarrow dW^k_{i,j} + \lambda \frac{1}{\sqrt{(W^k_{i,j})^2 + \varepsilon}}.$$
  Here, we explicitly excluded the parameters b^k, for which regularization is not needed.)

Full network:
To set up a full network of K layers, we have to specify the sizes n_k of each layer and to initialize the layers accordingly.
For such a net, we require the above routines to be applied layerwise (e.g. initialization, updates) or in a recursive way (e.g. forward propagation,back-propagation). For the forward propagation, we require a loop $k = 1, \dots, K$, in which x^k is calculated from x^{k-1}, x^0 being the input data. For the back-propagation we require a loop $k = K, \dots, 1$ calling the back-propagation routine of layer k with input v^k and retrieving a value v^{k-1}, where v^K is the error $\frac{dE}{dy}$ for an arbitrary loss function $E(y, \hat{y})$ where $y = f(\hat{x}, W, b)$ is the output of the net and $\hat{y}$ is from training the training sample.

Learning:
For learning, we focus on a stochastic gradient approach, which is as follows:

Dr. Frank Lenzen
Frank.Lenzen@iwr.uni-heidelberg.de

Input: $N \geq 1$, training samples $(\hat{x}^s, \hat{y}^s)$, $\lambda \geq 0$, $\mu \in (0,1)$
Output: W, b `// stored in the networks stucture`
Initialize Neural Network
Set τ to small value.
begin
  for $i = 1, \ldots, N$ **do**
    store current value $e_1 := E(W,b)$ of loss function.
    backup current W, b
    for $s = 1, \ldots, S$ **do**
      forward propagation with input $\hat{x}^s$, storing $\tilde{x}^k, x^k$
      calculate error v `// depends on loss function`
      back-propagate error v in to network
      update dW, db according to regularization with parameter λ
      perform a parameter update on W, B with rate τ
    end
    calculate loss $e_2 := E(W,b)$
    if $e2 \geq e1$ **then**
      restore old parameters from backup
      $\tau \leftarrow \mu\tau$
    end
  end
end

This is a stochastic descent approach, which is somehow between a stochastic gradient approach with fixed τ and a full line search. This versions seems to work well in practice. However, please feel free to try modifications.

Error for the loss function
Considering the quadratic loss $\frac{1}{2}\|f(x;W,b) - y\|^2$, the error v is given by $v := f(x;W,b) - y$. In case of a softmax loss function the error is

$$v_k = \frac{dE}{dy_k} = p_k - \delta_0(k - l)$$

where y_k is the $k - th$ component of $y = f(x;W,b)$ and $p_k = \frac{e^{y_k}}{\sum_j e^{y_j}}$.

Remarks on the implementation:
In your implementation, you might either choose an object-oriented approach by implementing a class for layer of neurons or in a function-oriented way. In the latter case it might be useful to at least store all relevant information of a neuronal layer in a struct.

Dr. Frank Lenzen
Frank.Lenzen@iwr.uni-heidelberg.de

The tasks in detail:

1. Implement the routines of a layer and a net of neurons described above.
2. Implement the stochastic gradient descent for the learning.
3. Test your routine first with fixed parameters W^k, b^k. Check also the correct calculation of the gradients.
4. Test your algorithm by approximating a given function $g : \mathbb{R} \to \mathbb{R}$, e.g. $g(x) = \cos(\alpha x)$. Generate training samples of g and learn a neural network of a few layers.
5. Test your implementation with the training sets provided as supplemental material. Use different kinds of loss functions. Your might also think of generating training samples on your own.