

# Regression

## 6. Regression with Neural Networks

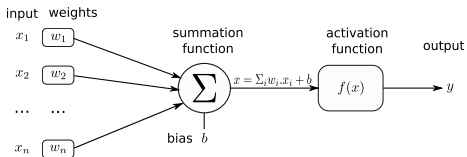
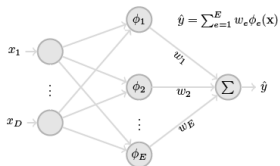
Olivier Sigaud

Sorbonne Université  
<http://people.isir.upmc.fr/sigaud>



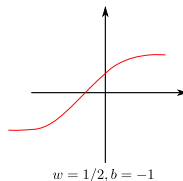
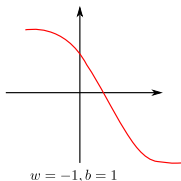
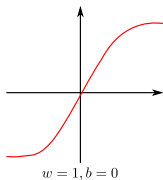
# Regression with Neural Networks

## The case of (feedforward) neural networks



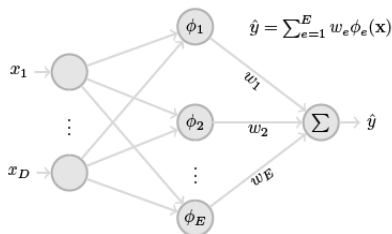
- ▶ The activation function is non local (sigmoid, ReLu, LeakyReLu, ...) vs Gaussians
- ▶ Weights of output layer: regression
- ▶ Weight of intermediate layer(s): tuning basis functions
- ▶ Shares the same structure as all basis function networks
- ▶ Sigmoids instead of Gaussians: better split of space in high dimensions

## Regression with neural networks: discovering features



- ▶ The backprop algo tunes both kinds of weights
- ▶ Discovers the adequate features by itself
- ▶ Deep versus shallow: get more tunable features with less parameters

## Regression with neural networks: variants

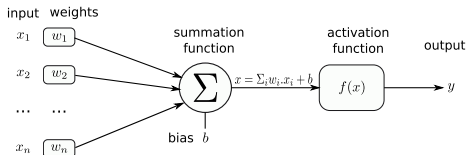
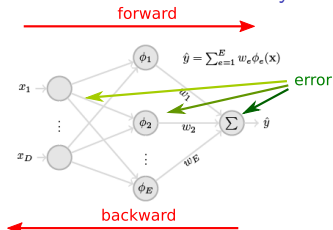


- ▶ If only the weights at the last layer are tuned, still defines a linear architecture (Extreme Learning Machine)
- ▶ Stochastic optimization of intermediate weights, linear regression on output weights?



Huang, G.-B., Zhou, H., Ding, X., & Zhang, R. (2012) Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529

## Gradient descent over multilayer neural networks



- ▶ The parameters  $\theta$  are the weights  $w_{ij}$  and biases  $b_i$
- ▶ In  $\nabla_{\theta}(\mathbf{L}(\theta))$ , the errors at a layer depends on the errors at the next layers
- ▶ Gradient computation cannot be performed in a single step
- ▶ Compute the gradient with the gradient backpropagation algorithm (backprop)
- ▶ It is technical and tedious to code for each specific network structure
- ▶ TensorFlow, pytorch and their ancestors (theano, caffe...) are built to provide gradient backpropagation for any tensor structure



Nielsen, M. A. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, 2015

<http://neuralnetworksanddeeplearning.com/chap2.html>

## Creating an neural network in pytorch

```
class NeuralNetwork(nn.Module):  
  
    def __init__(self, l1, l2, l3, l4, out, learning_rate):  
        super(NeuralNetwork, self).__init__()  
        self.relu = nn.ReLU()  
        self.sigmoid = nn.Sigmoid()  
        self.fc1 = nn.Linear(l1, l2)  
        self.fc2 = nn.Linear(l2, l3)  
        self.fc3 = nn.Linear(l3, l4)  
        self.fc4 = nn.Linear(l4, out)  
        self.optimizer = th.optim.Adam(self.parameters(), lr=learning_rate)  
  
    def f(self, x):  
        input = th.from_numpy(x).float()  
        hidden1 = self.sigmoid(self.fc1(input))  
        hidden2 = self.sigmoid(self.fc2(hidden1))  
        hidden3 = self.sigmoid(self.fc3(hidden2))  
        output = rescale(self.sigmoid(self.fc4(hidden3)))  
        return output
```

- ▶ Adam does better than SGD
- ▶ `f()` is often called `forward()`
- ▶ Other functions not shown

## Gradient descent in pytorch

- ▶ Computing the loss over a batch

```
for i in range(max_iter):  
    output = model.f(xt)  
    loss = func.mse_loss(output, yt)
```

- ▶ Applying gradient descent

```
def update(self, loss) -> None:  
    """  
    Apply a loss to a network using gradient backpropagation  
    :param loss: the applied loss  
    :return: nothing  
    """  
  
    self.optimizer.zero_grad()  
    loss.sum().backward()  
    self.optimizer.step()
```

- ▶ The backprop in one line!



Huang, G.-B., Zhou, H., Ding, X., and Zhang, R. (2012).  
Extreme learning machine for regression and multiclass classification.  
*IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529.



Nielsen, M. A. (2015).  
*Neural networks and deep learning*, volume 25.  
Determination press San Francisco, CA.