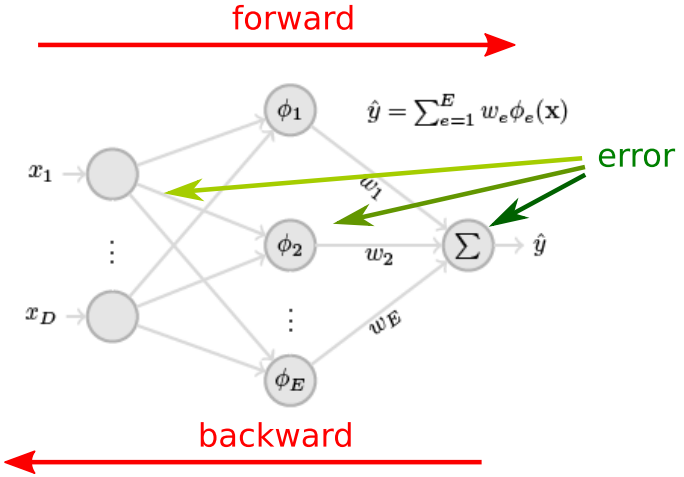
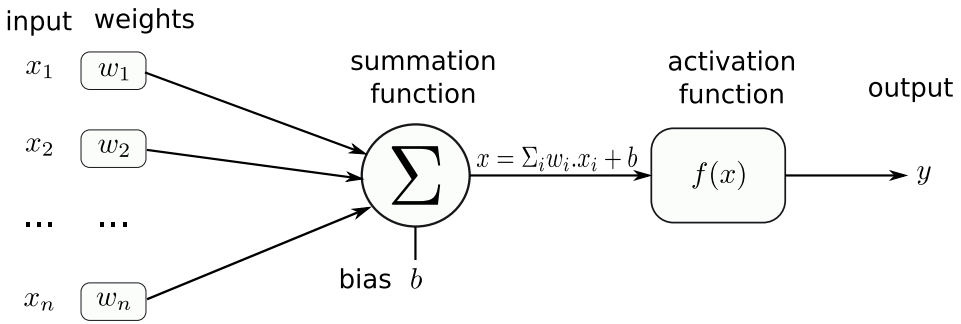
Regression in Neural Networks

Olivier Sigaud

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

Gradient descent over multilayer neural networks





- The parameters $\boldsymbol{\theta}$ are the weights w_{ij} and biases b_i
- In $\nabla_{\boldsymbol{\theta}}(\mathbf{L}(\boldsymbol{\theta}))$, 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 a little 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)
  model.update(loss)
```

- 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!

Any question?

Send mail to: `Olivier.Sigaud@upmc.fr`