

Cost Function and Backpropagation

Video: Cost Function
6 min

Reading: Cost Function
4 min

Video: Backpropagation Algorithm
11 min

Reading: Backpropagation Algorithm
10 min

Video: Backpropagation Intuition
12 min

Reading: Backpropagation Intuition
4 min

Backpropagation in Practice

Video: Implementation
Note: Unrolling Parameters
7 min

Reading: Implementation
Note: Unrolling Parameters
3 min

Video: Gradient Checking
11 min

Reading: Gradient Checking
3 min

Video: Random Initialization
6 min

Reading: Random Initialization
3 min

Video: Putting It Together
13 min

Reading: Putting It Together
4 min

Application of Neural Networks

Review

Gradient Checking

Gradient checking will assure that our backpropagation works as intended. We can approximate the derivative of our cost function with:

∂J(Θ)/∂Θ ≈ (J(Θ + ε) - J(Θ - ε)) / (2ε)

With multiple theta matrices, we can approximate the derivative **with respect to Θ_j** as follows:

∂J(Θ)/∂Θ_j ≈ (J(Θ₁, ..., Θ_j + ε, ..., Θ_n) - J(Θ₁, ..., Θ_j - ε, ..., Θ_n)) / (2ε)

A small value for ε (epsilon) such as ε = 10⁻⁴, guarantees that the math works out properly. If the value for ε is too small, we can end up with numerical problems.

Hence, we are only adding or subtracting epsilon to the Θ_j matrix. In octave we can do it as follows:

```
1 epsilon = 1e-4;
2 for i = 1:n,
3     thetaPlus = theta;
4     thetaPlus(i) += epsilon;
5     thetaMinus = theta;
6     thetaMinus(i) -= epsilon;
7     gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)
8 end;
9
```

We previously saw how to calculate the deltaVector. So once we compute our gradApprox vector, we can check that gradApprox = deltaVector.

Once you have verified **once** that your backpropagation algorithm is correct, you don't need to compute gradApprox again. The code to compute gradApprox can be very slow.

Mark as completed