

## Cost Function and Backpropagation

### Backpropagation in Practice

- Implementation Note: Unrolling Parameters 7 min
- Implementation Note: Unrolling Parameters 3 min
- Gradient Checking 11 min
- Gradient Checking 3 min
- Random Initialization 6 min
- Random Initialization 3 min
- Putting It Together 13 min
- 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:

$$\frac{\partial}{\partial \Theta} J(\Theta) \approx \frac{J(\Theta + \epsilon) - J(\Theta - \epsilon)}{2\epsilon}$$

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

$$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \dots, \Theta_j + \epsilon, \dots, \Theta_n) - J(\Theta_1, \dots, \Theta_j - \epsilon, \dots, \Theta_n)}{2\epsilon}$$

A small value for  $\epsilon$  (epsilon) such as  $\epsilon = 10^{-4}$ , guarantees that the math works out properly. If the value for  $\epsilon$  is too small, we can end up with numerical problems.

Hence, we are only adding or subtracting epsilon to the  $\Theta_j$  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  $\approx$  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.

✓ Completado

