1. COST FUNCTION AND BACKPROPAGATION.

1.1. Cost Functn:

NN (Classificatn)

L = Total # of layers in network.

s_l = # of units in layer l (excluding bias unit)

Binary classificatn: 1 output unit. $h_\theta(x) \in \mathbb{R}$

Multi-Class classfctn: K output units. $h_\theta(x) \in \mathbb{R}^K$

Cost Func:

Logistic Regression: $J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

NN:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-h_\theta(x^{(i)}))_k\right] - \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\theta_{ji}^{(l)})^2$$

1.2. Backpropagatn Algorithm:

$\min_\theta J(\theta)$

Need code to compute: $\rightarrow J(\theta)$

$\rightarrow -\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$

Gradient Computatn:

$(x,y) \leftarrow$ training example

$a^{(1)} = x$

$z^{(2)} = \theta^{(1)} . a^{(1)}$

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

$z^{(3)} = \theta^{(2)} . a^{(2)}$

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \theta^{(3)} a^{(3)}$

$a^{(4)} = g(z^{(4)}) = h_\theta(x)$ (add $a_0^{(4)}$)

Backpropagatn Algorithm:

Intuitn: $\delta_j^{(l)}$ = "error" of node j in layer l.

For each output unit (layer $L=4$)

$\delta_j^{(4)} = a_j^{(4)} - y_j \longrightarrow \delta^{(4)} = a^{(4)} - y$

$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \rightarrow a^{(3)} .* (1-a^{(3)})$

$\delta^{(2)} = (\theta^{(3)})^T \delta^{(3)} .* g'(z^{(2)}) \rightarrow a^{(2)} .* (1-a^{(2)})$

$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} . \delta_i^{(l+1)}$ (ignoring λ, if $\lambda=0$)

For the huge # of training set.

$\Delta_{ij}^{(l)} = 0$ (for $\forall\, i,j,l$) (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

for $i=1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagatn to compute $a^{(l)}$ for $l=2,3,\dots,L$

using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ $\cancel{\delta^{(1)}}$

$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \longrightarrow \Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T.$

$$\left.\begin{aligned} D_{ij}^{(l)} &= \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0 \\ D_{ij}^{(l)} &= \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0. \end{aligned}\right\} \quad \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

end.

1.3. Backpropagatn Intuitn:

$z_1^{(3)} = \Theta_{10}^{(2)} + \Theta_{11}^{(2)} \cdot a_1^{(2)} + \Theta_{12}^{(2)} \cdot a_1^{(2)}$

Backpropagatn: (running the feedforward algorithm, but doing it backwards.)

$cost(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log h_\Theta(x^{(i)})$ [Think of $cost(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$]

how well is the network doing on example i?

$\delta_2^{(2)} = \Theta_{12}^{(2)} \cdot \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}$

$\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}.$

2. BACKPROPAGATION IN PRACTICE.

2.1. Implementatn Note: Unrolling Paramtrs:

NN: ($L=4$)

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ – matrices (Theta1, Theta2, Theta3)

$D^{(1)}, D^{(2)}, D^{(3)}$ – matrices (D1, D2, D3)

"Unroll" into vectors.

ex/ $s_1=10,\ s_2=10,\ s_3=1$

$\Theta^{(1)} \in \mathbb{R}^{10\times 11}, \Theta^{(2)} \in \mathbb{R}^{10\times 11}, \Theta^{(3)} \in \mathbb{R}^{1\times 11}$

$D^{(1)} \in \mathbb{R}^{10\times 11}, D^{(2)} \in \mathbb{R}^{10\times 11}, D^{(3)} \in \mathbb{R}^{1\times 11}$

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];
Theta1 = reshape(thetaVec(1:110), 10, 11);
```

Learning Algorithm:

- have initial paramtrs $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$
- unroll to get initialTheta to pass to

fminunc (@costFunction, initialTheta, options).

function [jval, gradientVec]= costFunction (thetaVec)

- from thetaVec, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$
- use forward prop/back prop. to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$
- unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get gradientVec.

2.2. Gradient Checking:

It may look $J(\Theta)$ is decreasing, but you might end up with a NN that has a higher level of error than you would with a bug free implementatn.

gradApprox = (J(theta + ε) − J(theta − ε)) / (2ε)

```
for i=1 to n
    θp = θ;
    θp(i) = θp(i)+ε;
    θm = θ;
    θm(i) = θm(i)−ε;
    gradApprox(i) = (J(θp)−J(θm))/(2ε);
end
```

Check that

gradApprox ≈ DVec

gradApprox: from numerical

DVec: from back.prop

1) Implement back prop to compute DVec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$)
2) Implement numerical gradient check to compute gradApprox
3) Make sure they give similar values.
4) Turn off gradient check
5) Use back prop code for learning.

2.3. Random Initializatn:

→ Zero initializatn: $\Theta_{ij}^{(l)} = 0$ for $\forall$ i, j, l.

for every training examples $a_1^{(2)} = a_2^{(2)}$.

$$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta) \qquad \delta_1^{(2)} = \delta_2^{(2)}$$

$$\downarrow$$

updated weights would be equal to each other $\longleftarrow \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

⇒ After each update, paramtrs corresponding to inputs going into each of two hidden units are identical. $a_1^{(2)} = a_2^{(2)}$

⇒ $a_1^{(2)} = a_2^{(2)} = a_3^{(2)} = \dots = a_n^{(2)}$ means that ∀ of our hidden units are computing the exact same func. of the input! (highly redundant = equivalent with a unique feature, one unit on that layer).

INSTEAD!

→ Initialize each $\theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

Theta1 = rand (10,11) * (2ε) − ε; → $\boxed{-\varepsilon < \text{Theta1} < \varepsilon}$

Theta 2 = rand (1,11) * (2ε) − ε; → $\boxed{-\varepsilon < \text{Theta2} < \varepsilon}$

2.4. Putting It Together:

→ The more hidden units → the better.

Training a NN.

1. Randomly initialize weights
2. FP ($h_\theta(x^{(i)})$)
3. Compute $J(\theta)$
4. BP ($\frac{\partial}{\partial \theta_{jk}^{(l)}} J(\theta)$)

for i=1 to m
   FP & BP using $(x^{(i)}, y^{(i)})$
   (Get activatns $a^{(l)}$ & deltas $\delta^{(l)}$ for $l=2,3,\dots,L$)
   $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$
end

compute $\frac{\partial}{\partial \theta_{jk}^{(l)}} J(\theta)$

5. use grad. checking.
   then disable checking code.
6. Use grad. desc. or advanced optimizatn method with backprop. to try to minimize $J(\theta)$ as a fn of θ.

$J(\theta)$ — "non-convex", global minimum is not guaranteed, but in practice this is not a huge problem.

! What will if we try to maximize $J(\theta)$ at first, then initialize the weights with those (maximized $J(\theta)$), in order to, hopefully, end up with global minimum, or at least, with better local minimum?!

(3.) APPLICATION OF NN.

! What will it we apply BRIEF based logic into NNs, rather than CNNs.

→ Automous driving.

Once every 2 seconds, ALVINN digitizes a video img of the road ahead, and records the person's steering drn.

(3 layered Network)

→ This same procedure is repeated for other road types