Tuan Tran

A20357808

CS584 - Summer 2020.

1,

- ~~Gene.~~ In generative learning, we model the feature distribution and class priors → allow generating new examples and also do classification.
- In discriminative learning, we model $P(Y|X)$ → strictly doing prediction.
- Given nD ~~A~~ feature vector:

$$g(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n.$$

- Parameters $\theta_1 \dots \theta_n$ can be thought of as the components of the normal vector to the decision boundary

  θ_0 can be thought of as the negative distance from decision boundary to origin.

2, We have 2 options

- One vs all: ~~we~~ where we model the decision ~~boundary~~ boundaries that separate~~s~~ each class against ~~eve~~ all remaining classes
  ↳ This will produce k discriminant functions (one for each class)

- One vs each other where we separates each class against each of the remaining classes → this will produce $\frac{K(K-1)}{2}$ discriminant functions. (one for each pair of classes)

→ One vs each other produces more discriminant Functions and thus more parameters

3,

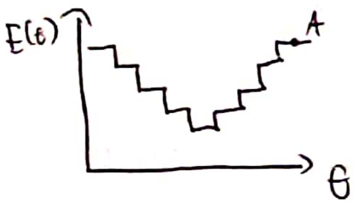
- Empirical Error:

$$E(\theta) = \# x^{*} + \# x^{**}$$

where $x^{*} = \{x^{(i)} \mid 1(y^{(i)}=0) \wedge \theta^T x > 0\}$

$x^{**} = \{x^{(i)} \mid 1(y^{(i)}=1) \wedge \theta^T x < 0\}$.

↳ The problem is that $E(\theta)$ is a piecewise function:



↳ We can't compute the gradient since we can easily stop at point A (gradient $= 0$).

→ Can't optimize using GD.

4,

PC: $E(\theta) = \sum_{x^{(i)} \in x^{*}} \theta^T x^{(i)} - \sum_{x^{(i)} \in x^{**}} \theta^T x^{(i)}$

→ $\nabla E(\theta) = \sum_{x^{(i)} \in x^{*}} x^{(i)} - \sum_{x^{(i)} \in x^{**}} x^{(i)}$

↳ Update:

$$\theta \leftarrow \theta - \eta\left(\sum_{x^{(i)} \in x^{*}} x^{(i)} - \sum_{x^{(i)} \in x^{**}} x^{(i)} \right)$$

5,

- In order to arrive at the logistic function as the hypothesis, we assume that the log of probabilities ratio can be modeled as a linear function:

$$\log \frac{P(y=1|x)}{P(y=0|x)} = \theta^T x.$$

- We simply equate $P(y=1|x) = \text{Sigmoid} = \frac{1}{1+e^{-\theta^T x}}$

→ the result is in the range of $(0, 1)$ where 0.5 represents complete uncertainty.

6,

$$l(\theta) = \log\left(\prod_{x^{(i)} \in C_1} P(y=1|x^{(i)}) \prod_{x^{(i)} \in C_0} P(y=0|x^{(i)})\right)$$

$$= \log \prod_{i=1}^{m} P(y=1|x^{(i)})^{y^{(i)}} P(y=0|x^{(i)})^{y^{(i)}}$$

$$= \sum_{i=1}^{m} y^{(i)} \log P(y=1|x^{(i)}) + (1-y^{(i)}) \underbrace{\log P(y=0|x^{(i)})}_{= \log(1 - P(y=1|x^{(i)}))}$$

where $h_\theta(x) = P(y=1|x)$

$$\rightarrow l(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

$$\frac{d}{d\theta} l(\theta) = \frac{d}{d\theta} \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))$$

$$= \sum_{i=1}^{m} y^{(i)} (1-h_\theta(x^{(i)})) x^{(i)} + (1-y^{(i)})(-h_\theta(x^{(i)})) x^{(i)}$$

$$= \sum_{i=1}^{m} x^{(i)} \left(y^{(i)} - y^{(i)} h_\theta(x^{(i)}) - h_\theta(x^{(i)}) + y^{(i)} h_\theta(x^{(i)})\right)$$

$$= \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x^{(i)}$$

-> ~~negative log like~~ gradient of negative log likelihood:

$$\frac{d}{d\theta}(-l(\theta)) = \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

- SGD update: For batch X_K:

$$\theta \leftarrow \theta - \eta \sum_{i \in X_K} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

- Should initialize parameters randomly ~~closely~~ close to 0
  Since as $\theta \to 0$, $h_\theta(x) \to 0.5$ -> represents ~~great~~ greatest uncertainty

7,

- For K classes, we use the softmax function since it outputs a distribution over all the ~~cla~~ classes and adds up to 1.
- Softmax is essentially a "one vs all" version of ~~logis~~ sigmoid.
- Update equation:

$$\theta_j \leftarrow \theta_j - \eta \left(h_{\theta_j}(x) - 1(y=j)\right) x^{(i)}$$

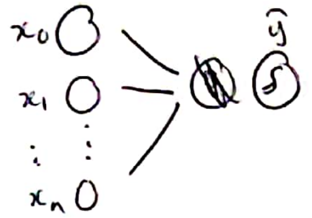
where $j = 1 \ldots K$ and θ_j = parameter vector for class j

⊛ Neural Net

⊛ 1,

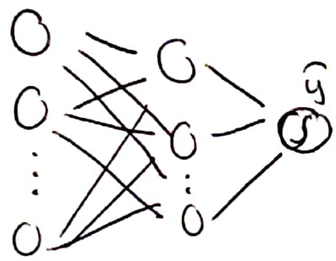
- Basic Structure :

A logistic unit :



-> We can stack mutiple logistic unit on top of each other to form a neuralnet,

Ex:



- Activation function is important as it introduces non-linearity into the neural net model -> give it ability to learn complex patterns.

- Several types of activation function : sigmoid, softmax, ReLU, leaky ReLU, linear (no activation), etc

2,

- ~~In~~ Feedforward net pushes the input ~~all th~~ forward all the way to output, Feedback net also does the forward push, but the output ~~loops~~ loops back to the input layer

- In the case of single output, we have exactly 1 parameter vector θ to plug into activation function

In K-class case, we have K parameter vectors $\theta_1 \dots \theta_K$

In terms of classification, we use sigmoid for single output and softmax for K-class outputs.

3,

- number of hidden units < number of ~~inp~~ inputs :

  → We are basically doing dimensionality reduction since we are condensing a larger number of features into a smaller dimensional space

  → Extracting the more important features

- number of hidden units > no of inputs :

  → We are doing non-linear mapping of features to higher dimensional space

  → Allows for more complex models.

4,

- Difficulty is that the update equations require knowing if $z^{(i)} = j$ or not where $z_j^{(i)}$ is the output of ~~the~~ unit j in hidden layer.

- Chain rule is used to derive update equations by moving backward one layer at a time, calculating gradient at each step. ~~Since~~ Since neural net is essentially a chain of functions, consider parameter v ~~and~~ into one output unit $\hat{y}$ ~~with~~ with error E

  $$\rightarrow \frac{\partial E}{\partial v} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v}.$$

~~8~~

5, Assume 2-layer

- Single output regression:

$$V = \begin{bmatrix} v_0 \\ \vdots \\ v_n \end{bmatrix} = \text{parameter vector of output layer}$$

$$w_j = \begin{bmatrix} w_{j0} \\ \vdots \\ w_{jn} \end{bmatrix} = \text{parameter vector of hidden unit } z_j.$$

$$E = \text{loss} = \frac{1}{2} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

$$\hookrightarrow \frac{\partial E}{\partial v} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v} = \frac{1}{2} \cdot 2 \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) z^{(i)}$$

$$= \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) z^{(i)}$$

+, Update for v:

$$v \leftarrow v - \eta \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) z^{(i)}$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial w_j}$$

$$= \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) v_j z_j^{(i)} (1 - z_j^{(i)}) x^{(i)}$$

+, Update for w_j:

$$w_j \leftarrow w_j - \eta \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) v_j z_j^{(i)} (1 - z_j^{(i)}) x^{(i)}$$

- Mutiple outputs: assume K outputs

→ v_j = parameter vector of $\hat{y}_j$, $j=1\ldots k$.

w_j = same definition as before.

$$E = \text{loss} = \frac{1}{2}\sum_{i=1}^{m} \underbrace{\sum_{l=1}^{K}}_{\text{sum over all } K \text{ classes}} (\hat{y}_l^{(i)} - y_l^{(i)})^2$$

+, $$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial v_j}$$

$$= \sum_{i=1}^{m} (\hat{y}_j^{(i)} - y_j^{(i)})\, z^{(i)}$$

→ Update for v_j where $j=1\ldots k$:

$$v_j \leftarrow v_j - \eta \sum_{i=1}^{m} (\hat{y}_j^{(i)} - y_j^{(i)})\, z^{(i)}$$

+, $$\frac{\partial E}{\partial w_j} = \underbrace{\sum_{l=1}^{K}}_{\text{summation because } w_j \text{ leads to } z_j \text{ which is input to all of output units.}} \frac{\partial E}{\partial \hat{y}_l} \cdot \frac{\partial \hat{y}_l}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_j}$$

(summation because w_j leads to z_j which is input to all of ~~k classes~~ output units.)

$$= \sum_{i=1}^{m}\sum_{l=1}^{K} (\hat{y}_l^{(i)} - y_l^{(i)})\, v_{lj}\, z_j^{(i)} (1 - z_j^{(i)})\, x^{(i)}$$

→ Update for w_j:

$$w_j \leftarrow w_j - \eta \sum_{i=1}^{m}\sum_{l=1}^{K} (\hat{y}_l^{(i)} - y_l^{(i)})\, v_{lj}\, z_j^{(i)} (1 - z_j^{(i)})\, x^{(i)}$$

- In case of regression, objective function is MSE, for multiclass, it's MSE over all classes.

7, Assume 2-layer:

- Single output Classification:

Using the sigmoid derivative rules

and $\frac{\partial E}{\partial v} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v}$

$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_j}$

where $E = \sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$

↳ we observe that $\frac{\partial E}{\partial v}$ and $\frac{\partial E}{\partial w_j}$ has same value as their regression counterpart.

→ $\therefore v \leftarrow v - \eta \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) z^{(i)}$

$w_j \leftarrow w_j - \eta \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) v_j z_j^{(i)} (1 - z_j^{(i)}) x^{(i)}$

- K-class Classification:

Similarly, we also ~~obt~~ obtain same gradient and update equations as the regression ~~counter~~ counterpart in the case of multiple outputs.

- Objective function used is the cross-entropy / negative log likelihood obtained from negating the ~~likelihood~~ likelihood.

6,

8,

- ~~Overfitt~~ Prevent overfitting by:
  - Earlystopping
  - L1, L2 regularization
  - Dropout.
- We can add regularization term to loss function:

Ex of 12:

$$E = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 + \lambda \|G\|^2$$

↳ update equation:

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j} - \lambda \|w_j\|$$

↳ Basically weight decay.

- Scaling input is important ~~some~~ since some input may have way bigger values than the other, which will have more influence ~~of~~ on output ~~to~~ even though ~~that may not~~ they may not be important features.

- One possible approach ~~to~~ adapt learning rate is learning rate decay, for ex:

$$\eta^{(i)} = \frac{\eta^{(i-1)}}{2}.$$

↳ . Useful since as we approach the minima, we ~~nee~~ may ~~m~~ need smaller and smaller learning rate to avoid overshooting the minima.

- ~~Moment~~ Update equation with momentum added:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial E}{\partial w_j} + \beta(w_j^{(t)} - w_j^{(t-1)})$$

↳ useful to help us ~~ov~~ overcome local minimum.

- Dropout is performed by randomly "shutdown" units in hidden layers during ~~train~~ training

→ ~~A~~ Prevent ~~co-adapt~~ co-adaptation

9,

CNN is used to process image

The idea is that we arrange the parameters in an ~~A to~~ n×n matrix called filter and we perform convolution (dot product) between ~~the~~ the filter and each n×n ~~ve~~ region of the image

↳ ~~Advan~~ ~~A to t~~ Advantages of doing this include ~~we get~~:

+, We can keep the ~~image~~ input image in its original form and not have to vectorize the image

+, There are ~~m~~ ~~much~~ fewer parameter compared to vectorizing the image and feed into MLP.

4 Deep learning:

1,