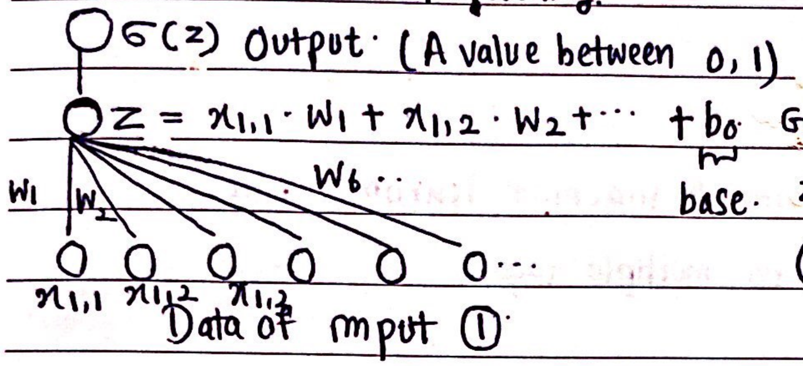
NOTES

Machine Learning.

Different trainings models.

Logistic regression.

Probability.



Simple model.

$z = x_1 \odot W + b_0$

Inner product.

Only ① filter (W)

Used for linearly seperable data.

There can be several different inputs (Ex:- MNSET 0,1,2‥ 9) for each of them the filter sets are different. bias values are different.

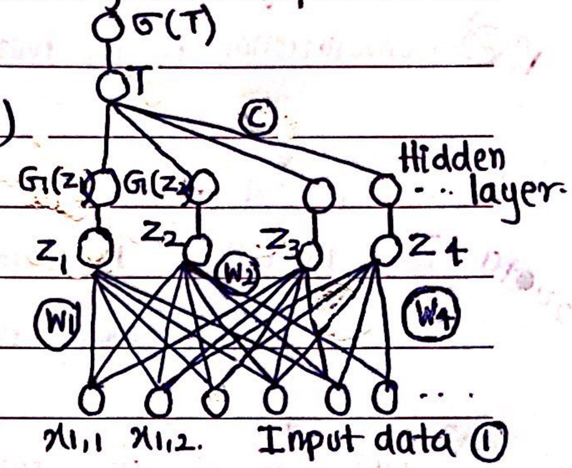
Then

$T_1 = C \odot G(W \odot x_1 + b) + t_1$

∴ Input to 2nd layer z_1 is send through another fung G (Softmax,)

(A neural network)

Multilayer Perceptron.



Complex model.

Multiple filters.

$W_1, W_2, W_3, \cdots$ C.

~~Used for non linear seperable~~ data.

$z_1 = x_{1,1} \cdot W_{1,1} + x_{1,2} \cdot W_{1,2} + \cdots + b_1$

$z_1 = x_1 \odot W_1 + b_1$

$z_2 = x_1 \odot W_2 + b_2$

Input to the 2nd layer isn't this z value.

∵ If we do that

$T = (C \odot \underbrace{z} + t)$

$\underbrace{z} = \underbrace{W} \odot x_1 + b \; (b_1, b_2, ..)$

$(z_1, z_2 \cdots) \quad (W_1, W_2, \cdots)$

$= (C \odot (W \odot x_1 + b) + t)$

$= F \odot x_1 + f$

This is same as LR

There are different models that we can use in ML.

Ex:- Logistic regression, MLP, CNN, . . .

Overfitting:
Using complex models can perform well on training data and not so well in real (Not generalized model). When we increase the depth of our model → increase the parameters of the model. → increase error rate.

Complex relationships may be too complex for reality.

How to choose a suitable model?

↙ Trying the model in real word by getting new real word data set

X Costly.

∴ Split our training data for 3 groups.

Training - Train the model.

Validation - Compare each approach. Use repeatedly to select best performance

Testing - Evaluate performance of our model by running the test set.

Equal to running new data (real world data)

Should only be used once. If not lead to bias.

Training set ← redefine the model

↓

learn parameters → estimate models preformance on validation set

↓ If the performance good

Test set.

NOTES

How to choose (best) optimized ~~p~~parameters?

The values of parameters when the loss function is minimum. – Gradiant desent.

$L(x_i, L_i)$ – The distance between the true label and the predicted label by the model for i^{th} input.

We need to take the average loss value.

$$AVGL = \frac{1}{N} \sum_{i=1}^{N} \underbrace{L(x_i, L_i)}$$

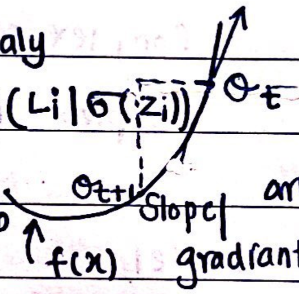
→ Normaly $= -\log P(L_i | \sigma(z_i))$

N – All the input.

$\sigma(z_i)$ – Predicted output value prob

Mostly used cruss entropy $= -\sigma(z_i) \log(L_i)$

Gradiant desent



Take the gradiant of the loss func in an arbitary point. Then (update) move the arbitary point in direction the gradiant ~~reduces~~. descent ([illegible])

$$\theta_{t+1} = \theta_t - \underbrace{\alpha_t}_{\text{step size}} \underbrace{\nabla f(\theta_t)}_{\text{gradiant at point } \theta_t}$$

In ML θ_t is the parameters that we have to learn.

We keep updating the parameters until loss func is minimum.

For large data set calculating gradient descent this way ~~was~~ required looking at every input data point ($\because \nabla AVGL = \nabla \frac{1}{N} \sum_{i=1}^{N} L(x_i, L_i) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i, L_i)$ have to calculate gradient of each data point and sum them)

Approximation – Randomly chossen j^{th} data point's gradiant approximately equal to $\nabla AVGL$

$$\nabla L(x_j, L_j) = \nabla AVGL$$

This is known as Stochastic gradient descent.

Not accurate as gradient descent

But fast. Does more updates withm a smgle tme.

May go m both gradient ascendmg and descendmg directions. But avg at descendmg way.

In SGD pomt update $\theta^{t+1} = \theta^{t} - \alpha^{t} \underbrace{\nabla f(\theta i)}_{\text{Estimated gradient}}$

$= \nabla f(\hat{\alpha}^{t})$

Early stoppmg.

Optimization goal - Do as well as possible m tramming set

Validation / Generalization goal - Maximize performance m real world.

Combme optimization with validation process.

Run trammg data set (with every iteration avg loss ↓)

~~While runnmg trammg data we will run Validation~~ set also once m a while. Not for parameter optimization. To estmate real world performance.

With every iteration compute avg loss. Before we choose optimized parameters when avg loss converge to 0. But now we will stop optimization when validation avg loss stops improvmg.

Advantage - Save computational cost.

Perform better in real world.