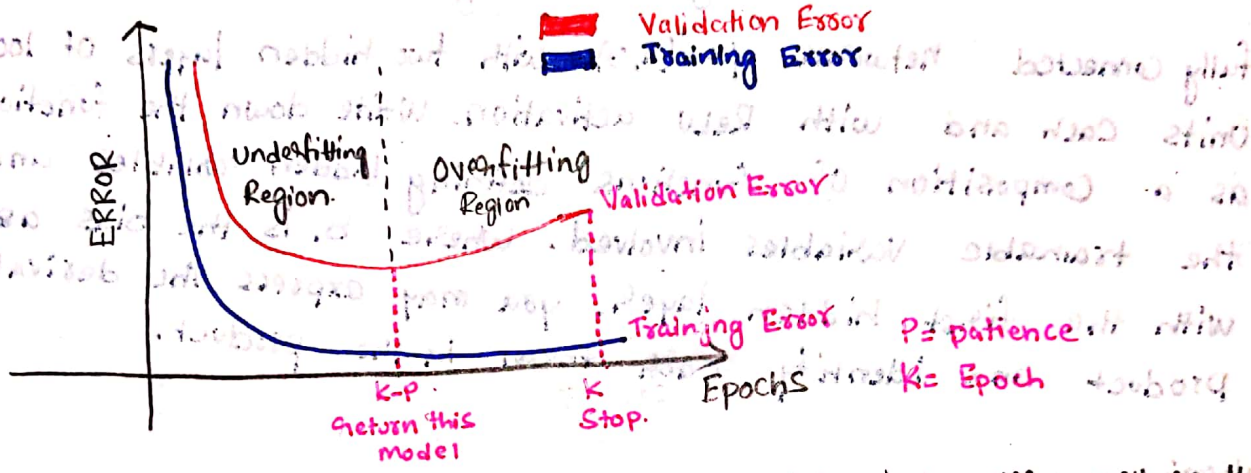
Final Exam

Saitham Sri Vatsavai

Problem 1: Sketch a typical learning curve for the training and validation sets, for a setting where overfitting occurs at some point. Assume that the training set and the validation set are of the same size. Label all the axis and label the curves that you sketch. State how to apply early stopping in the context of using a validation set.

Solution:



Overfitting:

The overfitting of the model occurs when a model trains very well on the training set but does not generalize well for validation set. Deep Neural networks are highly complex models thus it is very for them to overfit on the training set. where they learn training data exactly.

The above figure shows the variation of training error with validation over epochs. The model is said to be overfitting when the training error continues to decrease but validation error is not decreasing instead it starts increasing. This is shown in the overfitting Region in the above graph. In deep learning we need to maintain a bias variance tradeoff for the model to perform optimally. If a model has high bias then model is not trained well and we have high training and validation error. and model underfits. Similarly if we have high variance the model overfits the training set and does not perform well on test set or validation set. we need to maintain a bias variance tradeoff.

Early Stopping: It is a kind of Regularization technique. we use a patience parameter P, For the specified P epochs we observe the validation error and training error. If the validation error is increasing or remaining

Same while training error is decreasing then we can be sure Overfitting is happening. During this patience epochs we store the weights of the model at each epoch. When we see overfitting is occuring instead of further training, we stop the training and return the model weights at K-p where the Validation loss is minimum. This is called early Stopping.

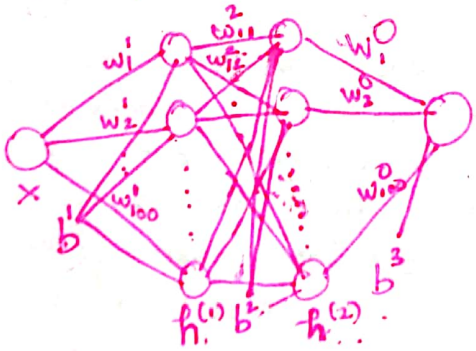
Problem 2: Consider Solving a binary classification problem using a fully connected network $y=f(x,\theta)$ with two hidden layers of 100 hidden units each and with ReLU activation. Write down the function $y=f(x,\theta)$ as a composition of functions defining hidden variables and clearly the trainable variables involved. Where b_1 is the bias associated with the first hidden layer. you may express the derivate as a product and identify each term in the product.

Solution:

The network is a feed forward Neural Network.

$y=f(x,\theta)$.

Training Set $= \{(x_i, y_i)\}_{i=1}^{N}$ → So we have only one feature for each training sample.



$\theta = \{w^{(l)}, b^{(l)}\}_{l=1}^{L}$

$x = [x_1\ x_2\ x_3\ x_4\ \ldots\ x_N]$ → training Set.

Trainable Parameters. θ {

$w^1 \in \mathbb{R}^{1\times 100}$ $\quad b^1 \in \mathbb{R}^{100}$ $\quad w^0 \in \mathbb{R}^{100}$

$w^2 \in \mathbb{R}^{100\times 100}$ $\quad b^2 \in \mathbb{R}^{100}$ $\quad b^3 \in \mathbb{R}^1$

ReLU → Activation. $h^{(1)} = g(w^{(1)}x + b^{(1)}) \in \mathbb{R}^{100}$

ReLU-Activation. $h^{(2)} = g(w^{(2)}h^{(1)} + b^{(2)}) \in \mathbb{R}^{100}$

Sigmoid-Activation $\hat{y} = \sigma(w^0 h^{(2)} + b^3) \in \{0,1\}$

$g(x) \rightarrow \text{ReLU} = \max\{x, 0\}$

$= \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$

The problem is a binary Classification, so the Loss function to be used is binary cross entropy loss.

$$L(y_i, \hat{y}_i) = -y_i \log \sigma(\hat{y}_i) + (1-y_i) \log (1 - \sigma(\hat{y}_i))$$

$$\mathcal{L} = \frac{1}{N} \sum_i L(y_i, \hat{y}_i).$$

$$\rightarrow \quad \frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial y^{(2)}} \frac{\partial y^{(2)}}{\partial y^{(1)}} \frac{\partial y^{(1)}}{\partial b_1}.$$

$$y^{(1)} = w'x + b'.$$

$$\frac{\partial y^{(1)}}{\partial b_1} = 1.$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial y^{(2)}} \cdot \frac{\partial y^{(2)}}{\partial y^{(1)}}}$$

$f(x_1, x_2) = x_1 + x_2 - \cos(x_1 + x_2)$

Problem 3: For $f(x) = x_1^2 + x_2^2 - \cos(x_1 + x_2)$ where $x = [x_1, x_2]$

1. Show that $[0,0]$ is a local minimum of $f(x)$
2. Starting with $[1,2]$ carry out two iterations of the gradient descent algorithm with a learning rate of 0.01;
3. What is approximately the rate of convergence of gradient descent.

Solution:

(1) $f(x) = x_1^2 + x_2^2 - \cos(x_1 + x_2)$. where $x = [x_1, x_2]$

for $f(x)$ to be local minimum gradient should be zero.

$$\frac{\partial f(x)}{\partial x_1} = \frac{\partial}{\partial x_1}\left(x_1^2 + x_2^2 - \cos(x_1 + x_2)\right).$$

$$= 2x_1 + 0 - (-\sin(x_1 + x_2)).$$

$$\frac{\partial f(x)}{\partial x_1} = 2x_1 + \sin(x_1 + x_2)$$

For local minimizer. $\frac{\partial f(x)}{\partial x_1} = 0.$

$$2x_1 + \sin(x_1 + x_2) = 0.$$

$$2x_1 = -\sin(x_1 + x_2).$$

$$\sin(x_1 + x_2) = -2x_1$$

$$x_1 + x_2 = \sin^{-1}(-2x_1).$$

$$\boxed{x_2 = \sin^{-1}(-2x_1) - x_1} \quad -(1)$$

$$\frac{\partial f(x)}{\partial x_2} = 2x_2 + \sin(x_1 + x_2). = 0$$

From (1)

$$2x_2 = -\sin(x_1 + x_2)$$

$$2x_2 = -\sin(x_1 + \sin^{-1}(-2x_1) - x_1)$$

$$2x_2 = -\sin(\sin^{-1}(-2x_1)). \quad [\because \sin(\sin^{-1} x) = x$$

$$2x_2 = 2x_1$$

$$\boxed{x_1 = x_2}$$

Thus. at [0,0] $x_1 = x_2$. So it will be local minimizer.

(2)

$$f(x_1,x_2) = x_1^2 + x_2^2 - \cos(x_1+x_2)$$

$\eta = 0.01$. [1, 2] $f(x_1,x_2) \longleftarrow 4.001$

First iteration: $x_1' = x_1 - \eta \frac{\partial f(x_1,x_2)}{\partial x_1}$.

$$x_2' = x_2 - \eta \cdot \frac{\partial f(x_1,x_2)}{\partial x_2}.$$

$$\frac{\partial f(x_1,x_2)}{\partial x_1} = 2x_1 + \sin(x_1+x_2).$$

$$\frac{\partial f(x_1,x_2)}{\partial x_2} = 2x_2 + \sin(x_1+x_2).$$

$$x_1' = 1 - 0.01 \times \frac{\partial f(x_1,x_2)}{\partial x_1}\Big|_{x_1=1,\ x_2=2}$$

$$x_1' = 1 - 0.01\left(2(1) + \sin(1+2)\right).$$

$$x_1' = 0.979.$$

$$x_2' = x_2 - \eta \cdot \frac{\partial f(x_1,x_2)}{\partial x_2}.$$

$$x_2' = 2 - 0.01 \times \left(2(x_2) + \sin(x_1+x_2)\right)$$

$$x_2' = 2 - 0.01 \times (2\times 2 + \sin(3))$$

$$x_2' = 1.95.$$

Update $\Rightarrow$ $x_1' \longleftarrow 0.979$ $x_2' \longleftarrow 1.959$. $f(x_1',x_2') \longleftarrow 3.79$

Second iteration:

$$x_1^2 = x_1' - \eta \frac{\partial f(x)}{\partial x_1}\Big|_{x_1=x_1',\ x_2=x_2'}$$

$$x_1^2 = 0.979 - 0.01\left(2(0.979) + \sin(0.979+1.959)\right)$$

$$x_1^2 = 0.958.$$

$$x_2^2 = x_2^1 - 0.01 \left.\frac{\partial f(x)}{\partial x_2}\right|_{x_1 = x_1^1,\ x_2 = x_2^1}$$

$$x_1^2 = (1.959) - 0.01 \times \left(2(1.959) + \sin(0.979 + 1.959)\right)$$

$$x_2^2 = 1.919$$

$x_1^2 \leftarrow 0.958$ $\quad x_2^2 \leftarrow 1.919$ $\quad f(x_1^2, x_2^2) \leftarrow 3.60$

If we observe with each iteration the cost function value is decreasing which the outcome of gradient descent (changing parameters x_1 and x_2 until we make Costfunction minimum.

(3) The rate of convergence is ~~approximate~~ the numbers of steps it would take to reach the ~~data~~ minima.

If the learning rate $\eta \leq 1/L$, after T steps

$$f(x_t) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\eta T}$$

$$\leq \frac{\|(1,2) - (0,0)\|^2}{2\eta T}$$

$f(x^*)$ is the optimal value.

Rate of Convergence $O\left(\frac{1}{T}\right)$

$$f(1,2) - f(0,0) \leq \frac{\|x_0 - x^*\|^2}{2\eta T}$$

$$4 - (-1) \leq \frac{5}{2 \times 0.01 \times T}$$

$$1 \leq \frac{5}{2 \times 0.01 \times T} \qquad T \times 0.01 \times 2 \leq 1$$

$$T \leq 50 \qquad T \leq \frac{100}{2}$$

Hence it would take around 50 iterations for gradient to converge with learning rate 0.01.

Problem (4): When training using mini-batch gradient descent algorithm it sometimes happens that the training loss goes up after performing an update operation/iteration. What are two possible reasons that can cause this to happen? Explain your answers.

Solution: The mini batch gradient descent algorithm is a mixture of both stochastic gradient deseent and batch gradient descent. The training set is divided into multiple groups called batches. Each batch has a number of training samples. At a time single batch is passed through the network which computes the loss of every sample in the batch and uses their average to update the parameters of the neural network. The two possible reasons that can cause training loss to go up after an updation are

(1) It approximates the gradient to reduce with a set of data points defined by batch size rather than all the dataset, so it may lead to bad approximation as that particular batch may not represent the complete data set accurately.

(2) It might partially depend on the value of learning rate a because if learning rate is high then with update we may miss local minima causing the loss to increase.

(3) It also depends on the batch size, if batch size is very less then also model with not able to update the weights in the

correct direction. It doesnot directly go to the local minima, rather it takes zig zag moments, which leads to increase in the loss.

Problem 5: Consider a CNN with two convolutional layers and a fully connected layers. The first layer is a valid convolution with a kernel of size $3\times3\times3\times16$ followed by 2×2 max pooling. The second layer is a valid convolution with a kernel of size $3\times3\times16\times32$. Then we have a flatten layer followed by a fully connected layer to produce an output of dimension 10. If the input has dimension $64\times64\times3$, write down the function $y=f(x,\theta)$, as a composition of functions defining hidden variables and identify trainable variables involved.

Solution:

Given $I \in \mathbb{R}^{M\times N}$.

$$S(i,j) = \sum_{m=0}^{K_{m-1}} \sum_{n=0}^{K_{n-1}} I\left(i_{s-p+m}, j_{s-p+n}\right) K(m,n).$$

where $0 \le i \le (M-K_m+2p)/s$.

$0 \le j \le (N-K_n+2p)/s$

$$S \in \mathbb{R}\left[\frac{m-K_m+2p}{s}+1\right] \times \left[\frac{N-K_n+2p}{s}+1\right]$$

Input $= I \in \mathbb{R}^{64\times64\times3}$

conv 1 layer $= K \in \mathbb{R}^{3\times3\times3\times16}$ → Channel = 16

Max pool layer $= 2\times2$

conv 2 layer $= K \in \mathbb{R}^{3\times3\times16\times32}$ → Channel = 32

Output = Softmax $\in \mathbb{R}^{10}$

$$S(i,j) = \sum_{k=0}^{C} \sum_{m=0}^{K_{m1}} \sum_{n=0}^{K_{m1}} I\left(i_{s\cdot p+m}, j_{s\cdot p+n}\right) K(m,n)$$

First Convolutional layer.

$$S(i,j) = \sum_{k_0=0}^{16} \sum_{m=0}^{K_{m1}} \sum_{n=0}^{K_{m1}} I\left(i_{1\cdot 0+m}, j_{1+n}\right) K'(m,n)$$

$K' \in \mathbb{R}^{3\times3\times16}$ b

trainable.

Activation layer.

$$h^{(1)}_{ij} = f\left(S(i,j) + b'_j\right)$$

trainable.

$h^{(1)} \in \mathbb{R}^{(60,60,32)}$ $h^{(1)}$ f is ReLU function and bi is biad.

$h^{(1)} \in \mathbb{R}^{m_1 \times n_1 \times c_1}$.

Max pooling layer. 2×2.

$$h^{(2)}(i,j,c) = \max h^{(1)}\left(s_i : s_i+f, s_j : s_j+f, c\right)$$

S-tride, f- spacial extent

$h^{(2)} \in \mathbb{R}^{(14,14,64)}$ $h^{(2)}(i,j,c) = \max h^{(1)}\left(s_i : s_i+2, s_j : s_j+2, c\right)$

Second Convolutional layer.

$$S^2(i,j) = \sum_{k=0}^{32} \sum_{m=0}^{K_{m1}} \sum_{n=0}^{K_{m1}} h^{(2)}\left(i_{1+m}, j_{1+n}\right) K'(m,n)$$

trainable.

$h^{(3)} \in \mathbb{R}^{(14,14,64)}$ $h^{(3)} = f\left(S^2(i,j) + b^2_j\right)$. trainable b^2 is bais.

f is a ReLU function and

Flatten layer: $h^{(4)} \in \mathbb{R}^{(12544,1)} \longrightarrow W \in \mathbb{R}^{12544}$ → Trainable

Output layer. $y = \text{Softmax}\left(h^{(4)} * W^0 + b^0\right) \in \mathbb{R}^{10}$

The activations of the output of a Convolution layers is calculated by

$$S_j^{(l)} = f\left(\sum_{i=1}^{C} \tilde{W}_{ij}^{(l)} * S_i^{(l-1)} + b_j^{(l)}\right).$$

l is the index of convolutional layer. $x_j^{(l)}$ denotes the j^{th} Channel of the output volume, $w_{ij}^{(l)}$ is a 3D filter kernel. Connecting i^{th} channel of the input channel volume to the j^{th} ~~output~~ Channel of output, $b_j^{(l)}$ denotes the bias term of the j^{th} output channel.

Kernels in all the layers and bias in all the layers are trainable parameters and max pool layers does not have ~~training~~ any training parameters.

Problem 6: PCA, Autoencoders, and GAN are three basic methods of Unsupervised learning. Describe one similarity and one difference between the following two methods: (1) PCA and Autoencoders (2) Autoencoders and GAN.

PCA:

→ PCA learns a linear transformation that projects the data into another space, where vectors of projections are defined by variance of the data. By restricting the dimensionality to a number of components that account for most of the variance of the dataset. We achieve dimensionality reduction.

Auto Encoders:

Autoencoders are neural networks that are used to reduce the data into a low dimensional latent space by stacking multiple non-linear transformations. They have encoder and decoder. The encoder maps the input to latent space and decoder reconstruct the input. They are trained using back propagation.

Comparison.

1) PCA is a linear transformation but Auto encoders are capable of modelling complex non linear functions.
2) PCA features are linearly uncorrelated as features are projected onto the orthogonal basis. Auto encoder features might have correlations
3) PCA are faster and computationally cheaper.
4) A single layered auto encoder with linear activation function is very similar to PCA.
5) Both are known for dimensional reduction problem.

Generative Adversarial Networks: Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that could have been drawn from the original dataset.

Comparison

Both GANs and Autoencoders are generative models, which learn a given data distribution rather than density.

The difference lies in how they learn this.

Autoencoders learn a given distribution comparing its input to its output; this good for learning hidden representation of data.

But it is not good for generating new data.

Generative Adversarial Networks use a discriminator to measure the distance between the generated and real data.

Discriminator receives some data as an input and returns a number between 0 and 1, 0 meaning that data is fake and 1 meaning real. The generators goal is to convince the discriminator to believe generated data is real.

The main advantage of GANs over Autoencoders is generating data is that they can be conditioned by different inputs

Autoencoders are more suitable for compressing data to the lower dimensions or generating semantic vectors from it. where GANs are more suitable for generating data.