Issues in training the deep neural networks

1. Gradients growing ever smaller or larger:
   1. Both issues make lower layers very hard to train.
   2. Now, if activation function sigmoid is being used, and let the value that we get is 5,10,100 sigmoid will only give us 1 every time, thus we will not know the change thus training will be hard.
   3. Here the loss component will not be received.
2. Insufficient training data for large networks.
3. Training may be extremely low
4. A model with millions of parameters would risk of overfitting.
5. Vanishing gradient
   1. Gradient gets smaller and smaller as the algorithm progresses down to the lower layers. As a result, connection weights are virtually in changes, and training never converge to a good solution
6. Exploding gradient

Solution by xavier

1. Combination of sigmoid and weight initialization techniques. (Normal dist mean = 0, std =1)
2. The variance if the outputs of each layer is much greater than the variance of its inputs.
3. Going forward in the network the variance keeps increasing after each layer until the activation function saturates at the top layers.

Glorot and He Initialization

1. Glorot and Bengio proposed:
   1. Variance of outputs of each layer = variance of its inputs, and the gradients to have equal variance before and after flowing through a layer in the reverse direction.
   2. The connection weight of each layer must be initialized randomly as shown
   3. Normal distribution with mean 0 and variance sigma square = 1/fan(avg)
   4. Uniform distribution between r and -r with r = root(3/fan(avg))
2. Glorot = none, tanh, sigmoid, softmax, sigma squate = 1/fan(avg)
3. He = ReLu, Leaky ReLu, ELU, GELU, Swish, Mish, sigma square = 2/fan(in)
4. LeCun = SELU sigma squred = 1/ fan(in) (fan in = inputs avg)

Fan(avg) and Fan(in) are examples of incremental research.

By default, Keras uses glorot.

He initialization

Import tensorflow as tf

Dense = tf.keras.layers.Dense(50, activation =’relu’, kernel\_initialization = ‘he\_normal’)

He initialization with a uniform distribution and based on fan(avg) using the code:

He\_avg\_init = tf.keras.initializer.VarianceScaling(scale =2, mode=’fan\_avg’, distribution=’uniform’)

Dense = tf.keras.Dense(50, activation=’sigmoid’, kernel\_initialization = he\_acg\_init)

Shallow models me ye problem nai hoti as gradient bada nai hota hai

Deep network = 4/5 hidden layer. Every layer i/p o/p variance must be same. Initialization starts when network starts with these weights. After initialization, gradient ki exploding position nai aegi

Batch Normalization (BN)

Normalization cannot happen with full data as the whole data is not exposed thus normalization happens in batches.

* Sergey Ioffe proposed a technique of batch normalization that addresses the problems of vanishing/exploding gradient
* Batch normalization acts like a regularizer.
* It adds some complexity to the model.
  + Cpu cycles extra lagega
* There is a runtime penelty
  + Prediction will be slower
* It is possible to fuse BN with other layers
* Estimates each input’s mean and standard deviation. It does so by evaluating the mean and standard deviation of the input over the current mini-batch.

What about at test time?

During training, BN standardizes its inputs, then rescales and offsets them.

Test ke time single instance hota hai

Single i/p (at the time of test jo input aaya hai) ka mean and variance nai aa sakta, then how to normalize that?

Solution:

* Use mean or variance of training but har batch ka different that oh kaunsa le, for that we take moving avg or exponential moving avg. Using this, moving avg of mean and variance normalize
* We may need to make prediction for individual instances rather than for batches of instances: in this case, we will have no way to compute each input’s mean and standard deviation thus we take out the moving average and feed it to the layer at time of test.