KTH Royal Institute of Technology
Stockholm

School of Electrical Engineering and
Computer Science

Scalable Machine Learning and Deep Learning - ID2223

Review Questions 3

Author
Emil Ståhl

Author
Selemawit Fsha

Author
Erik Kongpachith

November 27, 2021

1 Is it OK to initialize all the weights of a neural network to the same value as long as that value is selected randomly using He initialization? Is it okay to initialize the bias terms to 0?

No, all weights must be sampled independently, meaning that they should not all have the same initial value even if you use He initialization. The purpose of sampling weights to different values is to break the symmetry, this is not the case if the weights are initialized to the same value since all neurons of a layer now are the same, this results in that back propagation is unable to work correctly and break the symmetry. With all the weights being the same, is like having only one neuron per layer making it slow and impossible for the network to produce a good solution. Regarding initializing biases to 0, that is totally fine. You can even initialize biases just like weights without it making a difference on the model.

2 In which cases would you want to use each of the following activation functions: ELU, leaky ReLU, tanh, logistic, and softmax?

ELU

The ELU activation function is a good default. Exponential Linear Unit or its widely known name ELU is a function that tend to converge cost to zero faster and produce more accurate results. Different to other activation functions, ELU has a extra alpha constant which should be positive number. Benefits of ELU include that it becomes smooth slowly until its output equal to -α whereas RELU sharply smoothes, ELU is a strong alternative to ReLU, and unlike ReLU, ELU can produce negative outputs. However, for x $>$ 0, it can blow up the activation with the output range of [0, inf].

Leaky ReLU

If you need the neural network to be as fast as possible, you can use one of the leaky ReLU variants instead (e.g., a simple leaky ReLU using the default hyperparameter value). The simplicity of the ReLU activation function makes it a preferred option in many situations, despite the fact that it is generally outperformed by other functions regarding performance. However, the ReLU activation function's ability to output precisely zero can be useful in some cases. Moreover, it can sometimes benefit from optimized implementation as well as from hardware acceleration. It avoids and rectifies vanishing gradient problem. ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. One of its limitations is that it should

only be used within hidden layers of a neural network model. Some gradients can be fragile during training and can die. It can cause a weight update which will makes it never activate on any data point again. In other words, ReLu can result in dead neurons. In another words, for activations in the region ($x<0$) of ReLu, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes). This is called the dying ReLu problem.

tanh

The hyperbolic tangent (tanh) can be useful in the output layer if you need to output a number between –1 and 1, but nowadays it is not used much in hidden layers (except in recurrent nets). The gradient is stronger for tanh than sigmoid (derivatives are steeper). However, tanh also has the vanishing gradient problem.

Logistic

The logistic activation function is also useful in the output layer when you need to estimate a probability (e.g., for binary classification), but is rarely used in hidden layers. Advantages of logistic functions include smooth gradient preventing "jumps" in output values, output values bound between 0 and 1, normalizing the output of each neuron. Disadvantages are vanishing gradients for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction. Lastly, outputs not zero centered and it is computationally expensive

Softmax

Finally, the softmax activation function is useful in the output layer to output probabilities for mutually exclusive classes, but it is rarely used in hidden layers. Softmax is able to handle multiple classes only one class in other activation functions—normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class. Useful for output neurons, for neural networks that need to classify inputs into multiple categories.

3 What is batch normalization and why does it work?

Batch normalization makes the learning of layers in the network more independent of each other. It is a technique to address the problem that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. It consists of normalizing activation vectors from hidden layers using the first and the second statistical moments (mean and variance) of the current batch. This normalization step is applied right before (or right after) the nonlinear function. The BN layer first determines the empirical mean and the standard deviation of the activation values across the batch. It then normalizes the activation vector $Z^{(i)}$. That way, each neuron's output follows a standard normal distribution across the batch. At each iteration, the network computes the mean and the standard deviation corresponding to the current batch. Then it trains the scaling and shifting parameter through gradient descent, using an Exponential Moving Average (EMA) to give more importance to the latest iterations.

4 Does dropout slow down training? Does it slow down inference (i.e., making predictions on new instances)?

Yes, dropout does slow down training, in general by a factor of two. This is due to the estimation of the loss and gradient containing noise when dropping certain units. This can cause the optimizer to not always move in the right direction towards the local minimum. Thus, more training time is needed. However, it has no impact on inference speed since it is only turned on during training.

5 Consider a CNN composed of three convolutional layers, each with 3 × 3 filters, a stride of 2, and SAME padding. The lowest layer outputs 100 feature maps, the middle one outputs 200, and the top one outputs 400. The input images are RGB images of 200 × 300 pixels. What is the total number of parameters w in the CNN?

Lowest layer

3x3 filters
3 input channels for RGB
Outputs 100 feature maps
In each feature map we have 3x3x3 = 27 weights plus one additional bias weight = 28
For 100 output feature maps we have 28x100=2800 parameters

Middle layer

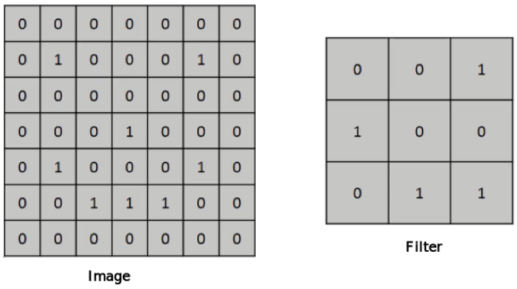
3x3 filters
Input 100 feature maps
Outputs 200 feature maps
In each feature map we have 3x3x100 = 900 weights plus one additional bias weight = 901
For 200 output feature maps we have 901x200=180200 parameters

Top layer

3x3 filters
Input 200 feature maps
Outputs 400 feature maps
In each feature map we have 3x3x200 = 1800 weights plus one additional bias weight = 1801
For 400 output feature maps we have 1801x400=720400 parameters

The total number of parameters w in the CNN is 2800+180200+720400=903400

6 Consider a CNN with one convolutional layer, in which it has a 3 × 3 filter (as shown below) and a stride of 2. Please write the output of this layer for the given input image (the left image in the following figure)?



The receptive field will scan areas with size 3x3 and with a stride of 2. This will result in 9 scans of 3x3 areas and the output of the layer will be of the size 3x3. The output will be the following:

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$