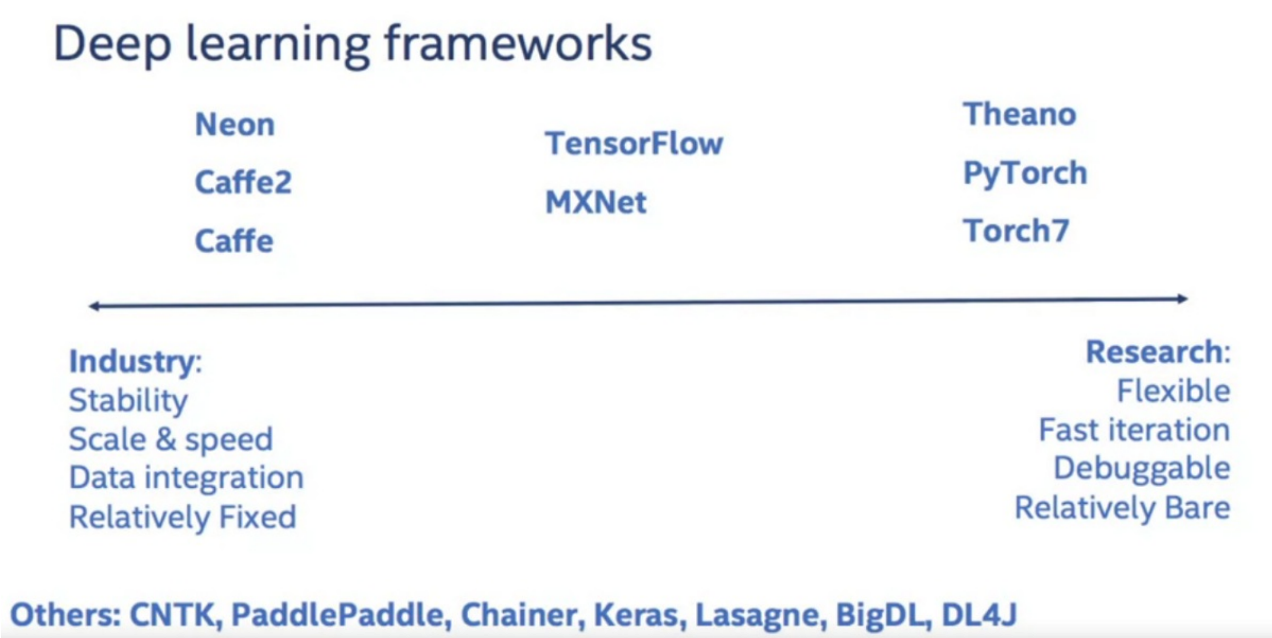
Practical Deep Learning — Intel Course (WEEK — 01)

medium.com/@aakashgoel12/practical-deep-learning-intel-course-week-01-7eae79b7357f

July 13, 2020



Deep learning framework

neon overview

Backend	NervanaGPU, NervanaCPU
Datasets	MNIST, CIFAR-10, Imagenet 1K, PASCAL VOC, Mini-Places2, IMDB, Penn Treebank, Shakespeare Text, bAbI, Hutter-prize, UCF101, flickr8k, flickr30k, COCO
Initializers	Constant, Uniform, Gaussian, Glorot Uniform, Xavier, Kaiming, IdentityInit, Orthonormal
Optimizers	Gradient Descent with Momentum, RMSProp, AdaDelta, Adam, Adagrad, MultiOptimizer
Activations	Rectified Linear, Softmax, Tanh, Logistic, Identity, ExpLin
Layers	Linear, Convolution, Pooling, Deconvolution, Dropout, Recurrent, Long Short-Term Memory, Gated Recurrent Unit, BatchNorm, LookupTable, Local Response Normalization, Bidirectional-RNN, Bidirectional-LSTM
Costs	Binary Cross Entropy, Multiclass Cross Entropy, Sum of Squares Error
Metrics	Misclassification (Top1, TopK), LogLoss, Accuracy, PrecisionRecall, ObjectDetection

Image 01: Neon Overview

The MNIST dataset contains 70,000 images that are 28 by 28 pixels in size. The goal is to classify the images as one-digit numbers from 0 through 9.

If this MLP contains one hidden layer containing 64 units, how many total parameters, including weights and biases, are in the network?

50890

Correct Response
The weights between the input layer and hidden layer is 784 times 64, the weights between the hidden layer and output layer is 64 times 10, and there are 64 plus 10 biases, giving a total of 50890 weights and biases in this network.

Image 02: Question — Parameter Calculation

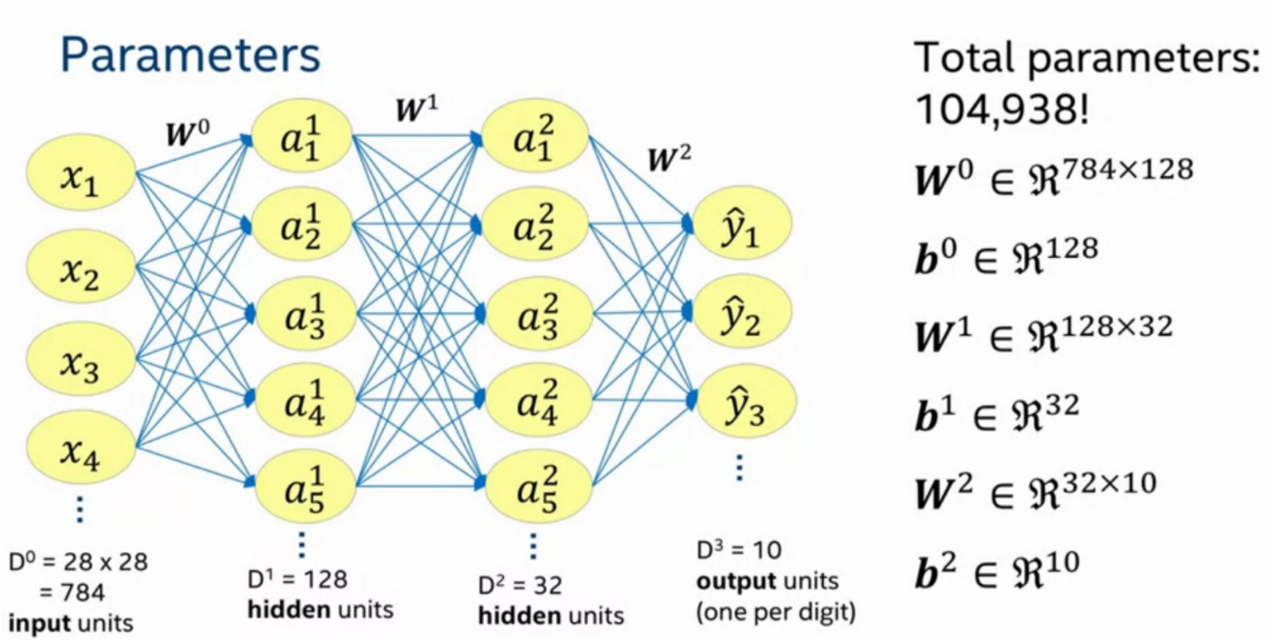


Image 03: NN (Parameters)

Order the following steps when training a neural network:

a. Cost

b. Forward-pass

c. Update weights

d. Randomly seed weights

e. Backward-pass

f. Fetch a batch of new data

Enter your answer as a string of letters a through f to represent the order. For example, one possible answer could be "bcfeda"

dfbaec

Correct Response

Image 04: NN Training Process

Activation Fn

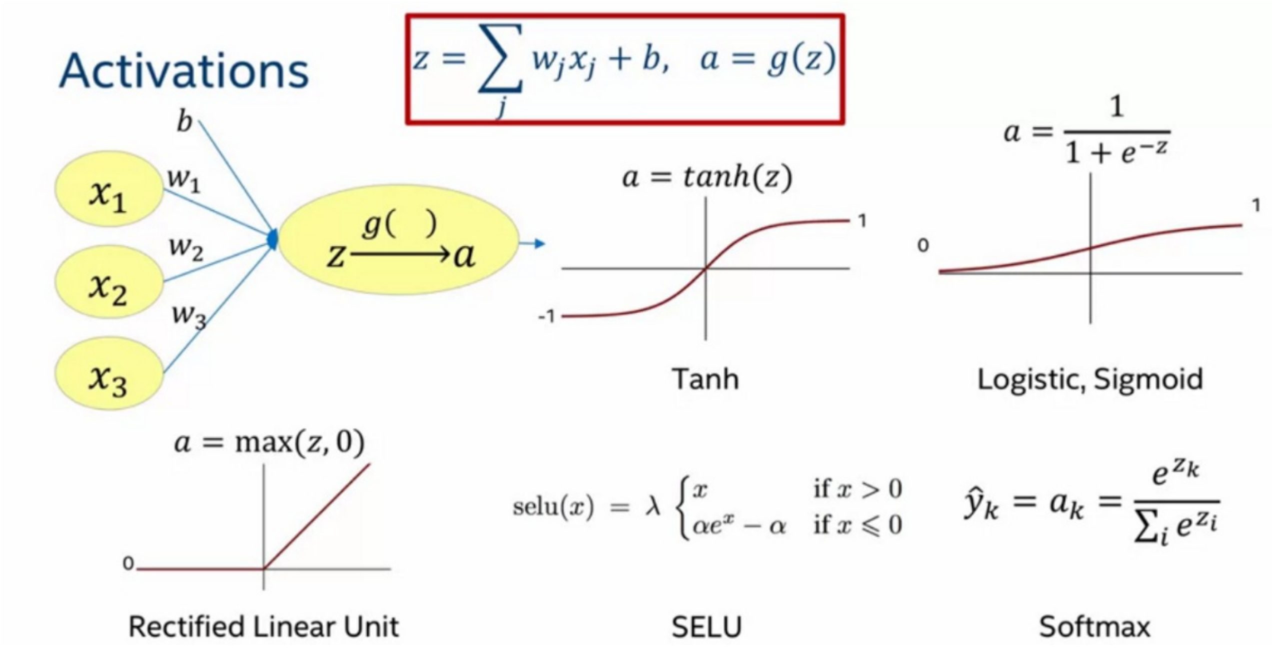


Image 05: Activation

- **TanH →** Suppose your i unit is responsible for detecting a particular feature. If your input z is near zero, then you're uncertain about the presence of that feature. In that case, your gradient or derivative is high, which encourages training. However, when your input z is too high or too low, your gradient is almost zero, causing the weight not to learn.
- **RELU →** The rectified linear unit activation function or RELU has become quite popular as it was found to accelerate the train process compared to the sigmoid or hyperbolic tangent functions. The RELU can be implemented by simply thresholding a matrix of inputs at zero and does not require computing exponentials. However, RELU units can be fragile during the training and can die. **E.g.** large gradient flowing through a RELU unit could cause the weights to update in such a way that the unit will never activate. To mitigate: Noisy RELU, leaky RELU and parametric RELU.
- SELU → **Very** good link https://mlfromscratch.com/activation-functions-explained/#/

Weight Initialization matrix

Initialization

Gaussian	Gaussian(mean, std)	
GlorotUniform	Uniform$(-k, k)$ $\quad k = \sqrt{\frac{6}{d_{in} + d_{out}}}$	Logistic
Xavier	Uniform(k, k) $\quad k = \sqrt{\frac{3}{d_{in}}}$	Logistic
Kaiming	Gaussian$(0, \sigma^2)$ $\quad \sigma = \sqrt{\frac{2}{d_{in}}}$	ReLU

http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

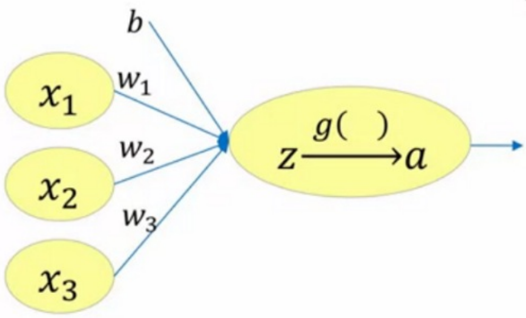
Image 06: Weight Initialization

As we backpropagate the gradients, they can exponentially increase or decrease, and for very deep networks, they can **explode** or **diminish** to zero.

To mitigate **this** is to initialize the weight such that the activations and backpropagations have **unit variance**.

FOR ***LOGISTIC/SIGMOID:***

Initialization



Logistic have linear activation at origin

$$a = g(z) = z$$

$$var[A] = \sum_{i=i}^{d_{in}} var[X]var[W]$$

$$= d_{in} var[X]var[W]$$

Forward prop: $var[W] = \frac{1}{d_{in}}$

Backward prop: $var[W] = \frac{1}{d_{out}}$

It can be shown, Compromise: $var[W] = \frac{2}{d_{in}+d_{out}}$

Glorot & Bengio 2010 "Understanding the difficulty of training deep feedforward neural networks"

Image 07: Weight Initialization (Logistic)

The ***variance of the activations*** is equal to the variance of the input times the variance of the weight, times the number of inputs d_in during the forward propagation, or times the number of outputs d_out during the backward propagation.

To derive above formula, refer to https://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

Suppose we have an input X with n components and a **linear** neuron with random weights W that spits out a number Y. What's the variance of Y? Well, we can write

$$Y = W_1 X_1 + W_2 X_2 + \cdots + W_n X_n$$

And from Wikipedia we can work out that $W_i X_i$ is going to have variance

$$\text{Var}(W_i X_i) = E[X_i]^2 \text{Var}(W_i) + E[W_i]^2 \text{Var}(X_i) + \text{Var}(W_i)\text{Var}(i_i)$$

Now if our inputs and weights both have mean 0, that simplifies to

$$\text{Var}(W_i X_i) = \text{Var}(W_i)\text{Var}(X_i)$$

Then if we make a further assumption that the X_i and W_i are all independent and identically distributed, we can work out that the variance of Y is

$$\text{Var}(Y) = \text{Var}(W_1 X_1 + W_2 X_2 + \cdots + W_n X_n) = n\text{Var}(W_i)\text{Var}(X_i)$$

Or in words: the variance of the output is the variance of the input, but scaled by $n\text{Var}(W_i)$. So if we want the variance of the input and output to be the same, that means $n\text{Var}(W_i)$ should be 1. Which means the variance of the weights should be

$$\text{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$

Image 08: Weight Initialization (Variance Derivation)

- Assuming the inputs have unit variance, to maintain unit variance in the forward propagation, the weights should be initialized to have a variance of one over d_in.
- To maintain unit variance in the backward propagation, the weights should be initialized to have a variance of one over d_out.

Let X have a uniform distribution on (a, b). The density function of X is

$f(x) = \frac{1}{b-a}$ if $a \leq x \leq b$ and 0 elsewhere

The the mean is given by

$$E[X] = \int_a^b \frac{x}{b-a} dx = \frac{b^2 - a^2}{2(b-a)} = \frac{b+a}{2}$$

The variance is given by $E[X^2] - (E[X])^2$

$$E[X^2] = \int_a^b \frac{x^2}{b-a} dx = \frac{b^3 - a^3}{3(b-a)} = \frac{b^2 + ba + a^2}{3}$$

The required variance is then

$$\frac{b^2 + ba + a^2}{3} - \frac{(b+a)^2}{4} = \frac{(b-a)^2}{12}$$

Image 09: Weight Initialization (Variance Derivation)

Continuous Uniform (a,b) has a mean $(a+b)/2$ and variance as $((b-a)^2)/12$

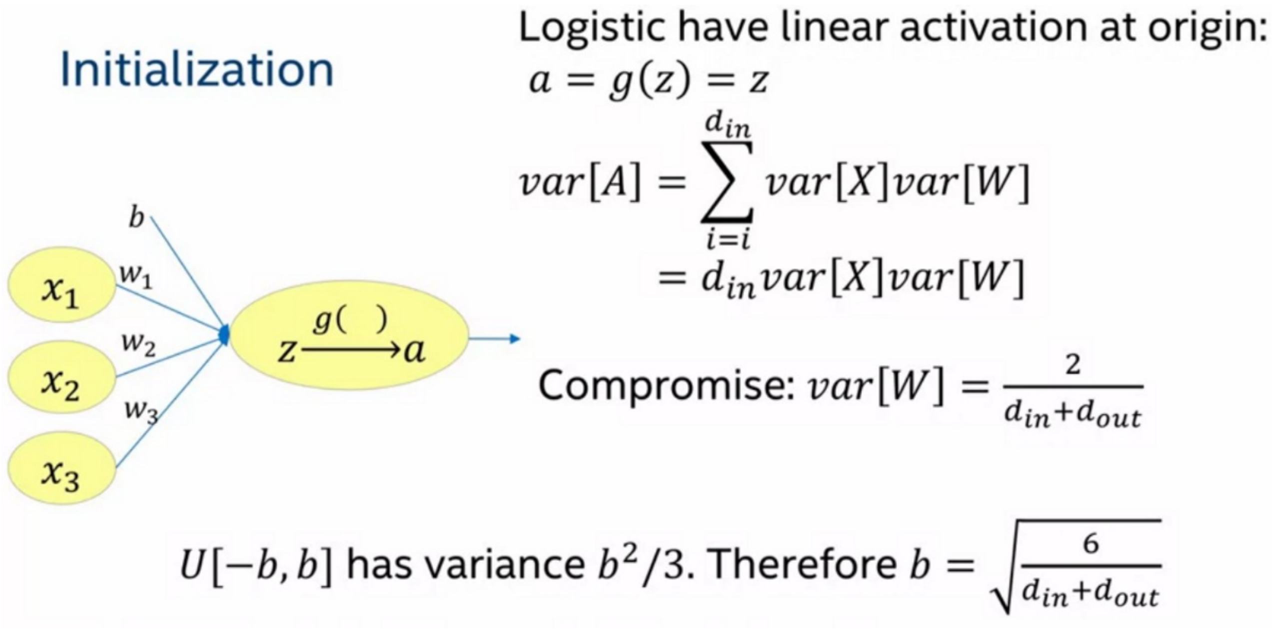


Image 10: Weight Initialization (Weight Derivation)

To get value of b, equate Var(b) to weight and we calculate.

FOR ***RELU***

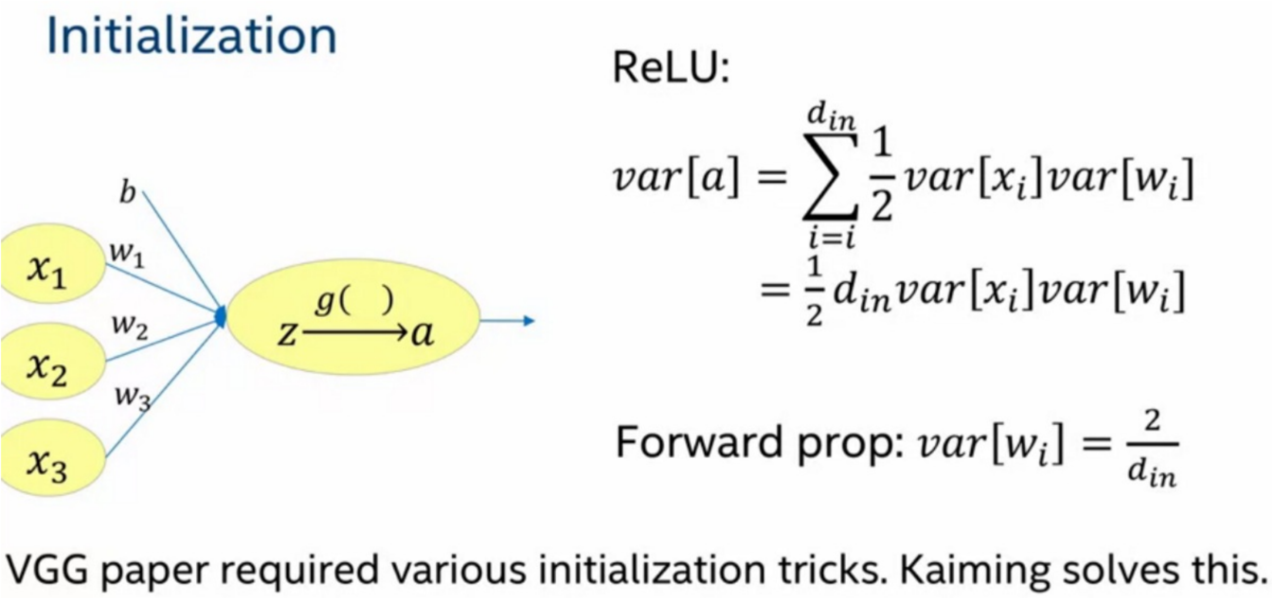


Image 11: Weight Initialization (RELU)

- variance of activation is half of what it was for sigmoid because RELU zeroes out the negative inputs.
- Assuming, unit variance in the inputs, to maintain unit variance in activations, the weights should be initialized to have variance of 2/d_in.

Question

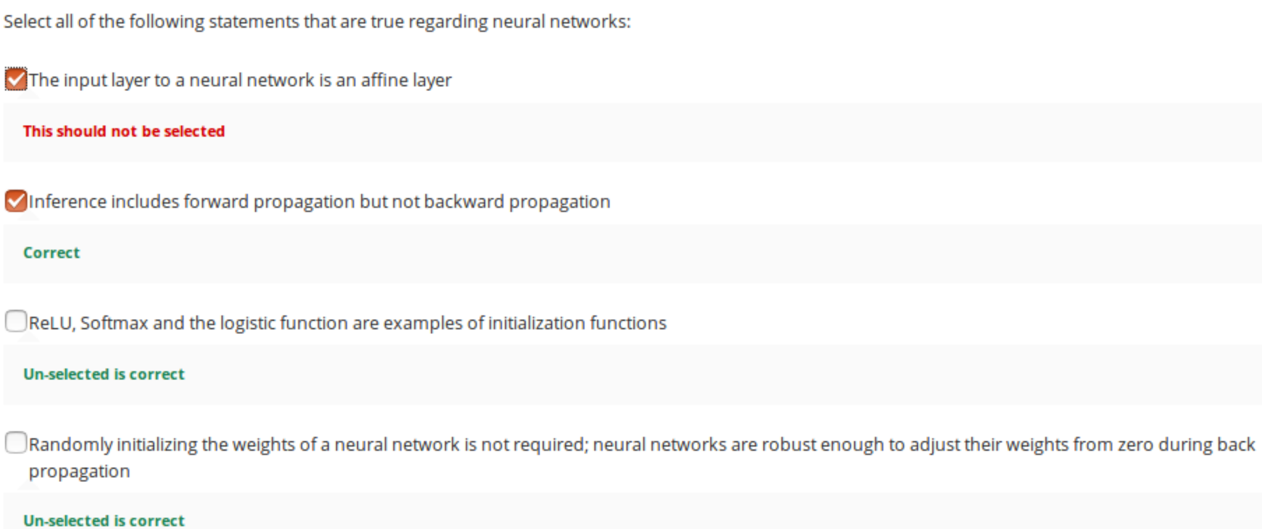


Image 12: Question — General

> **Optimization**
>
> **Gradient Descent —** Used to find local minimum of objective function. Which direction to move — When slope negative, go right. Slope +ve, go left i.e. you move in direction opposite to gradient. Cost fn. must have property of differentiable.

Gradient Descent has some issues for deep learning.

- It requires summing overall the cost of training samples & no. of training samples ca be millions in no..

- It converges to poor local minima. **Theory1 (Saddle Point):** Training stops @ sub-optimal value. **Theory2 (Sharp Minimums):** Optimization space has many sharp minimums, it don't explore optimization space and end to sharp minimum. Sharp Minimum don't generalize well because cost of validation on test dataset may be much different than the cost on training dataset.

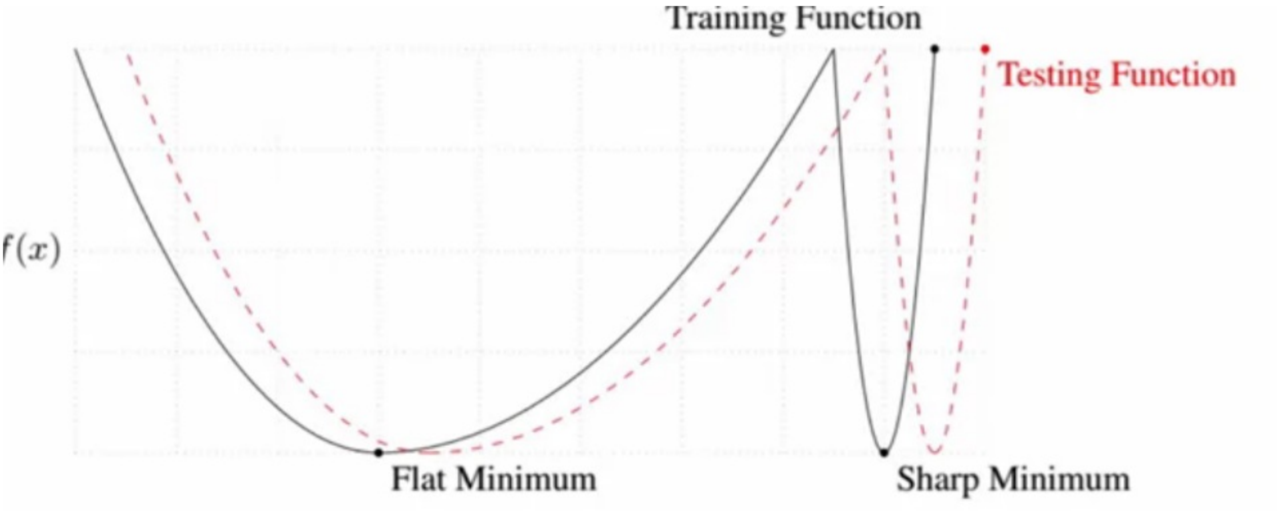


Image 13: Optimization

- **SGD (Stochastic Gradient Descent):** Mini-batch Gradient Descent. Solves Saddle point issue as global gradient is zero but not batch gradient descent. For each batch, costs are aggregated, weight are updated. Here, one epoch is M iterations i.e. M = N/batch_size
- **AdaGrad** — Dynamic Learning rate, normalize it by dividing all gradients. Take large steps when gradients small (during saddle point), small steps when gradients are large.
- **RMS PROP** — Instead all gradients, take only some gradients (few iterations) for normalize

QUIZ — 01

1. We have a dataset containing 90,000 black and white images of objects that are 64 by 64 pixels in size. Our goal is to classify the images into three classes: animals, vehicles and a class that represents everything else.

   If an MLP is used for this task, how many input units are there?

   4096

2. We have a dataset containing 90,000 black and white images of objects that are 64 by 64 pixels in size. Our goal is to classify the images into three classes: animals, vehicles and a class that represents everything else.

   If an MLP is used for this task, how many output units are there?

   3

Image 14: Quiz 1.1

3. We have a dataset containing 90,000 black and white images of objects that are 64 by 64 pixels in size. Our goal is to classify the images into three classes: animals, vehicles and a class that represents everything else.

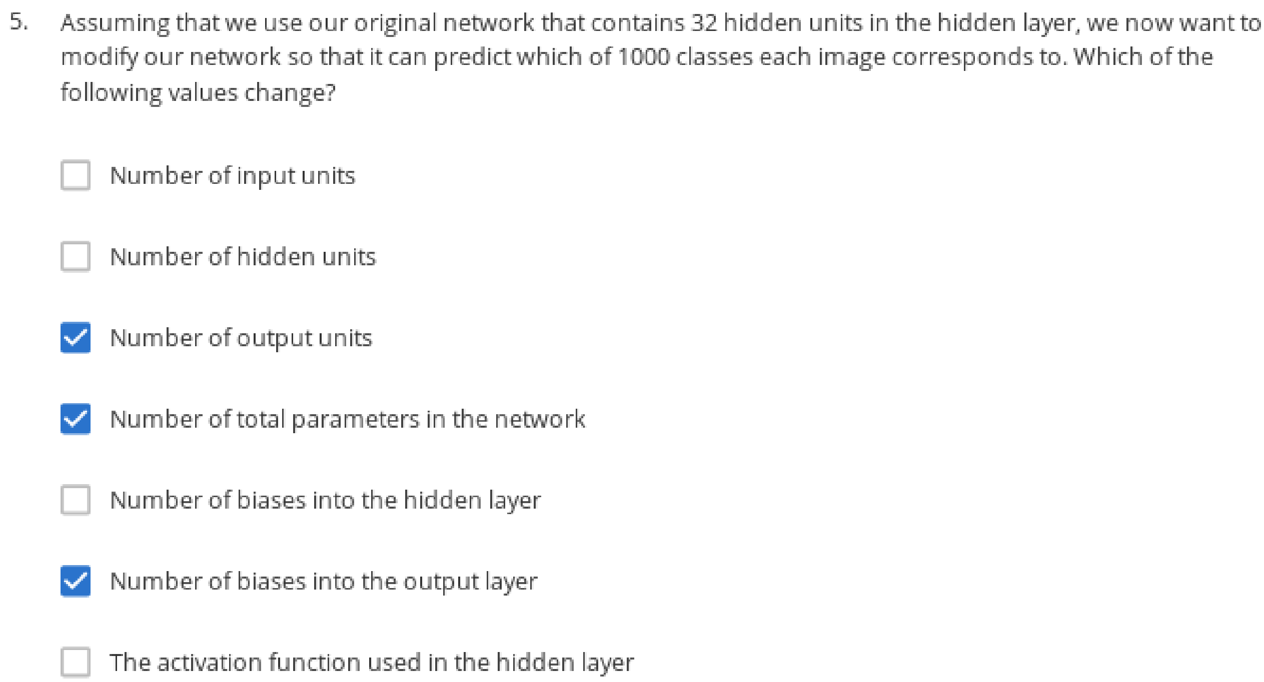
   If this MLP contains one hidden layer containing 32 units, how many total parameters, including weights and biases, are in the network?

   131203

4. Using the same dataset as above, we now change our network so that there are 64 hidden units in the hidden layer. Which of the following values change?

   - [ ] Number of input units
   - [ ] Number of output units
   - [x] Number of total parameters in the network
   - [x] Number of biases into the hidden layer
   - [ ] Number of biases into the output layer
   - [ ] The activation function used in the hidden layer

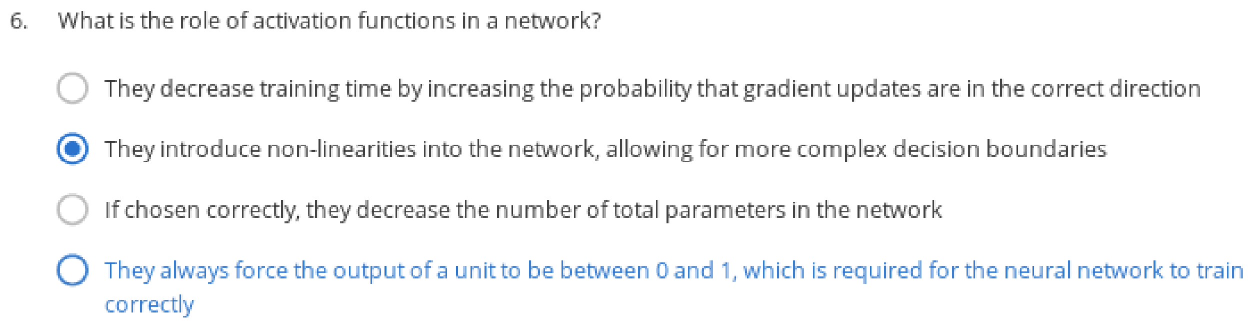
Image 15: Quiz 1.2



5. Assuming that we use our original network that contains 32 hidden units in the hidden layer, we now want to modify our network so that it can predict which of 1000 classes each image corresponds to. Which of the following values change?

- [ ] Number of input units
- [ ] Number of hidden units
- [x] Number of output units
- [x] Number of total parameters in the network
- [ ] Number of biases into the hidden layer
- [x] Number of biases into the output layer
- [ ] The activation function used in the hidden layer

Image 16: Quiz 1.3



6. What is the role of activation functions in a network?

- ( ) They decrease training time by increasing the probability that gradient updates are in the correct direction
- (•) They introduce non-linearities into the network, allowing for more complex decision boundaries
- ( ) If chosen correctly, they decrease the number of total parameters in the network
- ( ) They always force the output of a unit to be between 0 and 1, which is required for the neural network to train correctly

Image 17: Quiz 1.4

Useful resource Link

- https://link.springer.com/chapter/10.1007%2F978-3-642-35289-8_25
- **Visualize back prop:** https://www.youtube.com/watch?v=7HZk7kGk5bU
- *Adam: A Method for Stochastic Optimization*: https://arxiv.org/abs/1412.6980