Sol: 3

(a) Mini batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and efficiency of batch gradient descent. ~~It is the most common~~

~~The~~ Advantages :-

- The model update frequency is higher than batch gradient descent with which allows for a more robust convergence, avoiding local minima.
- Also, the batched update provides a computationally more efficient process.
- Batching, allows both the efficiency of not having all training data in memory (case when batch size = 1 or without mini batch) and algorithm implementations.

(b) A partial solution to the vanishing / Exploding gradients in NN is better or more careful choice of the random initialization of weights.

Suppose we have symmetrically initialized our weights.

$w[l] = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} > \text{Identity}$ l: hidden layer number.

then if we ignore biases for a minute,

$y' = w[L]\, w[L-1] \dots w[2]\, w[1]\, X$

~~so~~ $y' = w[L] \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{(L-1)} X = (1.5)^\wedge L$ (which will be very large) (exploding gradients)

also, if $w[l] = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix} < \text{Identity}$.

$y' = w[L] \begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix}^{(L-1)} X = (0.9)^\wedge L$ (which will be very smaller) (vanishing gradients)

(c) Most oftenly used regularization technique is L2 regularization

$J(w,b) = (1/m) * \text{Sum}\left(L(y(i), y'(i))\right) + \text{lambda}/2m * \text{sum}(|w[i]|^\wedge 2)$

lambda: regularization parameter.

(i) If lambda is too large – a lot of w's will be close to zeros which will make the NN simpler.

(ii) If lambda is good enough, it will just reduce some weights that makes the neural network overfit. Also, it will just make some of tanh activations roughly linear which will prevent overfitting.

Other methods to deal with overfitting :- Collect More data or try a different model

(d) Batch Normalization

~~We generally~~

Before training the model, we normalize the input by subtracting the mean and dividing by variance. This helped a lot for the shape of the cost function and for reaching the minimum point faster.

So, for any hidden layer, we can normalize A[l] to train W[l+1] and b[l+1] faster.

The batch normalization reduces the problem of input values changing (shifting).

Sol: 4

The number of parameters here (without including bias terms) will be = $(\cancel{3 \times 3) \times (3})$

$(fh \times fw) \times (\text{no. of input channels}) \times (\text{no. of output channels})$

$= 3 \times 3 \times 3 \times 8 = 216$

Input image = $N_1 \times N_2 \times 3$. N_1: height N_2: width.

So, output image dimensions : $O_H = (N_1 - fh + 2P)/s + 1$

So, $O_H = (N_1 - 3 + 2)/2 + 1$

$O_H = (N_1 - 1)/2 + 1 = \left(\frac{N_1 + 1}{2}\right)$

$O_W = (N_2 - fw + 2P)/s + 1$

$= (N_2 - 3 + 2)/2 + 1$

$O_W = (N_2 - 1)/2 + 1$

Output image = $O_H \times O_W \times (\text{no. of output channels})$

$= O_H \times O_W \times 8$

Classification algorithms such as logistic regression learn a probability distribution over classes conditioned on the input instead of directly producing an output y because

~~(a) After producing~~

After using different activations functions to ~~be~~ scale down the output values of our algorithm to (0,1) is more efficient and more error free. Introducing loss function which would directly classify the output to either 0 & 1 (in binary classification) will introduce large scale errors into the model, and so accuracy of the model will be low.

~~After~~ Using loss functions ~~such~~ and activations function such that the output is a probability, we can easily classify them of the basis of what they are more likely to be..

Q-2 Stochastic Gradient Descent with Momentum.

SGD with momentum is a method which helps accelerate gradient vectors in the right directions, thus leading to find fast ~~conversions~~ convergence.

For fast convergence of our model, what we want to do with the data is, we want some kind of 'moving' average which would 'denoise' the data and bring it closer to the original function. This can be achieved by using ~~an~~ ~~expon~~ the exponentially weighted averages, which are defined as $V_t = \beta V_{t-1} + (1-\beta) S_t$

(V: new sequence; S: old sequence)

~~For~~ We can use this exponential weighted averages to average over our gradients. Momentum is the moving average of our gradients. Our update equation can be written as

$$\boxed{\begin{aligned} V_t &= \beta V_t + (1-\beta)\nabla_w L(W, x, y) \\ W &= W - \alpha V_t \end{aligned}}$$

L: Loss function

∇w: gradient w.r.t weight

alpha: learning rate.

Momentum in Physics refers to a push of the ~~effe~~ something ~~th~~ possed by the body due to its motion and it's mass. The momentum here accelerates the Gradient descent algorithm ~~at~~ so, the convergence is achieved faster and more smoothly.