Problem 1:

(a)

Solution: Large W causes Wx to be large. When Wx is large, the gradient is small for sigmoid activation function. Hence, we will encounter the vanishing gradient problem.

(b)

Solution: We should use Recall here, as any false-negatives from the system are much more harmful.

(c)

Solution: **Performance wouldn't be affected**. The positioning of the Batch-Norm *immediately after* the Conv ensures that the BN layer can learn any bias previously learnt by the Conv layer. Indeed, original implementations of the https://arxiv.org/abs/1512.03385 had their convolutional biases switched off.

(d)

Solution: The model is suffering from a bias problem.
Increasing the amount of data reduces the variance, and is not likely to solve the problem.
A better approach would be to decrease the bias of the model by maybe adding more layers/ learnable parameters. It is possible that training converged to a local optimum. Training longer/using a better optimizer/ restarting from a different initialization could also work.

(e)

Solution: 1x1 convolution reduces the size of feature maps before sending them to layers with more expensive filters (3x3, 5x5). Using 1x1 convolution, we can stack more layers and allocate them on the training device. Check the inception paper for more details.

(f)

Solution: Simply normalizing the inputs of a layer may change what the layer can represent. For instance, normalizing the inputs of a sigmoid function would constrain them to the linear regime of the nonlinearity. Inserting learnable transformation in the batch norm layer restores the representation power of the network.

Problem 2:

(a)

$$\frac{\partial J}{\partial \mu_i} = \sum_{y_j=i} (\mu_i - X_j).$$

Setting that derivative to zero gives us

$$\begin{aligned} \sum_{y_j=i} \mu_i &= \sum_{y_j=i} X_j; \\ n_j \mu_i &= \sum_{y_j=i} X_j; \\ \mu_i &= \frac{1}{n_j} \sum_{y_j=i} X_j, \end{aligned}$$

which indeed is the mean of the sample points assigned to cluster *i*.

(b)

$$\begin{aligned} \frac{\partial L}{\partial \mu_1} &= \frac{\partial}{\partial \mu_1} \sum_{x_i \in S_1} (x_i - \mu_1)^\top (x_i - \mu_1) \\ &= \sum_{x_i \in S_1} 2(\mu_1 - x_i). \end{aligned}$$

Therefore the update formula is

$$\mu_1 \leftarrow \mu_1 + \epsilon \sum_{x_i \in S_1} (x_i - \mu_1).$$

(Note: writing 2ϵ instead of ϵ is fine.)

(c)

In the standard algorithm, we assign $\mu_1 \leftarrow \sum_{x_i \in S_1} \frac{1}{|S_1|} x_i$.

Comparing to the answer in (1), we set $\sum_{x_i \in S_1} \frac{1}{|S_1|} x_i = \mu_1 + \epsilon \sum_{x_i \in S_1} (x_i - \mu_1)$ and solve for ϵ.

$$\begin{aligned} \sum_{x_i \in S_1} \frac{1}{|S_1|} x_i - \sum_{x_i \in S_1} \frac{1}{|S_1|} \mu_1 &= \epsilon \sum_{x_i \in S_1} (x_i - \mu_1) \\ \sum_{x_i \in S_1} \frac{1}{|S_1|} (x_i - \mu_1) &= \epsilon \sum_{x_i \in S_1} (x_i - \mu_1). \end{aligned}$$

Thus $\epsilon = \frac{1}{|S_1|}$.

Problem 3:

(a)

$$J = CE(y, \hat{y}) = -\sum_i y_i \, log(\hat{y})$$

and

$$\hat{y} = \frac{exp(\theta_i)}{\sum_j exp(\theta_j)}$$

We know that $y_j = 0$ for $j \neq k$ and $y_k = 1$, so:

$$J = CE(y, \hat{y}) = -log(\hat{y}_k) = -log\left(\frac{exp(\theta_k)}{\sum_j exp(\theta_j)}\right)$$

$$= -\theta_k + log\left(\sum_j exp(\theta_j)\right)$$

$$\rightarrow \ \delta_1 = \frac{\partial J}{\partial \theta} = \ -\frac{\partial \theta_k}{\partial \theta} + \frac{\partial}{\partial \theta} log\left(\sum_j exp(\theta_j)\right)$$

Use the fact that $\frac{\partial \theta_k}{\partial \theta_k} = 1$ and $\frac{\partial \theta_k}{\partial \theta_j} = 0$ for $j \neq k$ to show that:

$$\frac{\partial \theta_k}{\partial \theta} = y$$

For the second part we write out the derivative for each individual element of θ and use the chain rule to get:

$$\frac{\partial}{\partial \theta_i} log\left(\sum_j exp(\theta_j)\right) = \frac{exp(\theta_i)}{\sum_j exp(\theta_j)} = \hat{y}_i$$

Hence,

$$\delta_1 = \frac{\partial J}{\partial \theta} = \hat{y} - y$$

(b)

$$\delta_2 = \frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial \theta}\frac{\partial \theta}{\partial h_2}\frac{\partial h_2}{\partial z_2} = \delta_1 \circ 1\{z_2 > 0\}$$

(c)

$$\delta_3 = \frac{\partial J}{\partial d} = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial d} + \frac{\partial J}{\partial \theta}\frac{\partial \theta}{\partial d} = W_2^T \delta_2 + \delta_1$$

(d)

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial d}\frac{\partial d}{\partial h_1}\frac{\partial h_1}{\partial x} + \frac{\partial J}{\partial d}\frac{\partial d}{\partial x} = W_1^T\left(\delta_3 \circ 1\{z_1 > 0\}\right) + \delta_3$$

Problem 4:

(a) Exploding gradients is a problem of large increase in the norm of the gradient during training. Backpropagation computes gradients using the chain rule and accumulating gradients through layers involves multiplying a large number of gradients. When all gradients are greater than 1, this can result in very large gradients. This in turn results in large updates to the network weights which may even lead to numerical overflow, hence the term "exploding gradients". This gets worse as the length of the sequence increases.

(b) In feed forward networks the gradients wrt earlier layers depend on the weight matrices after it. Since the weight matrices are different, larger and smaller weight matrices might balance out to not let the gradients vanish. RNNs on the other hand share weights across time steps, so the gradients of wrt earlier time steps will depend on the same matrix multiplied with itself many times. If the weight matrices increase the magnitude of a gradient, we get the "exploding gradient" problem, whereas if they decrease the magnitude of a gradient, we get the "vanishing gradient" problem.

(c) Some of the common ways to solve the "vanishing gradient problem" are:

(i) Use LSTM or GRU units

(ii) Use residual networks

(iii) Use ReLU instead of sigmoid activation function

(iv) Use fewer layers

(v) Use batch normalization

(vi) Truncate backpropagation

(d) We have

$$h_m = \sigma(\theta h_{m-1} + x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \theta \sigma'(\theta h_{m-1} + x_m)$$

It can easily be shown that the derivative of the sigmoid function is

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Since the sigmoid function is bounded and has outputs in $(0, 1)$, one can conclude that maximum value for $\sigma'(x)$ is 0.25. This can be seen from the graph or by taking the second derivative of this expression. Now let's see what happens when $k \to \infty$.

$$\begin{aligned}
\frac{\partial h_{m+k}}{\partial h_m} &= \prod_{i=m+k}^{m+1} \frac{\partial h_i}{\partial h_{i-1}} \\
&= \prod_{i=1}^{k-1} \theta\sigma'(\theta h_{i-1} + x_i) \\
&\leq \prod_{i=1}^{k-1} 0.25\theta \\
&= 0.25^{k-1}\theta^{k-1} \\
&\to 0 \quad \text{as} \quad k \to \infty \quad (\text{since } |\theta| < 1)
\end{aligned}$$

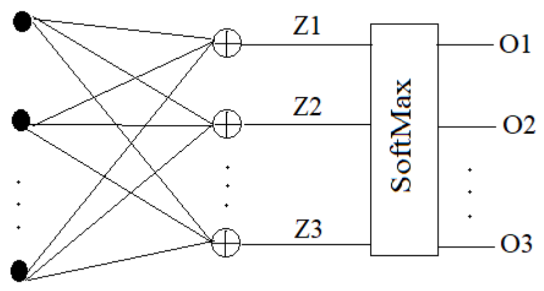
Problem 5:

الف)

Layer	Activation Map Dimension	Number of Weights	Number of Biases
Input	128*128*3	0	0
CONV-9-32	120*120*32	32*(9*9*3)	32
Pool-2	60*60*32	0	0
CONV-5-64	56*56*64	64(5*5*32)	64
Pool-2	28*28*64	0	0
CONV-5-64	24*24*64	64(5*5*64)	64
Pool-2	12*12*64	0	0
FC-6	6	6(12*12*64)	6

ب)

از تابع فعال ساز softmax با توجه به اینکه با یک مسئله کلاس بندی مواجه هستیم و شبکه ۶ خروجی دارد، استفاده می کنیم.



$$O_i = \frac{\exp(Z_i)}{\sum_{j=1}^{6} \exp(Z_j)}$$

این تابع عددی در بازه ۰ تا ۱ برمیگرداند و احتمال وقوع هر کدام از ۶ کلاس را در خروجی نشان می دهد. محدود بودن خروجی به یک بازه از یک طرف باعث جلوگیری از بزرگ شدن خروجی نورون ها شده و در نتیجه سربار محاسباتی در شبکه ایجاد نمی کند و از طرف دیگر یک تابع غیرخطی است که باعث می شود شبکه بتواند پترن های غیرخطی را نیز یاد بگیرد، همچنین این تابع مشتق پذیر است و برای برگشت گرادیان در شبکه مشکلی ایجاد نمی کند.

ج)

اضافه کردن لایه های بیشتر به شبکه تا زمانی که خطای validation در روند آموزش شبکه افزایش پیدا نکند می تواند مدل دقیق تری به دست دهد. اما اگر تعداد لایه های شبکه نسبت به داده ی آموزشی که در دست داریم تعادل نداشته باشد شبکه یک مدل پیچیده ایجاد خواهد کرد و در نتیجه با مشکل Overfit شدن شبکه مواجه خواهیم شد.

د)

از لایه Pooling برای کاهش سایز Feature map ها استفاده می شود و به نوعی باعث downsample شدن ورودی های لایه می شود و در نتیجه از این طریق ویژگی های مهم را نگه داشته و بقیه ویژگی ها را دور می ریزد. با این کار تعداد پارامترهای شبکه کاهش پیدا کرده و به نوعی عاملی برای کمتر شدن احتمال overfit شدن شبکه نیز هست، ضمن اینکه حافظه ی مورد نیاز شبکه نیز از این طریق کاهش می یابد.

ه)

استفاده از Stride بزرگتر از یک تأثیری مشابه استفاده از لایه های Max Pooling دارد.

Problem 6:

(a) A node h_i is dead if the argument of its ReLU function is always negative; this means for any input $\mathbf{x}$ we have $W_i^{(1)} \cdot \mathbf{x} + b_i^{(1)} \leq 0$, which can happen due to e.g., a large, negative $b_i^{(1)}$.

(b)

$$\frac{\partial l}{\partial b_i^{(1)}} = \frac{\partial l}{\partial y}\frac{\partial y}{\partial h_i}\frac{\partial h_i}{\partial b_i^{(1)}} = 1 \cdot W_i^{(2)} \cdot \mathbb{1}_{\{W_i^{(1)}\cdot\mathbf{x}+b_i^{(1)}>0\}}$$

$$\frac{\partial l}{\partial W_{ij}^{(1)}} = \frac{\partial l}{\partial y}\frac{\partial y}{\partial h_i}\frac{\partial h_i}{\partial W_{ij}^{(1)}} = 1 \cdot W_i^{(2)} \cdot x_j \cdot \mathbb{1}_{\{W_i^{(1)}\cdot\mathbf{x}+b_i^{(1)}>0\}}$$

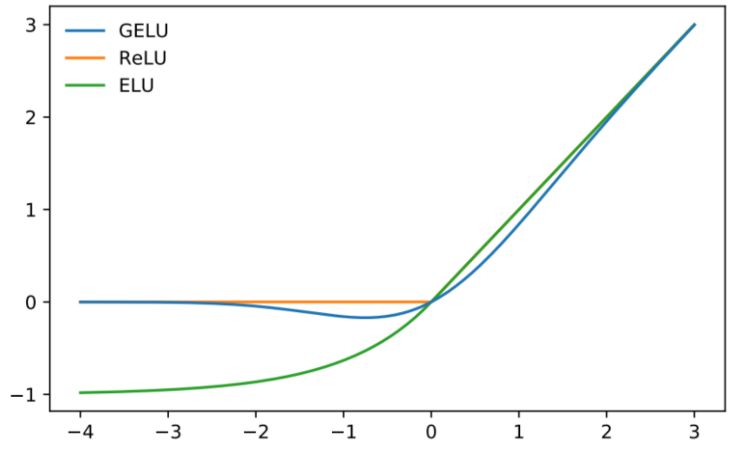
(c) Both gradients involve the term $\mathbb{1}_{\{W_i^{(1)}\cdot\mathbf{x}+b_i^{(1)}>0\}}$. If h_i is dead, it means we have $W_i^{(1)} \cdot \mathbf{x} + b_i^{(1)} \leq 0$ for all inputs. Then this term $\mathbb{1}_{\{W_i^{(1)}\cdot\mathbf{x}+b_i^{(1)}>0\}}$ is guaranteed to be zero for all inputs and thus, the gradient of the neuron's parameters will be zero for all inputs and parameters will not change in gradient-based learning. So h_i will remain "dead."

(d) Some of the common methods to address the problem of dead neurons in ReLU are:

(i) Leaky ReLU

(ii) ELU

(iii) GeLU


GELU
ReLU
ELU