[redacted]

Question I . . . . **(28 Marks)**

Use the space provided for answering short questions. Be precise while answering the short questions.

(a) **(2 Marks)** What nonlinearity is preferred in deep networks and why ?

Leeky Relu is used/preferred for deep Networks because it gives α if value is less than 0 and ~~(x)~~ dx if value is greater than 0.

2

(b) **(2 Marks)** For what two reasons, analytical gradient is preferred over the numerical gradient.

[redacted]

(c) Given following response of 4-class SVM classifier on a test example x, *i.e.* $[-9, 2, 0.001, 5]$:

i. **(1 Mark)** to which example class x belongs.

ii. **(2 Marks)** what will be the score of Softmax classifier ?

SVM = max(0,

[redacted]

score classifier.

$-9 \rightarrow 4.50$

$2 \rightarrow -1.00$

$.001 \rightarrow$ ~~-5.00~~

$\rightarrow -8.501$

(d) **(2 Marks)** Prove that softmax loss contribution from a single example $L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$ can be equivalently written as $L_i = -f_{y_i} + \log\sum_j e^{f_j}$.

(e) **(2 Marks)** Why we need to normalize the randomly initialized weights by the square-root of fan-in?

[redacted]

$\frac{w}{\sqrt{n}}$

(f) **(1 Mark)** Consider you are training a softmax classifier for 7 classes and you have initialized your weights randomly what loss value you should expect from your code.

[redacted]

ok

(g) **(1 Mark)** Consider that now you are training a multi-class SVM classifier for 7 classes and you have initialized your weights randomly what loss value you should expect from your code.

[redacted]

(h) **(1 Mark)** What is the best strategy of checking whe[redacted]network system is working fine.

[redacted]

(i) **(2 Marks)** What it means by "vanishing gradients" and why are they considered a problem during optimization?

[redacted]

(j) **(3 Marks)** How Dropout is applied during testing, write proper code. What does its application signifies?

[redacted]

(k) **(2 Marks)** Differentiate between gradient descent, stochostic gradient descent and batch gradient descent, which is more preferred one and why?

[redacted]

(l) *(1 Mark)* During gradient descent, if your cost function value start increasing with epochs, what might be the reason. How can you rectify that ?

[redacted]

(m) *(1 Mark)* During gradient descent, if your cost function value start increasing with epochs but after some epochs it gets stuck and no improvement is seen over the epochs, what might be the reason. How can you rectify that ?

It can be due to the local minma, where it may stuck. It can be solved by momentum Gradient Descent.

(n) **(2 Marks)** During back-propogation while computing derivatives *w.r.t.* weights why we have to compute sum of derivatives across the examples.

The weights are distrobuted across the examples that's why we use sum of derivaties of all weights for a particuler [illegible] example and vice verse.

Continue...

$\alpha \quad x<0$

$\alpha x \quad x>0$

Question II ....(10 Marks)

One of the way of limiting number of weights in a neural network is weight sharing among the neurons. This not only allows in reduction of possible parameters but also help to model the local spatial structure in the input. Now given the following architecture (See Figure 1) of a neural network, find the expression for the derivatives of the shared-input weights $w_{11}^{(1)}$. Here $a_i^{(2)}$ and $a_i^{(3)}$ are simple affine combinations of weights and inputs followed by ReLU units, that each $a_i^{(l)}$ first calculates the dot product of inputs and weights and apply the ReLU non-linearity on its output.

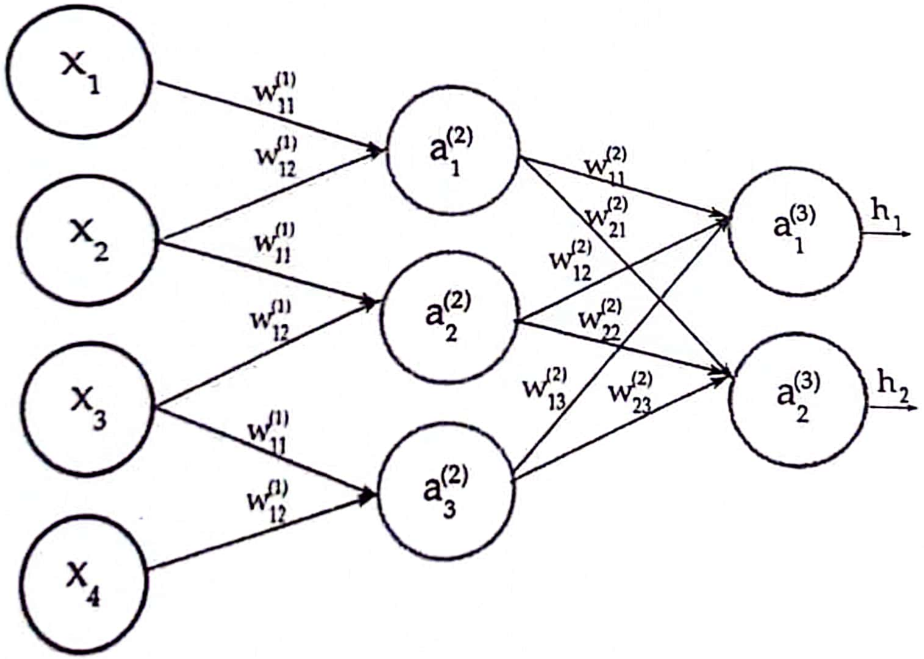

Figure 1: *Here in the first layer weights* ($w_{11}^{(1)}$ and $w_{12}^{(1)}$ are shared among neurons $a_i^{(2)}$) whereas second layer is a simple affine (fully connected) layer which means each neuron has its own weights.

Question III..(10 Marks)

Given that following functions have already been defined, now write code using these functions for building a 3-layer neural network with multi-class SVM loss (architecture diagram is given in Figure 2). Your code should be able to compute the class scores as well as should produce the derivatives *w.r.t.* all the weights. Store each layer weights and biases derivatives in separate variables preferably in a dictionary. Assume there are 50,000 examples from 3 classes each having 30 features and there are 20 neurons in each layer 1 and layer 2.

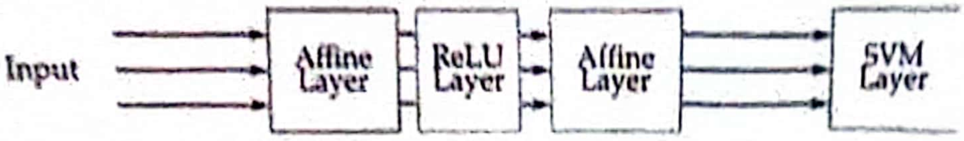

Figure 2: An example of multi-layer architecture

```
def affine_forward(x, w, b):
    """
    Computes the forward pass for an affine (fully-connected) layer.

    Inputs:
    x - Input data, of shape (N, d_1, ..., d_k)
    w - Weights, of shape (D, M)                    [redacted]
    b - Biases, of shape (M,)

    Returns a tuple of:
    - out: output, of shape (N, M)
    - cache: (x, w, b)
    """
def affine_backward(dout, cache):
    """
    Computes the backward pass for an affine layer.

    Inputs:
    - dout: Upstream derivative, of shape (N, M)    [redacted]
    - cache: Tuple of:
      - x: Input data, of shape (N, d_1, ... d_k)
      - w: Weights, of shape (D, M)

    Returns a tuple of:
    - dx: Gradient with respect to x, of shape (N, d1, ..., d_k)
    - dw: Gradient with respect to w, of shape (D, M)
    - db: Gradient with respect to b, of shape (M,)
    """
def relu_forward(x):
    """
    Computes the forward pass for a layer of rectified linear units (ReLUs).

    Input:
    - x: Inputs, of any shape                       [redacted]

    Returns a tuple of:
    - out: Output, of the same shape as x
    - cache: x
    """
```

```
def relu_backward(dout, cache):
    """
    Computes the backward pass for a layer of rectified linear units (ReLUs).

    Input:
    - dout: Upstream derivatives, of any shape
    - cache: Input x, of same shape as dout

    Returns:
    - dx: Gradient with respect to x
    """
def svm_loss(x, y):
    """
    Computes the loss and gradient using for multiclass SVM classification.

    Inputs:
    - x: Input data, of shape (N, C) where x[i, j] is the score for the jth cl
      for the ith input.
    - y: Vector of labels, of shape (N,) where y[i] is the label for x[i] and
      0 <= y[i] < C

    Returns a tuple of:
    - loss: Scalar giving the loss
    - dx: Gradient of the loss with respect to x
    """
```