

CS230: Deep Learning

Spring Quarter 2018

Stanford University

Midterm Examination

180 minutes

	Problem	Full Points	Your Score
1	Multiple Choice	14	
2	Short Answers	28	
3	Loss functions	17	
4	Batch Norm	15	
5	"Who let the cars in?"	4	
6	Numpy coding	10	
7	X-Network	23	
8	Practical case Study	11	
Total		122	

The exam contains 33 pages including this cover page.

- This exam is **closed book i.e. no laptops, notes, textbooks, etc. during the exam**. However, you may use one A4 sheet (front and back) of notes as reference.
- In all cases, and especially if you're stuck or unsure of your answers, **explain your work, including showing your calculations and derivations!** We'll give partial credit for good explanations of what you were trying to do.

Name: _____

SUNETID: _____@stanford.edu

The Stanford University Honor Code:

I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

Signature: _____

Question 1 (Multiple Choice Questions, 14 points)

For each of the following questions, circle the letter of your choice. There is only ONE correct choice unless explicitly mentioned. No explanation is required.

- (a) **(2 points)** Which of the following is true about max-pooling?
- (i) It allows a neuron in a network to have information about features in a larger part of the image, compared to a neuron at the same depth in a network without max pooling.
 - (ii) It increases the number of parameters when compared to a similar network without max pooling.
 - (iii) It increases the sensitivity of the network towards the position of features within an image.
- (b) **(2 points)** In order to backpropagate through a max-pool layer, you need to pass information about the positions of the max values from the forward pass.
- (i) True
 - (ii) False
- (c) **(2 points)** Consider a simple convolutional neural network with one convolutional layer. Which of the following statements is true about this network? (Check all that apply.)
- (i) It is scale invariant.
 - (ii) It is rotation invariant.
 - (iii) It is translation invariant.
 - (iv) All of the above.
- (d) **(2 points)** You are training a Generative Adversarial Network to generate nice iguana images, with mini-batch gradient descent. The generator cost $\mathcal{J}^{(G)}$ is extremely low, but the generator is not generating meaningful output images. What could be the reason? (Circle all that apply.)
- (i) The discriminator has poor performance.
 - (ii) Your generator is overfitting.
 - (iii) Your optimizer is stuck in a saddle point.
 - (iv) None of the above.

- (e) **(2 points)** Mini-batch gradient descent is a better optimizer than full-batch gradient descent to avoid getting stuck in saddle points.
- (i) True
 - (ii) False
- (f) **(2 points)** Using "neural style transfer" (as seen in class), you want to generate an RGB image of the Great Wall of China that looks like it was painted by Picasso. The size of your image is 100x100x3 and you are using a pretrained network with 1,000,000 parameters. At every iteration of gradient descent, how many updates do you perform?
- (i) 10,000
 - (ii) 30,000
 - (iii) 1,000,000
 - (iv) 1,030,000
- (g) **(2 points)** You are building a model to predict the presence (labeled 1) or absence (labeled 0) of a tumor in a brain scan. The goal is to ultimately deploy the model to help doctors in hospitals. Which of these two metrics would you choose to use?

(i) $\text{Precision} = \frac{\text{True positive examples}}{\text{Total predicted positive examples}}$

(ii) $\text{Recall} = \frac{\text{True positive examples}}{\text{Total positive examples}}.$

Question 2 (Short Answers, 28 points)

The questions in this section can be answered in 3-4 sentences. Please be short and concise in your responses.

- (a) Consider an input image of shape $500 \times 500 \times 3$. You flatten this image and use a fully connected layer with 100 hidden units.

(i) **(1 point)** What is the shape of the weight matrix of this layer?

(ii) **(1 point)** What is the shape of the corresponding bias vector?

- (b) **(2 points)** Consider an input image of shape $500 \times 500 \times 3$. You run this image in a convolutional layer with 10 filters, of kernel size 5×5 . How many parameters does this layer have?

- (c) **(2 points)** You forward propagate an input x in your neural network. The output probability is \hat{y} . Explain briefly what $\frac{\partial \hat{y}}{\partial x}$ is.

- (d) **(2 points)** Why is it necessary to include non-linearities in a neural network?

- (e) **(2 points)** The universal approximation theorem states that a neural network with a single hidden layer can approximate any continuous function (with some assumptions on the activation). Give one reason why you would use deep networks with multiple layers.

- (f) **(3 points)** Consider a dataset $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ where each example $x^{(i)}$ contains a sequence of 100 numbers:

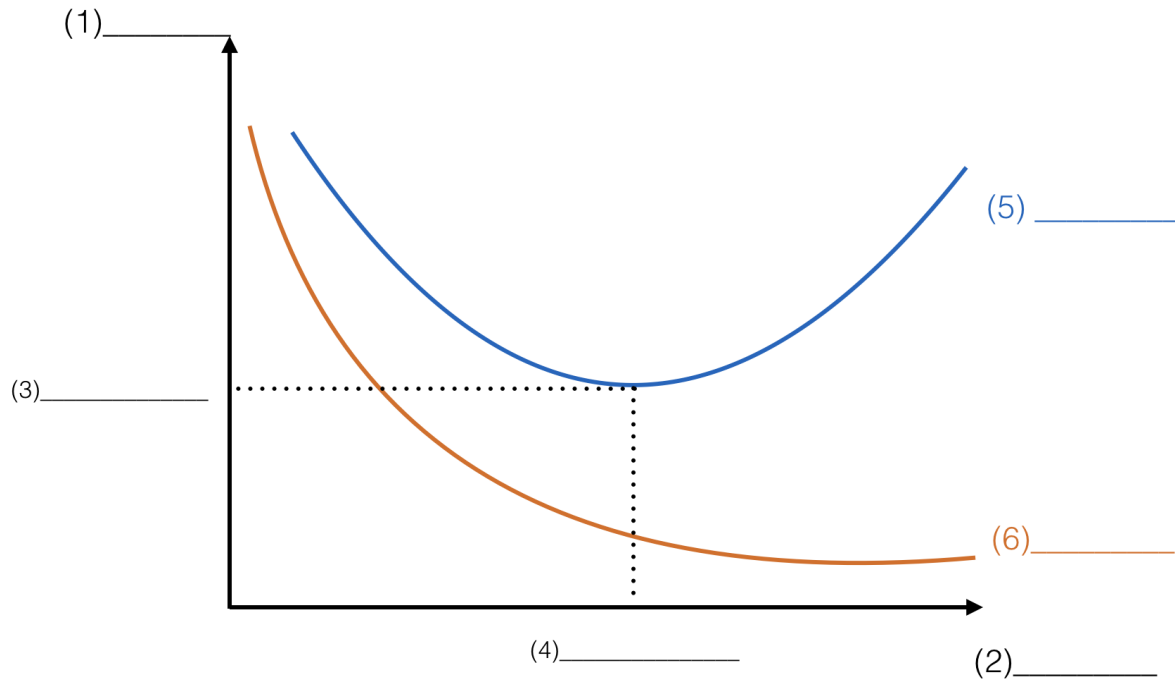
$$x^{(i)} = (x^{(i)<1>}, x^{(i)<2>}, \dots, x^{(i)<100>})$$

You have a very accurate (but not perfect) model that predicts $x^{<t+1>}$ from $(x^{<1>}, \dots, x^{<t>})$. Given $x^{<1>}$, you want to generate $(x^{<2>}, \dots, x^{<100>})$ by repeatedly running your model. Your method is:

1. Predict $\hat{x}^{<2>}$ from $x^{<1>}$
2. Predict $\hat{x}^{<3>}$ from $(x^{<1>}, \hat{x}^{<2>})$
3. ...
4. Predict $\hat{x}^{<100>}$ from $(x^{<1>}, \hat{x}^{<2>}, \dots, \hat{x}^{<99>})$

The resulting $\hat{x}^{<100>}$ turns out to be very different from the true $x^{<100>}$. Explain why.

- (g) **(3 points)** You want to use the figure below to explain the concept of early stopping to a friend. Fill-in the blanks. (1) and (2) describe the axes. (3) and (4) describe values on the vertical and horizontal axis. (5) and (6) describe the curves. Be precise.



- (h) **(2 point)** Look at the grayscale image at the top of the collection of images below. Deduce what type of convolutional filter was used to get each of the lower images. Explain briefly and include the values of these filters. The filters have a shape of (3,3).



- (i) **(1 point)** When the input is 2-dimensional, you can plot the decision boundary of your neural network and clearly see if there is overfitting.

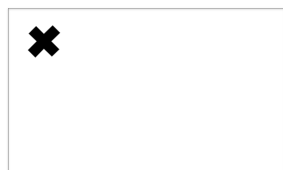
How do you check overfitting if the input is 10-dimensional?

- (j) **(2 points)** What is the advantage of using Inverted Dropout compared to its older version (“Normal” dropout)? Mention the difference of implementation.

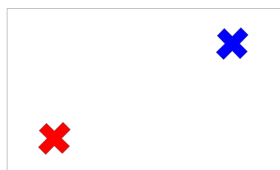
- (k) **(1 point)** Give a method to fight exploding gradient in fully-connected neural networks.

- (l) **(2 points)** You are using an Adam optimizer. Show why the bias correction naturally disappears when the numbers of steps to compute the exponential moving averages gets large.

- (m) Consider the two images below:



(A) Image 1



(B) Image 2

Figure 1: (A) Image 1 is a binary image. Each pixel has value 1 (white) or 0 (black). There’s only a single channel. (B) Image 2 is a color image. Each pixel has value 0 (absence of that color) or 1 (presence of that color), but there are 3 color channels (RGB). The cross on the left is red, and the cross on the right is blue.

You have access to an algorithm (\star) that can efficiently find the position of the maximum value in a matrix.

- (i) **(2 points)** You would like to detect the 3×3 cross in Image 1. Hand-engineer a filter to be convolved over Image 1. Given the output of this "convolved" operation, the algorithm (\star) should give you the position of the cross.
- (ii) **(2 points)** You would like to detect the 3×3 red cross in Image 2 but not the blue cross. Hand-engineer a filter to be convolved over Image 2. Given the output of this "convolved" operation, the algorithm (\star) should give you the position of the red cross.

Question 3 (Loss Functions, 17 points + 3 bonus points)

Equipped with cutting-edge Deep Learning knowledge, you are working with a biology lab. Specifically, you're asked to build a classifier that predicts the animal type from a given image into four ($n_y = 4$) classes: **dog**, **cat**, **iguana**, **mouse**. There's always exactly one animal per image. You decide to use cross-entropy loss to train your network. Recall that the cross-entropy (CE) loss for a single example is defined as follows:

$$\mathcal{L}_{CE}(\hat{y}, y) = - \sum_{i=1}^{n_y} y_i \log \hat{y}_i$$

where $\hat{y} = (\hat{y}_1, \dots, \hat{y}_{n_y})^\top$ represents the predicted probability distribution over the classes and $y = (y_1, \dots, y_{n_y})^\top$ is the ground truth vector, which is zero everywhere except for the correct class (e.g. $y = (1, 0, 0, 0)^\top$ for **dog**, and $y = (0, 0, 1, 0)^\top$ for **iguana**).

- (a) **(2 points)** Suppose you're given an example image of an iguana. If the model correctly predicts the resulting probability distribution as $\hat{y} = (0.25, 0.25, 0.3, 0.2)^\top$, what is the value of the cross-entropy loss? You can give an answer in terms of logarithms.

- (b) **(2 points)** After some training, the model now incorrectly predicts **mouse** with distribution $\langle 0.0, 0.0, 0.4, 0.6 \rangle$ for the same image. What is the new value of the cross-entropy loss for this example?

- (c) **(2 points)** Surprisingly, the model achieves lower loss for a misprediction than for a correct prediction. Explain what implementation choices led to this phenomenon.

- (d) **(2 points)** Given your observation from question (c), you decide to train your neural network with the accuracy as the objective instead of the cross-entropy loss. Is this a good idea? Give one reason. Note that the accuracy of a model is defined as

$$\text{Accuracy} = \frac{(\text{Number of correctly-classified examples})}{(\text{Total number of examples})}$$

- (e) **(3 points)** After tuning the model architecture, you find that *softmax* classifier works well. Specifically, the last layer of your network computes logits $z = (z_1, \dots, z_{n_y})^\top$, which are then fed into the softmax activation. The model achieves 100% accuracy on the training data. However, you observe that the training loss doesn't quite reach zero. Show why the cross-entropy loss can never be zero if you are using a softmax activation.

- (f) **(2 points)** The classifier you trained worked well for a while, but its performance suddenly dropped. It turns out that the biology lab started producing chimeras (creatures that consist of body parts of different animals) by combining different animals together. Now each image can have multiple classes associated with them; for example, it could be a picture of a dog with mouse whiskers, cat ears and an iguana tail! Propose a way to label new images, where each example can simultaneously belong to multiple classes.

- (g) **(2 points)** The lab asks you to build a new classifier that will work on chimeras as well as normal animals. To avoid extra work, you decide to retrain a new model with the same architecture (softmax output activation with cross-entropy loss). Explain why this is problematic.
- (h) **(2 points)** Propose a different activation function for the last layer and a loss function that are better suited for this multi-class labeling task.
- (i) **(Bonus: 3 points extra credit)** Prove the following lower bound on the cross-entropy loss for an example with L correct classes:

$$\mathcal{L}_{CE}(\hat{y}, y) \geq L \log L$$

Assume you are still using softmax activation with cross-entropy loss with ground truth vector $y \in \{0, 1\}^{n_y}$ with L nonzero components.

Question 4 (Batch Norm, 15 points)

In this question, you will explore the importance of good balance between positive and negative labeled examples within a mini-batch, especially when your network includes batch normalization layers.

Recall the batch normalization layer takes values $z = (z^{(1)}, \dots, z^{(m)})$ as input and computes $z_{\text{norm}} = (z_{\text{norm}}^{(1)}, \dots, z_{\text{norm}}^{(m)})$ according to:

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{(\sigma^2) + \epsilon}} \quad \text{where} \quad \mu = \frac{1}{m} \sum_{i=1}^m z^{(i)} \quad (\sigma^2) = \frac{1}{m} \sum_{i=1}^m (z^{(i)} - \mu)^2$$

It then applies a second transformation to get $\tilde{z} = (\tilde{z}^{(1)}, \dots, \tilde{z}^{(m)})$ using learned parameters γ and β :

$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

- (a) **(1 point)** Explain the purpose of ϵ in the expression for $z_{\text{norm}}^{(i)}$.

You want to use a neural network with batch normalization layers (described in Figure 2) to build a classifier that can distinguish a hot dog from not a hot dog. (For the rest of this question, assume $\epsilon = 0$.)

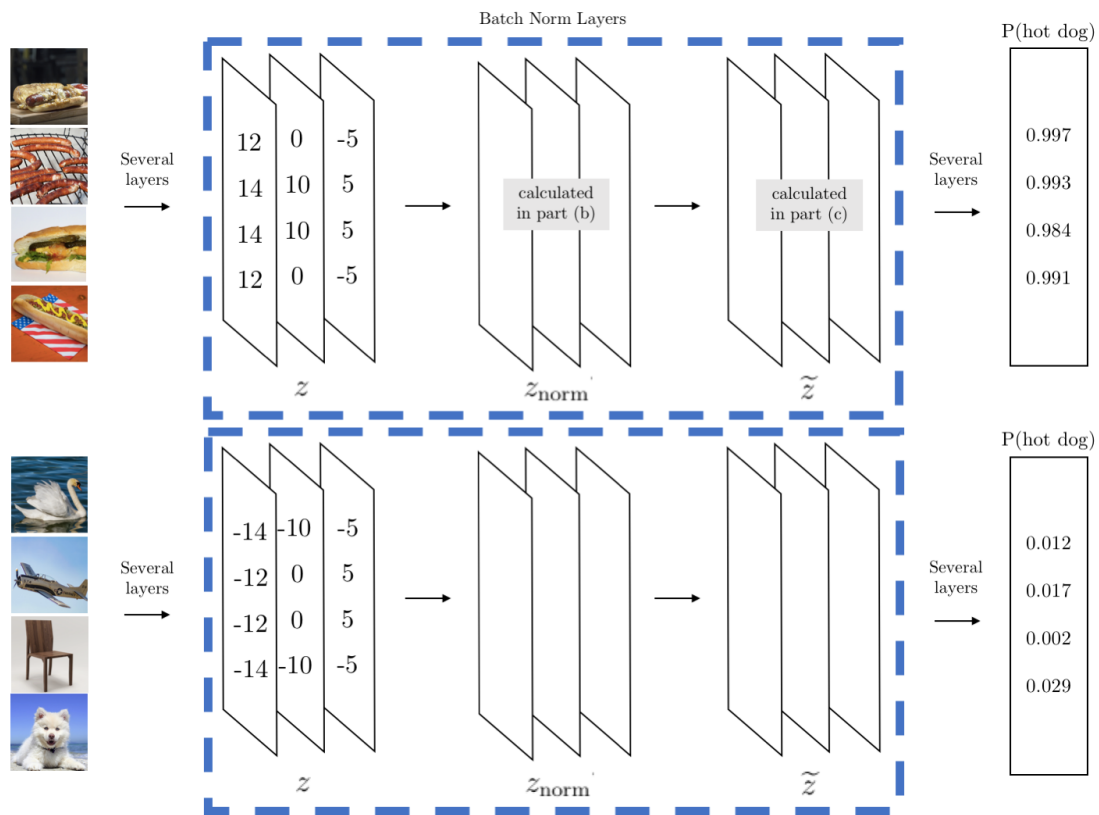


Figure 2: Hot Dog Network

- (b) **(2 points)** You forward propagate a batch of m examples in your network. The input $z = (z^{(1)}, \dots, z^{(m)})$ of the batch normalization layer has shape $(n_h = 3, m = 4)$, where n_h represents the number of neurons in the pre-batchnorm layer:

$$\begin{pmatrix} 12 & 14 & 14 & 12 \\ 0 & 10 & 10 & 0 \\ -5 & 5 & 5 & -5 \end{pmatrix}$$

What is z_{norm} ? Express your answer as a matrix with shape $(3, 4)$.

(c) **(2 points)** Suppose $\gamma = (1, 1, 1)$ and $\beta = (0, -10, 10)$. What is \tilde{z} ? Express your answer as a matrix with shape $(3, 4)$.



(d) **(2 points)** Give 2 benefits of using a batch normalization layer.

(e) **(2 points)** Explain what would go wrong if the batch normalization layer only applied the first transformation (z_{norm}).

Recall that during training time, the batch normalization layer uses the mini-batch statistics to estimate μ and (σ^2) . However, at test time, it uses the moving averages of the mean and variance tracked (but not used) during training time.

(f) **(2 points)** Why is this approach preferred over using the mini-batch statistics during training *and* at test time?

Suppose you have the following dataset:

Description	Number of images	Example of an image
Photos of hot dogs	100k	
Photos of not hot dogs	100k	

You make the mistake of constructing each mini-batch entirely out of one of the two groups of data, instead of mixing both groups into each mini-batch. As a result, even though your training loss is low, when you start using the trained model at test time in your new mobile app, it performs very poorly.

- (g) **(4 points)** Explain why inclusion of the batch normalization layer causes a discrepancy between training error and testing error when mini-batches are constructed with poor mixing. *Hint: Consider the example values given in Figure 2. What are the moving-averages over the two batches depicted and how do they compare to the batch statistics?*

Question 5 (Numpy coding, 10 points)

In this question, you will implement a fully-connected network. The architecture is *LINEAR* \rightarrow *RELU* \rightarrow *DROPOUT* \rightarrow *BATCHNORM*. This is a dummy architecture that has been made up for this exercise.

The code below implements the forward propagation, but some parts are missing. You will need to fill the parts between the tags (START CODE HERE) and (END CODE HERE) **based on the given instructions**. Note that we are using only numpy (not tensorflow), and the relu function has been imported for you.

```
import numpy as np
from utils import relu

def forward_propagation_with_dropout_and_batch_norm(X, parameters, keep_prob = 0.5):
    """
    Implements the forward propagation: LINEAR -> RELU -> DROPOUT -> BATCHNORM.

    Arguments:
    X -- input data of shape (n_x, m)
    parameters -- python dictionary containing your parameters:
        W -- weight matrix of shape (n_y, n_x)
        b -- bias vector of shape (n_y, 1)
        keep_prob -- probability of keeping a neuron active during drop-out, scalar
        gamma -- shifting parameter of the batch normalization layer, scalar
        beta -- scaling parameter of the batch normalization layer, scalar
        epsilon -- stability parameter of the batch normalization layer, scalar

    Returns:
    A2 -- output of the forward propagation
    cache -- tuple, information stored for computing the backward propagation
    """

    # retrieve parameters
    W = parameters["W"]
    b = parameters["b"]
    gamma = parameters["gamma"]
    beta = parameters["beta"]
    epsilon = parameters["epsilon"]

    ### START CODE HERE ###
    # LINEAR
    Z1 =

    # RELU
    A1 =
```

```

# DROPOUT
# Step 1: initialize matrix D1 = np.random.rand(..., ...)
D1 =
# Step 2: convert entries of D1 to 0 or 1 (using keep_prob as the threshold)
D1 =
# Step 3: shut down some neurons of A1
A1 =
# Step 4: scale the value of neurons that haven't been shut down
A1 =

# BATCHNORM
# Step 1: compute the vector of means; you can use np.mean(..., axis=...)
mu =
# Step 2: compute the vector of variances; you can use np.var(..., axis=...)
var =
# Step 3: normalize the activations (don't forget about stability)
A1_norm =
# Step 4: scale and shift
A2 =
### END CODE HERE ###

cache = (Z1, D1, A1, W, b, A1_norm, mu, var, gamma, beta, epsilon)

return A2, cache

```

Question 6 ("Who let the cars in?", 4 points)

You have just moved into your new holiday mansion Mar-a-Yolo and would like to build a car detection model to use at your front gate. You have a database $\mathcal{D} = \{x_1, \dots, x_N\}$ of the N cars allowed to enter your mansion. A training example for your network is a pair of images $(x^{(1)}, x^{(2)})$ where $x^{(1)}$ and $x^{(2)}$ are different images of the **same** car. You forward propagate $(x^{(1)}, x^{(2)})$ through your network to get encodings $(e^{(1)}, e^{(2)})$.

Using these encodings, you compute the loss (distance between encodings):

$$\mathcal{L} = \|e^{(1)} - e^{(2)}\|^2 \quad (1)$$

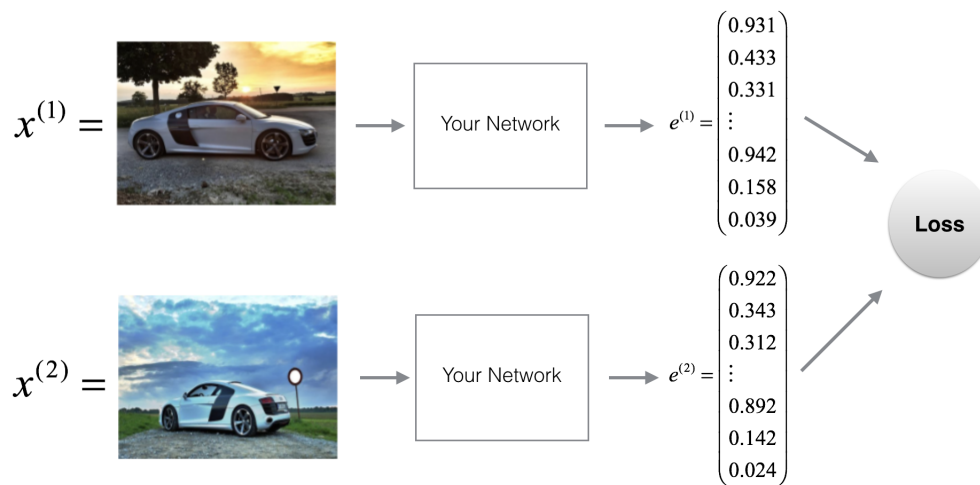


Figure 3: Car detection model

You've chosen your loss such that pictures of the same car have similar encodings. You hope that your network learned to output meaningful encodings to identify cars.

- (a) **(2 points)** You pick a pair of (different) unseen images of the same car and compute the loss \mathcal{L} . The loss is extremely low, but your model performs poorly when deployed. What could be a possible problem with your approach?

You decide to implement the triplet training scheme, as used in class for face recognition and retrain your network using this scheme.

- (b) **(2 points)** Explain how you would use this newly trained network to decide whether a car in front of your mansion should be allowed to enter.

Question 7 (X-Network, 23 points)

An X-neuron, as opposed to a neuron, takes vectors as input, and outputs vectors. There are three stages in the forward propagation for an X-neuron.

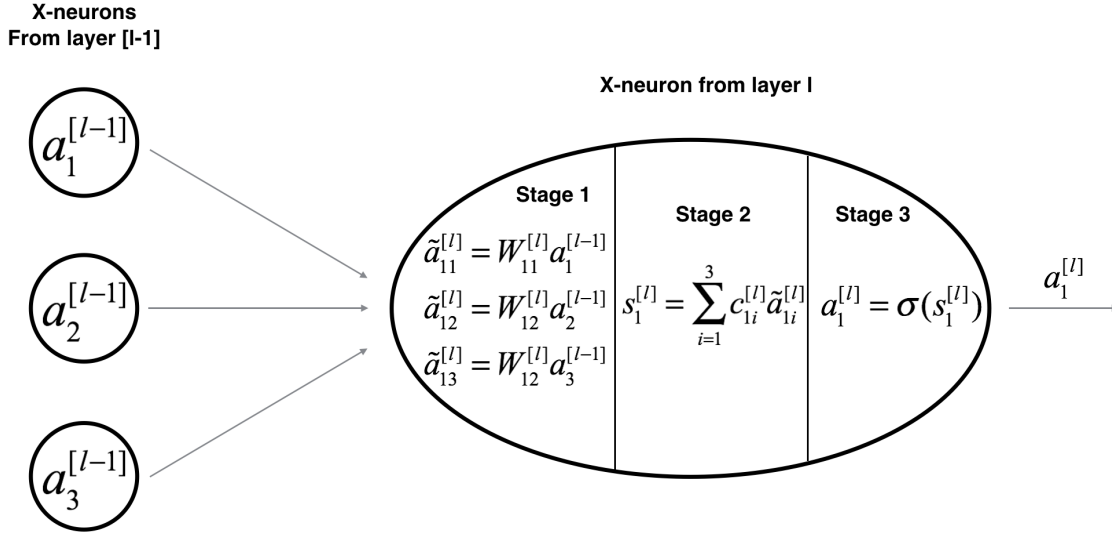


Figure 4: $(a_1^{[l-1]}, a_2^{[l-1]}, a_3^{[l-1]})$ are vector outputs of the previous layer's X-neurons (assuming there are only 3 X-neurons in the previous layer). The forward propagation in an X-neuron is split into 3 stages. $a_1^{[l]}$ is the vector output of layer l 's X-neuron.

Stage 1 is the linear transformation. For a given X-neuron j with inputs from X-neurons in the previous layer indexed with i :

$$\tilde{a}_{ji}^{[l]} = W_{ji}^{[l]} a_i^{[l-1]}$$

Stage 2 is the linear combination of the outputs of stage 1.

$$s_j^{[l]} = \sum_{i=1}^3 c_{ij}^{[l]} \tilde{a}_{ji}^{[l]}$$

Stage 3 is the non-linear sigmoid activation of the X-neuron. It is applied element-wise to its input vector $s_j^{[l]}$.

$$a_j^{[l]} = \sigma(s_j^{[l]})$$

Note that with the above scheme, every X-neuron of a layer is connected to every X-neuron of the previous layer, just like in classic fully-connected networks. The difference is that each X-neuron outputs a vector.

Consider the following 3-layer X-network, made of X-neurons exclusively.

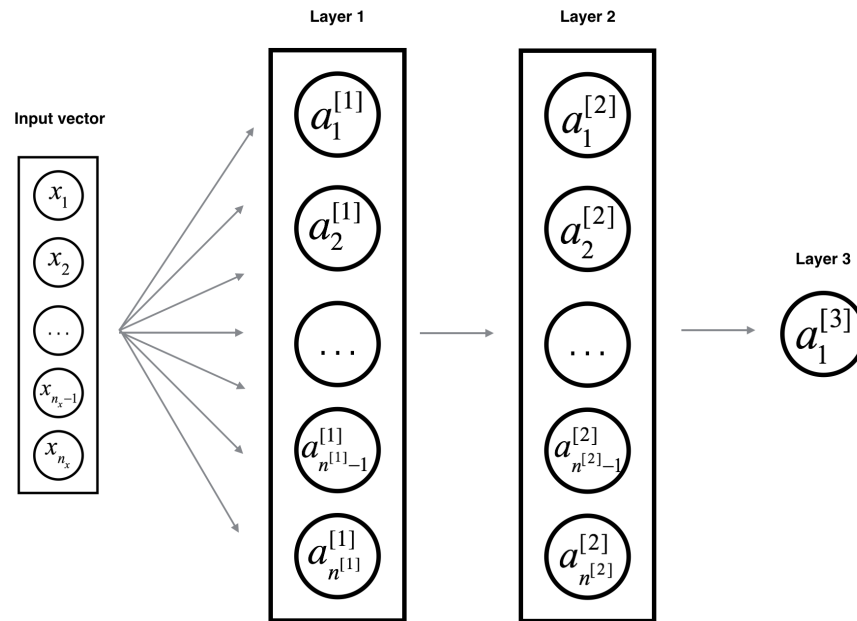


Figure 5: The input is a vector x of shape $(n_x, 1)$, which is duplicated and given to every X-neuron of the first layer. The X-network consists of three layers in which layer 1 has $n^{[1]}$ X-neurons, layer 2 has $n^{[2]}$ X-neurons, and layer 3 has a single X-neuron. Although for simplicity we did not draw all arrows, every layer is fully-connected to the next one.

(a) Given that all X-neurons in your network output a vector of shape $(D, 1)$:

(i) **(2 points)** What is the dimension of $W_{ji}^{[2]}$, for a single X-neuron in layer 2 of your X-network?

(ii) **(2 points)** What is the dimension of $W_{ji}^{[1]}$, for a single X-neuron in layer 1 of your network?

- (b) **(2 points)** For the j^{th} X-neuron of the 2^{nd} layer, how many weights ($c_{ij}^{[l]}$) does it have in its stage 2?
- (c) Consider the Stage 2 computation of the j^{th} X-neuron in layer 2:
- (i) **(2 points)** How would you vectorize this sum as a matrix product?
- (ii) **(1 point)** What is the dimension of the output of stage 2 of the j^{th} X-neuron?
- (d) The loss function \mathcal{L} is a scalar that has been calculated from the output $a_1^{[3]}$ from the final layer's X-neuron. You are given the derivative $\frac{\partial \mathcal{L}}{\partial a_1^{[2]}}$.
- (i) **(2 points)** What is the dimension of this gradient?
- (ii) **(3 points)** Given this derivative, write the derivative $\frac{\partial \mathcal{L}}{\partial s_1^{[2]}}$ in terms of $\frac{\partial \mathcal{L}}{\partial a_1^{[2]}}$ and $a_1^{[2]}$?
- (e) **(3 points)** Using the derivative calculated in question (d), write down the gradient $\frac{\partial \mathcal{L}}{\partial w_{1i}^{[2]}}$.

You decide to use your X-network to predict if an image contains a falafel or not. The falafel is a type of food, and this is a binary classification application.

Given an image, you run it through your X-network, and use the output activation $a_1^{[3]}$ to make a prediction. You do this by using the $L2$ norm of the output as a measure of probability of the presence of a falafel. You define the loss function for one training example as follows:

$$\mathcal{L} = y \max(0, m^+ - \|a_1^{[3]}\|)^2 + \lambda(1 - y) \max(0, \|a_1^{[3]}\| - m^-)^2$$

You have set $\lambda = 1$, $m^- = 0.1$ and $m^+ = 0.9$ (they are hyperparameters). Here, y is the label which tells you whether the image truly has a falafel ($y = 1$) or not ($y = 0$).

(f) **(2 point)** Draw the graph of \mathcal{L} against $\|a_1^{[3]}\|$ assuming the picture contains a falafel.

(g) Recall that your output activation is sigmoid, and the output of every X-neuron is of dimension D . Considering an image that is **not** a falafel:

(i) **(2 points)** What is the maximum value the loss can have?

(ii) **(2 points)** What is the minimum value of the loss and when is this minimum reached?

Question 8 (Practical case study: Online advertising, 11 points)

Learning and predicting user response plays a crucial role in online advertising.

- (a) **(2 points)** Let's say you have a website with a lot of traffic. You would like to build a network that computes the probability of a user clicking on a given advertisement on your website. This is a supervised learning setting. What dataset do you need to train such a network?

- (b) **(2 points)** Choose an appropriate cost function for the problem and give the formula of the cost.

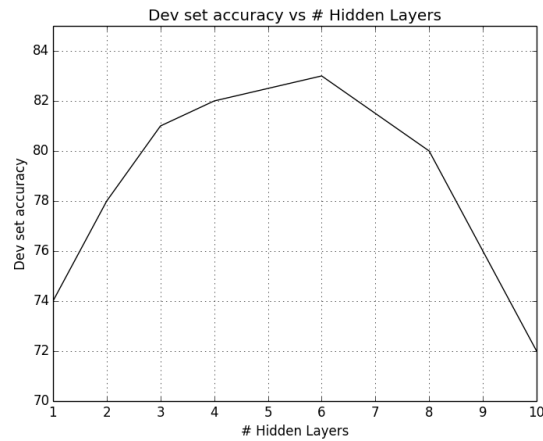
- (c) **(2 points)** Your website sells sport footwear and you have already collected a dataset of 1 million examples from past visits. Your friend, who works in the high fashion industry, offers to let you use their dataset as it has similar descriptors. However, you are concerned about the impact of this different distribution dataset in the performance of your predictive system. Explain how you would use your friend's dataset.

- (d) **(1 point)** How would you assess the impact of the new dataset?

- (e) **(2 points)** Initially, you decide to build a fully connected neural network for the problem. This baseline model would have L hidden layers, one input layer and an output layer. The number of neurons in the l^{th} layer is $n^{[l]}$. Write down the number of parameters of this model as a function of L and $n^{[l]}$ for $l = 0 \dots L$.

Note: The input layer is considered to be layer number 0

- (f) **(2 points)** Based on the information provided in the graph below, what type of problem will you encounter as the number of hidden layer approaches 10? Mention possible solutions to this problem.



Extra Page 1/6

Extra Page 2/6

Extra Page 3/6

Extra Page 4/6

Extra Page 5/6

Extra Page 6/6

END OF PAPER