CS230: Deep Learning

Winter Quarter 2018

Stanford University

Midterm Examination

180 minutes

	Problem	Full Points	Your Score
1	Multiple Choice	7	
2	Short Answers	22	
3	Coding	7	
4	Backpropagation	12	
5	Universal Approximation	19	
6	Optimization	9	
7	Case Study	25	
8	AlphaTicTacToe Zero	11	
9	Practical industry-level questions	8	
Total		120	

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, 7 points)

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

(a) (1 point) You want to map every possible image of size 64×64 to a binary category (cat or non-cat). Each image has 3 channels and each pixel in each channel can take an integer value between (and including) 0 and 255. How many bits do you need to represent this mapping?

(i) $256^{3^{64\times 64}}$
(ii) $256^{3\times 64\times 64}$
(iii) $(64 \times 64)^{256\times 3}$
(iv) $(256 \times 3)^{64\times 64}$

Solution: (ii)

(b) (1 point) The mapping from question (a) clearly can not be stored in memory. Instead, you will build a classifier to do this mapping. Recall the simple single hidden layer classifier you used in the assignment on classifying images as cat vs non-cat. You use a single hidden layer of size 100 for this task. Each weight in the $W^{[1]}$ and $W^{[2]}$ matrices can be represented in memory using a float of size 64 bits. How many bits do you need to store your two layer neural network (you may ignore the biases $b^{[1]}$ and $b^{[2]}$)?

(i) $64 \times ((256 \times 3 \times 100) + (64 \times 64 \times 1))$
(ii) $64 \times ((64 \times 64 \times 3 \times 100) + (100 \times 1))$
(iii) $64 \times ((256^3 \times 64 \times 64 \times 100) + (100 \times 64))$
(iv) $64 \times (256 \times 3 \times 64 \times 64 \times 100)$

Solution: (ii)

(c) (1 point) Suppose you have a 3-dimensional input $x = (x_1, x_2, x_3) = (2, 2, 1)$ fully connected to 1 neuron with activation function g_i. The forward propagation can be written:

$$z = (\sum_{k=1}^{3} w_k x_k) + b$$
$$a_i = g_i(z)$$

After training this network, the values of the weights and bias are: $w = (w_1, w_2, w_3) = (0.5, -0.2, 0)$ and $b = 0.1$. You try 4 different activation functions (g_1, g_2, g_3, g_4) which respectively output the values $(a_1, a_2, a_3, a_4) = (0.67, 0.70, 1.0, 0.70)$. What is a valid guess for the activation functions g_1, g_2, g_3, g_4?

(i) sigmoid, tanh, indicator function, linear

(ii) linear, indicator function, sigmoid, ReLU

(iii) sigmoid, linear, indicator function, leaky ReLU

(iv) ReLU, linear, indicator function, sigmoid

(v) sigmoid, tanh, linear, ReLU

Recall that the indicator function is:

$$I(x)_{x\geq 0} = \begin{cases} 0 \quad if \quad x < 0 \\ 1 \quad if \quad x \geq 0 \end{cases}$$

Solution: iii

(d) (2 points) A common method to accelerate the training of Generative Adversarial Networks (GANs) is to update the Generator k (≥ 1) times for every 1 time you update the Discriminator.

(i) True

(ii) False

(iii) It depends on the architecture of the GAN.

Solution: ii

(e) (2 points) BatchNorm layers are really important when it comes to training GANs. However, the internal parameters of the BatchNorm (γ and β) are highly correlated to the input mini-batch of examples, which leads to the generated images being very similar to each other in a mini-batch.

(i) True

(ii) False

Solution: ii

Question 2 (Short Answers, 22 points)

Please write concise answers.

(a) (2 points) What's the risk with tuning hyperparameters using a test dataset?

Solution: The model will not generalize well to unseen data because it **overfits** the test set. Tuning model hyperparameters to a test set means that the hyperparameters may overfit to that test set. If the same test set is used to estimate performance, it will produce an overestimate. Using a separate validation set for tuning and test set for measuring performance provides unbiased, realistic measurement of performance.

(b) (2 points) Explain why dropout in a neural network acts as a regularizer.

Solution: There were several acceptable answers:

(1) Dropout is a form of model averaging. In particular, for a layer of H nodes, we sampling from 2^H architectures, where we choose an arbitrary subset of the nodes to remain active. The weights learned are shared across all these models means that the various models are regularizing the other models.

(2) Dropout helps prevent feature co-adaptation, which has a regularizing effect.

(3) Dropout adds noise to the learning process, and training with noise in general has a regularizing effect.

(4) Dropout leads to more sparsity in the hidden units, which has a regularizing effect. (Note that in one of the lecture videos, this was phrased as dropout "shrinking the weights" or "spreading out the weights". We will also accept this phrasing.)

Answers that defined dropout without explaining why it acts as a regularizer got no credit. Answers that included one or more of the above explanations but also made claims that were incorrect did not receive full credit.

(c) (3 points) Why do we often refer to L2-regularization as "weight decay"? Derive a mathematical expression to explain your point.

Solution: In the case of L2 regularization, we can derive the following update rule for the weights:

$$W = (1 - \alpha\lambda)W - \frac{\partial J}{\partial W}$$

where α is the learning rate and λ is the regularization hyperparameter ($\alpha\lambda << 1$). This shows that at every iteration W's value is pushed closer to zero.

(d) (2 points) Explain what effect will the following operations generally have on the bias and variance of your model. Fill in one of 'increases', 'decreases' or 'no change' in each of the cells:

	Bias	Variance
Regularizing the weights		
Increasing the size of the layers (more hidden units per layer)		
Using dropout to train a deep neural network		
Getting more training data (from the same distribution as before)		

Solution: 1) Increases, Decreases 2) Decreases, Increases 3) Increases, Decreases 4) No change, Decreases

(e) (3 points) Suppose you are initializing the weights $W^{[l]}$ of a layer with **uniform** random distribution $U(-\alpha, \alpha)$. The number of input and output neurons of the layer l are $n^{[l-1]}$ and $n^{[l]}$ respectively.

Assume the input activation and weights are independent and identically distributed, and have mean zero. You would like to satisfy the following equations:

$$E[z^{[l]}] = 0$$

$$Var[z^{[l]}] = Var[a^{[l-1]}]$$

What should be the value of α?
Hints: If X is a random variable distributed uniformly $U(-\alpha, \alpha)$, then $E(X) = 0$ and $Var(X) = \frac{\alpha^2}{3}$. Use the following relation seen in-class:

$$Var(z^{[l]}) = n^{[l-1]} Var(W^{[l]}) Var(a^{[l-1]})$$

Solution: $\alpha = \sqrt{\frac{3}{n^{[l-1]}}}$.

(f) Consider the graph in Figure 1 representing the training procedure of a GAN:

(i) (2 points) Early in the training, is the value of $D(G(z))$ closer to 0 or closer to 1? Explain.

Solution: The value of $D(G(z))$ is closer to 0 because early in the training D is much better than G. One reason is that G's task (generating images that look like real data) is a lot harder to learn than D's task (distinguishing fake images from real images).

(ii) (2 points) Two cost functions are presented in figure (1), which one would you choose to train your GAN? Justify your answer.

Solution: I would use the "non-saturating cost" because it leads to much higher gradients early in the training and thus helps the generator learn quicker.

(iii) (2 points) You know that your GAN is trained when D(G(z)) is close to 1. True / False ? Explain.

Solution: False, at the end of the training G is able to fool D. So $D(G(z))$ is close to 0.5 which means that D is randomly guessing.

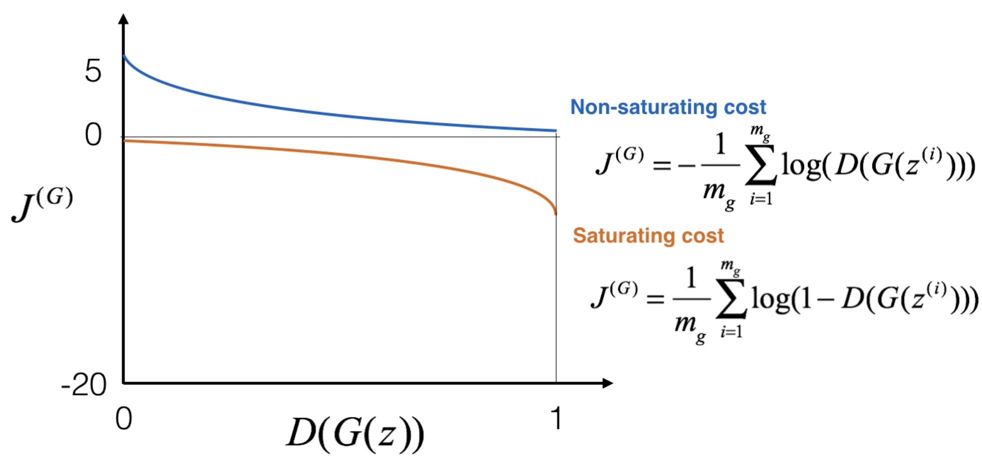


Figure 1: Cost function of the generator plotted against the output of the discriminator when given a generated image $G(z)$. Concerning the discriminator's output, we consider that 0 (resp. 1) means that the discriminator thinks the input "has been generated by G" (resp. "comes from the real data").

(g) (2 points) A neural network has been encrypted on a device, you can access neither its architecture, nor the values of its parameters. Is it possible to create an adversarial example to attack this network? Explain why.

Solution: Yes. 1. You can train a different neural network for the same task and build an adversarial example to this network. This example will be also an adversarial example for the encrypted network. 2. (Also fine) computing the gradients or the cost with respect to the inputs can be numerically approximated.

(h) In a neural network, consider a layer that has $n^{[l-1]}$ inputs, $n^{[l]}$ outputs and uses a linear activation function. The input $a^{[l-1]}$ is independently and identically distributed, with zero mean and unit variance.

(i.) (1 point) How can you initialize the weights of this layer to ensure the output $z^{[l]}$ has the same variance as $a^{[l-1]}$ during the forward propagation?

Solution: In forward pass: initialize the weight randomly with variance of $\frac{1}{n^{[l-1]}}$.

(ii.) (1 point) How can you initialize the weights of this layer to ensure the gradient of input have unit variance as the gradient of the output during back-propagation? Explain your answer.

Solution: In back-propagation: initialize the weight randomly with variance of $\frac{1}{n^{[l]}}$.

Question 3 (Coding, 7 points)

In this question you are asked to implement a training loop for a classifier. The input data is X, of shape (n_x, m) where m is the number of training examples. You are using a 2-layer neural network with:

- one hidden layer with n_h neurons
- an output layer with n_y neurons.

The code below is meant to implement the training loop, but some parts are missing. Between the tags (START CODE HERE) and (END CODE HERE), implement:

(i) the parameters initialization: initialize all your parameters, the weights should be initialized with Xavier initialization and the biases with zeros.

(ii) the parameters update: update your parameters with Batch Gradient Descent with momentum.

We won't penalize syntax errors.

```
import numpy as np

def train(X_train, Y_train, n_h, n_y, num_iterations, learning_rate, beta):
  """
  Implement train loop of a two layer classifier
  Arguments:
  X_train -- training data
  Y_train -- labels
  n_h -- size of hidden layer
  n_y -- size of output layer
  num_iteration -- number of iterations
  learning_rate -- learning rate, scalar
  beta -- the momentum hyperparameter, scalar

  Returns:
  W1, W2, b1, b2 -- trained weights and biases
  """

  m = X_train.shape[1] # number of training examples
  n_x = X_train.shape[0] # size of each training example
```

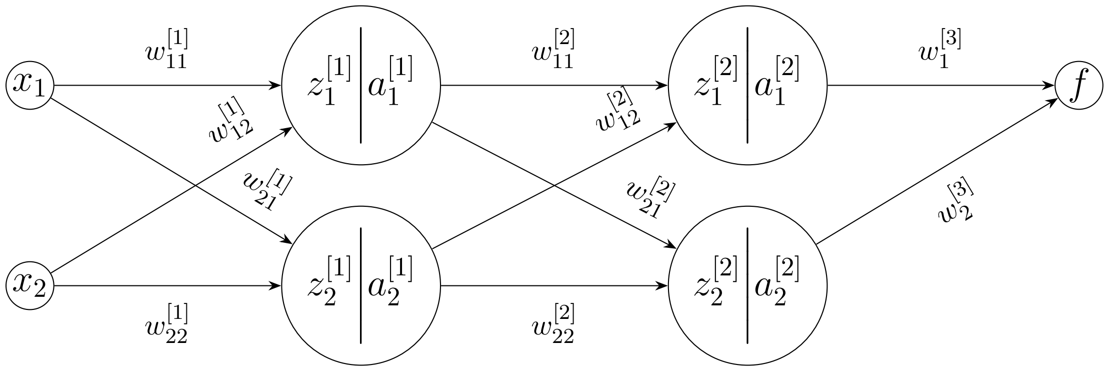
```
# initialize parameters
### START CODE HERE ###











### END CODE HERE ###

# training loop
for i in range(num_iterations):

    # Forward propagation
    a1, cache1 = activation_forward(X, W1, b1, activation = "relu")
    a2, cache2 = activation_forward(a1, W2, b2, activation = "sigmoid")

    # Compute cost
    cost = compute_cost(a2, Y_train)

    # Backward propagation
    da2 = - 1./m * (np.divide(Y_train, a2) - np.divide(1 - Y_train, 1 - a2))
    da1, dW2, db2 = activation_backward(da2, cache2, activation = "sigmoid")
    dX, dW1, db1 = activation_backward(da1, cache1, activation = "relu")

    # Update parameters with momentum
    ### START CODE HERE ###











    ### END CODE HERE ###
return W1, W2, b1, b2
```

Solution:

```
## Initializing
W1 = np.random.randn(nh, nx) * np.sqrt( 2 / (nx + nh))
# other type of xavier intializations like
# np.sqrt( 1 / (nx)) or np.sqrt( 1 / (nh))
# are also acceptable solutions
b1 = np.zeros((nh, 1))
W2 = np.random.randn(ny, nh) * np.sqrt( 2 / (nh + ny))
b2 = np.zeros((ny, 1))

vdW1 = np.zeros(W1.shape)
vdb1 = np.zeros(b1.shape)
vdW2 = np.zeros(W2.shape)
vdb2 = np.zeros(b2.shape)

## Updating
vdW1 = beta * vdW1 + (1 - beta) * dW1
vdW2 = beta * vdW2 + (1 - beta) * dW2
vdb1 = beta * vdb1 + (1 - beta) * db1
vdb2 = beta * vdb2 + (1 - beta) * db2

W1 -= learning_rate * vdW1
W2 -= learning_rate * vdW2
b1 -= learning_rate * vdb1
b2 -= learning_rate * vdb2
# Solutions with bias correction are acceptable:
"""
W1 -= learning_rate * vdW1 / (1 - beta ** (i + 1))
W2 -= learning_rate * vdW2 / (1 - beta ** (i + 1))
b1 -= learning_rate * vdb1 / (1 - beta ** (i + 1))
b2 -= learning_rate * vdb2 / (1 - beta ** (i + 1))
"""
```

Question 4 (Backpropagation, 12 points)

Consider this three layer network:



$$Z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad , \quad A^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \end{bmatrix}$$

$$Z^{[2]} = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} \quad , \quad A^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[2]}) \\ \sigma(z_2^{[2]}) \end{bmatrix}$$

Given that $f = w_1^{[3]} a_1^{[2]} + w_2^{[3]} a_2^{[2]}$, compute :

(i) (3 points) $\delta_1 = \frac{\partial f(x)}{\partial z_1^{[2]}}$

(ii) (3 points) $\delta_2 = \frac{\partial f(x)}{\partial Z^{[2]}}$

(iii) (3 points) $\delta_3 = \frac{\partial f(x)}{\partial Z^{[1]}}$

(iv) (3 points) $\delta_4 = \frac{\partial f(x)}{\partial w_{11}^{[1]}}$

Solution:

(i) $\frac{\partial f(x)}{\partial z_1^{[2]}} = w_1^{[3]} \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} = w_1^{[3]} \sigma(z_1^{[2]})(1 - \sigma(z_1^{[2]})) = w_1^{[3]} a_1^{[2]}(1 - a_1^{[2]})$

$$f = \begin{bmatrix} w_1^{[3]} \; w_2^{[3]} \end{bmatrix} A^{[2]}, \qquad A^{[2]} = \sigma(Z^{[2]})$$

(ii)

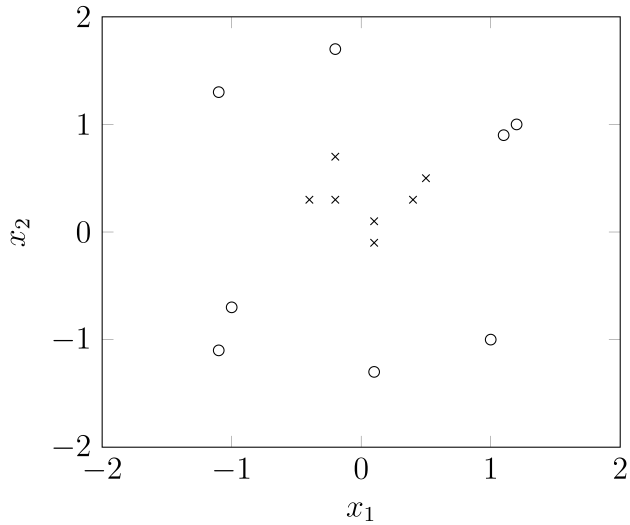
$$\frac{\partial f}{\partial Z^{[2]}} = \frac{\partial f}{\partial A^{[2]}} . \frac{\partial A^{[2]}}{\partial Z^{[2]}} = \begin{bmatrix} w_1^{[3]} \; w_2^{[3]} \end{bmatrix}^T \circ A^{[2]} \circ (1 - A^{[2]})$$

(iii) $\frac{\partial f(x)}{\partial Z^{[1]}} = \frac{\partial f(x)}{\partial Z^{[2]}} . \frac{\partial Z^{[2]}}{\partial Z^{[1]}} = \frac{\partial f(x)}{\partial Z^{[2]}} . \frac{\partial Z^{[2]}}{\partial A^{[1]}} . \frac{\partial A^{[1]}}{\partial Z^{[1]}} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \end{bmatrix} \delta_2 \circ A^{[1]} \circ (1 - A^{[1]})$

(iv) $\delta_4 = \frac{\partial f(x)}{\partial w_{11}} = \frac{\partial f(x)}{\partial Z^{[1]}} . \frac{\partial Z^{[1]}}{\partial w_{11}} = \delta_3^T \begin{bmatrix} x_1 \\ 0 \end{bmatrix}$

Question 5 (Universal Approximation, (19 points))

Consider the following binary classification task:



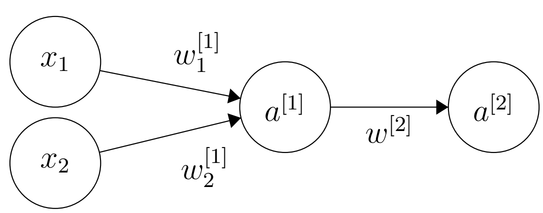
(a.) Let's begin by modeling this problem with a simple 2 layer network: with an activation function $g^{[1]}$ and a sigmoid output unit ($\sigma(z) = \frac{1}{1+e^{-z}}$):

$$z^{[1]} = w_1^{[1]}x_1 + w_2^{[1]}x_2 + b^{[1]}$$

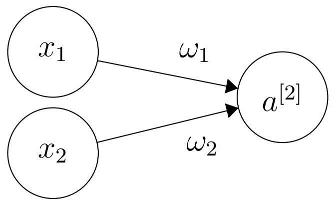
$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$



(i.) (2 points) Show that if $g^{[1]}$ is a linear activation function, $g^{[1]}(z) = \alpha z$, then the above network can be reduced to the single layer network shown below. Give the new weights and bias for this network $\omega_1, \omega_2, \beta_1$.



Solution:

$$a^{(2)} = \sigma(w^{(2)}a^{(1)} + b^{(2)})$$

$$a^{(2)} = \sigma(\alpha w^{(2)}w_1^{(1)}x_1 + \alpha w^{(2)}w_2^{(1)}x_2 + b^{(2)} + \alpha w^{(2)}b^1)$$

$$a^{(2)} = \sigma(\omega_1 x_1 + \omega_2 x_2 + \beta_1)$$

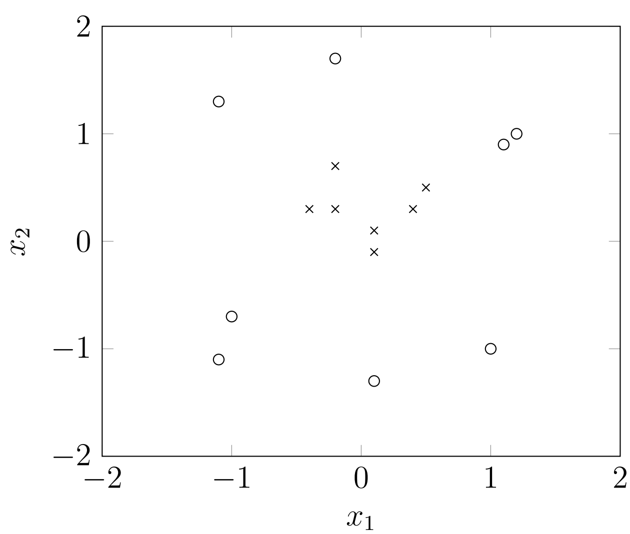
which makes,

$$\omega_1 = \alpha w^{(2)}w_1^{(1)}$$

$$\omega_2 = \alpha w^{(2)}w_2^{(1)}$$

$$\beta_1 = b^{(2)} + \alpha w^{(2)}b^1$$

(ii.) (2 points) If we use a threshold of $a^{[2]} > 0.5$, what is the form of decision rules that can be learned using a $g(z) = \alpha z$. Draw an example of a possible decision rule on the plot below, and write down the equation of the decision rule.



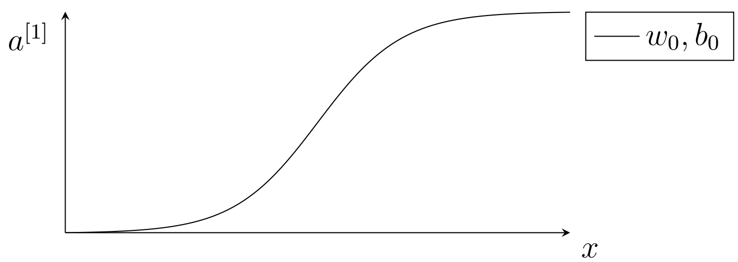
Solution: That makes our decision rule a linear boundary, with any equation of the form:

$$\omega_1 x_1 + \omega_2 x_2 + \beta_1 > 0$$

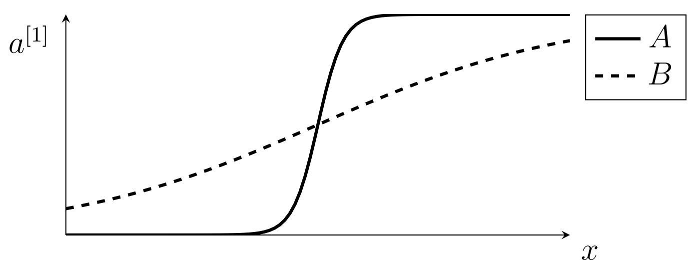
Note: The equation given above should have weights specified and should match the line drawn on the plot for full credit.

(b.) Instead of a linear activation function, let's examine the effects of defining $g^{[1]}$ as a sigmoid activation: $g^{[1]}(z) = \sigma(z) = \frac{1}{1+e^{-z}}$. In order to simplify the graphical representation, assume that you have a single neuron activation $a^{[1]} = g^{[1]}(z^{[1]}) = \sigma(wx + b)$ where w, x and b are scalars.

Let's plot $a^{[1]}$ against the input x for parameters $(w, b) = (w_0, b_0)$:

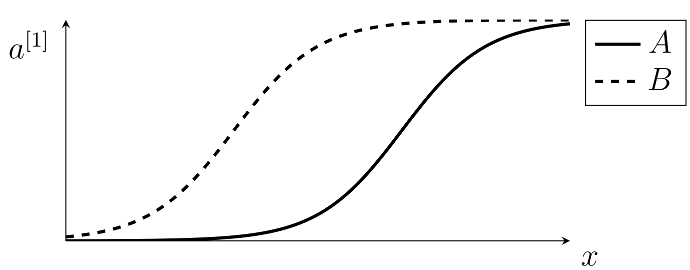


(i.) (1 point) For a $\Delta > 0$, which line corresponds to the change w to $w_0 + \Delta$? Circle **A** or **B**, given that we hold b at b_0.



Solution: A

(ii.) (1 point) For a $\Delta > 0$, which line corresponds to the change b to $b_0 + \Delta$? Circle **A** or **B**, given that we hold w at w_0.

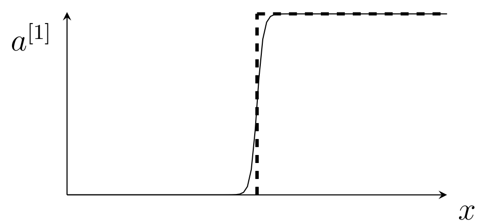


Solution: B

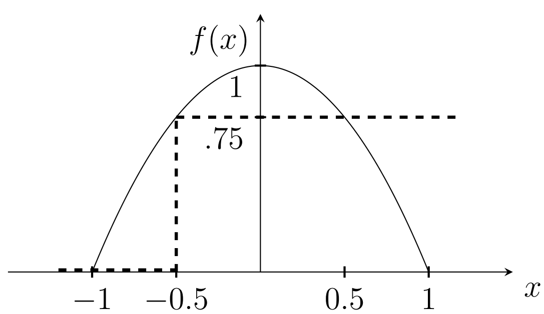
(c.) Given the responses above, it can be shown that for certain choices of w and b, the sigmoid response can closely approximate a step function:

$$a^{[1]} = \sigma(wx + b) \approx step(x) = 1\{x \geq s\}$$

with $s = -b/w$.

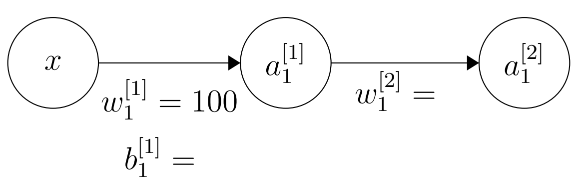


(i.) (2 points) For a scalar input x, you wish to approximate $f(x) = 1 - x^2$ with the dotted line below (a step function),



using the network,

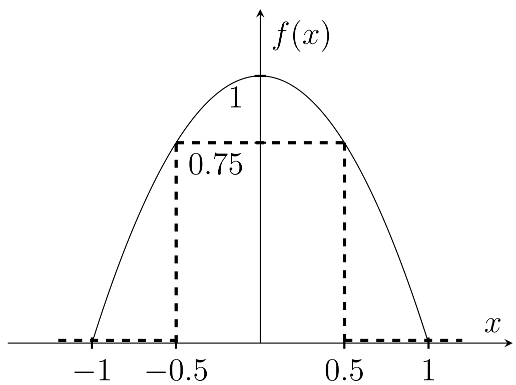
$$z_1^{[1]} = w_1^{[1]} x + b_1^{[1]}$$
$$a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_1^{[2]} = w_1^{[2]} a_1^{[1]}$$
$$a_1^{[2]} = z_1^{[2]}$$



fill in the above values of $b_1^{[1]}$ and $w_1^{[2]}$ that result in the dotted approximation.

Solution: $b_1^{[1]} = 50,\ w_1^{[2]} = .75$

(ii.) (4 points) You now want to go a step further and approximate using a pulse as shown below:



If you use the network below,

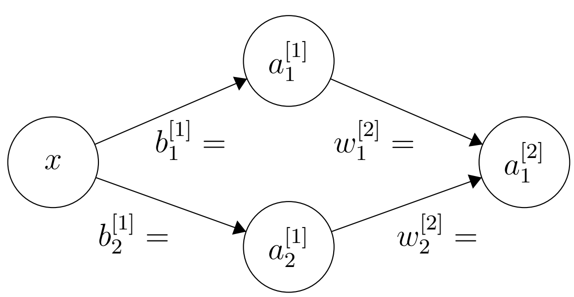
$$Z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]} & 0 \\ 0 & w_2^{[1]} \end{bmatrix} \begin{bmatrix} x \\ x \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \end{bmatrix}$$

$$Z^{[2]} = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \end{bmatrix} = \begin{bmatrix} w_1^{[2]} & 0 \\ 0 & w_2^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix}$$

$$a_1^{[2]} = z_1^{[2]} + z_2^{[2]}$$

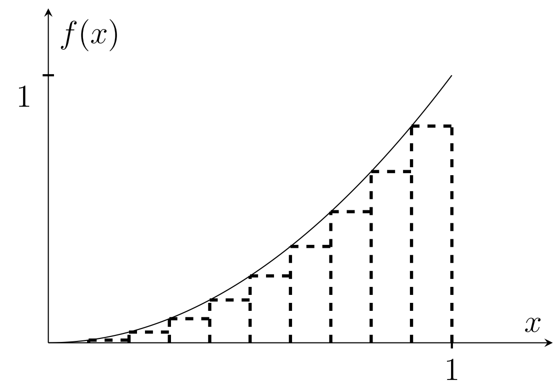
with $w_1^{[1]} = w_2^{[1]} = 100$,



Fill in the parameters $b_1^{[1]}$,$b_2^{[1]}$ and $w_1^{[2]}$,$w_2^{[2]}$ to approximate the dotted pulse function.

Solution: $b_1^{[1]} = 50,\ b_2^{[1]} = -50,\ w_1^{[2]} = .75,\ w_1^{[2]} = -.75$

(d.) You can extend the above scheme to closely approximate almost any function. Here you will approximate $f(x) = x^2$ over [0,1) using step impulses plotted below,



with each step impulse as

$$\tilde{f}(x; w, s_1, s_2) = w * 1\{s_1 \leq x < s_2\}$$

and your approximation as a sum of impulses,

$$\hat{f}(x) = \sum_{\hat{x} \in S} \tilde{f}(x; \hat{x}^2, \hat{x}, \hat{x} + \epsilon)$$

We define $S = \{0, \epsilon, 2\epsilon, ..., 1 - \epsilon\}$ where ϵ is a small positive scalar such that all bins have equal width.

(i.) (5 points) What is the maximum error $|f(x) - \hat{f}(x)|$ of your approximation over [0,1)? Give your answer as a function of ϵ, the impulse width.
Hint: for any $x \in [0, 1)$ consider $\hat{x}_n$ to be the smallest element of S such that $\hat{x}_n + \epsilon > x$, and find an upper bound to $|f(x) - \hat{f}(x)|$ that depends only on ϵ.

Solution: For any $x \in [0, 1]$ let $\hat{x}_n$ be the first element of S s.t. $\hat{x}_n + \epsilon > x$,

$$|f(x) - \hat{f}(x)| \leq |f(\hat{x}_n + \epsilon) - \hat{f}(x)|$$

$$|f(x) - \hat{f}(x)| \leq |(\hat{x}_n + \epsilon)^2 - \hat{x}_n^2| = |\hat{x}_n^2 + 2\hat{x}_n\epsilon + \epsilon^2 - \hat{x}_n^2|$$

$$|f(x) - \hat{f}(x)| \leq |2\hat{x}_n\epsilon + \epsilon^2|$$

and over our domain, $\hat{x}_n \leq 1 - \epsilon$

$$|f(x) - \hat{f}(x)| \leq 2\epsilon - \epsilon^2$$

(ii.) (2 points) As you've just seen, you can approximate functions arbitrarily well with 1 hidden layer. State a reason and explain why, in practice, you would use deeper networks.

Solution: There are a lot of possible answers to this question. For full credit we expected a specific reason and explanation. Some common answers were that deeper networks can approximate similar functions but often with less required parameters (think about the circuits example in lecture). Another is that deep networks are learning compositions of functions/layers of features which can be both easier to learn and improve generalization of networks over learning a single function.

Question 6 (Optimization, 9 points)

For these questions, we expect you to be concise and precise in your answers.

(a) (2 points) What problem(s) will result from using a learning rate that's too high? How would you detect these problems?

Solution: Cost function does not converge to an optimal solution and can even diverge. To detect, look at the costs after each iteration (plot the cost function v.s. the number of iterations). If the cost oscillates wildly, the learning rate is too high. For batch gradient descent, if the cost increases, the learning rate is too high.

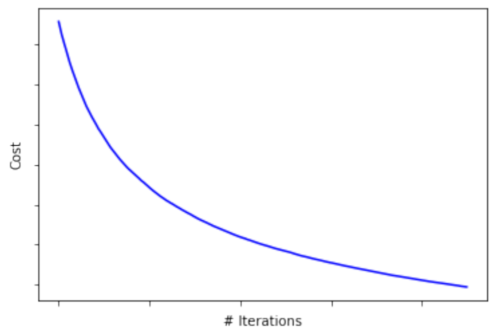
(b) (2 points) What problem(s) will result from using a learning rate that's too low? How would you detect these problems?

Solution: Cost function may not converge to an optimal solution, or will converge after a very long time. To detect, look at the costs after each iteration (plot the cost function v.s. the number of iterations). The cost function decreases very slowly (almost linearly). You could also try higher learning rates to see if the performance improves.

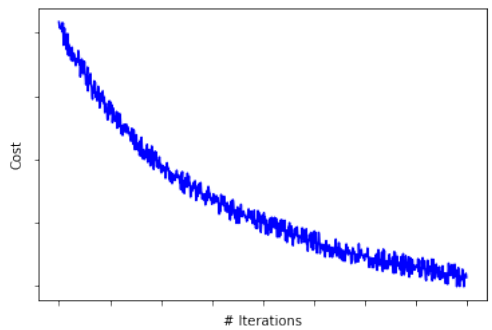
(c) (2 points) What is a saddle point? What is the advantage/disadvantage of Stochastic Gradient Descent in dealing with saddle points?

Solution: Saddle point - The gradient is zero, but it is neither a local minima nor a local maxima. Also accepted - the gradient is zero and the function has a local maximum in one direction, but a local minimum in another direction.
SGD has noisier updates and can help escape from a saddle point

(d) (1 point) Figure 2 below shows how the cost decreases (as the number of iterations increases) when two different optimization algorithms are used for training. Which of the graphs corresponds to using batch gradient descent as the optimization algorithm and which one corresponds to using mini-batch gradient descent? Explain.



(a) Graph A



(b) Graph B

Figure 2

Solution: Batch gradient descent - Graph A, Minibatch - Graph B. Batch gradient descent - the cost goes down at every single iteration (smooth curve). Mini-batch - does not decrease at every iteration since we are just training on a mini-batch (noisier)

(e) (2 points) Figure 3 below shows how the cost decreases (as the number of iterations increases) during training. What could have caused the sudden drop in the cost? Explain one reason.

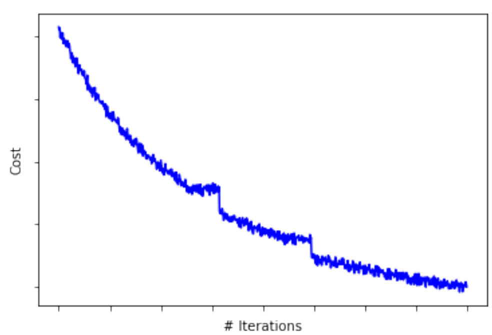


Figure 3

Solution: Learning rate decay

Question 7 (Case Study: semantic segmentation on microscopic images, 25 points)

You have been hired by a group of health-care researchers to solve one of their major challenges dealing with cell images: determining which parts of a microscope image corresponds to which individual cells.
In deep learning, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. In your case, you want to locate the cells and their boundaries in microscopic images.
Here are three examples of input images and the corresponding target images:

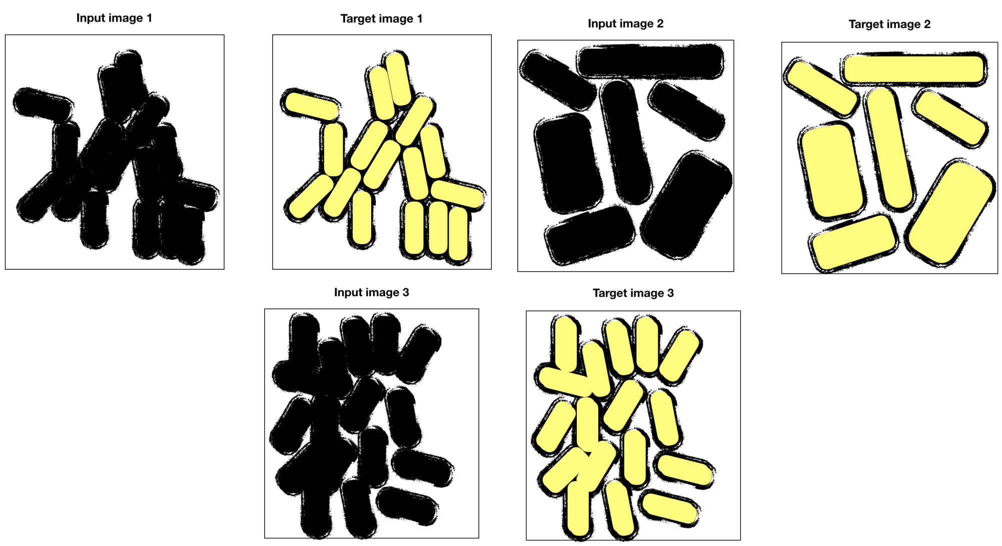


Figure 4: The input images are taken from a microscope. The target images have been created by the doctors, they labeled the pixels of the input image such that 1 represents the presence of a cell while 0 represents the absence of a cell. The target image is the superposition of the labels and the input image (light grey pixels you can see inside the cells correspond to label 1, indicating the pixel belongs to a cell). A good algorithm will segment the data the same way the doctors have labeled it.

In other words, this is a classification task where each pixel of the target image is labeled as 0 (this pixel is not part of a cell) or 1 (this pixel is part of a cell).

Dataset: Doctors have collected 100,000 images from microscopes and gave them to you. Images have been taken from three types of microscopes: A (50,000 images), B (25,000 images) and C (25,000 images). The doctors who hired you would like to use your algorithm on images from microscope C.

(a) (3 points) Explain how you would split this dataset into train, dev and test sets. Give the exact percentage split, and give reasons to your choices.

Solution:

- Split has to be roughly 90,5,5. Not 60,20,20.
- Distribution of dev and test set have to be the same (contain images from C).
- There should be C images in the training as well, more than in the test/dev set.

(b) (2 points) Can you augment this dataset? If yes, give only 3 distinct methods you would use. If no, explain why (give only 2 reasons).

Solution:

Those methods would work for augmentation based on the images shown above.

- cropping
- adding random noise
- changing contrast, blurring.
- flip
- rotate

You have finished the data processing, and are wondering you could solve the problem using a neural network. Given a training examples $x^{(i)}$ (flattened version of an RGB input image, of shape $(n_x, 1)$ and its corresponding label $y^{(i)}$ (flattened version of labels of shape $(n_y, 1)$) answer the following questions:

(c) (2 points) What is the mathematical relation between n_x and n_y?

Solution: $n_x = 3 \times n_y$

(d) (2 points) Write down the cross-entropy loss function $\mathcal{L}^{(i)}$ for one training example.

Solution: $-\sum_{i=1}^{n_y}(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$ Summation over all pixel value with cross entropy loss.

(e) (1 point) Write down the cost function $\mathcal{J}$, i.e. generalize the loss function to a batch of m training examples.

Solution: $-\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{n_y}(y_i^{(k)} log(\hat{y}_i^{(k)}) + (1 - y_i^{(k)}) log(1 - \hat{y_i}^{(k)}))$

You have coded your neural network (model M1) and have trained it for 1000 epochs.

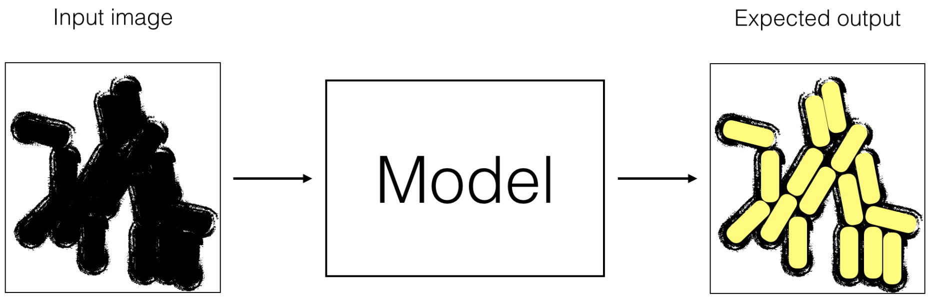


Figure 5: Your model (M1) takes as input a cell image taken from a microscope and outputs a matrix of 0s and 1s indicating for each pixel if it is part of a cell or not. You then superpose this matrix on the input image to see the results. For the given input image, M1 should ideally output the target image above.

Your model is not performing well. One of your friends suggested to use transfer learning using another labeled dataset made of 1,000,000 microscope images for skin disease classification. A model (M2) has been trained on this dataset on a 10-class classification. The images are the same size as those of the dataset the doctors gave you. Here is an example of input/output of the model M2.

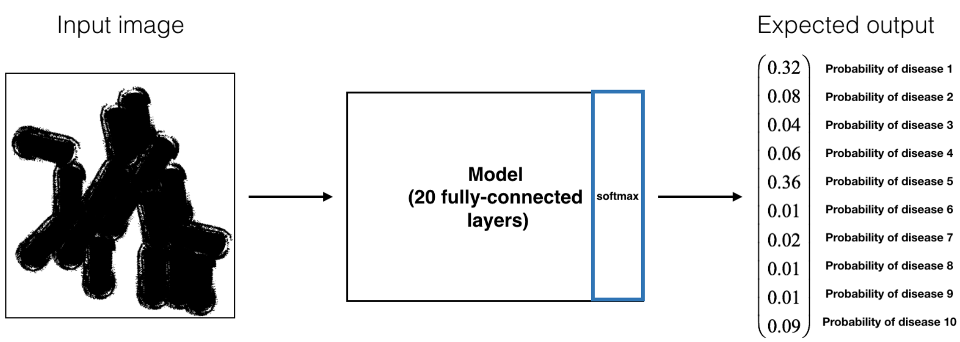


Figure 6: The model (M2) takes as input a cell image taken from a microscope and outputs a 10-dimensional vector of values indicating the probabilities of presence of each of the 10 considered deceases in the image.

(f) (3 points) Explain in details how you would use transfer learning in this case. If this process adds hyperparameters, describe each one of them.

Solution: Transfer learning is a technique where we can use the weights of model M2 in our model M1. This is possible because M1 and M2 are trained on the same kind of input, and will likely learn similar low-level features.

Performing transfer learning from M2 to M1 means taking the parameters (and architecture) of model M2 up to the l^{th} layer, and stack l' randomly initialized layers along with a classification head with n_y classes (for our predictions). Among the l layers taken from M2, we can freeze the first l_f layers and retrain the rest of them.

We will choose these hyperparameters (l, l', l_f) by tuning on the dev set.

You now have a trained model, you would like to define a metric computing the per-pixel accuracy. An accuracy of 78% means that 78% of the pixels have been classified correctly by the model, while 22% of the pixels have been classified incorrectly.

(g) (2 points) Write down the formula defining the accuracy for a single example using the labels y and the output of the softmax $\hat{y}$.

Solution: If the probability output p_j of your network for pixel j is more than 0.5, you predict 1 ($= \hat{y}_j$). Otherwise you predict 0.
$\frac{1}{n_y}\sum_{i=1}^{n_y} 1(y_i = \hat{y}_i)$

Your model's accuracy on the test set is 96%. You thus decide to present the model to the doctors. They are very unhappy, and argue that they would like to visually distinguish cells. They show you the following prediction output, which does not clearly separate distinct cells.

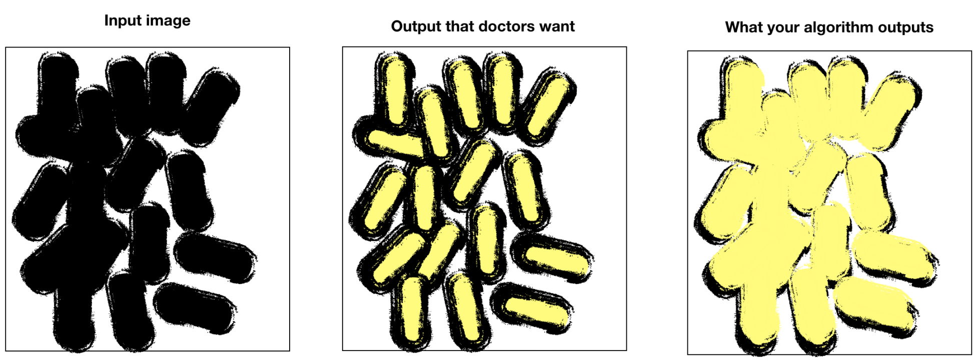


Figure 7: On the left, the input image. In the middle, the type of segmentation that would have satisfied the doctors. On the right, the output of your algorithm which struggles to separate distinct cells.

(h) (4 points) How can you correct your model and/or dataset to satisfy the doctors' request? Explain in details.

Solution: Modify the dataset in order to label the boundaries between cells. On top of that, change the loss function to give more weight to boundaries or penalize false positives.

You have solved the problem, the doctors are really satisfied. They have a new task for you: a binary classification. They give you a dataset containing images similar to the previous ones. The difference is that each image is labeled as 0 (there are no cancer cells on the image) or 1 (there are cancer cells on the image).
Strong from your previous experience with neural networks, you easily build a state-of-the-art model to classify these images with 99% accuracy. The doctors are astonished and surprised, they ask you to explain your network's predictions. More specifically,

(i) (3 points) Given an image classified as 1 (cancer present), can you figure out based on which cell(s) the model predicted 1? Explain.

> **Solution:** Gradient of output w.r.t. input X (saliency map), occlusion of cells/regions to observe difference in prediction, class activation maps. Partial for mentioning largest activations, weights.
> NOTE: For Winter '18 due to ambiguity ('can you' vs 'how would you'), we gave partial credit to all answers with logically correct explanations.

Your model detects cancer on cells (test set) images with 99% accuracy, while a doctor would on average perform 97% accuracy on the same task.

(j) (3 points) Is this possible? Explain.

> **Solution:** A neural network will very unlikely achieve better accuracy than the human who labeled the data. If the dataset was entirely labeled by this one doctor with 97% accuracy, it is unlikely that the model can perform at 99% accuracy.
>
> However in the more probable case where the data is annotated by multiple doctors, the network will learn from these several doctors and be able to outperform the one doctor with 97% accuracy. In this case, a panel composed of the doctors who labeled the data would likely perform at 99% accuracy or higher.

Question 8 (AlphaTicTacToe Zero)

DeepMind recently invented AlphaGo Zero – a Go player that learns purely through self-play and without any human expert knowledge. AlphaGo Zero was able to impressively defeat their previous player AlphaGo, which was trained on massive amounts of human expert games. AlphaGo in its turn had beat the human world Go champion- a task conceived nearly impossible 2 years ago! Unsurprisingly, at its core, AlphaGo Zero uses a neural network.

Your task is to build a neural network that we can use as a part of AlphaTicTacToe Zero. The board game of TicTacToe uses a grid of size 3×3, and players take turns to mark an $\times$ (or $\bigcirc$) at any unoccupied square in the grid until either player has 3 in a row or all nine squares are filled.

(a) (2 points) The neural network we need takes the grid at any point in the game as the input. Describe how you can convert the TicTacToe grid below into an input for a neural network.

$\times$	$\bigcirc$	
	$\times$	$\times$
	$\bigcirc$	

Solution: Possible answers: vector of 9 values (+1 for $\times$, -1 for $\bigcirc$, 0 for empty) OR equivalent matrix/vector of 18/27 values, or a similar scheme. Two examples:

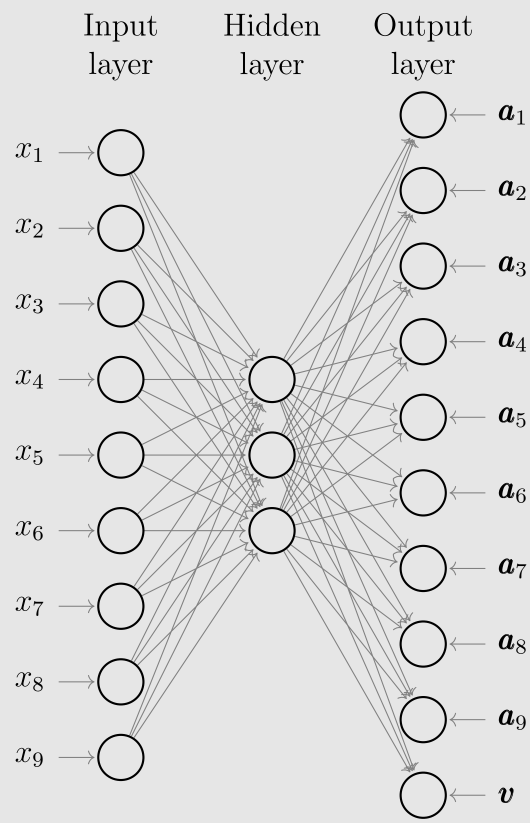
[1,-1,0,0,1,1,0,-1,0]

1	-1	0
0	1	1
0	-1	0

(b) (3 points) The neural network we require has 2 outputs. The first is a vector $\vec{a}$ of 9 elements, where each element corresponds to one of the nine squares on the grid. The element with the highest value corresponds to the square which the current player should play next. The second output is a single scalar value $\boldsymbol{v}$, which is a continuous value in [-1,1]. A value closer to 1 indicates that the current state is favorable for the current player, and -1 indicates otherwise.

Roughly sketch a fully-connected single hidden layer neural network (hidden layer of size 3) that takes the grid as input (in its converted form using the scheme described in part (a)) and outputs $\vec{a}$ and $\boldsymbol{v}$. In your sketch, clearly mark the input layer, hidden layer and the outputs. You need not draw all the edges between two layers, but make sure to draw all the nodes. Remember, the same neural network must output both $\vec{a}$ and $\boldsymbol{v}$.

Solution:



Input as in previous part, hidden layer of size 3. Output layer should have 9 nodes marked for $\vec{a}$ and 1 node for $\boldsymbol{v}$. (3 points)

(c) (i) (1 point) As described above, each element in the output $\vec{\boldsymbol{a}}$ corresponds to a square on the grid. More formally, $\vec{\boldsymbol{a}}$ defines a probability distribution over the 9 possible moves, with higher probability assigned to the better move. What activation function should be used to obtain $\vec{\boldsymbol{a}}$?

Solution: Softmax

(ii) (1 point) The output $\boldsymbol{v}$ is a single scalar with value in [-1,1]. What activation function should be used to obtain $\boldsymbol{v}$?

Solution: tanh

(d) (4 points) During the training, given a state t of the game (grid), the model predicts $\vec{\boldsymbol{a}}^{<t>}$ (vector of probabilities) and $\boldsymbol{v}^{<t>}$. Assume that for every state t of the game, someone has given you the best move $\vec{\boldsymbol{y_a}}^{<t>}$ to do (one-hot vector) and the corresponding value $\boldsymbol{y_v}^{<t>}$ for $\boldsymbol{v}^{<t>}$

In terms of $\vec{\boldsymbol{a}}^{<t>}$, $\boldsymbol{v}^{<t>}$, $\vec{\boldsymbol{y_a}}^{<t>}$ and $\boldsymbol{y_v}^{<t>}$, propose a valid loss function for training on a single game with T steps. Explain your choice.

Solution: $L = \sum_{t=1}^{T} -\vec{\boldsymbol{y_a}}^{<t>} \cdot \log(\vec{\boldsymbol{a}}^{<t>}) + (\boldsymbol{y_v}^{<t>} - \boldsymbol{v}^{<t>})^2$ (3 points)

Since $\vec{\boldsymbol{a}}^{<t>}$ specifies a probability distribution, we use the cross-entropy loss. We use a squared loss for $\boldsymbol{v}^{<t>}$. L1 loss and a modified cross entropy loss are also acceptable for $\boldsymbol{v}^{<t>}$. If a cross entropy loss is used for $\boldsymbol{v}^{<t>}$, then $\boldsymbol{v}^{<t>}$ must be rescaled to $[0, 1]$. The loss is summed over T steps.

Question 9 (Practical industry-level questions)

You want to solve the following problem: *build a trigger word detection algorithm to spot the word* ***cardinal*** *in a 10 second long audio clip.*

(a.) (4 points) Explain in a short paragraph what is the best practice for building the dataset.

Solution:
To build the training set:

* Data collection can be done by collecting clips of positive ("cardinal") and negative (other words) words, as well as background noise clips. Keep these in 3 separate files.
* Data synthesis can be then performed by overlaying words from the positive/negative clips on the background clips. The labels can be placed simultaneously because the insertion index of the positive word is known (chosen).

Building the dev/test set is different because it needs to represent real conditions:

* Record 10sec audio clips with positive and negative words.
* label by hand

(b.) (2 points) Give 2 pros and 2 cons of embedding your model on a smartphone device instead of using it on a server.

Solution: Pros:

* faster to predict
* works offline

Cons:

* model is heavy, can take smartphone's memory
* it is harder to update the model than if it was on a server

(c.) (2 points) You have coded a model and trained on the audio dataset you have built, the training accuracy indicates that there is a problem. Other than spending time checking your code, what is a good strategy to quickly know if the problem is due to an error in your code, or from the fact that your model is not complex/deep enough to understand the dataset?

Solution: Try to overfit 1 training example. Failing to overfit 1 training example is a huge hint that your code has a problem.

Extra Page 1/3

Extra Page 2/3

Extra Page 3/3

END OF PAPER