CS230: Deep Learning

Fall Quarter 2018

Stanford University

Midterm Examination

180 minutes

	Problem	Full Points	Your Score
1	Multiple Choice Questions	10	
2	Short Answer Questions	35	
3	Attacks on Neural Networks	15	
4	Autonomous Driving Case Study	27	
5	Traversability Estimation Using GANs	14	
6	LogSumExp	16	
Total		117	

The exam contains 25 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, 10 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. There is no penalty for a wrong answer.

(a) **(1 point)** Which of the following techniques does NOT prevent a model from overfitting?

(i) Data augmentation

(ii) Dropout

(iii) Early stopping

(iv) None of the above

Solution: (iv)

(b) **(3 points)** Consider the following data sets:

- $X_{\text{train}} = (x^{(1)}, x^{(2)}, ..., x^{(m_{\text{train}})}), Y_{\text{train}} = (y^{(1)}, y^{(2)}, ..., y^{(m_{\text{train}})})$
- $X_{\text{test}} = (x^{(1)}, x^{(2)}, ..., x^{(m_{\text{test}})}), Y_{\text{test}} = (y^{(1)}, y^{(2)}, ..., y^{(m_{\text{test}})})$

You want to normalize your data before training your model. Which of the following propositions are true? **(Circle all that apply.)**

(i) The normalizing mean and variance computed on the training set, and used to train the model, should be used to normalize test data.

(ii) Test data should be normalized with its own mean and variance before being fed to the network at test time because the test distribution might be different from the train distribution.

(iii) Normalizing the input impacts the landscape of the loss function.

(iv) In imaging, just like for structured data, normalization consists in subtracting the mean from the input and multiplying the result by the standard deviation.

Solution: (i) and (iii)

(c) **(2 points)** Which of the following is true, given the optimal learning rate?

(i) Batch gradient descent is always guaranteed to converge to the global optimum of a loss function.

(ii) Stochastic gradient descent is always guaranteed to converge to the global optimum of a loss function.

(iii) For convex loss functions (i.e. with a bowl shape), batch gradient descent is guaranteed to eventually converge to the global optimum while stochastic gradient descent is not.

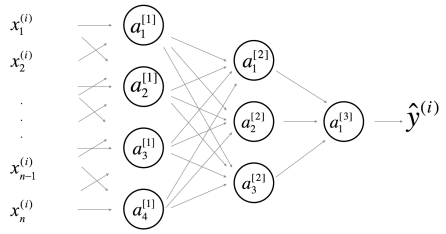
(iv) For convex loss functions (i.e. with a bowl shape), stochastic gradient descent is guaranteed to eventually converge to the global optimum while batch gradient descent is not.

(v) For convex loss functions (i.e. with a bowl shape), both stochastic gradient descent and batch gradient descent will eventually converge to the global optimum.

(vi) For convex loss functions (i.e. with a bowl shape), neither stochastic gradient descent nor batch gradient descent are guaranteed to converge to the global optimum.

Solution: (iii)

(d) **(1 point)** You design the following 2-layer fully connected neural network.



All activations are sigmoids and your optimizer is stochastic gradient descent. You initialize all the weights and biases to zero and forward propagate an input $x \in \mathbb{R}^{n \times 1}$ in the network. What is the output $\hat{y}$?

(i) -1

(ii) 0

(iii) 0.5

(iv) 1

Solution: (iii)

(e) **(1 point)** Consider the model defined in question (d) with parameters initialized with zeros. $W^{[1]}$ denotes the weight matrix of the first layer. You forward propagate a batch of examples, and then backpropagate the gradients and update the parameters. Which of the following statements is true?

(i) Entries of $W^{[1]}$ may be positive or negative

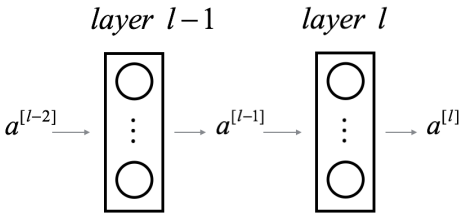
(ii) Entries of $W^{[1]}$ are all negative

(iii) Entries of $W^{[1]}$ are all positive

(iv) Entries of $W^{[1]}$ are all zeros

Solution: (i)

(f) **(2 points)** Consider the layers l and $l-1$ in a fully connected neural network:



The forward propagation equations for these layers are:

$$
\begin{aligned}
z^{[l-1]} &= W^{[l-1]}a^{[l-2]} + b^{[l-1]} \\
a^{[l-1]} &= g^{[l-1]}(z^{[l-1]}) \\
z^{[l]} &= W^{[l]}a^{[l-1]} + b^{[l]} \\
a^{[l]} &= g^{[l]}(z^{[l]})
\end{aligned}
$$

Which of the following propositions is true? Xavier initialization ensures that :

(i) $Var(W^{[l-1]})$ is the same as $Var(W^{[l]})$.

(ii) $Var(b^{[l]})$ is the same as $Var(b^{[l-1]})$.

(iii) $Var(a^{[l]})$ is the same as $Var(a^{[l-1]})$, at the end of training.

(iv) $Var(a^{[l]})$ is the same as $Var(a^{[l-1]})$, at the beginning of training.

Solution: (iv)

Question 2 (Short Answer Questions, 35 points)

Please write concise answers.

(a) **(2 points)** You are training a logistic regression model. You initialize the parameters with 0's. Is this a good idea? Explain your answer.

Solution: There is no symmetry problem with this approach. In logistic regression, we have $a = Wx + b$ where a is a scalar and W and x are both vectors. The derivative of the binary cross entropy loss with respect to a single dimension in the weight vector $W[i]$ is a function of $x[i]$, which is in general different than $x[j]$ when $i \neq j$.

(b) **(2 points)** You design a fully connected neural network architecture where all activations are sigmoids. You initialize the weights with large positive numbers. Is this a good idea? Explain your answer.

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.

(c) **(2 points)** You are given a dataset of 10×10 grayscale images. Your goal is to build a 5-class classifier. You have to adopt one of the following two options:

- the input is flattened into a 100-dimensional vector, followed by a fully-connected layer with 5 neurons
- the input is directly given to a convolutional layer with five 10×10 filters

Explain which one you would choose and why.

Solution: The 2 approaches are the same. But the second one seems better in terms of computational costs (no need to flatten the input). We accept the answer "the 2 approaches are the same".

(d) **(2 points)** You are doing full batch gradient descent using the entire training set (not stochastic gradient descent). Is it necessary to shuffle the training data? Explain your answer.

Solution: It is not necessary. Each iteration of full batch gradient descent runs through the entire dataset and therefore order of the dataset does not matter.

(e) **(2 points)** You would like to train a dog/cat image classifier using mini-batch gradient descent. You have already split your dataset into train, dev and test sets. The classes are balanced. You realize that within the training set, the images are ordered in such a way that all the dog images come first and all the cat images come after. A friend tells you: "you absolutely need to shuffle your training set before the training procedure."

Is your friend right? Explain.

Solution: Yes, there is a problem. The optimization is much harder with mini-batch gradient descent because the loss function moves by a lot when going from the one type of image to another.

(f) **(2 points)** You want to evaluate the classifier you trained in (e). Your test set $(\mathcal{X}_{\text{test}}, \mathcal{Y}_{\text{test}})$ is such that the first m_1 images are of dogs, and the remaining images are of cats. After shuffling $\mathcal{X}_{\text{test}}$ and $\mathcal{Y}_{\text{test}}$, you evaluate your model on it to obtain a classification accuracy $a_1\%$. You also evaluate your model on $\mathcal{X}_{\text{test}}$ and $\mathcal{Y}_{\text{test}}$ without shuffling to obtain accuracy $a_2\%$. What is the relationship between a_1 and a_2 ($>$, $<$, $=$, $\leq$, $\geq$)? Explain.

Solution: $a_1 = a_2$ When evaluating on the test set, the only form of calculation that you do is a single metric (e.g. accuracy) on the **entire** test set. The calculation of this metric on the entire test set does not depend on the ordering.

(g) **(2 points)** Data augmentation is often used to increase the amount of data you have. Should you apply data augmentation to the test set? Explain why.

Solution: Both answers are okay but need to be justified. If no, then explain that we want to test on real data only. If yes, then explain in which situation doing data augmentation on test set might make sense (e.g. as an ensemble approach in image classifiers).

(h) **(2 points)** Weight sharing allows CNNs to deal with image data without using too many parameters. Does weight sharing increase the bias or the variance of a model?

Solution: Increases bias.

(i) **(2 points)** You'd like to train a fully-connected neural network with 5 hidden layers, each with 10 hidden units. The input is 20-dimensional and the output is a scalar. What is the total number of trainable parameters in your network?

Solution: (20+1)*10 + (10+1)*10*4 + (10+1)*1

(j) **(3 points)** Consider the figure below:

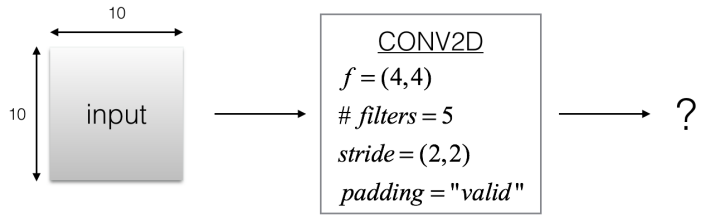


Figure 1: Input of shape $(n_H, n_W, n_C) = (10, 10, 1)$; There are five 4×4 convolutional filters with 'valid' padding and a stride of $(2, 2)$

What is the output shape after performing the convolution step in Figure 1? Write your answer in the following format: (n_H, n_W, n_c).

Solution: (h=4,w=4,c=5)

(k) **(2 points)** Recall that $\sigma(z) = \frac{1}{1+e^{-z}}$ and $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. Calculate $\frac{\partial \sigma(z)}{\partial z}$ in terms of $\sigma(z)$ and $\frac{\partial tanh(z)}{\partial z}$ in terms of $tanh(z)$.

Solution: Gradient for sigmoid: $\sigma(z) * (1 - \sigma(z))$
Gradient for tanh: $1 - tanh^2(z)$

(l) **(2 points)** Assume that before training your neural network the setting is:
(1) The data is zero centered.
(2) All weights are initialized independently with mean 0 and variance 0.001.
(3) The biases are all initialized to 0.
(4) Learning rate is small and cannot be tuned.

Using the result from (k), explain which activation function between *tanh* and *sigmoid* is likely to lead to a higher gradient during the first update.

Solution: *tanh*. During initialization, expected value of z is 0.
Derivative of σ w.r.t. z evaluated at zero $= 0.5 * 0.5 = 0.25$.
Derivative of *tanh* w.r.t. z evaluated at zero $= 1$.
tanh has higher gradient magnitude close to zero.

(m) You want to build a 10-class neural network classifier, Given a cat image, you want to classify which of the 10 cat breeds it belongs to.

(i) **(2 points)** What loss function do you use? Introduce the appropriate notation and write down the formula of the loss function.

Solution: You would want to use the cross entropy loss given by $\mathcal{L} = -\sum_{i=1}^{n} y_i log(\hat{y}_i)$

(ii) **(2 points)** Assuming you train your network using mini-batch gradient descent with a batch size of 64, what cost function do you use? Introduce the appropriate notation and write down the formula of the cost function.

Solution: If there are m training examples, $\mathcal{J} = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}^{(i)}$

(iii) **(3 points)** One of your friends has trained a cat vs. non-cat classifier. It performs very well and you want to use transfer learning to build your own model. Explain what additional hyperparameters (due to the transfer learning) you will need to tune.

Solution: The parameters you would need to choose are: 1) How many layers of the original network to keep. 2) How many new layers to introduce 3) How many of the layers of the original network would you want to keep frozen while fine tuning.

(n) **(3 points)** A binary classification problem could be solved with the two approaches described below:

Approach 1: Simple Logistic Regression (one neuron)
Your output will be $\hat{y} = \sigma(W_l x + b_l)$
Classify as 0 if $\hat{y} \leq 0.5$ and 1 otherwise.

Approach 2: simple softmax regression (two neurons)
Your output will be $\hat{y} = softmax(W_s x + b_s) = [\hat{y}_1, \hat{y}_2]^T$
Classify as 0 if $\hat{y}_1 \geq \hat{y}_2$ and 1 otherwise.

Approach 2 involves twice as many parameters as approach 1. Can approach 2 learn more complex models than approach 1?
If yes, give the parameters (W_s, b_s) of a function that can be modeled by approach 2 but not by approach 1. If no, show that (W_s, b_s) can always be written in terms of (W_l, b_l).

Solution: No. For approach 1, the classifier is $\sigma(w_1 x + b_1) \geq 0.5$ which is equivalent to $w_1 x + b_1 \geq 0$.

Now, for approach 2, define w_2^1 as the weights for the first output and w_2^2 as the weights for the second output. Similarly, define b_2^1 as the bias for the first output and b_2^2 as the bias for the second output.

For approach 2, the classifier is $\frac{exp(w_2^1 x + b_2^1)}{exp(w_2^1 x + b_2^1) + exp(w_2^2 x + b_2^2)} \geq \frac{exp(w_2^2 x + b_2^2)}{exp(w_2^1 x + b_2^1) + exp(w_2^2 x + b_2^2)}$.
Since the denominator is positive and same on both sides, this is equivalent to $exp(w_2^1 x + b_2^1) \geq exp(w_2^2 x + b_2^2)$.
Since exp is a monotonic increasing function, this is equivalent to $w_2^1 x + b_2^1 \geq w_2^2 x + b_2^2$.
This is equivalent to $(w_2^1 - w_2^2)x + (b_2^1 - b_2^2) \geq 0$.
Given any w_2 and b_2 in approach 2, we can get the exact same model by setting $w_1 = w_2^1 - w_2^2$ and $b_1 = b_2^1 - b_2^2$ in approach 1.

Alternatively, one can argue based on degrees of freedom.

Note that in this case, we are asking about model complexity, which is independent of loss function.

Question 3 (Attacks on Neural Networks, 15 points)

Alice and Bob are deep learning engineers working at two rival start-ups. They are both trying to deliver the same neural network-based product.

Alice and Bob do not have access to each other's models and code. However they can query each other's models as much as they'd like.

(a) **(2 points)** Name a type of neural network attack that Alice cannot use against Bob's model, and explain why it cannot be used in this case.

Solution: White-box attacks
White box attacks require access to the weights of the model whereas black box attacks do not.

(b) **(3 points)** How can Alice forge an image x_{iguana} which looks like an iguana but will be wrongly classified as a plant by Bob's model? Give an iterative method and explicitly mention the loss function.

Solution: $L = ||\hat{y} - y_{iguana}|| + \gamma||x - x_{plant}||$

(c) **(3 points)** It is possible to add an invisible perturbation η to an image x, such that $\tilde{x} = x + \eta$ is misclassified by a model. Assuming you have access to the target model, explain how you would find η.

Solution: η can be chosen using a method such as the Fast Gradient Sign Based Method. It is an iterative method that requires access to the model (a white-box attack)

(d) **(2 points)** Given that you have obtained η, you notice that $|\eta| << 1$. Explain why even though the adversarial noise has a small magnitude, it can cause a large change in the output of a model.

Solution: Since the dimensionality of the images is very large, even though the noise is small, it can cause a large swing in the output. To illustrate consider $\tilde{x} = x + \eta$. While passing this through a single layer, $W\tilde{x} = Wx + W\eta = \sum_j W_{ij}\eta_j$. If j, is very large, this can have a significant contribution.

(e) **(3 points)** Alice doesn't have access to Bob's network. How can she still generate an adversarial example using the method described above?

Solution: Adversarial examples are transferable, and so Alice can use Fast Gradient Sign Based Method on a different model, built for the same task (cat vs. non-cat classification), and it is likely that this adversarial example will also be misclassified by Bob's model.

(f) **(2 points)** To defend himself against Alice's attacks, Bob is thinking of using dropout. Dropout randomly shuts down certain neurons of the network, and makes it more robust to changes in the input. Thus, Bob has the intuition that the network will be less vulnerable to adversarial examples. Is Bob correct? Explain why or why not.

Solution: No, dropout isn't used at test time while Alice will forge her adversarial examples by querying a network at test time.

Question 4 (Autonomous driving case study, 27 points)

As the CEO of an autonomous driving company Potato Inc., you are trying to build a pipeline to predict a car's steering angle $\theta \in [-1, 1]$ from an input image.

(a) After setting up a camera on the hood of your car, an expert driver in your team has collected the dataset $\mathcal{X}$ in a city where most roads are straight. She always drives in the center of the lane.

(i) **(2 points)** Describe two problems that can occur if you train a model on $\mathcal{X}$ and test it on real-world data.

Solution: System hasn't seen situations where you're not driving sensibly, or away from the centre of the lane. This can lead to an accumulation of errors if your model output slightly differs from the correct output at a given time step. Also, a majority of outputs from the steering will be 0, if the driver was always in the center of the lane, and most of the roads were straight.

(ii) **(3 points)** Without collecting new data, how can you help your model generalize to real world data? Describe the details of how you would implement the approach.

Solution: Data augmentation. Slight left and right translations of the images, with a corresponding change in steering angle. For example, if you translate an image to the right, add some small quantity ϵ to the steering angle (if positive implies steering to the left), so that you're now steering to the left.

(b) **(2 points)** Give two reasons why you wouldn't use an end-to-end model to predict the steering direction (θ) from the input image.

Solution: 1 - If there's not enough data. 2 - If it is easier to collect data for submodules than end-to-end data. For instance, collecting high variance data while driving a car around with a camera on the hood is costly. Finding images with cars/pedestrians and labelling with bounding boxes might be easier, and helps train "car detector" and "pedestrian detector" submodules.

(c) You finally design the following pipeline:

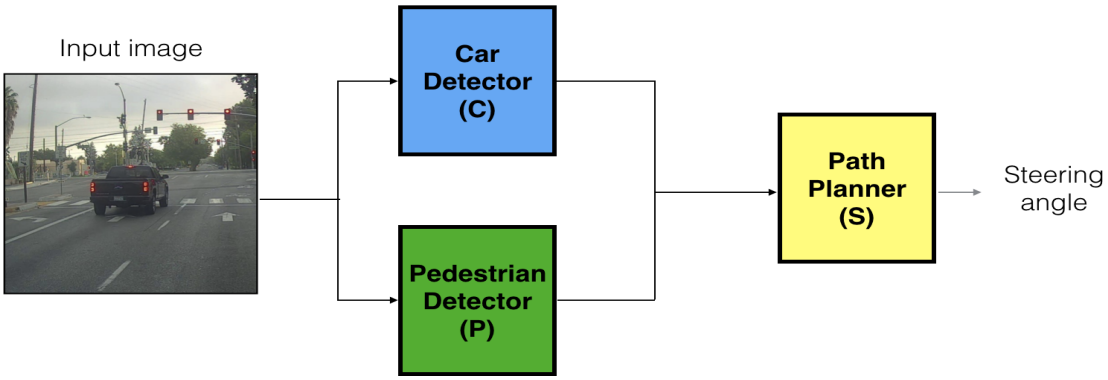


Figure 2: The input camera image is given to two modules: the Car Detector $\mathcal{C}$ and the Pedestrian Detector $\mathcal{P}$. $\mathcal{C}$ outputs a set of bounding boxes localizing the cars. $\mathcal{P}$ outputs a set of bounding boxes localizing the pedestrians. The bounding boxes are then given to a Path Planner $\mathcal{S}$ which outputs the steering angle. Assume all these submodules are supervised learning algorithms.

(i) **(3 points)** What data do you need to train the submodules in the pipeline presented in Figure 2?

Solution:

- To train the Car Detector, we need to collect X_c (images from a camera hood of a car) and Y_c (bounding box labels localizing the cars.)
- To train the Pedestrian Detector, we need to collect X_p (images from a camera hood of a car) and Y_p (bounding box labels localizing the pedestrians.)
- To train the Path planner, we need to collect X_s (bounding boxes localizing the cars and the pedestrians) and Y_s (steering angle.)

(ii) **(3 points)** Explain how you would collect this data.

Solution:

- (X_c, Y_c) : Put a camera on the hood of a car, label the bounding boxes by hand. You can also download images from roads online or use online datasets such as COCO or PASCAL VOC.
- (X_p, Y_p) : Put a camera on the hood of a car, label the bounding boxes by hand. You can also download images from roads online or use online datasets such as COCO or PASCAL VOC.
- (X_s, Y_s) : Put a camera on the hood of a car, label the bounding boxes by

hand. Track the steering angle θ using a sensor in the car. Note that the most challenging to train seems to be the path planner. You collect images of roads with and without cars and pedestrians. Each image should be labelled with bounding boxes around cars and pedestrians, and indicate the true steering angle.
You can thus set-up a camera on the hood of your car and a sensor capturing the steering angle. Drive it in various environment, track the live variations in steering angle mapped to the camera's video stream. Finally, label the video frames with bounding boxes around pedestrians and cars.
To boost the performance of your car detector and pedestrian detector, you can also add images from other sources such as PASCAL VOC and COCO which have bounding box labels.

(d) **(2 points)** Propose a metric to measure the performance of your pipeline model.

Solution:

- Sum of absolute deviations (i.e. L1 distance) between ground truth and predicted steering angle.
- Sum of squared errors (i.e. L2 distance) between ground truth and predicted steering angle.
- (not expected) You can also design metrics for submodules. For $\mathcal{C}$ and $\mathcal{P}$ it might be mAP (/mean IoU)

(e) Assume that you have designed a metric that scores your pipeline between 0% (bad) and 100% (good.) On unseen data from the real world, your entire pipeline gets a score of 54%.

(i) **(2 points)** Define Bayes error and human level error. How do these two compare with each other? ($\leq$, $\geq$, $=$)

Solution: Bayes error is a lower bound on the minimum error that can be achieved. Human level error is the error achieved by an expert human on the same task. The Bayes error is $\leq$ human level error.

(ii) **(2 points)** How would you measure human level error for your autonomous driving task?

Solution: Possible solution: Create a simulator with the same path followed by the car while recording data, and have an expert try following the path.

Calculate the error between the original path and the expert's new path to get an estimate of human level error on this task.

(iii) **(3 points)** Human level performance is 96% for your task. Your pipeline should do a lot better! Assuming one component (among $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{S}$) is failing, explain how you would identify which one it is.

Solution: Replace every component in turn with the ground truth. For example if you are testing whether $\mathcal{C}$ is failing, supply $\mathcal{S}$ with ground truth bounding boxes of all the cars, and see if performance of the overall pipeline increases. If it increases significantly, it is likely that the failing component is $\mathcal{C}$.

(iv) **(3 points)** You have identified that the failing component is $\mathcal{P}$. $\mathcal{P}$ fails at detecting pedestrians in 3 distinct settings: at night, when it is snowing, and when it is raining. You cannot solve the 3 problems at once. How would you decide which of the 3 problems to focus on first?

Solution: Do an error analysis. Ascertain how much of your error is due to each of the three cases, by using images only from these cases. Then focus on the case which contributes the most to your error. You could also choose to based your choice in practical situations. If Potato Inc. is in a place which has high rainfall, you might want to focus on the errors made on images with rain.

(f) **(2 points)** After fixing the only failing component, your pipeline gets a score of 67%. What could be wrong?

Solution: Each of the modules has been trained independently and on perfect outputs from the other components. It is likely that small perturbations/errors in the output of previous components is affecting the performance of the other components, leading to a snowballing of errors.

Question 5 (Traversability Estimation Using GANs, 14 points)

In robot navigation, the traversability problem aims to answer the question: can the robot traverse through a situation?

Figure 3: Example of different situations. *Left*: traversable; *Right*: non-traversable

You want to estimate the traversability of a situation for a robot. Traversable data is easy to collect (e.g. going through a corridor) while non-traversable data is very costly (e.g. going down the stairs). You have a **large and rich** dataset $\mathcal{X}$ of traversable images, but no non-traversable images.

The question you are trying to answer is: 'Is it possible to train a neural network that classifies whether or not a situation is traversable using only dataset $\mathcal{X}$?' More precisely, if a non-traversable image was fed into the network, you want the network to predict that it is non-traversable. In this part, you will use a Generative Adversarial Network (GAN) to solve this problem.

(a) Before considering the traversability problem, let us do a warm-up question. Consider that you have trained a network $f_w : \mathbb{R}^{n_x \times 1} \to \mathbb{R}^{n_y \times 1}$. The parameters of the network are denoted w. Given an input $x \in \mathbb{R}^{n_x \times 1}$, the network outputs $\hat{y} = f_w(x) \in \mathbb{R}^{n_y \times 1}$.

Given an arbitrary output $\hat{y}^*$, you would like to use gradient descent optimization to generate an input x^* such that $f_w(x^*) = \hat{y}^*$.

(i) **(2 points)** Write down the formula of the l_2 loss function you would use.

Solution: $\mathcal{L}(x^*, \hat{y}) = \|f_w(x^*) - \hat{y}^*\|_2^2$

(ii) **(2 points)** Write down the update rule of the gradient descent optimizer in terms of l_2 norm.

Solution: $x^*_{t+1} = x^*_t - \alpha \cdot \frac{\partial \|f_w(x) - \hat{y}^*\|_2^2}{\partial x}|_{x=x^*_t}$

(iii) **(2 points)** Calculate the gradient of the loss in your update rule in terms of $\frac{\partial f_w(x)}{\partial w}$, and $\frac{\partial f_w(x)}{\partial x}$ (it is not necessary to use both terms).

Solution: $x^*_{t+1} = x^*_t - 2\alpha \cdot \left(\frac{\partial f_w(x)}{\partial x}|_{x=x^*_t}\right)^T \cdot (f_w(x^*_t) - \hat{y}^*)$

(b) Now, let us go back to the traversability problem. In particular, imagine that you have successfully trained a **perfect** GAN with generator G and discriminator D on $\mathcal{X}$. As a consequence, given a code $z \in \mathbb{R}^C$, $G(z)$ will look like a traversable image.

(i) **(2 points)** Consider a new image x. How can you find a code z such that the output of the generator $G(z)$ would as close as possible to x?

Solution: We can apply the backpropagation technique developed in part (a), where z plays the role of x^*, and x plays the role of y^*.

(ii) **(2 points)** Suppose you've found z such that $G(z)$ is the closest possible value to x out of all possible z. How can you decide if x represents a traversable situation or not? Give a qualitative explanation.

Solution: We compare $G(z)$ to the image x in the sense that if $\|G(z) - x\|_2$ is "big" then x is non-traversable, and vice versa.

(iii) **(2 points)** Instead of using the method above, Amelia suggests directly running x through the discriminator D. Amelia believes that if $D(x)$ predicts that it is a real image, then x is likely a traversable situation. Else, it is likely to be a non-traversable situation. Do you think that Amelia's method would work and why?

Solution: The reason is if the GAN is trained perfectly, which is the case here, then the discriminator cannot tell if a generated image by the generator is real or fake, that is, traversable or non-traversable.

(c) **(2 points)** The iterative method developed in part (a) is too slow for your self-driving application. Given $\hat{y}^*$ you need to generate x^* in real time. Come up with a method using an additional network to generate x^* faster, e.g., with a single forward propagation.

Solution: We can train a second network to spit out the inverse of the network in hand.

Some ideas for this question are borrowed from the paper titled ***GONet: A Semi-Supervised Deep Learning Approach For Traversability Estimation*** *(Hirose et al.).*

Question 6 (LogSumExp, 16 points)

Consider training a neural network for K-class classification using softmax activation and cross-entropy (CE) loss (see Fig. 4).

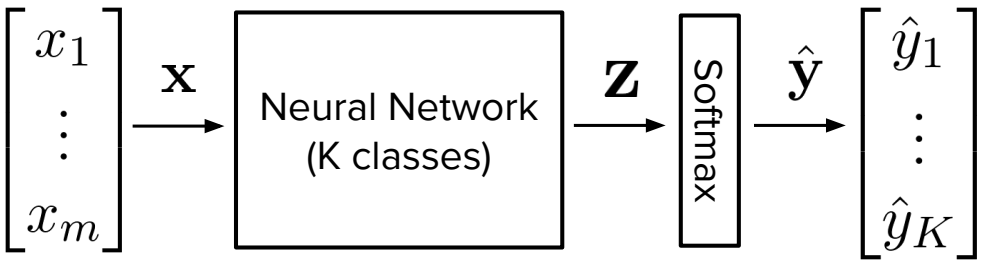


Figure 4: Diagram of the K-class classification network.

For a given input $\mathbf{x}$ and its one-hot encoded label $\mathbf{y} \in \{0,1\}^K$, the network outputs a logits (i.e. pre-softmax) vector $\mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$. Assume that the correct class is c ($y_c = 1$ and $y_i = 0$ for all $i \neq c$).
Recall the following definitions:

- softmax:
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}) = \left\langle \frac{\exp(z_1)}{Z}, \ldots, \frac{\exp(z_K)}{Z} \right\rangle$$
where $Z = \sum_{i=1}^{K} \exp(z_i)$ is the normalizing constant.

- Cross-Entropy Loss
$$\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

(a) **(1 point)** What problem would arise if the predicted logits $\mathbf{z}$ contain very large values? (Hint: consider evaluating the CE-loss for $\mathbf{z} = \langle 100000, 100000, 100000 \rangle$)

Solution: Due to the exp function, numerical overflow may occur because exp(100000) is too large.

In the following questions, you will express the cross-entropy loss in a different way.

(b) **(2 points)**

LogSumExp (LSE), defined below, is an operation commonly encountered in Deep Learning:

$$\text{LSE}(x_1, \ldots, x_n) = \log \sum_{i=1}^{n} \exp(x_i) = \log(\exp(x_1) + \cdots + \exp(x_n)) \tag{1}$$

Express the loss $\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y})$ in terms of the logits vector $\mathbf{z}$ and the LSE function. (Hint: $\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$)

Solution:

$$\begin{aligned}
\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y}) &= -\log \hat{y}_c \\
&= -\log \frac{\exp(z_c)}{Z} \\
&= -\log(\exp(z_c)) + \log(Z) \\
&= -z_c + \text{LSE}(\mathbf{z})
\end{aligned}$$

Thus, $\text{LSE}(\mathbf{z}) - z_c$

(c) **(2 point)** Compute the following partial derivative:

$$\frac{\partial}{\partial z_j} \text{LSE}(\mathbf{z})$$

Solution: $\dfrac{\exp(z_j)}{\sum_{i=1}^{K} \exp(z_i)}$ or $\hat{y}_j$ or $(\text{softmax}(\mathbf{z}))_j$

(d) **(2 point)** Compute the following partial derivative (for the correct class c):

$$\frac{\partial}{\partial z_c}\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y})$$

(Hint: use Part (b))

Solution: $\hat{y}_c - 1$

(e) **(2 point)** Compute the following partial derivative (for an incorrect class $j \neq c$):

$$\frac{\partial}{\partial z_j}\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y})$$

Solution: $\hat{y}_j$

(f) **(2 points)** Using the results of Part (d) and (e), express the following gradient using $\hat{\mathbf{y}}$ and $\mathbf{y}$:

$$\frac{\partial}{\partial z}\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y})$$

Solution: $\hat{\mathbf{y}} - \mathbf{y}$

(g) **(3 points)** Prove the following identity for some fixed constant $\alpha \in \mathbb{R}$:

$$\text{LSE}(x_1, \ldots, x_n) = \text{LSE}(x_1 - \alpha, \ldots, x_n - \alpha) + \alpha$$

(Hint: $\exp(a + b) = \exp(a)\exp(b)$).

Solution:

$$\begin{aligned}
\text{LSE}(x_1, \ldots, x_n) &= \log \sum_{i=1}^{n} \exp(x_i) \\
&= \log \sum_{i=1}^{n} \exp(\alpha) \cdot \exp(x_i - \alpha) \\
&= \log \left[\exp(\alpha) \sum_{i=1}^{n} \exp(x_i - \alpha) \right] \\
&= \log(\exp(\alpha)) + \log(\sum_{i=1}^{n} \exp(x_i - \alpha)) \\
&= \alpha + \text{LSE}(x_1 - \alpha, \ldots, x_n - \alpha)
\end{aligned}$$

(h) **(2 points)** Explain how the above identity can be used to avoid overflow.

Solution: You can set $\alpha = \max\{x_1, \ldots, x_n\}$ to compute $\text{LSE}(x)$ without overflow.

Extra Page 1/3

Extra Page 2/3

Extra Page 3/3

END OF PAPER