CS230: Deep Learning

Winter Quarter 2020
Stanford University

Midterm Examination

180 minutes

	Problem	Full Points	Your Score
1	Neural Network	12	
2	Loss Functions	12	
3	Optimization	11	
4	Batch Normalization	9	
5	DL Strategy	4	
6	Adversarial Attacks and GANs	6	
7	CNNs	6	
Total		60	

The exam contains 28 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.

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 (Neural Network, 12 points)

You want to build a model to predict whether a startup will raise funding (label = 1) or not (label = 0) in its first year. You have access to a dataset of examples with 300 input features, including the number of founders, the number of employees, the variance in the age of the employees, etc. As a baseline model, you decide to train a logistic regression that outputs a probability $\hat{y}^{(i)} \in [0, 1]$ that a startup described by a vector of features $x^{(i)}$ raises funding in its first year, where the predicted label of an input is chosen to be 1 when $\hat{y}^{(i)} \geq 0.5$ and 0 otherwise.

(a) **(1 point)** If the shape of $x^{(i)}$ is $(n_x, 1)$, what should n_x be? (1 word)

Solution: 300

(b) **(2 points)** A logistic regression has trainable weights w and bias b.

(i) How many weights does the logistic regression model have? (1 word)

Solution: 300 or (300 x 1)

(ii) What is the dimension of b? (1 word)

Solution: 1 or (1 x 1)

(c) **(2 points)** Consider the prediction $\hat{y}^{(i)} \in [0, 1]$.

(i) What is the dimension of $\hat{y}^{(i)}$? (1 word)

Solution: 1 or (1 x 1)

(ii) Write the output $\hat{y}^{(i)}$ in terms of the input $x^{(i)}$ and the parameters w and b. (1 formula)

Solution: $\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$ or $\sigma(wx^{(i)} + b)$.

After training your baseline logistic regression model, you decide to train a single hidden layer neural network with 80 hidden neurons on the same dataset.

(d) **(2 points)** How many weights and biases does this neural network have? (2 sentences)

Solution:
Weight Parameters: 300*80 + 80*1 = 24080 (1 point)
Bias Parameters: 80 + 1 = 81 (1 point)
Total Parameters = 24080 + 81 = 24161

(e) **(1 point)** Recall that a neural network with a single hidden layer is sufficient to approximate any continuous function (with some assumptions on the activation). Why would you use neural networks with multiple layers? (1 sentence)

Solution: Potential answers:
- fewer hidden units, or fewer parameters, or more efficiently (fewer weights)

We did not reward points for saying better to optimize or generalize. We also did not reward points for saying can capture more features or more complex functions.

(f) **(3 point)** While you remembered to use a sigmoid activation in your neural network's output layer, you forgot to use intermediate non-linear activations. In this part, you will demonstrate that your implementation is equivalent to logistic regression.

Let $W^{[l]}$ and $b^{[l]}$ denote the weights and biases respectively of your neural network at layer l, starting at $l = 1$.

(i) Let $\hat{y}^{(i)}$ represent the output probability of your neural network given a feature vector $x^{(i)}$ as input. Write an expression for $\hat{y}^{(i)}$ in terms of $x^{(i)}$ and the neural network parameters. (1 formula)

Solution: $\hat{y}^{(i)} = \sigma(W^{[2]}(W^{[1]}x^{(i)} + b^{[1]}) + b^{[2]})$

(ii) There exists a single logistic regression model with weights w' and bias b' that outputs the same probability that your neural network does for all possible inputs. Write the parameters w' and b' of this logistic regression model in terms of your neural network parameters. (2 equations, one for w', and one for b')

Solution:
$w' = W^{[2]}W^{[1]}$ if the convention being used is that w' is a row vector, $(W^{[2]}W^{[1]})^T$ if it's a column vector
$b' = W^{[2]}b^{[1]} + b^{[2]}$

(g) **(1 point)** You decide to use ReLU as your hidden layer activation, and also insert a ReLU before the sigmoid activation such that $\hat{y} = \sigma(\text{ReLU}(z))$, where z is the pre-activation value for the output layer. What problem are you going to encounter? (1 sentence)

Solution: Since the final ReLU never outputs a negative number, all outputs of the network will always be at least 0.5, so all inputs will be classified as positive.

Question 2 (Loss Functions, 12 points)

Many supermarket customers use the yellow creamy spot on the outside of a watermelon to evaluate its level of sweetness. To help customers who aren't aware of this fact, you decide to build an image classifier to predict whether a watermelon is sweet (label=1) or not (label=0).

(a) **(1 point)** You've built your own labeled dataset, chosen a neural network architecture, and are thinking about using the mean squared error (MSE) loss to optimize model parameters. Give one reason why MSE might not be a good choice for your loss function. (1 sentence)

Solution: When performing binary classification, the outputs are constrained from 0 to 1. When using MSE, we place an upper bound of 1 on the loss, when intuitively it should be infinity (you are being incorrect as possible). BCE does this, and would be a more natural choice.

(b) **(1 point)** You decide to use the binary cross-entropy (BCE) loss to optimize your network. Write down the formula for this loss (for a single example) in terms of the label y and prediction $\hat{y}$. (1 formula)

Solution: $L(y, \hat{y}) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$

(c) **(1 point)** You want to sanity-check your implementation of the BCE loss. What value does the loss take for a prediction of $\hat{y} = 0.9$ on a negative ($y = 0$) example. *You can simply fill in the formula with the numbers, without needing to calculate it.* (1 formula)

Solution: $-\log 0.1$

(d) **(3 points)** Compute the total cost, J, of the network averaged across the following dataset of 3 examples using the binary cross entropy loss. $Y^T = (1,\ 0,\ 0)$, and $\hat{Y^T} = (0.1,\ 0.2,\ 0.7)$. *You can simply fill in the formula, without needing to simplify it. There is no penalty on the weights.* (1 formula)

Solution: $J = -\frac{1}{3}\left(\log 0.1 + \log 0.8 + \log 0.3\right).$

(e) **(3 points)** You add L2 regularization to your loss function.

(i) For a particular trainable weight W, write the update formula for W when the standard gradient descent optimizer is used with L2 regularization. Write your answer in terms of the learning rate α, L2 regularization hyperparameter λ, and the binary cross entropy cost function J_{BCE}. (1 formula)

Solution: $W = (1 - 2\alpha\lambda)W - \alpha\frac{\partial J_{BCE}}{\partial W}$. This answer assumes that the L2 penalty term for this weight is λW^2

(ii) You decide to train one model with L2 regularization (model A) and one without (model B). How would you expect model A's weights to compare to model B's weights? (1 sentence)

Solution: The weights of model A will generally be smaller in magnitude than those of model B.

(iii) Explain one difference between L1 and L2 regularization. (1 sentence)

Solution: L1 leads to weight sparsity, while L2 leads to smaller weights generally but not necessarily with as many weights that are exactly 0. L1 adds a penalty term to the cost function that is proportional to the sum of the absolute values of the weights, while L2 adds one that is proportional to the sum of the squares of the weights.

The price of a watermelon depends on its weight, rather than its level of sweetness. Thus supermarkets don't care about a watermelon's level of sweetness as much as customers do.

Supermarkets give you a new dataset of watermelon images and their corresponding weight in pounds, and ask you to build another image classifier to predict the weight of a watermelon.

(f) **(1 point)** What is one advantage to reusing the weights of your previous sweetness classifier on the new task? (1-2 sentences)

Solution: Shared weights might be useful on both tasks.

(g) **(2 point)** You decide to use a single unified neural network to predict both the level of sweetness and the weight of a watermelon given an image.

(i) Write down the dimension of a new label y for the new network. (1 word)

Solution: 2 (or 2 transpose, or 2×1 or 1×2)

(ii) Propose a new loss function to train the unified model. Assume no regularization and write your answer in terms of the new y and $\hat{y}$. (1 formula)

Solution: An acceptable solution should include an MSE loss term for the weight and a cross entropy term for sweetness.

Question 3 (Optimization, 11 points)

You want to build a classifier that predicts the musical instrument given an audio clip. There are 50 unique types of instruments, and you've collected a dataset of sounds of each, first collecting audio clips of guitars, then clips of violins, and so on for all the different instruments.

(a) **(1 point)** You use batch gradient descent (BGD) to optimize your loss function, but you have been getting poor training loss. You search your code for potential bugs and realize that you're not shuffling the training data. Would shuffling the training fix this problem? Explain your reasoning. (1-2 sentences)

Solution: Shuffling the dataset will not have an impact on the gradients that are used to perform an update, as either way the entire dataset is used (since you are using batch GD).

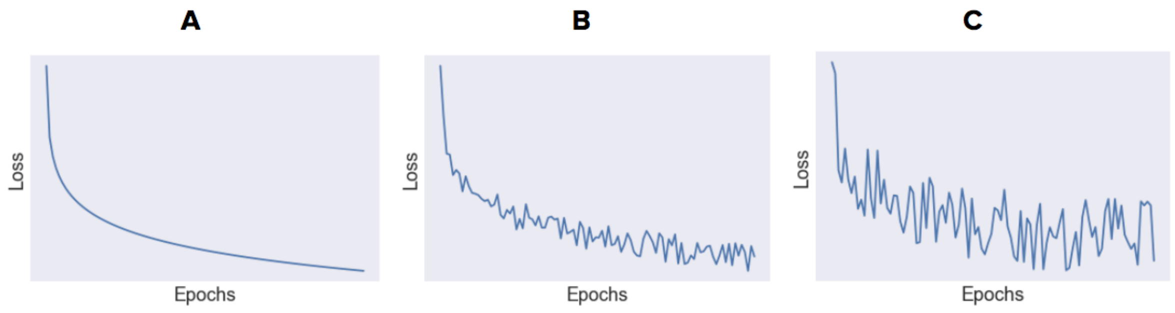
(b) **(1 point)** You are deciding whether you should optimize your network parameters using mini-batch gradient descent (MBGD) or stochastic gradient descent (SGD) (i.e. batch size of 1). Give one reason to choose MBGD over SGD. (1-2 sentences)

Solution: Compared to stochastic gradient descent, with minibatch gradient descent, we can make use of the efficiency benefits of vectorization to quickly process more training examples. In addition, the updates are less noisy.

(c) **(1 point)** Give one reason to use MBGD over BGD. (1-2 sentences)

Solution: Large datasets often can't be held in GPU memory, so minibatch gradient descent circumvents this issue. Also, additional noise in updates can often help to escape local minima and saddle points. Also, if your dataset size is very big, it may take a long time to perform even a single update for BGD.

(d) **(1 point)** Label the training loss curves A, B, and C with whether they were likely generated with SGD, MBGD, or BGD *(select one for each)*.



Solution: Batch (A), minibatch (B), stochastic (C).

(e) **(2 points)** You decide to tune your model's learning rate.

(i) What is one typical sign of a learning rate being too large? (1 sentence)

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 large.

(ii) What is one typical sign of a learning rate being too small? (1 sentence)

Solution: Cost function may take a very long time to converge. 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. You could also try higher learning rates to see if the performance improves.

(f) **(2 points)** You now decide to use gradient descent with momentum.

(i) For gradient descent with momentum, write down the update rule for a particular trainable weight W. Use learning rate α and momentum hyperparameter β, and let J denote the cost function. (2 equations)

Solution: Momentum takes exponentially weighted averages of the weight gradients:

$$V = \beta V + (1-\beta)\frac{\partial J}{\partial W}$$
$$W = W - \alpha V$$

(ii) Explain how momentum speeds up learning compared to standard gradient descent. (1 sentence)

Solution: Momentum speeds up learning by cancelling out oscillations in the gradients and ensures that the update is performed in the direction of maximum change. It does this by using exponentially weighted averages of the gradients.

(g) **(3 points)** The code below is meant to implement a single step of the training loop using the Adam optimizer, but some parts are missing. Finish the implementation of each line marked TODO. Recall the parameter update equations for Adam optimization:

$$V = \beta_1 V + (1-\beta_1)\frac{\partial J}{\partial W}$$
$$S = \beta_2 S + (1-\beta_2)\left(\frac{\partial J}{\partial W}\right)^2$$
$$V_{corr} = \frac{V}{1-\beta_1^t}$$
$$S_{corr} = \frac{S}{1-\beta_2^t}$$
$$W = W - \frac{\alpha}{\sqrt{S_{corr}+\epsilon}}V_{corr}$$

```
def optim_adam(weights_dict, gradients_dict, cache_dict, step):
    """
    v is VdW, s is SdW, v_corr is VcorrdW, s_corr is ScorrdW.
    """
    lr, beta1, beta2, eps = 1e-3, 0.9, 0.999, 1e-8
    for weight_name in weights_dict:
        w = weights_dict[weight_name]
        grad = gradients_dict[weight_name]
        v = cache_dict["v" + weight_name]
        s = cache_dict["s" + weight_name]

        # TODO: Exp weighted avg of grad
        v =

        # TODO: Exp weighted avg of grad^2
        s =

        # TODO: Bias correction. divide by (1 - beta1^step))
        v_corr =

        # TODO: Bias correction. divide by (1 - beta2^step))
        s_corr =

        # TODO: Update rule for Adam
        w =

        cache_dict["v" + weight_name] = v
        cache_dict["s" + weight_name] = s
        weights_dict[weight_name] = w
```

Solution:

```
v = beta1 * v + (1 - beta1) * grad
s = beta2 * s + (1 - beta2) * (grad ** 2)
v_corr = v / (1 - (beta1 ** step))
s_corr = s / (1 - (beta2 ** step))
w = w - lr * v_corr / np.sqrt(s_corr + eps)
```

Question 4 (Batch Normalization Questions, 9 points)

This question focuses on batch normalization.

(a) **(2 points)** Which of the following statements are true about batch normalization? **(Circle all that apply.)**

(i) Batch normalization makes processing a single batch faster, reducing the training time while keeping the number of updates fixed. This allows the network to spend the same amount of time performing more updates to reach the minima.

(ii) Batch normalization weakens the coupling between earlier/later layers, which allows for independent learning.

(iii) Batch normalization normalizes the output distribution to be more uniform across dimensions.

(iv) Batch normalization mitigates the effects of poor weight initialization and allows the network to initialize our weights to smaller values close to zero.

Solution: (0.5 points for each)

(i) False. There is no reason to assume that batch normalization makes processing a single batch faster. Quite to the contrary, we can expect batch normalization to make each training iteration slower because of the extra processing required during the forward pass and the additional hyperparameters that require training during back-propagation. That being said, we expect the network to converge faster than it would without batch normalization, so training of the network is expected to be completed faster overall.

(ii) True. We are normalizing the input *to each layer* with batch normalization, so it certainly helps with lessening the coupling between layers and makes the learning process more independent.

(iii) True. This is essentially the purpose of batch normalization, to normalize each layer's inputs (which are in turn the outputs of the layer before it) given the mean and variance of the values in the current mini-batch. As seen in the course material, we usually normalize to zero mean and unit variance.

(iv) False. This one is a little tricky. The first part is certainly true, as batch normalization mitigates the effects of poor weight initialization and allows us to be less concerned with how we initialize our weights, which is generally not an easy issue to resolve optimally, so batch normalization really helps us out here. But the second part of this statement doesn't hold, because we are simply limiting the effect of our weight initialization and then normalizing the input to each layer, weakening the impact of the initial weights determined but not necessarily "allowing the network to initialize our weights to smaller values close to zero".

(b) **(3 points)** On the next page, complete the implementation of batch normalization's forward propagation in numpy code. The following formulas may be helpful:

$$z^{(i)}_{\text{norm}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

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

```python
def forward_batchnorm(Z, gamma, beta, eps, cache_dict, beta_avg, mode):
    """
    Performs the forward propagation through a BatchNorm layer.

    Arguments:
    Z -- input, with shape (num_examples, num_features)
    gamma -- vector, BN layer parameter
    beta -- vector, BN layer parameter
    eps -- scalar, BN layer hyperparameter
    beta_avg -- scalar, beta value to use for moving averages
    mode -- boolean, indicating whether used at 'train' or 'test' time

    Returns:
    out -- output, with shape (num_examples, num_features)
    """

    if mode == 'train':
        # TODO: Mean of Z across first dimension
        mu =

        # TODO: Variance of Z across first dimension
        var =

        # Take moving average for cache_dict['mu']
        cache_dict['mu'] = beta_avg * cache_dict['mu'] + \
            (1 - beta_avg) * mu

        # Take moving average for cache_dict['var']
        cache_dict['var'] = beta_avg * cache_dict['var'] + \
            (1 - beta_avg) * var
    elif mode == 'test':
        # TODO: Load moving average of mu
        mu =

        # TODO: Load moving average of var
        var =

    # TODO: Apply z_norm transformation
    Z_norm =

    # TODO: Apply gamma and beta transformation to get Z tilde
    out =

    return out
```

Solution: 1 point for each pair of consecutive variables.

```
def forward_batchnorm(X, gamma, beta, eps, cache_dict, beta_avg, mode):
    out = None
    if mode == 'train':
        mu = np.mean(X, axis=0)
        var = np.var(X, axis=0)
        cache_dict['mu'] = beta_avg * cache_dict['mu'] + \
            (1 - beta_avg) * mu
        cache_dict['var'] = beta_avg * cache_dict['var'] + \
            (1 - beta_avg) * var
        X = (X - mu) / np.sqrt(var + eps)
        out = gamma * X + beta
    elif mode == 'test':
        mu = cache_dict['mu']
        var = cache_dict['var']
        X = (X - mu) / np.sqrt(var + eps)
        out = gamma * X + beta
        return out
```

(c) **(1 point)** What is the role of the ϵ hyperparameter in batch normalization? (1 sentence)

Solution: It prevents division by 0 for features with variance 0.

(d) **(1 point)** What problem does using batch normalization have with a batch size of 1? (1-2 sentences)

Solution: For this question specifically, if we just have a minibatch size of 1, the output from the BN layer will always be 0 ($x - \mu$ is 0, we multiply that by γ which will still be 0, and we add β. Since β is a 'bias' parameter it is initialized to 0), and so we won't be able to learn meaningful features.

In general, normalizing over a batch doesn't make sense when you just have a small number of examples per batch. Batch normalization at train time will get messed up because your mean/variance estimates will be super noisy (using just a few samples we are trying to estimate the true population/distribution). The same applies at test time - we normally use a moving average of the mean/variance, collected while training, at test time, but this moving average will also be noisy and not indicative of the true distribution. Either explanation is acceptable.

(e) **(2 points)** You are applying batch normalization to a fully connected (dense) layer with an input size of 10 and output size of 20. How many training parameters does this layer have, including batch normalization parameters? (2 sentences).

Solution: Batch normalization is typically applied to pre-activations in a dense layer, of which there are 20 for this layer. Gamma and beta are scalars for each input. As such we have 20 gammas and 20 betas. This is in addition to the $10 \times 20 = 200$ weights in the dense layer (no bias terms when batch norm is applied as the betas make them redundant). This gives a total of 240 parameters.

Question 5 (Deep Learning Strategy, 4 points)

You're asked to build an algorithm estimating the risk of premature birth for pregnant women using ultrasound images.

(a) **(2 point)** You have 500 examples in total, of which only 175 were examples of preterm births (positive examples, label = 1). To compensate for this class imbalance, you decide to duplicate all of the positive examples, and then split the data into train, validation and test sets. Explain what is a problem with this approach. (1-2 sentences)

Solution: Train data leaks into val/test sets since it is split after duplication. This is a serious issue that leads to overestimates of model performance.

1pt for other valid but less direct problems, such as potential overfitting of positive examples, mismatch between the test set and the true data distribution, etc. 0pts for other answers / incorrect statements such as "there is no benefit of having the same data appear more than once" (there is a benefit: it helps balance the loss on positive / negative examples. Training on the same data is a very common technique – recall that often, many epochs are used during training. GD is an iterative process and can benefit from using the same data more than once.)

(b) **(1 point)** You fix the issue. Subject matter experts tell you that the model should absolutely not miss preterm births, but false positives are okay. Your best model achieves 100% recall. Does it mean the model works well? Explain. (1 sentence)

Solution: Not necessarily. 100% recall is necessary to meet the experts' criteria, but is not sufficient for judging whether the model works well, since e.g. a trivial model can get 100% recall by always predicting positive.

(c) **(1 point)** In order to estimate human-level performance, you asks subject matter experts to perform the task at hand, and measures their F1 scores. Which of the following experiments would give the best estimate of Bayes error on this task? (Circle the correct option.)

	Experiment	F1 score
Option A	Single Student	0.20
Option B	Group of doctors	0.88
Option C	Single doctor	0.80

Solution: B. Bayes optimal error corresponds to the lowest possible error that can be achieved on a task, which we can approximate with the error of a group of doctors.

Question 6 (Adversarial Attacks and GANs, 6 points)

(a) **(2 points)** Which of the following statements are true regarding adversarial attacks? **(Circle all that apply.)**

(i) If you generate an adversarial example to fool a cat classifier A, there's a chance it will fool another cat classifier B.

(ii) The Fast Gradient Sign Method is an iterative method that can generate adversarial examples.

(iii) Using dropout is an effective defense against adversarial attacks.

(iv) You can create an adversarial attack against a neural network that has been encrypted on a device, where you can access neither its architecture nor its parameters.

Solution: (0.5 points for each)

(i) True. Classifiers that have similar behaviors will also be fooled.

(ii) False. FGSM is not iterative.

(iii) False. Dropout is disabled during test time.

(iv) True. Adversarial attack on black-box models is possible.

(b) **(1 point)** Recall the Fast Gradient Sign Method for generating adversarial examples:

$$x^* = x + \varepsilon \cdot \text{sign}(\frac{\partial J}{\partial x})$$

Given $x = \begin{bmatrix}1 & 2 & 3\end{bmatrix}^\top$, $\frac{\partial J}{\partial x} = \begin{bmatrix}0.5 & -0.5 & 1\end{bmatrix}^\top$, and $\varepsilon = 0.01$. What would the resulting adversarial example be? Show your work.

Solution: $x^* = x + \varepsilon \text{ sign}(\frac{\partial J}{\partial x}) = \begin{bmatrix}1\\2\\3\end{bmatrix} + 0.01\begin{bmatrix}1\\-1\\1\end{bmatrix} = \begin{bmatrix}1.01\\1.99\\3.01\end{bmatrix}$

(c) **(1 point)** The magnitude of ε needed to create the adversarial example increases with the dimension of x. Do you agree with this statement? Explain your reasoning (1-2 sentences).

Solution: This statement is false, and to see why we can consider the Fast Gradient Sign Method applied to a logistic regression model. For a logistic regression model where $\hat{y} = \sigma(wx + b)$, the gradient of $\hat{y}$ with respect to x is proportional to w^T, so we can take our adversarial example to be $x^* = x + \varepsilon \cdot \text{sign}(w^T)$. Note that this ensures that any element of x^* can deviate from the corresponding element of x by at most ε, so the deviation of the adversarial example from the original input is always small.

The output of the model on x^* is

$$\hat{y}^* = \sigma(w(x + \varepsilon \cdot \text{sign}(w^T)) + b) = \sigma(wx + b + \varepsilon \|w\|_1)$$

where $\|w\|_1$ denotes the sum of the absolute values of the elements of w. If x and w have dimension n and the average magnitude of the elements of w is m, then the change in the input to the sigmoid function grows as $\Theta(\varepsilon m n)$. This grows linearly with the dimension of x for fixed ε. This means that even when ε is small, we can perturb the elements of the input by at most ε and still achieve a large deviation in the output when the dimension of x is large.

(d) **(1 point)** Given the two options of (A) saturating cost and (B) non-saturating cost, which cost function would you choose to train a GAN? Explain your reasoning. (1-2 sentences)

Solution: Non-saturating cost is generally considered preferable, as the gradient towards the beginning of training is larger.

(e) **(1 point)** You are training a standard GAN, and at the end of the first epoch you take note of the values of the generator and discriminator losses. At the end of epoch 100, the values of the loss functions are approximately the same as they were at the end of the first epoch. Why are the quality of generated images at epoch 1 and epoch 100 not necessarily similar? (1-2 sentences)

Solution: You should not necessarily expect them to be the same since the losses are with respect to different quality models over time. That is, the loss of the generator at epochs 1 and 100 are with respect to a discriminator which might have significantly improved, and the same follows for the loss of the discriminator.

Question 7 (CNNs, 6 points + 2 extra credit points)

(a) **(1 point)** Give a reason why one would use a 1×1 convolution. Hint: what does performing a 1×1 convolution achieve in terms of the resulting output volume? (1 sentence)

Solution: A (1×1) convolution would be useful if you wanted to reduce the number of channels while preserving the spatial dimensions.

(b) **(1 point)** What would you set the padding of a 2D CONV layer to be (as a function of the filter width f) to ensure that the output has the same dimension as the input? Assume the stride is 1. (1 formula)

Solution: $\frac{f-1}{2}$

(c) **(2 points)** You have an input volume of $32 \times 32 \times 3$. What are the dimensions of the resulting volume after convolving a 5×5 kernel with zero padding, stride of 1, and 2 filters? (1 formula)

Solution: $(28 \times 28 \times 2)$

(d) **(2 point)** How many weights and biases would the layer defined in (c) have? (1 formula)

Solution: Number of parameters $= (F * F * D) * K + K = 5 * 5 * 3 * 2 + 2 = 152$

(e) **(Extra credit: 1 point)** You want to process time-series data with a 1D CONV that has the same configuration as the layer presented in (c) but with a kernel of size 5. The input volume of shape $T \times 3$ models three fluctuating values over time. How many weights and biases does this layer have? Assume the same configuration (padding, stride, number of filters) as in (c) and show your work.

Solution: Number of parameters $= (F * D) * K + K = (5 * 3) * 2 + 2 = 32$

(f) **(Extra credit: 1 point)** You want to process a video with a 3D CONV that has the same configuration as the layer presented in (c) but with kernel of shape $5 \times 5 \times 5$. The input video can be seen as a sequence of images indexed by time, i.e. a volume of shape $W \times H \times T \times 3$. How many weights and biases does this layer have? Assume the same configuration (padding, stride, number of filters) ass in (c) and show your work.

Solution: Number of parameters $= (F*F*F*D)*K+K = 5*5*5*3*2+2 = 752$

Extra Page

END OF PAPER