CS230: Deep Learning

Fall Quarter 2019

Stanford University

Midterm Examination

180 minutes

	Problem	Full Points	Your Score
1	Multiple Choice	14	
2	Short Answers	38	
3	Convolutional Architectures	12	
4	Numpy Coding	10	
5	Backpropagation	32	
6	Fun with Activation Functions	11	
7	Softmax (Bonus)	7	
Total		124	

The exam contains 32 pages including this cover page.

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

Name: _____

SUNETID: _____@stanford.edu

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

Signature: _____

.

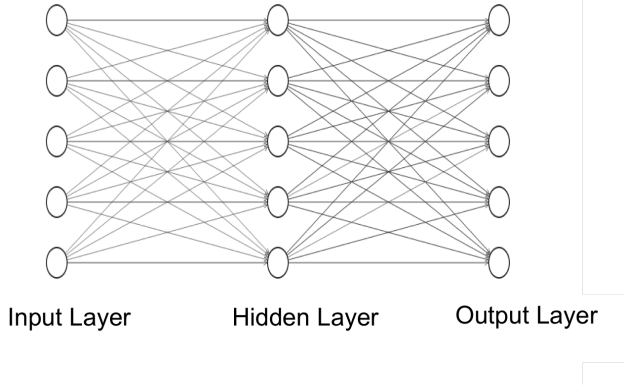
Question 1 (Multiple Choice Questions, 14 points)

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

(a) **(2 points)** A 2-layer neural network with 5 neurons in each layer has a total of 60 parameters (i.e. weights and biases)

(i) True

(ii) False



Solution: True

(b) **(2 points)** Given an input volume of shape (10, 10, 3), you consider using one of the two following layers:

- Fully-connected layer with 2 neurons, with biases
- Convolutional layer with three 2x2 filters (with biases) with 0 padding and a stride of 2

If you use the fully-connected layer, the input volume is "flattened" into a column vector before being fed into the layer. What is the **difference** in the number of trainable parameters between these two layers?

(i) The fully-connected layer has 566 fewer parameters

(ii) The convolutional layer has 566 fewer parameters

(iii) The convolutional layer has 564 fewer parameters

(iv) The convolutional layer has 563 fewer parameters

(v) None of the above

Solution: iv

(c) **(2 points)** You decide to use the convolutional layer. What will be the size of the output of the convolutional layer described above?

(i) (5, 5, 4)

(ii) (5, 5, 3)

(iii) (4, 4, 4)

(iv) (4, 4, 3)

(v) None of the above

Solution: ii

(d) **(2 points)** Assume the vectors v and u are defined as numpy arrays. The vectors are multiplied together using the * operator: v*u. Which of the following shapes are valid, meaning the operation will succeed without errors? (Circle all that apply)

(i) v = (5, 1), u = (5, 1)

(ii) v = (1, 5), u = (50, 5)

(iii) v = (1, 1), u = (50, 5)

(iv) v = (1, 5), u = (1, 50)

(v) None of the above

Solution: i, ii, iii

(e) **(2 points)** Which of the following is true about the vanishing gradient problem? (Circle all that apply)

(i) Tanh is usually preferred over sigmoid because it doesn't suffer from vanishing gradients

(ii) Vanishing gradient causes deeper layers to learn more slowly than earlier layers

(iii) Leaky ReLU is less likely to suffer from vanishing gradients than sigmoid

(iv) Xavier initialization can help prevent the vanishing gradient problem

(v) None of the above

Solution: iii, iv

(f) **(2 points)** The backpropagated gradient through a tanh non-linearity is always smaller or equal in magnitude than the upstream gradient. (Recall: if z = tanh(x) then $\frac{\partial z}{\partial x} = 1 - z^2$)

(i) True

(ii) False

Solution: True

(g) **(2 points)** Consider a trained logistic regression. Its weight vector is W and its test accuracy on a given data set is A. Assuming there is no bias, dividing W by 2 won't change the test accuracy.

(i) True

(ii) False

Solution: True

Question 2 (Short Answers, 38 points)

The questions in this section can be answered in 2-4 sentences. Please be concise in your responses.

(a) **(2 points)** The gradient estimated during a step of mini-batch gradient descent has on average a lower bias when the data is i.i.d. (independent and identically distributed). True or False? Explain why.

Solution: True. The examples in a batch should be i.i.d. because mini-batch gradient descent uses an empirical estimate of the gradient from a small batch. If the examples are correlated, then the gradient estimates will become biased and the model will fail to learn.

(b) **(2 points)** You have two data sets of similar size for a binary classification task. However, one contains almost entirely positive examples, and the other contains only negative examples. You would like to use both sets to train your model. Describe a scenario in which combining these two data sets could lead to a failure of the model to learn.

Solution: Imagine training on mini-batches constructed from dataset 1 (mostly positive examples, then training on mini-batches from dataset 2 (only negative examples). The model will likely forget what it learned from the positive examples and will learn to always predict negative examples.

(c) **(2 points)** Why do the layers in a deep architecture need to be non-linear?

Solution: Without nonlinear activation functions, each layer simply performs a linear mapping of the input to the output of the layer. Because linear functions are closed under composition, this is equivalent to having a single (linear) layer. Thus,

no matter how many such layers exist, the network can only learn linear functions.

(d) **(3 points)** Cite 3 layers commonly used in a convolutional neural network.

Solution: CONV, POOL, FC, DROPOUT, etc.

(e) **(4 points)** Alice recommends the use of convolutional neural networks instead of fully-connected networks for image recognition tasks since convolutions can capture the spatial relationship between nearby image pixels.

Bob points out that fully-connected layers can capture spatial information since each neuron is connected to all of the neurons in the previous layer.

Both are correct, but describe two reasons we should prefer Alice's approach to Bob's.

Solution: (i) Computational tractability. (ii) Explicit hierarchical representation of features. (iii) Reduces overfitting. (iv) Translation invariant.

(f) **(2 points)** You're solving a binary classification task. The final two layers in your network are a ReLU activation followed by a sigmoid activation. What will happen?

Solution: Using ReLU then sigmoid will cause all predictions to be positive.

(g) **(2 points)** You are searching the best learning rate for your model. You decide to test the following values between 0.01 and 1:

- learning rate = 0.01
- learning rate = 0.16
- learning rate = 0.21
- learning rate = 0.84
- learning rate = 0.94

Is that a good method? Explain why.

Solution: No. A better method would be to use random search on a log scale. Here, we search four values between 0.1 and 1, and search only one between 0.01 and 0.1.

(h) You are randomly (with a uniform distribution) searching for the best β_1 parameter for the Adam optimizer in the range of values $[0.9; 0.999]$.

(i) **(2 points)** What is the probability of choosing β_1 such that $\beta_1 < 0.94$?

(ii) **(2 points)** What is the probability of choosing β_1 such that $\beta_1 > 0.99$?

(iii) **(2 points)** Propose a better search method to find the best β_1 hyperparameter value.

Solution:
i) $\frac{0.04}{0.099}$

ii) $\frac{0.999-0.99}{0.099}$

iii) A better method would be to use random search on a log scale need to be checked

(i) You're solving a binary classification task.

(i) **(2 points)** You first try a logistic regression. You initialize all weights to 0.5. Is this a good idea? Briefly explain why or why not.

Solution: Yes. For logistic regression with a convex cost function you'll have just a single optimal point and it does not matter where you start, the starting point just changes the number of epochs to reach to that optimal point.

(ii) **(2 points)** Then, you try a 4-layer neural network. You initialize all weights to 0.5. Is this a good idea? Briefly explain why or why not.

Solution: No, initializing all weights to zeros does not break the symmetry. All hidden units will have identical influence on the cost, which will lead to identical gradients. Thus, both neurons will evolve symmetrically throughout training, effectively preventing different neurons from learning different things.

(j) Variations of Gradient Descent

(i) **(2 points)** Describe one advantage of using mini-batch gradient descent instead of full-batch gradient descent.

Solution: less computationally expensive, faster convergence

(ii) **(2 points)** Describe one advantage of using mini-batch gradient descent instead of stochastic gradient descent with batch size 1.

Solution: faster convergence, less divergence, can leverage efficient vectorized libraries

(iii) **(2 points)** Describe one advantage of using Adam optimizer instead of vanilla gradient descent.

Solution: per-parameter updates lead to faster convergence, momentum helps avoid getting stuck in saddle point

(k) Consider a model trying to learn an encoding of some input $x \in \mathbb{R}$. The goal is to encode the input x using $z = w_1 x \in \mathbb{R}$, then accurately reconstruct the original x from the encoded representation using $\hat{x} = w_2 z \in \mathbb{R}$. Here, $(w_1, w_2) \in \mathbb{R} \times \mathbb{R}$. The model is trained with the squared reconstruction error:

$$L(W) = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)} - w_2 w_1 x^{(i)})^2$$

(i) **(2 points)** What is the set of solutions for w_1 and w_2 which makes loss zero?

Solution: $w_2 w_1 = 1$

(ii) **(3 points)** Does the loss have a saddle point? Where?

Solution: Yes, $w_1 = w_2 = 0$

Question 3 Convolutional Architectures (12 points)

Consider the convolutional neural network defined by the layers in the left column below. Fill in the shape of the output volume and the number of parameters at each layer. You can write the shapes in the numpy format (e.g. (128,128,3)).

Notation:

- CONV5-N denotes a convolutional layer with N filters with height and width equal to 5. Padding is 2, and stride is 1.
- POOL2 denotes a 2x2 max-pooling layer with stride of 2 and 0 padding.
- FC-N denotes a fully-connected layer with N neurons

Layer	Activation Volume Dimensions	Number of parameters
Input	$32\times32\times1$	0
CONV5-10		
POOL2		
CONV5-10		
POOL2		
FC10		

Solution:

Layer	Output Volume Shape	Number of parameters
Input	$32\times32\times1$	0
CONV5-10	$32\times32\times10$	$10\times(5\times5\times1+1)$
POOL2	$16\times16\times10$	0
CONV5-10	$16\times16\times10$	$10\times(5\times5\times10+1)$
POOL2	$8\times8\times10$	0
FC10	10×1	$10\times(8\times8\times10+1)$

Question 4 (Numpy Coding, 10 points)

In this question, you will mine hard examples from a large training set. Examples for which the model's predictions are very different from the ground truth are called "hard examples." Your goal is to sample hard examples after each training epoch, so that they could be passed to your model for faster training and better performance.
Below, we use 0 and 1 to label the negative and positive classes, respectively. We use `y_hat` to denote predictions (such that `y_hat` > 0.5 indicates predictions for the positive class) and `y` to denote true labels.
Fill in the blanks in the code below. Your code should compile, so please use correct syntax.

```
import numpy as np

def mine_hard_examples(X, y, y_hat, min_sample_size):
    """
    y_hat -- numpy array of shape (m,); model predictions
    y -- numpy array of shape (m,); ground-truth labels
    """
    ### START CODE HERE ###
    # 1) Compute a vector of booleans indicating whether
    # or not the examples are misclassified
    misclassified =

    # 2) Compute the absolute difference between y and y_hat
    absolute_difference =

    # 3) Create a boolean vector indicating whether or not
    # the examples are hard given a threshold of 0.7
    # (i.e., misclassified with absolute_difference > 0.7)
    mask =

    # 4) Compute the sample size selected by the mask array
    sample_size =

    assert sample_size < len(mask)
    if sample_size < min_sample_size:
        # 5) Randomly flip 0's in mask to ensure we return
        # min_sample_size examples. We'll use flip_probs
        # to select supplemental indices to set in mask;
        # recall that flip_probs must sum to 1
        flip_probs =

        ### END CODE HERE ###
        idx = np.random.choice(
            len(mask),
```

```
            min_sample_size - sample_size,
            replace=False,
            p=flip_probs)
        mask[idx] = ~mask[idx]
    return X[mask], y[mask]
```

Solution:

```
import numpy as np

def mine_hard_examples(X, y, y_hat, min_sample_size):
    # np.logical_xor(y_hat > 0.5, y) is also ok;
    # The ^ operator is acceptable as well below
    # as long as mask is converted to bool later;
    misclassified = (y_hat > 0.5) != y
    absolute_difference = np.abs(y - y_hat)
    # np.logical_and is also acceptable below
    mask = misclassified & (absolute_difference > 0.7)
    # other ways to count are also acceptable
    sample_size = np.sum(mask)
    assert sample_size < len(mask)
    if sample_size < min_sample_size:
        # using np.logical_not(mask) below is also acceptable
        # any equivalent length of X, y, etc. is also OK
        flip_probs = ~mask / (len(mask) - sample_size)
        idx = np.random.choice(
            len(mask),
            min_sample_size - sample_size,
            replace=False,
            p=flip_probs)
        mask[idx] = ~mask[idx]
    return X[mask], y[mask]

np.random.seed(0)
cases = [
    (0, 0.3, False),  # TN
    (1, 0.7, True),   # TP; randomly selected/flipped
    (0, 0.7, True),   # easy FP; randomly selected/flipped
    (1, 0.3, False),  # easy FN
    (0, 0.7 + np.finfo(float).eps, True),  # hard FP
    (1, 0.3 - np.finfo(float).eps, True),  # hard FN
]
X = np.arange(len(cases))
y = np.array([c[0] for c in cases])
```

```
y_hat = np.array([c[1] for c in cases])
actual, _ = mine_hard_examples(X, y, y_hat, 4)
expected = X[[c[2] for c in cases]].tolist()
assert actual.tolist() == expected, f'{actual} != {expected}'
```

Question 5 (Backpropagation , 32 points)

You're trying to classify RGB images in giraffe present (1) and giraffe absent (0) using a deep neural network. Unfortunately, your data set is imbalanced. The class counts are:

2000 images with a giraffe

200 examples with no giraffe

(a) **(2 points)** Name two data augmentation techniques you could use to help address the class imbalance problem.

Solution: Gaussian blur, translation, reflection

Instead of data augmentation, you want to experiment with other techniques. Here's the architecture of your network:

$$z_1 = W_1 x^{(i)} + b_1$$
$$a_1 = \text{ReLU}(z_1)$$
$$z_2 = W_2 a_1 + b_2$$
$$\hat{y}^{(i)} = \sigma(z_2)$$
$$L^{(i)} = \alpha * y^{(i)} * \log(\hat{y}^{(i)}) + \beta * (1 - y^{(i)}) * \log(1 - \hat{y}^{(i)})$$
$$J = -\frac{1}{m} \sum_{i=1}^{m} L^{(i)}$$

The dimensions are as follows: $\hat{y}^{(i)} \in \mathbb{R}$, $y^{(i)} \in \mathbb{R}$, $x^{(i)} \in \mathbb{R}^{D_x \times 1}$, $W_1 \in \mathbb{R}^{D_{a_1} \times D_x}$, $W_2 \in \mathbb{R}^{1 \times D_{a_1}}$. Note m is the size of the dataset and that the RGB images are flattened into vectors of length D_x before being fed into the network.

(b) **(2 points)** What are the dimensions of b_1 and b_2?

Solution:
$b_1 \in \mathbb{R}^{D_{a_1} \times 1}$
$b_2 \in \mathbb{R}^{1 \times 1}$

(c) Let's focus on the hyperparameters α and β of the loss function.

(i) **(2 points)** Why are α and β useful?

Solution: Weighting how much each class contributes to the loss function can help gradient descent because the network will take larger steps when learning from instances of the underrepresented class

(ii) **(4 points)** What is a reasonable pair of values for (α, β)? Provide specific values for these weightings.

Solution: $\alpha = .1$, $\beta = 1$. Roughly, the ratio should be somewhere near $\beta = 10 * \alpha$ but not ridiculously large or small.

(d) Backpropagation,

Hint: For these derivations, it will save you time to refer to earlier answers using symbols. For example, gradients computed in an earlier part can be denoted δ_1, δ_2, etc.

(i) **(3 points)** What is $\frac{\partial J}{\partial \hat{y}}$?

Solution:

$$-\frac{1}{m}\sum_i \delta_1^{(i)}$$

where

$$\delta_1^{(i)} = \alpha * \frac{y^{(i)}}{\hat{y}^{(i)}} - \beta * \frac{(1-y^{(i)})}{1-\hat{y}^{(i)}}$$

(ii) **(2 points)** What is $\frac{\partial \hat{y}^{(i)}}{\partial z_2}$?

Solution:

$$\delta_2 = \sigma(z_2)(1-\sigma(z_2))$$

(iii) **(2 points)** What is $\frac{\partial z_2}{\partial a_1}$?

Solution:

$$\delta_3 = W_2$$

(iv) **(3 points)** What is $\frac{\partial a_1}{\partial z_1}$?

Solution:

$$\delta_4 = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

(v) **(2 points)** What is $\frac{\partial z_1}{\partial W_1}$?

Solution:

$$x^{(i)^T}$$

(vi) **(3 points)**What is $\frac{\partial J}{\partial W_1}$?
Hint: reuse work from previous parts.

Solution: There's multiple ways to express this. An example (please verify):

$$\delta_5 = -\frac{1}{m} \sum_i \delta_1^{(i)} * \delta_2 * (\delta_3 \circ \delta_4) * \delta_5$$

Where $\delta_1^{(i)}, \delta_2$ are scalars,
$\delta_3, \delta_4 \in \mathbb{R}^{D_{a_1} \times 1}$,
$\delta_5 \in \mathbb{R}^{1 \times D_x}$

(vii) **(2 points)** You decide to to add L2 regularization to this model. Write your new cost function. Assume that the value of any regularization constant(s) is 1.

Solution:

$$J = -\sum_i \Big(\alpha * (1 - y^{(i)}) * \log(1 - \hat{y}^{(i)}) + \beta * y^{(i)} * \log(\hat{y}^{(i)})\Big) + \|W_2\|_2^2 + \|W_1\|_2^2$$

(viii) **(3 points)** Using this new cost function, write down the update rule for W_1. Hint: You should reuse some of your work from previous parts. Assume you are using gradient descent without optimizers. Use η as your learning rate.

Solution:

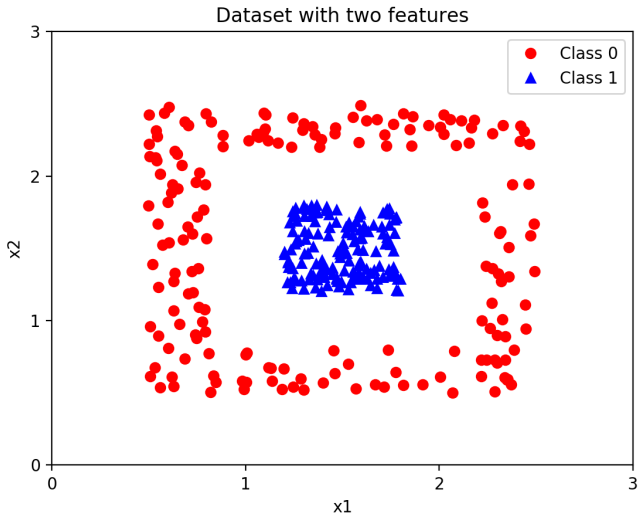
$$W_1' = W_1 - \eta * \Big(\frac{\partial J}{\partial W_1} + 2 * W_1\Big)$$

(ix) **(2 points)** Suppose you used L1 regularization instead. How would you expect the weights learned using L1 regularization to differ from those learned using L2 regularization?

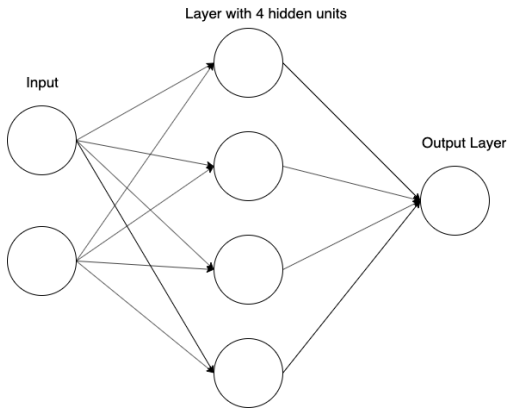
Solution: L1 weights will probably be more sparse.

Question 6 (Fun with Activation Functions, 11 points)

(a) You have a dataset where each example contains two features, x1 and x2, and a binary label. Here's a plot of the dataset:



You want to develop a model to perform binary classification. Suppose you're using a small neural network with the architecture shown below.



Below is a precise definition of the network. Note h is the activation function, $w_{i,j}$ denotes the jth weight in the ith hidden unit in the network, and $w_{output,j}$ denotes the jth weight in the output layer. The biases follow similar rules.

$$a_i = h(w_{i,1} * x_1 + w_{i,2} * x_2 + b_i)$$

$$\text{output} = f\Big(\sum_{j=1}^{4}(a_j * w_{\text{output},j}) + b_{\text{output}}\Big)$$

Note that h is the activation function for the hidden units and f is the activation function for the output layer. For all of these questions, f is defined as:

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x >= 0 \end{cases}$$

(i) **(3 points)** If you used the function $h(x) = c * x$ for some $c \in \mathbb{R}$, is it possible for this model to achieve perfect accuracy on this dataset? If so, provide a set of weights that achieves perfect accuracy. If not, briefly explain why.

Solution: No; the decision boundary for this network would become linear due to composition of matrix multiplication (something of the form prediction $= \mathbf{1}\{wx + b > 0\}$.

(ii) **(4 points)** Now assume h is a modified version of the sign function:

$$h(x) = \begin{cases} 0 & x < 0 \\ 1 & x >= 0 \end{cases}$$

Is it possible for this model to achieve perfect accuracy? If so, provide a set of weights that achieves perfect accuracy. If not, briefly explain why.

Solution: The key is to have each hidden node evaluate one of the sides of the separating square and the output layer checks that all conditions are true (or false, depending on how the hidden weights are set). For example:
$w_1 = (0, -1)$, $b_1 = 2$
$w_2 = (-1, 0)$, $b_2 = 2$
$w_3 = (0, 1)$, $b_3 = -1$
$w_4 = (1, 0)$, $b_4 = -1$
$w_{\text{output}} = (1, 1, 1, 1)$, $b_1 = -4$

(b) Recall the activation functions ReLU:

$$\text{ReLU}(x) = \max(0, x)$$

(i) **(2 points)** Consider an alternative to ReLU called Exponential Linear Unit (ELU).

$$\text{ELU}(x) = \begin{cases} x & x >= 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

What is the gradient for ELU?

Solution:

$$\begin{cases} 1 & x >= 0 \\ \alpha e^x & x < 0 \end{cases}$$

(ii) **(2 points)** Name one advantage of using ELU over ReLU.

Solution: Non-zero gradient everywhere avoids dying ReLU problem

Question 7 (Bonus, Softmax, 7 points)

This is a bonus question. Please spend your time wisely.

Softmax takes in an n-dimensional vector x and outputs another n-dimensional vector y:

$$y_i = \frac{e^{x_i}}{\sum_k e^{x_k}}$$

In this question we're going to compute the gradient of y with respect to x. Let $\delta_{ij} = \frac{\partial y_i}{\partial x_j}$.

(i) **(3 points)** Derive an expression for δ_{ii}.
Hint: Recall the Quotient Rule of Calculus: Let $h(x) = \frac{f(x)}{g(x)}$. Then

$$\frac{\partial h}{\partial x} = \frac{\frac{\partial f(x)}{\partial x} * g(x) - \frac{\partial g(x)}{\partial x} * f(x)}{\big(g(x)\big)^2}$$

Solution:

$$\delta_{ii} = y_i * (1 - y_i)$$

(ii) **(2 points)** Now derive an expression for δ_{ij}, where $i \neq j$.

Solution:

$$\delta_{ij} = -y_i * y_j$$

(iii) **(2 points)**Write a general expression for δ_{ij}. Hint: Feel free to use curly brace notation to distinguish between the two cases where $i = j$ and $i \neq j$. For a concrete example, see how ELU was defined in the previous question.

Solution:

$$\delta_{ij} = \begin{cases} y_i(1 - y_j) & i = j \\ -y_i * y_j & i \neq j \end{cases}$$

Extra Page 1/6

Extra Page 2/6

Extra Page 3/6

Extra Page 4/6

Extra Page 5/6

Extra Page 6/6

END OF PAPER