EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Midterm Exam

Exam location: Wheeler

PRINT your student ID: _____

PRINT AND SIGN your name: _____, _____ _____
(last) (first) (signature)

PRINT your discussion section: _____
Row Number (front row is 1): _____ Seat Number (left most is 1): _____

Name and SID of the person to your left: _____

Name and SID of the person to your right: _____

Name and SID of the person in front of you: _____

Name and SID of the person behind you: _____

Section 0: Pre-exam questions (5 points)

1. Honor Code: Please copy the following statement in the space provided below and sign your name. (1 point or $-\infty$)

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I will follow the rules and do this exam on my own.

2. What's your favorite thing about this semester? (4 points)

Do not turn this page until the proctor tells you to do so. You can work on Section 0 above before time starts.

PRINT your name and student ID: _____

3. Normalization Layers (10 points)

Recall the pseudocode for a batchnorm layer (with learnable scale and shift) in a neural network:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// standardize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(a) (5pts) If our input data (1-dimensional) to batchnorm follows roughly the distribution on the left:

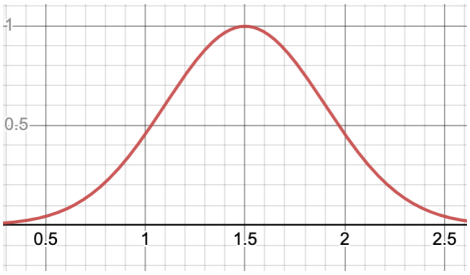


Figure 1: Gaussian with mean $\mu = 1.5$, variance $\sigma^2 = 0.16$

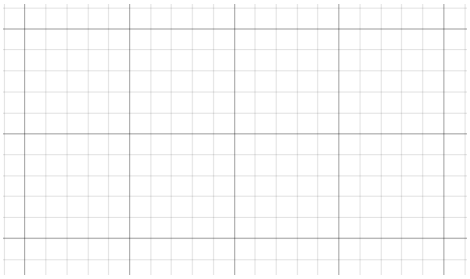


Figure 2: Blank grid for your answer

What does our data distribution look like after batch normalization with $\beta = 3$ and $\gamma = 1$ parameters? Draw your answer on the blank grid above, give a scale to the horizontal axis, and label β. You can assume that the batch-size is very large.

(Note: You do not have to give a scale to the vertical axis.)

Solution: Shifted and scaled gaussian is still a gaussian. Note that mean is at $\beta = 3$, and stdev is $\gamma = 1$.

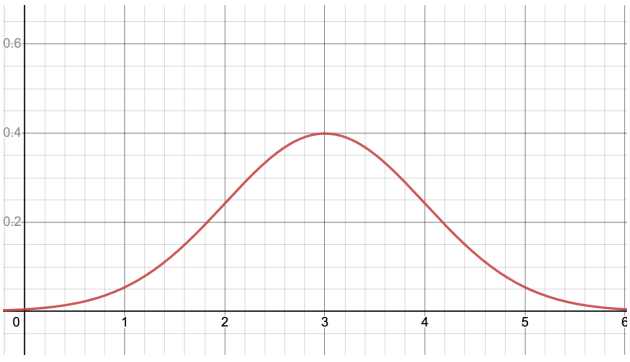


Figure 3: Solution

PRINT your name and student ID: _____

(b) (5pts) Say our input data (now 2-dimensional) to the batchnorm layer follows a Gaussian distribution. The mean and contours (level sets) of points that are 1 and 2 stdev away from the mean are shown below. **On the same graph, draw what the mean, 1-SD, and 2-SD contours would look like after batchnorm without any shifting/scaling (i.e. $\beta = 0$ and $\gamma = 1$).** You can assume that the batch-size is very large.

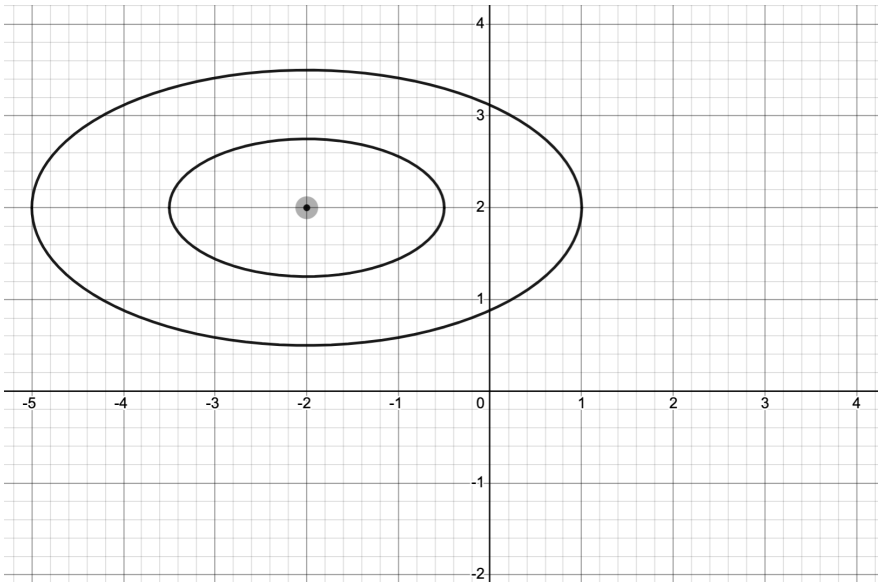


Figure 4: Draw your answer on the grid

Solution:

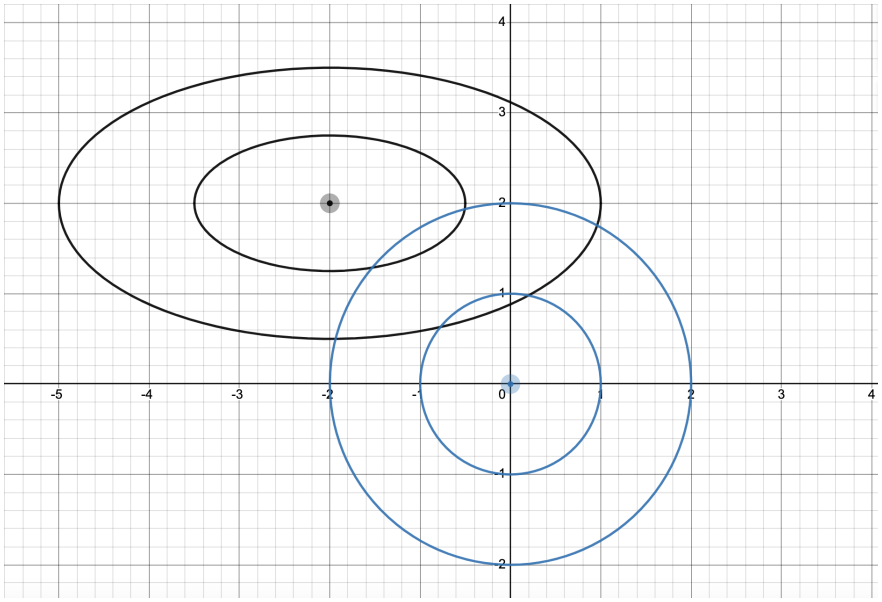


Figure 5: Solution

4. Gradients: exploding and vanishing (6 points)

For deep neural nets, vanishing gradients make it very hard to train some aspects of the network since the corresponding training updates barely change anything. Exploding gradients force the use of tiny learning rates and these tiny learning rates similarly make it hard to train.

Which of these "tricks" help with vanishing and exploding gradients?

If a trick helps with neither, don't mark anything. If it helps primarily with vanishing gradients, just mark the column for vanishing gradients. Similarly, if it helps primarily with exploding gradients, just mark that column. If it helps with both vanishing and exploding gradients, mark both columns.

Algorithmic & Model Tools	Vanishing Gradient	Exploding Gradient
He Initialization (relative to Xavier Init)	☐	☐
Skip/Residual Connections	☐	☐
Batch Normalization	☐	☐

Solution:

Algorithmic & Model Tools	Vanishing Gradient	Exploding Gradient
He Initialization (relative to Xavier Init)	✓	
Skip/Residual Connections	✓	
Batch Normalization	✓	✓

He initialization is different from Xavier initialization is because with ReLU activations in the previous layer, we are more afraid of having half of the incoming things being 0. So, He initialization increases the size of the weights to make sure that the sum that comes in is a good size, and thus helps with the vanishing gradient problem relative to Xavier initalization. Students will not be penalized for selecting "Exploding gradient", since Xavier initialization (compared to random initialization) addresses exploding gradients, too.

Skip connections help with the vanishing gradient problem by having an identity "shortcut" route that gradients can flow through which is unaffected by the linear layers and nonlinearities.

Batch normalization keeps the activation sizes controlled and thus helps with both the vanishing and exploding gradient problems.

PRINT your name and student ID: _____

5. Backprop through a simple RNN (18 points)

Consider the following 1D RNN with no nonlinearities, a 1D hidden state, and 1D inputs u_t at each timestep. (Note: There is only a single parameter w, no bias). This RNN expresses unrolling the following recurrence relation, with hidden state h_t at unrolling step t given by:

$$h_t = w \cdot (u_t + h_{t-1}) \tag{1}$$

The computational graph of unrolling the RNN for three timesteps is shown below:

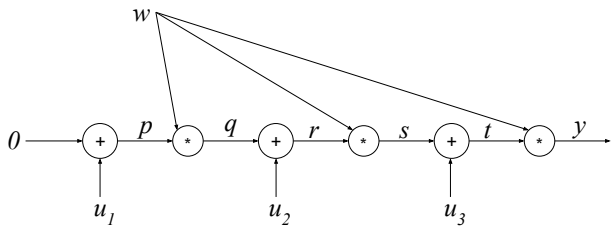


Figure 6: Illustrating the weight-sharing and intermediate results in the RNN.

where w is the learnable weight, u_1, u_2, and u_3 are sequential inputs, and p, q, r, s, and t are intermediate values.

(a) (4pts) **Fill in the blanks for the intermediate values during the forward pass, in terms of w and the u_i's:**

$$p = u_1 \qquad q = w \cdot u_1 \qquad r = u_2 + q = u_2 + w \cdot u_1$$

$$s = w \cdot r = w \cdot u_2 + w^2 \cdot u_1$$

$$t = \_\_\_\_\_$$

Solution: $t = u_3 + s = u_3 + w \cdot u_2 + w^2 \cdot u_1$

$$y = \_\_\_\_\_$$

Solution: $y = w \cdot t = w \cdot u_3 + w^2 \cdot u_2 + w^3 \cdot u_1$

(b) (4pts) **Using the expression for y from the previous subpart, compute $\frac{dy}{dw}$.**

Solution: $\frac{dy}{dw} = u_3 + 2wu_2 + 3w^2u_1$

PRINT your name and student ID: _____

(c) (2pts) **Fill in the blank for the missing partial derivative of y with respect to the nodes on the backward pass.** You may use values for p, q, r, s, t, y computed in the forward pass and downstream derivatives already computed.

$$\frac{\partial y}{\partial t} = w \qquad \frac{\partial y}{\partial s} = w \qquad \frac{\partial y}{\partial r} = \frac{\partial y}{\partial s} \cdot w \qquad \frac{\partial y}{\partial q} = \frac{\partial y}{\partial r} \cdot 1$$

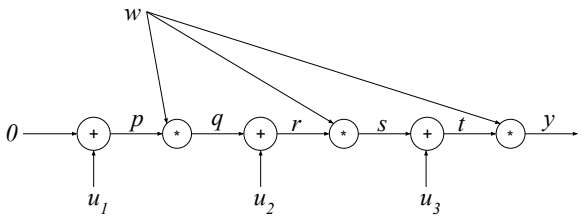
$$\frac{\partial y}{\partial p} = \_\_\_\_\_$$

Solution: $\frac{\partial y}{\partial p} = \frac{\partial y}{\partial q} \cdot w = w \cdot w \cdot w = w^3$

(d) (8pts) **Calculate the partial derivatives along each of the three outgoing edges from the learnable w in Figure 6, replicated below.** (e.g., the right-most edge has a relevant partial derivative of t in terms of how much the output y is effected by a small change in w as it influences y through this edge. You need to compute the partial derivatives for the other two edges yourself.)

You can write your answers in terms of the p, q, r, s, t and the partial derivatives of y with respect to them.

Use these three terms to find the total derivative $\frac{dy}{dw}$.



(HINT: You can use your answer to part (b) to check your work.)

Solution: Along the right edge, we have $t = u_3 + wu_2 + w^2u_1$ (This is provided for you).
Along the middle edge, we have $r \cdot \frac{\partial y}{\partial s} = r \cdot w = (u_2 + wu_1)w = wu_2 + w^2u_1$
Along the left edge, we have $p \cdot \frac{\partial y}{\partial q} = p \cdot w^2 = u_1w^2$
Adding all of these up, we have

$$\begin{aligned} \frac{dy}{dw} &= t + r \cdot \frac{\partial y}{\partial s} + p \cdot \frac{\partial y}{\partial q} && (2) \\ &= (u_3 + wu_2 + w^2u_1) + (u_2 + wu_1)w + w^2u_1 && (3) \\ &= 3w^2u_1 + 2wu_2 + u_3 && (4) \end{aligned}$$

Which is same as answer to (b) so everything checks out.

PRINT your name and student ID: _____

6. Argmax Attention (10 points)

Recall from lecture that we can think about attention as being *queryable softmax pooling*. In this problem, we ask you to consider a hypothetical argmax version of attention where it returns exactly the value corresponding to the key that is most similar to the query, where similarity is measured using the traditional inner-product.

(a) (6pts) Perform argmax attention with the following keys and values:

Keys:

$$\left\{ \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 3 \\ 4 \end{bmatrix}, \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}$$

Corresponding Values:

$$\left\{ \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \right\}$$

using the following query:

$$\mathbf{q} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

What would be the output of the attention layer for this query? Remember, to simplify calculations, use an argmax instead of softmax. For example, $\mathrm{softmax}([1,3,2])$ becomes $\mathrm{argmax}([1,3,2]) = [0,1,0]$.

Solution: Compute inner products of query with the keys:

$$\langle [1,1,2]^\top, [1,2,0]^\top \rangle = 3$$

$$\langle [1,1,2]^\top, [0,3,4]^\top \rangle = 11$$

$$\langle [1,1,2]^\top, [5,0,0]^\top \rangle = 5$$

$$\langle [1,1,2]^\top, [0,0,1]^\top \rangle = 2$$

We take argmax rather than softmax, and have $\mathrm{argmax}([3,11,5,2]) = [0,1,0,0]$

Now, take weighted sum of value vectors (in this case all are zeroed out except for the one corresponding to the highest dot-product between query and key)

$0 \cdot [2,0,1]^\top + 1 \cdot [1,4,3]^\top + 0 \cdot [0,-1,4]^\top + 0 \cdot [1,0,-1]^\top$

So final output is $\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$

(b) (4pts) Note that instead of using *softmax* we used `argmax` to generate outputs from the attention layer. **How does this design choice affect our ability to usefully train models involving attention?**

(Hint: think about how the gradients flow through the network in the backward pass. Can we learn to improve our queries or keys during the training process?)

Solution: It wouldn't work in real life, since the gradients only flow through the one element that is selected by argmax and thus most of the parameters in the transformer layers would remain the same if we did gradient-based training. The reason is that generically, the argmax is not sensitive to small changes in the keys and queries, since any such tiny perturbations will not change the winner. Consequently, the gradients with respect to the keys and queries will always be zero. This means that the keys and queries will never be improved — or more precisely, the functions used to generate keys and queries from the state will never get updated.

PRINT your name and student ID: _____

7. CNNs (16 points)

Suppose that, like in the homework, we are training a CNN to classify whether an image has a horizontal edge, a vertical edge, or no edge.

(a) (10pts) We are going to describe a convolutional neural net using the following pieces:

- CONV3-F denotes a convolutional layer with F different filters, each of size $3 \times 3 \times C$, where C is the depth (i.e. number of channels) of the activations from the previous layer. Padding is 1, and stride is 1.
- POOL2 denotes a 2×2 max-pooling layer with stride 2 (pad 0)
- FLATTEN just turns whatever shape input tensor into a one-dimensional array with the same values in it.
- FC-K denotes a fully-connected layer with K output neurons.

Note: All CONV3-F and FC-K layers have biases as well as weights. Do not forget the biases when counting parameters.

Now, we are going to use this network to do inference on a single input. **Fill in the missing entries in this table of the size of the activations at each layer, and the number of parameters at each layer. You can/should write your answer as a computation (e.g.** $128 \times 128 \times 3$**) in the style of the already filled-in entries of the table.**

Layer	Number of Parameters	Dimension of Activations
Input	0	$28 \times 28 \times 1$
CONV3-10	**Solution:** $3 \times 3 \times 1 \times 10 + 10$	$28 \times 28 \times 10$
POOL2	0	$14 \times 14 \times 10$
CONV3-10	$3 \times 3 \times 10 \times 10 + 10$	**Solution:** $14 \times 14 \times 10$
POOL2	**Solution:** 0	**Solution:** $7 \times 7 \times 10$
FLATTEN	0	490
FC-3	**Solution:** $490 \times 3 + 3$	3

PRINT your name and student ID: _____

(b) (6pts) Consider a new architecture: CONV2-3 → ReLU → CONV2-3 → ReLU → GAP (Global Average Pool) → FC-3. Each CONV2-3 layer has stride of 1 and padding of 1. Note that we use circular padding (i.e. wrap-around) for this task. Instead of using zeros, circular padding makes it as though the virtual column before the first column is the last column and the virtual row before the first row is the last row — treating the image as though it was on a torus.

Here, the GAP layer is an average pooling layer that computes the per-channel means over the entire input image.

You are told the behavior for an input image with a horizontal edge, $\mathbf{x}_1$ and an image with a vertical edge, $\mathbf{x}_2$:

$$\mathbf{x}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Suppose we knew that the GAP output features when fed $\mathbf{x}_1$ and $\mathbf{x}_2$ are

$$\mathbf{g}_1 = f(\mathbf{x}_1) = \begin{bmatrix} 0.8 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{g}_2 = f(\mathbf{x}_2) = \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix}$$

Use what you know about the invariances/equivariances of convolutional nets to compute the $\mathbf{g}_i$ corresponding to the following $\mathbf{x}_i$ images.

- $\mathbf{x}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ **Solution:** $\mathbf{g}_3 = f(\mathbf{x}_3) = \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix}$

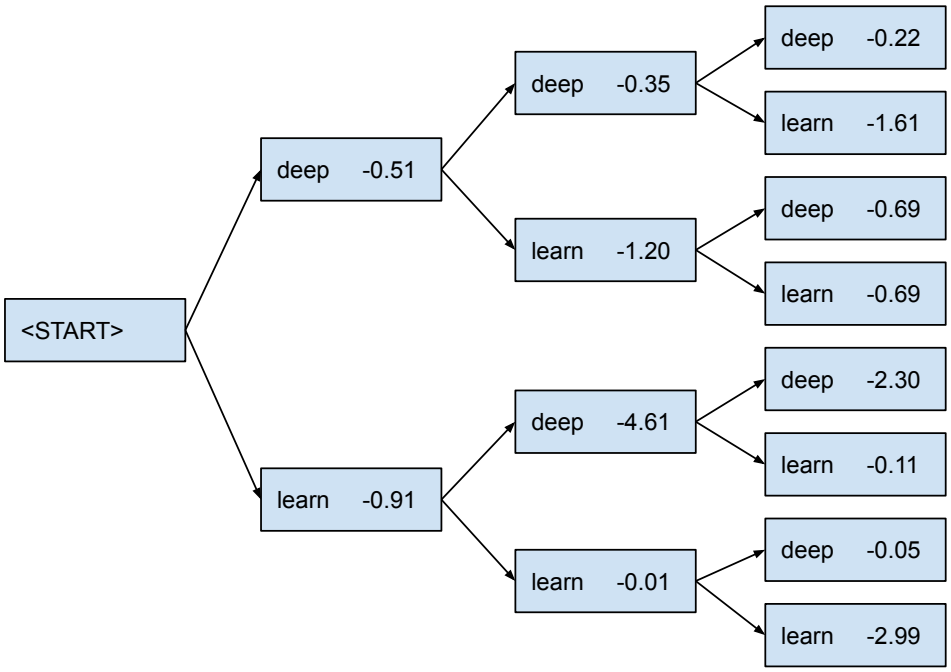
- $\mathbf{x}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ **Solution:** $\mathbf{g}_4 = f(\mathbf{x}_4) = \begin{bmatrix} 0.8 \\ 0 \\ 0 \end{bmatrix}$

Solution: In both cases, this is because the input is just a circular shift of the examples earlier and we know that this CNN is invariant to circular shifts.

PRINT your name and student ID: _____

8. Beam Search (14 points)

Consider a simplified RNN language decoder that outputs one of two words at each timestep. The log-probabilities of each word at a given timestep are shown next to the word. A tree structure (like you saw in discussion) is used to visualize what the next set of possibilities would be assuming that we committed to earlier choices.



(a) (5pts) In exhaustive search, we look at all the possible sequences and pick the most likely one. **With exhaustive search, what is the sequence that this RNN decoder would emit?**

Solution: "learn learn deep"

One can exhaustively calculate the sum of log-probabilities for each sequence and choose sequence with the largest value.
deep deep deep: -1.08
deep deep learn: -2.47
deep learn deep: -2.4
deep learn learn: -2.4
learn deep deep: -7.82
learn deep learn: -5.63
learn learn deep: -0.97
learn learn learn: -3.91
Students do not need to show all work.

(b) (4pts) In greedy search, we just pick the highest probability choice at each step and commit to it. **Using greedy search, what is the sequence that this RNN decoder would emit?**

Solution: "deep deep deep"

At the first step, we choose deep since -0.51 > -0.91
At the second step, we choose deep since -0.35 > -1.20
At the third step, we choose deep since -0.22 > -1.61

(c) (5pts) In beam search, we keep the past best k possibilities in memory and see what could come next. We then keep the best k possibilities of these in memory, and continue moving forward. **Using beam search with beam size $k = 2$, what sequence would this RNN decoder emit?**

Solution: "learn learn deep"
After the first step, our beam contains:
deep: -0.51
learn: -0.91

In the second step, we expand and get:
deep deep: -0.86
deep learn: -1.71
learn deep: -5.52
learn learn: -0.92

We only keep the top two in our beam, so after the second step our beam would contain:
deep deep: -0.86
learn learn: -0.92

In the third step, we expand only the sequences in our beam and get:
deep deep deep: -1.08
deep deep learn: -2.47
learn learn deep: -0.97
learn learn learn: -3.91

Of which "learn learn deep" is the best.

PRINT your name and student ID: _____

9. Machine Translation (10 points)

Consider the following Machine Translation problem:

- You are learning an Encoder-Decoder model to translate sentences from Spanish into English.
- This Encoder-Decoder model consists of a single-layer RNN encoder and a single-layer RNN decoder.
- The first hidden state of the decoder is initialized with the last hidden state of the encoder.
- Words are converted into learned embeddings of length H (hidden state size), before they are passed to the model.

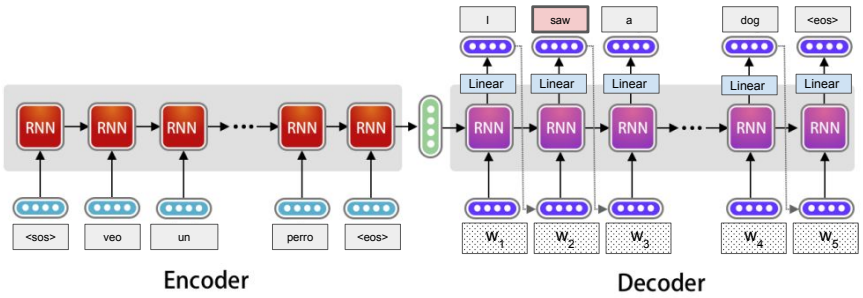


Figure 7: Translation model. The ovals with horizontal dots are learned encodings of words. The tokens $< sos >$ and $< eos >$ are "Start of Sequence" and "End of Sequence" tokens respectively. The boxes $w_1 \dots w_5$ represent the word tokens passed into the RNN decoder at each timestep (you'll fill in which tokens go here). **Solution:** Diagram modified from `http://www.adeveloperdiary.com/data-science/deep-learning/nlp/machine-translation-recurrent-neural-network-pytorch/`

(a) (4pts) Your teammate proposes stacking the encoder and decoder vertically rather than horizontally. Instead of passing the final hidden state of the encoder h_T into the decoder's first hidden state, at each timestep t, the encoder's hidden state h_t gets passed as an input to timestep t of the decoder. **State one problem with this proposed design change.**

Solution: Reasonable answers could include (a) This model does not include an easy way to handle when different-length English sentences are appropriate to translate Spanish sentences. (b) The ith English word generated can only condition on the first i Spanish words, but words in later positions may be important for translation (e.g. if the word order in the two languages is different).

PRINT your name and student ID: _____

(b) (3pts) In the example shown the correct translation is "I see a dog," but the translation that happened to be sampled from the model incorrectly states "I **saw** a dog".

What five tokens will be passed into the decoder during training for $w_1, w_2, \ldots, w_5$?

(HINT: Remember, during training we have access to correct supervision for translations. Don't forget that you also have special tokens $< sos >$ and $< eos >$ for the beginning and end of a sentence.)

Solution: <SOS>, I see, a, dog

During training, we pass in the ground-truth previous token.

(c) (3pts) Continuing the previous part, **what five tokens would be passed into the decoder at evaluation time for $w_1, w_2, \ldots, w_5$ when a translation is being generated?**.

(Here, you can assume that the decoder only emits a single possibility for each word.)

Solution: <SOS>, I saw, a, dog

When a translation is being generated, there is no ground-truth previous token, so we pass in the output generated by the model.

PRINT your name and student ID: _____

10. Graph Neural Networks (22 points)

(a) (8pts) For an undirected graph with no labels on edges, the function that we compute at each layer of a Graph Neural Network must respect certain properties so that the same function (with weight-sharing) can be used at different nodes in the graph. Let's focus on a single particular "layer" ℓ. For a given node i in the graph, let $\mathbf{s}_i^{\ell-1}$ be the self-message (i.e. the state computed at the previous layer for this node) for this node from the preceeding layer, while the preceeding layer messages from the n_i neighbors of node i are denoted by $\mathbf{m}_{i,j}^{\ell-1}$ where j ranges from 1 to n_i. We will use w with subscripts and superscripts to denote learnable scalar weights. If there's no superscript, the weights are shared across layers. Assume that all dimensions work out.

Tell which of these are valid functions for this node's computation of the next self-message $\mathbf{s}_i^{\ell}$.

For any choices that are not valid, briefly point out why.

Note: we are *not* asking you to judge whether these are useful or will have well behaved gradients. Validity means that they respect the invariances and equivariances that we need to be able to deploy as a GNN on an undirected graph.

- $\mathbf{s}_i^{\ell} = w_1 \mathbf{s}_i^{\ell-1} + w_2 \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{m}_{i,j}^{\ell-1}$

  Solution: This is valid because it is permutation invariant to the ordering of neighbors. This is the classic averaging form. Notice that a dependence on the number of neighbors is fine.

- $\mathbf{s}_i^{\ell} = \max(w_1^{\ell} \mathbf{s}_i^{\ell-1}, w_2 \mathbf{m}_{i,1}^{\ell-1}, w_2 \mathbf{m}_{i,2}^{\ell-1}, \ldots, w_2 \mathbf{m}_{i,n_i}^{\ell-1})$ where the max acts component-wise on the vectors.

  Solution: This is valid because it is permutation invariant to the ordering of neighbors. The max is another classic permutation-invariant operation.

- Assume scalar states and messages for this item. $s_i^{\ell} = s_i^{\ell-1} \prod_{j=1}^{n_i} w_j m_{i,j}^{\ell-1}$

  Solution: We intended for this one to be marked invalid because there is a different weight w_j for each neighbor. However, it turns out that the multiplication operation itself is permutation invariant and scalar multiplication commutes. (If we had made these weight matrices and not scalars, it would not have been permutation invariant.) And so, this is actually valid. However, for grading we will consider both responses to be correct.

PRINT your name and student ID: _____

[Extra page. If you want the work on this page to be graded, make sure you tell us on the problem's main page.]

PRINT your name and student ID: _____

(b) (6pts) We are given the following simple graph on which we want to train a GNN. The goal is binary node classification (i.e. classifying the nodes as belonging to type 1 or 0) and we want to hold back nodes 1 and 4 to evaluate performance at the end while using the rest for training. We decide that the surrogate loss to be used for training is the average binary cross-entropy loss.

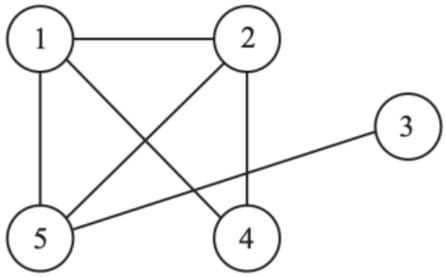


Figure 8: Simple Undirected Graph

nodes	1	2	3	4	5
y_i	0	1	1	1	0
$\hat{y}_i$	a	b	c	d	e

Table 1: y_i is the ground truth label, while $\hat{y}_i$ is the predicted probability of node i belonging to class 1 after training.

Table 1 gives you relevant information about the situation.

Please compute the training loss at the end of training.

Remember that with n training points, the formula for average binary cross-entropy loss is

$$\frac{1}{n}\sum_{x}\left(y(x)\log\frac{1}{\hat{y}(x)}+(1-y(x))\log\left(\frac{1}{1-\hat{y}(x)}\right)\right)$$

where the x in the sum ranges over the training points and $\hat{y}(x)$ is the network's predicted probability that the label for point x is 1.

Solution:

Since our testing nodes are nodes $\{1,4\}$, a mask will be used to remove their contribution to the loss. We therefore get:

$$\mathcal{L}(y,\hat{y}) = -\frac{1}{3}[y_2\times\log(\hat{y}_2)+y_3\times\log(\hat{y}_3)+y_5\times\log(\hat{y}_5)+(1-y_5)\times\log(1-\hat{y}_5)] \quad (5)$$

$$= -\frac{1}{3}[(1.\times\log(b))+(1.\times\log(c))+(0\times\log(e))+\log(1-e)] \quad (6)$$

$$= -\frac{1}{3}[\log(b)+\log(c)+\log(1-e)] \quad (7)$$

$$= -\frac{1}{3}[\log\big(bc(1-e)\big)] \quad (8)$$

Students didn't have to simplify to get full credit here.

PRINT your name and student ID: _____

(c) (8pts) Suppose we decide to use the following update rule for the internal state of the nodes at layer ℓ.

$$\mathbf{s}_i^{\ell} = \mathbf{s}_i^{\ell-1} + W_1 \frac{\sum_{j=1}^{n_i} \tanh\left(W_2 \mathbf{m}_{i,j}^{\ell-1}\right)}{n_i} \tag{9}$$

where the $\tanh$ nonlinearity acts element-wise.

For a given node i in the graph, let $\mathbf{s}_i^{\ell-1}$ be the self-message for this node from the preceeding layer, while the preceeding layer messages from the n_i neighbors of node i are denoted by $\mathbf{m}_{i,j}^{\ell-1}$ where j ranges from 1 to n_i. We will use W with subscripts and superscripts to denote learnable weights in matrix form. If there's no superscript, the weights are shared across layers.

- **Which of the following design patterns does this update rule have?** (Select all that apply)
  - ☐ Residual connection
  - ☐ Batch normalization
  - ☐ Attention

  Solution: This rule shows the residual connection pattern since the previous layer's state gets added in. This kind of residual connection enables gradients to flow back to earlier layers more easily. None of the other patterns are present.
- **If the dimension of the state $\mathbf{s}$ is d-dimensional and W_2 has k rows, what are the dimensions of the matrix W_1?**

  Solution: W_1 needs to return something that can add to the state. This means that it needs d rows. It also has to be able to act on vectors that are k dimensional since W_2 has k rows and we say that the $\tanh$ operation acts element-wise. This means that W_1 needs k columns. Putting this together, the dimensions of W_1 are $d \times k$.
- If we choose to use the state $\mathbf{s}_i^{\ell-1}$ itself as the message $\mathbf{m}^{\ell-1}$ going to all of node i's neighbors, **please write out the update rules corresponding to (9) giving $\mathbf{s}_i^{\ell}$ for the graph in Figure 8 for nodes $i = 2$ and $i = 3$ in terms of information from earlier layers.** Expand out all sums.

  Solution: We can write the mean-pooling/ average pooling as in the the neighborhood N_i is of node i:
  Therefore:
  i = 2:
  $$s_2^l = s_2^{l-1} + \frac{W_1}{3}(\tanh\left(W_2 s_1^{l-1}\right) + \tanh\left(W_2 s_4^{l-1}\right) + \tanh\left(W_2 s_5^{l-1}\right))$$
  i = 3:
  $$s_3^l = s_3^{l-1} + \frac{W_1}{1}(\tanh\left(W_2 s_5^{l-1}\right))$$
  The above equations can be found by reducing the sum in equation 11 and applying the accurate neighbors from Figure 8.

PRINT your name and student ID: _____

11. Regularization and Dropout (14 points)

You saw one perspective on the implicit regularization of dropout in HW, and here, you will see another one. Recall that linear regression optimizes the following learning objective:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - X\mathbf{w}||_2^2 \tag{10}$$

One way of using *dropout* during SGD on the d-dimensional input features $\mathbf{x}_i$ involves keeping each feature at random $\sim_{i.i.d} Bernoulli(p)$ (and zeroing it out if not kept) and then performing a traditional SGD step.

It turns out that such dropout makes our learning objective effectively become

$$\mathcal{L}(\check{\mathbf{w}}) = E_{R\sim Bernoulli(p)}\left[||\mathbf{y} - (R \odot X)\check{\mathbf{w}}||_2^2\right] \tag{11}$$

where $\odot$ is the element-wise product and the random binary matrix $R \in \{0,1\}^{n\times d}$ is such that $R_{i,j} \sim_{i.i.d} Bernoulli(p)$. We use $\check{\mathbf{w}}$ to remind you that this is learned by dropout.

Recalling how Tikhonov-regularized (generalized ridge-regression) least-squares problems involve solving:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 \tag{12}$$

for some suitable matrix Γ, it turns out we can manipulate (11) to eliminate the expectations and get:

$$\mathcal{L}(\check{\mathbf{w}}) = ||\mathbf{y} - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 \tag{13}$$

with $\check{\Gamma}$ being a diagonal matrix whose j-th diagonal entry is the norm of the j-th column of the training matrix X.

(a) (4pts) **How should we transform the $\check{\mathbf{w}}$ we learn using (13) (i.e. with dropout) to get something that looks a solution to the traditionally regularized problem (12)?**

(Hint: This is related to how we adjust weights learned using dropout training for using them at inference time. PyTorch by default does this adjustment during training itself, but here, we are doing dropout slightly differently with no adjustments during training.)

Solution: Choose $\mathbf{w} = p\check{\mathbf{w}}$. I.e., we need to multiply $\check{\mathbf{w}}$ by p.

The idea here is to absorb the p term into $\mathbf{w}$, and then choose Γ accordingly.

By choosing $\mathbf{w} = p\check{\mathbf{w}}$ (and thus $\check{\mathbf{w}} = \frac{1}{p}\mathbf{w}$), we would have

$$\begin{aligned}
\mathcal{L}(w) &= ||y - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 && (14)\\
&= ||y - X\mathbf{w}||_2^2 + p(1-p)||\check{\Gamma}\frac{\mathbf{w}}{p}||_2^2 && (15)\\
&= ||y - X\mathbf{w}||_2^2 + ||\sqrt{\frac{(1-p)}{p}}\check{\Gamma}\mathbf{w}||_2^2 && (16)\\
&= ||y - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 && (17)
\end{aligned}$$

Where we chose $\Gamma = \sqrt{\frac{1-p}{p}}\check{\Gamma}$

PRINT your name and student ID: _____

(b) (6pts) With the understanding that the Γ in (12) is an invertible matrix, **change variables in (12) to make the problem look like classical ridge regression:**

$$\mathcal{L}(\widetilde{\mathbf{w}}) = ||\mathbf{y} - \widetilde{X}\widetilde{\mathbf{w}}||_2^2 + \lambda||\widetilde{\mathbf{w}}||_2^2 \tag{18}$$

Explicitly, what is the changed data matrix $\widetilde{X}$ in terms of the original data matrix X and Γ?

Solution: To make the regularization term in (12) look like ridge regression, we'll let $\widetilde{\mathbf{w}} = \Gamma\mathbf{w}$, so $\mathbf{w} = \Gamma^{-1}\widetilde{\mathbf{w}}$. Plugging this into (12) gives us

$$||\mathbf{y} - X\Gamma^{-1}\tilde{\mathbf{w}}||_2^2 + ||\tilde{\mathbf{w}}||_2^2 \tag{19}$$

From this, we see our modified data matrix should be

$$\tilde{X} = X\Gamma^{-1}.$$

We also accept solutions which rescale the given answer by a positive constant.

(c) (4pts) Continuing the previous part, with the further understanding that Γ is a *diagonal* invertible matrix with the j-th diagonal entry proportional to the norm of the j-th column in X, **what can you say about the norms of the columns of the effective training matrix $\widetilde{X}$** and **speculate briefly on the relationship between dropout and batch-normalization.**

Solution:

Let's say each x_j, $\tilde{x}_j$ is the j-th column vector of X, $\tilde{X}$. Recall that Γ is defined to be a diagonal matrix whose j-th diagonal entry is the norm of the j-th column of X and $\tilde{X} = cX\Gamma^{-1}$, where c is a recaling positive constant of that you found in the previous part.

$$\tilde{X} = cX\Gamma^{-1} = \begin{bmatrix}\tilde{x}_1 & \tilde{x}_2 & \dots & \tilde{x}_d\end{bmatrix} = c\begin{bmatrix}x_1 & x_2 & \dots & x_d\end{bmatrix}\begin{bmatrix}\frac{1}{\|x_1\|_2} & 0 & \dots & 0 \\ 0 & \frac{1}{\|x_2\|_2} & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & \frac{1}{\|x_d\|_2}\end{bmatrix}$$

$$= \begin{bmatrix}c\frac{x_1}{\|x_1\|_2} & c\frac{x_2}{\|x_2\|_2} & \dots & c\frac{x_d}{\|x_d\|_2}\end{bmatrix}$$

Therefore, $\tilde{x}_j = c\frac{x_j}{\|x_j\|_2}$ and $\left\|\tilde{x}_j\right\|_2 = c$. I.e., the dropout makes each column vector of the matrix X constant-norm of c in effect. (Again, c is what you included in 11. (b))

Note that this is similar to batch-normalization's standardization and scaling operations, which makes the column vectors have the same variance of c.

PRINT your name and student ID: _____

12. Self-supervised Linear Autoencoders (14 points)

We consider linear models consisting of two weight matrices: an encoder $W_1 \in \mathbb{R}^{k \times m}$ and decoder $W_2 \in \mathbb{R}^{m \times k}$ (assume $1 < k < m$). The traditional autoencoder model learns a low-dimensional embedding of the n points of training data $\mathbf{X} \in \mathbb{R}^{m \times n}$ by minimizing the objective,

$$\mathcal{L}(W_1, W_2; \mathbf{X}) = \frac{1}{n} ||\mathbf{X} - W_2 W_1 \mathbf{X}||_F^2 \tag{20}$$

We will assume $\sigma_1^2 > \cdots > \sigma_k^2 > \sigma_{k+1}^2 \geq 0$ are the $k+1$ largest eigenvalues of $\frac{1}{n}\mathbf{X}\mathbf{X}^\top$. The assumption that the $\sigma_1, \ldots, \sigma_k$ are positive and distinct ensures identifiability of the principal components.

Consider an ℓ_2-regularized linear autoencoder where the objective is:

$$\mathcal{L}_\lambda(W_1, W_2; \mathbf{X}) = \tfrac{1}{n} ||\mathbf{X} - W_2 W_1 \mathbf{X}||_F^2 + \lambda \|W_1\|_F^2 + \lambda \|W_2\|_F^2. \tag{21}$$

where $\|\cdot\|_F^2$ represents the Frobenius norm squared of the matrix (i.e. sum of squares of the entries).

(a) (9pts) You want to use SGD-style training (involving the training points one at a time) and self-supervision to find W_1 and W_2 which optimize (21) by treating the problem as a neural net being trained in a supervised fashion. Briefly answer the following questions:

- **How many linear layers do you need?**
  - ☐ 0
  - ☐ 1
  - ☐ 2
  - ☐ 3

  Solution: **2**, we would use two linear layers, one for the encoder, one for the decoder.
- **What is the loss function that you will be using?**
  - ☐ `nn.L1Loss`
  - ☐ `nn.MSELoss`
  - ☐ `nn.CrossEntropyLoss`

  Solution: We should use **MSE-Loss** to train the model (reconstruction under l2-loss) since what we want is for each vector to be close to its reconstruction in a squared-error sense.
- **Which of the following would you need to optimize (21)? (Select all that are needed)**
  - ☐ Weight Decay
  - ☐ Dropout
  - ☐ Layer Norm
  - ☐ Batch Norm
  - ☐ SGD optimizer
  - ☐ Adam optimizer

  Solution: We need to use **Weight Decay** to achieve the desired regularization and of the optimizers listed, the **SGD-Optimizer** is the one that would work the best.

  Because we have potentially non-unique solutions, the Adam optimizer might end up bringing its own different bias to the problem since it does not actually take gradient steps. That's why the Adam optimizer isn't a choice that gets full credit here.

PRINT your name and student ID: _____

(b) (5pts) **Do you think that the solution to (21) when we use a small nonzero λ has an inductive bias towards finding a W_2 matrix with approximately orthonormal columns? Argue why or why not?**
(Hint: Think about the SVDs of $W_1 = U_1\Sigma_1 V_1^\top$ and $W_2 = U_2\Sigma_2 V_2^\top$. You can assume that if a $k \times m$ or $m \times k$ matrix has all k of its nonzero singular values being 1, then it must have orthonormal rows or columns. Remember that the Frobenius norm squared of a matrix is just the sum of the squares of its singular values. Further think about the minimizer of $\frac{1}{\sigma^2} + \sigma^2$. Is it unique?)

Solution: If there were no regularization terms, we know that all the optimizers have to have W_2W_1 acting like a projection matrix that projects onto the k largest singular vectors of X.

This means that the W_2W_1 to minimize the main loss has to be the identity when restricted to the subspace spanned by the k largest singular vectors of X.

Therefore we would expect W_1, W_2 be approximate psuedo-inverses of each other since they are not square, and the rank of either one is at most k.

Therefore regularizing by penalizing the Frobenius norms forces us to consider:

$$\begin{aligned}\|W_1\|_F^2 + \|W_2\|_F^2 &= \|\Sigma_1\|_F^2 + \|\Sigma_2\|_F^2 \\ &= \sum_{i=1}^{k} \left(\sigma_i^2 + \frac{1}{\sigma_i^2}\right)\end{aligned}$$

where W_1 is bringing the σ_i terms and its approximate pseudo-inverse W_2 is bringing the $\frac{1}{\sigma_i}$ for its singular values.

Minimizing $\frac{1}{\sigma^2} + \sigma^2$ by taking derivatives results in setting $0 = -\frac{2}{\sigma^3} + 2\sigma$ which has a unique non-negative real solution at $\sigma = 1$, and so this is the unique minimizer since clearly this expression goes to ∞ as $\sigma \to \infty$ or $\sigma \to 0$.

If the λ is small enough, then the optimization essentially decouples: the main loss forces W_2 and W_1 to be pseudoinverses and to have the product W_2W_1 project onto the supspace spanned by the k singular vectors of X whose singular values are largest; and the regularization term forces the individual W_2 and W_1 to have all nonzero singular values as close to 1 as possible.

Once $\sigma_i \approx 1$, the matrix has approximately orthonormal columns for W_2 and approximately orthonormal rows for W_1. You can see this by simply writing $W = U\Sigma V^\top$ and then noticing for a tall W that $W^\top W = V\Sigma^\top\Sigma V^\top \approx VIV^\top = I$ and similarly for $WW^\top$ for a wide W.

PRINT your name and student ID: _____

[Doodle page! Draw us something if you want or give us suggestions or complaints. You can also use this page to report anything suspicious that you might have noticed.

If needed, you can also use this space to work on problems. But if you want the work on this page to be graded, make sure you tell us on the problem's main page.]

