EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Final Exam

Exam location: Pimentel 1

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 below. (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 182 topic? (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. Depthwise Separable Convolutions (10 points)

Depthwise separable convolutions are a type of convolutional operation used in deep learning for image processing tasks. Unlike traditional convolutional operations, which perform both spatial and channel-wise convolutions simultaneously, depthwise separable convolutions decompose the convolution operation into two separate operations: Depthwise convolution and Pointwise convolution.

This can be viewed as a low-rank approximation to a traditional convolution. For simplicity, throughout this problem, we will ignore biases while counting learnable parameters.

(a) (5 pts) Suppose the input is a three-channel 224×224-resolution image, the kernel size of the convolutional layer is 3×3, and the number of output channels is 4.

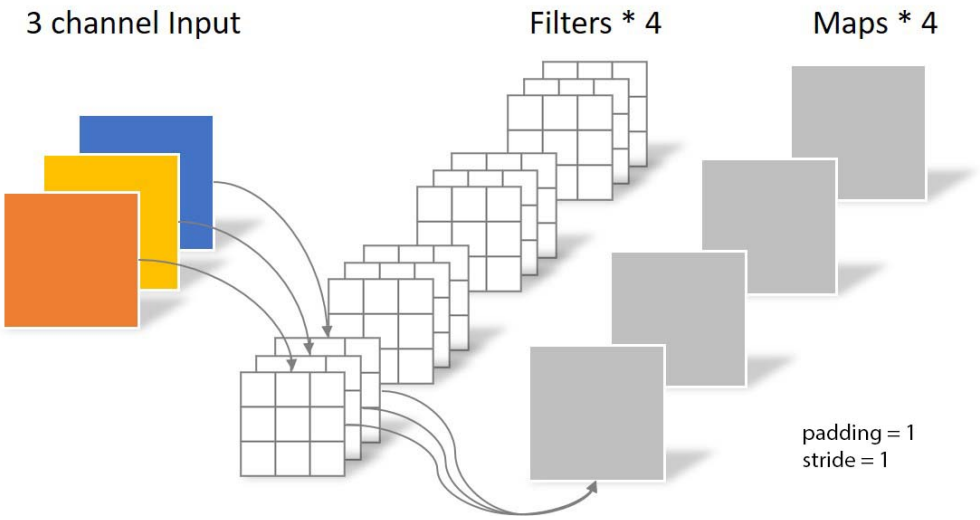


Figure 1: Traditional convolution.

What is the number of learnable parameters in the traditional convolution layer?

PRINT your name and student ID: _____

(b) (5 pts) Depthwise separable convolution consists of two parts: depthwise convolutions (Fig.2) followed by pointwise convolutions (Fig.3). Suppose the input is still a three-channel 224×224-resolution image. The input first goes through depthwise convolutions, where the number of output channels is the same as the number of input channels, and there is no "cross talk" between different channels. Then, this intermediate output goes through pointwise convolutions, which is basically a traditional convolution with the filter size being 1×1. Assume that we have 4 output channels.

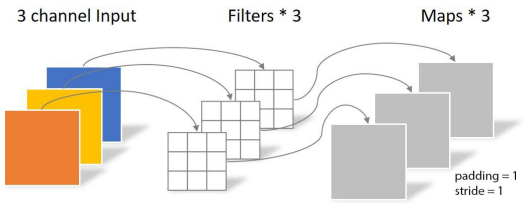


Figure 2: Depthwise convolution.

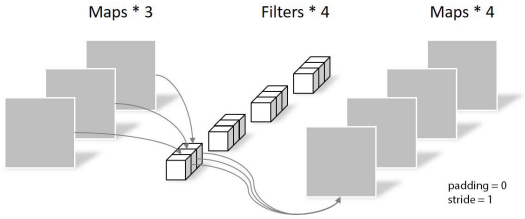


Figure 3: Pointwise convolution

What is the total number of learnable parameters of the depthwise separable convolution layer which consists of both depthwise and pointwise convolutions?

PRINT your name and student ID: _____

4. Weight Decay and Adam (11 points)

Because Adam can use different learning rates for different weights, it can have different implicit regularization behavior than traditional SGD or SGD with momentum. Consequently, it can interact differently when combined with other approaches for explicit regularization — even if those other approaches are equivalent with traditional SGD.

Two such regularization approaches are putting an L2 penalty on weights in the loss function itself (in the style of explicit ridge regularization) and alternatively, adding a weight-decay term during learning updates. For a simplified variant of Adam (without momentum), we have:

- **L2 regularization** can be expressed as: $\boldsymbol{\theta_{t+1}} \leftarrow \boldsymbol{\theta_t} - \alpha \mathbf{M}_t(\nabla f_t(\boldsymbol{\theta_t}) + \lambda_w \boldsymbol{\theta_t})$
- **Weight decay** can be expressed as: $\boldsymbol{\theta_{t+1}} \leftarrow (1 - \lambda_r)\boldsymbol{\theta_t} - \alpha \mathbf{M}_t \nabla f_t(\boldsymbol{\theta_t})$

Here $\mathbf{M_t}$ denotes the adaptive term in the Adam update step — think of this as a diagonal matrix.

(a) (6 pts) **When will the updates for L2 regularization and weight decay be identical?**
(Hint: your answer should involve things like $\alpha, \lambda_w, \lambda_r$, and $\mathbf{M_t}$ in some way.)

(b) (5 pts) **Where should you add $\lambda_r \boldsymbol{\theta_{t-1}}$ to the Adam code below to get explicit weight decay?**

Algorithm 1 Adam Optimizer (without bias correction)

1: **Given** $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$
2: **Initialize** time step $t \leftarrow 0$, parameter $\theta_{t=0} \in \mathbb{R}^n$, $m_{t=0} \leftarrow \mathbf{0}$, $v_{t=0} \leftarrow \mathbf{0}$
3: **Repeat**
4: $t \leftarrow t + 1$
5: $\nabla f_t(\theta_{t-1}) \leftarrow$ SelectBatch(θ_{t-1})
6: $g_t \leftarrow \nabla f_t(\theta_{t-1}) +$ ____**(A)**____
7: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)(g_t +$ ____**(B)**____$)$
8: $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 +$ ____**(C)**____
9: $\theta_t \leftarrow \theta_{t-1} - (\alpha \cdot \frac{m_t}{\sqrt{v_t}} +$ ____**(D)**____$)$
10: **Until** the stopping condition is met

◯ **A** on line 6 above
◯ **B** on line 7 above
◯ **C** on line 8 above
◯ **D** on line 9 above

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: _____

5. Multiplicative Regularization beyond Dropout (15 points)

In dropout, we get a regularizing effect by multipling the activations of the previous layer by iid coin tosses to randomly zero out many of them during each SGD update. Here, we will consider a linear-regression problem but instead of randomly multiplying each input feature with a 0 or a 1 during SGD updates, we will multiply each feature of our input with an iid random sample of a normal distribution with mean μ and variance σ^2. In other words, we perform the elementwise product $R \odot X$, where R is a matrix where every iid entry $R_{ij} \sim \mathcal{N}(\mu, \sigma^2)$ and $\odot$ represents elementwise multiplication.

It turns out that the expected training loss

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{R_{ij} \sim \mathcal{N}(\mu, \sigma^2)} \left[||\mathbf{y} - (R \odot X)\mathbf{w}||_2^2 \right]$$

can be put in the form

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - \_\_\textbf{(A)}\_\_X\mathbf{w}||_2^2 + \_\_\textbf{(B)}\_\_||\Gamma\mathbf{w}||_2^2$$

where $\Gamma = (\text{diag}(X^\top X))^{1/2}$.

What are (A) and **(B)**?

Select one choice for **(A)**:

- ◯ μ
- ◯ 2μ
- ◯ $\frac{\mu}{2}$
- ◯ σ
- ◯ 2σ
- ◯ $\frac{\sigma}{2}$

Select one choice for **(B)**:

- ◯ μ^2
- ◯ $2\mu^2$
- ◯ $\frac{\mu^2}{2}$
- ◯ σ^2
- ◯ $2\sigma^2$
- ◯ $\frac{\sigma^2}{2}$

Show some work below to justify your choices. Correct answers with incorrect or no supporting work will not receive full credit.

(Hint: As the problem title suggests, this should look similar to the derivation of dropout regularization you saw in the homework.)

(Further Hint: You might find it helpful to think about the case $\mu = 1$ and $\sigma^2 = 0$ to help you pick as well as $\mu = 0$ and $\sigma^2 = \infty$.)

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: _____

6. Tensor Rematerialization (23 points)

You want to train a neural network on a new chip designed at UC Berkeley. Your model is a 10 layer network, where each layer has the same fixed input and output size of s. The chip your model will be trained on is specialized for model evaluation, and thus can run forward passes very fast. However, it is severely memory constrained and can only fit $2s$ activations in memory (in addition to the inputs and other optimizer state).

To train despite this memory limitation, your friend suggests using a training method called tensor rematerialization. She proposes using SGD with a batch size of 1, and only storing the activations of every 5th layer during an initial forward pass to evaluate the model. During backpropagation, she suggests recomputing activations on-the-fly for each layer by loading the relevant last stored activation from memory, and rerunning forward through layers up till the current layer.

Figure 4 illustrates this approach. Activations for Layer 5 and Layer 10 are stored in memory from an initial forward pass through all the layers. Consider when weights in layer 7 are to be updated during backpropagation. To get the activations for layer 7, we would load the activations of layer 5 from memory, and then run them through layer 6 and layer 7 to get the activations for layer 7. These activations can then be used (together with the gradients from upstream) to compute the gradients to update the parameters of Layer 7, as well as get ready to next deal with layer 6.

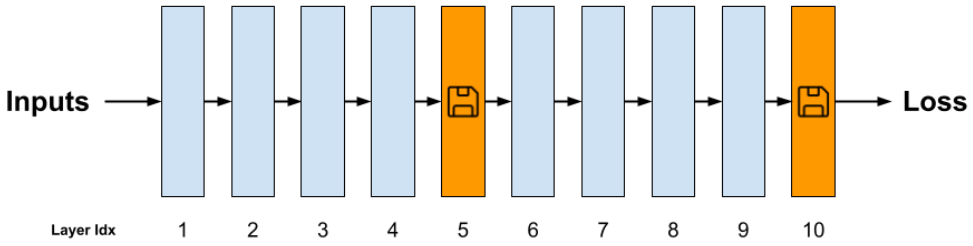


Figure 4: Tensor rematerialization in action - Layer 5 and Layer 10 activations are stored in memory along with the inputs. Activations for other layers are recomputed on-demand from stored activations and inputs.

(a) (10 pts) Assume a forward pass of a single layer is called a `fwd` operation. **How many `fwd` operations are invoked when running a single backward pass through the entire network?** Do not count the initial forward passes required to compute the loss, and don't worry about any extra computation beyond activations to actually backprop gradients.

(b) (5 pts) Assume that each memory access to fetch activations or inputs is called a `loadmem` operation. **How many `loadmem` operations are invoked when running a single backward pass?**

(c) (8 pts) Say you have access to a local disk which offers practically infinite storage for activations and a `loaddisk` operation for loading activations. You decide to not use tensor rematerialization and instead store all activations on this disk, loading each activation when required. Assuming each `fwd` operation takes 20ns and each `loadmem` operation (which loads from memory, not local disk) takes 10ns for tensor rematerialization, **how fast (in ns) should each `loaddisk` operation be to take the same time for one backward pass as tensor rematerialization?** Assume activations are directly loaded to the processor registers from disk (i.e., they do not have to go to memory first), only one operation can be run at a time, ignore any caching and assume latency of any other related operations is negligible.

PRINT your name and student ID: _____

7. Fine-tuning Large Models for Multiple Tasks (20 points)

In the context of fine-tuning large pre-trained foundation models for multiple tasks, consider the following three scenarios:

(1) Using a foundation model with different soft prompts tuned per task, while keeping the main model frozen. Assume prompts have a token length of 5.

(2) Using a foundation model held frozen, with task-specific low-rank adapters (i.e. we train A, B matrices to allow us to replace the relevant weight matrix W with $W + AB$ where A is initialized to zero and B is randomly initialized) fine-tuned for the attention-parameters (key, value, and query weight matrices) in the top four layers.

(3) Full-fine tuning of the entire model using the data from the multiple target tasks simultaneously.

You can assume that the foundation model has 13B parameters with a max context length of 512, hidden_dim 5120, Multi-head-attention with 40 heads and 40 layers, trained with a dataset consisting of 1T tokens.

(a) (5 pts) **Which of these scenarios is most likely to lead to catastrophic forgetting?** (select one)

- ◯ Scenario 1
- ◯ Scenario 2
- ◯ Scenario 3
- ◯ None of the above

(b) (5 pts) **What is the total number of trainable parameters using soft prompt tuning?**

(c) (4 pts) Suppose we decide to use meta-learning to get better initializations for the A and B matrices to be used for task-specific low-rank adaptation.

Assume you have a large family of tasks with lots of relevant training data. **Which meta-learning approach do you think will be more practically effective given this setting:** (select one)

- ◯ MAML with a number of inner iterations k of around 10. (Recall that MAML requires you to compute the gradients to the initial condition before the inner training iterations through the training iterations but on held-out test data.)
- ◯ REPTILE using a large batch of task-specific data at a time. (Recall that REPTILE uses the net movement from the initial condition of this meta-iteration as an approximation for the meta-gradient and just moves the meta-learned initial-condition a small step in that direction.)

(d) (6 pts) To better defend against catastrophic forgetting during your meta-learning approach in the previous part, **which of the following would you consider doing:** (mark all that apply)

- ☐ During meta-learning, include occasional gradient updates to A and B on random subsets of the original training data with the original self-supervised training loss.
- ☐ During meta-learning, include occasional gradient updates to A and B using the original self-supervised training loss but on the new training data from the training family of tasks.
- ☐ After your meta-learning updates are done, reinitialize the A matrices to zero before actually fine-tuning on new tasks.

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: _____

8. Analyzing Distributed Training (12 points)

For real-world models trained on lots of data, the training of neural networks is parallelized and accelerated by running workers on distributed resources, such as clusters of GPUs. In this question, we will explore three popular distributed training paradigms:

All-to-All Communication: Each worker maintains a copy of the model parameters (weights) and processes a subset of the training data. After each iteration, each worker communicates with every other worker and updates its local weights by averaging the gradients from all workers.

Parameter Server: A dedicated server, called the parameter server, stores the global model parameters. The workers compute gradients for a subset of the training data and send these gradients to the parameter server. The server then updates the global model parameters and sends the updated weights back to the workers.

Ring All-Reduce: Arranges n workers in a logical ring and updates the model parameters by passing messages in a circular fashion. Each worker computes gradients for a subset of the training data, splits the gradients into n equally sized chunks and sends a chunk of the gradients to their neighbors in the ring. Each worker receives the gradient chunks from its neighbors, updates its local parameters, and passes the updated gradient chunks along the ring. After $n-1$ passes, all gradient chunks have been aggregated across workers, and the aggregated chunks are passed along to all workers in the next $n-1$ steps. This is illustrated in Figure 5.

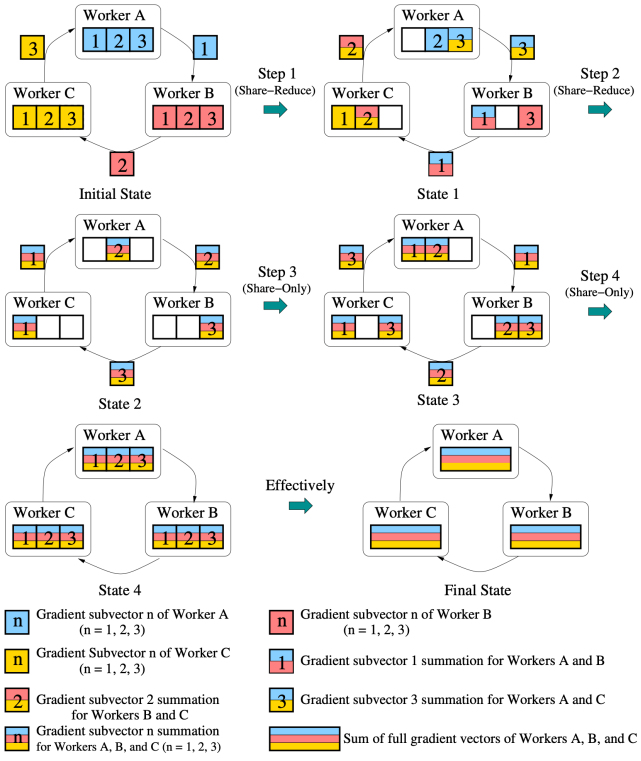


Figure 5: Example of Ring All-Reduce in a 3 worker setup. Source: Mu Et. al, *GADGET: Online Resource Optimization for Scheduling Ring-All-Reduce Learning Jobs*

For each of the distributed training paradigms, fill in the total number of messages sent and the size of each message. Assume that there are n workers and the model has p parameters, with p divisible by n.

	Number of Messages Sent	**Size of each message**
All-to-All		p
Parameter Server	$2n$	
Ring All-Reduce	$n(2(n-1))$	

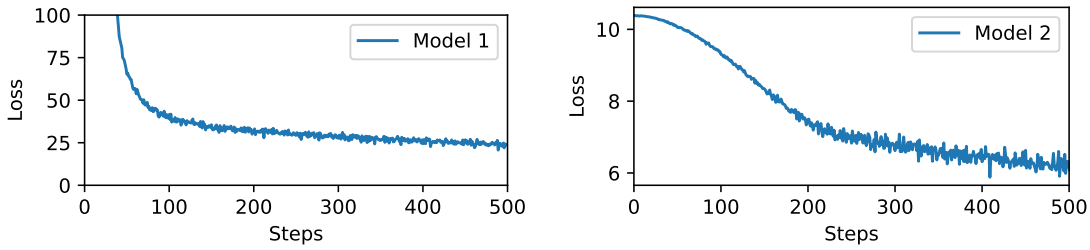
PRINT your name and student ID: _____

9. Debugging Transformers (24 points)

You're implementing a Transformer encoder-decoder model for document summarization (a sequence-to-sequence NLP task). You write the initialization of your embedding layer and head weights as below:

```
1  class Transformer(nn.Module):
2      def __init__(self, n_words, max_len, n_layers,
3                   d_model, n_heads, d_ffn, p_drop):
4          super().__init__()
5          self.emb_word = nn.Embedding(n_words, d_model)
6          self.emb_pos = nn.Embedding(max_len, d_model)
7
8          # Initialize embedding layers
9          self.emb_word.weight.data.normal_(mean=0, std=1)
10         self.emb_pos.weight.data.normal_(mean=0, std=1)
11
12         self.emb_ln = nn.LayerNorm(d_model)
13         self.encoder_layers = nn.ModuleList([
14             TransformerLayer(False, d_model, n_heads, d_ffn, p_drop)
15             for _ in range(n_layers)
16         ])
17         self.decoder_layers = nn.ModuleList([
18             TransformerLayer(True, d_model, n_heads, d_ffn, p_drop)
19             for _ in range(n_layers)
20         ])
21         self.lm_head = nn.Linear(d_model, n_words)
22         # Share lm_head weights with emb_word
23         self.lm_head.weight = self.emb_word.weight
24         self.criterion = nn.CrossEntropyLoss(ignore_index=-100)
```

After training this model, you compare your implementation with your friend's by looking at the loss curves:



(a) Your model's loss - 23.4

(b) Your friend's model's loss - 6.1

Figure 6: Comparing your model's loss vs your friend's model's loss. Your model is doing significantly worse.

Your friend suggests that there's something wrong with how the head gets initialized. **Identify the bug in your initialization, fix it by correcting the buggy lines, and briefly explain why your fix should work.**

Hint: remember that `d_model` *is large in transformer models.*

Hint: your change needs to impact line 9 somehow since that is where the head is initialized.

PRINT your name and student ID: _____

The bug: (brief description)

The fix (code): (Just show anything you change and/or add to the code.)

Why the fix should work: (brief explanation)

PRINT your name and student ID: _____

10. Diffusion Fun: To Infinity and Beyond! (28 points)

(a) (8 pts) One of the critical parts of being able to train a denoising diffusion model is to be able to quickly sample z_t and z_{t+1} given z_0 where z_0 is our initial data point. Gaussian diffusions make this easy to do.

Suppose that we are working in discrete time $t = 1, 2, \ldots$ and we know that the conditional variance for the one-step forward diffusion at time t is given by β_t. This means that in terms of conditional distributions, we have:

$$q(z_t|z_{t-1}) = \mathcal{N}(z_t; \underbrace{\sqrt{1-\beta_t}\, z_{t-1}}_{\text{mean}}, \overbrace{\beta_t I}^{\text{covariance}})$$

What is α_t so that the conditional distribution of z_t given z_0 is just Gaussian with a known mean and variance as follows:

$$q_(z_t|z_0) = \mathcal{N}(z_t; \sqrt{\alpha_t} z_0, (1-\alpha_t)I)$$

Your answer should involve the β_i values.

(b) (8 pts) Notice that the forward diffusion looks similar wherever you start. Suppose we wanted to quickly generate a sample of z_τ given z_t where $\tau > t$ and both are integers.

What is $\gamma_{\tau,t}$ so that the conditional distribution of z_τ given z_t is just Gaussian with a known mean and variance as follows:

$$q_(z_\tau|z_t) = \mathcal{N}(z_\tau; \sqrt{\gamma_{\tau,t}} z_t, (1-\gamma_{\tau,t})I)$$

Your answer should just involve the α_t and α_τ values.

Hint: Think about the means and leverage the previous part.

PRINT your name and student ID: _____

(c) (12 pts) At every step of the forward Gaussian diffusion, the "signal" of the original example gets weaker while the amount of "noise" gets higher. Consider the case of scalar z for simplicity. If the original z_0 comes from a zero-mean distribution with variance 1, then α_t represents the signal energy and $1-\alpha_t$ represents the noise energy. The ratio $\xi_t = \frac{\alpha_t}{1-\alpha_t}$ is the signal-to-noise ratio and decreases monotonically from $+\infty$ at $t=0$ where $\alpha_0 = 1$ down to $\alpha_\infty = 0$ when $t = +\infty$ and the signal has been completely lost.

However, this means that we can forget about discrete time entirely and just think about the continuous quantity $\eta = \frac{1}{\xi}$ where η naturally ranges from 0 to ∞. We can consider $\widetilde{z}(\eta)$ to be such that $\widetilde{z}(0) = z_0$ and $\widetilde{z}(\eta) = z_{t(\eta)}$ where $t(\eta)$ is whatever hypothetical t satisfies $\frac{1-\alpha_t}{\alpha_t} = \eta$.

Given $0 < \eta_1 < \eta_2 < \infty$, show how you would generate a sample of the pair $\widetilde{z}(\eta_1), \widetilde{z}(\eta_2)$ given access to z_0 and two iid standard normal random variables V_1, V_2 so that these two $\widetilde{z}(\eta_1), \widetilde{z}(\eta_2)$ are distributed in a manner that is compatible with a forward diffusion process sampled at two distinct times where the noise-to-signal ratios are η_1 and η_2 respectively.

PRINT your name and student ID: _____

11. Variational Information Bottleneck (26 points)

In class, you saw tricks that we introduced in the context of Variational Auto-Encoders to allow ourselves to get the latent space to have a desirable distribution. It turns out that we can use the same spirit even with tasks different than auto-encoding.

Consider a prediction task that maps an input source X to a target Y through a latent variable Z, as shown in the figure below. Our goal is to learn a latent encoding that is maximally useful for our target task, while trying to be close to a target distribution $r(Z)$.

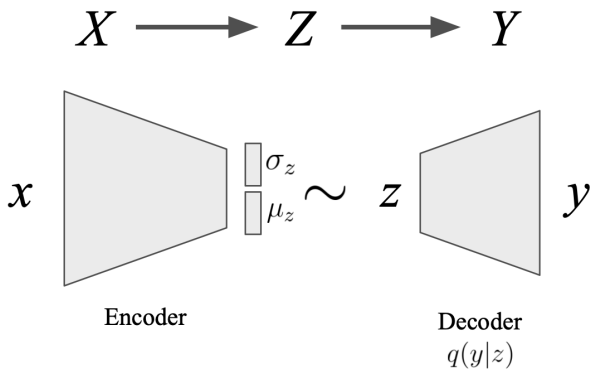


Figure 7: Overview of a VIB that maps an input X to the target Y through a latent variable Z (top). We use deep neural networks for both the encoder and task-relevant "decoder."

(a) (8 pts) Assume that we decide to have the encoder network (parameterized by θ_e) take both an input x and some independent randomness V to emit a random sample Z in the latent space drawn according to the Gaussian distribution $p_{\theta_e}(Z|x)$.

For this part, assume that we want Z to be a scalar Gaussian (conditioned on x) with mean μ and variance σ^2 with the encoder neural network emitting the two scalars μ and σ as functions of x. Assume that V is drawn from iid standard $\mathcal{N}(0,1)$ Gaussian random variables.

Draw a block diagram with multipliers and adders showing how we get Z from μ and σ along with V.

PRINT your name and student ID: _____

(b) (6 pts) Assume that our task is a classification-type task and the "decoder" network (parameterized by θ_d) emits scores for the different classes that we run through a softmax to get the distribution $q_{\theta_d}(y|z)$ over classes when the latent variable takes value z.

To train our networks using our N training points, we want to use SGD to approximately minimize the following loss:

$$L = \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{z\sim p_{\theta_e}(z|x_n)} \left[\underbrace{\log q_{\theta_d}(y_n|z)}_{\text{task loss}} \right] + \beta \overbrace{D_{KL}(p_{\theta_e}(Z|x_n)||r(Z))}^{\text{latent regularizer}} \tag{1}$$

where the y_n is the training label for input x_n and during training, we draw fresh randomness V each time we see an input, and we set $r(Z)$ to be a standard Gaussian $\mathcal{N}(0,1)$.

If we train using SGD treating the random samples of V as a part of the enternal input, **select all the loss terms that contribute (via backprop) to the gradients used to learn the encoder and decoder parameters**:

For encoder parameters θ_e: ☐ task loss ☐ latent regularizer

For decoder parameters θ_d: ☐ task loss ☐ latent regularizer

(c) (4 pts) Let's say we implemented the above information bottleneck for the task of MNIST classification. **Which of the curves in Figure 8 below best represents the trend of the *validation error* (on held-out data) with increasing regularization strength parameter β?** (select one)

◯**(a)** ◯**(b)** ◯**(c)** ◯**(d)**

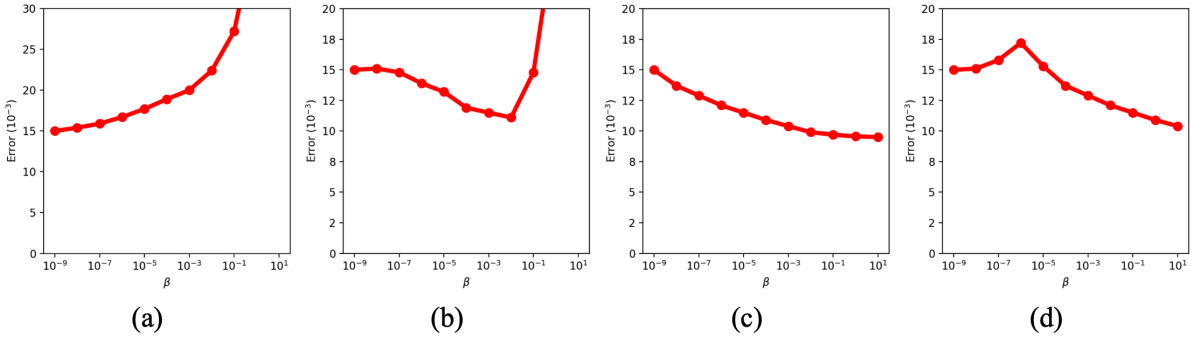


Figure 8: Validation error (on held-out data) profiles for different values of β.

PRINT your name and student ID: _____

(d) (8 pts) Let's say we implemented the above information bottleneck for the task of MNIST classification for three digits, and set the dimension of the latent space Z to 2. Figure 9 below shows the latent space embeddings of the input data, with different symbols corresponding to different class labels, for three choices of $\beta \in \{10^{-6}, 10^{-3}, 10^{0}\}$. Now answer these two questions:

i. **Guess the respective values of β used to generate the samples.** (select one for each fig)
(HINT: Don't forget to look at the axis labels to see the scale.)
(a) ◯ $\beta = 10^{-6}$ ◯ $\beta = 10^{-3}$ ◯ $\beta = 10^{0}$
(b) ◯ $\beta = 10^{-6}$ ◯ $\beta = 10^{-3}$ ◯ $\beta = 10^{0}$
(c) ◯ $\beta = 10^{-6}$ ◯ $\beta = 10^{-3}$ ◯ $\beta = 10^{0}$

ii. **Which of the three experiments in Figure 9 results in a *better* latent space for the prediction task?** (select one)
◯**(a)** ◯**(b)** ◯**(c)**

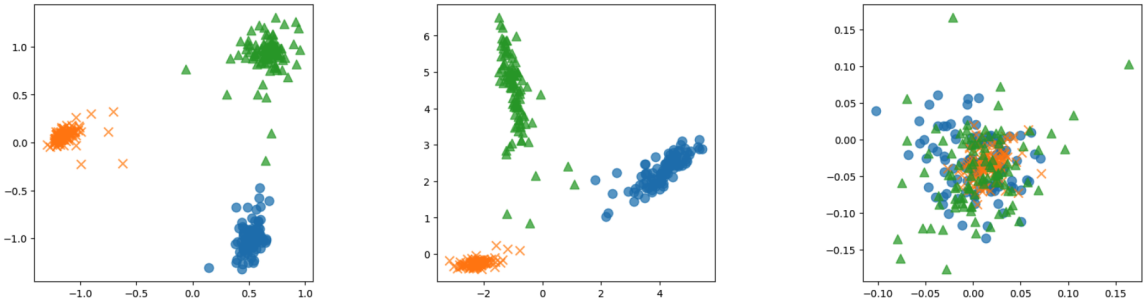


Figure 9: MNIST VIB with 2D latent space.

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: _____

12. LayerNorm and Transformer Models (63 points)

Consider a simplified formulation of LayerNorm written in PyTorch:

```
1  def f(x):
2      mu = x.mean(dim=-1)
3      z = x - mu
4      return z
5
6  def h(z):
7      sigma = (z ** 2).mean(dim=-1).sqrt()
8      y = z / sigma
9      return y
10
11 def LN(x):
12     return h(f(x))
```

Here the input $\mathbf{x} \in \mathbb{R}^d$ is a vector $[x_1, x_2, \ldots, x_d]^\top$ and the output $\mathbf{y} \in \mathbb{R}^d$ is a vector $[y_1, y_2, \ldots, y_d]^\top$ of the same shape.

For the demeaning function f, we have $f(\mathbf{x}) = \mathbf{z}$, with $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$. Given the upstream gradient $\mathbf{g}^{(z)} := (\frac{\partial \mathcal{L}}{\partial \mathbf{z}})^\top = [\frac{\partial \mathcal{L}}{\partial z_1}, \frac{\partial \mathcal{L}}{\partial z_2}, \ldots, \frac{\partial \mathcal{L}}{\partial z_d}]^\top \in \mathbb{R}^d$, we derive the backpropagation of f here for your convenience.

Since $\mathbf{z} = f(\mathbf{x}) = \mathbf{x} - \mu\mathbf{1}$ and $\mu = \frac{1}{d}\sum_{i=1}^d x_i = \frac{1}{d}\mathbf{1}^\top\mathbf{x}$, we obtain in vector notation:

$$\mathbf{z} = (\mathbf{I} - \frac{1}{d}\mathbf{1}\mathbf{1}^\top)\mathbf{x} \tag{2}$$

The Jacobian of f is thus:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \mathbf{I} - \frac{1}{d}\mathbf{1}\mathbf{1}^\top \tag{3}$$

The downstream gradient $\mathbf{g}^{(x)} := (\frac{\partial \mathcal{L}}{\partial \mathbf{x}})^\top = [\frac{\partial \mathcal{L}}{\partial x_1}, \frac{\partial \mathcal{L}}{\partial x_2}, \ldots, \frac{\partial \mathcal{L}}{\partial x_d}]^\top$ after backpropagation is thus:

$$\mathbf{g}^{(x)} = (\frac{\partial \mathcal{L}}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{x}})^\top = (\mathbf{I} - \frac{1}{d}\mathbf{1}\mathbf{1}^\top)\mathbf{g}^{(z)} \tag{4}$$

Notice, as expected, any "DC" component of the upstream gradient that wants to increase or decrease all the components by the same amount will not pass backward through the mean-removal function.

PRINT your name and student ID: _____

(a) (6 pts) For demeaning function f, suppose we linearly scale the input $\mathbf{x}' = a\mathbf{x}$, where $a > 0$; suppose the upstream gradient $\mathbf{g}^{(z)}$ remains unchanged. Let $\mathbf{g}^{(x')} := [\frac{\partial \mathcal{L}}{\partial x'_1}, \frac{\partial \mathcal{L}}{\partial x'_2}, \ldots, \frac{\partial \mathcal{L}}{\partial x'_d}]^\top$. **Which two of the following statements are true?**

- ☐ $\|f(\mathbf{x}')\|_2 = \|f(\mathbf{x})\|_2$
- ☐ $\|f(\mathbf{x}')\|_2 = a\|f(\mathbf{x})\|_2$
- ☐ $\|f(\mathbf{x}')\|_2 = \frac{1}{a}\|f(\mathbf{x})\|_2$
- ☐ $\|\mathbf{g}^{(x')}\|_2 = \|\mathbf{g}^{(x)}\|_2$
- ☐ $\|\mathbf{g}^{(x')}\|_2 = a\|\mathbf{g}^{(x)}\|_2$
- ☐ $\|\mathbf{g}^{(x')}\|_2 = \frac{1}{a}\|\mathbf{g}^{(x)}\|_2$

(b) (15 pts) For the normalizing function h, we have $h(\mathbf{z}) = \mathbf{y}$, where $\mathbf{z}, \mathbf{y} \in \mathbb{R}^d$. Given upstream gradient $\mathbf{g}^{(y)} := [\frac{\partial \mathcal{L}}{\partial y_1}, \frac{\partial \mathcal{L}}{\partial y_2}, \ldots, \frac{\partial \mathcal{L}}{\partial y_d}]^\top \in \mathbb{R}^d$, **derive the downstream gradient** $\mathbf{g}^{(z)} := [\frac{\partial \mathcal{L}}{\partial z_1}, \frac{\partial \mathcal{L}}{\partial z_2}, \ldots, \frac{\partial \mathcal{L}}{\partial z_d}]^\top$**, as a function of** d, $\mathbf{g}^{(y)}$ and $\mathbf{z}$. Recommended notation: denote variable `sigma` by σ.

Hint: To solve this question, it is easier to employ vector arithmetic and vector calculus. We strongly recommend expressing the normalizing function h in vector form as a first step. Alternatively, you could compute elementwise partial derivatives and simplify them, though this approach is more involved.

Hint: For a vector $\mathbf{u}$, the following relationship holds: $\frac{\partial \|\mathbf{u}\|_2}{\partial \mathbf{u}} = \frac{\mathbf{u}^\top}{\|\mathbf{u}\|_2}$. You may use this without proof in this question.

Hint: To partially check your work, which directional component in $\mathbf{g}^{(y)}$ shouldn't be able to pass through this backprop through a normalization function?

PRINT your name and student ID: _____

(c) (6 pts) For normalizing function h, suppose we linearly scale the input $\mathbf{z}' = a\mathbf{z}$, where $a > 0$; suppose upstream gradient $\mathbf{g}^{(y)}$ is unchanged and let $\mathbf{g}^{(z')} := [\frac{\partial \mathcal{L}}{\partial z'_1}, \frac{\partial \mathcal{L}}{\partial z'_2}, \ldots, \frac{\partial \mathcal{L}}{\partial z'_d}]^\top$. **Which two of the following choices are true?**

- ☐ $\|h(\mathbf{z}')\|_2 = \|h(\mathbf{z})\|_2$
- ☐ $\|h(\mathbf{z}')\|_2 = a\|h(\mathbf{z})\|_2$
- ☐ $\|h(\mathbf{z}')\|_2 = \frac{1}{a}\|h(\mathbf{z})\|_2$
- ☐ $\|\mathbf{g}^{(z')}\|_2 = \|\mathbf{g}^{(z)}\|_2$
- ☐ $\|\mathbf{g}^{(z')}\|_2 = a\|\mathbf{g}^{(z)}\|_2$
- ☐ $\|\mathbf{g}^{(z')}\|_2 = \frac{1}{a}\|\mathbf{g}^{(z)}\|_2$

(d) (9 pts) Consider the code below for Transformer layers, with dropouts omitted for simplicity. Here the `LN` layer is defined as above, which is layer normalization without `eps` and affine transformation parameterized β and γ. There are two types described below with both sharing the same constructor code.

```
# Constructor
self.self_attn = MultiheadAttention(d_model, n_heads)
self.self_attn_ln = LN
self.fc1 = nn.Linear(d_model, d_ffn)
self.fc2 = nn.Linear(d_ffn, d_model)
self.ffn_ln = LN

# Forward (Type 1)
residual = x
x = self.self_attn(x, x, x, padding_mask, causal=False)
x = self.self_attn_ln(x + residual)       # Normalize after
residual = x
x = self.fc2(F.relu(self.fc1(x)))
x = self.ffn_ln(x + residual)             # Normalize after

# Forward (Type 2)
residual = x
x = self.self_attn_ln(x)           # Normalize before attention
x = self.self_attn(x, x, x, padding_mask, causal=False) + residual
residual = x
x = self.ffn_ln(x)                 # Normalize before MLP
x = self.fc2(F.relu(self.fc1(x))) + residual
```

Select all options that are true:

- ☐ The Type 1 code implements a type of Transformer encoder-style layer.
- ☐ The Type 1 code implements a type of Transformer decoder-style layer.
- ☐ The Type 2 code implements a type of Transformer encoder-style layer.
- ☐ The Type 2 code implements a type of Transformer decoder-style layer.
- ☐ Type 1 and Type 2 implementations are mathematically equivalent.
- ☐ Type 1 and Type 2 implementations are not mathematically equivalent.

PRINT your name and student ID: _____

(e) (10 pts) Consider a Transformer model consisting of L Type 2 Transformer layers. We will investigate the behavior of the magnitudes of activations during the forward pass, as well as the gradients during the backward pass. To simplify our analysis, we assume that both the self-attention and feedforward modules are initialized as *identity* mappings, such that for any input vector $\mathbf{x}$:

- `self.self_attn(x, x, x) == x`
- `self.fc2(F.relu(self.fc1(x))) == x`

With identity self-attention and feed-forward modules, the Transformer now only consists of layer normalizations and residual connections, so each layer essentially simplifies to $\mathbf{x} \mapsto \mathbf{x} + \mathrm{LN}(\mathbf{x}) + \mathrm{LN}(\mathbf{x} + \mathrm{LN}(\mathbf{x}))$.

Suppose the input to the Transformer, denoted by $\mathbf{x}^{(0)}$, is already normalized, which means it satisfies $x_1^{(0)} + x_2^{(0)} + \cdots + x_d^{(0)} = 0$ and $\|\mathbf{x}^{(0)}\|_2 = \sqrt{d}$. Let the output of the ℓ-th layer be $\mathbf{x}^{(\ell)}$. **Determine $\mathbf{x}^{(\ell)}$ as a function of ℓ and $\mathbf{x}^{(0)}$, where $\ell = 1, 2, \ldots, L$.**

Hint: $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \mathrm{LN}(\mathbf{x}^{(0)}) + \mathrm{LN}(\mathbf{x}^{(0)} + \mathrm{LN}(\mathbf{x}^{(0)})) = 3\mathbf{x}^{(0)}$.

You might want to compute $\mathbf{x}^{(2)}$ to help see the general pattern.

PRINT your name and student ID: _____

NOTE: You should only attempt this part if you've already gotten all the other points that you think you can get on this exam.

(f) (20 pts) In the previous part, we examined the forward propagation of a Type 2 Transformer. In this section, we will investigate the behavior of gradient norms during the backward propagation. Retaining all assumptions from the previous part, we additionally assume that the loss is $\mathcal{L}$ and the upstream gradient of the last layer, denoted by $\mathbf{g}^{(L)} := (\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}})^\top$, satisfies $\|\mathbf{g}^{(L)}\|_2 = \sqrt{d}$. Your task is to **prove that the norm of the upstream gradient for the ℓ-th layer $\mathbf{g}^{(\ell)} := (\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(\ell)}})^\top$ is less than or equal to** $\frac{2L+1}{2\ell+1}\sqrt{d}$, i.e.

$$\|\mathbf{g}^{(\ell)}\|_2 \leq \frac{2L+1}{2\ell+1}\sqrt{d}, \text{ for any } \ell = 1, 2, \ldots, L \tag{5}$$

which suggests that the lower layers usually get gradients of greater magnitudes. As an example, consider the L-th layer (the last layer), for which the norm of the upstream gradient is $\|\mathbf{g}^{(L)}\|_2 = \sqrt{d} = \frac{2L+1}{2L+1}\sqrt{d}$.
You may use the following lemma without proof:

Lemma 1. If $\mathbf{y} = \mathrm{LN}(\mathbf{x})$, then

$$\left\|\frac{\partial \mathcal{L}}{\partial \mathbf{x}}\right\|_2 \leq \frac{\sqrt{d}}{\|\mathbf{x}\|_2}\left\|\frac{\partial \mathcal{L}}{\partial \mathbf{y}}\right\|_2$$

.

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: _____

[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.]

