Problem Set 2
EE 148a
Winter 2026

Originally Prepared By: Andrew Zabelo and Armeet Jatyani

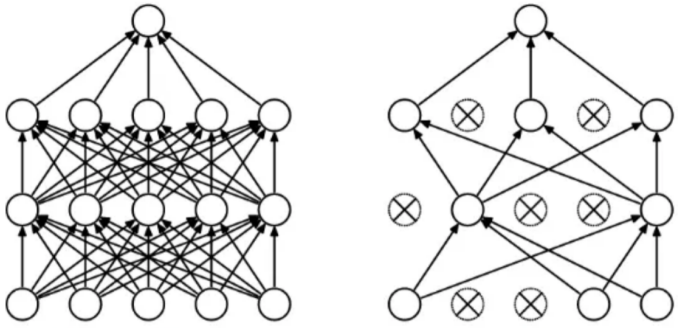
Collaboration is allowed, just list the names of the people you collaborated in the header. You are allowed to use AI to solve the coding problems.

You do not need to upload your code as part of the submission and the coding problems are not graded. However, coding problems can be a useful resource for your projects.

Question 1: Dropout (5 points)

1A Dropout equivalence to ℓ_2 regularization (5 points)

Dropout is a regularization technique used in neural networks that randomly omits (sets to zero with probability p) features being passed to specific layers during each training iteration. This forces the model to learn redundant representations, improving its ability to generalize to unseen data.



(a) Standard Neural Net (b) After applying dropout.

Show that OLS Regression with dropout is equivalent to Ridge Regression by proving that the expected squared loss with dropout is of the following form:

$$\mathbb{E}[\mathcal{L}(w)] = \|y - f(p)Xw\|^2 + g(p)\|\Gamma w\|^2$$

where $f(p)$ and $g(p)$ are functions of p, and Γ is a diagonal matrix with the standard deviations of features in data matrix X (derived from the Gram matrix X^TX)[1]. This result should provide some insight into how dropout performs a similar regularization role as an ℓ_2 norm penalty on model parameters. More generally, this equivalence is approximately true for multilayer perceptrons.

1B Dropout Code (0 points)

Create a copy of this Dropout Colab Notebook and rename it to Dropout_{first_name}_last_name}.ipynb. Follow the instructions to generate a Loss vs. Dropout Rate plot demonstrating equivalence between dropout and ℓ_2 regularization.

[1]Hint: first find the mean and variance of data matrix X after a dropout mask is applied

Question 2: Batch Normalization (5 points)

Question 2A BatchNorm Backpropagation (5 points)

Batch Normalization is a technique that normalizes the input and applies an affine transformation across the batch dimension. It has been observed to improve training stability by mitigating internal covariance shift and reducing dependency on initialization.

Specifically, if $\mathbf{X}$ is an $N \times D$ data matrix for the batch with batch size N and D features, the $D \times 1$ vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$ are found by taking the mean and variance, respectively, for each feature across the batch dimension:

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} X_{ij}$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (X_{ij} - \mu_j)^2$$

Using these values, Z-score normalization is applied to obtain $\hat{\mathbf{X}}$, with ϵ being a small constant added for numerical stability.

$$\hat{X}_{ij} = \frac{X_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Finally, an affine transformation is applied to $\hat{\mathbf{X}}$ using learnable parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, which are representable by $D \times 1$ vectors:

$$Y_{ij} = BN_{\gamma,\beta}(X_{ij}) = \gamma_j \hat{X}_{ij} + \beta_j$$

Given $\frac{\partial \mathcal{L}}{\partial Y}$[2], derive expressions for the following:

- gradient of β: $\frac{\partial \mathcal{L}}{\partial \beta}$
- gradient of γ: $\frac{\partial \mathcal{L}}{\partial \gamma}$
- gradient signal for input[3] : $\frac{\partial \mathcal{L}}{\partial X}$

2B Batch Normalization Code (0 points)

Create a copy of this BatchNorm Colab Notebook and rename it to BatchNorm_{first_name}_last_name}.ipynb. Follow the instructions to implement the forwards and backwards passes of BatchNorm from scratch and verify that your implementation matches the behavior of PyTorch's built-in BatchNorm function.

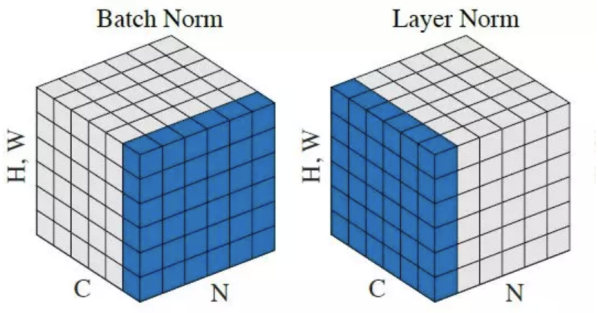
[2] $\mathcal{L}$ is the loss and Y is the output of BatchNorm, making $\frac{\partial \mathcal{L}}{\partial Y}$ the gradient signal with respect to the output of BatchNorm)

[3] Hint 1: start with the D=1 case. Hint 2: express σ^2 in terms of X and μ. Hint 3: apply chain rule and product rule on the vector quantities. Hint 4: the derivative of a vector function w.r.t. a vector is a matrix (the Jacobian)

Question 3: Layer Normalization (10 points)

3A LayerNorm Backpropagation (5 points)

Layer Normalization is very similar to Batch Normalization. Rather than normalizing across the batch dimension for each feature, we instead normalize across the feature dimension for each example in the batch. The affine transformation is still applied across the batch dimension. Thus, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$ are now $N \times 1$ vectors, while $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are still $D \times 1$. LayerNorm is more commonly used in NLP tasks, while BatchNorm is more prevalent in Computer Vision tasks.



$$\mu_i = \frac{1}{D}\sum_{j=1}^{D} X_{ij}$$

$$\sigma_i^2 = \frac{1}{D}\sum_{j=1}^{D} (X_{ij} - \mu_i)^2$$

$$\hat{X}_{ij} = \frac{X_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$$Y_{ij} = LN_{\gamma,\beta}(X_{ij}) = \gamma_j \hat{X}_{ij} + \beta_j$$

Given $\frac{\partial \mathcal{L}}{\partial Y}$[4], derive expressions for the following:

- gradient of β: $\frac{\partial \mathcal{L}}{\partial \beta}$
- gradient of γ: $\frac{\partial \mathcal{L}}{\partial \gamma}$
- gradient signal for input[5] : $\frac{\partial \mathcal{L}}{\partial X}$

[4] $\mathcal{L}$ is the loss and Y is the output of LayerNorm, making $\frac{\partial \mathcal{L}}{\partial Y}$ the gradient signal with respect to the output of LayerNorm)

[5] Hint: most of your work from solving problem 2A can be reused, adjusting for the fact that the Z Score (normalization) transform is done across a different axis.

3B Layer Normalization Code (0 points)

Create a copy of this LayerNorm Colab Notebook and rename it to LayerNorm_{first_name}_last_name}.ipynb. Follow the instructions to implement the forwards and backwards passes of LayerNorm from scratch and verify that your implementation matches the behavior of PyTorch's built-in LayerNorm function.

3C BatchNorm LayerNorm Puzzle (5 points)

Alice generates a random symmetric 16×16 matrix $\boldsymbol{X}$, which can be thought of as representing a batch with 16 features and 16 examples. She keeps $\boldsymbol{X}$ private.

Alice then performs Batch Normalization on this matrix with $\gamma = 1$ and $\beta = 0$, resulting in matrix $\boldsymbol{Y}$. $\boldsymbol{Y}$ is public.

Alice also performs Layer Normalization on the same matrix $\boldsymbol{X}$, with the same γ and β, resulting in matrix $\boldsymbol{Z}$. She keeps $\boldsymbol{Z}$ private.

Bob knows that $\boldsymbol{X}$ was symmetric and has access to $\boldsymbol{Y}$.

How many elements in $\boldsymbol{Z}$ does Bob know with certainty?

What if γ and β were arbitrary (i.e., not necessarily 1 and 0 respectively)? How many elements in $\boldsymbol{Z}$ would Bob know with certainty?

Question 4: Regularization, Optimizers, Augmentation in MNIST (40 points)

In this question, we will demonstrate several concepts involving regularization, optimizers, and data augmentation through the MNIST dataset. Throughout this problem, we will train a simple fully connected neural network in different ways without concerning ourselves with model architecture or hyperparameter tuning.

Create a copy of this MNIST Colab Notebook and rename it to MNIST_{first_name}_last_name}.ipynb.. Begin by running the starter code to download the MNIST dataset. Follow the instructions to implement various methods. Unless otherwise stated, train for 10 epochs with a learning rate of 0.01, a batch size of 64, no momentum, and reset the random seed to 42 before every run. **Please note again that you do not need to submit the solutions for the coding problems (which are the problems assigned 0 points).**

4A: Regularization (0 points)

1. (0 points) Plot the distribution of weights of a newly initialized model. Let X be a random variable that is the value of a parameter chosen at random from our uninitialized model. Write an expression for the PDF of X. Why is it beneficial for the standard deviation of the initial distribution of a layer to decrease as layer width increases? [6]

2. (0 points) Plot the distribution of weights of a model trained using SGD. Qualitatively, what is the new distribution of weights?

3. (0 points) Now plot the distribution of weights of a model trained using SGD and weight decay 0.01. Qualitatively, what changed about the distribution? In theory, why should this help a model's ability to generalize to unseen data?

4. (5 points) ℓ_2 regularization on weights (not to be confused with ℓ_2 regularization on model outputs) and weight decay are often used interchangeably, but they are in fact distinct methods. Prove that for standard SGD, ℓ_2 regularization and weight decay are equivalent.

   Weight decay update: $w_{t+1} \leftarrow w_t - (\nabla_w \mathcal{L} + \lambda_{weightdecay} w_t)$
   ℓ_2 regularized update: $w_{t+1} \leftarrow w_t - (\nabla_w \mathcal{L})$ where $\mathcal{L} = \lambda_{\ell_2} \sum_i w_i^2$

4B: SGD and Weight Decay (5 points)

1. (5 points) ℓ_2 regularization on weights (not to be confused with ℓ_2 regularization on model outputs) and weight decay are often used interchangeably, but they are in fact distinct methods. Prove that for standard SGD, ℓ_2 regularization and weight decay are equivalent.

   Weight decay update: $w_{t+1} \leftarrow w_t - (\nabla_w \mathcal{L} + \lambda_{weightdecay} w_t)$
   ℓ_2 regularized update: $w_{t+1} \leftarrow w_t - (\nabla_w \mathcal{L})$ where $\mathcal{L} = \lambda_{\ell_2} \sum_i w_i^2$

[6] Hint: PyTorch's initialization of linear layers is neither Kaiming nor Xaiver. You may use Google to figure out the initialization method.

4C: Optimizers (15 points)

1. (5 points) Derive the bias correction term $\frac{1}{1-\beta^t}$ for the Adam optimizer (i.e. show that the term is necessary to un-bias the estimates for the first and second moments)[7].

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

2. (5 points) We showed that ℓ_2 regularization is equivalent to weight decay when using SGD. Why is it problematic to add an ℓ_2 regularization penalty to the loss when using SGD with momentum? What if you use the RMSProp optimizer instead? What if you use the Adam optimizer?

3. (5 points) Explain how AdamW correctly penalizes weights in contrast with Adam.[8]

4D: Comparing Optimizers (0 points)

1. (0 points) Run the following and report the validation accuracies. Which 2 methods perform best? Why do the others perform worse?

   - Adam, default
   - Adam, ℓ_2_lambda=0.005
   - Adam, weight_decay=0.01
   - AdamW, ℓ_2_lambda=0.005
   - AdamW, weight_decay=0.01

[7]Hint: treat the initialization to 0 of a moment as a measurement of the moment yielding 0. Then a valid bias correction term is one that removes the contribution of this first 0 measurement from the exponentially decaying weighted average

[8]Hint: See AdamW paper.

4E: Data Augmentation (0 points)

Data augmentation is the practice of artificially increasing the size and diversity of a training dataset by applying transformations to which the labels are invariant (especially useful for smaller datasets). In this part, we will use small_dataset, a 10% subset of the 50000 images from the full training split. For this entire part, train using SGD, 10 epochs, batch size 64, learning rate 0.01, momentum 0.9, and reset the random seed to 42 before every run.

1. Some common transformations for images are rotations, translations, blurring, scaling, shear, Gaussian noise addition, etc. Implement a method for each of these.[9]

2. Display the effects of each augmentation one example of each digit 0-9 (augmented images should be noticeably different from the original in most cases and should still be clearly recognizable). Include your plot in your submission.

3. By definition, labels should be largely invariant to the augmentations. This means that a good augmentation can be applied to the whole dataset and should not significantly impact performance. First, train the model with small_dataset as a baseline and note the best validation accuracy. Then, write a method that applies a specific augmentation to every image in small_dataset. Finally, for each type of augmentation, experiment with different strengths (i.e. vary the angle_range, max_shift, etc.), apply it on small_dataset and train a model using the new dataset, and report the following:[10]

   - An example of an augmentation strength that decreases accuracy from the baseline by $\leq$1%
   - An example of an augmentation strength that decreases accuracy from the baseline by 1-3%
   - An example of an augmentation strength that decreases accuracy by $\geq$3%

4. Why is setting the allowable range for rotations too high a bad idea? Try this by setting the angle_range to (-180,180) and plot a confusion matrix to deduce which digit is misclassified as which other digit most often. Include your plot in your submission.

5. Develop your own method to randomly apply your augmentations on the small_train_dataset to augment it back to the original size of 50000 images. Improve your augmentation methods until the best validation accuracy is at least 90% (you must use SGD, 10 epochs, batch size=64, 0.01 learning rate, 0.9 momentum for all). Report your validation accuracies when training on the following datasets:

   - Small dataset (5000 images)
   - Small dataset copied 10 times [11] (50000 images)
   - Your augmented dataset (50000 images)
   - Original dataset (50000 images)

[9] You are free to use external python libraries

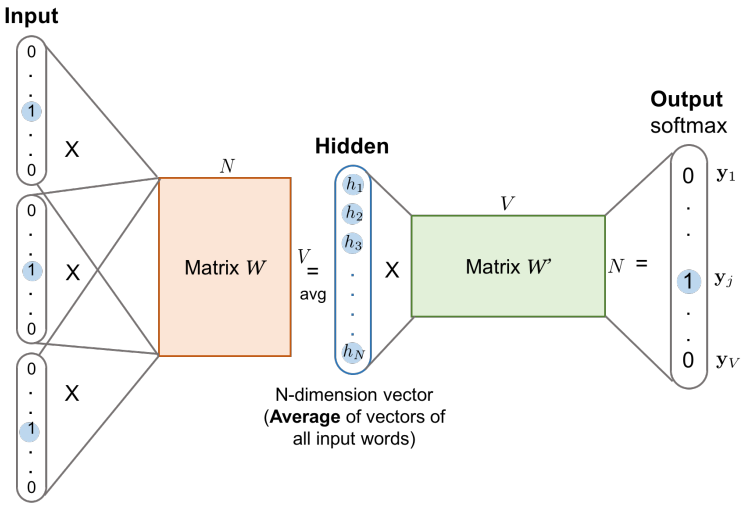
[10] Hint: modularizing your code to iterate quickly

[11] This is a control to show that augmentations improve model performance beyond just doing more iterations

Question 5[12]: Word2Vec (0 points)

5A: Implementing Continuous Bag of Words (CBOW) (0 points)

Word embeddings are dense vector representations of words, where semantically similar words are represented by vectors that are closer together in the vector space. The Word2Vec paper introduced two methods for learning word embeddings: Skip-Gram and Continuous Bag of Words (CBOW). Both methods are motivated by the observation that the semantic concepts of a word can be inferred from neighboring words. Skip-Gram predicts surrounding context words from a single word, while CBOW predicts a single word from surrounding context words.



In this question, you will implement CBOW and train it with the Shakespeare dataset (a tiny dataset by NLP standards). Create a copy of this CBOW Colab Notebook and rename it to CBOW_{first_name}_last_name}.ipynb.

- Begin by running the provided code to download the raw dataset.
- Process the dataset by:
  - Filtering out words that have less than 5 occurrences.
  - One-hot encoding the remaining words.
  - Using a sliding window to create input-target pairs of target words and the surrounding context words (recommended: 2 before and 2 after the target).[13]
- Implement the CBOW model:
  - Use one linear layer to encode context words into the embedding space and take the average (experiment with different embedding dimensions).
  - Use another linear layer to decode the embedding back into the dimensionality of the vocabulary, followed by a LogSoftmax. This means that we should use Negative Log Likelihood (NLL) Loss. Alternatively, we could have used CrossEntropy Loss without any Softmax layers.
  - Add a method to find the closest word in the vocabulary to a given embedding via cosine similarity.

[12]This question will likely involve 6 to 12 hours time training, so it is advisable to tackle this problem first and not to wait until the day before the deadline

[13]Hint: create a mapping from words to indices and vice versa. Then, pass indices to your model and one-hot encode them at runtime for more efficient memory usage

- Implement a training loop. Save the loss of every batch and use it to generate two plots after every epoch:
  - Loss versus batches processed (since the beginning of training, not just the current epoch).
  - Loss versus batches processed in log-log scale (since the beginning of training, not just the current epoch).
- Choose an optimizer, batch size, learning rate, number of epochs, and regularization, and train your CBOW model.[14] You may need to do some hyperparameter tuning to learn good embeddings. Feel free to use additional techniques as you see fit.

5B: Semantically similar clusters (0 points)

Find 10 different clusters of 10 semantically similar words and briefly state what the words have in common (given the small dataset, the results may be slightly noisy and they need not be exact synonyms). For example: the cluster with the word 'my' at the center is ['my', 'thy', 'your', 'his', 'hastings", 'our', 'antony's', 'petruchio's', 'their', 'her', 'timon's'], representing possession. Your results should be reproducible from your code submission.

Bonus Question: Semantically meaningful directions (0 points)

Demonstrate the existence of semantically meaningful directions in the embedding space by finding 3 different vector analogies. Examples of vector analogies are embed("man") + embed("royal") $\approx$ embed("king") or embed("king") - embed("man") + embed("woman") $\approx$ embed("queen").[15]

Note that this is a more difficult task than part B) and may require more careful model training. Your results should be reproducible from your code submission.

[14] Hint 1: print the closest words to specific words during training to see if your model begins to learn semantic relationships. Use your log-log plot to see if the loss has converged, and to compare different runs. Both tips can help you determine whether a run is worth continuing, allowing you to iterate faster.

[15] Since king and queen and woman and queen are likely already close in the embedding space, valid demonstration would need to show that replacing any word in the analogy with any one of several irrelevant word no longer works. For instance, embed("man") + embed("tree") $\not\approx$ embed("king") or embed("king") - embed("tree") + embed("woman") $\not\approx$ embed("queen").