Assignment 2

Release Date: 2025-02-21

Deadline: Monday, March 10th, at 1:00pm.

Submission: You must submit two files through MarkUs: (1) a PDF file containing your writeup, titled `a2-writeup.pdf`, and (2) your code file `a2-code.ipynb`. There will be sections in the notebook for you to write your responses. Your writeup may be typed or handwritten, but please ensure it is legible. Make sure that the relevant outputs (e.g. `print_gradients()` outputs, plots, etc.) are included and clearly visible.

See the syllabus on the course website for detailed policies. You may ask questions about the assignment on Piazza.

The assignment is marked out of 100 points. 10 points are allocated to neatness, so be sure that your answers and code are clear and readable!

You may notice that some questions are worth 0 points, which means we will not mark them in this assignment. However, we think these questions are educationally valuable and may help you prepare for the tests, so we recommend you spend some time thinking about them.

Important Instructions

Read the following before attempting the assignment.

Collaboration Policy

You are welcome to work together with other students on the homework. You are also welcome to use any resources you find (online tutorials, textbooks, papers, etc.) to help you complete the homework. However, you must write up your submission by yourself and not use any content generated by someone else or generative AI. You must cite all the collaboration and resources you used to complete each assignment. If you hand in homework that makes use of content that you did not create or you do not disclose the collaboration or resources, you will get a 0 for that homework. Note also that if you rely too much on outside resources, you will likely not learn the material and will do poorly on the exams, during which such resources will not be available.

Generative AI Policy

You may ask general questions about concepts related to the homework questions. However, you may not directly ask them for hints on homework assignment questions; e.g., you should not be copying and pasting directly from the assignment handout into the chat interface. Also, you may not directly use the outputs of AI chatbots in your homework solutions (even paraphrased) unless instructed to do so.

You must include any chat transcripts related to a homework assignment along with your submission for the assignment. We will interpret the above policy leniently when judging whether the GenAI use is appropriate, though we reserve the right to update the GenAI policy if we find that students are using it in a way that reduces the educational value of the homeworks.

If you use GenAI, then your transcripts should be submitted on MarkUs with filenames starting with `chat_transcript`.

1 Optimization [14pt]

This week, we will continue investigating the properties of optimization algorithms, focusing on stochastic gradient descent and adaptive gradient descent methods. For a refresher on optimization, refer to: `https://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/readings/L07%20Optimization.pdf`.

We will continue using the linear regression model established in Homework 1. Given n pairs of input data with d features and scalar labels $(\mathbf{x}_i, t_i) \in \mathbb{R}^d \times \mathbb{R}$, we want to find a linear model $f(\mathbf{x}) = \hat{\mathbf{w}}^T\mathbf{x}$ with $\hat{\mathbf{w}} \in \mathbb{R}^d$ such that the squared error on training data is minimized. Given a data matrix $X \in \mathbb{R}^{n\times d}$ and corresponding labels $\mathbf{t} \in \mathbb{R}^n$, the objective function is defined as:

$$\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 \tag{1}$$

1.1 Mini-Batch Stochastic Gradient Descent (SGD)

Mini-batch SGD performs optimization by taking the average gradient over a mini-batch, denoted $\mathcal{B} \in \mathbb{R}^{b\times d}$, where $1 < b \ll n$. Each training example in the mini-batch, denoted $\mathbf{x}_j \in \mathcal{B}$, is randomly sampled without replacement from the data matrix X. Assume that X is full rank. Where $\mathcal{L}$ denotes the loss on $\mathbf{x}_j$, the update for a single step of mini-batch SGD at time t with scalar learning rate η is:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta}{b}\sum_{\mathbf{x}_j\in\mathcal{B}} \nabla_{\mathbf{w}_t}\mathcal{L}(\mathbf{x}_j, \mathbf{w}_t) \tag{2}$$

Mini-batch SGD iterates by randomly drawing mini-batches and updating model weights using the above equation until convergence is reached.

1.1.1 Minimum Norm Solution [10pt]

Recall Question 1.3.2 from Homework 1. For an overparameterized linear model, gradient descent starting from zero initialization finds the unique minimum norm solution $\mathbf{w}^*$ such that $X\mathbf{w}^* = \mathbf{t}$. Let $\mathbf{w}_0 = \mathbf{0}$, $d > n$. Assume mini-batch SGD also converges to a solution $\hat{\mathbf{w}}$ such that $X\hat{\mathbf{w}} = \mathbf{t}$. Show that mini-batch SGD solution is identical to the minimum norm solution $\mathbf{w}^*$ obtained by gradient descent, i.e., $\hat{\mathbf{w}} = \mathbf{w}^*$.

Hint: Be more specific as to what other solutions? Or is $\mathbf{x}_j$ or $\mathcal{B}$ contained in span of X? Do the update steps of mini-batch SGD ever leave the span of X?

1.2 Adaptive Methods

We now consider the behavior of adaptive gradient descent methods. In particular, we will investigate the RMSProp method. Let w_i denote the i-th parameter. A scalar learning rate η is used.

At time t for parameter i, the update step for RMSProp is shown by:

$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{v_{i,t}} + \epsilon} \nabla_{w_{i,t}} \mathcal{L}(w_{i,t}) \tag{3}$$

$$v_{i,t} = \beta(v_{i,t-1}) + (1-\beta)(\nabla_{w_{i,t}} \mathcal{L}(w_{i,t}))^2 \tag{4}$$

We begin the iteration at $t = 0$, and set $v_{i,-1} = 0$. The term ϵ is a fixed small scalar used for numerical stability. The momentum parameter β is typically set such that $\beta \geq 0.9$ Intuitively, RMSProp adapts a separate learning rate in each dimension to efficiently move through badly formed curvatures (see lecture slides/notes).

1.2.1 Minimum Norm Solution [4pt]

Consider the overparameterized linear model ($d > n$) for the loss function defined in Section 1. Assume the RMSProp optimizer converges to a solution. Provide a proof or counterexample for whether RMSProp always obtains the minimum norm solution.

Hint: Compute a simple 2D case. Let $\mathbf{x}_1 = [2, 1]$, $w_0 = [0, 0]$, $t = [2]$.

1.2.2 [0pt]

Consider the result from the previous section. Does this result hold true for other adaptive methods (Adagrad, Adam) in general? Why might making learning rates independent per dimension be desirable?

2 Gradient-based Hyper-parameter Optimization [24pt]

In this problem, we will implement a simple toy example of *gradient-based hyper-parameter optimization*.

Often in practice, hyper-parameters are chosen by trial-and-error based on a model evaluation criterion. Instead, *g*radient-based hyper-parameter optimization computes gradient of the evaluation criterion w.r.t. the hyper-parameters and uses this gradient to directly optimize for the best set of hyper-parameters. For this problem, we will optimize for the learning rate of gradient descent in a regularized linear regression problem.

Specifically, given n pairs of input data with d features and scalar label $(\mathbf{x}_i, t_i) \in \mathbb{R}^d \times \mathbb{R}$, we wish to find a linear model $f(\mathbf{x}) = \hat{\mathbf{w}}^\top \mathbf{x}$ with $\hat{\mathbf{w}} \in \mathbb{R}^d$ and a L2 penalty, $\lambda \|\hat{\mathbf{w}}_2^2\|$, that minimizes the squared error of prediction on the training samples. λ is a hyperparameter that modulates the impact of the L2 regularization on the loss function. Using the concise notation for the data matrix $X \in \mathbb{R}^{n \times d}$ and the corresponding label vector $\mathbf{t} \in \mathbb{R}^n$, the squared error loss can be written as:

$$\tilde{\mathcal{L}} = \frac{1}{n} \|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 + \lambda \|\hat{\mathbf{w}}\|_2^2.$$

Starting with an initial weight parameters $\mathbf{w}_0$, gradient descent (GD) updates $\mathbf{w}_0$ with a learning rate η for t number of iterations. Let's denote the weights after t iterations of GD as $\mathbf{w}_t$, the loss as $\mathcal{L}_t$, and its gradient as $\nabla_{\mathbf{w}_t}$. The goal is the find the optimal learning rate by following the gradient of $\mathcal{L}_t$ w.r.t. the learning rate η.

2.1 Optimal Learning Rates

In this section, we will take a closer look at the gradient w.r.t. the learning rate. To simplify the computation for this section, consider an unregularized loss function of the form $\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2$. Let's start with the case with only one GD iteration, where GD updates the model weights from $\mathbf{w}_0$ to $\mathbf{w}_1$.

2.1.1 [8pt]

Write down the expression of $\mathbf{w}_1$ in terms of $\mathbf{w}_0$, η, $\mathbf{t}$ and X. Then use the expression to derive the loss $\mathcal{L}_1$ in terms of η.
Hint: If the expression gets too messy, introduce a constant vector $\mathbf{a} = X\mathbf{w}_0 - \mathbf{t}$

2.1.2 [0pt]

Determine if $\mathcal{L}_1$ is convex w.r.t. the learning rate η.
Hint: A function is *convex* if its second order derivative is positive

2.1.3 [4pt]

Write down the derivative of $\mathcal{L}_1$ w.r.t. η and use it to find the optimal learning rate η^* that minimizes the loss after one GD iteration. Show your work.

2.2 Weight decay and L2 regularization

Although well studied in statistics, L2 regularization is usually replaced with explicit weight decay in modern neural network architectures:

$$\mathbf{w}_{i+1} = (1 - \lambda)\mathbf{w}_i - \eta\nabla\mathcal{L}_i(X) \tag{5}$$

In this question you will compare regularized regression of the form $\tilde{\mathcal{L}} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 + \tilde{\lambda}\|\hat{\mathbf{w}}\|_2^2$ with unregularized loss, $\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2$, accompanied by weight decay (equation 5).

2.2.1 [8pt]

Write down two expressions for $\mathbf{w}_1$ in terms of $\mathbf{w}_0$, η, $\mathbf{t}$, λ, $\tilde{\lambda}$, and X. The first one using $\tilde{\mathcal{L}}$, the second with $\mathcal{L}$ and weight decay.

2.2.2 [4pt]

How can you express $\tilde{\lambda}$ (corresponding to L2 loss) so that it is equivalent to λ (corresponding to weight decay)?
Hint: Think about how you can express $\tilde{\lambda}$ in terms of λ and another hyperparameter.

2.2.3 [0pt]

Adaptive gradient update methods like RMSprop (equation 4) modulate the learning rate for each weight individually. Can you describe how L2 regularization is different from weight decay when adaptive gradient methods are used? In practice it has been shown that for adaptive gradients methods weight decay is more successful than l2 regularization.

3 Trading off Resources in Neural Net Training [16pt]

3.1 Effect of batch size

When training neural networks, it is important to select an appropriate batch size. In this question, we will investigate the effect of batch size on some important quantities in neural network training.

3.1.1 Batch size vs. learning rate

Batch size affects the stochasticity in optimization, and therefore affects the choice of learning rate. We demonstrate this via a simple model called the noisy quadratic model (NQM). Despite the simplicity, the NQM captures many essential features in realistic neural network training **?**.

For simplicity, we only consider the scalar version of the NQM. We have the quadratic loss $\mathcal{L}(w) = \frac{1}{2}aw^2$, where $a > 0$ and $w \in \mathbb{R}$ is the weight that we would like to optimize. Assume that we only have access to a noisy version of the gradient — each time when we make a query for the gradient, we obtain $g(w)$, which is the true gradient $\nabla\mathcal{L}(w)$ with additive Guassian noise:

$$g(w) = \nabla\mathcal{L}(w) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2).$$

One way to reduce noise in the gradient is to use minibatch training. Let B be the batch size, and denote the minibatch gradient as $g_B(w)$:

$$g_B(w) = \frac{1}{B}\sum_{i=1}^{B} g_i(w), \quad \text{where } g_i(w) = \nabla\mathcal{L}(w) + \epsilon_i, \quad \epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2).$$

(a) [4pt] As batch size increases, how do you expect the optimal learning rate to change? Briefly explain in 2-3 sentences.

(Hint: Think about how the minibatch gradient noise change with B.)

3.1.2 Training steps vs. batch size

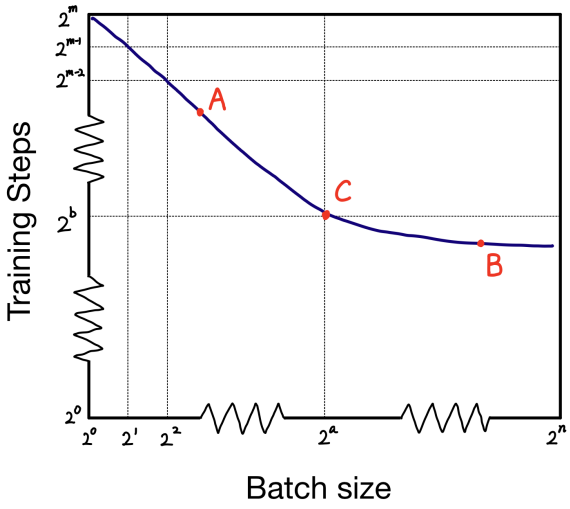


Figure 1: A cartoon illustration of the typical relationship between training steps and the batch size for reaching a certain validation loss (based on **?**). Learning rate and other related hyperparameters are tuned for each point on the curve.

For most of neural network training in the real-world applications, we often observe the relationship of training steps and batch size for reaching a certain validation loss as illustrated in Figure 1.

(a) [4pt] For the three points (A, B, C) on Figure 1, which one has the most efficient batch size (in terms of best resource and training time trade-off)? Assume that you have access to scalable (but not free) compute such that minibatches are parallelized efficiently. Briefly explain in 1-2 sentences.

(b) [4pt] Figure 1 demonstrates that there are often two regimes in neural network training: the noise dominated regime and the curvature dominated regime. In the noise dominated regime, the bottleneck for optimization is that there exists a large amount of gradient noise. In the curvature dominated regime, the bottleneck of optimization is the ill-conditioned loss landscape. For points A and B on Figure 1, which regimes do they belong to, and what would you do to accelerate training? Fill each of the blanks with **one** best suited option.

Point A: Regime: _____. Potential way to accelerate training: _____.

Point B: Regime: _____. Potential way to accelerate training: _____.

Options:

- Regimes: noise dominated / curvature dominated.
- Potential ways to accelerate training: use higher order optimizers / seek parallel compute

3.2 Model size, dataset size and compute

We have seen in the previous section that batch size is an important hyperparameter during training. Besides efficiently minimizing the training loss, we are also interested in the test loss. Recently, researchers have observed an intriguing relationship between the test loss and hyperparameters such as the model size, dataset size and the amount of compute used. We explore this relationship for neural language models in this section. The figures in this question are from **?**.

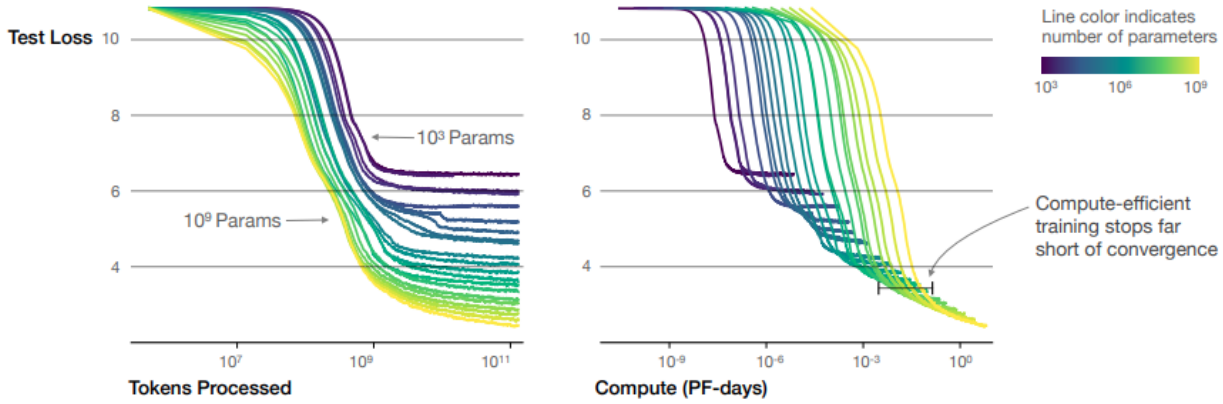


Figure 2: Test loss of language models of different sizes, plotted against the dataset size (tokens processed) and the amount of compute (in petaflop/s-days).

(a) [4pt] Previously, you have trained a neural language model and obtained somewhat adequate performance. You have now secured more compute resources (in PF-days), and want to improve

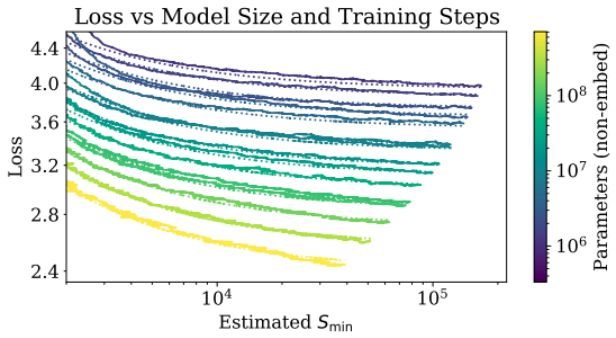


Figure 3: Test loss for different sized models after the initial transient period, plotted against the number of training steps (S_{min}) when using the critical batch sizes (the batch sizes that separate the two regimes in Question 3.1.2).

the model test performance (assume you will train from scratch). Which of the following is the best option? Give a brief explanation (2-3 sentences).

A. Train the same model with the same batch size for more steps.

B. Train the same model with a larger batch size (after tuning learning rate), for the same number of steps.

C. Increase the model size.

Programming Assignment

Image Colourization as Classification

In this section, we will perform image colourization using two convolutional neural networks (Figure 4a and b). Given a grayscale image, we wish to predict the color of each pixel. We have provided a subset of 24 output colours, selected using k-means clustering[1]. The colourization task will be framed as a pixel-wise classification problem, where we will label each pixel with one of the 24 colours. For simplicity, we measure distance in RGB space. This is not ideal but reduces the software dependencies for this assignment.

We will use the CIFAR-10 data set, which consists of images of size 32x32 pixels. For most of the questions we will use a subset of the dataset. The data loading script is included with the notebooks, and should download automatically the first time it is loaded.

Helper code for Sections 4 and 5, along with the main training loop and utilities for data manipulation, is provided at `https://colab.research.google.com/drive/1gTvOQ9nvIufcD8IPosd_7F6JDkrz1VBq?usp=sharing`. Make a copy of the Colab notebook to setup for this question and answer the following questions.

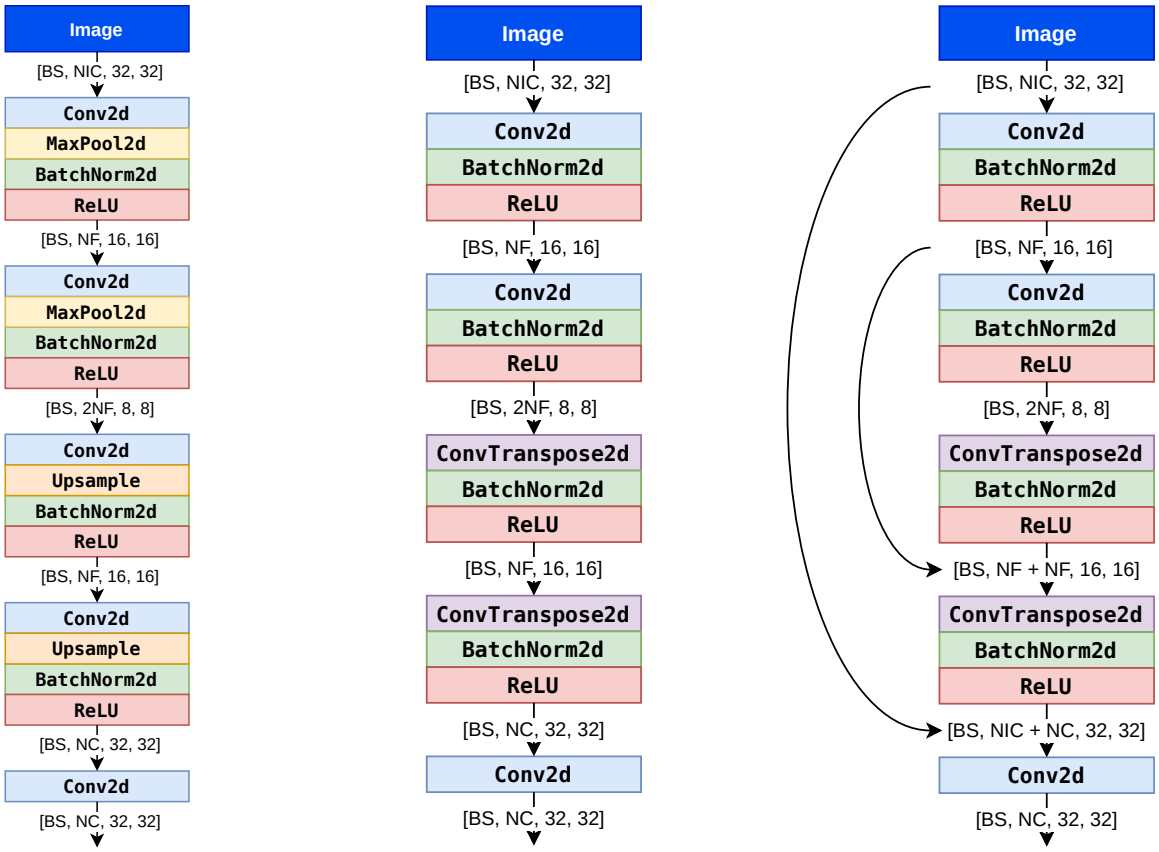
4 Pooling and Upsampling [20pt]

4.1 [8pt]

Complete the model `PoolUpsampleNet`, following the diagram in Figure 4a. Use the PyTorch layers `nn.Conv2d`, `nn.ReLU`, `nn.BatchNorm2d`, `nn.Upsample`[2], and `nn.MaxPool2d`. Your CNN should be configurable by parameters `kernel`, `num_in_channels`, `num_filters`, and `num_colours`. In the diagram, `num_in_channels`, `num_filters` and `num_colours` are denoted **NIC**, **NF** and **NC** respectively. Use the following parameterizations (if not specified, assume default parameters):

- `nn.Conv2d`: The number of input filters should match the second dimension of the *input* tensor (e.g. the first `nn.Conv2d` layer has **NIC** input filters). The number of output filters should match the second dimension of the *output* tensor (e.g. the first `nn.Conv2d` layer has **NF** output filters). Set kernel size to parameter `kernel`. Set padding to the `padding` variable included in the starter code.
- `nn.BatchNorm2d`: The number of features should match the second dimension of the output tensor (e.g. the first `nn.BatchNorm2d` layer has **NF** features).
- `nn.Upsample`: Use `scaling_factor = 2`.

[1] `https://en.wikipedia.org/wiki/K-means_clustering`
[2] `https://machinethink.net/blog/coreml-upsampling/`



(a) `PoolUpsampleNet` (b) `ConvTransposeNet` (c) `UNet`

Figure 4: Three network architectures that we will be using for image colourization. Numbers inside square brackets denote the shape of the tensor produced by each layer: **BS**: batch size, **NIC**: `num_in_channels`, **NF**: `num_filters`, **NC**: `num_colours`. For this assignment, we will skip the UNet architecture until future lectures.

- `nn.MaxPool2d`: Use `kernel_size = 2`.

Note: grouping layers according to the diagram (those not separated by white space) using the `nn.Sequential` containers will aid implementation of the `forward` method.

4.2 [4pt]

Run main training loop of `PoolUpsampleNet`. This will train the CNN for a few epochs using the cross-entropy objective. It will generate some images showing the trained result at the end. Do these results look good to you? Why or why not?

4.3 [8pt]

Compute the number of weights, outputs, and connections in the model, as a function of **NIC**, **NF** and **NC**. Compute these values when each input dimension (width/height) is doubled. Report all 6 values.

Note:

1. Please ignore biases when answering the questions.
2. Please ignore `nn.BatchNorm2d` when answering the number of weights, outputs and connections, but we still accept answers that do.

Hint:

1. `nn.Upsample` does not have parameters (this will help you answer the number of weights).
2. Think about when the input width and height are both doubled, how will the dimension of feature maps in each layer change? If you know this, you will know how dimension scaling will affect the number of outputs and connections.

5 Strided and Transposed Dilated Convolutions [16pt]

For this part, instead of using `nn.MaxPool2d` layers to reduce the dimensionality of the tensors, we will increase the step size of the preceding `nn.Conv2d` layers, and instead of using `nn.Upsample` layers to increase the dimensionality of the tensors, we will use *transposed* convolutions. Transposed convolutions aim to apply the same operations as convolutions but in the opposite direction. For example, while increasing the stride from 1 to 2 in a convolution forces the filters to skip over every other position as they slide across the input tensor, increasing the stride from 1 to 2 in a transposed convolution adds "empty" space around each element of the input tensor, as if reversing the skipping over every other position done by the convolution. We will be using a `dilation rate of 1` for the transposed convolution. Excellent visualizations of convolutions and transposed

convolutions have been developed by Dumoulin and Visin [2018] and can be found on their GitHub page[3].

5.1 [4pt]

Complete the model `ConvTransposeNet`, following the diagram in Figure 4b. Use the PyTorch layers `nn.Conv2d`, `nn.ReLU`, `nn.BatchNorm2d` and `nn.ConvTranspose2d`. As before, your CNN should be configurable by parameters `kernel`, `dilation`, `num_in_channels`, `num_filters`, and `num_colours`. Use the following parameterizations (if not specified, assume default parameters):

- `nn.Conv2d`: The number of input and output filters, and the kernel size, should be set in the same way as Section 4. For the first two `nn.Conv2d` layers, set `stride` to 2 and set `padding` to 1.
- `nn.BatchNorm2d`: The number of features should be specified in the same way as for Section 4.
- `nn.ConvTranspose2d`: The number of input filters should match the second dimension of the *input* tensor. The number of output filters should match the second dimension of the *output* tensor. Set `kernel_size` to parameter `kernel`. Set `stride` to 2, set `dilation` to 1, and set both `padding` and `output_padding` to 1.

5.2 [4pt]

Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. Plot the training curve, and include this plot in your write-up.

5.3 [4pt]

How do the results compare to Section 4? Does the `ConvTransposeNet` model result in lower validation loss than the `PoolUpsampleNet`? Why may this be the case?

5.4 [4pt]

How would the `padding` parameter passed to the first two `nn.Conv2d` layers, and the `padding` and `output_padding` parameters passed to the `nn.ConvTranspose2d` layers, need to be modified if we were to use a kernel size of 4 or 5 (assuming we want to maintain the shapes of all tensors shown in Figure 4b)? Note: PyTorch documentation for `nn.Conv2d`[4] and `nn.ConvTranspose2d`[5] includes equations that can be used to calculate the shape of the output tensors given the parameters.

[3] `https://github.com/vdumoulin/conv_arithmetic`
[4] `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`
[5] `https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html`

5.5 [0pt]

Re-train a few more `ConvTransposeNet` models using different batch sizes (e.g., 32, 64, 128, 256, 512) with a fixed number of epochs. Describe the effect of batch sizes on the training/validation loss, and the final image output quality. You do *not* need to attach the final output images.

References

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.