MIT CSAIL

6.s978 Deep Generative Models

Fall 2024

# Problem Set 2

**Posted:** Tuesday, Sep. 24, 2024 **Due:** Tuesday 11:59 pm, Oct. 08, 2024

We provide a Python notebook with the code to be completed. You can run it locally or on Google Colab. To use Colab, upload it to Google Drive and double-click the notebook (or right-click and select Open with Google Colaboratory), which will allow you to complete the problems without setting up your own environment.

**Submission Instructions:** Please submit two files on Canvas: Submit (1) Your report named `kerberos.pdf` which should include your answers to all required questions with images/plots showing your results as well as the code you wrote (e.g., using screenshots); (2) The provided Python notebook with the relevant source code and with all the cells run.

**Attention: If you fail to include your code in your report in your submission, we will not be able to give you credit for your code, so please do not forget.**

**Late Submission Policy:** If your Problem Set is submitted within 7 days of the original deadline, you will receive partial credit. Such submissions will be penalized by a multiplicative coefficient that linearly decreases from 1 to 0.5, step-wise on each day's 11:59pm cutoff.

In this problem set, you will be experimenting with an autoregressive generative model – PixelCNN using PyTorch. To reduce training time, we recommend using GPU acceleration. Colab comes with free GPU support. On Colab, select GPU as your runtime type as follows: **Runtime** → **Change runtime type** → **Hardware accelerator** → **GPU** → **Save**.

If you exceed your GPU usage limit on Colab, don't fret. You can also complete your problem set using a regular CPU in a reasonable amount of time.

**Problem 0** *Notebook Submission, .ipynb version* (1 point, required)

**Problem 1** *Negative Log-likelihood v.s. Cross Entropy* (20 points)

In this problem, you will prove that the Negative Log-Likelihood (NLL) is equivalent to the Cross-Entropy Loss for binary sequence autoregressive models.

Consider a binary sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, where each $x_i$ represents a binary variable, i.e., $x_i \in \{0, 1\}$. The goal of an autoregressive model is to predict the sequence by modeling the conditional probability of each element $x_i$ given all previous elements in the sequence $x_1, x_2, \ldots, x_{i-1}$. An autoregressive model predicts the probability of each binary variable $x_i$

as:

$$p(x_i \mid x_1, x_2, \ldots, x_{i-1}),$$

where $p(x_i \mid x_1, x_2, \ldots, x_{i-1})$ is the probability that the autoregressive model assigns to the event $x_i = 1$, and $1 - p(x_i \mid x_1, x_2, \ldots, x_{i-1})$ is the probability of the event $x_i = 0$.

The Negative Log-Likelihood for the entire sequence $\mathbf{x}$ is defined as:

$$\text{NLL}(\mathbf{x}) = -\sum_{i=1}^{T} \log p(x_i \mid x_1, x_2, \ldots, x_{i-1}).$$

Here, the NLL measures how well the model predicts the actual observed sequence $\mathbf{x}$. The lower the NLL, the better the model's predictions align with the true sequence. Note that the NLL is a common metric used in the earlier papers on autoregressive models [Van+16].

The Cross-Entropy Loss for the sequence $\mathbf{x}$ is defined as:

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{T} \left(y_i \log p(x_i \mid x_1, x_2, \ldots, x_{i-1}) + (1 - y_i) \log(1 - p(x_i \mid x_1, x_2, \ldots, x_{i-1}))\right),$$

where $y_i$ represents the true label for each $x_i$ (in this binary case, $y_i = x_i$).

> Prove that the NLL is equal to the Cross-Entropy Loss for binary sequence autoregressive models. Show all steps clearly.

In the following two problems, we will train PixelCNN on MNIST, exploring both unconditional and conditional generation.

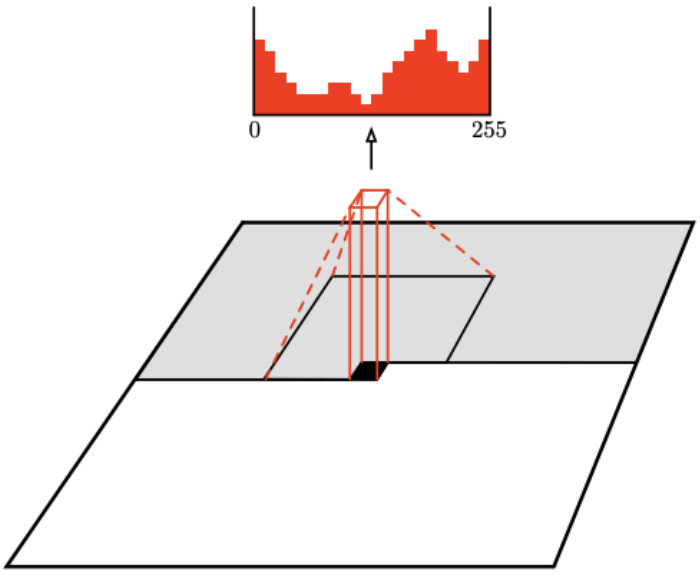


Figure 1: Illustration of PixelCNN.

**Problem 2** *Regular PixelCNN on MNIST* (30 points)

PixelCNNs are a type of autoregressive generative model that generate images by modeling the joint distribution of pixels as a product of conditional distributions. In PixelCNN, the pixels are generated one by one in a specific order, with each pixel conditioned on the previously generated pixels. The model uses masked convolutions to ensure that the prediction of each pixel only depends on the already generated pixels, following the raster scan order.

Figure 1 illustrates how PixelCNN maps a neighborhood of pixels to predict the next pixel. To generate pixel $x_i$, the model can only condition on the previously generated pixels $x_1, \ldots, x_{i-1}$. This conditioning is achieved by applying a mask to the convolutional filters, as shown in the right figure. The mask depicted is a type A mask, which differs from a type B mask where the weight for the central pixel is also set to 1.

(a) Implement the `MaskedConv2d()` function, which applies a mask to the convolutional filters in a 2D convolutional layer. The mask ensures that each pixel is only influenced by the pixels that have been generated before it. The function should allow for both type A and type B masks, where type A masks do not allow the central pixel to influence itself, and type B masks do.

(b) Using your `MaskedConv2d()` function, implement the PixelCNN architecture. Your implementation should include several masked convolutional layers, nonlinear activations, and any necessary normalization layers. Ensure that your model is capable of generating images pixel by pixel.

(c) Train your PixelCNN model on the MNIST dataset. We will use Cross-Entropy Loss as the training objective. As proven in Problem 1, this is equal to minimizing the NLL (i.e., maximizing likelihood) for binary sequence autoregressive models. After training, evaluate the model using the provided code for unconditional generation of MNIST digits. You may need to adjust hyperparameters such as the learning rate, number of epochs, and batch size to achieve good results.

**Problem 3** *Conditional PixelCNN on MNIST* (30 points)

Conditional PixelCNNs extend the basic PixelCNN model by conditioning the generation process on additional information, such as class labels. In this problem, you will implement and train a Conditional PixelCNN on the MNIST dataset, where the model is conditioned on the digit class labels.

(a) Extend your implementation of the `MaskedConv2d()` function to incorporate a conditional bias based on class labels. Specifically, the conditional bias should be added to the output of each convolutional layer as follows:

$$W_\ell * x + b_\ell + V_\ell y,$$

where $W_\ell * x + b_\ell$ represents the masked convolution, $V_\ell$ is a 2D weight matrix, and $y$ is the one-hot encoded class label. The conditional bias $V_\ell y$ should be broadcasted spatially and added to the output channel-wise.

(b) Using your conditional `MaskedConv2d()` function, implement the Conditional PixelCNN architecture. Your model should include several conditional masked convolutional layers, with the class label influencing each layer's output through

the conditional bias. The overall architecture should closely resemble the regular PixelCNN but with the additional conditioning on class labels.

(c) Train your Conditional PixelCNN on the MNIST dataset, using the class labels as conditioning information. After training, evaluate the model's performance on the test set by generating images conditioned on specific digit classes. Visualize several generated samples for each digit (0-9) and discuss the quality of the conditional generation compared to the unconditional generation in Problem 2.

**Problem 4** *Maximizing Likelihood with Regularization* (20 points)

In practice, Maximum Likelihood (ML) estimation (equivalent to minimizing NLL) is often enhanced by incorporating regularization techniques. Regularization biases the ML estimates towards solutions that reflect certain properties, often informed by domain knowledge. One common form of regularization is $\ell_2$ regularization, also known as Tikhonov regularization, which is widely used to prevent overfitting by penalizing large parameter values. The $\ell_2$ regularization is often applied in the code by using the `weight_decay` parameter in the optimizer, which directly corresponds to the $\lambda\|\theta\|_2^2$ term in the regularization, where $\theta$ is the neural network parameter.

It is known that ML estimation with a specific regularizer can be equivalent to Bayesian estimation with a corresponding prior distribution. In this problem, you will explore this equivalence by showing that ML estimation with $\ell_2$ regularization is equivalent to Bayesian estimation with a Gaussian prior.

Consider a parameterized model (e.g., an NN) $p_\theta(\mathbf{x};\theta)$ where $\theta$ is the network parameter and $\mathbf{x}$ is the training data. The *regularized maximum likelihood estimate* with a penalty $\rho(\cdot)$ is given by:

$$\hat{\theta}_{\mathrm{RML}}(\mathbf{x}) = \arg\max_{\theta} \left[\log p_\theta(\mathbf{x};\theta) - \lambda\rho(\theta)\right],$$

where $\lambda > 0$ is the regularization parameter.

Now, consider the corresponding Bayesian estimation scenario where the parameter $\theta$ is treated as a random variable. The Maximum A Posteriori (MAP) estimate is given by:

$$\hat{\theta}_{\mathrm{MAP}}(\mathbf{x}) = \arg\max_{\theta} \left[\log p(\theta \mid \mathbf{x})\right].$$

(a) Using Bayes' theorem, derive the MAP estimator for $\theta$ and show that it can be rewritten in a form similar to the regularized ML estimator. *Hint: The non-Bayesian case of $p_\theta(\mathbf{x};\theta)$ can be treated as equal to the Bayesian case of $p(\mathbf{x} \mid \theta)$, where the former $\theta$ is a parameter value and the latter $\theta$ is a random variable.*

(b) Demonstrate the equivalence between the regularized ML estimator $\hat{\theta}_{\mathrm{RML}}(\mathbf{x})$ with $\ell_2$ regularization and the MAP estimator $\hat{\theta}_{\mathrm{MAP}}(\mathbf{x})$ with a Gaussian prior. Specifically, show what the Gaussian prior (its mean and variance) should be in the Bayesian setting to make the two estimators equivalent.

## References

[Van+16] Aaron Van den Oord et al. "Conditional image generation with pixelcnn decoders". In: *Advances in neural information processing systems* 29 (2016).