CS294-158 Deep Unsupervised Learning
UC Berkeley, Spring 2019
HW2: Flows
Due: February 26, 11:59pm

1 Flows in 2D

In this problem, you will train flow models that map 2D data to a uniform base distribution. We'll use this notation:

- Data space: $\mathcal{X} = \mathbb{R}^2$
- Latent/noise space: $\mathcal{Z} = (0,1)^2$, the open unit square
- Data distribution: $p_{\text{data}}(x)$
- Base distribution: $p(z) = \text{Uniform}((0,1)^2)$
- Parameterized flow: $f_\theta : \mathcal{X} \to \mathcal{Z}$

Train these two flow models:

1. An autoregressive flow built out of univariate mixture CDFs. Specifically, for a hyperparameter k (number of mixture components), define a flow $f_\theta : \mathcal{X} \to \mathcal{Z}$ by:

$$(z_1, z_2) = f_\theta(x_1, x_2) = (f_\theta(x_1), f_\theta(x_2 \mid x_1)) \tag{1}$$

$$f_\theta(x_1) = \int_{-\infty}^{x_1} \sum_{i=1}^{k} \pi_{1,i}(\theta)\, \mathcal{N}(t; \mu_{1,i}(\theta), \sigma^2_{1,i}(\theta))\, dt \tag{2}$$

$$f_\theta(x_2 \mid x_1) = \int_{-\infty}^{x_2} \sum_{i=1}^{k} \pi_{2,i}(x_1;\theta)\, \mathcal{N}(t; \mu_{2,i}(x_1;\theta), \sigma^2_{2,i}(x_1;\theta))\, dt \tag{3}$$

Here, $\{\pi_{1,i}, \mu_{1,i}, \sigma^2_{1,i}, \pi_{2,i}, \mu_{2,i}, \sigma^2_{2,i}\}_{i=1}^k$ are parameterized functions. Feel free to set them up as a MADE or as separate networks. Each $\pi_{1,i}$ and $\pi_{2,i}$ should be a probability distribution over $i \in \{1, \ldots, k\}$.

2. A RealNVP-like model of the following form:

$$(z_1, z_2) = (\sigma \circ f_{\theta,1} \circ \cdots \circ f_{\theta,n})(x_1, x_2) \tag{4}$$

Here, $\sigma : \mathbb{R}^2 \to (0,1)^2$ is a flow defined by applying the sigmoid function elementwise. Use RealNVP coupling layers for the flows $f_{\theta,i}$: see the paper [1]. For each coupling layer, treat one coordinate of the data as the conditioned part and the other coordinate as the transformed part. Be sure to flip the conditioning pattern every layer. We strongly recommend you to use activation normalization flows [4] between each coupling layer.

Run the following code to generate data for training these flows. This code will generate a dataset of labeled samples: the first element of the returned tuple is the data in $\mathbb{R}^2$, and the second element of the returned tuple is the label in $\{0, 1, 2\}$. You will train the flows on unlabeled samples, and you will use the labels for visualization purposes only. Take the first 80% of the samples as a training set and the remaining 20% as a test set.

```
import numpy as np
def sample_data():
    count = 100000
    rand = np.random.RandomState(0)
    a = [[-1.5, 2.5]] + rand.randn(count // 3, 2) * 0.2
    b = [[1.5, 2.5]] + rand.randn(count // 3, 2) * 0.2
    c = np.c_[2 * np.cos(np.linspace(0, np.pi, count // 3)),
              -np.sin(np.linspace(0, np.pi, count // 3))]
    c += rand.randn(*c.shape) * 0.2
    data_x = np.concatenate([a, b, c], axis=0)
    data_y = np.array([0] * len(a) + [1] * len(b) + [2] * len(c))
    perm = rand.permutation(len(data_x))
    return data_x[perm], data_y[perm]
```

Train both models by maximum likelihood. Recall the log likelihood formula for flows:

$$\log p_\theta(x) = \log p(f_\theta(x)) + \log \left| \det \frac{\partial f_\theta(x)}{\partial x} \right| \tag{5}$$

You will have to implement this formula for the flows in a way that is trainable by your deep learning framework of choice. You may want to think about these questions (there is no need to answer these in your writeup; we provide these questions only to guide you towards implementing your models correctly):

- What is $\log p(f_\theta(x))$ for the uniform base distribution that we are using in this problem?
- What is the determinant of the Jacobian of the autoregressive flow in Equation 1? What about the elementwise sigmoid flow in Equation 4?
- How can you ensure that your calculation of the Jacobian determinants are correct?

Provide these deliverables for both models

1. Plot negative log likelihood over the course of training, for both training and validation data. What is your final test set performance? Report in bits per dimension.

2. Show the learned density in the region $(-4,4)^2 \subset \mathcal{X}$. To do so, evaluate and exponentiate the result of Equation 5 on a dense grid in this region, and show the result using `pcolormesh` or `imshow` in Matplotlib.

3. Display samples.

4. Display latents for data. To do so, take *labeled* training data (x, y) and plot $z = f_\theta(x)$, colored by the label y. Comment on the appearance of the latent spaces for the two models.

5. Optional: map a grid in $\mathcal{X}$ into a distorted grid in $\mathcal{Z}$. Display the result.

2 High-dimensional data

In this problem, you will train a flow model to invertibly map a high dimensional dataset of celebrity faces to gaussian noise. The dataset is a low-resolution (32×32) version of the CelebA-HQ dataset [3]. Further, the dataset has been quantized to 2 bits per color channel as in the previous homework. As a pre-requisite, understanding the architectures used in [1] and [4] is recommended. We used the affine coupling flow from [1] and a form of data-dependent initialization [5] that normalizes activations from an initial forward pass with a minibatch.

We describe a reference architecture below that can achieve good performance, but you are free and **encouraged** to use different architectures and/or structure your code differently (for example, the reference below keeps track of two set of dimensions explicitly instead of using masking but a masking based solution can work).

```
"""
Read section 4.1 of RealNVP paper https://arxiv.org/pdf/1605.08803.pdf
and adapt it to 2-bit quantized image.
It is important that you take this operation into account for reporting bits/dim
"""
Preprocess(),

CheckerboardSplit() # Figure 3 in Dinh et al - (left)
for _ in range(4):
    AffineCoupling(), TupleFlip()
Inverse(CheckerboardSplit())

Squeeze(), # [b, h, w, c] --> [b, h//2, w//2, c*4]

ChannelSplit() # x1, x2 = tf.split(x, 2, axis=-1) - NHWC
for _ in range(3):
    AffineCoupling(), TupleFlip()
Inverse(ChannelSplit()) # x = tf.concat([x1, x2], axis=-1) - NHWC

CheckberboardSplit()
for _ in range(3):
    AffineCoupling() , TupleFlip()
```

```
Inverse(CheckerboardSplit())

Squeeze()

ChannelSplit()
for _ in range(3):
    AffineCoupling(), TupleFlip()
Inverse(ChannelSplit())

CheckberboardSplit()
for _ in range(3):
    AffineCoupling(), TupleFlip()
Inverse(CheckerboardSplit())
```

The Affine Coupling flow is implemented with the following architecture:

```
AffineCoupling((x1, x2)):
    # forward pass
    y1 = x1
    log_s, t = split(simple_resnet(x1), 2, axis=-1)
    y2 = tf.exp(log_s)*(y1 + t)
    return (y1, y2)

    def simple_resnet(x, *, n_filters=256, n_blocks=8):
        n_in = x.shape[-1].value
        n_out = n_in*2
        h = conv2d(x, kernel=(3, 3), n_filters, stride=(1, 1))
        for _ in range(n_blocks):
            _h = conv2d(h, kernel=(1, 1), n_filters, stride=(1, 1))
            _h = relu(_h)
            _h = conv2d(_h, kernel=(3, 3), n_filters, stride=(1, 1))
            h = relu(_h)
            h = conv2d(_h, kernel=(1,1), n_filters, stride=(1, 1))
            h = (h + _h)
        h = relu(h)
        x = conv2d(h, kernel=(3, 3), n_out, stride=(1, 1))
        return x

TupleFlip((x1, x2)):
    return (x2, x1)
```

The data-dependent initialization is executed at every convolution operation so that its weight and bias matrices are initialized to ensure that its activations are approximately zero mean and unit standard deviation after one forward pass with a minibatch of size 128. You could also instead try out the ActNorm Flow from [4] (Section 3.1) and make sure to account for its scaling factor in the bits/dim calculation. A sanity check to know if you have data-dependent initialization implemented correctly is that your training log-likelihood should start from approximately 2.2 bits/dim (since the dataset contains 2 bits per color channel).

Our reference implementation with the above details gets to train bits/dim of 0.58 in 20000 updates with a batch size 64 in approximately 2 hours. You should start seeing reasonable samples by now. We are providing you a validation set to report your log-likelihood scores as in the previous assignment. We use an Adam Optimizer with a warmup over 200 steps till a learning rate of `5e-4`. We didn't decay the learning rate but it is a generally recommended practice while training generative models. Figure 1 shows the samples from our reference implementation at the end of one hour of training. Figure 2 shows the interpolations with this model for 6 pairs of randomly picked images from the dataset.

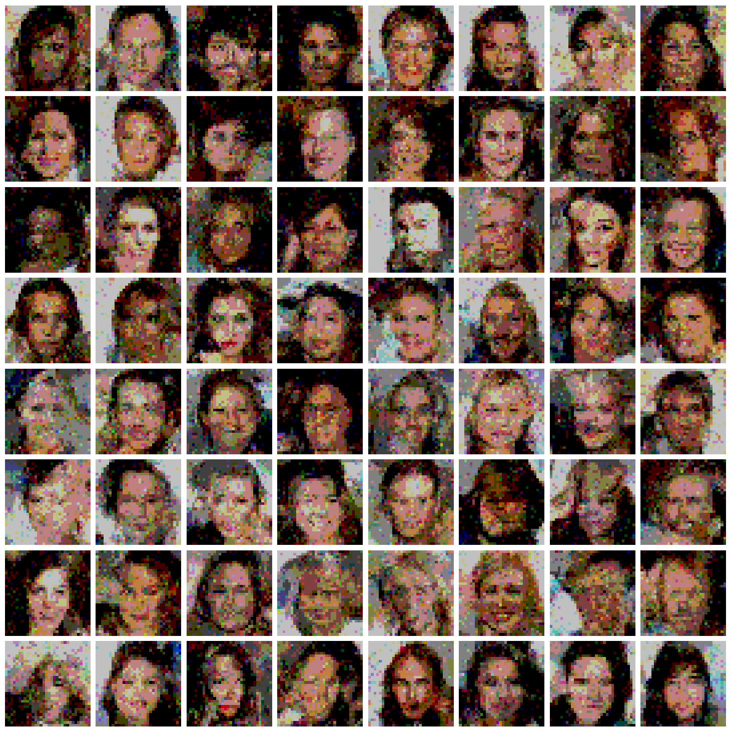

Figure 1: Samples from the flow model trained on 2-bit 32x32 CelebA

Provide these deliverables

1. Plot negative log likelihood over the course of training for both train and validation. What is your final validation set performance? Report in bits per dimension.

2. Propose a *bad* masking scheme (as opposed to checkerboard patterns) and show a comparison of samples and log-likelihood.

3. Latent space interpolations between 5 pairs of faces. You can pick whatever looks good to you.

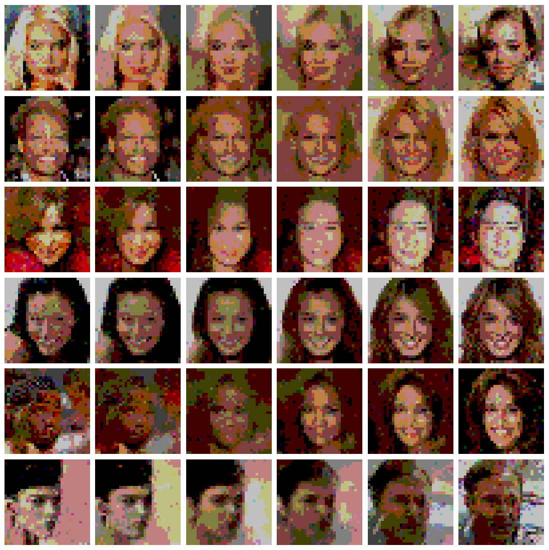

Figure 2: Interpolations from the flow model trained on 2-bit 32x32 CelebA

4. Display 100 samples.

3 Bonus Questions (Optional)

- Implement multi-scale RealNVP as described in Section 3.6 in [1] and examine if the latent space interpolations are better.
- Implement the 1x1 invertible convolution flow proposed by [4] and check if this allows you to remove `TupleFlip` and `CheckerboardSplit` in the flow architecture.
- Implement an autoregressive flow over color channels and check if this helps produce better samples.
- Attempt to implement a self-attentive architecture in the affine coupling flows. You may find [2] useful.

References

[1] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP". In: *International Conference on Learning Representations*. 2017.

[2] Jonathan Ho et al. "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design". In: *arXiv preprint arXiv:1902.00275* (2019).

[3] Tero Karras et al. "Progressive growing of gans for improved quality, stability, and variation". In: *arXiv preprint arXiv:1710.10196* (2017).

[4] Diederik P Kingma and Prafulla Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions". In: *Advances in Neural Information Processing Systems*. 2018.

[5] Tim Salimans and Durk P Kingma. “Weight normalization: A simple reparameterization to accelerate training of deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 901–909.