Mathematical Foundations of Deep Neural Networks, M1407.001200
E. Ryu
Fall 2022

Final Exam
Saturday, December 17, 2022, 1:00–5:00pm
4 hours, 7 questions, 100 points, 11 pages

This exam is open-book in the sense that you may use any non-electronic resource.
While we don't expect you will need more space than provided,
you may continue on the back of the pages.

Name: ______________________________

Do not turn to the next page until the start of the exam.

1. (15 points) *Layer normalization.* Remember that BatchNorm2D applied to 4-dimensional tensor $X \in \mathbb{R}^{B\times C\times P\times Q}$ is defined by

$$\hat{\mu}[:] = \frac{1}{BPQ}\sum_{b=1}^{B}\sum_{i=1}^{P}\sum_{j=1}^{Q} X[b,:,i,j]$$

$$\hat{\sigma}^2[:] = \frac{1}{BPQ}\sum_{b=1}^{B}\sum_{i=1}^{P}\sum_{j=1}^{Q} (X[b,:,i,j]-\hat{\mu}[:])^2$$

$$\mathrm{BN}_{\gamma,\beta}(X)[b,:,i,j] = \gamma[:]\frac{X[b,:,i,j]-\hat{\mu}[:]}{\sqrt{\hat{\sigma}^2[:]+\varepsilon}} + \beta[:],$$

while the running mean and variance is computed to later replace $\hat{\mu}$ and $\hat{\sigma}^2$ in test mode. BatchNorm2D can be implemented as follows:

```
class myBatchNorm(nn.Module):
  def __init__(self, num_features, momentum=0.9, epsilon=1e-05):
    super(MyBatchNorm, self).__init__()

    self.momentum = momentum
    self.insize = num_features
    self.epsilon = epsilon

    # init weight(gamma), bias(beta),running mean, var
    self.weight = nn.Parameter(torch.ones(self.insize))
    self.bias = nn.Parameter(torch.zeros(self.insize))
    self.run_mean = torch.zeros(self.insize)
    self.run_var = torch.ones(self.insize)

  def forward(self, input, mode):
    if mode == 'train':
      # mean over dims 0,2,3
      mean = input.mean(dim=(0, 2, 3)).view(1,-1,1,1)
      # var  over dims 0,2,3
      var = ((input - mean) ** 2).mean(dim=(0, 2, 3)).view(1,-1,1,1)

      run_mean_curr = self.momentum * self.run_mean
      self.run_mean = run_mean_curr + (1-self.momentum) * mean
      run_var_curr = self.momentum * self.run_var
      self.run_var = run_var_curr + (1-self.momentum) *var

      weight = self.weight.view(1, -1, 1, 1)
      bias = self.bias.view(1, -1, 1, 1)
      out = weight*(input-mean)/torch.sqrt(var+self.epsilon) + bias

     if mode == 'test':
      pass # in this problem, only consider train mode

    return out
```

In contrast, *layer normalization* is defined by

$$\hat{\mu}[:] = \frac{1}{CPQ}\sum_{c=1}^{C}\sum_{i=1}^{P}\sum_{j=1}^{Q} X[:,c,i,j]$$

$$\hat{\sigma}^2[:] = \frac{1}{CPQ}\sum_{c=1}^{C}\sum_{i=1}^{P}\sum_{j=1}^{Q} (X[:,c,i,j] - \hat{\mu}[:])^2$$

$$\mathrm{LN}_{\gamma,\beta}(X)[:,c,i,j] = \gamma[c,i,j]\frac{X[:,c,i,j] - \hat{\mu}[:]}{\sqrt{\hat{\sigma}^2[:] + \varepsilon}} + \beta[c,i,j].$$

Explain why layer norm does not need to distinguish train mode from test mode, and implement layer normalization.

2. (15 points) *Hierarchical Invertible Neural Transport (HINT) flow.* Let $n = 2^K$ for some $K \in \mathbb{N}$. Define the flow $f_\theta \colon \mathbb{R}^n \to \mathbb{R}^n$ recursively as follows. Let $f_\theta(x) = h_K(x)$ for $x \in \mathbb{R}^{2^K}$. For $k = K, K-1, \ldots, 1$, let

$$h_k(x) = \begin{bmatrix} h_{k-1}(x_{1:2^{k-1}}) \\ \hat{h}_{k-1}(x_{(2^{k-1}+1):2^k} \mid \psi_{k-1,\theta}(x_{1:2^{k-1}})) \end{bmatrix}$$

for $x \in \mathbb{R}^{2^k}$, where

$$\psi_{k-1,\theta}(x_{1:2^{k-1}}) = (s_{k-1,\theta}(x_{1:2^{k-1}}), t_{k-1,\theta}(x_{1:2^{k-1}}))$$
$$\hat{h}_{k-1}(x_{(2^{k-1}+1):2^k} \mid \psi_{k-1,\theta}(x_{1:2^{k-1}})) = e^{s_{k-1,\theta}(x_{1:2^{k-1}})} \odot x_{(2^{k-1}+1):2^k} + t_{k-1,\theta}(x_{1:2^{k-1}})$$

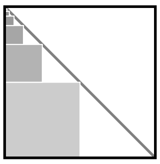
for $x \in \mathbb{R}^{2^k}$, where $\odot$ denotes the elementwise product. In other words, $\hat{h}_{k-1}$ is an affine coupling layer. Finally, $h_0(x)$ for $x \in \mathbb{R}$ is a 1D flow (and therefore is invertible). Assume we can evaluate $s_{0,\theta}, \ldots, s_{K-1,\theta}$, $t_{0,\theta}, \ldots, t_{K-1,\theta}$, h_0^{-1}, and h_0'.

(a) Describe an algorithm for computing $x = f_\theta^{-1}(z)$.

(b) Describe an algorithm for computing

$$\log \left| \frac{\partial f_\theta}{\partial x}(x) \right|.$$

Hint. The Jacobian matrix will have a lower-triangular structure:

$\frac{\partial f_\theta}{\partial x}(x) =$ 

3. (10 points) *Normalizing flow MLE minimizes KL-divergence.* Let p_{true} be a probability density function. We call

$$H(p_{\text{true}}) = -\int_{\mathbb{R}^d} p_{\text{true}}(x) \log p_{\text{true}}(x)\, dx$$

the (differential) *entropy* of p_{true}. Assume we have IID samples $X_1, \dots, X_N \sim p_{\text{true}}$. Consider the setup of training a flow model $f_\theta \colon \mathbb{R}^n \to \mathbb{R}^n$ with

$$\underset{\theta \in \mathbb{R}^p}{\text{maximize}} \quad \underbrace{\frac{1}{N} \sum_{i=1}^{N} \log p_Z(f_\theta(X_i)) + \log \left| \frac{\partial f_\theta}{\partial x}(X_i) \right|}_{\overset{\text{define}}{=} -\mathcal{L}(\theta)},$$

where p_Z is the density function of some prior distribution. Let p_θ be the density function $f_\theta^{-1}(Z)$, where $Z \sim p_Z$. Show that

$$\mathbb{E}[\mathcal{L}(\theta)] = D_{\text{KL}}(p_{\text{true}} \| p_\theta) + H(p_{\text{true}}).$$

4. (10 points) *VAE with Bernoulli likelihood.* For any $\mu \in [0,1]^n$, i.e., $(\mu)_i \in [0,1]$ for $i = 1, \dots, n$, let $\mathcal{B}(\mu)$ denote the distribution of an n independent Bernoulli random variables with means μ. In other words, if $X \sim \mathcal{B}(\mu)$, then

$$\mathbb{P}(X_i = 0) = 1 - \mu_i$$
$$\mathbb{P}(X_i = 1) = \mu_i$$

for $i = 1, \dots, n$ and $X_1, \dots, X_n$ are independent. Let our dataset $X^{(1)}, \dots, X^{(N)} \in \{0,1\}^n$ be "images" with each pixel value being 0 or 1. (As a concrete example, consider modifying the MNIST image to have pixel value 0 if the original pixel value is between 0 and 128 and 1 if between 128 and 255.) Consider the VAE setup with

$$\begin{aligned} p_Z &= \mathcal{N}(0, I) \\ q_\phi(z \mid x) &= \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x)) \text{ with diagonal } \Sigma_\phi \text{ with positive diagonals} \\ p_\theta(x \mid z) &= \mathcal{B}(f_\theta(z)), \end{aligned}$$

where $\mu_\phi \colon \{0,1\}^n \to \mathbb{R}^d$, $\Sigma_\phi \colon \{0,1\}^n \to \mathbb{R}^{d \times d}$, and $f_\theta \colon \mathbb{R}^d \to (0,1)^n$. (We implement f_θ as a deep neural network with the sigmoid activation function applied to the final output so that the outputs of f_θ are strictly between 0 and 1.) Describe the training objective that we can directly implement and backpropagate on in PyTorch.

Clarification. The training objective may not contain any expectations.

5. (10 points) *VAE prior scaling is unimportant.* Consider the VAE with training data $X^{(1)}, \ldots, X^{(N)} \in \mathbb{R}^n$ and

$$\begin{aligned} p_Z &= \mathcal{N}(0, \lambda^2 I) \quad \text{(note here)} \\ q_\phi(z \mid x) &= \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x)) \text{ with diagonal } \Sigma_\phi \text{ with positive diagonals} \\ p_\theta(x \mid z) &= \mathcal{N}(f_\theta(z), \sigma^2 I), \end{aligned}$$

where $\mu_\phi \colon \mathbb{R}^n \to \mathbb{R}^d$, $\Sigma_\phi \colon \mathbb{R}^n \to \mathbb{R}^{d \times d}$, $f_\theta \colon \mathbb{R}^d \to \mathbb{R}^n$, and $\sigma > 0$ is fixed.

(a) Show that the variational lower bound (VLB) changes as a function of $\lambda > 0$.

(b) Show that the VLB with $\lambda = 1$ is the same as the VLB with $\lambda > 0$, $\mu_\phi \mapsto \lambda \mu_\phi$, $\Sigma_\phi \mapsto \lambda^2 \Sigma_\phi$, and $f_\theta(z) \mapsto f_\theta(z/\lambda)$.

6. (20 points) *Geometric GAN.* For $r \in \mathbb{R}$, define

$$(r)_+ = \max\{0, r\} = \begin{cases} r & \text{if } r \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Consider a GAN with generator $G_\theta \colon \mathbb{R}^k \to \mathbb{R}^n$ and discriminator $D_\phi \colon \mathbb{R}^n \to \mathbb{R}$ trained with the *maximin* problem:

$$\underset{\theta \in \mathbb{R}^p}{\text{maximize}} \quad \underset{\phi \in \mathbb{R}^q}{\text{minimize}} \quad \mathbb{E}_{X \sim p_{\text{true}}}[(1 - D_\phi(X))_+] + \mathbb{E}_{Z \sim \mathcal{N}(0,I)}[(1 + D_\phi(G_\theta(Z)))_+],$$

where p_{true} is a density function.

(a) Let $a, b \in [0, \infty)$. Show that

$$h(y) = a(1 - y)_+ + b(1 + y)_+,$$

where $y \in (-\infty, \infty)$, is minimized at $y = -1$ or $y = +1$.

(b) Let p_θ be the density function of $G_\theta(Z)$ with $Z \sim \mathcal{N}(0, I)$. (Assume the density function p_θ exists for all $\theta \in \mathbb{R}^p$.) Assume that $D_\phi \colon \mathbb{R}^n \to \mathbb{R}$ is infinitely powerful, i.e., D_ϕ can represent any function from $\mathbb{R}^n$ to $\mathbb{R}$. Show that the minimax problem is equivalent to

$$\underset{\theta \in \mathbb{R}^p}{\text{maximize}} \quad \int \min\{p_{\text{true}}(x), p_\theta(x)\} \, dx. \tag{1}$$

(c) Further assume G_θ is infinitely powerful. Show that $p_\theta(x) = p_{\text{true}}(x)$ attains the maximum.

(d) For any probability density functions p and q, show that

$$\begin{aligned} D_{\text{TV}}(p, q) &\overset{\text{def}}{=} \frac{1}{2} \int_{\mathbb{R}^n} |p(x) - q(x)| \, dx \\ &= 1 - \int_{\mathbb{R}^n} \min\{p(x), q(x)\} \, dx. \end{aligned}$$

D_{TV} is acalled the *total variation distance* of p and q.

(e) Show that (1) is equivalent to

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad D_{\text{TV}}(p_{\text{true}}(x), p_\theta(x)).$$

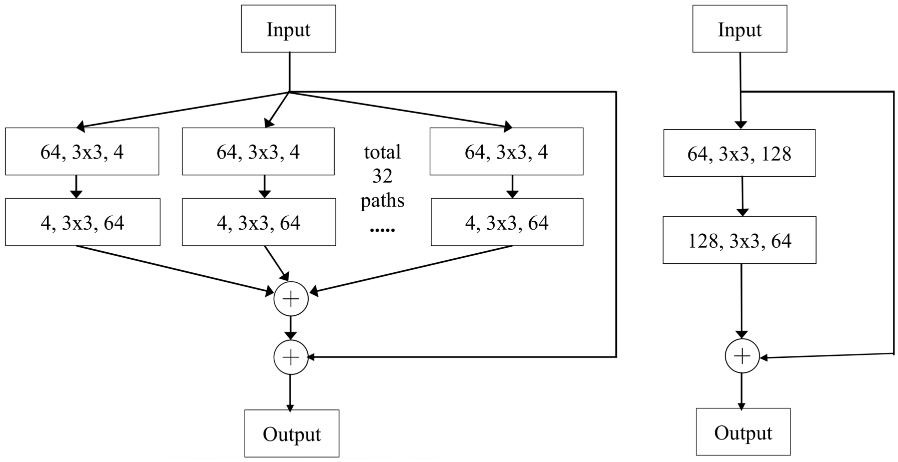
Hint. For (d), let $A = \{x \,|\, p(x) \leq q(z)\} \subseteq \mathbb{R}^n$ and $A^C = \{x \,|\, p(x) > q(z)\} \subseteq \mathbb{R}^n$ and use

$$\int_A p(x) \, dx = 1 - \int_{A^C} p(x) \, dx, \qquad \int_A q(x) \, dx = 1 - \int_{A^C} q(x) \, dx.$$

Remark. You can transform the maximin problem into a minimax problem by flipping the sign of the objective, i.e., the maximin problem is equivalent to

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \underset{\phi \in \mathbb{R}^q}{\text{maximize}} \quad -\mathbb{E}_{X \sim p_{\text{true}}}[(1 - D_\phi(X))_+] - \mathbb{E}_{Z \sim \mathcal{N}(0,I)}[(1 + D_\phi(G_\theta(Z)))_+].$$

7. (20 points) *2-layer ResNext block = 2-layer ResNet block.* Show that the following two blocks are equivalent.



Each box represents a convolutional layer with no bias followed the by ReLU activation function. The two blocks are formally defined as follows

```
class twoResNext(nn.Module):
  def __init__(self):
    super(twoResNext, self).__init__()
    self.layer1 = nn.ModuleList([
      nn.Conv2d(64, 4, kernel_size=3, padding=1, bias=False)
      for i in range(32)
    ])
    self.layer2 = nn.ModuleList([
      nn.Conv2d(4, 64, kernel_size=3, padding=1, bias=False)
      for i in range(32)
    ])

  def forward(self, x):
    out = 0
    for i in range(32):
      tmp = torch.nn.functional.relu(self.layer1[i](x))
      tmp = torch.nn.functional.relu(self.layer2[i](tmp))
      out += tmp
    return out + x

class twoResNet(nn.Module):
  def __init__(self):
    super(twoResNet, self).__init__()
    self.layer1
      = nn.Conv2d(64, 128, kernel_size=3, padding=1, bias=False)
    self.layer2
      = nn.Conv2d(128, 64, kernel_size=3, padding=1, bias=False)

  def forward(self, x):
    tmp = torch.nn.functional.relu(self.layer1(x))
    tmp = torch.nn.functional.relu(self.layer2(tmp))
    return tmp + x
```

Remark. This is why the “basic” ResNext blocks has depth 3.