Machine Learning - Assignment 2

Zhenrui Zheng

Chinese University of Hong Kong, Shenzhen
225040512@link.cuhk.edu.cn

Contents

Problem 1: Backpropagation for a MLP

Given a MLP defined as follows:

$$\begin{aligned}
&\boldsymbol{x} \in \mathbb{R}^D \\
&\boldsymbol{z} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}_1 \in \mathbb{R}^K \\
&\boldsymbol{h} = \mathrm{ReLU}(\boldsymbol{z}) \in \mathbb{R}^K \\
&\boldsymbol{a} = \mathbf{V}\boldsymbol{h} + \boldsymbol{b}_2 \in \mathbb{R}^C \\
&\mathcal{L} = \mathrm{CrossEntropy}(\boldsymbol{y}, \mathcal{S}(\boldsymbol{a})) \in \mathbb{R}
\end{aligned}$$

and the gradients of the loss with respect to logits a and hidden z:

$$\begin{aligned}
\boldsymbol{u}_2 &= \nabla_{\boldsymbol{a}}\mathcal{L} = (\boldsymbol{p} - \boldsymbol{y}) \in \mathbb{R}^C \\
\boldsymbol{u}_1 &= \nabla_{\boldsymbol{z}}\mathcal{L} = (\mathbf{V}^\top \boldsymbol{u}_2) \odot H(\boldsymbol{z}) \in \mathbb{R}^K
\end{aligned}$$

Then, we can derive the gradients for the parameters $\mathbf{V}, \boldsymbol{b}_2, \mathbf{W}, \boldsymbol{b}_1$ and the input $\boldsymbol{x}$ using the chain rule (Hereafter we always use the numerator layout):

$$\nabla_{\mathbf{V}}\mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{V}}\right]_{1,:} = \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}}\frac{\partial \boldsymbol{a}}{\partial \mathbf{V}}\right]_{1,:}$$

Where we know $\partial\mathcal{L}/\partial\boldsymbol{a} = \boldsymbol{u}_2^\top \in \mathbb{R}^{1\times C}$. For $\partial\boldsymbol{a}/\partial\mathbf{V}$, we can expand it according to definition:

$$\frac{\partial \boldsymbol{a}}{\partial \mathbf{V}} = \begin{pmatrix} \frac{\partial a_1}{\partial \mathbf{V}} \\ \vdots \\ \frac{\partial a_C}{\partial \mathbf{V}} \end{pmatrix} \in \mathbb{R}^{C\times(C\times K)}$$

Note that $a_c = \sum_{k=1}^{K} V_{ck}h_k + b_{2c}$, thus:

$$\frac{\partial a_c}{\partial V_{ij}} = \sum_{k=1}^{K} \frac{\partial}{\partial V_{ij}} V_{ck}h_k = \sum_{k=1}^{K} h_k \mathbb{I}(i = c, j = k)$$

Thus,

$$\frac{\partial \boldsymbol{a}}{\partial V_{ij}} = \begin{pmatrix} 0 & \cdots & 0 & h_j & 0 & \cdots & 0 \end{pmatrix}^\top \in \mathbb{R}^{C\times 1}$$

Where the non-zero element is at the i-th position. Thus,

$$\boldsymbol{u}_2^\top \frac{\partial \boldsymbol{a}}{\partial V_{ij}} = u_{2i}h_j$$

Thus,

$$\left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}}\frac{\partial \boldsymbol{a}}{\partial \mathbf{V}}\right]_{1,:} = \left[\boldsymbol{u}_2^\top \frac{\partial \boldsymbol{a}}{\partial \mathbf{V}}\right]_{1,:} = \boldsymbol{u}_2\boldsymbol{h}^\top \in \mathbb{R}^{C\times K}$$

We can do exactly the same thing for $\mathbf{W}$:

$$\nabla_{\mathbf{W}}\mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{W}}\right]_{1,:} = \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \mathbf{W}}\right]_{1,:} = \left[\boldsymbol{u}_1^\top \frac{\partial \boldsymbol{z}}{\partial \mathbf{W}}\right]_{1,:} = \boldsymbol{u}_1 \boldsymbol{x}^\top \in \mathbb{R}^{K\times D}$$

This is because $\boldsymbol{z} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}_1$ and $\boldsymbol{a} = \mathbf{V}\boldsymbol{h} + \boldsymbol{b}_2$ are identical in form.

For $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$, the derivation would be:

$$\nabla_{\boldsymbol{b}_2}\mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2}\right)^\top = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}}\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{b}_2}\right)^\top = \boldsymbol{u}_2 \in \mathbb{R}^C$$

Where $\partial\boldsymbol{a}/\partial\boldsymbol{b}_2 = \mathbb{I} \in \mathbb{R}^{C\times C}$. We can do exactly the same for $\boldsymbol{b}_1$ and get $\nabla_{\boldsymbol{b}_1}\mathcal{L} = \boldsymbol{u}_1 \in \mathbb{R}^C$.

For $\boldsymbol{x}$, the derivation would be:

$$\nabla_{\boldsymbol{x}}\mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\right)^\top = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)^\top = \left(\boldsymbol{u}_1^\top \mathbf{W}\right)^\top = \mathbf{W}^\top \boldsymbol{u}_1 \in \mathbb{R}^D$$

Where

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_1}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_K}{\partial x_1} & \cdots & \frac{\partial z_K}{\partial x_D} \end{pmatrix} = \mathbf{W}$$

Because

$$\frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j}(\sum_{k=1}^{K} W_{ik}x_k + b_{1i}) = W_{ij}$$

Problem 2: Gradient derivation for two-layer neural network

(a) Gradient of E wrt $\boldsymbol{z}_2$

Use the chain rule:

$$\frac{\partial E}{\partial \boldsymbol{z}_2} = \frac{\partial E}{\partial \hat{\boldsymbol{o}}} \frac{\partial \hat{\boldsymbol{o}}}{\partial \boldsymbol{z}_2}$$

Which can be expanded as:

$$\frac{\partial E}{\partial (\boldsymbol{z}_2)_j} = \sum_{i=1}^{K} \frac{\partial E}{\partial (\hat{\boldsymbol{o}}_i)} \frac{\partial (\hat{\boldsymbol{o}}_i)}{\partial (\boldsymbol{z}_2)_j}$$

Where the derivative of E wrt to $\hat{\boldsymbol{o}}_i$ is:

$$\frac{\partial E}{\partial (\hat{\boldsymbol{o}}_i)} = \frac{\partial}{\partial (\hat{\boldsymbol{o}}_i)} \left(- \sum_{k=1}^{K} \boldsymbol{o}_k \log(\hat{\boldsymbol{o}}_k) \right) = - \frac{\boldsymbol{o}_i}{\hat{\boldsymbol{o}}_i}$$

And the derivative of $\hat{\boldsymbol{o}}_i$ wrt to $(\boldsymbol{z}_2)_j$ is:

$$\frac{\partial (\hat{\boldsymbol{o}}_i)}{\partial (\boldsymbol{z}_2)_j} = \frac{\partial}{\partial (\boldsymbol{z}_2)_j} \left(\text{softmax}(\boldsymbol{z}_2)\right)_i = \frac{\partial}{\partial (\boldsymbol{z}_2)_j} \left(\frac{\exp((\boldsymbol{z}_2)_i)}{\sum_{k=1}^{K} \exp((\boldsymbol{z}_2)_k)} \right) = \hat{\boldsymbol{o}}_i (\mathbb{I}(i = j) - \hat{\boldsymbol{o}}_j)$$

Thus,

$$\begin{aligned}
\frac{\partial E}{\partial (\boldsymbol{z}_2)_j} &= \sum_{i=1}^{K} - \frac{\boldsymbol{o}_i}{\hat{\boldsymbol{o}}_i} \hat{\boldsymbol{o}}_i (\mathbb{I}(i = j) - \hat{\boldsymbol{o}}_j) \\
&= -\boldsymbol{o}_j (1 - \hat{\boldsymbol{o}}_j) + \sum_{i \neq j} \boldsymbol{o}_i \hat{\boldsymbol{o}}_j \\
&= -\boldsymbol{o}_j + \hat{\boldsymbol{o}}_j \sum_{i=1}^{K} \boldsymbol{o}_i \\
&= \hat{\boldsymbol{o}}_j - \boldsymbol{o}_j \quad (\text{Since} \sum_{i} \boldsymbol{o}_i = 1)
\end{aligned}$$

Thus, the vector form is:

$$\frac{\partial E}{\partial \boldsymbol{z}_2} = \begin{pmatrix} \frac{\partial E}{\partial (\boldsymbol{z}_2)_1} \\ \vdots \\ \frac{\partial E}{\partial (\boldsymbol{z}_2)_K} \end{pmatrix} = \begin{pmatrix} \hat{\boldsymbol{o}}_1 - \boldsymbol{o}_1 \\ \vdots \\ \hat{\boldsymbol{o}}_K - \boldsymbol{o}_K \end{pmatrix} = \hat{\boldsymbol{o}} - \boldsymbol{o} \qquad \dashleftarrow\!\!\dashleftarrow\!\!\dashleftarrow$$

(b) Gradient of E wrt $\mathbf{W}_2$ and $\boldsymbol{b}_2$

Similarly, we can use the chain rule:

$$\frac{\partial E}{\partial \mathbf{W}_2} = \frac{\partial E}{\partial \boldsymbol{z}_2} \frac{\partial \boldsymbol{z}_2}{\partial \mathbf{W}_2}$$

Which can be expanded as:

$$\frac{\partial E}{\partial (\mathbf{W}_2)_{ij}} = \sum_{k=1}^{K} \frac{\partial E}{\partial (\boldsymbol{z}_2)_k} \frac{\partial (\boldsymbol{z}_2)_k}{\partial (\mathbf{W}_2)_{ij}}$$

Where the derivative of $\boldsymbol{z}_2$ wrt to $(\mathbf{W}_2)_{ij}$ is:

$$\frac{\partial (\boldsymbol{z}_2)_k}{\partial (\mathbf{W}_2)_{ij}} = \frac{\partial}{\partial (\mathbf{W}_2)_{ij}} \left(\sum_{l=1}^{d} (\mathbf{W}_2)_{kl} \boldsymbol{h}_l + \boldsymbol{b}_{\boldsymbol{2}k} \right) = \boldsymbol{h}_j \mathbb{I}(k = i, l = j)$$

Thus,

$$\begin{aligned}
\frac{\partial E}{\partial (\mathbf{W}_2)_{ij}} &= \sum_{k=1}^{K} (\hat{\boldsymbol{o}}_k - \boldsymbol{o}_k) \boldsymbol{h}_j \mathbb{I}(k = i, l = j) \\
&= (\hat{\boldsymbol{o}}_i - \boldsymbol{o}_i) \boldsymbol{h}_j \\
\frac{\partial E}{\partial \mathbf{W}_2} &= (\hat{\boldsymbol{o}} - \boldsymbol{o}) \boldsymbol{h}^\top \qquad \dashleftarrow \cdots\cdots\cdots
\end{aligned}$$

For $\boldsymbol{b}_2$, the derivation would be:

$$\frac{\partial E}{\partial \boldsymbol{b}_2} = \frac{\partial E}{\partial \boldsymbol{z}_2} \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{b}_2}$$

Which can be expanded as:

$$\frac{\partial E}{\partial (\boldsymbol{b}_2)_j} = \sum_{k=1}^{K} \frac{\partial E}{\partial (\boldsymbol{z}_2)_k} \frac{\partial (\boldsymbol{z}_2)_k}{\partial (\boldsymbol{b}_2)_j}$$

Where the derivative of $\boldsymbol{z}_2$ wrt to $(\boldsymbol{b}_2)_j$ is:

$$\frac{\partial (\boldsymbol{z}_2)_k}{\partial (\boldsymbol{b}_2)_j} = \frac{\partial}{\partial (\boldsymbol{b}_2)_j} \left(\sum_{l=1}^{d} (\mathbf{W}_2)_{kl} \boldsymbol{h}_l + \boldsymbol{b}_{\boldsymbol{2}k} \right) = \mathbb{I}(k = j)$$

Thus,

$$\begin{aligned}
\frac{\partial E}{\partial (\boldsymbol{b}_2)_j} &= \sum_{k=1}^{K} (\hat{\boldsymbol{o}}_k - \boldsymbol{o}_k) \mathbb{I}(k = j) \\
&= (\hat{\boldsymbol{o}}_j - \boldsymbol{o}_j) \\
\frac{\partial E}{\partial \boldsymbol{b}_2} &= (\hat{\boldsymbol{o}} - \boldsymbol{o}) \qquad \dashleftarrow \cdots\cdots\cdots
\end{aligned}$$

(c) Gradient of E wrt $\boldsymbol{h}$ and $\boldsymbol{z}_1$

Similarly:

$$\frac{\partial E}{\partial \boldsymbol{h}} = \frac{\partial E}{\partial \boldsymbol{z}_2} \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{h}}$$

Which can be expanded as:

$$\frac{\partial E}{\partial (\boldsymbol{h})_j} = \sum_{k=1}^{K} \frac{\partial E}{\partial (\boldsymbol{z}_2)_k} \frac{\partial (\boldsymbol{z}_2)_k}{\partial (\boldsymbol{h})_j}$$

Where the derivative of $\boldsymbol{z}_2$ wrt to $(\boldsymbol{h})_j$ is:

$$\frac{\partial(\boldsymbol{z}_2)_k}{\partial(\boldsymbol{h})_j} = \frac{\partial}{\partial(\boldsymbol{h})_j}\left(\sum_{l=1}^{d}(\mathbf{W}_2)_{kl}\boldsymbol{h}_l + \boldsymbol{b}_{\mathbf{2}k}\right) = (\mathbf{W}_2)_{kj}$$

Thus,

$$\begin{aligned}\frac{\partial E}{\partial(\boldsymbol{h})_j} &= \sum_{k=1}^{K}(\hat{\boldsymbol{o}}_k - \boldsymbol{o}_k)(\mathbf{W}_2)_{kj} \\ \frac{\partial E}{\partial \boldsymbol{h}} &= \mathbf{W}_2^\top(\hat{\boldsymbol{o}} - \boldsymbol{o}) \qquad \dashleftarrow\end{aligned}$$

For $\boldsymbol{z}_1$, the derivation would be:

$$\frac{\partial E}{\partial \boldsymbol{z}_1} = \frac{\partial E}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}_1} = \frac{\partial E}{\partial \boldsymbol{h}} \odot \mathrm{ReLU}'(\boldsymbol{z}_1) = \mathbf{W}_1^\top(\hat{\boldsymbol{o}} - \boldsymbol{o}) \odot \mathbb{I}(\boldsymbol{z}_1 > 0) \qquad \dashleftarrow$$

(d) Gradient of E wrt $\boldsymbol{W}_1$ and $\boldsymbol{b}_1$

Similarly:

$$\frac{\partial E}{\partial \mathbf{W}_1} = \frac{\partial E}{\partial \boldsymbol{z}_1}\frac{\partial \boldsymbol{z}_1}{\partial \mathbf{W}_1}$$

Which can be expanded as:

$$\frac{\partial E}{\partial(\boldsymbol{W}_1)_{ij}} = \sum_{k=1}^{d}\frac{\partial E}{\partial(\boldsymbol{z}_1)_k}\frac{\partial(\boldsymbol{z}_1)_k}{\partial(\boldsymbol{W}_1)_{ij}}$$

Where the derivative of $\boldsymbol{z}_1$ wrt to $(\boldsymbol{W}_1)_{ij}$ is:

$$\frac{\partial(\boldsymbol{z}_1)_k}{\partial(\boldsymbol{W}_1)_{ij}} = \frac{\partial}{\partial(\boldsymbol{W}_1)_{ij}}\left(\sum_{l=1}^{n}(\mathbf{W}_1)_{kl}\boldsymbol{x}_l + \boldsymbol{b}_{\mathbf{1}k}\right) = \boldsymbol{x}_j\mathbb{I}(k = i, l = j)$$

Similar to $\partial E/\partial \mathbf{W}_2$, This results in an outer product of the upstream gradient $\frac{\partial E}{\partial \boldsymbol{z}_1}$ and the input to the layer, $\boldsymbol{x}$:

$$\frac{\partial E}{\partial \mathbf{W}_1} = \left[\mathbf{W}_2^\top(\hat{\boldsymbol{o}} - \boldsymbol{o}) \odot \mathbb{I}(\boldsymbol{z}_1 > 0)\right]\boldsymbol{x}^\top \qquad \dashleftarrow$$

For $\boldsymbol{b}_1$, the derivation would be:

$$\frac{\partial E}{\partial \boldsymbol{b}_1} = \frac{\partial E}{\partial \boldsymbol{z}_1}\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{b}_1} = \frac{\partial E}{\partial \boldsymbol{z}_1}\mathbf{I} = \mathbf{W}_1^\top(\hat{\boldsymbol{o}} - \boldsymbol{o}) \odot \mathbb{I}(\boldsymbol{z}_1 > 0) \qquad \dashleftarrow$$

Problem 3: Convolutional Neural Network

(a) Activation Volume Dimensions and Number of Parameters

The formula for the output dimension of a convolutional or pooling layer is:

$$O = \lfloor \frac{W_{in} - F + 2P}{S} \rfloor + 1$$

where W_{in} is the input dimension, F is the filter/pool size, P is padding, and S is stride. The number of parameters for each layer type is:

- Convolutional Layer (CONV): $(F \times F \times D_{in} + 1) \times D_{out}$
- Pooling Layer (POOL): 0 parameters
- Fully-Connected Layer (FC): $(N_{in} + 1) \times N_{out}$, where N_{in} is the total number of input neurons after flattening

Layer	Activation Volume Dimensions	Number of Parameters
INPUT	$32 \times 32 \times 1$	0
CONV5-10	$32 \times 32 \times 10$	$(5 \times 5 \times 1 + 1) \times 10 = 260$
POOL2	$16 \times 16 \times 10$	0
CONV5-10	$16 \times 16 \times 10$	$(5 \times 5 \times 10 + 1) \times 10 = 2,510$
POOL2	$8 \times 8 \times 10$	0
FC-10	10	$(8 \times 8 \times 10 + 1) \times 10 = 6,410$

(b) Reason for using a 1x1 Convolution

Less rigorously, we say convolutional neural networks process input in two ways:

- **spatial operations**, which capture spatial patterns between adjacent pixels, such as edges or textures, through receptive fields;
- **channel operations**, used to identify information such as colors in RGB space.

In the original convolutional neural network architecture, these two were simultaneously processed by the conv layer; however, in “depthwise separable convolution”, they are decoupled and such architectures have been proven to be faster and more efficient.

From this perspective, a 1×1 convolution is equivalent to independently processing channel information without changing spatial information, and is often used in conjunction with its complement, i.e., a “receptive field” network that only processes spatial information and does not involve channel information. Intuitively, if two components of a statistical distribution are (to some extent) independent, it will be easier to handle them separately, and the use of 1×1 convolution is consistent with such intuition.

As a result, for a layer with D_{out} output dimensions, the number of parameters required for “depthwise separable convolution” is only about $1/D_{\text{out}}$ of the original, without damaging (or even improving) the performance.

Problem 4: CNNs and Vision Transformers

1. **What are the two main operations used in CNNs to process images?**:
   B. Convolution and pooling.

2. **Why do Vision Transformers (ViTs) need positional encodings?**:
   B. To remember where each image patch came from.

3. **Which architecture typically works better with small datasets?**:
   B. CNN.
   CNN possess strong inductive biases, such as local receptive fields and weight sharing. These biases enable CNN to effectively learn meaningful locality features with a relatively smaller number of parameters. This generally leads to better performance for CNN on small datasets compared to ViT, while the later typically require large amounts of data to learn effective representations and reduce overfitting induced by global attention mechanism. (As an evidence, some follow-up work attempts to add regularization to pixel distance and claims performance improvements.)

4. **Patch Processing: An image of size 224×224 is divided into 16×16 patches for a Vision Transformer. (1) How many patches will there be? (2) If each patch is flattened into a vector, what will be its length for an RGB image?**:
   (1) There will be $(224//16) \times (224//16) = 14 \times 14 = 196$ patches.
   (2) The length of each patch will be $16 \times 16 \times 3 = 768$ for an RGB image.

5. **For each feature below, indicate whether it applies to**:
   (1) Processes image patches in sequential order: (V) or (N)
   Although ViT divides pixels into patches and processes them in a "sequential fashion", it is less appropriate to call it "sequential order" because all patches are simultaneously fed into the attention layer for computation, and there is no logical order between patches. Most ViT are discriminative models, so they are non-causal during training (no causal mask), but some generative ViT are indeed trained causally and are "sequential" during generation.
   (2) Can handle patches in parallel fashion: (B)
   (3) Uses weight sharing across spatial locations: (C) or (B)
   There's no doubt about CNN's spatial weight sharing, but for ViT, this concept is more ambiguous. The weight matrices ($\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$) for different patches are shared, while the patches are distributed across different spatial locations. Furthermore, within each patch, the parameters of the linear layers are also shared. Although for pixels at certain specific positions in the input, the weights they "see" may not be the same (e.g., within the same patch, the parameters for two different positions are different), and most positional encodings introduce some spatial bias. But at least during training, this heterogeneity can be reduced through spatial augmentation operations or by using true "relative" positional encodings.
   (4) Requires positional encoding of patches: (V)
   (5) Naturally preserves local spatial relationships: (C)
   (6) Naturally preserves global spatial relationships: (V)

Problem 5: Using Backpropagation for GANs

(1) Reason for objective $L(\theta_d, \theta_g)$

The objective function for the discriminator is given by:

$$L(\theta_d, \theta_g) = \frac{1}{n} \sum_{i=1}^{n} \log D(X^i) + \log\big(1 - D(G(Z^i))\big)$$

Given the discriminator acts as a binary classifier. $D(X^i)$ is the probability of the sample X^i being real. For a real sample X^i, the discriminator wants to maximize $D(X^i)$, which maximizes $\log D(X^i)$. For a fake sample $G(Z^i)$, the discriminator wants to classify it as fake, meaning minimizing $D(G(Z^i))$, which in turn maximizes $\log(1 - D(G(Z^i)))$.

Since the objective function L is a sum of these terms, maximizing L corresponds to correctly classifying both real and fake samples, which is the discriminator's goal. This is analogous to the cross-entropy loss used in binary classification, where minimizing the negative log-likelihood is equivalent to maximizing the likelihood.

(2) $\nabla_{\theta_d} L(\boldsymbol{X}; \theta_d, \theta_g)$ in terms of $\nabla_{\theta_d} D(.)$

We need to compute $\nabla_{\theta_d} L(\theta_d, \theta_g)$. Let $D_i^{(real)} = D(X^i; \theta_d)$ and $D_i^{(fake)} = D(G(Z^i); \theta_d)$. The objective function is

$$L(\theta_d, \theta_g) = \frac{1}{n} \sum_{i=1}^{n} \left[\log D_i^{(real)} + \log(1 - D_i^{(fake)})\right]$$

Taking the gradient with respect to θ_d:

$$\begin{aligned}
\nabla_{\theta_d} L(\theta_d, \theta_g) &= \frac{1}{n} \sum_{i=1}^{n} \left[\nabla_{\theta_d} \log D_i^{(real)} + \nabla_{\theta_d} \log(1 - D_i^{(fake)})\right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \left[\frac{1}{D_i^{(real)}} \nabla_{\theta_d} D_i^{(real)} + \frac{1}{1 - D_i^{(fake)}} \left(-\nabla_{\theta_d} D_i^{(fake)}\right)\right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \left[\frac{\nabla_{\theta_d} D(X^i; \theta_d)}{D(X^i; \theta_d)} - \frac{\nabla_{\theta_d} D(G(Z^i); \theta_d)}{1 - D(G(Z^i); \theta_d)}\right]
\end{aligned}$$

(3) $\partial L(\theta_d, \theta_g) / \partial z_d^{L_d}$

Let $A_d^{L_d} = D(\cdot; \theta_d)$ be the output of the discriminator. Given that the last layer activation is a sigmoid function, $A_d^{L_d} = \sigma(z_d^{L_d})$. We need to compute $\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^{L_d}}$. Let's consider a single sample X^i and $G(Z^i)$.

$$\begin{aligned}\frac{\partial L}{\partial z_d^{L_d}} &= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{\partial \log D(X^i;\theta_d)}{\partial z_d^{L_d}} + \frac{\partial \log(1-D(G(Z^i);\theta_d))}{\partial z_d^{L_d}}\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{D(X^i;\theta_d)}\frac{\partial D(X^i;\theta_d)}{\partial z_d^{L_d}(X^i)} + \frac{1}{1-D(G(Z^i);\theta_d)}\left(-\frac{\partial D(G(Z^i);\theta_d)}{\partial z_d^{L_d}(G(Z^i))}\right)\right]\end{aligned}$$

Since $D(\cdot;\theta_d) = \sigma(z_d^{L_d})$, we have $\frac{\partial D}{\partial z_d^{L_d}} = \frac{\partial \sigma(z_d^{L_d})}{\partial z_d^{L_d}} = \sigma(z_d^{L_d})(1-\sigma(z_d^{L_d})) = D(1-D)$. Substituting this into the expression, for each sample:

$$\begin{aligned}\left.\frac{\partial L}{\partial z_d^{L_d}}\right|_{X^i} &= \frac{1}{D(X^i;\theta_d)}D(X^i;\theta_d)(1-D(X^i;\theta_d)) = 1-D(X^i;\theta_d) \\ \left.\frac{\partial L}{\partial z_d^{L_d}}\right|_{Z^i} &= \frac{-1}{1-D(G(Z^i);\theta_d)}D(G(Z^i);\theta_d)(1-D(G(Z^i);\theta_d)) = -D(G(Z^i);\theta_d)\end{aligned}$$

Combining these:

$$\frac{\partial L(\theta_d,\theta_g)}{\partial z_d^{L_d}} = \frac{1}{n}\sum_{i=1}^{n}\left[(1-D(X^i;\theta_d)) - D(G(Z^i);\theta_d)\right]$$

This formula treats the gradient with respect to $z_d^{L_d}$ abstractly, summing over contributions from both real and fake data.

(4) Recursive breakdown of $\partial L(\theta_d,\theta_g)/\partial z_d^{L_d}$

For a given layer $l < L_d$, the pre-activation $z_d^{l+1} = \mathbf{W}_d^{l+1}A_d^l$, and activation $A_d^l = g_d^l(z_d^l)$. We want to express $\frac{\partial L(\theta_d,\theta_g)}{\partial z_d^l}$ in terms of $\frac{\partial L(\theta_d,\theta_g)}{\partial z_d^{l+1}}$. Using the chain rule:

$$\frac{\partial L}{\partial z_d^l} = \frac{\partial L}{\partial z_d^{l+1}}\frac{\partial z_d^{l+1}}{\partial z_d^l}$$

Firstly:

$$\begin{aligned}\frac{\partial z_d^{l+1}}{\partial z_d^l} &= \frac{\partial(\mathbf{W}_d^{l+1}A_d^l)}{\partial z_d^l} \\ &= \mathbf{W}_d^{l+1}\frac{\partial A_d^l}{\partial z_d^l}\end{aligned}$$

Since $A_d^l = g_d^l(z_d^l)$, the derivative $\frac{\partial A_d^l}{\partial z_d^l}$ is a diagonal matrix where the diagonal entries are the derivatives of the activation function for each element of z_d^l. Let $(g_d^l)'(z_d^l)$ denote this diagonal matrix. Thus, we have $\frac{\partial z_d^{l+1}}{\partial z_d^l} = \mathbf{W}_d^{l+1}(g_d^l)'(z_d^l)$. Substituting this back:

$$\frac{\partial L(\theta_d,\theta_g)}{\partial z_d^l} = \frac{\partial L(\theta_d,\theta_g)}{\partial z_d^{l+1}}\mathbf{W}_d^{l+1}(g_d^l)'(z_d^l)$$

Where we insist with the numerator layout, and thus gradients are row vectors.

(5) $\partial L(\theta_d, \theta_g)/\partial g(z_i; \theta_g)$ in terms of w_d^1 and $\partial L(\theta_d, \theta_g)/\partial z_d^1$

Let $G_i = G(Z^i; \theta_g)$ be the output of the generator for input noise Z^i. We want to compute $\frac{\partial L(\theta_d,\theta_g)}{\partial G_i}$ for a specific generated sample G_i. The pre-activation of the first layer of the discriminator for a given input G_i is $z_d^1(G_i) = \mathbf{W}_d^1 G_i$. Using the chain rule, we have:

$$\begin{aligned}
\frac{\partial L(\theta_d, \theta_g)}{\partial G_i} &= \frac{\partial L(\theta_d, \theta_g)}{\partial z_d^1(G_i)} \frac{\partial z_d^1(G_i)}{\partial G_i} \\
&= \frac{\partial L(\theta_d, \theta_g)}{\partial z_d^1(G_i)} \frac{\partial (\mathbf{W}_d^1 G_i)}{\partial G_i} \\
&= \frac{\partial L(\theta_d, \theta_g)}{\partial z_d^1(G_i)} \mathbf{W}_d^1 \frac{\partial G_i}{\partial G_i} \\
&= \frac{\partial L(\theta_d, \theta_g)}{\partial z_d^1(G(Z^i; \theta_g))} \mathbf{W}_d^1 \quad (\frac{\partial G_i}{\partial G_i} = \mathbf{I})
\end{aligned}$$

(6) $\nabla_{\theta_g} L(\theta_d, \theta_g)$ in terms of $\nabla_{\theta_g} G(Z^i; \theta_g)$

The objective function only depends on θ_g via $G(Z^i; \theta_g)$.

$$\begin{aligned}
\nabla_{\theta_g} L(\theta_d, \theta_g) &= \nabla_{\theta_g} \frac{1}{n} \sum_{i=1}^{n} \left[\log D(X^i; \theta_d) + \log(1 - D(G(Z^i; \theta_g); \theta_d))\right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta_g} \left[\log(1 - D(G(Z^i; \theta_g); \theta_d))\right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial L(\theta_d, \theta_g)}{\partial G(Z^i; \theta_g)} \nabla_{\theta_g} G(Z^i; \theta_g) \\
&= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial L(\theta_d, \theta_g)}{\partial z_d^1(G(Z^i; \theta_g))} \mathbf{W}_d^1 \nabla_{\theta_g} G(Z^i; \theta_g)
\end{aligned}$$

(7) mini-batch gradient descent update for θ_d

The discriminator is optimized to maximize the objective function. So its update rule is for gradient ascent.

$$\theta_d^{t+1} = \theta_d^t + \alpha \nabla_{\theta_d} L(\theta_d^t, \theta_g^t)$$

(8) mini-batch gradient descent update for θ_g

The generator is optimized to minimize the objective function. So its update rule is for gradient descent.

$$\theta_g^{t+1} = \theta_g^t - \alpha \nabla_{\theta_g} L(\theta_d^t, \theta_g^t)$$

Programming I: CNN Implementation

For the CIFAR-10 challenge, a convolutional neural network that achieved over 81% accuracy on the test set is implemented. The model architecture incorporates several modern deep learning techniques to improve performance.

Architecture Design

The advanced model consists of four main components:

1. **Deep Convolutional Layers**: Four convolutional blocks with increasing channel depth (64→128→256→512 channels)
2. **Batch Normalization**: Applied after each convolutional layer to stabilize training and improve convergence
3. **Max Pooling**: Applied after each conv block to reduce spatial dimensions (32→16→8→4→2)
4. **Multi-layer Fully Connected Network**: Three FC layers (1024→512→10) with dropout regularization

Key Technical Features

Batch Normalization: Each convolutional layer is followed by batch normalization, which:

- Normalizes inputs to each layer, reducing internal covariate shift
- Allows higher learning rates and faster convergence
- Acts as a regularizer, reducing the need for dropout in convolutional layers

Dropout Regularization: Applied in fully connected layers with a rate of 0.5 to prevent overfitting while maintaining model capacity.

Proper Weight Initialization:

- Kaiming normal initialization for convolutional and linear layers
- Prevents vanishing/exploding gradient problems

Training Optimizations

Adam Optimizer: Used instead of SGD for better convergence with:

- Learning rate: 0.001
- Weight decay: 1e-4 (L2 regularization)
- Adaptive learning rates for each parameter

Training Strategy: The model was trained for 10 epochs, achieving:

- Validation accuracy: 81.4% (exceeding the 70% requirement)
- Test accuracy: 81.46%
- Consistent performance across validation and test sets

Programming II: Attention Mechanism

We implement the scaled dot-product attention mechanism equivalent to the following:

```python
def scaled_dot_product_attention(query, key, value, attn_mask=None, dropout_p=0.0):
    "Compute 'Inefficient Scaled Dot Product Attention'"
    L, S = query.size(-2), key.size(-2)
    scale_factor = 1 / math.sqrt(query.size(-1))
    attn_bias = torch.zeros(L, S, dtype=query.dtype, device=query.device)

    if attn_mask is not None:
        if attn_mask.dtype == torch.bool:
            attn_bias.masked_fill_(attn_mask.logical_not(), float("-inf"))
        else:
            attn_bias = attn_mask + attn_bias

    attn_weight = query @ key.transpose(-2, -1) * scale_factor
    attn_weight += attn_bias
    attn_weight = torch.softmax(attn_weight, dim=-1)
    attn_weight = torch.dropout(attn_weight, dropout_p, train=True)
    return attn_weight @ value
```