Data 188 Introduction to Deep Learning
Spring 2026 Eric Kim

Discussion 06

This discussion covers computer vision and some review!

1. Computer Vision Architecture

(a) Design a two-layer ConvNet image classification model with the following features:

- Input is RGB image with $H = 224, W = 224$.
- Two `Conv2d` layers (with bias), each followed by `BatchNorm2d` and `ReLU`.
- Square filters of size 3×3.
- All intermediate spatial feature maps must have 8 channels.
- Spatial resolution must decrease by a factor of 2 after each convolution.
- Final classification layer for $K = 1000$ classes.

i. Specify the layer parameters (input channels, input features, output channels, output features, stride, padding, etc.) for each layer in the model architecture.

Solution: From the requirements, we can derive the following information that is true regardless of the specific architecture we choose:

- $C_{\text{in}} = 3$ for the first convolutional layer (since the input is an RGB image)
- $C = 8$ (for input/output channels of the intermediate feature maps)
- $k = 3$ (kernel size)
- stride $= 2$ (spatial resolution decreases by a factor of 2)
- out_features $= 1000$ (final linear classification layer for $K = 1000$ classes)

Let B be the batch size.

Approach 1: Use `Conv2d` with `stride=2` for spatial reduction.

- `Conv2d(in_channels=3, out_channels=8, kernel_size=3, stride=2, padding=1)` → Output shape: `[B, 8, 112, 112]`
- `BatchNorm2d(num_features=8)` → Output shape: `[B, 8, 112, 112]` (`num_features` is the number of output channels in the previous layer)
- `ReLU()` → Output shape: `[B, 8, 112, 112]`
- `Conv2d(in_channels=8, out_channels=8, kernel_size=3, stride=2, padding=1)` → Output shape: `[B, 8, 56, 56]`
- `BatchNorm2d(num_features=8)` → Output shape: `[B, 8, 56, 56]`
- `ReLU()` → Output shape: `[B, 8, 56, 56]`
- `AvgPool2d(kernel_size=56)` → Output shape: `[B, 8, 1, 1]` (global average pooling, for intuition see Lec 12, Slide 15)
- `Flatten()` → Output shape: `[B, 8]`
- `Linear(in_features=8, out_features=1000, bias=True)` → Output shape: `[B, 1000]`

- `CrossEntropyLoss()` → Output shape: `[1]`

Approach 2: Use `AvgPool2d` with `stride=2` for spatial reduction while the `Conv2d` layers use `stride=1`.

- `AvgPool2d(kernel_size=3, stride=2, padding=1)` → Output shape: `[B, 3, 112, 112]`
- `Conv2d(in_channels=3, out_channels=8, kernel_size=3, stride=1, padding=1)` → Output shape: `[B, 8, 112, 112]`
- `BatchNorm2d(num_features=8)` → Output shape: `[B, 8, 112, 112]`
- `ReLU()` → Output shape: `[B, 8, 112, 112]`
- `AvgPool2d(kernel_size=3, stride=2, padding=1)` → Output shape: `[B, 8, 56, 56]`
- `Conv2d(in_channels=8, out_channels=8, kernel_size=3, stride=1, padding=1)` → Output shape: `[B, 8, 56, 56]`
- `BatchNorm2d(num_features=8)` → Output shape: `[B, 8, 56, 56]`
- `ReLU()` → Output shape: `[B, 8, 56, 56]`
- `AvgPool2d(kernel_size=56)` → Output shape: `[B, 8, 1, 1]`
- `Flatten()` → Output shape: `[B, 8]`
- `Linear(in_features=8, out_features=1000, bias=True)` → Output shape: `[B, 1000]`
- `CrossEntropyLoss()` → Output shape: `[1]`

ii. Calculate the total number of trainable parameters in the model architecture you designed in part (i).

Solution: Approach 1:

- **Conv2d (Layer 1):** $(C_{\text{in}} \times C_{\text{out}} \times k \times k) + \text{bias} = (3 \times 8 \times 3 \times 3) + 8 = 224$ parameters.
- **BatchNorm2d (Layer 1):** $2 \times 8 = 16$ parameters (multiply `num_features` by 2 to account for the γ and β parameters).
- **Conv2d (Layer 2):** $(C_{\text{in}} \times C_{\text{out}} \times k \times k) + \text{bias} = (8 \times 8 \times 3 \times 3) + 8 = 584$ parameters.
- **BatchNorm2d (Layer 2):** $2 \times 8 = 16$ parameters.
- **AvgPool2d + Flatten:** 0 parameters.
- **Linear Layer:** $(C_{\text{in}} \times C_{\text{out}}) + \text{bias} = (8 \times 1000) + 1000 = 9,000$ parameters.
- **Total Parameters:** $224 + 16 + 584 + 16 + 9,000 = \mathbf{9,840}$

Approach 2:

- **AvgPool2d (Block 1):** 0 parameters.
- **Conv2d (Block 1):** $(3 \times 8 \times 3 \times 3) + 8 = 224$ parameters.
- **BatchNorm2d (Block 1):** $2 \times 8 = 16$ parameters.
- **AvgPool2d (Block 2):** 0 parameters.
- **Conv2d (Block 2):** $(8 \times 8 \times 3 \times 3) + 8 = 584$ parameters.
- **BatchNorm2d (Block 2):** $2 \times 8 = 16$ parameters.
- **AvgPool2d (Global) + Flatten:** 0 parameters.
- **Linear Layer:** $(8 \times 1000) + 1000$ (bias) $= 9,000$ parameters.
- **Total Parameters:** $224 + 16 + 584 + 16 + 9,000 = \mathbf{9,840}$

(b) Suppose our training batch size B is very small (e.g., $B = 1$). What specific change should we make to the model architecture to ensure training stability?

Solution: With $B = 1$, batch norm becomes unstable because the batch variance is zero and the running statistics cannot be properly estimated. To maintain stability, we replace batch norm with layer norm. Layer norm normalizes across the channel and spatial dimensions of each sample individually, making it independent of the batch dimension.

2. ConvNormAct

Suppose we have the following code:

```
def create_conv_norm_act(C: int) -> Module:
    return Sequential(
        Conv2d(
            in_channels=C,
            out_channels=C,
            kernel_size=3,
            stride=1,
            padding=1,
        ),
        BatchNorm2d(num_features=C),
        ReLU(),
    )
```

Implement the `residual_conv_norm_act_forward` function that adds a "skip connection" to the `ConvNormAct` block. Assume that we use elementwise-addition to implement the skip connection.

```
def residual_conv_norm_act_forward(
    Z_in: Tensor, # shape=[b, c, h, w]
    conv_norm_act: Module, # return value of create_conv_norm_act
) -> Tensor:
    Z_conv = ____________________________________________________
    return ____________________________________________________
```

Solution:

```
Z_conv = conv_norm_act(Z_in)
return Z_conv + Z_in # skip connection
```

Note that the shapes must be exactly the same for the elementwise addition to work!

3. Computational Graph

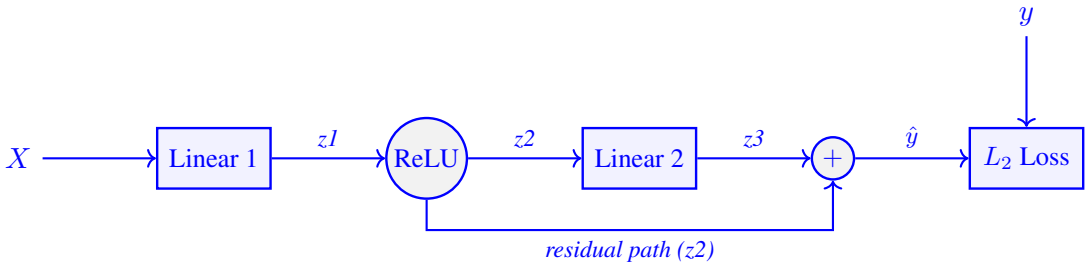
(a) Draw a computational graph for this single-item, scalar model:

```
# X (input) shape=[1]
# y (ground-truth) shape=[1]
linear1 = Linear(in_feats=1, out_feats=1, bias=False)
relu = ReLU()
linear2 = Linear(in_feats=1, out_feats=1, bias=False)
```



```
z1 = linear1(X)
z2 = relu(z1)
z3 = linear2(z2)
y_hat = z2 + z3
L = L2Loss(y_hat, y)
```

Solution:



(b) Let $X = 1$, and $y = 0.5$. Let the weights of the linear layers be $w_1 = 1$, $w_2 = 2$.

i. Compute the forward pass.

Solution:

- $z_1 = w_1 \cdot X = 1 \cdot 1 = 1$
- $z_2 = \text{ReLU}(z_1) = \max(0, z_1) = 1$
- $z_3 = w_2 \cdot z_2 = 2 \cdot 1 = 2$
- $\hat{y} = z_2 + z_3 = 1 + 2 = 3$
- $\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(3 - 0.5)^2 = 3.125$

ii. Compute the adjoints for all nodes (do the backward pass).

Solution:

- $\bar{\mathcal{L}} = 1$ (by definition, the adjoint of the loss with respect to itself is 1)
- $\bar{\hat{y}} = \bar{\mathcal{L}} \cdot \frac{\partial \mathcal{L}}{\partial \hat{y}} = 1 \cdot [2 \cdot \frac{1}{2} \cdot (\hat{y} - y)] = 3 - 0.5 = 2.5$
- $\bar{z_3} = \bar{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} = 2.5 \cdot 1 = 2.5$
- $\bar{z_2} = \underbrace{\bar{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2}}_{\text{Residual path}} + \underbrace{\bar{z_3} \cdot \frac{\partial z_3}{\partial z_2}}_{\text{Linear 2 path}} = 2.5 \cdot 1 + 2.5 \cdot w_2 = 2.5 + 2.5 \cdot 2 = 7.5$
- $\bar{z_1} = \bar{z_2} \cdot \frac{\partial z_2}{\partial z_1} = 7.5 \cdot \text{ReLU}'(z_1) = 7.5 \cdot 1 = 7.5$ (since $z_1 > 0$, the derivative of ReLU is 1, otherwise it would be 0)

iii. Compute the final gradients for the trainable parameters w_1 and w_2.

Solution:

- $\bar{w_2} = \bar{z_3} \cdot \frac{\partial z_3}{\partial w_2} = 2.5 \cdot z_2 = 2.5 \cdot 1 = 2.5$
- $\bar{w_1} = \bar{z_1} \cdot \frac{\partial z_1}{\partial w_1} = 7.5 \cdot X = 7.5 \cdot 1 = 7.5$

(c) Now consider $X \in \mathbb{R}^2, y \in \mathbb{R}^2, W_1 \in \mathbb{R}^{4\times 2}, W_2 \in \mathbb{R}^{2\times 4}$. This means that we left multiply weight matrices, e.g. $z_1 = W_1 X$ and $z_3 = W_2 z_2$. Additionally, recall:

- Vector L2 loss is defined as $\mathcal{L} = \frac{1}{2}\|\hat{y} - y\|_2^2$
- For the matrix multiplication operation $C = AB$, the gradients are $\frac{\partial \mathcal{L}}{\partial A} = \left(\frac{\partial \mathcal{L}}{\partial C}\right) B^T$ and $\frac{\partial \mathcal{L}}{\partial B} = A^T \left(\frac{\partial \mathcal{L}}{\partial C}\right)$

Express the gradients $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ symbolically using the upstream gradients.

Solution:

- $\bar{\mathcal{L}} = 1$ (by definition, the adjoint of the loss with respect to itself is 1)
- $\bar{\hat{y}} = \bar{\mathcal{L}} \cdot \frac{\partial \mathcal{L}}{\partial \hat{y}} = 1 \cdot [2 \cdot \frac{1}{2} \cdot (\hat{y} - y)] = (\hat{y} - y)$
- $\bar{z_3} = \bar{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} = \bar{\hat{y}} \cdot 1 = \bar{\hat{y}}$
- $\bar{z_2} = \underbrace{\bar{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2}}_{\text{Residual path}} + \underbrace{\frac{\partial z_3}{\partial z_2} \cdot \bar{z_3}}_{\text{Linear 2 path}} = \bar{\hat{y}} \cdot 1 + W_2^\top \cdot \bar{\hat{y}} = \bar{\hat{y}} + W_2^\top \cdot \bar{\hat{y}}$
  (see table above for the gradient of matrix multiplication, where $A = W_2$, $B = z_2$, and $C = z_3$)
- $\bar{z_1} = \bar{z_2} \cdot \frac{\partial z_2}{\partial z_1} = \bar{z_2} \odot \text{ReLU}'(z_1) = (\bar{\hat{y}} + W_2^\top \cdot \bar{\hat{y}}) \odot \text{ReLU}'(z_1)$
  where $\odot$ represents elementwise multiplication, and $\text{ReLU}'(z_1)$ is a vector of the same shape as z_1 where each element is 1 if the corresponding element of z_1 is greater than 0, and 0 otherwise.
- $\bar{W_2} = \bar{z_3} \cdot \frac{\partial z_3}{\partial W_2} = \bar{z_3} \cdot z_2^\top = \bar{\hat{y}} \cdot z_2^\top$
  (see table above for the gradient of matrix multiplication, where $A = W_2$, $B = z_2$, and $C = z_3$)
- $\bar{W_1} = \bar{z_1} \cdot \frac{\partial z_1}{\partial W_1} = \bar{z_1} \cdot X^\top = [(\bar{\hat{y}} + W_2^\top \cdot \bar{\hat{y}}) \odot \text{ReLU}'(z_1)] \cdot X^\top$
  (see table above for the gradient of matrix multiplication, where $A = W_1$, $B = X$, and $C = z_1$)

Note on dimensions: You might have noticed the dimensions of $z_2 \in \mathbb{R}^{4\times 1}$ and $z_3 \in \mathbb{R}^{2\times 1}$ are different and thus cannot be directly added to compute $\hat{y}$. For our class, we will not worry about the details but in practice one way to fix this issue is to perform a linear projection (e.g. matrix multiplication) on one or both inputs so that the dimensions match.

4. Miscellaneous Review!

(a) Take the Jacobian of the following vector function $\mathbf{f}(x, y, z)$:

$$\mathbf{f}(x, y, z) = \begin{bmatrix} x^2 + y^2 \\ 5z \\ x \sin(y) \end{bmatrix}$$

Solution: The Jacobian matrix $\mathbf{J}$ is defined as:

$$\mathbf{J} = \frac{\partial(f_1, f_2, f_3)}{\partial(x, y, z)} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} 2x & 2y & 0 \\ 0 & 0 & 5 \\ \sin(y) & x\cos(y) & 0 \end{bmatrix}$$

(b) What are logits? What is the softmax function? What loss do we use in softmax regression?

Solution: Logits are the raw scores output by the last layer of a neural network. They can take any real value $(-\infty, \infty)$. The softmax function squashes a vector of K logits $\mathbf{z}$ into a probability distribution where each value is in $(0, 1)$ and the sum is 1:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

In softmax regression (multi-class classification), we typically use the Cross-Entropy Loss:

$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i)$$

(c) Recall the equations for the Adam optimizer:

$$u_t = \beta_1 u_{t-1} + (1-\beta_1)\nabla_\theta L(\theta_t) \quad \text{First moment}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta L(\theta_t))^2 \quad \text{Second moment}$$
$$\hat{u}_t = \frac{u_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad \text{Bias correction}$$
$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{u}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad \text{Weight update}$$

What do the first moment, second moment, and bias correction equations do?

Solution:

- First moment: Smooths out the gradients to navigate noisy surfaces and speed up learning in consistent directions.
- Second moment: Scales the learning rate for each parameter individually based on the magnitude of recent gradients.
- Bias Correction: Compensates for the fact that u_t and v_t are initialized at zero, which otherwise creates a "cold start" problem where estimates are biased toward zero in early iterations.

(d) Normalize the following matrix using LayerNorm, then BatchNorm. Assume that `X.shape=[batchsize, dim]`. Ignore the learnable parameters of each normalization layer for this question (e.g. no need to scale and shift the normalized output).

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Solution: Let the rows be samples and columns be features. LayerNorm (Normalize across features/rows): For Row 1: $\mu = 1.5, \sigma = 0.5$. Normalized: $[-1, 1]$. For Row 2: $\mu = 3.5, \sigma = 0.5$. Normalized: $[-1, 1]$.

$$\text{LayerNorm}(X) = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$$

BatchNorm (Normalize across batch/columns): For Col 1: $\mu = 2, \sigma = 1$. Normalized: $[-1, 1]$. For

Col 2: $\mu = 3, \sigma = 1$. Normalized: $[-1, 1]$.

$$\text{BatchNorm}(X) = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$$

(e) Given a convolutional layer with a 3×3 kernel and a padding of 2, calculate the output dimensions for an input of size 5×8. Assume a stride of 1.

Solution: Using the convolution output size formula $O = \lfloor \frac{I-K+2P}{S} \rfloor + 1$ where O is the output size, I is the input size, K is the kernel size, P is the padding, and S is the stride:

- Height: $H_{out} = \lfloor \frac{5-3+2(2)}{1} \rfloor + 1 = \lfloor 2 + 4 \rfloor + 1 = 7$
- Width: $W_{out} = \lfloor \frac{8-3+2(2)}{1} \rfloor + 1 = \lfloor 5 + 4 \rfloor + 1 = 10$

The output dimensions are 7×10.

Contributors:

- Eric Kim, Andria Xu.

