NEURAL NETWORK CLASSIFICATION

– Matee Vadrukchid –

Part 1: Basic Pytorch

Part 1: Basic Pytorch

2.1 Assignment 1

Instruction

1. Create a **2D random** tensor `X1` of shape `(5, 6)` with `torch.randn`.
2. Print:
   - The **shape** of `X1`
   - The **data type** of `X1`
   - The **mean** of all elements in `X1`

Part 1: Basic Pytorch

2.2 Assignment 2

Instruction

1. Create an **integer** tensor `X2` of shape `(4,4)` with random integers in [0..10).
2. Convert it to **float** and store in `X2_float`.
3. Show the **difference** (`X2_float - X2`) and explain why the difference is zero or non-zero.

Part 1: Basic Pytorch

2.3 Assignment 3

Instruction

1. Create a random **float** tensor `X3` of shape `(3,2)`.
2. **Transpose** it to get shape `(2,3)`.
3. Perform **matrix multiplication** of `X3 @ X3_transposed`.
4. Print the resulting shape.

Part 1: Basic Pytorch

2.4 Assignment 4

Instruction

1. Create two tensors `X4_A` and `X4_B` of shape `(2,3)`, each with random integers in range [1..5).
2. Compute:
   - `X4_A + X4_B` (element-wise addition)
   - `X4_A - X4_B` (element-wise subtraction)
   - `X4_A * X4_B` (element-wise multiplication)
   - Print each result and shape.

Part 1: Basic Pytorch

2.5 Assignment 5

Instruction

1. Create a tensor `X5` of shape `(3, 4)` with `torch.linspace(1, 12, steps=12)` and then reshape it.
2. **Flatten** `X5` into 1D.
3. Print both the original shape and the flattened shape.

Part 1: Basic Pytorch

2.6 Assignment 6

Instruction

1. Create a random float tensor `X6` of shape `(4,4)`.
2. Compute the **column-wise sum** and the **row-wise sum**.
3. Print both results and confirm their shapes are `(4,)`.

Part 1: Basic Pytorch

2.7 Assignment 7

Instruction

1. Create two random float tensors `X7_A` and `X7_B` each of shape `(2,2)`.
2. **Concatenate** them along dimension **0** => result shape `(4,2)`.
3. **Concatenate** them along dimension **1** => result shape `(2,4)`.
4. Print both results.

Part 1: Basic Pytorch

2.8 Assignment 8

Instruction

1. Create a **3D** random tensor `X8` of shape `(2, 3, 4)` with `torch.randint`.
2. Compute the **maximum value** in the entire tensor.
3. Compute the **mean** along dimension **2**.
4. Print the shapes of the resulting tensors.

Part 1: Basic Pytorch

2.9 Assignment 9

Instruction

1. Create a random float tensor `X9` of shape `(3,3)`.
2. Compute the **determinant** of this matrix using `torch.linalg.det(X9)`. (PyTorch 1.10+ has `torch.linalg.det`)
3. Print the result and interpret it (could be near zero if the matrix is nearly singular).

Part 1: Basic Pytorch

2.10 Assignment 10

Instruction

1. Create a random float tensor `X10` of shape `(3,2)` using `torch.randn`.
2. Use `torch.stack` to replicate `X10` **3 times** along a **new dimension** => shape `(3, 3, 2)`.
3. Print the final shape.

Part 2: Softmax Function and the Categorical Cross-Entropy Loss Neural Network

Part 2: Softmax and Cross -Entropy Loss in NN

1. Definitions

1. **Logits**:

$$\mathbf{z} = (z_1, z_2, \ldots, z_K)$$

   These are the raw scores (unbounded) for each of the K classes.

2. **Softmax Function**:

   For each component $i \in \{1, \ldots, K\}$, we define

$$\hat{y}_i \;=\; \mathrm{softmax}(\mathbf{z})_i \;=\; \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}.$$

   $\hat{y}_i$ represents the predicted probability of class i.

Part 2: Softmax and Cross -Entropy Loss in NN

3. **Ground-Truth (Target) Vector**:

$$\mathbf{y} = (y_1, y_2, \ldots, y_K),$$

which is often a **one-hot** vector for the correct class (e.g., if the true class is 2 out of 3, then $\mathbf{y} = (0, 1, 0)$), or a probability distribution that sums to 1.

4. **Categorical Cross-Entropy Loss**:

$$\mathcal{L}(\mathbf{z}, \mathbf{y}) \;=\; -\sum_{i=1}^{K} y_i \, \log(\hat{y}_i).$$

We want $\frac{\partial \mathcal{L}}{\partial z_k}$ for each k.

Part 2: Softmax and Cross -Entropy Loss in NN

2. Rewrite the Loss in Terms of Logits z_i

First, note:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}.$$

Then

$$\log(\hat{y}_i) = \log\Big(\frac{e^{z_i}}{\sum_j e^{z_j}}\Big) = z_i - \log\Big(\sum_{j=1}^{K} e^{z_j}\Big).$$

Hence the loss becomes:

$$\begin{aligned}\mathcal{L}(\mathbf{z},\mathbf{y}) &= -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \\ &= -\sum_{i=1}^{K} y_i \Big(z_i - \log\big(\sum_{j=1}^{K} e^{z_j}\big)\Big) \\ &= -\sum_{i=1}^{K} y_i z_i + \sum_{i=1}^{K} y_i \log\big(\sum_{j=1}^{K} e^{z_j}\big).\end{aligned}$$

Factor out the term $\log\big(\sum_j e^{z_j}\big)$ in the second sum:

$$\mathcal{L}(\mathbf{z},\mathbf{y}) = -\sum_{i=1}^{K} y_i z_i + \log\Big(\sum_{j=1}^{K} e^{z_j}\Big) \sum_{i=1}^{K} y_i.$$

Since $\mathbf{y}$ is a distribution, $\sum_{i=1}^{K} y_i = 1$. Therefore,

$$\mathcal{L}(\mathbf{z},\mathbf{y}) = -\sum_{i=1}^{K} y_i z_i + \log\Big(\sum_{j=1}^{K} e^{z_j}\Big).$$

Part 2: Softmax and Cross -Entropy Loss in NN

3. Take the Derivative w.r.t. z_k

We now compute $\frac{\partial \mathcal{L}}{\partial z_k}$. From the expression

$$\mathcal{L} = -\sum_{i=1}^{K} y_i z_i \; + \; \log\Big(\sum_{j=1}^{K} e^{z_j}\Big),$$

we split it into two terms:

1. $-\sum_{i=1}^{K} y_i z_i$
2. $\log\left(\sum_{j=1}^{K} e^{z_j}\right)$

Part 2: Softmax and Cross -Entropy Loss in NN

3.1 Derivative of the First Term

$$-\sum_{i=1}^{K} y_i z_i$$

Since y_i is a constant w.r.t. z_k, the only term that matters is when $i = k$:

$$\frac{\partial}{\partial z_k}\left(-\sum_{i=1}^{K} y_i z_i\right) = -y_k.$$

Part 2: Softmax and Cross -Entropy Loss in NN

3.2 Derivative of the Second Term

$$\log\Big(\sum_{j=1}^{K} e^{z_j}\Big).$$

Use the chain rule:

$$\frac{d}{dx}(\log_a x) = \boxed{\frac{1}{x \ln a}}$$

$$\frac{\partial}{\partial z_k} \log\Big(\sum_{j} e^{z_j}\Big) = \frac{1}{\sum_{j=1}^{K} e^{z_j}} \frac{\partial}{\partial z_k} \Big(\sum_{j=1}^{K} e^{z_j}\Big).$$

Inside the sum $\sum_{j=1}^{K} e^{z_j}$, the derivative wrt z_k is e^{z_k} (because derivative of e^{z_j} w.r.t. z_k is e^{z_k} if $j = k$, else 0). Thus:

$$\frac{\partial}{\partial z_k} \log\Big(\sum_{j=1}^{K} e^{z_j}\Big) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} = \hat{y}_k.$$

(This is precisely the softmax component for class k.)

Putting it all together, the second term's derivative is:

$$\hat{y}_k.$$

Part 2: Softmax and Cross -Entropy Loss in NN

3.3 Combine Both Parts

Hence,

$$\frac{\partial \mathcal{L}}{\partial z_k} = \underbrace{(-y_k)}_{\text{from first term}} + \underbrace{\hat{y}_k}_{\text{from second term}} = \hat{y}_k - y_k.$$

Final result for each k:

$$\boxed{\frac{\partial \mathcal{L}}{\partial z_k} = \hat{y}_k - y_k.}$$

Part 2: Softmax and Cross -Entropy Loss in NN

4. Vector Form of the Gradient

If we write $\mathbf{z}$ as a vector and $\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{z})$ as the corresponding probability vector, then the gradient of $\mathcal{L}$ w.r.t. $\mathbf{z}$ is:

$$\nabla_{\mathbf{z}}\mathcal{L} = (\hat{y}_1 - y_1, \hat{y}_2 - y_2, \ldots, \hat{y}_K - y_K) = \hat{\mathbf{y}} - \mathbf{y}.$$

Thus, in one line:

$$\boxed{\nabla_{\mathbf{z}}\left[-\sum_i y_i \log(\hat{y}_i)\right] = \hat{\mathbf{y}} - \mathbf{y}.}$$

Part 2: Softmax and Cross -Entropy Loss in NN

Summary

1. **Softmax**: $\hat{y}_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$.
2. **Cross-Entropy**: $\mathcal{L} = -\sum_k y_k \log(\hat{y}_k)$.
3. **Resulting Gradient**: $\frac{\partial \mathcal{L}}{\partial z_k} = \hat{y}_k - y_k$.