Before starting

20 November 2024 17:55

Lec = 2

1. Boring but useful
2. Knowledge of Deep learning
3. Very similar to NumPy
4. Lecture flow
5. Very close to PyTorch official documentation
6. Watch like a lecture

What are Tensors

20 November 2024 17:56

Tensor is a specialized multi-dimensional array designed for mathematical and computational efficiency.

Real-World Examples

1. Scalars: 0-dimensional tensors (a single number)
   - Represents a single value, often used for simple metrics or constants.
   - Example:
     - Loss value: After a forward pass, the loss function computes a single scalar value indicating the difference between the predicted and actual outputs.
     - Example: 5.0 or -3.14
2. Vectors: 1-dimensional tensors (a list of numbers)
   - Represents a sequence or a collection of values.
   - Example:
     - Feature vector: In natural language processing, each word in a sentence may be represented as a 1D vector using embeddings.
     - Example: [0.12, -0.84, 0.33] (a word embedding vector from a pre-trained model like Word2Vec or Glove).
3. Matrices: 2-dimensional tensors (a 2D grid of numbers)
   - Represents tabular or grid-like data.
   - Example:
     - Grayscale images: A grayscale image can be represented as a 2D tensor, where each entry corresponds to the pixel intensity.
     - Example:
       [[0, 255, 128],
       [34, 90, 180]]
4. 3D Tensors: Coloured images (RGB)
   - Adds a third dimension, often used for stacking data.
   - Example:
     - RGB Images: A single RGB image is represented as a 3D tensor (width × height × channels).
     - Examples:
       - RGB Image (e.g., 256x256): Shape [256, 256, 3]
5. 4D Tensors: Batches of RGB images
   - Adds the batch size as an additional dimension to 3D data.
   - Example:
     - Batches of RGB Images: A dataset of coloured images is represented as a 4D tensor (batch size × width × height × channels).
     - Example: A batch of 32 images, each of size 128x128 with 3 colour channels (RGB), would have shape [32, 128, 128, 3].
6. 5D Tensors: Video data
   - Adds a time dimension for data that changes over time (e.g., video frames).
   - Example:
     - Video Clips: Represented as a sequence of frames, where each frame is an RGB image.
     - Example: A batch of 10 video clips, each with 16 frames of size 64x64 and 3 channels (RGB), would have shape [10, 16, 64, 64, 3].

Why Are Tensors Useful?

1. Mathematical Operations
   - Tensors enable efficient mathematical computations (addition, multiplication, dot product, etc.) necessary for neural network operations.
2. Representation of Real-world Data
   - Data like images, audio, videos, and text can be represented as tensors:
     - Images: Represented as 3D tensors (width × height × channels).
     - Text: Tokenized and represented as 2D or 3D tensors (sequence length × embedding size).
3. Efficient Computations

* Tensor

Summary

→ First, what is tensor,
A tensor is a data structure.
it mean the work that an array does in 1D, the same work tensor does in multi dimensions.

→ dimension mean how many direction it is spread
like we can see in fig 1

- In Rank 1 there is 1 direction, so it is 1 dimension.
- In Rank 2 there is 2 direction, so it is 2 dimension

Real world example of tensor

① 0-dimension :- it means it is not spread in any direction
Example :- Scalars → 1, 2, 3 - - - .

② 1-dimension :- it means it is spread in 1 direction
Like :- Vector, array

③ 2-dimension :- it means it is spread in 2 direction
Like :- Matrices (Grayscale image)

④ 3-dimension :- it means it is spread in 3 direction
Like :- Colored image (RGB)

⑤ 4-dimension :- it means it is spread in 4 direction
Like :- batch of RGB image
We can see in fig :-

⑥ 5-dimension :- it means it is spread in 5 direction
like :- Video data.

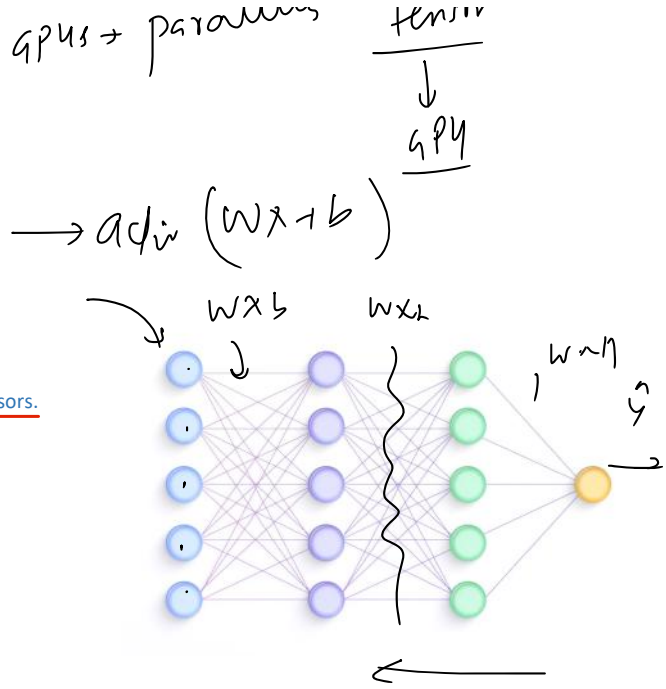
size).

3. Efficient Computations

- Tensors are optimized for hardware acceleration, allowing computations on GPUs or TPUs, which are crucial for training deep learning models.

Where Are Tensors Used in Deep Learning?

1. Data Storage
   - Training data (images, text, etc.) is stored in tensors.
2. Weights and Biases
   - The learnable parameters of a neural network (weights, biases) are stored as tensors.
3. Matrix Operations
   - Neural networks involve operations like matrix multiplication, dot products, and broadcasting—all performed using tensors.
4. Training Process
   - During forward passes, tensors flow through the network.
   - Gradients, represented as tensors, are calculated during the backward pass.



Summary

* Why are tensor useful

① Mathematical operation

With tensor we can easily do comman mathmatical operation.

like:- Addition, sub, Multi, dot product etc.

② Representation of real-world data

We can use tensor to map or represent real-world data like image, audio, video or text.

③ Efficient computation

We can run tensor on a GPU and do parallel computation.

example

if we can add two $\begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 8 \\ 9 & 4 \end{bmatrix}$ on a CPU, it will add element one by one, so it will take more time.

but if we represent it as a tensor and run it on a GPU, then all addition will run in parallel and it will be very fast.

* Where are tensor used in Deep learning

① Data storage :- We store data (like image, text etc.) in tensor.

② Weights and Bias :- Weights and bias are also stored in tensor (You can see in the figure)

③ Matrix operation :- In every step (like multiplying a * b) the operation is done using tensor.

④ Training process :- From forward pass to backward pass, the whole process is done with tensor.

Tensor-in-pytorch - colab-code-summary

PyTorch me, **tensor** ek fundamental data structure hai. Aap ise ek multi-dimensional array samajh sakte hain, jo numbers ko store karta hai. Ye NumPy arrays ki tarah hi hote hain, lekin inka sabse bada fayda yeh hai ki inhein GPU par bhi run kiya ja sakta hai, jisse calculations bahut fast ho jaati hain.[1][2][3]

Tensor Kaise Banaye? (How to Create a Tensor?)

Tensor banane ke kai tarike hain:

- **torch.empty(rows, cols)**: Yeh aapko ek uninitialized tensor bana kar deta hai. Isme pehle se memory me jo bhi values hongi, woh aa jayengi.
    - Example: torch.empty(2,3)
- **torch.zeros(rows, cols)**: Yeh ek tensor banata hai jiske saare elements zero hote hain.[1][4]
    - Example: torch.zeros(2,3)
- **torch.ones(rows, cols)**: Yeh ek tensor banata hai jiske saare elements one hote hain.[4]
    - Example: torch.ones(2,3)
- **torch.rand(rows, cols)**: Yeh 0 aur 1 ke beech ki random values se bhara hua tensor banata hai.[4][5]
    - Example: torch.rand(2,3)
- **torch.manual_seed(seed_number)**: Agar aap chahte hain ki aapke random numbers har baar same generate ho, toh torch.rand() se pehle is function ka use karein. Isse reproducibility aati hai.[6]
    - Example:

      codePython

      ```
      torch.manual_seed(100)
      torch.rand(2,3)
      ```

- **torch.tensor([[...],[...]])**: Aap Python list se direct tensor bana sakte hain.[5][7]

- Example: torch.tensor([[1,2,3],[4,5,6]])

Kuch Aur Tarike Tensor Banane Ke (Other Ways to Create Tensors)

- **torch.arange(start, end, step)**: Yeh ek 1D tensor banata hai jisme start se end-1 tak values hoti hain, aur har value ke beech step ka difference hota hai.
  - Example: torch.arange(1,10,2) (Output: tensor([1, 3, 5, 7, 9]))
- **torch.linspace(start, end, steps)**: Yeh start se end tak steps number of equally spaced values ka 1D tensor banata hai.
  - Example: torch.linspace(1,10,10) (Output: tensor([1., 2., ..., 10.]))
- **torch.eye(size)**: Yeh ek identity matrix (2D tensor) banata hai, jiske diagonal me 1s aur baaki sab jagah 0s hote hain.
  - Example: torch.eye(5)
- **torch.full((rows, cols), value)**: Yeh diye gaye shape ka ek tensor banata hai aur usko value se bhar deta hai.
  - Example: torch.full((3,3),5)

Tensor ka Shape jaanna (Understanding Tensor Shapes)

- **.shape**: Kisi bhi tensor ka shape (dimensions) janne ke liye is attribute ka use hota hai.
  - Example: x = torch.tensor([[1,2,3],[4,5,6]]) -> x.shape (Output: torch.Size([2, 3]))
- **torch.empty_like(x)**: Yeh x tensor ke jaisa hi ek empty tensor banata hai, jiska shape x ke barabar hota hai.[4]
- **torch.zeros_like(x)**: Yeh x tensor ke jaisa hi ek tensor banata hai jiske saare elements zero hote hain.[8]
- **torch.ones_like(x)**: Yeh x tensor ke jaisa hi ek tensor banata hai jiske saare elements one hote hain.[8]

- **torch.rand_like(x)**: Yeh x tensor ke jaisa hi ek random tensor banata hai. (Note: Iske liye x ka data type float hona chahiye).[4]

Tensor Data Types

Har tensor ka ek data type hota hai.

- **Data type check karna**: Tensor ke aage .dtype likh kar aap uska data type dekh sakte hain.
    - Example: x.dtype (Output: torch.int64)
- **Data type set karna**: Tensor banate waqt aap dtype parameter de sakte hain ya baad me .to() function se change kar sakte hain.
    - Example 1: torch.tensor([1,2,3], dtype=torch.float64)
    - Example 2: x.to(torch.float32)

Mathematical Operations

Tensors par alag-alag tarah ke math operations kar sakte hain.

1. Scalar Operations
Jab aap ek tensor ko kisi ek single number (scalar) se operate karte hain. Yeh operation tensor ke har element par apply hota hai.

- x + 2 (Har element me 2 add ho jayega)
- x * 2 (Har element 2 se multiply ho jayega)
- x ** 2 (Har element ka square ho jayega)

2. Element-wise Operations
Jab do same shape ke tensors ke beech operation hota hai. Ek tensor ka har element doosre tensor ke corresponding element se operate hota hai.

- a + b (Dono tensors ke corresponding elements add ho jayenge)
- a * b (Dono tensors ke corresponding elements multiply ho jayenge)

- **Kuch aur functions**:
    - torch.abs©: Har element ki absolute value (negative ko positive bana dega).
    - torch.round(d): Values ko round off kar dega.
    - torch.clamp(d, min=2, max=3): Values ko min aur max ke range me “clamp” yaani limit kar dega. Jo value 2 se kam hai wo 2 ban jayegi, aur jo 3 se zyada hai wo 3 ban jayegi.

3. Reduction Operations

Yeh operations tensor ko “reduce” karke ek single value ya chota tensor dete hain.

- torch.sum(e): Tensor ke saare elements ka sum.
- torch.mean(e): Tensor ke saare elements ka average.
- torch.max(e) / torch.min(e): Sabse badi / choti value batata hai.
- torch.argmax(e) / torch.argmin(e): Sabse badi / choti value ka **index** (position) batata hai.
- **Dimension ke saath**: dim=0 (column-wise operation) aur dim=1 (row-wise operation).
    - Example: torch.sum(e, dim=0) har column ka sum dega.

4. Matrix Operations

Yeh linear algebra ke operations hain jo 2D tensors (matrices) par apply hote hain.

- torch.matmul(f, g): Do matrices ka multiplication.
- torch.dot(v1, v2): Do 1D vectors ka dot product.
- torch.transpose(f, 0, 1): Matrix ka transpose (rows ko columns aur columns ko rows bana deta hai).
- torch.det(h): Matrix ka determinant.
- torch.inverse(h): Matrix ka inverse.

5. Comparison Operations

Yeh element-wise comparison karte hain aur ek boolean tensor (True/False values) return karte hain.

- i > j: Check karega i ka har element j ke corresponding element se bada hai ya nahi.
- i == j: Check karega elements barabar hain ya nahi.

6. Special Functions

Deep learning me use hone wale common functions.

- torch.log(k): Natural logarithm.
- torch.exp(k): Exponential (e^x).
- torch.sqrt(k): Square root.
- torch.sigmoid(k): Sigmoid activation.
- torch.softmax(k, dim=...) : Softmax activation.
- torch.relu(k): **ReLU** (Rectified Linear Unit). Agar value negative hai to 0 kar dega, positive hai to waisa hi rakhega.

In-place Operations

Jab aap chahte hain ki operation ka result ek naye tensor me store na ho, balki original tensor hi update ho jaye, to in-place operations ka use hota hai. Isse memory bachti hai.

- Inki pehchan _ (underscore) se hoti hai.
- Example: m.add_(n) ye m = m + n ke barabar hai, lekin ye m ko direct update kar deta hai.

Tensor ko Copy Karna (Copying a Tensor)

- **Galat Tarika (b = a)**: Isse tensor copy nahi hota, balki b usi tensor ko point karne lagta hai jisko a point kar raha hai. Agar aap a ko change karenge to b bhi change ho jayega.
- **Sahi Tarika (b = a.clone())**: clone() ek naya tensor banata hai jisme a ki saari values copy ho jaati hain. Ab a ko change karne se b par koi fark nahi padega.

Tensor ko GPU par Chalana (Tensor Operations on GPU)

Agar aapke paas compatible GPU hai, to aap calculations ko fast karne ke liye tensor ko GPU par move kar sakte hain.

1. **Device set karein**: device = torch.device('cuda')
2. **Tensor ko GPU par move karein**: b = a.to(device)
3. Ab b par kiye gaye saare operations GPU par run honge.

Tensor ko Reshape Karna (Reshaping Tensors)

- a.reshape(new_shape) ya a.view(new_shape): Tensor ke shape ko badalta hai, lekin total number of elements same rehne chahiye.
- a.flatten(): Multi-dimensional tensor ko ek single 1D tensor me badal deta hai.
- b.permute(2, 0, 1): Dimensions ki order ko change karta hai. Jaise image data me (height, width, channel) ko (channel, height, width) karna.
- c.unsqueeze(0): Ek nayi dimension add karta hai jiska size 1 hota hai. (Jaise image me batch dimension add karna).
- d.squeeze(0): Jis dimension ka size 1 hota hai, usko hata deta hai.

NumPy aur PyTorch

Aap aasani se PyTorch tensor aur NumPy array ke beech convert kar sakte hain.

- **Tensor se NumPy**: b = a.numpy()
- **NumPy se Tensor**: d = torch.from_numpy©

Important: Aksar, is tarah se convert karne par, dono (tensor aur array) ek hi memory location ko share karte hain. Matlab agar aap ek ko change karenge, to doosra bhi change ho jayega.