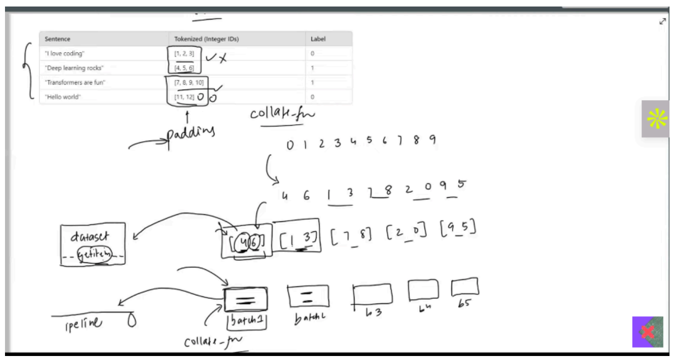
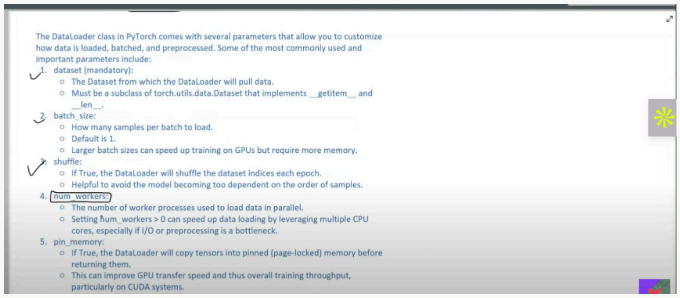
> - You can Custom Samplers as well but Why? - when u have a imbalanced dataset - 1 class have 99% of data and 2 class have 1% - if u random sampling ~100% of data will be off class 1 - make custom logic.

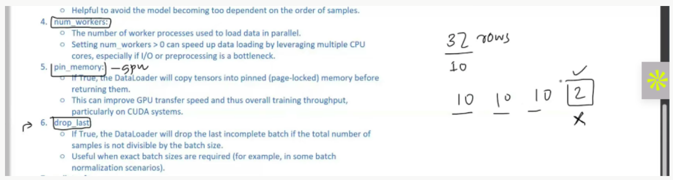
- Rows combine using Collate_fn - specifices how to combine samples from a dataset into a single batch .
  - be def , the dataloader uses a simple batch collation mech but this collate_fn allows to customize how the data shld be processed and batched

  - but why wld i need it? - if sampels are of diff sizes - cant be stacked - need to use padding - padding ka logic shal be written manually using collate_fn

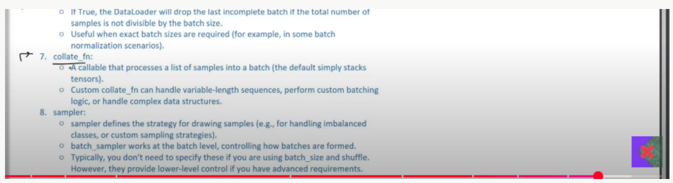




- pin memo also gpu = true



- drop last - left out batch ko drop ; if applies batch normalisation - drop it



basic dataset and lodaer implem : https://colab.research.google.com/drive/1D8rsmAODbfAiB1LjLeo-MgsxqpUaYhwJ

improving the batch grad desc code - applying mini batch grad desc - https://colab.research.google.com/drive/135PTL-ZQU9KMnu6OUBmICVouDS-RyVwL

Building an ANN or MLP pytorch (artifical neural network)

- Trying to build a ANN using whatever we learnt prevly
- kaggle - fashion MNIST - 700k 28*28 images ; rn only training on cpu so using 6k images ; imput layer *784 nodes) - hidden layer (128) - 64 both relu - output lauer (10 neurons softmax)j
- workflow - make dataloader objects for trian and text ; train loop ; eval

> ANN and handtypes accuracy calculation :
> https://colab.research.google.com/drive/1pYo1lOpbPAMkgXsBm3-XK37jLnTg1R2t

---

Training NN on GPU

- Check GPU availability :

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

- Move the Model to GPU

```
model = MyNN(X_train.shape[1])
model = model.to(device)
```

- Change the training loop by moving data to GPU - each batch of data is moved to gpu before processing.

```
for epoch in range(epochs):

  total_epoch_loss = 0

  for batch_features, batch_labels in train_loader:

    # move data to gpu
    batch_features, batch_labels = batch_features.to(device), batch_labels.to(device)

    # forward pass
    outputs = model(batch_features)
```

- Similarly Change the eval loop by moving data to gpu -

```
with torch.no_grad():

  for batch_features, batch_labels in test_loader:

    # move data to gpu
    batch_features, batch_labels = batch_features.to(device), batch_labels.to(device)
```

- Optimize the GPU Usage -
  a. Use Larger Batch Sizes - can better utilize gpu memory and reduce comp time per epoch

b. Enable DataLoader Pinning (use pin_memory = True) to speed up transfer form cpu to gpu ; $cpu(pagermemory) \rightarrow pinnedmemory \rightarrow GPUmai$

if pehle se hi pinned m rakhe then fast hoga

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, pin_memory=True)
```

https://colab.research.google.com/drive/17hjS23CgFqIZjuB2XKt2u7pVu1MpD2Qr?usp=sharing#scrollTo=0oISAHnU5GnT - gpu optimised code (88% accuracy on test but 100 on train data !!)

💡 Test→88%   Train→100%

if train-test > 10% then the model is OVERFITTED
- Doesnt give good results on unseen data

OPTIMISING THE NN (reducing the overfitting)

- Various solutions -
  1. adding more data (jitna data dikhega utna biases kam honge)

  2. reducing the complexity of NN arch (many hidden layers) - apna theek h

  3. Regularisation - loss function + penality (tries to minimize both loss and penality) - L1 and L2 regularisation (L2 is more used in ML)

  3. Dropouts - randomly turn off few layers

  4. Data augmentation - flip , rotate, tilt - alag alag variation -
  Works like a charm when using CNN tho

  5. Batch Normalisation - vaise to used for stablising training - have effect on regularisation also

  6. Early stopping - pehle epochs m hi rok do if loss isnt getting better

- We are going to apply reg , droputs , batch normalisation

DROPOUT(nn.Dropout) https://www.youtube.com/watch?v=gyTlcHVeBjM&t=726s