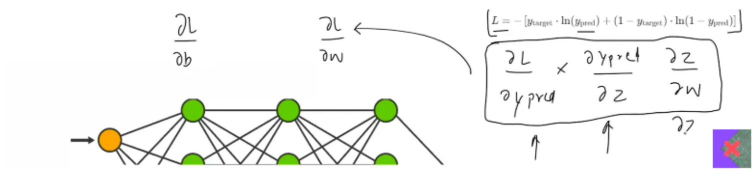
- Koi bhi NN is a nested functions.
- compute gradients of the loss , delL/delW and delL/delB



- Increasing complexity of these functions and nested functions , calculating gradient can be diffuculat - afet this gradients are updated - adjust parameter using an opti algo (grad descent)
- To solve this prob AUTOGRAD is introduced - saare derivatives ko calc - privdes automatic diff ing for tensor operations - enables grad computation using opti algo like grad descent.

> https://colab.research.google.com/drive/1sI52bmFHSMUELEbQ94NVIvIS3NPthJrS

Pytorch Training Pipelines (will make a small NN)

- Going to make a single neuron NN on breast cancer dataset.
- Loading dataset , preprocessing , training process(making model, forward pass, loss calculation , backprop, parameters update (using eg grad desc) , model eval

  https://colab.research.google.com/drive/1DD9whhBlgtXmigdCB3pm9kIK5TvdAliw

---

NN Module (imp) - torch.nn as nn

- Basically offers pre built layers , loss fns, activation fns and other utils.
- Last learn training pipeline ko imporve karenge - pytroch ka nn module and torch.optim modules - makes work v easy
- What will improve - manually created wts and bias and their interaction - replaced by NN module (hv functionality to crate neurons and layers) ; manual written loss function with inbuilt ; also activation function using nnmodu
- will use torch.optim instead of manully updating weights.
- Layers - nn.Linear , nn.Conv2D , nn.LSTM (recurrent layers)
- Act fns - nn.ReLU , nn.Sigmoid , nn.Tanh

- Loss - nn.CrossEntropyLoss , nn.MSELoss , nn.NLLoss
- contriner modules - nn.Sequential (o stack layers)
- other utils - regularisation and dropout.

> making a simple 5 feature binary classifier - https://colab.research.google.com/drive/15FsvB_yU0-wgcdl0vzMINshEFtyLFyHi

Rewriting the manual code with nn.Module : https://colab.research.google.com/drive/13hQ0Mmk3gxrLry-tMFhN50Bymk_PBBLK

- torch.optim provides variety of optim algo to update the parameters of the model - LR schdeduling and weight decay all very easy.
  - model.parameter() method is an iterator over all the trainable parameters (wts and b) in a model - optim usese sthese to compute grads and update the wts and b.

Dataset and DataLaoder class (pytorch)

```
from torch.utils.data import Dataset, DataLoader
```

- Biggest Flaw in prev codes - we are using **batch grad descent** - to updaet the parameter we are passing the whole dataset - loss - update parameters.

  - v memory ineff (pura data in RAM , imagine lakhs of image classification)
  - not very good convergence - pura data dekh rhe then ek baar data update kr rhe - have to update paramtetes more frequently (like SGD)
  - rather than loading entire dataset and uspr grad descent kro - instead load data in batches - x rows ko pass kro - loss-grad calc - grad descent - next batch - again.
  - called mini batch grad descent

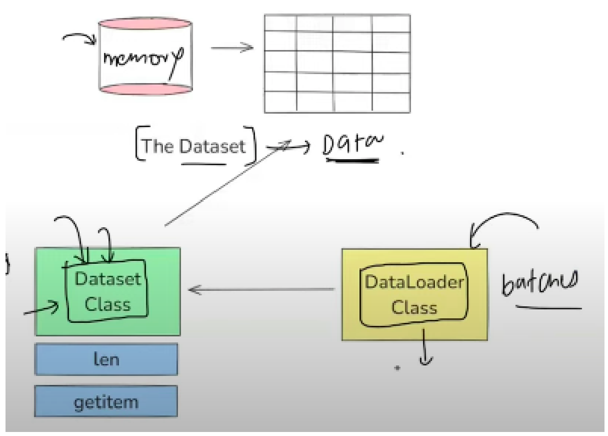
- Simply using a for loop inside epoch loop isnt the greatest choice (for start_idx in range(0, n_sampels, batch_size) :

  - Problems :
  1. No standard interface for data - X train y train se batches - sometimes data isnt easily avaliabel - eg data in diff folders - data ko lana

  2. no easy way to apply tranformation
  3. Shuffling and sampling (pehle dogs fir cats - shuffling is better - sampling is random batch of batch_size)
  4. batch managment and parallelization (multiple batches parellelly extract kaise kre)

```python
batch_size = 32
epochs = 25
n_samples = len(X_train_tensor)

for epoch in range(epochs):
    # Simply loop over the dataset in chunks of `batch_size`
    for start_idx in range(0, n_samples, batch_size):
        end_idx = start_idx + batch_size
        X_batch = X_train_tensor[start_idx:end_idx]
        y_batch = y_train_tensor[start_idx:end_idx]

        # Forward pass
        y_pred = model(X_batch)
        loss = loss_function(y_pred, y_batch.view(-1, 1))

        # Update step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch: {epoch+1}, Loss: {loss.item()}")
```


- To solve these problems pytorch gives us Datasat and DataLoader class.
- How they work ?? :
  - They decouple how data is loaded and how data is used for training.



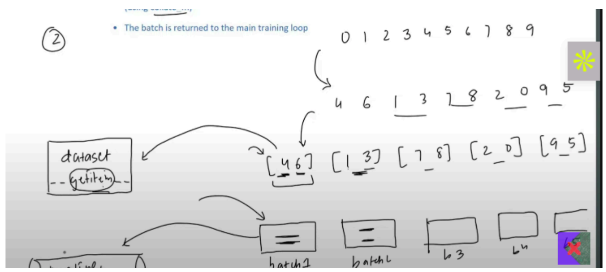
- **DataSet Class** knows where the data is in the memory and can load rows.
  - DataLoader class handles the batch making - decided the number of rows per batch
  - Dataloader class asks for rows from the DataSet class.
- CustomDataset(Dataset): - inherit from DataSet class
  - DataSet class is a abstract class - essentially a blueprint - whe u create a custom datast , you decide how data is loaded nd returned.
  - you have to make three classes inside it.
    - constructor("__init__(self, featues , labels)")- how data shld be read - pd.read_csv() - or images

load - memory se data load

- “__len__()” returns total number of samples (rows)
- “__getitem__(index)” - returns the data and label at the given index - row nikaal kr dega

- **DataLoader Class: handles batching shuffling and parrelel loading**
  -
  step 1 : at start of epoch(if shuffle = True) - shuffles the indices with help of sampler

  - step 2 : It divides the indicies into chunks of batch_size(eg 2) [4 6] [1 3] [5 6]...
  - step 3: for each index in the chunk , data samples are fecthed frm the DataSet object (getitem)
  - step4 : the samples are then collected and combined into a batch using collate function collate_fn - combines the rows of indicies in a batch
  - batch is returned to the main training loop



> 💡 note about data transformation-
>
> Inside **getitem** before return add your transformation(resizing , BnW, lemm, stopword etc)

> 💡 Parallelisation - the above image seems sequential - DataLoader mai workers ka concept hota h - add several workers .
>
> go thru this for workers :
> https://drive.google.com/file/d/1fILm74_ytGv5O06ZZEutD6cyd1mvL-Yj/view

> Note about the Sampler - in the dataloaser determines the strat for selecting samples from the dataset during data loading. - how indicies are choosen for each batch.
>
> - SequentialSampler - samples in the order they appear , when shuffle= False
>
> - RandomSampler - randomaly without replacement default when shuffle= True