Creating Models

- **Sequential - `torch.nn.Sequential()`**
  - Arguments
    - Takes in the layers, the activation functions in the order that you want them to run
- **Module -**
  -

Optimization Algorithms

- Optimizers define how the gradients in your model are computed (e.g. using GD, SGD, Adam, etc)
- torch.optim.SDG(model.parameters(), lr=3e-4), momentum=0.9)
- The optimizer takes as argument model parameters and it updates them using **`.step()`** based on the current gradient, which is stored in **`.grad`** of each parameter

Criterion

- Your loss functions
- `criterion = torch.nn.CrossEntropyLoss()`
  - Criterion is a callable function
- `loss.backward()` computes and accumulates the gradient by addition for each parameter
- Loss knows which parameters to update .grad for because when it is created with `required_grad==True`, it is added to the computation graph as a leaf
- **This works because torch.tensors (vs. numpy arrays) have an additional LAYER, a computational graph leading to the associated matrix**

Steps

1. Define your model class (with constructor and forward()
2. Set hyperparameters
3. Create Loader
4. Create an instance of your model
5. Define a criterion
6. Define an optimizer
7. Have a training loop, for each epoch, for each batch:
   a. Move data to device
   b. Move targets to device
   c. Forward Prop:
      - Compute predictions
      - Compute loss with the criterion
   d. Backward Prop
      - optimizer.zerograd()
      - loss.backward()
8. Have a check_accuracy loop, where we test our accuracy on training set and on test set at the end