Department of Mathematics
and Computer Science

Deep Learning

Introduction to PyTorch Part 2

Lukas Galke Poech & Gianluca Barmina & Mogens H. From – Slides from Lucas Dyssel

Fall 2023

SDU

Deep Learning Workflow

- **Defining the problem and assembling a dataset**
  - What will your input data be?
  - What type of problem are you facing? Classification? Regression?
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- **Choosing a measure of success**
  - How do you measure if the model is successful
  - Not to be confused with the loss function which is something we use measure how close we are to the goal achieving our goal
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- **Decide on the evaluation protocol**
  - Hold-out validation
  - Cross-validation
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- **Prepare your data**
  - Clean data
  - Normalize data
  - Data augmentation
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- **Define a model**
  - Activation functions
  - Loss function
  - Optimization
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- **Scaling up: Make the model overfit**
  - Add layers
  - Make the layers bigger
  - Train for more epochs
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- **Regularizing your model and tuning you hyperparameters**
  - This will take the most time
  - Add dropout
  - Try different architectures: add or remove layers
  - Add L1 and/or L2 regularization
- Finalize your final model

Regularization

```
class MyNetwork(nn.Module):
    def __init__(self, input_size,
output_size):
        super(NNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 16)
        self.fc2 = nn.Linear(16, output_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.softmax(self.fc2(x))
        return x

net = MyNetwork(input_size=10, output_size=10)

######### INSIDE OF TRAINING LOOP #########
loss = criterion(outputs, targets)
loss += L2_lambda * torch.norm(net.fc1.weight,
p=2)
```

- **MyNetwork** is a standard neural network
- Adding the L2 regularization term to the loss function should happen inside of the training loop:
  - **L2_lambda** is a hyperparameter which determines the strength of of L2
  - **torch.norm(MyModel.fc1, p=2)** calculates the L2 norm (Euclidian norm) of the weights of the first fully connected layer
    - **p=1** is L1 regularization and **p=2** is L2 regularization
  - **loss += L2_lambda * torch.norm(MyModel.fc1, p=2)** the combined regularization term is added to the loss, penalizing large weight

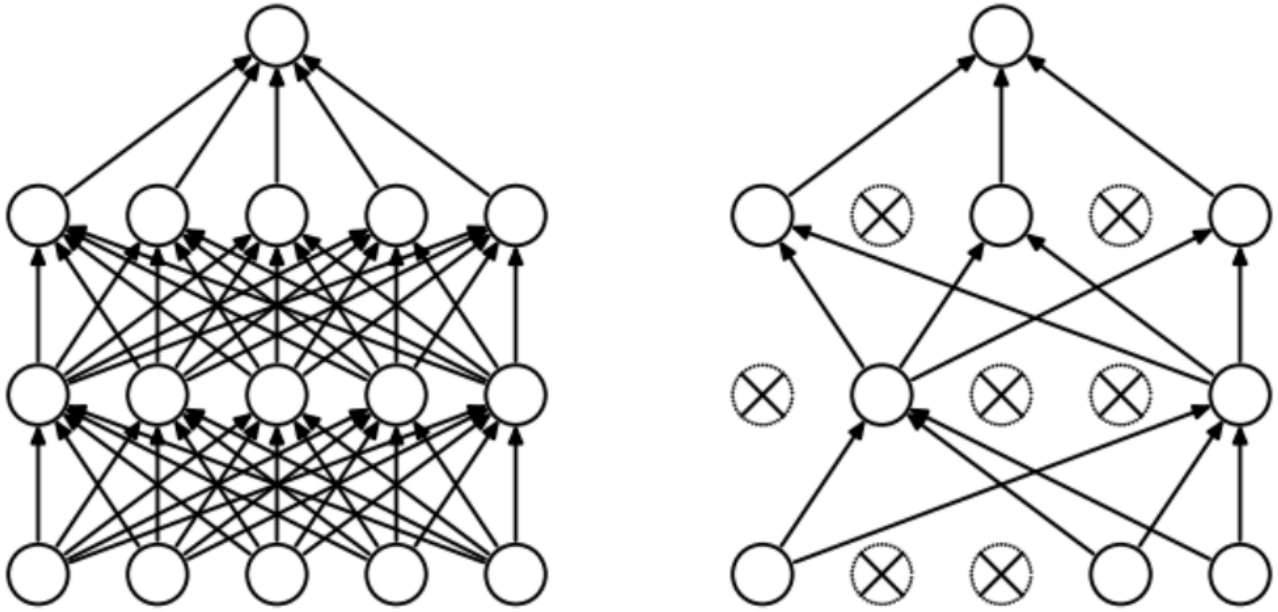
Dropout

- **p (float)** – Probability of an element to be zeroed. Default: 0.5
- **inplace (bool)** – If set to True, will do this operation in-place. Default: False

```
torch.nn.Dropout(p=0.5, inplace=False)

# Example
class NNetwork(nn.Module):
    def __init__(self):
        super(NNetwork, self).__init__()
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        # Dropout is used inbetween two layers
        x = self.dropout(x)
```



(a) Standard Neural Net (b) After applying dropout.

Budhiraja, A. (2018, March 6). *Learning Less to Learn Better—Dropout in (Deep) Machine learning*. Medium. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- **Finalize your final model**
  - Save and distribute the model