Deep Learning

Exercise 5: Binary and Multi-class Classification

Room: **BIN-1-B.01**
Instructor: Manuel Günther
Email: guenther@ifi.uzh.ch
Office: AND 2.54

Friday, March 26, 2021

Outline

Outline

1. PyTorch
   - Local Installation
   - Datatype `torch.tensor` and its Operations
   - Building a Network
   - The Training Loop

Local Installation

Cuda Installation

- Install Cuda driver (if you have a Cuda-enabled GPU)
  - → List of supported GPUs: https://developer.nvidia.com/cuda-legacy-gpus
  - → Install driver for your OS https://www.nvidia.com/Download

PyTorch Installation

- Install from `pytorch` channel via Conda:

  ```
  conda install pytorch torchvision cudatoolkit -c pytorch
  ```

- Extensive documentation and tutorials:
  - → Tutorials: https://pytorch.org/tutorials
  - → Reference documentation: https://pytorch.org/docs/stable

Everything already installed in Google Colaboratory

Datatype `torch.tensor` and its Operations

What is a `torch.tensor`

- A multi-dimensional array that
  - → Has a specific datatype: `torch.float`, `torch.long`, ...
  - → Can be in CPU and GPU memory: `device="cpu"`
  - → Supports automatic differentiation: `backward` function and `grad` member
  - → Needs to be `detach`'ed from `grad` for processing with `numpy`

`torch.tensor` Creation

- Construct from data: `torch.tensor([[0,1],[2,3]])`
- Construct from `numpy`: `torch.as_tensor(data)`
- Initialize: `torch.empty(shape)`, `torch.zeros(shape)`, `torch.ones(shape)`
- Parameters `dtype` and `requires_grad`

Datatype `torch.tensor` and its Operations

Operations on `torch.tensor`'s

- Math operations: `+`, `+=`, `*`, `*=`, `/`, `**`
- Trigonometric: `torch.sin`, `torch.acos`, `torch.exp`, `torch.log`
- Reduction operations: `torch.max`, `torch.mean`, `torch.std`
- Matrix operations: `torch.inner`, `torch.outer`, `torch.matmul`
- **All operations support automatic differentiation**
  → More details on this in next lecture
- Use `with torch.no_grad():` block to avoid computing gradients
  → Particularly useful for validation and testing

Building a Network

Types of Layers (for Today)

- Fully connected layer `torch.nn.Linear`
  - → `in_features` $= D$; no artificial bias neuron required
  - → `out_features` $= K$
  - → `bias = True`
- Activation functions:
  - → `torch.nn.Sigmoid`, `torch.nn.Tanh`, `torch.nn.Softmax`

Building a Network by Connecting Layers

- Simplest variant via `torch.nn.Sequential(layer1, layer2, layer3, ...)`
- Other variants will follow

The Training Loop

Training Networks in PyTorch

- PyTorch training loops are very verbose
  - → Many things to be implemented, but allows very flexible code
- Typical training loop:
  1. compute network output on training data
  2. compute loss for training data
  3. perform gradient descent on training data
  4. optionally: compute training accuracy
- Typical validation loop:
  1. compute network output on validation data
  2. optionally: compute loss for validation data
  3. compute validation accuracy

The Training Loop

Loss Functions

- Compare output of network with target values
- `torch.nn.MSELoss`: $\mathcal{J}^{L_2}$
  - $\rightarrow$ Used mainly for regression tasks
- `torch.nn.BCEWithLogitsLoss`:
  - $\rightarrow$ Combines $\mathcal{J}^{\mathrm{BCE}}$ with σ activation
  - $\rightarrow$ Works directly on logit values; no σ activation required in network
- `torch.nn.CrossEntropyLoss`:
  - $\rightarrow$ Combines $\mathcal{J}^{\mathrm{CCE}}$ and SoftMax activation
  - $\rightarrow$ Works directly on logit values; no SoftMax activation required in network

The Training Loop

Learning Strategy

- `torch.optim.SGD` for training
  - → `params`: All optimizable weights
  - → `lr`: Learning rate η
  - → `momentum`: μ

Example Instantiation

```
network = Network()
loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(
  params=network.parameters(),
  lr=0.01, momentum=0.9
)
```

Example Training Loop

```
for epoch in range(epochs):
  # DO NOT FORGET:
  optimizer.zero_grad()
  Z = network(X_train)
  J = loss(Z, T_train)
  J.backward()
  # perform parameter update
  optimizer.step()

  # compute test accuracy
  with torch.no_grad():
    Z = network(X_val)
    acc = accuracy(Z, T_val)
```

Outline

2. Binary and Categorical Classification
   - Datasets
   - Network Training
   - Training and Evaluation

Datasets

Binary Classification of Spam Emails

- Dataset from UCI
- Inputs: 58 attributes
  - → 48 relative number of occurrences of specific key words
  - → 6 relative number of occurrences of specific key characters
  - → 3 counts according to capital letters
- Standardization of data required
- Output: spam (1) or not spam (0)

Dataset URL

http://archive.ics.uci.edu/ml/datasets/Spambase

Datasets

Categorical Classification of Wine Types

- Input: chemical analysis of wines
  - $\rightarrow$ Alcohol, acid, magnesium, color intensity, ...
- Output: one of three different wine types
- Simple dataset, easy to solve

Dataset URL

http://archive.ics.uci.edu/ml/datasets/wine

Datasets

Task 1: Dataset Loading

- Download dataset from UCI
- Extract from zip file in Python
- Read CSV file `wine.data` or `spambase.data`
  - → All values are numerical
- Compute target tensors T
  - → Last column for spam, first column for wine
  - → Convert to $\mathbb{R}^{N\times 1}$ for spam, $\mathbb{N}^{N}$ for wine
- Compute input tensors X using remaining columns

Test 1: Check Datasets

- Check that shapes and data types of X and T are as expected

Datasets

Task 2: Training and Validation Split

- Split (X, T) into training (80 %) and validation (20 %)
  $\rightarrow$ Four different sets: (X_t, T_t, T_v, T_v)
- What do we need to consider?

Task 3: Input Data Standardization

- Compute mean and standard deviation from input data
  $\rightarrow$ Which dataset should be used here?
- Standardize X_t and X_v with this mean and std

Network Training

Task 4: Network Implementation

- Create two-layer fully-connected network
  - $\rightarrow$ Input dimension D, number of hidden neurons K and output dimension O
- Add one activation function (tanh) between these layers
- What kind of output activation do we need?
- You can make use of `torch.nn.Sequential`

Network Training

Task 5: Accuracy Computation

- Implement function to compute accuracy
  $\rightarrow$ Take network output Z and targets T as input
- Split implementation based on task
  $\rightarrow$ Binary vs. categorical classification

Test 2: Test Accuracy Implementation

- Design input and target data to test accuracy function
- Test both binary and categorical classification

Network Training

Task 6: Training Loop

- Implement a function to perform gradient descent
  - → Provide all necessary parameters
- Use `torch.optim.SGD` as optimizer
  - → We will use gradient descent here, but SGD works well
- Loop over 10'000 epochs:
  - → Compute training loss and accuracy
  - → Perform gradient descent step/parameter update
  - → Compute validation loss and accuracy
- Return losses and accuracies for all epochs

Training and Evaluation

Task 7: Plotting Function

- Take losses and accuracies from training and validation
- Plot losses together and accuracies together in two plots

Task 8: Binary Classification

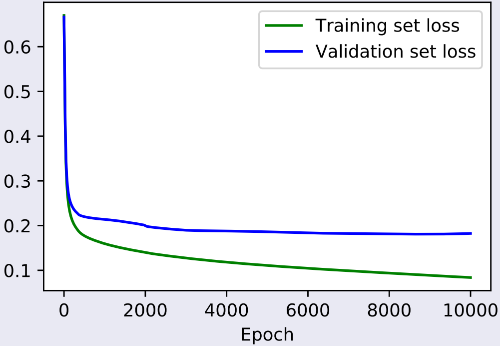
- Load spam dataset, split into train/val, standardize inputs
- Instantiate network with appropriate values for D, K and O
- Use binary cross-entropy loss function
- Train network and plot accuracies and losses

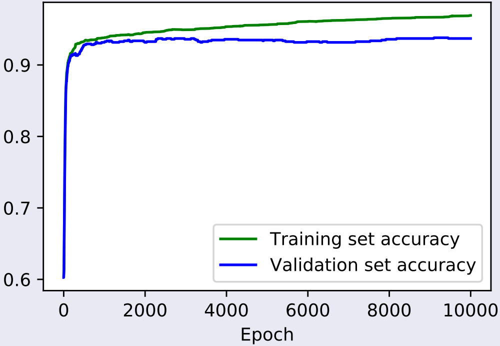
Task 9: Categorical Classification

- Repeat with wine dataset, use categorical cross-entropy loss

Training and Evaluation

Exemplary Binary Classification Results





Training and Evaluation

Exemplary Categorical Classification Results