Deep Learning

Exercise 12: RBF Networks

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

Friday, May 27, 2022

Outline

Outline

Custom Layers with Learnable Parameters

Custom Layers in PyTorch

- Derive from `torch.nn.Module`

In Constructor

- Call base class constructor
- Define `Parameter` as members
  - → Wraps a `torch.Tensor`
  - → Will be found in `parameters()`
- Initialize parameters

In Forward

- Parameter acts like tensor
- Utilize only PyTorch functions

Functionality in `torch.nn.init`

- Normal distributed:
  `torch.nn.init.normal_`
- Uniformly distributed
  `torch.nn.init.uniform_`
- Constant:
  `torch.nn.init.constant_`
  `torch.nn.init.ones_`
- Xavier:
  `torch.nn.init.xavier_normal_`

Custom Layers with Learnable Parameters

Example Custom Layer Implementation

```
class MyLayer(torch.nn.Module):
  def __init__(self, K, D):
    # base class constructor
    super(MyLayer, self).__init__()
    # instantiate parameter, for example
    self.param = torch.nn.Parameter(torch.empty((K, D)))
    # initialize parameter, for example
    torch.nn.init.normal_(self.param, 0, 1)

  def forward(self, x):
    # utilize parameter, for example
    return torch.matmul(self.param, x)
```

Radial Basis Function Layer

RBF Layer Implementation

- Parameter $\mathbf{W}$
  - $\rightarrow$ Requires suitable Initialization
- Distance-based activation

$$a_r = \|\vec{w}_r - \vec{x}\|$$

- Problem: matrix shapes
  - $\rightarrow$ Weight matrix $\mathbf{W} \in \mathbb{R}^{R \times K}$
  - $\rightarrow$ Input matrix $\mathbf{X} \in \mathbb{R}^{B \times K}$
  - $\rightarrow$ Activation $\mathbf{A} \in \mathbb{R}^{B \times R}$

Batch Implementation

- Bring $\mathbf{X}$ and $\mathbf{W}$ to $\mathbb{R}^{B \times R \times K}$
  - $\rightarrow$ Logical copies of $\mathbf{X}$ and $\mathbf{W}$
  - $\rightarrow$ Add singular dimension: `tensor.unsqueeze(dim=...)`
  - $\rightarrow$ Logical (no physical) copies: `tensor.expand(B,R,K)`
- Compute distances:

$$\mathcal{A} = (\mathcal{W} - \mathcal{X})^2 \in \mathbb{R}^{B \times R \times K}$$

- Sum over dimension K
  - $\rightarrow$ $a_{b,r} = \sum_{k=1}^{K} a_{b,r,k}$

Outline

2. Radial Basis Function Network
   - Dataset
   - Radial Basis Function
   - Visualization

Dataset

Task 1: Dataset

- We use the default MNIST training and validation sets
  → Select appropriate batch sizes

Radial Basis Function

Task 2: Radial Basis Function Layer

- Implement a layer in PyTorch to compute activation $\mathbf{A} \in \mathbb{R}^{B\times R}$
- Instantiate weight parameter $\mathbf{W} \in \mathbb{R}^{R\times K}$ and initialize to $[-2, 2]$
- Implement `forward` function using tensor operations

Task 3: Radial Basis Function Activation

- Implement activation function as layer in PyTorch
- Learnable parameters `sigma2=`$2\vec{\sigma} \odot \vec{\sigma}$ with $\vec{\sigma} \in \mathbb{R}^K$

  $\rightarrow$ Initialize constantly as 1
- Implement Gaussian activation:

$$\vec{h} = \mathcal{N}_{\vec{0},\vec{\sigma}}(\vec{a}) = e^{-\frac{\vec{a}\odot\vec{a}}{2\vec{\sigma}\odot\vec{\sigma}}}$$

Radial Basis Function

Test 1: RBF Layer and Activation

- Instantiate RBF layer and Activation for $K = 4$ and $R = 10$
- Create some data in batch size $B = 12$
- Forward input through RBF layer and activation
  - $\rightarrow$ Make sure that the implementation does not raise exceptions
  - $\rightarrow$ Check size of output

Radial Basis Function

Task 4: Radial Basis Function Network

- Implement convolutional network similar to Assignment 8
  - → 2 convolutional layers with Q_1 and Q_2 channels
  - → 2 times maximum pooling
  - → 2 times ReLU activation
  - → One fully-connected layer to produce K outputs (no ReLU here)
- One radial basis function layer with K inputs and R outputs
- One radial basis function activation
- One fully-connected layer with R inputs and $O = 10$ outputs
- Output of `forward` is logits and deep features (K-dimensional)

Radial Basis Function

Task 5: Training and Validation Loop

- Instantiate RBF network:
  - $\rightarrow$ $Q_1 = 32$, $Q_2 = 64$, $K = 2$, $R = 100$, and $O = 10$
  - $\rightarrow$ We want later to visualize 2D features
- Instantiate categorical cross-entropy loss
- Instantiate optimizer of your choice
- Train for 20 epochs and report validation set accuracy

Visualization

Task 6: Deep Feature Extraction

- Iterate through validation set
- Extract 2D features and store in separate lists per target class

Task 7: Deep Feature Visualization

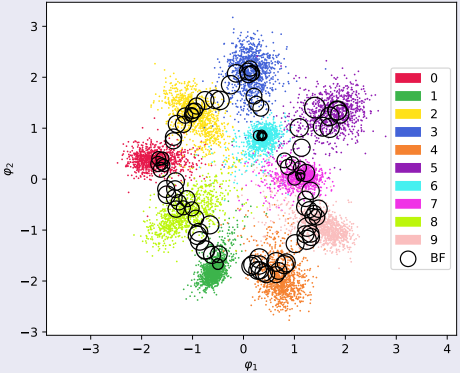
- Obtain 10 different colors: one per target class
- Plot a dot for each sample in 2D feature space (via `pyplot.scatter`)

Task 8: Basis Function Visualization

- Plot a black circle for each learned basis function
  - → Maybe use `o` marker with size according to `sigma2`
  - → Might need to be scaled with large constant for nice results

Visualization

Example with $K = 100$



Example with $K = 10$