Deep Learning

Exercise 12: RBF Networks

Instructor: Manuel Günther
Email: guenther@ifi.uzh.ch
Office: AND 2.54

Friday, May 28, 2021

Outline

Outline

Radial Basis Function Layer

Custom Layer in PyTorch

- Derive class from `torch.nn.Module`
  → Same as for networks
- Call base class constructor

Learnable Parameters

1. Create a `torch.tensor`
2. Wrap with `torch.nn.Parameter`
3. Store in your model `self`
4. Some python magic
5. Parameter in `parameters`

Parameter Instantiation

```
class MyLayer(torch.nn.Module):
  def __init__(self, ...):
    # base class constructor ...
    self.param = torch.nn.Parameter(
          torch.tensor(...))

class MyNetwork(torch.nn.Module):
  def __init__(self,...):
    # base class constructor ...
    self.mylayer = MyLayer(...)

network = MyNetwork(...)
network.parameters()
```

Radial Basis Function Layer

RBF Layer

- Two parameters: $\mathbf{W}$ and $\vec{\sigma}$
- Distance-based activation

$$a_k = \|\vec{w}_k - \vec{x}\|$$

- Activation function:

$$h_k = \mathcal{N}_{0,\sigma_k}(a_k) = e^{-\frac{a_k^2}{2\sigma_k^2}}$$

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

Batch Implementation

- Bring $\mathbf{X}$ and $\mathbf{W}$ to $\mathbb{R}^{B \times K \times D}$
  - $\rightarrow$ Logical copies of $\mathbf{X}$ and $\mathbf{W}$
- Compute distances:

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

$$\rightarrow a_{b,k,d} = \left(w_{k,d} - x_d^{[b]}\right)^2$$

- Sum over dimension D

$$\rightarrow a_{b,k} = \sum_{d=1}^{D} a_{b,k,d}$$

- $\mathbf{H} = e^{-\frac{\mathbf{A}}{2\vec{\sigma}^2}}$ broadcastable

Radial Basis Function Layer

RBF Layer Initialization

- $\mathbf{W}$ and $\vec{\sigma}$ need initialization
- Default initialization inappropriate
- Good values: $\mathbf{W} \sim \mathcal{N}_{0,1}$ and $\vec{\sigma} = \vec{1}$
  - $\rightarrow$ Better initialization possible
- See module `torch.nn.init`

RBF Layer Implementation

- Initialization `__init__` see above
- Processing: `forward(self, X)`
  - $\rightarrow$ Receives batch `X`
  - $\rightarrow$ Returns batch `H`

Logical Copies in PyTorch

- Add singular dimension:
  `tensor.unsqueeze(dim=...)`
- Logical (no physical) copies:
  `tensor.expand(B,K,D)`

Implementation

```
class RBFLayer(torch.nn.Module):
  def __init__(self):
    # Parameters

  def forward(self, X):
    # expand X and W
    # compute distance
    # compute activation
    # Gaussian of distances
```

Outline

2 MNIST Classification

MNIST Classification

MNIST Classification

- `Dataset` and `Dataloader`
- Classify 10 classes

Task 1: Deep RBF Network

- Convolutional layers
  - $\rightarrow$ Pooling and `Sigmoid`/`ReLU`
  - $\rightarrow$ `BatchNorm` where appropriate
- Fully-connected layer $K^{(\mathrm{FC})} = 2$
- RBF layer (own) $K^{(\mathrm{RBF})} = 100$
- Fully-connected layer $O = 10$

Task 2: Network Training

- Categorical cross-entropy loss
- Stochastic gradient descent
- Low learning rate ($\eta \approx$ `1e-4`)
  - $\rightarrow$ Despite batch norm!
- Train for 100+ epochs
- Measure validation accuracy
  - $\rightarrow$ Accuracy $\sim$ 10% random guess
  - $\rightarrow$ Accuracy $>$ 90% good
- Save best validation model

Outline

Visualization

Task 3: Plot Features

- Test set images
- Deep features **before** RBF
  - → 2-element FC features
- Color according to target
- Scatter plot in 2D space
  - → `pyplot.scatter()`

Task 4: Plot Centers

- Centers from RBF layer
- Plot into 2D space as +
  - → Change size with variance

Learned 2D Features and Centers