Deep Learning

Exercise 8: Open-Set Learning

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

Friday, April 30, 2020

Outline

Outline

1. Open-Set Training
   - MNIST Split
   - Network Definition
   - Loss Functions

MNIST Split

MNIST Dataset

- MNIST total 10 classes
  - → 10 different digits
- Split into 3 categories:
  - → 4 known classes, e.g.: 4,5,8,9
  - → 4 known unknown classes, e.g.: 0,2,3,7
  - → 2 unknown unknown classes, e.g.: 1,6
- Train known and known unknown
- Evaluate all three categories

Task 1: Target Vectors

- One-hot targets for knowns:
  - → $4 \Rightarrow$ (1,0,0,0)
  - → $5 \Rightarrow$ (0,1,0,0)
  - → $8 \Rightarrow$ (0,0,1,0)
  - → $9 \Rightarrow$ (0,0,0,1)
- Equal targets for unknowns:
  - → $0 \Rightarrow$ (.25,.25,.25,.25)
  - → $2 \Rightarrow$ (.25,.25,.25,.25)
  - → $3 \Rightarrow$ (.25,.25,.25,.25)
  - → $7 \Rightarrow$ (.25,.25,.25,.25)

Datasets Loaders

Example MNIST Dataset

```
# obtain datasets
transform = torchvision.transforms.ToTensor()
train_set = torchvision.datasets.MNIST(
              root="/temp/MNIST",
              train=True, download=True,
              transform=transform
            )
test_set = torchvision.datasets.MNIST(
              root="/temp/MNIST",
              train=False, download=True,
              transform=transform
            )
# loaders
train_loader = torch.utils.data.DataLoader(
    train_set, shuffle=True, batch_size=64
)
test_loader = torch.utils.data.DataLoader(
    test_set, shuffle=False, batch_size=100
)
```

While Training

```
for inputs, targets in train_loader():
    # remove unknown unknown inputs
    ...
    # convert targets to one-hot
    targets = convert_target(targets)
    # forward inputs
    logits = network(inputs)
    # call loss function
    J = loss(logits, targets)
    ...
```

Network Definition

Convolutional Network

- Adaptation from Exercise 6
- 2 convolutions with padding
  - → 1 input, 32 outputs, 5×5 kernels
  - → 32 input, 32 outputs, 5×5 kernels
  - → 2×2 Maximum pooling after each
  - → Sigmoid activation after each
- Batch normalization (important)
- Linear layer with 50 outputs
- Sigmoid activation
- Linear layer with 4 outputs

Task 2: Network Implementation

```
class Network():
    def __init__(self):
        self.conv1 = torch.nn.Conv2d(...)
        self.conv2 = torch.nn.Conv2d(...)
        self.pooling = torch.nn.MaxPool2d(...)
        self.activation = torch.nn.Sigmoid()
        self.bn = torch.nn.BatchNorm2d(...)

        self.fc1 = torch.nn.Linear(7*7*32, 50)
        self.fc2 = torch.nn.Linear(50, 4)

    def forward(self, input):
        ...
        return logits
```

Loss Functions

Default Implementation

- `torch.nn.CrossEntropyLoss` requires target class t
  $\rightarrow$ We have target vector $\vec{t}$

Easiest Implementation

$$\mathcal{J}^{\text{CE}} = -\sum_{o=1}^{O} t_o \ln y_o$$

Other Implementation

$$\mathcal{J}^{\text{CE}} = -\sum_{o=1}^{O} t_o z_o + \frac{1}{O} \ln \sum_{o=1}^{O} e^{z_o}$$

Task 3: Loss Function via Autograd

- To implement:

```
def loss(logits, targets):
```

- $\ln \vec{y}$ from logits
  $\rightarrow$ `torch.log_softmax`
- $\ln \sum_{o=1}^{O} e^{z_o}$
  $\rightarrow$ `torch.logsumexp`

\+ Remember that you get a batch
  $\rightarrow$ Check the `dim` parameter

- To return: a `torch.tensor` with one element (check `torch.mean`)

Outline

2 Evaluation

Evaluation

During Training

- Compute average confidence
  - $\rightarrow$ y_t for known samples
  - $\rightarrow$ $1 - \max\limits_o y_o + \frac{1}{O}$ for unknowns

After Training

- Evaluate confidences $\vec{y}$
  - $\rightarrow$ False Positive Rate for unknown
  - $\rightarrow$ Correct Classification Rate for known
- Use fixed threshold $\tau = 0.5$

False Positive Rate (FPR)

$$\frac{\left|\left\{x^{[n]} \mid t^{[n]} = 0 \wedge \max\limits_{1\leq o \leq O} y_o^{[n]} \geq \tau\right\}\right|}{\left|\{x^{[n]} \mid t^{[n]} = 0\}\right|}$$

Correct Classification Rate (CCR)

$$\frac{\left|\left\{x^{[n]} \mid t^{[n]} > 0 \wedge \arg\max\limits_{1\leq o \leq O} y_o^{[n]} = t^{[n]} \wedge y_{t^{[n]}}^{[n]} \geq \tau\right\}\right|}{\left|\{x^{[n]} \mid t^{[n]} > 0\}\right|}$$

Task 4

- Train network for 20 epochs
- Evaluate trained network

Outline

3 Autograd Implementation
- Autograd Extension in PyTorch

Autograd Extension in PyTorch

Loss and Derivative of $\mathcal{J}^{\text{CE}}$

$$\mathcal{J} = -\sum_{o=1}^{O} t_o \ln y_o \qquad \frac{\partial \mathcal{J}}{\partial z_o} = y_o - t_o$$

Implementation in PyTorch

- Implement a loss as `pytorch.autograd.Function`
- Online documentation
- Two static functions:
  - $\rightarrow$ `forward(ctx, params)`
  - $\rightarrow$ `backward(ctx, result)`
- Store elements in `ctx`

Optional Task: Autograd Function

```
class CELoss(torch.autograd.function):
    @staticmethod
    def forward(ctx, logits, targets):
        # save for backward
        ctx.save_for_backward(logits, targets)
        # implement loss function
        J = ...
        return J

    @staticmethod
    def backward(ctx, J):
        # get stored values
        logits, targets = ctx.saved_tensors
        # compute derivatives
        gradient = ...
        # gradient for each input of forward
        return gradient, None

# assign loss function
loss = CELoss.apply
```