

COMP90051

Statistical Machine Learning

Workshop Week 7

Xudong Han

https://github.com/HanXudong/COMP90051_Workshops

Pytorch

- [Pytorch](#) is an open-source Python library designed for fast **matrix computations** on CPU/GPU. This includes both standard linear algebra and deep learning-specific operations. It is based on the neural network backend of the [Torch library](#). A central feature of Pytorch is its use of Automatic on-the-fly differentiation ([Autograd](#)) to compute derivatives of (almost) all computations involving tensors, so we can make use of gradient-based updates to optimize some objective function. In this workshop we will introduce some fundamental operations in Pytorch and reimplement the Perceptron and logistic regression classifiers in Pytorch.

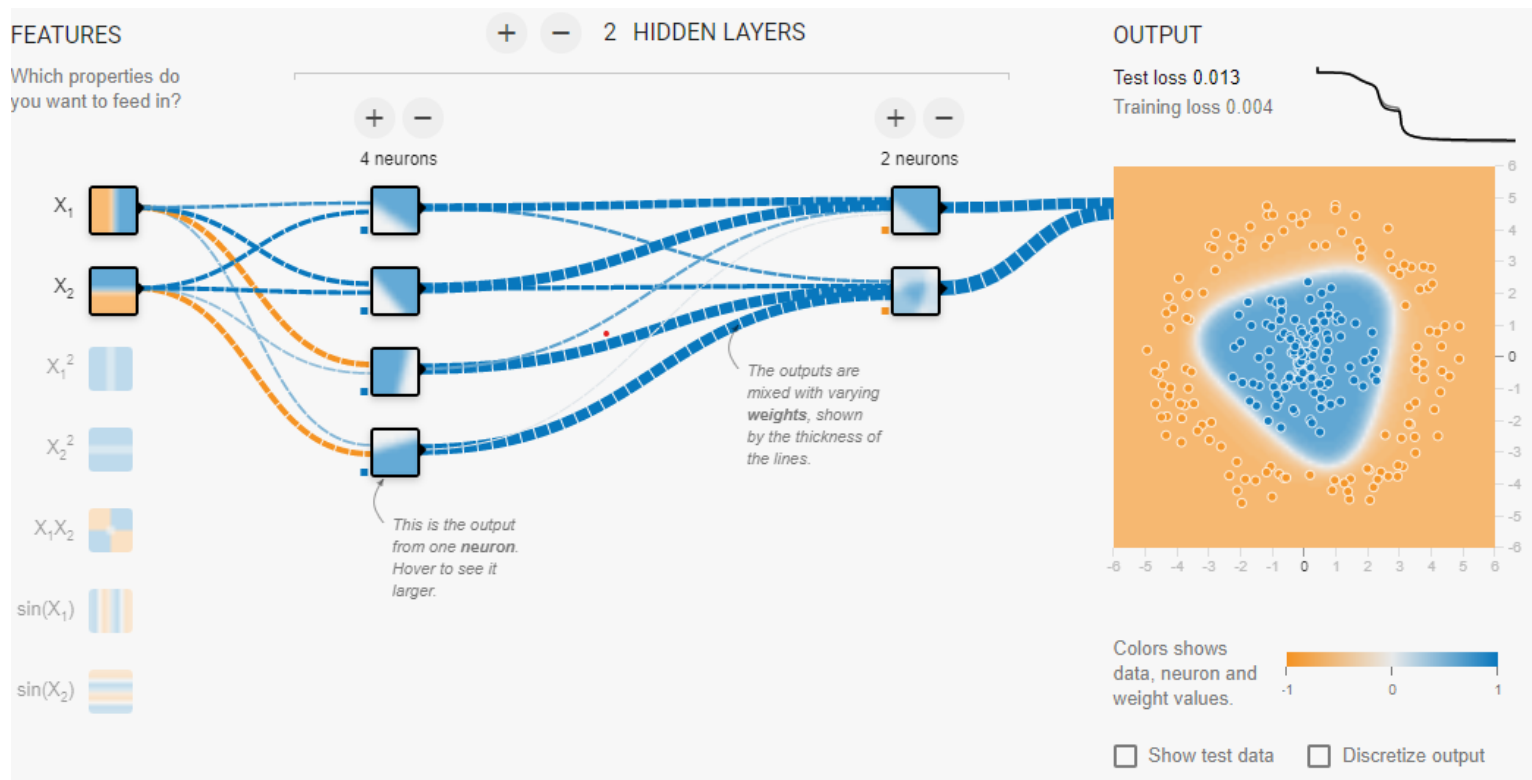
Perceptron training algorithm

PERCEPTRON(\mathbf{w}_0)

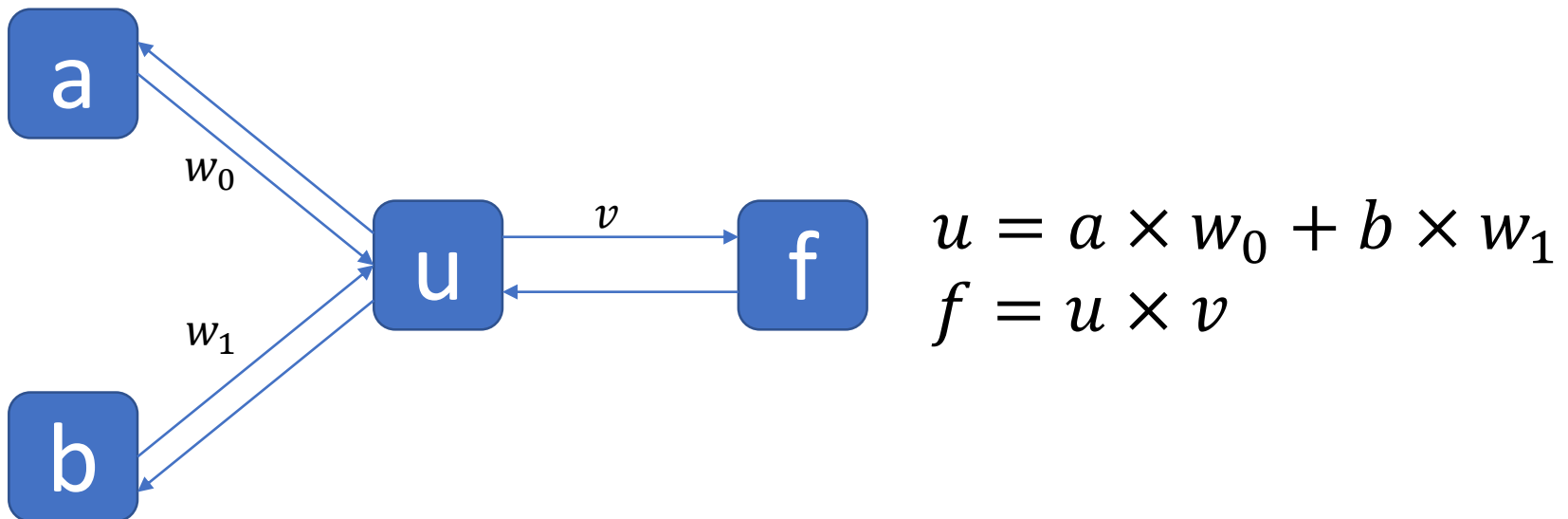
```
1   $\mathbf{w}_1 \leftarrow \mathbf{w}_0$        $\triangleright$  typically  $\mathbf{w}_0 = \mathbf{0}$ 
2  for  $t \leftarrow 1$  to  $T$  do
3      RECEIVE( $\mathbf{x}_t$ )
4       $\hat{y}_t \leftarrow \text{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t)$ 
5      RECEIVE( $y_t$ )
6      if  $(\hat{y}_t \neq y_t)$  then
7           $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$      $\triangleright$  more generally  $\eta y_t \mathbf{x}_t, \eta > 0$ .
8      else  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$ 
9  return  $\mathbf{w}_{T+1}$ 
```

Computational Model

- <https://playground.tensorflow.org>

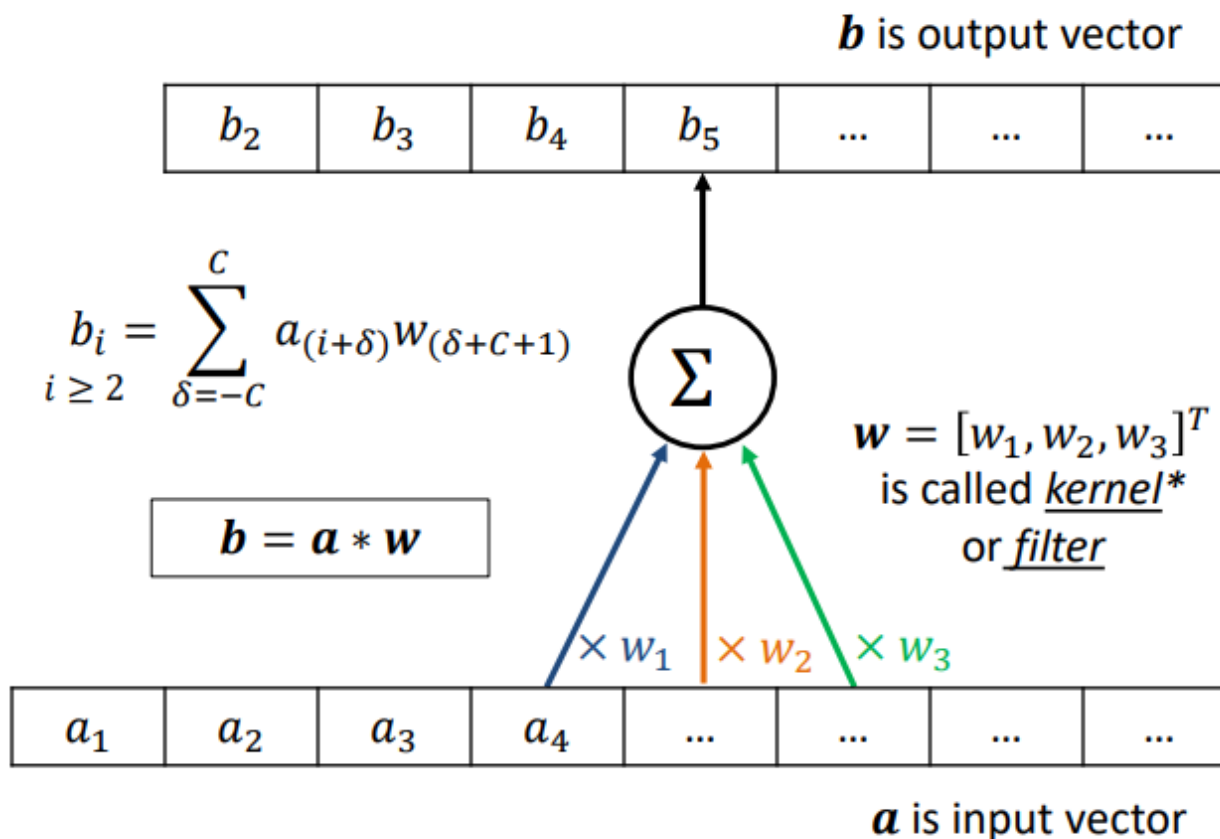


- MSE $L = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
- $\frac{\partial L}{\partial v} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial v} = 2(\hat{y}_i - y_i) \times u = -50$
- $\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial u} \frac{\partial u}{\partial w_0} = 2(\hat{y}_i - y_i) \times v \times a = -20$
- $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial u} \frac{\partial u}{\partial w_1} = 2(\hat{y}_i - y_i) \times v \times b = -30$



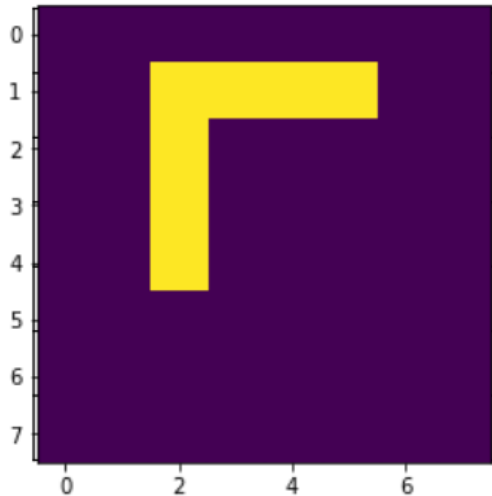
Model Design in PyTorch

Convolutional Neural Networks



Idea of filters

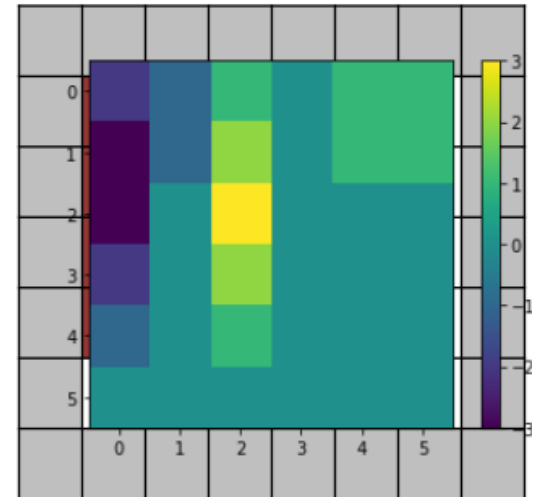
Filters are learned from training data!



Filters/Kernels

-1	0	1
-1	0	1
-1	0	1

Patterns/Features



1	1	1
0	0	0
-1	-1	-1

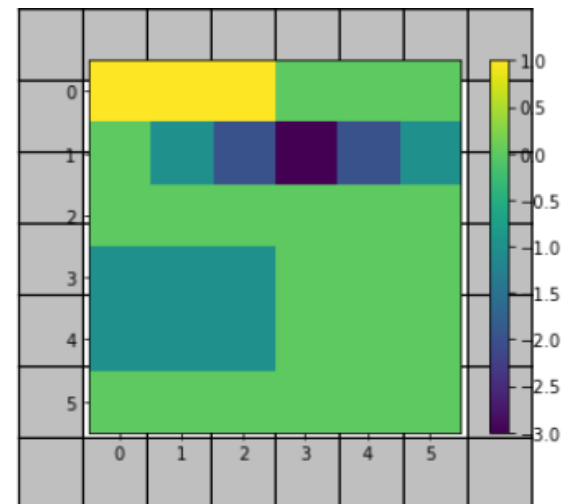


Image Classification on CIFAR-10

<https://www.cs.toronto.edu/~kriz/cifar.html>

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



Model design with torch.nn.Module

- Implement the constructor `__init__(self, ...)`. Here we define all network parameters.
- Override the forward method `forward(self, x)`. This accepts the input tensor `x` and returns our desired model output.
- Provided your operations are autograd-compliant, the backward pass is implemented **automatically** as PyTorch walks the computational graph backward.

```

import torch.nn as nn
import torch.nn.functional as F

class LogisticRegressionModel(nn.Module):

    def __init__(self, n_features, n_classes):
        super(LogisticRegressionModel, self).__init__()

        # Register weight matrix and bias term as model parameters - automatically tracks operations for gradient computation
        self.W = torch.nn.Parameter(torch.nn.init.xavier_uniform_(torch.empty([n_features, n_classes]))) # Weights
        self.b = torch.nn.Parameter(torch.zeros([n_classes])) # Biases

    def forward(self, x):
        """
        Forward pass for logistic regression.
        Input: Tensor x of shape [N,C,H,W]
        Output: Logits W @ x + b
        """
        batch_size = x.shape[0]

        x = x.view(batch_size, -1) # Flatten image into vector, retaining batch dimension
        out = torch.matmul(x, self.W) + self.b # Compute scores
        return out

```

```
def train(model, train_loader, test_loader, optimizer, n_epochs=10):  
    """  
    Generic training loop for supervised multiclass learning  
    """  
  
    LOG_INTERVAL = 250  
    running_loss, running_accuracy = list(), list()  
    start_time = time.time()  
    criterion = torch.nn.CrossEntropyLoss()  
  
    for epoch in range(n_epochs): # Loop over training dataset `n_epochs` times  
  
        epoch_loss = 0.  
  
        for i, data in enumerate(train_loader): # Loop over elements in training set  
  
            x, labels = data  
  
            logits = model(x)  
  
            predictions = torch.argmax(logits, dim=1)  
            train_acc = torch.mean(torch.eq(predictions, labels).float()).item()  
  
            loss = criterion(input=logits, target=labels)  
  
            loss.backward() # Backward pass (compute parameter gradients)  
            optimizer.step() # Update weight parameter using SGD  
            optimizer.zero_grad() # Reset gradients to zero for next iteration
```

Convolutional Networks

- Convolutional Layer #1 | 8 5×5 filters with a stride of 1, ReLU activation function.
- Max Pooling #1 | Kernel size 2 with a stride of 1.
- Convolutional Layer #2 | 16 5×5 filters with a stride of 1, ReLU activation function.
- Max Pooling #2 | Kernel size 2 with a stride of 1.
- Fully Connected Layer #1 | 400 input units (flattened convolutional output), 256 output units.
- Fully Connected Layer #2 | 256 input units, 10 output units - yields logits for classification.

```
OUT_C1 = 8
OUT_C2 = 16
DENSE_UNITS = 256
```

```
class BasicConvNet(nn.Module):
    def __init__(self, out_c1, out_c2, dense_units, n_classes=10):
        super(BasicConvNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=out_c1, kernel_size=5)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=out_c1, out_channels=out_c2, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, dense_units)
        self.logits = nn.Linear(dense_units, n_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        out = self.logits(x)
        return out
```

```
conv2D_model = BasicConvNet(OUT_C1, OUT_C2, DENSE_UNITS)
```

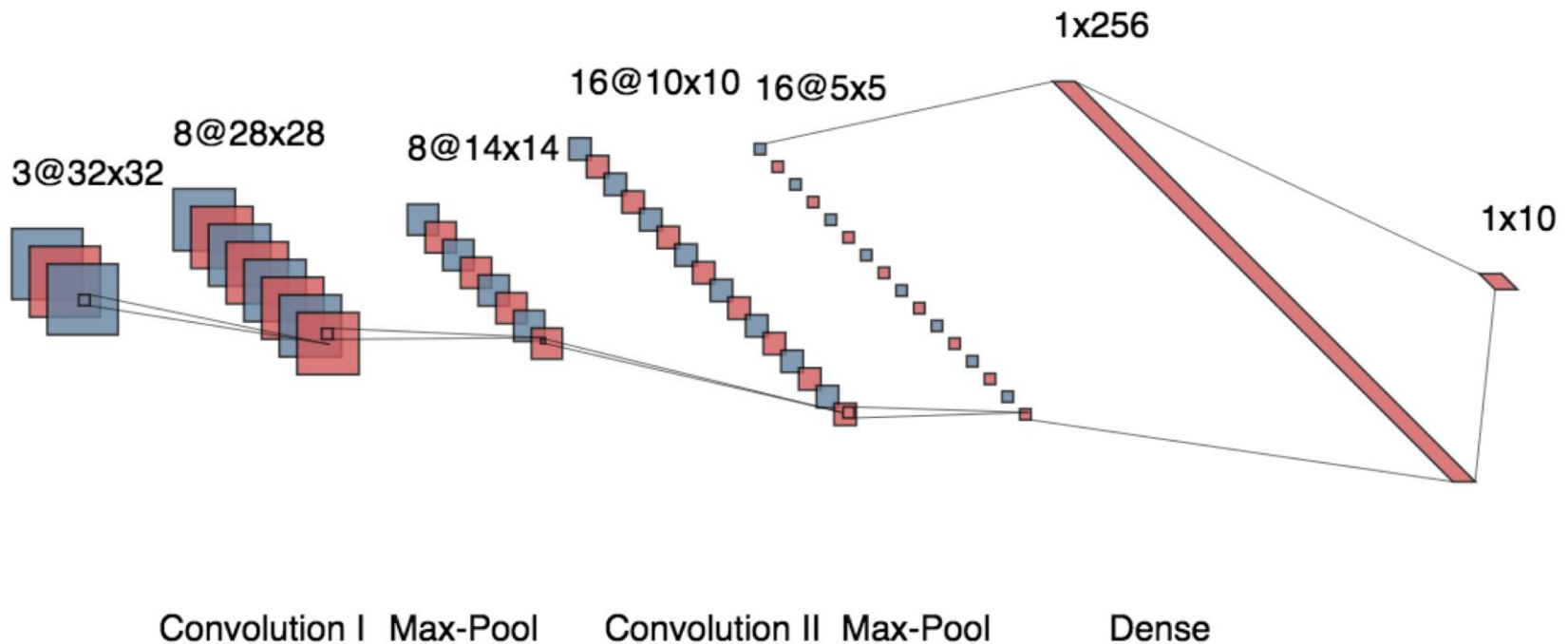
Downsampling Max-Pool

1	2	3	4
1	3	2	5
3	2	1	5
2	4	5	3

kernel_size=2, stride=2

3	5
4	5

Calculate the number of parameters



Convolution I: $3 \times 8 \times 5 \times 5 + 8$

<https://pytorch.org/docs/stable/nn.html#convolution-layers>

Dense I: $16 \times 5 \times 5 \times 256 + 256$

Autoencoders

