

# COMP90051

## Statistical Machine Learning

### Workshop Week 6

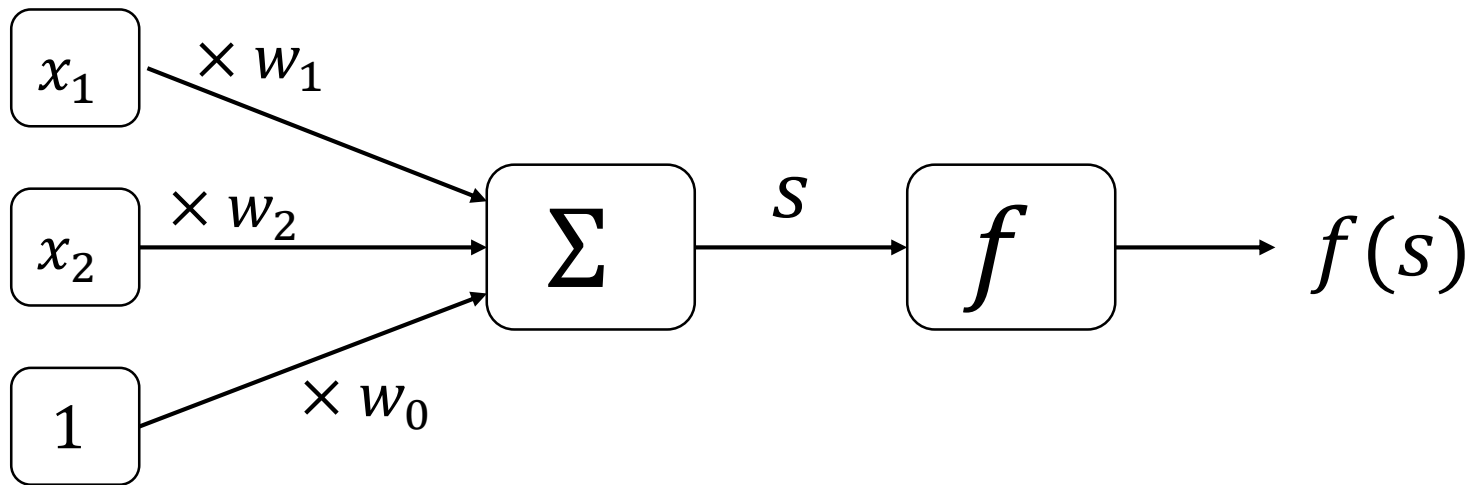
Xudong Han

[https://github.com/HanXudong/COMP90051\\_Workshops](https://github.com/HanXudong/COMP90051_Workshops)

# Your tutor

- Xudong Han
- [xudong.han@unimelb.edu.au](mailto:xudong.han@unimelb.edu.au)
- Slides  
[https://github.com/HanXudong/COMP90051\\_Worksheets](https://github.com/HanXudong/COMP90051_Worksheets)

# Review of the perceptron



- $$f(s) = \begin{cases} 1 & \text{if } s \geq 0, \\ -1 & \text{otherwise} \end{cases}$$

# 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  $\mathbf{w}_{T+1}$ 
```

# 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.

# Computational Model

- TensorFlow: Static Graphs

**Nodes** represent operations (e.g. Matrix multiplication).

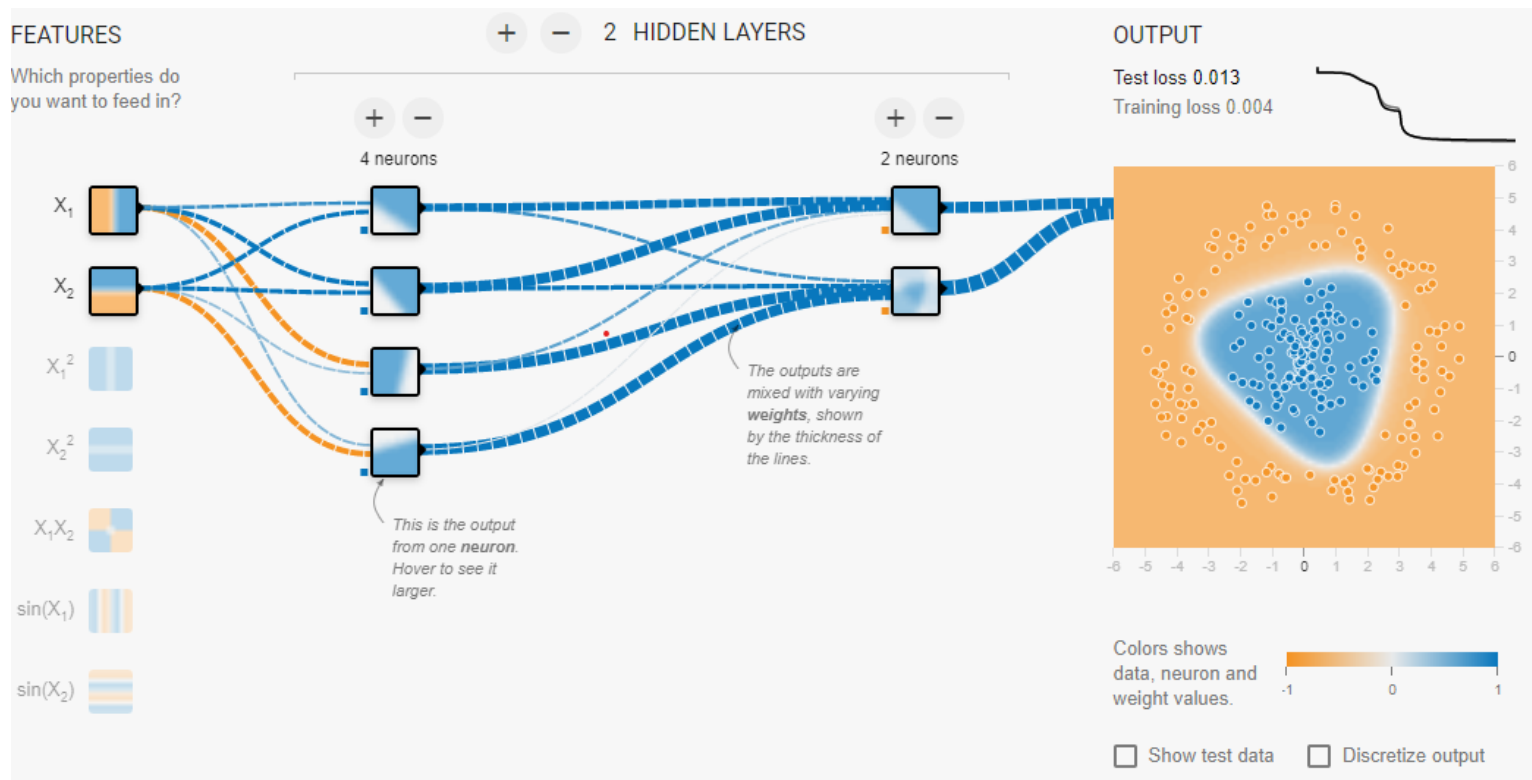
**Edges** represent tensors and flow between nodes.

- User defines computational graph beforehand, to be executed at later stage.

• Scalar	Vector	Matrix	Tensor
0	1	2	3+

# Computational Model

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



# Computational Model

- Pytorch: Dynamic  
Necessary computation graph metadata is generated automatically **each time** an operation is executed.
- Multiple advantages:
  - I. Allows different computational architecture for each training example/batch - **more flexible** algorithms, especially for dynamically sized data.
  - II. Control flow (if, while) can be implemented in host language.



# Basic Operations

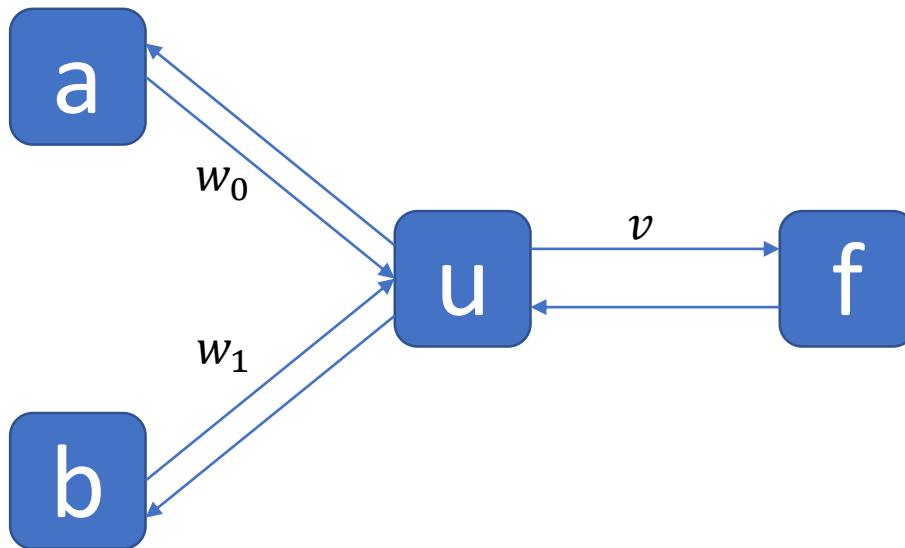
- The basic API is extremely similar to NumPy
- Try!
- <https://pytorch.org/docs/stable/torch.html#math-operations>

# AUTOGRAD MECHANICS

- <https://pytorch.org/docs/stable/notes/autograd.html>

```
>>> x = torch.randn(5, 5)  # requires_grad=False by default
>>> y = torch.randn(5, 5)  # requires_grad=False by default
>>> z = torch.randn((5, 5), requires_grad=True)
>>> a = x + y
>>> a.requires_grad
False
>>> b = a + z
>>> b.requires_grad
True
```

# Autograd



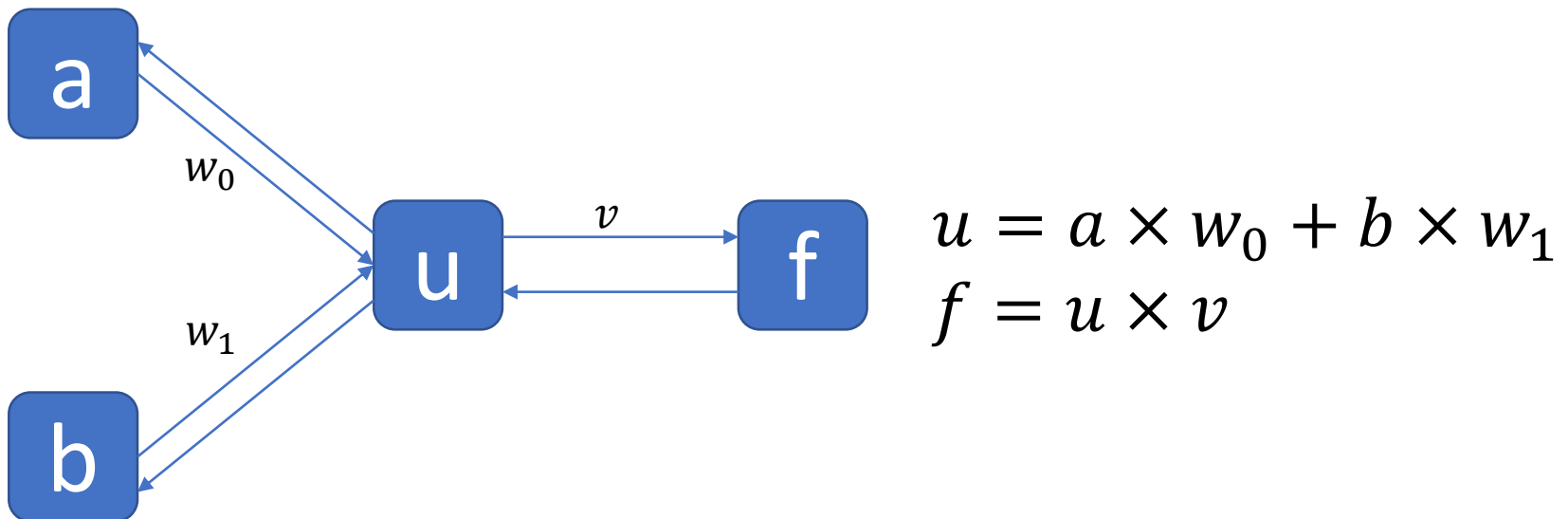
$$u = a \times w_0 + b \times w_1$$
$$f = u \times v$$

Choice of loss function  
MSE

Data:  
 $a = 2, b = 3, y = 10$

Init:  
 $w_0 = w_1 = v = 1$

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



```
import torch
```

```
a = torch.tensor(2.0)
```

```
b = torch.tensor(3.0)
```

```
w0 = torch.tensor(1.0, requires_grad = True)
```

```
w1 = torch.tensor(1.0, requires_grad = True)
```

```
v = torch.tensor(1.0, requires_grad = True)
```

```
u = a*w0 + b*w1
```

```
f = v*u
```

```
print(u, f)
```

```
criterion = torch.nn.MSELoss()
```

```
loss = criterion(f, torch.tensor(10.0))
```

```
print(loss)
```

```
loss.backward()
```

```
print(w0.grad, w1.grad, v.grad)
```