

# Lecture 1 Linear Classification

Fei Tan



@econdojo



@BusinessSchool101



Saint Louis University

Introduction to Neural Networks

July 12, 2025

# Image Classification



|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 08 | 02 | 22 | 97 | 38 | 15 | 00 | 40 | 00 | 75 | 04 | 05 | 07 | 78 | 52 | 12 | 50 | 77 | 97 | 29 |
| 49 | 49 | 99 | 40 | 17 | 81 | 18 | 57 | 60 | 87 | 17 | 40 | 98 | 43 | 69 | 45 | 74 | 56 | 62 | 00 |
| 81 | 49 | 31 | 73 | 55 | 79 | 14 | 29 | 93 | 71 | 40 | 67 | 58 | 85 | 30 | 03 | 49 | 13 | 36 | 65 |
| 52 | 70 | 95 | 23 | 04 | 60 | 11 | 42 | 68 | 44 | 88 | 56 | 01 | 32 | 56 | 71 | 37 | 02 | 36 | 91 |
| 22 | 31 | 16 | 71 | 51 | 60 | 05 | 89 | 41 | 92 | 36 | 54 | 22 | 40 | 40 | 28 | 66 | 33 | 13 | 80 |
| 24 | 47 | 33 | 60 | 99 | 03 | 45 | 02 | 44 | 75 | 33 | 53 | 78 | 36 | 84 | 20 | 35 | 17 | 12 | 50 |
| 02 | 95 | 81 | 28 | 64 | 23 | 67 | 10 | 26 | 38 | 40 | 67 | 59 | 54 | 70 | 66 | 18 | 38 | 64 | 70 |
| 67 | 26 | 20 | 68 | 02 | 62 | 12 | 20 | 95 | 63 | 94 | 39 | 63 | 08 | 40 | 91 | 66 | 49 | 94 | 21 |
| 24 | 35 | 38 | 05 | 66 | 73 | 99 | 26 | 97 | 17 | 78 | 78 | 96 | 83 | 14 | 88 | 34 | 89 | 63 | 72 |
| 21 | 36 | 23 | 09 | 75 | 00 | 76 | 44 | 20 | 45 | 35 | 14 | 00 | 61 | 33 | 97 | 34 | 31 | 33 | 95 |
| 78 | 17 | 53 | 28 | 22 | 75 | 31 | 67 | 15 | 94 | 03 | 80 | 04 | 62 | 16 | 14 | 09 | 53 | 56 | 92 |
| 16 | 39 | 05 | 42 | 96 | 35 | 31 | 47 | 55 | 58 | 88 | 24 | 00 | 17 | 54 | 24 | 36 | 29 | 85 | 57 |
| 86 | 56 | 00 | 48 | 35 | 71 | 89 | 07 | 05 | 44 | 44 | 37 | 44 | 60 | 21 | 58 | 51 | 54 | 17 | 58 |
| 19 | 80 | 81 | 68 | 05 | 94 | 47 | 69 | 28 | 73 | 92 | 13 | 86 | 52 | 17 | 77 | 04 | 89 | 55 | 40 |
| 04 | 32 | 08 | 83 | 97 | 35 | 99 | 16 | 07 | 97 | 57 | 32 | 16 | 26 | 26 | 79 | 33 | 27 | 98 | 66 |
| 65 | 36 | 68 | 87 | 57 | 62 | 20 | 72 | 03 | 46 | 33 | 67 | 46 | 55 | 12 | 32 | 63 | 93 | 53 | 69 |
| 04 | 42 | 16 | 73 | 35 | 36 | 39 | 11 | 24 | 94 | 72 | 18 | 08 | 46 | 29 | 32 | 40 | 62 | 76 | 36 |
| 20 | 69 | 36 | 41 | 72 | 30 | 23 | 88 | 34 | 52 | 82 | 69 | 82 | 67 | 59 | 85 | 74 | 04 | 36 | 16 |
| 20 | 73 | 35 | 29 | 78 | 31 | 90 | 01 | 74 | 31 | 49 | 71 | 38 | 24 | 61 | 16 | 23 | 57 | 05 | 54 |
| 01 | 70 | 54 | 71 | 83 | 51 | 54 | 69 | 16 | 92 | 33 | 48 | 61 | 43 | 52 | 01 | 89 | 29 | 62 | 48 |

What the computer sees

image classification

82% cat  
15% dog  
2% hat  
1% mug

# The Road Ahead...

- ① Linear Classifiers
- ② Gradient Descent
- ③ Backpropagation

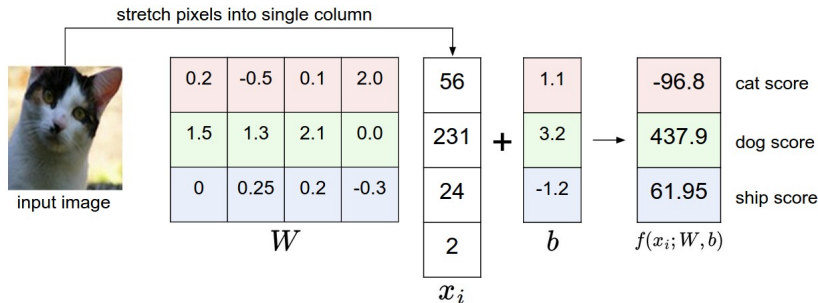
# Score Function

## Linear score

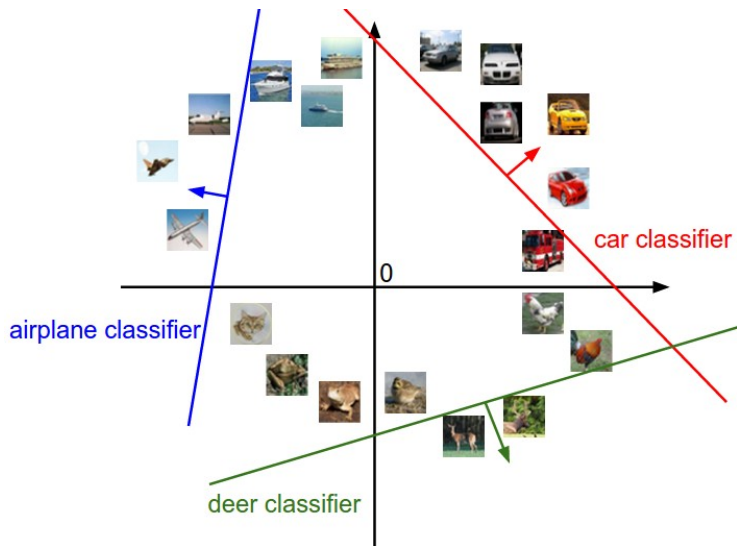
$$f(x_i, W, b) = Wx_i + b$$

- ▶ Notation
  - ▶ dataset of  $N$  examples  $\{(x_i, y_i)\}_{i=1}^N$ ,  $D$  (normalized) features  $x_i \in \mathbb{R}^D$ ,  $K$  categories of *single* attribute  $y_i \in 1, \dots, K$
  - ▶  $K \times D$  weights  $W$ ,  $K \times 1$  bias  $b$
- ▶ Pipeline
  - ▶ split data into training, validation, and test sets
  - ▶ train parameters and (cross) validate hyperparameters
  - ▶ evaluate on test set only once at the end

# Algebraic Interpretation



# Geometric Interpretation



# Loss Function

## Regularized loss

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W), \quad \lambda > 0$$

- ▶ Examples of data loss
  - ▶ multiclass support vector machine (SVM) classifier uses hinge loss:  $L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + \Delta)$ ,  $\Delta > 0$
  - ▶ Softmax classifier uses cross-entropy loss:  $L_i = -\log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right)$
- ▶ Examples of regularization/prior
  - ▶  $L^1$  regularization:  $R(W) = \sum_{i,j} |W_{ij}|$
  - ▶  $L^2$  regularization:  $R(W) = \sum_{i,j} W_{ij}^2$

## Loss Function (Cont'd)

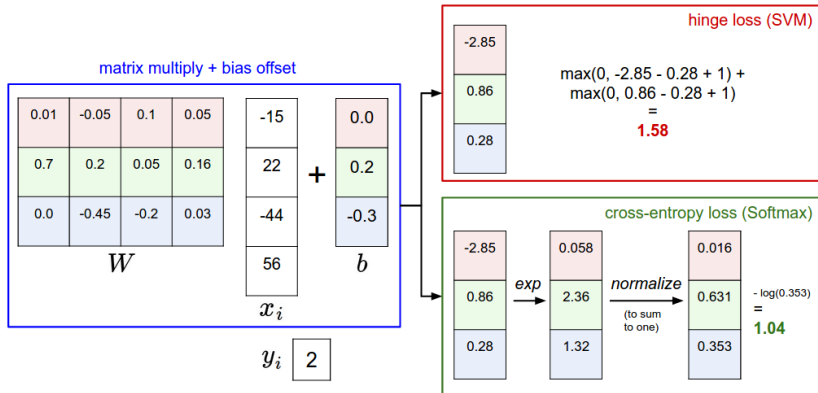
```
import numpy as np

# SVM classifier
def L1_i(x_i, y_i, W, b):
    delta = 1.0
    f = W.dot(x_i) + b
    margins = np.maximum(0, f - f[y_i] + delta)
    margins[y_i] = 0 # ignore true class
    return np.sum(margins)

# Softmax classifier
def L2_i(x_i, y_i, W, b):
    f = W.dot(x_i) + b
    f -= np.max(f) # avoid potential blowup
    p = np.exp(f) / np.sum(np.exp(f))
    return -np.log(p[y_i])
```



# SVM vs. Softmax



# The Road Ahead...

- ① Linear Classifiers
- ② Gradient Descent
- ③ Backpropagation

## Derivative of 1-D function

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

- ▶ Gradient of multi-D function
  - ▶ vector of partial derivatives in each dimension
  - ▶ examples of 2-D function

$$f(x, y) = xy \quad \rightarrow \quad \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] = [y, x]$$

$$f(x, y) = x + y \quad \rightarrow \quad \nabla f = [1, 1]$$

$$f(x, y) = \max(x, y) \quad \rightarrow \quad \nabla f = [\mathbb{1}(x \geq y), \mathbb{1}(x \leq y)]$$

# Numerical Gradient

```
import numpy as np

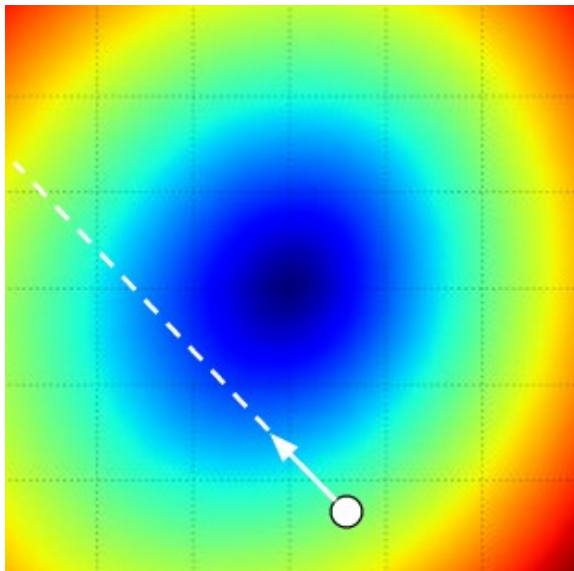
def num_grad(f, x): # finite difference method
    fx = f(x)
    grad = np.zeros(x.shape)
    h = 0.00001
    it = np.nditer(x, flags=['multi_index'],
        op_flags=['readwrite'])
    while not it.finished:
        ix = it.multi_index
        old_value = x[ix]
        x[ix] = old_value + h
        fxh = f(x)
        x[ix] = old_value
        grad[ix] = (fxh - fx) / h # alternatively
            [f(x+h)-f(x-h)]/2h
        it.iternext()
    return grad
```

# Gradient Descent

- ▶ Repeated local search to minimize loss function
  - ▶ update in *negative* gradient direction
  - ▶ validate learning rate (step size)
- ▶ Mini-batch/stochastic gradient descent

```
while True:
    data_batch = sample_data(training_data,
                              256)
    weights_grad = eval_grad(loss_fun,
                              data_batch, weights)
    weights += - step_size * weights_grad  #
        in-place update
```

## Gradient Descent (Cont'd)



# The Road Ahead...

- ① Linear Classifiers
- ② Gradient Descent
- ③ Backpropagation

# Backpropagation

## Chain rule

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

### ► Example of composite function

$$f(v(p(x,y), q(z,w))) = 2 \left[ \underbrace{xy}_{p \text{ (* gate)}} + \underbrace{\max(z,w)}_{q \text{ (max gate)}} \right]$$

$\underbrace{\hspace{15em}}_{v \text{ (+ gate)}}$

► forward pass:  $[x, y, z, w] = [3, -4, 2, -1]$ ,  $f = -20$

► backward pass:  $\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial w} \right] = [-8, 6, 2, 0]$



# Backpropagation (Cont'd)

```
# Forward pass
```

```
x = 3; y = -4; z = 2; w = -1
```

```
p = x * y          # -12
```

```
q = max(z, w)      # 2
```

```
v = p + q          # -10
```

```
f = 2 * v           # -20
```

```
# Backward pass
```

```
dfdvdv = 2
```

```
dvdvp = 1
```

```
dpdx = y
```

```
dpdy = x
```

```
dvdq = 1
```

```
dqdz = (z > w)
```

```
dqdw = (w > z)
```

```
dfdx = dfdv * dvdvp * dpdx # -8
```

```
dfdy = dfdv * dvdvp * dpdy # 6
```

```
dfdvdz = dfdv * dvdq * dqdz # 2
```

```
dfdvw = dfdv * dvdq * dqdw # 0
```

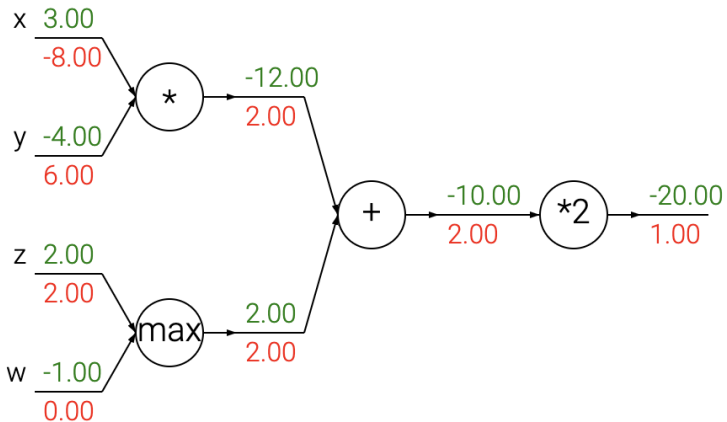
# PyTorch Implementation

```
import torch

# Forward pass
x = torch.tensor(3., requires_grad=True)
y = torch.tensor(-4., requires_grad=True)
z = torch.tensor(2., requires_grad=True)
w = torch.tensor(-1., requires_grad=True)
p = x * y      # tensor(-12., grad_fn=<MulBackward0>)
q = max(z, w)  # tensor(2., grad_fn=<MaxBackward0>)
v = p + q      # tensor(-10., grad_fn=<AddBackward0>)
f = 2 * v      # tensor(-20., grad_fn=<MulBackward0>)

# Backward pass
f.backward()   # compute gradients
print(x.grad)  # tensor(-8.)
print(y.grad)  # tensor(6.)
print(z.grad)  # tensor(2.)
print(w.grad)  # tensor(0.)
```

# PyTorch Computation Graph



# References

- ▶ [cs231n.stanford.edu](http://cs231n.stanford.edu) – CS231n: Deep Learning for Computer Vision
- ▶ [github.com/karpathy/micrograd](https://github.com/karpathy/micrograd) – A tiny scalar-valued autograd engine and a neural net library on top of it with PyTorch-like API
- ▶ Tang (2013), “Deep Learning using Linear Support Vector Machines”, [arXiv:1306.0239](https://arxiv.org/abs/1306.0239)