Image classification: Linear classifier

Computer Vision

Winter Semester 20/21

Goethe University

Acknowledgement: Some images are from various sources: UCF, Stanford cs231n, etc.

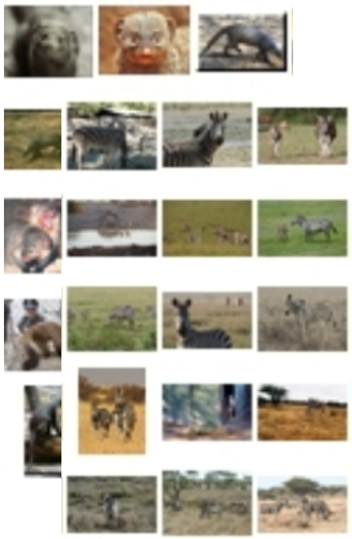
What we did last week

- Image histogram
- Image classification:
  - data-driven approach
  - K-nn

Today's class

- Image classification:
    - Linear classifier
    - Gradient descent

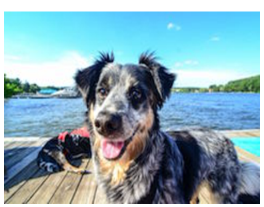
Data driven approach



Training/Learning (usually offline)

Training set: images with known class information → Learn a model using some algorithm → A model for this specific classification problem and classifier

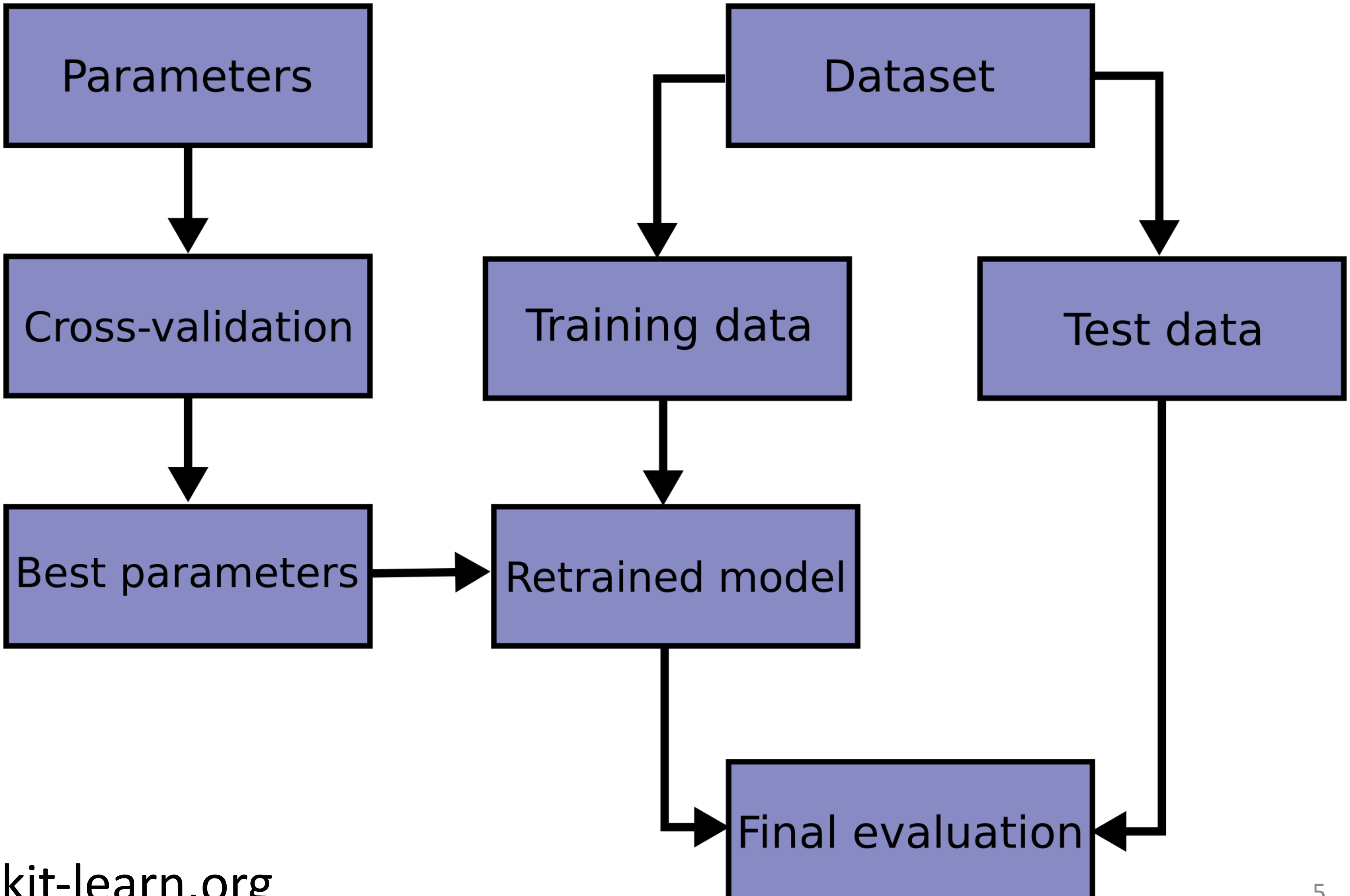
Testing/Evaluation (usually online)



Testing set (with label during evaluation, without label in an application) → Classifier algorithm → Predicted class information for this new image (compare with ground-truth during evaluation)

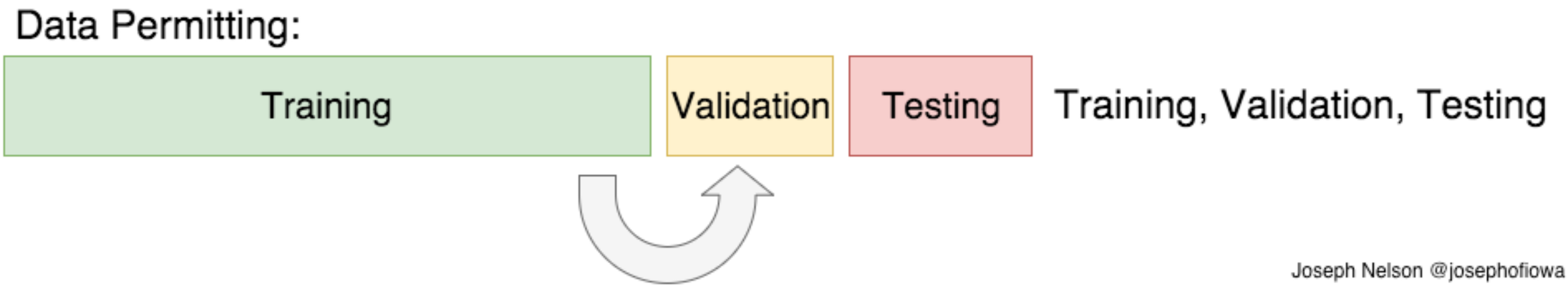
For practically use, testing time should be small

Training, validation, testing

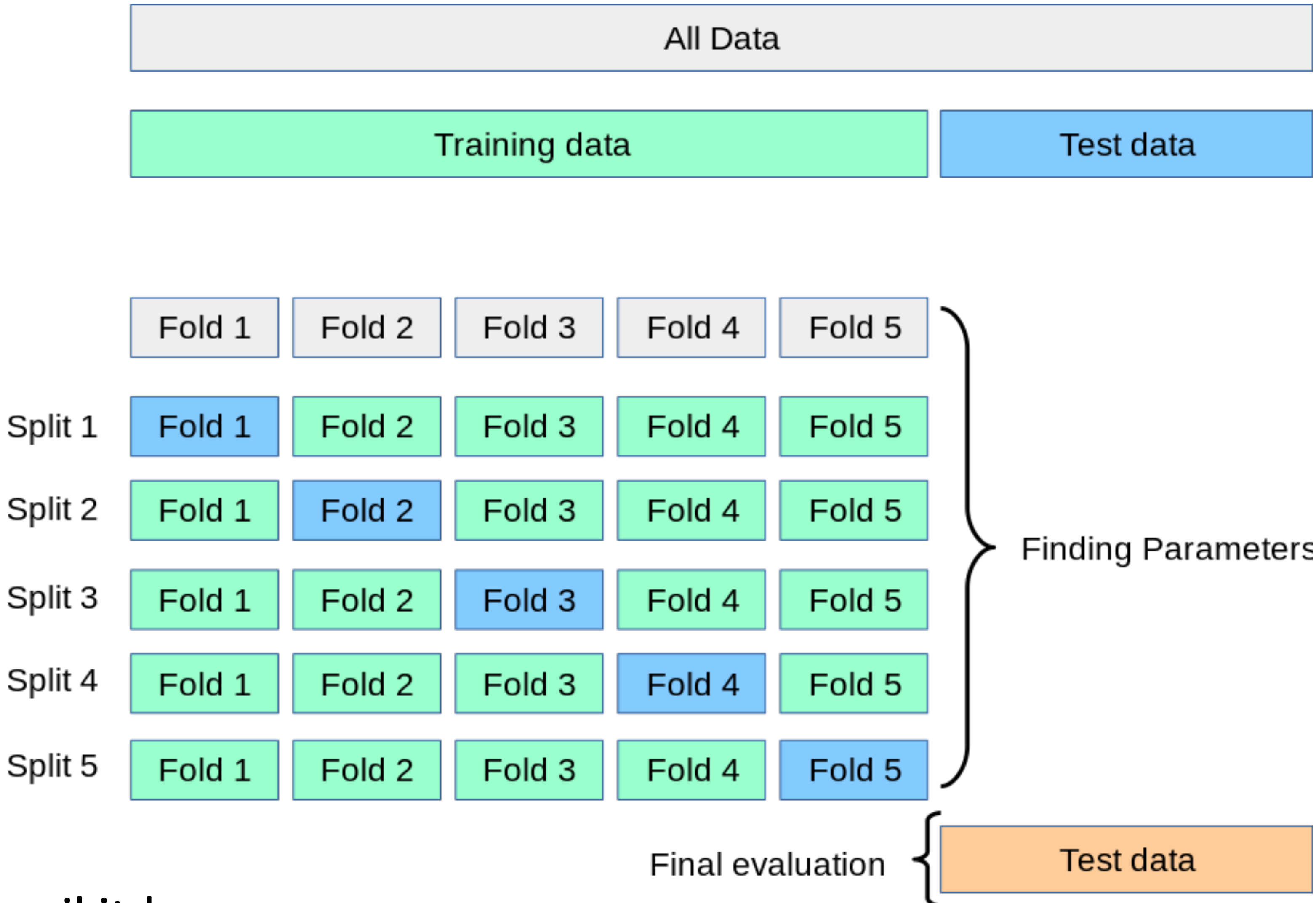


From: scikit-learn.org

Training, validation, testing



K-fold cross-validation



From: scikit-learn.org

Image classification with a Linear Classifier

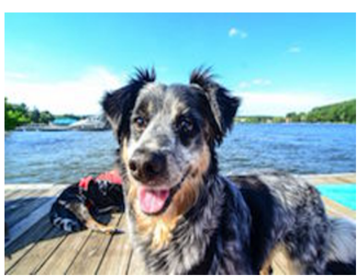
- Given a test image x, produce the confidence score for each class using linear transformation (total: K classes)
- Higher confidence score for a class -> more likely to be the ground-truth class
- Test image x: flatten to a Dx1 column vector, D is the image resolution times number of channel

Input x: Dx1
Weight W: KxD
Bias b: Kx1
Score s:Kx1

Image classification with a Linear Classifier

Score function:

$$s = f(x; W, b) = Wx + b$$



Input x: Dx1
Weight W: KxD
Bias b: Kx1
Score s:Kx1

Linear Classifier

$$s = f(x; W, b) = Wx + b$$

- **Testing**: W, b are fixed, x is the input
- **Training**: Given N training samples (x_i, y_i), y_i takes value in [1,...,K], learn W and b
- The ground truth class is y_i

Linear Classifier

$$s = f(x; W, b) = Wx + b$$

Training: (x_i, y_i) are given and fixed; W, b are the variables to be determined

Example:

K=3, {cat, <u>dog</u>, ship}

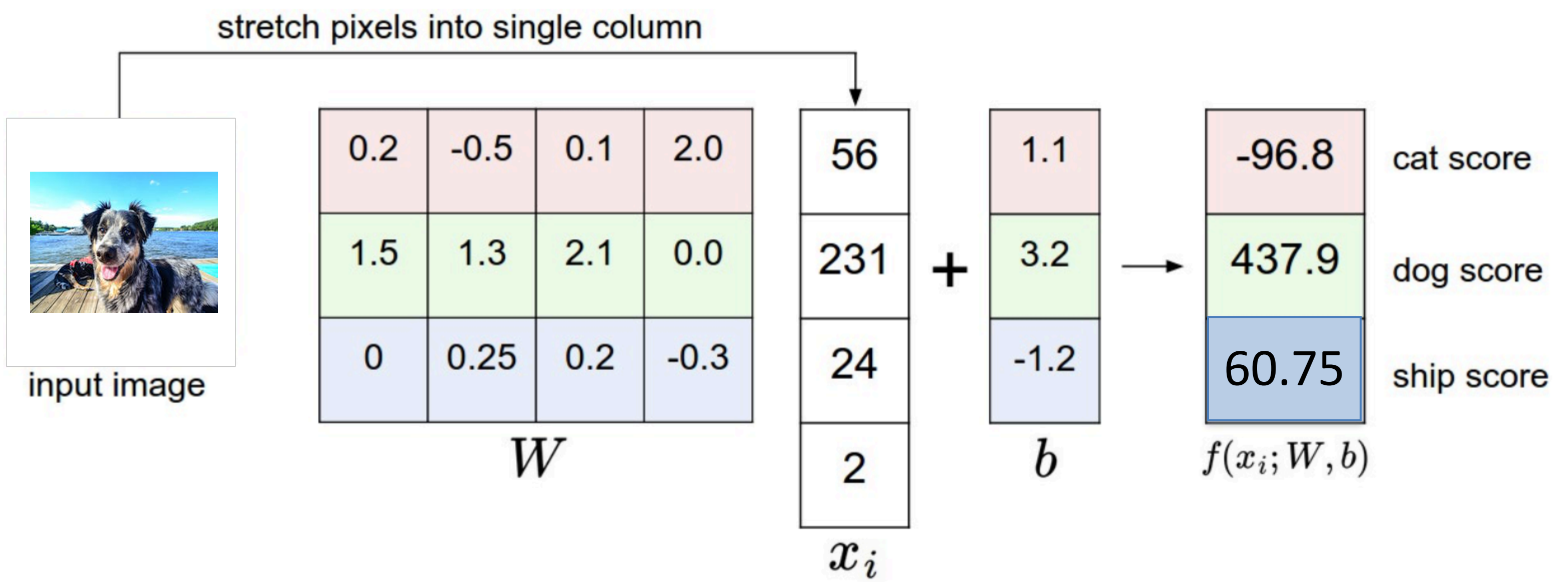
x_i= 

y_i = 2

Linear Classifier

$$s = f(x; W, b) = Wx + b$$

- **Testing**: W, b are fixed, x is the input



Linear Classifier

$$s = f(x; W, b) = Wx + b$$

- Training: learn W, b to discriminate the classes

- Each row of W extracts the features of a specific class from input

Linear Classifier

$$s = f(x; W, b) = Wx + b$$

- **Shorthand notation (equivalent to the above notation):**

$$\underbrace{s = [W\ b]}_{W}\underbrace{[x\ 1]^T}_{x}$$

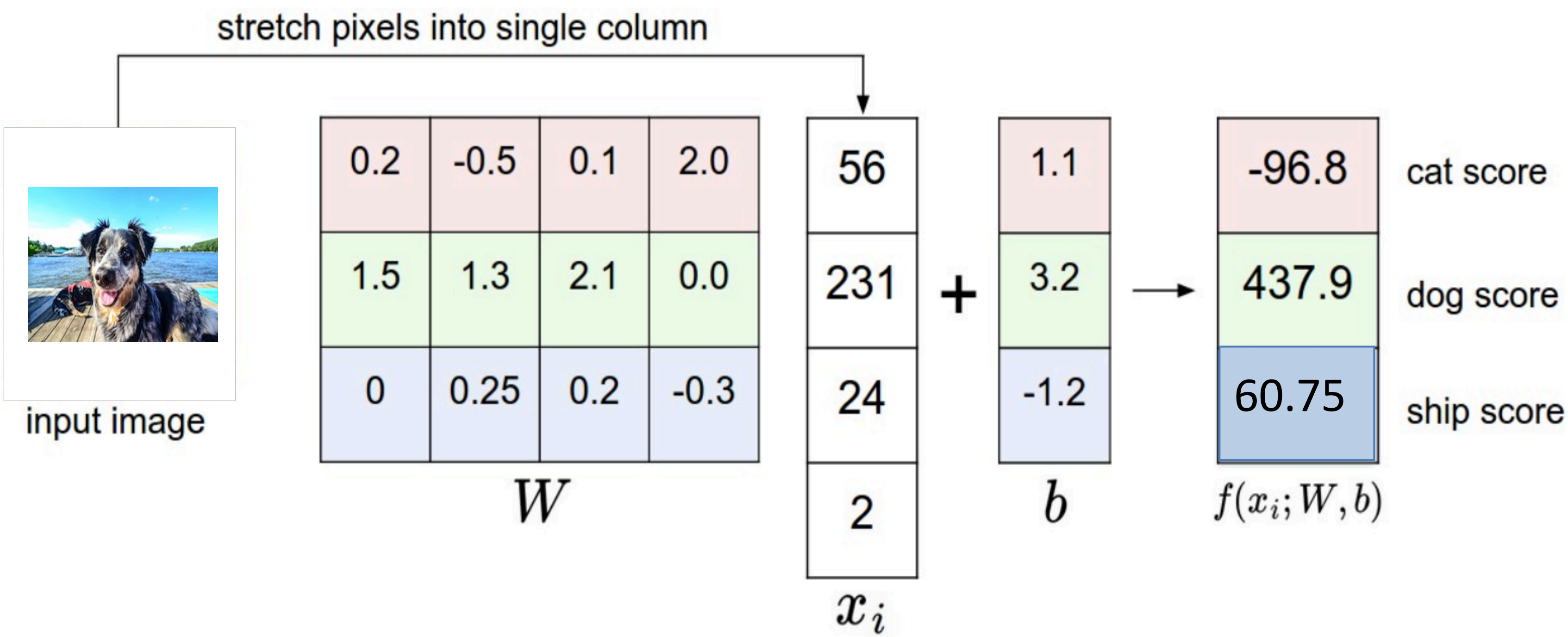
$$s = f(x; W) = Wx$$

Input x: (D+1)x1
Weight W: Kx(D+1)
Score s:Kx1

Linear Classifier

$$s = f(x; W, b) = Wx + b$$

- **QUESTION: For image classification, what other ways can we compute x from the original image?**



Loss function

- **Training**:

Given:

- N training samples (x_i, y_i),

- y_i takes value in [1,...,K],

- learn W

$$s = f(x; W, b) = Wx + b$$

Loss function

- **Training**: Given N training samples (x_i, y_i), y_i takes value in [1,...,K], learn W

- Loss function: measure how consistent are the ground-truth labels and the score function outputs, for some W

- Small loss: good W

Loss function

- **Training**: Given N training samples (x_i, y_i), y_i takes value in [1,...,K], learn W

- Loss function: measure how consistent are the ground-truth labels and the score function outputs, for some W

- Small loss: good W

EXAMPLES OF LOSS FUNCTIONS:

- Softmax classifier with cross-entropy loss
- Multiclass Support Vector Machine (SVM) loss

Softmax classifier

- Regard output of the score function f(x; W) as the *unnormalized log probability* of each class

Softmax classifier

- Regard output of the score function f(x; W) as the *unnormalized log probability* of each class

$$f(x; W, b) = Wx + b$$

- Probability of each class can be obtained by applying a softmax function (exp, then normalize):

$$\mathrm{softmax}(f) = \frac{e^{f_m}}{\sum_{j=1}^{K} e^{f_j}}$$ For the m-th class

Softmax classifier

- Probability of each class can be obtained by applying a softmax function (exp, then normalize):

$$\mathrm{softmax}(f) = \frac{e^{f_m}}{\sum_{j=1}^{K} e^{f_j}}$$

For the m-th class

- Cross-entropy loss (apply –log(.) to only the ground-truth class):

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

For the i-th training sample

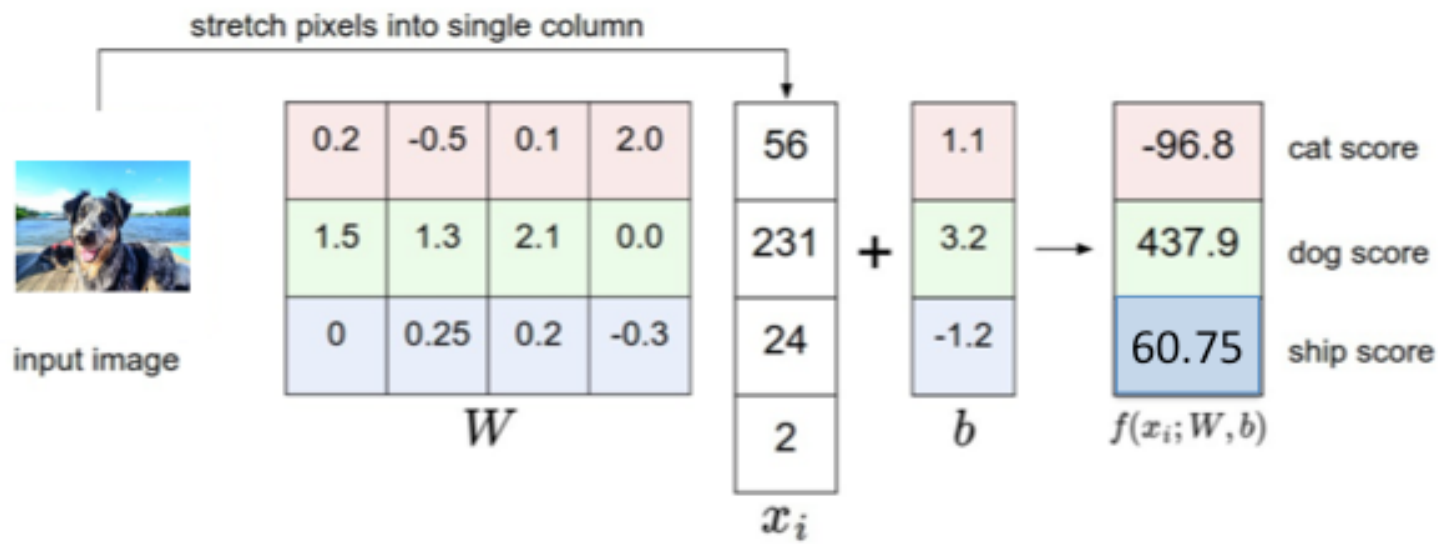
Softmax classifier

- Cross-entropy loss (apply –log(.) to only the ground-truth class):

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

For the i-th training sample

Example: a dog (which looks like a dog):

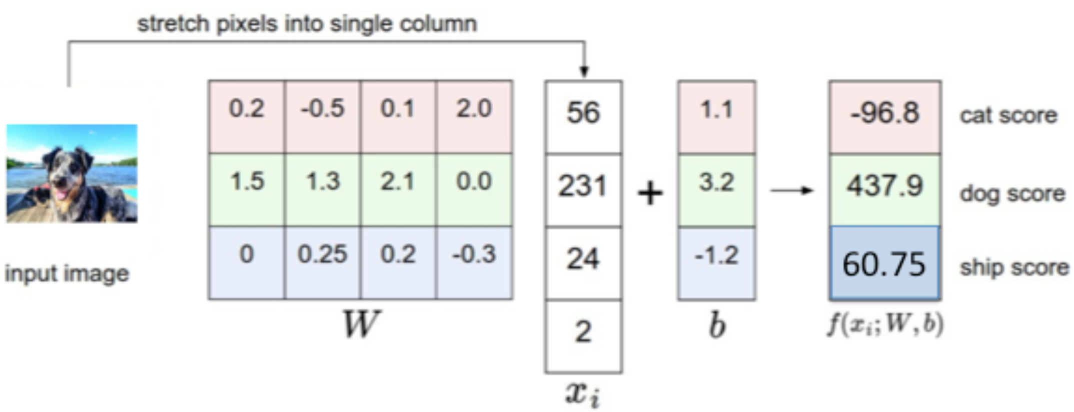


Softmax classifier

- Cross-entropy loss (apply –log(.) to only the ground-truth class):

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

For the i-th training sample



Example: a dog (which looks like a dog)

```
print(np.exp(f))
print(np.exp(f) / sum(np.exp(f)))
```

y_i=2

L_i = -log(1) = 0

```
[ 9.12628762e-043   1.50505935e+190   2.41762966e+026]
[ 6.06373937e-233   1.00000000e+000   1.60633510e-164]
```

Probability

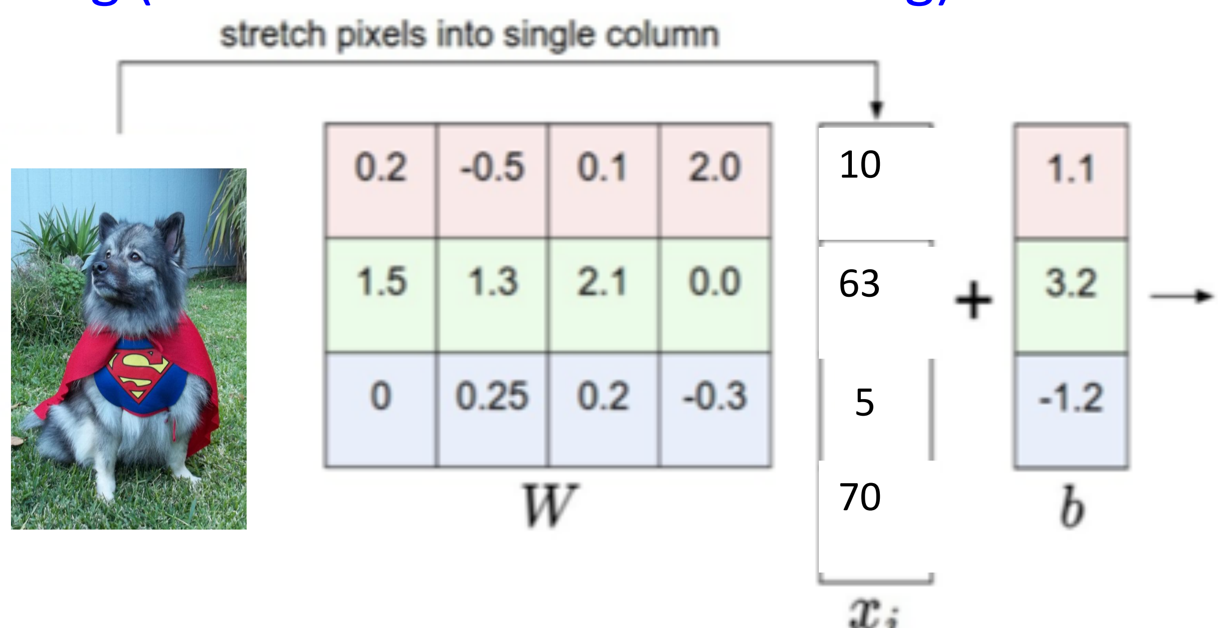
Softmax classifier

- Cross-entropy loss (apply –log(.) to only the ground-truth class):

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

For the i-th training sample

Exercice: a dog (which does not look like a dog):



Softmax classifier

- The entire loss for N training samples (x_i, y_i):

$$L = \frac{1}{N} \sum_i L_i$$

- We determine W to minimize this loss given the training dataset

- Additional regularization of W

Understanding Softmax classifier

- **Cross-entropy loss:**
  - First apply softmax function
  - Then apply –log(.) to only the ground-truth class

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

For the i-th training sample

Understanding Softmax classifier

- **Cross-entropy loss:**
  - First apply softmax function
  - Then apply –log(.) to only the ground-truth class

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

For the i-th training sample

- The term $\frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$ is the probability of the correct class (i.e., y_i)

- Therefore, want this to be large, i.e., $\max_W$ log(.)
- Thus, want this to be small $\min_W$ –log(.)

Understanding Softmax classifier

- Why min -log(.) or max log(.)?

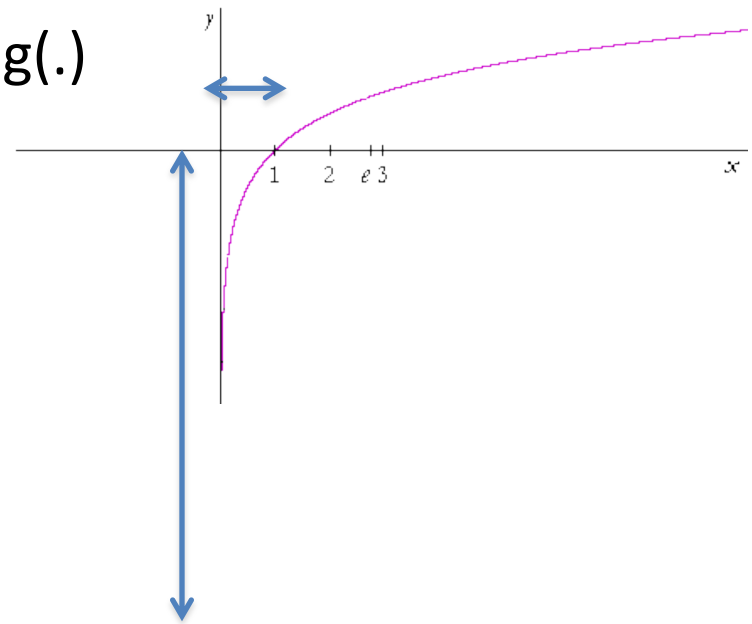
$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

- The term $\frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$ is the probability, between [0,1]

Understanding Softmax classifier

- Stretch the numerical range during min/max
- Often used when working with probability
- p1xp2 is small: log(p1xp2) = log(p1) + log(p2)
- Maximum Likelihood Estimation (MLE)
  - Minimize the negative log likelihood of the correct class

log(.)



Understanding Softmax classifier

- Why exp before normalization?

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

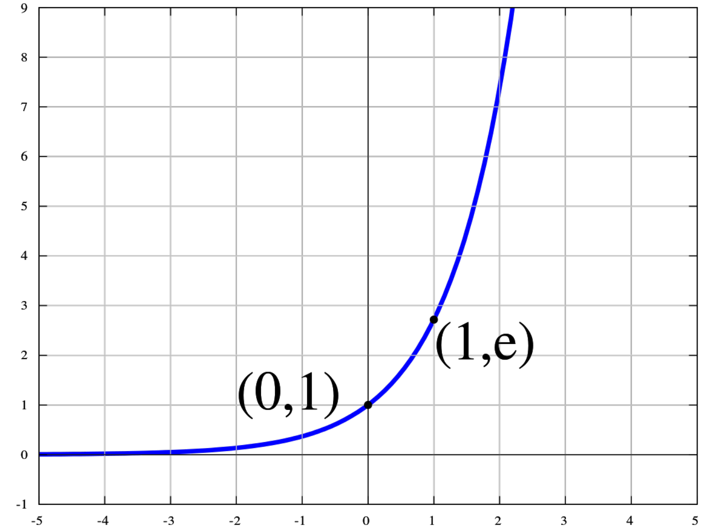
- Much higher confidence if the activation is large (clear images)

```
s = np.array([1,2])
print(np.exp(s) / sum(np.exp(s)))

[  0.26894142   0.73105858]


s = np.array([10,20])
print(np.exp(s) / sum(np.exp(s)))

[  4.53978687e-05   9.99954602e-01]
```



Understanding Softmax classifier

Another interpretation of the cross-entropy loss

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

- Difference between:
  - The model (estimated) prob Q: $\mathrm{softmax}(f) = \frac{e^{f_m}}{\sum_{j=1}^{K} e^{f_j}}$
  - The data (true) prob P: [0,0,....,1,....,0] (1 at the y_i-th position)
  - Measured by Kullback-Leibler (KL) divergence (D_{KL}=0 when P, Q are "the same"

Understanding Softmax classifier

Another interpretation of the cross-entropy loss

- The model (estimated) prob Q: $\mathrm{softmax}(f) = \dfrac{e^{f_m}}{\sum_{j=1}^{K} e^{f_j}}$

- The data (true) prob P: [0,0,....,1,....,0] (1 at the y_i-th position)

- Measured by Kullback-Leibler (KL) divergence (D_{KL}=0 when P, Q are "the same"

$$D_{\mathrm{KL}}(P\|Q) = -\sum_i P(i)\,\log\frac{Q(i)}{P(i)}$$

Want Q to be close to P

Understanding Softmax classifier

- Another interpretation of the cross entropy loss

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

- Difference between P and Q as measured by Kullback-Leibler (KL) divergence

$$D_{\mathrm{KL}}(P\|Q) = -\sum_i P(i) \log \frac{Q(i)}{P(i)}$$

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= -\sum_x p(x)\log q(x) + \sum_x p(x)\log p(x) \\ &= H(P,Q) - H(P) \end{aligned}$$

Cross- entropy of P and Q

Entropy of P: H(P) = 0 in this case

Softmax classifier

- Additional regularization of W (L2 or L1 norm of weights):

L2:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

L1:

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

- Prefer small $W_{k,l}$, less likely to overfit the training dataset

- Regularize only W, not the bias b

Softmax classifier

- Additional regularization of W (L2 or L1 norm of weights)

$$R(W) = \sum_k \sum_l W_{k,l}^2 \qquad R(W) = \sum_k \sum_l |W_{k,l}|$$

- The entire loss for N training samples (x_i, y_i): data loss and regularization loss

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

- We determine W to minimize this loss given the training dataset

Softmax classifier

- Additional regularization of W (L2 or L1 norm of weights)

$$R(W) = \sum_k \sum_l W_{k,l}^2 \qquad R(W) = \sum_k \sum_l |W_{k,l}|$$

- The entire loss for N training samples (x_i, y_i): data loss and regularization loss

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

- We determine W to minimize this loss given the training dataset
- **QUESTION**: how can we determine lambda?

Multiclass SVM loss

- **SVM loss:** The correct class has a score higher than the incorrect class by some fixed margin d

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + d)$$

- For this test image i, **zero score** contributed by class j iff

$0 >= s_j - s_{yi} + d$

$s_{yi} >= s_j + d$

Correct class score s_{yi} is higher than class j score s_j by at least d (otherwise, +ve. contribution of loss from class j)

Multiclass SVM loss

EXAMPLE

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + d)$$

- S = [13,-7,11]
- y_i= 1
- d=10

For test image i

L_i = max(0, -7-13+10) + max(0, 11-13+10)

= 0 + 8

- Ground-truth score 13 is higher than -7 by more than the margin d=10
- Ground-truth score 13 is not higher than 11 by d=10

Train W so that the correct class y_i has a score higher than the incorrect classes by at least d

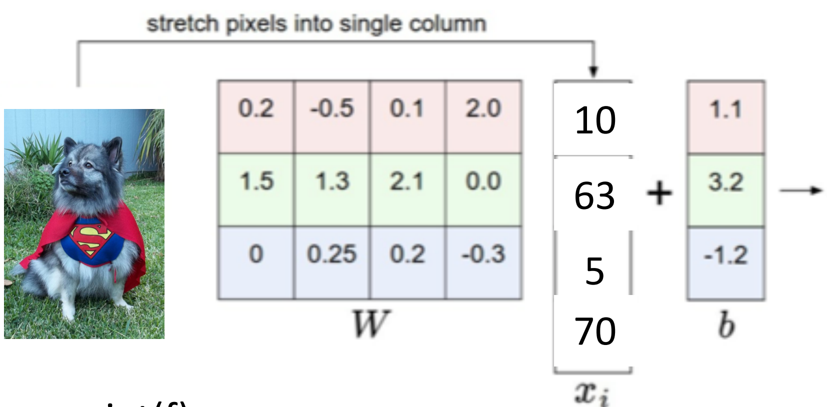
Softmax classifier

EXERCISE SOLUTION:
a dog (which does not look like a dog)

- Cross-entropy loss (apply –log(.) to only the ground-truth class):

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

For the i-th training sample




```
print(f)
print(np.exp(f))
print(np.exp(f) / sum(np.exp(f)))
[ 112.1      110.6       -5.45]
[ 4.83516636e+48   1.07887144e+48   4.29630469e-03]
[ 8.17574476e-01   1.82425524e-01   7.26458780e-52]
```

y_i=2
L_i = -log(0.182) = 1.7

← Probability

Linear Classifier

$$s = f(x; W, b) = Wx + b$$

- After learning of the parameter W, do not need the training data in deployment

- Fast in deployment

- How to learn W?

Today’s class

- Image classification:
  - Linear classifier
  - **Gradient descent**

Linear Classifier

- **Testing**: W, b are fixed, x is the input
- **Training**: Given N training samples (x_i, y_i), y_i takes value in [1,...,K], learn W and b

$$s = f(x; W, b) = Wx + b$$

Training: (x_i,y_i) are given and fixed; W, b are the variables to be determined

Learn W using loss function L(W)

- Try different W (randomly), choose the one with the min loss function

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

N training samples

- W is very large: Kx(D+1)
- Even larger in deep neural network

- Start from a random W, iteratively improve W (reduce L(W)): ***Gradient descent***

Learn W using loss function L(W)

- Try different W (randomly), choose the one with the min loss function

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

N training samples

- W is very large: Kx(D+1)
- Even larger in deep neural network

- Start from a random W, iteratively improve W (reduce L(W)): ***Gradient descent***

- Note: $L(W) = L(W; (x_1,y_1), (x_2,y_2), ...(x_i,y_i)...(x_N,y_N))$

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

- **Gradient**: a vector of partial derivatives in each dimension

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$w_l' = w_l - \gamma \frac{\partial L}{\partial w_l}$$

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

N training samples

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$w_l' = w_l - \gamma \frac{\partial L}{\partial w_l}$$

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

$$\frac{\partial L}{\partial w_l} = \frac{1}{N} \sum_i \frac{\partial L_i}{\partial w_l} + \lambda \frac{\partial R(W)}{\partial w_l}$$

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$w'_l = w_l - \gamma \frac{\partial L}{\partial w_l}$$

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

$$\frac{\partial L}{\partial w_l} = \frac{1}{N} \sum_i \frac{\partial L_i}{\partial w_l} + \lambda \frac{\partial R(W)}{\partial w_l}$$

-Sum gradients for all (partial) training samples for one w_l
-Make one update of W once we have the whole gradient vector (dim: Kx(D+1))

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

ΔW=-γ

$$\Delta W = -\gamma \nabla L$$

$$\frac{\partial L}{\partial w_l} = \frac{1}{N}\sum_i \frac{\partial L_i}{\partial w_l} + \lambda \frac{\partial R(W)}{\partial w_l}$$

-Sum gradients for all (partial) training samples for one w_l
-Make one update of W once we have the whole gradient vector (dim: Kx(D+1))

Gradient of one training sample: SVM loss

- SVM loss

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + d)$$

$$L_i = \sum_{j \neq y_i} \max(0, \mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d)$$

- $\mathbf{w}_j$: j-th row of W
- Loss function of one training sample: $L_i(W; (x_i, y_i))$

Gradient of one training sample

- SVM loss

$$L_i = \sum_{j \neq y_i} \max(0, \mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d)$$

- For row j, $\mathbf{w}_\mathbf{j}$, j = y_i

$$\nabla_{\mathbf{w}_j} L_i = -\left[\sum_{j \neq y_i} \mathbf{I}(\mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d > 0)\right] x_i$$

- For row j, $\mathbf{w}_\mathbf{j}$, j <> y_i

$$\nabla_{\mathbf{w}_j} L_i = \mathbf{I}(\mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d > 0) x_i$$

$\mathbf{I}$(cond) = 1 if cond is true, 0 otherwise

Gradient of one training sample

- SVM loss

$$L_i = \sum_{j \neq y_i} \max(0, \mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d)$$

- For row j, $\mathbf{w_j}$, j = y_i

$$\nabla_{\mathbf{w}_j} L_i = -\left[\sum_{j \neq y_i} \mathbf{I}(\mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d > 0) \right] x_i$$

- Justification:

  All the (K-1) terms of L_i involve w_{yi}; some are zero; for non-zero one, grad = $-x_i$

  $\mathbf{I}$(cond) = 1 if cond is true, 0 otherwise

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$\nabla_{\mathbf{w}_j} L_i = -\left[\sum_{j \neq y_i} \mathbf{I}(\mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d > 0)\right] x_i$$

dim: D+1

ΔW=-γ 

i.e. $-c\, x_i$

c is the number of terms with loss

$$\frac{\partial L}{\partial w_l} = \frac{1}{N}\sum_i \frac{\partial L_i}{\partial w_l} + \lambda \frac{\partial R(W)}{\partial w_l}$$

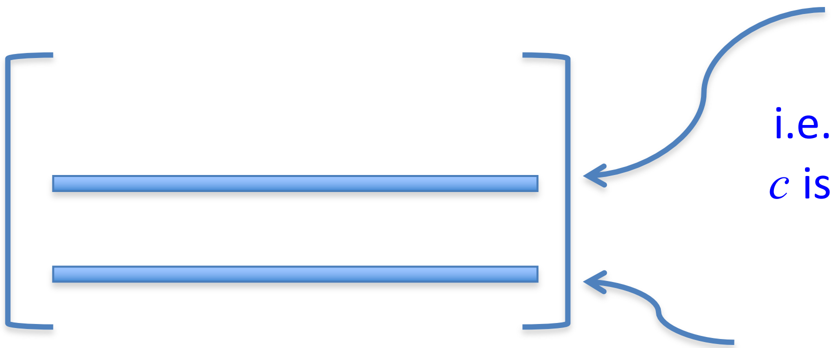
Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$\nabla_{\mathbf{w}_j} L_i = -\left[\sum_{j \neq y_i} \mathbf{I}(\mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d > 0)\right] x_i$$

dim: D+1

ΔW=-γ



i.e. -$c\ x_i$

c is the number of terms with loss

$$\nabla_{\mathbf{w}_j} L_i = \mathbf{I}(\mathbf{w}_j^T x_i - \mathbf{w}_{y_i}^T x_i + d > 0) x_i \quad j{=}y_i$$

dim: D+1

J=i.e. 0 (if no loss due to w_j) or x_i

$$\frac{\partial L}{\partial w_l} = \frac{1}{N} \sum_i \frac{\partial L_i}{\partial w_l} + \lambda \frac{\partial R(W)}{\partial w_l}$$

Learn W by gradient descent

- Mini-batch gradient descent / stochastic gradient descent: use small batch (64, 128, 256) for one update of W

$$\frac{\partial L}{\partial w_l} = \frac{1}{N_{batch}} \sum_i \frac{\partial L_i}{\partial w_l} + \lambda \frac{\partial R(W)}{\partial w_l}$$

- Random sampling without replacement

- Mini-batch: average for each update of W
- An epoch: go through the entire training dataset (multiple updates of W)

Gradient of one training sample: Cross-entropy loss

- Cross-entropy loss

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_{j=1}^{K} e^{f_j}}$$

- Or, p_m is the probability of the m-th class (output of softmax function)

$$p_m = \frac{e^{f_m}}{\sum_{j=1}^{K} e^{f_j}}$$

- Then

$$L_i = -\log p_{y_i}$$

Gradient (linear classifier, cross-entropy loss)

- Gradient matrix (for updating W):

$$\nabla_W L_i =$$

$$\frac{\partial L_i}{\partial \mathbf{w}_m} = p_m x_i \qquad m \neq y_i$$

dim: D+1

$$\frac{\partial L_i}{\partial \mathbf{w}_m} = (p_{y_i} - 1) x_i \qquad m = y_i$$

dim: D+1

See derivation in the document in Moodle

Today’s class

- Image classification:
  - Linear classifier
  - Gradient descent

Next week's class

- ❖ Deep learning
- ❖ Convolutional Neural Networks