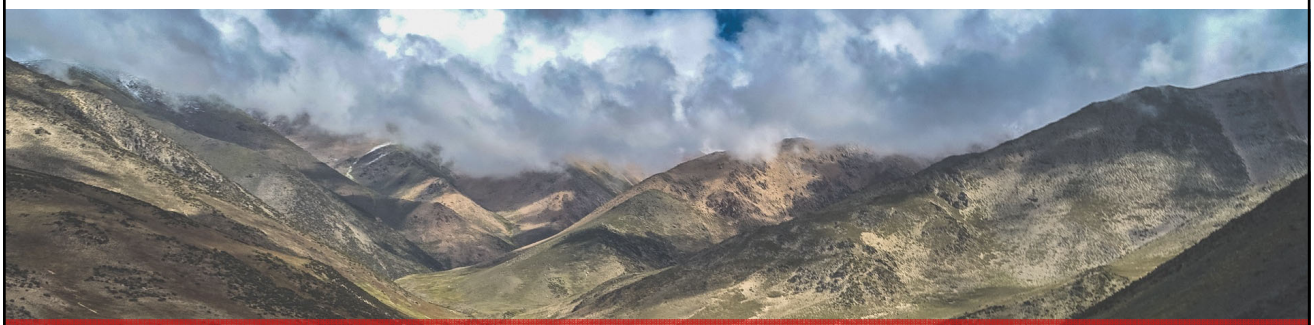


Module 3: Loss functions and optimization

5LSM0: Convolutional neural networks for computer vision

Fons van der Sommen

Electrical Engineering / VCA research group

Last time

(Computer) vision is hard

- Viewpoint alteration
- Illumination
- Deformation
- Occlusion
- Intraclass variation
- Background clutter

Last time

Linear classification

Image → Function → Prediction

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

1	2		
3	4		

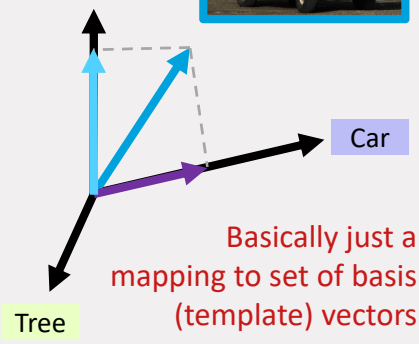
→

1
3
2
4

$\mathbf{x}$

0.2	0.4	-0.1	0.6
0.3	0.8	0.2	-0.1
-0.5	0.6	0.4	-0.7

3.6	Car
2.6	Boat
-3.0	Tree



Basically just a mapping to set of basis (template) vectors

Last time

Feature-based image classification

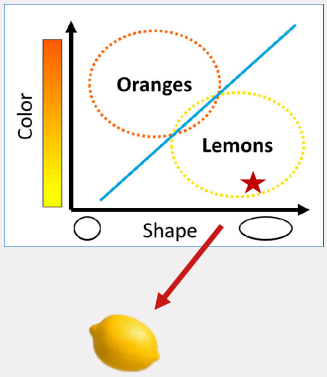
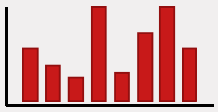
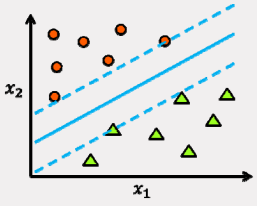


Image → Feature computation → Classification → Prediction

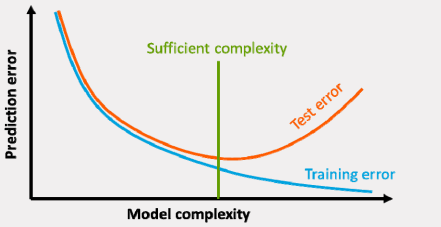


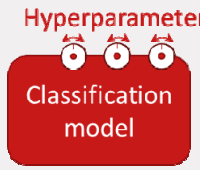


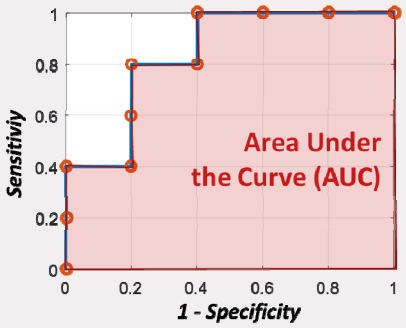
3.6	Boat
2.6	Car
-3.0	Tree

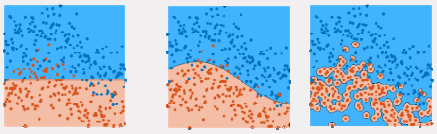
Last time

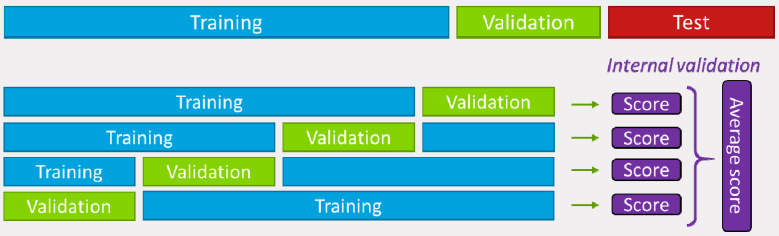
Model performance evaluation











This time

Loss functions

- How do we measure the quality of a certain prediction?
- Hinge-loss
- Cross-entropy loss

Optimization

- How can we achieve minimal loss?

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Loss functions

Loss functions

What would be straightforward loss function?



Car	**4.1**	1.2	-0.1
Boat	2.1	**1.3**	-2.2
Tree	-0.5	1.5	**3.1**

Suppose we have

- 3 training examples
- 3 classes

Linear classifier

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Given a certain weight matrix W

- How good is it?

A loss function yields a score for matrix W, telling us how good it is.

Loss functions



Car	**4.1**	1.2	-0.1
Boat	2.1	**1.3**	-2.2
Tree	-0.5	1.5	**3.1**

Given data set of examples $\{x_i, y_i\}_i^N$,

- x_i is an image, y_i is its label (integer)

We can define a total "loss" as

$$L = \frac{1}{N}\sum L_i(\underbrace{f(x_i, W)}_{=\hat{y}_i}, y_i)$$

… telling us how *unhappy* we are with the current option!

Loss functions



Car	**4.1**	1.2	-0.1
Boat	2.1	**1.3**	-2.2
Tree	-0.5	1.5	**3.1**

Option: multi-class SVM loss

- Classifier yields a score vector s
- We can define loss:

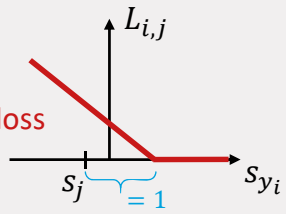
$$L_i = \sum_{i \neq y_i} \begin{cases} 0 & if\ s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{i \neq y_i} \max(0, \underbrace{s_j}_{\text{Score for other classes}} - \underbrace{s_{y_i}}_{\text{Score for true class}} + \underbrace{1}_{\text{Safety margin}})$$

Loss functions

$$L_i = \sum_{i \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Hinge loss





Car	**2.1**	2.1	-0.5
Boat	1.4	**1.3**	1.5
Tree	-0.1	-2.2	**3.1**
Losses:	**0.3**	**1.8**	**0**

$$L_1 = \max(0, 1.4 - 2.1 + 1.0) + \max(0, -0.1 - 2.1 + 1.0) = 0.3$$

$$L_2 = \max(0, 2.1 - 1.3 + 1.0) + \max(0, -2.2 - 1.3 + 1.0) = 1.8$$

$$L_3 = \max(0, -0.5 - 3.1 + 1.0) + \max(0, 1.5 - 3.1 + 1.0) = 0$$

$$L_{total} = \frac{1}{N}\sum_{i=1}^{N} L_i = 0.7$$

*Image from https://www.toolstation.com/chrome-plated-butt-hinge/p36609

Loss functions



Car	**4.1**	1.2	-0.1
Boat	2.1	**1.3**	-2.2
Tree	-0.5	1.5	**3.1**
Losses:	**0.3**	**1.8**	**0**

Some questions:

I. What happens to the loss if the tree scores change a bit?

II. What is the min/max possible loss?

III. At initialization the weights of W are so small, $s \approx \mathbf{0}$. What is the loss?

IV. What if the sum was over all classes?

V. What if we used mean instead of sum?

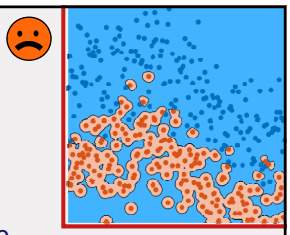
VI. What if we used a square loss?

$$L_i = \sum_{i \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

VII. If we found a W that achieves zero loss, is this W unique?

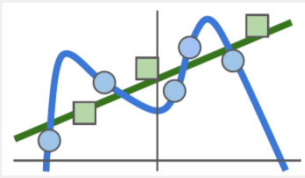
Also see cs231n lecture 3 slides 18-23

Loss functions



Observation

- The hinge loss only cares about being right on the training data, nothing more…
- Question: what are you prone to when using this approach?



*

Regularization

- Add a term express your preference for (the properties of) W

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) \; + \; \lambda R(W)$$

Loss: penalty for being wrong about the data

Regularizer: penalty for using undesirable W

Occam's Razor: go for the least complex option!

*Figure from cs231n lecture 3 slide 29

Loss functions

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Hyperparameter

Some commonly used regularizers

- L_2 regularization $R_{L2}(W) = \sum_k W_k^2$ → Favors small weights
  $w_{L2} = [0.25 \quad 0.25 \quad 0.25 \quad 0.25]$
- L_1 regularization $R_{L1}(W) = \sum_k |W_k|$ → Favors sparsity
  $w_{L1} = [0 \quad 1 \quad 0 \quad 0]$
- Elastic net $R_E(W) = \beta R_{L2} + (1-\beta)R_{L1}$
- Drop-out
- Batch normalization

Techniques used for training ConvNets

→ *More on this in Module 6!*

Example

$x = [1 \quad 1 \quad 1 \quad 1]$

$w_{L2}^T x = w_{L1}^T x = 1$

Loss functions

Q: Why is there is minus in front of the log?

Car	**4.1**
Boat	2.1
Tree	-0.5

Cross-entropy loss

- Use scores to compute a probability distribution over the classes

Softmax function

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s_i = \left(f(x_i, W)\right)$$

- Aim: maximize the probability for the right class $P(Y = y_i|X = x_i)$
- Equivalent to maximizing $\log\left(P(Y = y_i|X = x_i)\right)$, which is mathematically easier
- This whould yield a score for desirability, hence the *loss* would be

$$L_i = -\log P(Y = k|X = x_i) = -\log\left(\frac{e^{s_k}}{\sum_j e^{s_j}}\right)$$

Loss functions

Q2: Min/max of softmax loss?

Q3: How do the scores look for zero loss?

Q4: What is the loss at initialization? ($s \approx 0$)

$$L_i = -\log\left(\frac{e^{s_k}}{\sum_j e^{s_j}}\right)$$

	Unnormalized log-probabilities	exp →	Unormalized probabilities	normalize →	Probabilities
Car	**4.1**		**60.3**		**0.87**
Boat	2.1		8.2		0.12
Tree	-0.5		0.6		0.01

Loss:

$$L_i = -\log(0.87) = 0.14$$

Q1: Does this loss care about the scores for the other classes?

Loss functions

Where are we now?

We have

- some data set (x_i, y_i), data points x_i and labels y_i
- a score function $s = f(x, W)$, which could for example be linear, i.e. $Wx + b$
- a loss function to measure the badness of the proposed score function $f(x, W)$

Cross-entropy loss $L_i = -\log\left(\frac{e^{s_k}}{\sum_j e^{s_j}}\right)$

Hinge loss $L_i = \sum_{i \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Total loss $L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$

Cool! … but how can we find the best W??

TU/e

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Optimization

Not required, but again a huge shout-out to pexels.com for these nice images!

Optimization

Follow the slope

- Slope is equivalent to the derivative of a function $\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$
- The gradient is a vector of partial derivatives
- Compute the gradient to find the steepest slope
  - *The gradient points towards the direction of greatest increase*
- Then move on the way down using the opposite direction of the maximum gradient direction
- Note: you can find the slope in any direction by taking the dot product of the unit vector in that direction with the gradient at that point

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

Optimization

How could you evaluate the gradient for a given point (W)

- Option: use finite differences to evaluate the effect of a small step in every direction

Current W | Current W | h_1 | (approximate) gradient dW

$$\begin{bmatrix} 0.42 \\ 2.12 \\ -0.42 \\ 5.32 \\ -2.97 \\ -0.04 \\ 2.55 \end{bmatrix} \quad \begin{bmatrix} 0.42 \\ 2.12 \\ -0.42 \\ 5.32 \\ -2.97 \\ -0.04 \\ 2.55 \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} -0.7 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \frac{(2.94827 - 2.94834)}{0.0001} = -0.7$$

Loss 2.94834 | Loss 2.94827

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

Optimization

How could you evaluate the gradient for a given point (W)

- Option: use finite differences to evaluate the effect of a small step in every direction

Current W | Current W | h_1 | (approximate) gradient dW

$$\begin{bmatrix} 0.42 \\ 2.12 \\ -0.42 \\ 5.32 \\ -2.97 \\ -0.04 \\ 2.55 \end{bmatrix} \quad \begin{bmatrix} 0.42 \\ 2.12 \\ -0.42 \\ 5.32 \\ -2.97 \\ -0.04 \\ 2.55 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.0001 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} -0.7 \\ 0.8 \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \frac{(2.94842 - 2.94834)}{0.0001} = 0.8$$

Loss 2.94834 | Loss 2.94842

Optimization

How could you evaluate the gradient for a given point (W)

- Option: use finite differences to evaluate the effect of a small step in every direction

Current W | Current W | h_1 | (approximate) gradient dW

$$\begin{bmatrix} 0.42 \\ 2.12 \\ -0.42 \\ 5.32 \\ -2.97 \\ -0.04 \\ 2.55 \end{bmatrix} \quad \begin{bmatrix} 0.42 \\ 2.12 \\ -0.42 \\ 5.32 \\ -2.97 \\ -0.04 \\ 2.55 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.0001 \end{bmatrix} \quad \begin{bmatrix} -0.7 \\ 0.8 \\ 1.2 \\ 4.5 \\ -2.7 \\ 3.3 \\ -5.2 \end{bmatrix}$$

Loss 2.94834 | Loss 2.94842

This is terrible…

$$\frac{(2.94842 - 2.94834)}{0.0001} = -5.2$$

Optimization

Much better solution:

- Calculus!
- The loss is just a function of W

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

- We can derive an analytical solution for the gradient $\nabla_W L$

Gradient of the loss function with respect to W

(tells us what happens to the loss if we change W)

Optimization

How could you evaluate the gradient for a given point (W)

- Option: use finite differences to evaluate the effect of a small step in every direction

Current W

$$\begin{bmatrix} 0.42 \\ 2.12 \\ -0.42 \\ 5.32 \\ -2.97 \\ -0.04 \\ 2.55 \end{bmatrix}$$

Analytic expression

gradient dW

$$\begin{bmatrix} -0.7 \\ 0.8 \\ 1.2 \\ 4.5 \\ -2.7 \\ 3.3 \\ -5.2 \end{bmatrix}$$

Much better…

Q: What COULD we use the finite differences method for?

Optimization

How does this work?

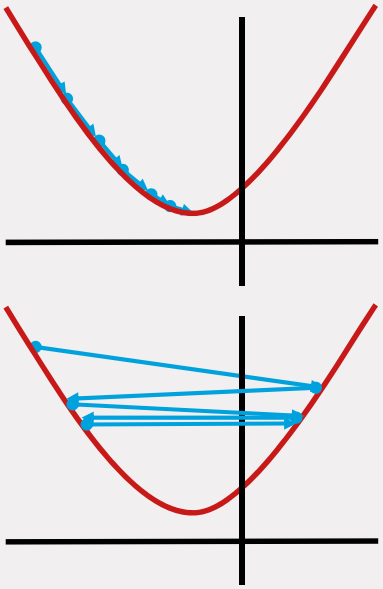
- Start at some point W in your loss landscape
- Compute the gradient $\nabla_W L$
- Take a step in that direction

Q: What is a crucial parameter in this procedure?

- Update function:

$$W \leftarrow W - \eta \nabla_W L$$

Notice that you don't actually know at what loss value you'll end up… You just move left or right



Optimization

How does this work?

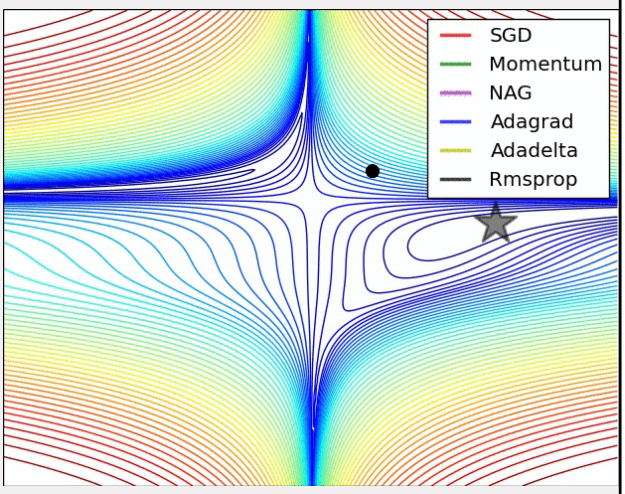
- Start at some point W in your loss landscape
- Compute the gradient $\nabla_W L$
- Take a step in that direction

Q: What is a crucial parameter in this procedure?

- Update function:

$$W \leftarrow W - \eta \nabla_W L$$

Some alternatives…



Optimization

Stochastic Gradient Descent (SGD)

- Our loss function waits with updating until it has seen all the data points in the set

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Vanilla gradient descent

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(f(x_i, W), y_i) + \lambda \nabla_W R(W)$$

Sample mini-batches without replacement

Period to check all samples in the training data is called an **epoch**

- But N can be very large, so it can take ages before you take a step…
  - *Both the loss and the gradient require you to wait*
- Use "mini-batches" instead, to approximate the loss function (and its gradient)
  - *Randomly sample (if your batch size = 1 → Stochastic Gradient Descent (SGD)*

Summary

Loss functions

- Use to determine the quality of a certain prediction
- Typically includes a regularization term to control the complexity
- Commonly used loss functions
  - *Hinge loss:* $L_i = \sum_{i \neq y_i} \max(0, s_j - s_{y_i} + 1)$ → *enforces a margin of 1 between true class and other classes*
  - *Cross-entropy loss:* $L_i = -\log\left(\frac{e^{s_k}}{\sum_j e^{s_j}}\right)$ → *converts class scores into probabilities*

Optimization

- Use to find the best W (i.e. the one yielding minimal loss)
- Compute gradient of the loss function to determine the direction to move
- (Mini-batch / stochastic) gradient descent

Next time

- **How can we efficiently find the gradient of the loss function?**
- **How can we combine linear classifiers to perform non-linear classification?**