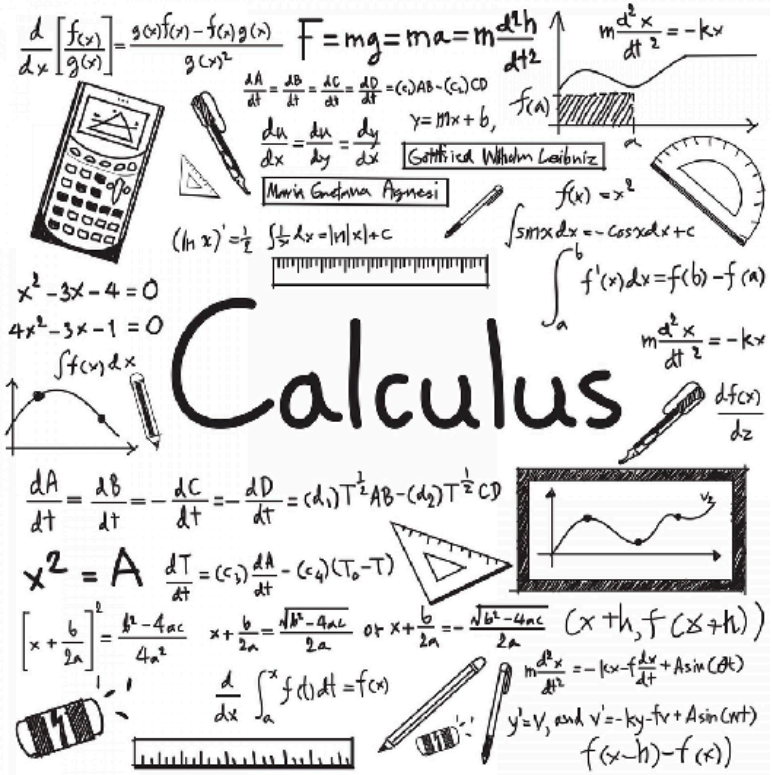
Introduction to (Deep) Learning

Gradient

slides adapted from courses.d2l.ai/berkeley-stat-157

Matrix



slides adapted from courses.d2l.ai/berkeley-stat-157

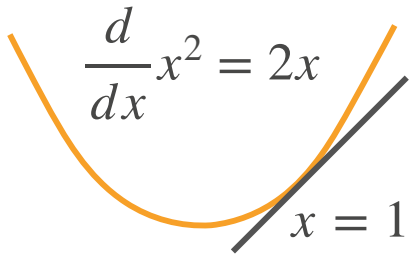
Review Scalar Derivative

y	a	x^n	$\exp(x)$	$\log(x)$	$\sin(x)$
$\frac{dy}{dx}$	0	nx^{n-1}	$\exp(x)$	$\frac{1}{x}$	$\cos(x)$

a is not a function of x

y	$u+v$	uv	$y=f(u),\ u=g(x)$
$\frac{dy}{dx}$	$\frac{du}{dx}+\frac{dv}{dx}$	$\frac{du}{dx}v+\frac{dv}{dx}u$	$\frac{dy}{du}\frac{du}{dx}$

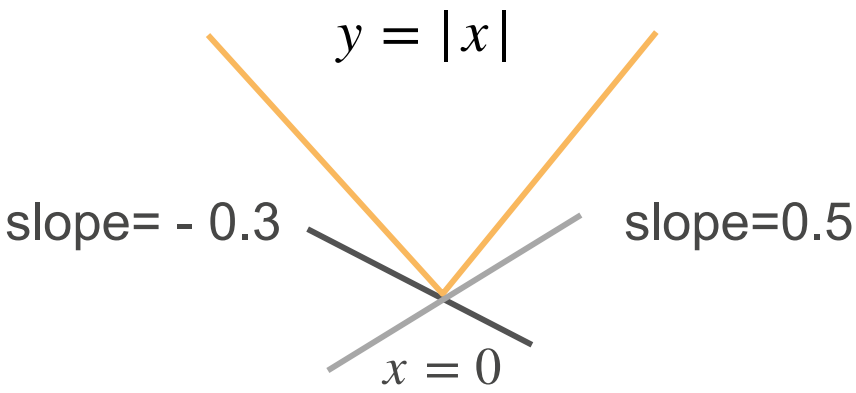
Derivative is the slope of the tangent line



The slope of the tangent line is 2

Subderivative

- Extend derivative to non-differentiable cases



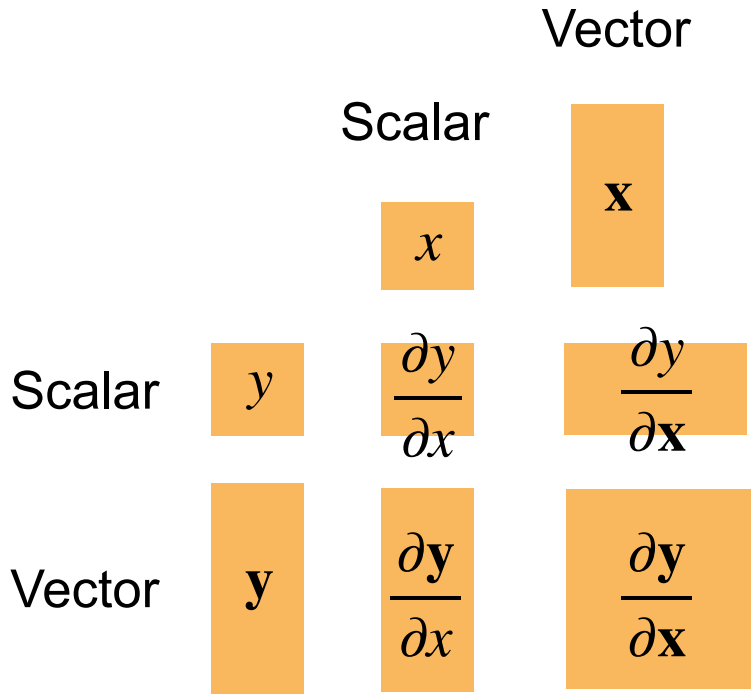
$$\frac{\partial |x|}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ a & \text{if } x = 0, \quad a \in [-1,1] \end{cases}$$

Another example:

$$\frac{\partial}{\partial x} \max(x,0) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ a & \text{if } x = 0, \quad a \in [0,1] \end{cases}$$

Gradients

- Generalize derivatives into vectors



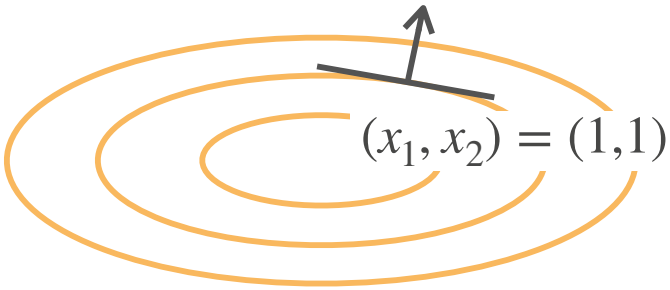
$\partial y / \partial \mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_n} \right]$$

	x	$\mathbf{x}$
y	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \mathbf{x}}$
$\mathbf{y}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

$$\frac{\partial}{\partial \mathbf{x}} x_1^2 + 2x_2^2 = [2x_1, 4x_2]$$

Direction (2, 4), perpendicular to the contour lines



Examples

y	a	au	$\text{sum}(\mathbf{x})$	$\|\mathbf{x}\|^2$
$\frac{\partial y}{\partial \mathbf{x}}$	$\mathbf{0}^T$	$a\frac{\partial u}{\partial \mathbf{x}}$	$\mathbf{1}^T$	$2\mathbf{x}^T$

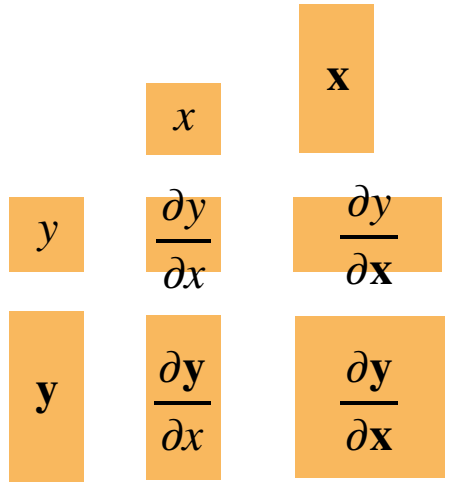
a is not a function of $\mathbf{x}$

$\mathbf{0}$ and $\mathbf{1}$ are vectors

y	$u + v$	uv	$\langle \mathbf{u}, \mathbf{v} \rangle$
$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}}v + \frac{\partial v}{\partial \mathbf{x}}u$	$\mathbf{u}^T\frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^T\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

$\partial \mathbf{y} / \partial x$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \qquad \frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}$$



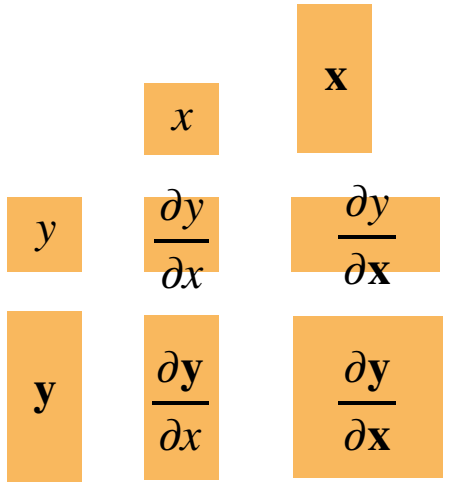
$\partial y / \partial \mathbf{x}$ is a row vector, while $\partial \mathbf{y} / \partial x$ is a column vector

It is called numerator-layout notation. The reversed version is called denominator-layout notation

$$\partial \mathbf{y} / \partial \mathbf{x}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial \mathbf{x}} \\ \frac{\partial y_2}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial y_m}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \ldots, \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}, \ldots, \frac{\partial y_2}{\partial x_n} \\ \vdots \\ \frac{\partial y_m}{\partial x_1}, \frac{\partial y_m}{\partial x_2}, \ldots, \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$



Examples

$\mathbf{y}$	$\mathbf{a}$	$\mathbf{x}$	$\mathbf{Ax}$	$\mathbf{x}^T\mathbf{A}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{A}$	$\mathbf{A}^T$

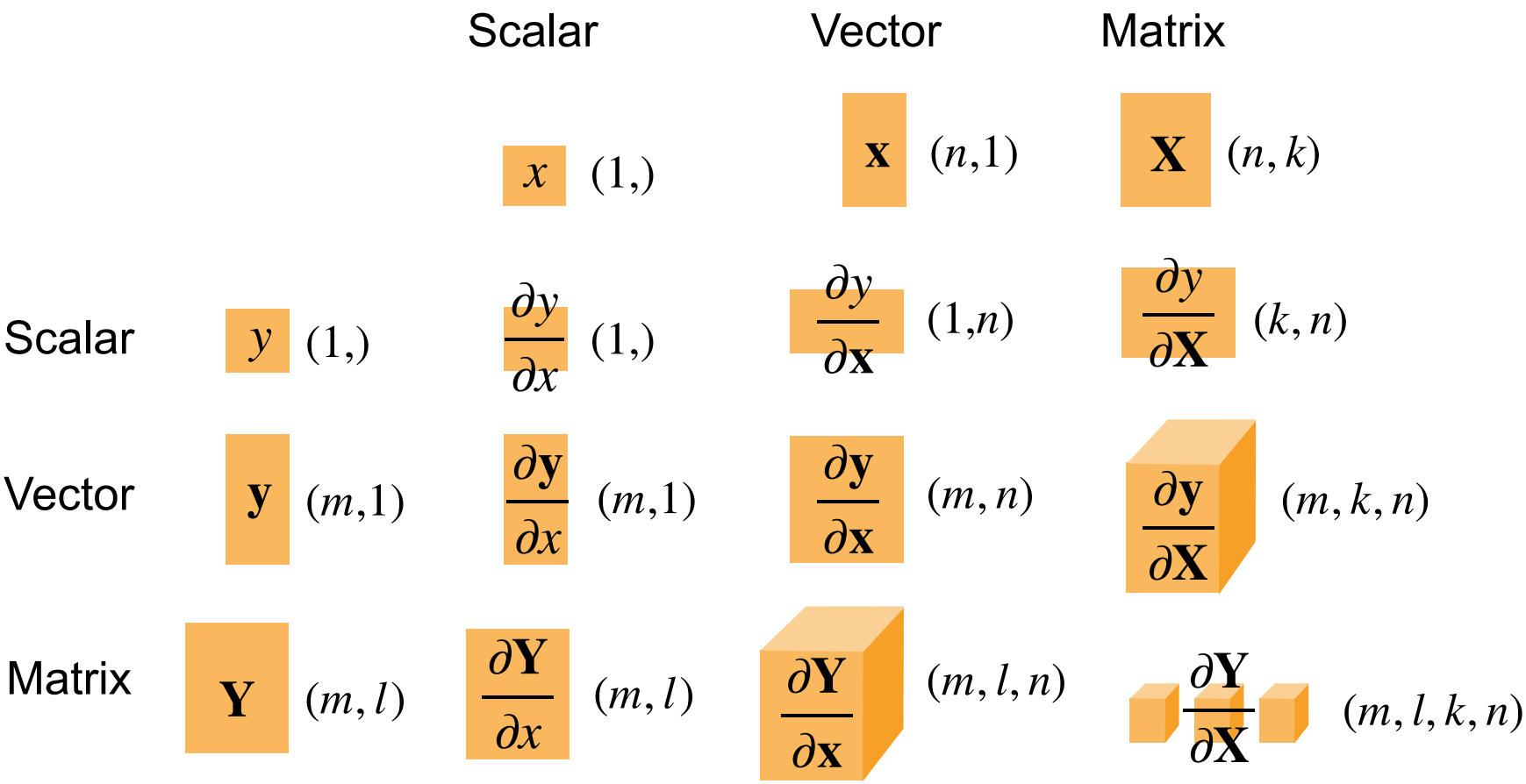
$\mathbf{y}$	$a\mathbf{u}$	$\mathbf{Au}$	$\mathbf{u} + \mathbf{v}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$a\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\mathbf{A}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$

$\mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{m \times n}$

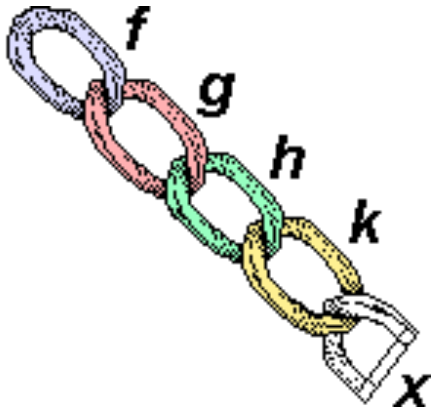
a, $\mathbf{a}$ and $\mathbf{A}$ are not functions of $\mathbf{x}$

$\mathbf{0}$ and $\mathbf{I}$ are matrices

Generalize to Matrices



Chain Rule



slides adapted from courses.d2l.ai/berkeley-stat-157

Generalize to Vectors

- Chain rule for scalars:

$$y = f(u),\ u = g(x) \qquad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

- Generalize to vectors straightforwardly

$$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial \mathbf{x}} \qquad \frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$$(1,n)\quad (1,)\ (1,n) \qquad (1,n)\quad (1,k)\ (k,n) \qquad (m,n)\ (m,k)\ (k,n)$$

Example 1

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

Assume $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, \quad y \in \mathbb{R}$

$$z = \left(\langle \mathbf{x}, \mathbf{w} \rangle - y\right)^2$$

Compute $\frac{\partial z}{\partial \mathbf{w}}$

$$\begin{aligned}
\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial \mathbf{w}} \\
&= \frac{\partial b^2}{\partial b} \frac{\partial a - y}{\partial a} \frac{\partial \langle \mathbf{x}, \mathbf{w} \rangle}{\partial \mathbf{w}} \\
&= 2b \cdot 1 \cdot \mathbf{x}^T \\
&= 2\left(\langle \mathbf{x}, \mathbf{w} \rangle - y\right) \mathbf{x}^T
\end{aligned}$$

Decompose

$$\begin{aligned}
a &= \langle \mathbf{x}, \mathbf{w} \rangle \\
b &= a - y \\
z &= b^2
\end{aligned}$$

Example 2

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

Assume $\mathbf{X} \in \mathbb{R}^{m \times n}, \quad \mathbf{w} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m$

$$z = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

Compute $\frac{\partial z}{\partial \mathbf{w}}$

Decompose

$$\mathbf{a} = \mathbf{X}\mathbf{w}$$
$$\mathbf{b} = \mathbf{a} - \mathbf{y}$$
$$z = \|\mathbf{b}\|^2$$

$$\begin{aligned}
\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial \mathbf{b}} \frac{\partial \mathbf{b}}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \\
&= \frac{\partial \|\mathbf{b}\|^2}{\partial \mathbf{b}} \frac{\partial \mathbf{a} - \mathbf{y}}{\partial \mathbf{a}} \frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} \\
&= 2\mathbf{b}^T \times \mathbf{I} \times \mathbf{X} \\
&= 2\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T \mathbf{X}
\end{aligned}$$

Introduction to (Deep) Learning

Multilayer Perceptron

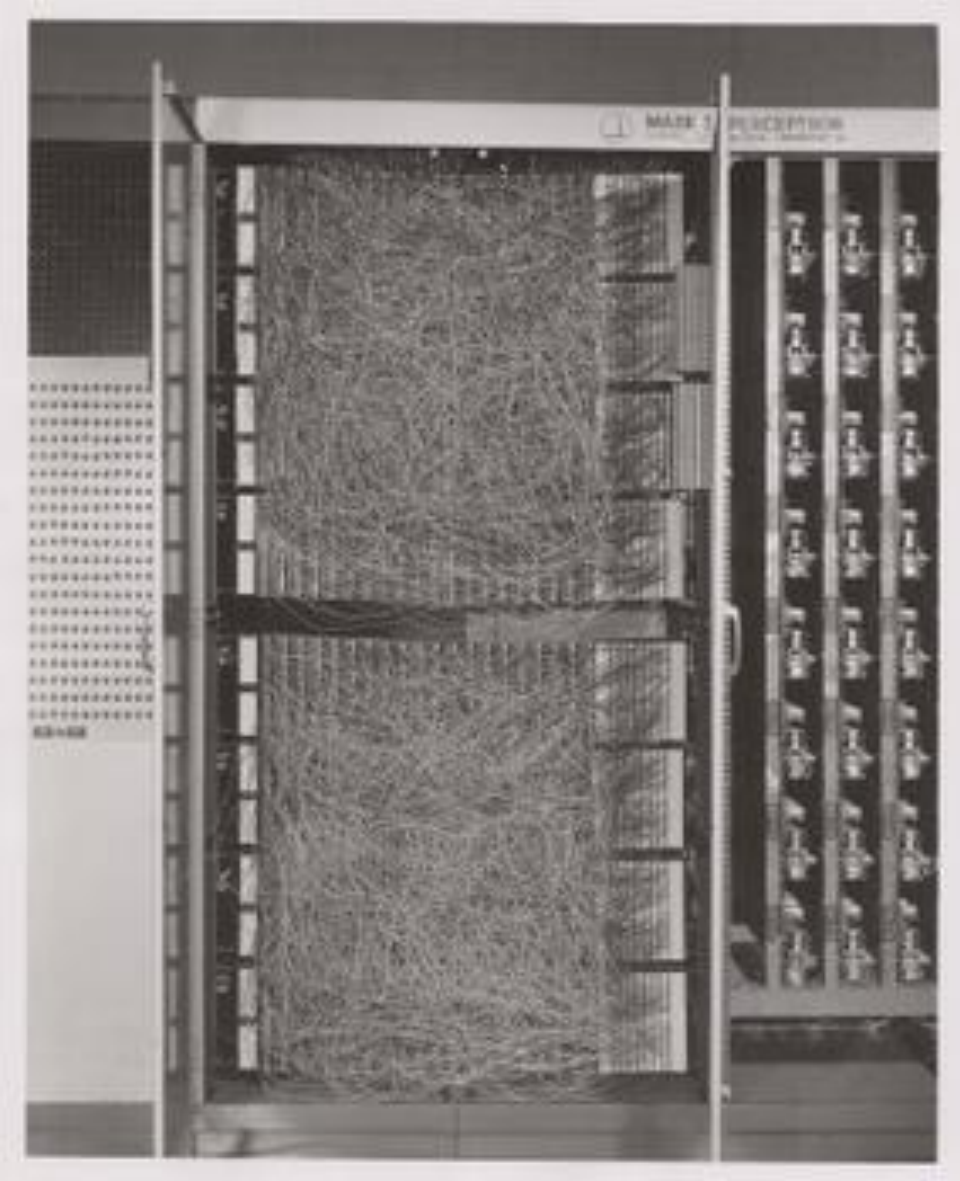
slides adapted from courses.d2l.ai/berkeley-stat-157

Outline

- **Single Layer Perceptron**
  - Decision Boundary
  - XOR is hard
- **Multilayer Perceptron**
  - Layers
  - Nonlinearities
  - Computational Cost

Perceptron

Mark I Perceptron, 1960
(wikipedia.org)

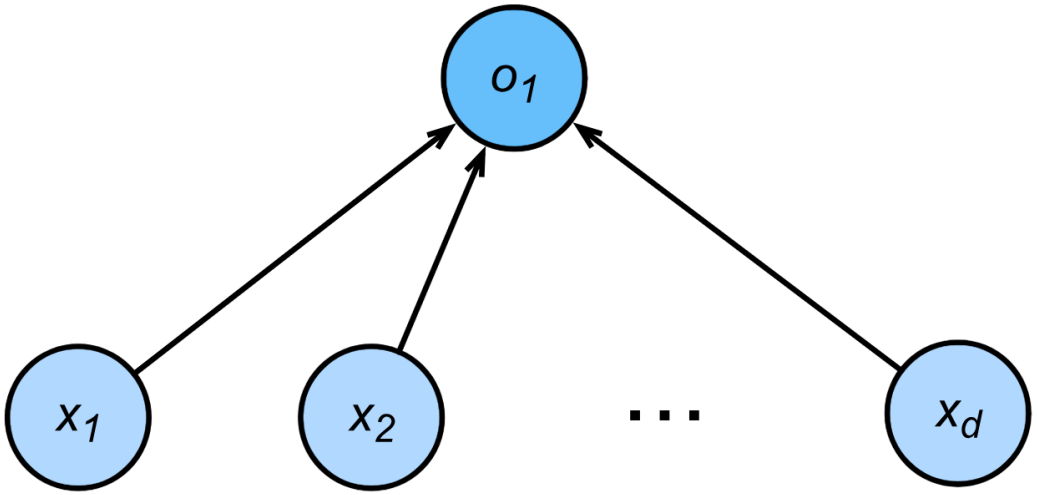


slides adapted from courses.d2l.ai/berkeley-stat-157

Perceptron

- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$o = \sigma\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) \qquad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

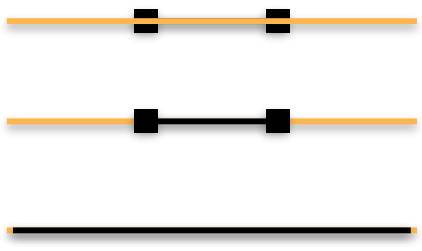


Perceptron

- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

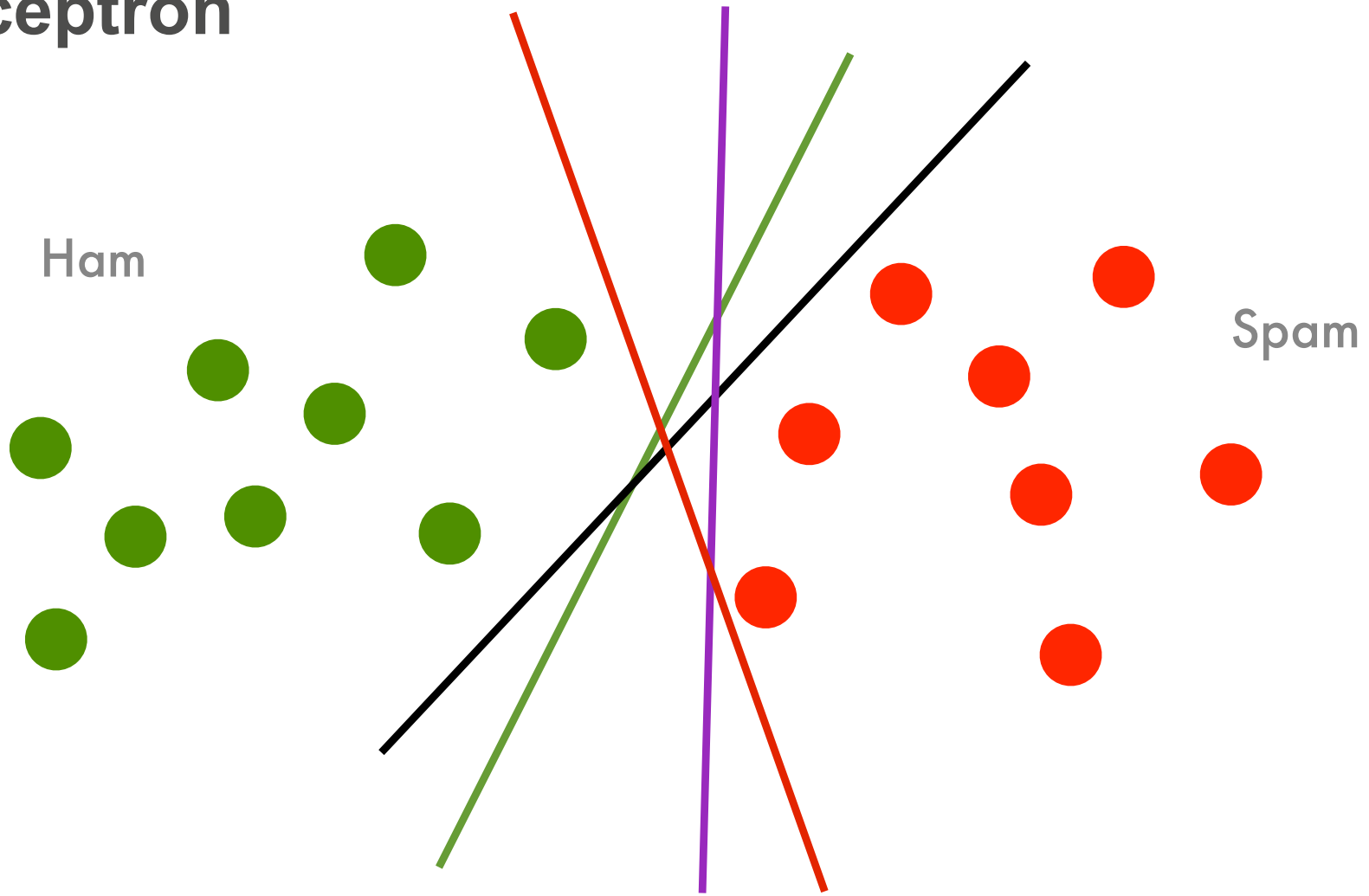
$$o = \sigma\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) \qquad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Binary classification (0 or 1)
  - Vs. scalar real value for regression
  - Vs. probabilities for logistic regression



slides adapted from courses.d2l.ai/berkeley-stat-157

Perceptron



slides adapted from courses.d2l.ai/berkeley-stat-157

Training the Perceptron

initialize $w = 0$ and $b = 0$

repeat

  if $y_i \left[\langle w, x_i \rangle + b\right] \leq 0$ **then**

    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$

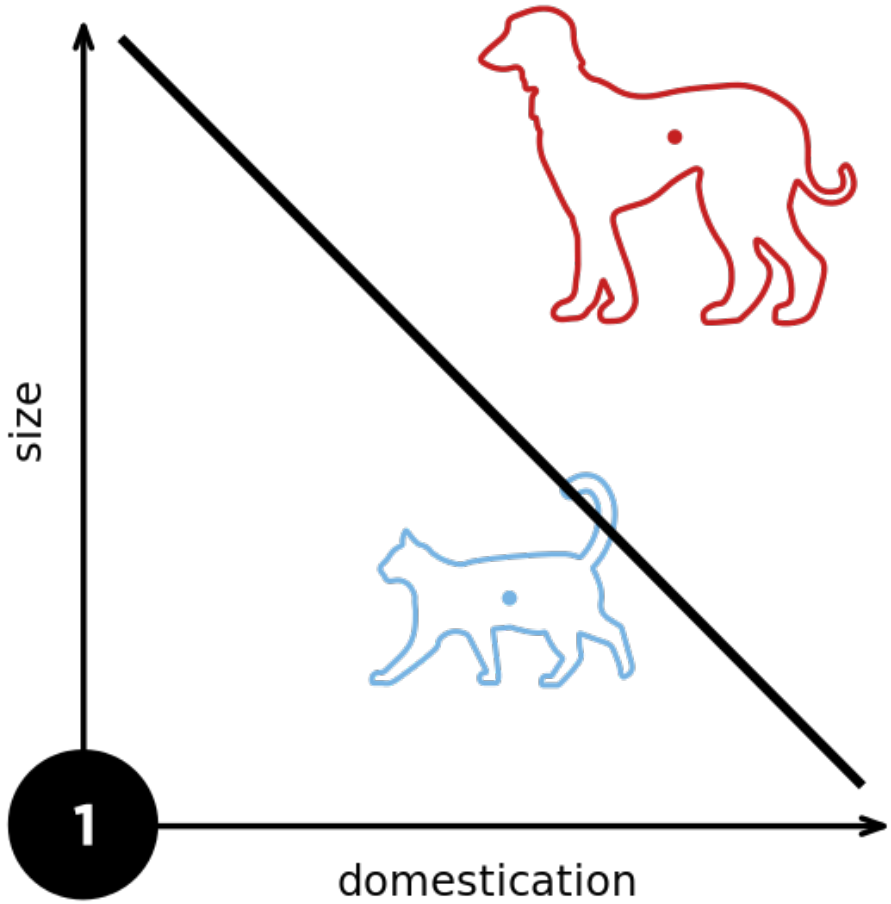
  end if

until all classified correctly

Equals to SGD (batch size is 1) with the following loss

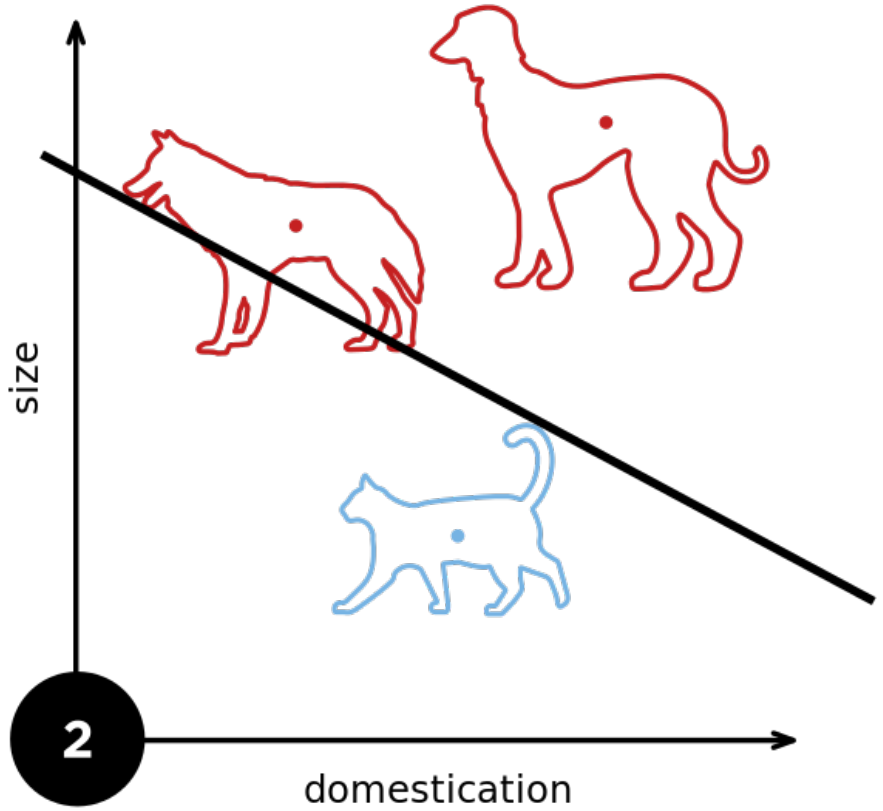
$$\ell(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\langle \mathbf{w}, \mathbf{x} \rangle)$$

slides adapted from courses.d2l.ai/berkeley-stat-157



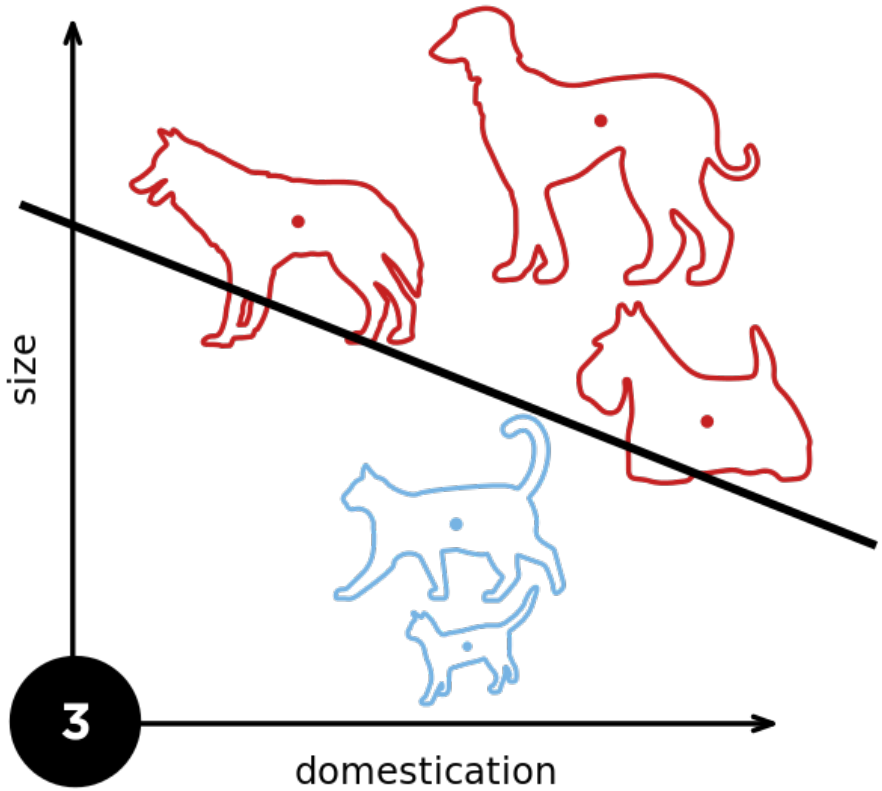
slides adapted from courses.d2l.ai/berkeley-stat-157

From wikipedia



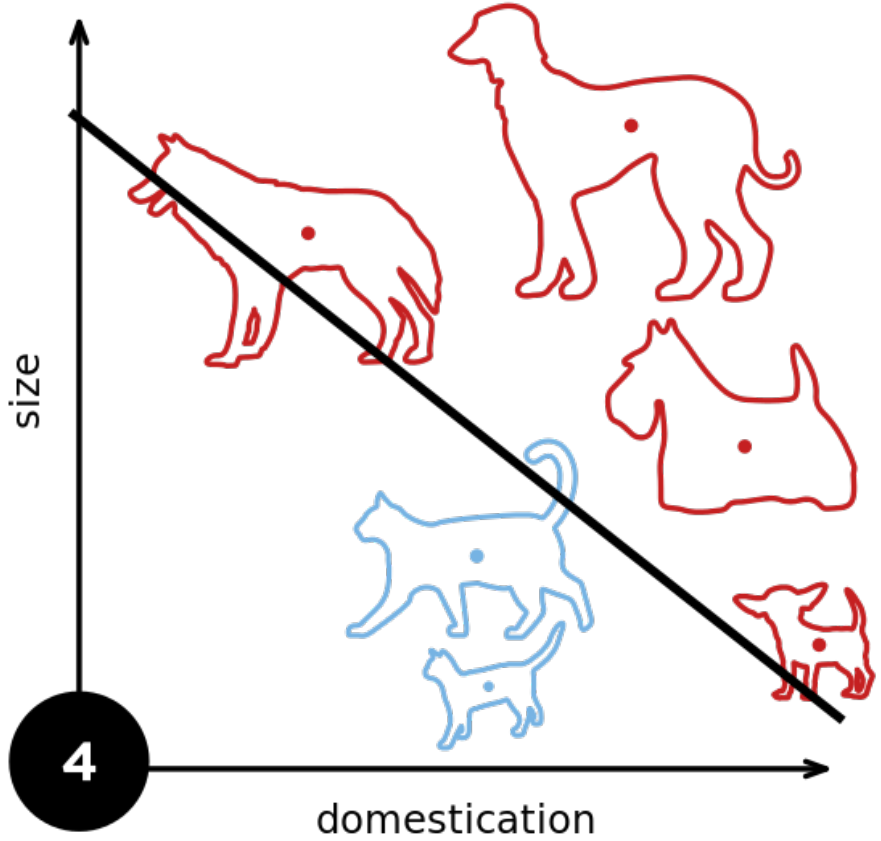
slides adapted from courses.d2l.ai/berkeley-stat-157

From wikipedia



slides adapted from courses.d2l.ai/berkeley-stat-157

From wikipedia



slides adapted from courses.d2l.ai/berkeley-stat-157

From wikipedia

Convergence Theorem

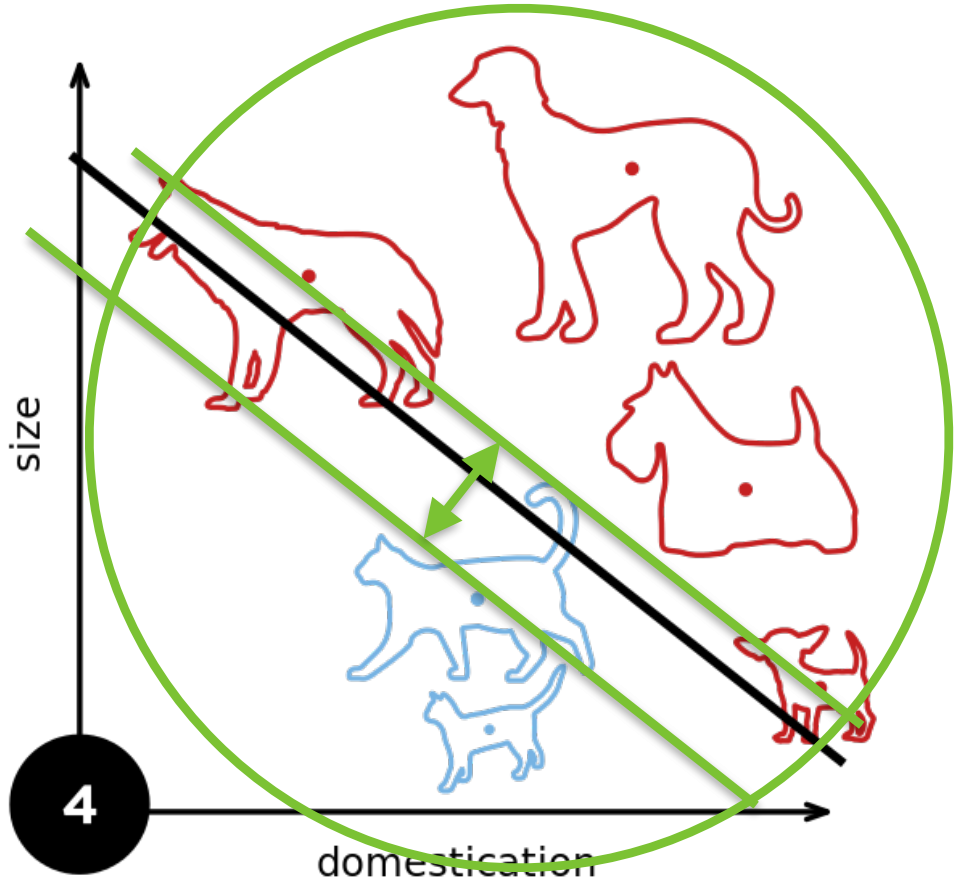
- Radius r enclosing the data
- Margin ρ separating the classes

$$y(\mathbf{x}^\top \mathbf{w} + b) \geq \rho$$

for $\|\mathbf{w}\|^2 + b^2 \leq 1$

- Guaranteed that perceptron will converge after

$$\frac{r^2 + 1}{\rho^2} \text{ steps}$$

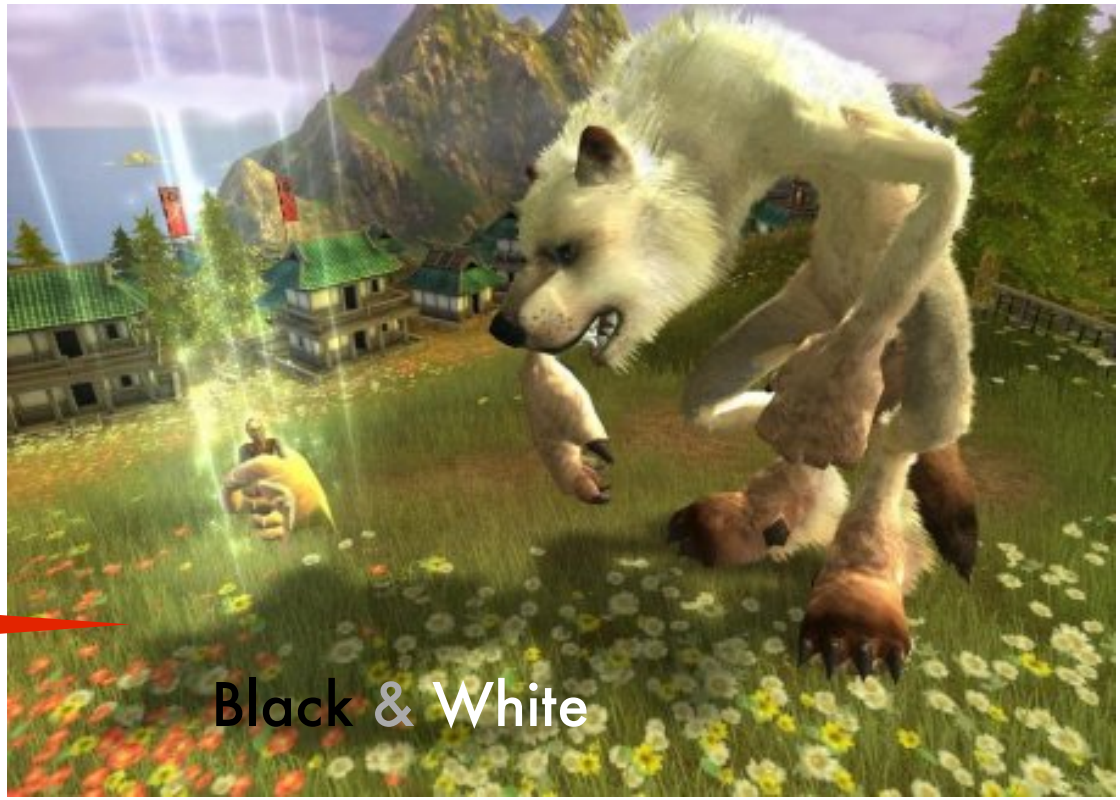


slides adapted from courses.d2l.ai/berkeley-stat-157

Consequences

- Only need to store errors. This gives a compression bound for perceptron.
- Fails with noisy data

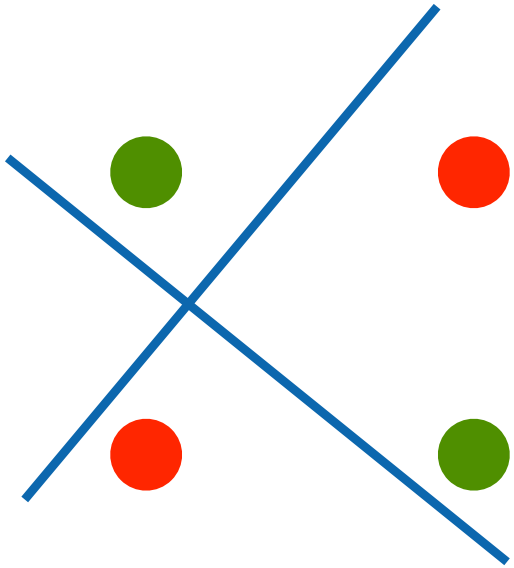
do NOT train your avatar with perceptrons



slides adapted from courses.d2l.ai/berkeley-stat-157

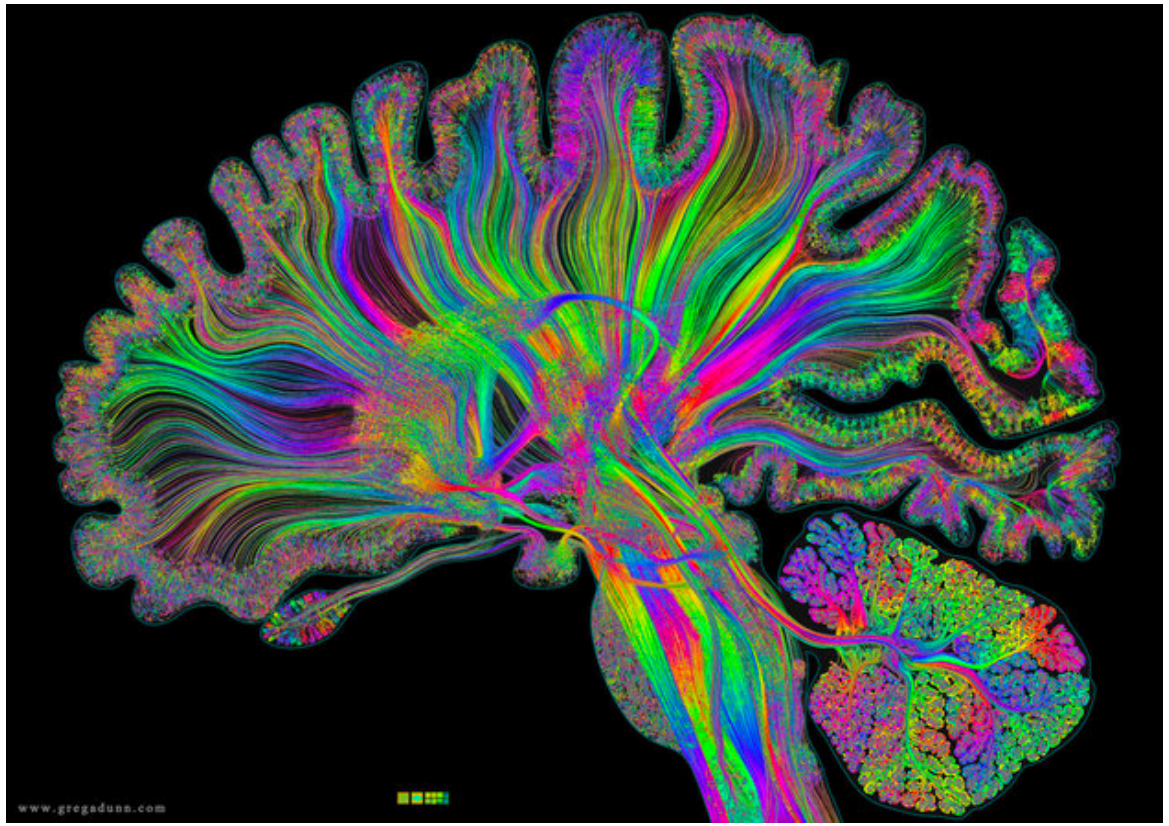
XOR Problem (Minsky & Papert, 1969)

The perceptron cannot learn an XOR function
(neurons can only generate linear separators)



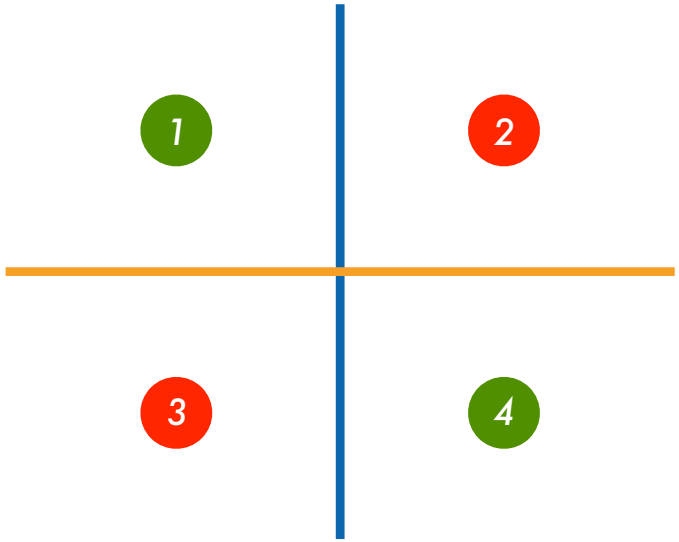
slides adapted from courses.d2l.ai/berkeley-stat-157

Multilayer Perceptron

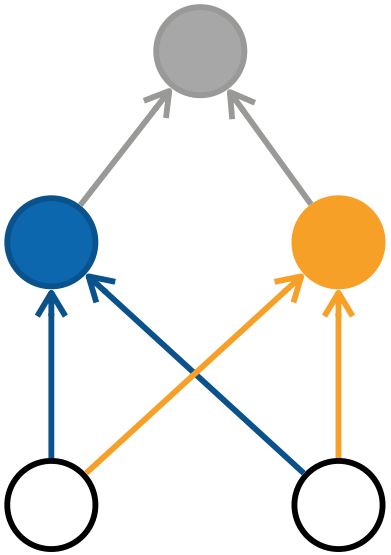


slides adapted from courses.d2l.ai/berkeley-stat-157

Learning XOR

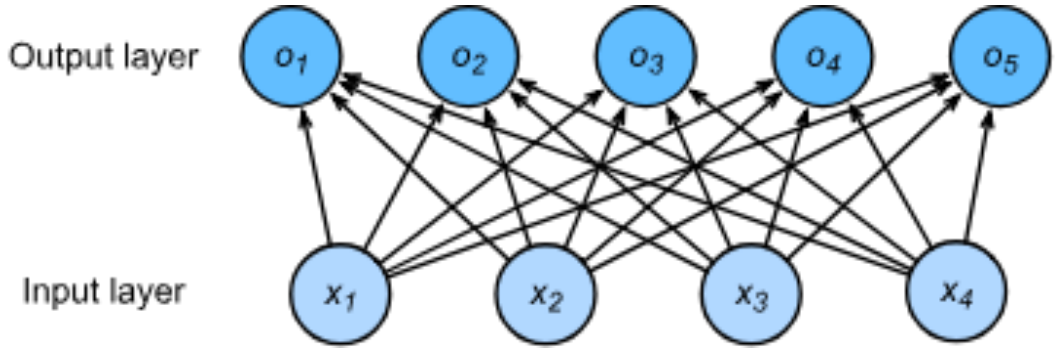


	1	2	3	4
	+	-	+	-
	+	+	-	-
product	+	-	-	+



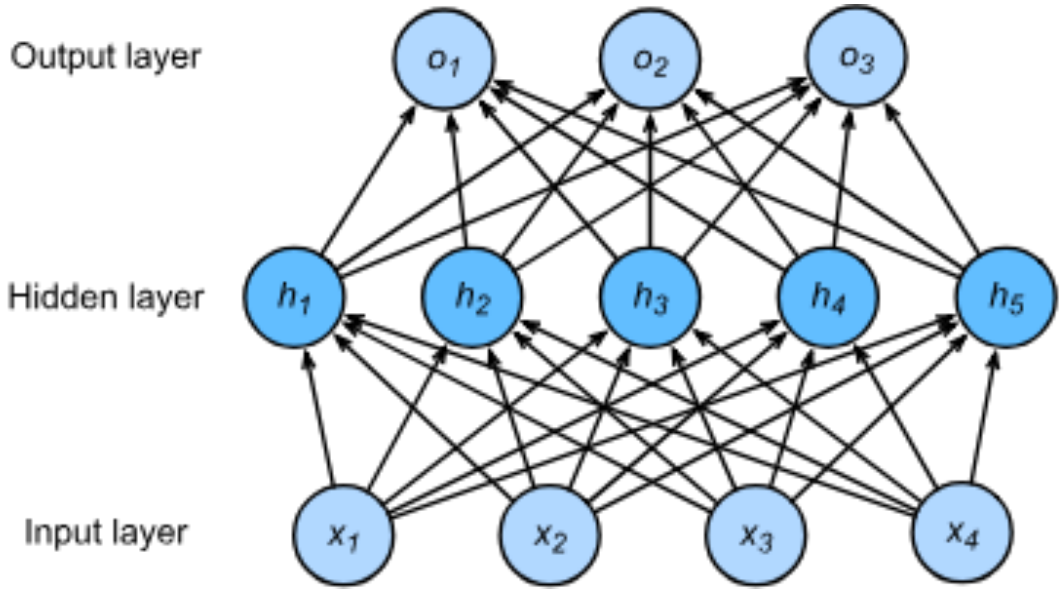
slides adapted from courses.d2l.ai/berkeley-stat-157

Single Hidden Layer



slides adapted from courses.d2l.ai/berkeley-stat-157

Single Hidden Layer



Hyperparameter - size m of hidden layer

slides adapted from courses.d2l.ai/berkeley-stat-157

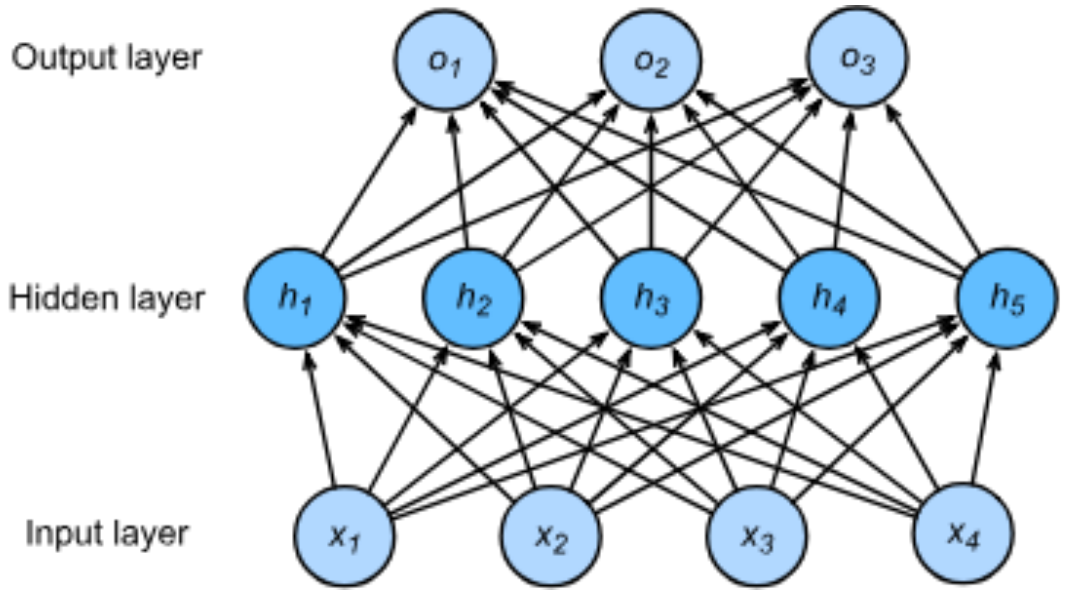
Single Hidden Layer

- Input $\mathbf{x} \in \mathbb{R}^n$
- Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- Output $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + b_2$$

σ is an element-wise activation function



slides adapted from courses.d2l.ai/berkeley-stat-157

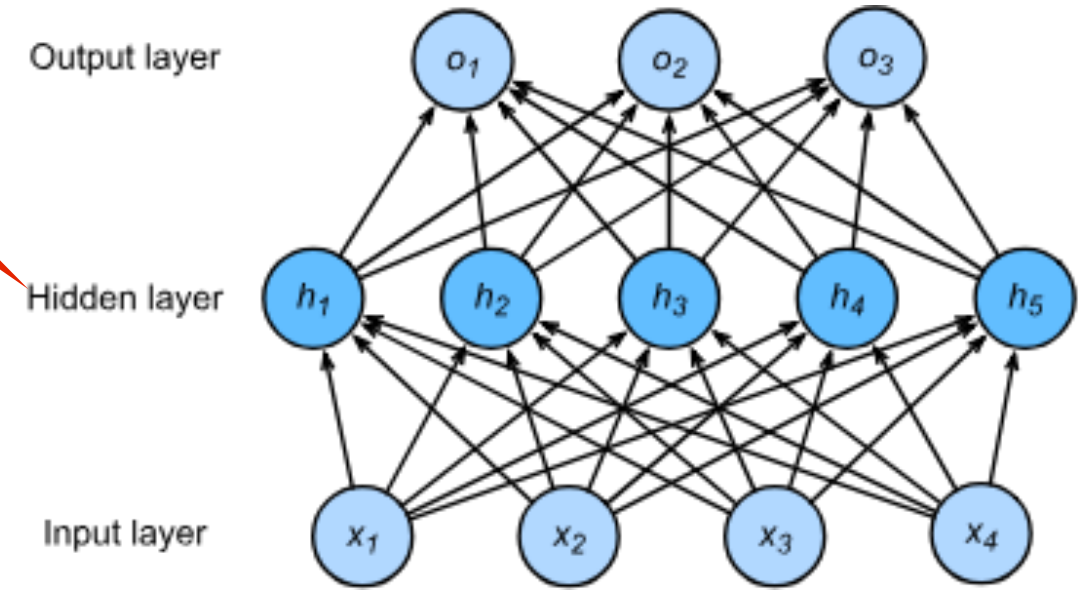
Single Hidden Layer

Why do we need an a nonlinear activation?

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + b_2$$

σ is an element-wise activation function



slides adapted from courses.d2l.ai/berkeley-stat-157

Single Hidden Layer

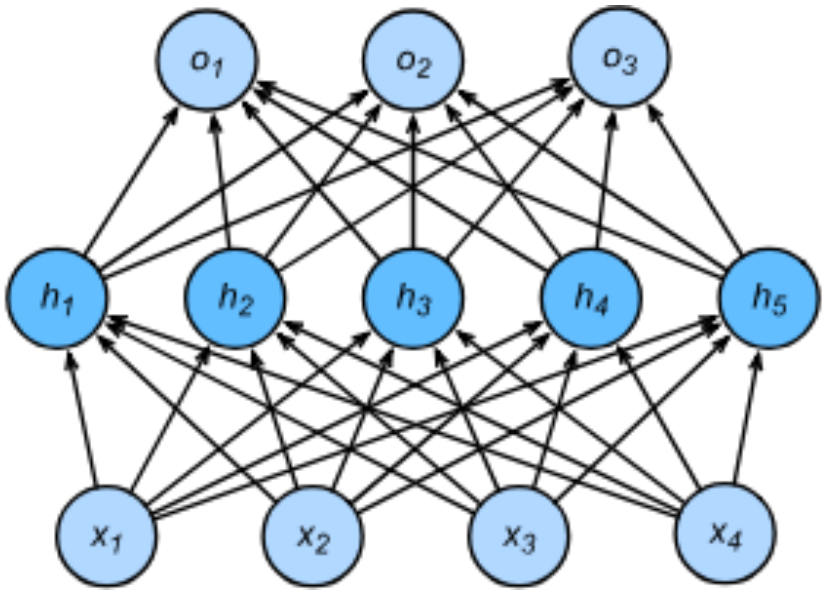
Why do we need an a nonlinear activation?

$$\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + b_2$$

hence $o = \mathbf{w}_2^\top\mathbf{W}_1\mathbf{x} + b'$

Linear …

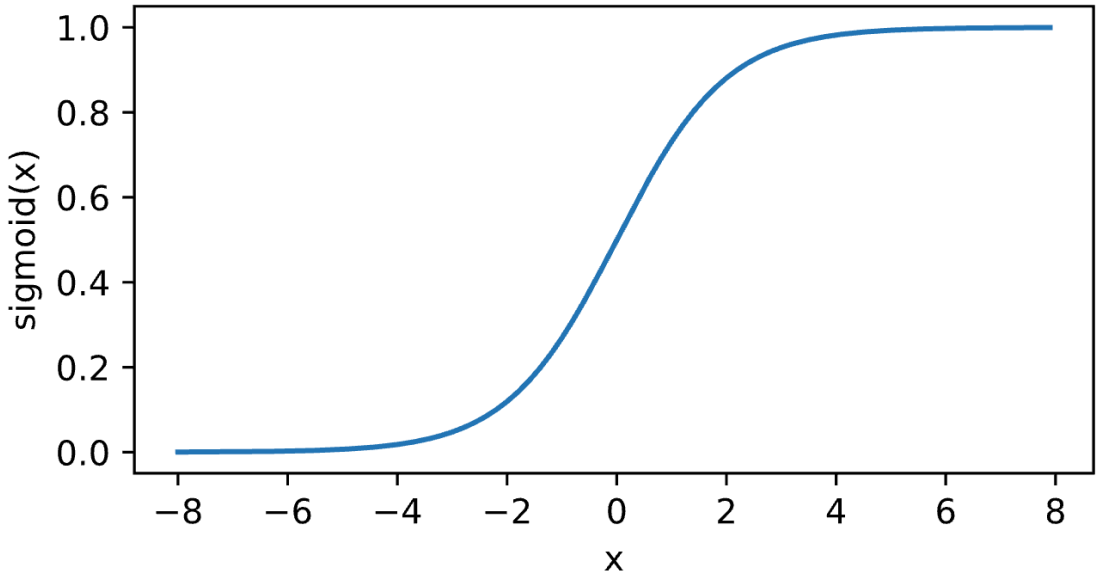


slides adapted from courses.d2l.ai/berkeley-stat-157

Sigmoid Activation

Map input into (0, 1), a soft version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

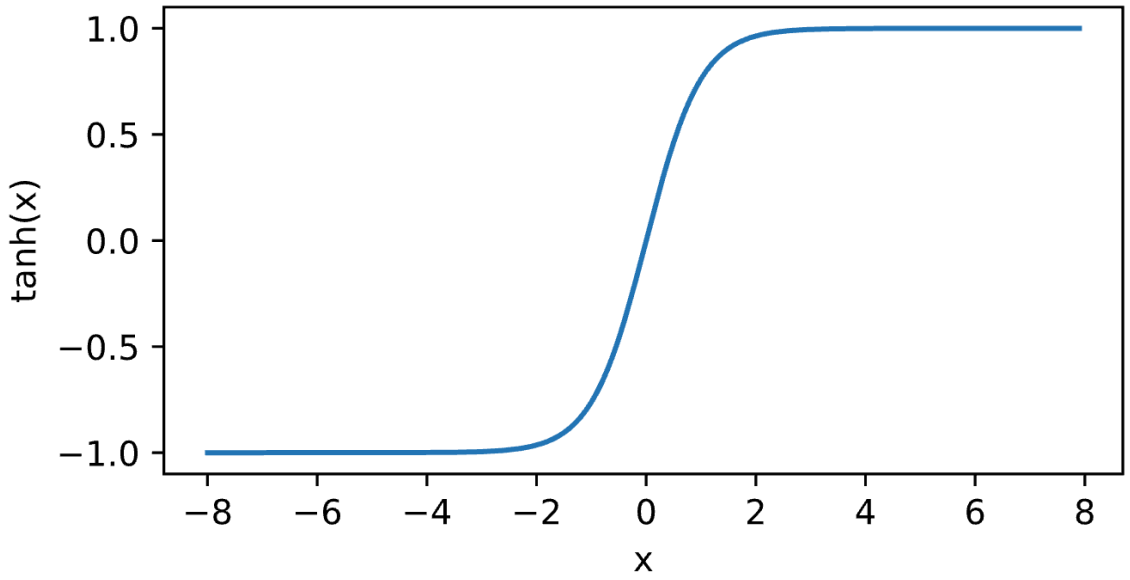


slides adapted from co

Tanh Activation

Map inputs into (-1, 1)

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

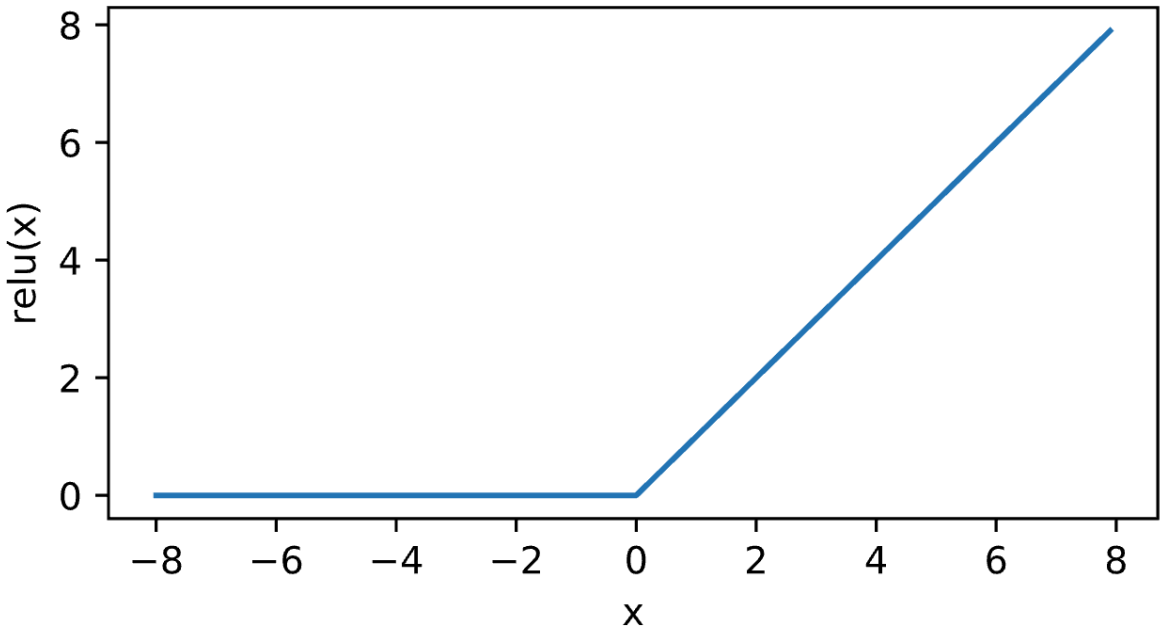


slides adapted from cc

ReLU Activation

ReLU: rectified linear unit

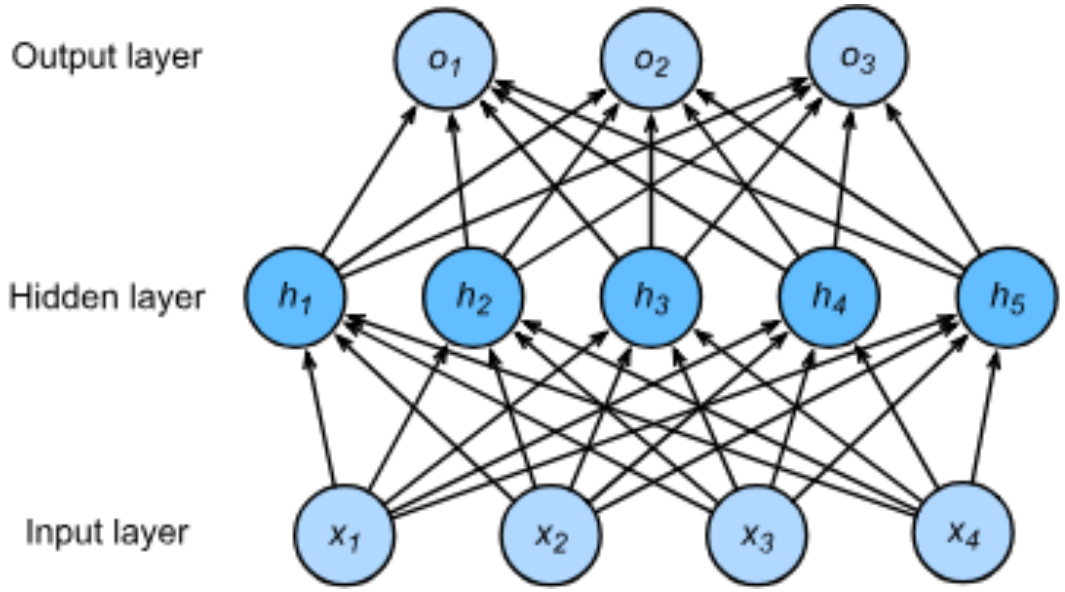
$$\text{ReLU}(x) = \max(x, 0)$$



slides adapted from co

Multiclass Classification

$$y_1, y_2, \ldots, y_k = \text{softmax}(o_1, o_2, \ldots, o_k)$$



slides adapted from courses.d2l.ai/berkeley-stat-157

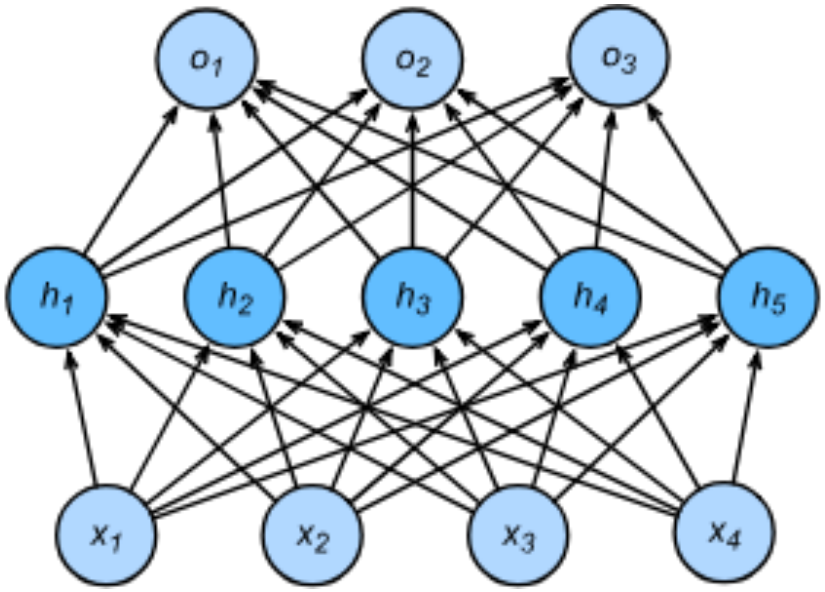
Multiclass Classification

- Input $\mathbf{x} \in \mathbb{R}^n$
- Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ and $\mathbf{b}_1 \in \mathbb{R}^m$
- Output $\mathbf{W}_2 \in \mathbb{R}^{m \times d}$ and $\mathbf{b}_2 \in \mathbb{R}^d$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(o)$$



slides adapted from courses.d2l.ai/berkeley-stat-157

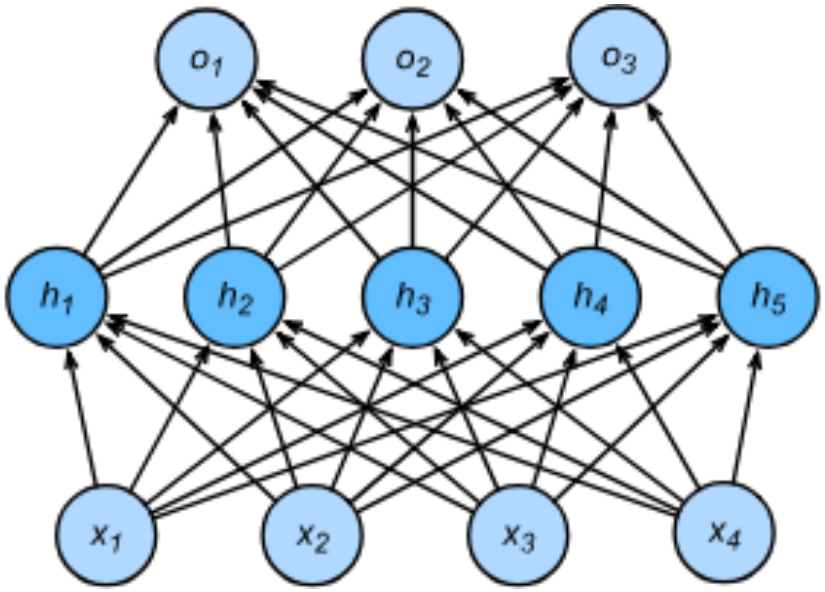
Multiclass Classification

- Input $\mathbf{x} \in \mathbb{R}^n$
- Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ and $\mathbf{b}_1 \in \mathbb{R}^m$
- Output $\mathbf{W}_2 \in \mathbb{R}^{m \times d}$ and $\mathbf{b}_2 \in \mathbb{R}^d$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(o)$$

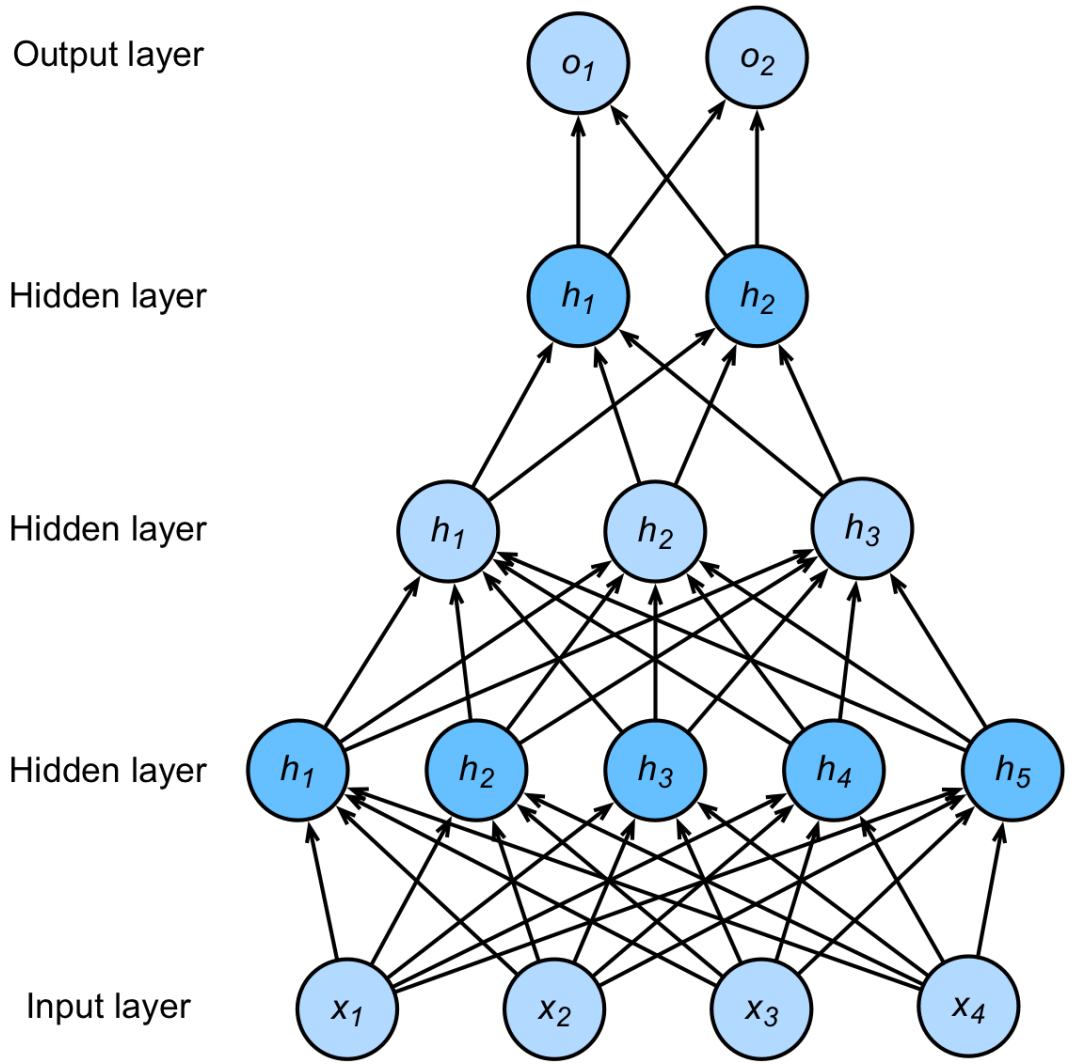


slides adapted from courses.d2l.ai/berkeley-stat-157

Multiple Hidden Layers

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{o} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$

Hyper-parameters
- # of hidden layers
- Hidden size for each layer



slides adapted from courses.d2l.ai/berkeley-stat-157

Summary

- Perceptron
  - Simple updates
  - Limited function complexity
- Multilayer Perceptron
  - Multiple layers add more complexity
  - Nonlinearity is needed
  - Simple composition (but architecture search needed)

slides adapted from courses.d2l.ai/berkeley-stat-157

Introduction to (Deep) Learning

Model Selection, Weight Decay, Dropout

slides adapted from courses.d2l.ai/berkeley-stat-157

Model Evaluation

adapted from courses.d2l.ai/berkeley-stat-157

Training Error and Generalization Error

- Training error: model error on the training data
- Generalization error: model error on new data
- Example: practice a future exam with past exams
  - Doing well on past exams (training error) doesn't guarantee a good score on the future exam (generalization error)
  - Student A gets 0 error on past exams by rote learning
  - Student B understands the reasons for given answers

adapted from courses.d2l.ai/berkeley-stat-157

Validation Dataset and Test Dataset

- Validation dataset: a dataset used to evaluate the model
  - E.g. Take out 50% of the training data
  - Should not be mixed with the training data (#1 mistake)
- Test dataset: a dataset can be used once, e.g.
  - A future exam
  - The house sale price I bid
  - Dataset used in private leaderboard in Kaggle

adapted from courses.d2l.ai/berkeley-stat-157

K-fold Cross Validation

- Useful when not sufficient data
- Algorithm:
  - Partition the training data into *K* parts
  - For *i = 1, …, K*
    - Use the *i*-th part as the validation set, the rest for training
  - Report the averaged the *K* validation errors
- Popular choices: *K = 5 or 10*

adapted from courses.d2l.ai/berkeley-stat-157

Underfitting Overfitting

Image credit: hackernoon.com

adapted from courses.d2l.ai/berkeley-stat-157

Underfitting and Overfitting

		Data complexity	
		Simple	Complex
Model capacity	Low	Normal	Underfitting
	High	Overfitting	Normal

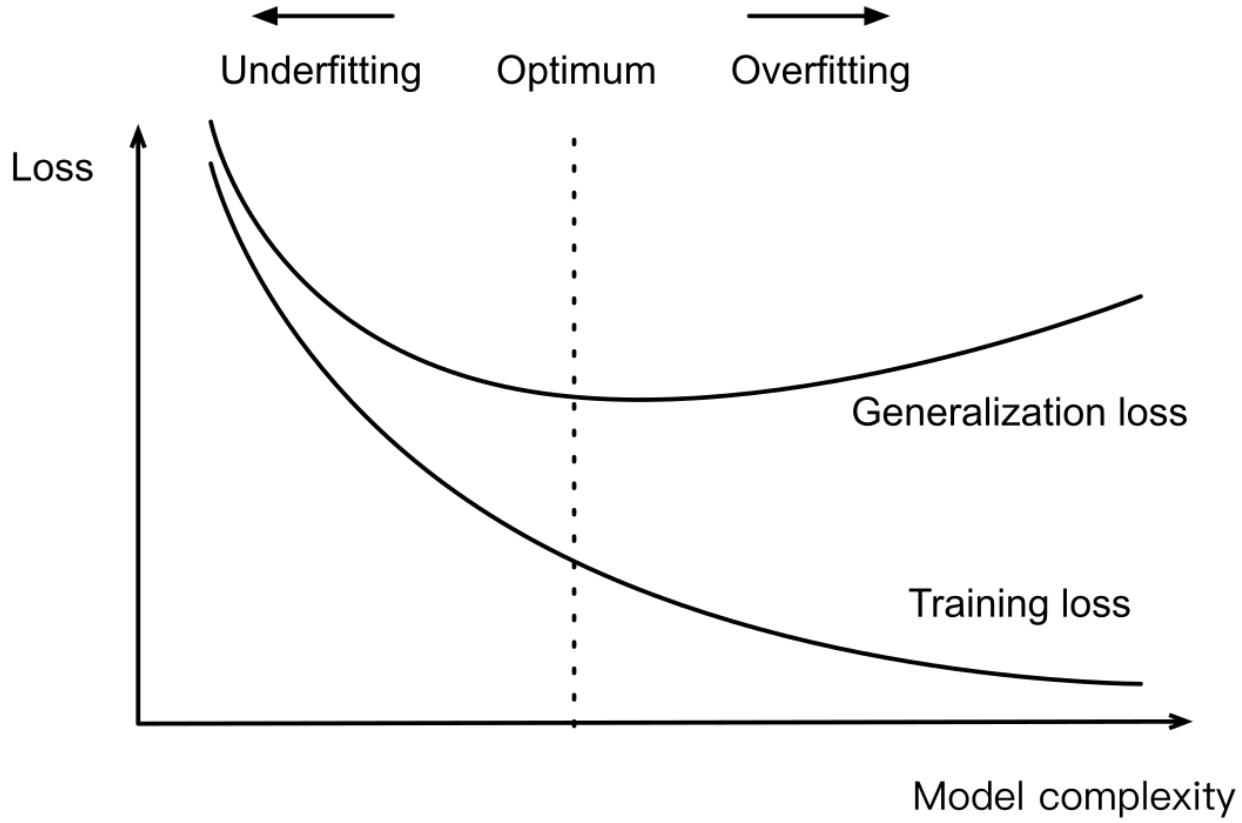
adapted from courses.d2l.ai/berkeley-stat-157

Model Capacity

- The ability to fit variety of functions
- Low capacity models struggles to fit training set
  - Underfitting
- High capacity models can memorize the training set
  - Overfitting

adapted from courses.d2l.ai/berkeley-stat-157

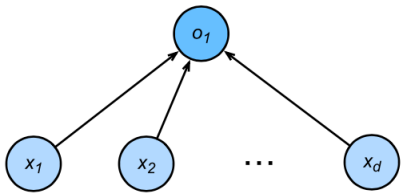
Influence of Model Complexity

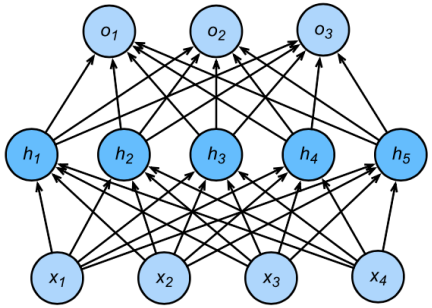


adapted from courses.d2l.ai/berkeley-stat-157

Estimate Model Capacity

- It's hard to compare complexity between different algorithms
  - e.g. tree vs neural network
- Given an algorithm family, two main factors matter:
  - The number of parameters
  - The values taken by each parameter





adapted from courses.d2l.ai/berkeley-stat-157

VC Dimension

- A center topic in Statistic Learning Theory
- For a classification model, it's the size of the largest dataset, no matter how we assign labels, there exist a model instance to classify them perfectly

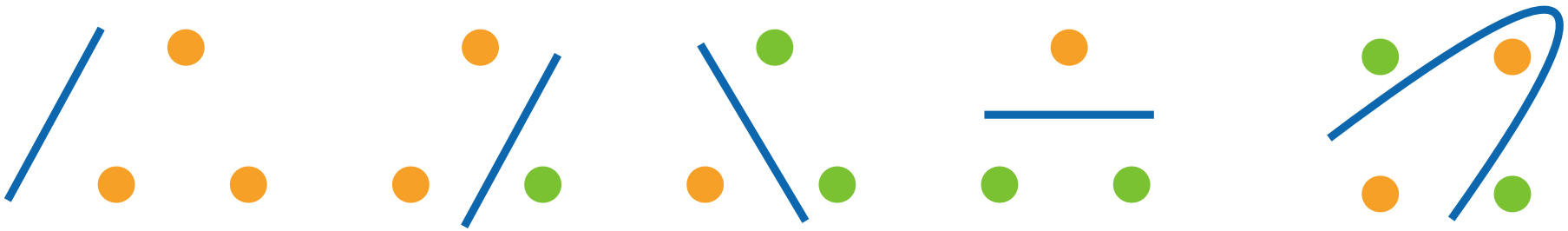
Vladimir **V**apnik

Alexey **C**hervonenkis

adapted from courses.d2l.ai/berkeley-stat-157

VC-Dimension for Linear Classifier

- 2-D perceptron: VCdim = 3
  - Can classify any 3 points, but not 4 points (xor)



- Perceptron with *N* parameters: VCdim = N
- Some Multilayer Perceptrons: VCdim = $O(N \log_2(N))$

adapted from courses.d2l.ai/berkeley-stat-157

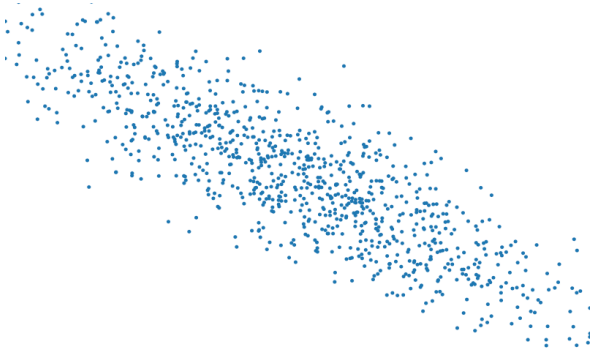
Usefulness of VC-Dimension

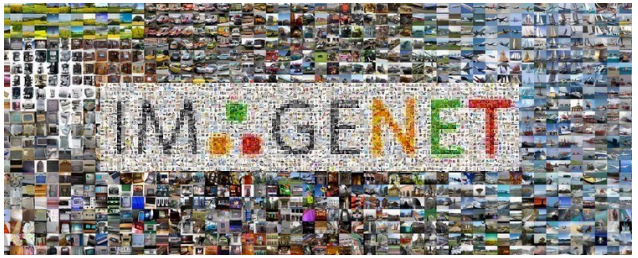
- Provides theory insights why a model works
  - Bound the gap between training error and generalization error
- Rarely used in practice with deep learning
  - The bounds are too loose
  - Difficulty to compute VC-dimension for deep neural networks
- Same for other statistic learning theory tools

adapted from courses.d2l.ai/berkeley-stat-157

Data Complexity

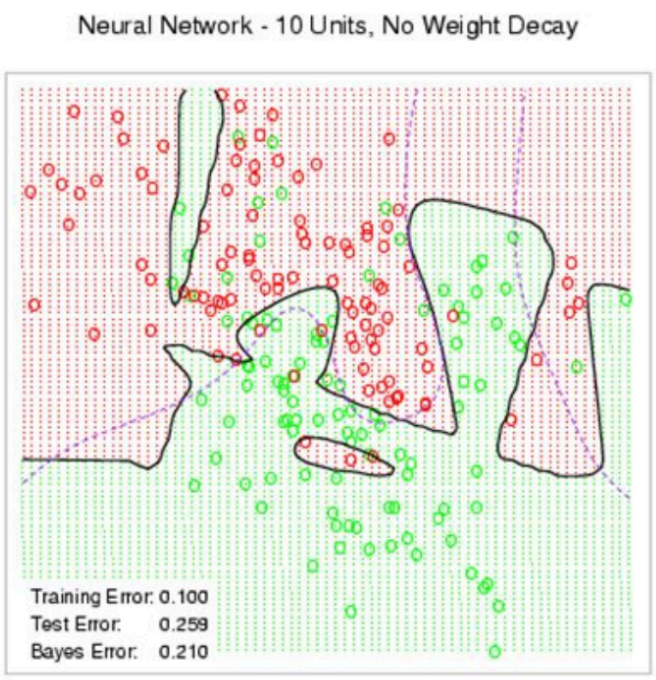
- Multiple factors matters
  - # of examples
  - # of elements in each example
  - time/space structure
  - diversity

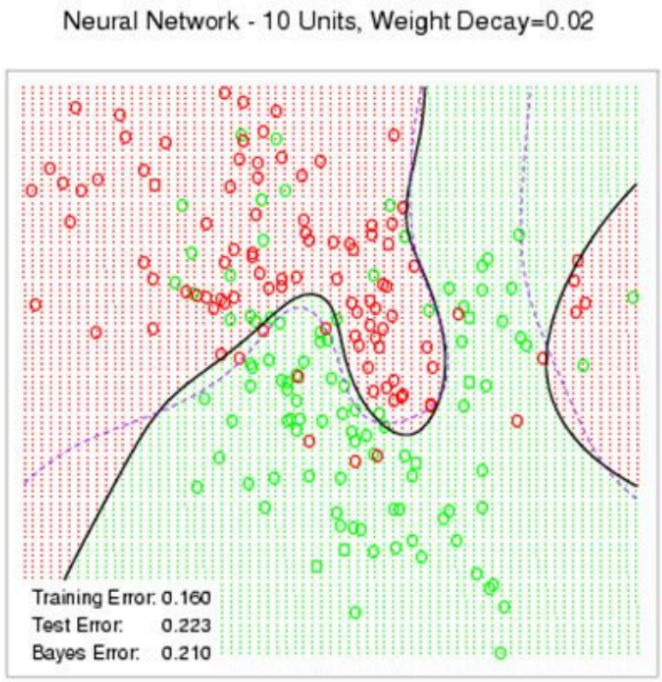




adapted from courses.d2l.ai/berkeley-stat-157

Weight Decay





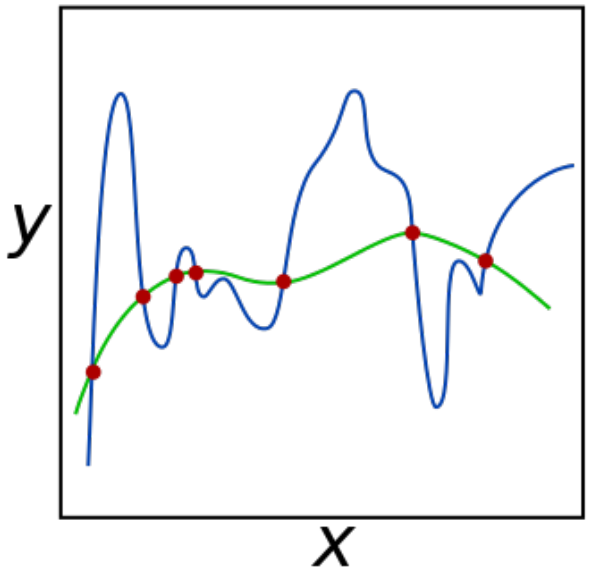
adapted from courses.d2l.ai/berkeley-stat-157

Squared Norm Regularization as Hard Constraint

- Reduce model complexity by limiting value range

$$\min \ \ell(\mathbf{w}, b) \quad \text{subject to} \ \ \|\mathbf{w}\|^2 \leq \theta$$

  - Often do not regularize bias b
    - Doing or not doing has little difference in practice
  - A small θ means more regularization



adapted from courses.d2l.ai/berkeley-stat-157

Squared Norm Regularization as Soft Constraint

- For each θ, we can find λ to rewrite the hard constraint version as

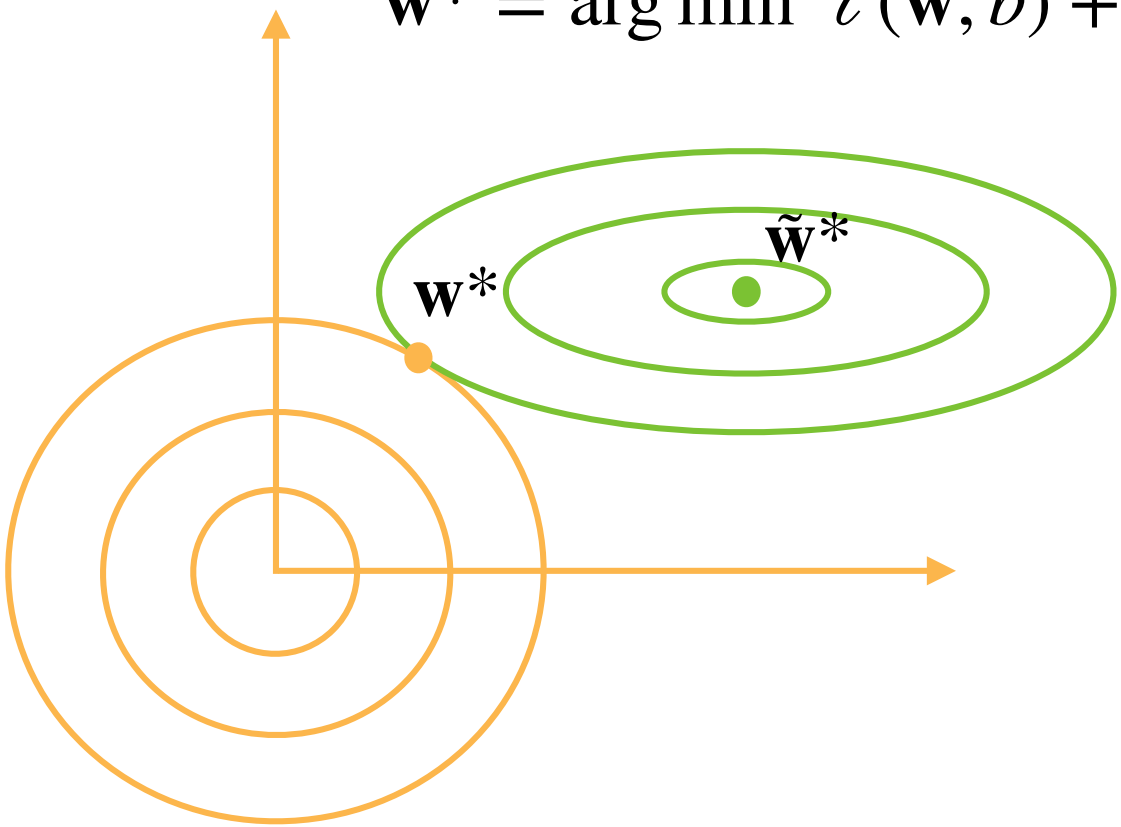
$$\min \ \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

  - Can prove by Lagrangian multiplier method
  - Hyper-parameter λ controls regularization importance
  - $\lambda = 0$: no effect
  - $\lambda \to \infty, \mathbf{w}^* \to \mathbf{0}$

adapted from courses.d2l.ai/berkeley-stat-157

Illustrate the Effect on Optimal Solutions

$$\mathbf{w}^* = \arg\min \ \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



$$\tilde{\mathbf{w}}^* = \arg\min \ \ell(\mathbf{w}, b)$$

adapted from courses.d2l.ai/berkeley-stat-157

Update Rule

- Compute the gradient

$$\frac{\partial}{\partial \mathbf{w}} \left(\ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right) = \frac{\partial \ell(\mathbf{w}, b)}{\partial \mathbf{w}} + \lambda \mathbf{w}$$

- Update weight at time t

$$\mathbf{w}_{t+1} = (1 - \eta\lambda)\mathbf{w}_t - \eta \frac{\partial \ell(\mathbf{w}_t, b_t)}{\partial \mathbf{w}_t}$$

  - Often $\eta\lambda < 1$, so also called weight decay in deep learning

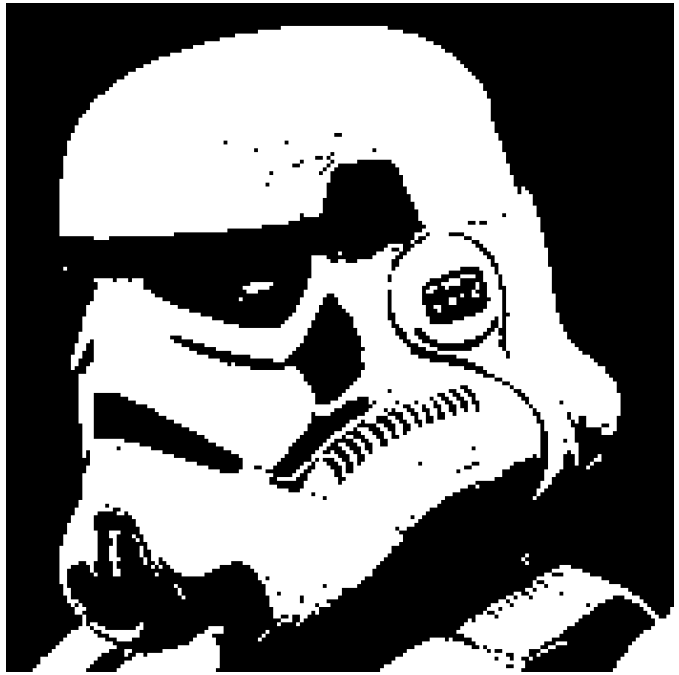
adapted from courses.d2l.ai/berkeley-stat-157

Dropout

adapted from courses.d2l.ai/berkeley-stat-157

Motivation

- A good model should be robust under modest changes in the input
  - Training with input noise equals to Tikhonov Regularization
  - Dropout: inject noises into internal layers



adapted from courses.d2l.ai/berkeley-stat-157

Add Noise without Bias

- Add noise into ***x*** to get ***x'***, we hope

$$\mathbf{E}[\mathbf{x}'] = \mathbf{x}$$

- Dropout perturbs each element by

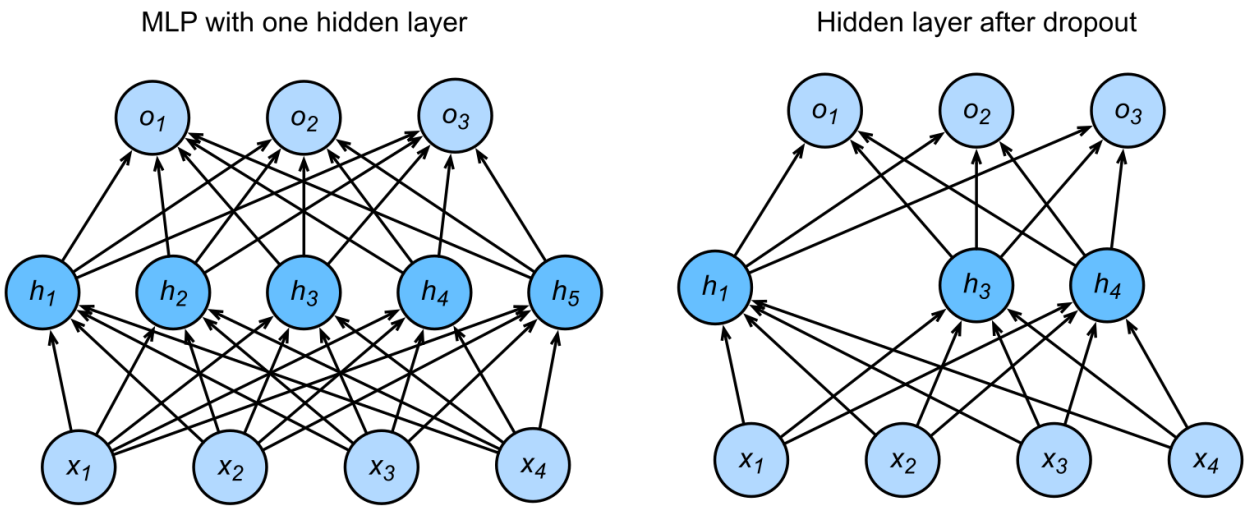
$$x_i' = \begin{cases} 0 & \text{with probablity } p \\ \frac{x_i}{1-p} & \text{otherise} \end{cases}$$

adapted from courses.d2l.ai/berkeley-stat-157

Apply Dropout

- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$
$$\mathbf{o} = \mathbf{W}_2\mathbf{h}' + \mathbf{b}_2$$
$$\mathbf{y} = \text{softmax}(o)$$



adapted from courses.d2l.ai/berkeley-stat-157

Introduction to (Deep) Learning

Convolutional and Pooling Layers

adapted from courses.d2l.ai/berkeley-stat-157


Convolutions

Classifying Dogs and Cats in Images

- Use a good camera
- RGB image has 36M elements
- The model size of a single hidden layer MLP with a 100 hidden size is 3.6 Billion parameters
- Exceeds the population of dogs and cats on earth (900M dogs + 600M cats)

Dual

12MP

wide-angle and telephoto cameras

adapted from courses.d2l.ai/berkeley-stat-157

Flashback - Network with one hidden layer

Output layer

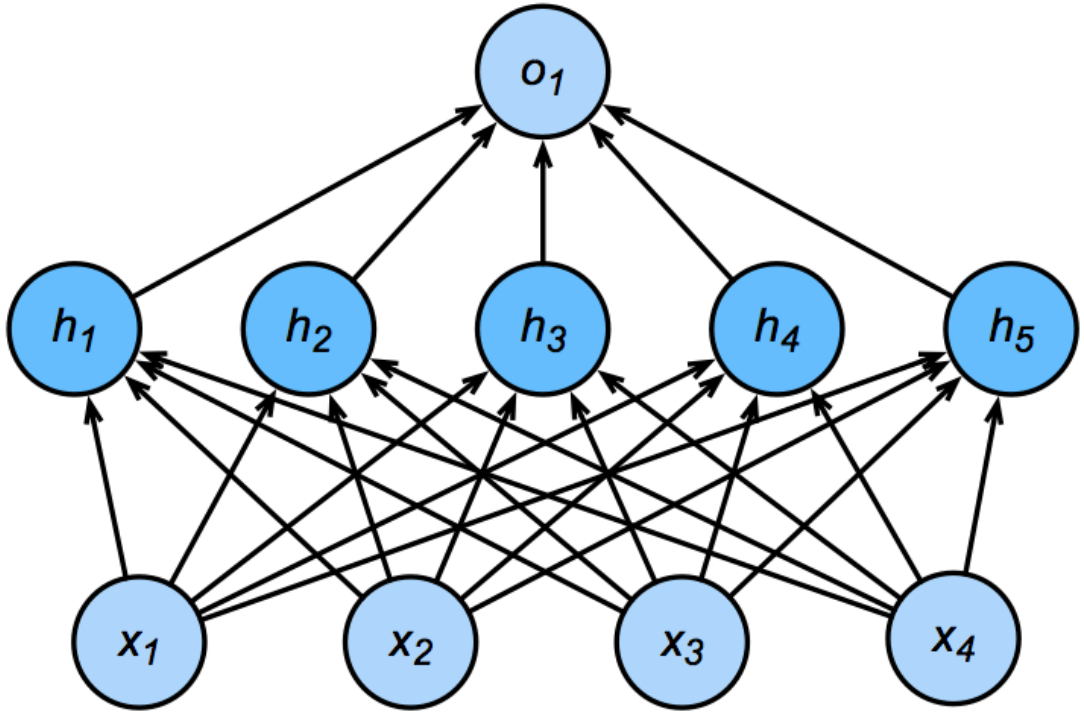
100 neurons

Hidden layer

3.6B parameters = 14GB

36M features

Input layer



$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

adapted from courses.d2l.ai/berkeley-stat-157

Where is Waldo?

adapted from courses.

Two Principles

- Translation Invariance
- Locality

adapted from courses.d2l.ai/berkeley-

Rethinking Dense Layers

- Reshape inputs and output into matrix (width, height)
- Reshape weights into 4-D tensors (h,w) to (h',w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

V is re-indexes W such as that

$$v_{i,j,a,b} = w_{i,j,i+a,j+b}$$

adapted from courses.d2l.ai/berkeley-stat-157

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

- A shift in *x* also leads to a shift in *h*
- *v* should not depend on *(i,j)*. Fix via $v_{i,j,a,b} = v_{a,b}$

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

That's a ~~2-D convolution~~
cross-correlation

adapted from courses.d2l.ai/berkeley-stat-157

Idea #2 - Locality

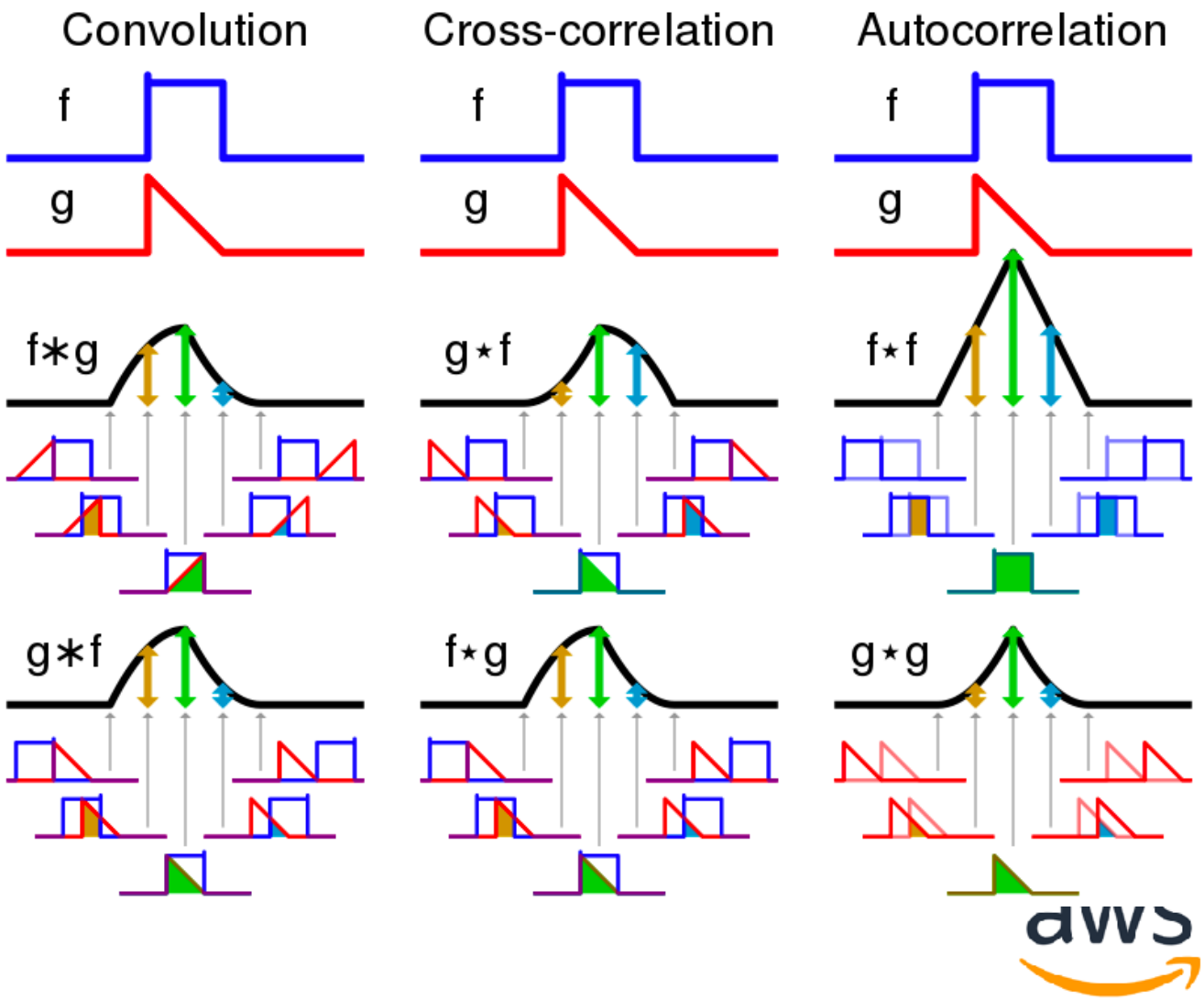
$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

- We shouldn't look very far from x(i,j) in order to assess what's going on at h(i,j)
- Outside range $|a|, |b| > \Delta$ parameters vanish $v_{a,b} = 0$

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$

adapted from courses.d2l.ai/berkeley-stat-157

Convolution



2-D Cross Correlation

Input

0	1	2
3	4	5
6	7	8

*

Kernel

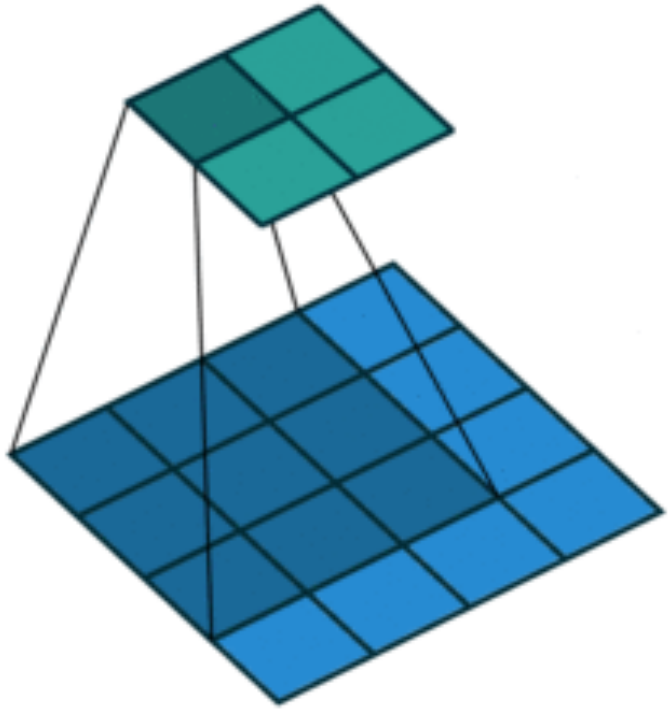
0	1
2	3

=

Output

19	25
37	43

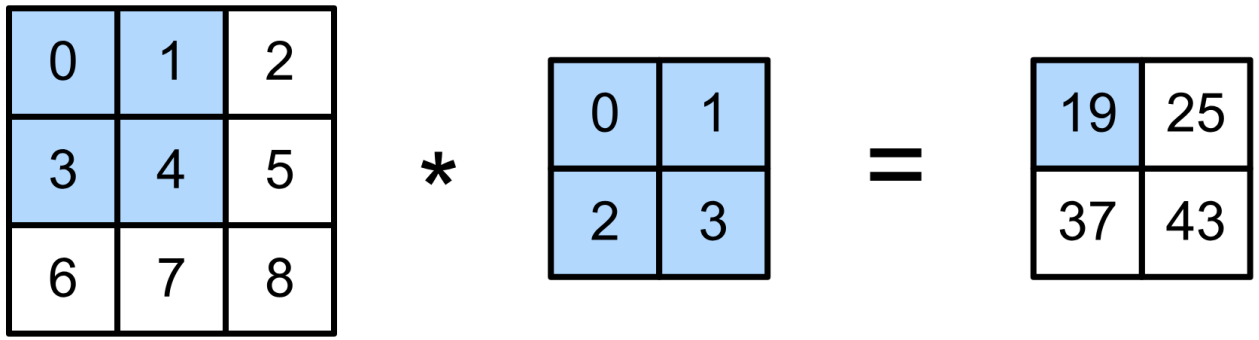
$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$



(vdumoulin@ Github)

adapted from courses.d2l.ai/berkeley-stat-157

2-D Convolution Layer



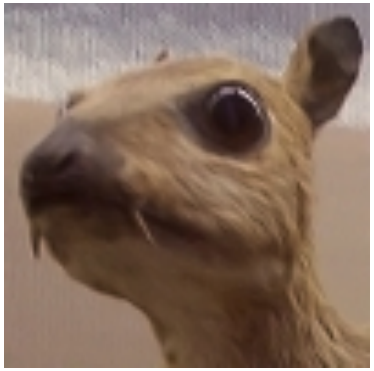
- $\mathbf{X} : n_h \times n_w$ input matrix
- $\mathbf{W} : k_h \times k_w$ kernel matrix
- b: scalar bias
- $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- **W** and *b* are learnable parameters

adapted from courses.d2l.ai/berkeley-stat-157

Examples

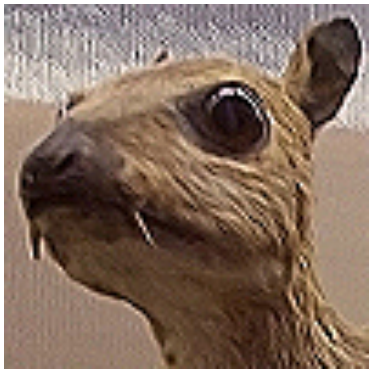


(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Sharpen

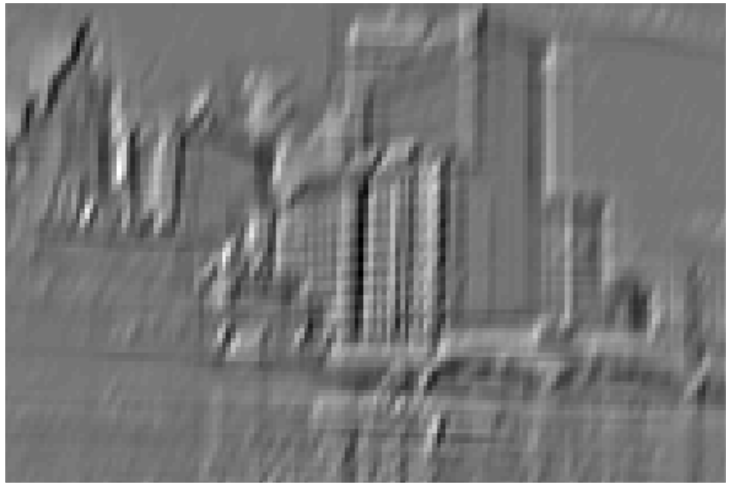
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

adapted from courses.d2l.ai/berkeley-stat-157

Examples

(Rob Fergus)



adapted from courses.d2l.ai/berkeley-stat-157

Cross Correlation vs Convolution

- 2-D Cross Correlation

$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{a,b} x_{i+a,j+b}$$

- 2-D Convolution

$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{-a,-b} x_{i+a,j+b}$$

- No difference in practice during to symmetry

adapted from courses.d2l.ai/berkeley-stat-157

1-D and 3-D Cross Correlations

- 1-D

$$y_i = \sum_{a=1}^{h} w_a x_{i+a}$$

  - Text
  - Voice
  - Time series

- 3-D

$$y_{i,j,k} = \sum_{a=1}^{h} \sum_{b=1}^{w} \sum_{c=1}^{d} w_{a,b,c} x_{i+a,j+b,k+c}$$

  - Video
  - Medical images

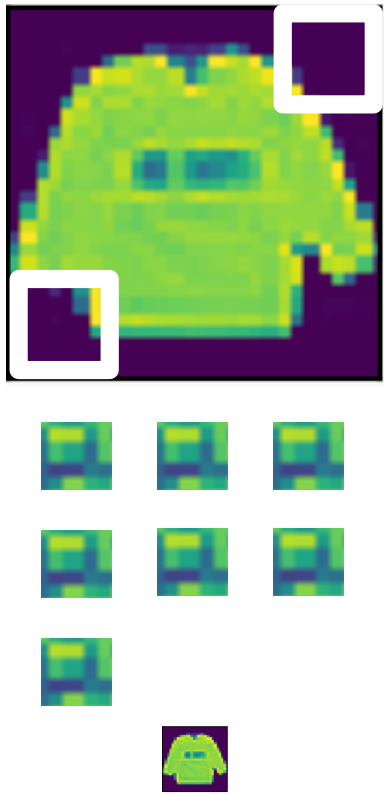
adapted from courses.d2l.ai/berkeley-stat-157


Padding and Stride

Padding

- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to

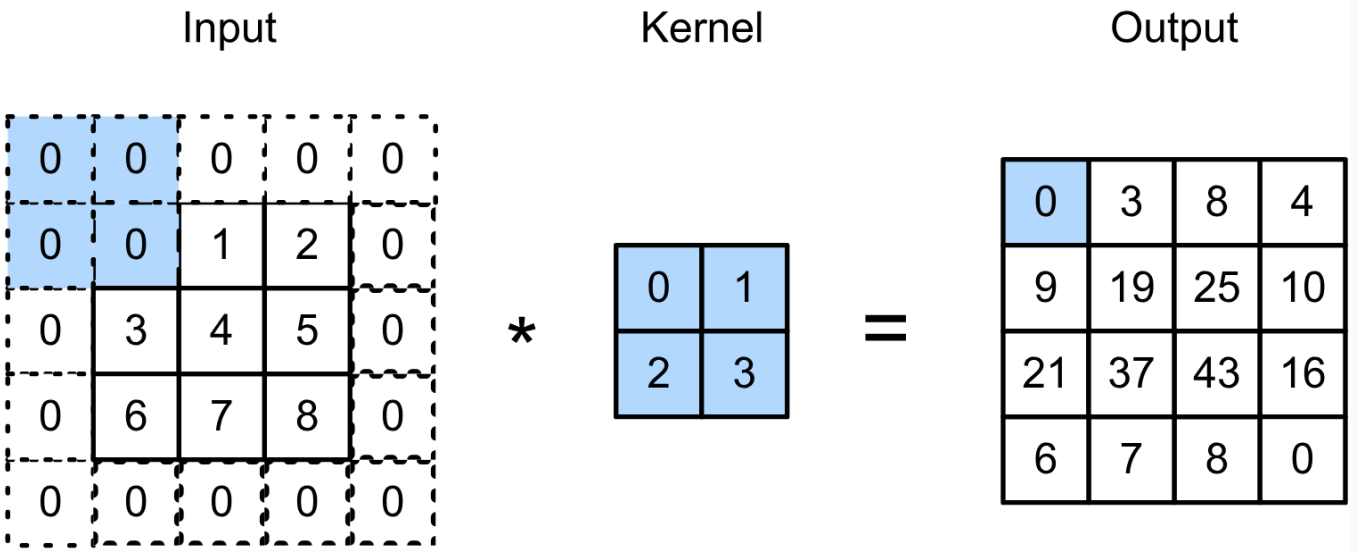
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

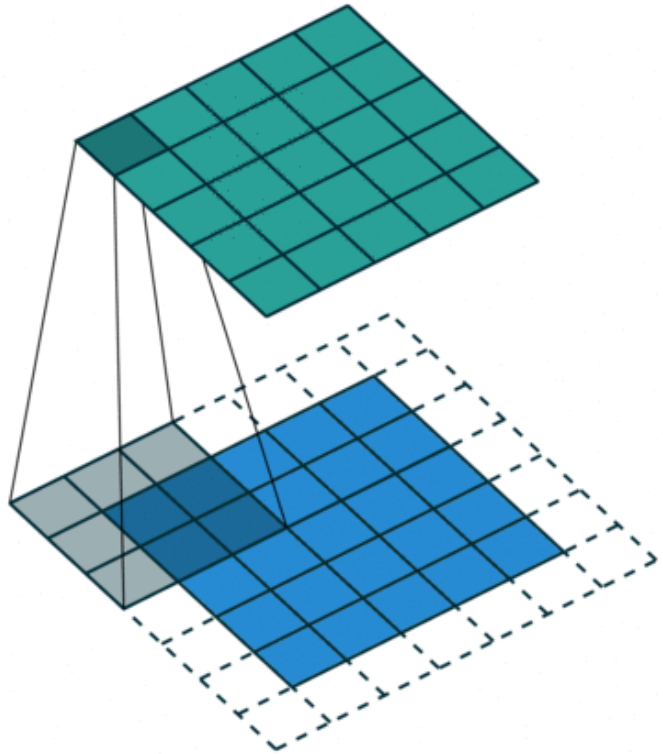


adapted from courses.d2l.ai/berkeley-stat-157

Padding

Padding adds rows/columns around input





$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

adapted from courses.d2l.ai/berkeley-stat-157

Padding

- Padding p_h rows and p_w columns, output shape will be

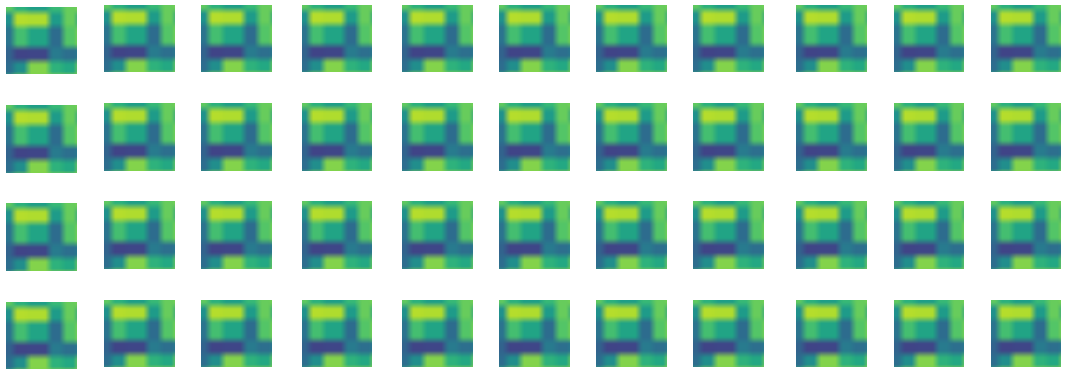
$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- A common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$
  - Odd k_h : pad $p_h/2$ on both sides
  - Even k_h : pad $\lceil p_h/2 \rceil$ on top, $\lfloor p_h/2 \rfloor$ on bottom

adapted from courses.d2l.ai/berkeley-stat-157

Stride

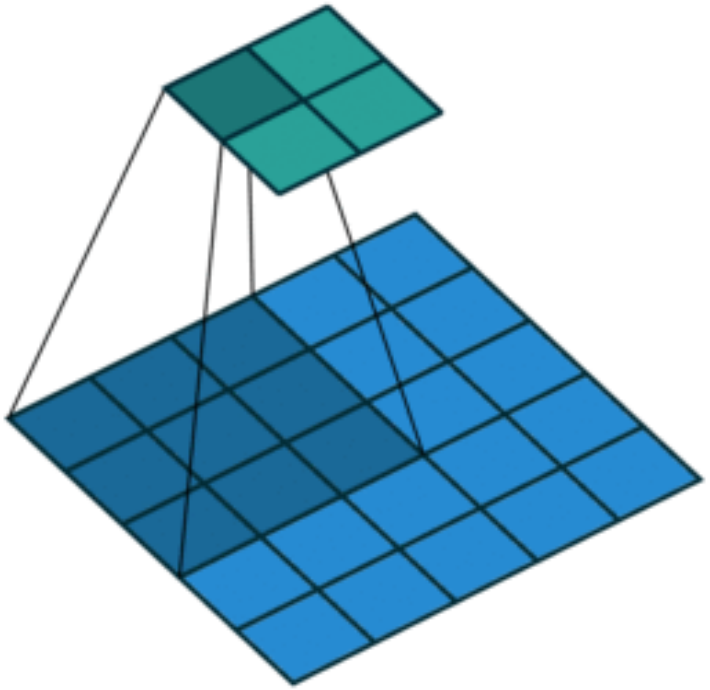
- Padding reduces shape linearly with #layers
  - Given a 224 x 224 input with a 5 x 5 kernel, needs 44 layers to reduce the shape to 4 x 4
  - Requires a large amount of computation



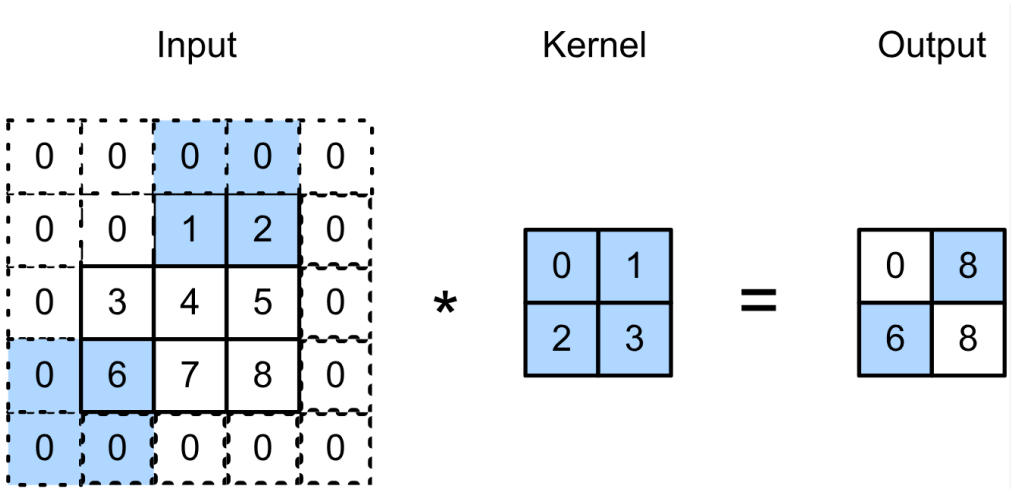
adapted from courses.d2l.ai/berkeley-stat-157

Stride

- Stride is the #rows/#columns per slide



Strides of 3 and 2 for height and width



$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

adapted from courses.d2l.ai/berkeley-stat-157

Stride

- Given stride s_h for the height and stride s_w for the width, the output shape is

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- With $p_h = k_h - 1$ and $p_w = k_w - 1$

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- If input height/width are divisible by strides

$$(n_h/s_h) \times (n_w/s_w)$$

adapted from courses.d2l.ai/berkeley-stat-157


Multiple Input and
Output Channels

Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

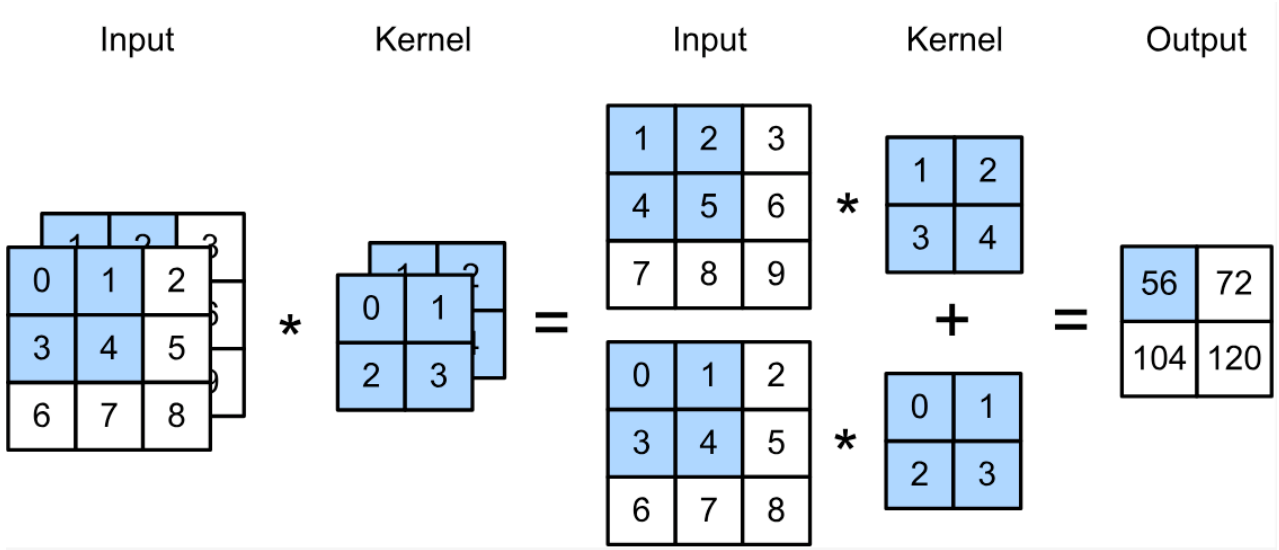
adapted from courses.d2l.ai/berkel

Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

- Have a kernel for each channel, and then sum results over channels



$$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4)$$
$$+(0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3)$$
$$= 56$$

adapted from courses.d2l.ai/berkeley-stat-157

Multiple Input Channels

- $\mathbf{X}$: $c_i \times n_h \times n_w$ input
- $\mathbf{W}$: $c_i \times k_h \times k_w$ kernel
- $\mathbf{Y}$: $m_h \times m_w$ output

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

adapted from courses.d2l.ai/berkeley-stat-157

Multiple Output Channels

- No matter how many inputs channels, so far we always get single output channel
- We can have multiple 3-D kernels, each one generates a output channel
- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$

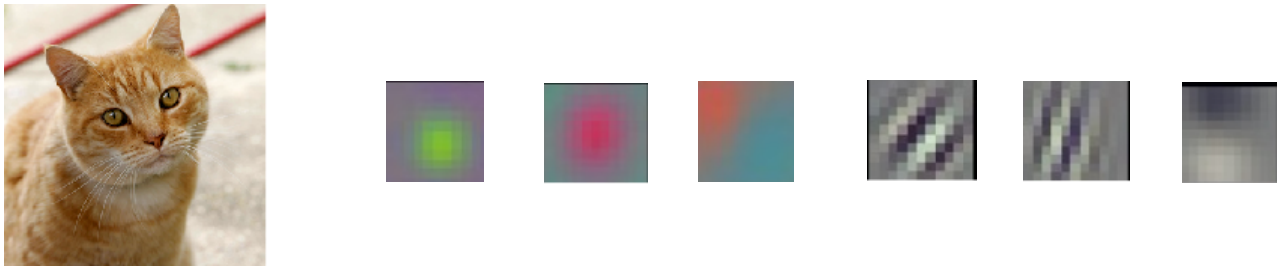
$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

for $i = 1, \ldots, c_o$

adapted from courses.d2l.ai/berkeley-stat-157

Multiple Input/Output Channels

- Each output channel may recognize a particular pattern

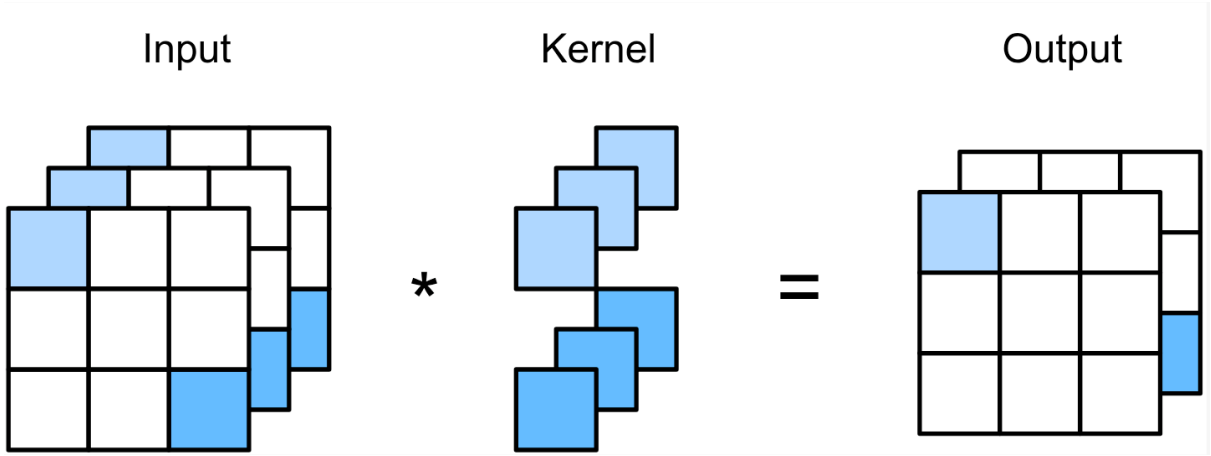


- Input channels kernels recognize and combines patterns in inputs

adapted from courses.d2l.ai/berkeley-stat-157

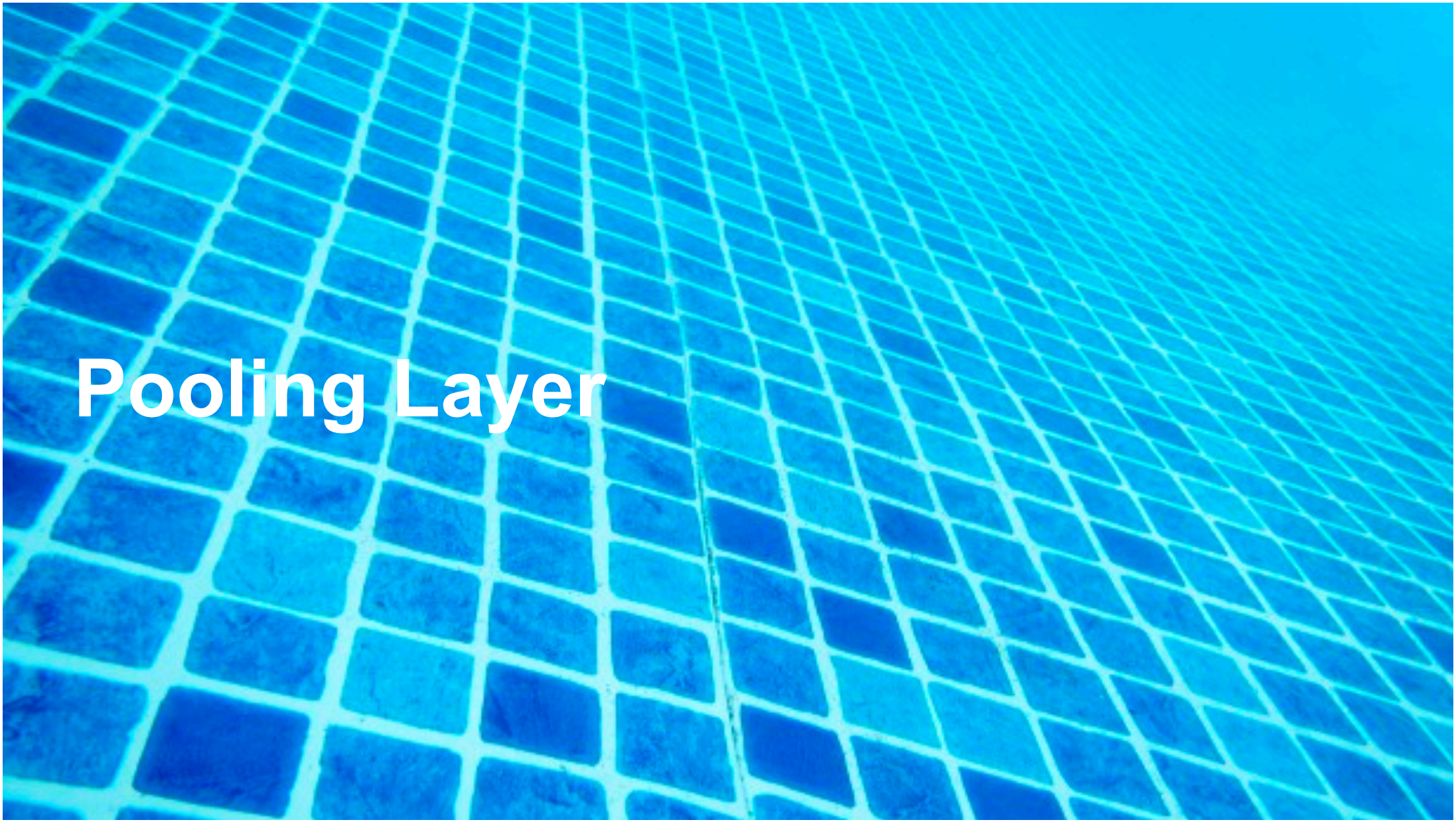
1 x 1 Convolutional Layer

$k_h = k_w = 1$ is a popular choice. It doesn't recognize spatial patterns, but fuse channels.



Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.

adapted from courses.d2l.ai/berkeley-stat-157


Pooling Layer

Pooling

- Convolution is sensitive to position
  - Detect vertical edges

X

```
[[1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
```

Y

```
[[ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
```

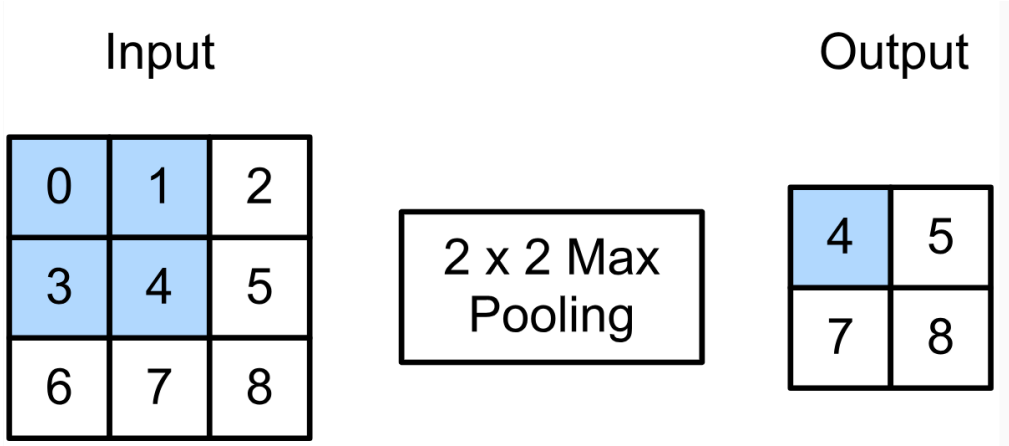
0 output with 1 pixel shift

- We need some degree of invariance to translation
  - Lighting, object positions, scales, appearance vary among images

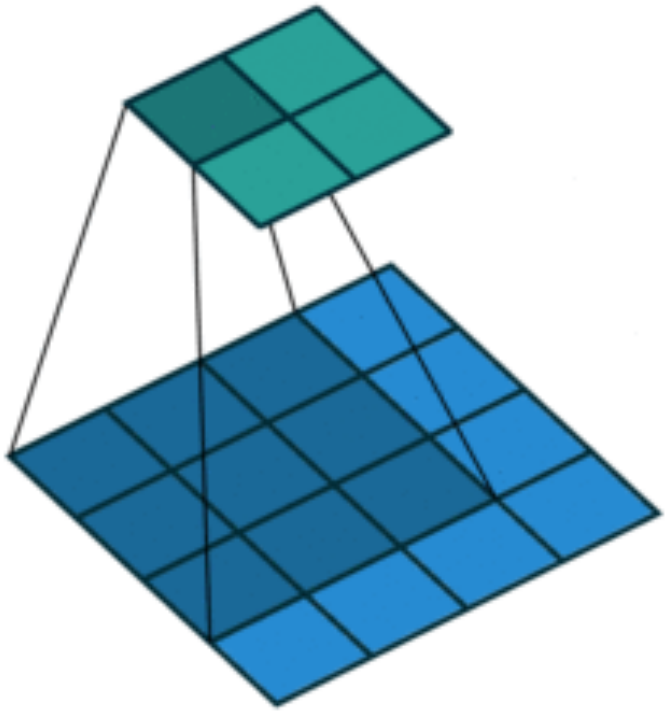
adapted from courses.d2l.ai/berkeley-stat-157

2-D Max Pooling

- Returns the maximal value in the sliding window



$$\max(0,1,3,4) = 4$$



adapted from courses.d2l.ai/berkeley-stat-157

2-D Max Pooling

- Returns the maximal value in the sliding window

Vertical edge detection

```
[[1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
```

Conv output

```
[[ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
```

2 x 2 max pooling

```
[[ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
```

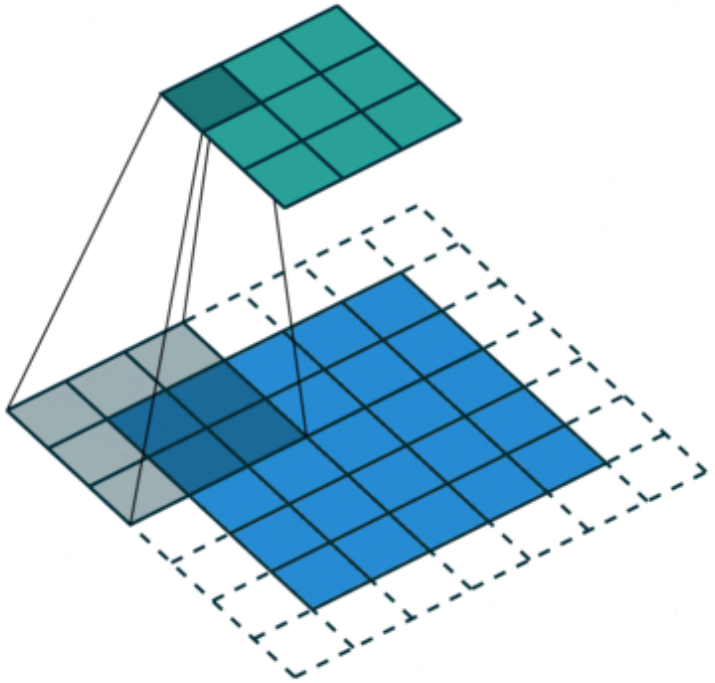
Tolerant to 1 pixel shift

adapted from courses.d2l.ai/berkeley-stat-157

Padding, Stride, and Multiple Channels

- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel

#output channels = #input channels

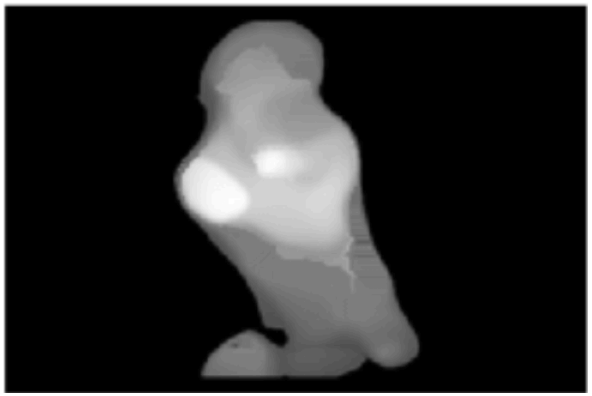


adapted from courses.d2l.ai/berkeley-stat-157

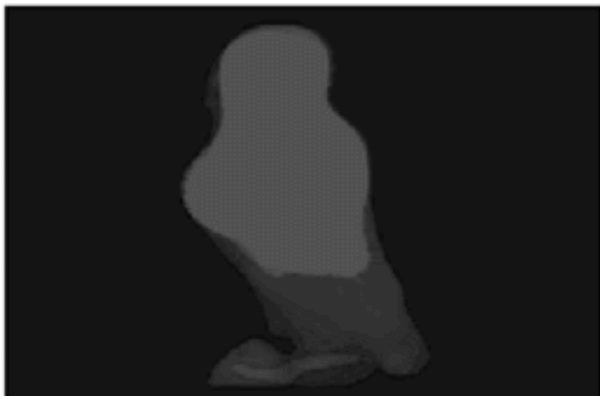
Average Pooling

- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window

Max pooling

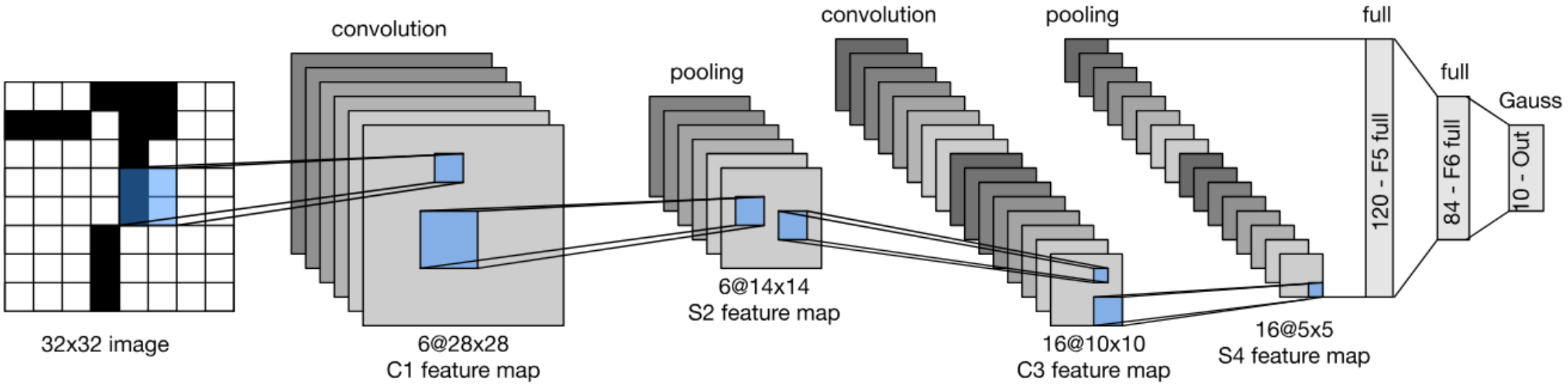


Average pooling



adapted from courses.d2l.ai/berkeley-stat-157

LeNet Architecture



gluon-cv.mxnet.io