CASUS
CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

Convolutional Neural Networks

Data Science Tutorial I
APS March Meeting 2024

Karan Shah

www.casus.science

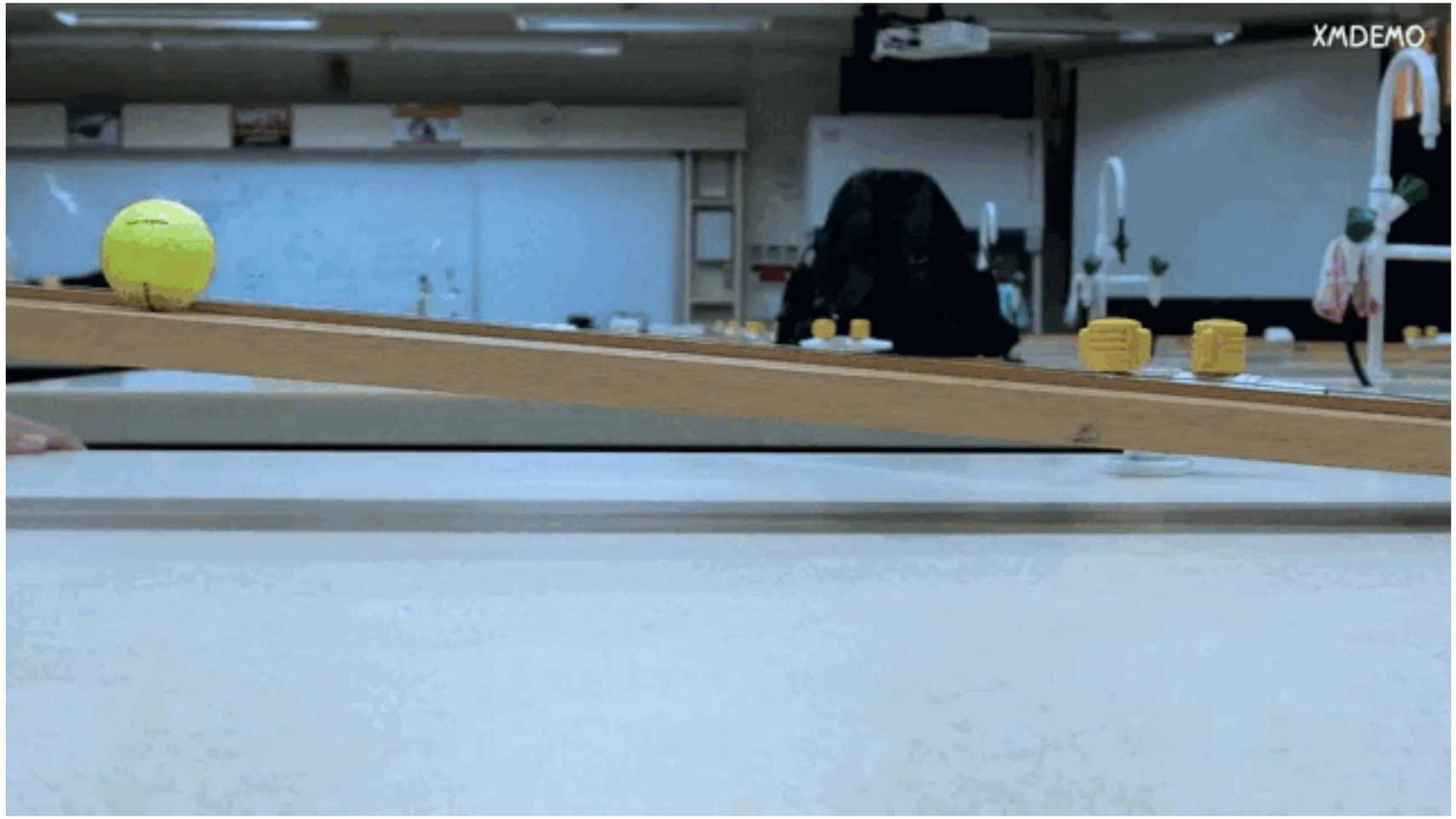
HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF
TECHNISCHE UNIVERSITÄT DRESDEN
CBG Max Planck Institute of Molecular Cell Biology and Genetics
HELMHOLTZ CENTRE FOR ENVIRONMENTAL RESEARCH – UFZ
Uniwersytet Wrocławski
SPONSORED BY THE
Federal Ministry of Education and Research
STAATSMINISTERIUM FÜR WISSENSCHAFT UND KUNST
Freistaat SACHSEN

Agenda

- Sample Problem: Object detection in a video
  - Brief Computer Vision Intro

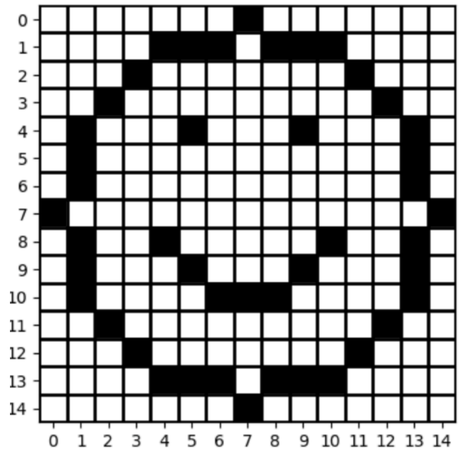
- Machine Learning
  - Neural Networks
    - Structure
    - Training
  - Convolutional Neural Networks
    - Convolutions
    - Structure
    - Training
    - Applications

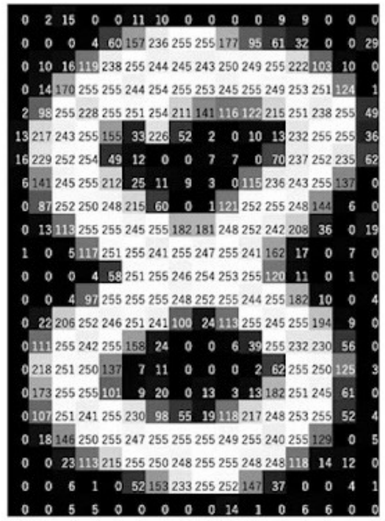
Problem: Detecting motion in undergrad lab experiments



Brief Computer Vision Introduction:
Images are matrices too

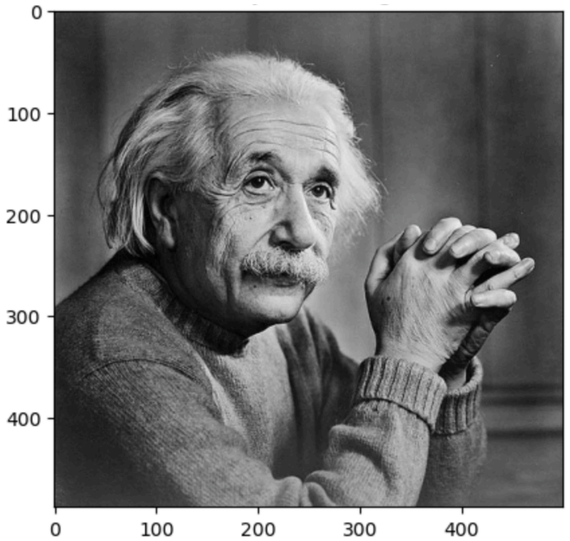
```
[[1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.],
[1., 1., 1., 1., 0., 0., 0., 1., 0., 0., 0., 1., 1., 1., 1.],
[1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1.],
[1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1.],
[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1.],
[1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.],
[1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.],
[0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.],
[1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1.],
[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1.],
[1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 1., 1., 0., 1.],
[1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1.],
[1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1.],
[1., 1., 1., 1., 0., 0., 0., 1., 0., 0., 0., 1., 1., 1., 1.],
[1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.]]
```



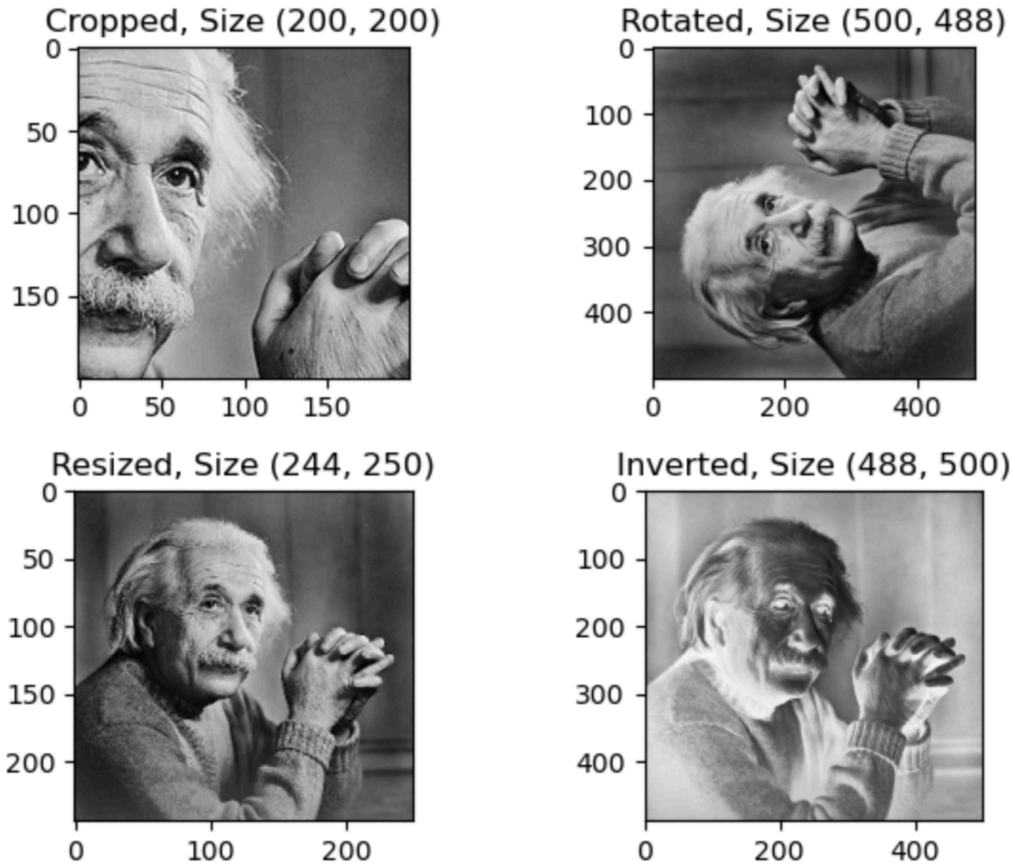


0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

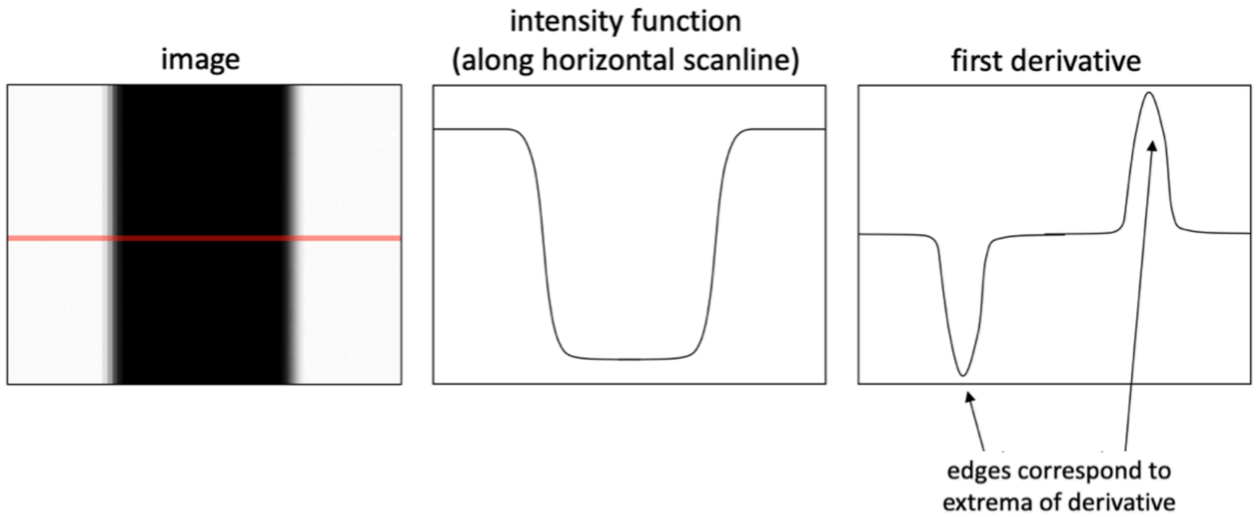
Brief Computer Vision Introduction:
Basic manipulations



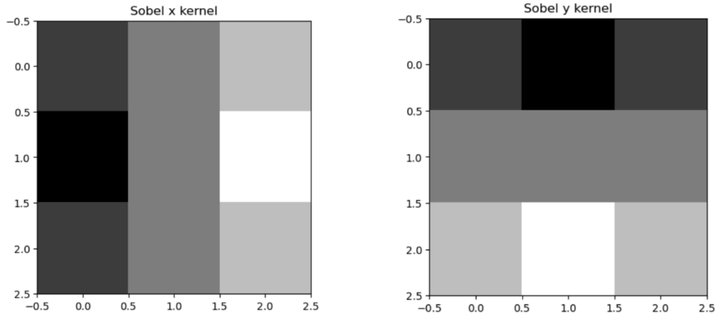
Original, Size (500, 500)

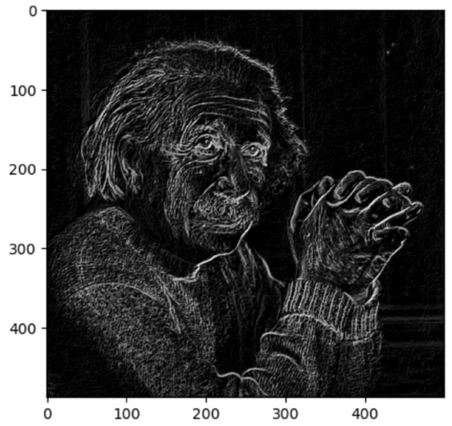


Brief Computer Vision Introduction:
Example of convolution

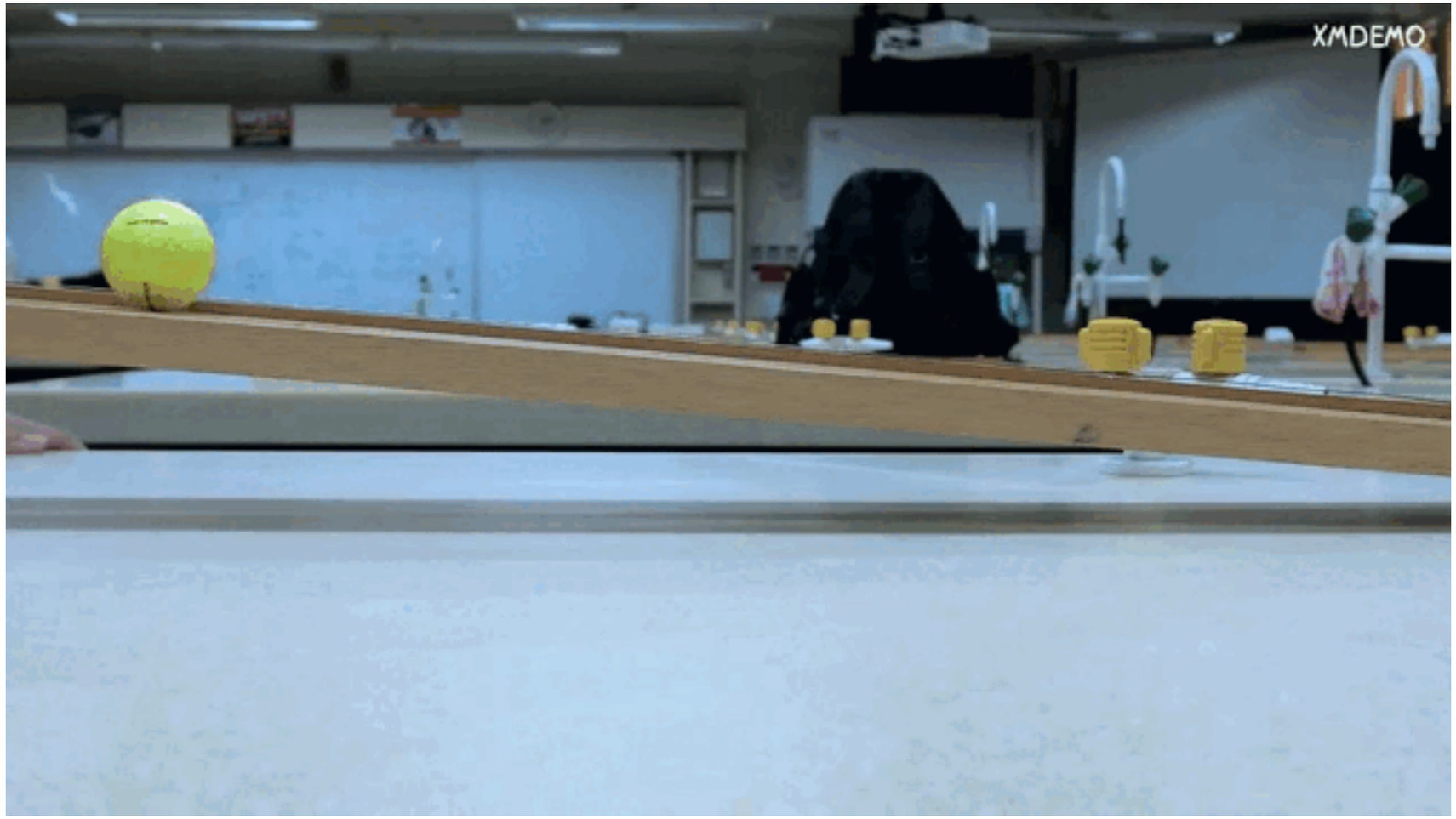


$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$





Solution: Use a series of matrix operations



Solution: Use a series of matrix operations

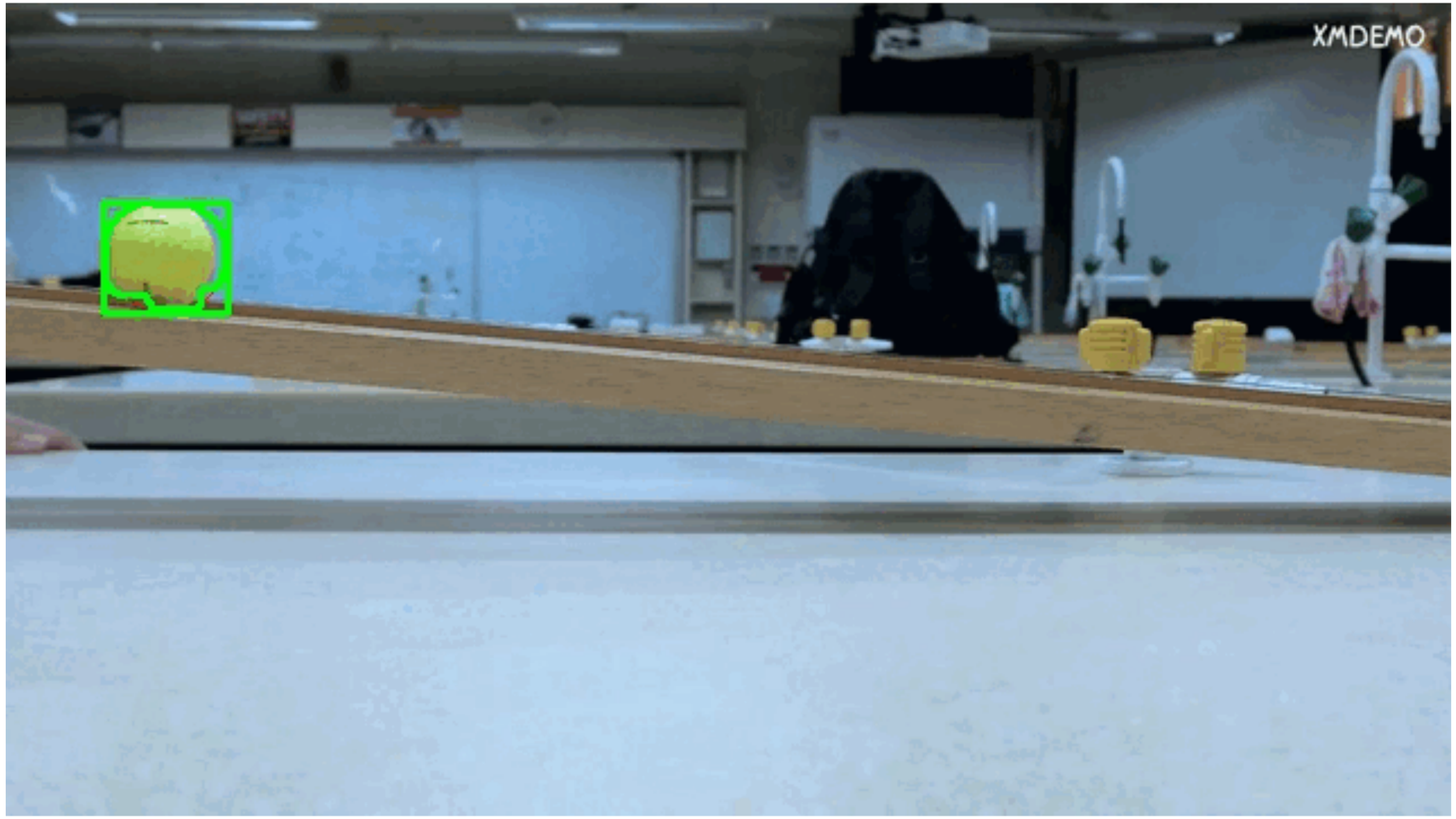
Solution: Use a series of matrix operations

Solution: Use a series of matrix operations

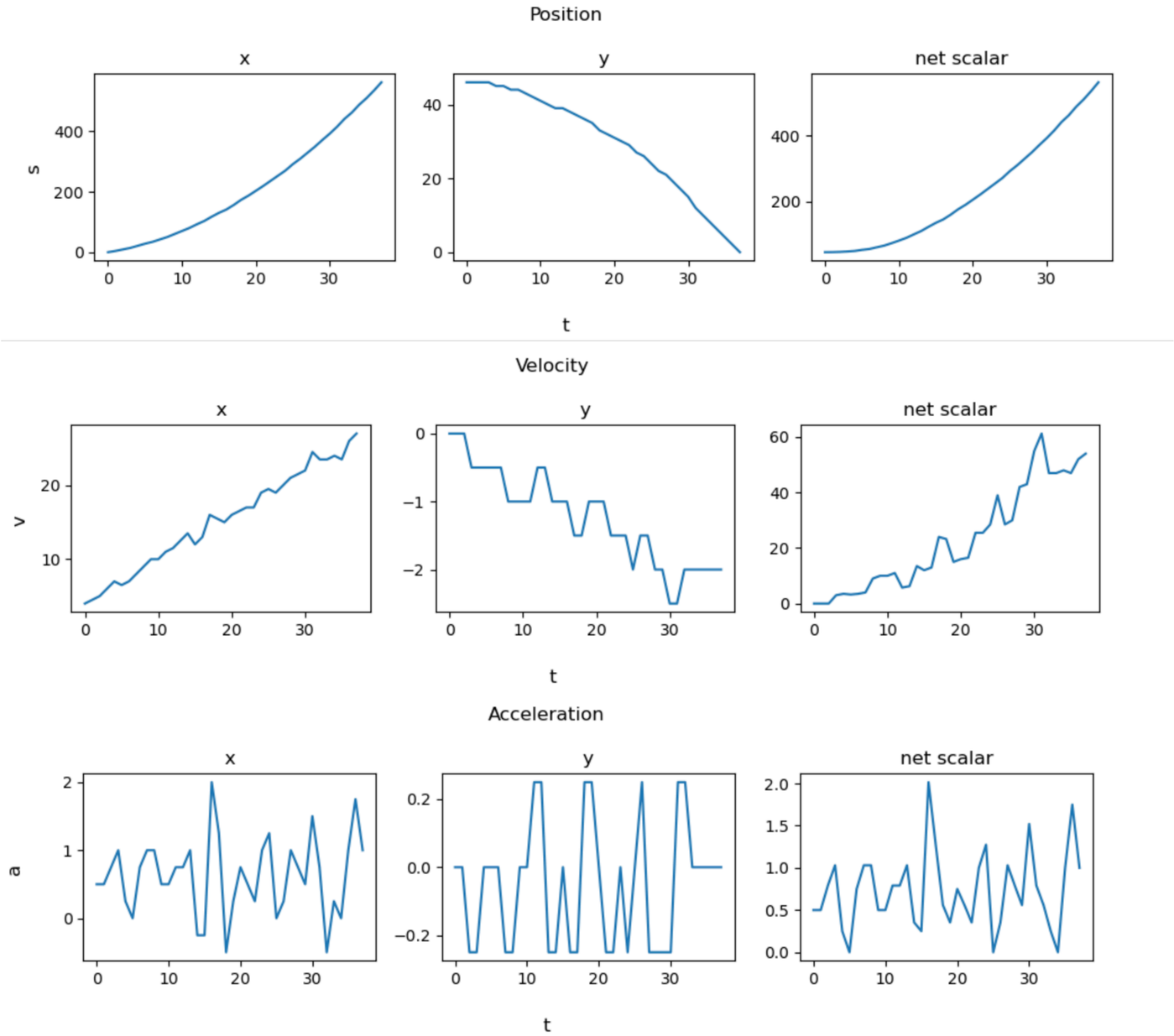
Solution: Use a series of matrix operations

Solution: Use a series of matrix operations

Solution: Use a series of matrix operations



Result

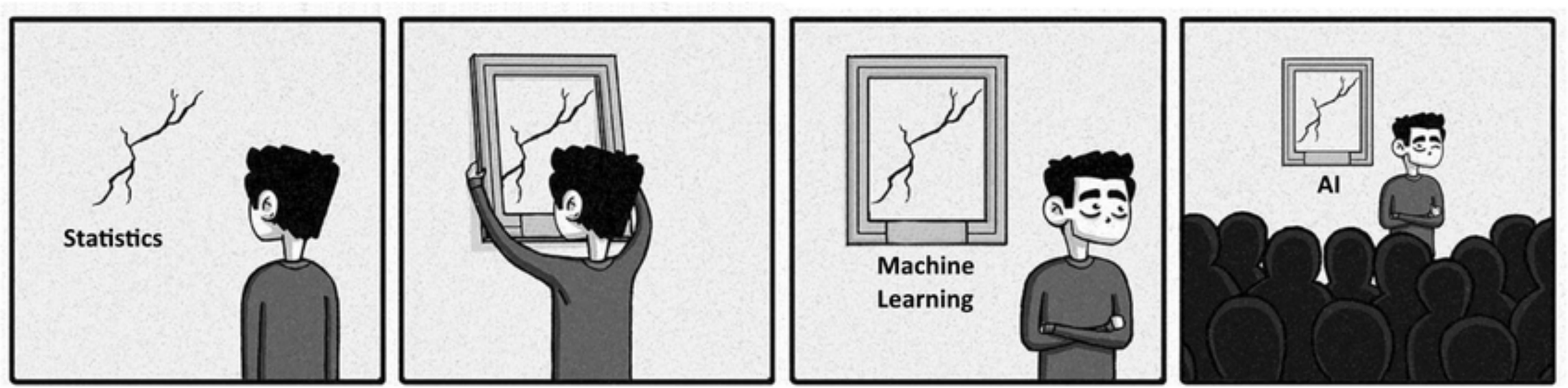

Position
x
y
net scalar
s
t
Velocity
x
y
net scalar
v
t
Acceleration
x
y
net scalar
a
t

Drawbacks

- Noisy outputs
- Need to tweak filter parameters for each video
- Might not generalize to other cases

Machine Learning

What is Machine Learning?



What is Machine Learning?

The study of computer algorithms that improve automatically through experience [1]

Fig 1. General applications of ML

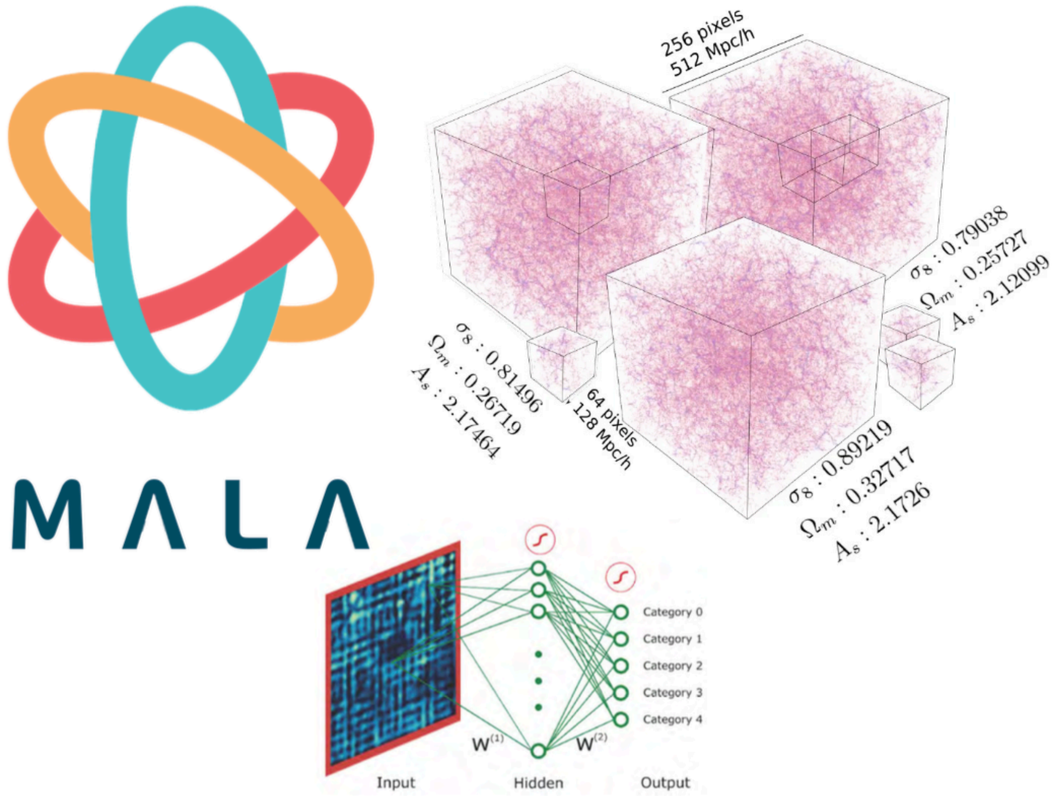
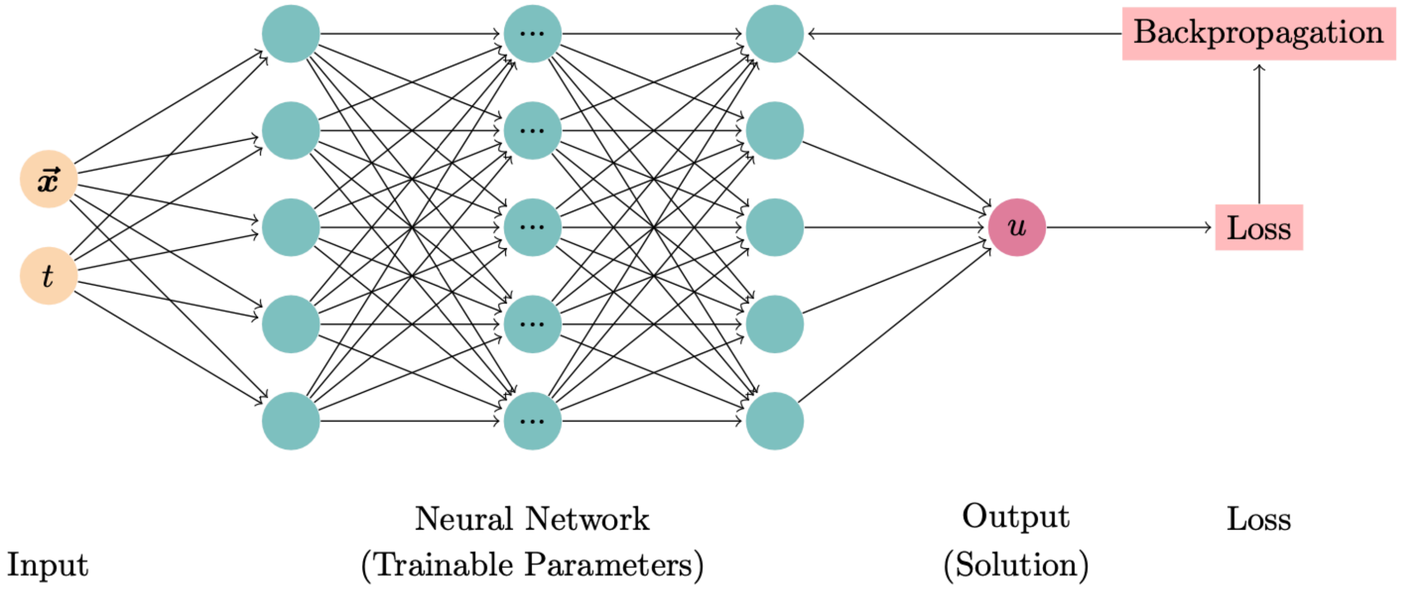


Fig 2. Science applications of ML[2,3,4]

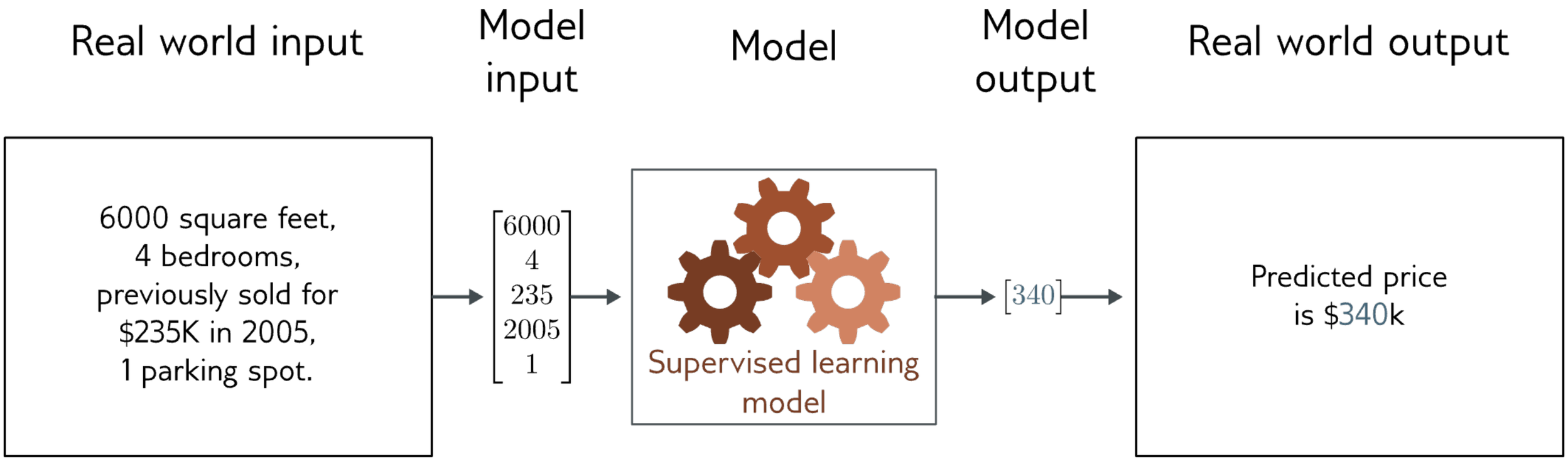
What is a neural network?

Optimise loss function based on the difference between true and predicted values



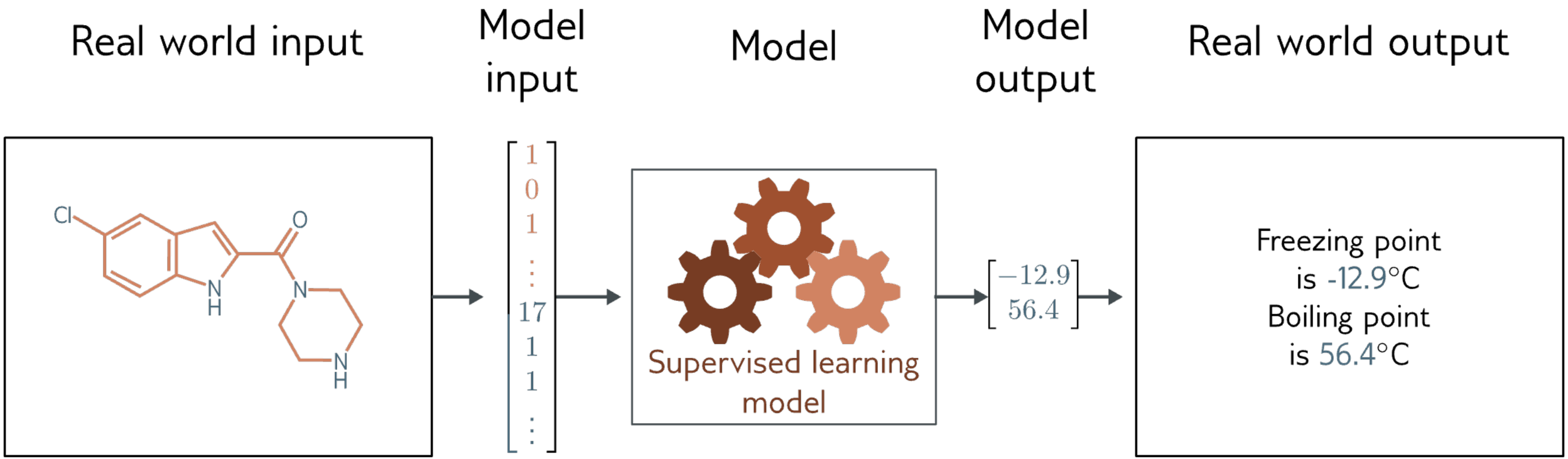
Feedforward Neural Network

What can you do with it?
Regression



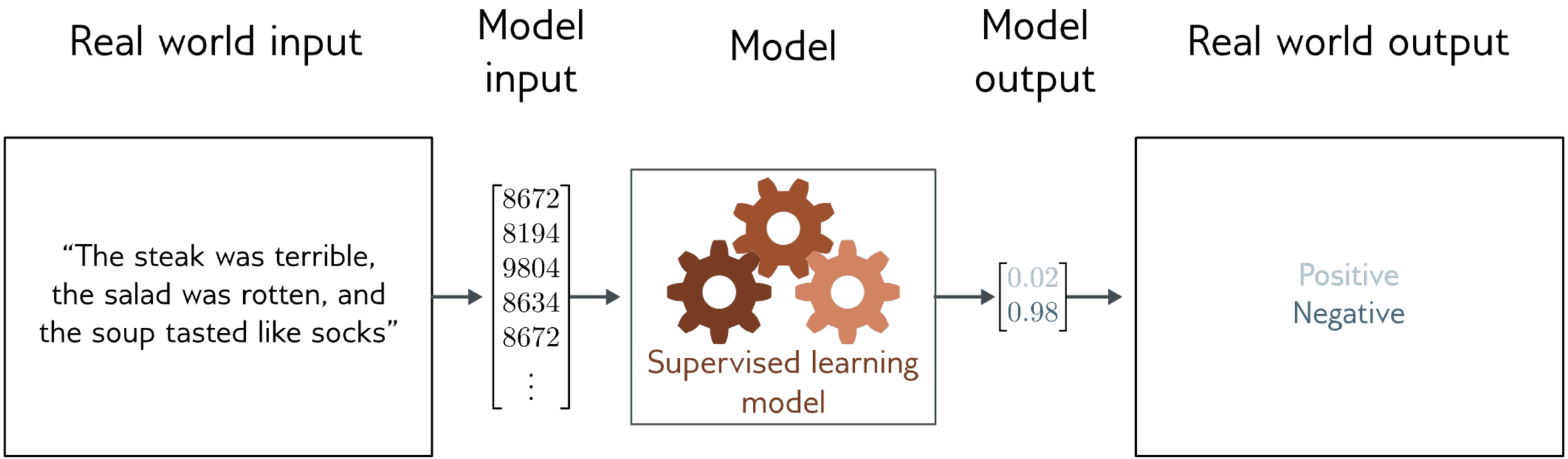
- Univariate regression problem (one output, real value)

Graph regression



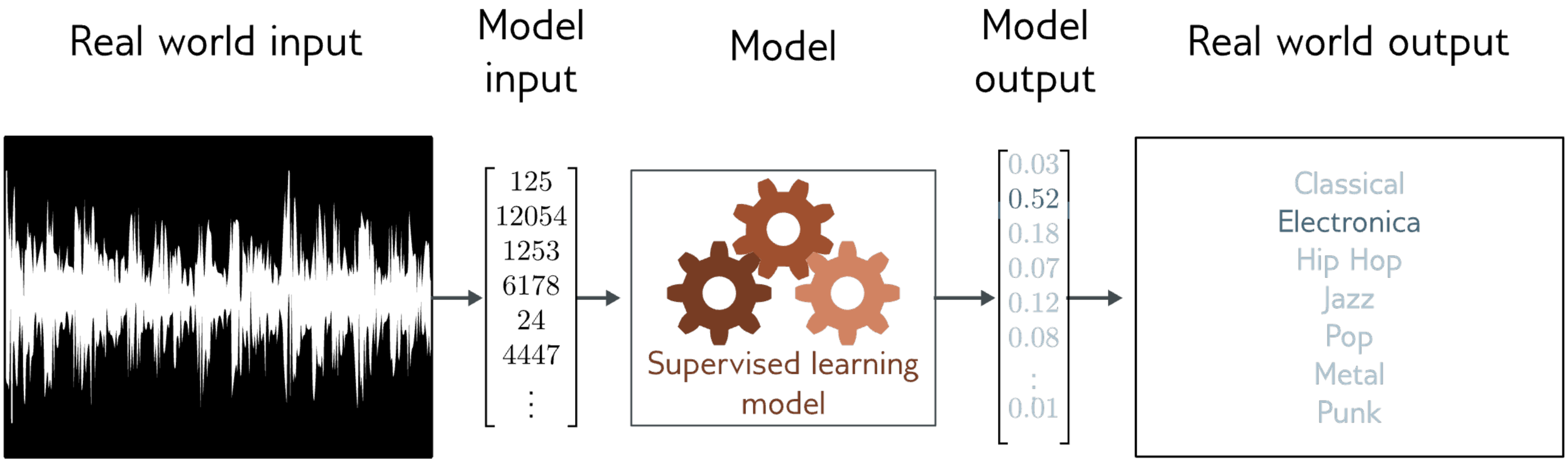
- Multivariate regression problem (>1 output, real value)

Text classification



- Binary classification problem (two discrete classes)

Music genre classification



- Multiclass classification problem (discrete classes, >2 possible values)

Why Neural Networks?

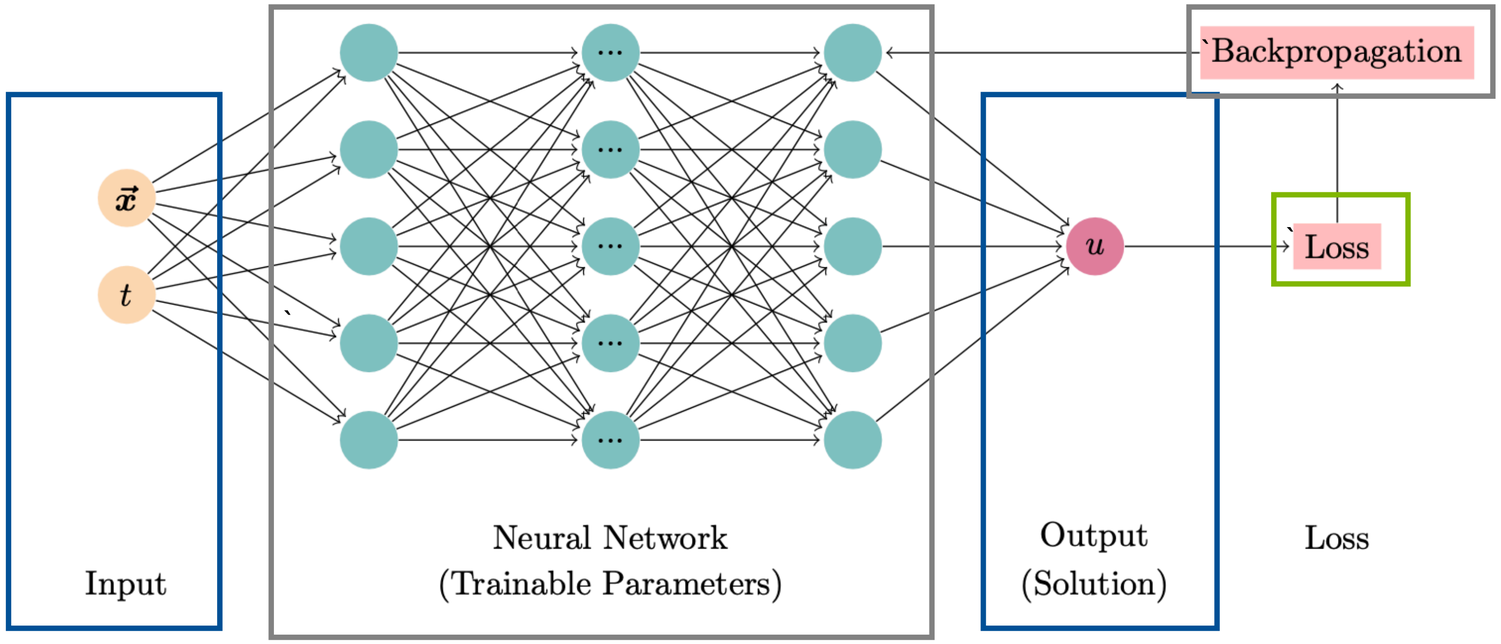
Universal Approximation Theorem

“a formal proof that, with enough hidden units, a shallow neural network can describe any continuous function to arbitrary precision”

Hornik et al., Multilayer feedforward networks are universal approximators, Neural Networks (1989)

What is a neural network?

Optimise loss function based on the difference between true and predicted values



Feedforward Neural Network

What is a neural network?

Vector function with tunable parameters θ

$$\mathbf{f}(\cdot;\theta) : \mathbb{R}^N \rightarrow (0,1)^K$$

Sample s in dataset S:

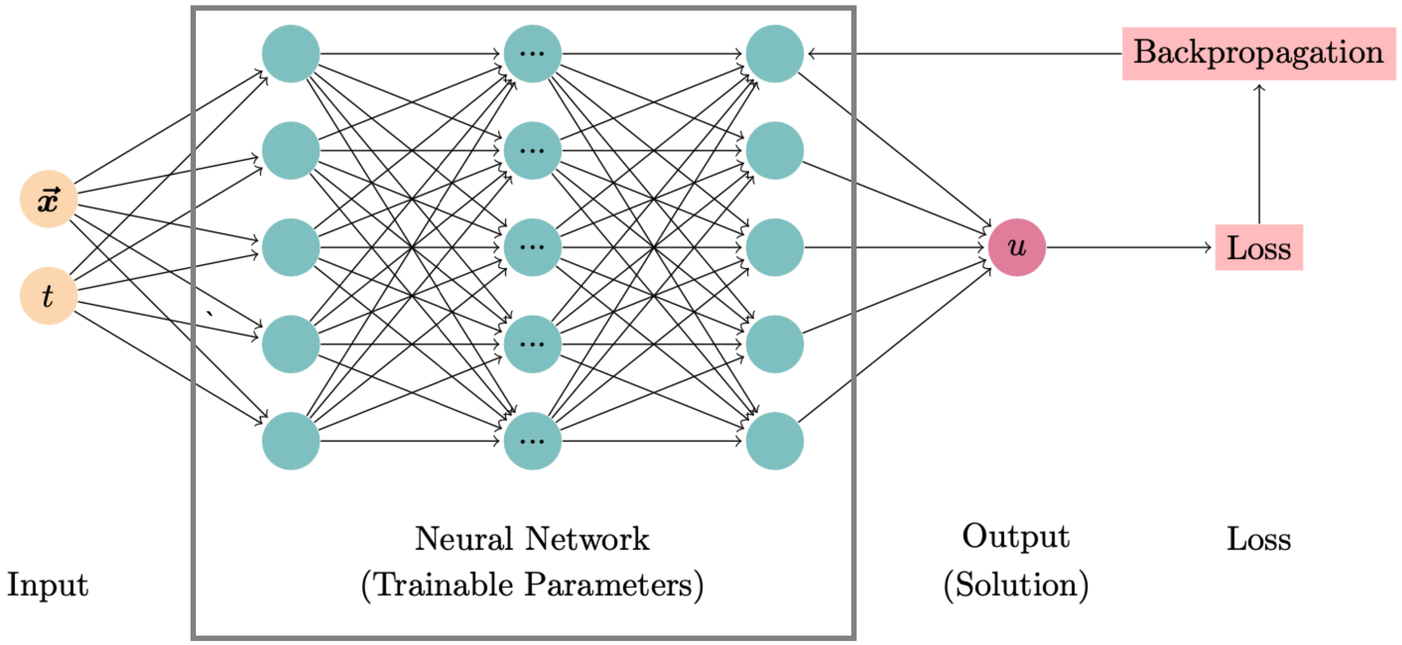
- input: $\mathbf{x}^s \in \mathbb{R}^N$
- expected output: $y^s \in [0, K-1]$

Output is a conditional probability distribution:

$$\mathbf{f}(\mathbf{x}^s;\theta)_c = P(Y = c | X = \mathbf{x}^s)$$

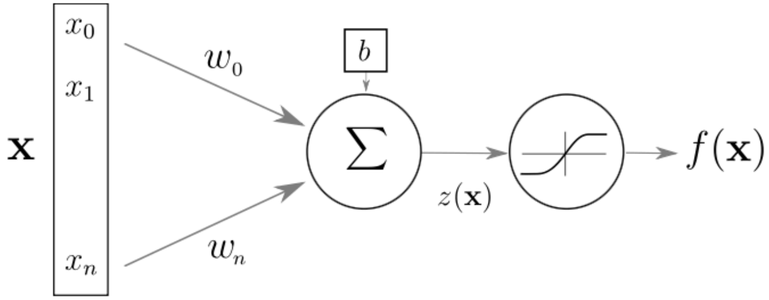
What is a neural network?

Optimise loss function based on the difference between true and predicted values



Feedforward Neural Network

Trainable Parameters: Perceptron

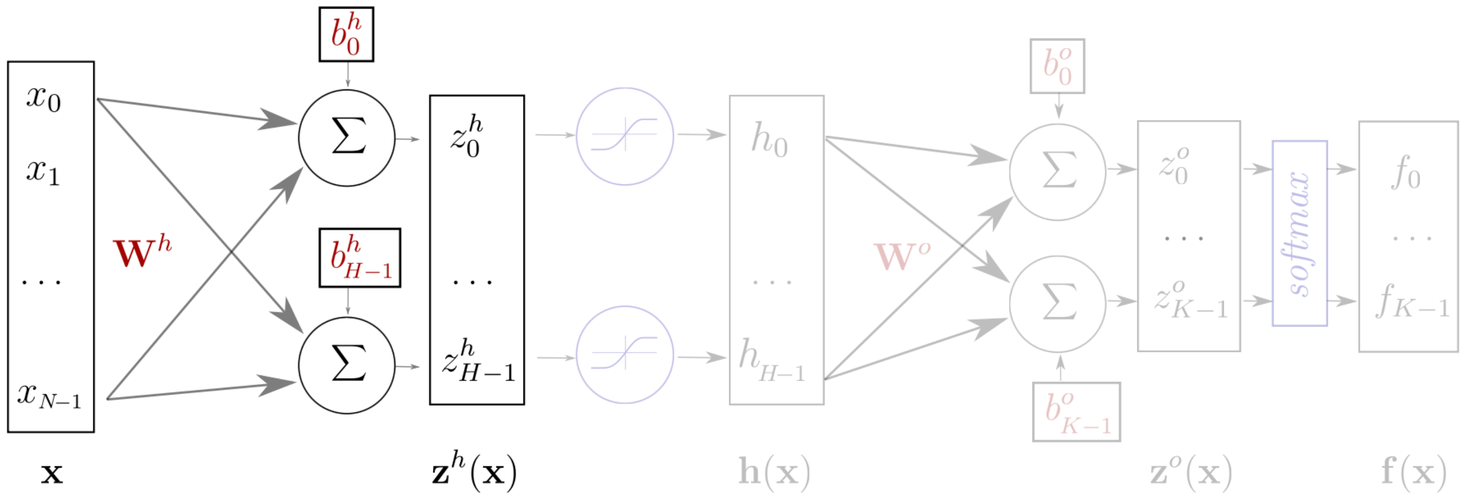


$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

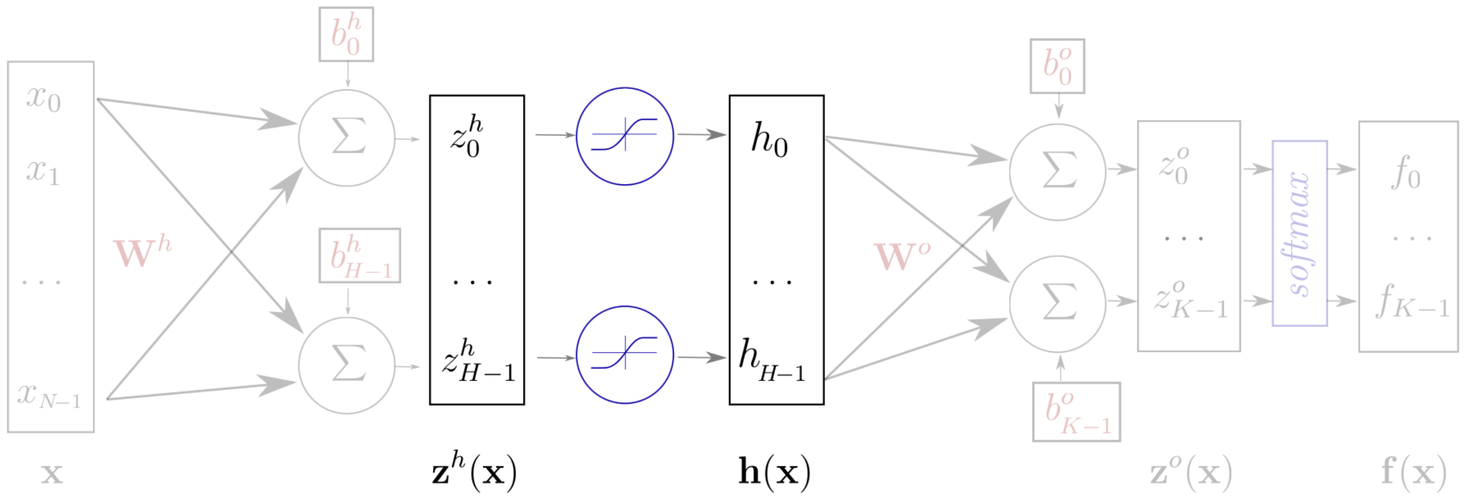
- $\mathbf{x}, f(\mathbf{x})$ input and output
- $z(\mathbf{x})$ pre-activation
- $\mathbf{w}, b$ weights and bias
- g activation function

Trainable Parameters: Multiple layers



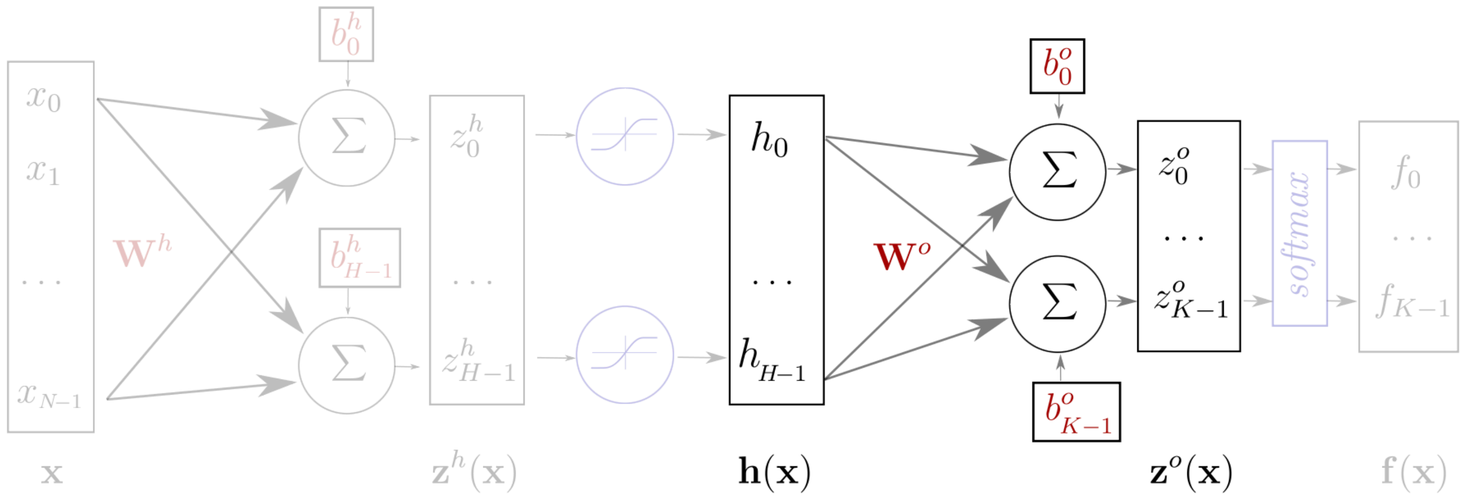
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

Trainable Parameters: Multiple layers



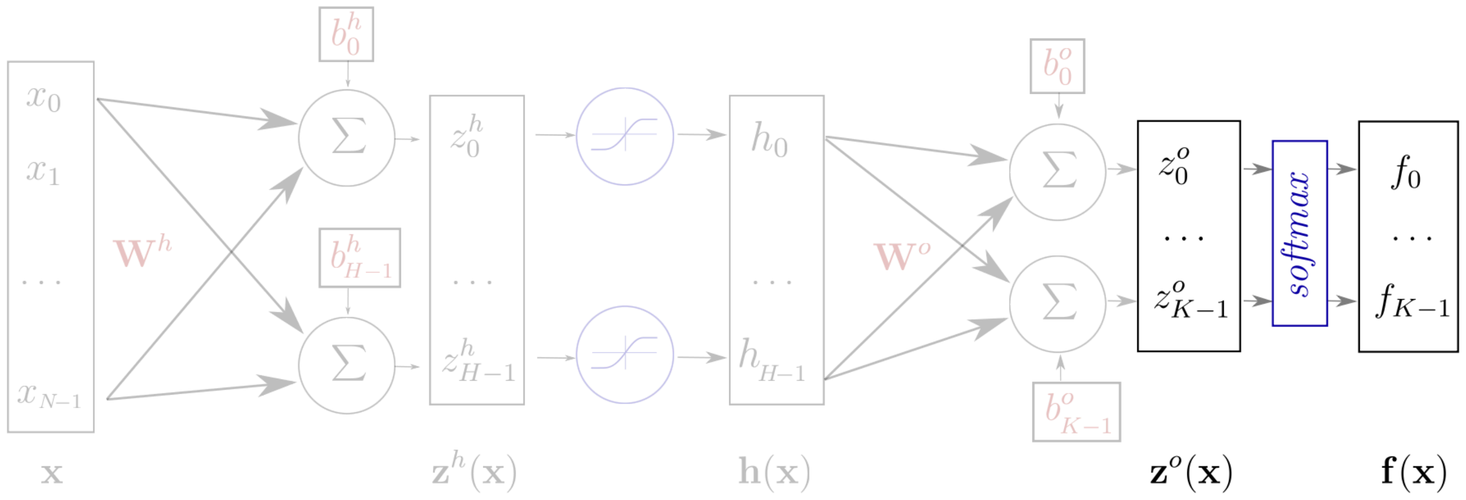
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

Trainable Parameters: Multiple layers



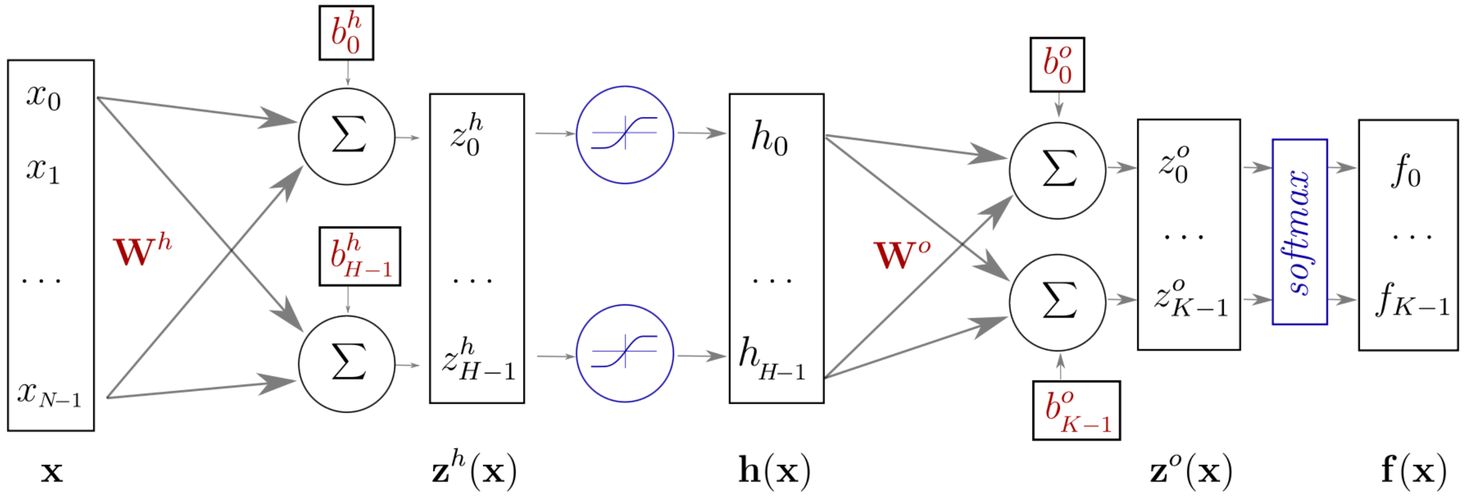
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

Trainable Parameters: Multiple layers

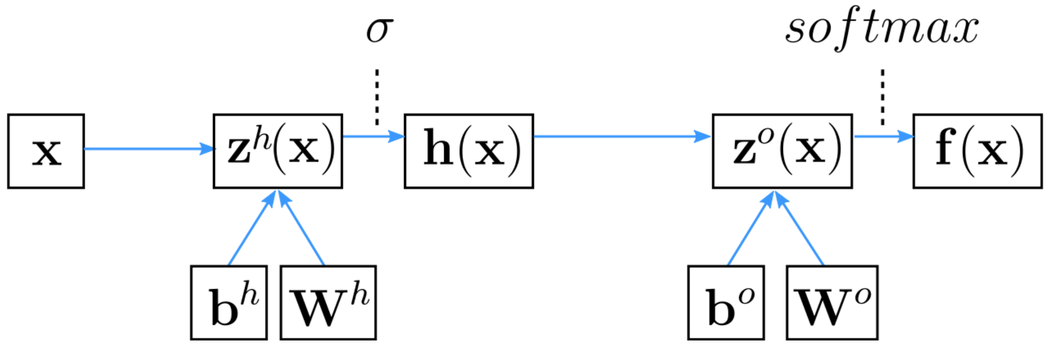


- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

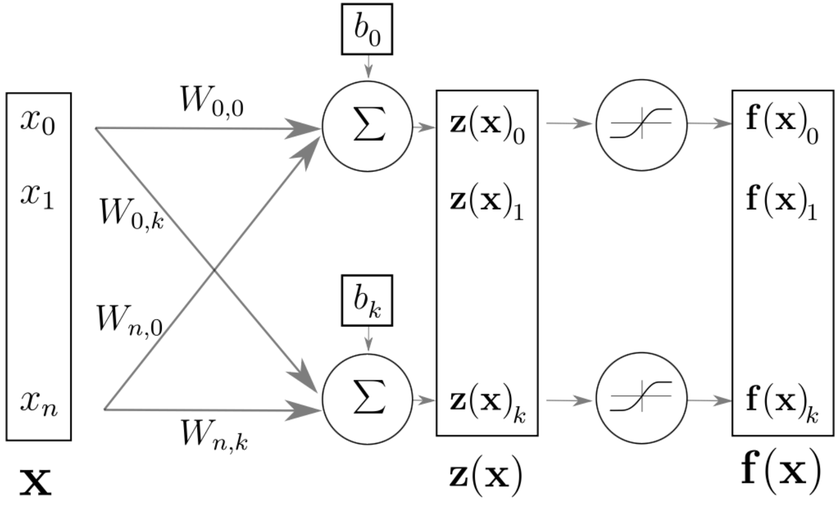
Trainable Parameters: Multiple layers



Alternate representation



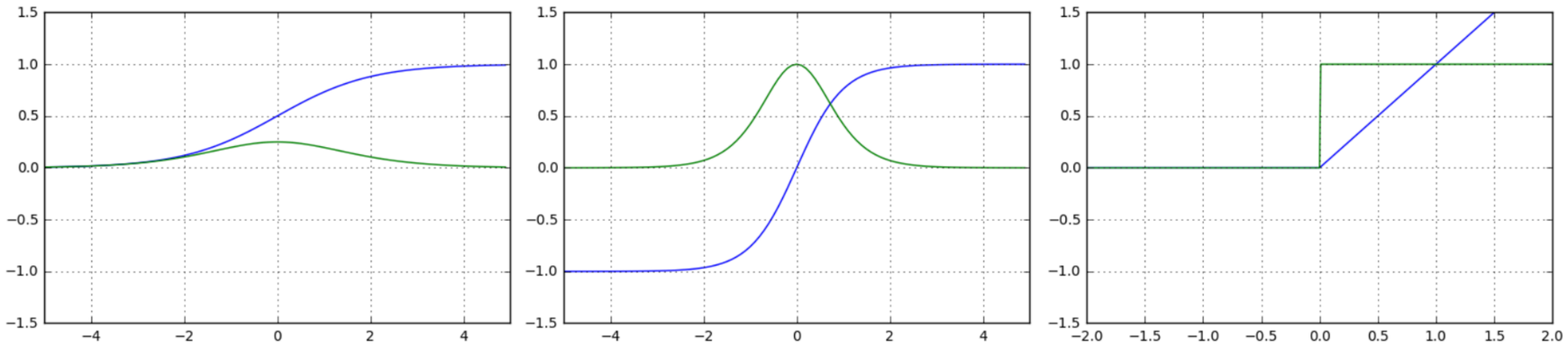
Trainable Parameters: Multiple layers



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

- $\mathbf{W}, \mathbf{b}$ now matrix and vector

Trainable Parameters: Activation Function



$$\mathrm{sigm}(x) = \frac{1}{1+e^{-x}}$$

$$\mathrm{sigm}'(x) = \mathrm{sigm}(x)(1-\mathrm{sigm}(x))$$

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

$$\tanh'(x) = 1-\tanh(x)^2$$

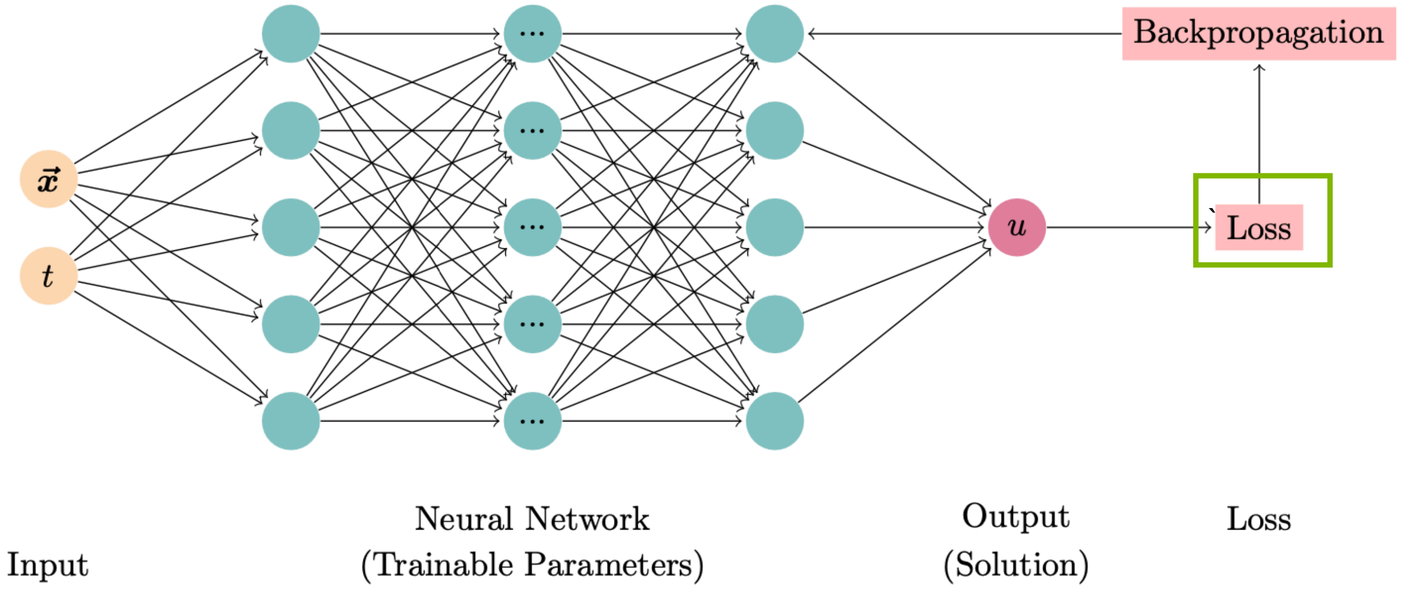
$$\mathrm{relu}(x) = \max(0,x)$$

$$\mathrm{relu}'(x) = 1_{x>0}$$

- blue: activation function
- green: derivative

What is a neural network?

Optimise loss function based on the difference between true and predicted values



Feedforward Neural Network

Loss Function

$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

$$\frac{\partial softmax(\mathbf{x})_i}{\partial x_j} = \begin{cases} softmax(\mathbf{x})_i \cdot (1 - softmax(\mathbf{x})_i) & i = j \\ -softmax(\mathbf{x})_i \cdot softmax(\mathbf{x})_j & i \neq j \end{cases}$$

Loss Function

$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

$$\frac{\partial softmax(\mathbf{x})_i}{\partial x_j} = \begin{cases} softmax(\mathbf{x})_i \cdot (1 - softmax(\mathbf{x})_i) & i = j \\ -softmax(\mathbf{x})_i \cdot softmax(\mathbf{x})_j & i \neq j \end{cases}$$

- vector of values in (0, 1) that add up to 1
- $p(Y = c | X = \mathbf{x}) = \mathrm{softmax}(\mathbf{z}(\mathbf{x}))_c$
- the pre-activation vector $\mathbf{z}(\mathbf{x})$ is often called "the logits"

Loss Function

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

Loss Function

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

Loss Function

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

example $y^s = 3$

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = l\left(\begin{bmatrix} f_0 \\ \dots \\ f_3 \\ \dots \\ f_{K-1} \end{bmatrix}, \begin{bmatrix} 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{bmatrix}\right) = -\log\ f_3$$

Loss Function

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

The cost function is the negative likelihood of the model computed on the full training set (for i.i.d. samples):

$$L_S(\theta) = -\frac{1}{|S|} \sum_{s \in S} \log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

Loss Function

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

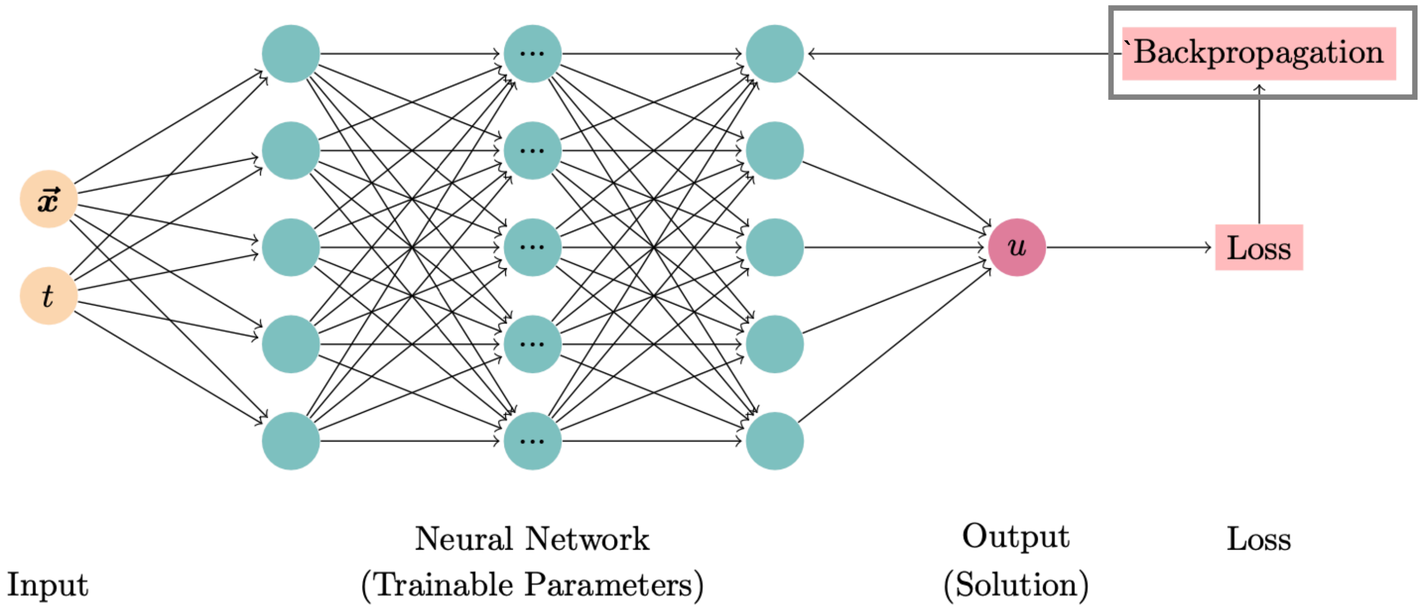
The cost function is the negative likelihood of the model computed on the full training set (for i.i.d. samples):

$$L_S(\theta) = -\frac{1}{|S|} \sum_{s \in S} \log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s} + \lambda \Omega(\theta)$$

$\lambda \Omega(\theta) = \lambda(||W^h||^2 + ||W^o||^2)$ is an optional regularization term.

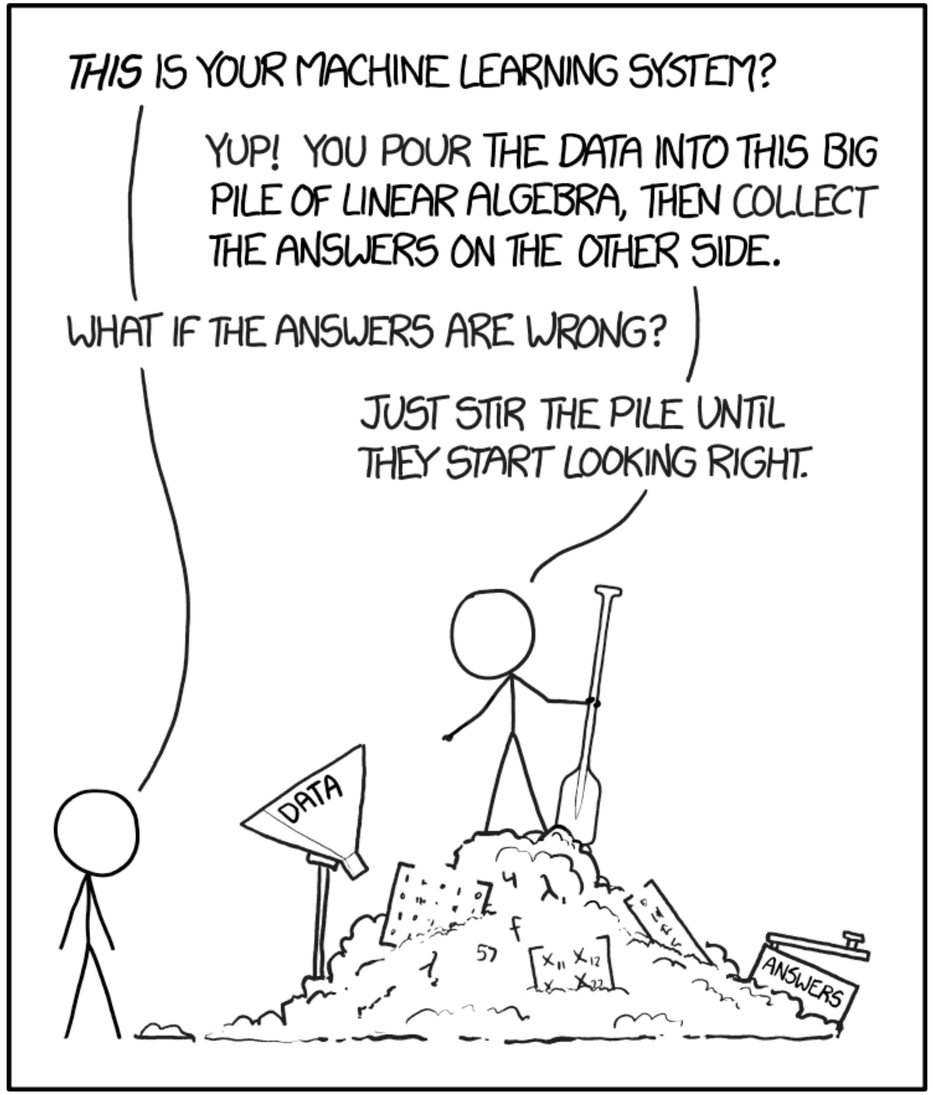
What is a neural network?

Optimise loss function based on the difference between true and predicted values



Feedforward Neural Network

How do we tune the weights?



Stochastic Gradient Descent

Initialize θ randomly

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla_\theta \mathcal{L}_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

Automatic Differentiation

Output Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^o_{i,j}}$

Output bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^o_i}$

Hidden Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^h_{i,j}}$

Hidden bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^h_i}$

Automatic Differentiation

$$x \longrightarrow \mathbf{v} \longrightarrow z$$

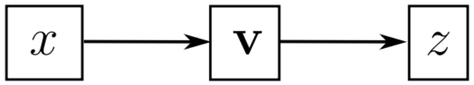
$$z = u(\mathbf{v}(x))$$

$$\frac{\partial z}{\partial x} = ?$$

$$\boxed{v_j}$$

$$\mathbf{v}$$

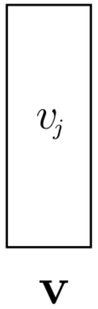
Automatic Differentiation



$$z = u(\mathbf{v}(x))$$

chain-rule

$$\frac{\partial z}{\partial x} = \sum_j \frac{\partial z}{\partial v_j} \frac{\partial v_j}{\partial x}$$



Automatic Differentiation

$$x \longrightarrow \mathbf{v} \longrightarrow z$$

$$z = u(\mathbf{v}(x))$$

chain-rule

$$\frac{\partial z}{\partial x} = \sum_j \frac{\partial z}{\partial v_j}\frac{\partial v_j}{\partial x} = \nabla u . \frac{\partial \mathbf{v}}{\partial x}$$

v_j	$\frac{\partial v_j}{\partial x}$	$\frac{\partial z}{\partial v_j}$
$\mathbf{v}$	$\frac{\partial \mathbf{v}}{\partial x}$	∇u

Automatic Differentiation

$$\mathbf{x} \longrightarrow \mathbf{v} \longrightarrow z$$

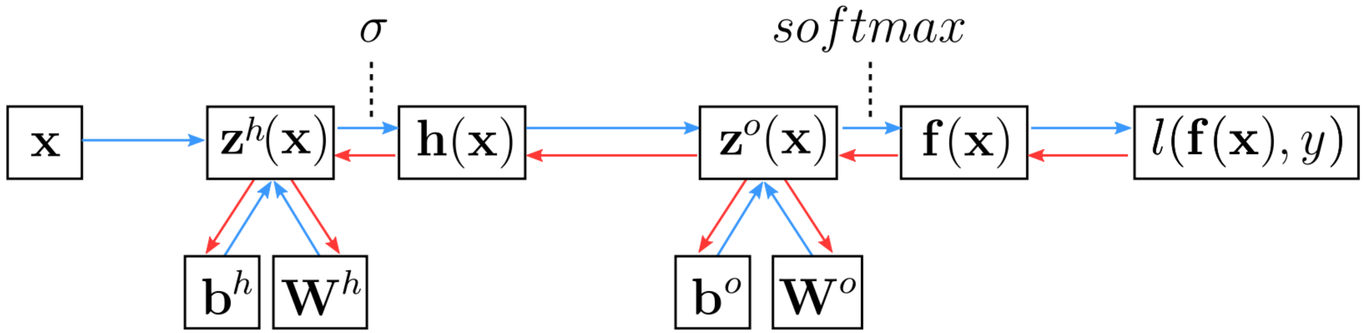
$$z = u(\mathbf{v}(\mathbf{x}))$$

chain-rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial v_j}\frac{\partial v_j}{\partial x_i} = \nabla u . \frac{\partial \mathbf{v}}{\partial x_i}$$

v_j	$\frac{\partial v_j}{\partial x_i}$	$\frac{\partial z}{\partial v_j}$
$\mathbf{v}$	$\frac{\partial \mathbf{v}}{\partial x_i}$	∇u

Automatic Differentiation



Compute partial derivatives of the loss

- $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{\partial -\log \mathbf{f}(\mathbf{x})_y}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y} = \frac{\partial \boldsymbol{l}}{\partial \mathbf{f}(\mathbf{x})_i}$
- $\frac{\partial \boldsymbol{l}}{\partial \mathbf{z}^o(\mathbf{x})_i} = ?$

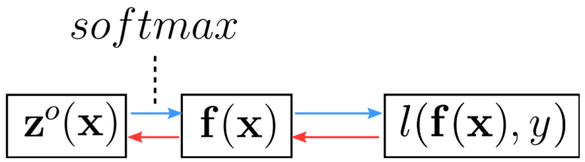
Automatic Differentiation

$$\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} = \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

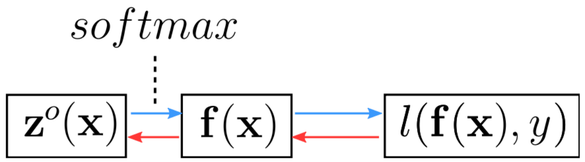
$$\frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y}$$

$$\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o(\mathbf{x}))$$



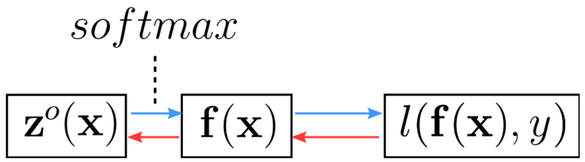
Automatic Differentiation

$$
\begin{aligned}
\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} &= \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= -\frac{1}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_y}{\partial \mathbf{z}^o(\mathbf{x})_i}
\end{aligned}
$$



Automatic Differentiation

$$
\begin{aligned}
\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} &= \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= -\frac{1}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_y}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \begin{cases} -\frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y (1 - softmax(\mathbf{z}^o(\mathbf{x}))_y) & \textbf{if } i = y \\ \frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y softmax(\mathbf{z}^o(\mathbf{x}))_i & \textbf{if } i \neq y \end{cases} \\
&= \begin{cases} -1 + \mathbf{f}(\mathbf{x})_y & \textbf{if } i = y \\ \mathbf{f}(\mathbf{x})_i & \textbf{if } i \neq y \end{cases}
\end{aligned}
$$

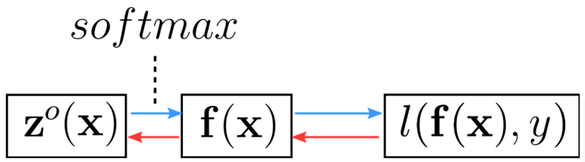


Automatic Differentiation

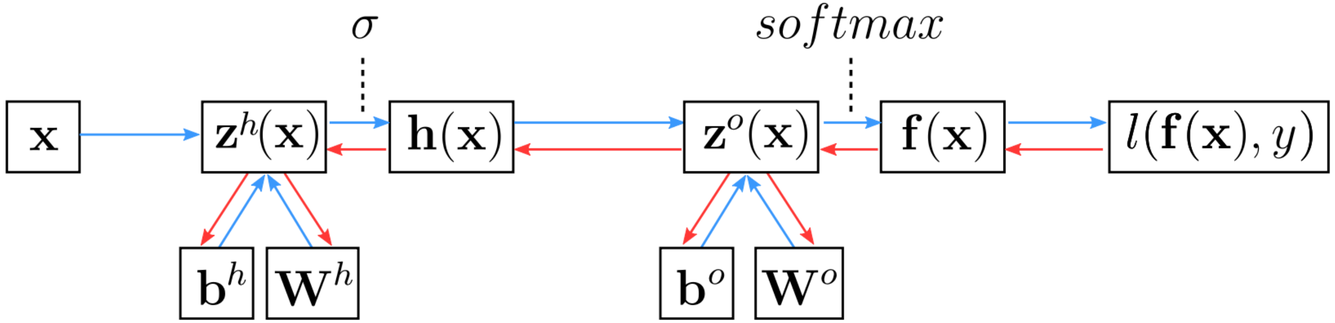
$$
\begin{aligned}
\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} &= \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= -\frac{1}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_y}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \begin{cases} -\frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y (1 - softmax(\mathbf{z}^o(\mathbf{x}))_y) & \textbf{if } i = y \\ \frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y softmax(\mathbf{z}^o(\mathbf{x}))_i & \textbf{if } i \neq y \end{cases} \\
&= \begin{cases} -1 + \mathbf{f}(\mathbf{x})_y & \textbf{if } i = y \\ \mathbf{f}(\mathbf{x})_i & \textbf{if } i \neq y \end{cases}
\end{aligned}
$$

$$\nabla_{\mathbf{z}^o(\mathbf{x})} l(\mathbf{f}(\mathbf{x}), y) = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$$

$\mathbf{e}(y)$: one-hot encoding of y



Automatic Differentiation

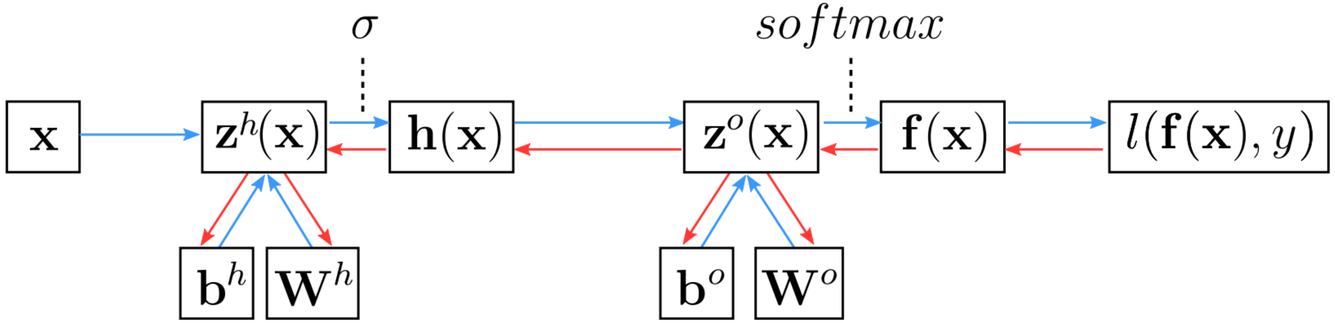


Gradients

- $\nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$
- $\nabla_{\mathbf{b}^o} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$

because $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$ and then $\frac{\partial \mathbf{z}^o(\mathbf{x})_i}{\partial \mathbf{b}^o_j} = 1_{i=j}$

Automatic Differentiation



Partial derivatives related to $\mathbf{W}^o$

- $\frac{\partial \boldsymbol{l}}{\partial W^o_{i,j}} = \sum_k \frac{\partial \boldsymbol{l}}{\partial \mathbf{z}^o(\mathbf{x})_k} \frac{\partial \mathbf{z}^o(\mathbf{x})_k}{\partial W^o_{i,j}}$
- $\nabla_{\mathbf{W}^o} \boldsymbol{l} = (\mathbf{f}(\mathbf{x}) - \mathbf{e}(y)).\, \mathbf{h}(\mathbf{x})^\top$

Automatic Differentiation

Compute activation gradients

- $\nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$

Compute layer params gradients

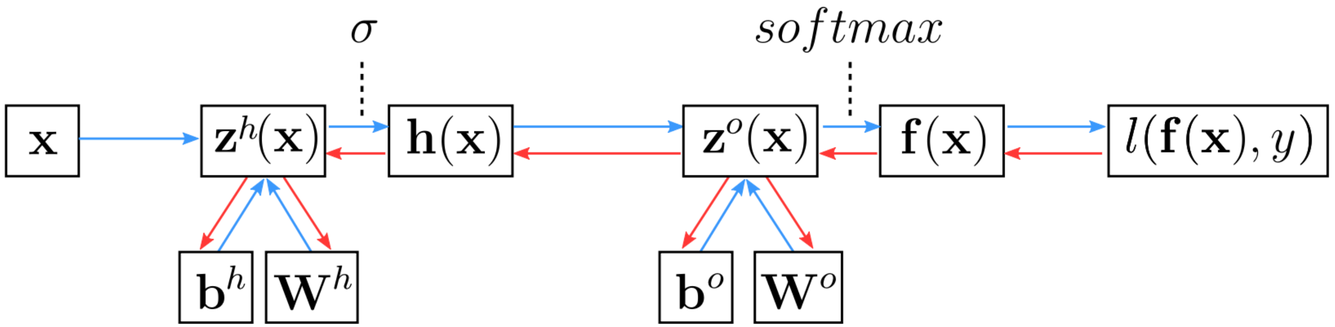
- $\nabla_{\mathbf{W}^o} \boldsymbol{l} = \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} \cdot \mathbf{h}(\mathbf{x})^\top$
- $\nabla_{\mathbf{b}^o} \boldsymbol{l} = \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l}$

Compute prev layer activation gradients

- $\nabla_{\mathbf{h}(\mathbf{x})} \boldsymbol{l} = \mathbf{W}^{o\top} \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l}$
- $\nabla_{\mathbf{z}^h(\mathbf{x})} \boldsymbol{l} = \nabla_{\mathbf{h}(\mathbf{x})} \boldsymbol{l} \odot \sigma'(\mathbf{z}^{\mathbf{h}}(\mathbf{x}))$

Automatic Differentiation

- Modern deep learning frameworks compute derivatives automatically
- You just have to specify the model and the loss
- How? Automatic differentiation
- Each component knows how to compute its own derivative
  - ReLU knows how to compute deriv of output w.r.t. input
  - Linear function knows how to compute deriv of output w.r.t. input
  - Linear function knows how to compute deriv of output w.r.t. parameter
- You specify how the order of the components
- It can compute the chain of derivatives
- Works with branches as long as it's still an acyclic graph



Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

Generating Loss Functions

- Binary classification: $y \in [0, 1]$
  - $Y|X = \mathbf{x} \sim Bernoulli(b = f(\mathbf{x}; \theta))$
  - output function: $logistic(x) = \frac{1}{1+e^{-x}}$
  - loss function: binary cross-entropy
- Multiclass classification: $y \in [0, K-1]$
  - $Y|X = \mathbf{x} \sim Multinoulli(\mathbf{p} = \mathbf{f}(\mathbf{x}; \theta))$
  - output function: $softmax$
  - loss function: categorical cross-entropy
- Continuous output: $\mathbf{y} \in \mathbb{R}^n$
  - $Y|X = \mathbf{x} \sim \mathcal{N}(\mu = \mathbf{f}(\mathbf{x}; \theta), \sigma^2 \mathbf{I})$
  - output function: Identity
  - loss function: square loss

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

Optimizers

Accumulate gradients across successive updates:

$$m_t = \gamma m_{t-1} + \eta \nabla_\theta L_{B_t}(\theta_{t-1})$$
$$\theta_t = \theta_{t-1} - m_t$$

γ is typically set to 0.9

Larger updates in directions where the gradient sign is constant to accelerate in low curvature areas

Nesterov accelerated gradient

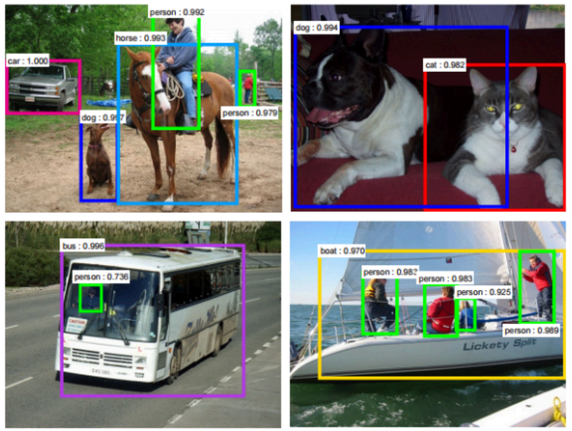
$$m_t = \gamma m_{t-1} + \eta \nabla_\theta L_{B_t}(\theta_{t-1} - \gamma m_{t-1})$$
$$\theta_t = \theta_{t-1} - m_t$$

Better at handling changes in gradient direction.

Convolutional Neural Networks

[Krizhevsky 2012]

[Ciresan et al. 2013]



[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

Convolutional Neural Networks

Convolutional Neural Networks

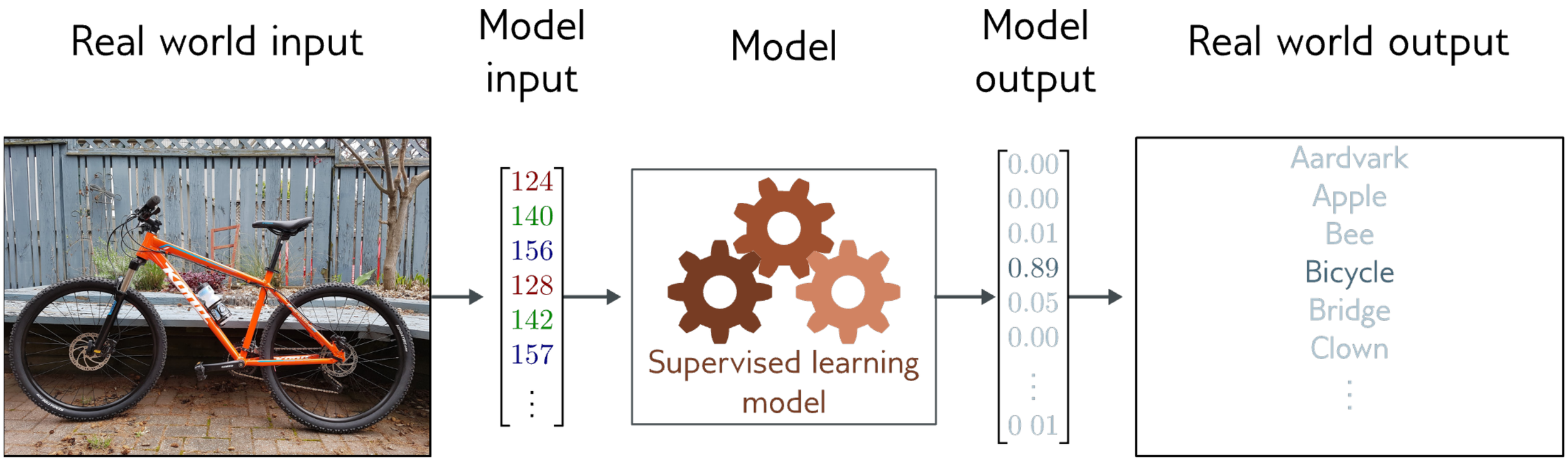
Speech recognition & speech synthesis

Natural Language Processing

Protein/DNA binding prediction

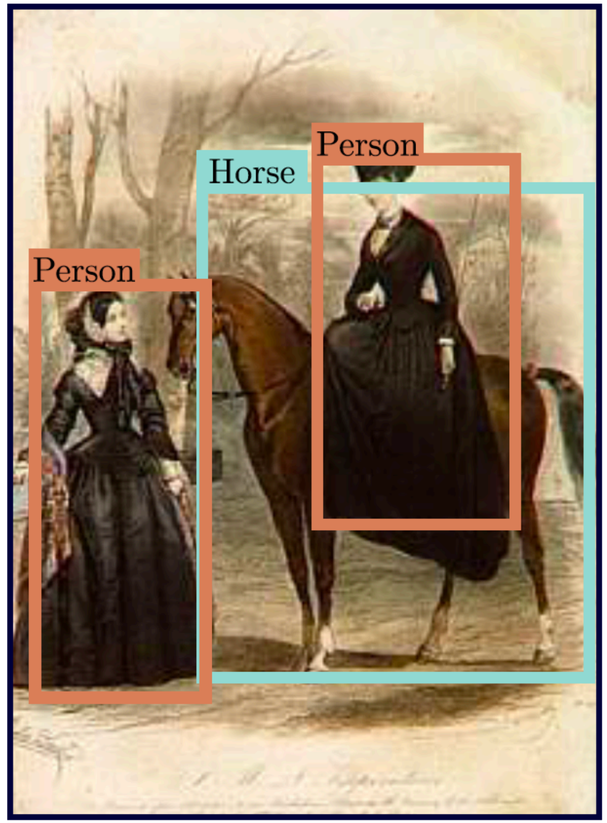
Any problem with a spatial (or sequential) structure

Image classification



- Multiclass classification problem (discrete classes, >2 possible classes)
- Convolutional network

Object detection



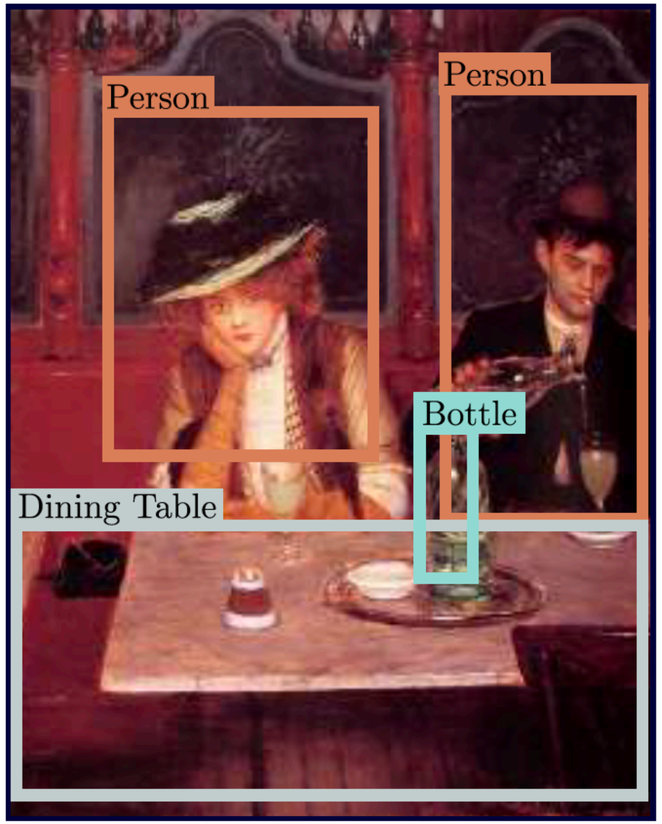
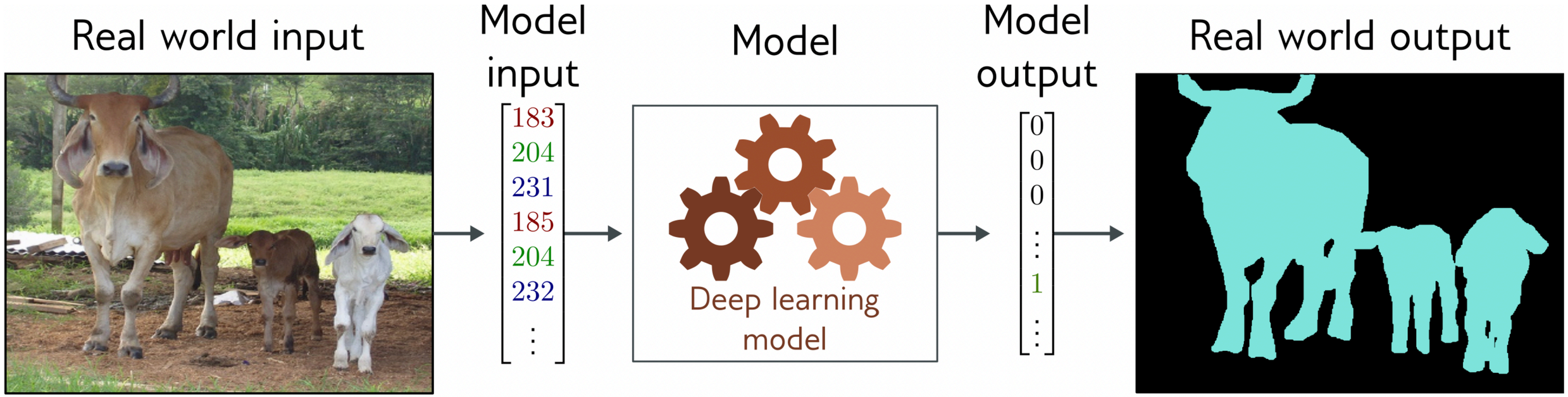


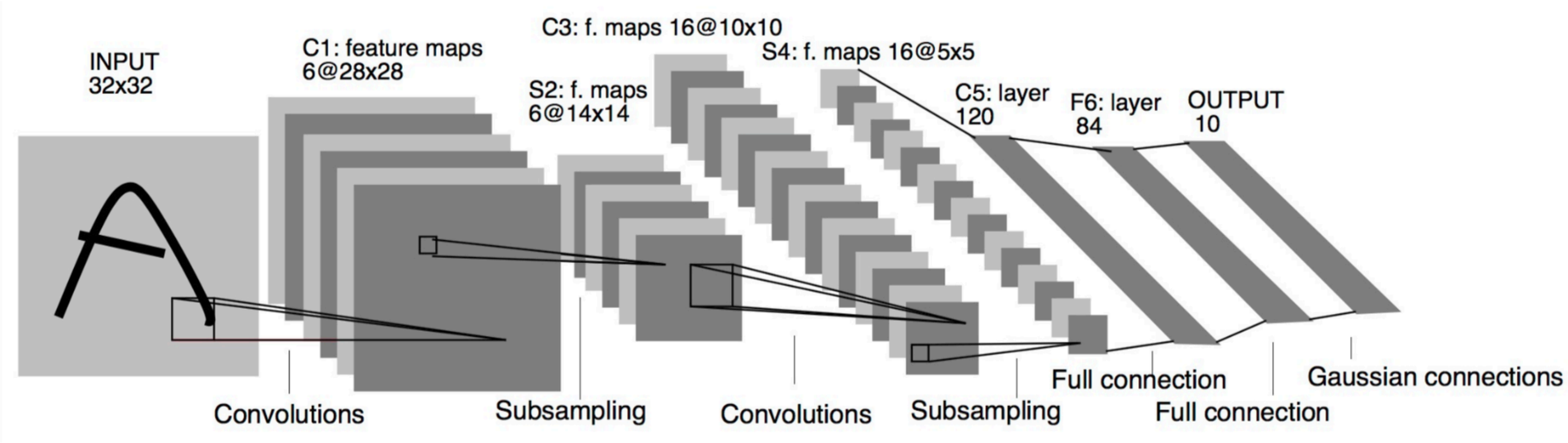
Image segmentation



- Multivariate binary classification problem (many outputs, two discrete classes)
- Convolutional encoder-decoder network

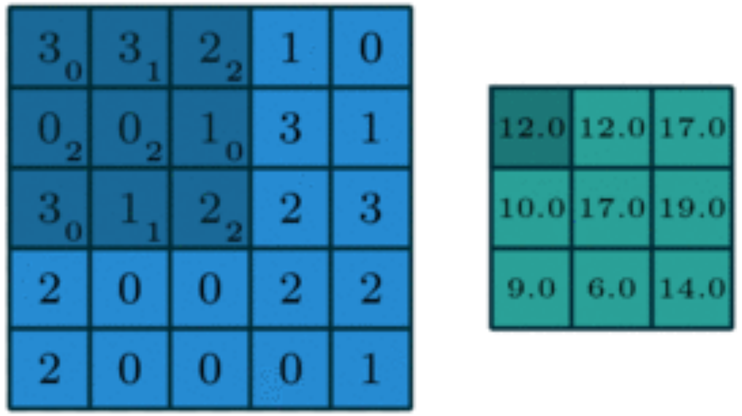
Convolutional Neural Networks

CNN = Convolutional Neural Networks = ConvNet



Original ConvNet by Le Cunn (1998)

Convolutional Neural Networks



- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

The activation obtained by sliding the 3×3 window and computing:

$$z(x) = relu(\mathbf{w}^T x + b)$$

Convolutional Neural Networks

Local connectivity

- A neuron depends only on a few local input neurons
- Translation invariance

Convolutional Neural Networks

Local connectivity

- A neuron depends only on a few local input neurons
- Translation invariance

Comparison to Fully connected

- Parameter sharing: reduce overfitting
- Make use of spatial structure: **strong prior** for vision!

Convolutional Neural Networks

Discrete convolution (actually cross-correlation) between two functions f and g:

$$(f \star g)(x) = \sum_{a+b=x} f(a).\, g(b) = \sum_{a} f(a).\, g(x+a)$$

2D-convolutions (actually 2D cross-correlation):

$$(f \star g)(x, y) = \sum_{n} \sum_{m} f(n, m).\, g(x+n, y+m)$$

f is a convolution **kernel** or **filter** applied to the 2-d map g (our image)

Convolutional Neural Networks

- Image: im of dimensions 5×5
- Kernel: k of dimensions 3×3

$$(k \star im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m).\, im(x + n - 1, y + m - 1)$$

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

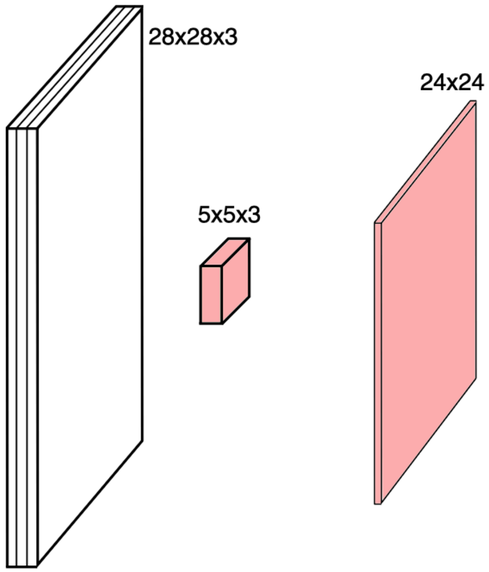
Convolutional Neural Networks

Colored image = tensor of shape `(height, width, channels)`

Convolutional Neural Networks

Colored image = tensor of shape `(height, width, channels)`

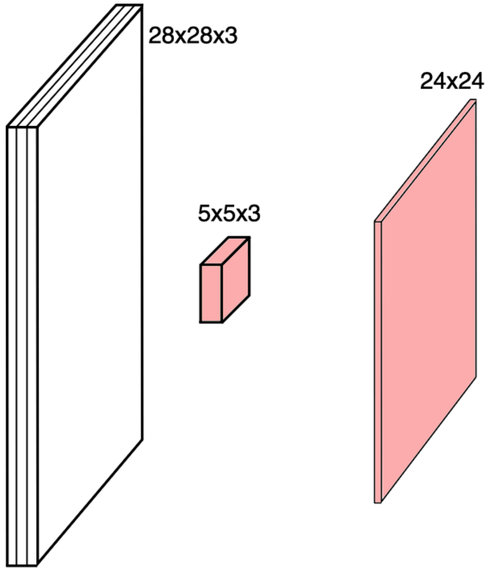
Convolutions are usually computed for each channel and summed:



Convolutional Neural Networks

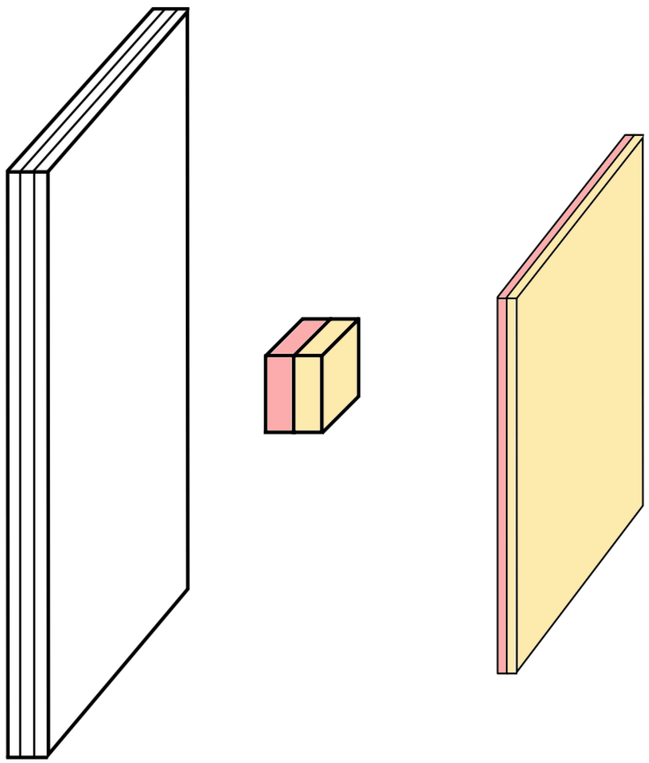
Colored image = tensor of shape `(height, width, channels)`

Convolutions are usually computed for each channel and summed:

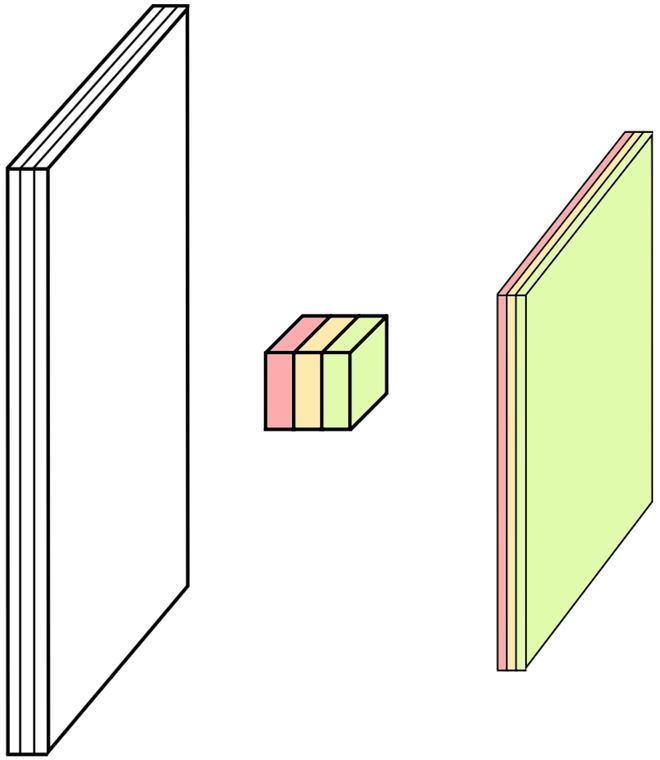


$$(k \star im^{color}) = \sum_{c=0}^{2} k^c \star im^c$$

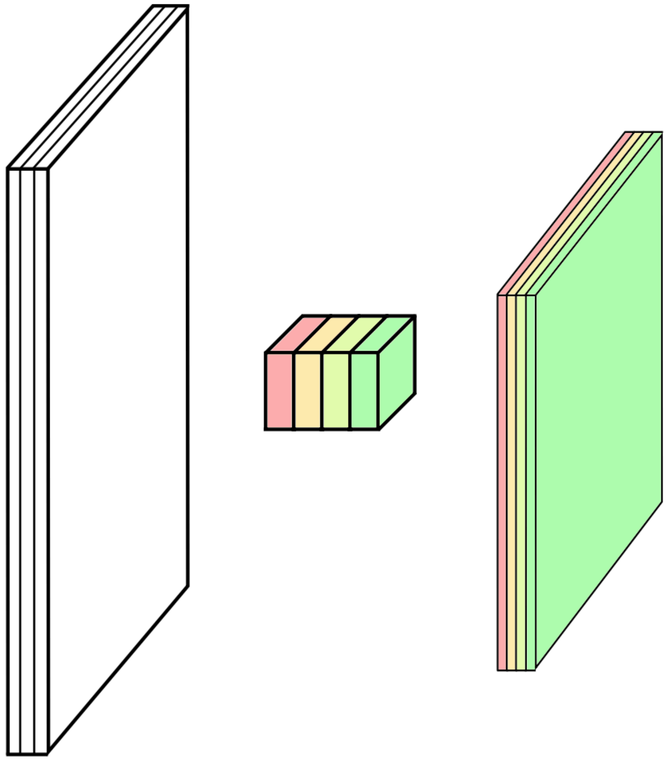
Convolutional Neural Networks: Multiple Convolution



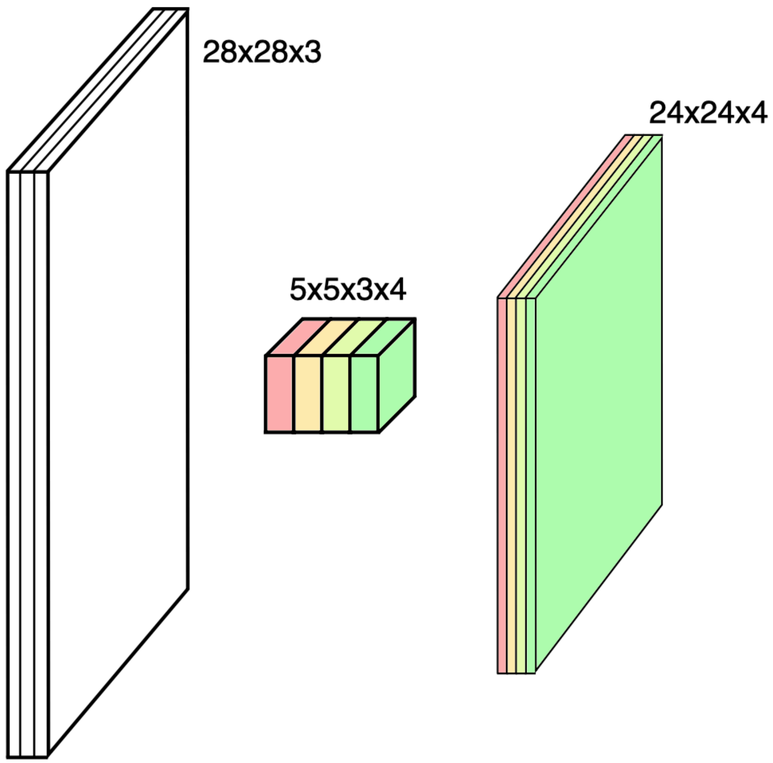
Convolutional Neural Networks: Multiple Convolution



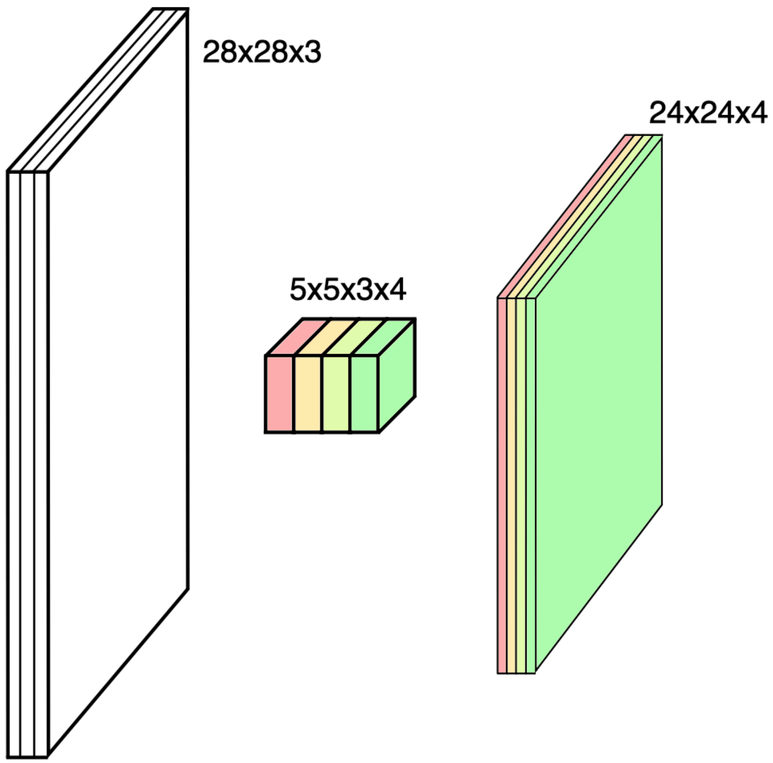
Convolutional Neural Networks: Multiple Convolution



Convolutional Neural Networks: Multiple Convolution



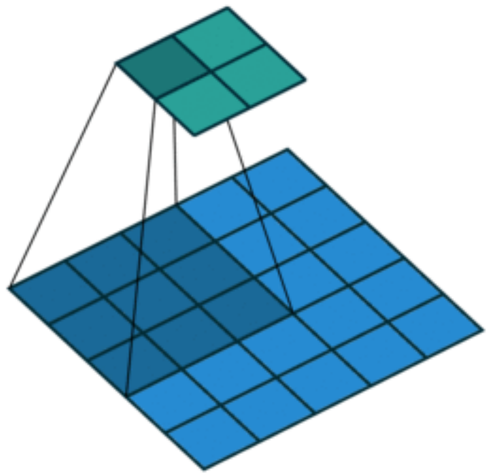
Convolutional Neural Networks: Multiple Convolution



- Kernel size aka receptive field (usually 1, 3, 5, 7, 11)
- Output dimension: `length - kernel_size + 1`

Convolutional Neural Networks: Strides

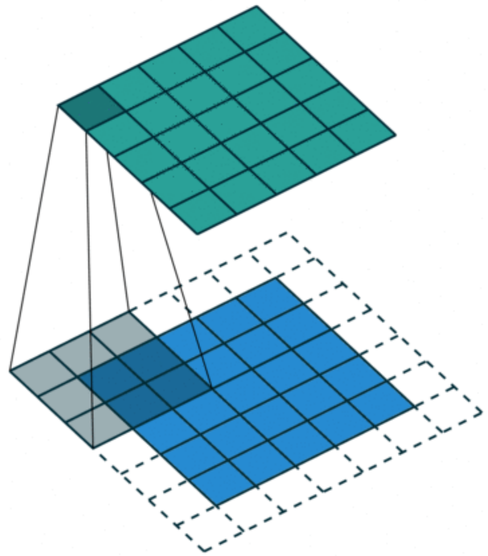
- Strides: increment step size for the convolution operator
- Reduces the size of the output map



Example with kernel size 3×3 and a stride of 2 (image in blue)

Convolutional Neural Networks: Padding

- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually: fill with 0s

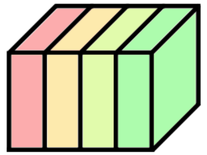


Convolutional Neural Networks: Shape

Kernel or **Filter** shape (F, F, C^i, C^o)

- $F \times F$ kernel size,
- C^i input channels
- C^o output channels

5x5x3x4



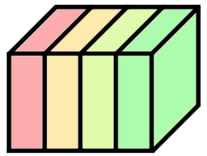
Number of parameters: $(F \times F \times C^i + 1) \times C^o$

Convolutional Neural Networks: Shape

Kernel or **Filter** shape (F, F, C^i, C^o)

- $F \times F$ kernel size,
- C^i input channels
- C^o output channels

5x5x3x4



Number of parameters: $(F \times F \times C^i + 1) \times C^o$

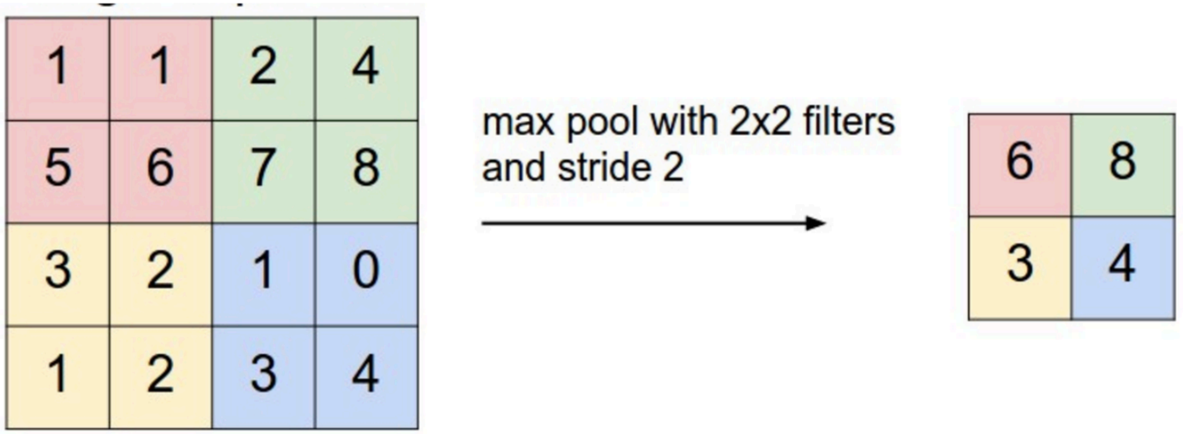
Activations or **Feature maps** shape:

- Input (W^i, H^i, C^i)
- Output (W^o, H^o, C^o)

$$W^o = (W^i - F + 2P)/S + 1$$

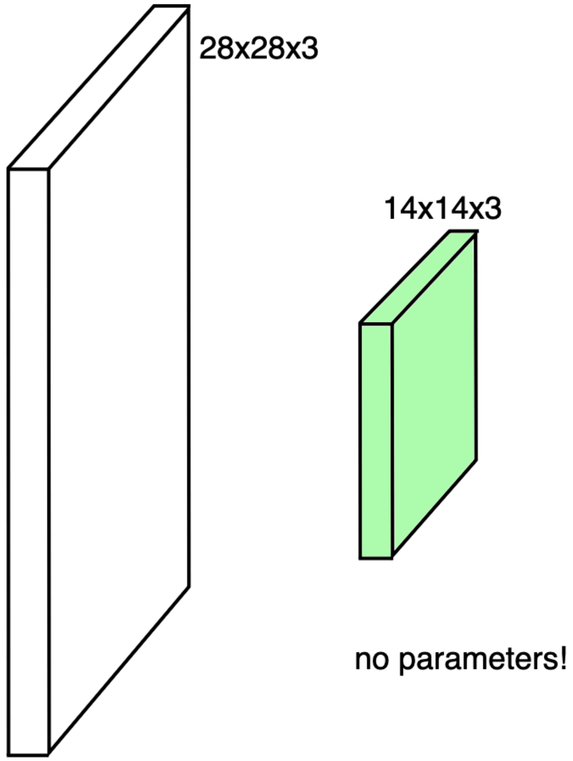
Convolutional Neural Networks: Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



Convolutional Neural Networks: Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



Setting up a Convolutional Net

Input

Setting up a Convolutional Net

Input

Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

Setting up a Convolutional Net

Input

Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

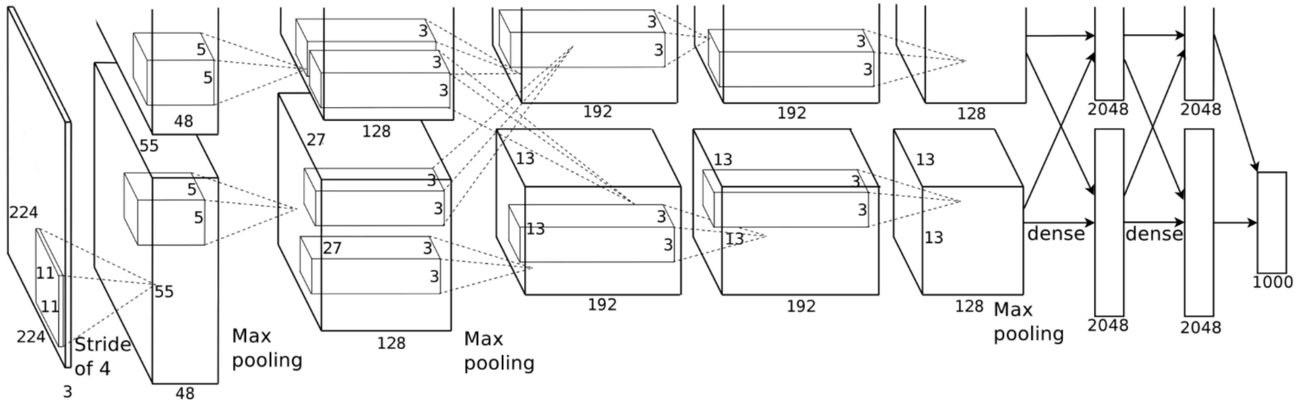
Output

- Fully connected layers
- Softmax

Setting up a Convolutional Net: Augmentation

Setting up a Convolutional Net: Augmentation

AlexNet

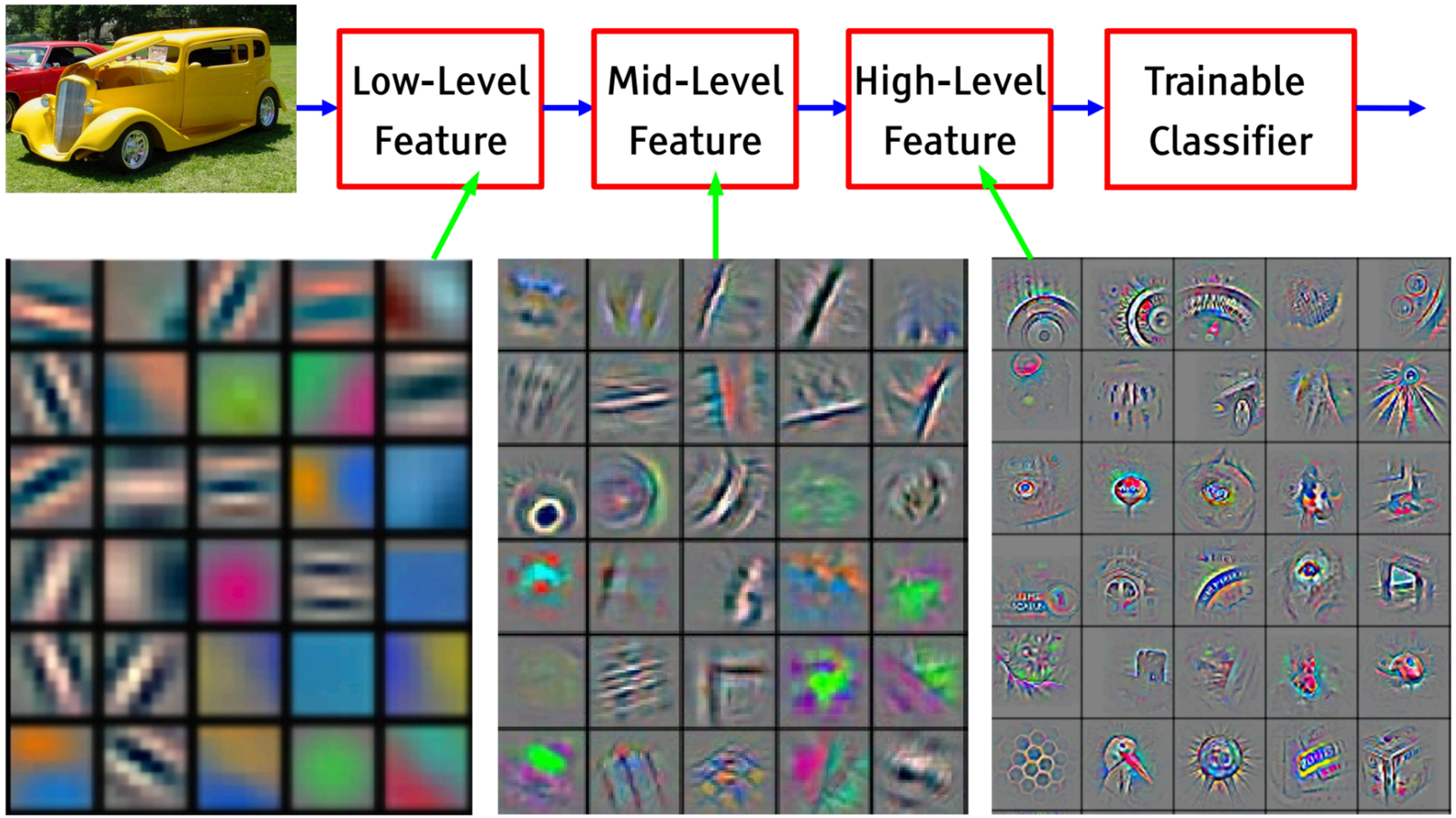


First conv layer: kernel 11x11x3x96 stride 4

- Kernel shape: `(11,11,3,96)`
- Output shape: `(55,55,96)`
- Number of parameters: `34,944`
- Equivalent MLP parameters: `43.7 x 1e9`

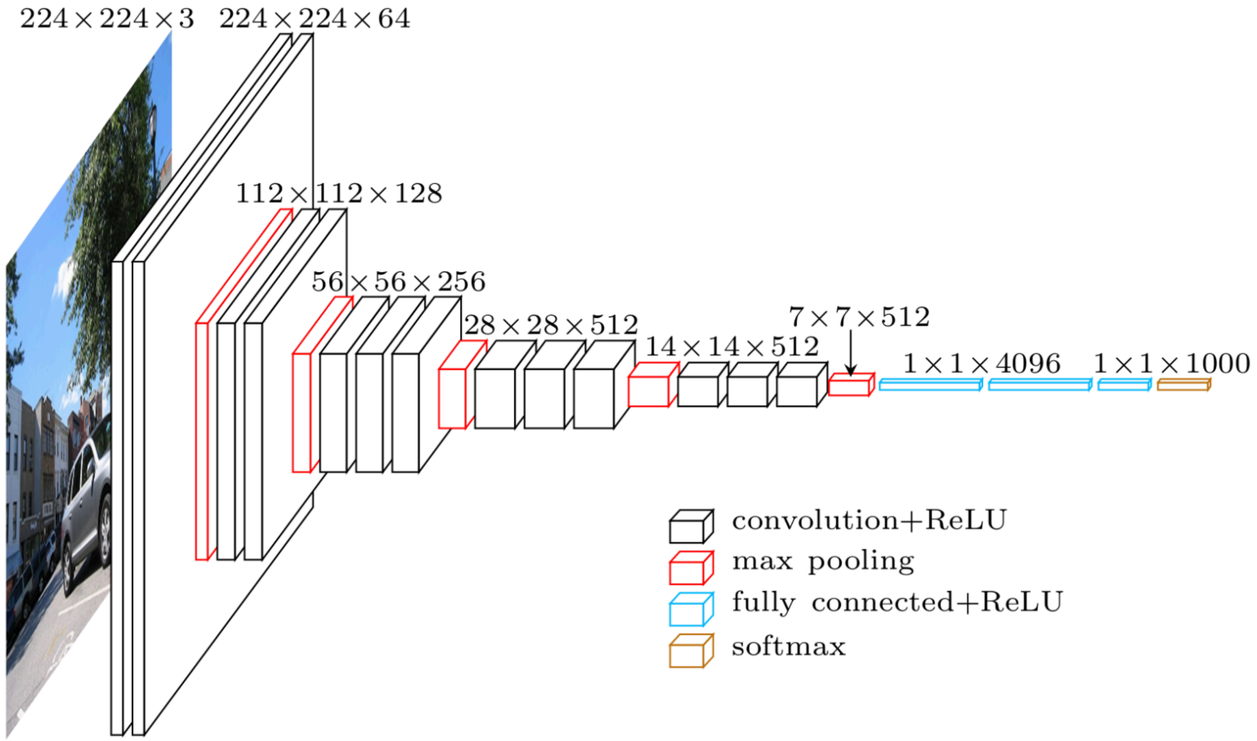
Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

AlexNet



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

VGG 16



Simonyan, Karen, and Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014)

VGG 16

```
            Activation maps          Parameters
INPUT:      [224x224x3]    = 150K    0
CONV3-64:   [224x224x64]   = 3.2M    (3x3x3)x64    =       1,728
CONV3-64:   [224x224x64]   = 3.2M    (3x3x64)x64   =      36,864
POOL2:      [112x112x64]   = 800K    0
CONV3-128:  [112x112x128]  = 1.6M    (3x3x64)x128  =      73,728
CONV3-128:  [112x112x128]  = 1.6M    (3x3x128)x128 =     147,456
POOL2:      [56x56x128]    = 400K    0
CONV3-256:  [56x56x256]    = 800K    (3x3x128)x256 =     294,912
CONV3-256:  [56x56x256]    = 800K    (3x3x256)x256 =     589,824
CONV3-256:  [56x56x256]    = 800K    (3x3x256)x256 =     589,824
POOL2:      [28x28x256]    = 200K    0
CONV3-512:  [28x28x512]    = 400K    (3x3x256)x512 =   1,179,648
CONV3-512:  [28x28x512]    = 400K    (3x3x512)x512 =   2,359,296
CONV3-512:  [28x28x512]    = 400K    (3x3x512)x512 =   2,359,296
POOL2:      [14x14x512]    = 100K    0
CONV3-512:  [14x14x512]    = 100K    (3x3x512)x512 =   2,359,296
CONV3-512:  [14x14x512]    = 100K    (3x3x512)x512 =   2,359,296
CONV3-512:  [14x14x512]    = 100K    (3x3x512)x512 =   2,359,296
POOL2:      [7x7x512]      =  25K    0
FC:         [1x1x4096]     = 4096    7x7x512x4096  = 102,760,448
FC:         [1x1x4096]     = 4096    4096x4096     =  16,777,216
FC:         [1x1x1000]     = 1000    4096x1000     =   4,096,000

TOTAL activations: 24M x 4 bytes ~=  93MB / image (x2 for backward)
TOTAL parameters: 138M x 4 bytes ~= 552MB (x2 for plain SGD, x4 for Adam)
```

YOLO

S × S grid on input

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." CVPR (2016)

YOLO

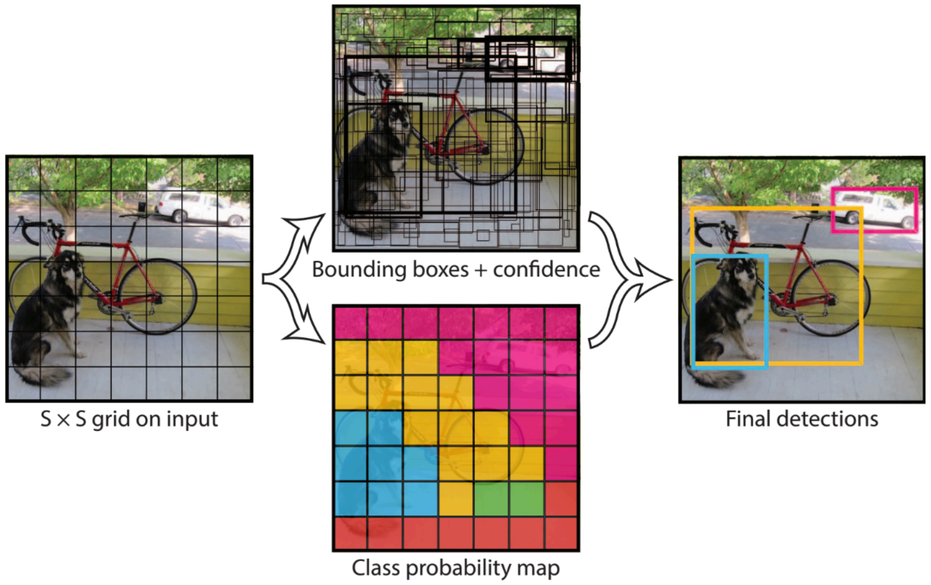
S × S grid on input

For each cell of the $S \times S$ predict:

- B **boxes** and **confidence scores** C ($5 \times B$ values) + **classes** c

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." CVPR (2016)

YOLO

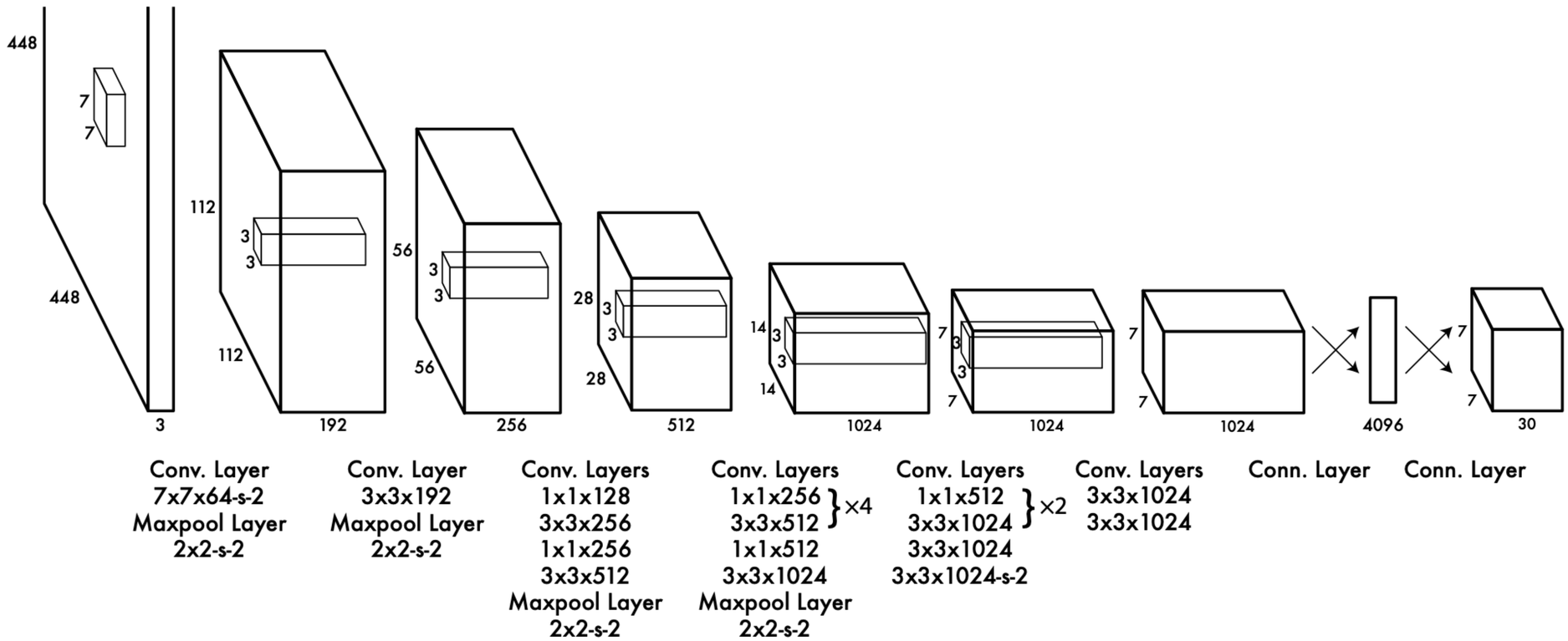


For each cell of the $S \times S$ predict:

- B **boxes** and **confidence scores** C ($5 \times B$ values) + **classes** c

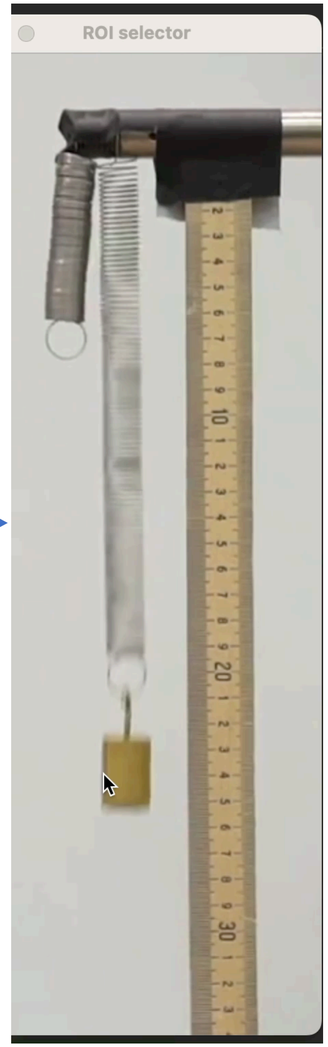
Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." CVPR (2016)

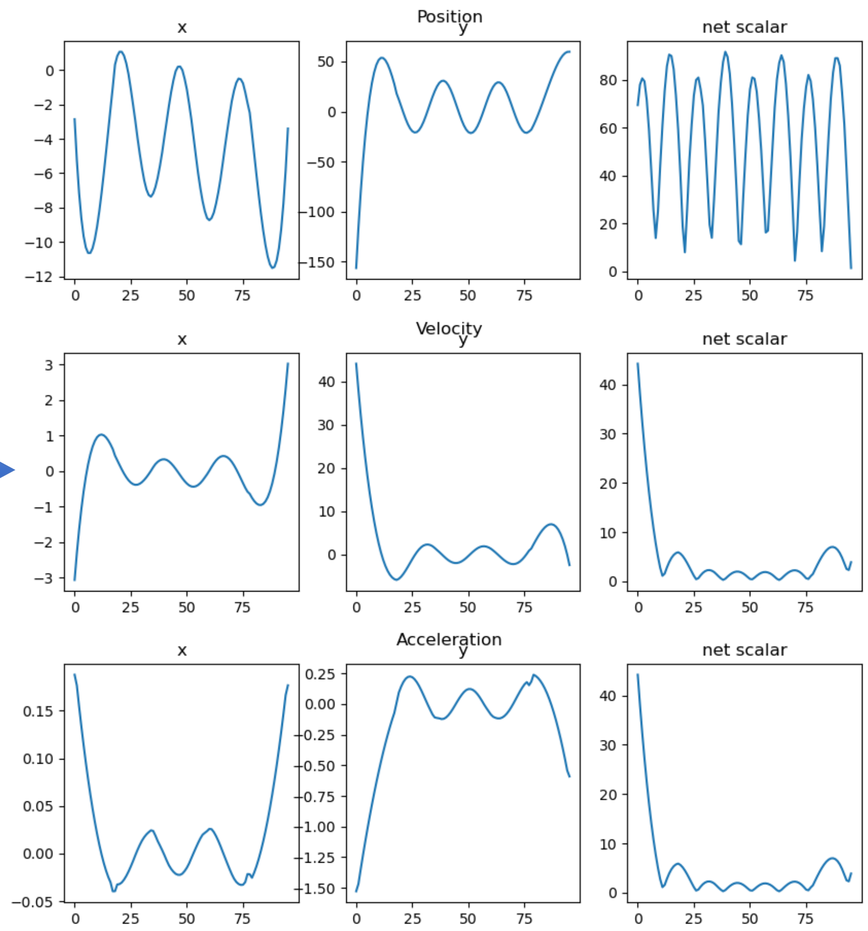
YOLO


448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2
Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x256
3x3x512
×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x512
3x3x1024
×2
3x3x1024
3x3x1024-s-2
Conv. Layers
3x3x1024
3x3x1024
Conn. Layer
Conn. Layer

Lab videos with YOLO

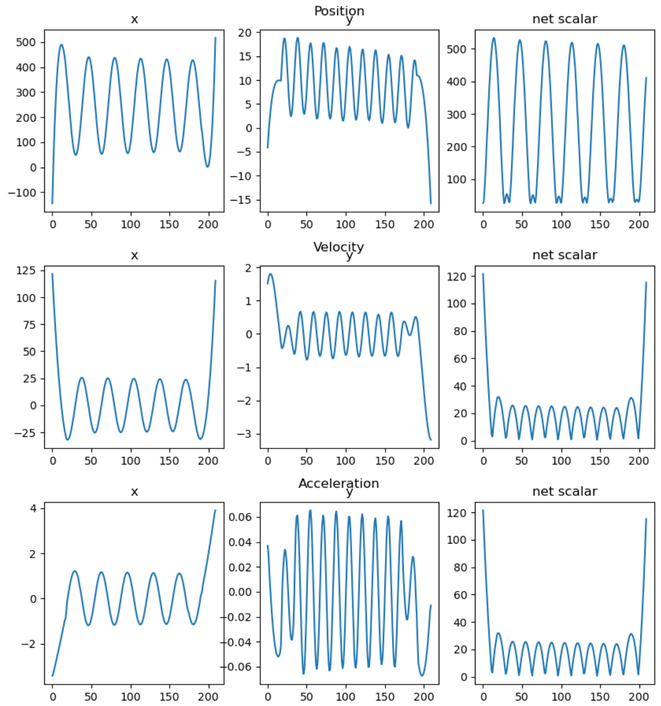
```
python 03_auto_tracking.py spring
spring
Found directory spring. Moving there.
[INFO] Read video file in spring
Detection mode:
1. Manual Region of Interest Selection
2.  Automatic using YOLO
(Default 1)1
[INFO] Detection mode 1
[INFO] Manual detection
Select a ROI and then press SPACE or ENTER button!
```





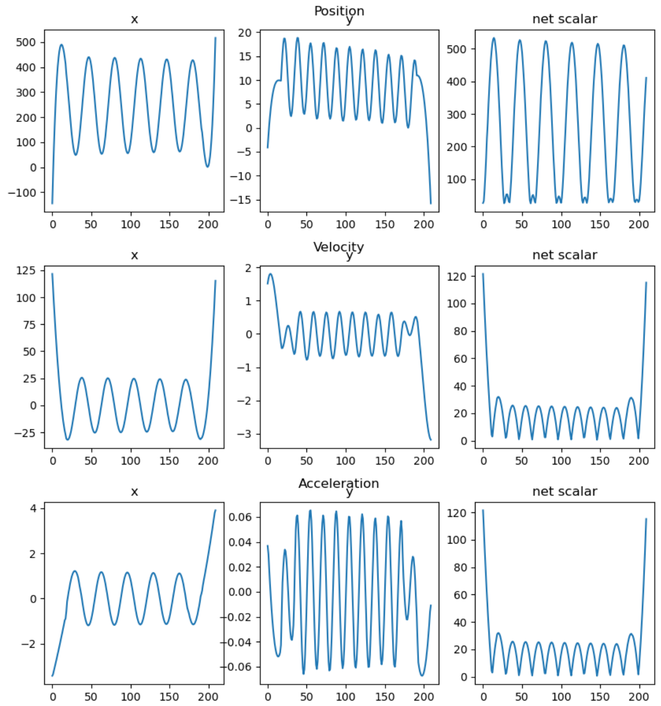
Lab videos with YOLO

Still hit-or-miss sometimes, but much better tracking than our initial implementation



Lab videos with YOLO

Still hit-or-miss sometimes, but much better tracking than our initial implementation



Using CNNs for research

Training a model on ImageNet from scratch takes **days or weeks**.

Using CNNs for research

Training a model on ImageNet from scratch takes **days or weeks**.

Many models trained on ImageNet and their weights are publicly available!

Using CNNs for research

Training a model on ImageNet from scratch takes **days or weeks**.

Many models trained on ImageNet and their weights are publicly available!

Transfer learning

- Use pre-trained weights, remove last layers to compute representations of images
- Train a classification model from these features on a new classification task
- The network is used as a generic feature extractor
- Better than handcrafted feature extraction on natural images

Using CNNs for research

Training a model on ImageNet from scratch takes **days or weeks**.

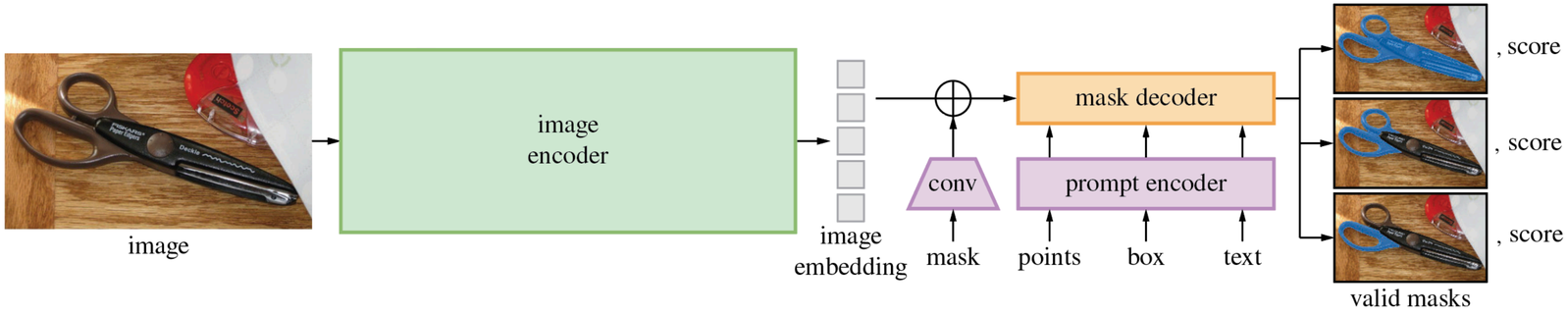
Many models trained on ImageNet and their weights are publicly available!

Fine-tuning

Retraining the (some) parameters of the network (given enough data)

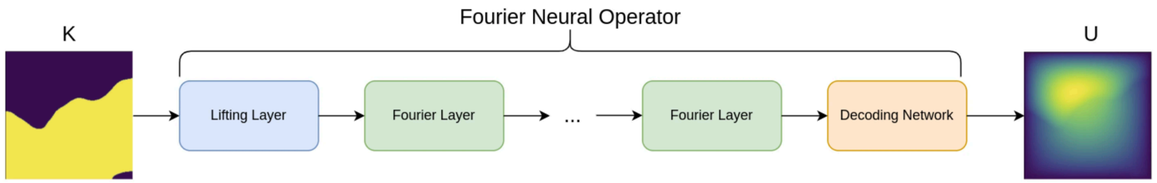
- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning

Advanced applications: Segment anything



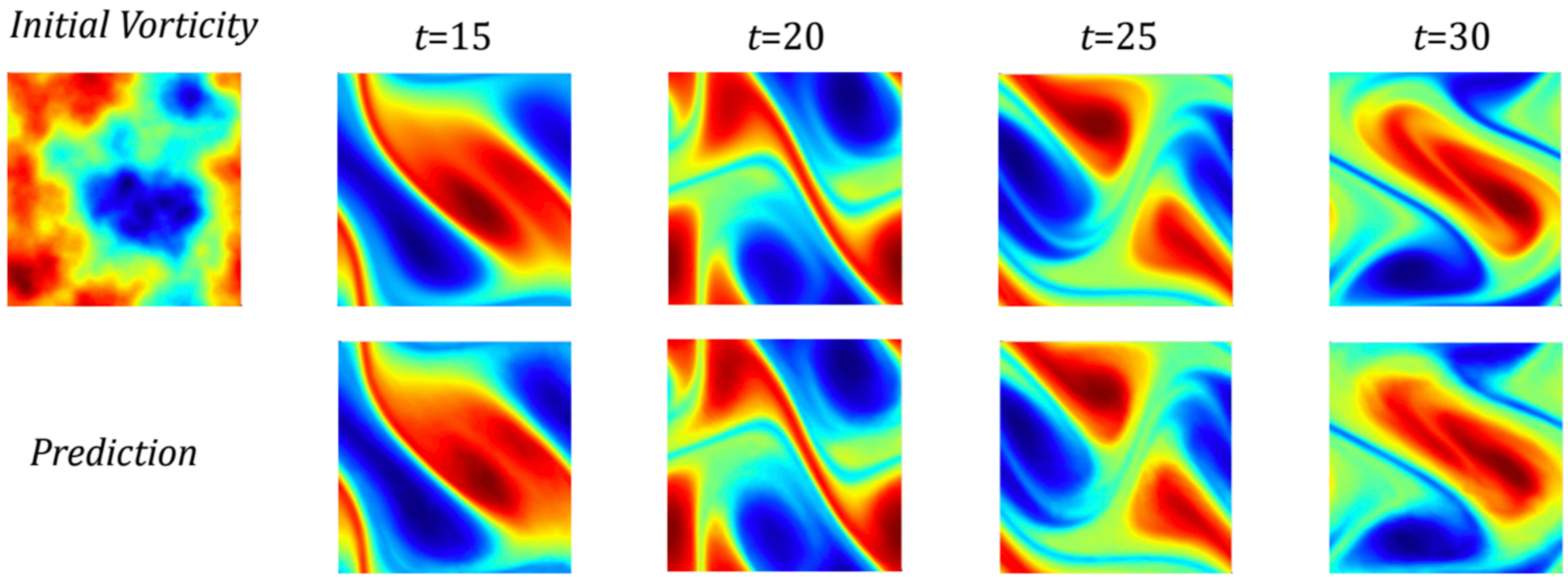
Segment Anything, Alexander et al, Meta AI (2023)

Physics-Informed Machine Learning



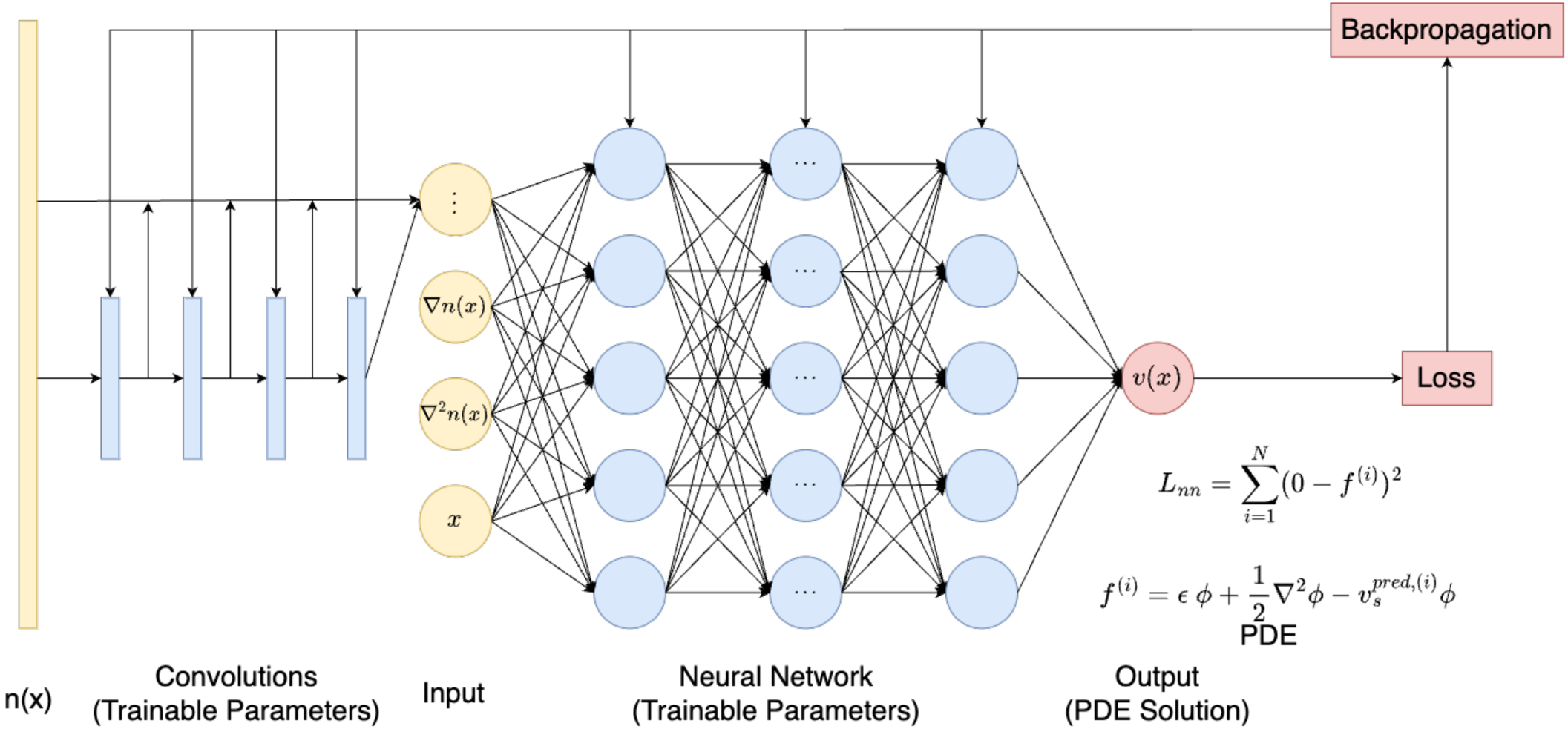
$$-\nabla \cdot (k(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}), \quad \mathbf{x} \in D,$$

Darcy flow prediction



FNO predictions for Navier Stokes propagation

Physics-Informed Neural-Network

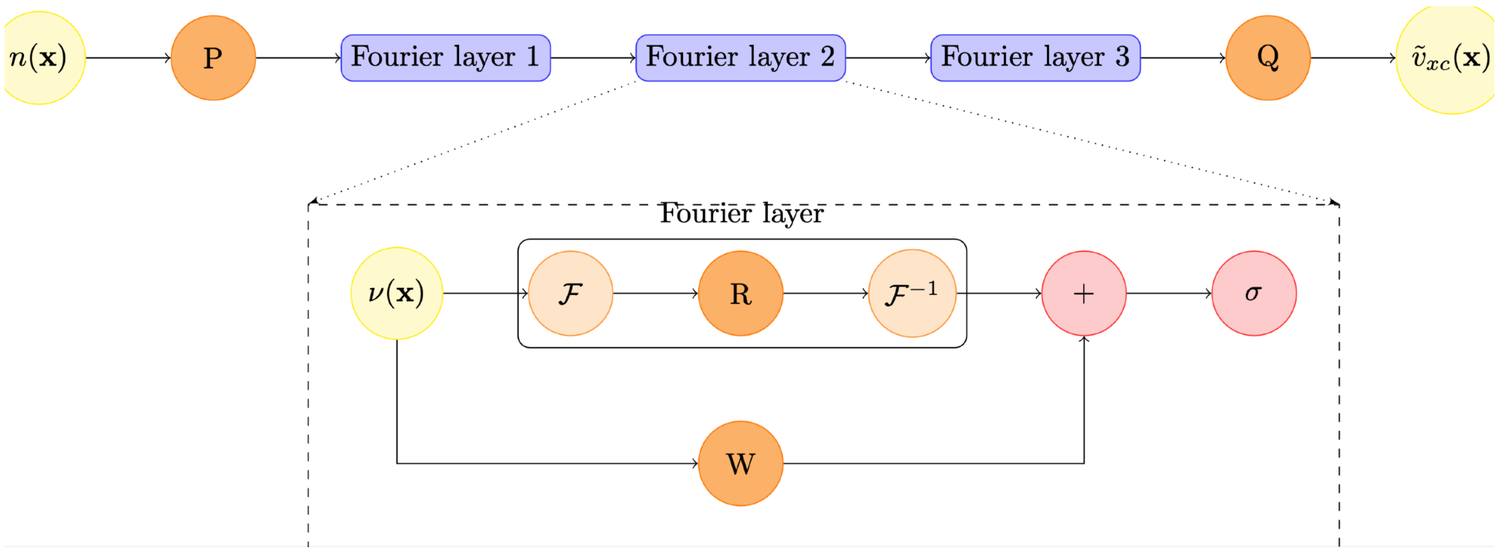


$$L_{nn} = \sum_{i=1}^{N} (0 - f^{(i)})^2$$

$$f^{(i)} = \epsilon\,\phi + \frac{1}{2}\nabla^2\phi - v_s^{pred,(i)}\phi$$

PDE

Fourier Neural Operators



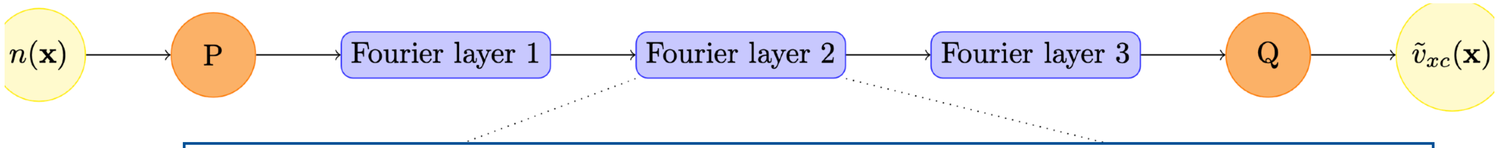
FNO Architecture

Each layer is defined as: $v_{l+1}(x) = \sigma_{(l+1)}\left(W_l v_l(x) + \left(\mathcal{K}_l(a;\lambda)v_l\right)(x)\right)$

For FNOs, the kernel function is: $\left(\mathcal{K}(\phi)v_t\right)(x) = \mathcal{F}^{-1}\left(R_\phi \cdot \left(\mathcal{F}v_t\right)\right)(x)$

With loss $L_{FNO} = MSE(v_{\mathrm{XC}}, \tilde{v}_{\mathrm{XC}}) = \frac{1}{N}\sum_{i=1}^{N}\left|v_{\mathrm{XC}}^{(i)} - \tilde{v}_{\mathrm{XC}}^{(i)}\right|^2$

Fourier Neural Operators



APS March Meeting 2024

Monday–Friday, March 4–8, 2024; Minneapolis & Virtual

Session G18: Machine Learning for Materials Science II
11:30 AM–2:30 PM, Tuesday, March 5, 2024
Room: M100I

Sponsoring Unit: GDS
Chair: Antonia Statt, University of Illinois at Urbana-Champaign

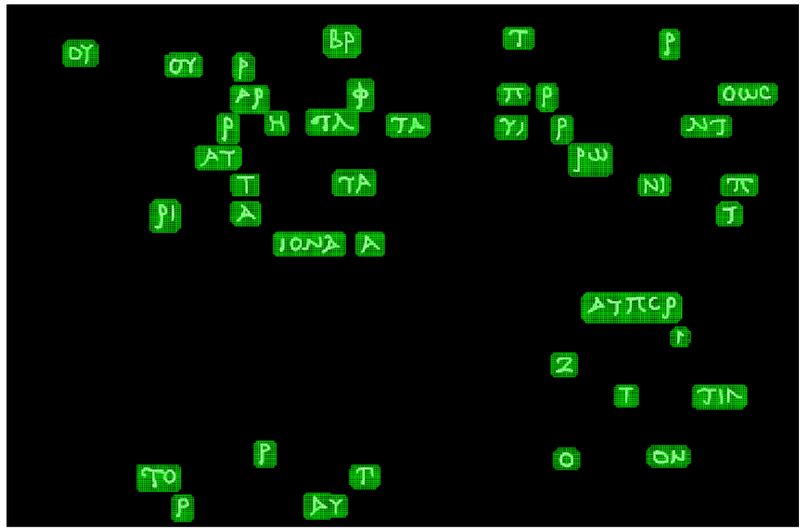
Abstract: G18.00010 : Physics-Informed Machine Learning for Addressing Challenges in Static and Time-Dependent Density Functional Theory
2:06 PM–2:18 PM

Each layer is definec

For FNOs, the kerne

With loss $L_{FNO} = MSE(v_{\mathrm{XC}}, \tilde{v}_{\mathrm{XC}}) = \frac{1}{N}\sum_{i=1}^{N}\left|v_{\mathrm{XC}}^{(i)} - \tilde{v}_{\mathrm{XC}}^{(i)}\right|^2$

Extra: Scroll Prize



Vesuvius challenge: https://scrollprize.org/grandprize

Summary

- We went from linear regression to machine learning
- Multi-layer (deep) neural networks
- Convolutional neural networks
- How you can apply it to your research
- Example applications

Acknowledgements

- APS GDS Leadership, especially William and Julie
- My supervisor Attila Cangi, and my group at CASUS

- Data Science course by Olivier Grisel and Charles Ollion: https://github.com/m2dsupsdlclass/lectures-labs
- Understanding Deep Learning book by Simon J.D. Prince: https://udlbook.github.io/udlbook/

Thank You

Email: k.shah@hzdr.de

Optimizers

- SGD (with Nesterov momentum)
  - Simple to implement
  - Very sensitive to initial value of η
  - Need learning rate scheduling
- Adam: adaptive learning rate scale for each param
  - Global η set to 3e-4 often works well enough
  - Good default choice of optimizer (often)