

# Knowledge-guided Data Science

Shen Liang  
Data Intelligence Institute of Paris (diiP)

diiP Summer School (dSDS 2024)

# Outline

- Motivation for Knowledge-guided Data Science
- Knowledge Sources and Representations
- Fusing Knowledge with Data
- Conclusion

# Outline

- Motivation for Knowledge-guided Data Science
- Knowledge Sources and Representations
- Fusing Knowledge with Data
- Conclusion

# Data-driven Methods

- Data science, as the name suggests, is **primarily driven by data**, that is, it **directly learns from raw or transformed data**.
- Machine Learning (ML) in a nutshell: Learn a mapping  $f$  from pre-defined **input  $x$**  to **output (as known as *label*)  $y$** . That is:  $y = f(x)$

Task	Input $x$	Output $y$
Industrial fault detection	Vibration signal of a machine	Either “normal” or <b>specific type of fault</b>
Power consumption prediction	Power consumption of the last 30 days ( <b>one value per day</b> )	Power consumption of the next 30 days ( <b>one value per day</b> )
Medical image segmentation	Original medical image	Labels (e.g. <b>is_part_of_tumor</b> , <b>not_part_of_tumor</b> ) for each pixel
Question answering	Question text	Answer text
...	...	...

# Data-driven Methods

- Data science, as the name suggests, is **primarily driven by data**, that is, it **directly learns from raw or transformed data**.
- Machine Learning (ML) in a nutshell: Learn a mapping  $f$  from pre-defined **input  $x$**  to **output (as known as *label*)  $y$** . That is:  $y = f(x)$
- The classic data-driven ML pipeline:
  - **Training** the ML model: feed it with **a training set of known  $(x, y)$  pairs**, so that it **learns a mapping  $f$**  such that for each training example  $x$ , the **difference between  $f(x)$  and  $y$**  is minimized.
  - **Testing / Predicting** with the ML model: feed  $f$  with **an example  $x'$  that was not in the training set**, and **let it predict its corresponding  $y'$** .

# Data-driven Methods

- Data science, as the name suggests, is **primarily driven by data**, that is, it **directly learns from raw or transformed data**.
- Machine Learning (ML) in a nutshell: Learn **a mapping  $f$**  from pre-defined **input  $x$**  to **output  $y$** .
- The classic data-driven ML pipeline:
  - **Training** the ML model: feed it with **a training set of known  $(x, y)$  pairs**, so that it **learns a mapping  $f$**  such that for each training example  $x$ , the **difference between  $f(x)$  and  $y$**  is minimized.
  - **Testing / Predicting** with the ML model: feed  $f$  with **an example  $x'$  that was not in the training set**, and **let it predict its corresponding  $y'$** .
- The ML model **only learns from the training examples**, **without drawing on pre-existing domain knowledge**.

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimãtasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afi]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Note: What will be discussed should only be considered a toy example. **This is *not* how a real text-to-speech system works!**

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input $x$	Output $y$
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input $x$	Output $y$
Natation	

Let's do this in a **purely data-driven (learn-by-example)** manner. That is, we solely learn from examples in the training set.



# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
<b><u>N</u>ation</b>	<b>[nasjɔ̃]</b>
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
<b><u>N</u>ation</b>	<b>[na</b>

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Aliment <u>ation</u>	[alimɔ̃ <u>ta</u> sjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Nat <u>ation</u>	[nat <u>a</u> ]

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Na <u>tion</u>	[na <u>sjɔ̃</u> ]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimenta <u>tion</u>	[alimãta <u>sjɔ̃</u> ]
Papa	[papa]
Simula <u>tion</u>	[simyla <u>sjɔ̃</u> ]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Nata <u>tion</u>	[nata <u>sjɔ̃</u> ]

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input $x$	Output $y$
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input $x$	Output $y$
Natation	[natasjɔ̃]

So far, the purely data-driven approach has served us well!

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbjɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	

Let's see if the purely data-driven approach works too on this one.

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
<b><u>Papa</u></b>	<b>[papa]</b>
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
<b><u>Pas</u></b>	<b>[pa]</b>

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbjɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
<b>S</b> imulation	[ <b>s</b> imylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pa <b>s</b>	[pa

Ah oh, how to pronounce **s** when it is at the end of a word?

Do we pronounce it as [**s**] as in **S**imulation?

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbjɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
<b>S</b> imulation	[ <b>s</b> imylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pa <b>s</b>	[pa

Ah oh, how to pronounce **s** when it is at the end of a word?

Do we pronounce it as [**s**] as in **S**imulation?

**(News flash: WRONG!)**



# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pa <u>s</u>	[pa

Ah oh, how to pronounce s when it is at the end of a word?

- Data-driven methods tend **not to work well on small training sets**, where the **training examples cannot sufficiently represent all cases!**
- Don't worry: **domain knowledge** is here to help!

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbjɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pa <u>s</u>	[pa

We now switch to a **domain knowledge-driven** approach, utilizing the following pre-existing knowledge:

*In French, except for q, c, l, r and f, the other consonant letters are usually not pronounced at the end of a word.*

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pa <u>s</u>	[pa]

We now switch to a **domain knowledge-driven** approach, utilizing the following pre-existing knowledge:

*In French, except for q, c, l, r and f, the other consonant letters are usually not pronounced at the end of a word.*

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]

**Knowledge-driven methods can excel** in the face of **small training data!**

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
R <u>o</u> yal	[r <u>w</u> ajal]
<u>o</u> bjet	[ɔ̃bʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
M <u>o</u> yen	[m <u>w</u> ajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]

Another advantage of knowledge-driven methods is their **high explainability**, that is, their ability to explain **why** they make a certain prediction (rather than others) for a testing example.

E.g. In *royal* and *moyen*, why is o seemingly pronounced as [wa], rather than [ɔ̃] as in *objet*?

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
R <u>o</u> yal	[r <u>wa</u> jal]
Objet	[ɔ̃bʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
M <u>o</u> yen	[m <u>wa</u> jɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]

In *royal* and *moyen*, why is o seemingly pronounced as [wa], rather than [ɔ̃] as in *objet*?

Domain knowledge: When y is between *two vowels*, it should be treated as i+i in pronunciation.

- *Royal* = *roi+ial*; *Moyen* = *moi+ien*

So it is not o that is pronounced as [wa], but the implicit *oi* !

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]

The knowledge-driven method is so cool!  
Why bother using a data-driven one?

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbjɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]
Maïs	

Well, let's see how a knowledge-driven method fairs with this one...



# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input $x$	Output $y$
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[nɛif]
Laique	[lɛik]

Testing examples	
Input $x$	Output $y$
Natation	[natasjɔ̃]
Pas	[pa]
Maïs	[mai]

To begin with, we currently do not have knowledge on how to pronounce *maï*, so we need to use a data-driven approach to bypass this.

But this is not the main problem with a knowledge-driven approach. After all, we can always add additional domain knowledge to tackle this.

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]
Maï <u>s</u>	[mai

The real problem with a knowledge-driven approach is: How to pronounce the s?

If we draw on our previous knowledge, *except for q, c, l, r and f, the other consonant letters are usually not pronounced at the end of a word*

we should make it silent, **which is WRONG!**

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbjɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]
Maï <u>s</u>	[maï <u>s</u> ]

If we draw on our previous knowledge, *except for q, c, l, r and f, the other consonant letters are usually not pronounced at the end of a word*

we should make it silent, **which is WRONG!**

This is a **special case** where the **s** at the end of the word is not silent!

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbʒɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]
Maï <u>s</u>	[mai <u>s</u> ]

Domain knowledge can often be **(over)-simplified summarizations** of **complex real-world cases**, which **inevitably leads to a degree of error**.

*Except for q, c, l, r and f, the other consonant letters are **usually (not always!)** not pronounced at the end of a word.*

# Data-driven vs Knowledge-driven: A Toy Example

- Suppose we are trying to train a model that transforms **written French words** into their **pronunciations**.

Training set	
Input x	Output y
Ticket	[tikɛ]
Nation	[nasjɔ̃]
Royal	[rwajal]
Objet	[ɔbjɛ]
Alimentation	[alimɑ̃tasjɔ̃]
Papa	[papa]
Simulation	[simylasjɔ̃]
Affiche	[afiʃ]
Moyen	[mwajɛ̃]
Naïf	[naif]
Laïque	[laik]

Testing examples	
Input x	Output y
Natation	[natasjɔ̃]
Pas	[pa]
Maïs	[mais]

Domain knowledge can often be **(over)-simplified summarizations** of **complex real-world cases**, which **inevitably leads to a degree of error**.

In such cases, it is usually better to **draw on a data-driven approach** on **a larger training set**.

# Data-driven vs Knowledge-driven: A Summary

	Data-driven Methods	Knowledge-driven Methods
General methodology	Learn from raw training examples	Draw on pre-existing domain knowledge
Robustness against small data	Low	High
Explainability	Low	High
Adaptivity to complex real-world cases	High	Low

- Data-driven and knowledge-driven methods nicely complement each other. Thus, it is best that we fuse them together by integrating knowledge into a data-driven method, hence knowledge-guided data science.
  - Synonyms of *knowledge-guided*: informed, knowledge-informed, physics-informed, physics-guided, mechanism-guided, ...

# Outline

- Motivation for Knowledge-guided Data Science
- Knowledge Sources and Representations
- Fusing Knowledge with Data
- Conclusion

# Sources of Knowledge

- Natural sciences: universal laws of physics, bio-molecular descriptions of genetic sequences, material-forming production processes...
- Social Sciences: effects in social networks, the syntax and semantics of language...
- Expert Knowledge: working experience of an expert...
- World Knowledge: common sense



# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
  - E.g. Mass-energy equivalence

$$E = m \cdot c^2$$

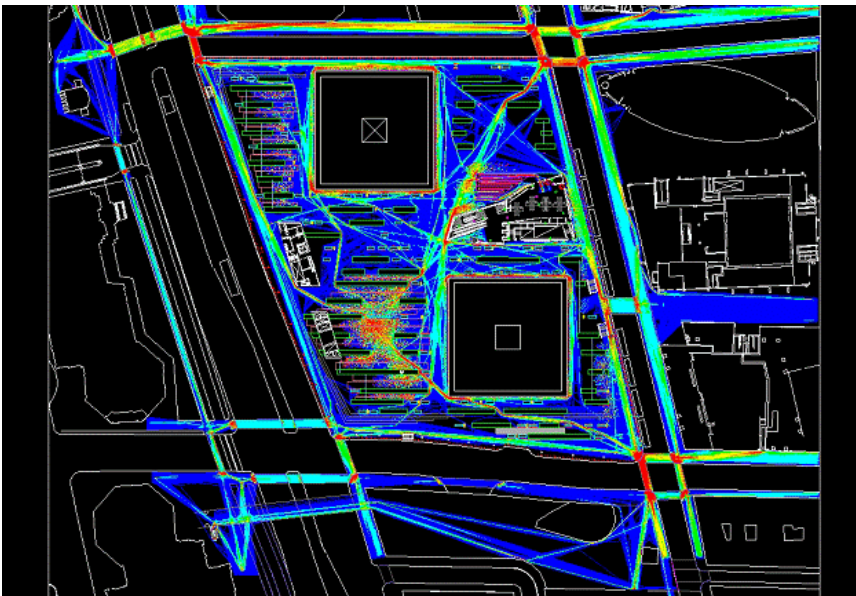
$$v \leq c$$

# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
- Logic rules: mainly from world knowledge, can also come from natural and social sciences
  - E.g. **if** I am **not** taller than 185cm, **and** you are as tall as **or** shorter than I am, **then** you are **not** taller than 185cm.

# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
- Logic rules: mainly from world knowledge, can also come from natural and social sciences
- Simulation results: mainly from natural sciences



Simulated Pedestrian Traffic

*Credit: Louis Berger Group*

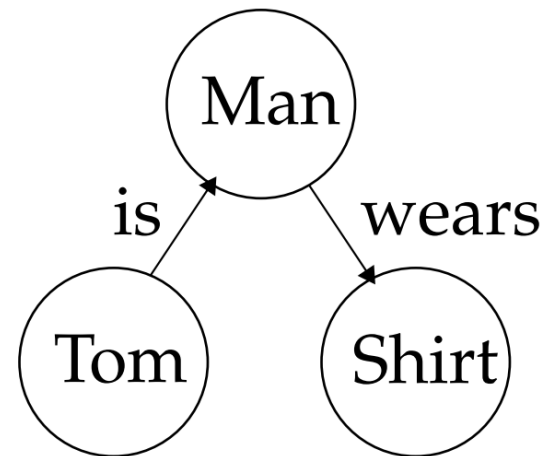
# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
- Logic rules: mainly from world knowledge, can also come from natural and social sciences
- Simulation results: mainly from natural sciences
- Differential equations: mainly from natural sciences
  - E.g. Burgers' equation, useful in fluid mechanics, nonlinear acoustics, gas dynamics, traffic flow, etc.

$$\begin{aligned} \partial_t u + u \partial_x u - (0.01/\pi) \partial_{xx} u &= 0, & (t, x) &\in (0, 1] \times (-1, 1), \\ u(0, x) &= -\sin(\pi x), & x &\in [-1, 1], \\ u(t, -1) = u(t, 1) &= 0, & t &\in (0, 1]. \end{aligned}$$

# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
- Logic rules: mainly from world knowledge, can also come from natural and social sciences
- Simulation results: mainly from natural sciences
- Differential equations: mainly from natural sciences
- Knowledge graphs: mainly from world knowledge, can also come from natural and social sciences
  - Knowledge graphs can effectively encode knowledge in the form of (**subject**, *predicate*, object) triples. E.g.
    - **Tom** *is* a man.
    - A **man** *wears* a shirt.

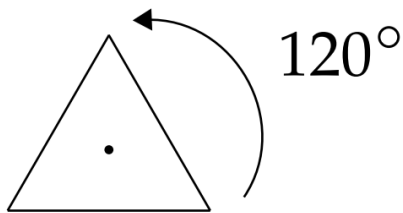


# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
- Logic rules: mainly from world knowledge, can also come from natural and social sciences
- Simulation results: mainly from natural sciences
- Differential equations: mainly from natural sciences
- Knowledge graphs: mainly from world knowledge, can also come from natural and social sciences
- Probabilistic relations: mainly from expert knowledge, can also come from natural science and world knowledge
  - If you experience shifting difficulties when driving your car, you **may** have a problem with your gearbox.

# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
- Logic rules: mainly from world knowledge, can also come from natural and social sciences
- Simulation results: mainly from natural sciences
- Differential equations: mainly from natural sciences
- Knowledge graphs: mainly from world knowledge, can also come from natural and social sciences
- Probabilistic relations: mainly from expert knowledge, can also come from natural science and world knowledge
- Invariances: mainly from natural science, can also come from world knowledge



E.g. If you rotate an equilateral triangle around its center by an angle of 120 degrees, it will look exactly the same.

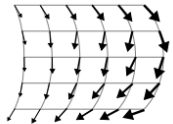
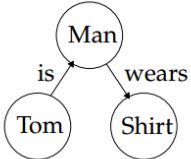
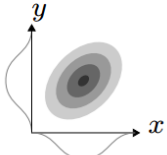
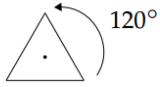

# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
- Logic rules: mainly from world knowledge, can also come from natural and social sciences
- Simulation results: mainly from natural sciences
- Differential equations: mainly from natural sciences
- Knowledge graphs: mainly from world knowledge, can also come from natural and social sciences
- Probabilistic relations: mainly from expert knowledge, can also come from natural science and world knowledge
- Invariances: mainly from natural science, can also come from world knowledge
- Human feedback: mainly from expert knowledge, can also come from world knowledge
  - - AI: How do you find the email I generated for you?
  - Human: It's too informal. To start with, you should begin with *Dear Sir*, not *Hi Bob*.



# Representations of Knowledge

- Algebraic equations: mainly from natural science, can also come from expert knowledge
- Logic rules: mainly from world knowledge, can also come from natural and social sciences
- Simulation results: mainly from natural sciences
- Differential equations: mainly from natural sciences
- Knowledge graphs: mainly from world knowledge, can also come from natural and social sciences
- Probabilistic relations: mainly from expert knowledge, can also come from natural science and world knowledge
- Invariances: mainly from natural science, can also come from world knowledge
- Human feedback: mainly from expert knowledge, can also come from world knowledge

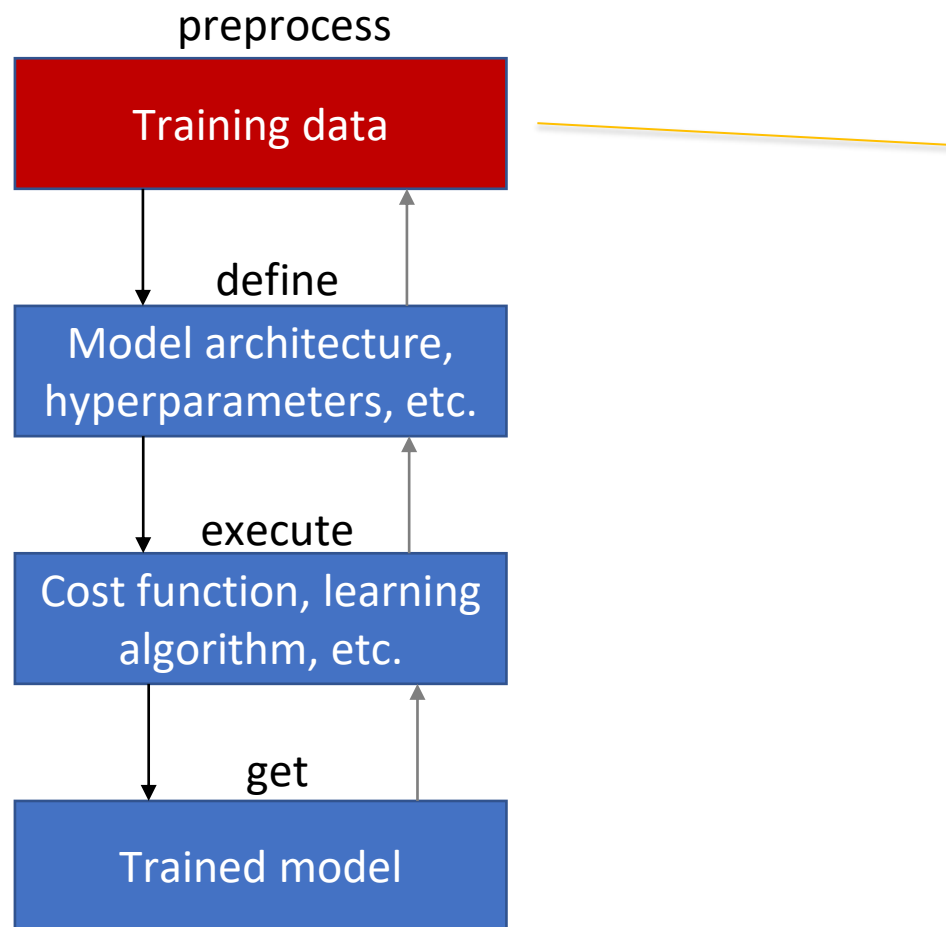
Algebraic Equations	Logic Rules	Simulation Results	Differential Equations	Knowledge Graphs	Probabilistic Relations	Invariances	Human Feedback
$E = m \cdot c^2$ $v \leq c$	$A \wedge B \Rightarrow C$		$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$ $F(x) = m \frac{d^2 x}{dt^2}$				

# Outline

- Motivation for Knowledge-guided Data Science
- Knowledge Sources and Representations
- **Fusing Knowledge with Data**
- Conclusion

# Data-driven Model Training Pipeline

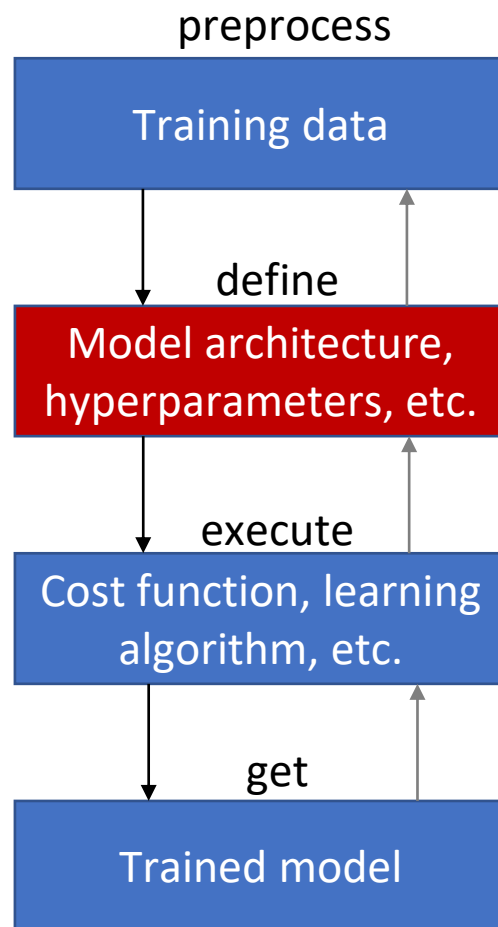
## Data-driven training pipeline for an ML model



- Prepare the training data-label pairs  $(x, y)$  examples.
- The input  $x$  can both be the **original data**, or **features** extracted from the original data.
  - E.g. If the original data is **text**, you can extract **the number of occurrences of each word** as a feature.

# Data-driven Model Training Pipeline

## Data-driven training pipeline for an ML model



- Define the model you want to train.

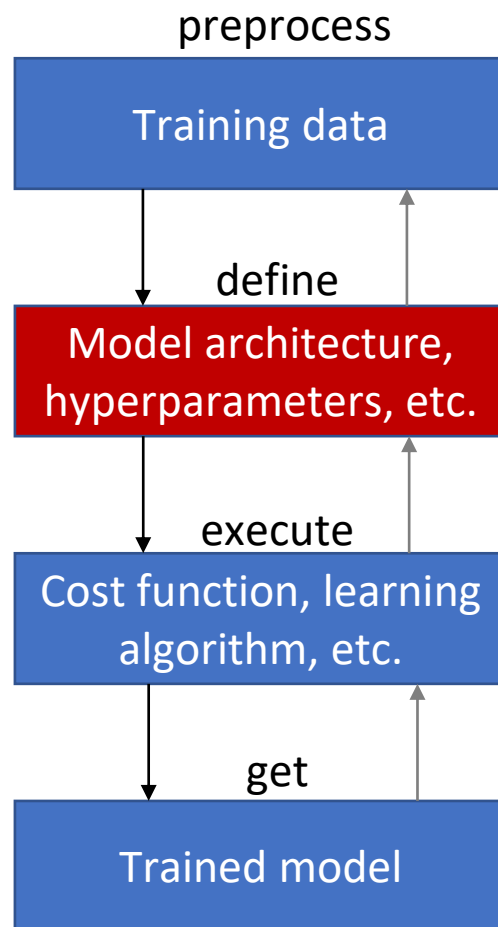
### 1. Model architecture:

- **Shallow models:** linear regression, logistic regression, support vector machines, etc.
- **Deep models:** convolutional, recurrent, transformers ...

(Note: Deep models can automatically extract features from the input data, which overlays with the previous step. However, *in the previous step, the features are manually defined*; here the features are *automatically learned*.)

# Data-driven Model Training Pipeline

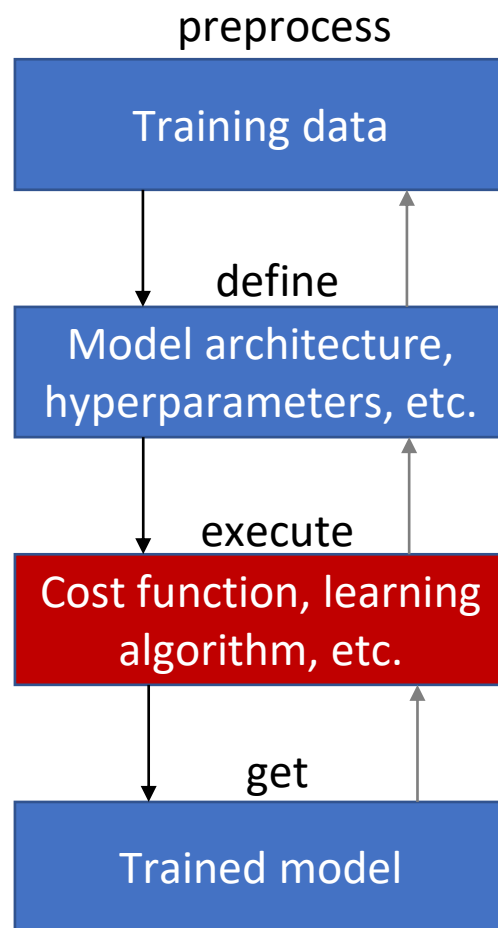
## Data-driven training pipeline for an ML model



- Define the model you want to train.
1. Model architecture:
    - Shallow models: linear regression, logistic regression, support vector machines, etc.
    - Deep models: convolutional, recurrent, transformers ...
  2. Hyperparameters
    - E.g. number of layers in a deep neural network, number of neurons in each layer, etc.
    - **Hyperparameters** vs **Parameters**
      - **Hyperparameters:** pre-set manually, does not change in the training process
      - **Parameters:** variables that are optimized during the training process (in fact, training an ML model is essentially optimizing its parameters so that it aligns with the training data).

# Data-driven Model Training Pipeline

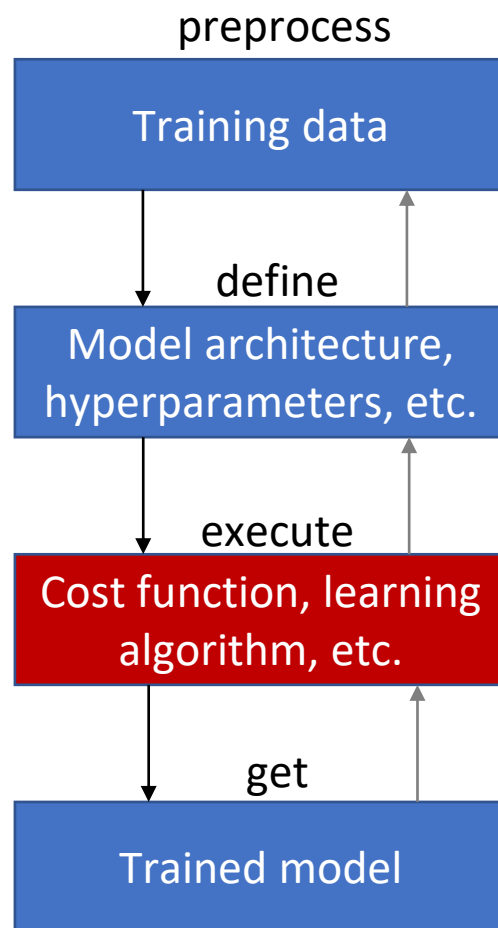
## Data-driven training pipeline for an ML model



- Decide how you want to train the defined model
- 1. **Cost/Lost function:** quantifies how good your model is; you get to decide what *good* means:
  - **Basic:** the eventual prediction result should be as accurate as possible;
  - the prediction criterion (decision boundary) should be as simple as possible;
  - certain important intermediate outputs should be as accurate as possible;
  - ...

# Data-driven Model Training Pipeline

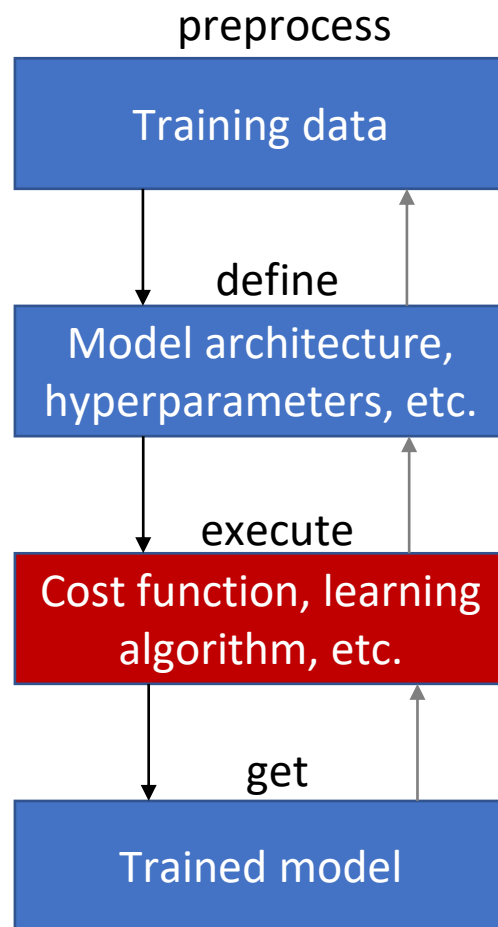
## Data-driven training pipeline for an ML model



- Decide how you want to train the defined model
- 1. **Cost/Lost function:** quantifies how good your model is; you get to decide what *good* means:
  - **Basic:** the eventual prediction result should be as accurate as possible;
  - the prediction criterion (decision boundary) should be as simple as possible;
  - certain important intermediate outputs should be as accurate as possible;
  - ...

# Data-driven Model Training Pipeline

## Data-driven training pipeline for an ML model

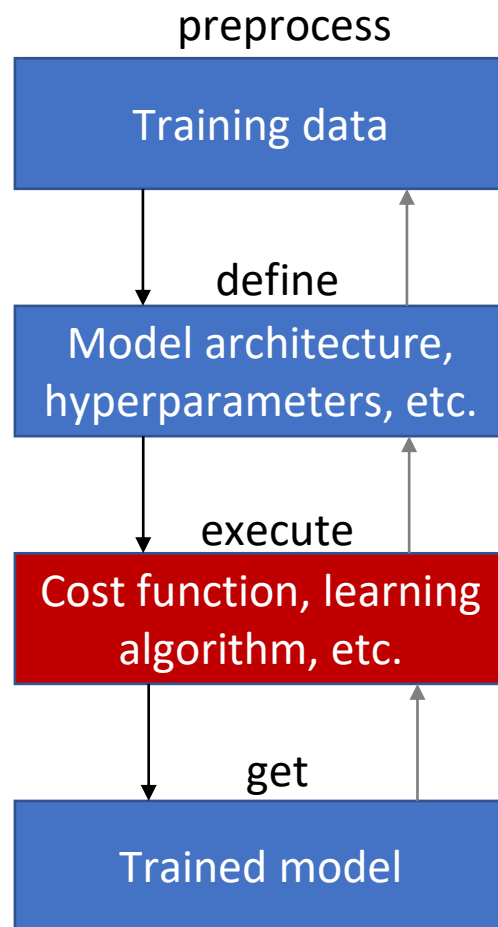


- Decide how you want to train the defined model
  1. **Cost/Lost function**: quantifies **how good** your model is; you get to decide **what good means**:
    - If there are multiple ways to define *good*, you can have **multiple terms in your cost function**. In this case, the cost function is the **(weighted) sum of all these terms** (and you get to decide the weight of each term yourself).



# Data-driven Model Training Pipeline

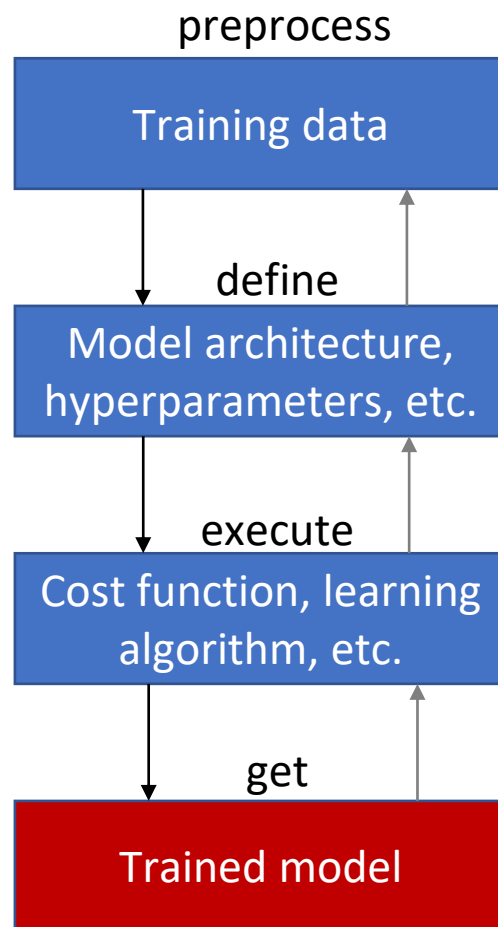
## Data-driven training pipeline for an ML model



- Decide how you want to train the defined model
  1. **Cost/Lost function**: quantifies **how good** your model is; you get to decide **what good means**.
  2. **Learning algorithm**: **the means by which the model becomes good** (with **good** defined by the cost function).

# Data-driven Model Training Pipeline

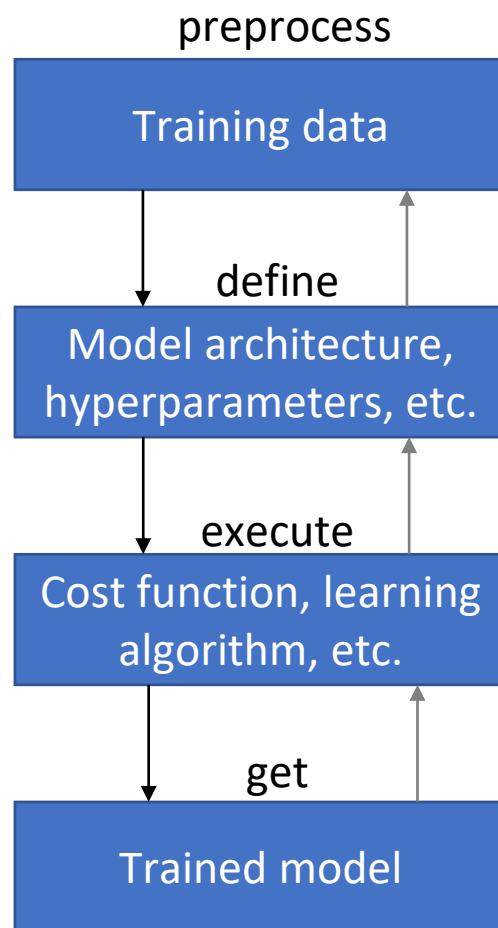
Data-driven training pipeline for an ML model



While the training can be done automatically, you can always **evaluate and refine** the trained model **after the initial training is complete** !

# Integrating Knowledge into a Data-driven pipeline

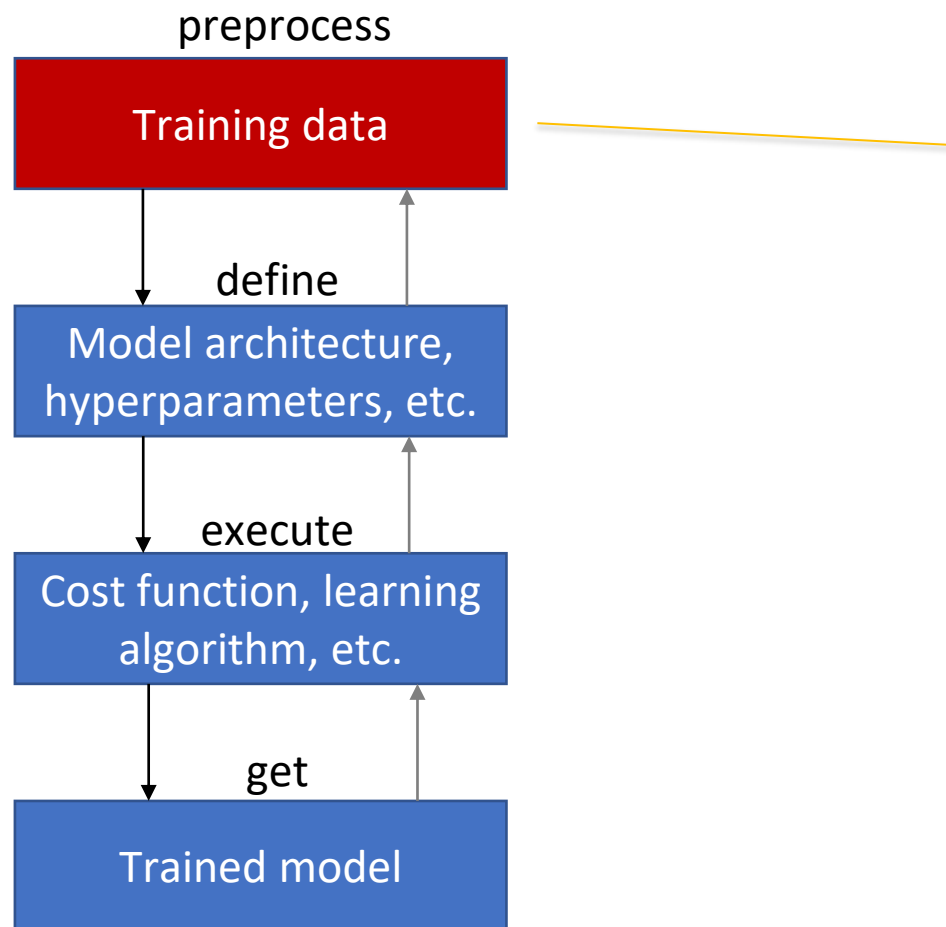
## Data-driven training pipeline for an ML model



We can integrate domain knowledge into **any step** of the training process of a data-driven machine learning model!

# Integrating Knowledge into the training data

## Data-driven training pipeline for an ML model

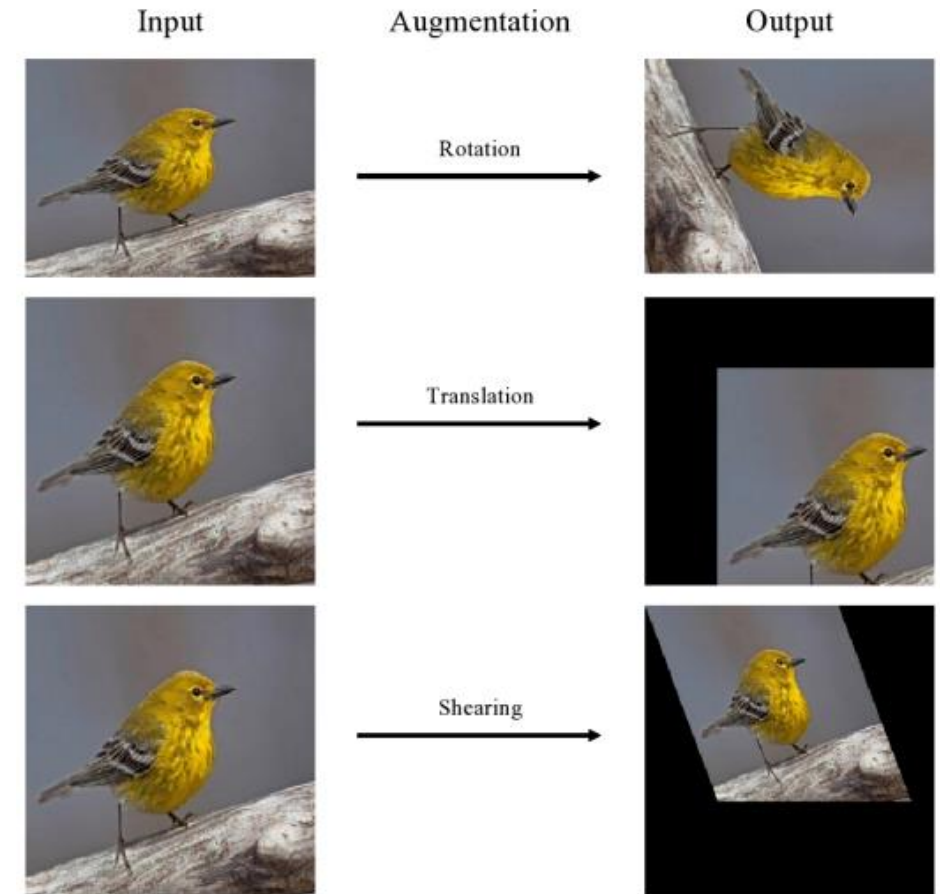


- Prepare the training data-label pairs  $(x, y)$  examples.
- The input  $x$  can both be the **original data**, or **features** extracted from the original data.

**You can enrich/refine the dataset with prior knowledge.**

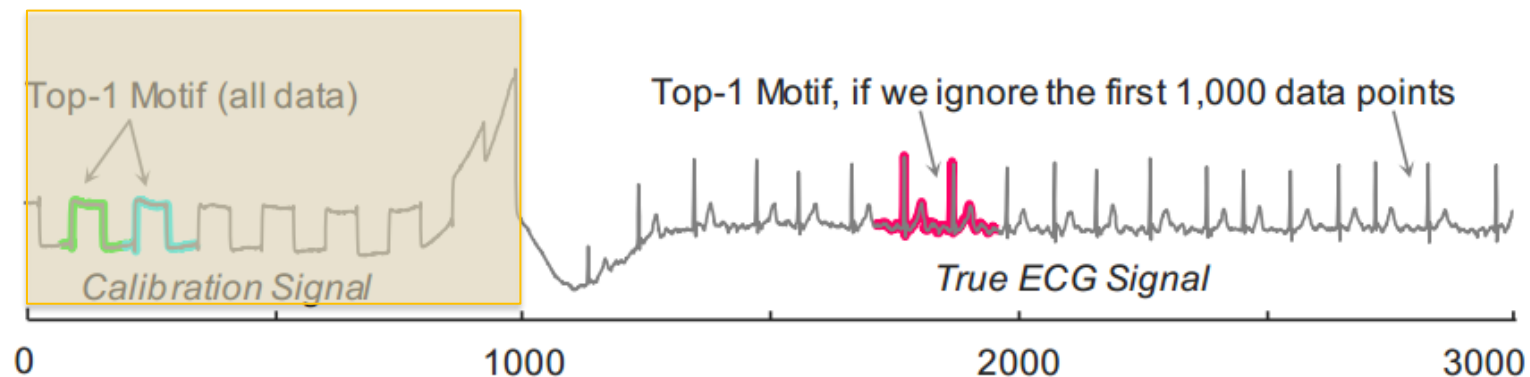
# Generating New Training Data (Data Augmentation)

- Case: Image classification
  - Input  $x$ : an image
  - Output  $y$ : a label describing the image (e.g. bird)
  - Note: many data augmentation methods are not specific to image classification; they can also be used for object detection, image segmentation, etc.
- Challenge: requires large numbers of labeled images to train the model; too laborious for humans
- Solution: use **common sense knowledge** to **devise transformation methods** that **generate new images carrying the same labels as the original ones**; add the newly generated images.
  - E.g. **Geometric transformations** tend not to alter the labels of the images.



# Filtering Bad Data

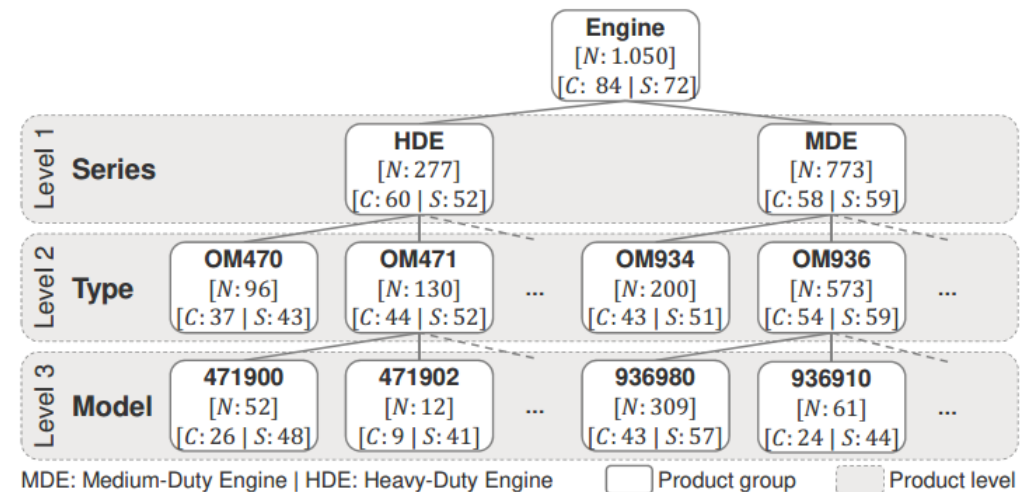
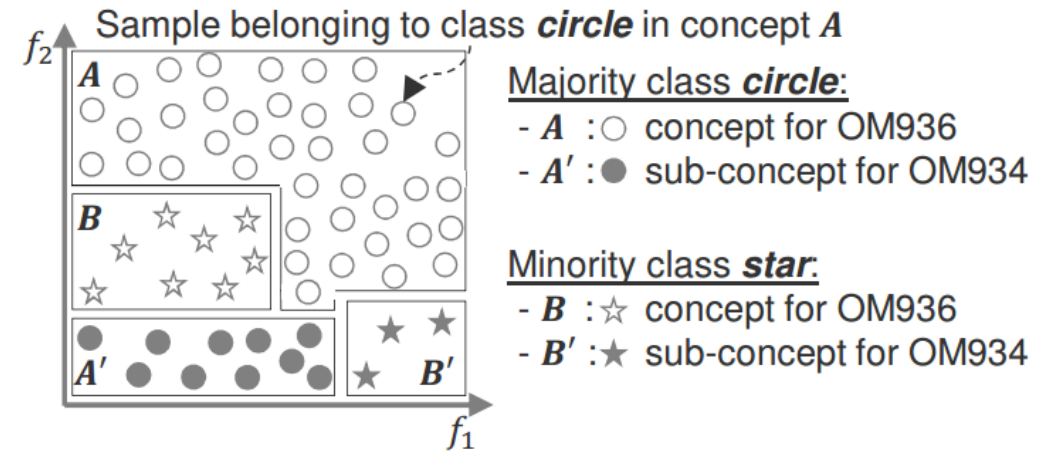
- Case: discovery of time series motifs (frequently recurring patterns)
  - **Input x**: a single (very long) time series (i.e. a signal, a waveform, a curve...; e.g. a 48h electrocardiography/ECG recording)
  - **Output y**: motifs that are ranked; the higher the ranking, the more interesting they are to the user (e.g. the most typical ECG heartbeat pattern in the recording)
  - Note: this is technically not a machine learning use case, but the idea still holds.
- Challenge: The top motif discovered is **not what is expected by the domain experts**.
- Solution: let **domain experts identify segments** in time series where **expected motifs are likely (or unlikely) to be found**.



In this ECG recording, the domain expert **removed the first 1000 data points**, because they correspond to the **calibration signal** at the beginning of the recording, **not the actual heartbeats**.

# Dividing The Training Set

- Case: finding faulty parts for truck engines
  - Input  $x$ : sensor signals
  - Output  $y$ : faulty parts
- Challenge: high intra-class heterogeneity, that is, even if two engines can have the same kinds of faulty parts, their symptoms may be very different because they are of different models.
- Solution: Using knowledge on engine model taxonomy to divide the training set; train an individual classifier for each engine series/type/model.



# Generating Training Labels

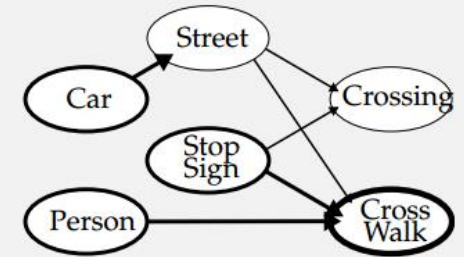
- Case: relation extraction
  - Input  $x$ : some text (e.g. *Barack Obama was born in Honolulu.*)
  - Output  $y$ : (subject, predicate, object) **triples** that characterize the relations (predicates) among entities (subjects or objects) mentioned in the text. [e.g. (*Barack Obama, born in, Honolulu*)]
- Challenge: The training of a relation extraction model can require large amounts of training text. it is **highly laborious to manually label** (namely, provide the correct triple corresponding to the text) **all of them**.
- Solution: We exploit a large pre-existing knowledge base (that contains very many triples) to **generate the labels**. This process is called *distant supervision*.



# Generating Training Labels

- Case: relation extraction
  - Input  $x$ : some text (e.g. *Barack Obama was born in Honolulu.*)
  - Output  $y$ : (subject, predicate, object) **triples** that characterize the relations (predicates) among entities (subjects or objects) mentioned in the text. [e.g. (*Barack Obama, born in, Honolulu*)]
- Challenge: The training of a relation extraction model can require large amounts of training text. It is highly laborious to manually label (namely, provide the correct triple corresponding to the text) all of them.
- Solution: We exploit a large pre-existing knowledge base (that contains very many triples) to generate the labels. This process is called *distant supervision*.
- **Basic assumption:** if two entities participate in a relation, any sentence that contains those two entities might express that relation. E.g.
  - Suppose we have the triple (*Barack Obama, born in, Honolulu*) in the knowledge base.
  - The distant supervision algorithm searches the (initially unlabeled) training text for **all sentences that simultaneously mention *Barack Obama* and *Honolulu***. E.g.
    - Barack Obama was born in Honolulu.
    - President Barack Obama was born in Honolulu, Hawaii.
    - ...
  - We can then **automatically label these sentences as having the relation *born in***. Namely, their corresponding triple is (*Barack Obama, born in, Honolulu*).

# Knowledge as Model Input



- Knowledge graphs are often directly used as part of the model input.
- Case: Image Classification
  - Input  $x$ : an image
  - Output  $y$ : a label describing the image (e.g. bird)
- Challenge: What if for certain labels, there are not enough training images?
  - E.g. the training set may only have one image with the label *cross walk*. This is not enough to train a data-driven model!
- We input into the model, apart from the images, a knowledge graph encoding the relationship between the rare target label and more common concepts. E.g.
  1. While we do not have enough data for *cross walk*, we have many for *car*, *person* and *stop sign*.
  2. We also have a knowledge graph telling us that an image containing these objects is likely to be that of a *cross walk*.
  3. Thus, we can determine if an image is a *cross walk* or not by detecting *car*, *person* and *stop sign* from it!

# Generating New Features

- Case: fault diagnosis for wind turbines
  - Input  $x$ : wind turbine sensor data
  - Output  $y$ : whether the turbine is faulty or not
- Based on the directly observable features, **generate new features by knowledge**.
  - E.g. Inside the turbine resides two temperature sensors. **Normally their readings should not differ much**. Thus, we can use **the difference between the two readings** as a new feature.
- The **generated features** are fed into the machine learning model **alongside the directly observable ones**.

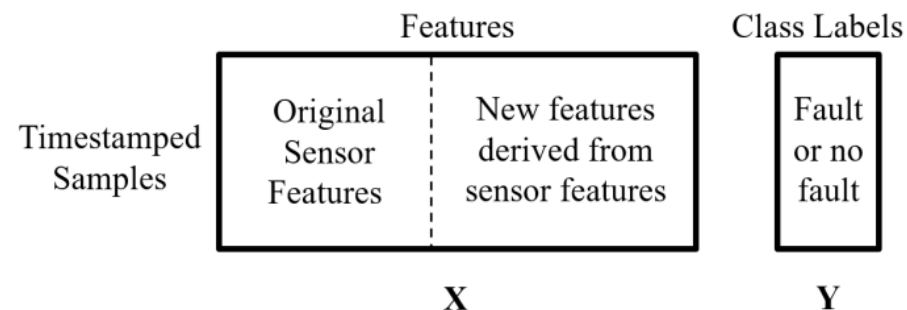
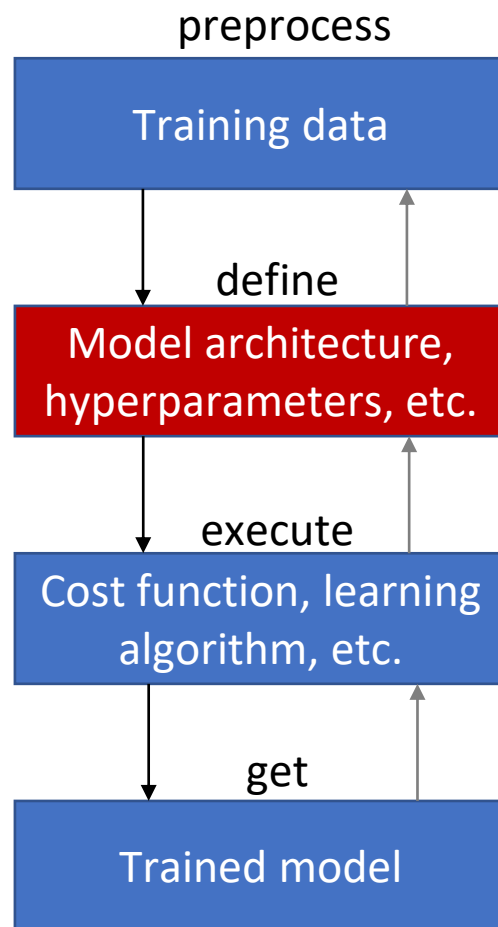


Table III  
EXAMPLE FEATURES FROM KNOWLEDGE OF WIND TURBINES

Average Of	Front and Rear Bearing Temperatures Rotor Temperatures Stator Temperatures ...
Difference Between	Max and Min of wind speed Max and Average of wind speed Min and Average of wind speed Front and Rear Bearing Temperature Nacelle Ambient Temperatures Generator Temperature and Nacelle Temperature ...
Ratio of	Average power to Available Power (from wind, technical reasons, force majeure reasons, force external reasons) ...

# Integrating Knowledge into a Data-driven pipeline

## Data-driven training pipeline for an ML model

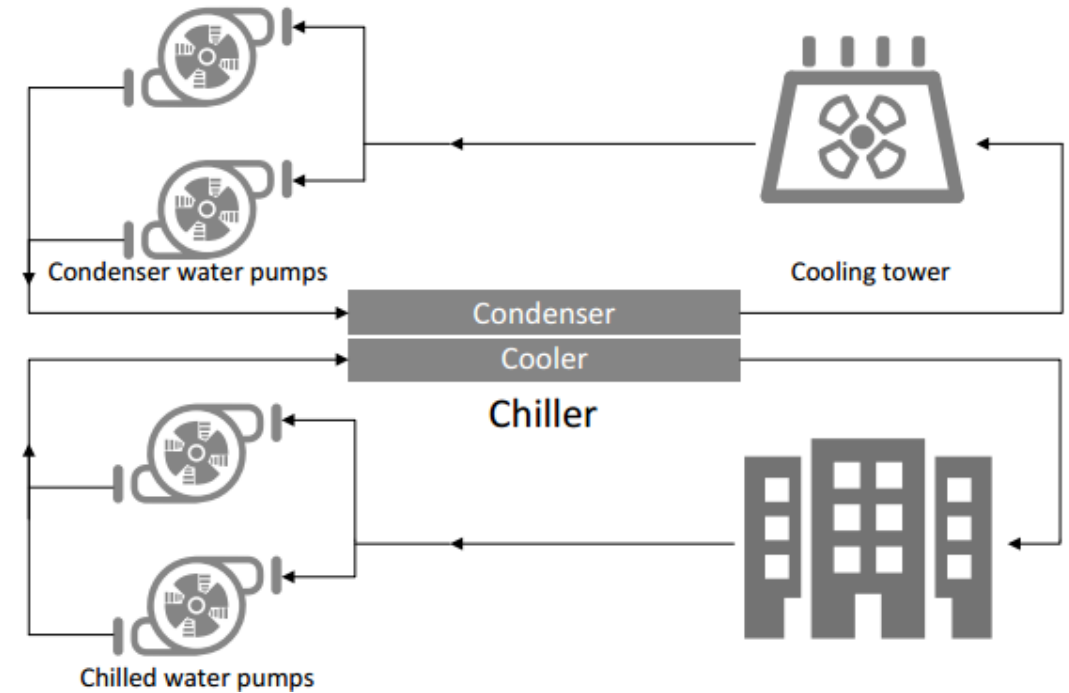


- Define the model you want to train.
  1. Model architecture:
    - Shallow models: linear regression, logistic regression, support vector machines, etc.
    - Deep models: convolutional, recurrent, transformers ...
  2. Hyperparameters
    - E.g. number of layers in a deep neural network, number of neurons in each layer, etc.

**You can use knowledge to guide the selection/design of the model architecture, etc.**

# Guiding Model Design

- Case: Modelling power of water pumps and cooling tower fans in a chiller plant
  - Input  $x$ : shaft speed
  - Output  $y$ : power
- Knowledge: Power  $y$  is proportional to the cube of shaft speed  $x^3$ .
- By this knowledge, we can simply construct a polynomial regression model, rather than a sophisticated deep learning model.



$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \dots + \alpha_k x^3 + \epsilon$$

where,

- $y$  : predicted value of power
- $x$  : control parameter
- $a_i$  : regression coefficient,  $i \in \{0, 1, \dots, k\}$
- $\epsilon$  : error term

# Guiding Model Design

- Case: tunnel construction work progress identification
  - **Input x:** an image taken in the tunnel
  - **Output y:** the type of work shown in the image
- Knowledge: knowledge on **the equipment used in each work type**
- Use data-driven methods to **identify the equipment**, and use a **probabilistic model** to encode the **relations between the work processes and the equipment**.
  - E.g. Suppose we have **identified equipment A** in the image, and by domain knowledge we **know A is usually used in work type B**, thus the **current work type has a higher probability to be B**.



Excavator



Loader



Shovel



Breaker



Mixer



Jumbo



Mortar Truck



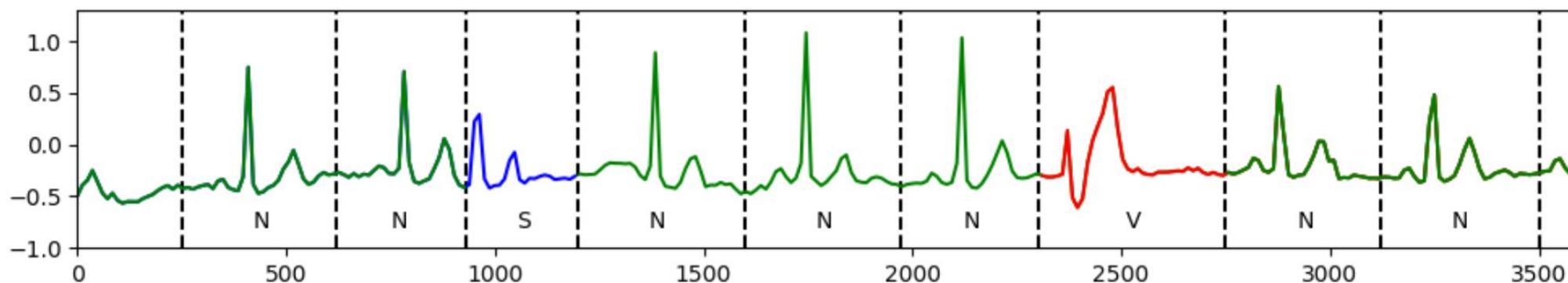
Erector/  
Spray Machine



Dump Truck

# Fusing Domain-specific Features with Deep Learning

- Unlike **shallow models** that rely on **pre-defined features**, **deep learning models learn their own features** that are **directly driven by the eventual task** (known as the “**downstream task**”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - **Input  $x$** : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - **Output  $y$** : the label of each input heartbeat. Here we have 3 labels:  $N$  (which *roughly* means *normal*, but not quite),  $V$  (which means *ventricular ectopic beat*), and  $S$  (which means *supraventricular ectopic beat*)



# Fusing Domain-specific Features with Deep Learning

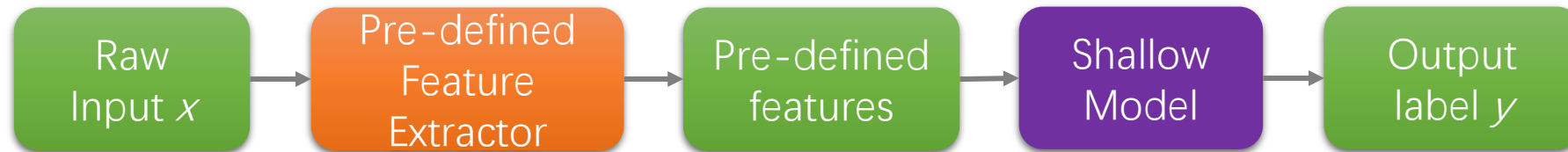
- Unlike shallow models that rely on pre-defined features, deep learning models learn their own features that are directly driven by the eventual task (known as the “downstream task”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - Input  $x$ : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - Output  $y$ : the label of each input heartbeat. Here we have 3 labels:  $N$ ,  $V$ ,  $S$
- This is how a **shallow model** would undertake this task





# Fusing Domain-specific Features with Deep Learning

- Unlike shallow models that rely on pre-defined features, deep learning models learn their own features that are directly driven by the eventual task (known as the “downstream task”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - Input  $x$ : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - Output  $y$ : the label of each input heartbeat. Here we 3 labels:  $N, V, S$
- This is how a **shallow model** would undertake this task

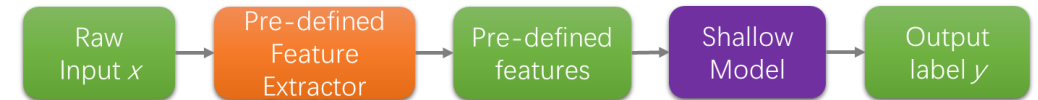


I need to accurately predict the labels of the input ECG heartbeats with the pre-defined features. I can only work on what I have been given...

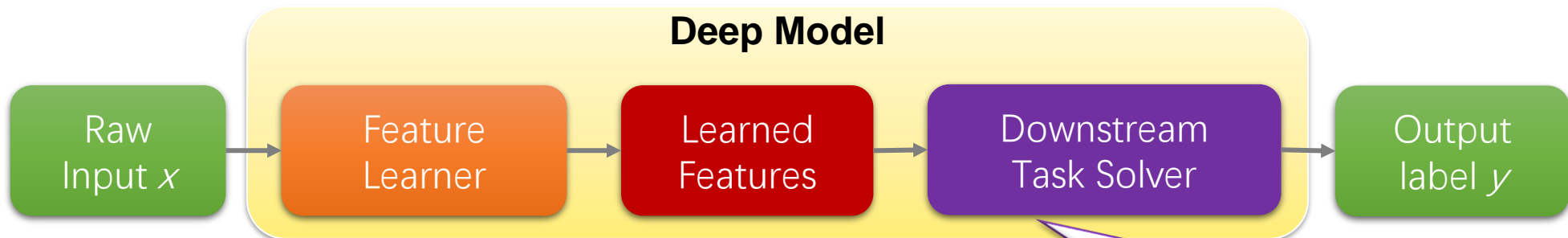
# Fusing Domain-specific Features with Deep Learning

- Unlike shallow models that rely on pre-defined features, deep learning models learn their own features that are directly driven by the eventual task (known as the “downstream task”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - Input  $x$ : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - Output  $y$ : the label of each input heartbeat. Here we have 3 labels:  $N, V, S$

• This is how a **shallow model** would undertake this task



• By contrast, this is how a deep model would do.

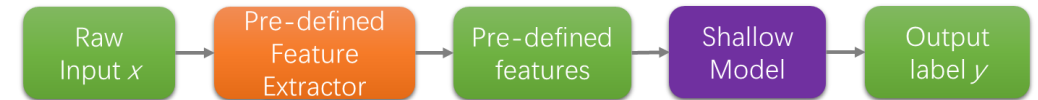


Hey **Feature Learner**, give me some features that can help me accurately predict the labels of these ECG heartbeats!

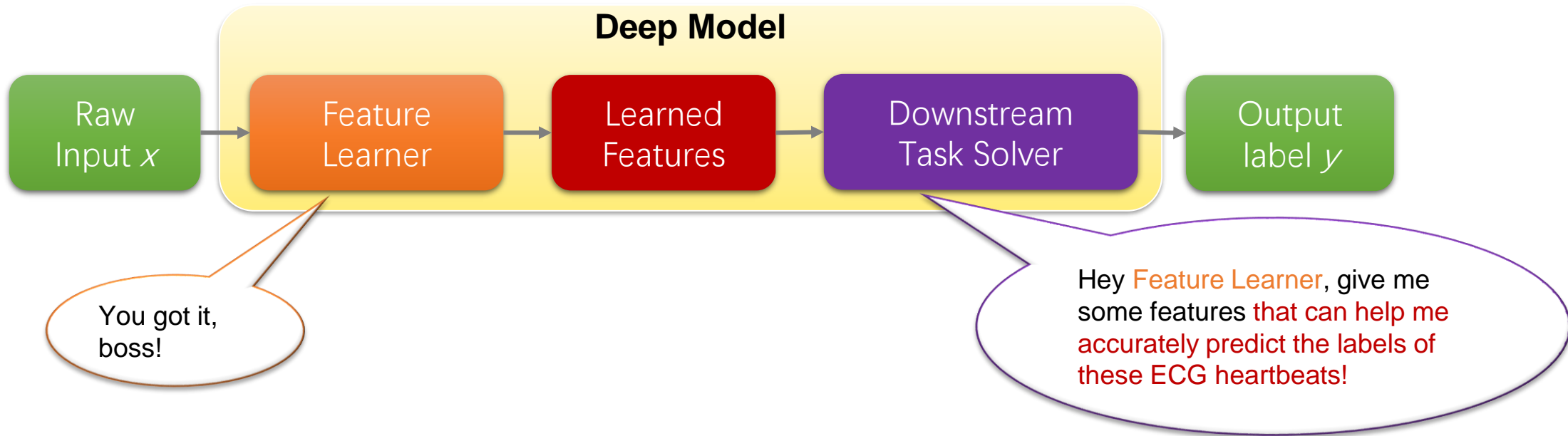
# Fusing Domain-specific Features with Deep Learning

- Unlike shallow models that rely on pre-defined features, deep learning models learn their own features that are directly driven by the eventual task (known as the “downstream task”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - Input  $x$ : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - Output  $y$ : the label of each input heartbeat. Here we have 3 labels:  $N, V, S$

• This is how a **shallow model** would undertake this task



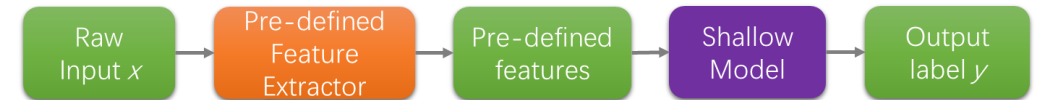
• By contrast, this is how a deep model would do.



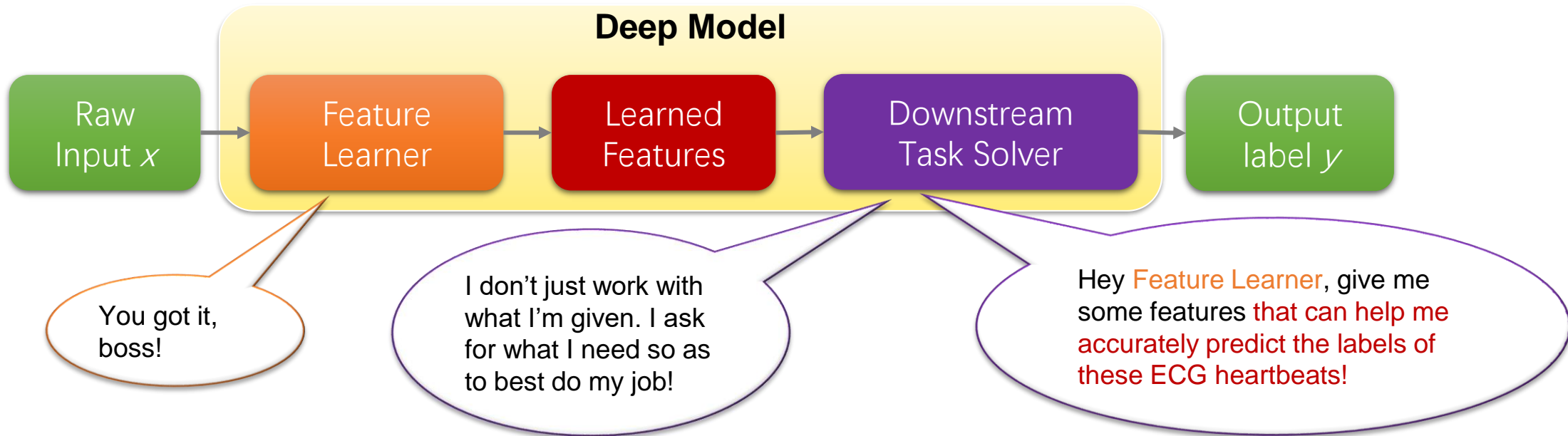
# Fusing Domain-specific Features with Deep Learning

- Unlike shallow models that rely on pre-defined features, deep learning models learn their own features that are directly driven by the eventual task (known as the “downstream task”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - Input  $x$ : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - Output  $y$ : the label of each input heartbeat. Here we have 3 labels:  $N, V, S$

• This is how a **shallow model** would undertake this task



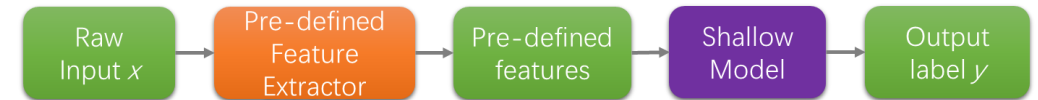
• By contrast, this is how a deep model would do.



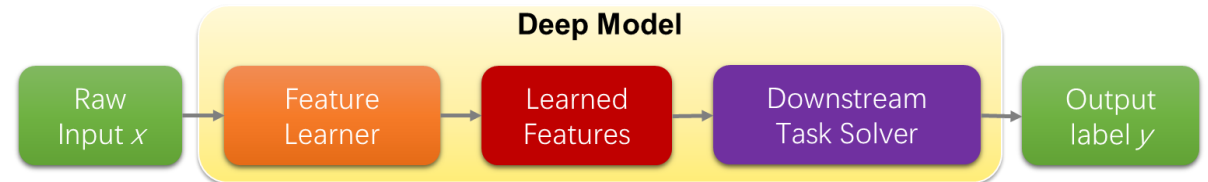
# Fusing Domain-specific Features with Deep Learning

- Unlike shallow models that rely on pre-defined features, deep learning models learn their own features that are directly driven by the eventual task (known as the “downstream task”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - Input  $x$ : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - Output  $y$ : the label of each input heartbeat. Here we have 3 labels:  $N, V, S$

• This is how a **shallow model** would undertake this task



• This is how a deep model would do.

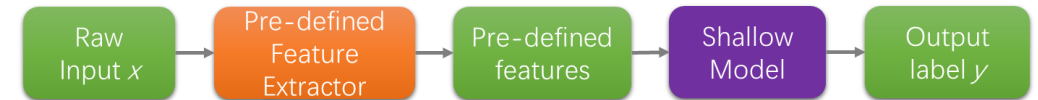


- The features are **tailored to the downstream task** (e.g. For ECG heartbeat classification, the features are tailored to maximizing the accuracy of the prediction of the heartbeat labels).

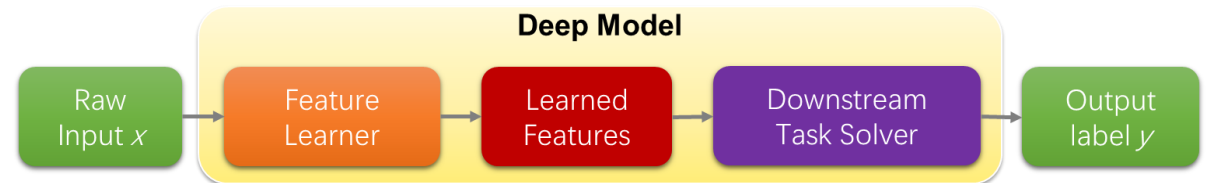
# Fusing Domain-specific Features with Deep Learning

- Unlike shallow models that rely on pre-defined features, deep learning models learn their own features that are directly driven by the eventual task (known as the “downstream task”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - Input  $x$ : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - Output  $y$ : the label of each input heartbeat. Here we have 3 labels:  $N, V, S$

- This is how a **shallow model** would undertake this task



- This is how a deep model would do.

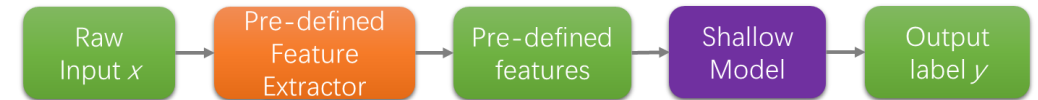


- The features are **tailored to the downstream task** (e.g. For ECG heartbeat classification, the features are tailored to maximizing the accuracy of the prediction of the heartbeat labels).
- Compared with **pre-defined features** in shallow models, which are **decoupled from the downstream task** (which means they may or may not be useful in undertaking the task), **the deep learned features can usually better contribute to the eventual prediction.**
- However, deep learned features **have weaknesses** as compared with pre-defined ones.

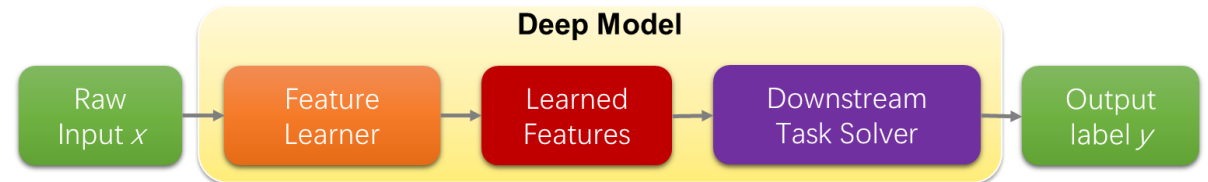
# Fusing Domain-specific Features with Deep Learning

- Unlike shallow models that rely on pre-defined features, deep learning models learn their own features that are directly driven by the eventual task (known as the “downstream task”).
- E.g. ECG heartbeat classification for cardiac arrhythmia detection
  - Input  $x$ : an electrocardiograph (ECG) signal that encompasses one or more heartbeats
  - Output  $y$ : the label of each input heartbeat. Here we have 3 labels:  $N, V, S$

- This is how a **shallow model** would undertake this task



- This is how a deep model would do.



- However, deep learned features **have weaknesses** as compared with pre-defined ones.
  - **Poor explainability**: deep learning are notorious for being black-box models; in particular, **it is often hard for humans to understand what the deep learned features mean.**

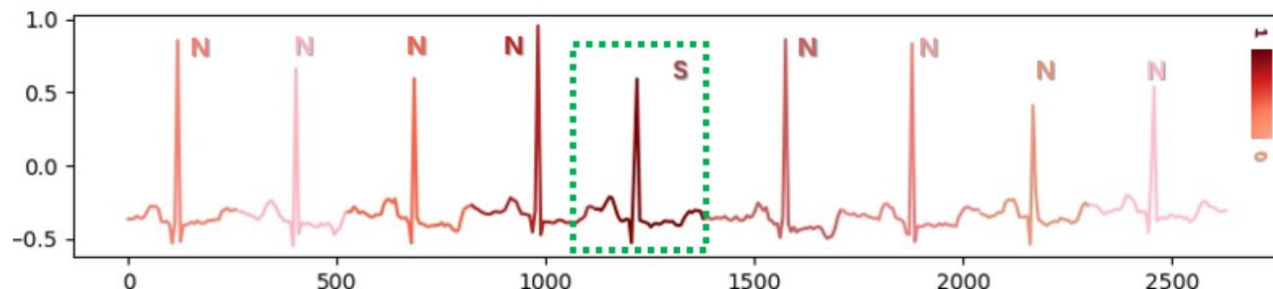
# Fusing Domain-specific Features with Deep Learning

- Deep learned features **have weaknesses** as compared with pre-defined ones.
  - Poor explainability: deep learning are notorious for being black-box models; in particular, it is often hard for humans to understand what the deep learned features mean.
  - **Limitations in model architecture**: the quality of the learned features (i.e. their usefulness in solving the downstream task) is closely related to the architecture of the feature learner. **If there are intrinsic limitations in the feature learner, the usefulness of the features are likely to be limited.**



# Fusing Domain-specific Features with Deep Learning

- Deep learned features **have weaknesses** as compared with pre-defined ones.
  - Poor explainability: deep learning are notorious for being black-box models; in particular, it is often hard for humans to understand what the deep learned features mean.
  - **Limitations in model architecture**: the quality of the learned features (i.e. their usefulness in solving the downstream task) is closely related to the architecture of the feature learner. **If there are intrinsic limitations in the feature learner, the usefulness of the features are likely to be limited.**
    - For example, in ECG heartbeat classification where we label each heartbeat as one of  $(N, V, S)$ , **the periodicity of multiple heartbeats** are **equally important** as **the morphology of singular heartbeats**.
    - In particular,  $N$  and  $S$  can often **have very similar morphology**, while the distinction between the two **heavily relies on their periodical differences**.



The key distinction between the  $S$  example and its  $N$  context is that **the  $S$  example is slightly closer to its preceding heartbeat!**

# Fusing Domain-specific Features with Deep Learning

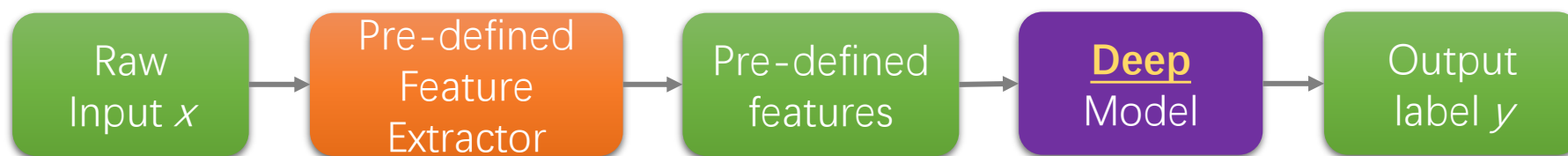
- Deep learned features **have weaknesses** as compared with pre-defined ones.
  - Poor explainability: deep learning are notorious for being black-box models; in particular, it is often hard for humans to understand what the deep learned features mean.
  - **Limitations in model architecture**: the quality of the learned features (i.e. their usefulness in solving the downstream task) is closely related to the architecture of the feature learner. **If there are intrinsic limitations in the feature learner, the usefulness of the features are likely to be limited.**
    - For example, in ECG heartbeat classification where we label each heartbeat as one of  $(N, V, S)$ , **the periodicity of multiple heartbeats** are **equally important** as **the morphology of singular heartbeats**.
    - In particular,  $N$  and  $S$  can often **have very similar morphology**, while the distinction between the two **heavily relies on their periodical differences**.
    - However, most deep ECG feature learners (especially those that only take a single heartbeat as input) **can not effectively capture the periodical characteristics**, leading to poor performance when predicting  $S$  examples.

# Fusing Domain-specific Features with Deep Learning

- Deep learned features **have weaknesses** as compared with pre-defined ones.
  - Poor explainability: deep learning are notorious for being black-box models; in particular, it is often hard for humans to understand what the deep learned features mean.
  - Limitations in model architecture: the quality of the learned features (i.e. their usefulness in solving the downstream task) is closely related to the architecture of the feature learner. If there are intrinsic limitations in the feature learner, the usefulness of the features are likely to be limited.
- To address these issues, we may **integrate pre-defined features** (especially those defined using **domain knowledge**) **into deep learning!**

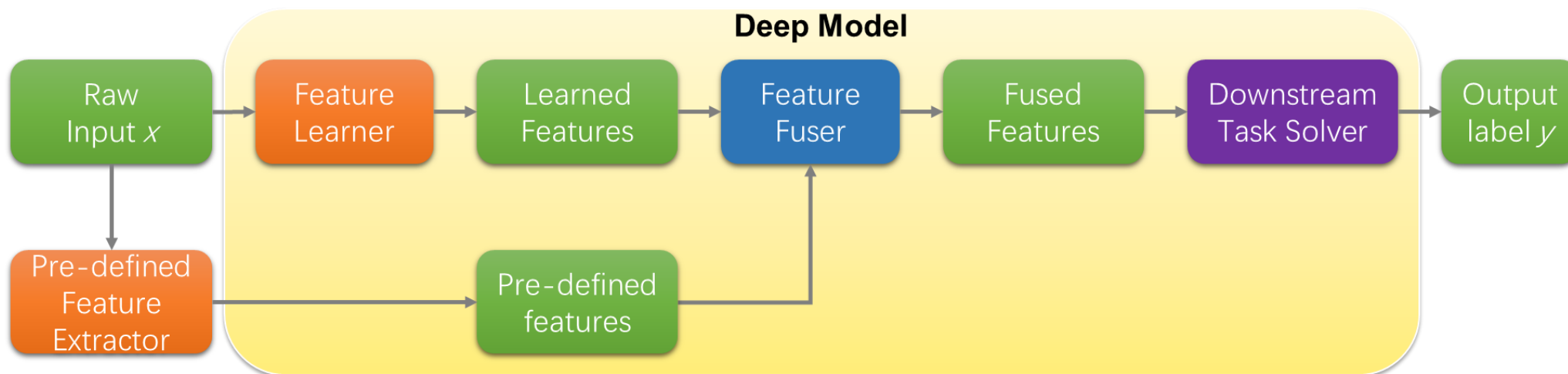
# Fusing Domain-specific Features with Deep Learning

- Deep learned features have weaknesses as compared with pre-defined ones.
- To address these issues, we may integrate pre-defined features (especially those defined using domain knowledge) into deep learning!
- A **conventional** way of fusing pre-defined features is to **use pre-defined features (rather than raw data) as the input of a deep model**, as if the deep model were a shallow model. This way, the deep model can **refine the pre-defined features** with its feature learner.



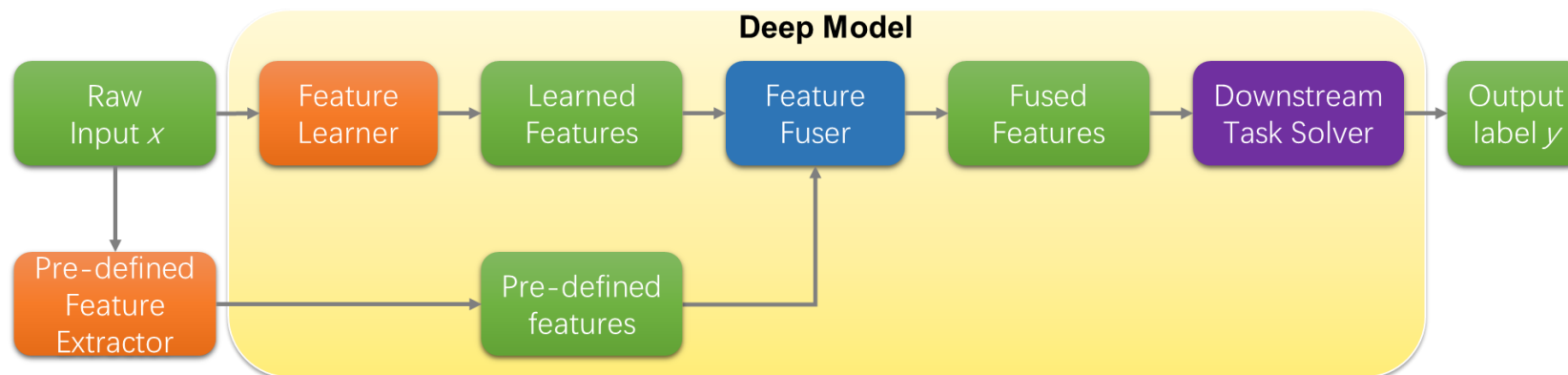
# Fusing Domain-specific Features with Deep Learning

- Deep learned features have weaknesses as compared with pre-defined ones.
- To address these issues, we may integrate pre-defined features (especially those defined using domain knowledge) into deep learning!
- A conventional way of fusing pre-defined features is to use pre-defined features (rather than raw data) as the input of a deep model, as if the deep model were a shallow model. This way, the deep model can refine the pre-defined features with its feature learner.
- However, there is another way to fuse the pre-defined features, which is to **use the pre-defined features as part of the deep model, directly fusing them (e.g. concatenating) with the deep learned features.**



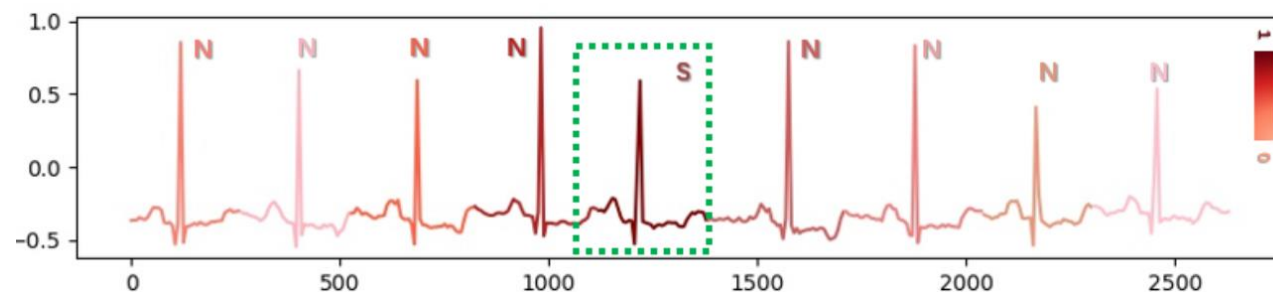
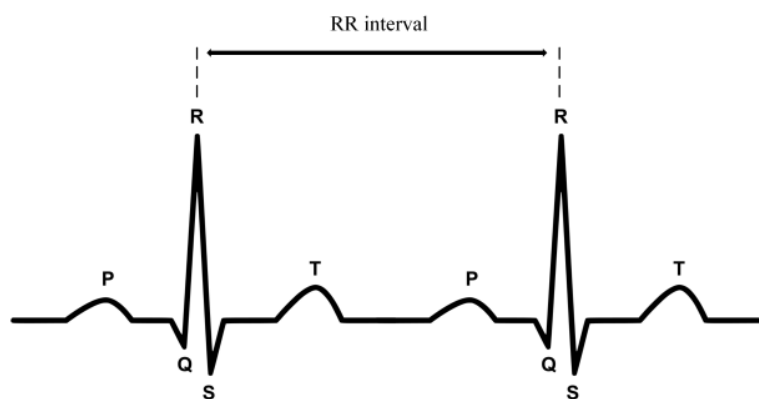
# Fusing Domain-specific Features with Deep Learning

- Deep learned features have weaknesses as compared with pre-defined ones.
- To address these issues, we may integrate pre-defined features (especially those defined using domain knowledge) into deep learning!
- A conventional way of fusing pre-defined features is to use pre-defined features (rather than raw data) as the input of a deep model, as if the deep model were a shallow model. This way, the deep model can refine the pre-defined features with its feature learner.
- However, there is another way to fuse the pre-defined features, which is to **use the pre-defined features as part of the deep model, directly fusing them (e.g. concatenating) with the deep learned features.**
- In this setting, the **pre-defined features** are **parallel complements** to the learned features. In particular, if the pre-defined features are based on **domain knowledge**, then we are **directly injecting knowledge into the network.**



# Fusing Domain-specific Features with Deep Learning

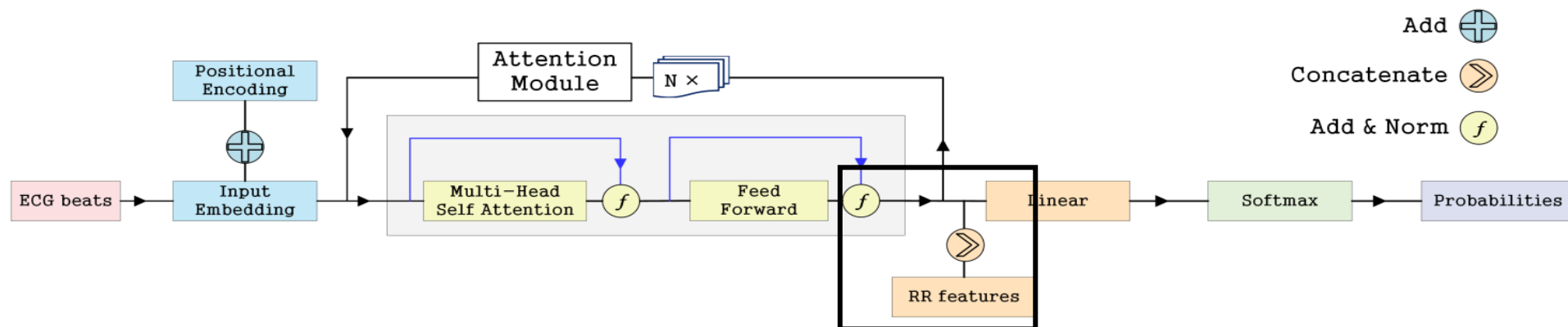
- A conventional way of fusing pre-defined features is to use pre-defined features (rather than raw data) as the input of a deep model, as if the deep model were a shallow model. This way, the deep model can refine the pre-defined features with its feature learner.
- However, there is another way to fuse the pre-defined features, which is to use the pre-defined features as part of the deep model, directly fusing them (e.g. concatenating) with the deep learned features.
- In this setting, the pre-defined features are parallel complements to the learned features. In particular, if the pre-defined features are based on domain knowledge, then we are directly injecting knowledge into the network.
- Let's go back to the ECG heartbeat classification task. While the neural network may not be able to capture the **periodicity characteristics** vital to distinguishing between *N* and *S* heartbeats, there are **domain-specific features called *RR-intervals in cardiology*** that can.



The *S* example has a smaller RR-interval with its preceding heartbeat than *N* examples.

# Fusing Domain-specific Features with Deep Learning

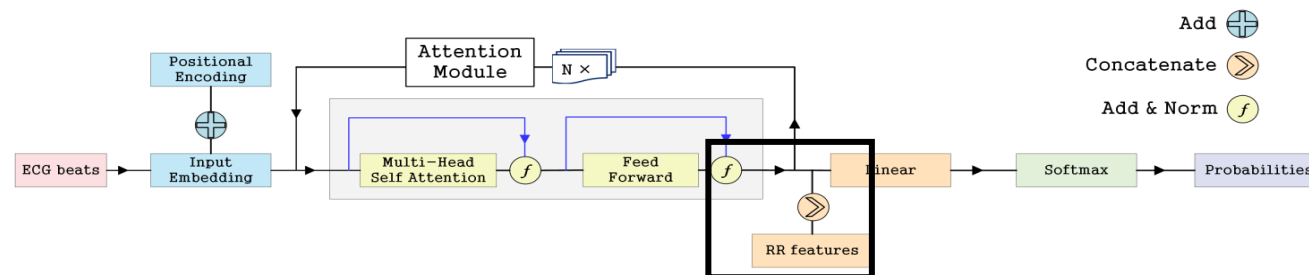
- A conventional way of fusing pre-defined features is to use pre-defined features (rather than raw data) as the input of a deep model, as if the deep model were a shallow model. This way, the deep model can refine the pre-defined features with its feature learner.
- However, there is another way to fuse the pre-defined features, which is to use the pre-defined features as part of the deep model, directly fusing them (e.g. concatenating) with the deep learned features.
- In this setting, the pre-defined features are parallel complements to the learned features. In particular, if the pre-defined features are based on domain knowledge, then we are directly injecting knowledge into the network.
- Let's go back to the ECG heartbeat classification task. While the neural network may not be able to capture the **periodicity characteristics** vital to distinguishing between  $N$  and  $S$  heartbeats, there are **domain-specific features called RR-intervals in cardiology** that can.
- Accordingly, we can **fuse (e.g. concatenate) the RR-interval values with the deep learned features**. This way, the **deep feature learner** can mainly focus on capturing **morphological features** (which is what it is good at), while the **periodicity features** are left in the capable hands of the **domain-specific RR-interval features**!





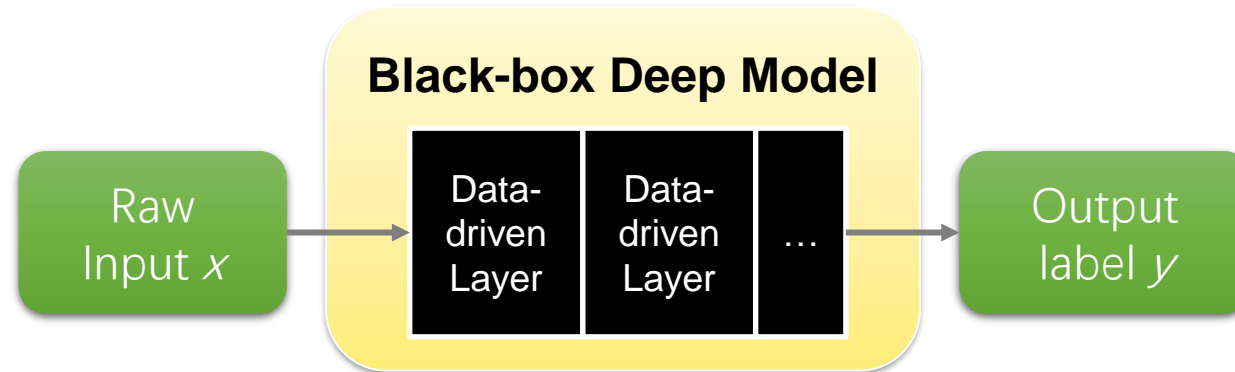
# Fusing Domain-specific Features with Deep Learning

- A conventional way of fusing pre-defined features is to use pre-defined features (rather than raw data) as the input of a deep model, as if the deep model were a shallow model. This way, the deep model can refine the pre-defined features with its feature learner.
- However, there is another way to fuse the pre-defined features, which is to use the pre-defined features as part of the deep model, directly fusing them (e.g. concatenating) with the deep learned features.
- In this setting, the pre-defined features are parallel complements to the learned features. In particular, if the pre-defined features are based on domain knowledge, then we are directly injecting knowledge into the network.
- Going back to the ECG heartbeat classification task. While the neural network may not be able to capture the **periodicity characteristics** vital to distinguishing between  $N$  and  $S$  heartbeats, there are **domain-specific features called RR-intervals in cardiology** that can.
- Accordingly, we can **fuse (e.g. concatenate) the RR-interval values with the deep learned features**. This way, the **deep feature learner** can mainly focus on capturing **morphological features** (which is what it is good at), while the **periodicity features** are left in the capable hands of the **domain-specific RR-interval features**!
- An added benefit of fusing domain-specific features from **within** the deep model is that they can **seamlessly interact with and enhance the deep-learned features**.
  - For example, by fusing RR-intervals with deep-learned features, not only do the deep-learned features encode morphological characteristics, **it can also encode some periodicity characteristics passed on to it by the RR-intervals**.



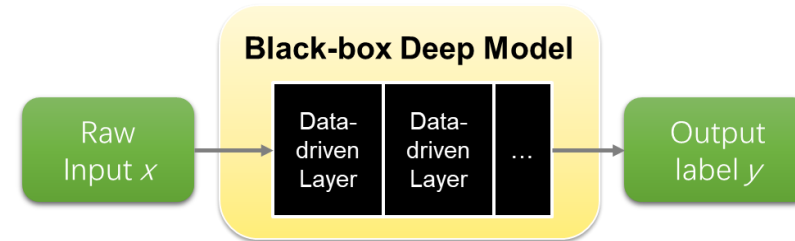
# Knowledge as Layers in Deep Networks

- As mentioned previously, conventional deep neural networks, with every (hidden) layer being data-driven, tend to be **black boxes**. That is, we **cannot fully comprehend (and thus have limited control of)** how the deep model maps input  $x$  to output  $y$ .

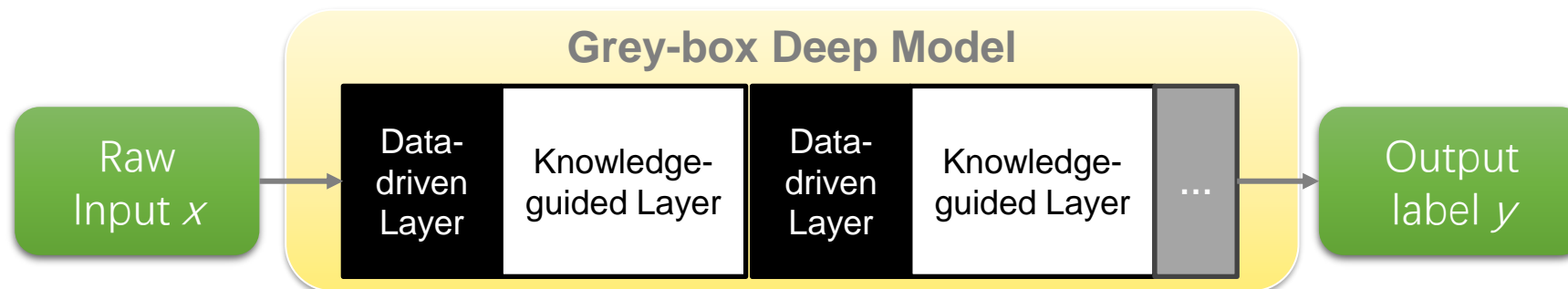


# Knowledge as Layers in Deep Networks

- As mentioned previously, conventional deep neural networks, with every (hidden) layer being data-driven, tend to be black boxes. That is, we cannot fully comprehend (and thus have limited control of) how the deep model maps input  $x$  to output  $y$ .

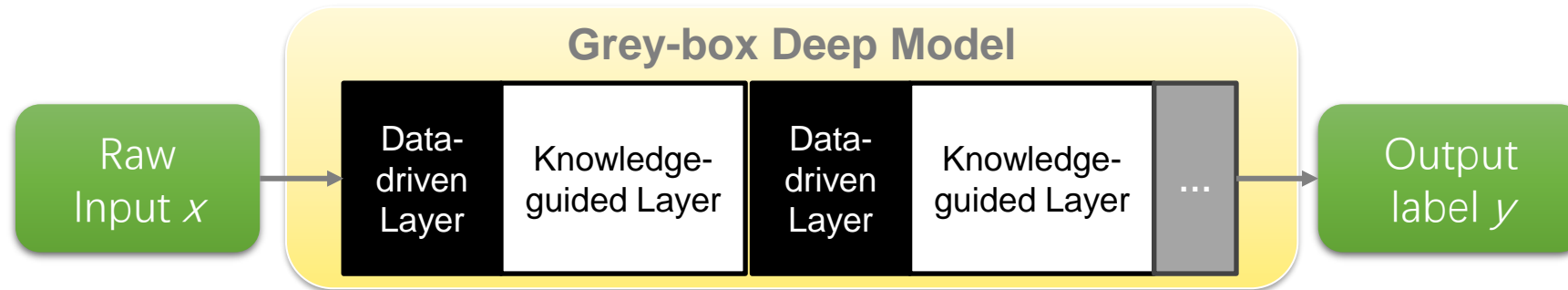


- However, if we have some **domain knowledge** that offers us viable options on the **intermediate steps** to take when mapping  $x$  to  $y$ , we can encode such intermediate steps as **(hidden) layers in the deep model**, creating **grey-box models**.



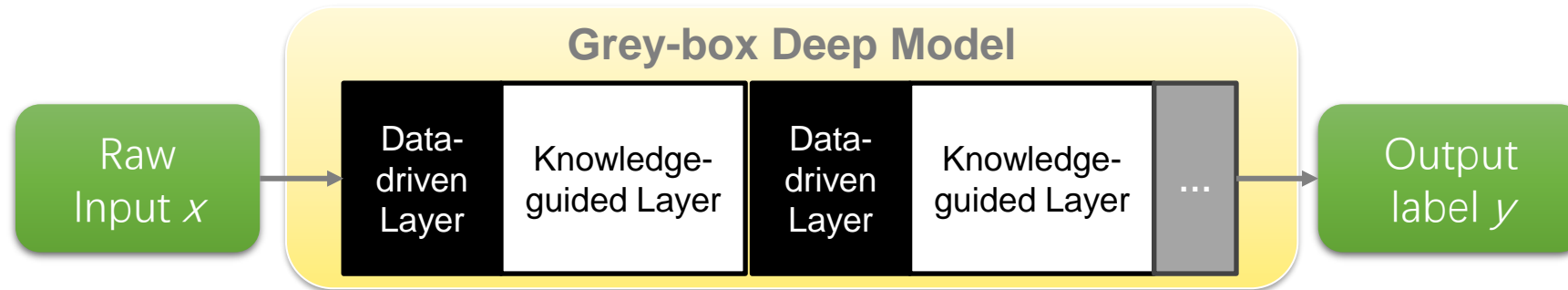
# Knowledge as Layers in Deep Networks

- However, if we have some **domain knowledge** that offers us viable options on the **intermediate steps** to take when mapping  $x$  to  $y$ , we can encode such intermediate steps as **(hidden) layers in the deep model**, creating **grey-box models**.
- The benefits of such a model are two-fold:
  - it has **better explainability** than a black-box mode;
  - it gives us **more control** over the model, enabling us to use carefully curated domain knowledge to enhance the deep model (e.g. **improve its performance on small training sets**).



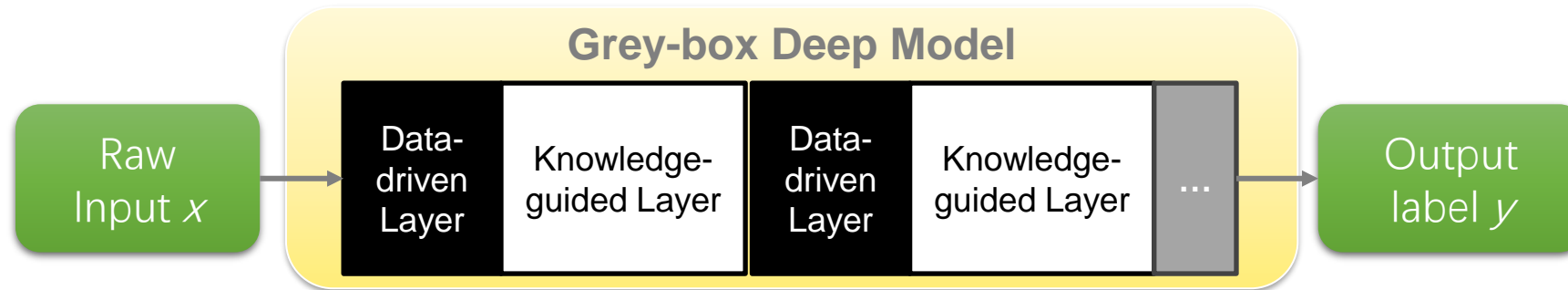
# Knowledge as Layers in Deep Networks

- However, if we have some **domain knowledge** that offers us viable options on the **intermediate steps** to take when mapping  $x$  to  $y$ , we can encode such intermediate steps as **(hidden) layers in the deep model**, creating **grey-box models**.
- We usually have **two types domain knowledge** that can serve as layers in deep networks.
  - Important intermediate outputs
  - Domain-specific mathematical models



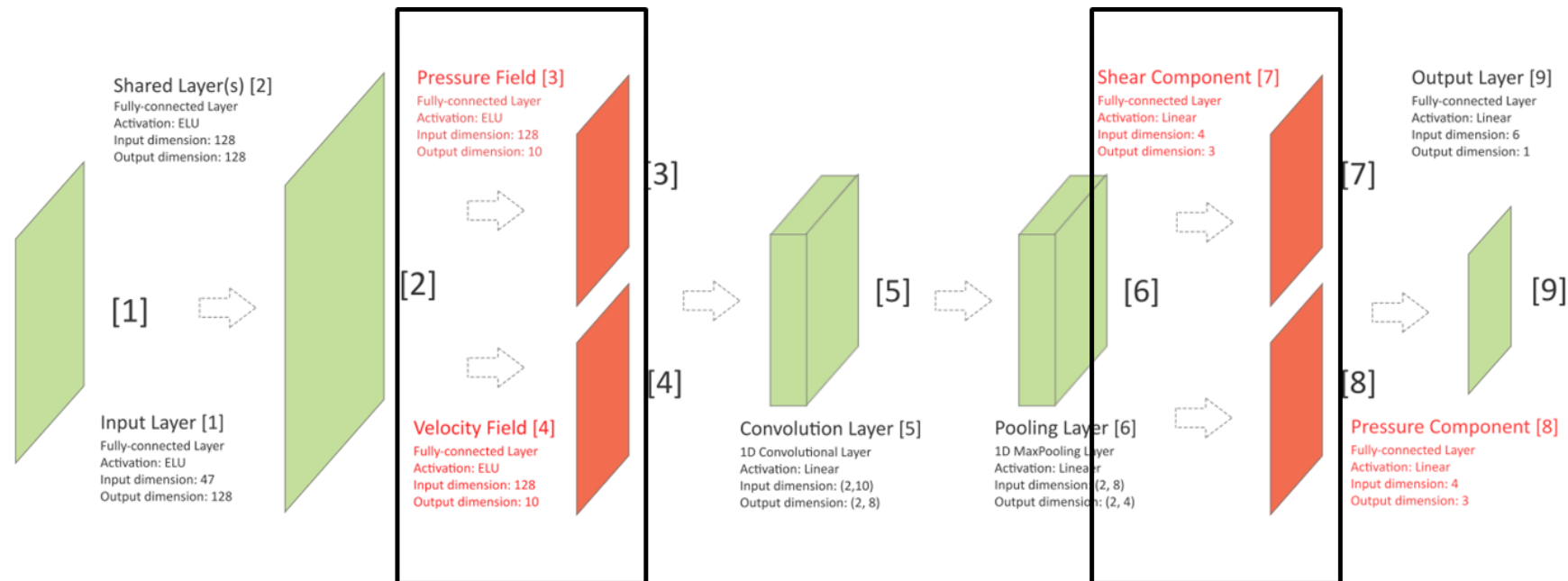
# Knowledge as Layers in Deep Networks

- However, if we have some domain knowledge that offers us viable options on the intermediate steps to take when mapping  $x$  to  $y$ , we can encode such intermediate steps as (hidden) layers in the deep model, creating grey-box models.
- We usually have two types domain knowledge that can serve as layers in deep networks.
  - [Important intermediate outputs](#)
  - Domain-specific mathematical models



# Knowledge as Layers in Deep Networks

- Case: particle drag force prediction in assembly
  - Background: we are interested in a single particle among a collection of particles in a fluid
  - **Input  $x$ : spatial arrangement of particles near the particle-of-interest**
  - **Output  $y$ : the drag force experienced by the particle-of-interest**
- **Important intermediate outputs: *pressure and velocity fields, shear component***



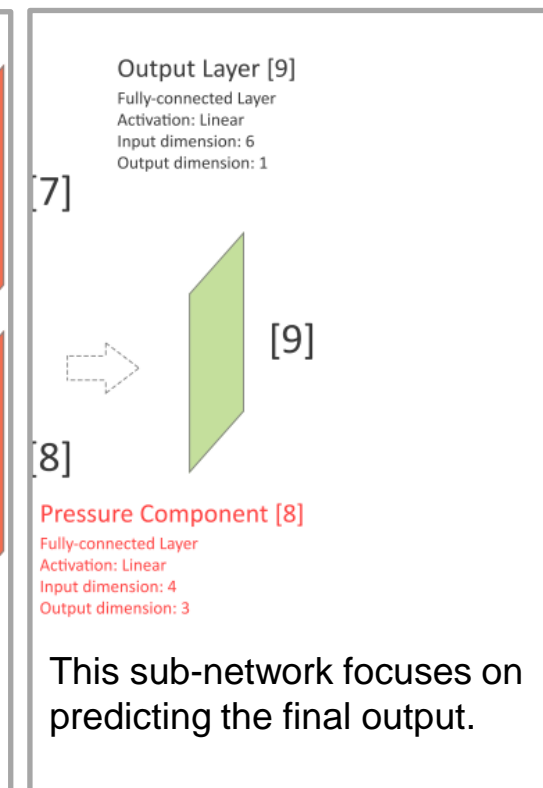
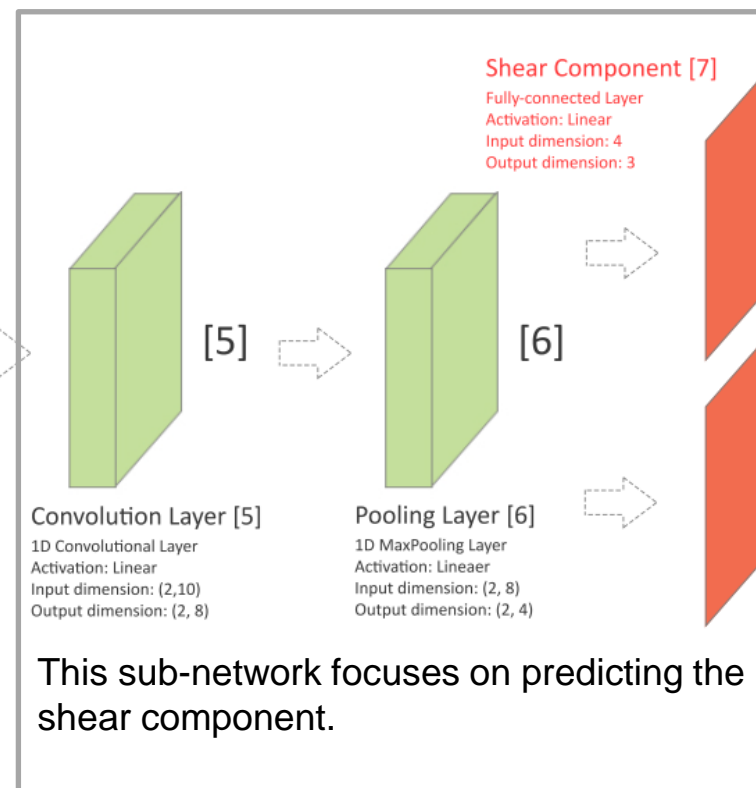
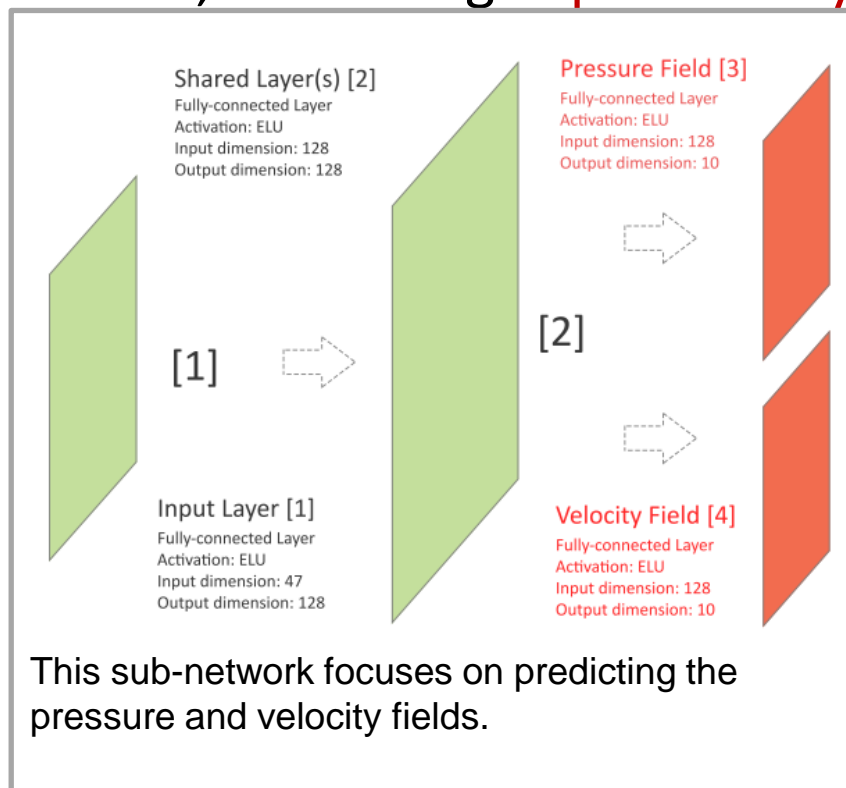
# Knowledge as Layers in Deep Networks

- Case: particle drag force prediction in assembly
  - Background: we are interested in a single particle among a collection of particles in a fluid
  - Input  $x$ : spatial arrangement of particles near the particle-of-interest
  - Output  $y$ : the drag force experienced by the particle-of-interest
- The benefit of having important intermediate outputs (especially if we know their groundtruth values) is that we can monitor their predicted values, compare them with their groundtruth values, and guide the network to reduce the prediction error of not only the eventual output  $y$ , but also the prediction errors of the intermediate outputs.
  - This can be done by altering the cost function of the deep model, which we will discuss later!
- In this way, not only can we get multiple outputs with one single network, but we can reduce the chance of the model overfitting the training set (that is, the model only works far worse on unseen examples than on training data).
- Moreover, we can roughly **know what each sub-network is doing** in the deep model, enhancing **explainability**.



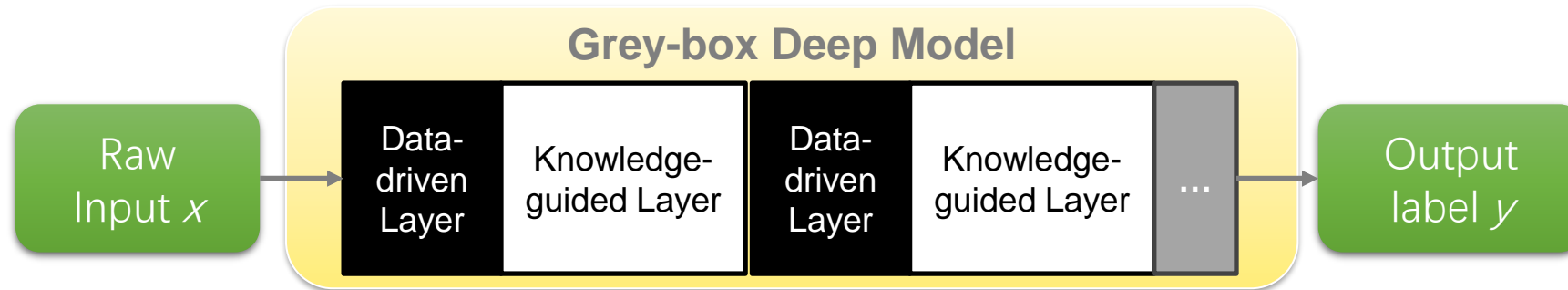
# Knowledge as Layers in Deep Networks

- By integrating intermediate outputs into the network, not only can we get multiple outputs with one single network, but we can reduce the chance of the model overfitting the training set (that is, the model only works far worse on unseen examples than on training data).
- Moreover, we can roughly **know what each sub-network is doing** in the deep model, enhancing **explainability**.



# Knowledge as Layers in Deep Networks

- However, if we have some domain knowledge that offers us viable options on the intermediate steps to take when mapping  $x$  to  $y$ , we can encode such intermediate steps as (hidden) layers in the deep model, creating grey-box models.
- We usually have two types domain knowledge that can serve as layers in deep networks.
  - Important intermediate outputs
  - [Domain-specific mathematical models](#)



# Knowledge as Layers in Deep Networks

- In many applications, there are **domain-specific mathematical models** (often called ***mechanism models***) that **directly maps the input  $x$  to output  $y$** . They take on the general form of

$$y = f(x; \theta)$$

where  $f$  is the mapping from  $x$  to  $y$ , with  $\theta$  being **model parameters** whose values must be either manually set or, more often than not, **automatically found**.

- E.g. One way of **predicting the viscosity of melts** is to use the VFT model:

$$y = A + \frac{(12 - A)(T_g - C)}{x - C}$$

where **input  $x$**  is the temperature, **output  $y$**  is the base-10 logarithm of the melt viscosity (namely, the actual viscosity is  $10^y$ ), while  **$A, C, T_g$  are model parameters**.

# Knowledge as Layers in Deep Networks

- In many applications, there are domain-specific mathematical models (often called mechanism models) that directly maps the input  $x$  to output  $y$ . They take on the general form of

$$y = f(x; \theta)$$

where  $f$  is the mapping from  $x$  to  $y$ , with  $\theta$  being model parameters whose values must be either manually set or, more often than not, automatically found.

- The use of mechanism models is often made challenging by two factors.
  - The accuracy of mechanism models are directly related to the setting of model parameters  $\theta$ . Traditionally, this is done with classic mathematical optimization methods (e.g. least squares, likelihood maximization...), whose effectiveness may be limited.

# Knowledge as Layers in Deep Networks

- In many applications, there are domain-specific mathematical models (often called *mechanism models*) that directly maps the input  $x$  to output  $y$ . They take on the general form of

$$y = f(x; \theta)$$

where  $f$  is the mapping from  $x$  to  $y$ , with  $\theta$  being model parameters whose values must be either manually set or, more often than not, automatically found.

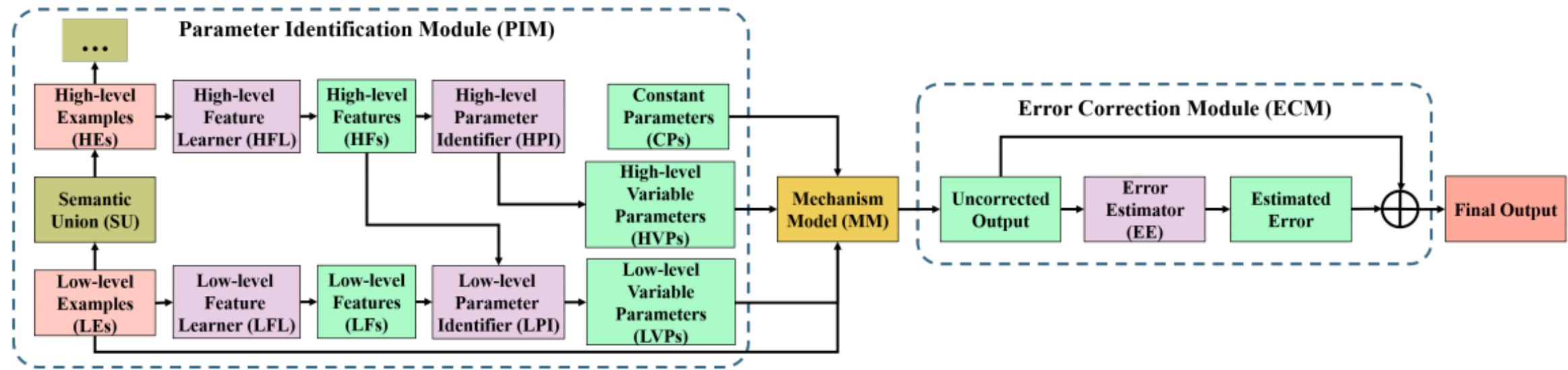
- The use of mechanism models is often made challenging by two factors.
  - The accuracy of mechanism models are directly related to the setting of model parameters  $\theta$ . Traditionally, this is done with classic mathematical optimization methods (e.g. least squares, likelihood maximization...), whose effectiveness may be limited.
  - Mechanism models are often (overly) simplified summary of complex real-world data, thus they inherently entail a level of error. In fact, many mechanism models have explicit error terms in them. However, even these error terms can rely on overly simplified assumptions (e.g. the error distribution is Gaussian).

# Knowledge as Layers in Deep Networks

- The use of mechanism models is often made challenging by two factors.
  - The accuracy of mechanism models are directly related to the setting of model parameters  $\theta$ . Traditionally, this is done with classic mathematical optimization methods (e.g. least squares, likelihood maximization...), whose effectiveness may be limited.
  - Mechanism models are often (overly) simplified summary of complex real-world data, thus they inherently entail a level of error. In fact, many mechanism models have explicit error terms in them. However, even these error terms can rely on overly simplified assumptions (e.g. the error distribution is Gaussian).
- To address these issues, note that many mechanism models can directly fit into neural networks (without causing your code to report error when training the networks).
- Thus, we can use **the mechanism model as the core of a neural network**, while **using data-driven deep sub-networks** to **optimize its parameters  $\theta$** , and **estimate its error**.
- Compared with conventional non-deep parameter optimization and error estimation methods, these **deep sub-networks tend to yield better outcomes** due to their **adaptability to complex real-world data**.

# Knowledge as Layers in Deep Networks

- We can use the mechanism model as the core of a neural network, while using data-driven deep sub-networks to optimize its parameters  $\theta$ , and estimate its error.



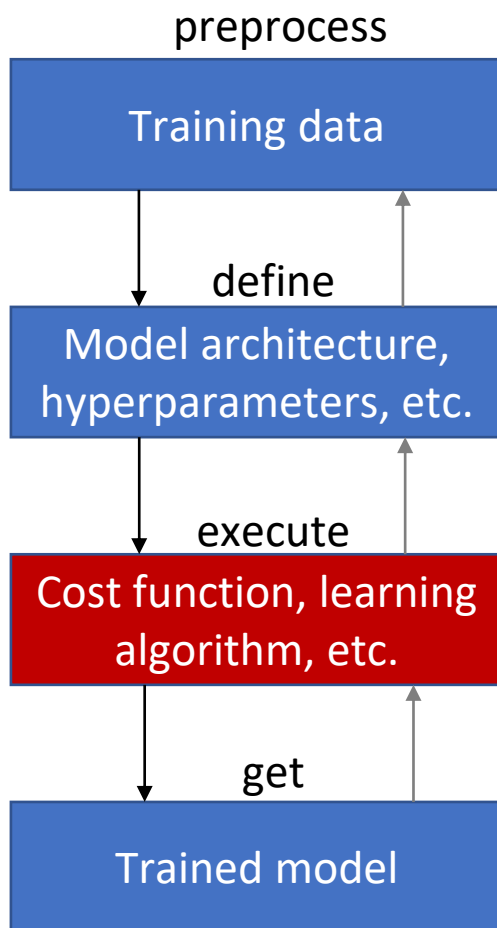
# Knowledge as the Entire Deep Model

- Sometimes, the **domain knowledge** can be **hard to directly exploit in its original form**. In such cases, we can **use an entire deep model to emulate it**, which can be **easier to handle** than in its original form.
- A typical example of this is the **Physic-Informed Neural Networks (PINNs)**.
  - In physics related domains, knowledge is often presented as **Partial Differential Equations (PDEs)**, which can be **hard to solve** in their raw form.
  - PINNs **emulate PDEs as deep neural networks**, and **use the computational power of modern deep learning frameworks** (PyTorch, Tensorflow, etc.) to efficiently solve the PDEs.
  - *We will look into how PINNs work in detail in our hands-on session!*



# Data-driven Model Training Pipeline

## Data-driven training pipeline for an ML model

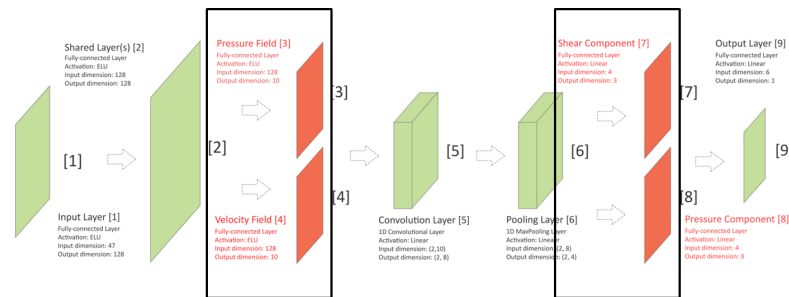


- Decide how you want to train the defined model
1. **Cost/Lost function**: quantifies **how good** your model is; you get to decide **what good means**.
    - If there are multiple ways to define *good*, you can have **multiple terms in your cost function**. In this case, the cost function is **the (weighted) sum of all these terms** (and you get to decide the weight of each term yourself).
  2. **Learning algorithm**: **the means by which the model becomes good** (with *good* defined by the cost function).

**You can integrate knowledge into the cost function, etc.**

# Knowledge as Loss Function Terms

- One way to define *good* in the loss function is **for the model to comply with domain knowledge**.
- Recall that we can **use important intermediate results as layers** in deep networks. In such cases, we can define *good* as **being able to accurately predict these intermediate results**.

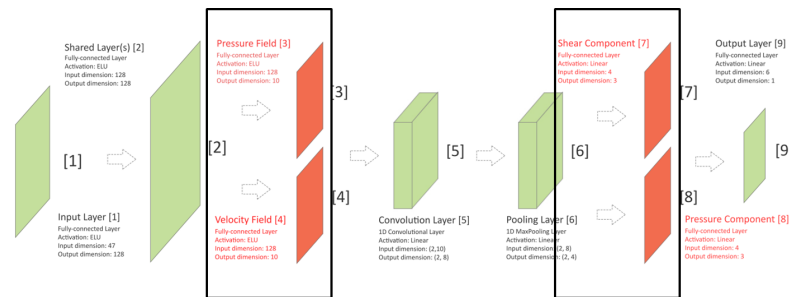


$$Loss_{MSE} = \lambda_P MSE(\mathbf{P}, \hat{\mathbf{P}}) + \lambda_V MSE(\mathbf{V}, \hat{\mathbf{V}}) + \lambda_{FP} MSE(\mathbf{F}^P, \hat{\mathbf{F}}^P) + \lambda_{FS} MSE(\mathbf{F}^S, \hat{\mathbf{F}}^S) + \boxed{MSE(F, \hat{F})}$$

Main loss term:  
Prediction error for the  
final output  $y$

# Knowledge as Loss Function Terms

- One way to define *good* in the loss function is **for the model to comply with domain knowledge**.
- Recall that we can **use important intermediate results as layers** in deep networks. In such cases, we can define *good* as **being able to accurately predict these intermediate results**.



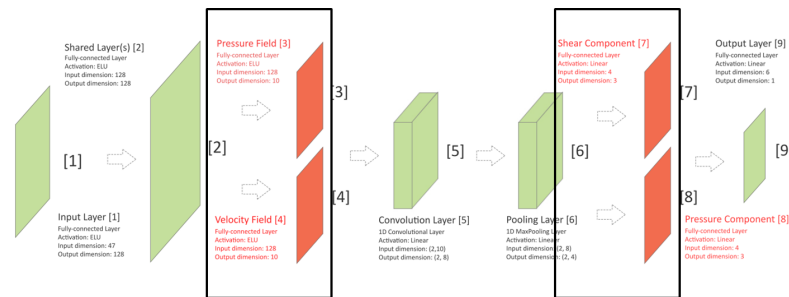
$$Loss_{MSE} = \lambda_P MSE(\mathbf{P}, \hat{\mathbf{P}}) + \lambda_V MSE(\mathbf{V}, \hat{\mathbf{V}}) + \lambda_{FP} MSE(\mathbf{F}^P, \hat{\mathbf{F}}^P) + \lambda_{FS} MSE(\mathbf{F}^S, \hat{\mathbf{F}}^S) + MSE(F, \hat{F})$$

Secondary loss terms:

Prediction errors on **important intermediate outputs**

# Knowledge as Loss Function Terms

- One way to define *good* in the loss function is **for the model to comply with domain knowledge**.
- Recall that we can **use important intermediate results as layers** in deep networks. In such cases, we can define *good* as **being able to accurately predict these intermediate results**.

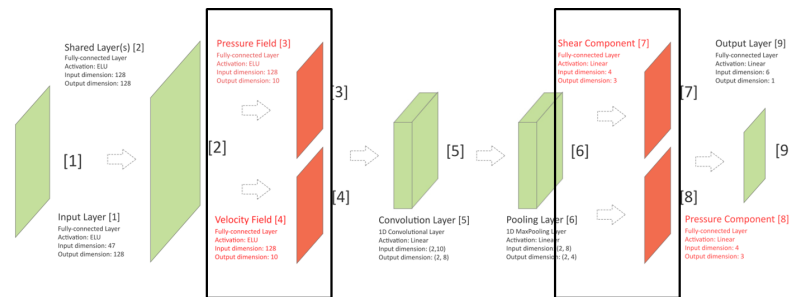


$$Loss_{MSE} = \lambda_P MSE(\mathbf{P}, \hat{\mathbf{P}}) + \lambda_V MSE(\mathbf{V}, \hat{\mathbf{V}}) + \lambda_{FP} MSE(\mathbf{F}^P, \hat{\mathbf{F}}^P) + \lambda_{FS} MSE(\mathbf{F}^S, \hat{\mathbf{F}}^S) + MSE(F, \hat{F})$$

**Weights** of the secondary loss terms, indicating their importance as compared with the main loss term

# Knowledge as Loss Function Terms

- One way to define *good* in the loss function is **for the model to comply with domain knowledge**.
- Recall that we can **use important intermediate results as layers** in deep networks. In such cases, we can define *good* as **being able to accurately predict these intermediate results**.



$$Loss_{MSE} = \lambda_P MSE(\mathbf{P}, \hat{\mathbf{P}}) + \lambda_V MSE(\mathbf{V}, \hat{\mathbf{V}}) + \lambda_{FP} MSE(\mathbf{F}^P, \hat{\mathbf{F}}^P) + \lambda_{FS} MSE(\mathbf{F}^S, \hat{\mathbf{F}}^S) + MSE(F, \hat{F})$$

- The eventual loss function is **all loss terms added together in a weighted fashion**. To train the model is to **minimize the loss function**.

# Knowledge as Loss Function Terms

- One way to define *good* in the loss function is for the model to comply with domain knowledge.
- Recall that we can use important intermediate results as layers in deep networks. In such cases, we can define *good* as being able to accurately predict these intermediate results.
- Another way to define *good* is **being able to comply with domain-specific rules and regulations** (e.g. physics laws, manufacturing specification limits, security protocols, non-discrimination policies, ...).

# Knowledge as Loss Function Terms

- Another way to define *good* is **being able to comply with domain-specific rules and regulations** (e.g. physics laws, manufacturing specification limits, security protocols, non-discrimination policies, ...).
- Case: lake temperature modeling
  - **Input  $x$ : physical variables governing the dynamics of lake temperature**
  - **Output  $y$ : water temperature (at a given depth  $d$  and a certain time  $t$ )**
- The water density  $\rho$  is has **a one-to-one mapping** from temperature  $y$  (for each  $y$ , there is a unique  $\rho$  corresponding to it; and vice-versa), which means **by predicting  $y$ , we can implicitly predict  $\rho$ .**
- This comes in handy as there is **a physics rule concerning the relationship between density  $\rho$  and the depth  $d$**  that can help us define **a knowledge-guided loss function** for  $y$ . Specifically:
  - **At a given time  $t$ , the large  $d$  is, the larger  $\rho$  is** (namely: the deeper the water, the denser).

$$\rho[d_1, t] - \rho[d_2, t] \leq 0 \quad \text{if } d_1 < d_2$$

# Knowledge as Loss Function Terms

- The water density  $\rho$  has a one-to-one mapping from temperature  $y$  (for each  $y$ , there is a unique  $\rho$  corresponding to it; and vice-versa), which means by predicting  $y$ , we can implicitly predict  $\rho$ .
- This comes in handy as there is a physics rule concerning the relationship between density  $\rho$  and the depth  $d$  that can help us define a knowledge-guided loss function for  $y$ . Specifically:
  - At a given time  $t$ , the larger  $d$  is, the larger  $\rho$  is
- Based on this rule, we can define the following loss term (in which  $\hat{\rho}$  is the predicted  $\rho$  value, derived from the predicted  $y$ ):

$$\rho[d_1, t] - \rho[d_2, t] \leq 0 \quad \text{if } d_1 < d_2$$

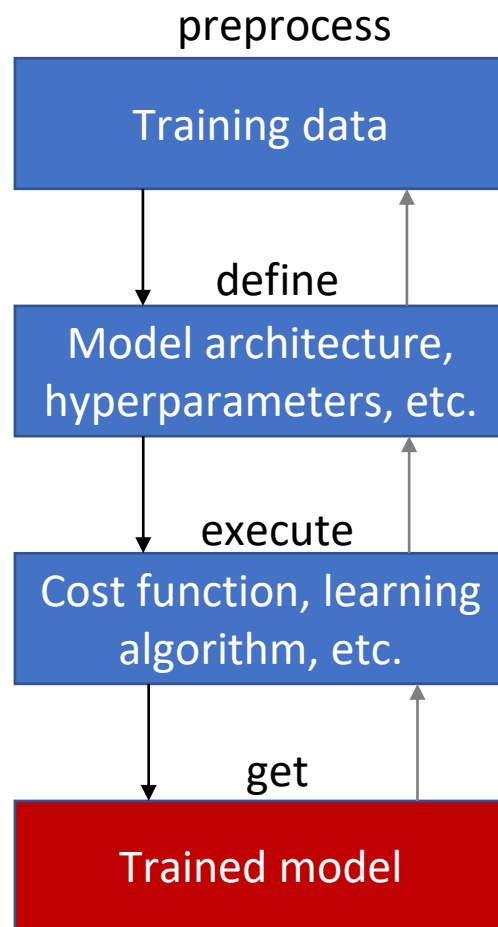
$$\text{ReLU}(\hat{\rho}[d_i, t] - \hat{\rho}[d_{i+1}, t])$$

- $\hat{\rho}[d_i, t] - \hat{\rho}[d_{i+1}, t]$  is the difference between the predicted densities from a smaller to a large depth. **It is expected to be non-positive by the physics rule.**
- ReLU is a commonly used function in neural networks that is defined as  $\text{ReLU}(z) = \begin{cases} z, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$
- By applying ReLU, we note that **the loss term is 0 if  $\hat{\rho}[d_i, t] - \hat{\rho}[d_{i+1}, t]$  is non-positive as expected; otherwise, the loss term is positive.**
- To minimize the loss term, the model must make the best effort to **keep  $\hat{\rho}[d_i, t] - \hat{\rho}[d_{i+1}, t]$  non-positive, as it should do!**



# Data-driven Model Training Pipeline

**Data-driven training pipeline for an ML model**



While the training can be done automatically, you can always **evaluate and refine** the trained model **after the initial training is complete!**

**You can evaluate and refine the model with knowledge!**

# Model Evaluation

- Like what we did in the cost function, when we evaluate a trained model, we also have multiple ways to define *good*, and again, one of them is **being able to comply with domain-specific rules and regulations**.
- Consider our previous lake temperature modeling case, where we had the physics rule:

$$\rho[d_1, t] - \rho[d_2, t] \leq 0 \quad \text{if } d_1 < d_2$$

- For model evaluation, we can use **the number of times this rule is violated** as an **evaluation metric**.

# Model Enhancement

- For a trained model, we can **enhance it with expert knowledge**. Specifically, we can **identify the examples that have been incorrectly predicted** by the model, and **present them to a domain expert**, who can **tell us why** they have been incorrectly predicted.
  - If the expert actually **agrees with the model's prediction**, then this means **the labels of the examples in the dataset (which serve as groundtruth) are wrong**. We should then **correct that label in the dataset**.
  - If the expert determines that **an inevitable external factor (that cannot be handled by the model itself)** has caused the false prediction, we then have a better understanding of the **intrinsic limitations of the model**. We should then **inform the users of these limitations**.
  - If the expert determines that the model **fails to take into account certain factors (that it theoretically can)**, then we know **there are flaws in our model training**. We should **make improvements and retrain the model according**.

# Outline

- Motivation for Knowledge-guided Data Science
- Knowledge Sources and Representations
- Fusing Knowledge with Data
- **Conclusion**

# Conclusion

- Domain knowledge is a powerful tool to help address challenges in data-driven methods, such as small data, lack of explainability, etc.
- On the other hand, data-driven methods can help exploit overly simplified domain knowledge to better address complex real-world scenarios.
- It is thus desirable to fuse the two together. This can be done in any step of the data-driven machine learning pipeline!

# Thanks!

Knowledge-guide Data Science

Shen Liang

diiP Summer School (dSDS 2024), Jun. 12, 2024. UPCité, Paris, France.