# Expressive Power of Graph Neural Networks

Pablo Barceló[1], Egor Kostylev[2], **Mikaël Monet**[1], Jorge Pérez[1], Juan Reutter[1]

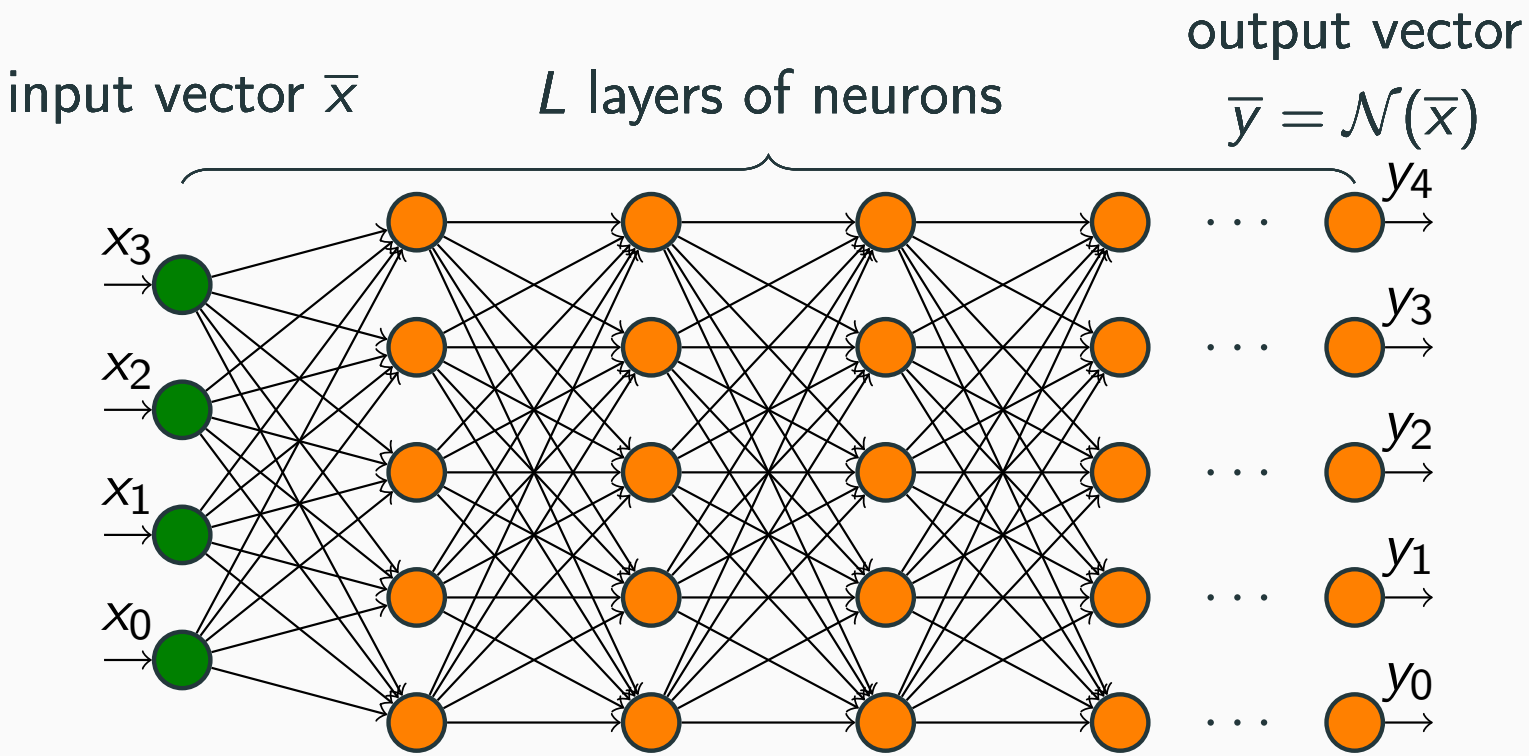
July 26th, 2019

[1]Millennium Institute for Foundational Research on Data, Chile

[2]Oxford University, UK

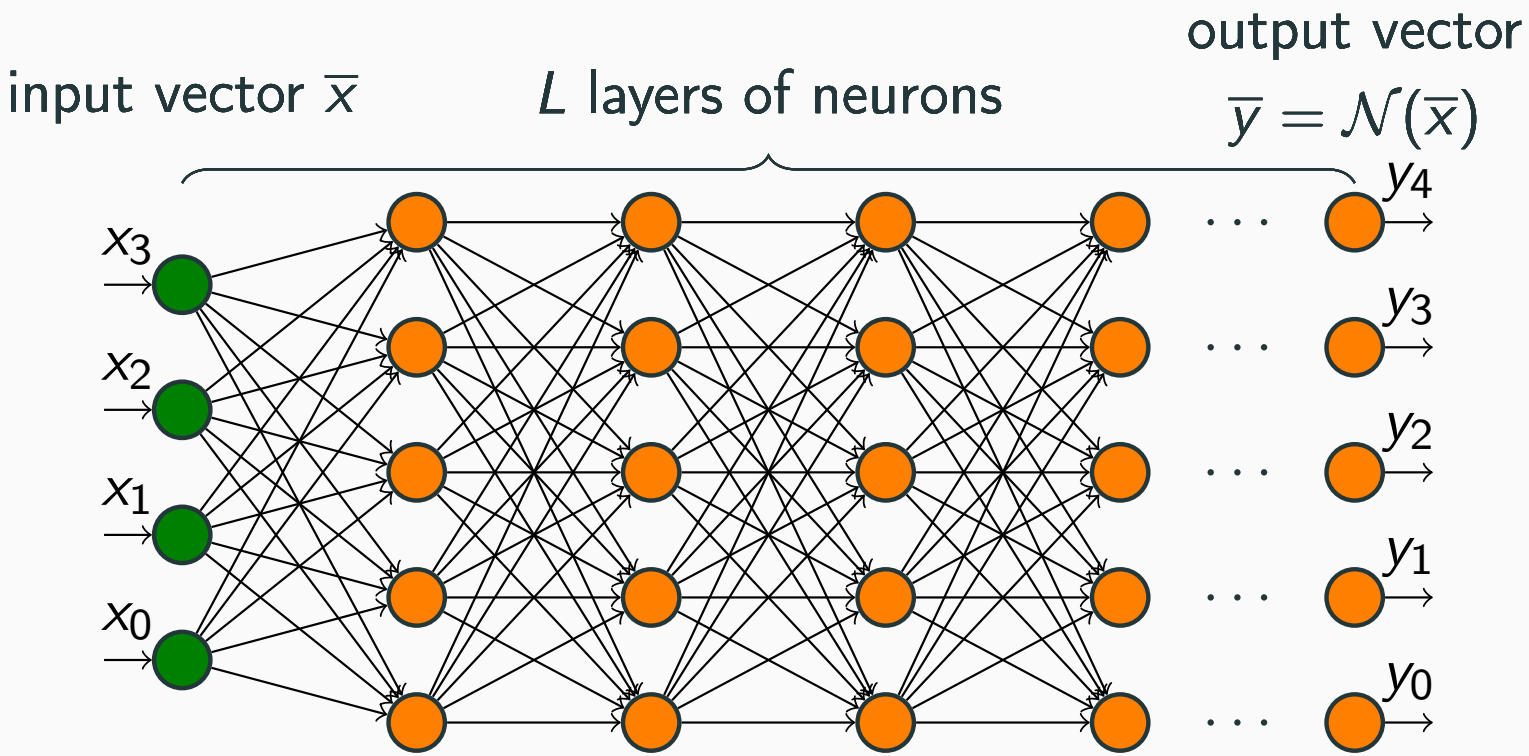
# Motivation: why *graph* neural networks?

# Neural Networks (NNs)




A fully connected neural network $\mathcal{N}$.

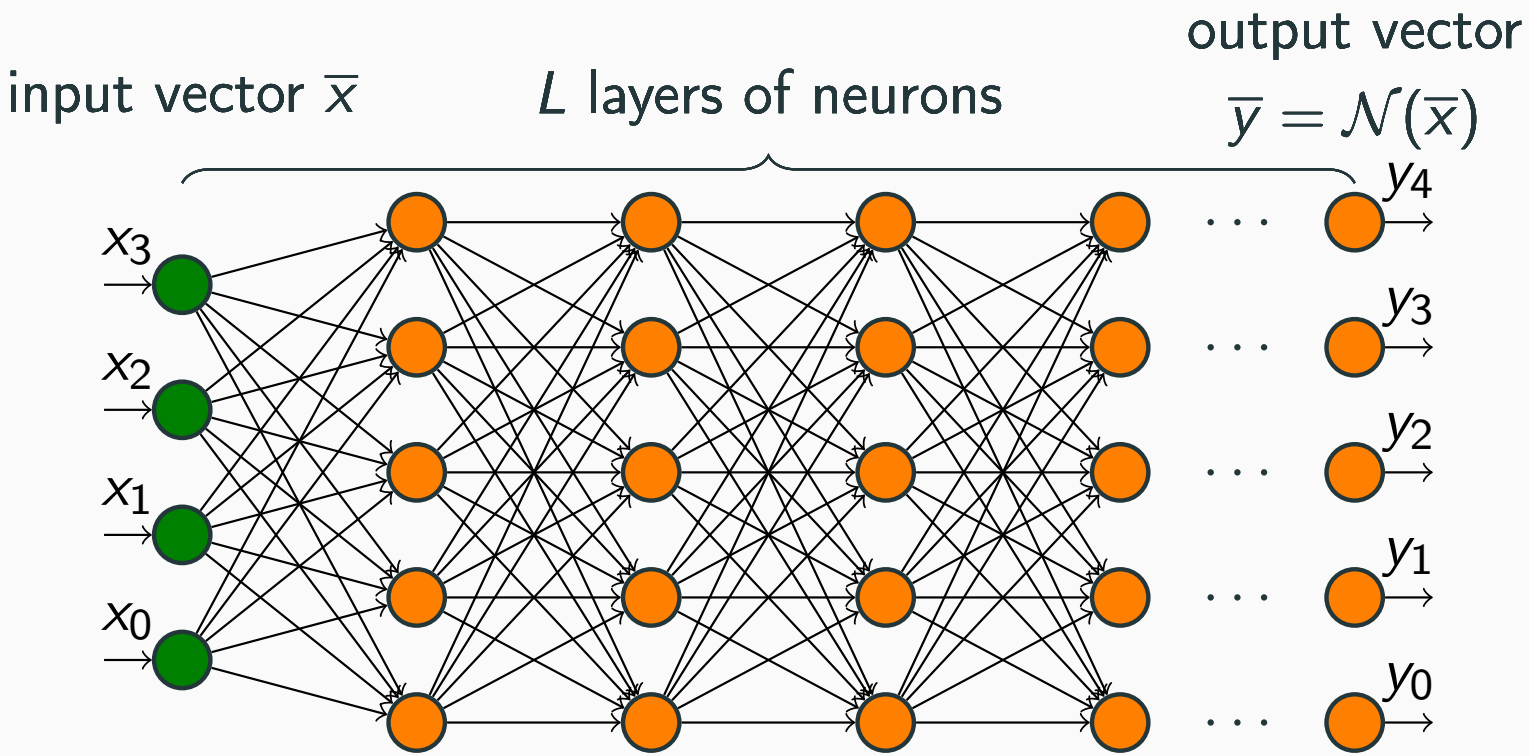
# Neural Networks (NNs)




A fully connected neural network $\mathcal{N}$.

- Weight $w_{n' \to n}$ between two consecutive neurons

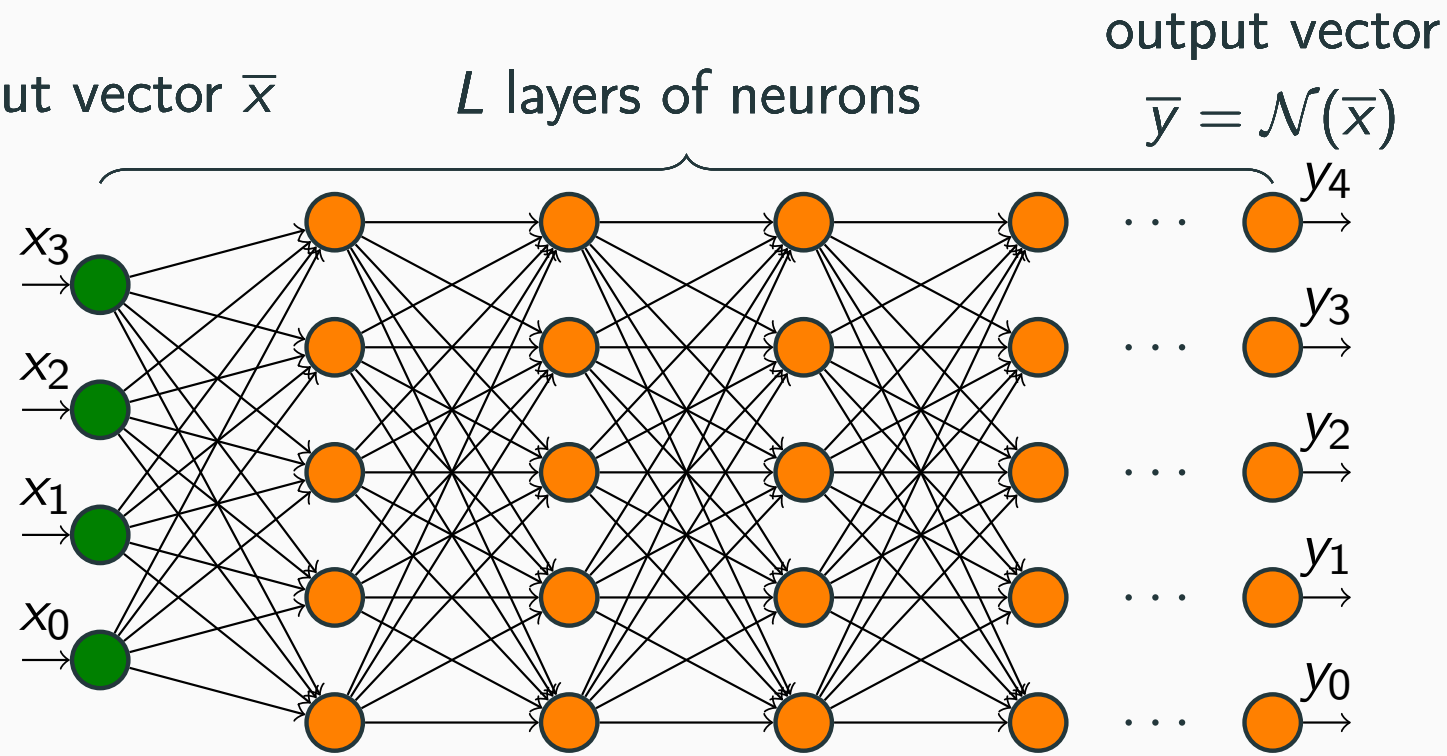
# Neural Networks (NNs)




A fully connected neural network $\mathcal{N}$.

- Weight $w_{n' \to n}$ between two consecutive neurons
- Compute left to right $\lambda(n) \stackrel{\text{def}}{=} f(\sum w_{n' \to n} \times \lambda(n'))$

# Neural Networks (NNs)




A fully connected neural network $\mathcal{N}$.

- Weight $w_{n' \to n}$ between two consecutive neurons
- Compute left to right $\lambda(n) \stackrel{\text{def}}{=} f(\sum w_{n' \to n} \times \lambda(n'))$
- Goal: find the weights that "solve" your problem (classification, clustering, regression, etc.)

# Finding the weights

- Goal: find the weights that "solve" your problem

$\rightarrow$ minimize $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where $g$ is what you want to learn

# Finding the weights

- Goal: find the weights that "solve" your problem

$\rightarrow$ minimize $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where $g$ is what you want to learn

$\rightarrow$ use backpropagation algorithm

# Finding the weights

- Goal: find the weights that "solve" your problem

$\rightarrow$ minimize $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where $g$ is what you want to learn

$\rightarrow$ use backpropagation algorithm

- **Problem**: when a layer has many neurons, there are a lot of weights...

# Finding the weights

- Goal: find the weights that "solve" your problem

$\rightarrow$ minimize $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where $g$ is what you want to learn

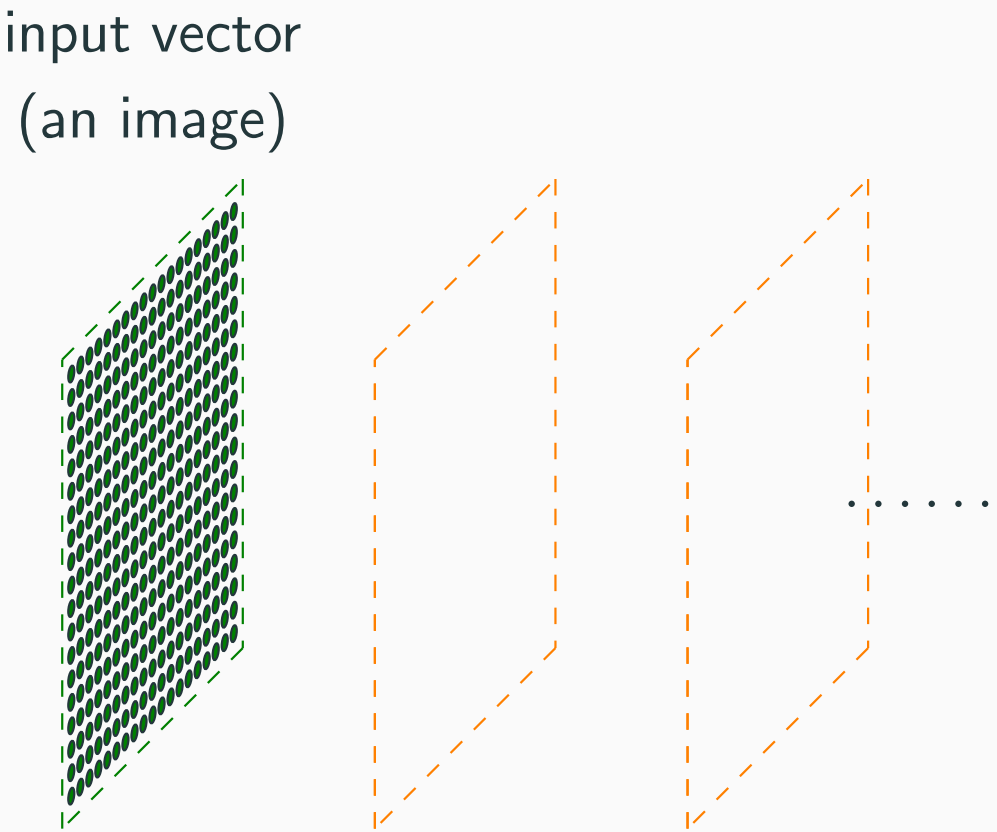
$\rightarrow$ use backpropagation algorithm

- **Problem**: when a layer has many neurons, there are a lot of weights...

$\rightarrow$ example: input is a $250 \times 250$ pixels image, and we want to build a fully connected NN with 500 neurons per layer

$\rightarrow$ between each consecutive layers we have $250 \times 250 \times 500 = 31,250,000$ weights

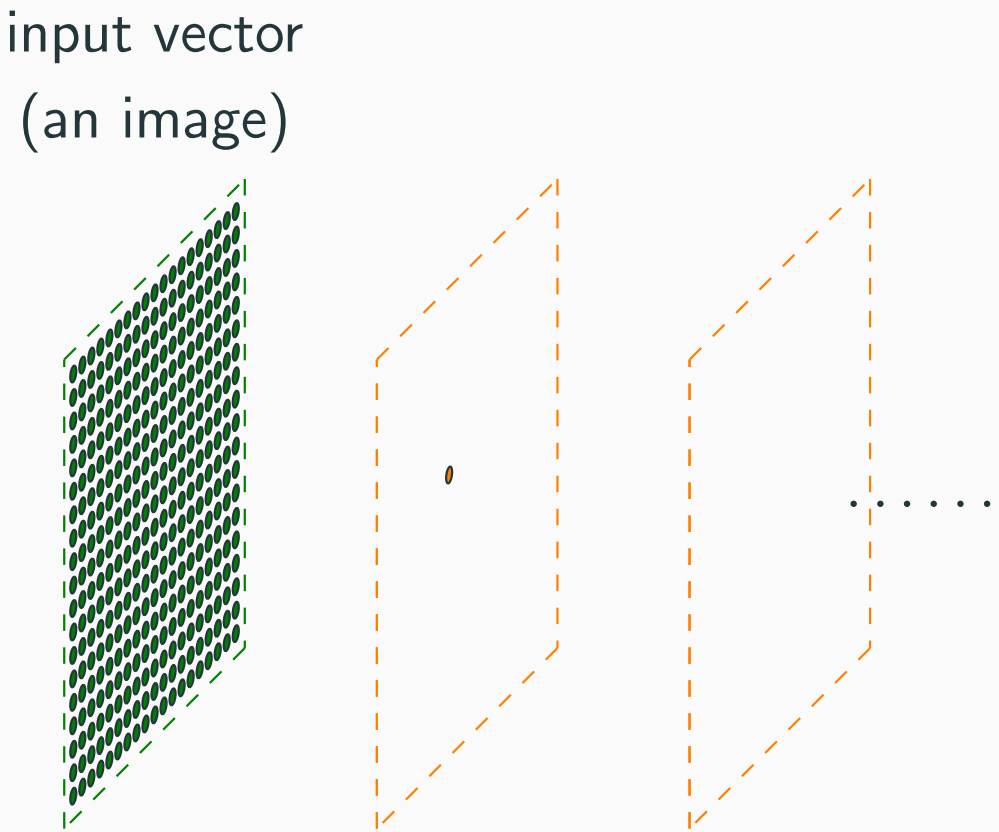
# Convolutional Neural Networks




A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

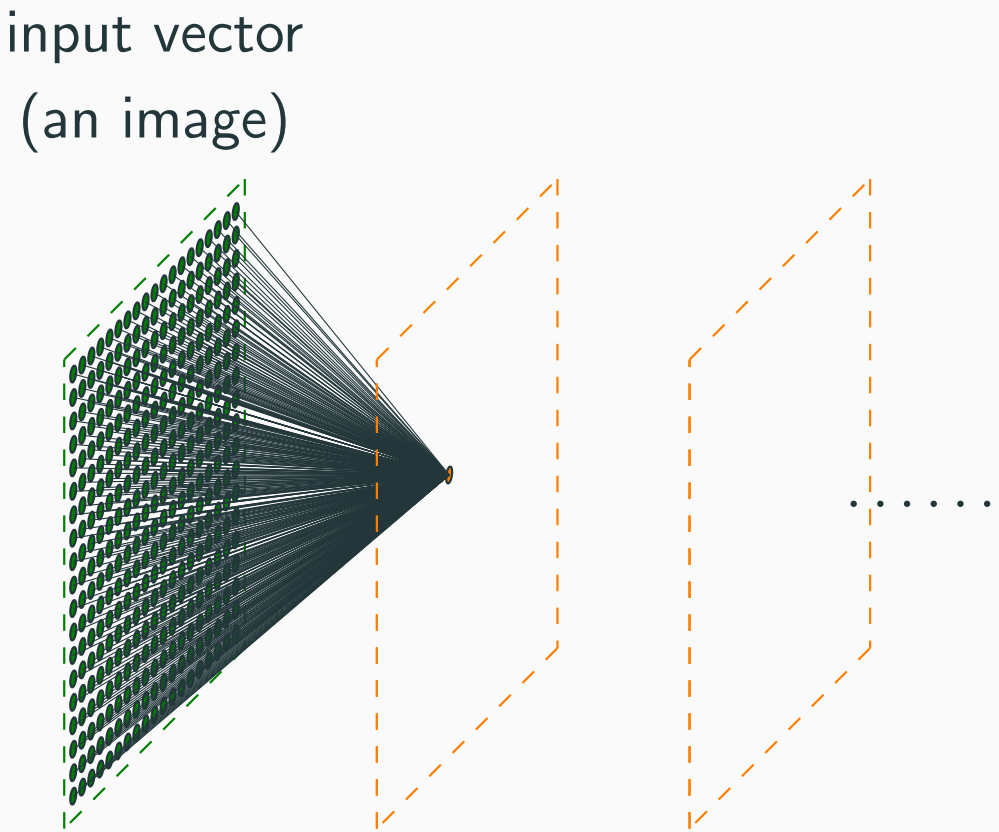
# Convolutional Neural Networks




A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

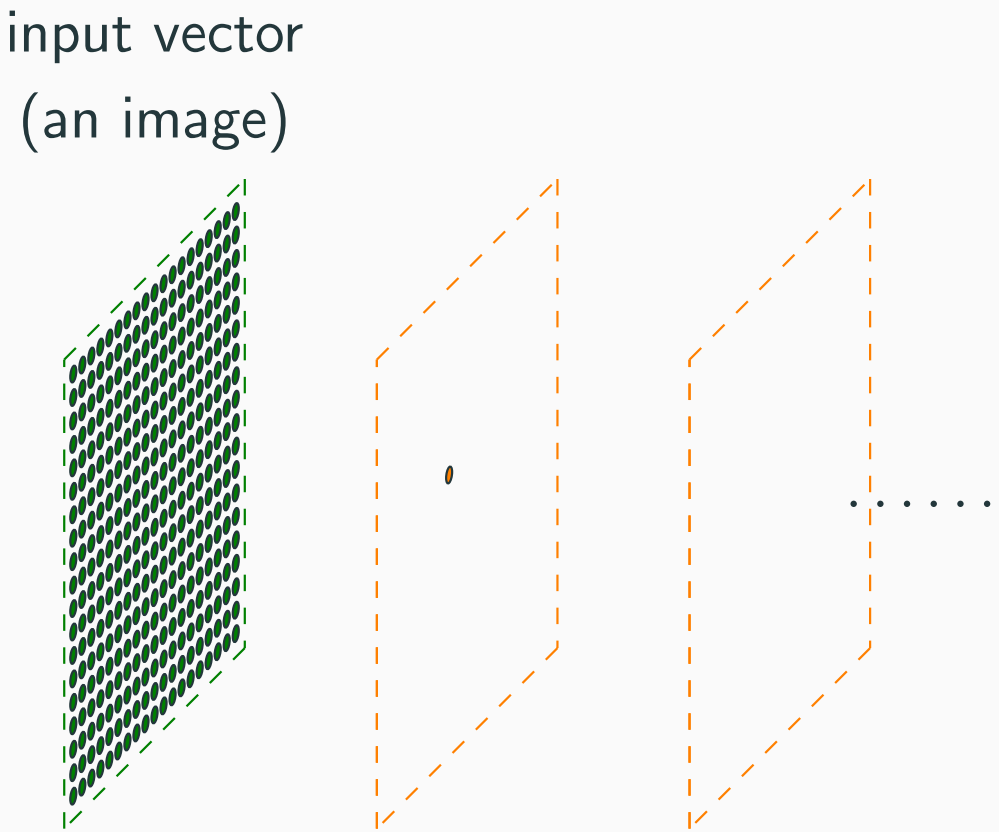
# Convolutional Neural Networks




A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

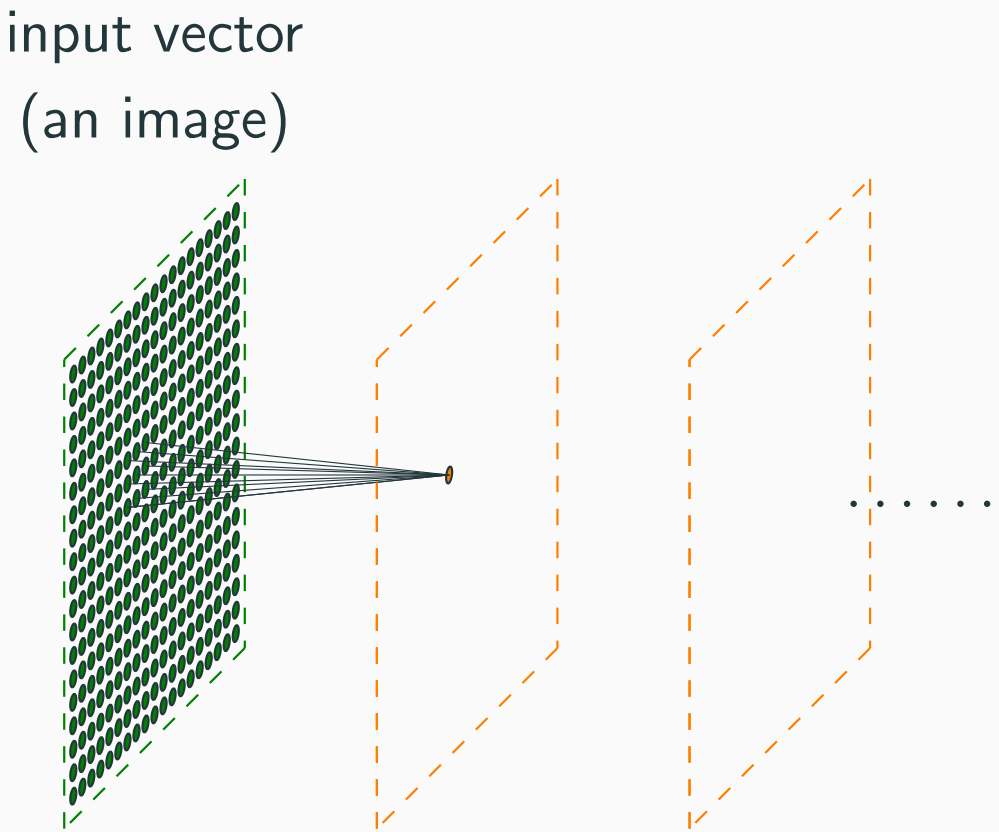
# Convolutional Neural Networks




A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

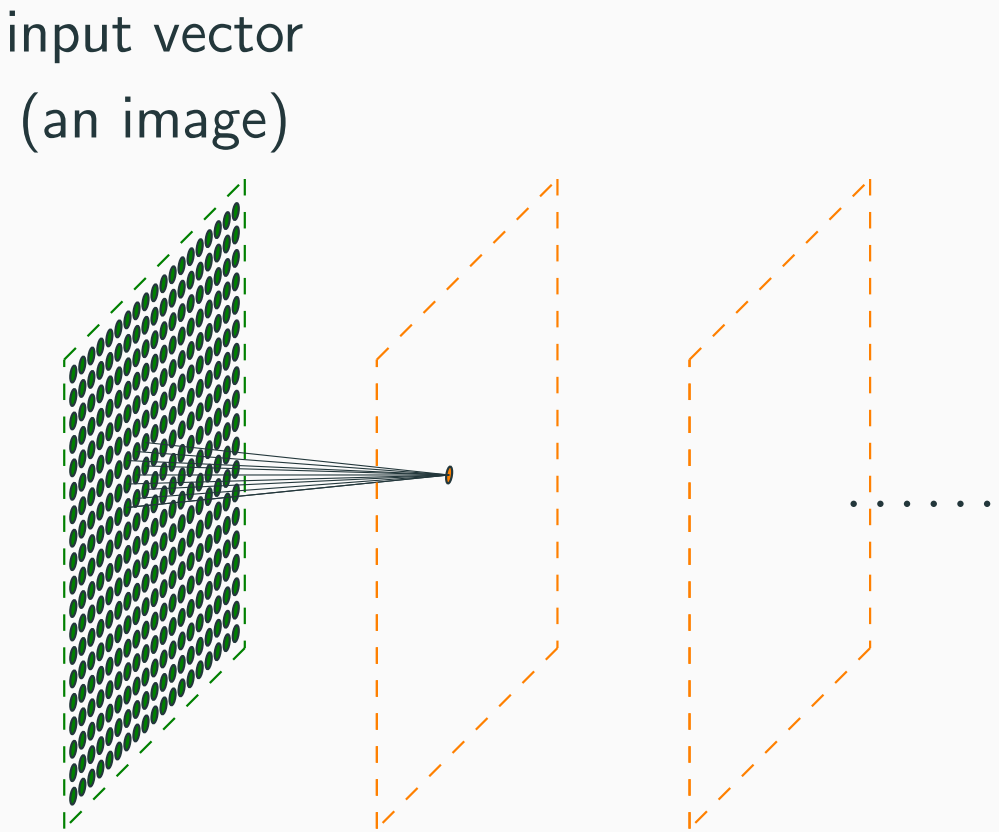
# Convolutional Neural Networks




A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

# Convolutional Neural Networks

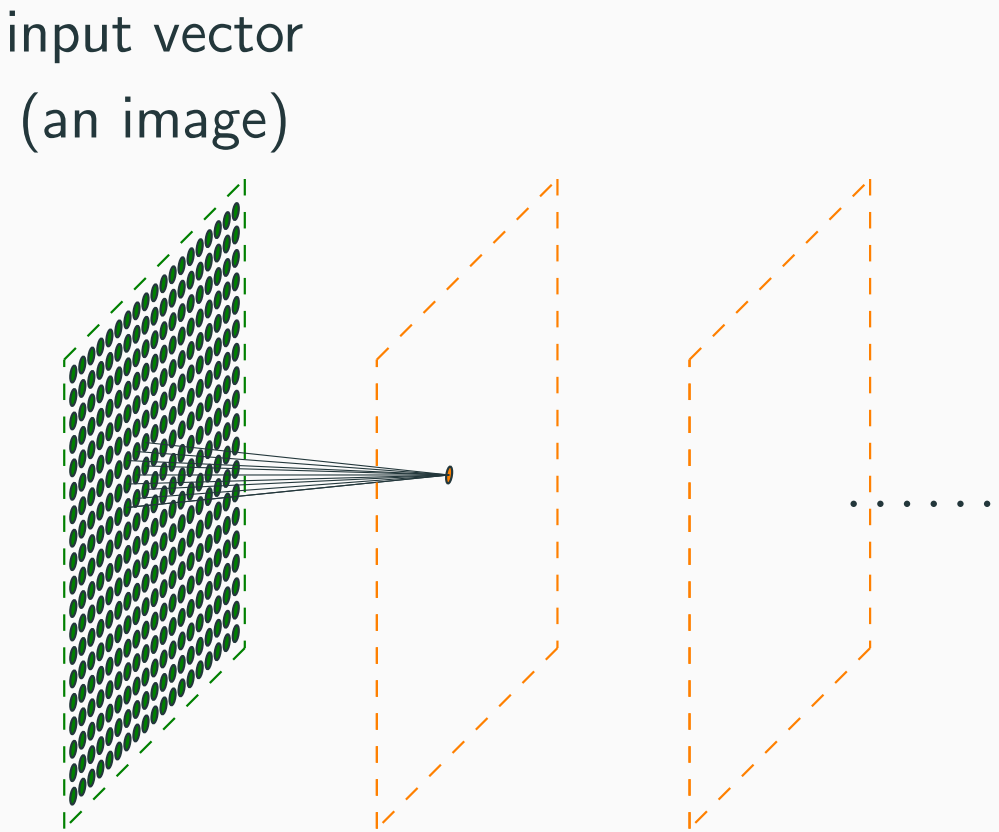



A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

$\rightarrow$ fewer weights to learn (e.g, $500 * 9 = 4,500$ for the first layer)

# Convolutional Neural Networks



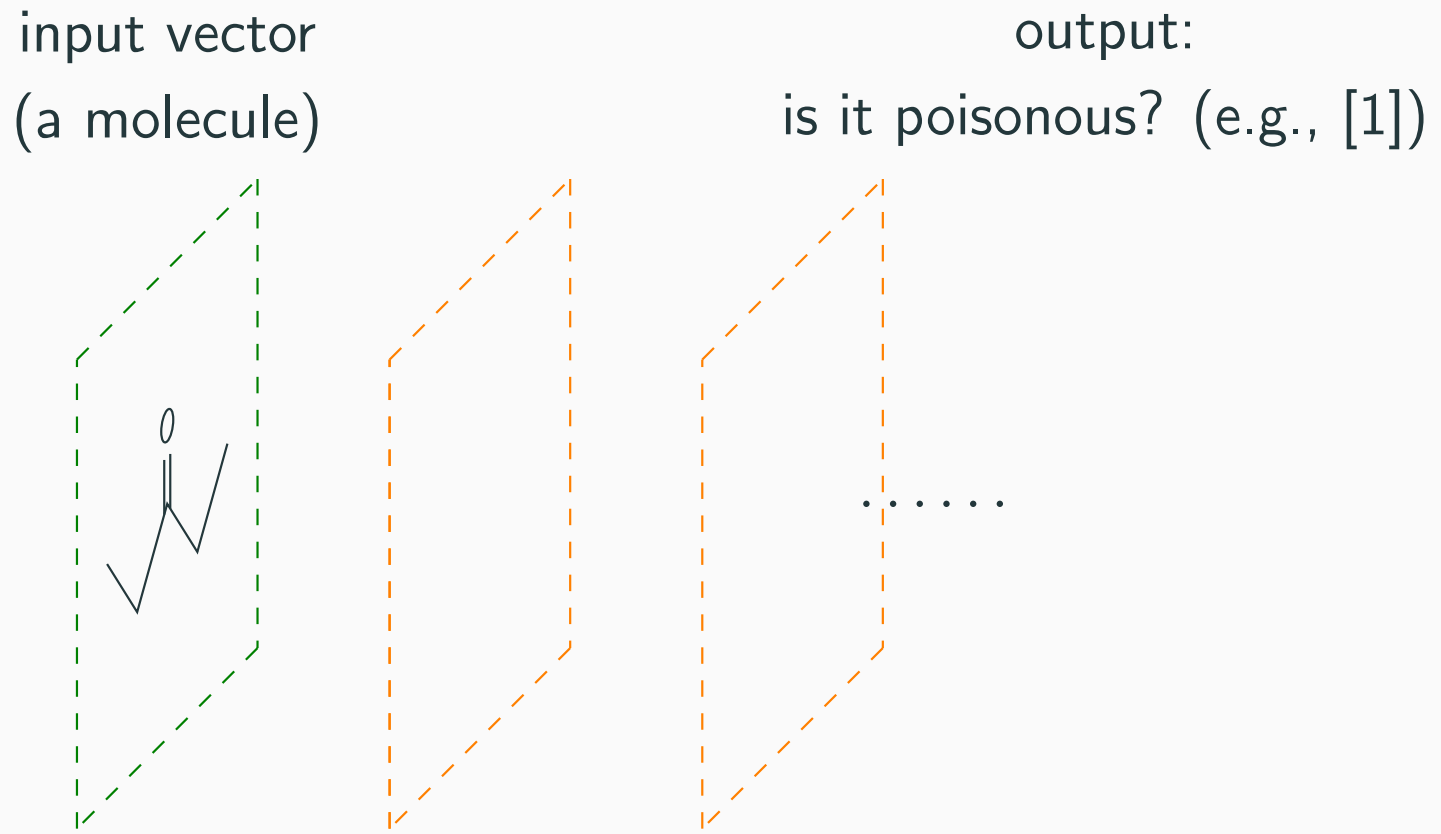

A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

$\rightarrow$ fewer weights to learn (e.g, $500 * 9 = 4,500$ for the first layer)

$\rightarrow$ other advantage: recognize patterns that are local

# Graph Neural Networks (GNNs)

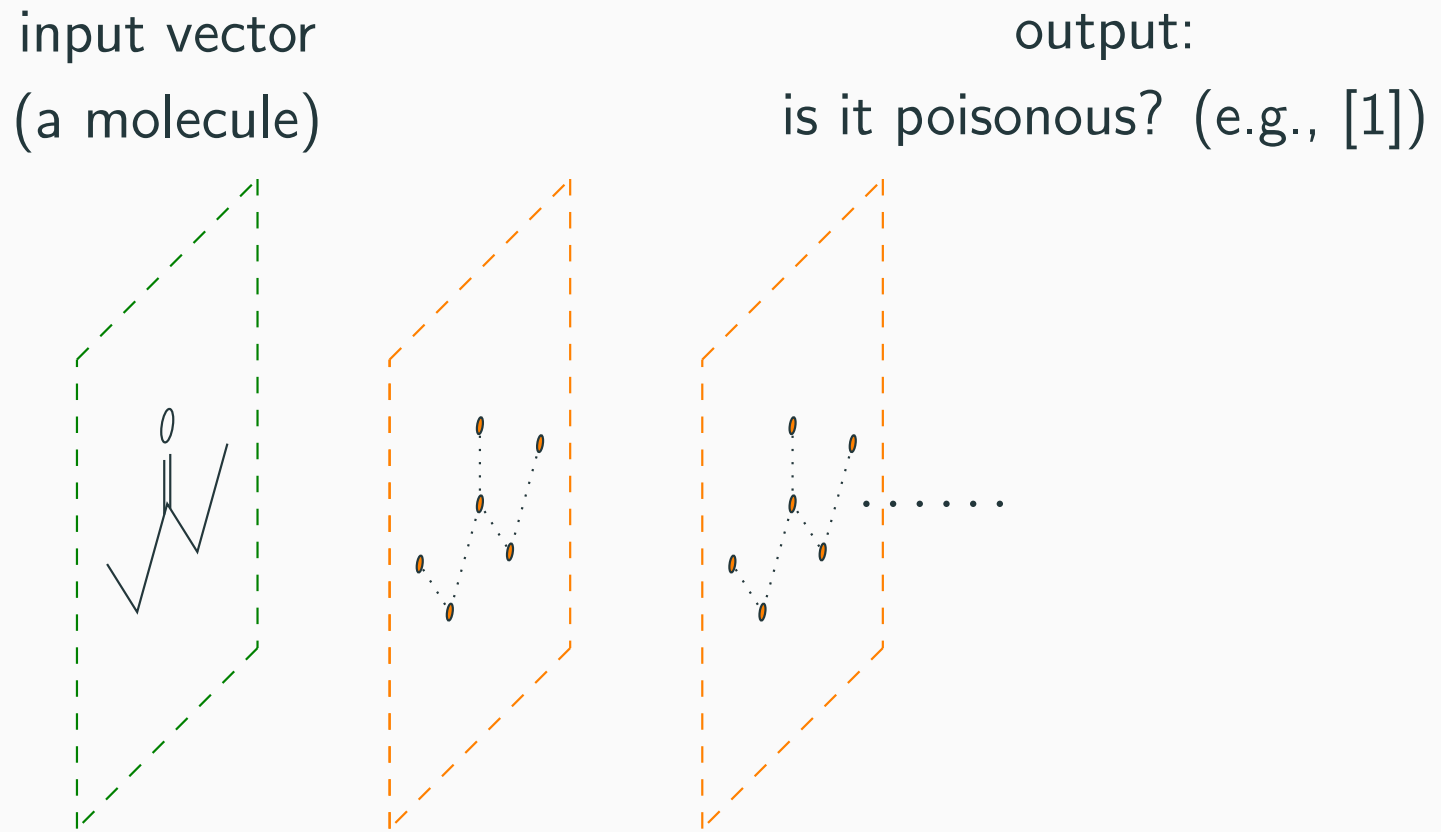



A graph neural network.

- **Idea**: use the structure of the data (here, a grid)

→ GNNs generalize this idea to allow *any* graph as input (in particular, not constrained with fixed-size inputs anymore)

# Graph Neural Networks (GNNs)




A graph neural network.

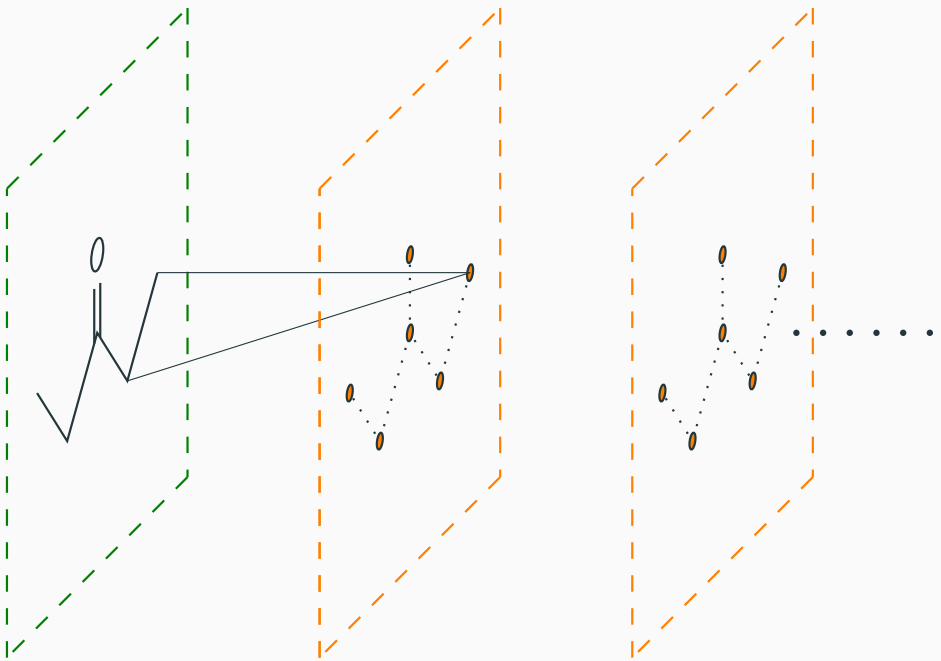
- **Idea**: use the structure of the data (here, a grid)

→ GNNs generalize this idea to allow *any* graph as input (in particular, not constrained with fixed-size inputs anymore)

# Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous? (e.g., [1])



A graph neural network.

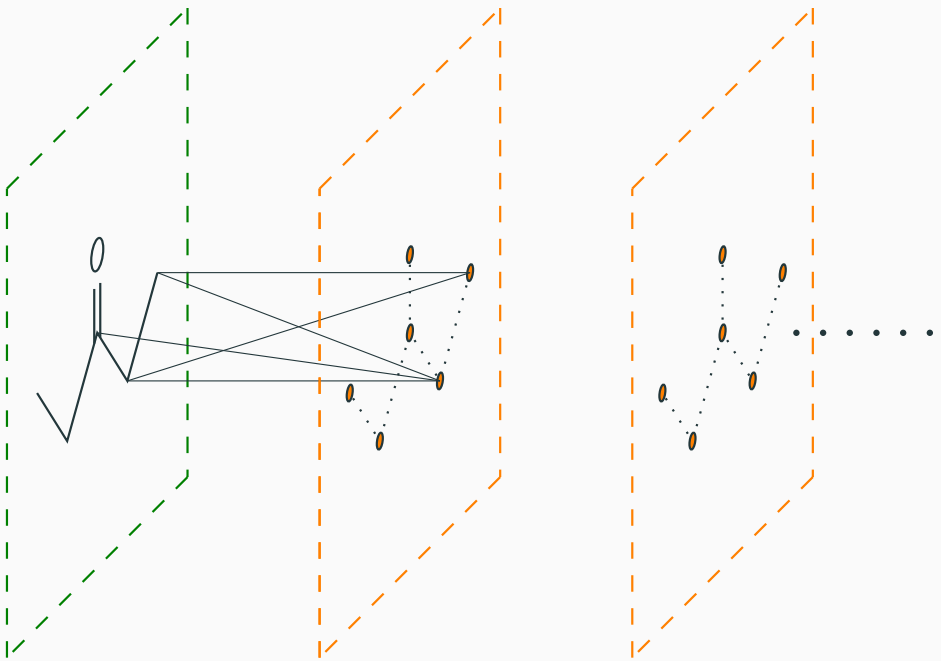
- **Idea**: use the structure of the data (here, a grid)

→ GNNs generalize this idea to allow *any* graph as input (in particular, not constrained with fixed-size inputs anymore)

# Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous? (e.g., [1])



A graph neural network.

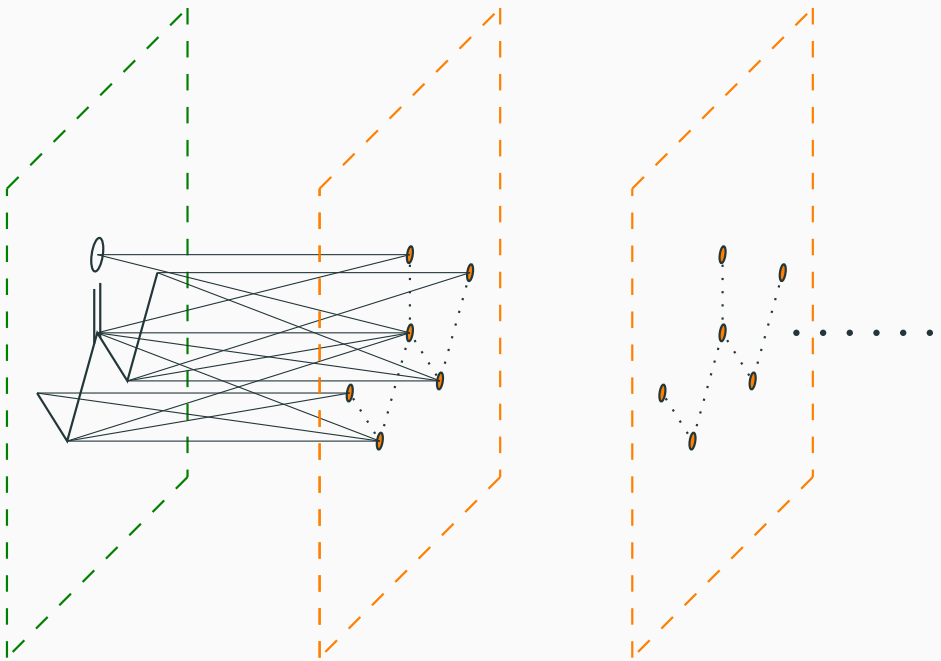
- **Idea**: use the structure of the data (here, a grid)

→ GNNs generalize this idea to allow *any* graph as input (in particular, not constrained with fixed-size inputs anymore)

# Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous? (e.g., [1])



A graph neural network.

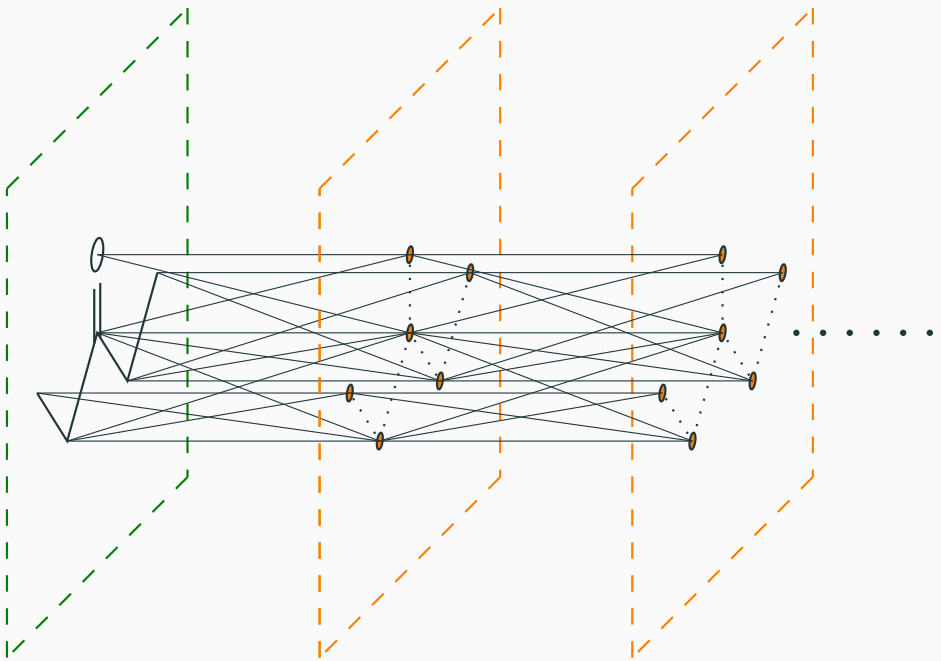
- **Idea**: use the structure of the data (here, a grid)

→ GNNs generalize this idea to allow *any* graph as input (in particular, not constrained with fixed-size inputs anymore)

# Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous? (e.g., [1])



A graph neural network.

- **Idea**: use the structure of the data (here, a grid)

→ GNNs generalize this idea to allow *any* graph as input (in particular, not constrained with fixed-size inputs anymore)

**Goal of the project**: what can we do with graph neural networks? (from a *theoretical* perspective)

Thanks for your attention!

# Bibliography I


David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams.
**Convolutional networks on graphs for learning molecular fingerprints.**
In *Advances in neural information processing systems*, pages 2224–2232, 2015.