6.7900 Machine Learning (Fall 2024)

Lecture 12: a brief intro to neural network models

(supporting slides)

Outline

- Modeling sequences with state space models: Recurrent Neural Networks (**RNNs**)
- Modeling images (and related data): Convolutional Neural Networks (**CNNs**)
- Modeling graphs, data on graphs: Graph Neural Networks (**GNNs**)
- Modeling sequences, images, etc with **Transformers**

Outline

- Modeling sequences with state space models: Recurrent Neural Networks (**RNNs**)
- Modeling images (and related data): Convolutional Neural Networks (**CNNs**)
- Modeling graphs, data on graphs: Graph Neural Networks (**GNNs**)
- Modeling sequences, images, etc with **Transformers** (next lecture)

Images, fully connected layers

- Transforming high resolution images using fully connected layers would be computational expensive, statistically not efficient (lacking translational “equivariance”)

$\mathscr{W} \times$

$d^2 x d^2$

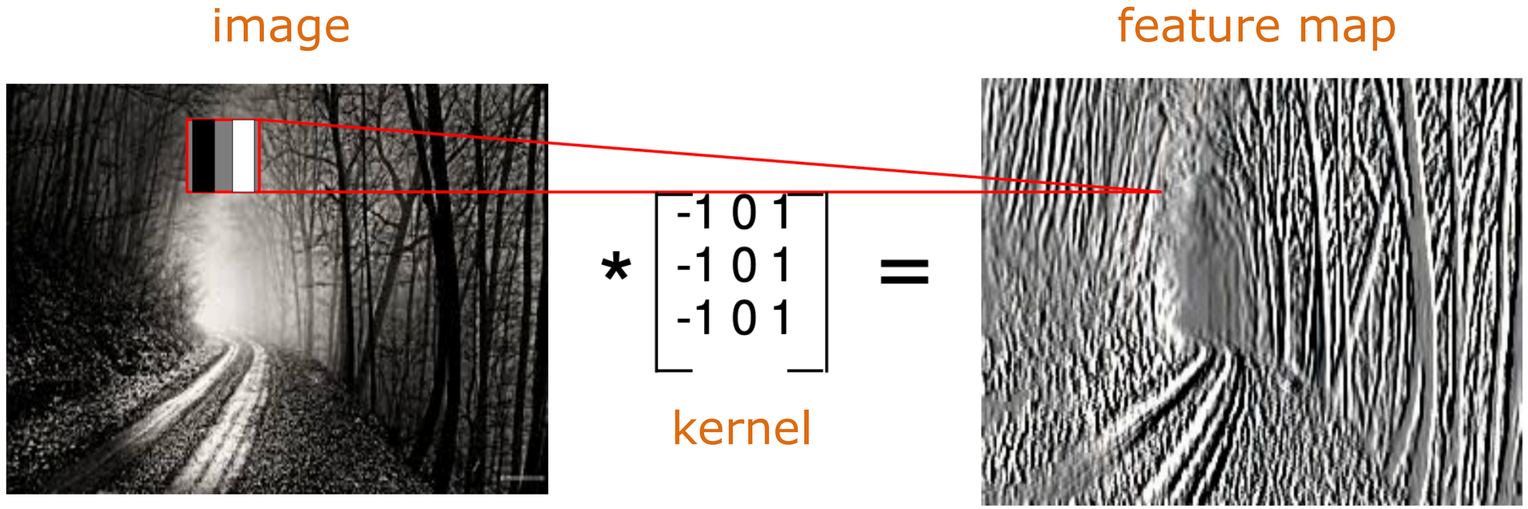
dxd

$=$

dxd

Image features with equivariance

- If we instead run a little feature detector over the image, we only need a few parameters (local linear + non-linear detector)
- The resulting "feature map" also shifts with the image (= translational equivariance)

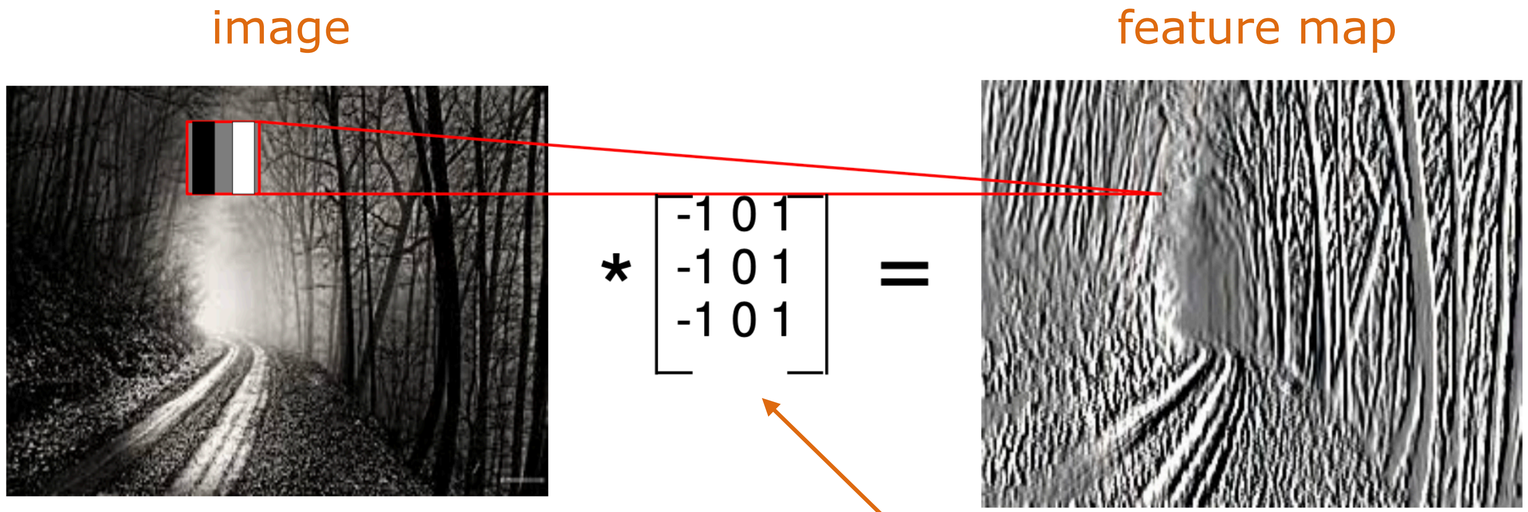


- (we will need to extract many such feature maps from the same image)

[Figure from: Ranzato, 2015]

Image features with equivariance

- A small feature detector run through an image = convolutional layer (can be easily parallelized)



$$h^n_{i,j} = \max\left\{0, \sum_{k=0}^{H-1}\sum_{l=0}^{W-1} h^{n-1}_{i+k,j+l} w_{kl}\right\}$$

feature map

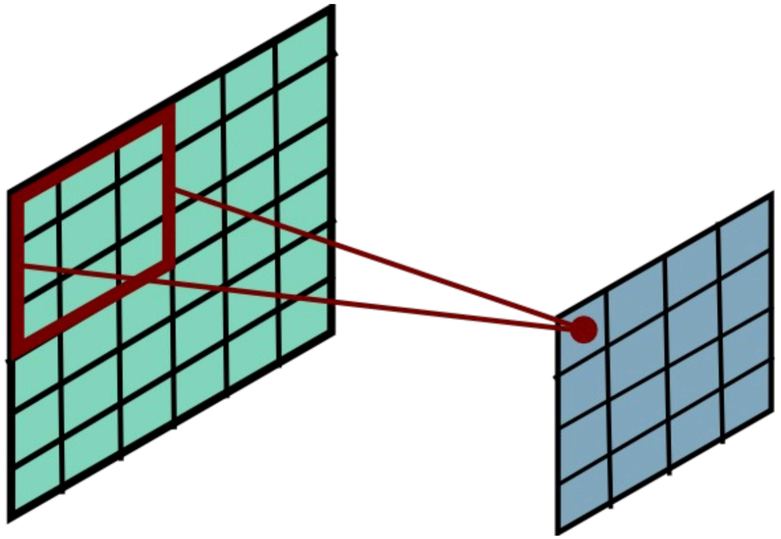
image or previous feature map

filter/kernel of height H and width W

[Figure from: Ranzato, 2015]

Images and features

- A small feature detector run through an image defines a “convolutional layer”
- A larger “stride” (step size when moving the detector across) yields a smaller feature map

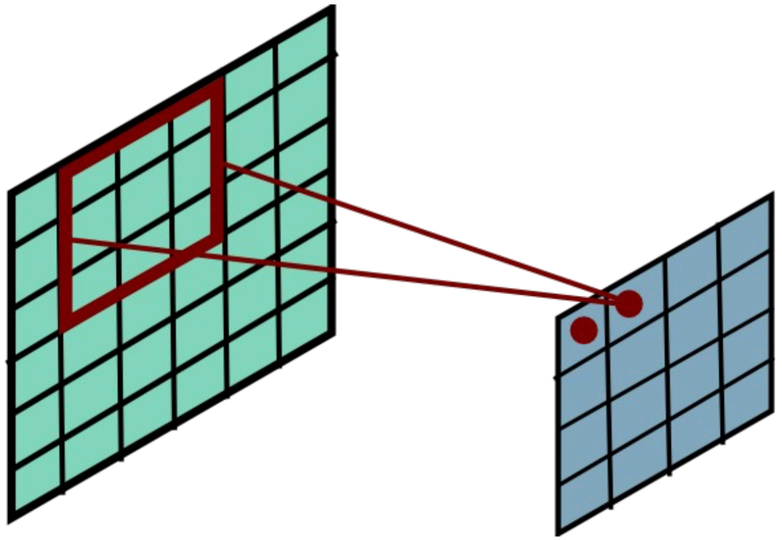


convolution stride = 1

[Figure from: Ranzato, 2015]

Images and features

- A small feature detector run through an image defines a “convolutional layer”
- A larger “stride” (step size when moving the detector across) yields a smaller feature map

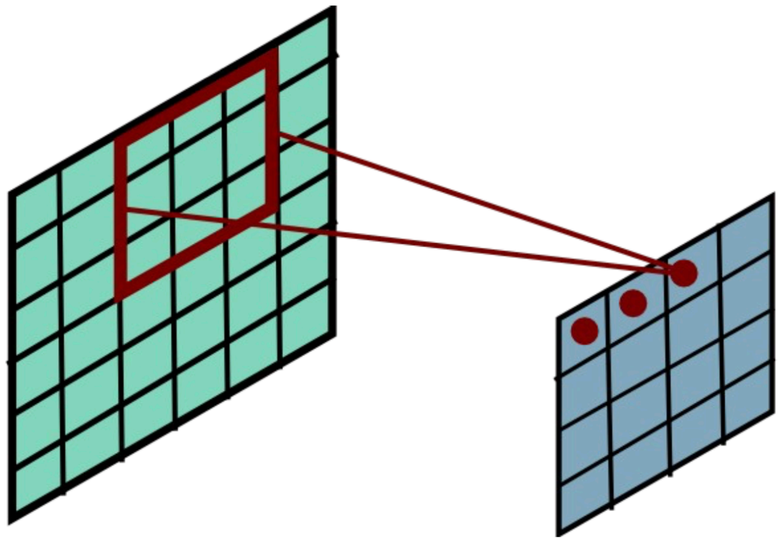


convolution stride = 1

[Figure from: Ranzato, 2015]

Images and features

- A small feature detector run through an image defines a “convolutional layer”
- A larger “stride” (step size when moving the detector across) yields a smaller feature map

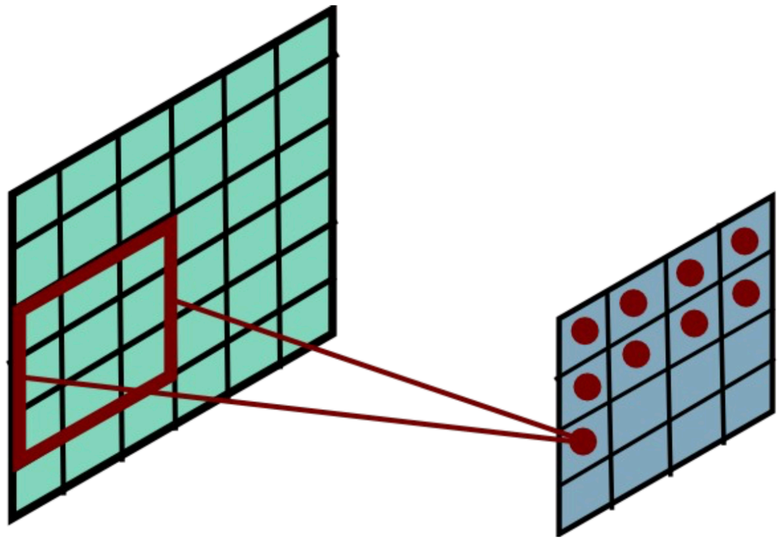


convolution stride = 1

[Figure from: Ranzato, 2015]

Images and features

- A small feature detector run through an image defines a “convolutional layer”
- A larger “stride” (step size when moving the detector across) yields a smaller feature map

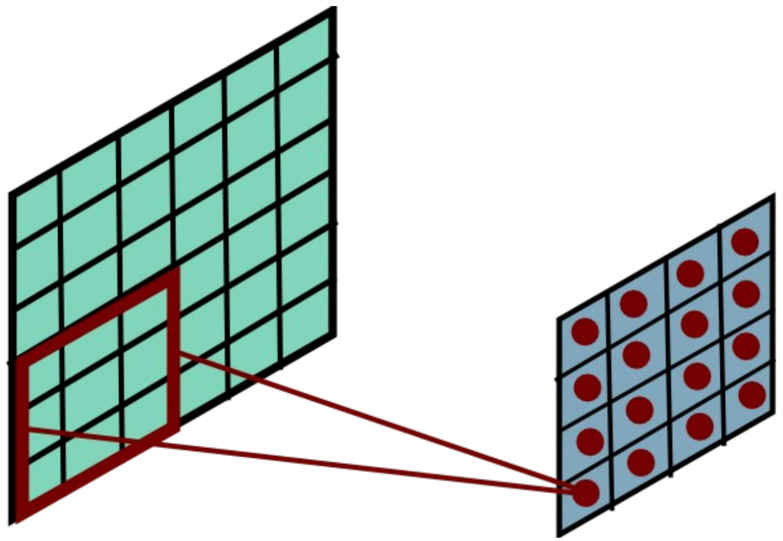


convolution stride = 1

[Figure from: Ranzato, 2015]

Images and features

- A small feature detector run through an image defines a “convolutional layer”
- A larger “stride” (step size when moving the detector across) yields a smaller feature map

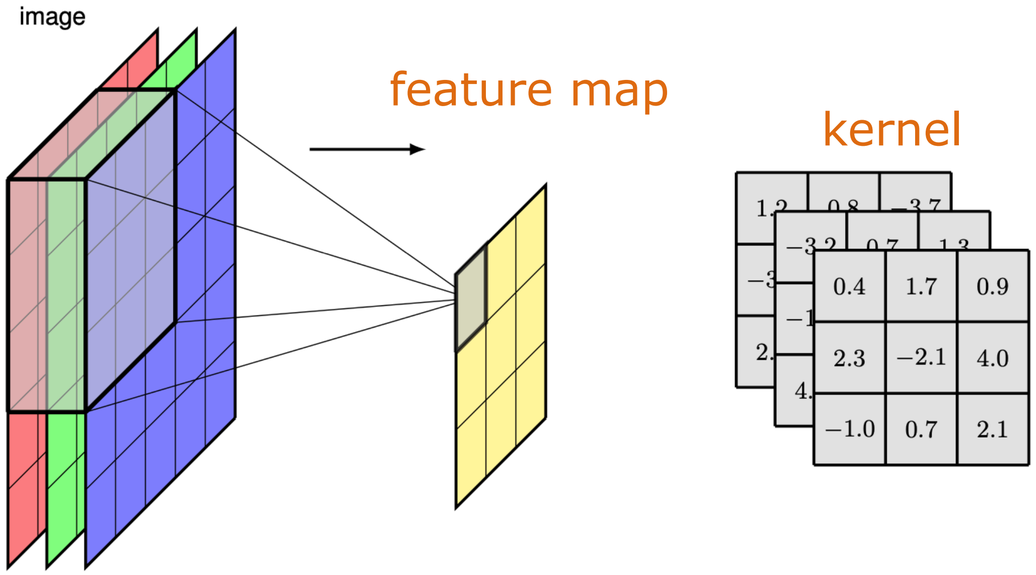


convolution stride = 1

[Figure from: Ranzato, 2015]

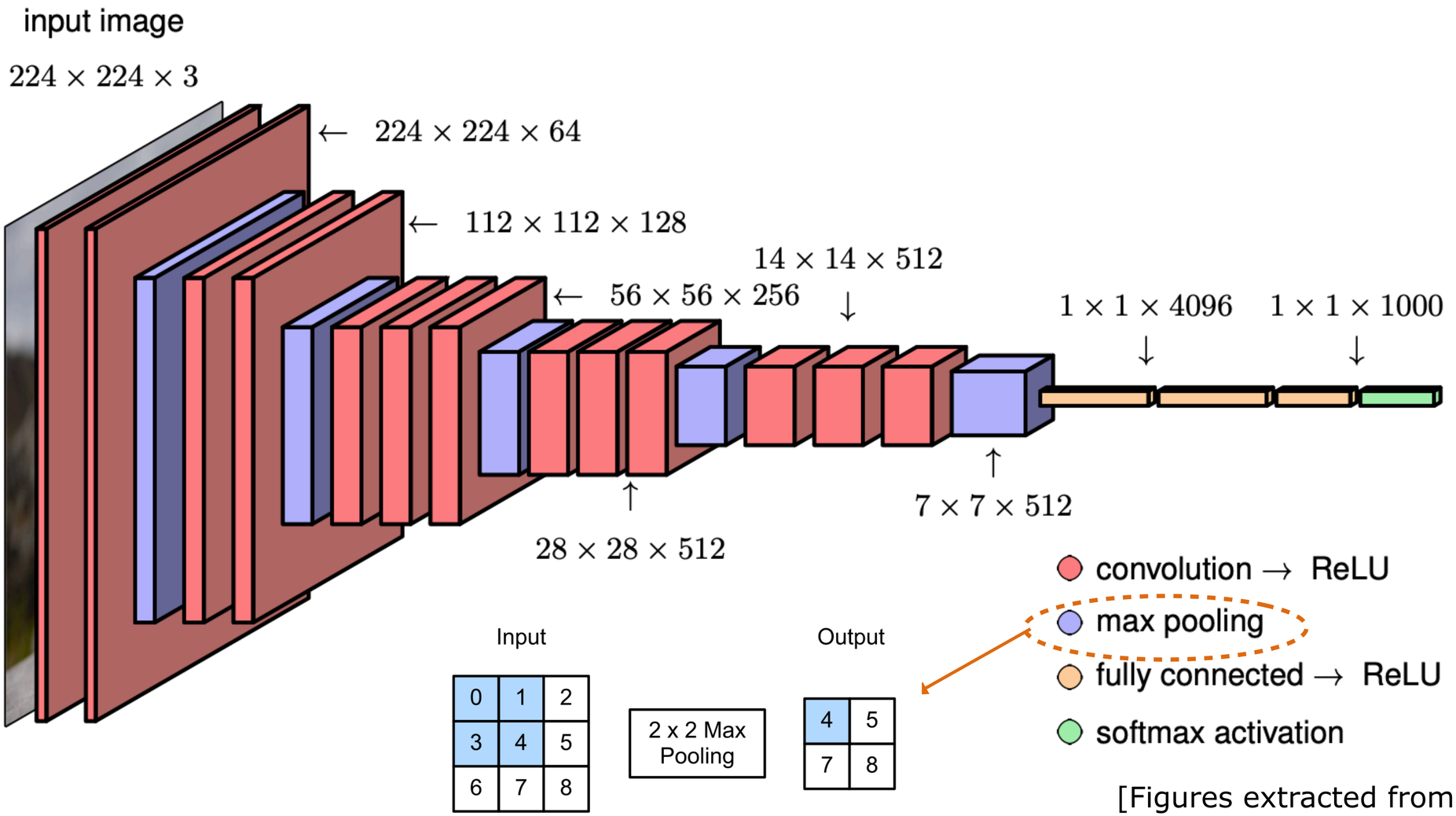
Images and features

- A small feature detector run through an image defines a “convolutional layer”
- A larger “stride” (step size when moving the detector across) yields a smaller feature map



An example of a multi-dimensional convolution (e.g., across feature maps) [figure from Bishop]

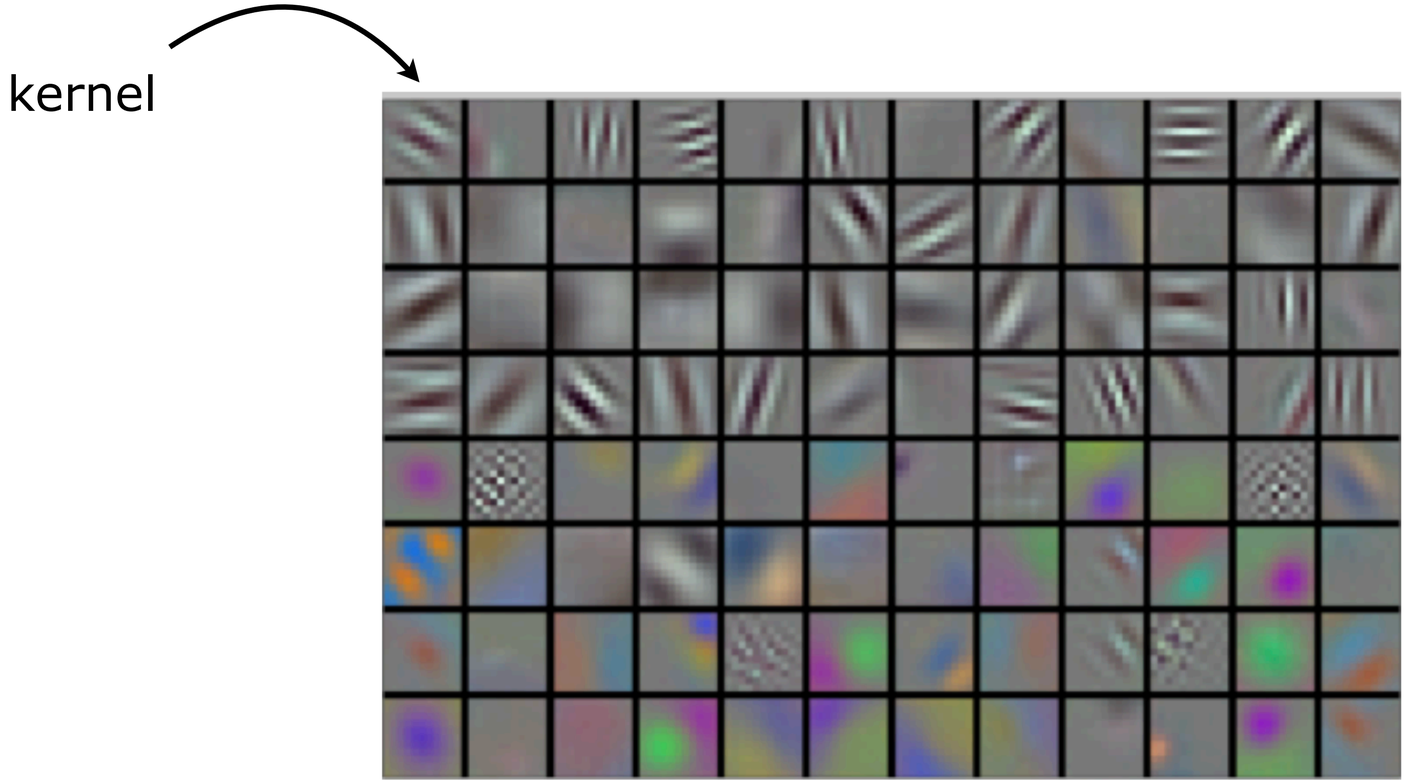
A simple convolutional neural network (CNN)

- We can extract multiple features from images in each stage and then build on those features (feature maps of feature maps etc).
- 

[Figures extracted from Murphy & Bishop]

What kind of features (kernels) do CNNs learn?

- Layer 1 features or kernels (from a different architecture)

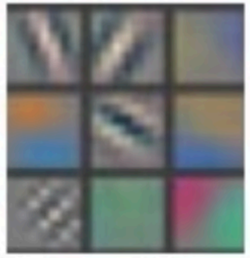


[Zeiler and Fergus, ECCV 2014]

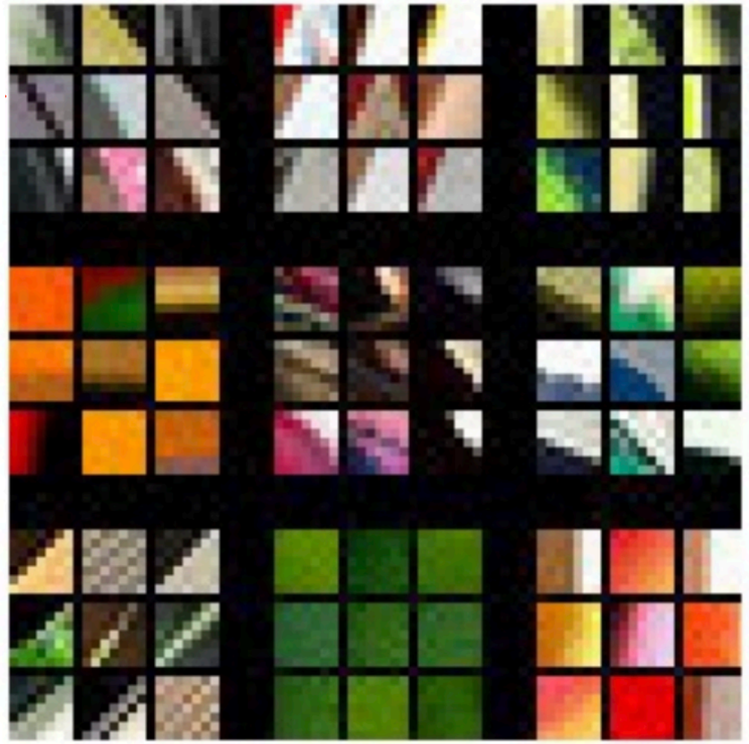
What kind of features (kernels) do CNNs learn?

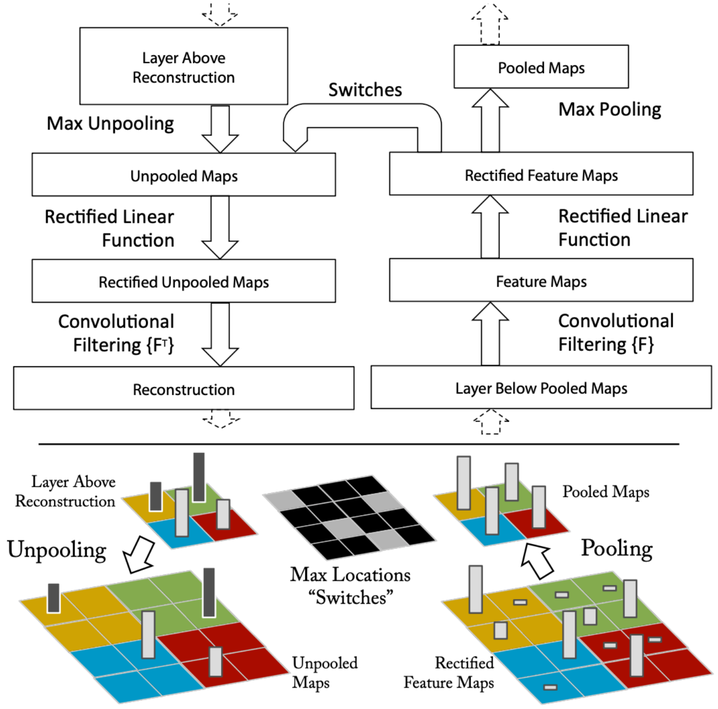
- Layer 1 features

kernel



top 9 image patches responsible for the highest feature activation

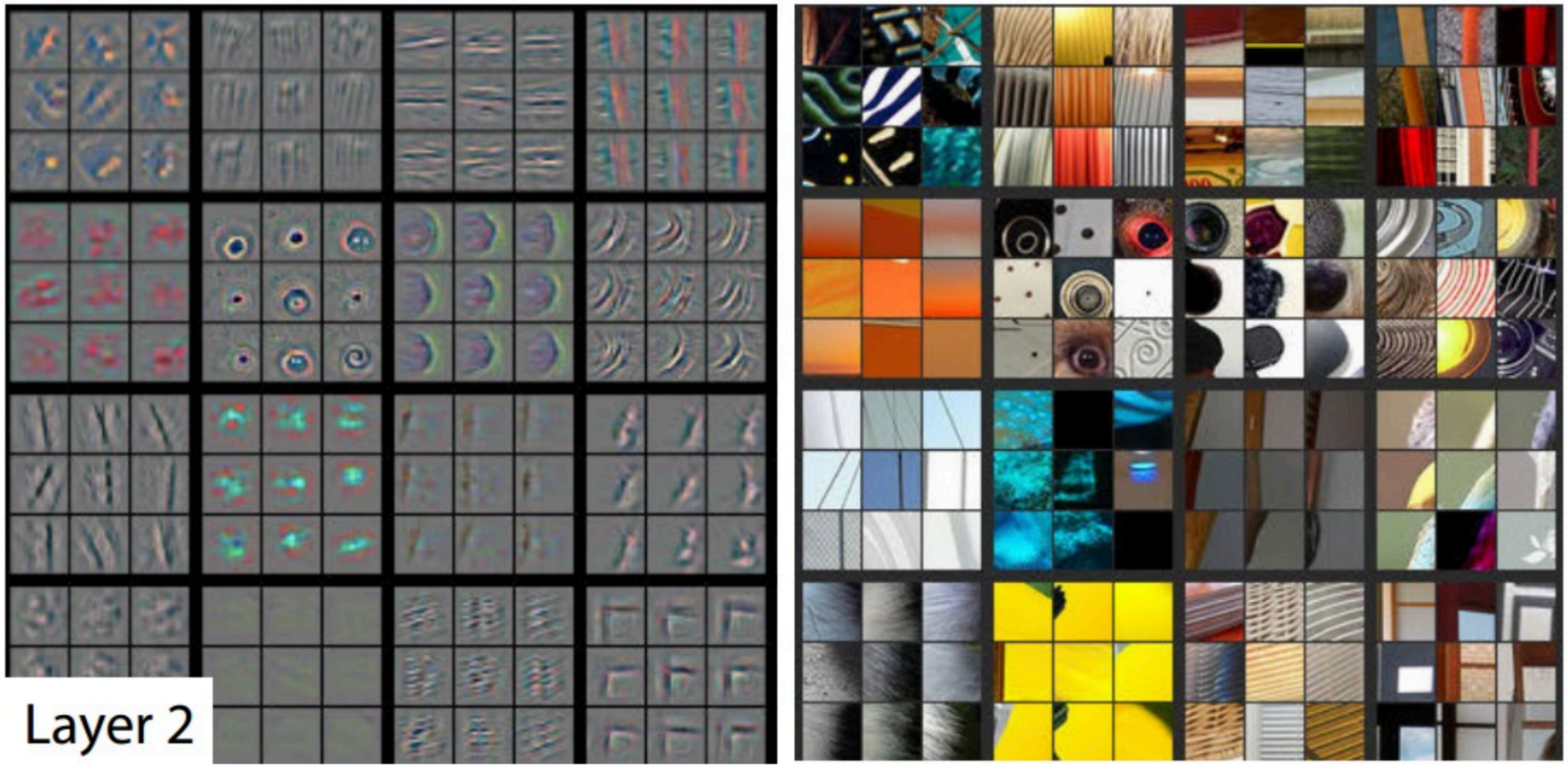




[Zeiler and Fergus, ECCV 2014]

What kind of features (kernels) do CNNs learn?

- Layer 2 features

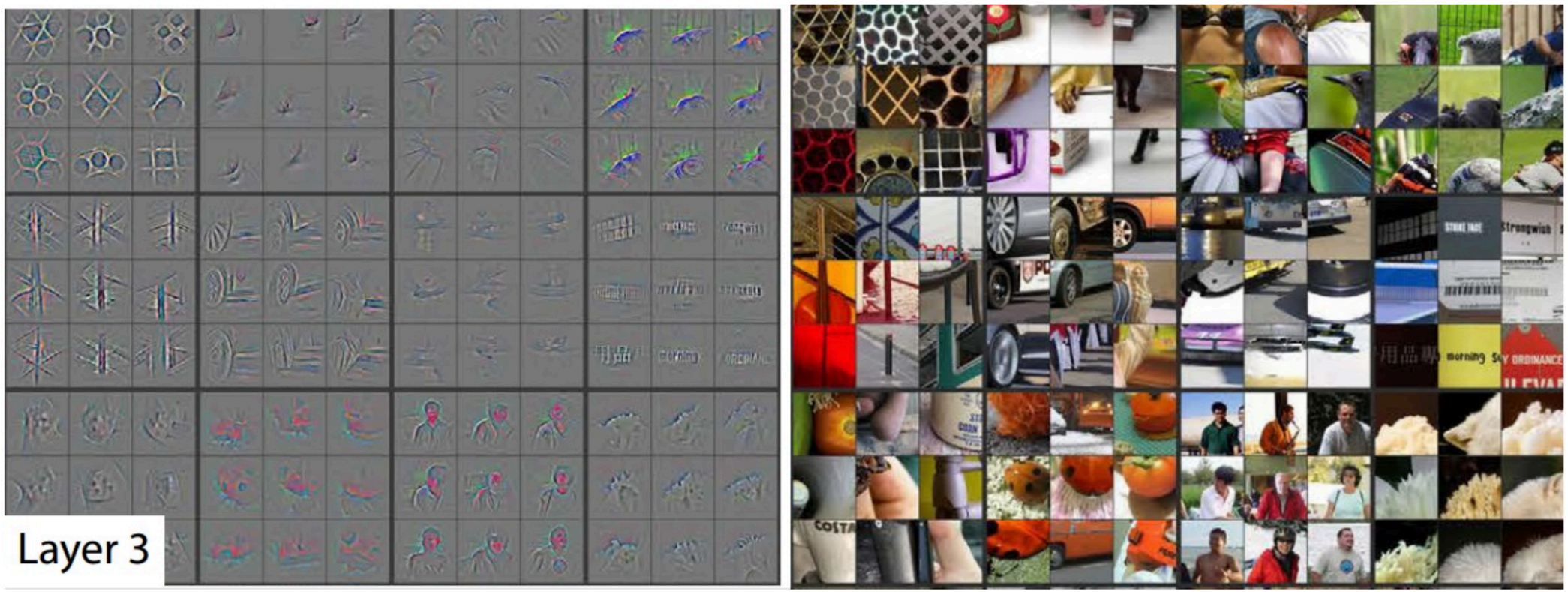


- (note that these features operate on layer 1 features and are thus more complex in terms of their relation to the images; the underlying image patches are also now larger)

[Zeiler and Fergus, ECCV 2014]

What kind of features (kernels) do CNNs learn?

- Layer 3 features

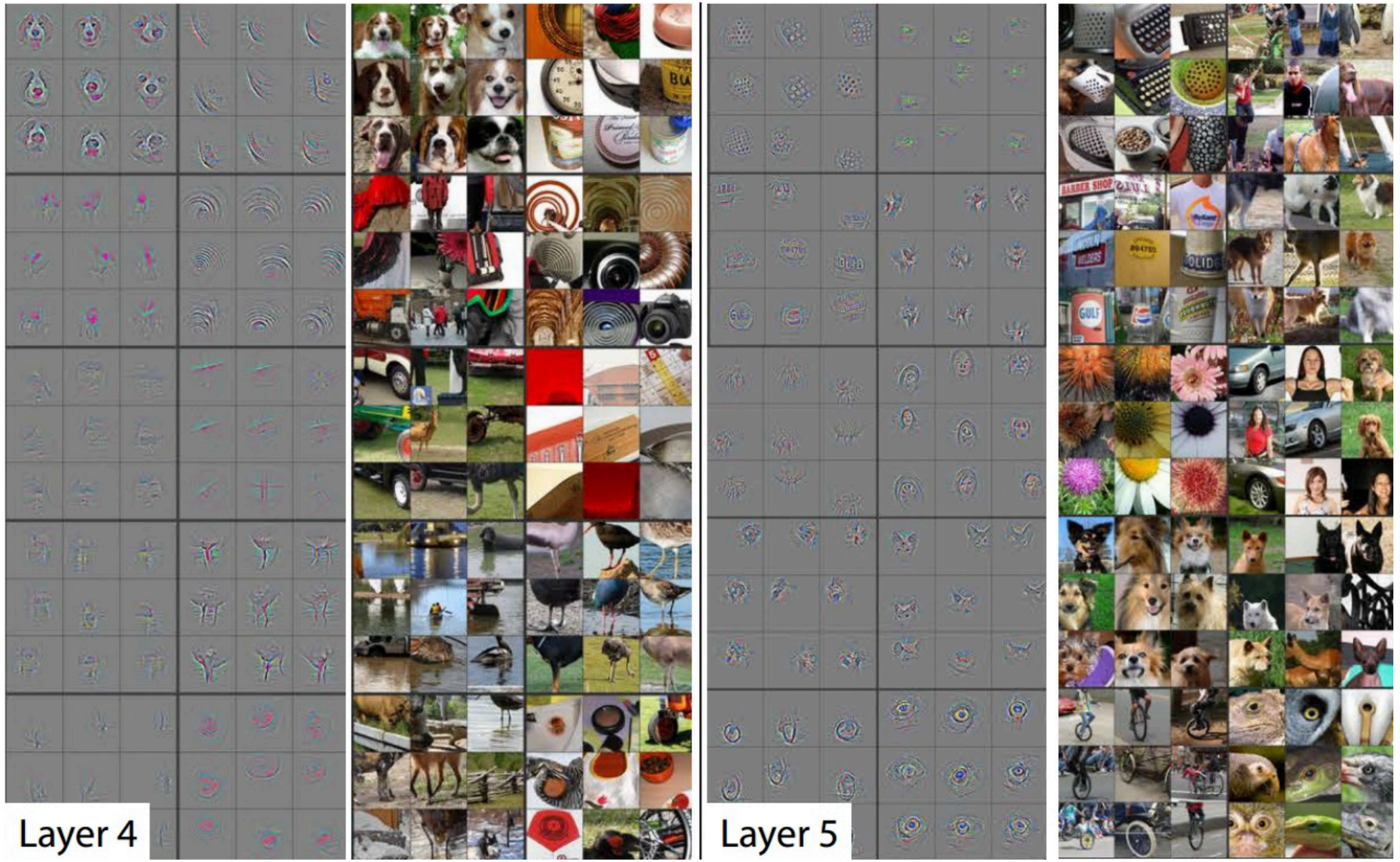


- (note that these features operate on layer 2 features and are thus more complex in terms of their relation to the images; the underlying image patches are also now larger)

[Zeiler and Fergus, ECCV 2014]

What kind of features (kernels) do CNNs learn?

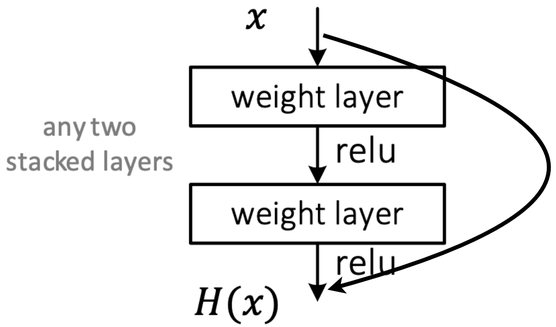
- Layers 4 and 5



Some early CNN developments

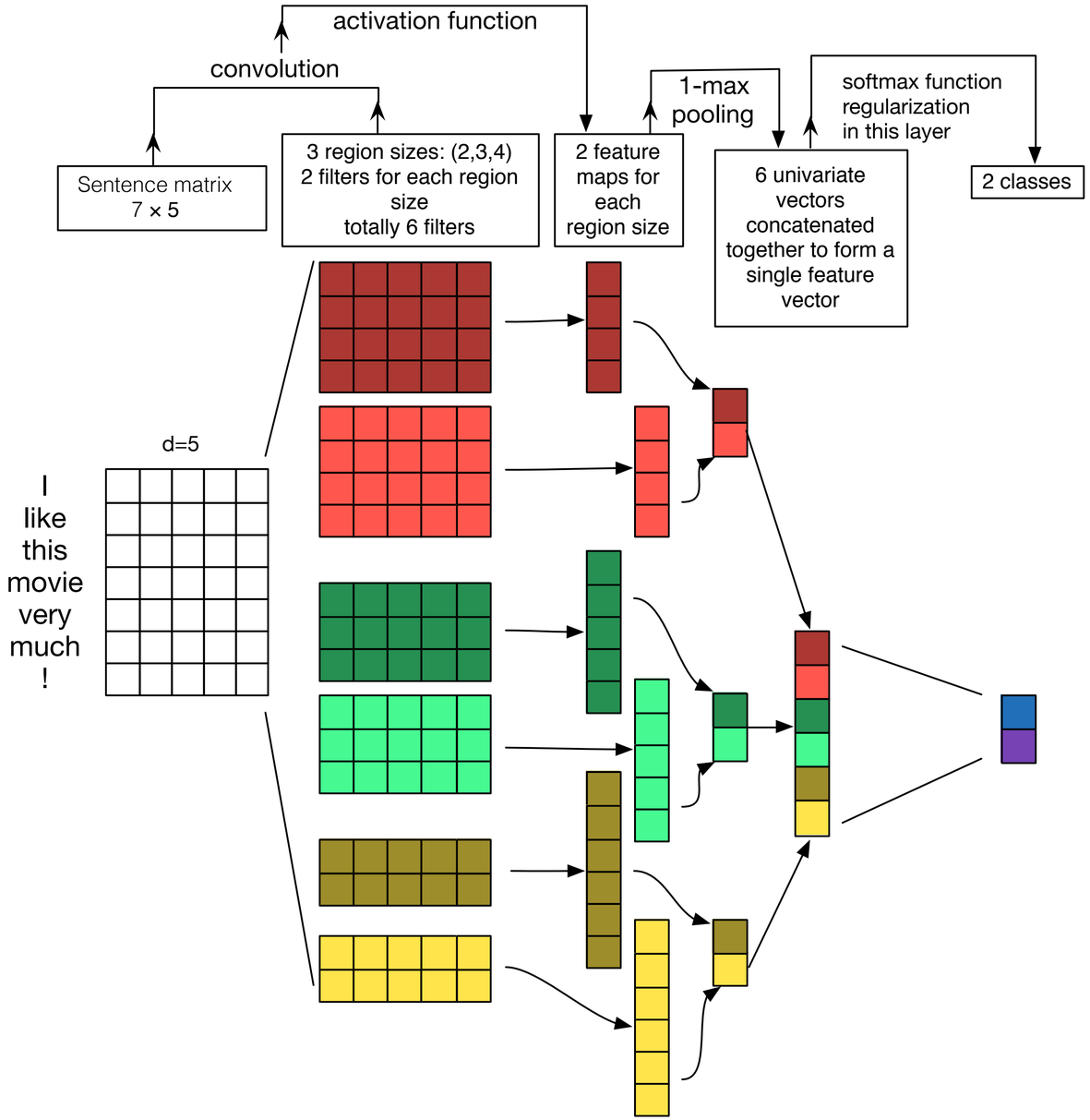
- Rapidly increasing numbers of layers

- Residual or skip connections (easier to learn, network initially linear)



[He et al. 2016]

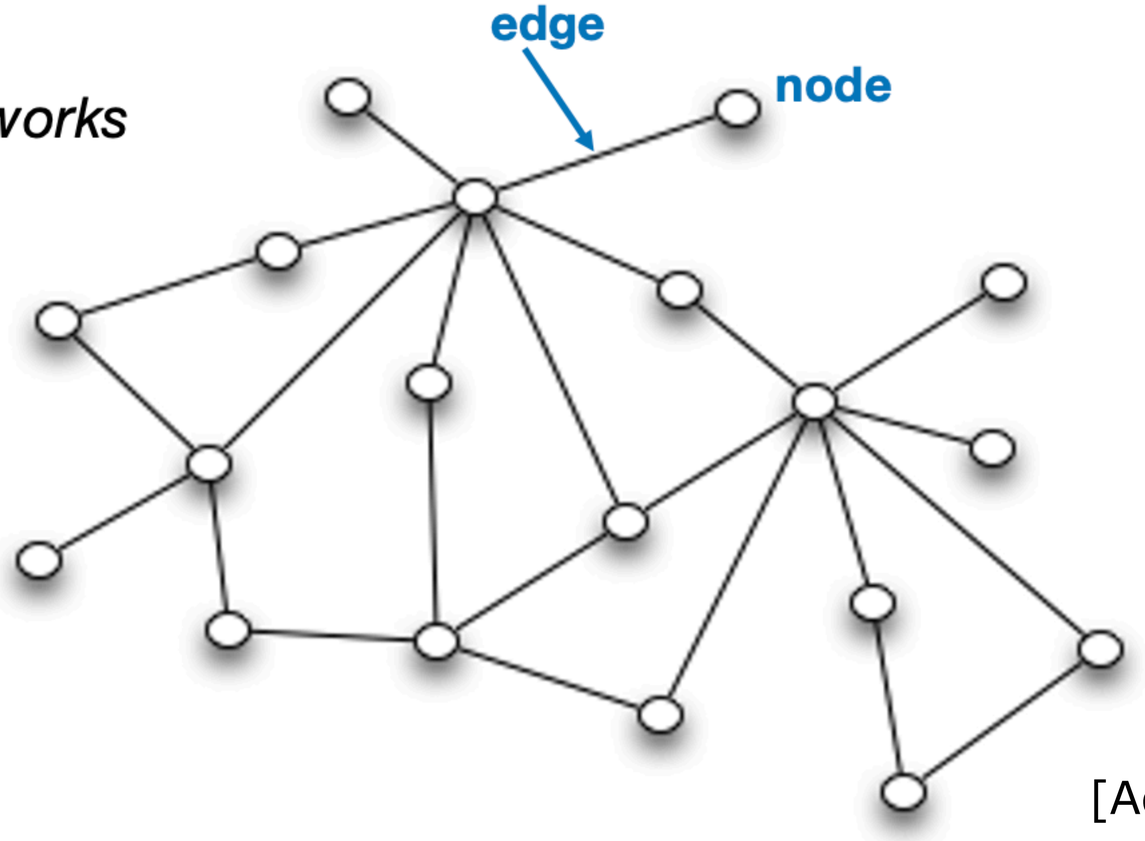
CNNs are not limited to images…



[Zhang et al., 2015]

Graphs and networks

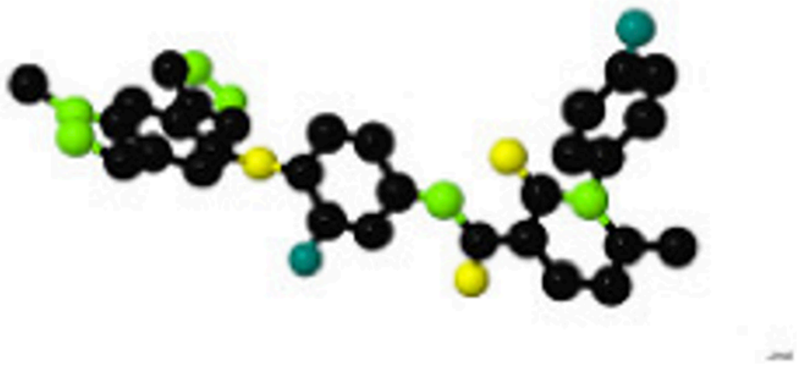
- nodes and connections (edges)
- possibly side information (attributes) for each node
- many examples:
  - *social networks*
  - *traffic networks*
  - *protein interaction networks*
  - *climate networks*
  - *brain*
  - *citation networks*
  - *internet …*



[Adapted from S. Jegelka]

Graphs and networks

- E.g., predicting a label for any given graph



Predict a label for the entire graph

Graph
Molecule

→

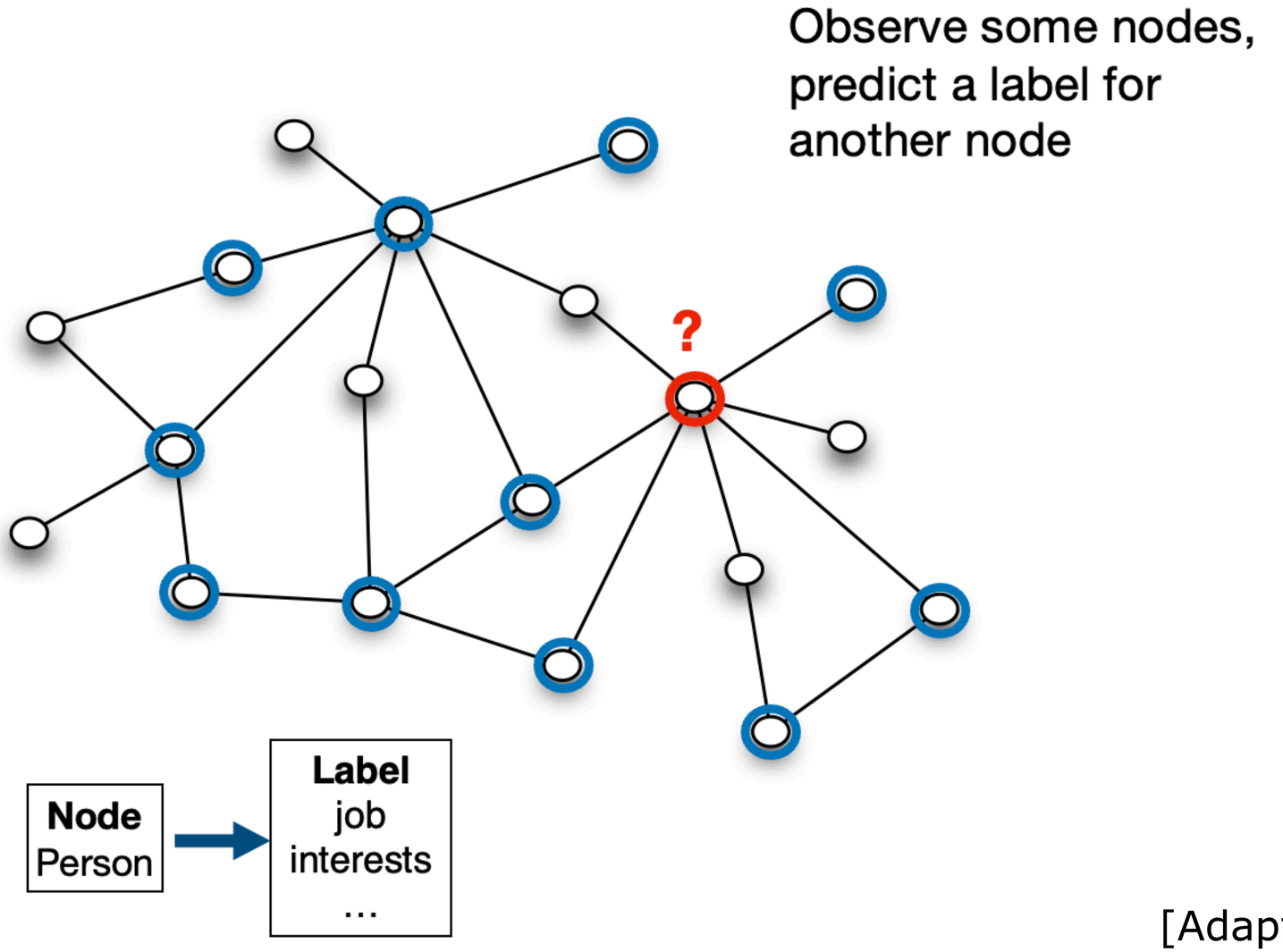
Property
Solubility
Toxicity
Drug efficacy
…

(Duvenaud et al, 2015, Stokes et al 2020,…)

[Adapted from S. Jegelka]

Graphs and networks

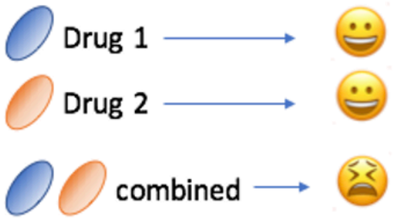
- E.g., predicting node features in a social network



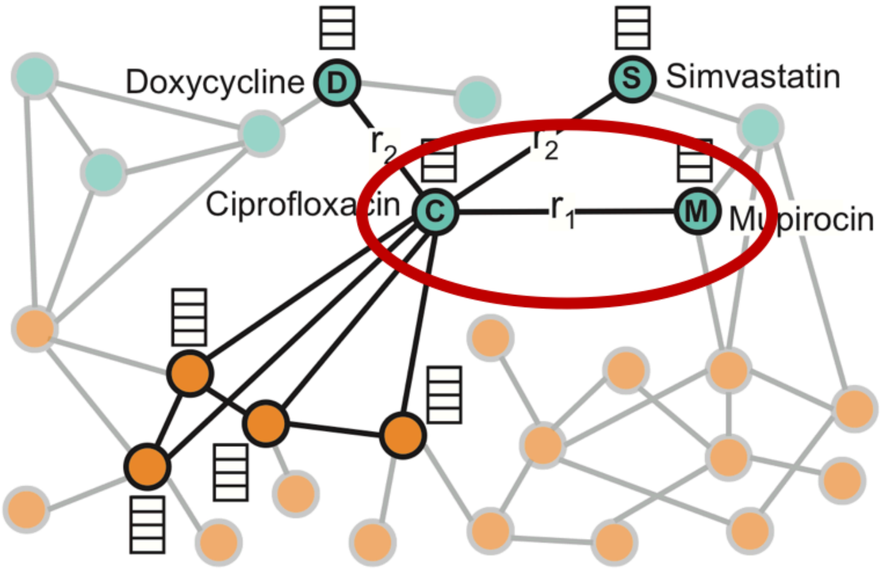
[Adapted from S. Jegelka]

Graphs and networks

- E.g., predicting polypharmacy side effects



Predict if there is an edge between two nodes



Drugs

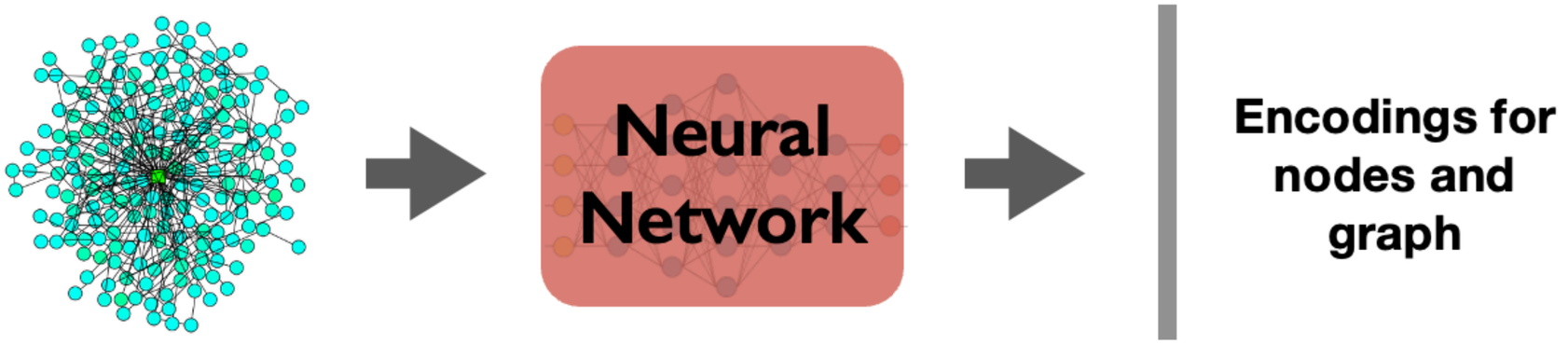
Proteins

Pair of Nodes Drugs	→	Edge Interaction type

(Zitnik et al, 2018)

Neural networks for graphs

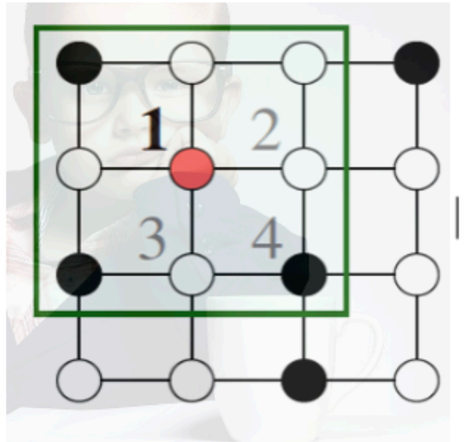
- Need a good representation of nodes or of the entire graph
- Capture node features and neighborhood relations
- Graph Neural Networks (GNNs)



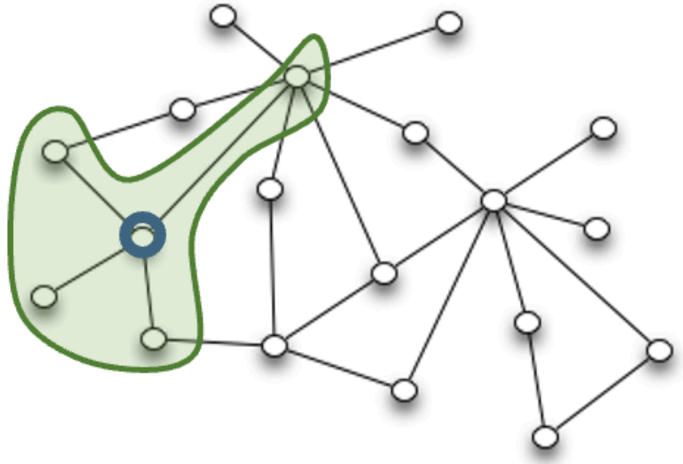
[Adapted from S. Jegelka]

Graph neural networks

- Compare to CNNs…



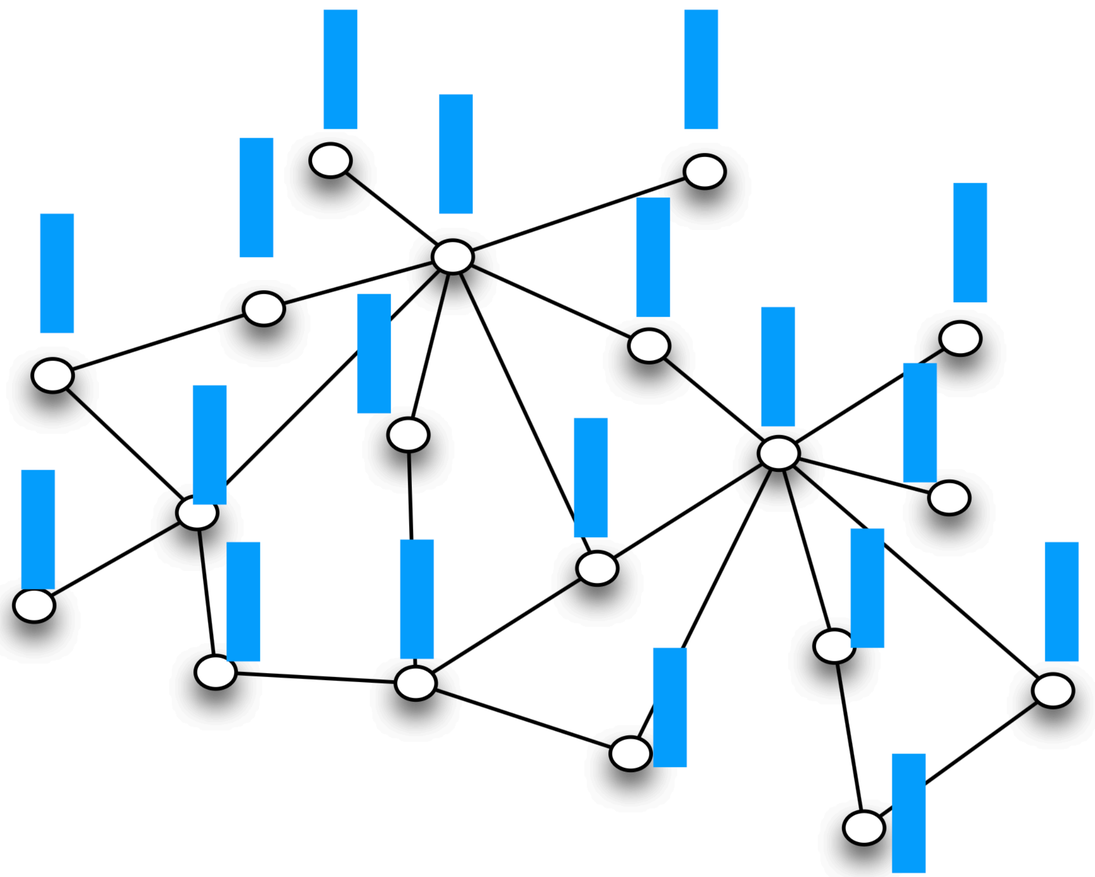
Encode local neighborhoods (patches)



Encode local neighborhoods (nodes that can be reached within *K* hops)

[Adapted from S. Jegelka]

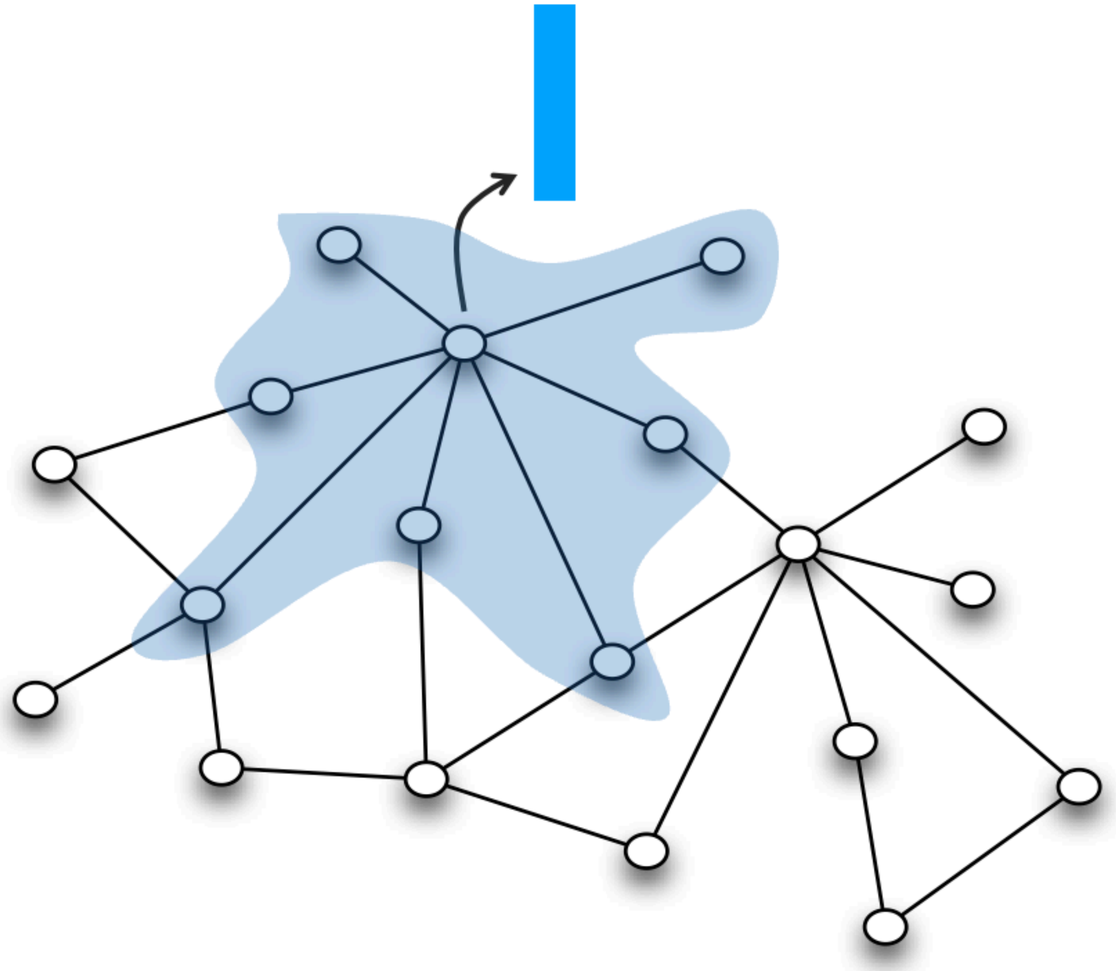
GNNs at a higher level



Input: a graph with node feature vectors

- We could in addition include features on the edges (e.g., types of relations in a social graph, chemical bonds in a molecular graph, etc)

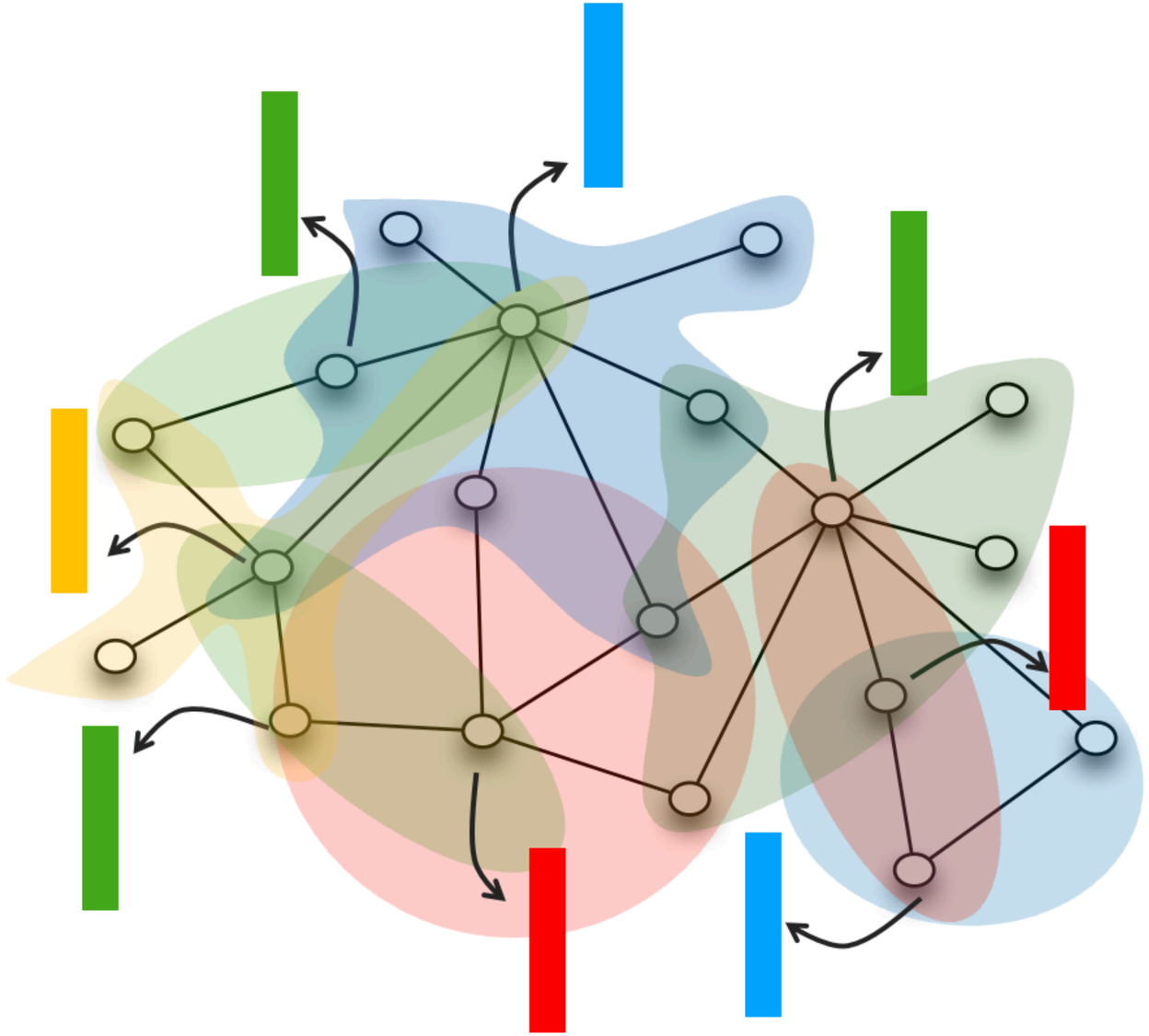
GNNs at a higher level



1. Node embeddings:
encode the local neighborhood
of each node separately
via aggregations

[Adapted from S. Jegelka]

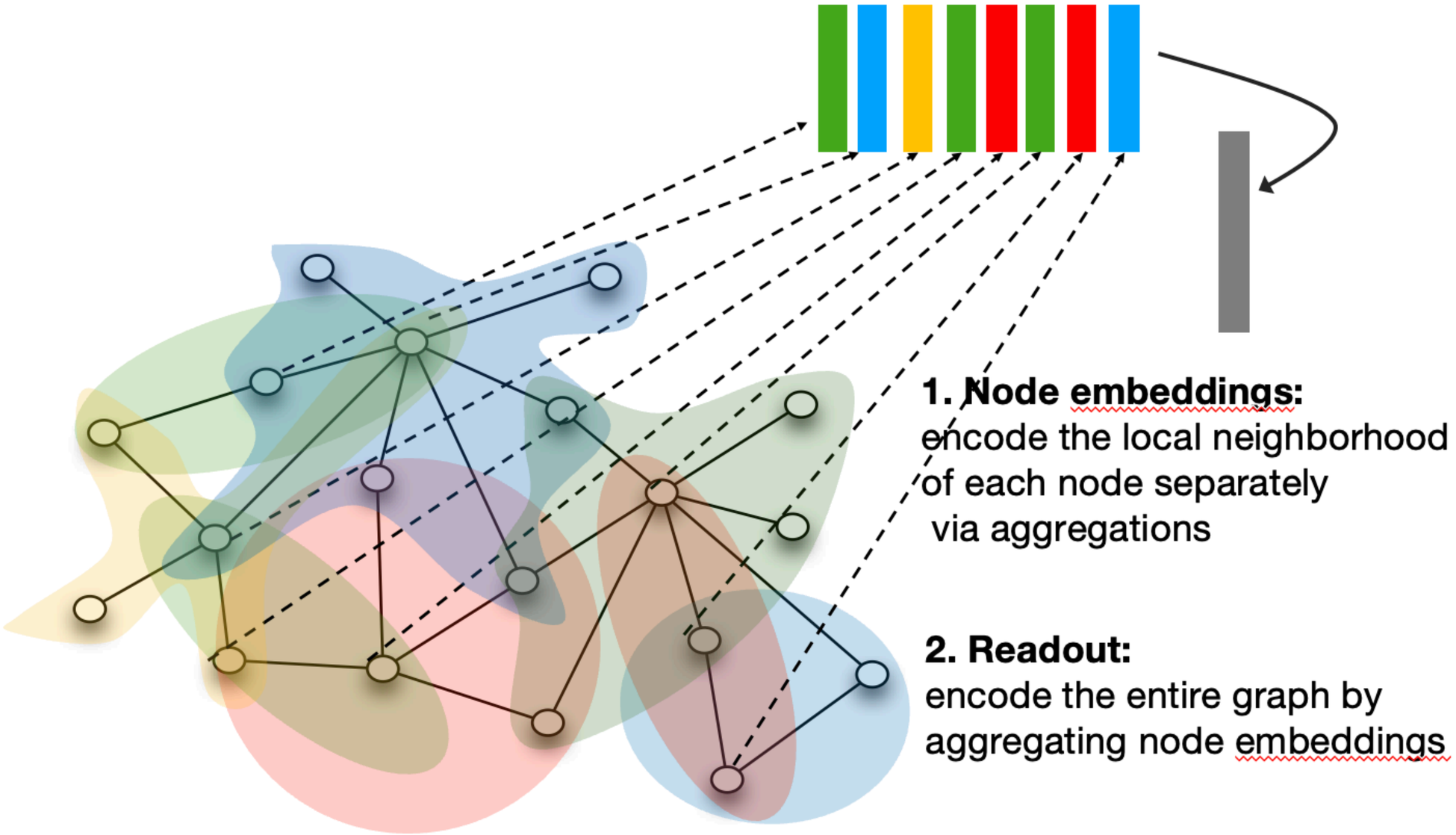
GNNs at a higher level



1. Node embeddings:
encode the local neighborhood
of each node separately
via aggregations

[Adapted from S. Jegelka]

GNNs at a higher level

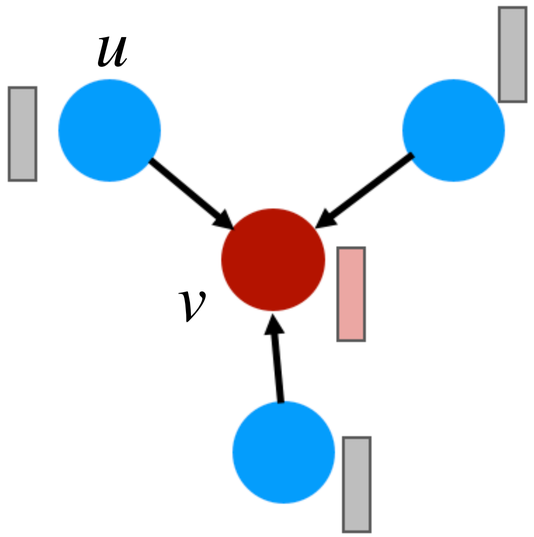


1. Node embeddings:
encode the local neighborhood
of each node separately
via aggregations

2. Readout:
encode the entire graph by
aggregating node embeddings

[Adapted from S. Jegelka]

Towards GNN update equations



feature description of node u in round k-1

In each round *k*:

Aggregate over neighbors

$$m_{\mathcal{N}(v)}^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)$$

input is a multiset

Update: Combine with current node

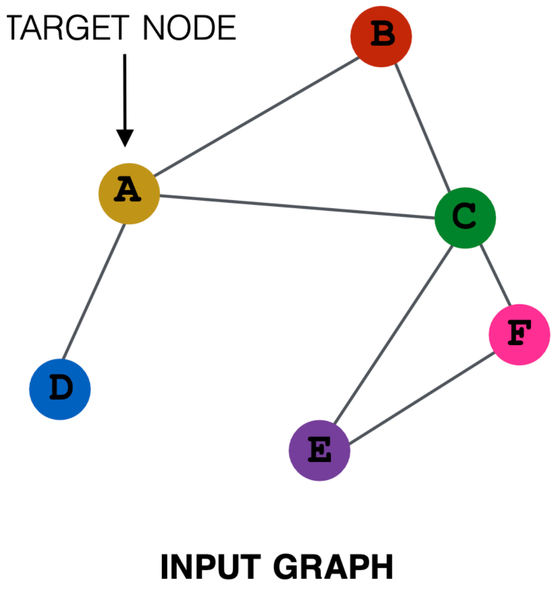
$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, m_{\mathcal{N}(v)}^{(k)}\right)$$

- Note: the aggregation + combination step is permutation invariant

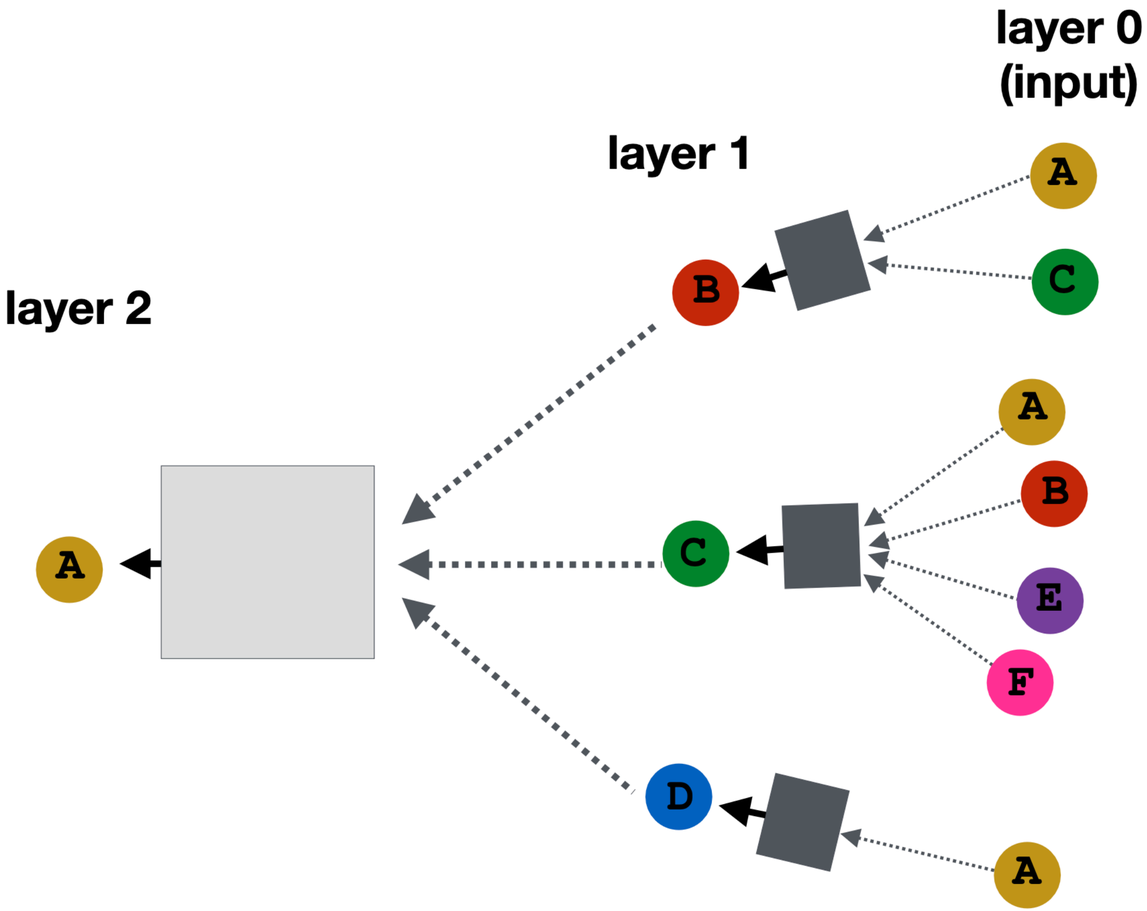
[Adapted from S. Jegelka]

A GNN and its associated computation tree

- We learn the gray boxes (aggregation + combine); all the boxes of the same color have the same parameters



INPUT GRAPH



[Illustration adapted from J Leskovec]

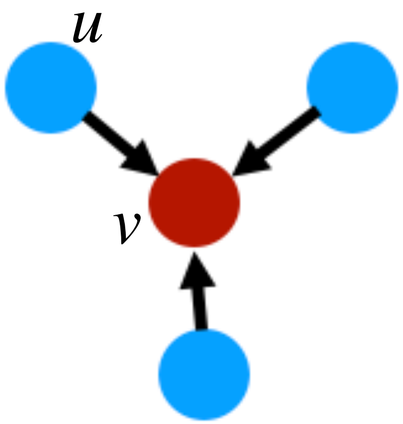
Examples of GNN updates

- Sum, average:

$$m_{\mathcal{N}(v)} = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} h_u$$

Input node attribute of neighbors

- Coordinate-wise min/max



- General form (with small neural networks = MLPs):

$$\mathbf{m}_{\mathcal{N}(v)}^{(k)} = \mathrm{MLP}_2\Big(\sum_{u \in \mathcal{N}(v)} \mathrm{MLP}_1\big(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}\big) \Big)$$

Learned aggregation function

learned

- Update (combine):

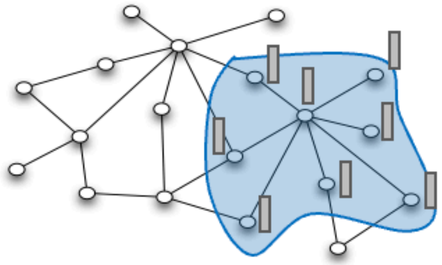
$$h_v^{(k)} = \sigma\left(W_{\text{self}} h_u^{(k-1)} + W_{\text{neigh}} m_{\mathcal{N}(v)}^{(k)} + b \right)$$

learned

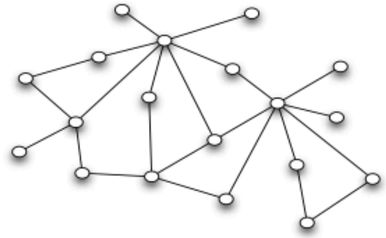
[Adapted from S. Jegelka]

How to train a GNN

- **What is a data point?**



Node + its neighborhood (computation tree)



Full graph

- **What to specify?**
  - Aggregate, combine and readout functions
  - Loss function on prediction

- **Train** with SGD

[Adapted from S. Jegelka]

Representational power of GNNs*

- Theorem [Xu et al. 2019, here informal]: Aggregation based GNNs are at most as powerful as the 1-dim Weisfeiler-Lehman graph isomorphism test

associated computation trees

- Coloring:

$$c^{(t)} : V(G) \to \Sigma$$

$$c^{(t)}(v) = \mathrm{Hash}\Big(c^{(t-1)}(v),\ \{\!\{c^{(t-1)}(u) \mid u \in \mathcal{N}(v)\}\!\}\Big)$$

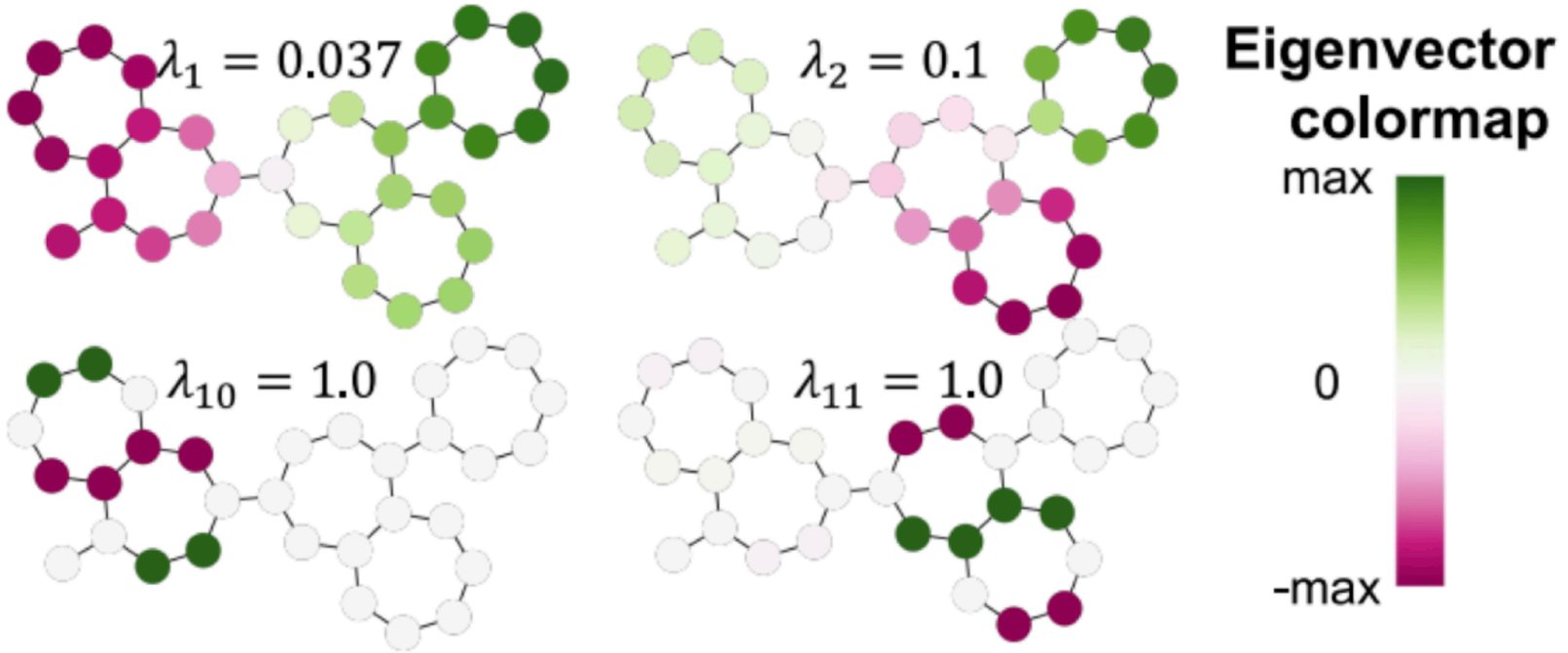
- GNN:

$$\mathbf{h}_v^{(t)} = f_{\mathrm{Update}}\Big(\mathbf{h}_v^{(t-1)},\ f_{\mathrm{Agg}}\big(\{\!\{\mathbf{h}_u^{(t-1)} \mid u \in \mathcal{N}(v)\}\!\}\big)\Big)$$

[Adapted from S. Jegelka]

Graph Laplacian and positional encoding*

- Graph Laplacian $L = D - A$, $D_{ii} = \sum_j A_{ij}$, $A =$ adjacency matrix (size nxn)
- Eigenvectors of the graph Laplacian (all eigenvalues non-negative, $\lambda_0 = 0$) contain important structural information about the graph



(Kreuzer, Beaini, Hamilton, Létourneau, Tossou 2021)

References

- Zeiler et al., “Visualizing and Understanding Convolutional Networks”, https://arxiv.org/abs/1311.2901
- He et al., “Deep residual learning for image recognition”, https://arxiv.org/abs/1512.03385
- Murphy, “Probabilistic Machine Learning: An Introduction”, chapter 14
- Bishop & Bishop, “Deep Learning”, chapter 10