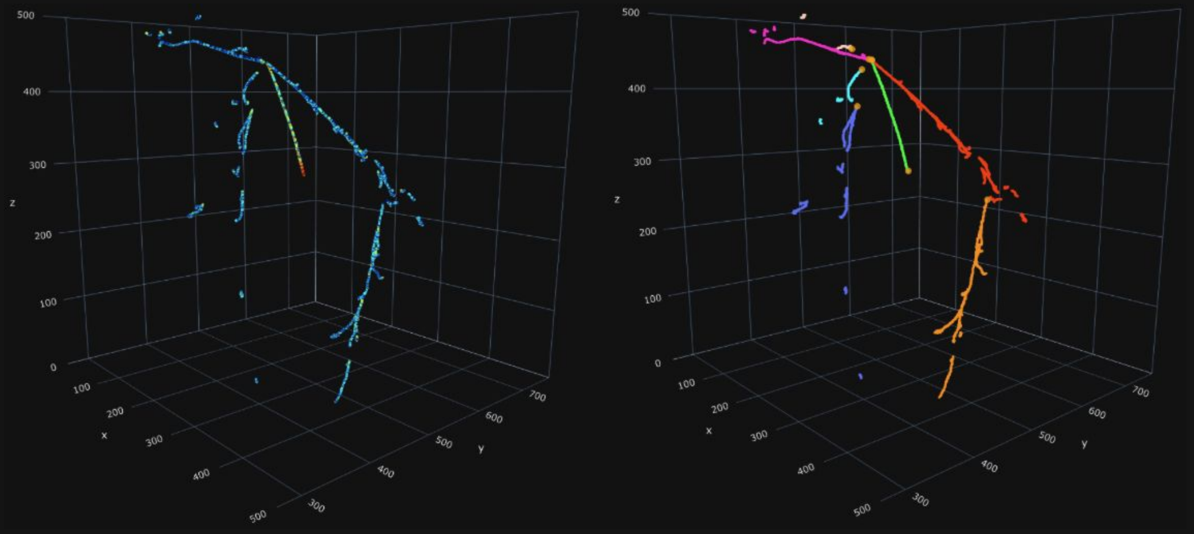
Machine Learning Basics
For ICARUS ML Workshop

Original image credit: xkcd



Kazuhiro Terao
SLAC National Accelerator Laboratory

Machine Learning

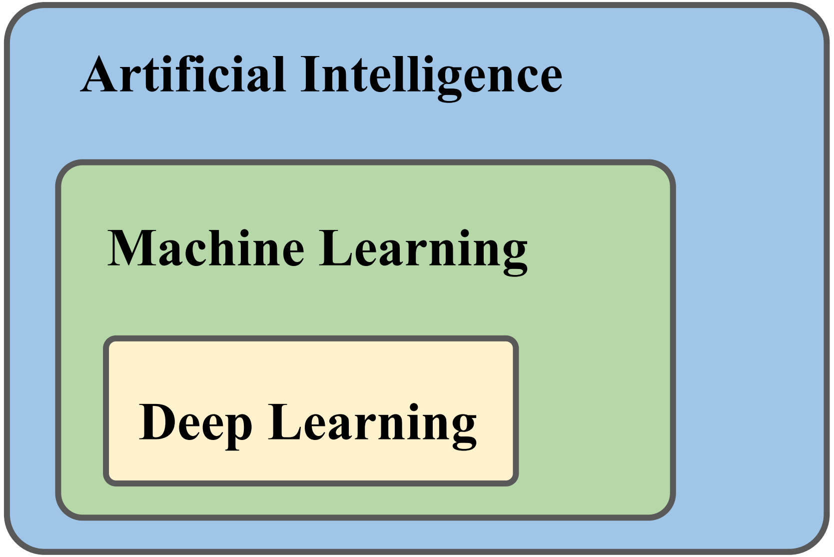
Machine Learning, Deep Learning, AI … what are they?

Machine Learning

Machine Learning, Deep Learning, AI … what are they?

Machine Learning

Machine Learning, Deep Learning, AI … what are they?



Artificial Intelligence

- A computer with intelligence

Machine Learning

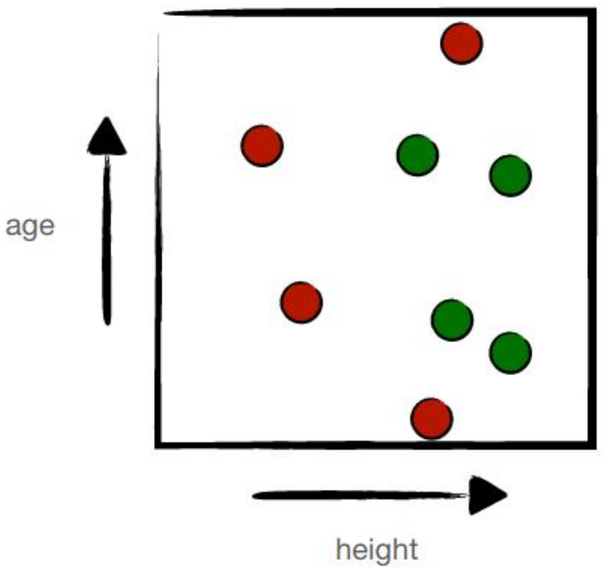
- Process to generate an intelligent algorithm from data.

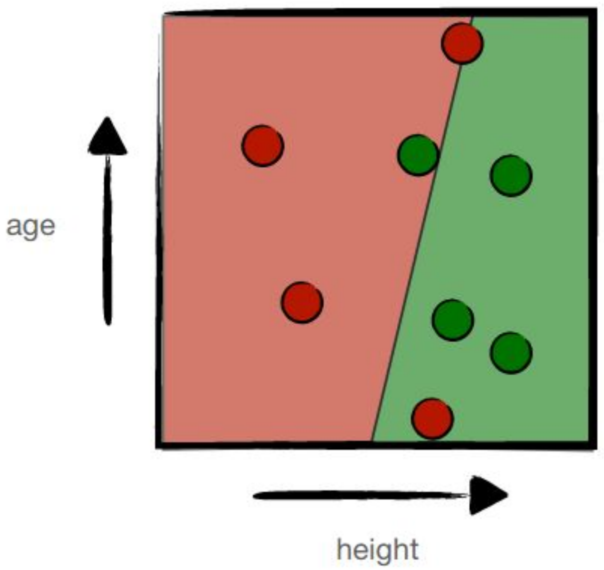
Deep Learning

- ML methods that aim at complex pipelines working on low-level data

Machine Learning

Turning data into an algorithm

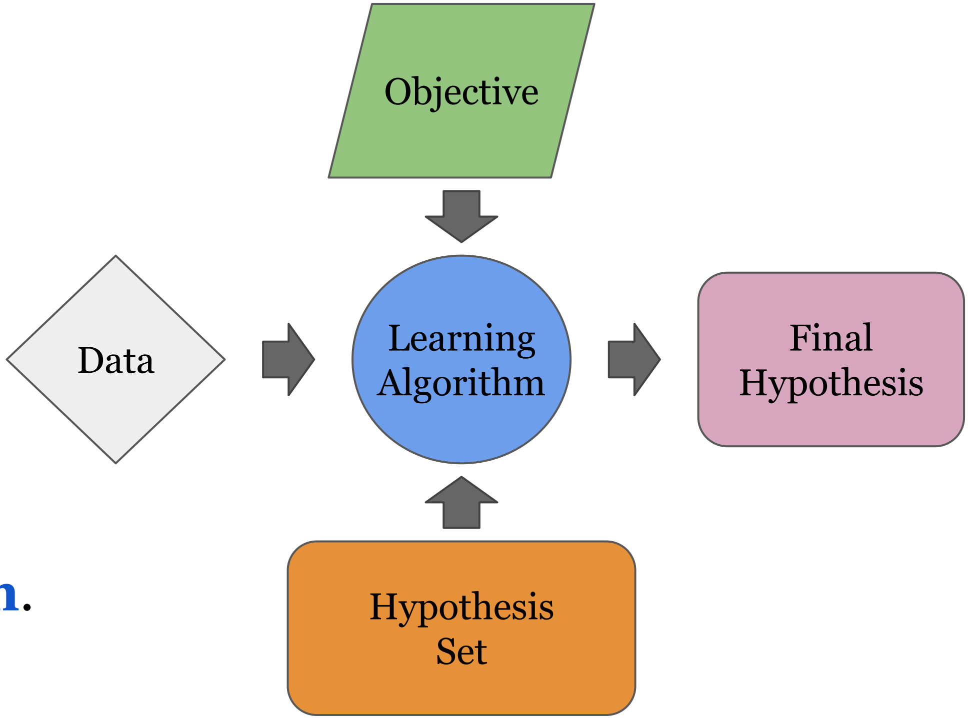




Machine Learning

How to machine learn?

1. Prepare **data set**.
2. Propose a search space of **potential algorithms**.
3. Define a performance metric (**objective to learn**).
4. Provide a **learning algorithm**.



Hypothesis Set

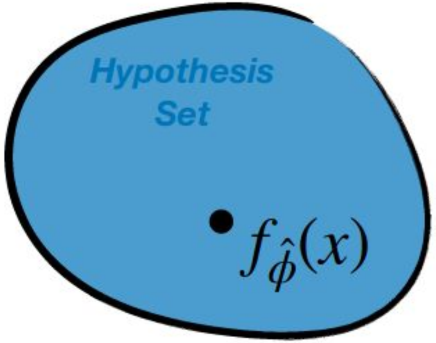
Algorithm = a numeric program with input and output

$$f(x) : \mathcal{X} \to \mathcal{Y}$$

Popular choice to form a “set of candidate algorithm”: **parametrization**

$$f_{\boxed{\phi}}(x) : \mathcal{X} \to \mathcal{Y}$$

parameters



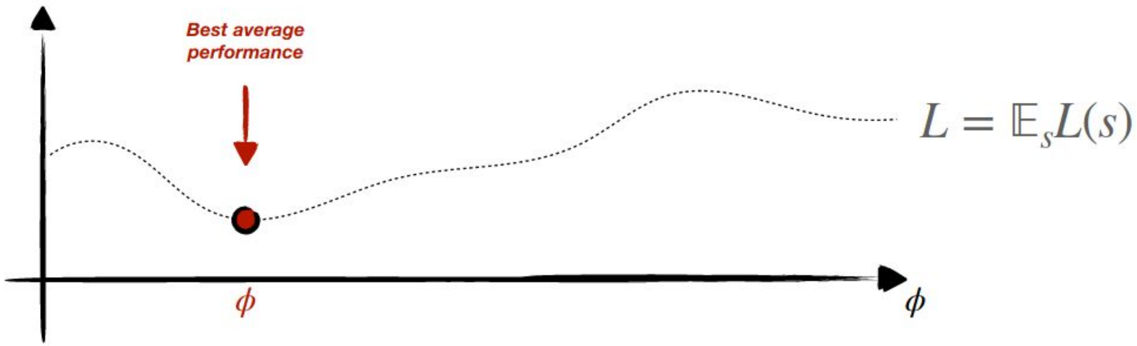
Learning = find parameters: $\hat{\phi} = \text{argmax}_{\phi}\text{Perf}$

Statistical Learning

Assume: data is a stochastic sample. $s \sim p(s)$

In order to learn, we must estimate the performance usually measured as a "risk" or "loss" (i.e. the lower the better). The true expectation loss is:

$$\bar{L}_\phi = E_{s \sim p(s)} \, L_\phi(s) = \int \mathrm{d}s \, L(s) p(s)$$



Statistical Learning

Assume: data is a stochastic sample. $s \sim p(s)$

In order to learn, we must estimate the performance usually measured as a "risk" or "loss" (i.e. the lower the better). The true expectation loss is:

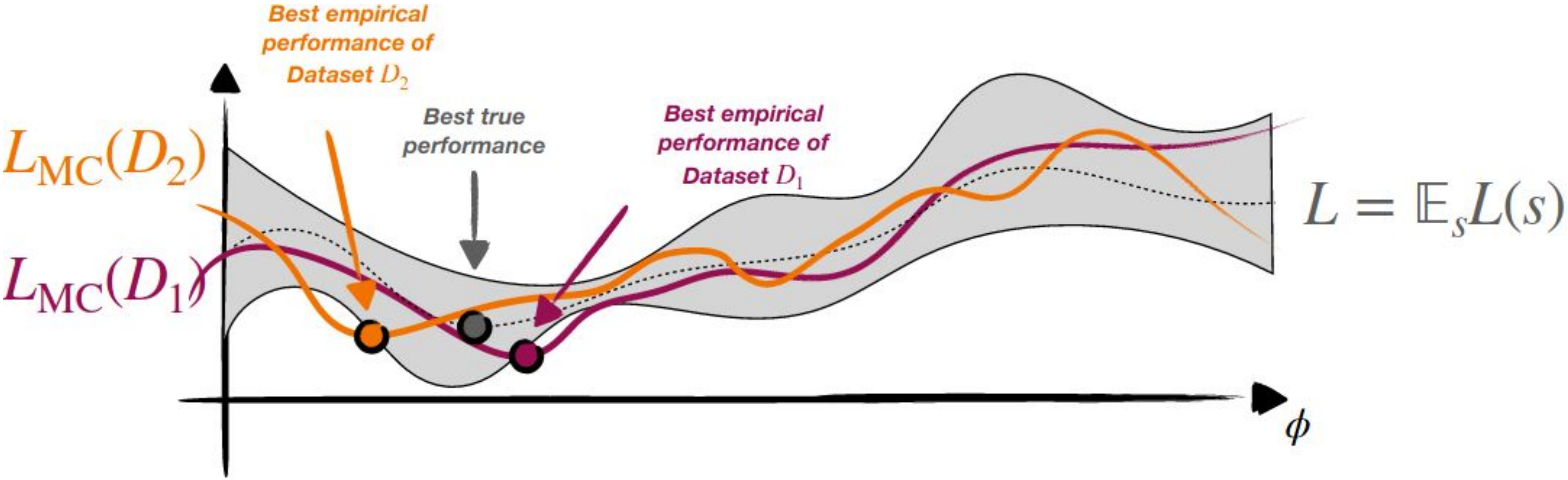
$$\bar{L}_\phi = E_{s\sim p(s)}\, L_\phi(s) = \int \mathrm{d}s\; L(s)p(s)$$

However, we only have data set (sample "s") and have no access to p(s). We can, however, have an unbiased approximation, the **empirical loss**

$$\bar{L} \approx \underbrace{L_{\mathrm{MC}}}_{\text{"Monte Carlo" estimate}}(D) = \frac{1}{N}\sum_i L(s_i)$$

Empirical Risk Minimization

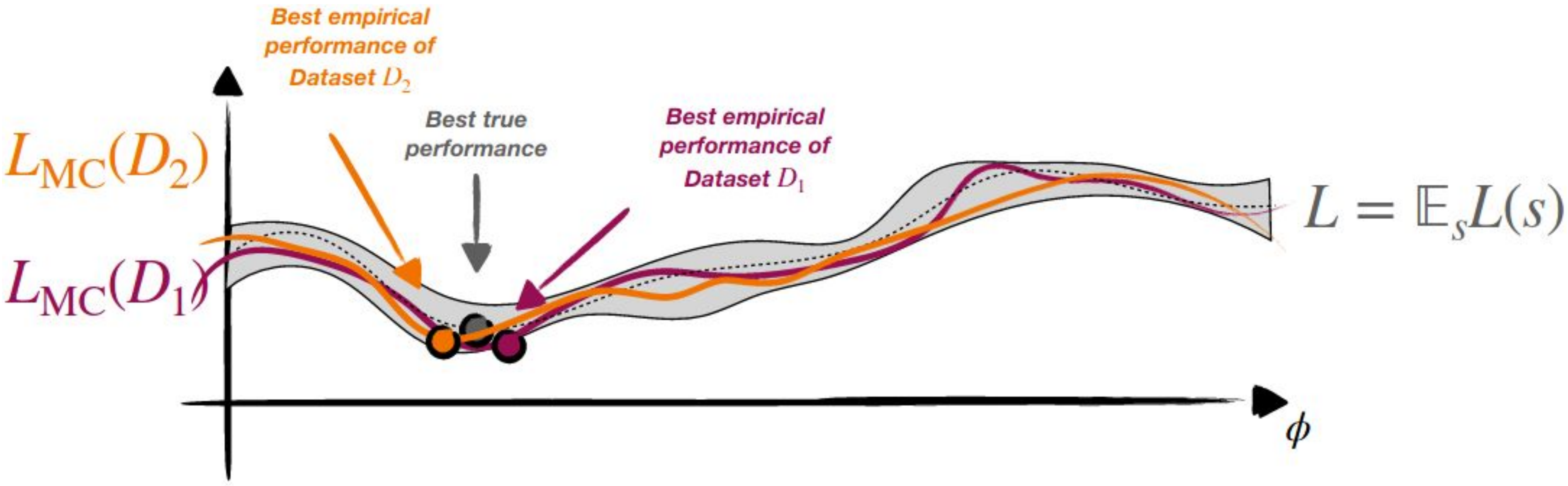
Minimization of empirical loss = choose a hypothesis from the set such that it performs best for the given dataset. The data size is critical.



Small Data

Empirical Risk Minimization

Minimization of empirical loss = choose a hypothesis from the set such that it performs best for the given dataset. The data size is critical.



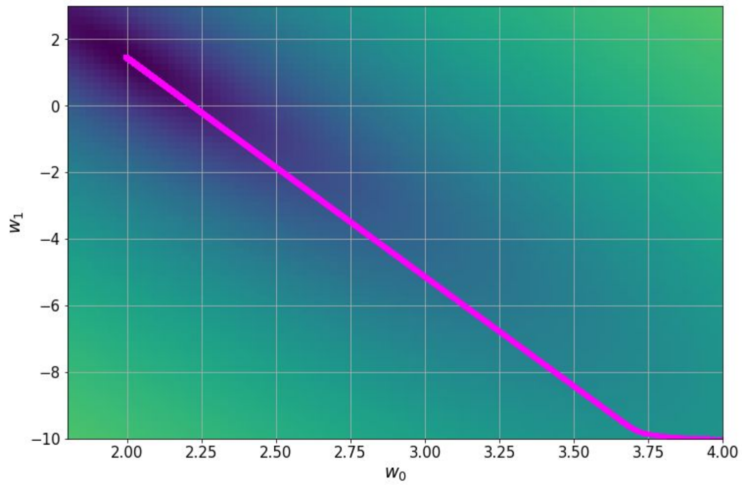
Big Data

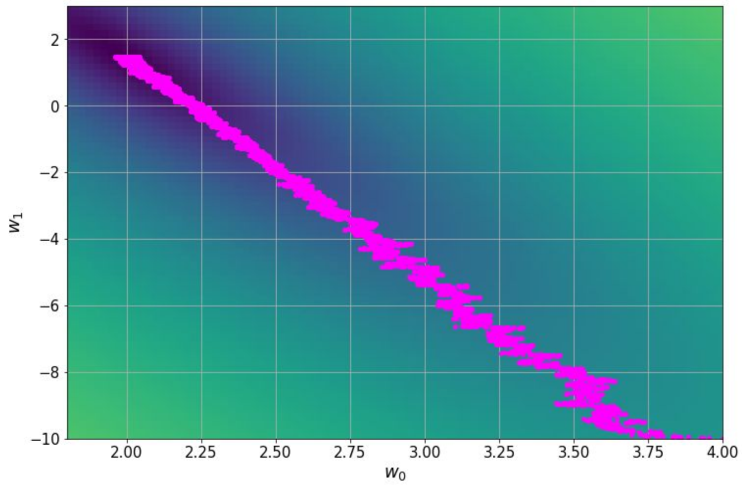
Learning algorithm: gradient-based optimization

Gradient descent (GD): $\theta_t = \theta_{t-1} - \lambda \nabla_\theta \mathcal{L}(x, \theta)$

Stochastic GD (SGD) approx. using **subset of data**.

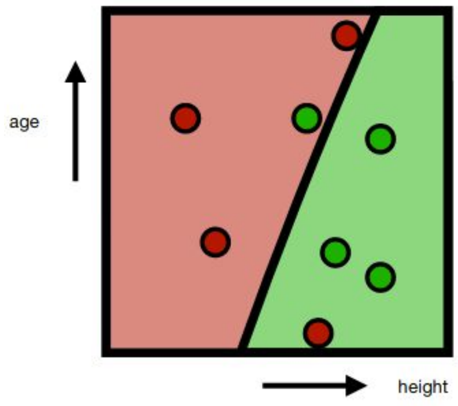
1. Create a batch = random subset of data.
2. Compute the gradient for the batch and update the parameters.





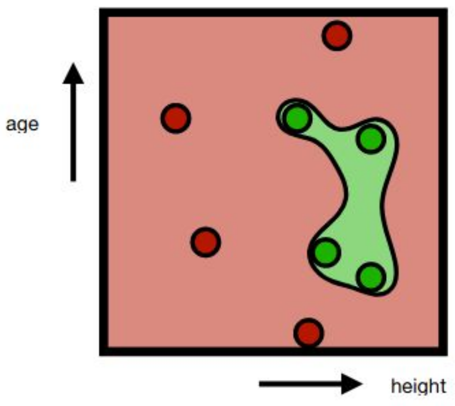
Hypothesis: simple or complex?

Simple: line



Small Data

Complex: curves

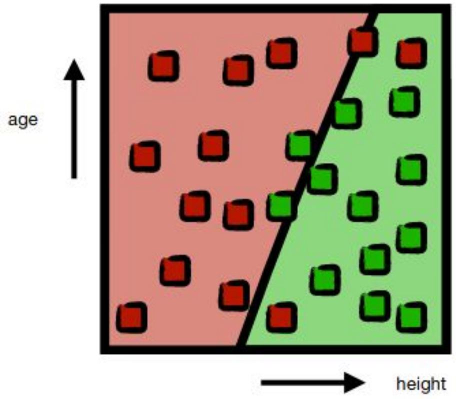


Empirical risk: 25%

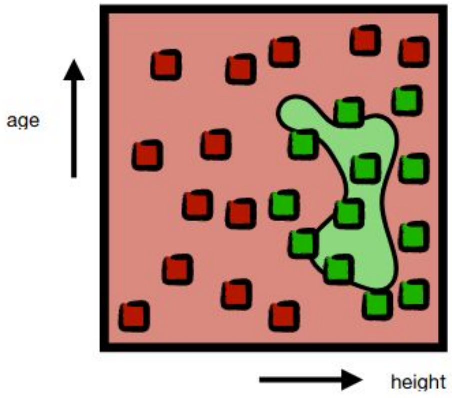
Empirical risk: 0%

Hypothesis: simple or complex?

Simple: line



Complex: curves



Small Data

Empirical risk: 25%

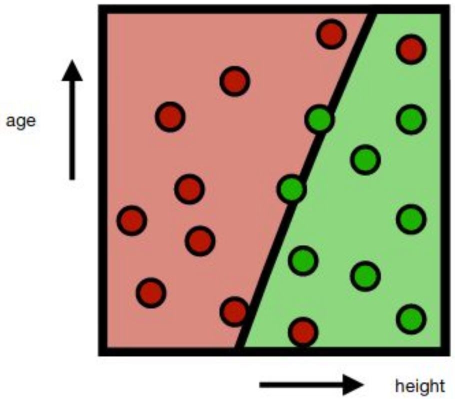
True risk: ~16%

Empirical risk: 0%

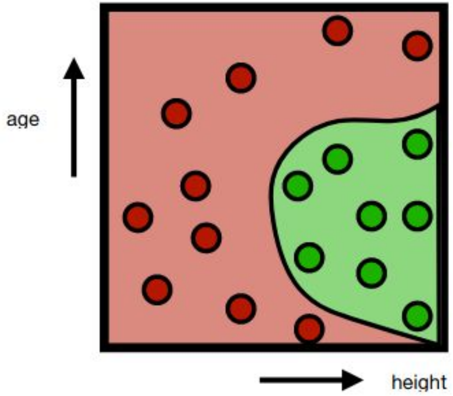
True risk: ~50%

Hypothesis: simple or complex?

Simple: line



Complex: curves



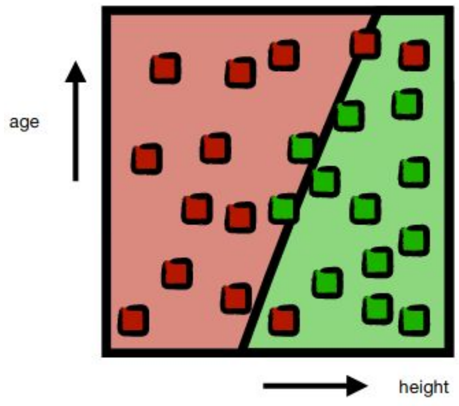
Big Data

Empirical risk: 25%

Empirical risk: 0%

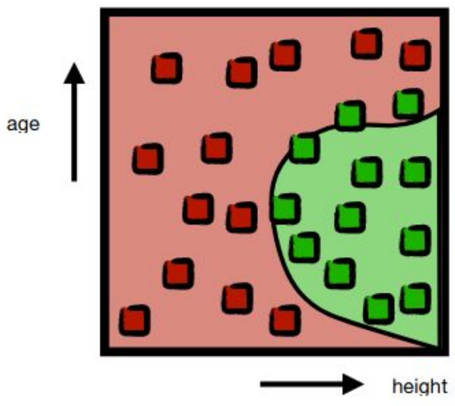
Hypothesis: simple or complex?

Simple: line



Big Data

Complex: curves



Empirical risk: 25%

True risk: ~16%

Empirical risk: 0%

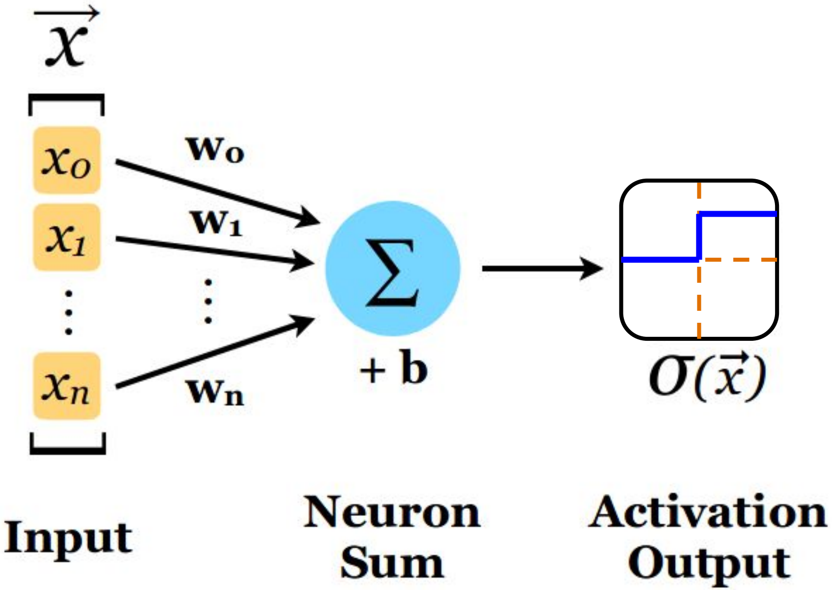
True risk: ~8%

Neural Networks

Introduction to Neural Network

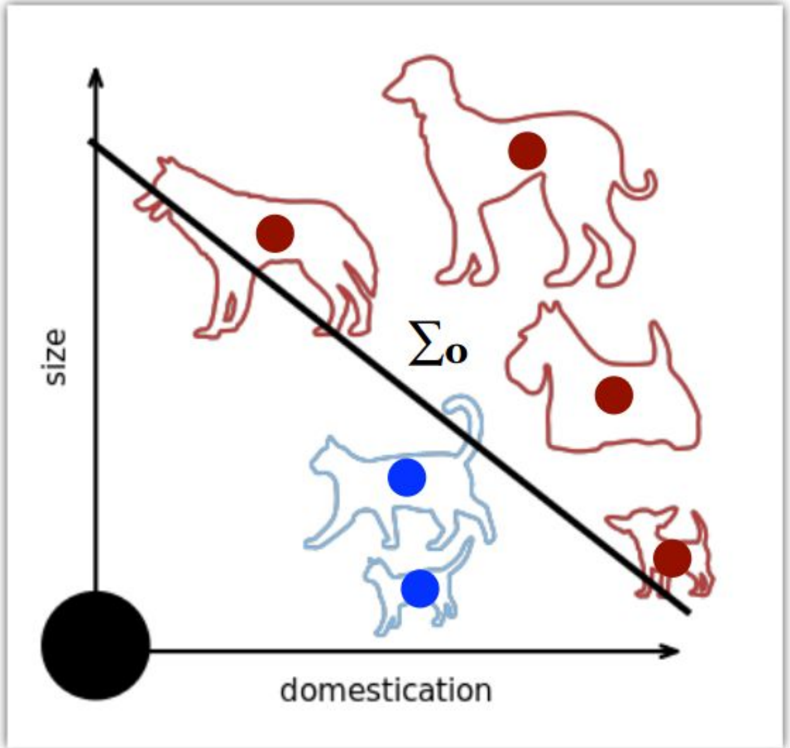
The basic unit of a neural net is the ***perceptron*** (loosely based on a real neuron)

Takes in a vector of inputs (x). Commonly inputs are summed with weights (w) and offset (b) then run through activation.



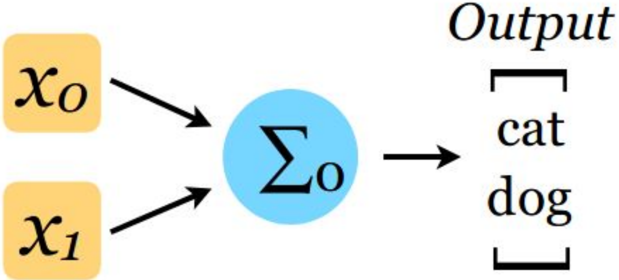
$$\sigma(\vec{x}) = \begin{cases} 1 & \vec{w}_i \cdot \vec{x} + b_i \geq 0 \\ 0 & \vec{w}_i \cdot \vec{x} + b_i < 0. \end{cases}$$

Imagine using two features to separate cats and dogs



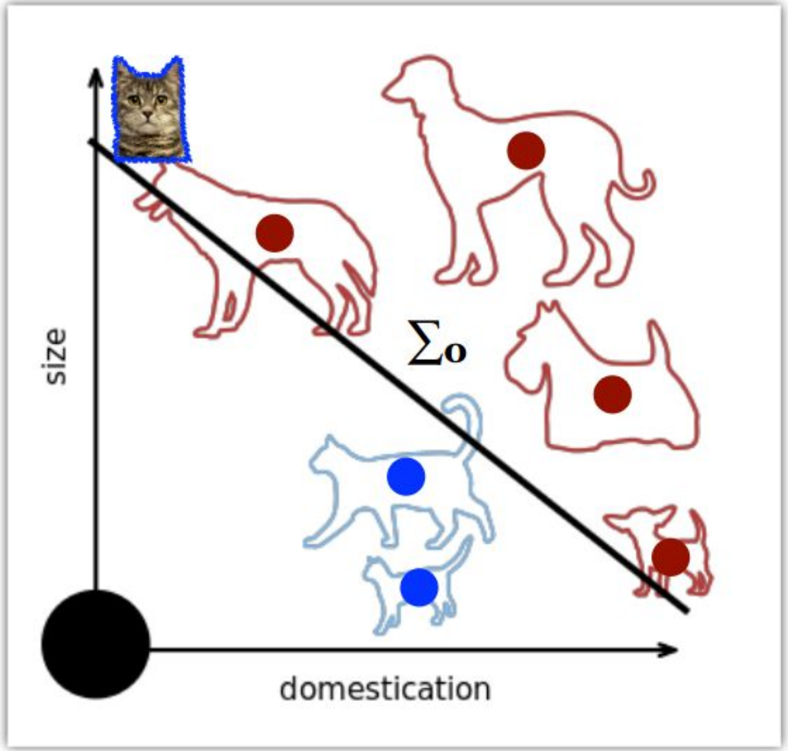
from **[wikipedia]()**

$$\sigma(\vec{x}) = \begin{cases} \vec{w}_i \cdot \vec{x} + b_i & \vec{w}_i \cdot \vec{x} + b_i \geq 0 \\ 0 & \vec{w}_i \cdot \vec{x} + b_i < 0. \end{cases}$$

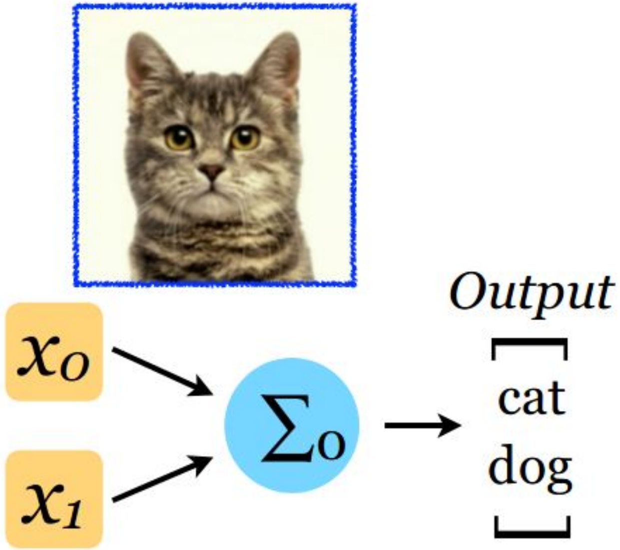


By picking a value for **w** and **b**, we define a boundary between the two sets of data

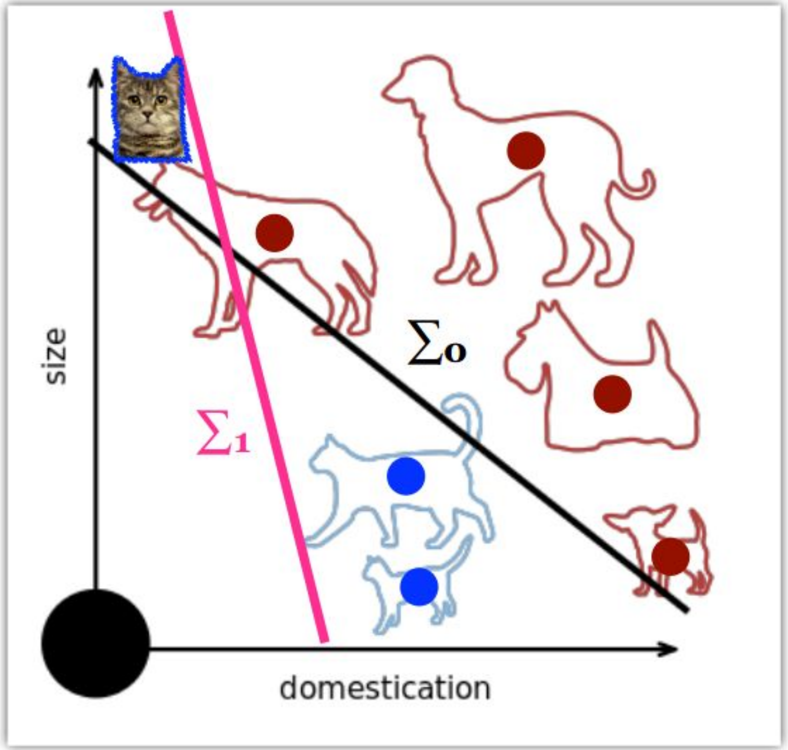
What if we have a new data point?



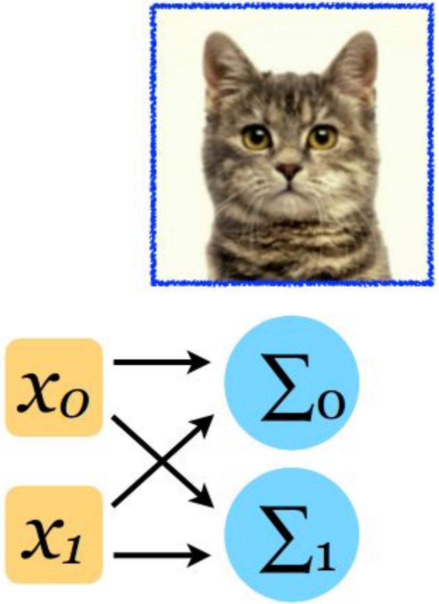
from [wikipedia]



What if we have a new data point?

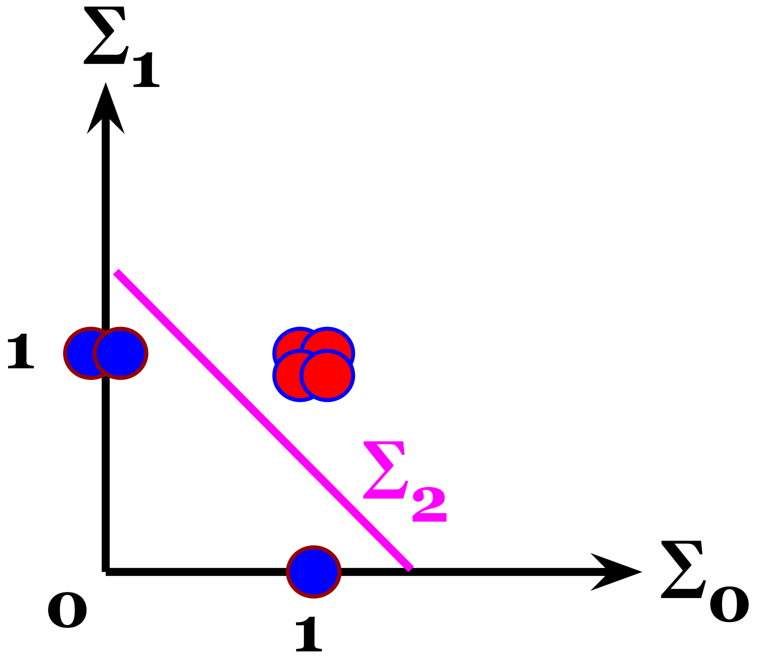


from [**wikipedia**]

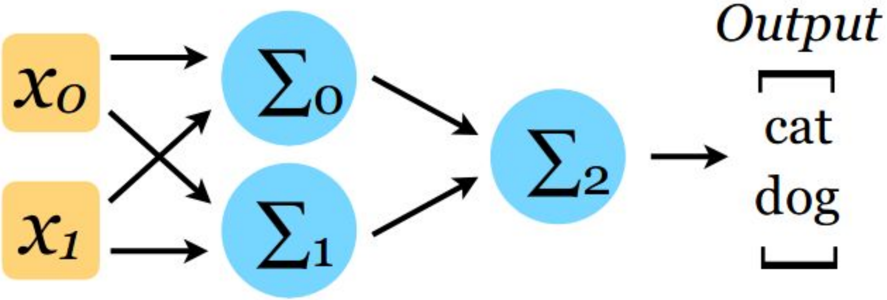


We can **add another perceptron** to help (but does not yet solve the problem)

What if we have a new data point?



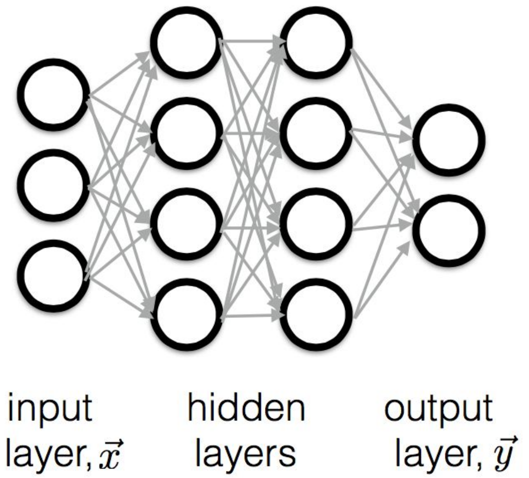
from wikipedia



Another layer can classify based on preceding layer's output (of **non-linear activation**)

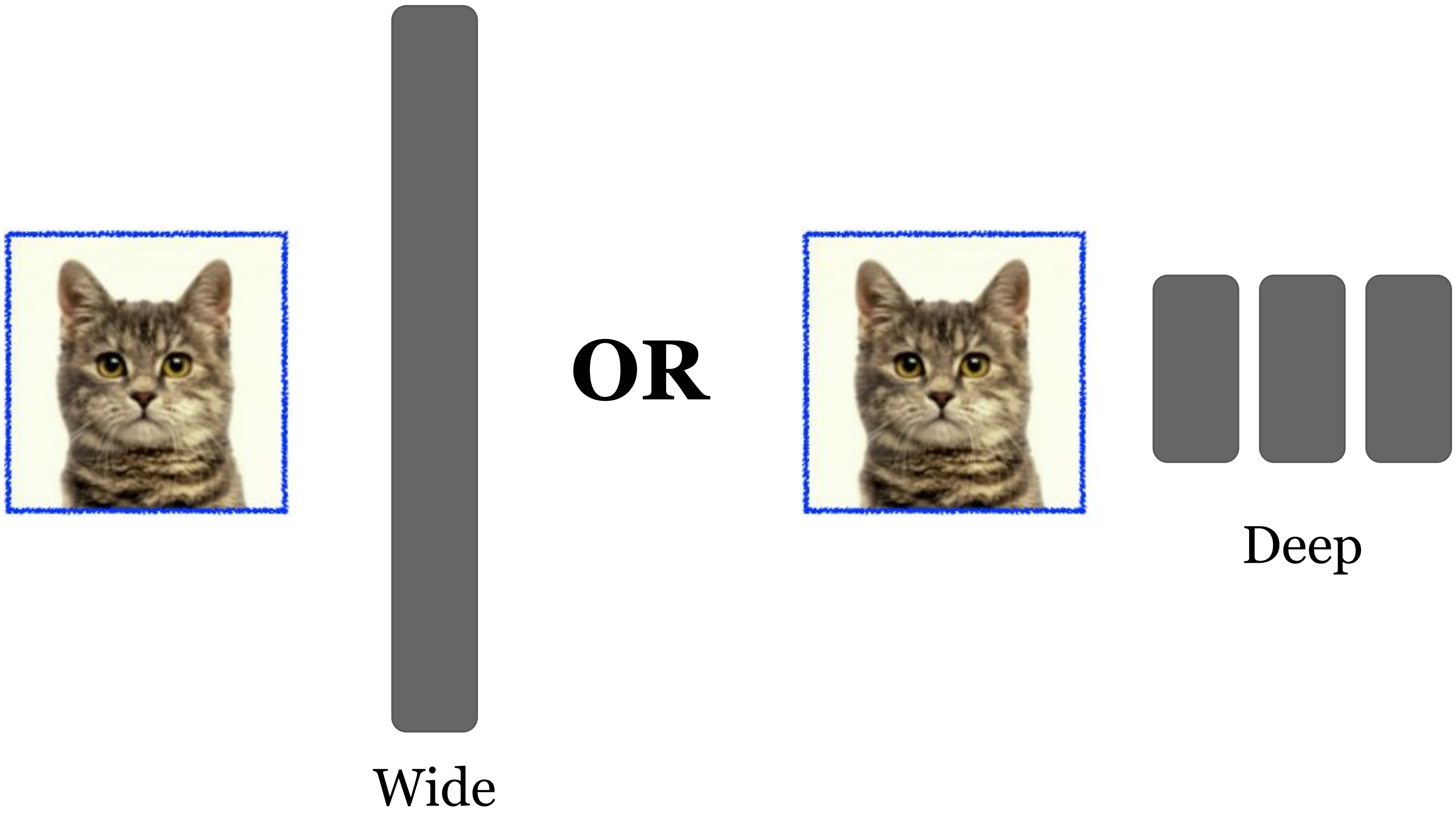
Vanilla neural net

Multi-Layer Perceptrons (MLP)



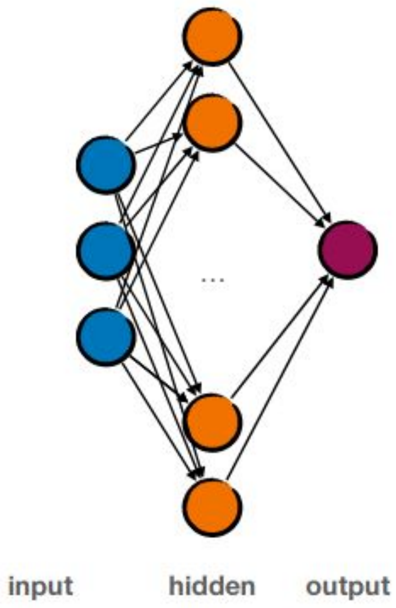
A traditional neural network consists of a stack of layers of such neurons where each neuron is ***fully connected*** to other neurons of the neighbor layers

Neural Network: Architecture Choice



Universal Approximation Theorem

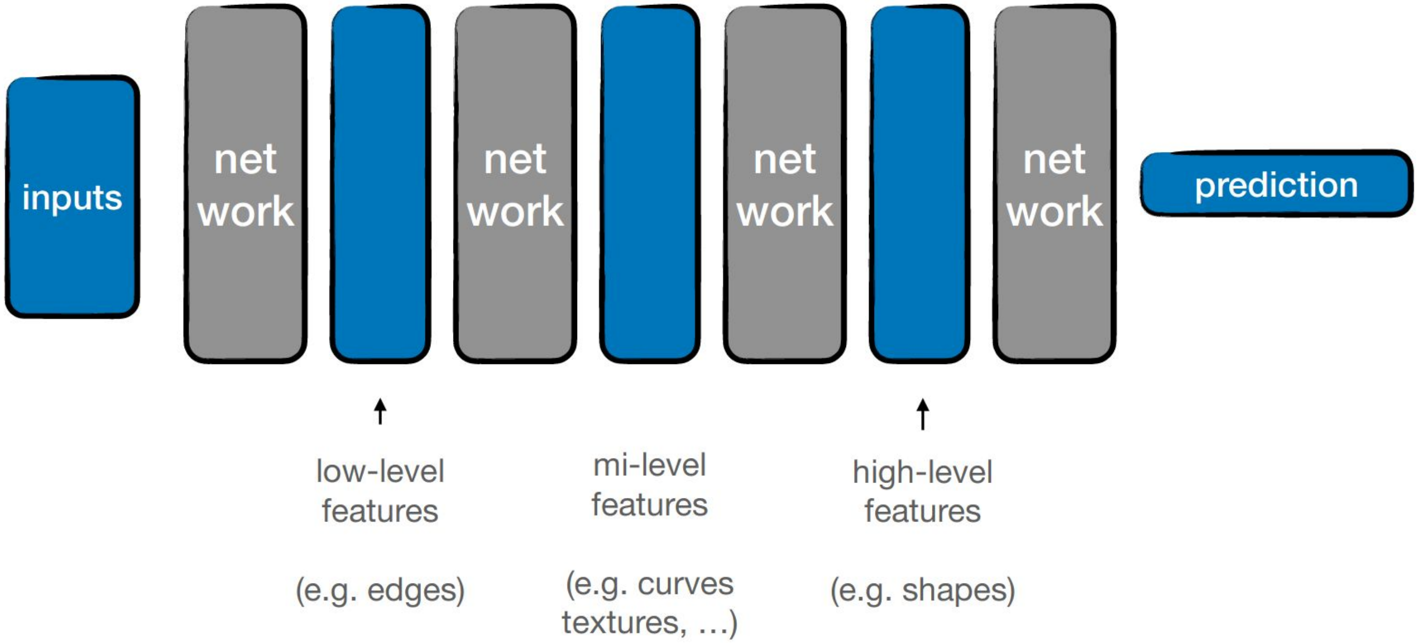
It can be shown that a MLP with single hidden layer is a universal function approximator (can represent any function).



Why do we need a deep network?

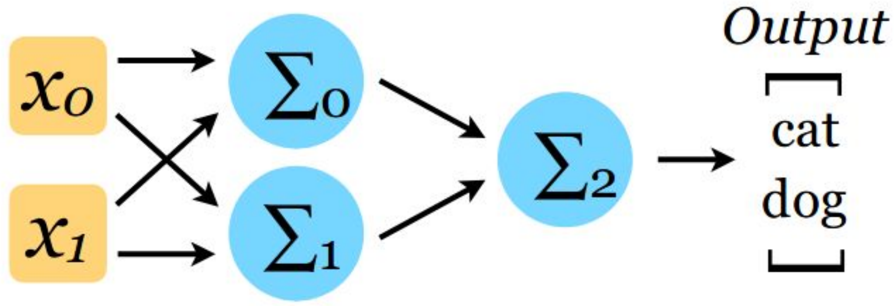
Benefits of the depth

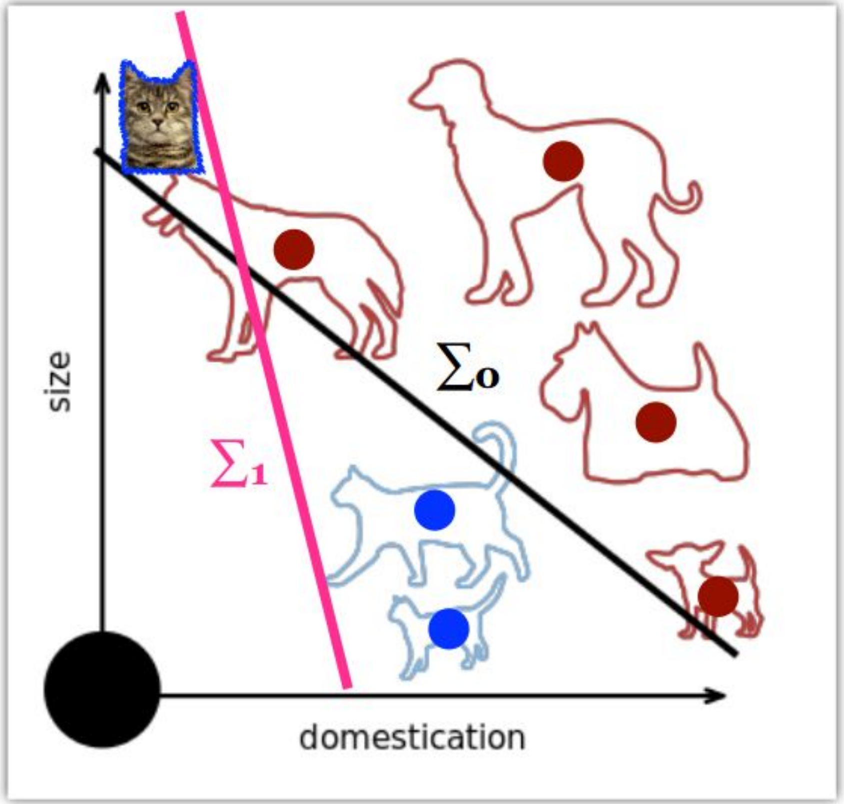
A neural network becomes exponentially more expressive with the depth due to composition of features into higher level concepts.

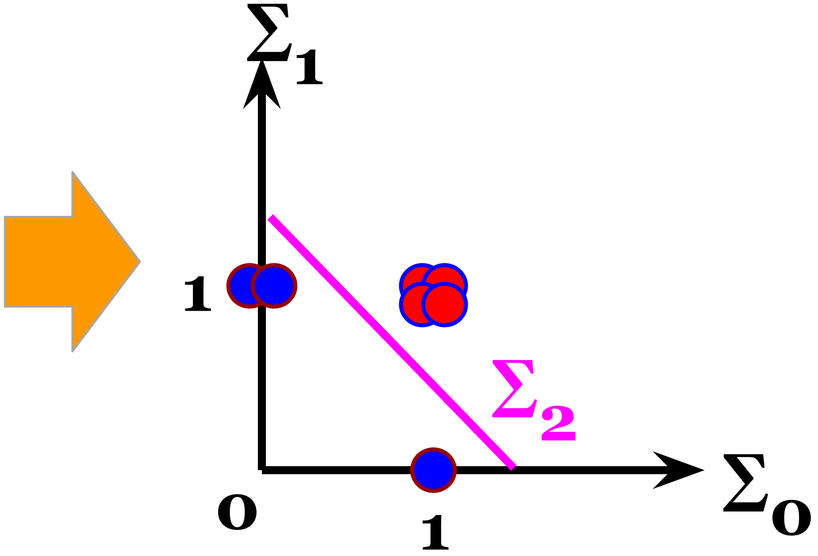


Convolutional Neural Network
for Image Data

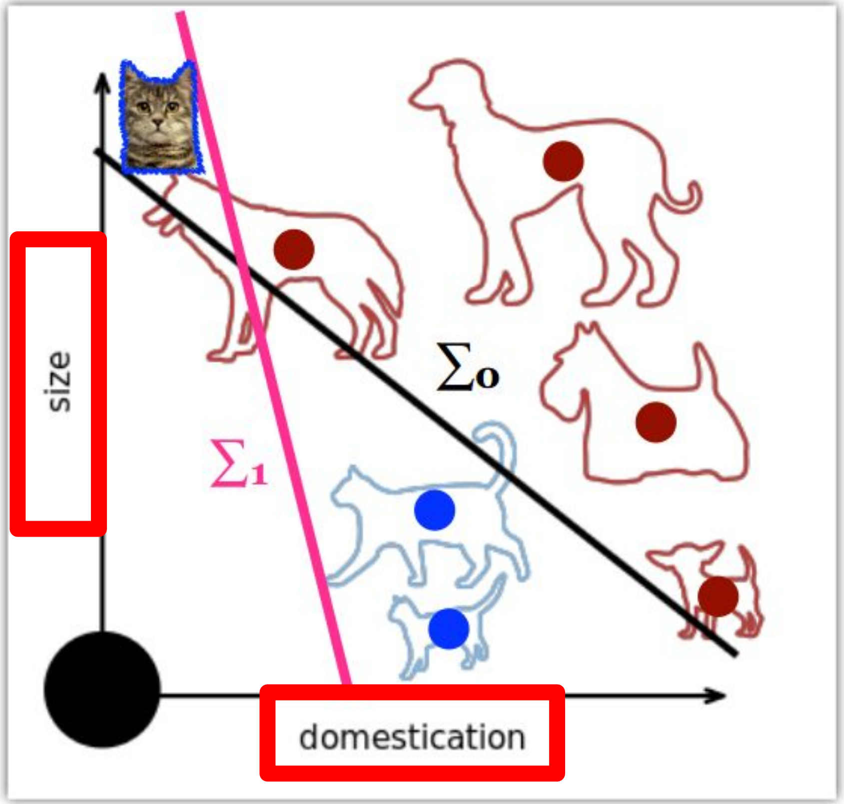
Recap: classification...

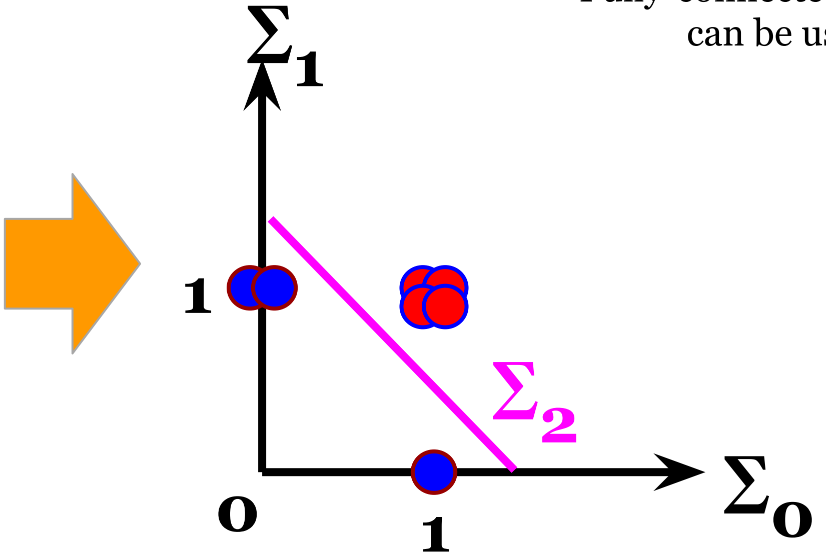


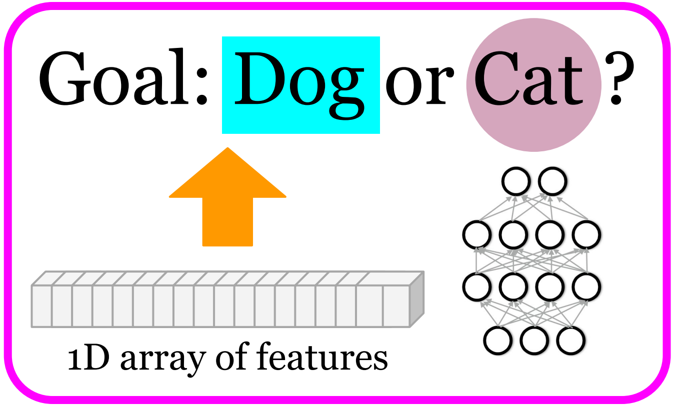




Recap: classification...





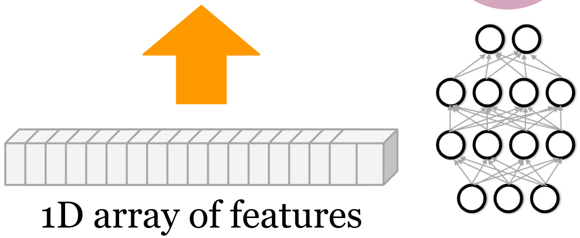


Fully-connected NN
can be useful.

Next step:

How?

Goal: Dog or Cat ?

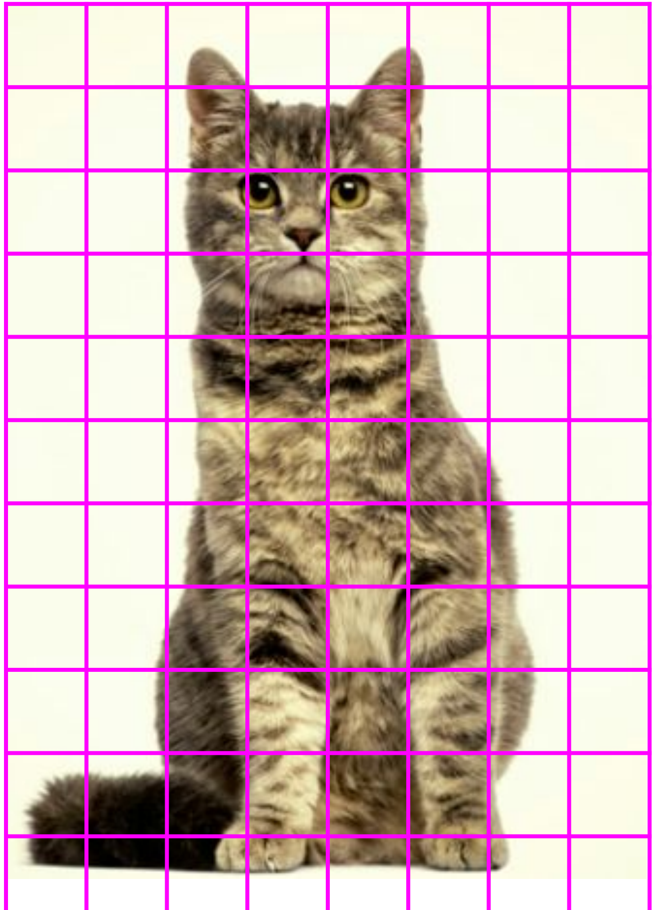


Fully-connected NN
can be useful.

How can we extract
“features” from image?

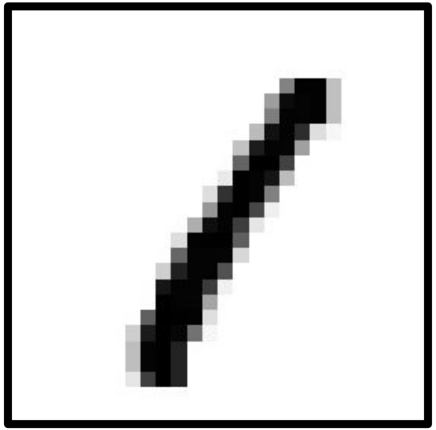
Deep Learning

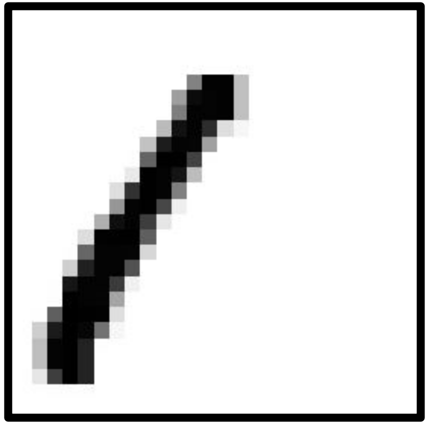
Next step:



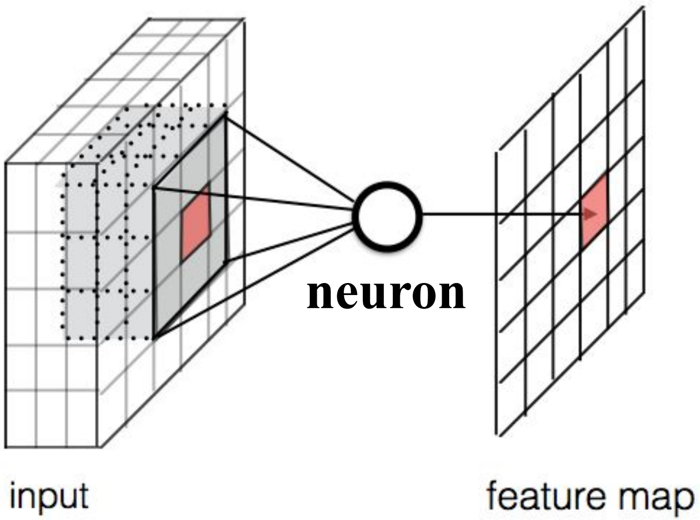
How about flattened image + MLP?

- For an input image of 100x100 pixels RGB image, how many weights does 1 neuron carry? 30,000 for just 1 neuron!
- Two image of the same cat, but in a different position w.r.t. the frame. Would neuron react the same? No! Position information is encoded!





CNNs introduce a ***limitation to MLP*** by forcing a neuron to look at only local, translation invariant features

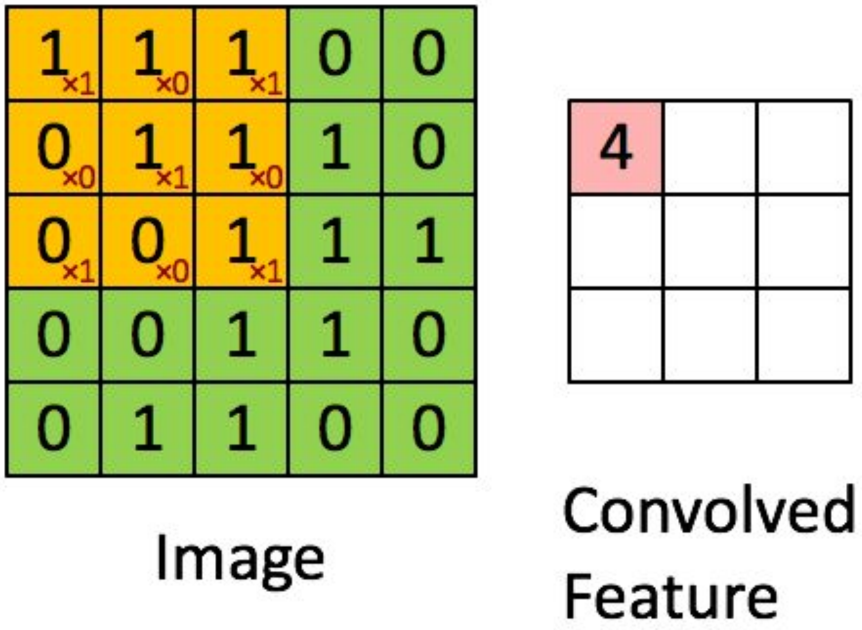


$$f_{i,j}(X) = \sigma\left(W_i \cdot X_j + b_i\right),$$

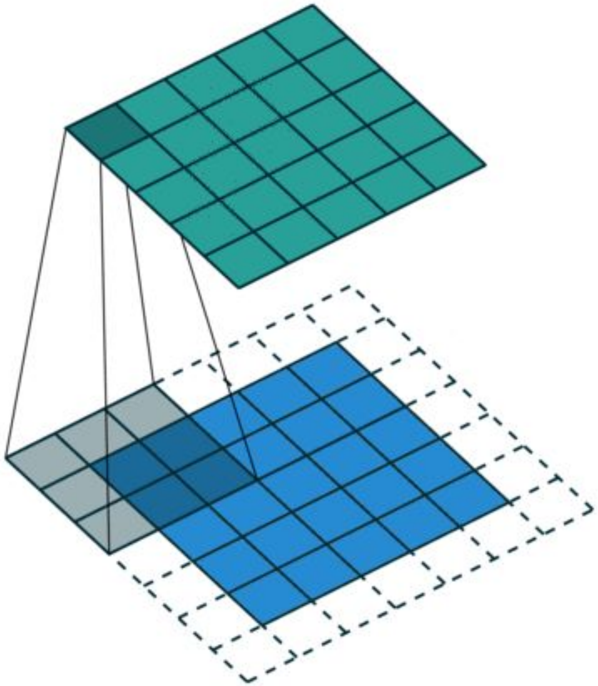
Still a linear transformation!
Weights=matrix, output=scalar
Analyze a fixed-size, local sub-matrix from the input.

- Traverse over 2D space to process the whole input
- **Locality** and **translation-invariance**

Convolution 3x3
Stride 1, no padding

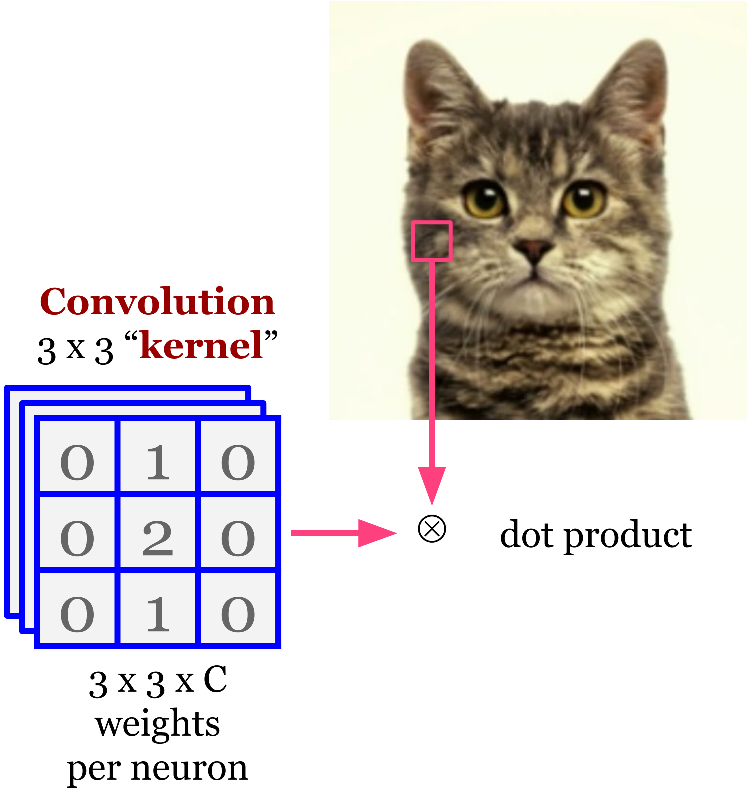


Convolution 3x3
Stride 1, padding 1

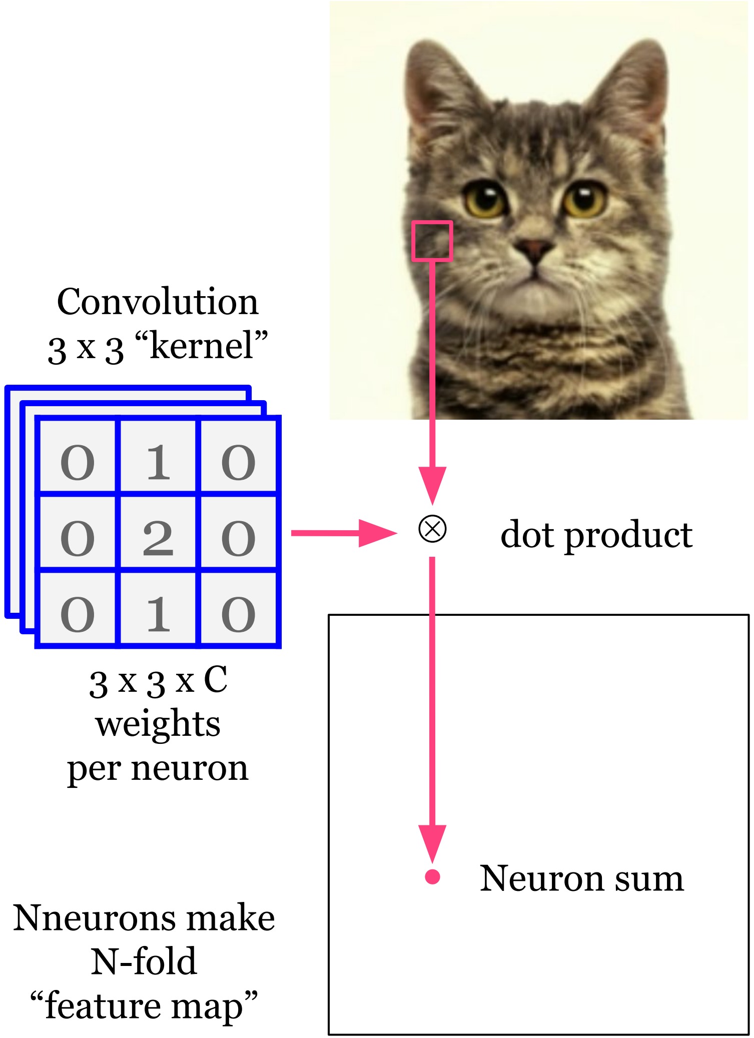


Goal: Dog or Cat ?

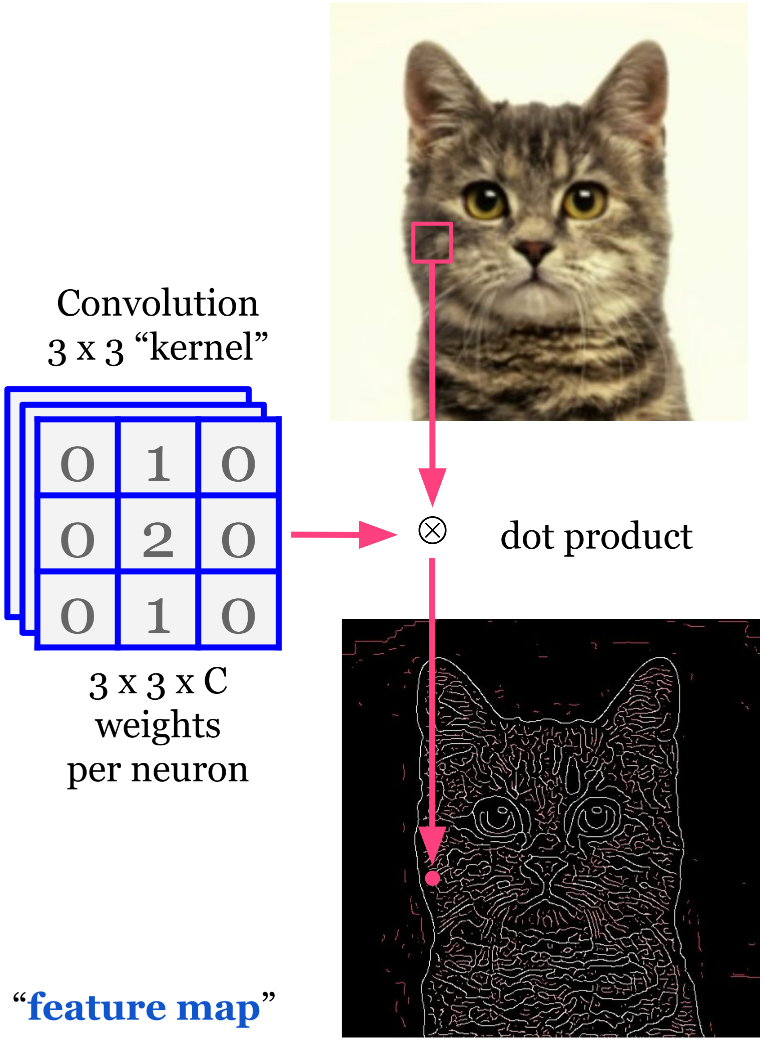
Goal: Dog or Cat?



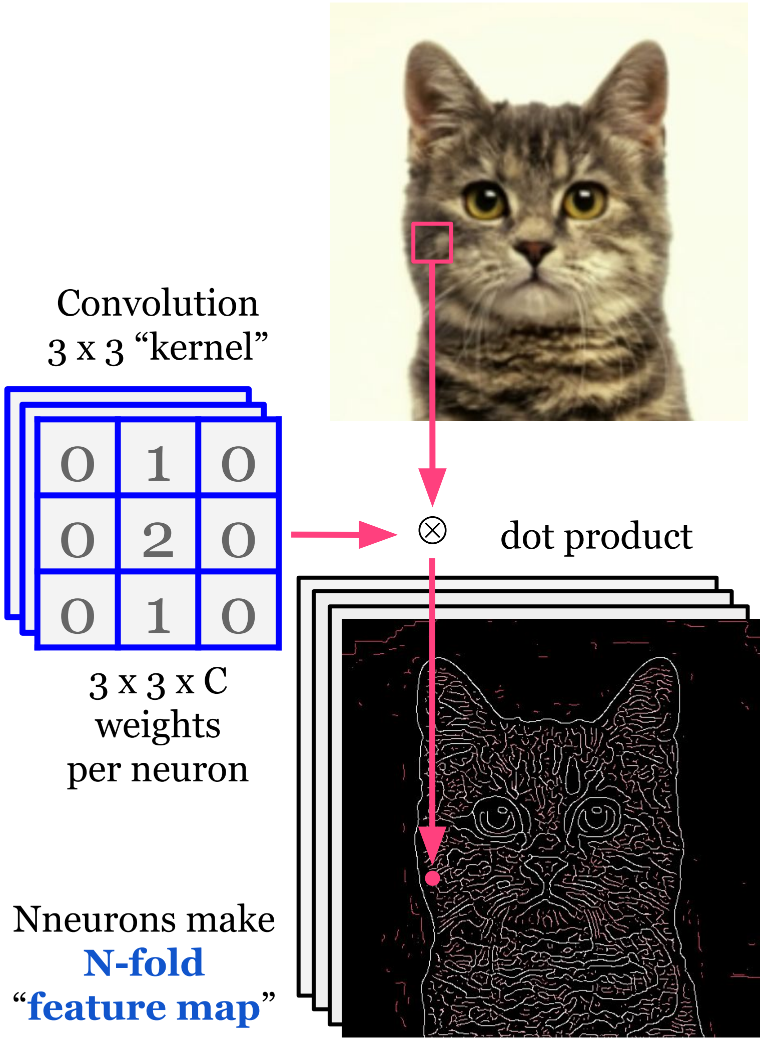
Goal: Dog or Cat?



Goal: Dog or Cat?

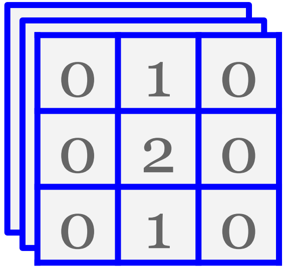


Goal: Dog or Cat ?

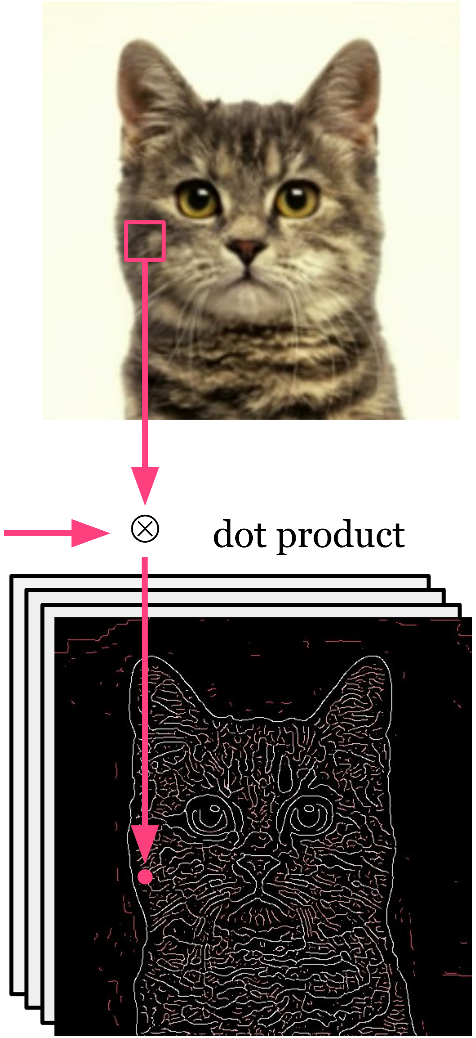


Goal: Dog or Cat ?

Convolution
3 x 3 “kernel”

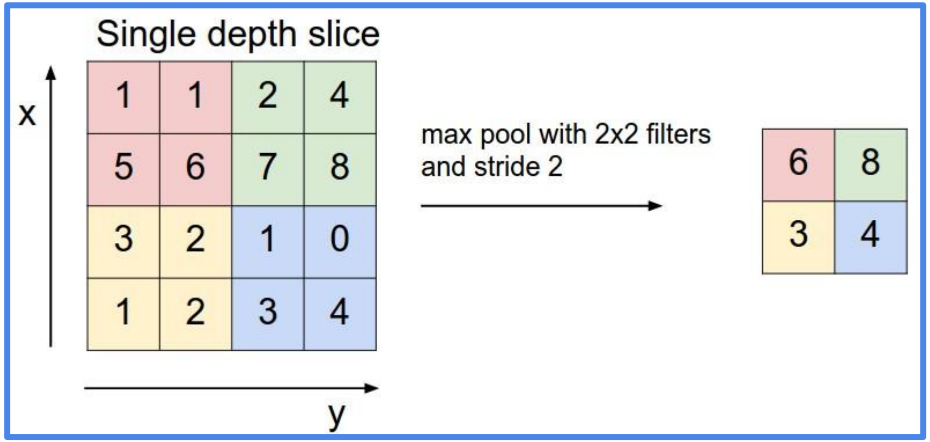


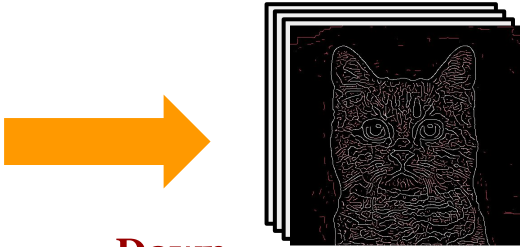
3 x 3 x C
weights
per neuron



Nneurons make
N-fold
“feature map”

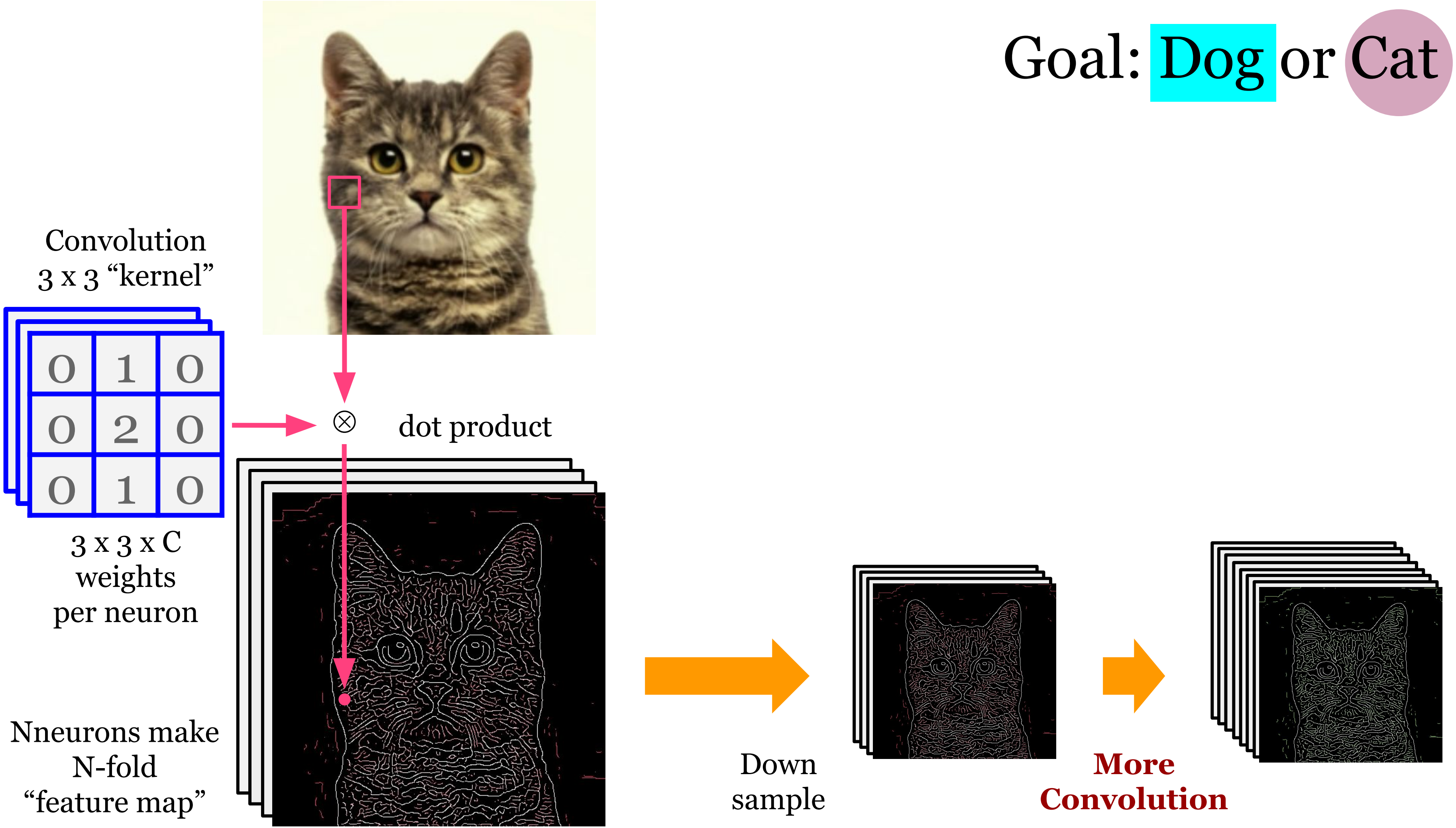
e.g) max pooling

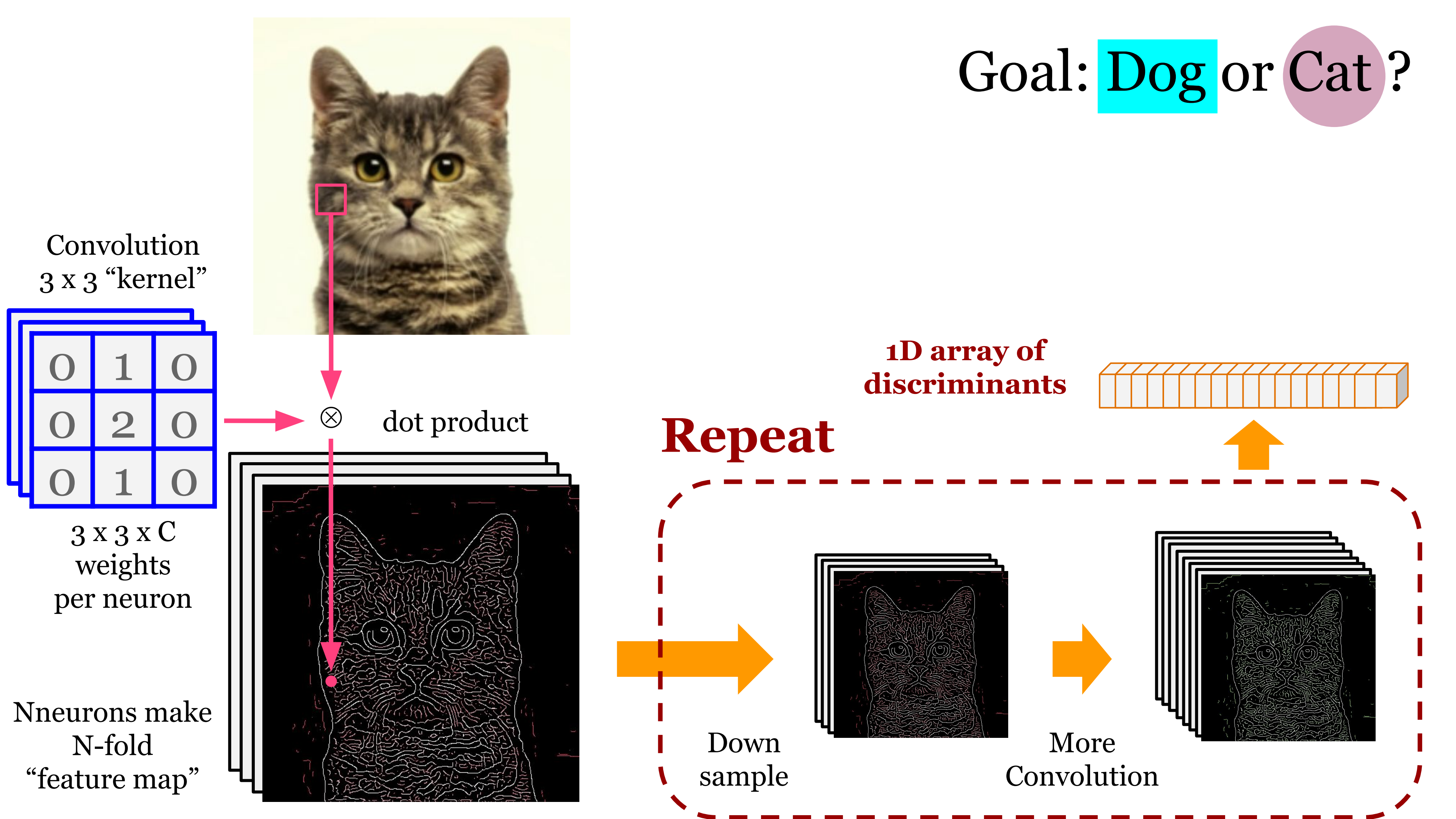




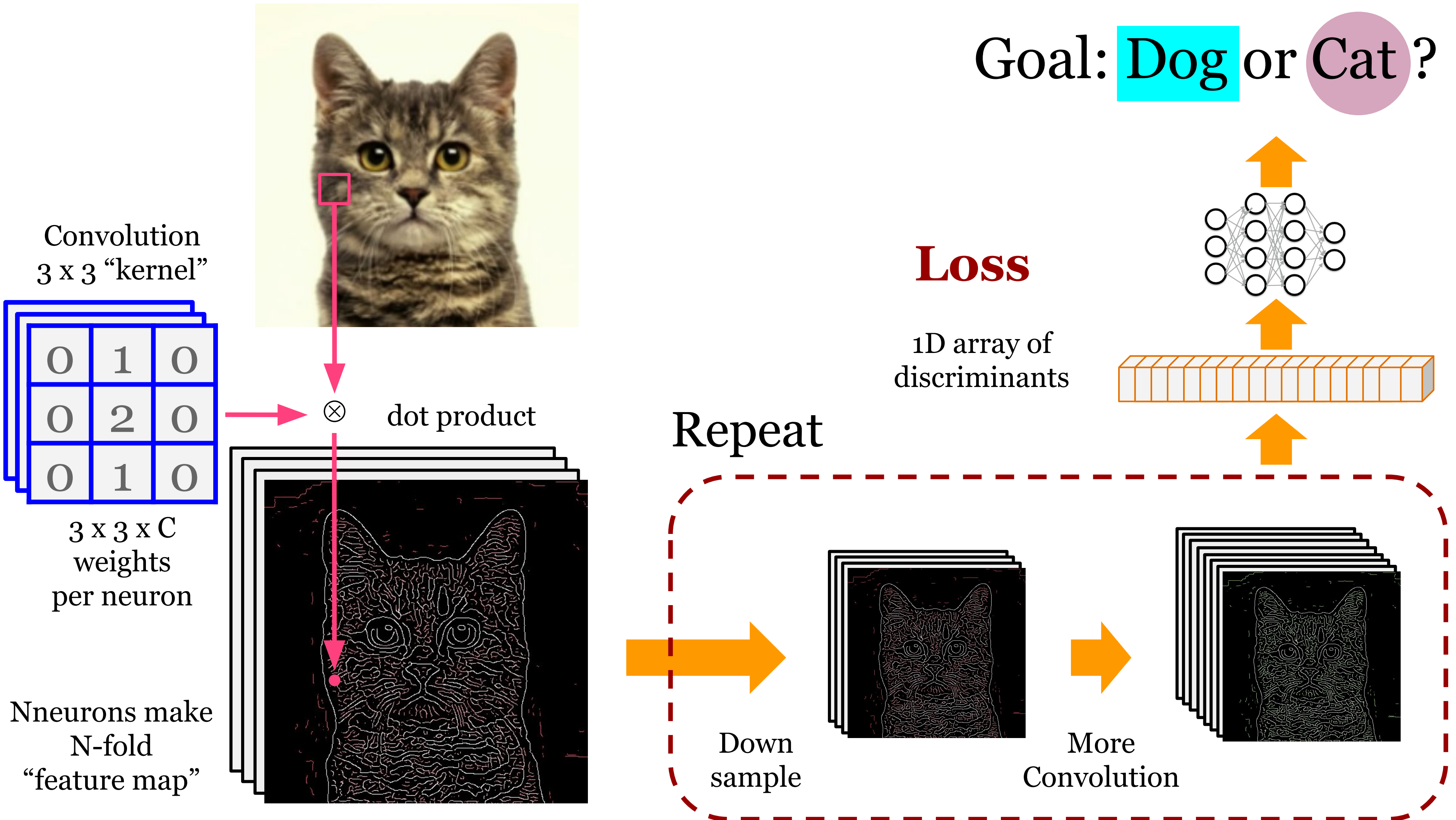
Down sample

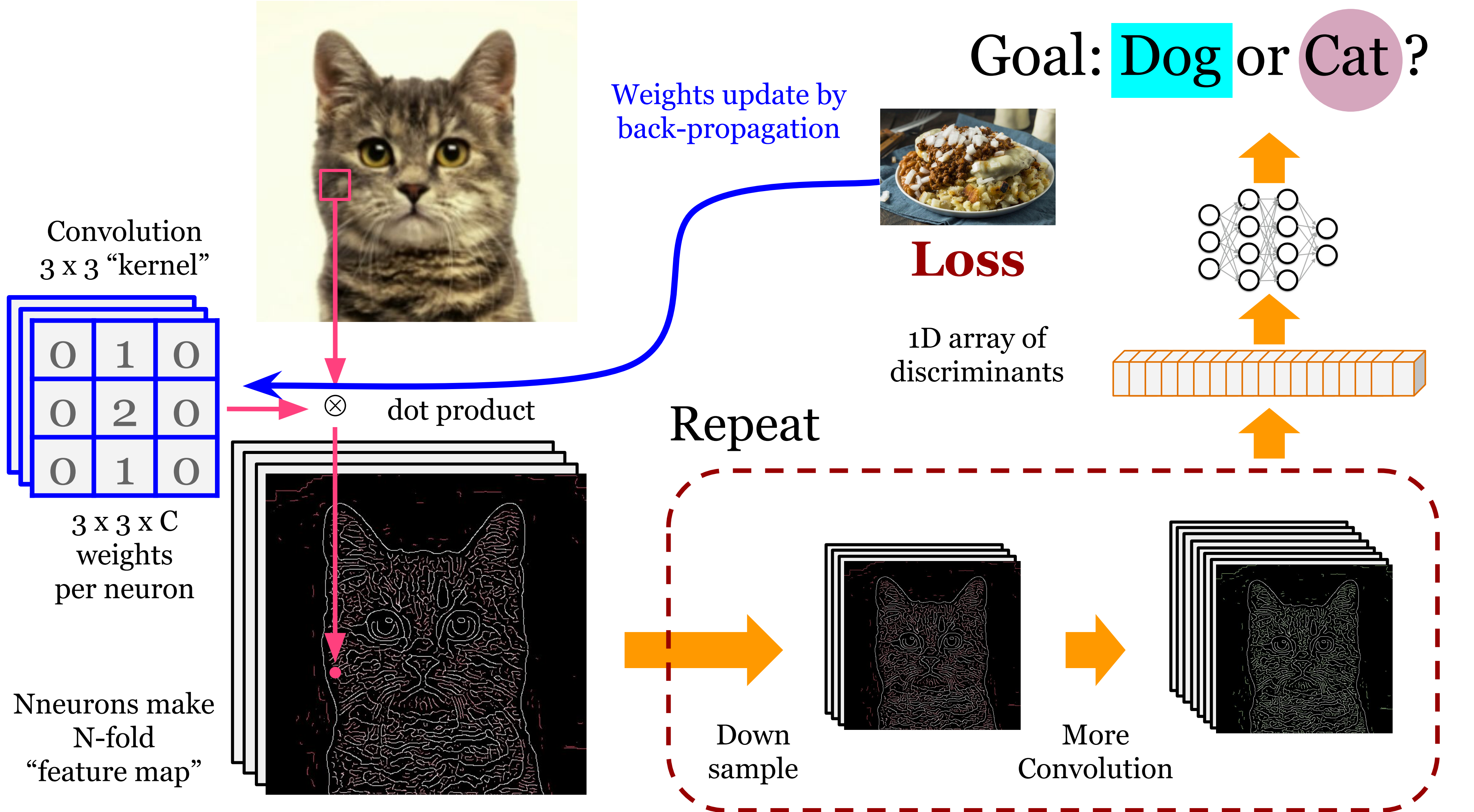
Goal: Dog or Cat ?





Goal: Dog or Cat ?
Convolution
3 x 3 "kernel"
0 1 0
0 2 0
0 1 0
3 x 3 x C
weights
per neuron
Nneurons make
N-fold
"feature map"
dot product
1D array of
discriminants
Repeat
Down
sample
More
Convolution


Goal: Dog or Cat ?
Convolution
3 x 3 “kernel”
0 1 0
0 2 0
0 1 0
3 x 3 x C
weights
per neuron
dot product
Nneurons make
N-fold
“feature map”
Loss
1D array of
discriminants
Repeat
Down
sample
More
Convolution

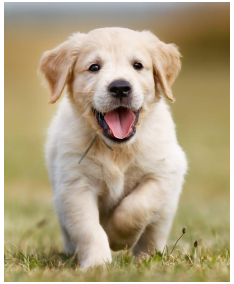


Goal: Dog or Cat ?
Weights update by back-propagation
Loss
Convolution 3 x 3 "kernel"
0 1 0
0 2 0
0 1 0
3 x 3 x C weights per neuron
dot product
1D array of discriminants
Repeat
Nneurons make N-fold "feature map"
Down sample
More Convolution

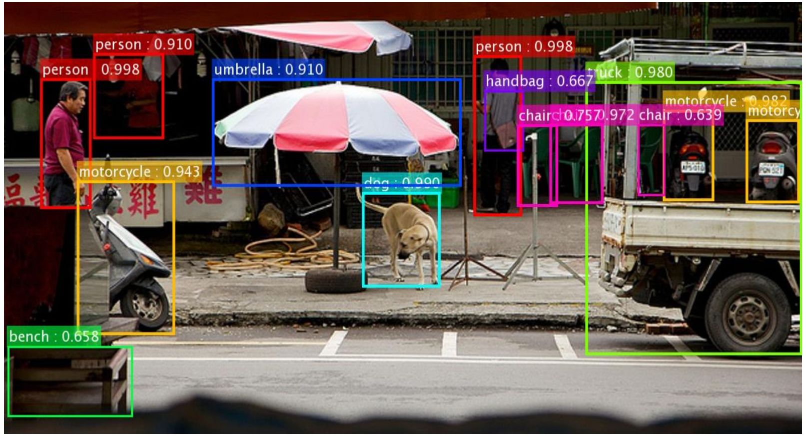
- **CNNs** are “**feature extraction machine**”
  - Consists of a “convolution layer” with “kernels”
  - A chain of parallelizable linear algebra operations
- **CNN** seen as a **geometrical data transformer**

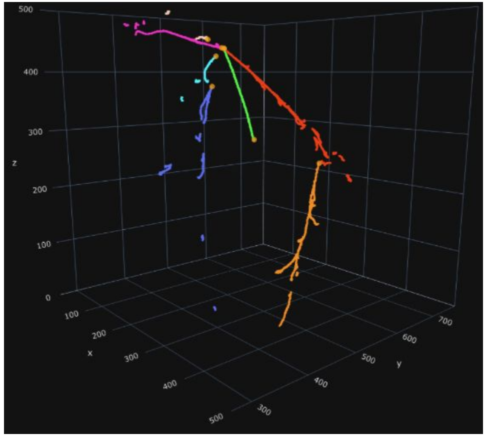
Covered





Later in this lecture



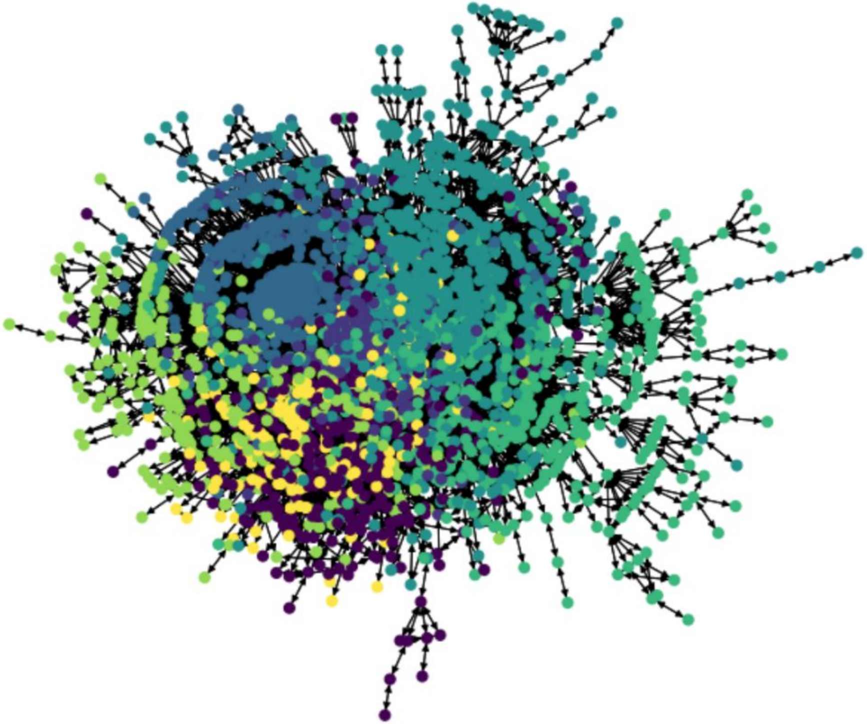


Graph Neural Networks

Graph Neural Networks

How do we analyze an unstructured data?

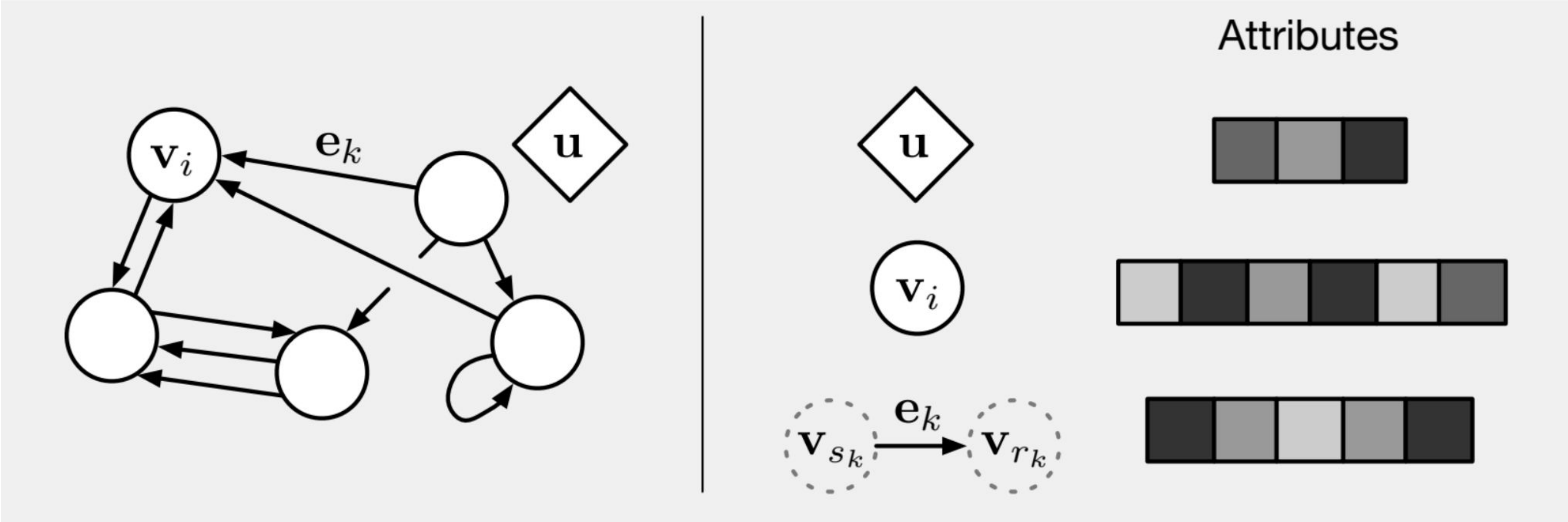
A social network



Citation/Reference Map

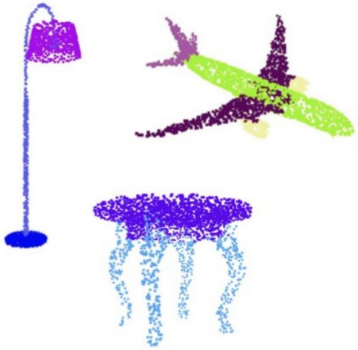
Graph Neural Networks

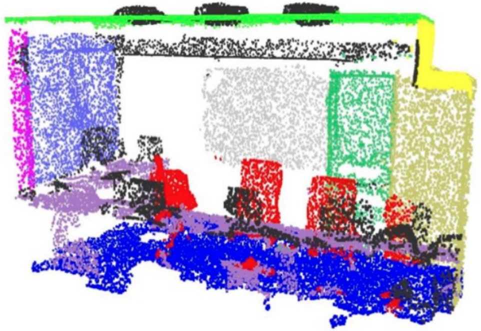
- A set of N_v nodes $\{\mathbf{v}_i\}$ that represent entities
- A set of N_e edges $\{\mathbf{e}_i\}$ that represent correlations between entities
- A global state $\mathbf{u}$ that represent the whole graph

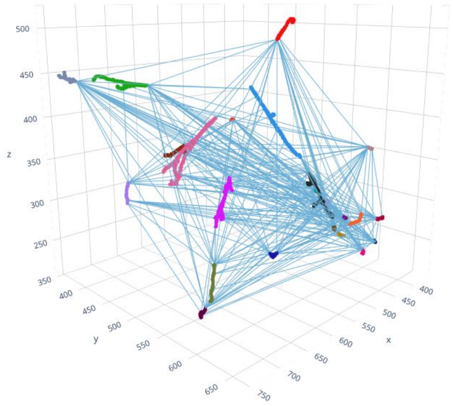


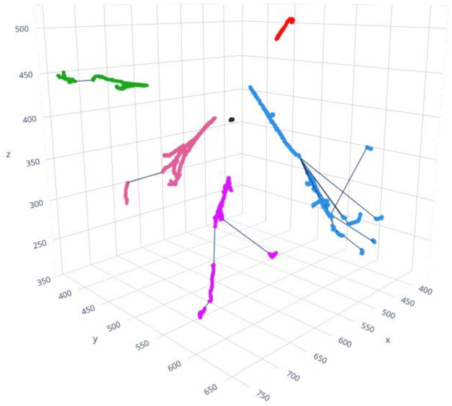
Tasks for GNNs

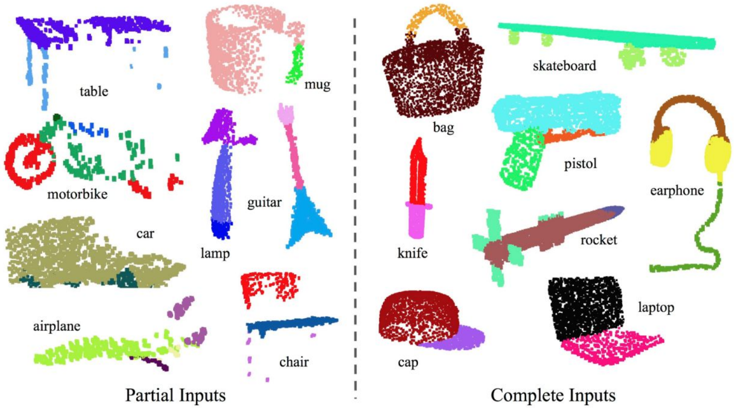
- Node classification/regression
- Edge classification/regression
- Graph classification/regression











Feature Engineering: How?

Recall CNN: a filter (neuron) analyzed locally connected pixel features

GNN: a filter analyze features of a target by including connected neighbors

One of the successful first attempts is called a "**Graph Convolution**"

- **Node feature matrix** V: shape (N, N_v) for N nodes with N_v features
- **Adjacency matrix** A: shape (N, N) for N nodes representing the connections between nodes (i.e. entries are 0 or 1)

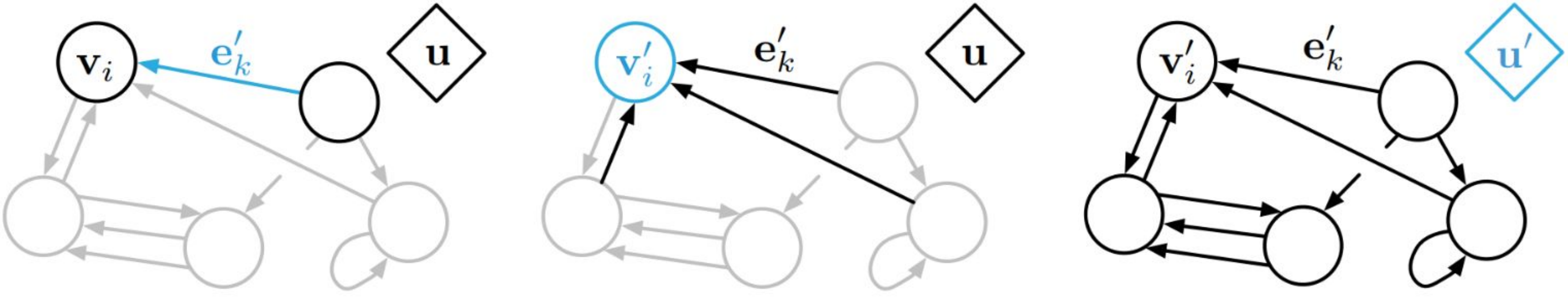
$$\mathbf{v}'_i = \sigma\left(\sum_{j=1}^{N} A_{ij}\mathbf{v}_j W\right)$$

A node is updated by incorporating features from itself and other connected nodes! However...

- Lacking edge and graph level features
- Only supports a simple sum of connected nodes

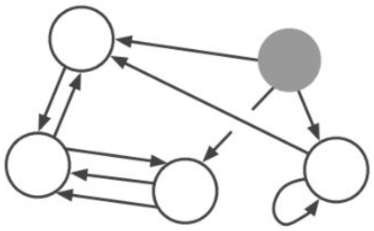
Generic Implementation: Message Passing

- **Edge update**: $\mathbf{e}'_k = \phi_e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right)$ takes the original edge, connected nodes, and global state into a learnable function ϕ_e
- **Node update**: $\mathbf{v}'_\ell = \phi_v\left(\rho_{e,v}\left(E'_\ell\right), \mathbf{v}_\ell, \mathbf{u}\right)$ takes the original node, global state, and edge features aggregated by $\rho_{e,v}$ into a learnable function ϕ_v
- **Global update**: $\mathbf{u}' = \phi_u\left(\rho_{e,u}\left(E'\right), \rho_{v,u}\left(V'\right), \mathbf{u}\right)$ takes the original global state, aggregated edge features, and aggregated node features into a learnable function ϕ_u

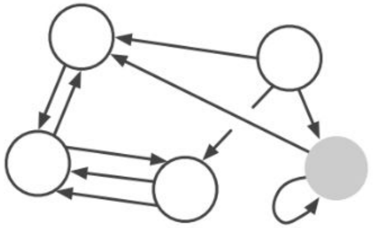


(a) Edge update (b) Node update (c) Global update

Generic Implementation: Message Passing

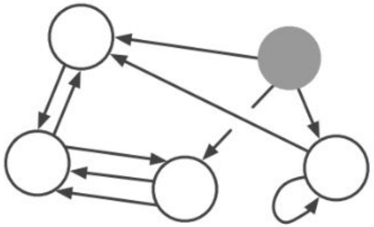


$m = 0$

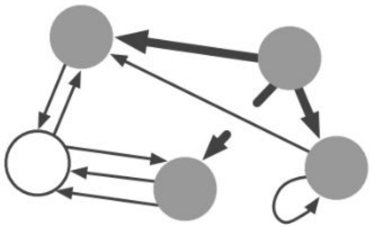


$m = 0$

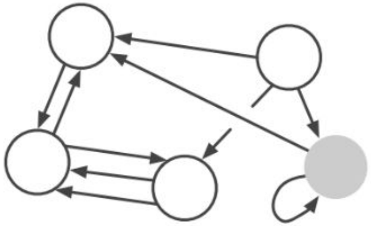
Generic Implementation: Message Passing



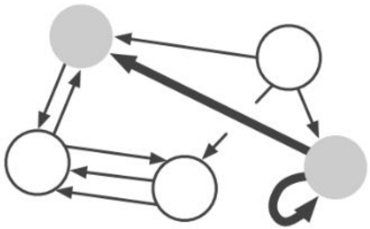
$m = 0$



$m = 1$

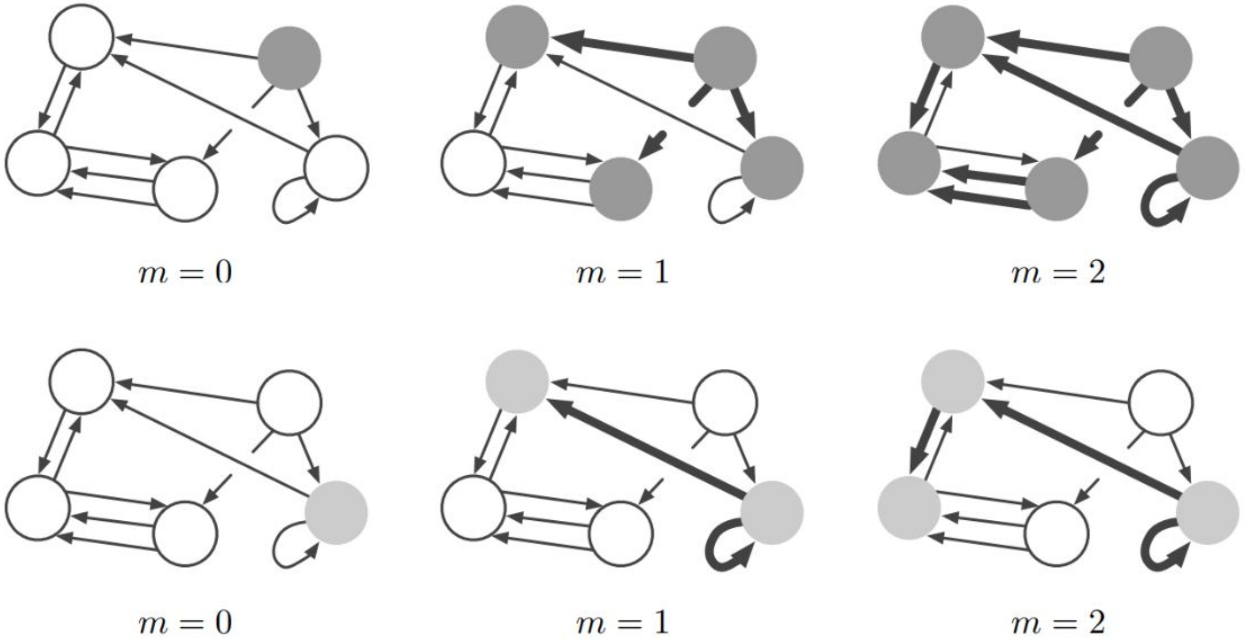


$m = 0$

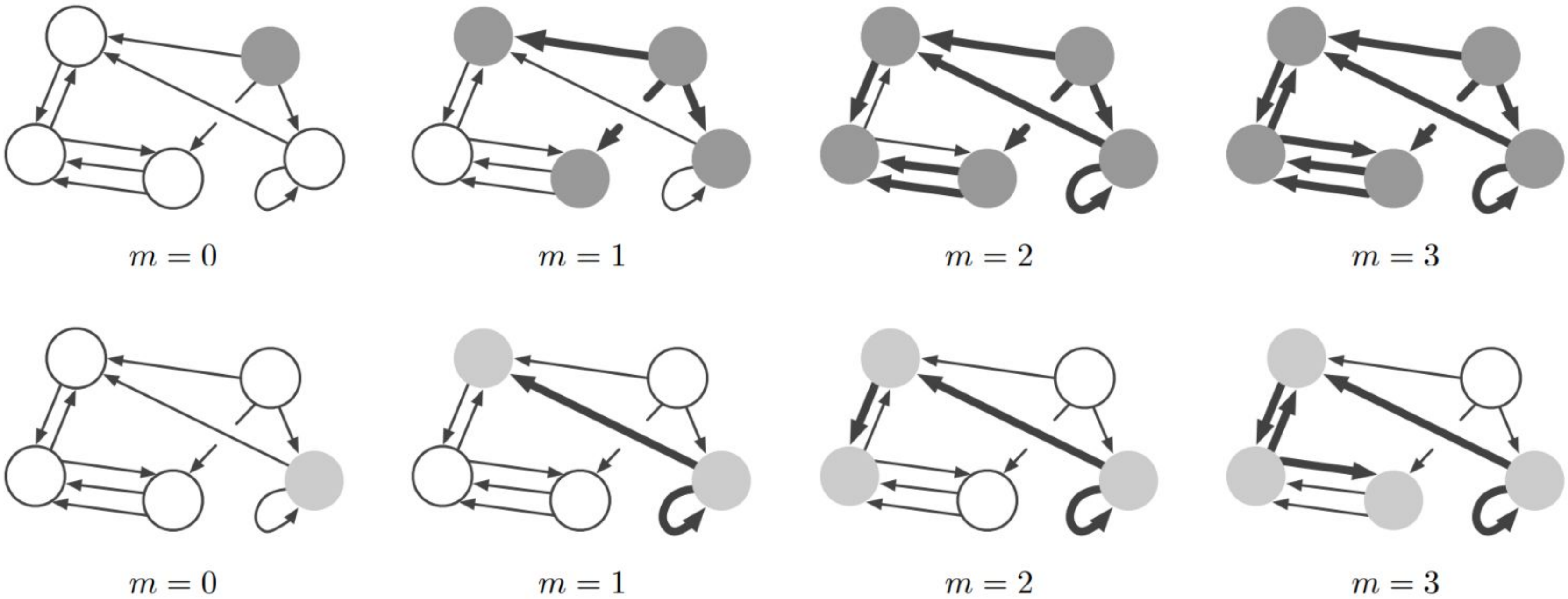


$m = 1$

Generic Implementation: Message Passing



Generic Implementation: Message Passing



Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations