BROWN
Computer Science

CS1951A: Data Science

Lecture 19: Deep learning

Lorenzo De Stefani

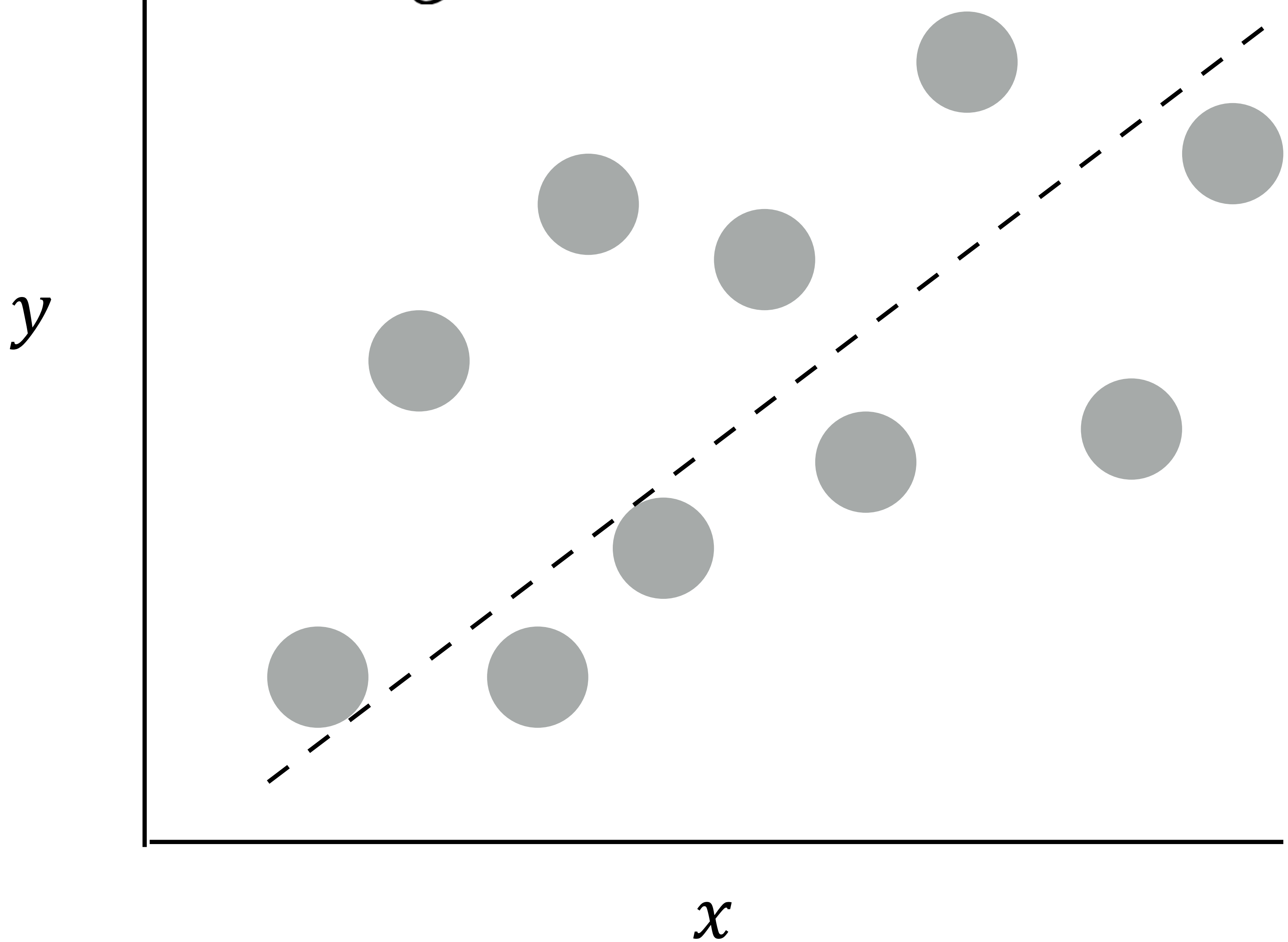
Spring 2022

Outline

- Deep Learning — roughly what is it?
- Why is it such a big deal (now)?
- Should I use deep learning?

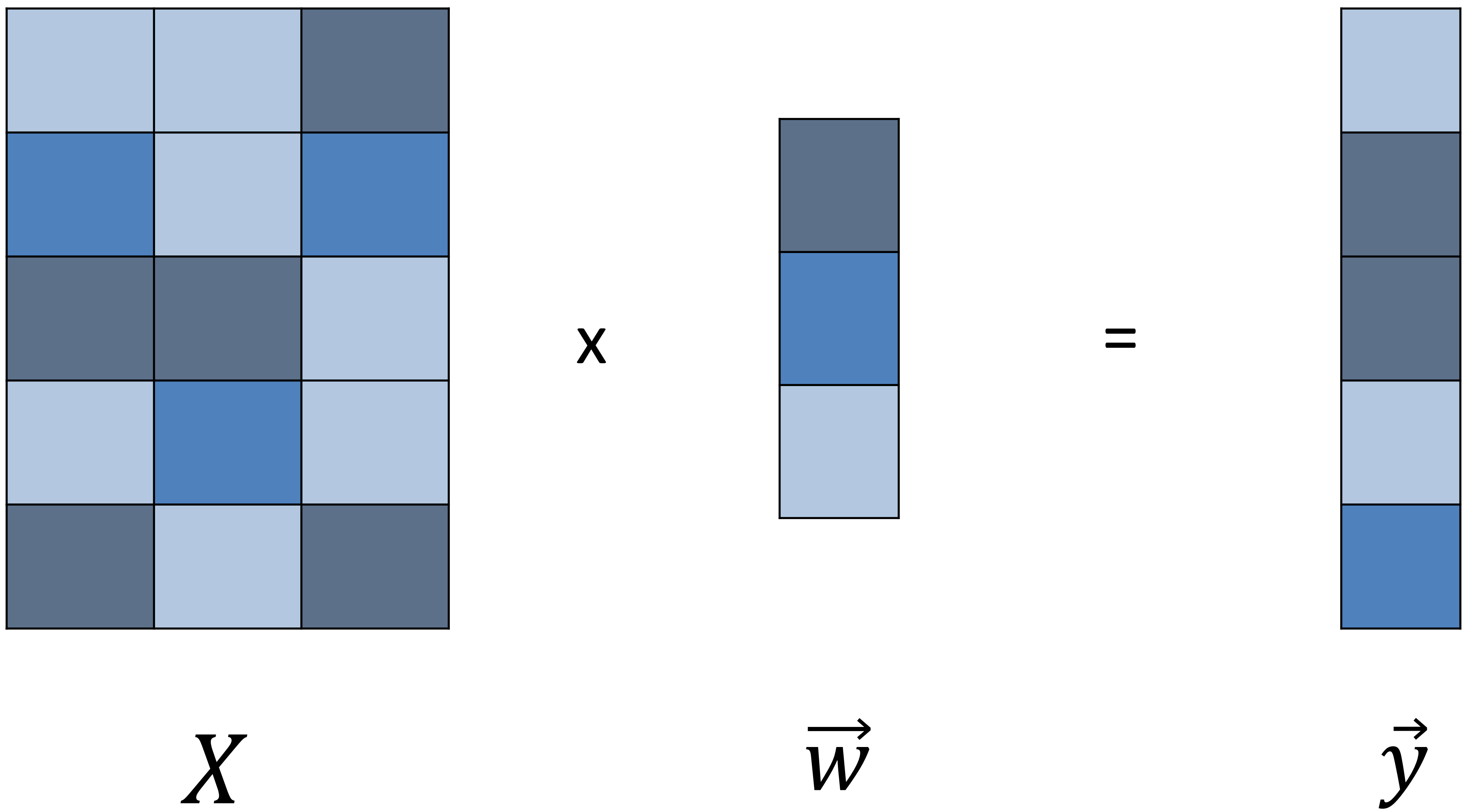
Linear Regression

$$y = wX + b$$



Linear Regression

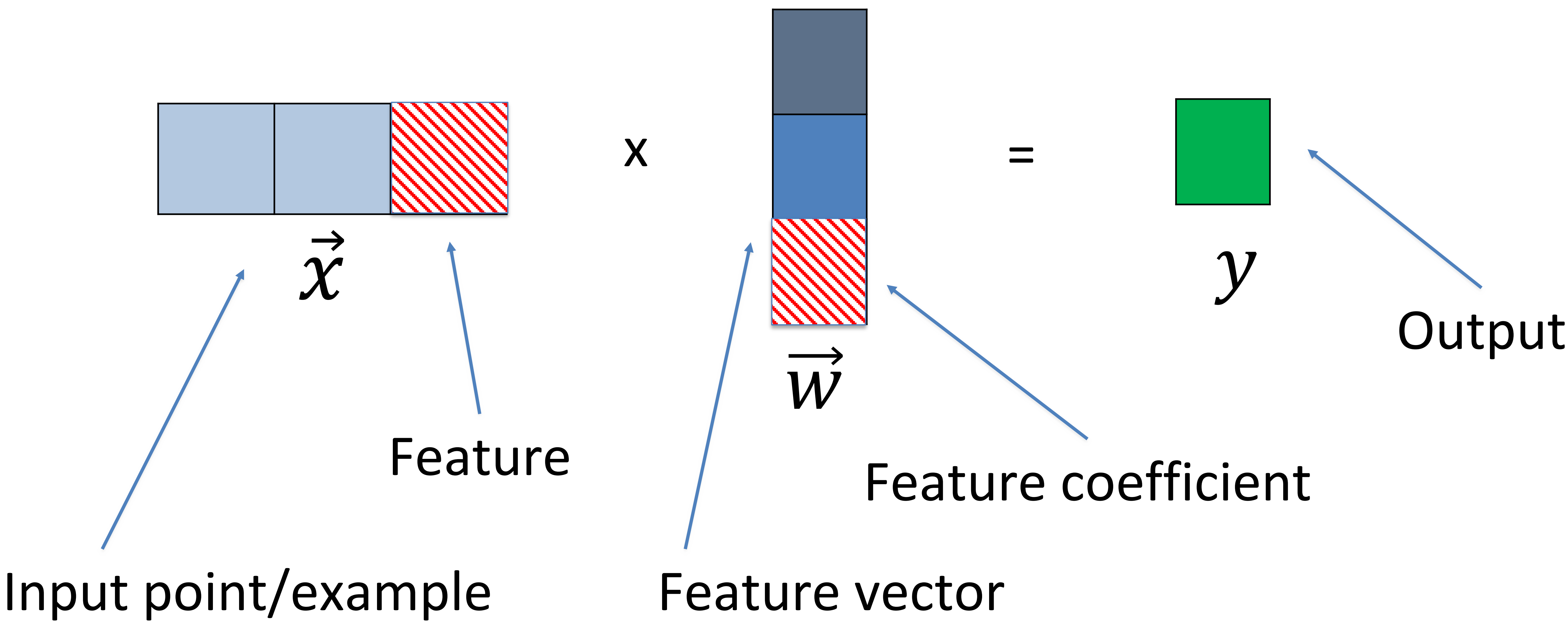
$$\vec{y} = \vec{w}X + b$$



The most basic "network"

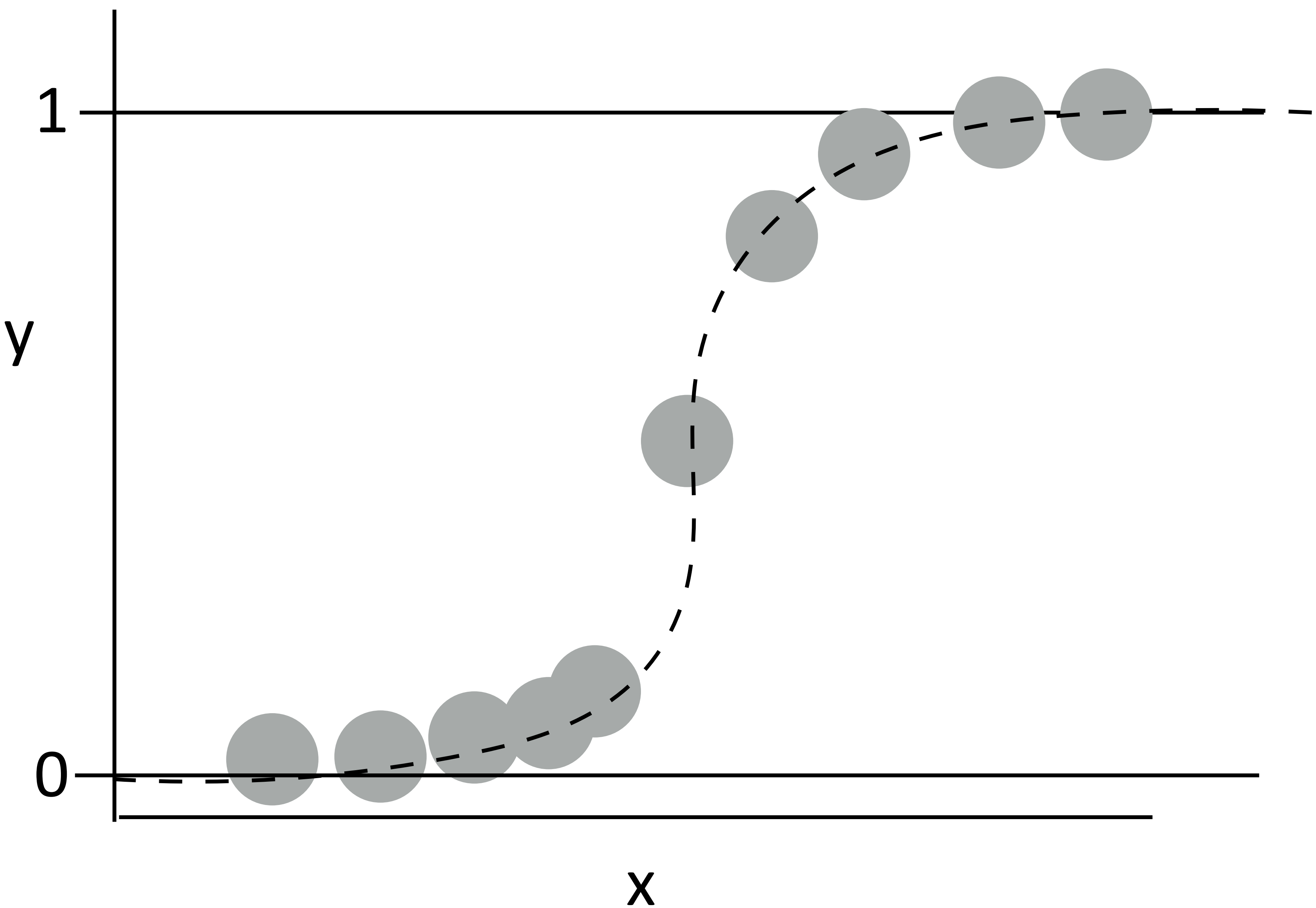
Perceptron: online linear regression

$$y = \vec{w} \cdot \vec{x}$$



Logistic Regression

$$y = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x})}}$$

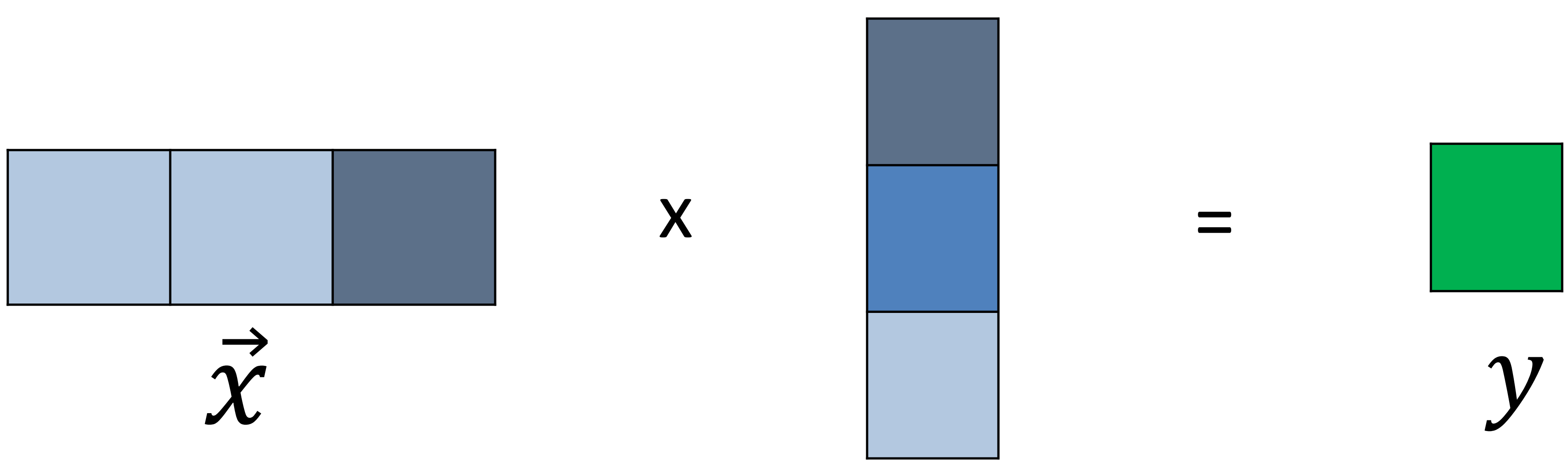


The most basic network

$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$

Threshold/activation value

- Activation function
- Non –linear behavior
- Perceptron algorithm
- Single-layer perceptron

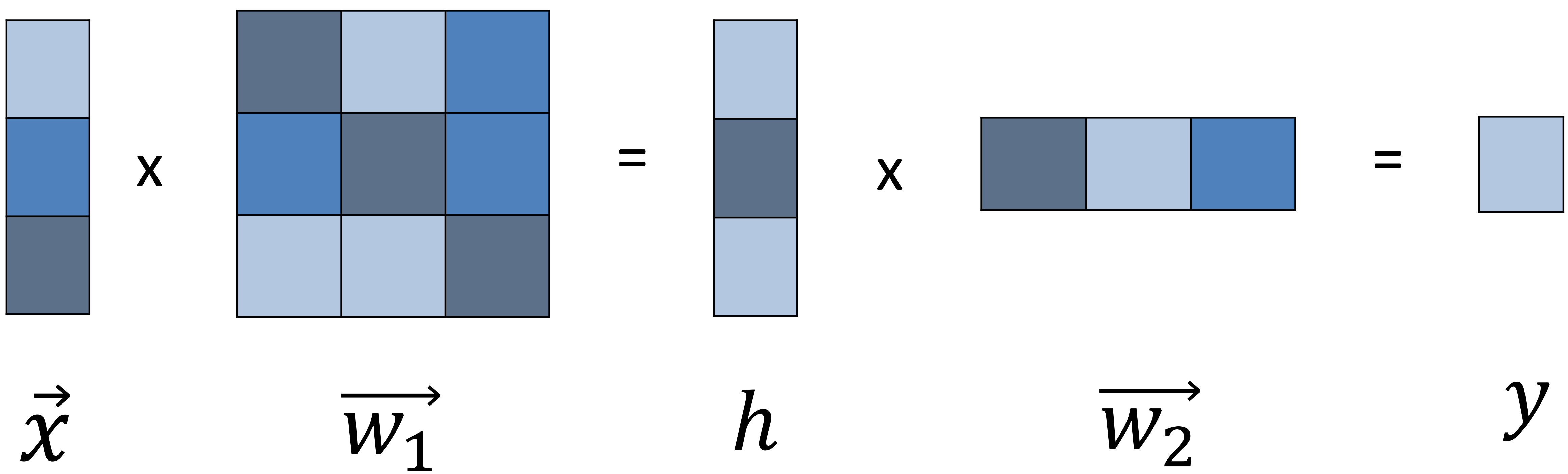


$\vec{x}$ $\vec{w}$ y

Binary Output

The most basic network

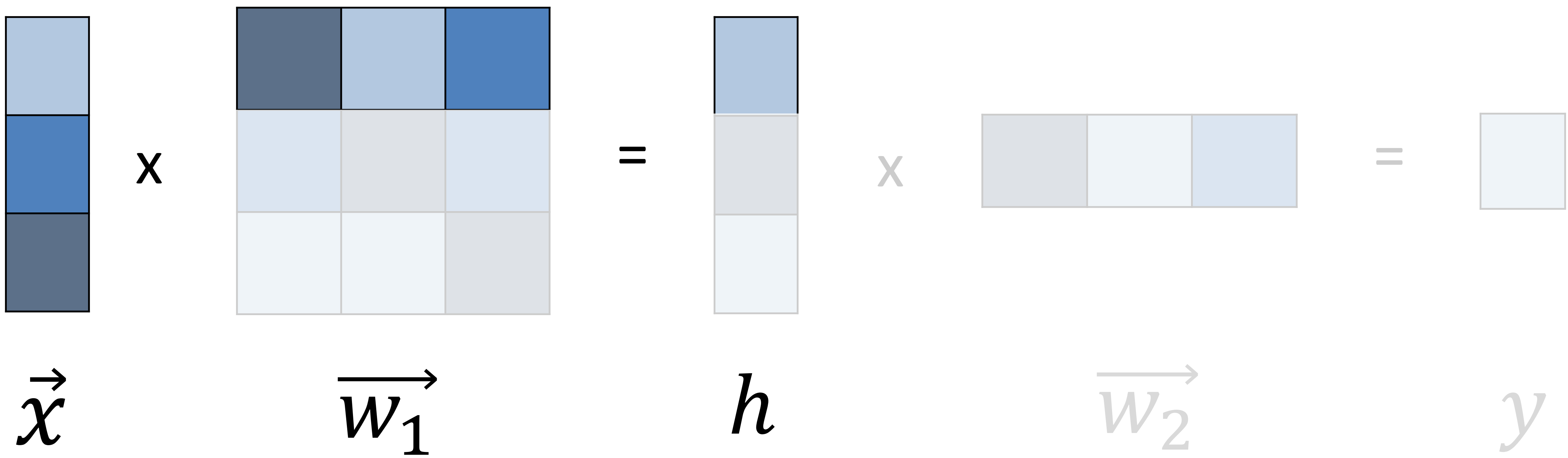
$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$



The most basic network

$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$

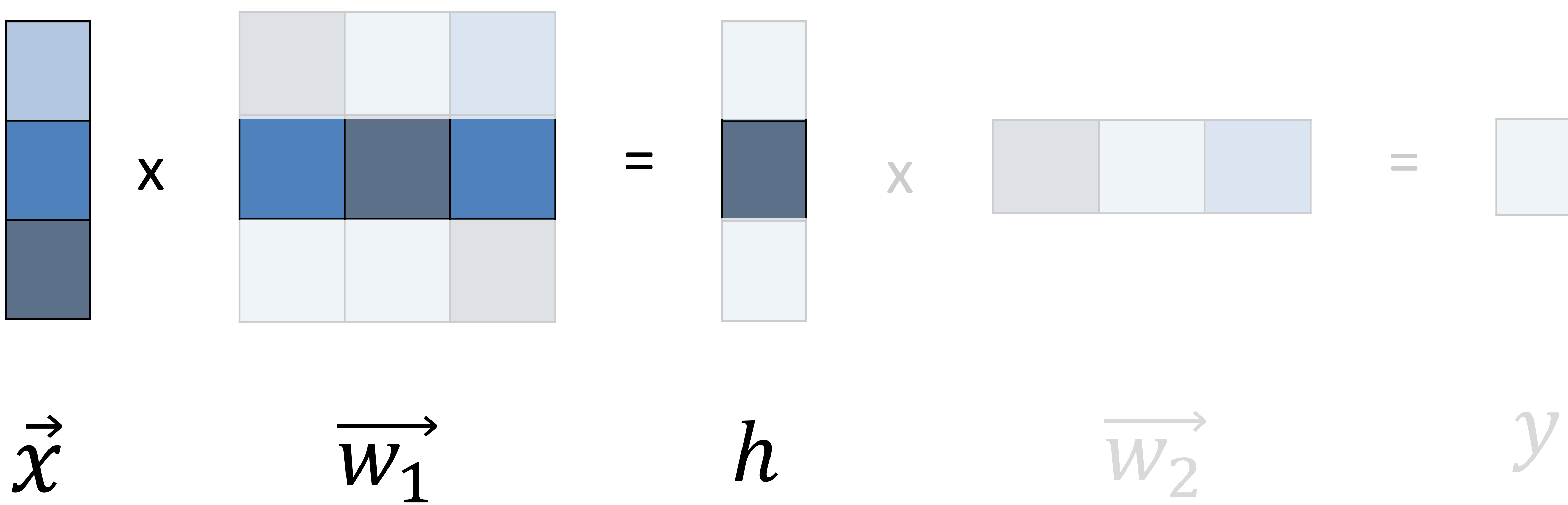
just a logistic regression



The most basic network

$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$

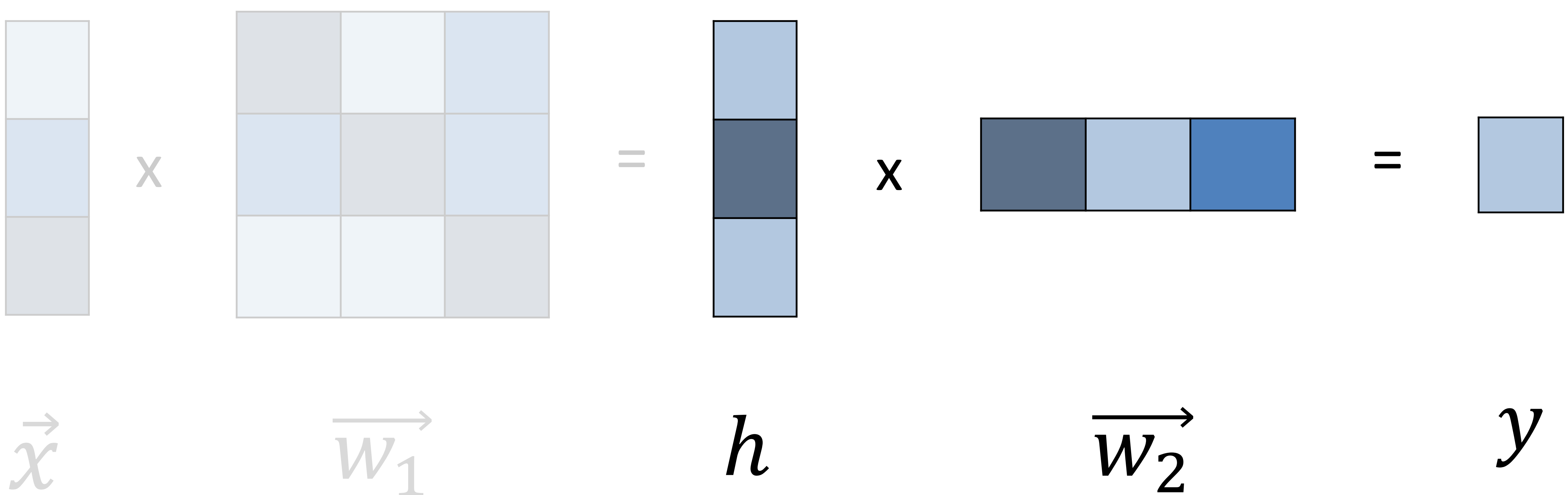
And another logistic regression



The most basic network

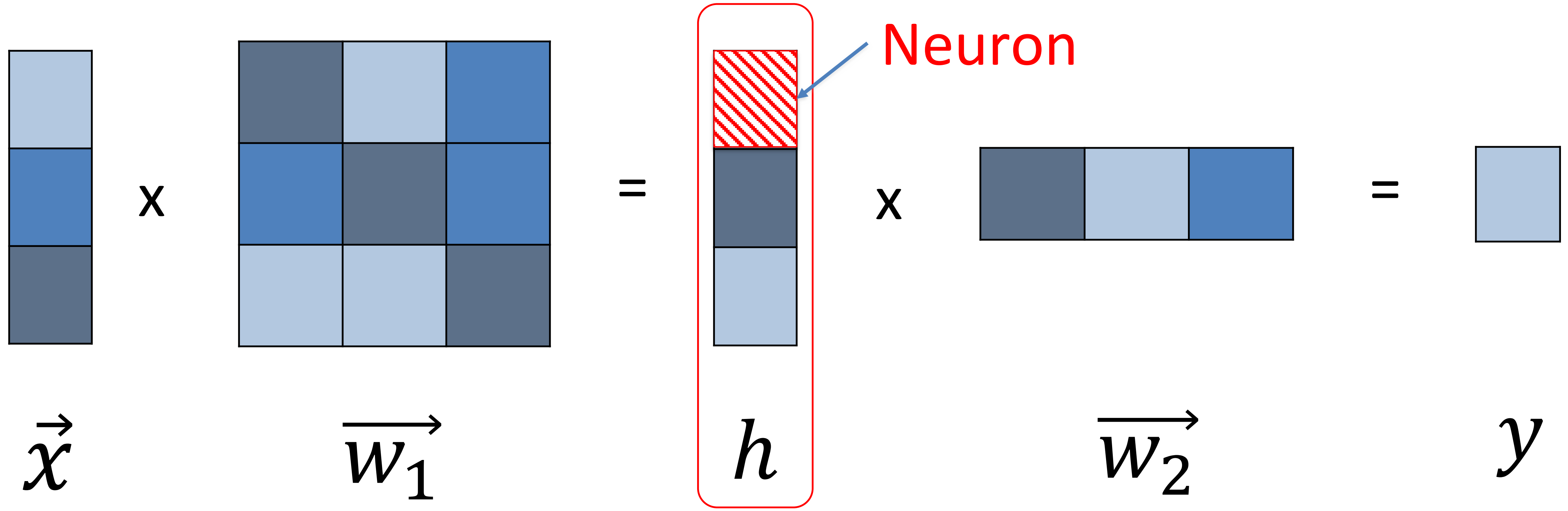
$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$

And another logistic regression



The most basic network

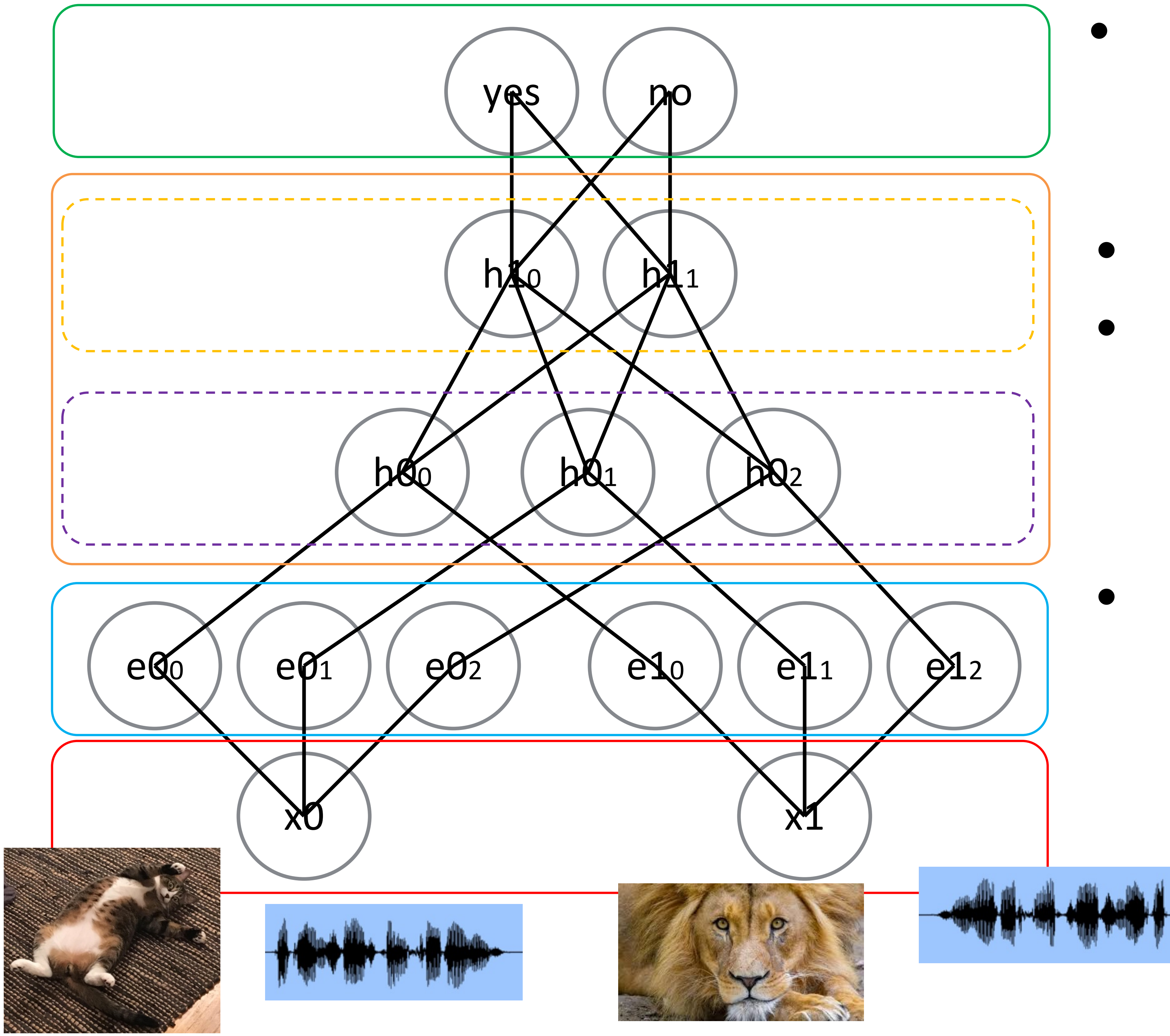
$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$



- Hidden State
- New feature vector
- Exploring different dimension of input

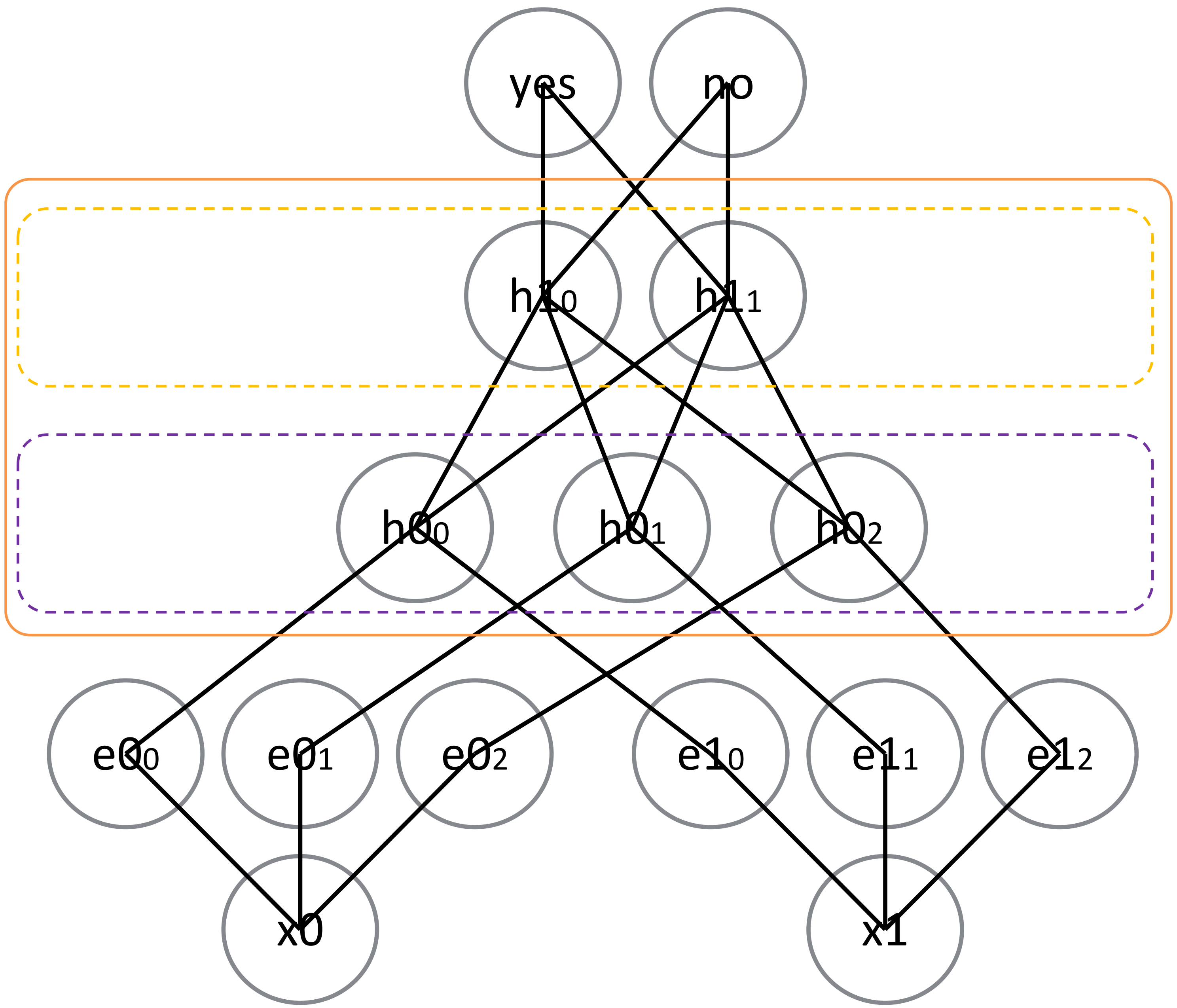
Same idea, illustrated a bit differently…

The most basic network



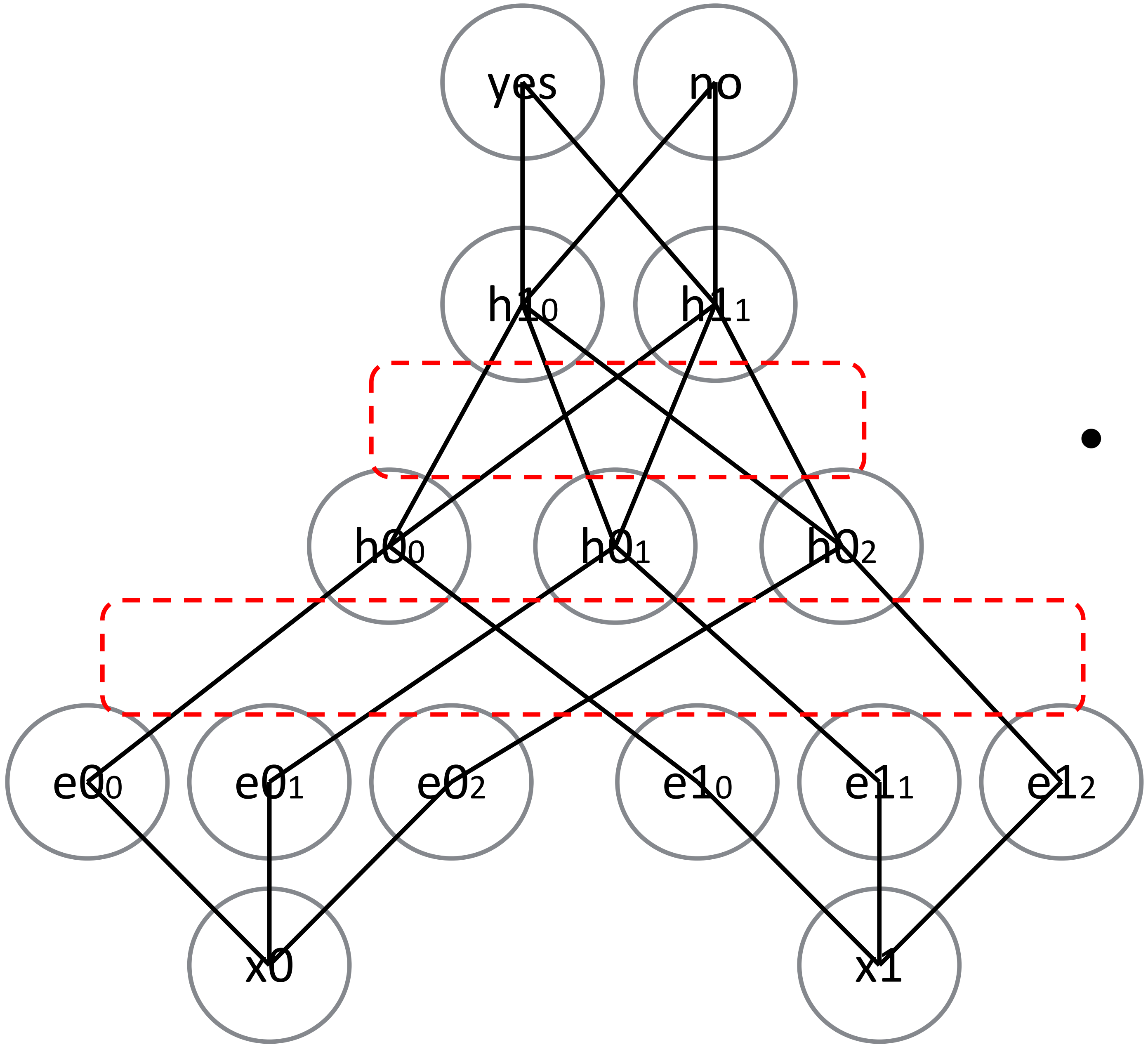
- Outputs
  Possible classes
- Hidden units
- One or more hidden layers/levels
- Input feature representation
  embedding
- Inputs

The most basic network



- Can have arbitrarily many
- Training more units requires more data
- In theory, “deeper” is not better than “wider”. In practice it seems it often is. We aren’t sure why yet…

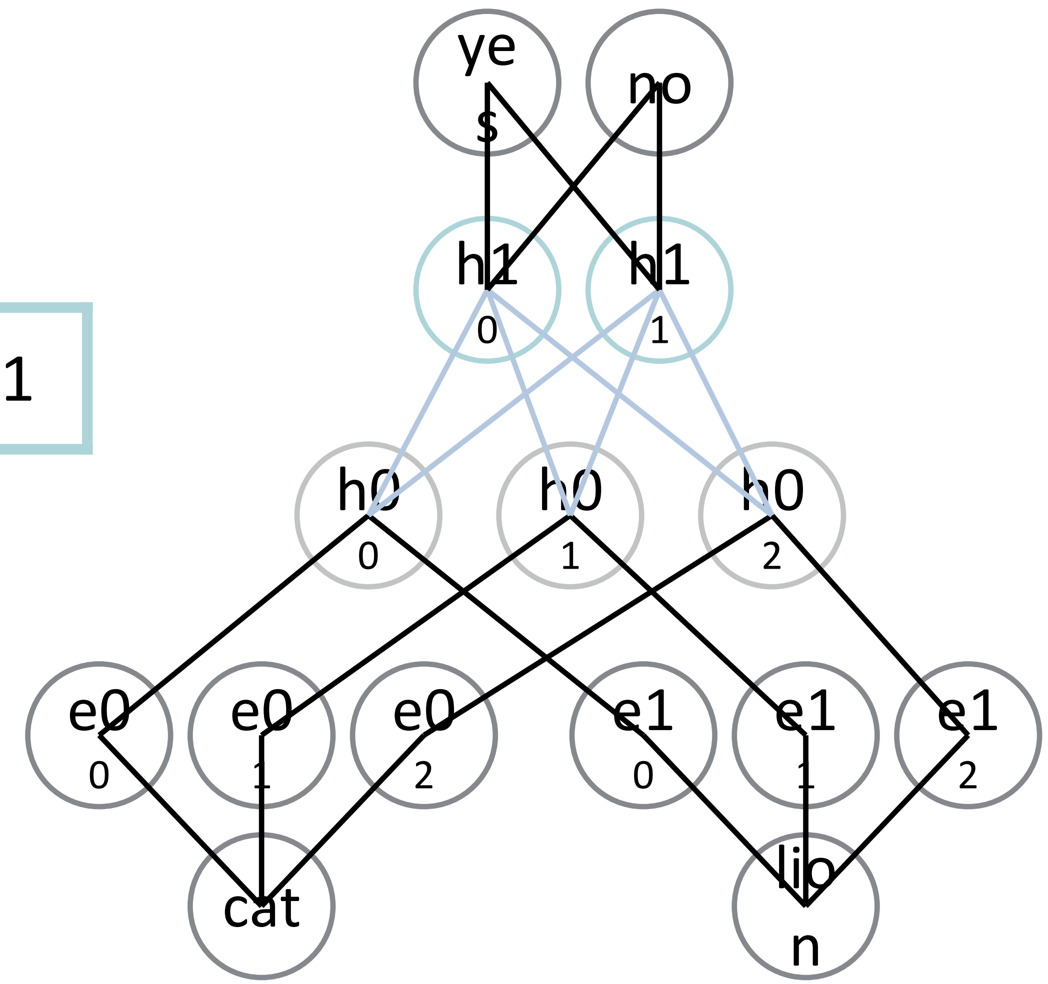
The most basic network



- Weight matrices

The most basic network

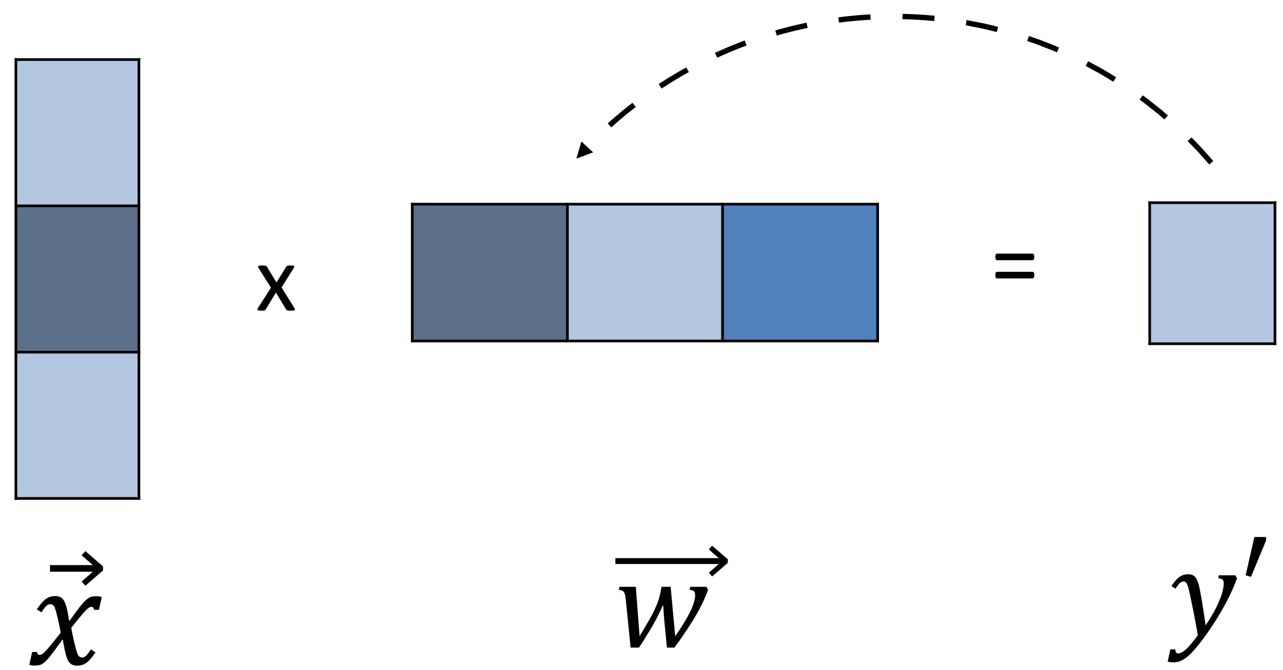
$$\begin{bmatrix} \mathbf{h00} & \mathbf{h01} & \mathbf{h02} \end{bmatrix} \times \begin{bmatrix} \mathbf{w00} & w01 \\ \mathbf{w10} & w11 \\ \mathbf{w20} & w21 \end{bmatrix} = \begin{bmatrix} h10 & h11 \end{bmatrix}$$



$$h10 = w00h00 + w10h01 + w20h02$$

$$h11 = w01h00 + w11h01 + w21h02$$

Training

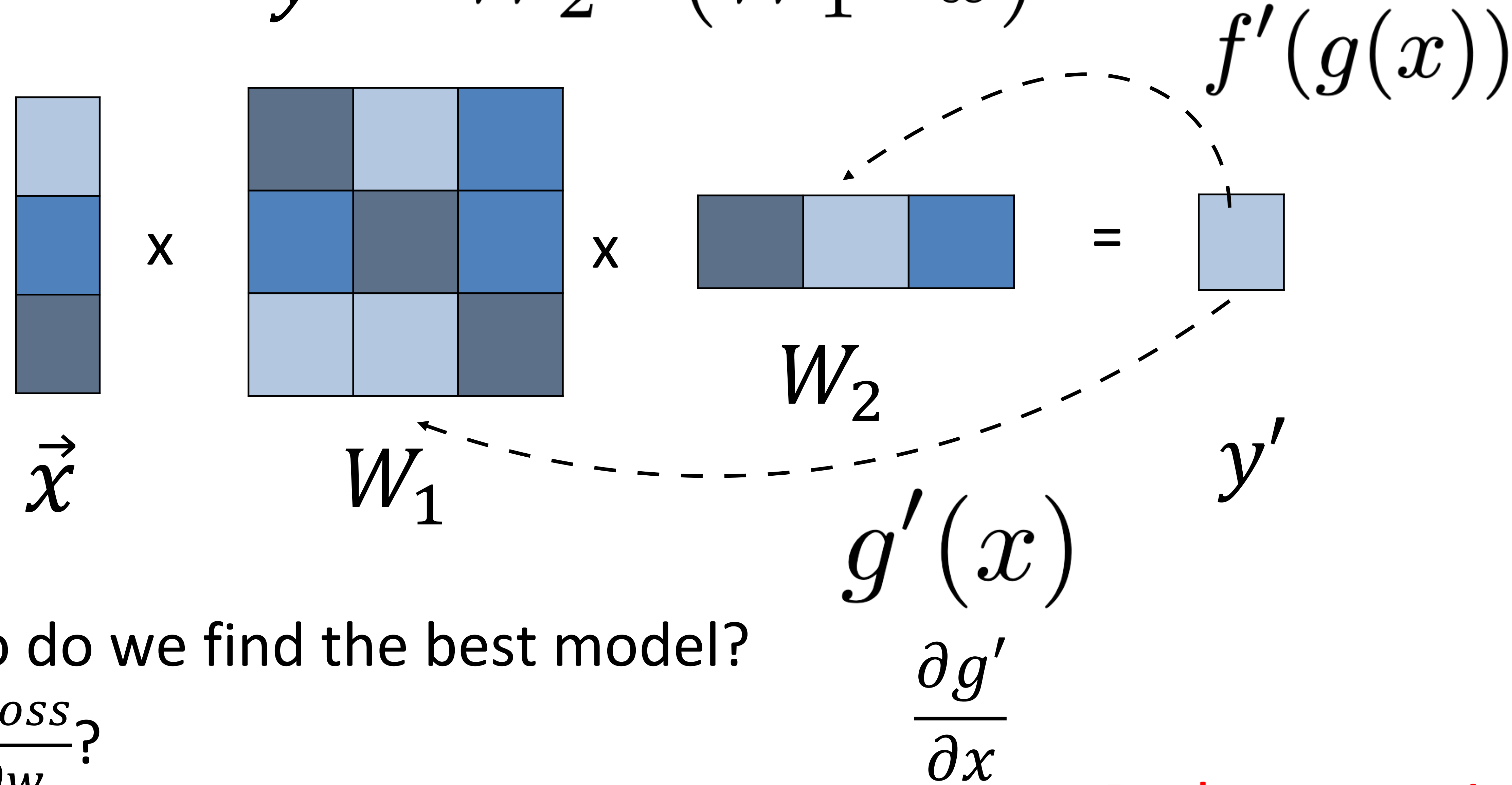


$$\frac{\partial \text{loss}}{\partial w}$$

$$loss(y, y')$$

Training

$$y' = W_2 \cdot (W_1 \cdot \vec{x})$$



- Ho do we find the best model?
- $\frac{\partial loss}{\partial w}$?
- $g'(\vec{x}) = W_1 \cdot \vec{x}$
- $f'(\vec{x}) = W_2 g'(\vec{x})$
- $\frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$

Backpropagation

backprop

Training

Loss Functions?

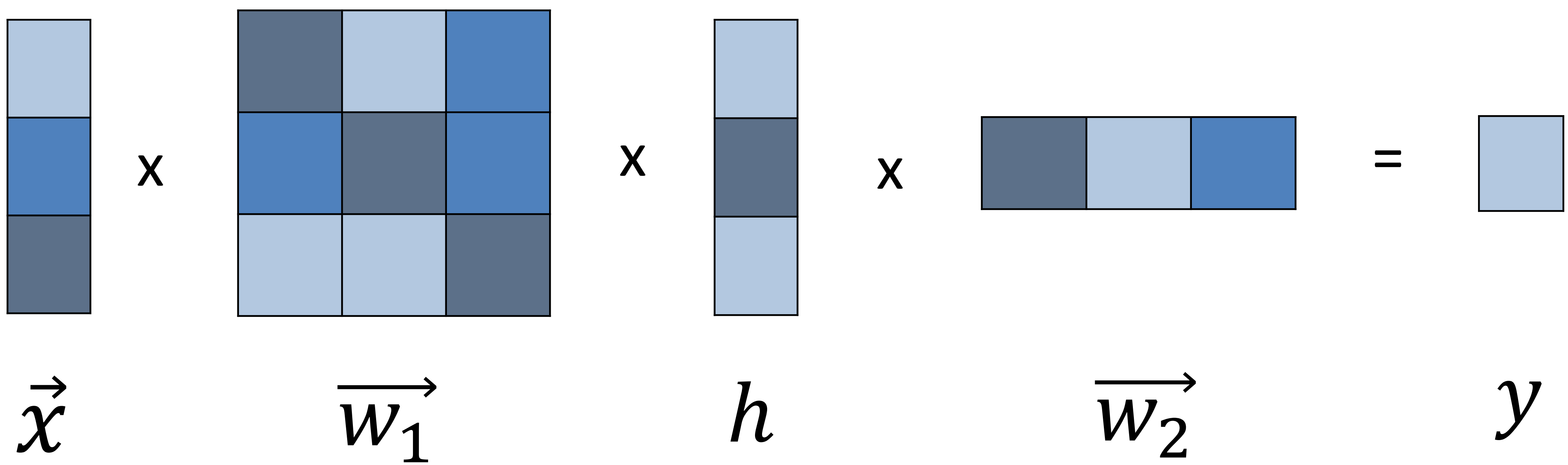
- Can be any differentiable function f(pred, true) = L
- Commonly MSE if true is continuous
- Commonly Cross Entropy if true is categorical

x w Backpropagation y

backprop

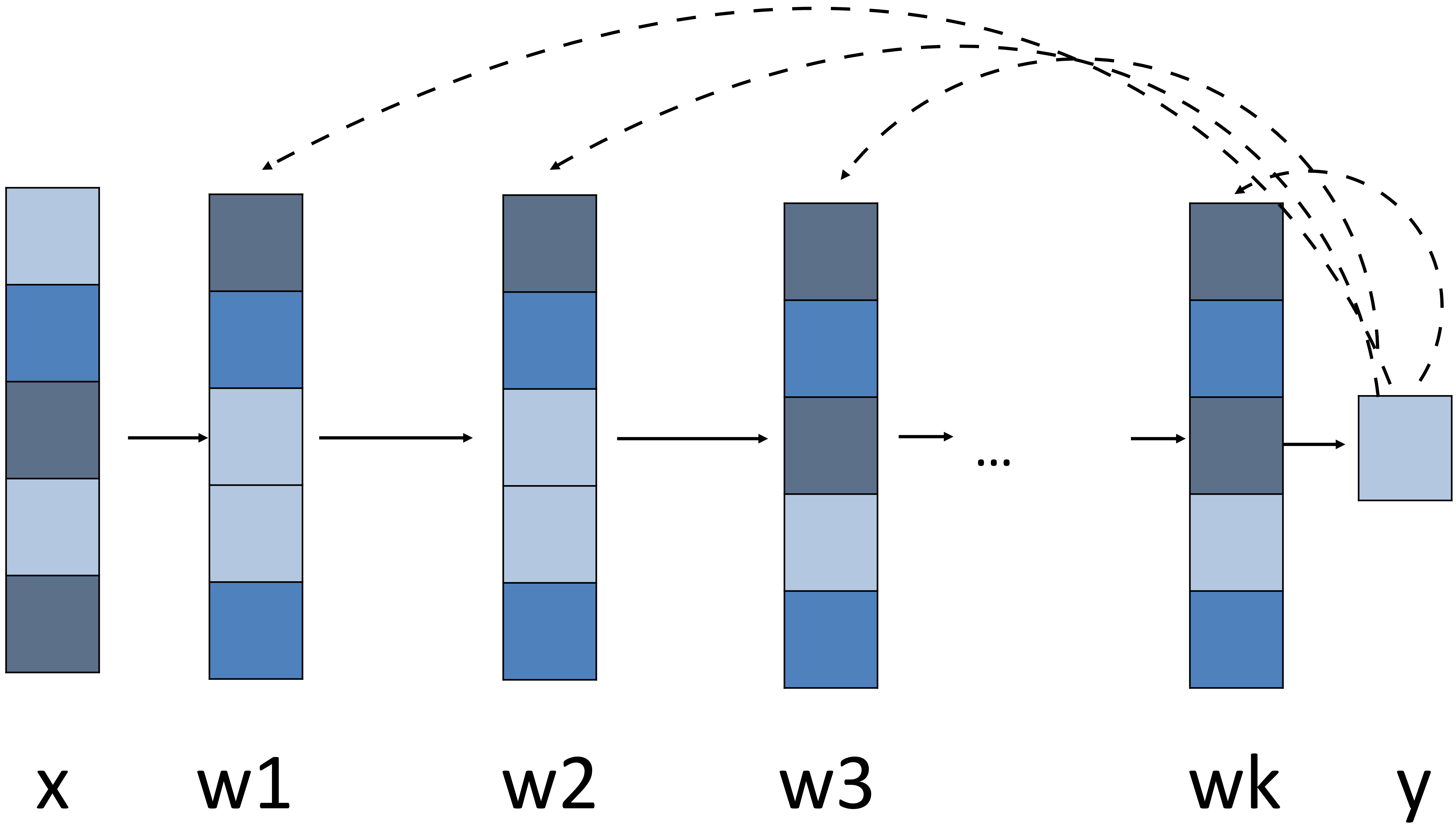
Getting "Deep"

$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$

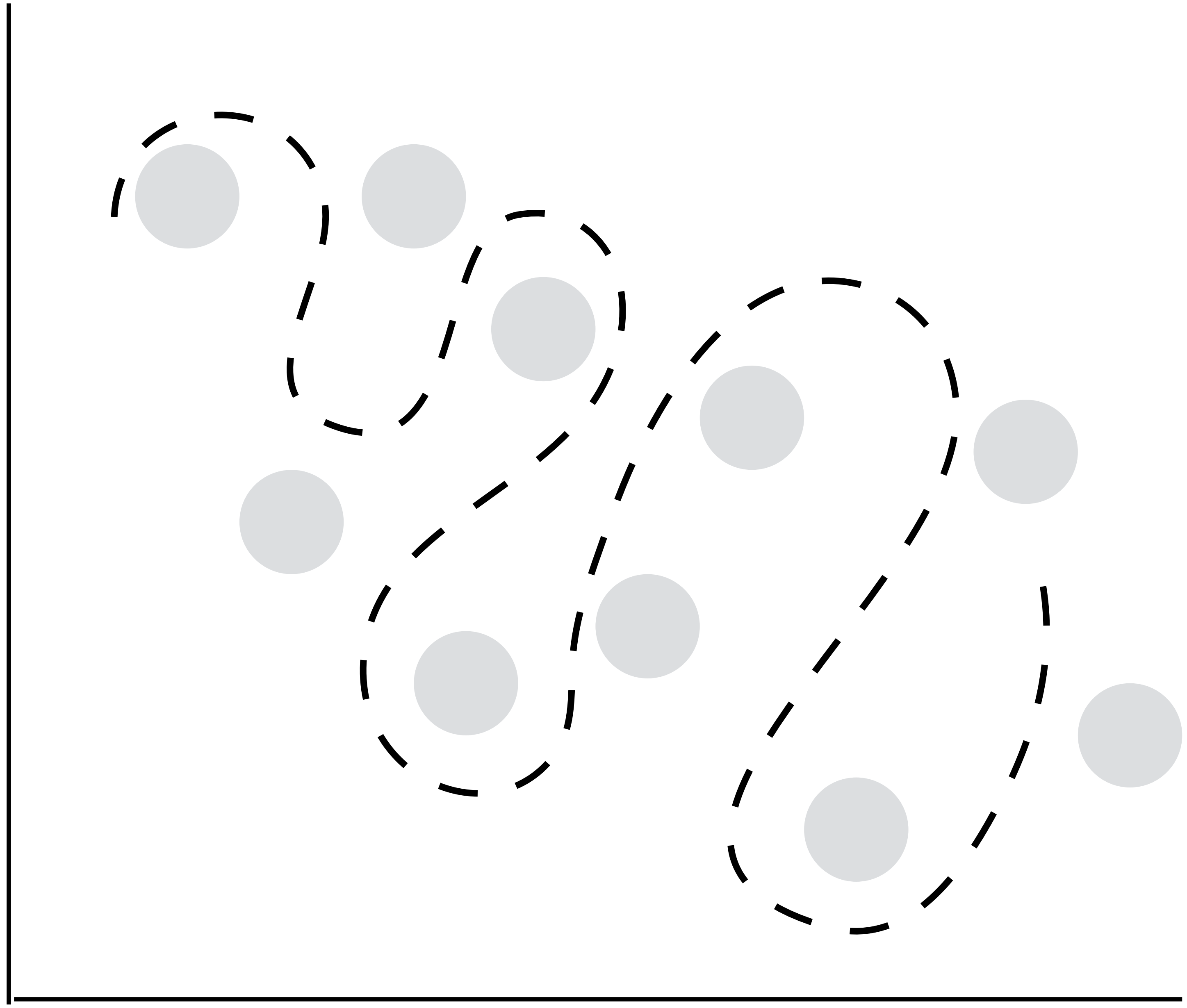


- New feature vectors
- Any correction that may improve the performance of the model

Complexity



Many nonlinear parameters = High flexibility = High complexity



Nothing new, nothing fancy

- Neural Networks have been around for a long time (1980's)
- A vanilla Multi Layer Perceptron (MLP) can theoretically approximate any function ("universal approximator")
- Note: "can" != "do"

What changed recently?

Deep learning became viable for a few reasons….

- Backpropagation allows building deep networks and actually train them
- GPUs: and we can train them fast
- Data: and we can train on enough data that they actually converge to something useful

Why are they so much better, though?

- "Its how the brain works."
  - —> NO!
- End-to-end training—optimize directly for the thing you care about
- Dense/denoised representations—similar inputs get similar predictions
- Uniform representations across sub-disciplines of AI (i.e. vision, language, sensor inputs)
  - "its all just vectors anyway"

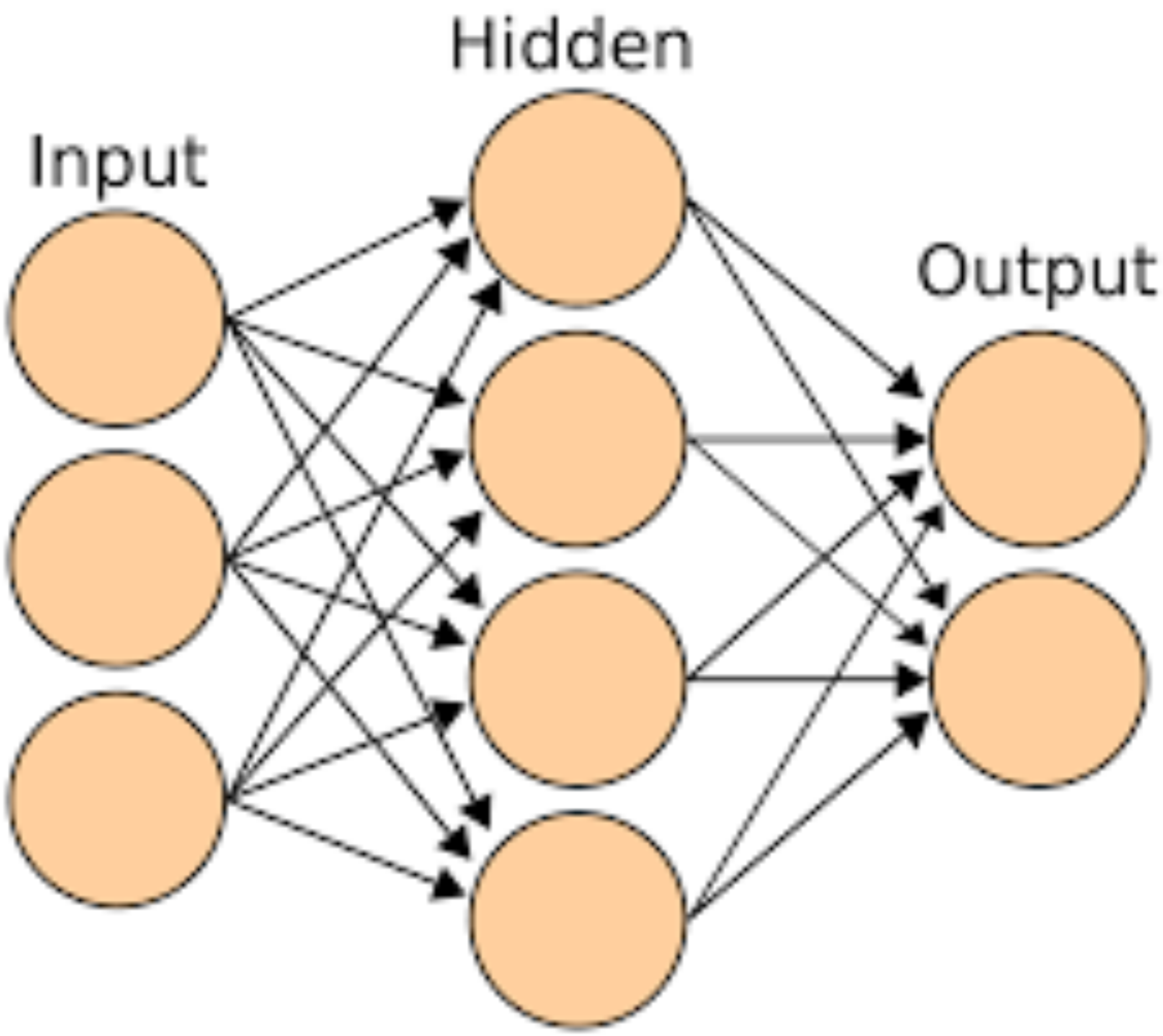
NNs as classifiers

- You already have linear regression, naive bayes, logistic regression, svm…
- Now you have neural nets too!

Multilayer Perceptron

"Feed Forward Net"

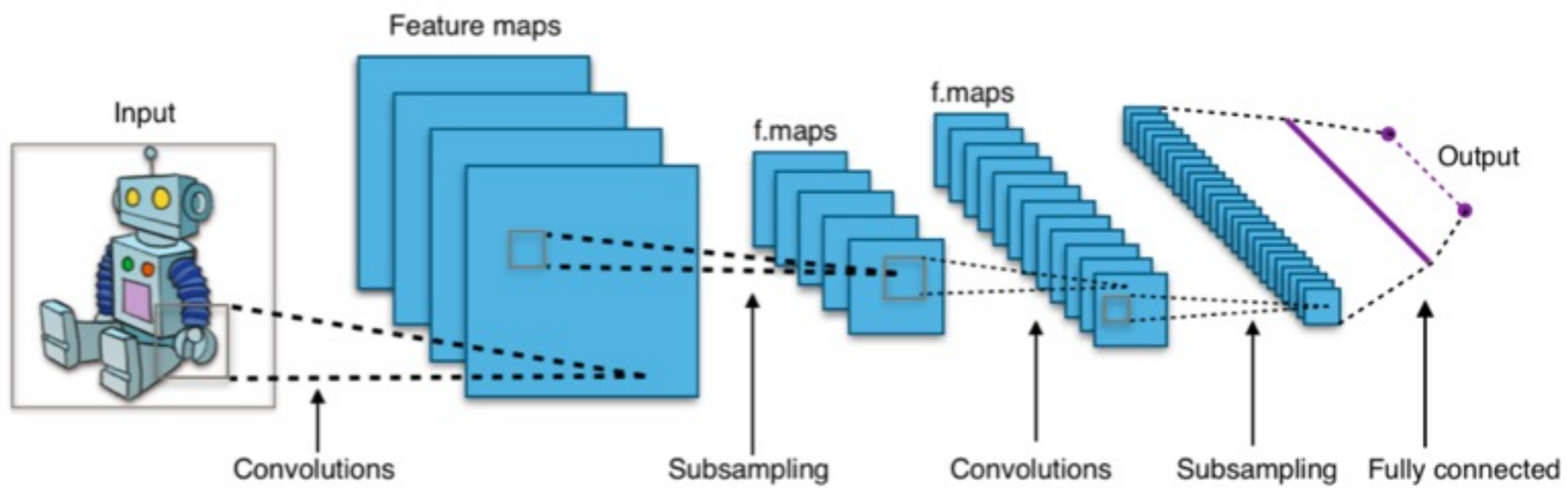
"Fully Connected Layer"



Arbitrary, non-linear combinations of input features.

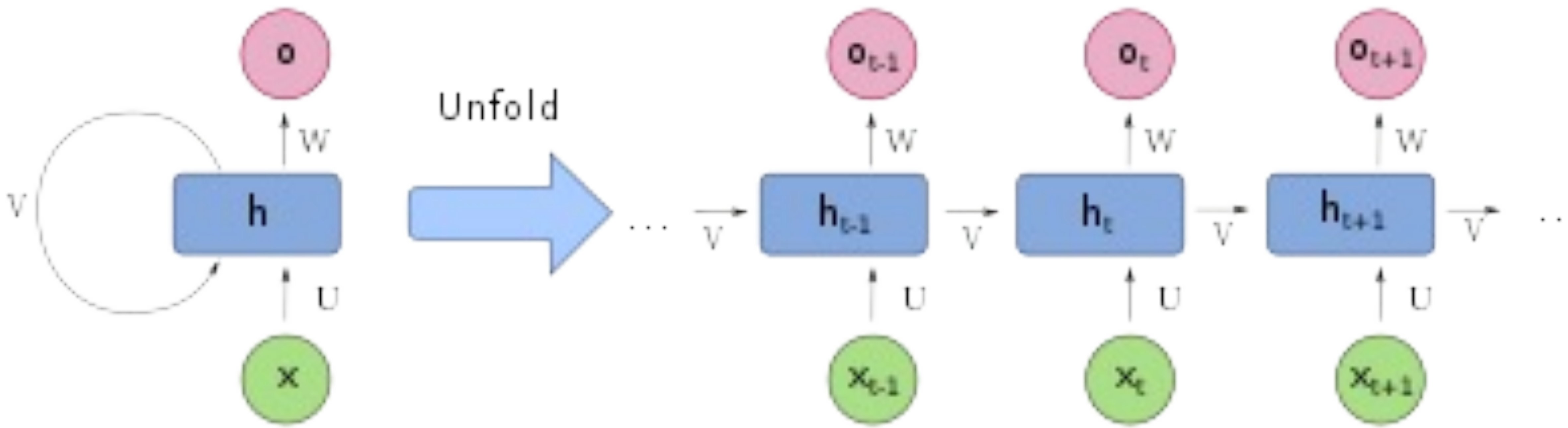
No prior on the structure of those features.

Convolutional Neural Net (CNN)



Used for vision. Assumes spatial structure to the data.

Recurrent Neural Net (RNN)

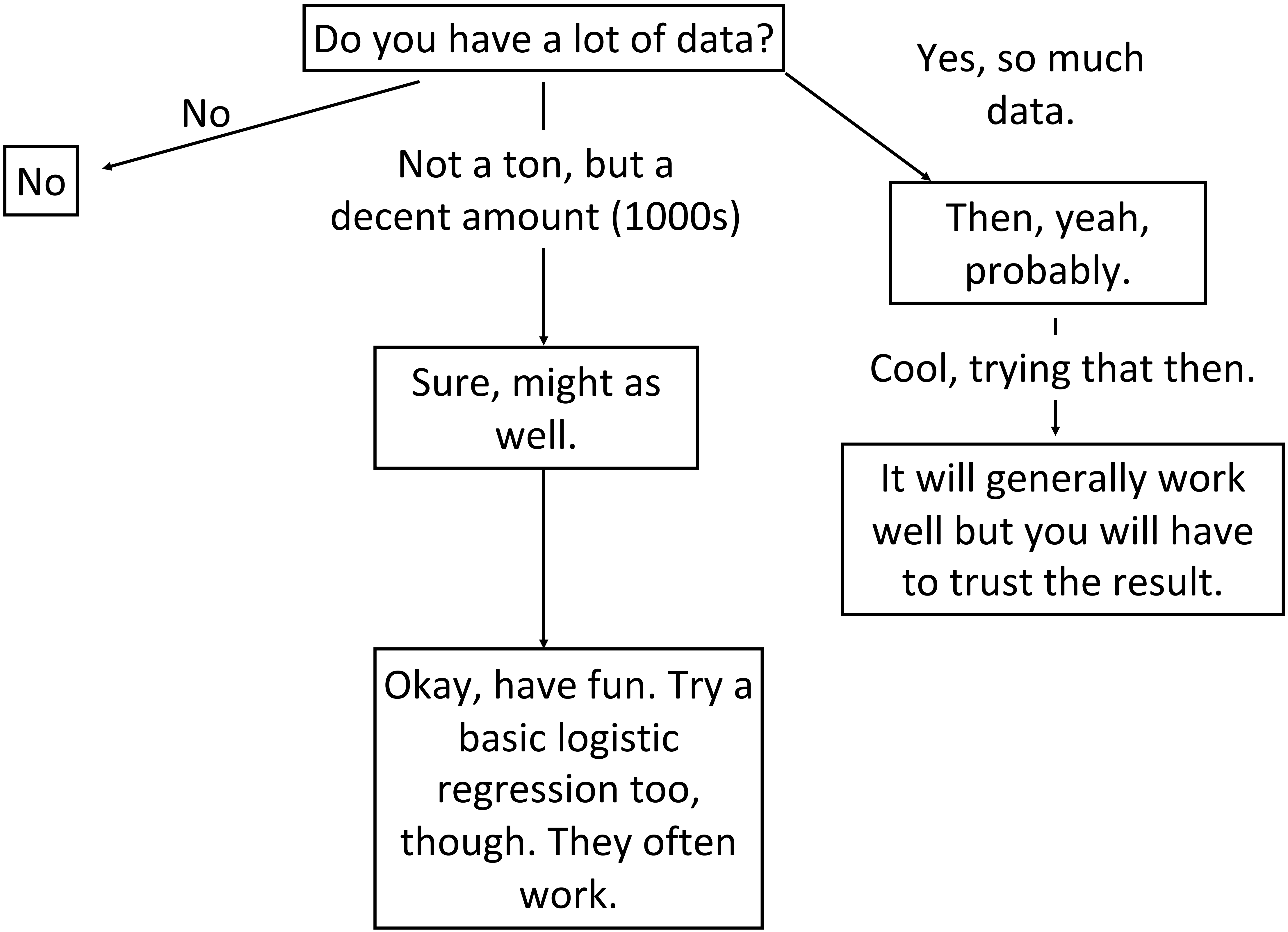


Used for language (and other things).
Assumes linear/temporal structure to the data.

So why haven’t NNs solved everything?

- Mostly require (massive!) supervised learning. Better use of deep RL/unsupervised pretraining?
- End-to-end-training hurts generalizability. Inductive biases on the hypothesis space?
- The ***big*** reason: its really really hard to formulate most problems as ML problems

Should I use deep learning?

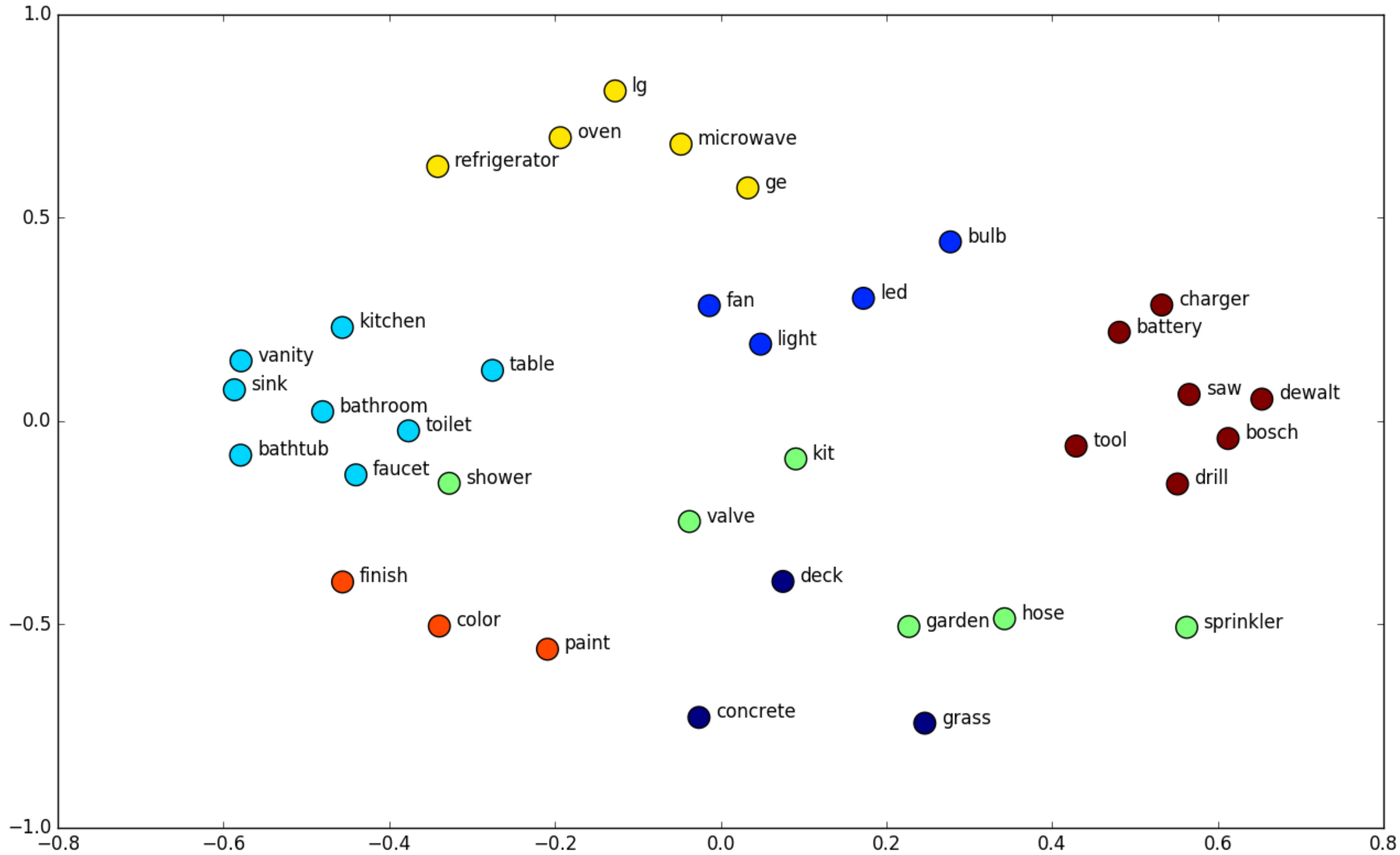


Transfer Learning

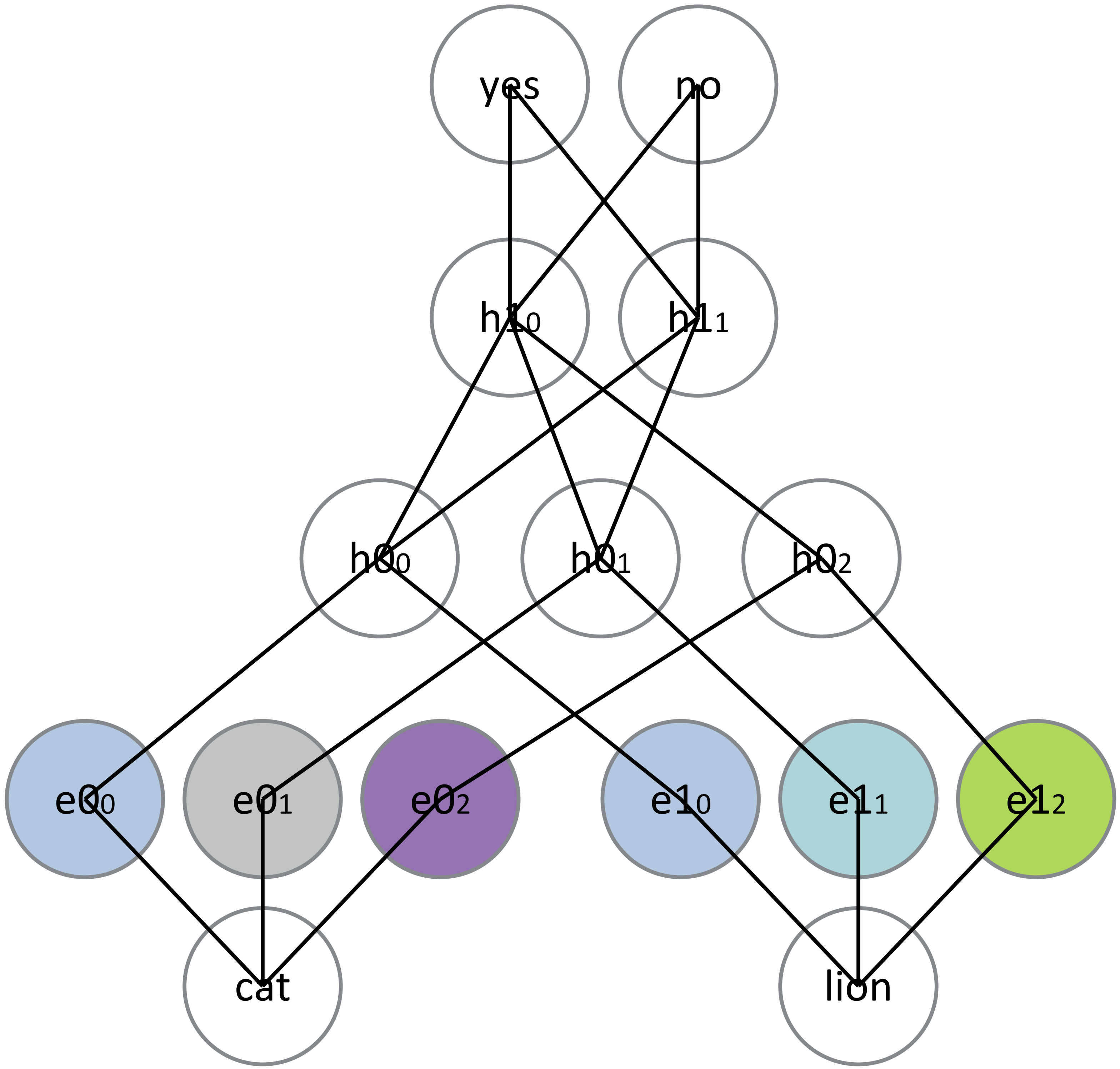
a.k.a. “Pretraining”,
“Representation Learning”…

- Train a model to do some task T1 (for which you have a lot of data)
- Let the model converge. Now your hidden states contain whatever features were good for T1
- Maybe these features are good for some other task T2 too? Maybe you can now do T2 with less training?

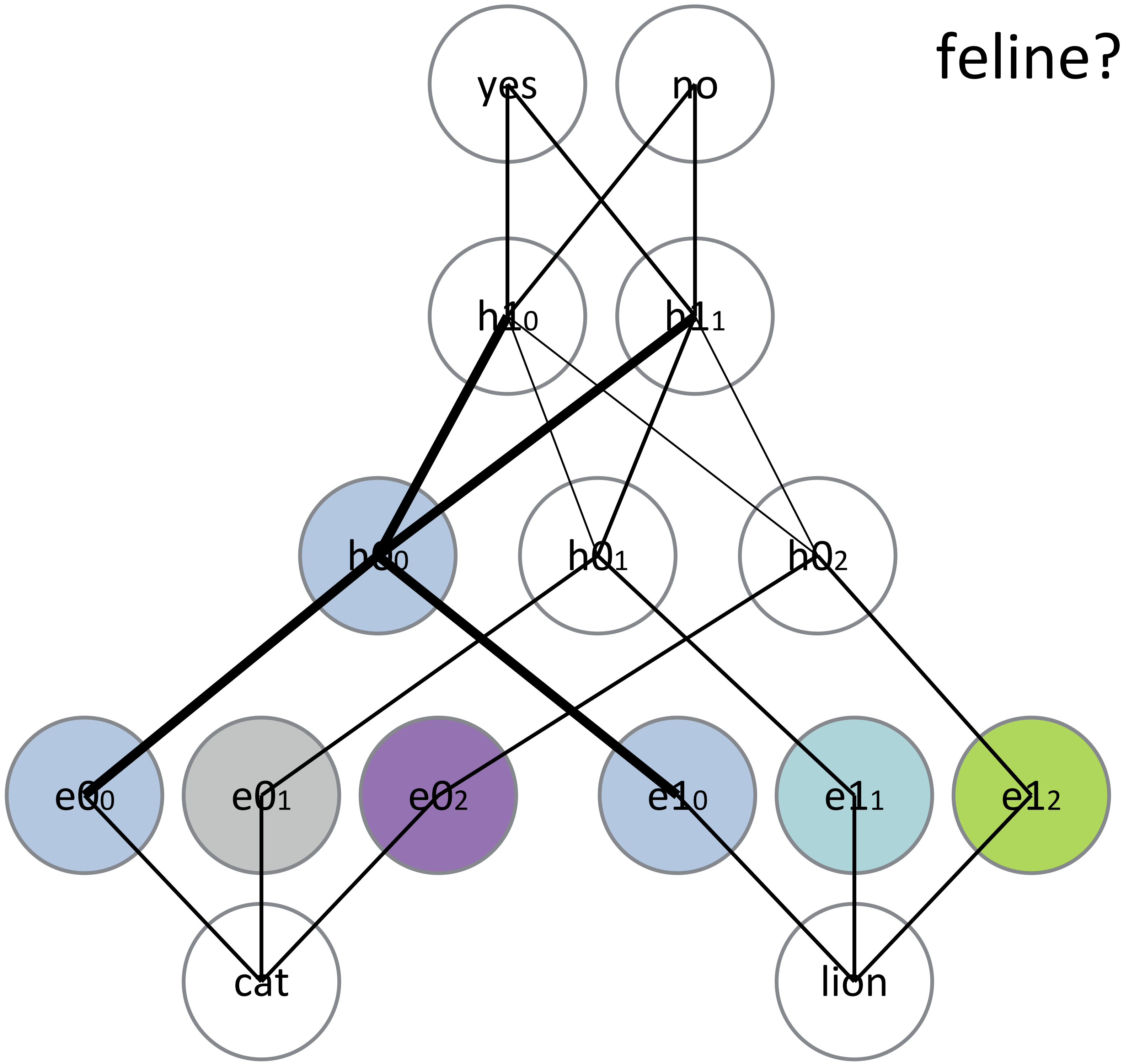
Word Embeddings



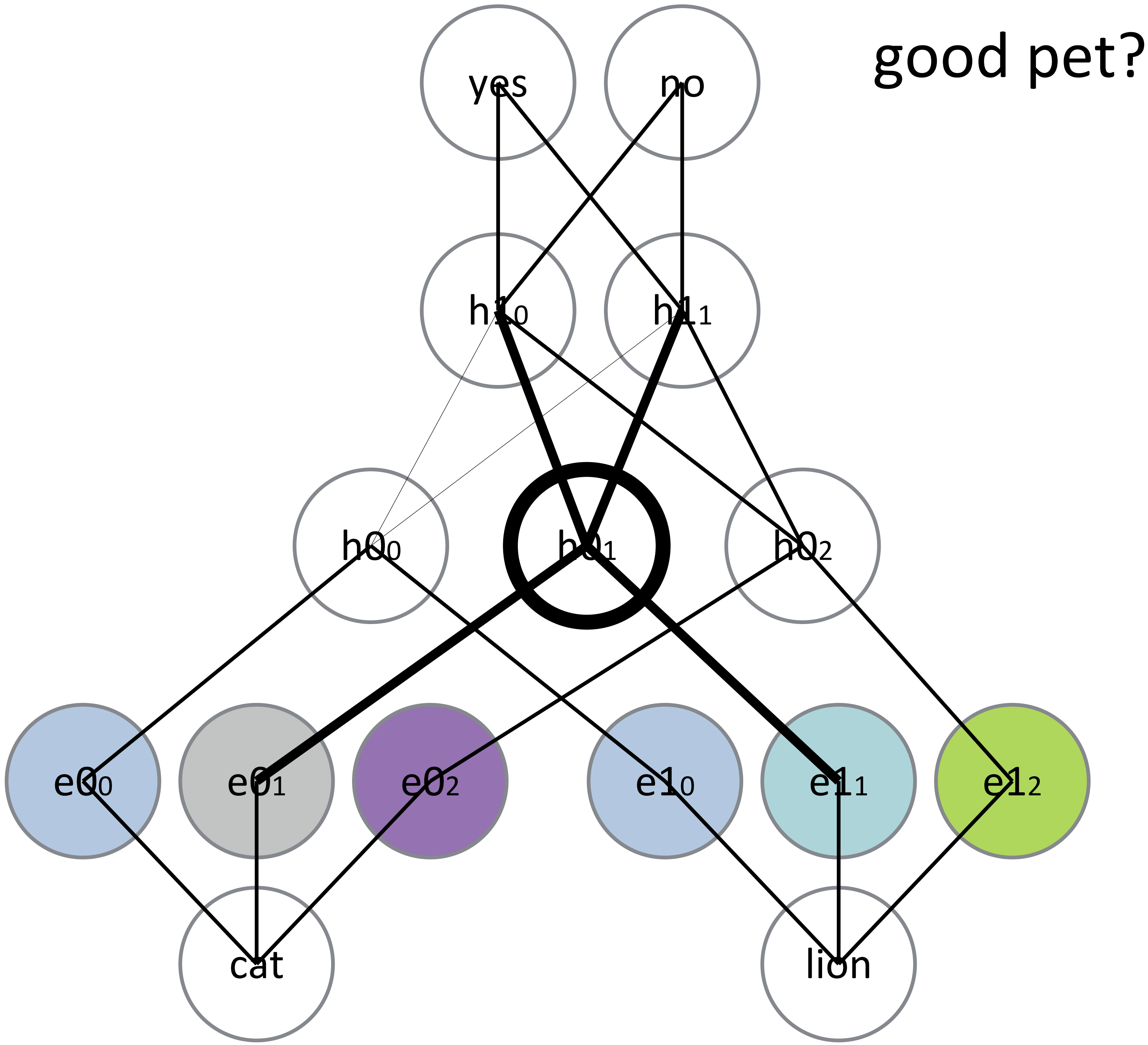
Representation Learning



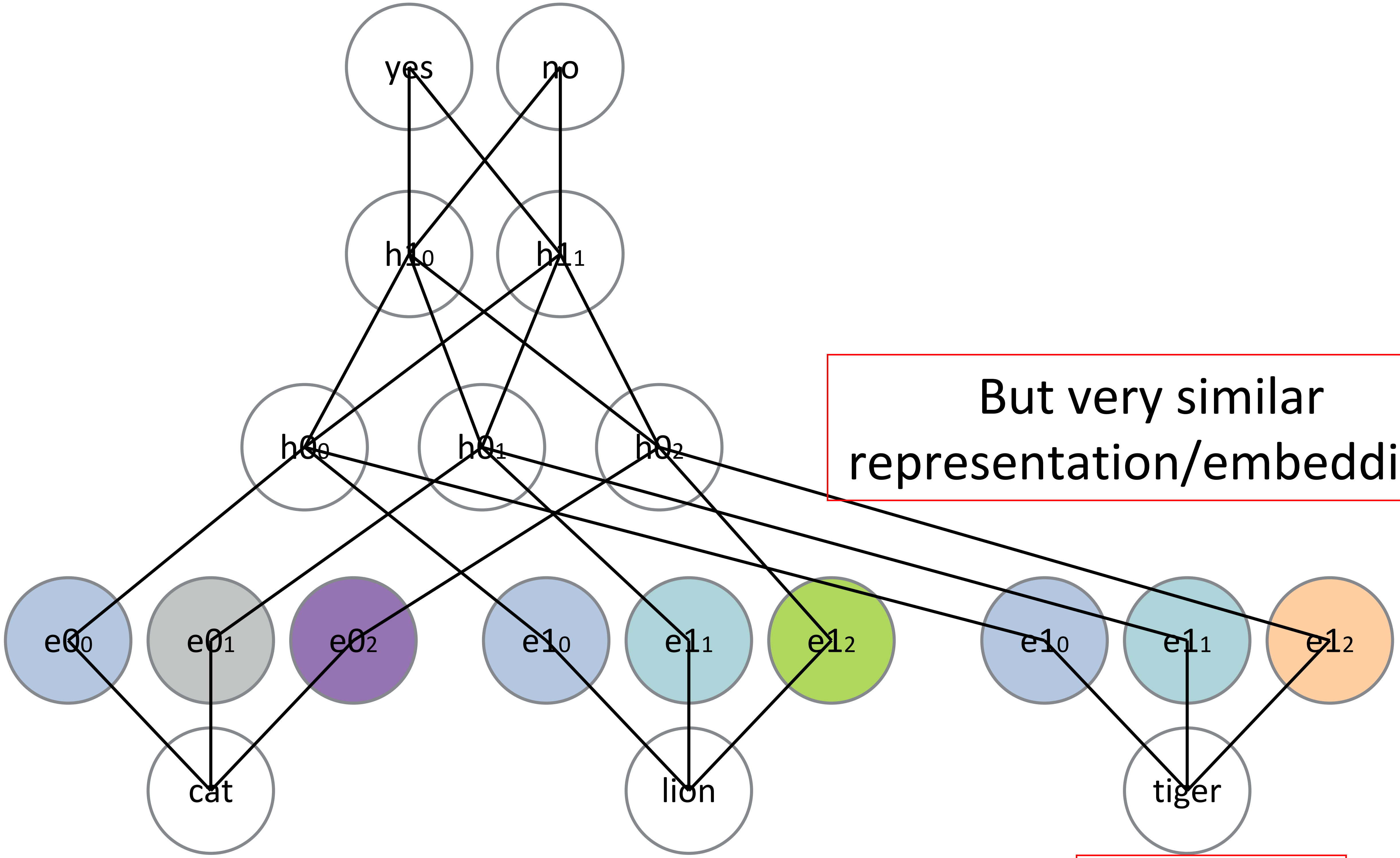
Representation Learning



Representation Learning



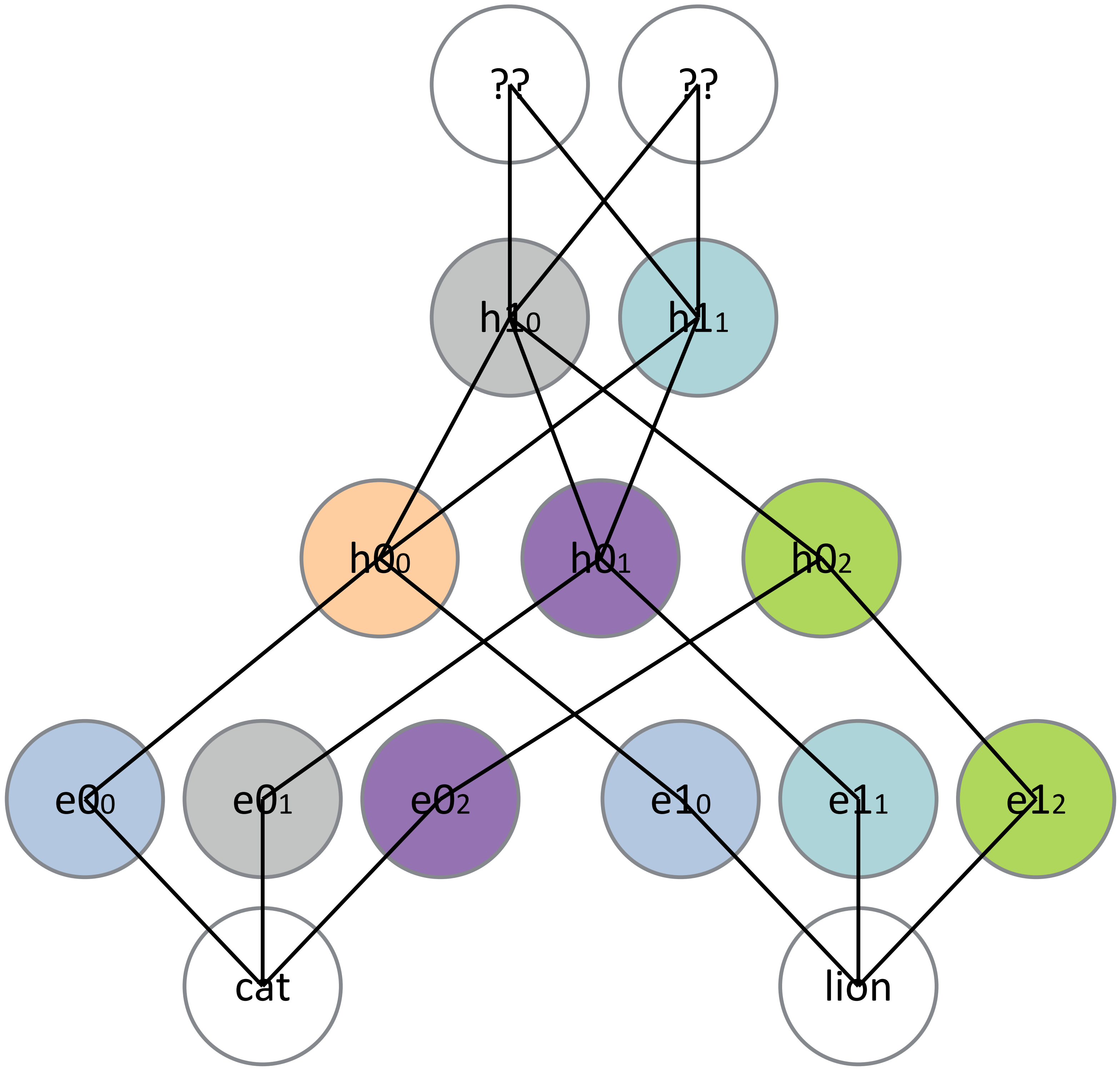
Representation Learning



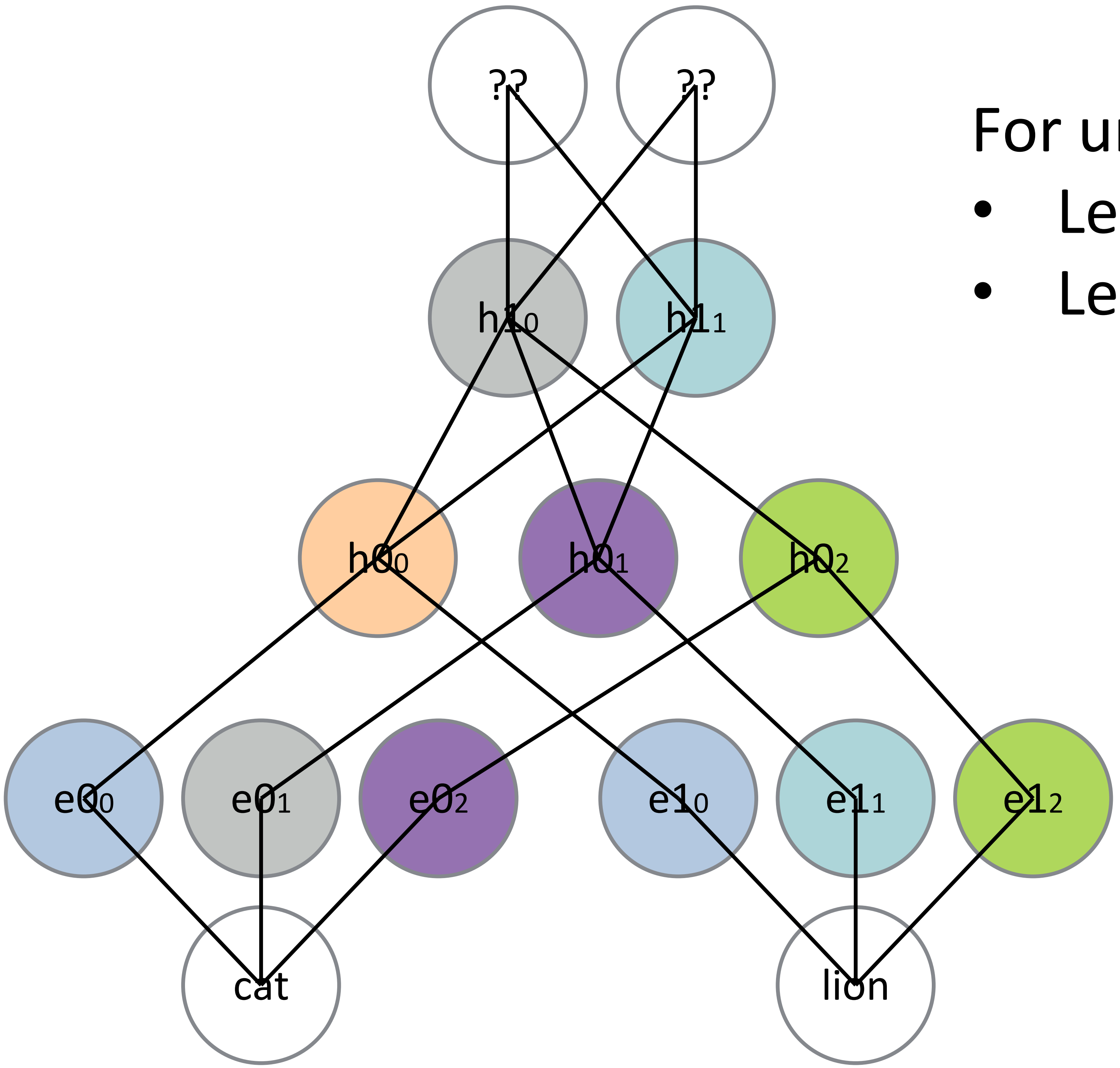
But very similar representation/embedding

New object

Representation Learning



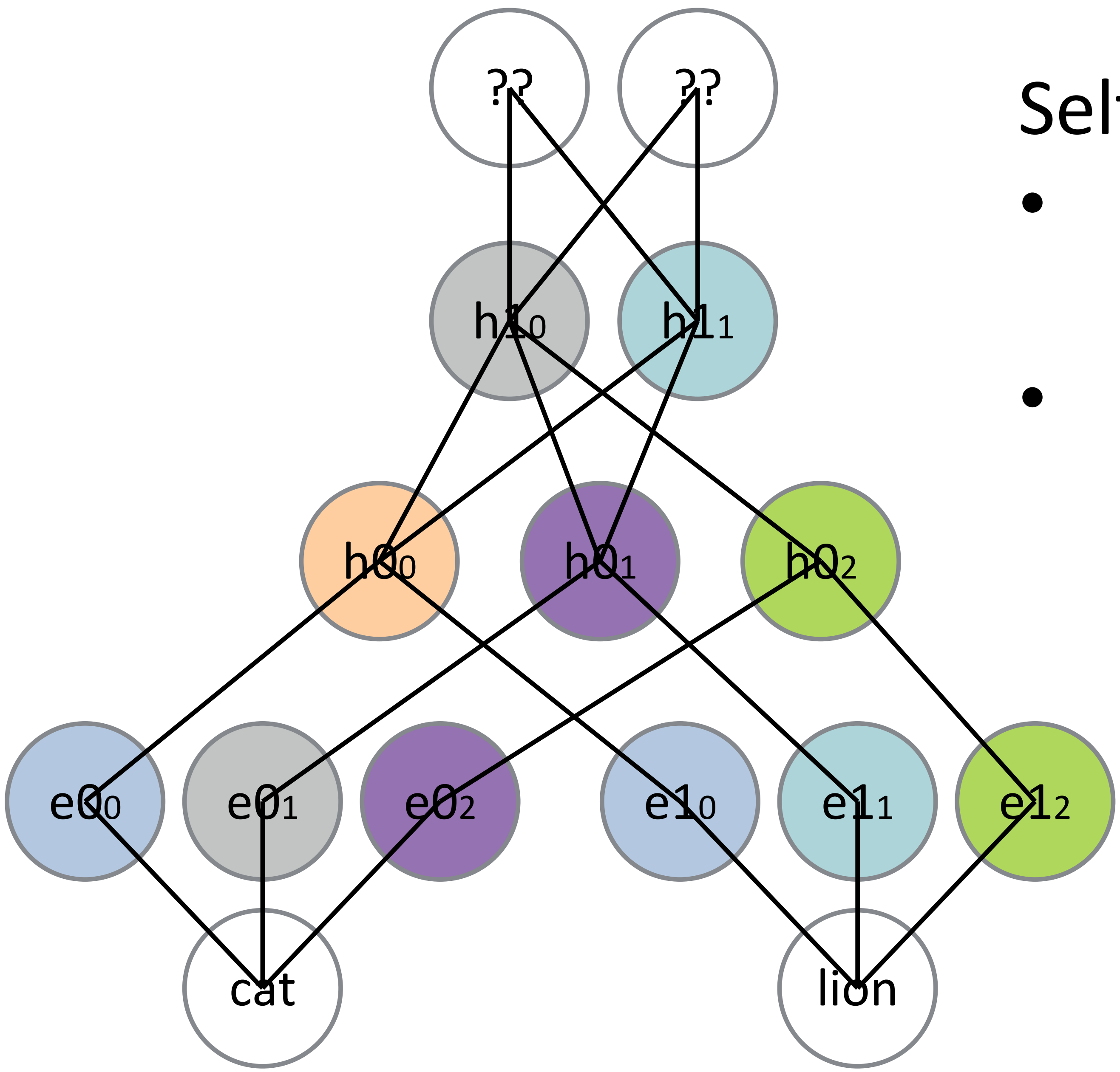
Representation Learning



For unsupervised learning:

- Learn input embedding
- Learn network coefficients

Representation Learning



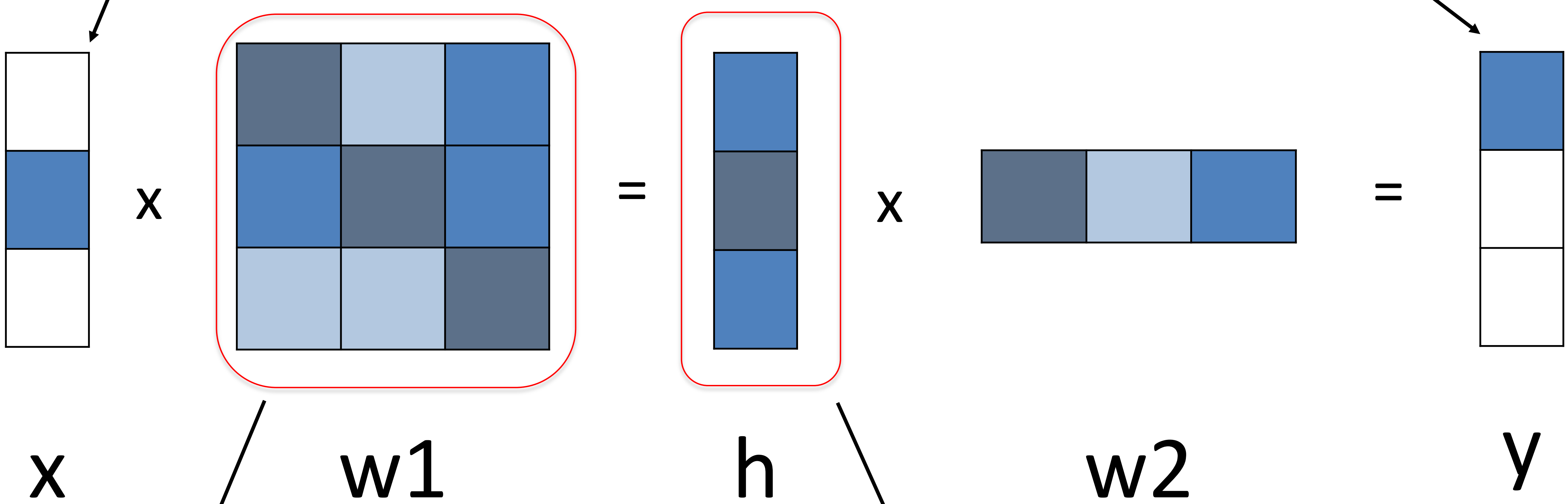
Self- supervised learning

- Predicting the future “one step at a time”
- Language modeling,

Word Embeddings

one hot encoding, i.e.: the word "congress"

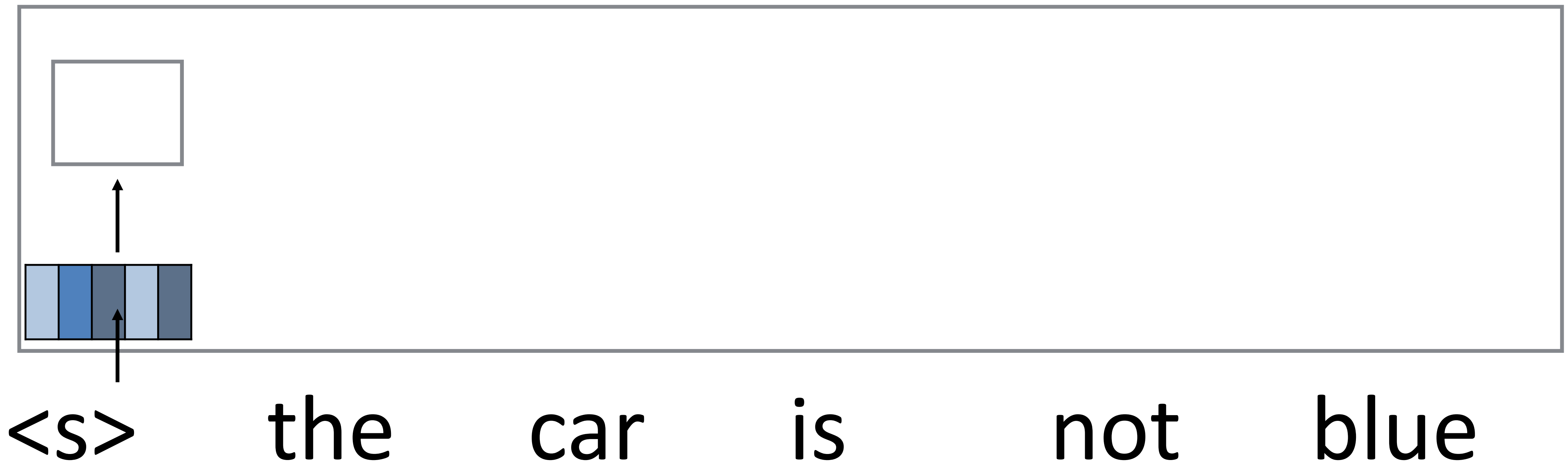
predict one hot encoding of next word, i.e.:the word "stagnated"



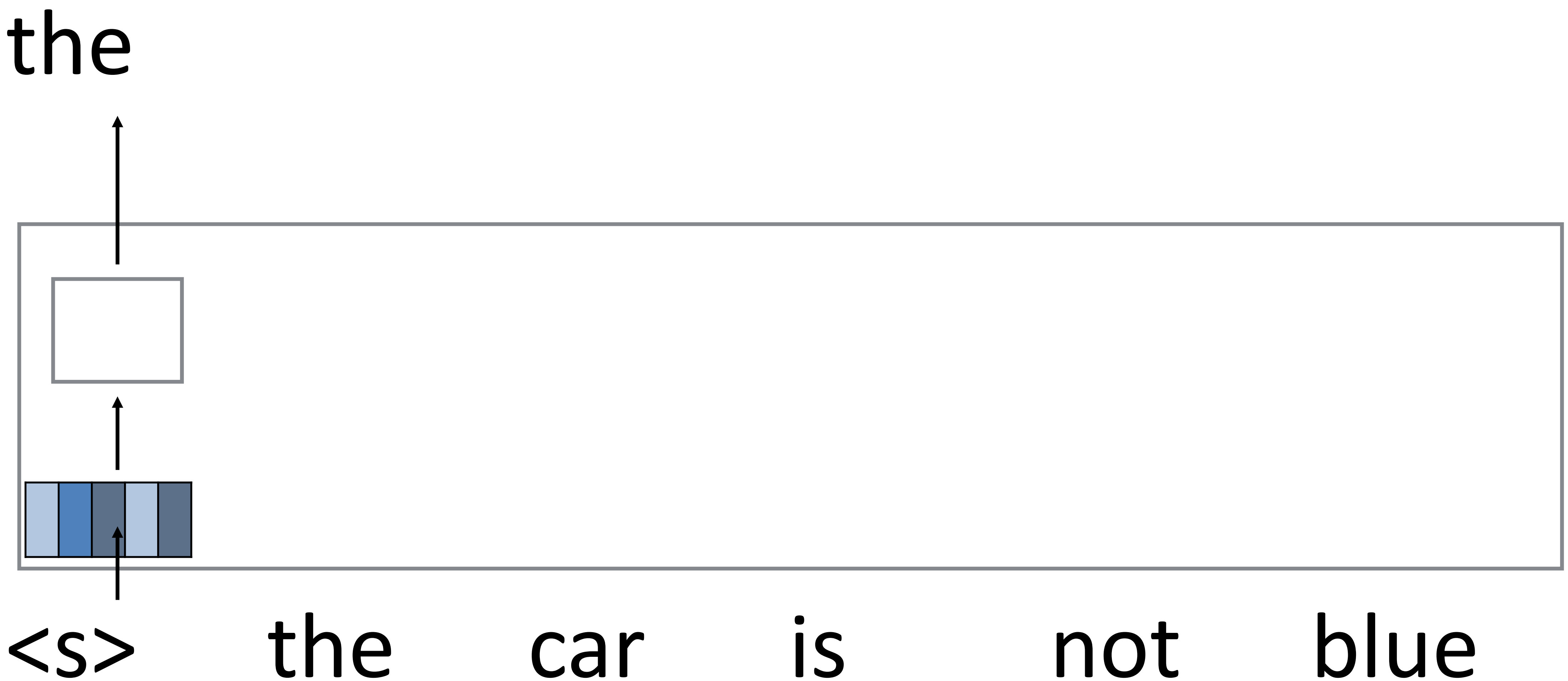
Embedding Matrix

Can be used to denote parameter sharing over similar words.

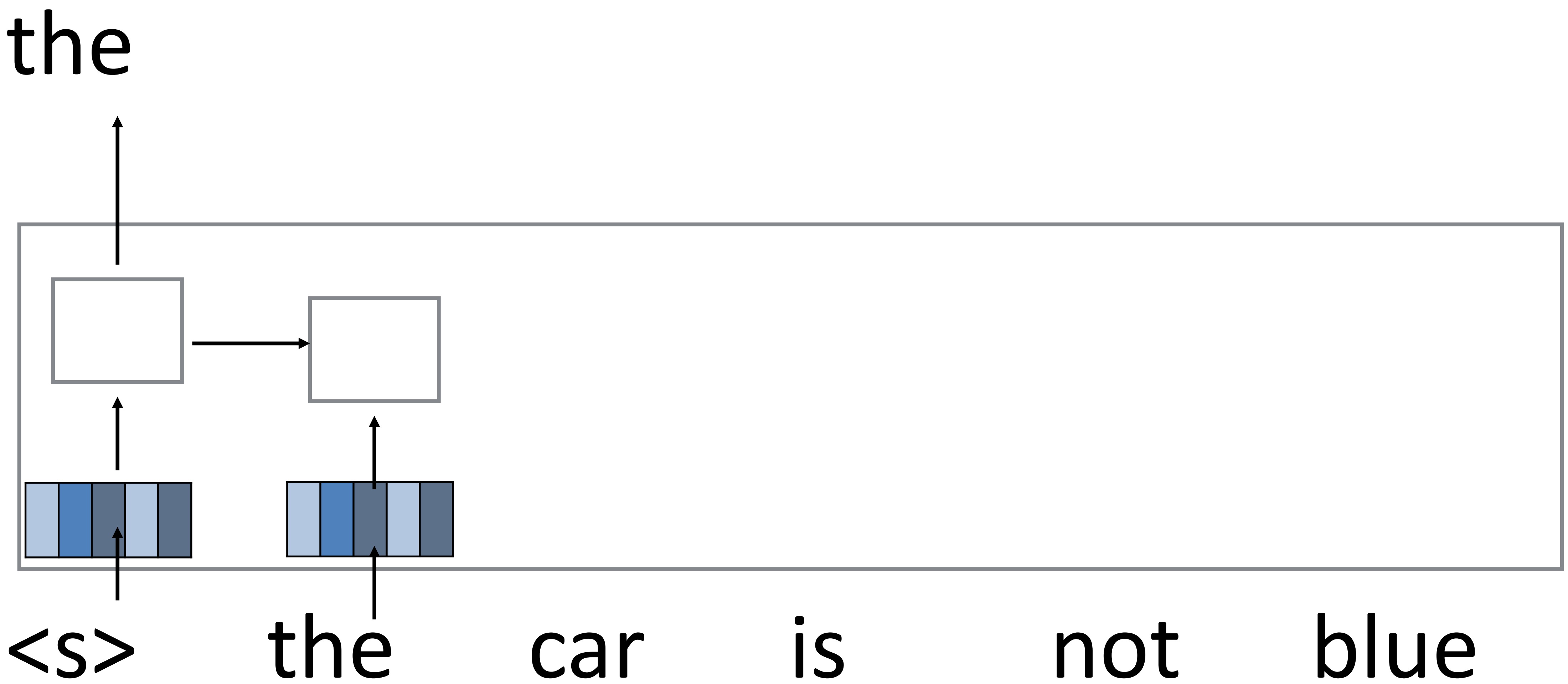
Sentence Embeddings



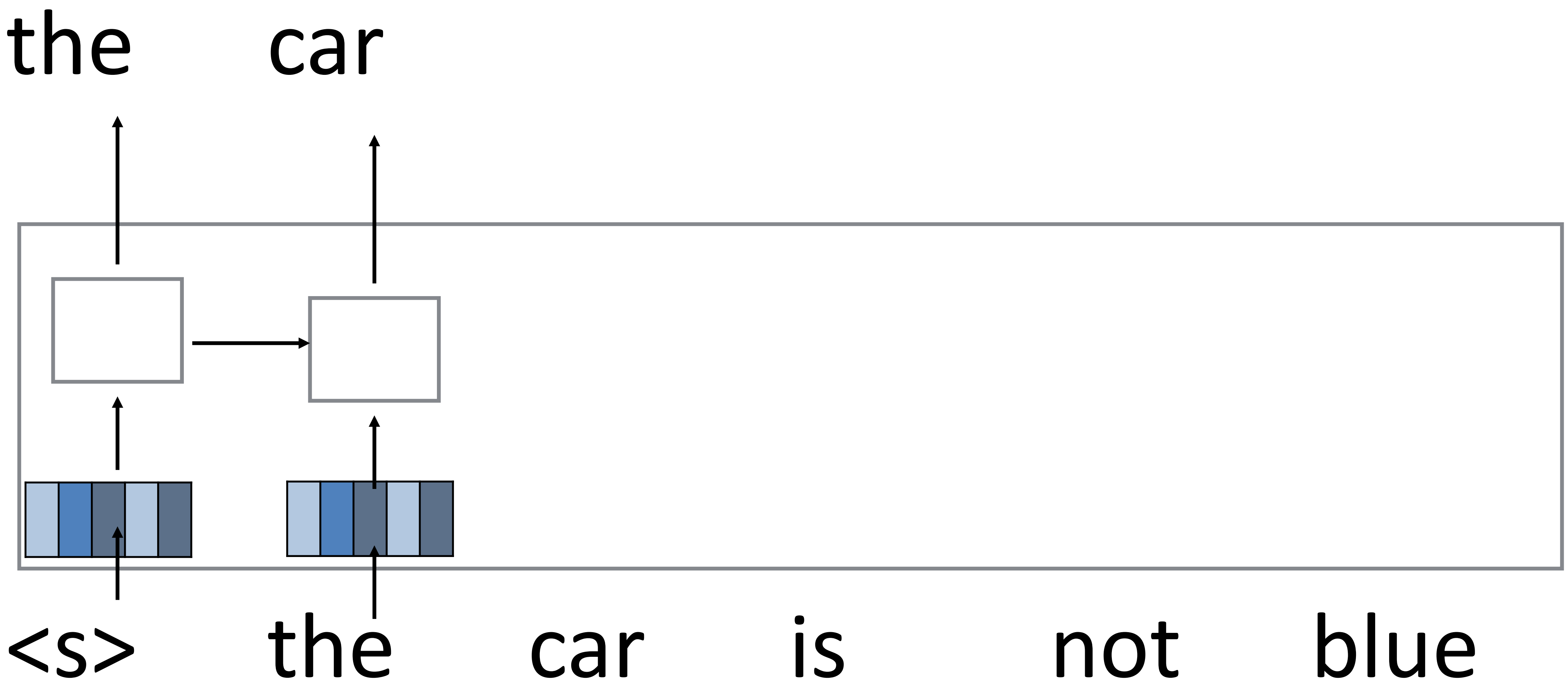
Sentence Embeddings



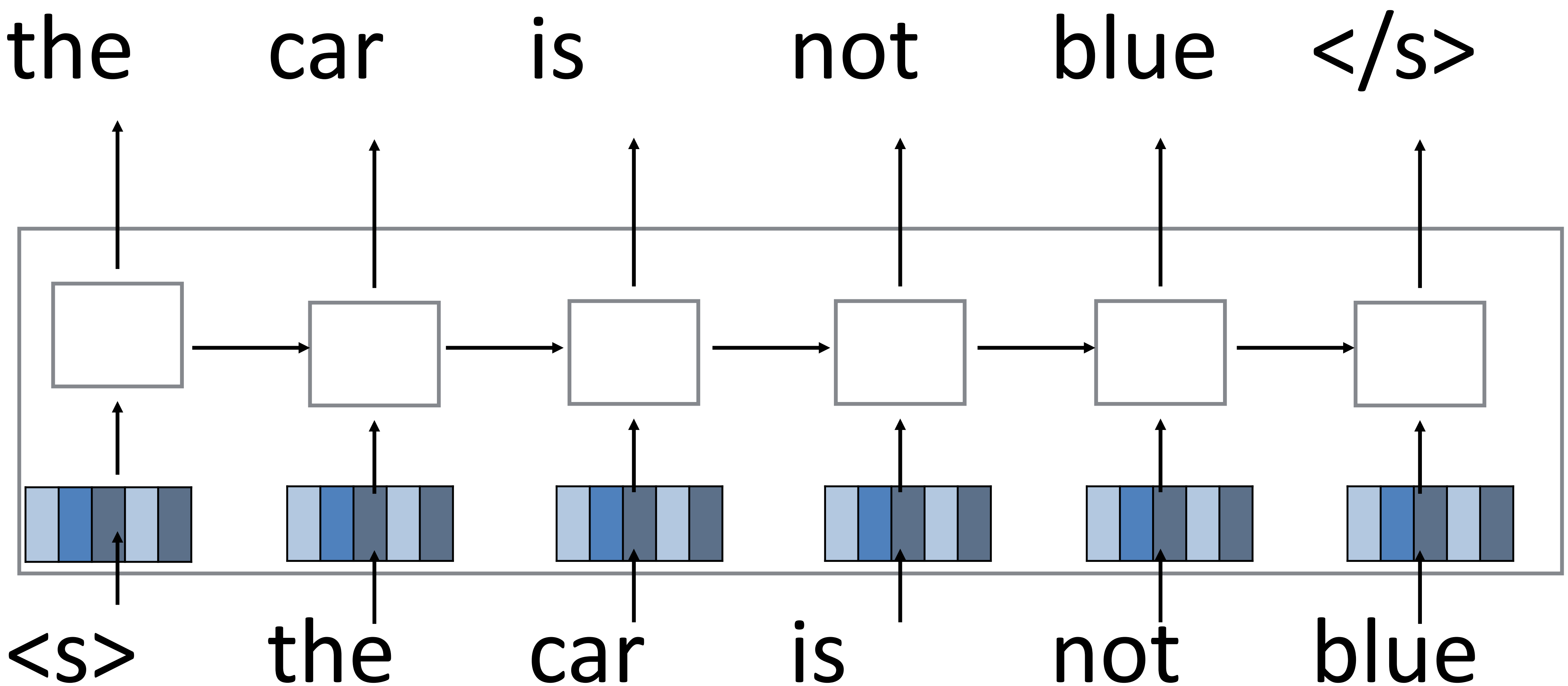
Sentence Embeddings



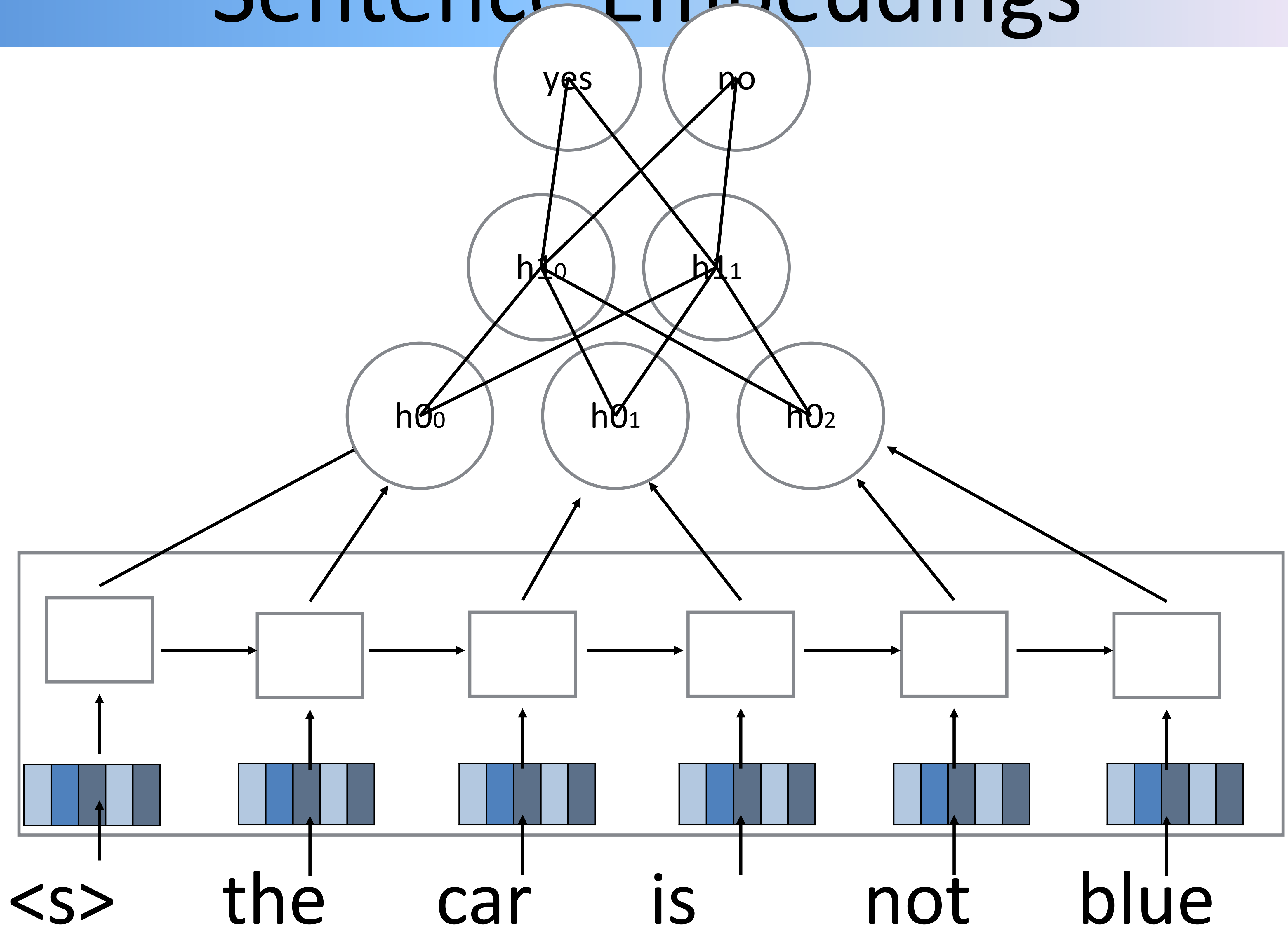
Sentence Embeddings



Sentence Embeddings



Sentence Embeddings



Resources

- Simply neural net classifier for images: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
- Simple recurrent network for sequence modeling: https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html