Deep Learning

April 18, 2019

Data Science CSCI 1951A

Brown University

Instructor: Ellie Pavlick

HTAs: Wennie Zhang, Maulik Dang, Gurnaaz Kaur

Announcements

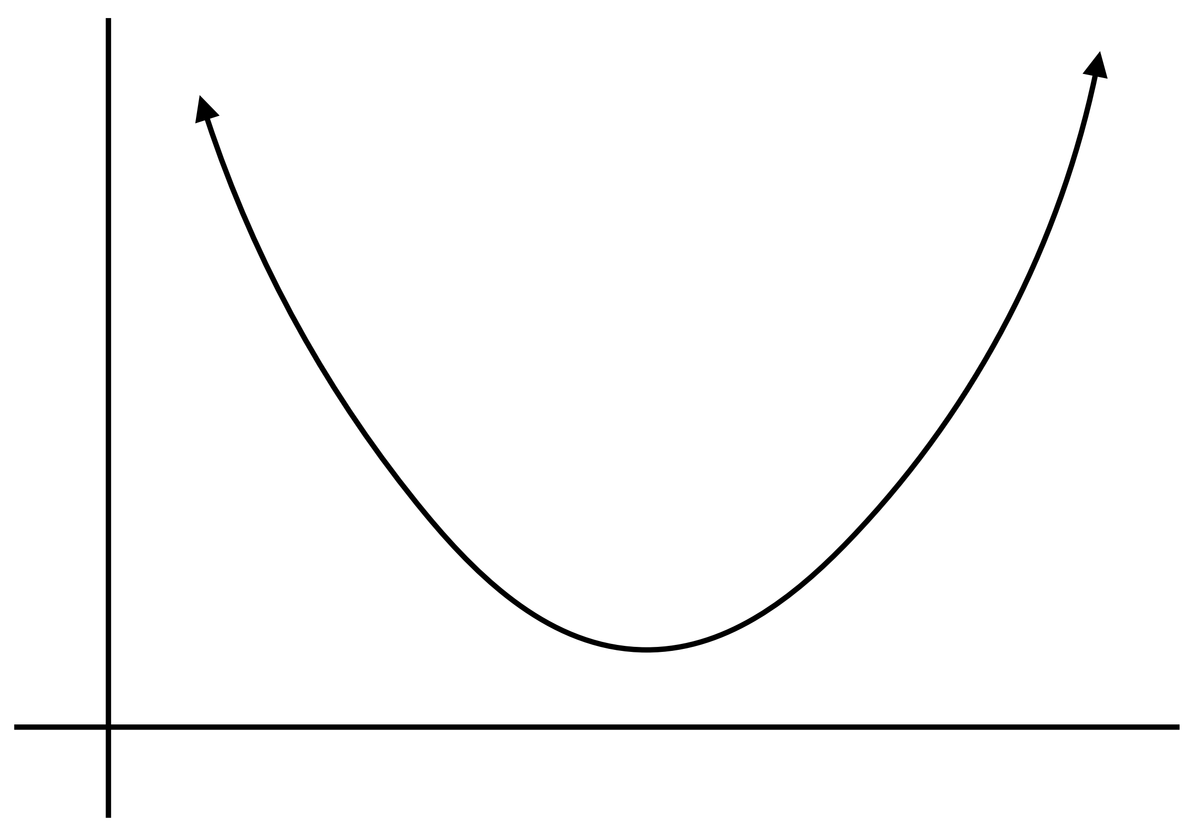
- Extra Office Hours next week—talk to me about your project woes (getting close to your last chance)

Today

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

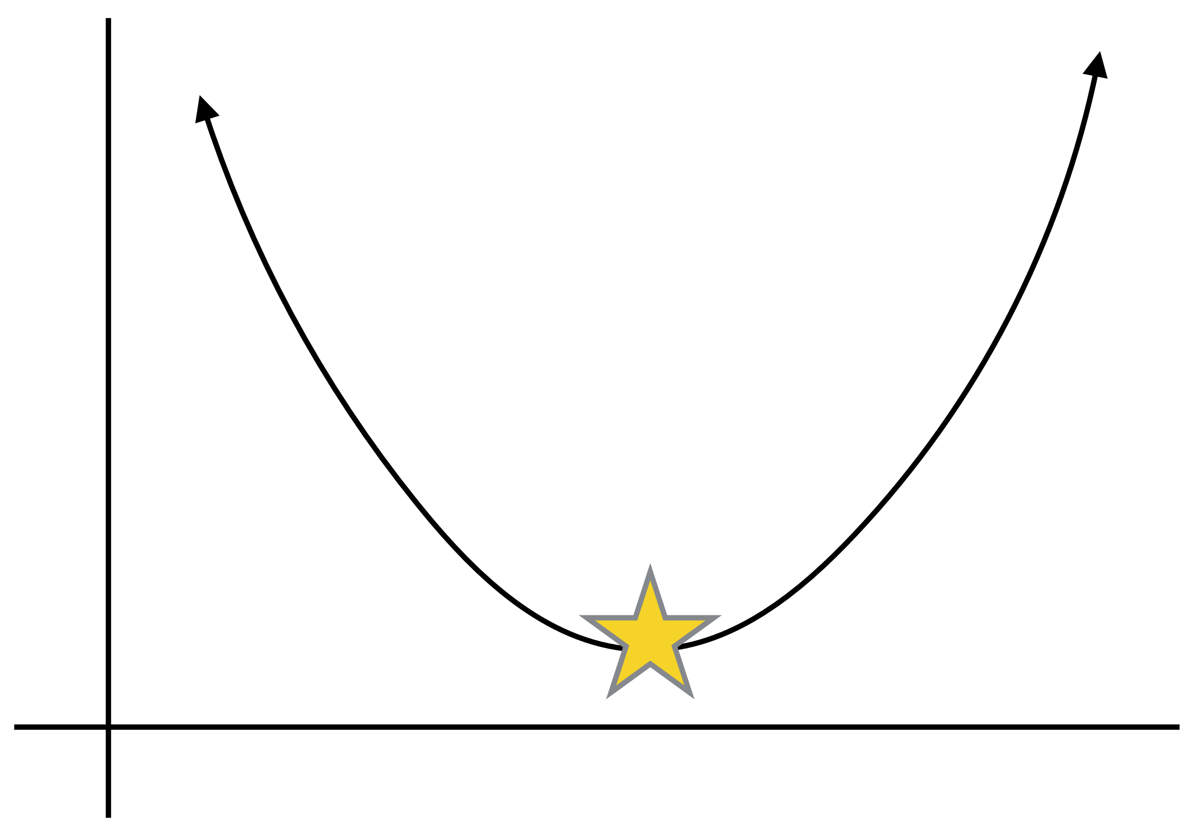
Gradient Descent

minimize $$\sum_{i=1}^{n}(Y_i - \hat{Y})^2$$



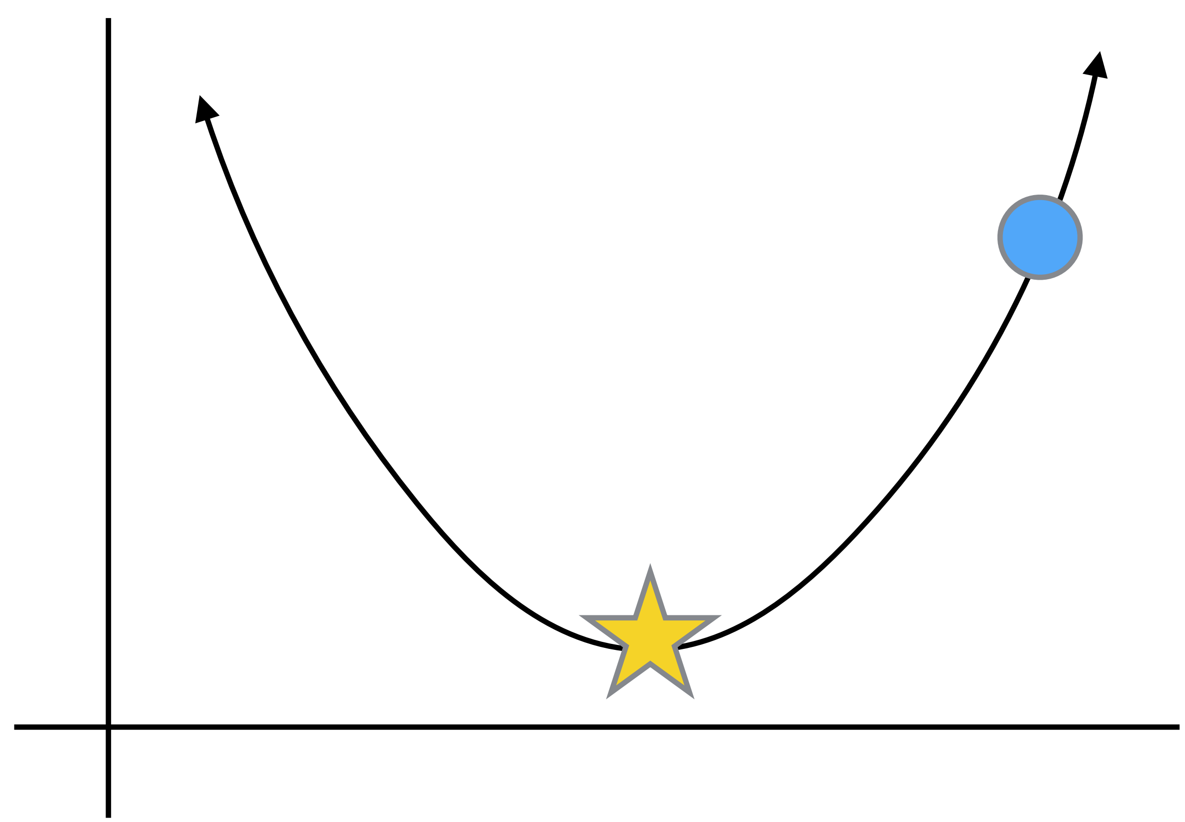
Gradient Descent

minimize $\sum_{i=1}^{n}(Y_i - \hat{Y})^2$



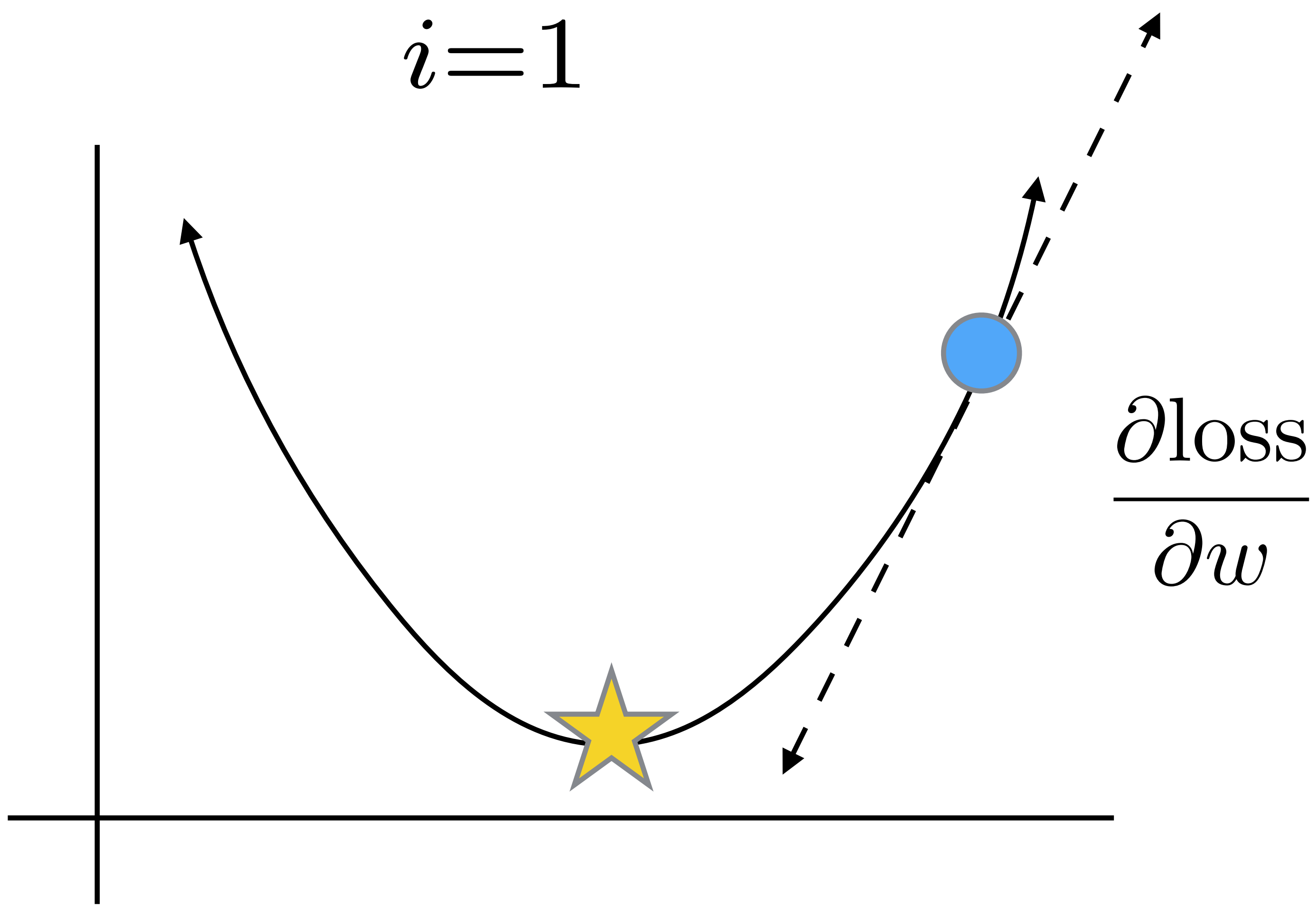
Gradient Descent

minimize $\sum_{i=1}^{n}(Y_i - \hat{Y})^2$



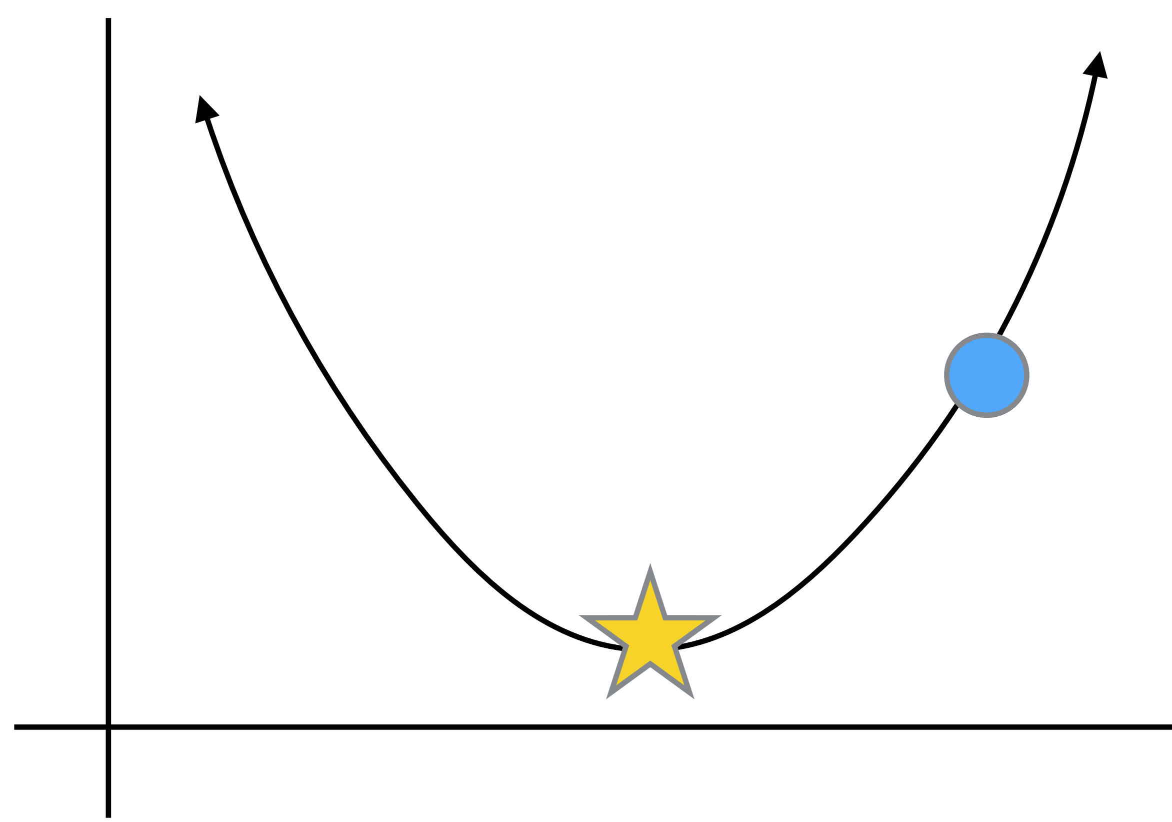
Gradient Descent

minimize $$\sum_{i=1}^{n}(Y_i - \hat{Y})^2$$



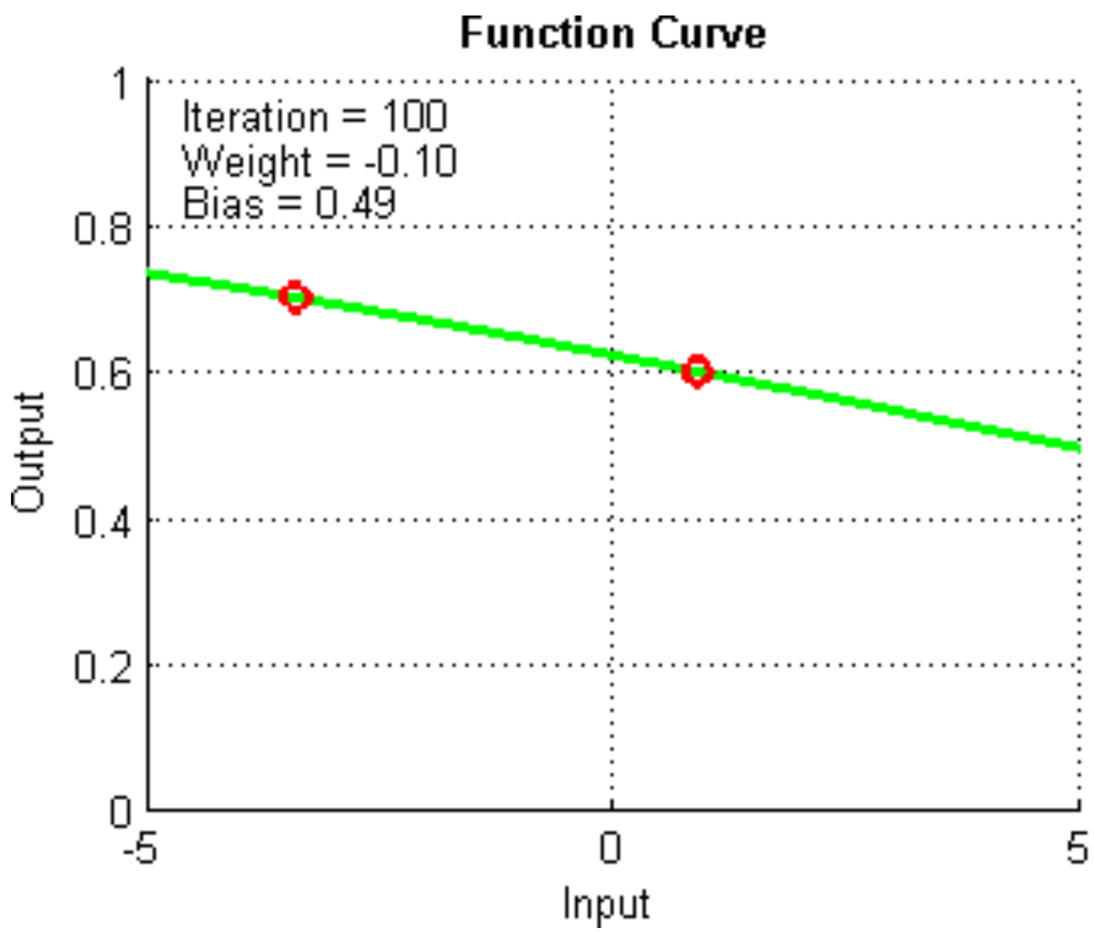
Gradient Descent

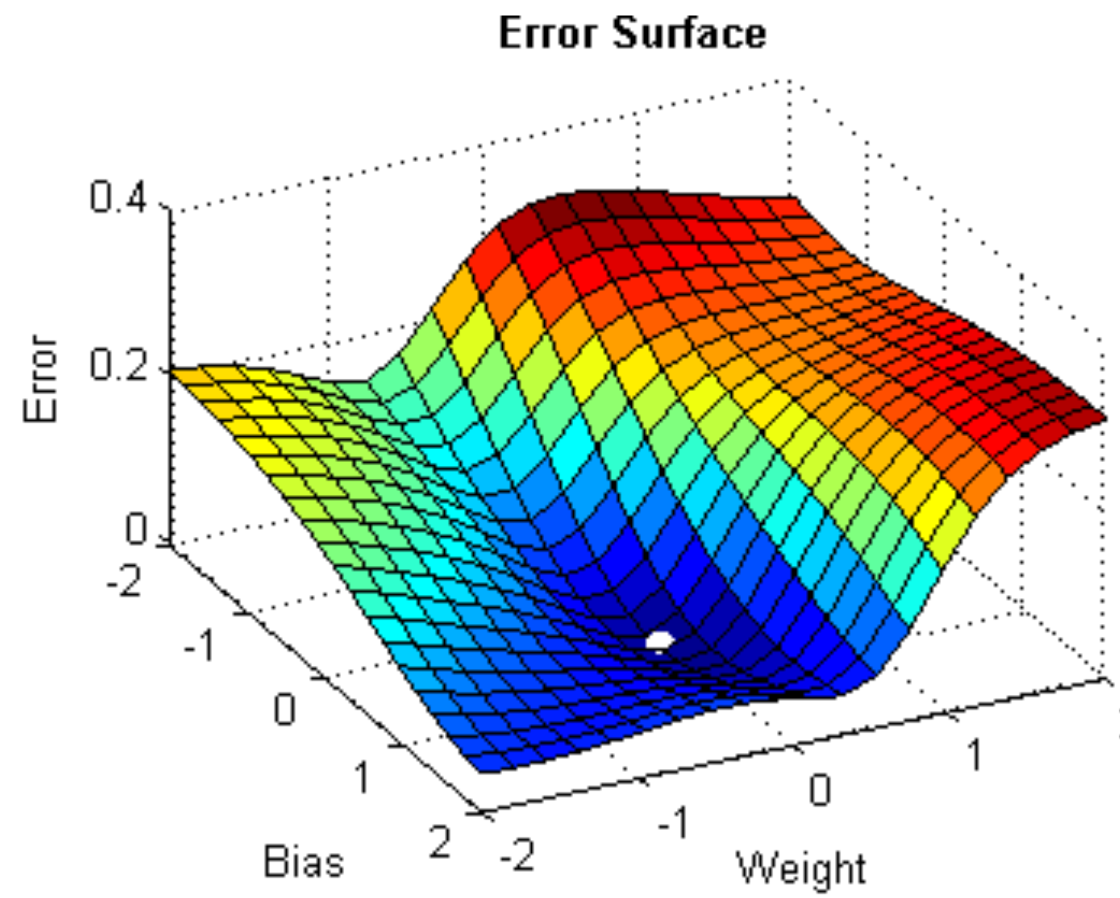
minimize $$\sum_{i=1}^{n}(Y_i - \hat{Y})^2$$



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

Gradient Descent

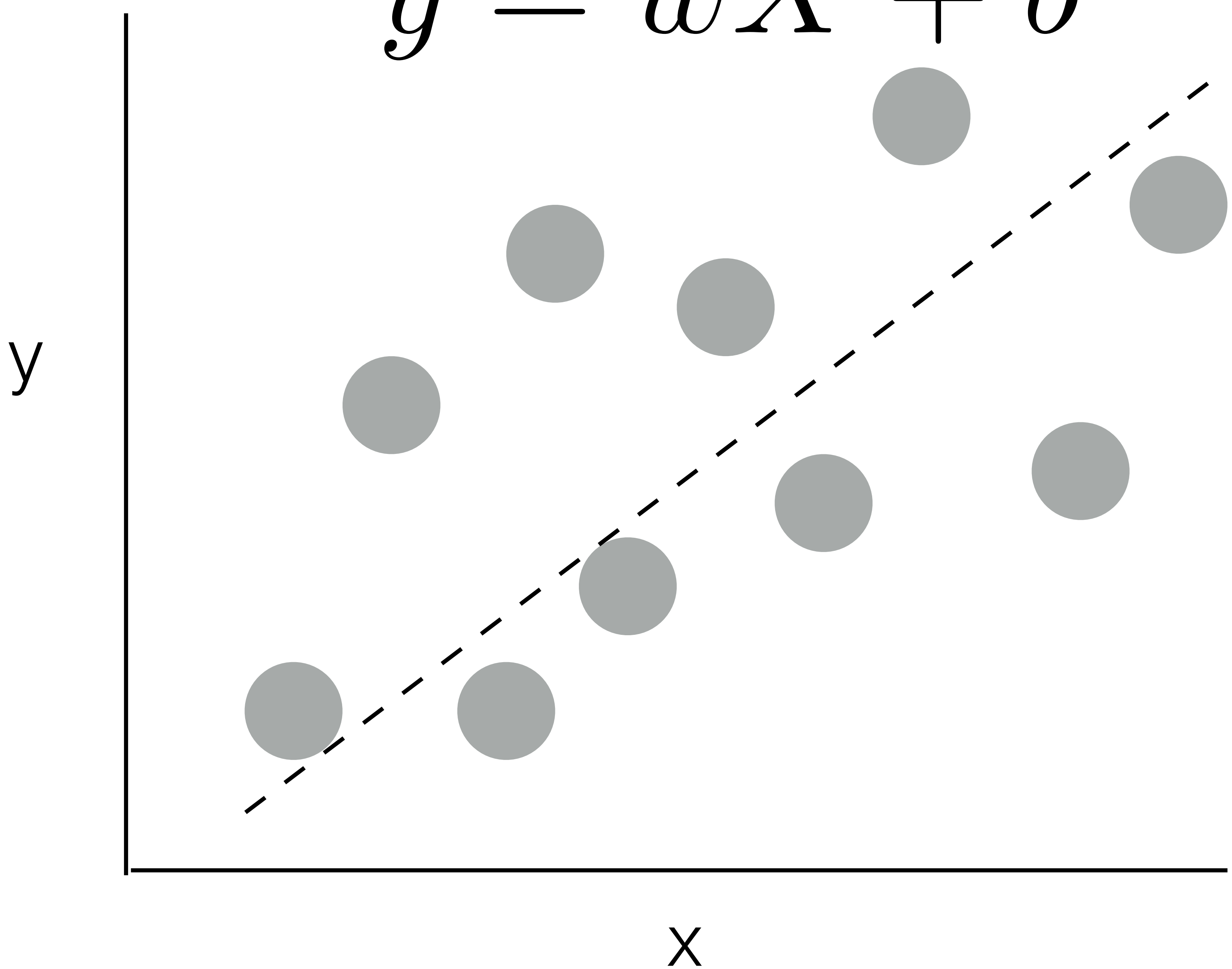




https://independentseminarblog.com/2018/01/12/moving-below-the-surface-3-gradient-descent-william/

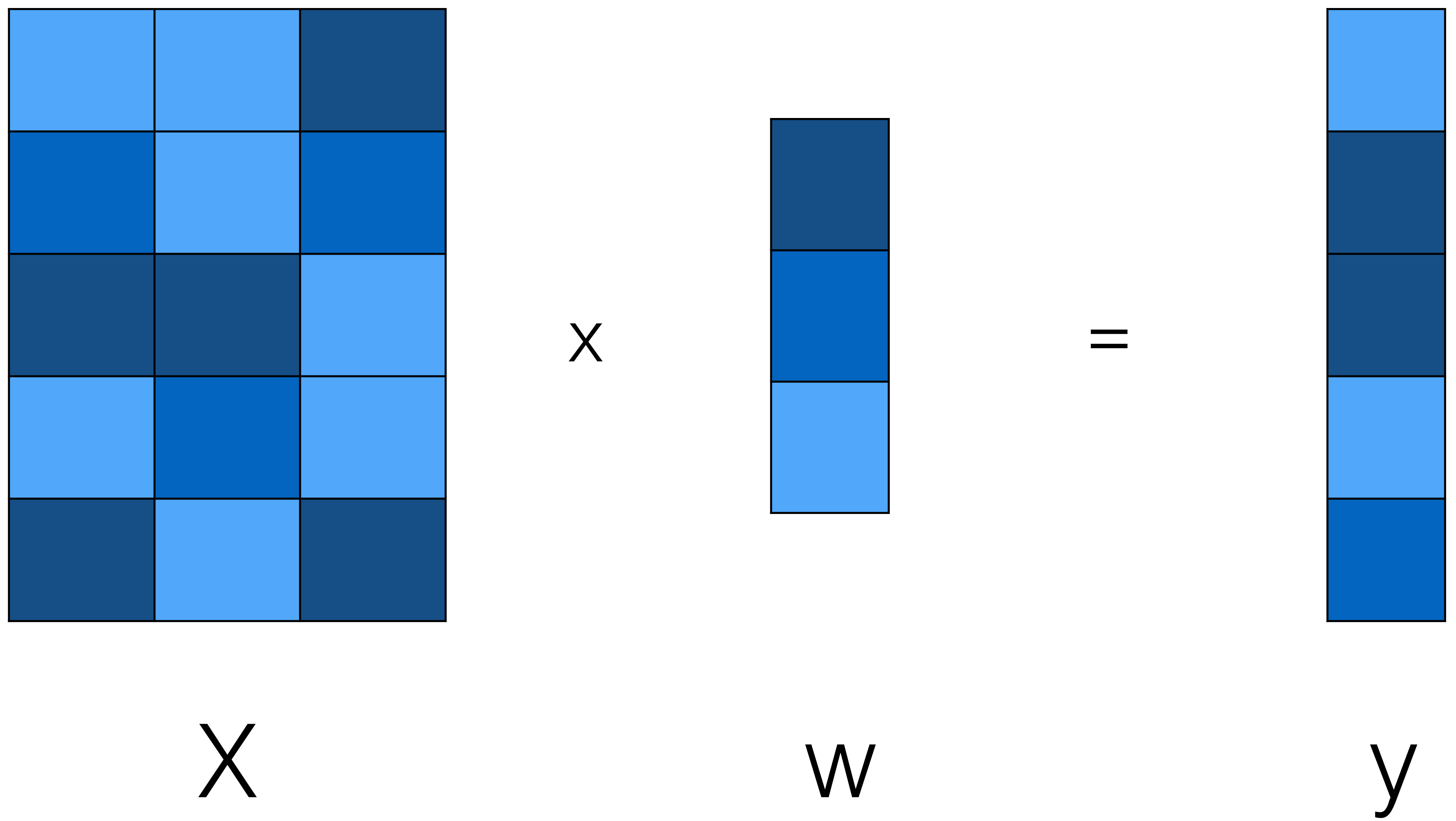
Linear Regression

$$y = wX + b$$



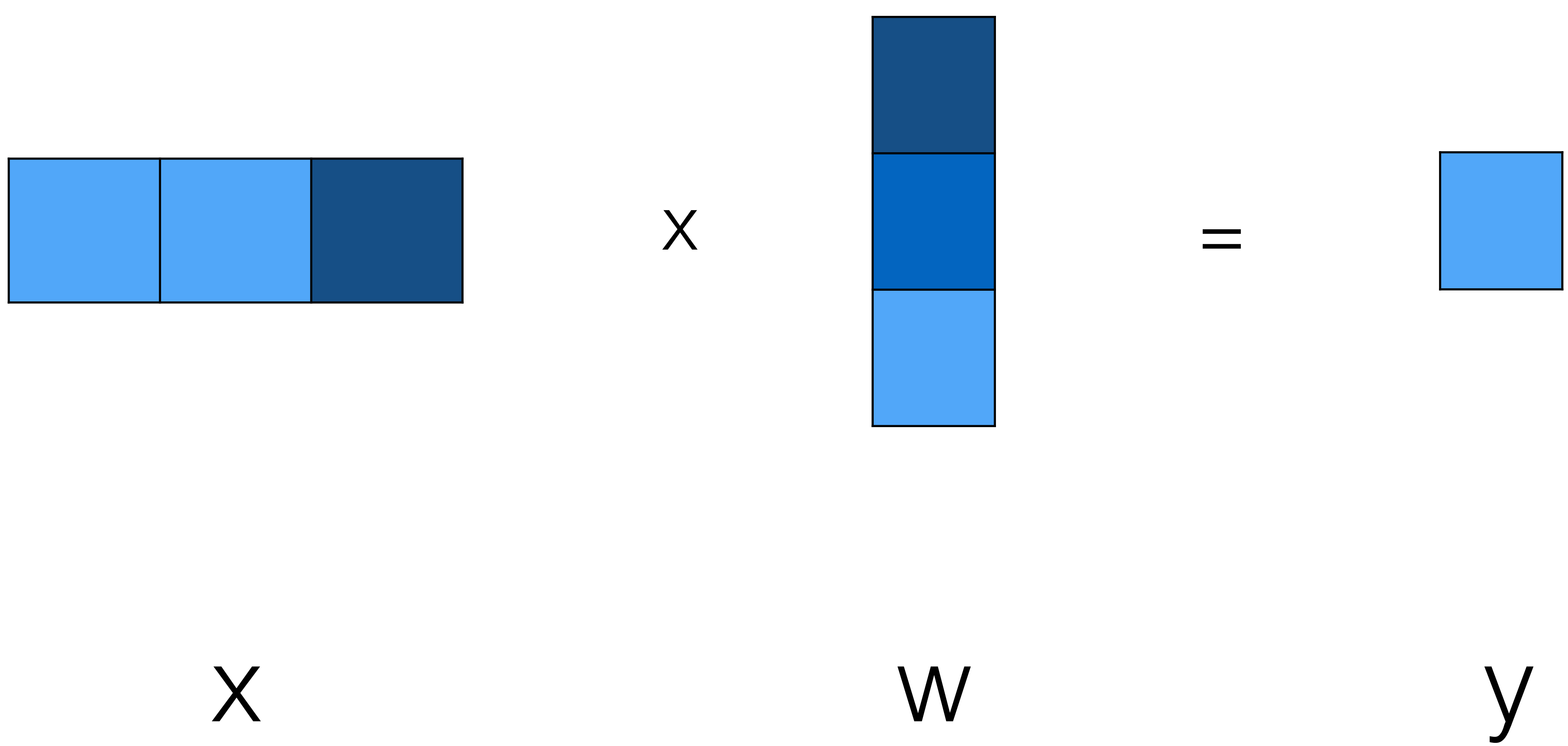
Linear Regression

$$y = wX + b$$



The most basic “network”

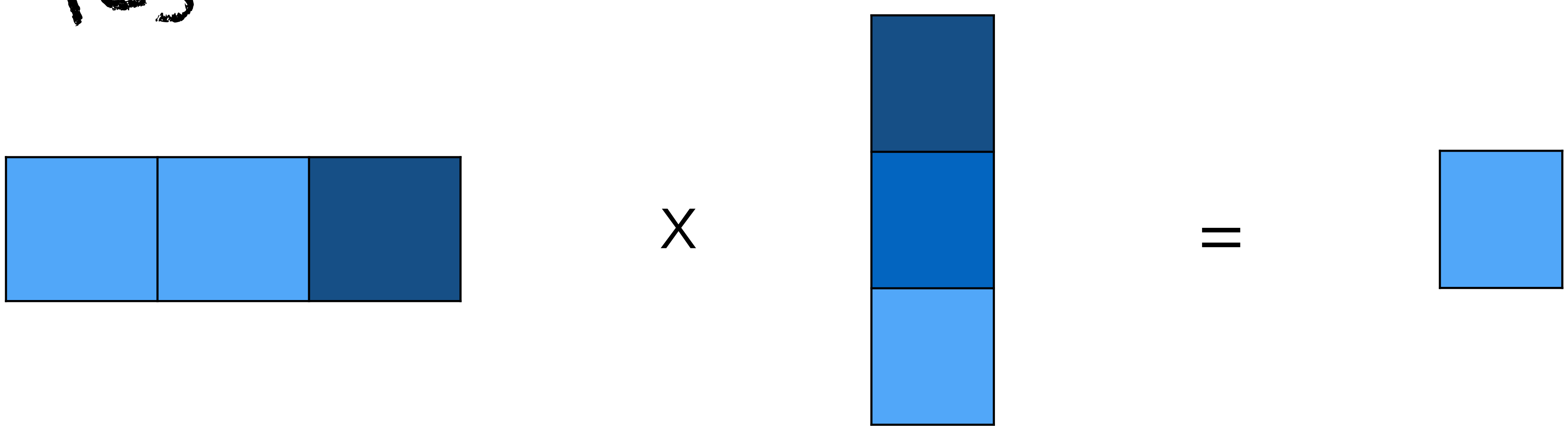
$$y = \vec{w} \cdot \vec{x}$$



The most basic “network”

“Perceptron”
(on-line linear
regression)

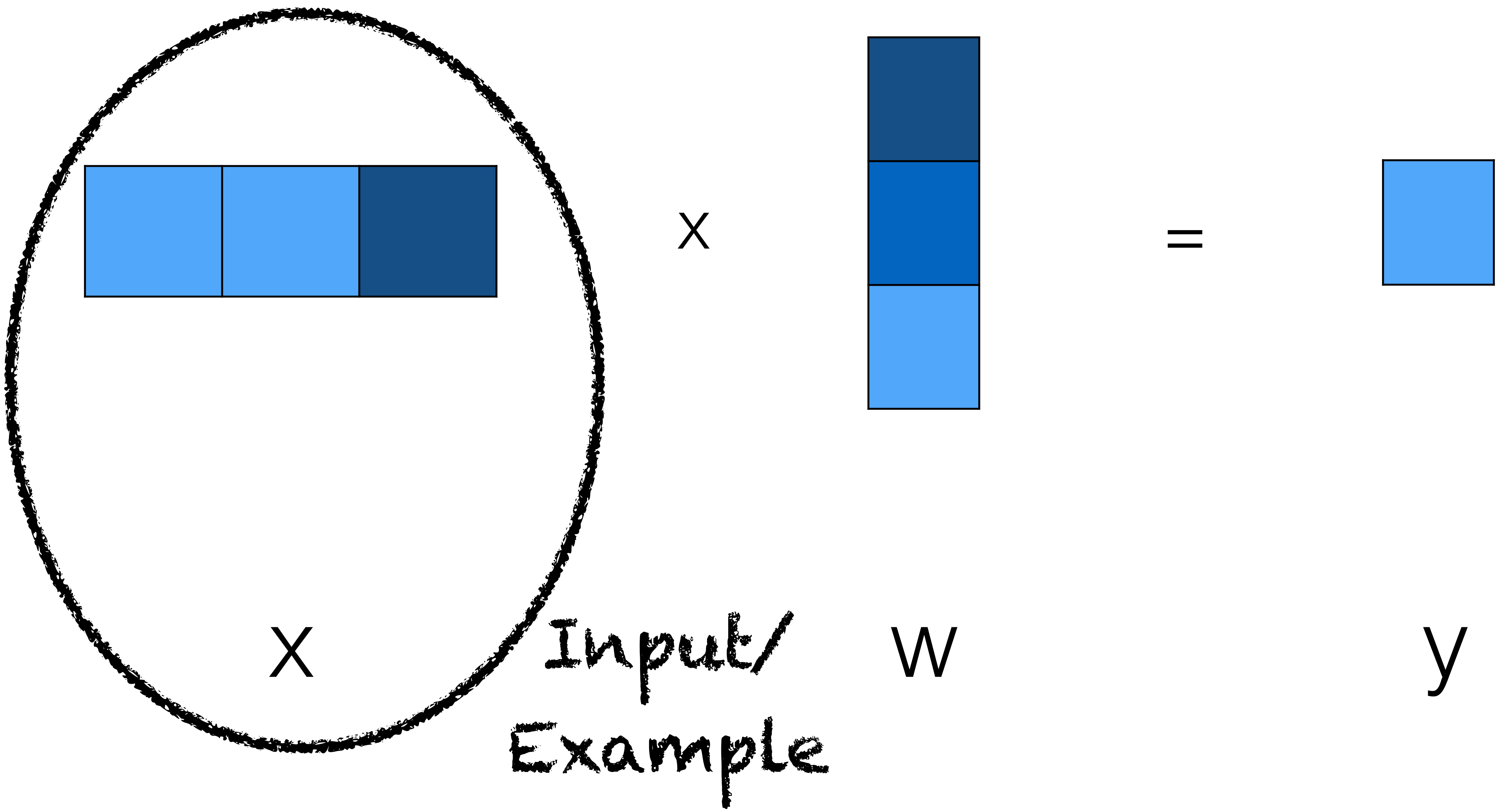
$$y = \vec{w} \cdot \vec{x}$$



x w y

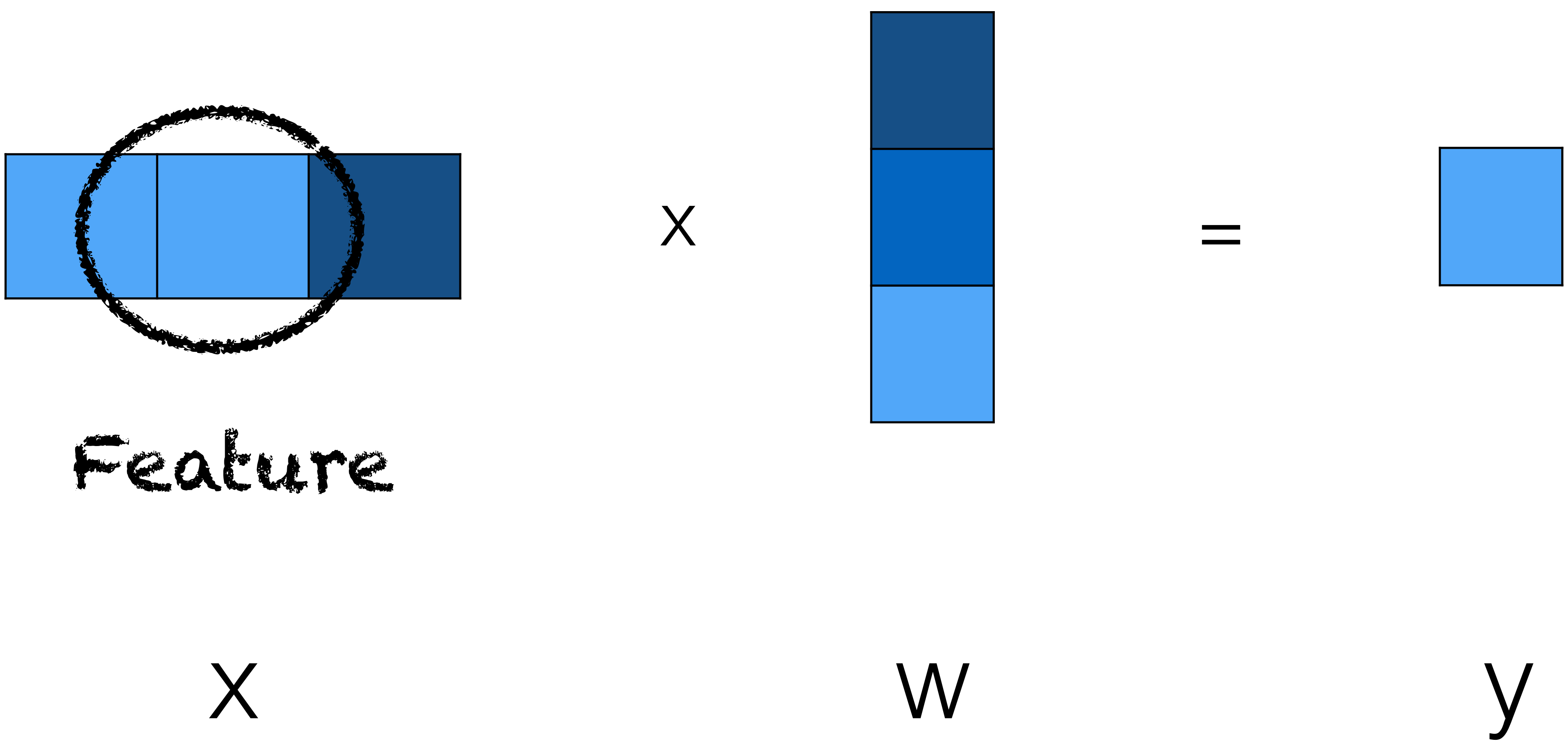
The most basic "network"

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



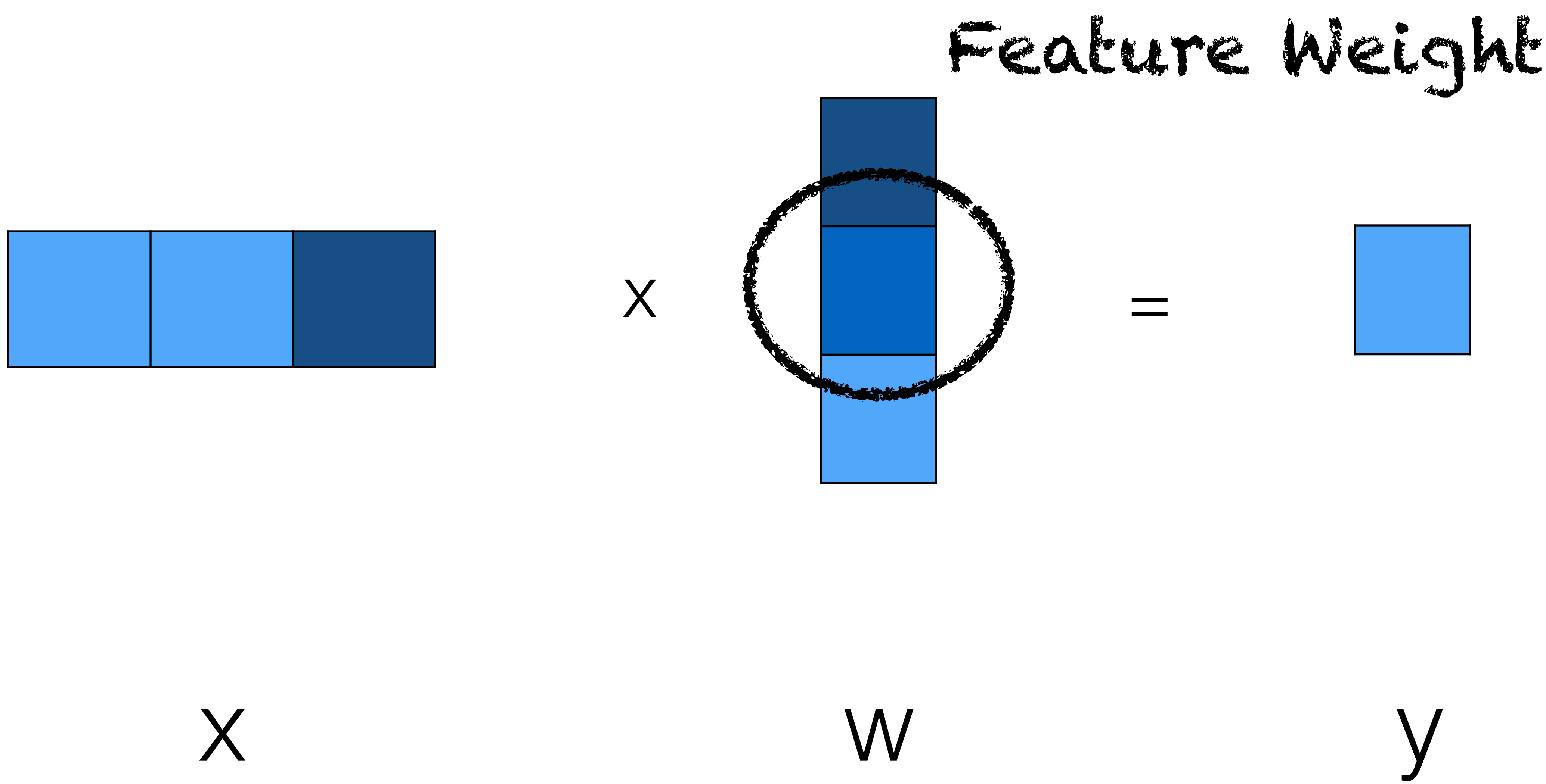
The most basic “network”

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



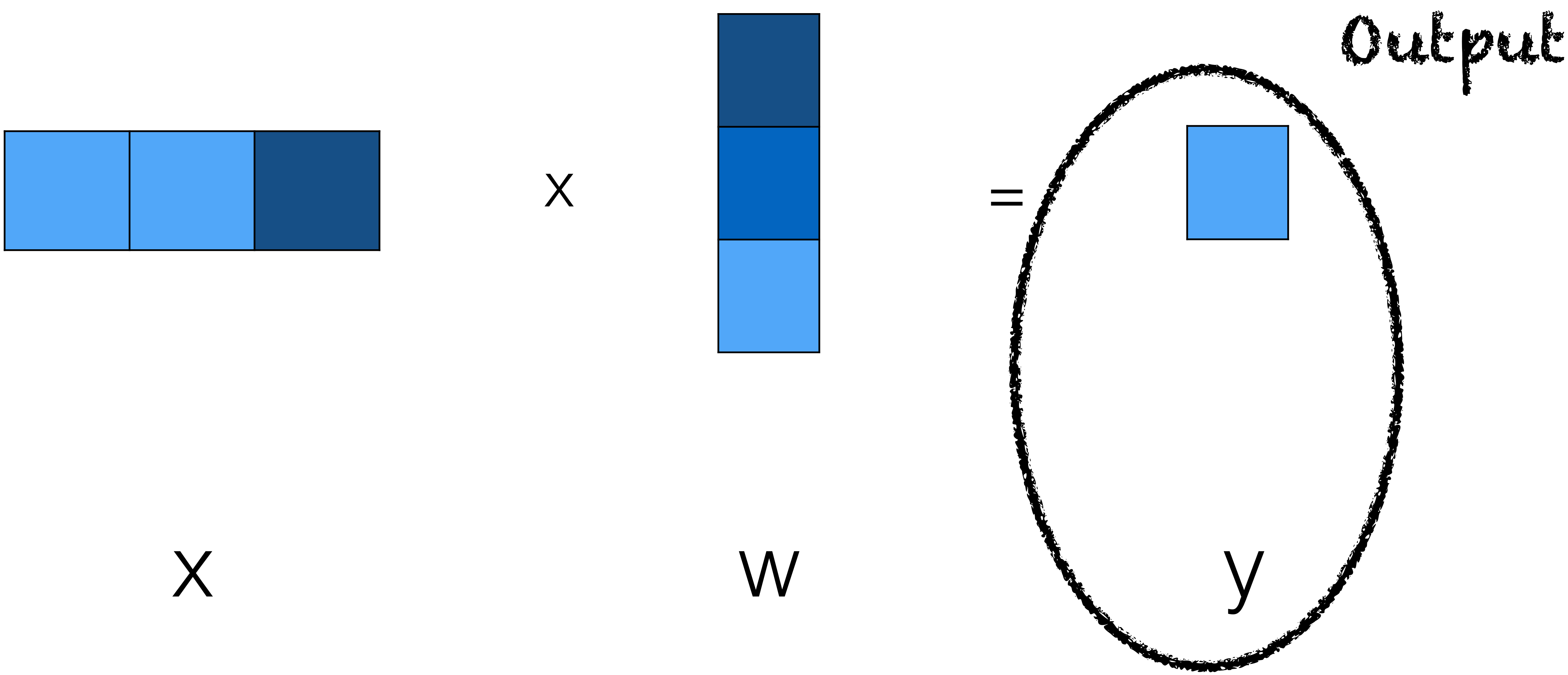
The most basic “network”

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



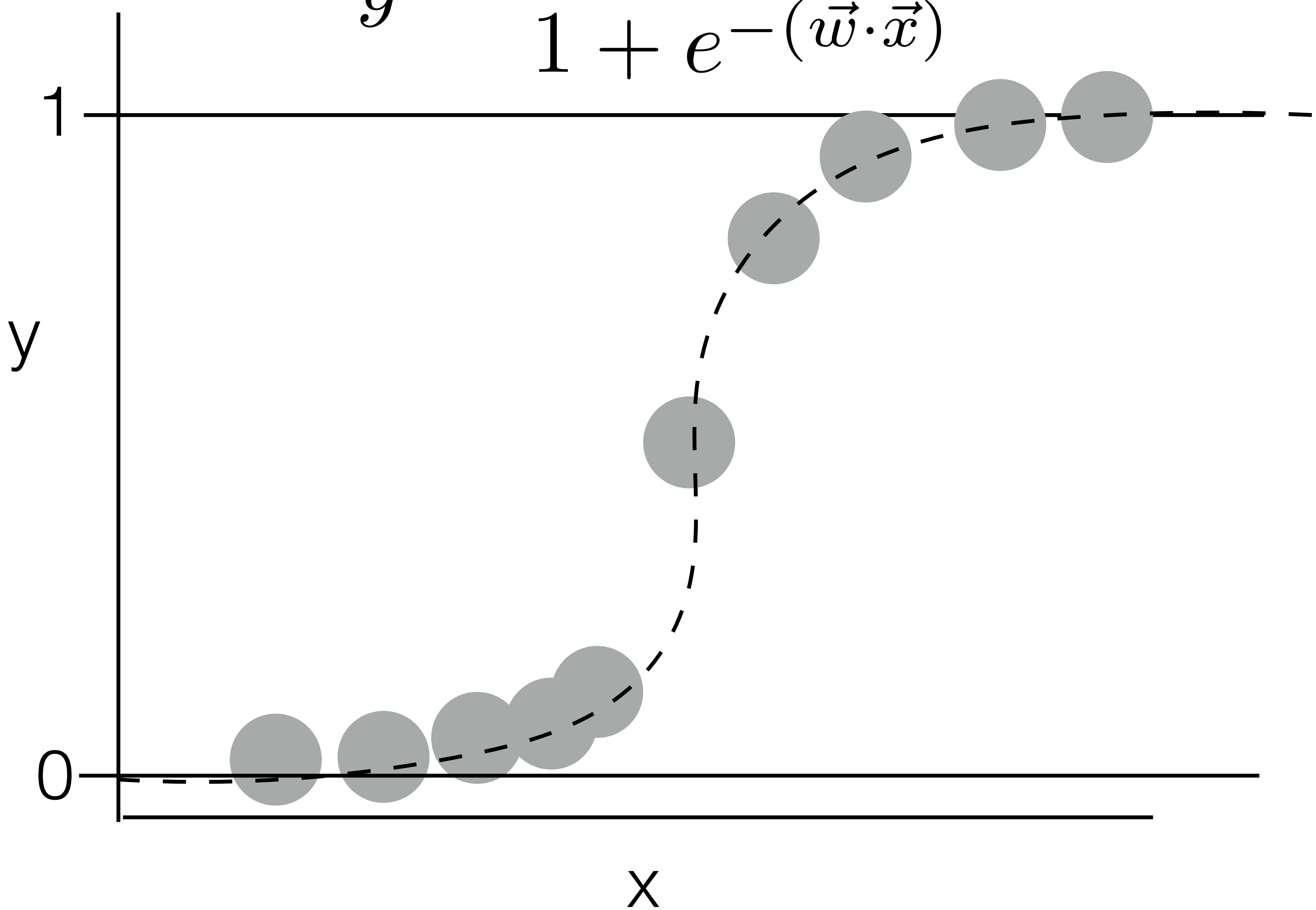
The most basic "network"

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



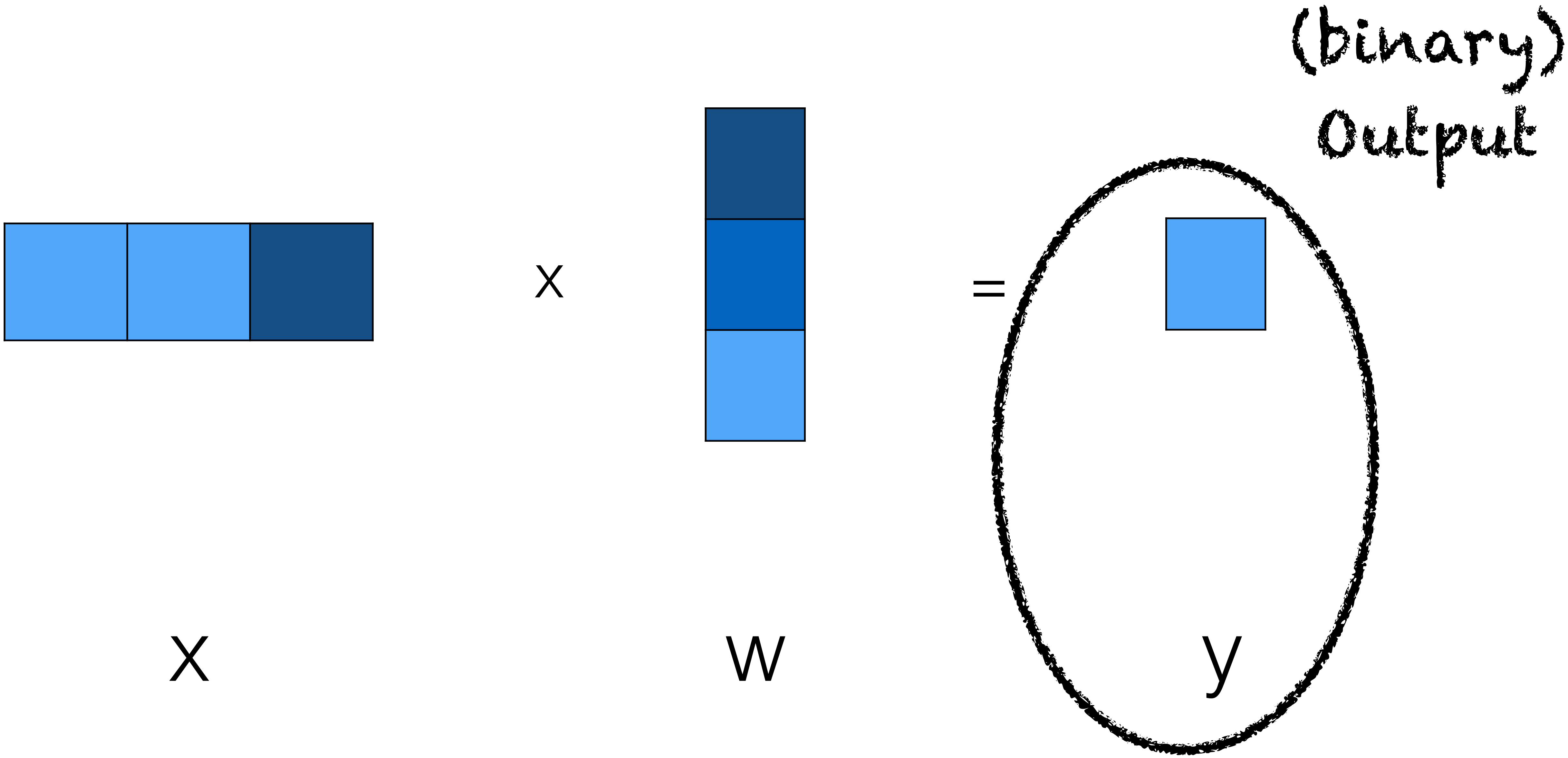
Logistic Regression

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



The most basic network

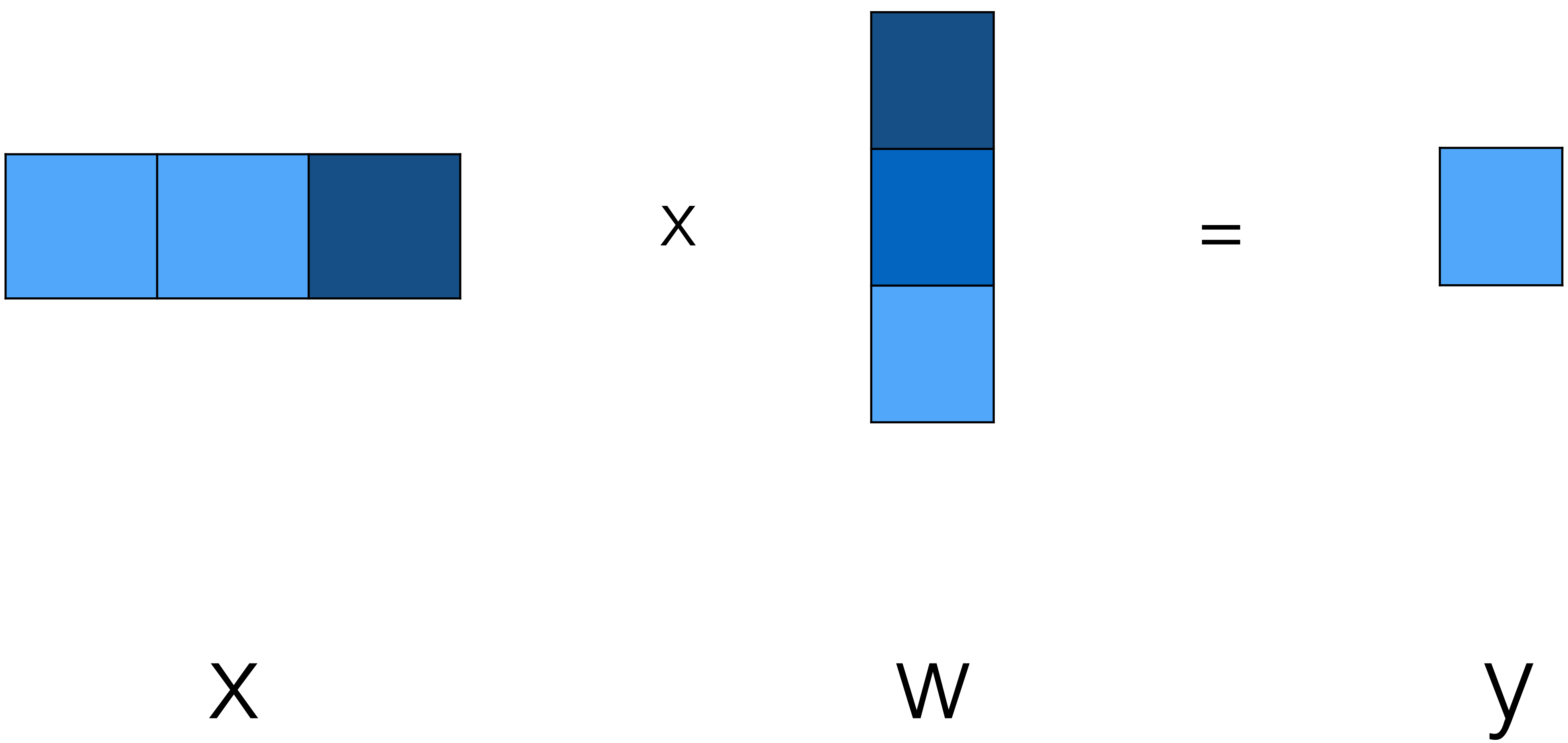
$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$



The most basic network

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

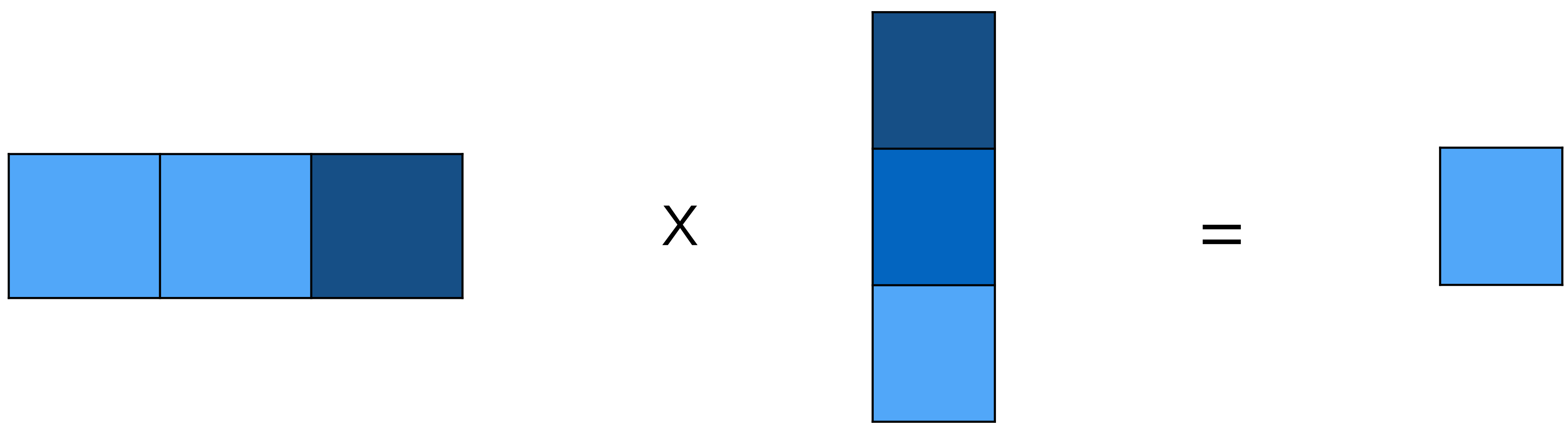
"activation function"



The most basic network

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

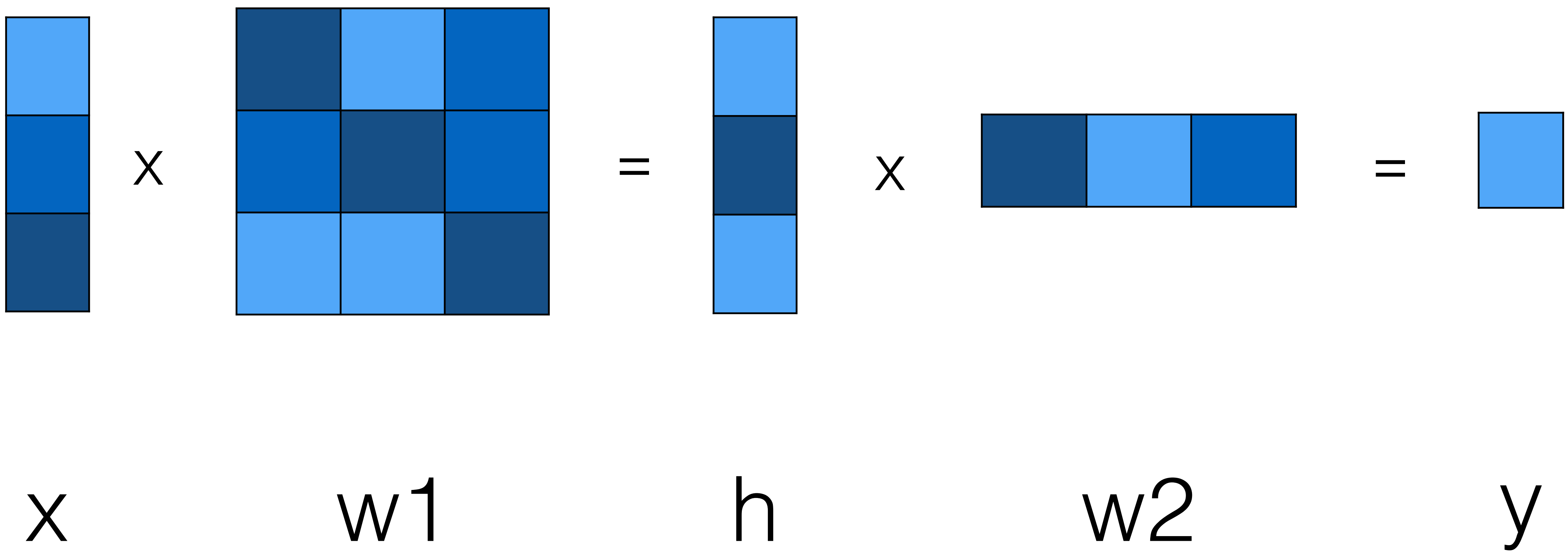
= non-linearity



x w y

The most basic network

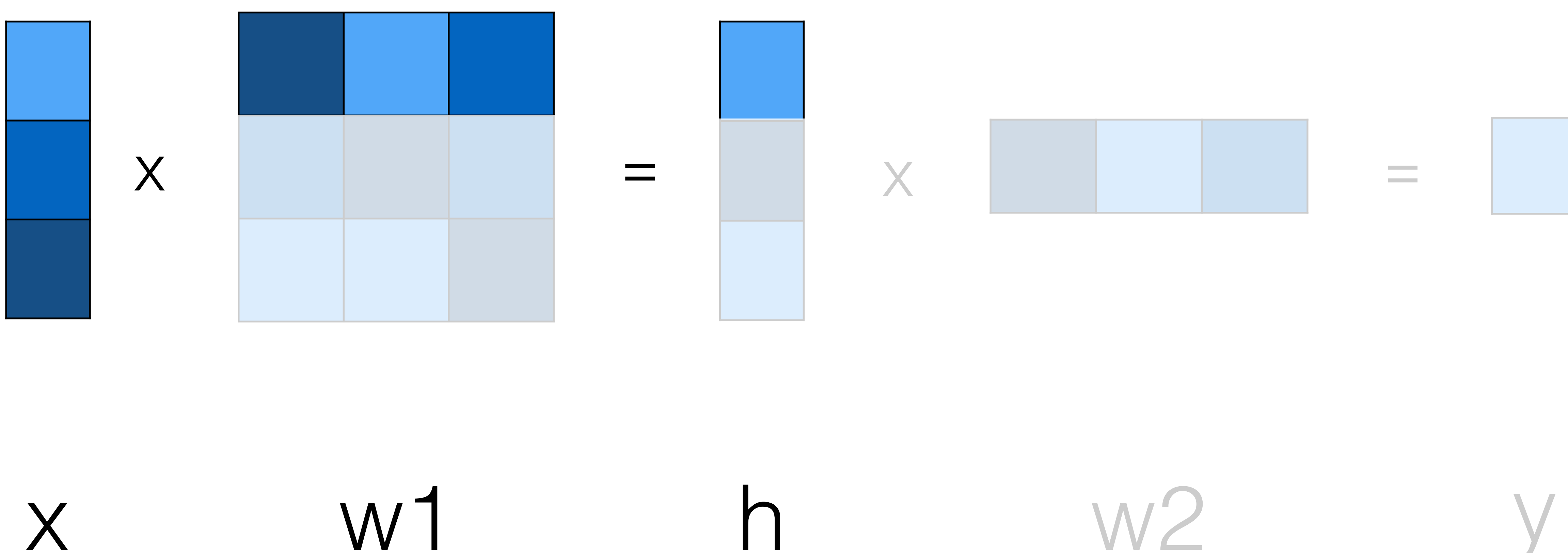
$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$



The most basic network

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

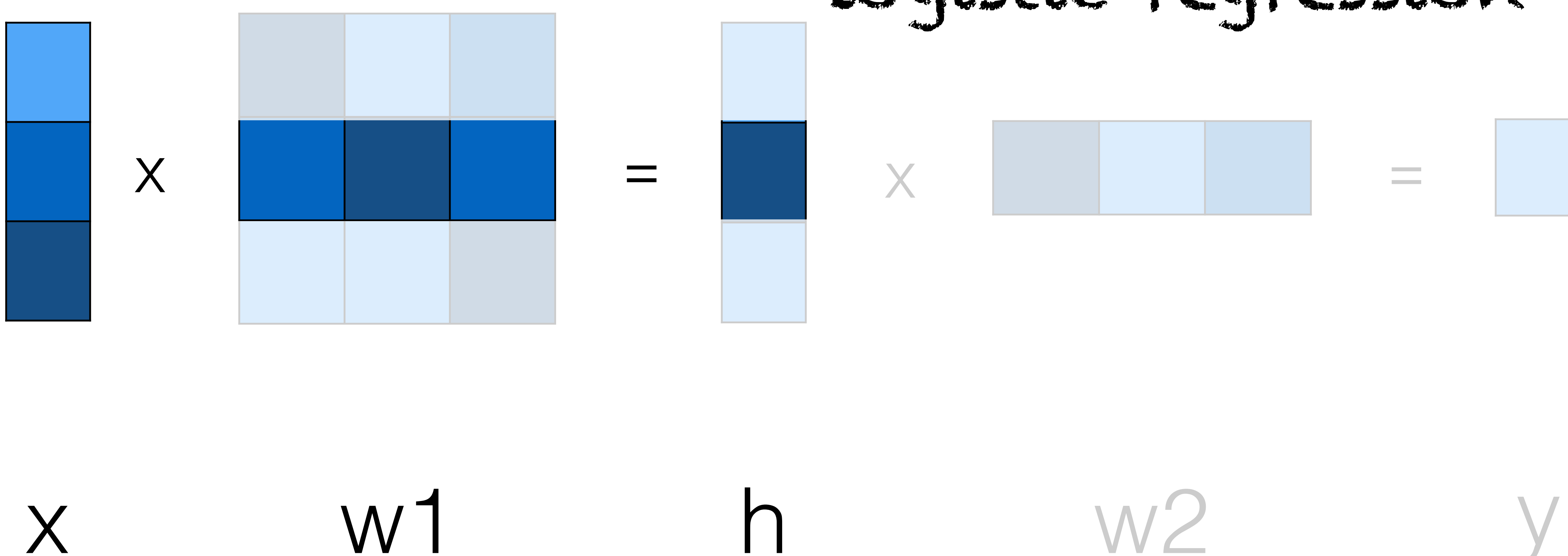
just a logistic regression



The most basic network

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

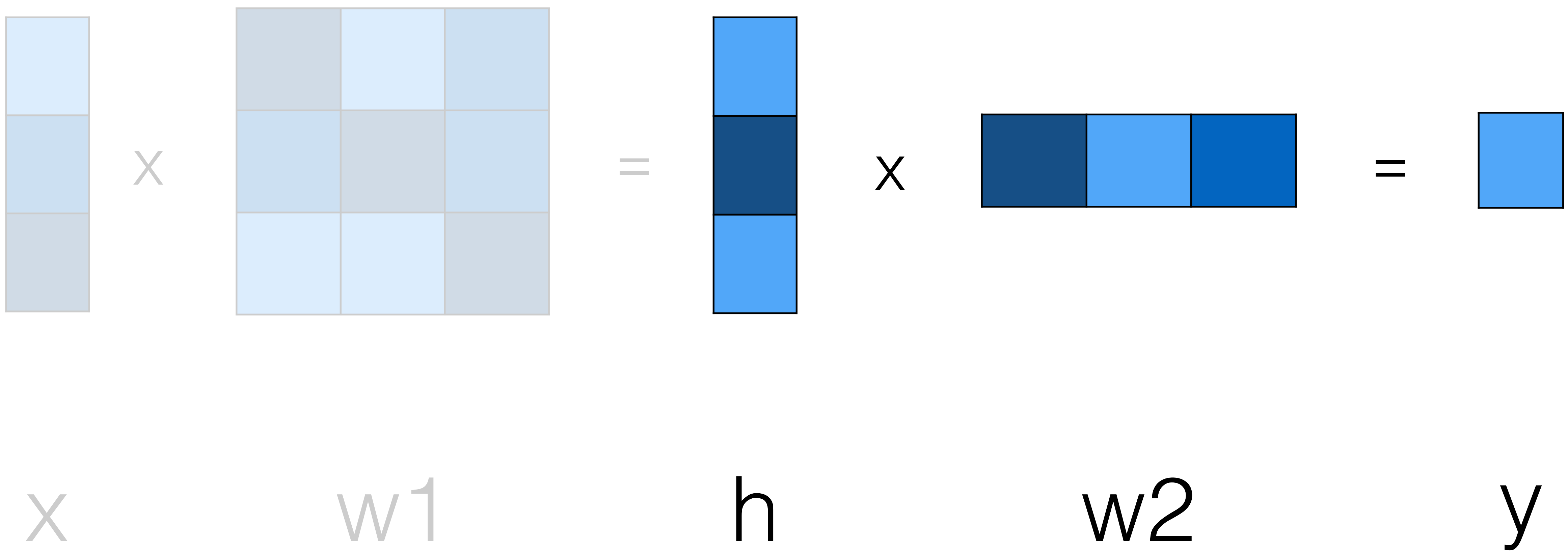
and another
logistic regression



The most basic network

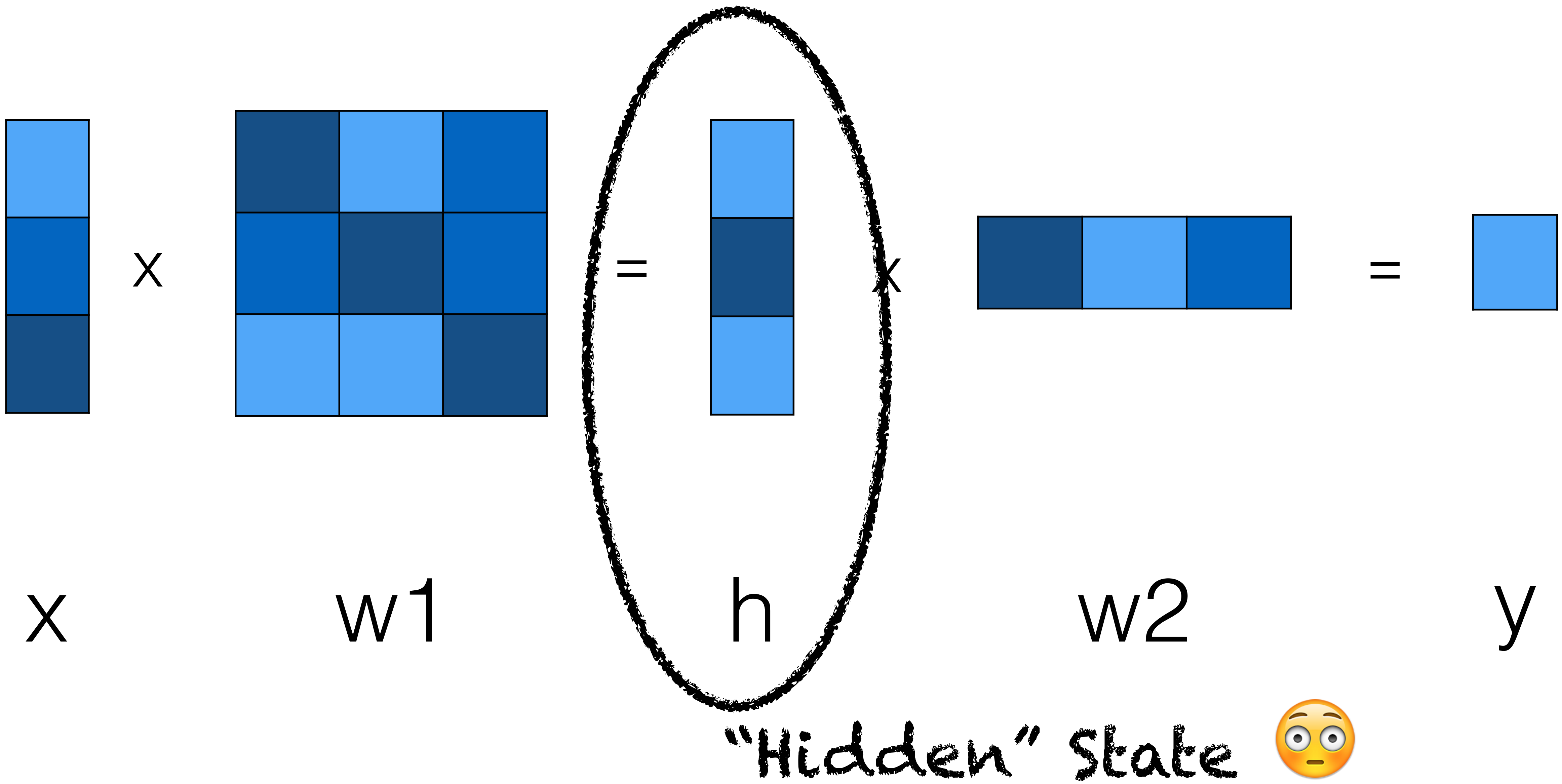
$$y = 1 \text{ if } \vec{w} \cdot \vec{x} > \tau \text{ else } 0$$

so many logistic regressions



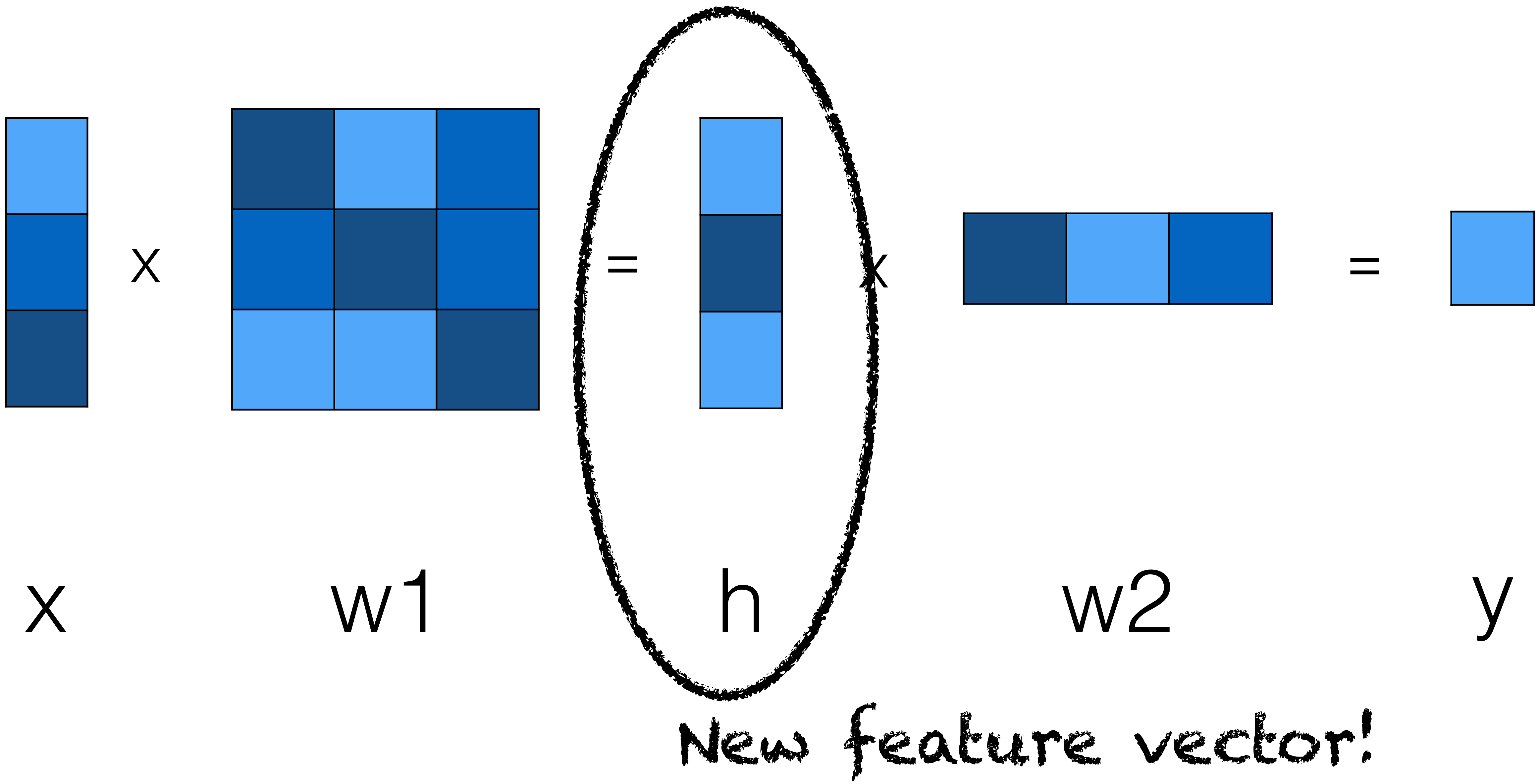
The most basic network

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



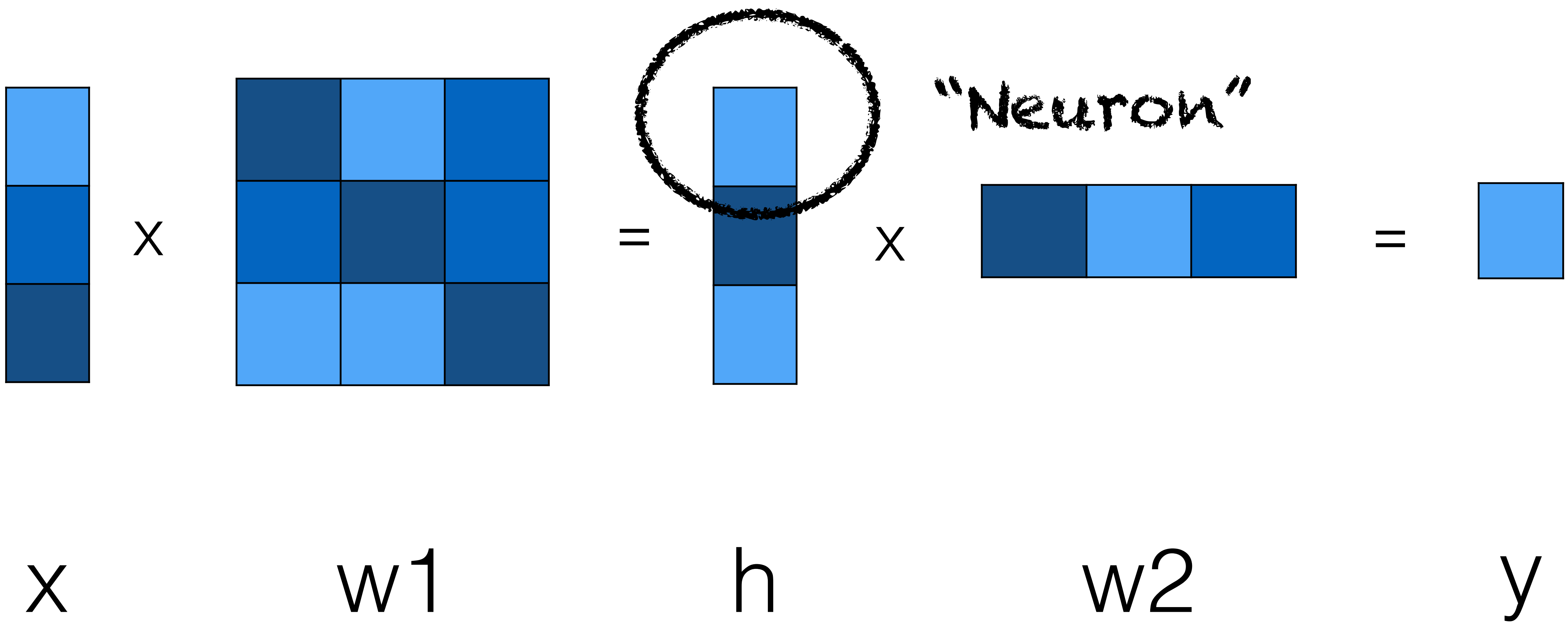
The most basic network

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



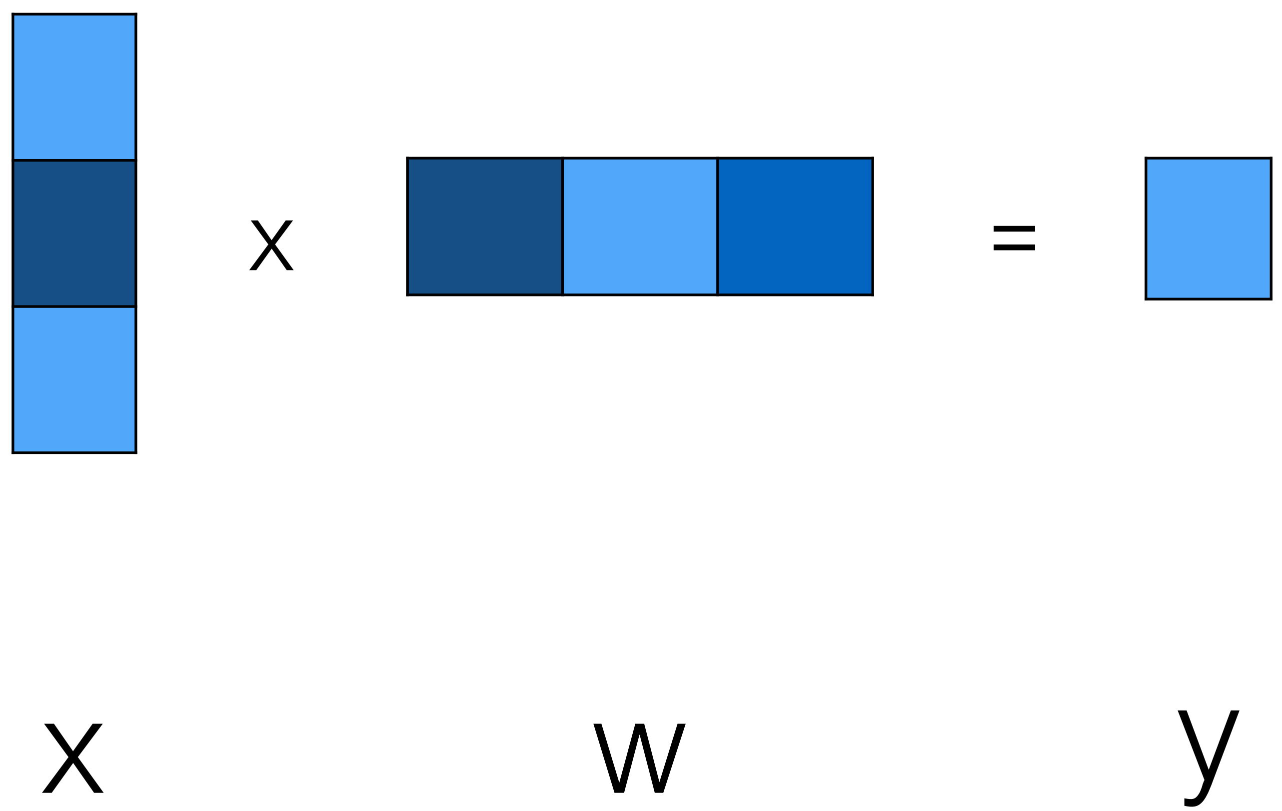
The most basic network

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



The most basic network

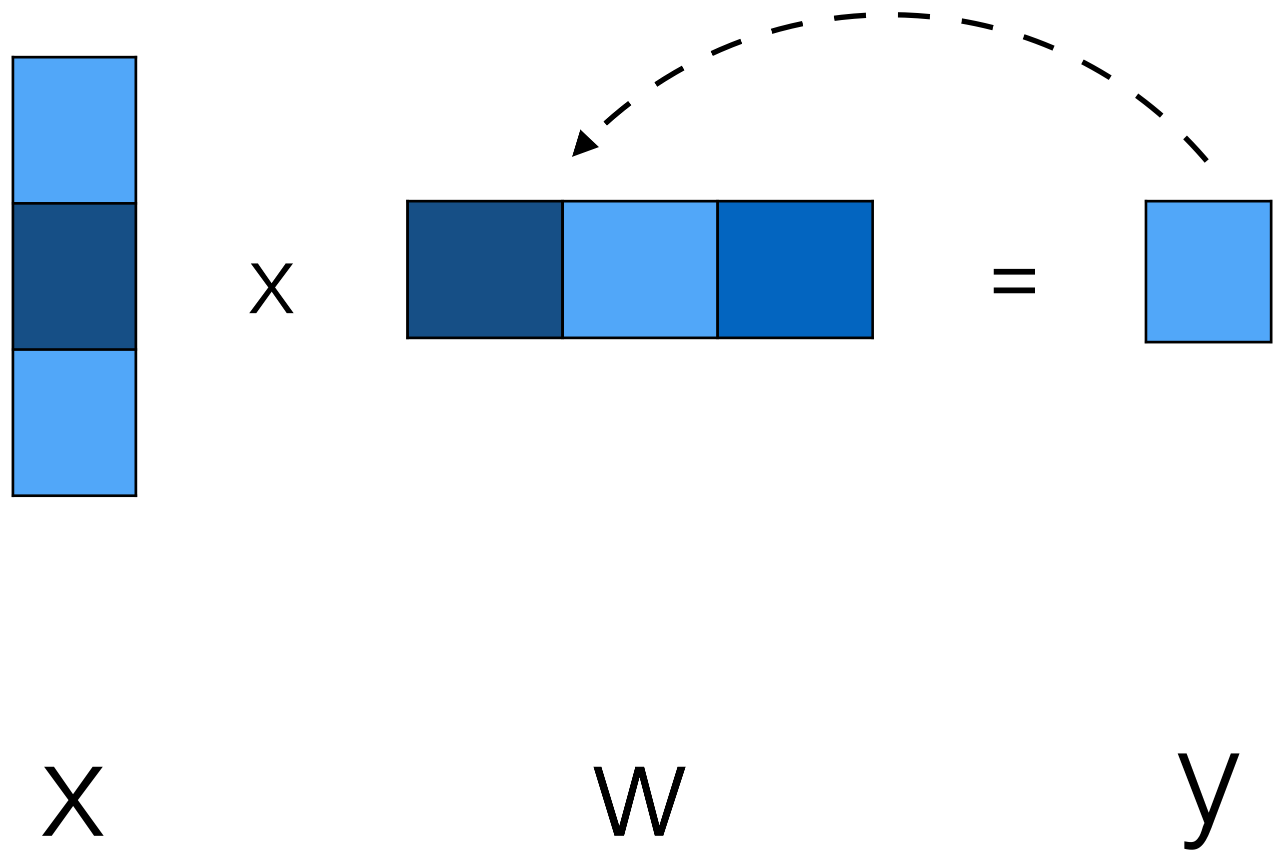
$$y = \vec{w} \cdot \vec{x}$$



The most basic network

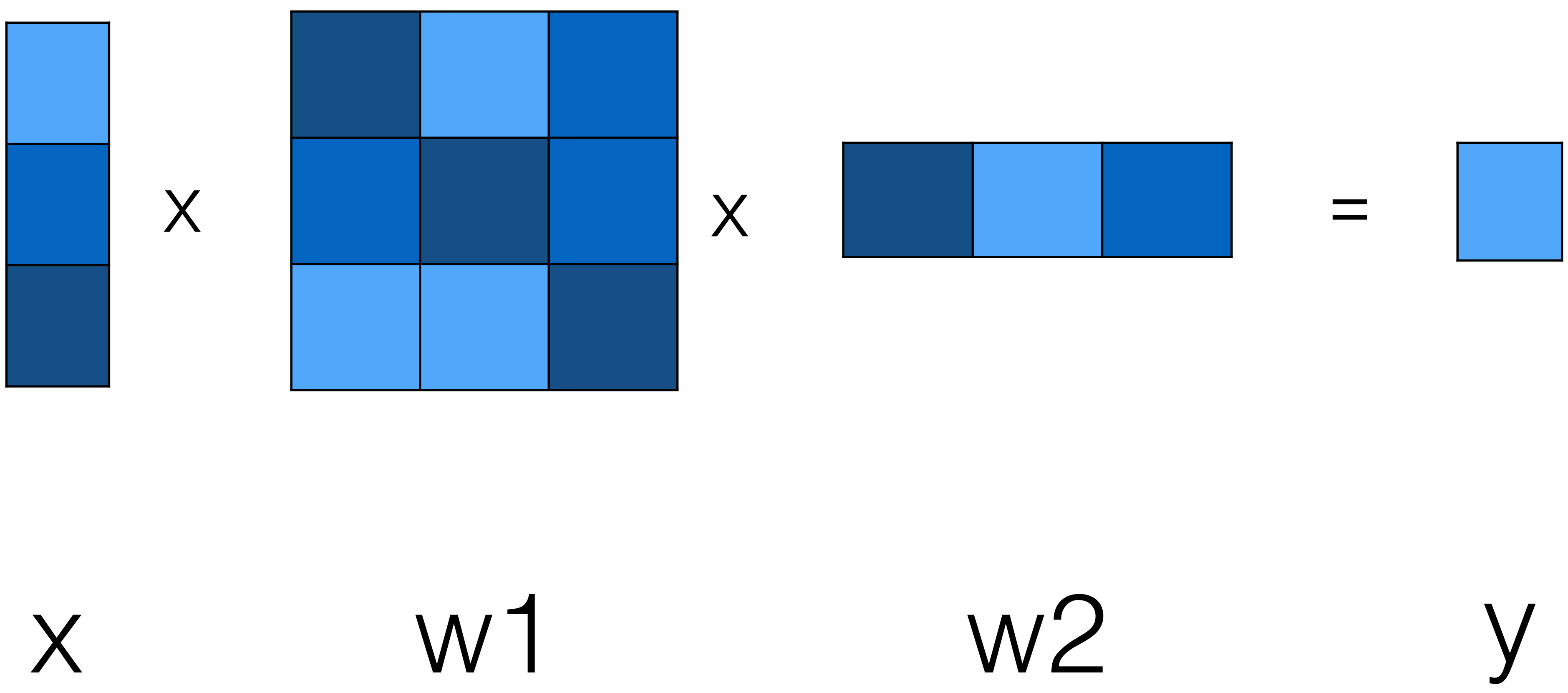
$$y = \vec{w} \cdot \vec{x}$$

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



The most basic network

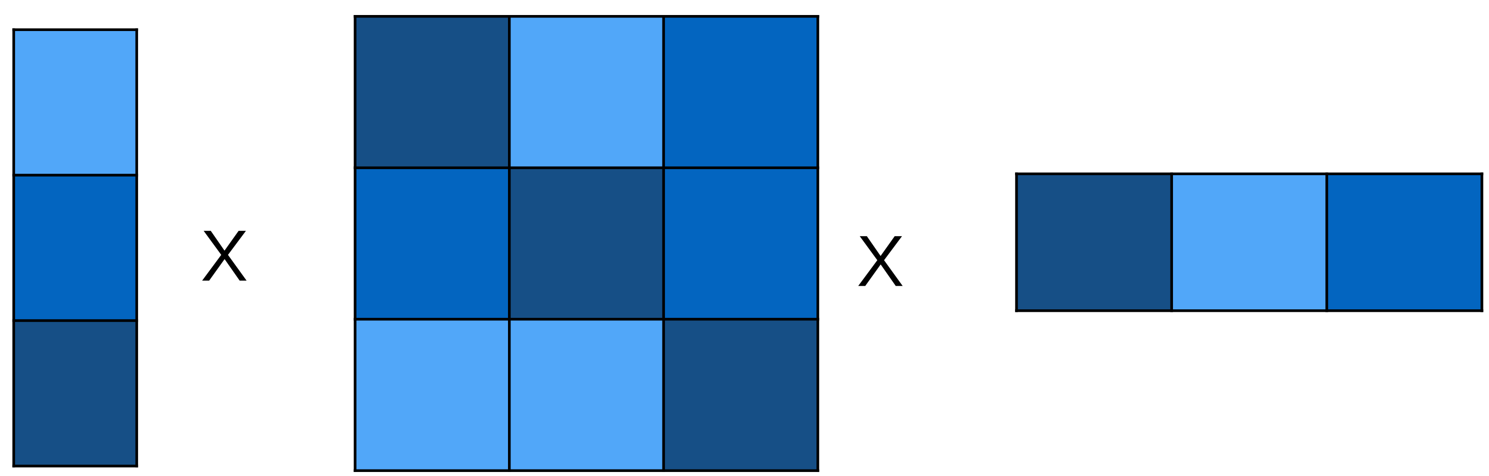
$$y = W_2 \cdot (W_1 \cdot \vec{x})$$



The most basic network

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

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



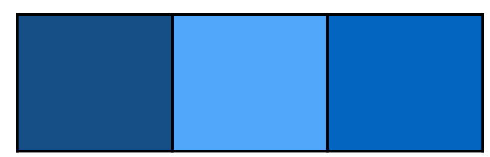
x w1 w2 y

The most basic network

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

$$f = \mathcal{L}(W_2 \cdot g(\vec{x}))$$

$$g = W_1 \cdot \vec{x}$$



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

x w1 w2 y

The most basic network

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

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

$$f = \mathcal{L}(W_2 \cdot g(\vec{x}))$$
$$g = W_1 \cdot \vec{x}$$

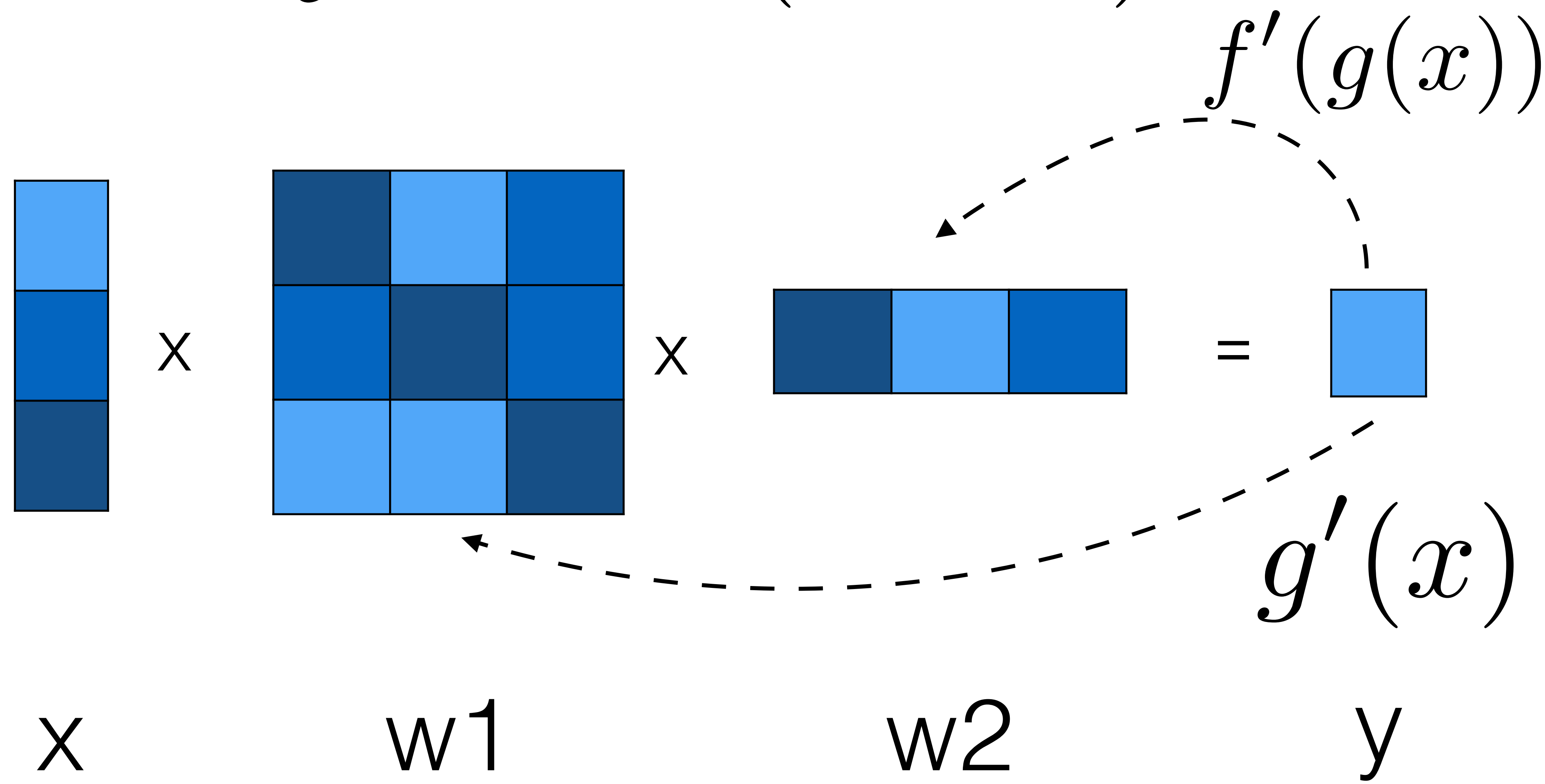
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

x w1

Chain Rule!

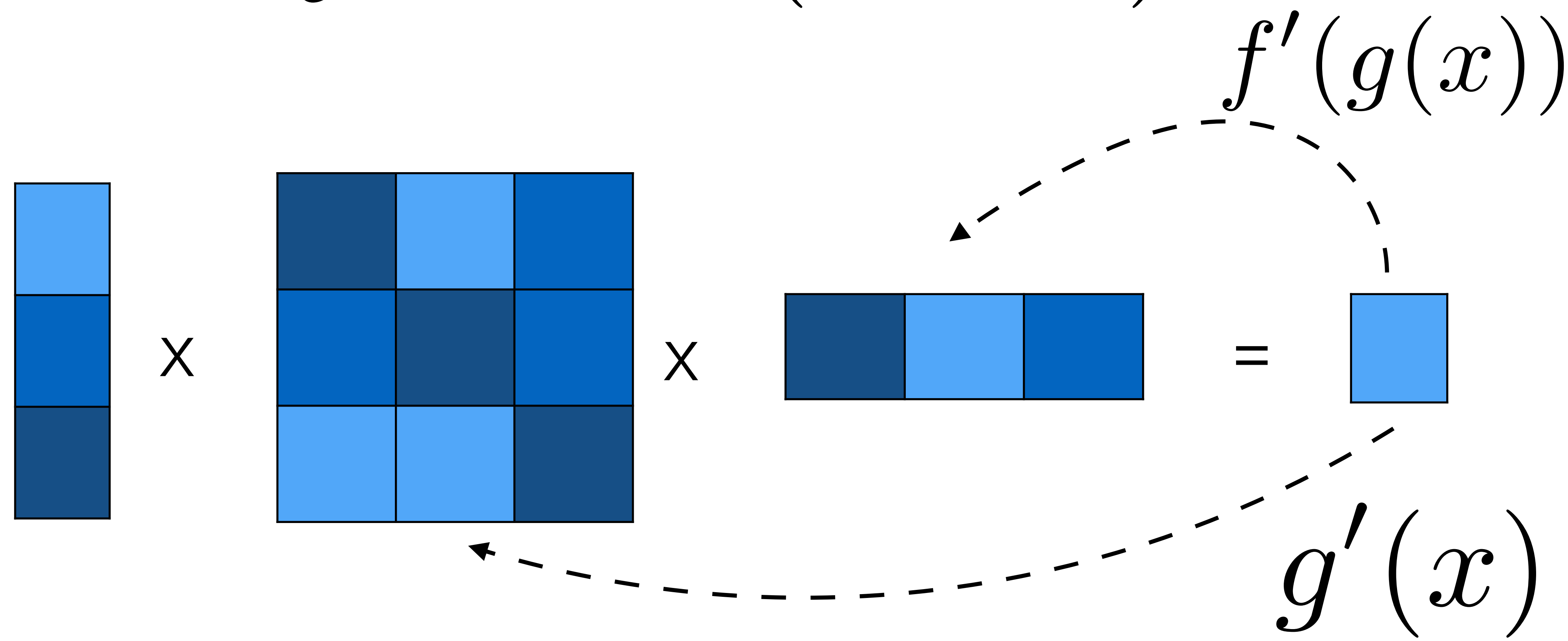
The most basic network

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



The most basic network

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



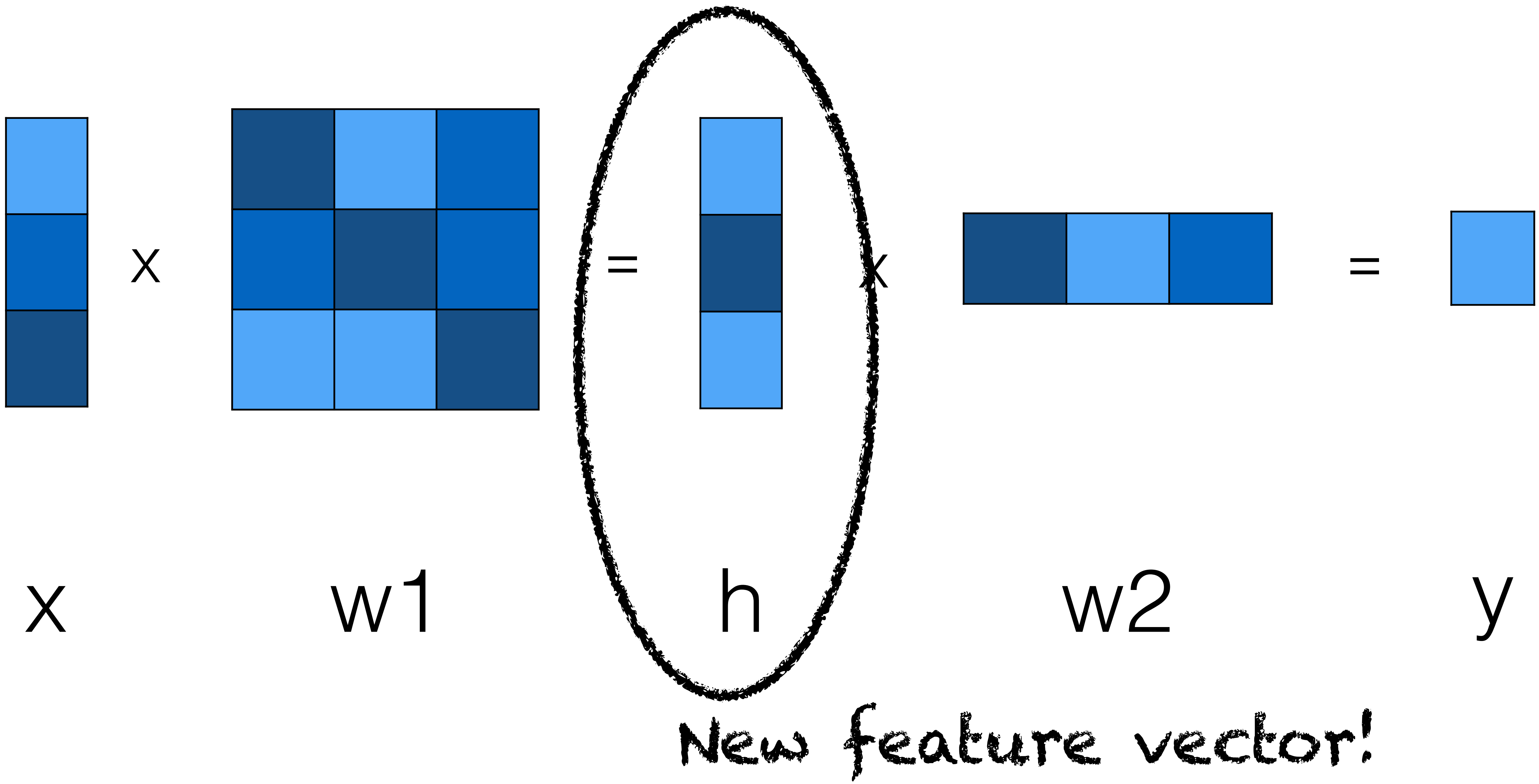
X

Backpropagation
("backprop")

y

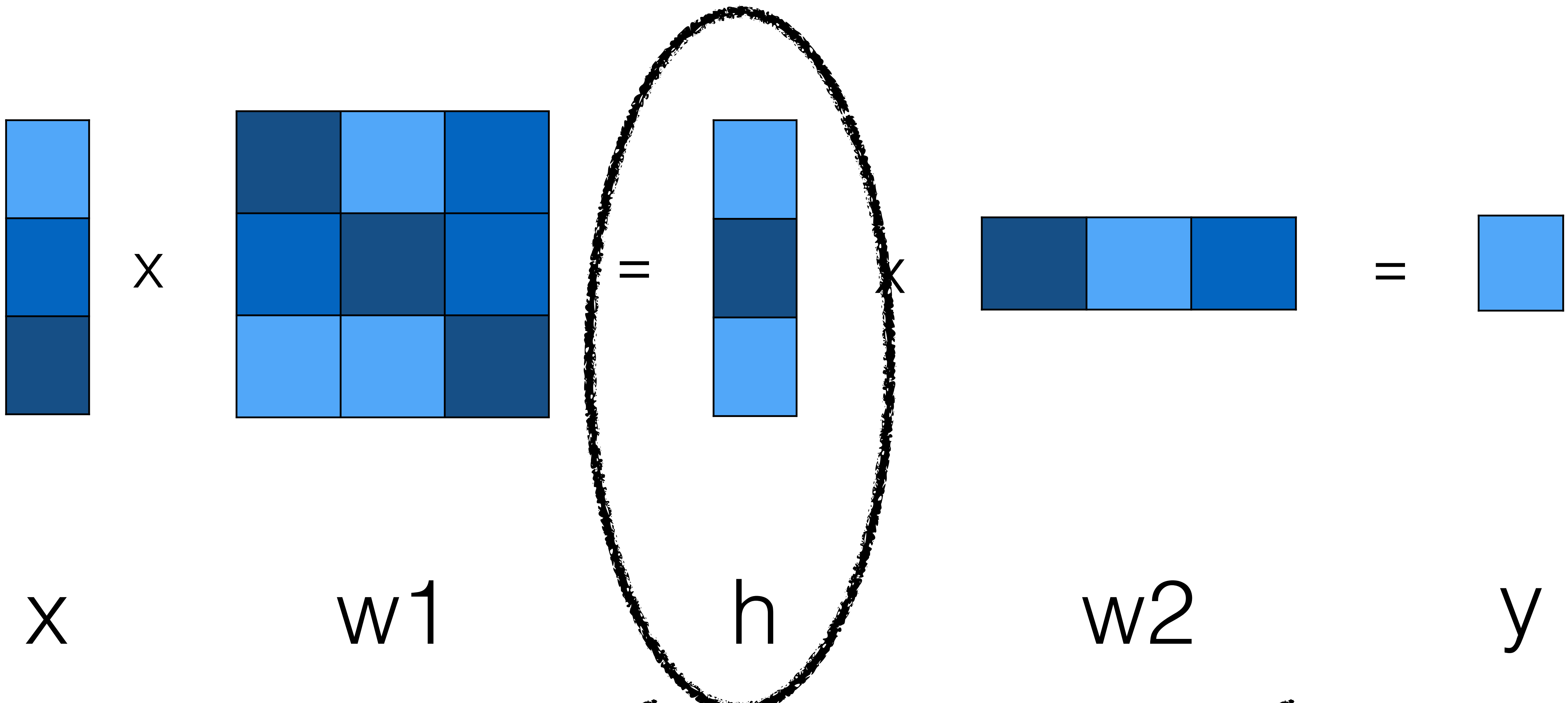
The most basic network

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



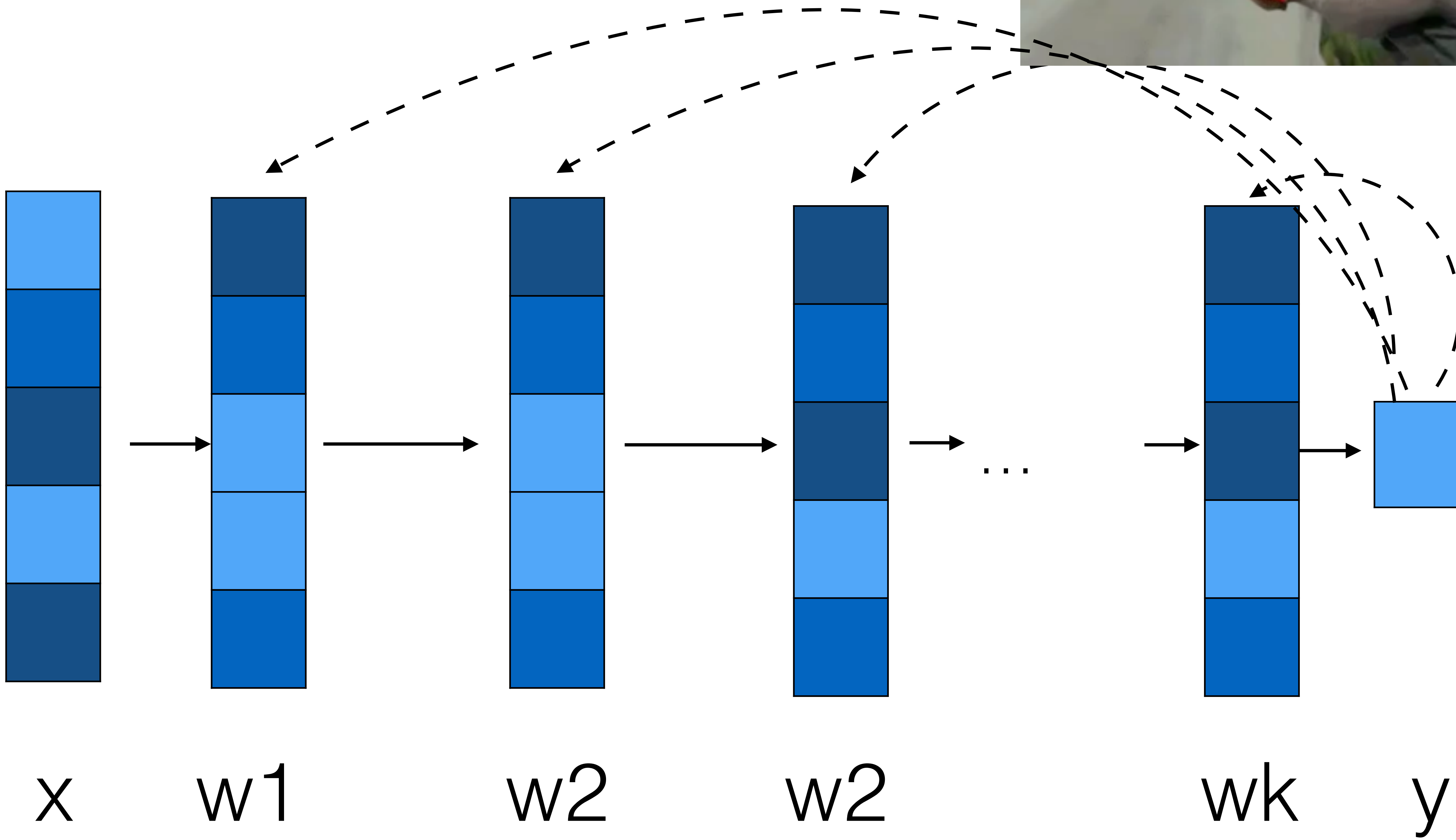
The most basic network

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



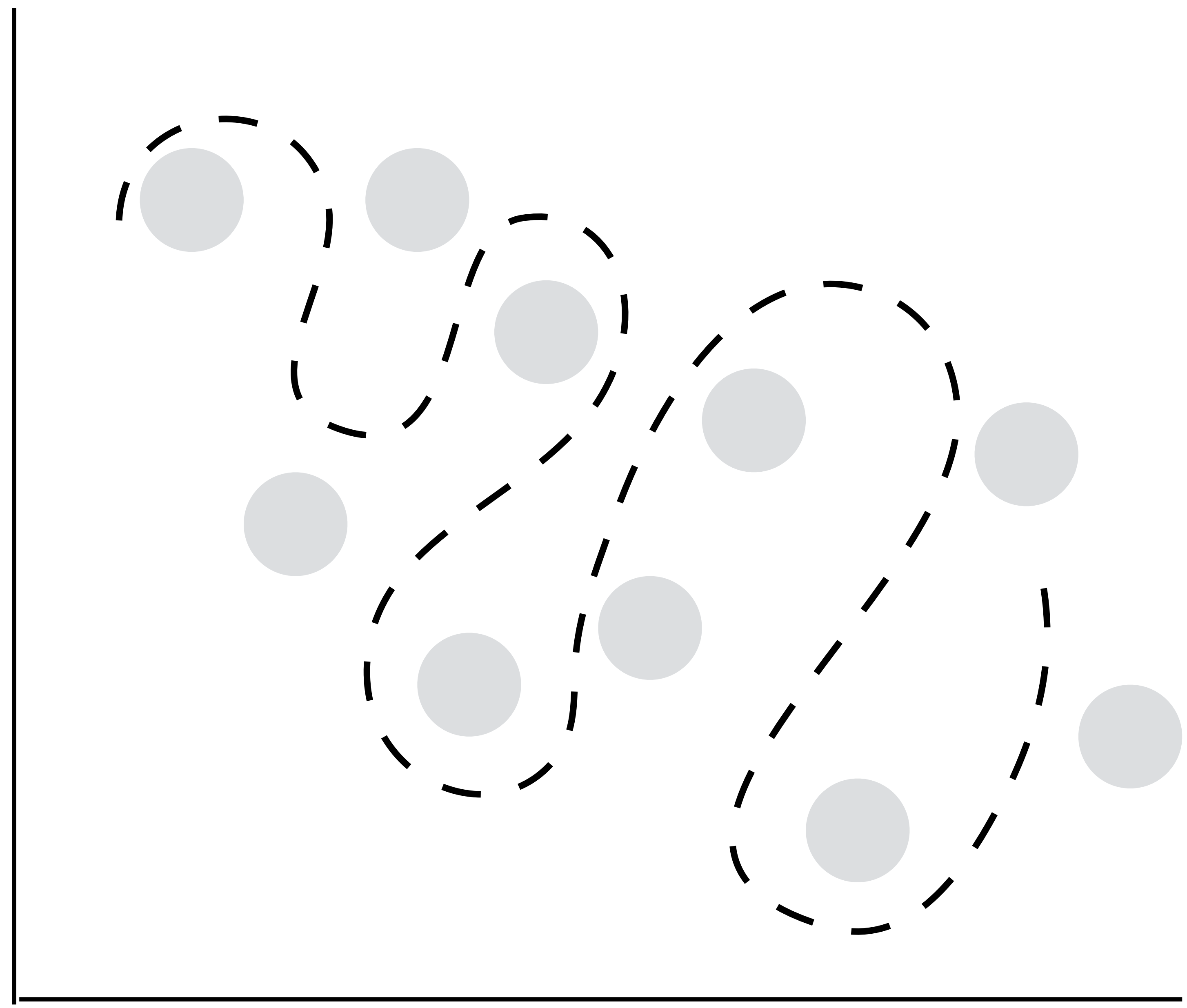
Specifically, whatever features help you solve the task best!

Weeeeeeee!!!!



* vector dimensions not to scale :)

tons of nonlinear parameters
= tons of flexibility



Nothing new, nothing fancy

Nothing new, nothing fancy

- NNs have been around since forever (the 80s!! :-0)

Nothing new, nothing fancy

- NNs have been around since forever (the 80s!! :-0)
- A vanilla MLP can theoretically approximate any function (“universal approximator”)

Nothing new, nothing fancy

- NNs have been around since forever (the 80s!! :-0)
- A vanilla MLP can theoretically approximate any function (“universal approximator”)
- (Note: “can” != “do”)

Yes, okay, but obviously something has changed recently

Yes, okay, but obviously something has changed recently

- Recently became viable for a few reasons, e.g.…

Yes, okay, but obviously something has changed recently

- Recently became viable for a few reasons, e.g.…
- Backprop (/autodiff): now we can build deep networks and actually train them

Yes, okay, but obviously something has changed recently

- Recently became viable for a few reasons, e.g.…
- Backprop (/autodiff): now we can build deep networks and actually train them
- GPUs: and we can train them fast

Yes, okay, but obviously something has changed recently

- Recently became viable for a few reasons, e.g.…
- Backprop (/autodiff): now we can build 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?

Why are they so much better, though?

"Its how the brain works."

Why are they so much better, though?

- “Its how the brain works.” —> NO!

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

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

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 (like MF, more in a bit)

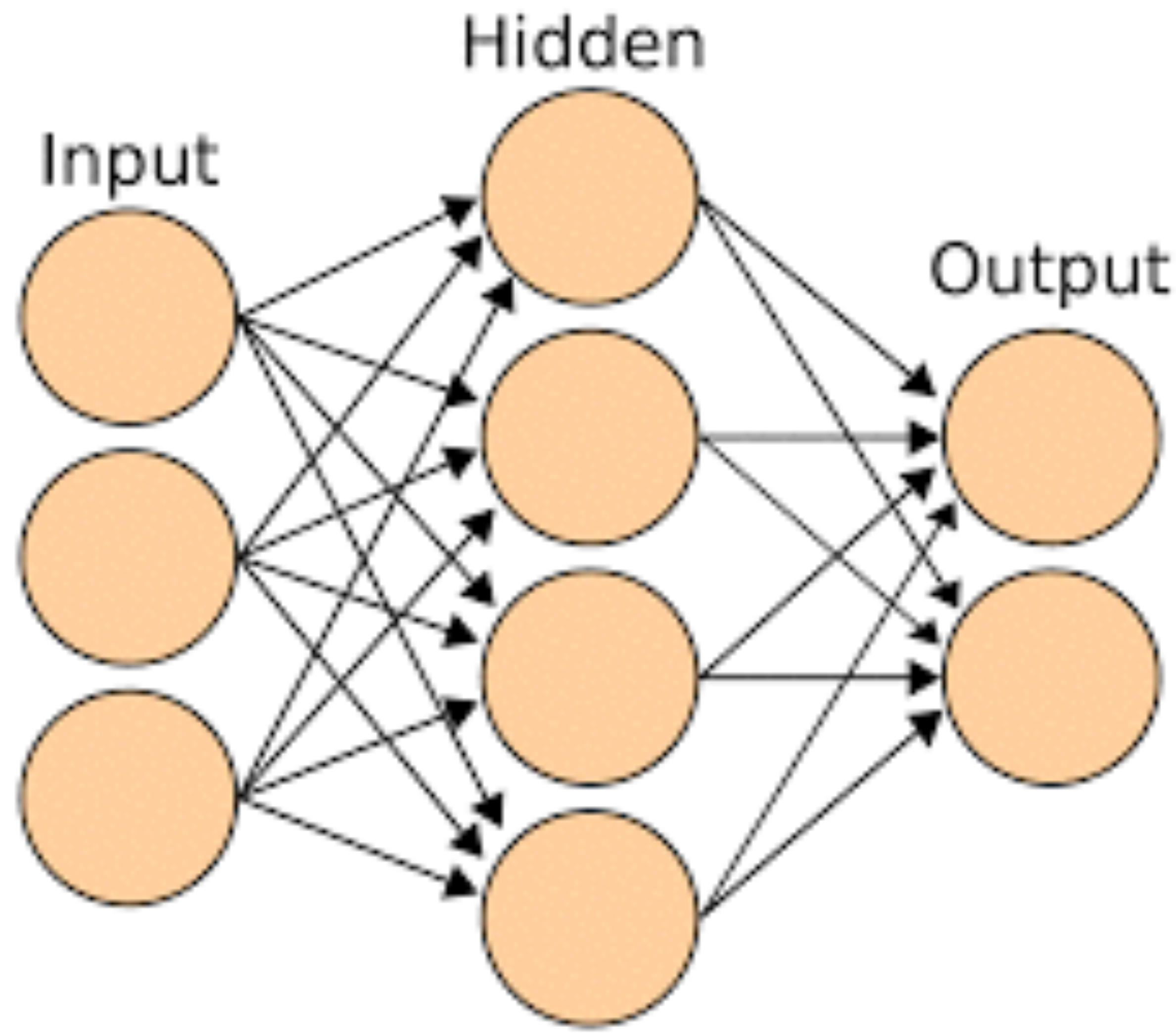
Why are they so much better, though?

- “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 (like MF, more in a bit)
- Uniform representations across sub-disciplines of AI (i.e. vision, language, sensor inputs)—“its all just vectors anyway”

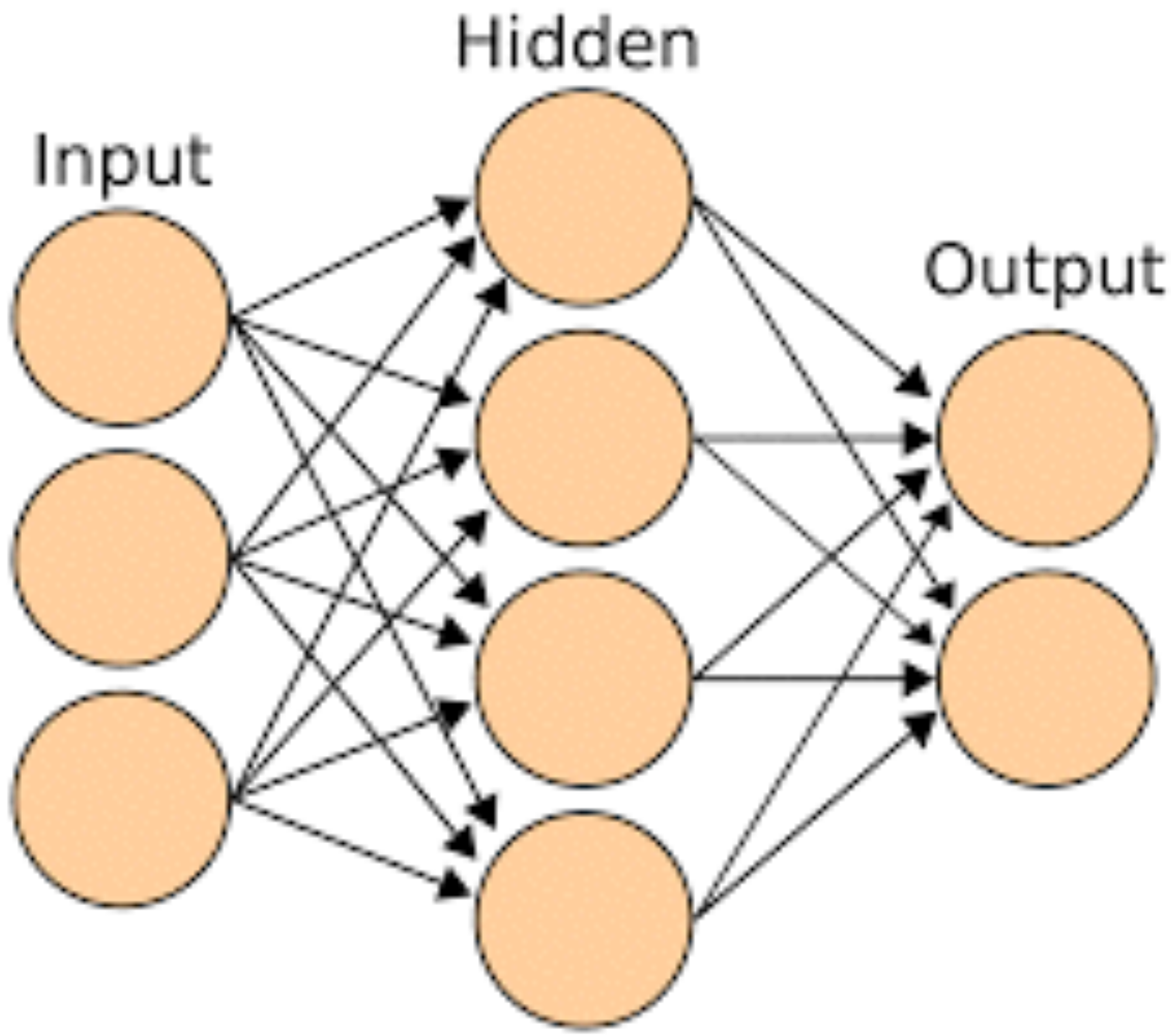
NNs as classifiers

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

Multilayer Perceptron



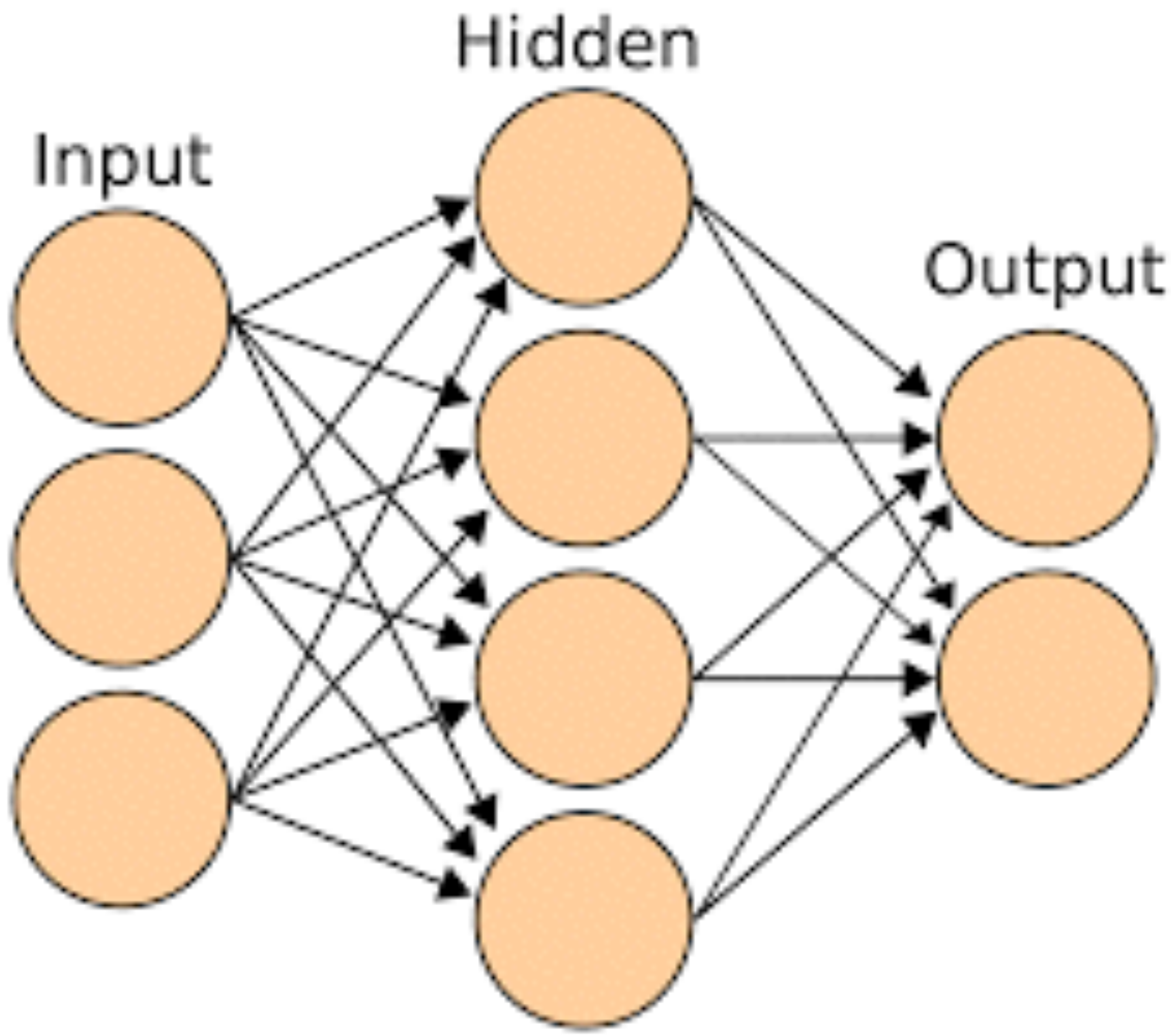
Multilayer Perceptron



“Feed Forward Net”

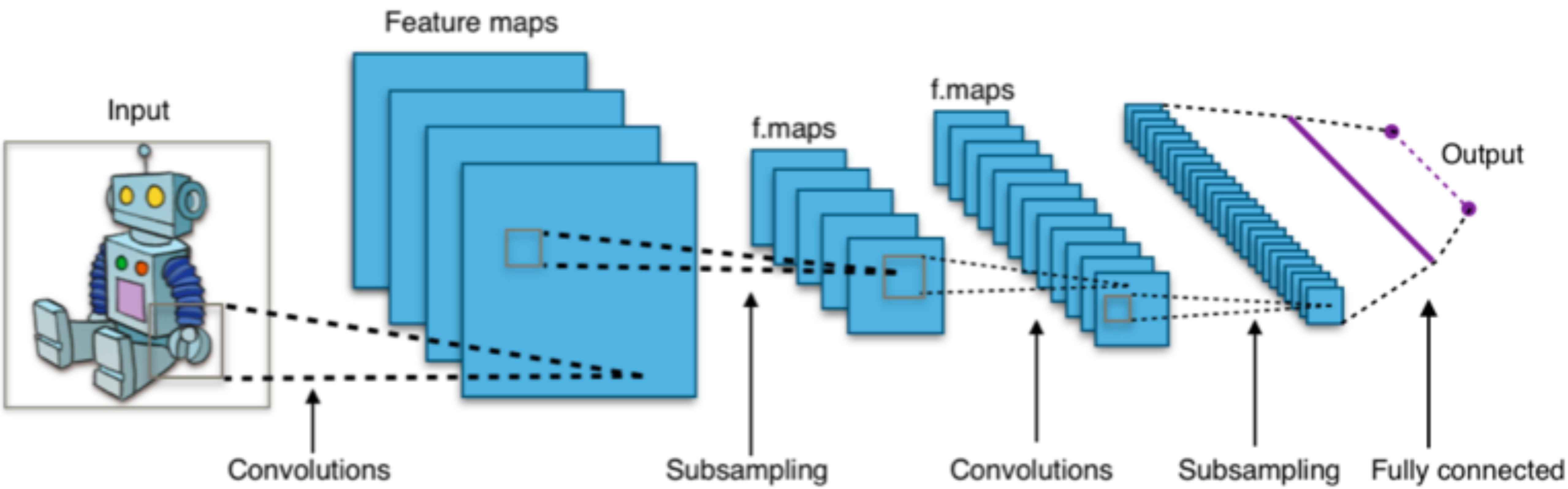
“Fully Connected Layer”

Multilayer Perceptron

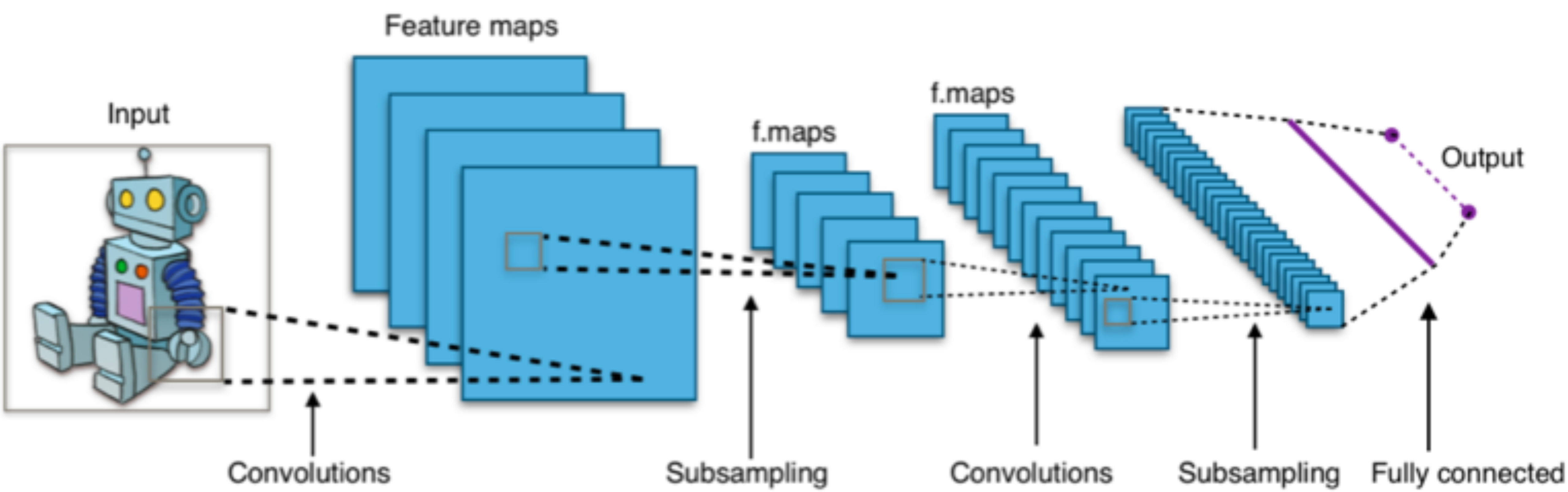


Arbitrary, non-linear combinations of input features.
No prior on the structure of those features.

Convolutional Neural Net (CNN)

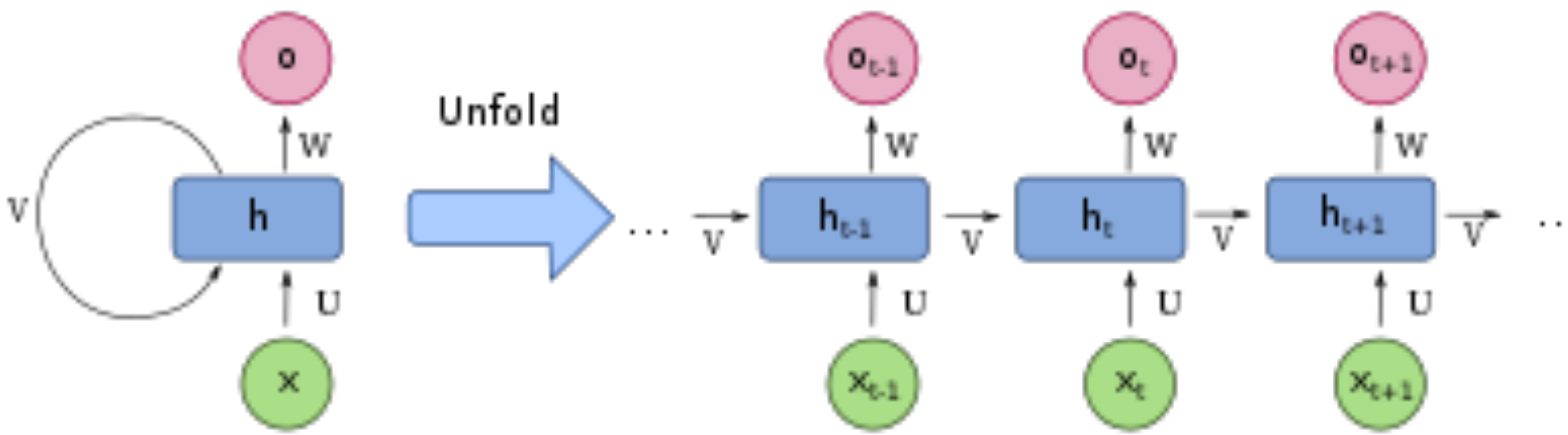


Convolutional Neural Net (CNN)

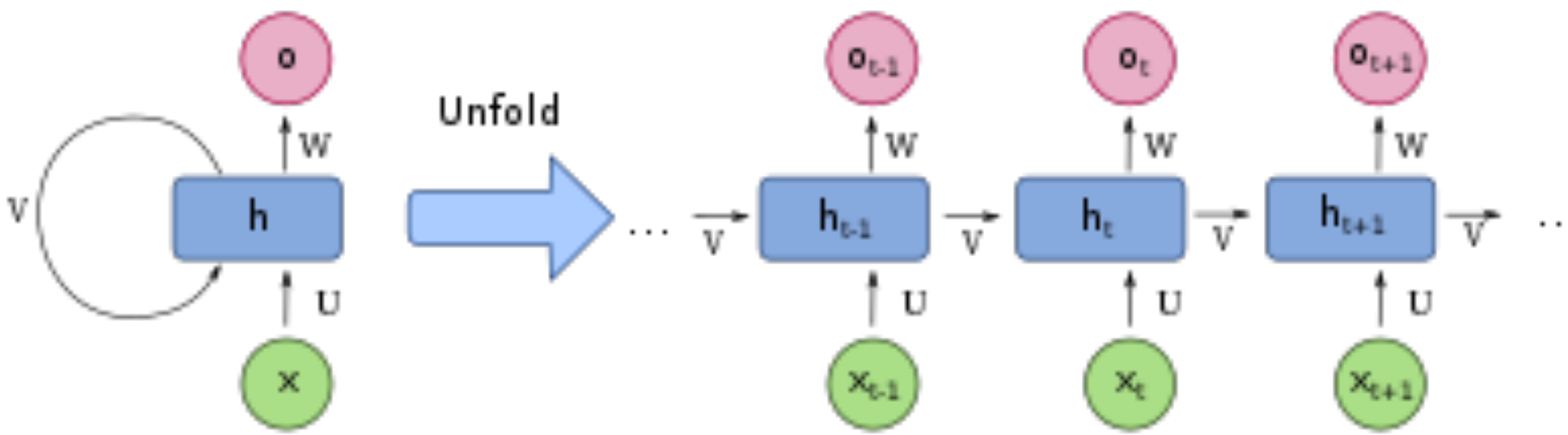


Used for vision. Assumes spatial structure to the data.

Recurrent Neural Net (RNN)



Recurrent Neural Net (RNN)



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

Transfer Learning

Transfer Learning

- Train a model to do some task T1 (for which you have a lot of data)

Transfer 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

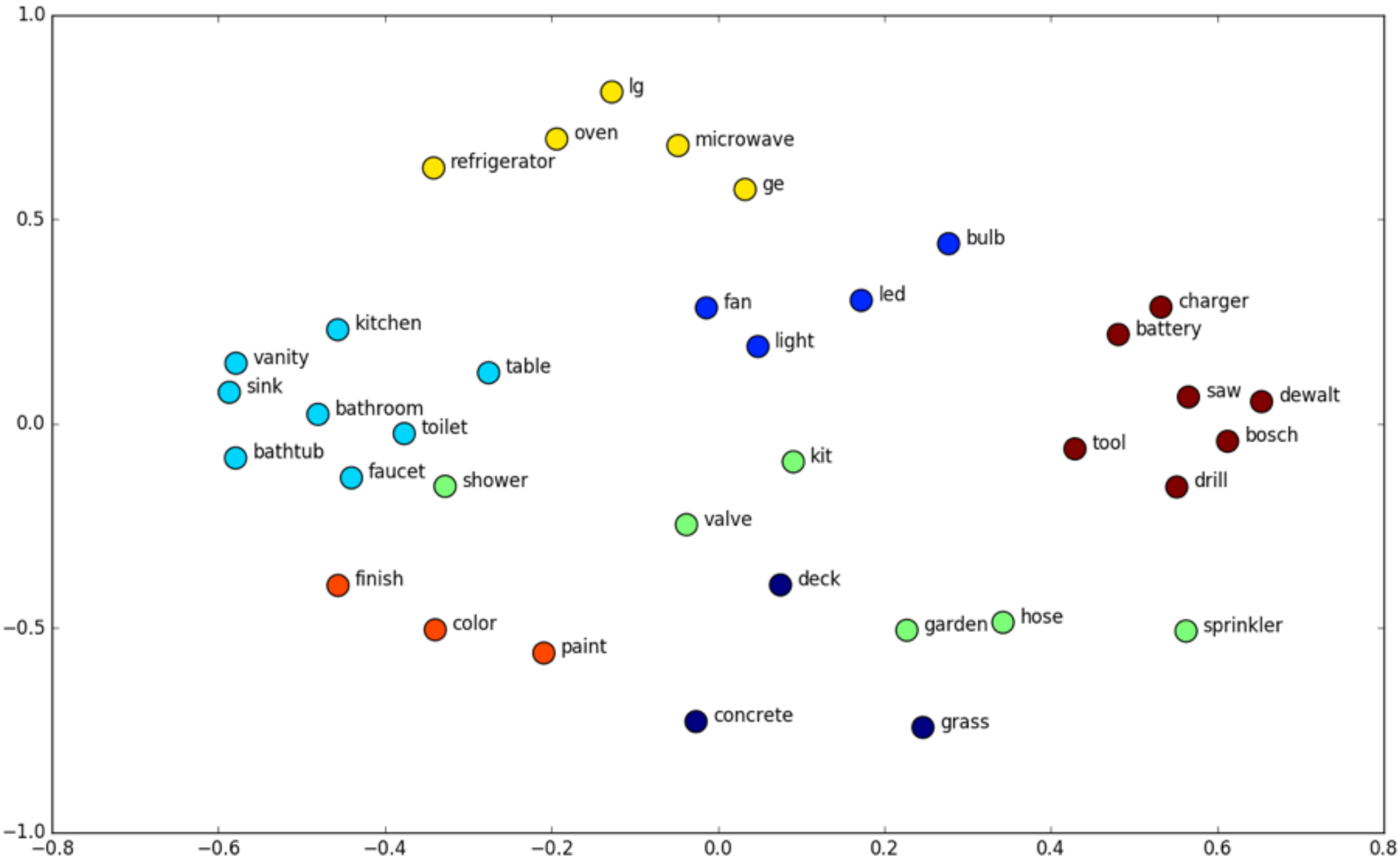
Transfer Learning

- 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?

"Pretraining"

- 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



Word Embeddings

Factorization of the term-context matrix

	the	congress	parliament	US	UK
the	1	1	1	1	1
congress	1	1	0	1	0
parlaiment	1	0	1	1	1
US	1	1	1	1	0
UK	1	0	1	0	1

Word Embeddings

	the	con-gress	parlia-ment	US	UK
the	1	1	1	1	1
congress	1	1	0	1	0
parlaiment	1	0	1	1	1
US	1	1	1	1	0
UK	1	0	1	0	1

Embeddings!

=

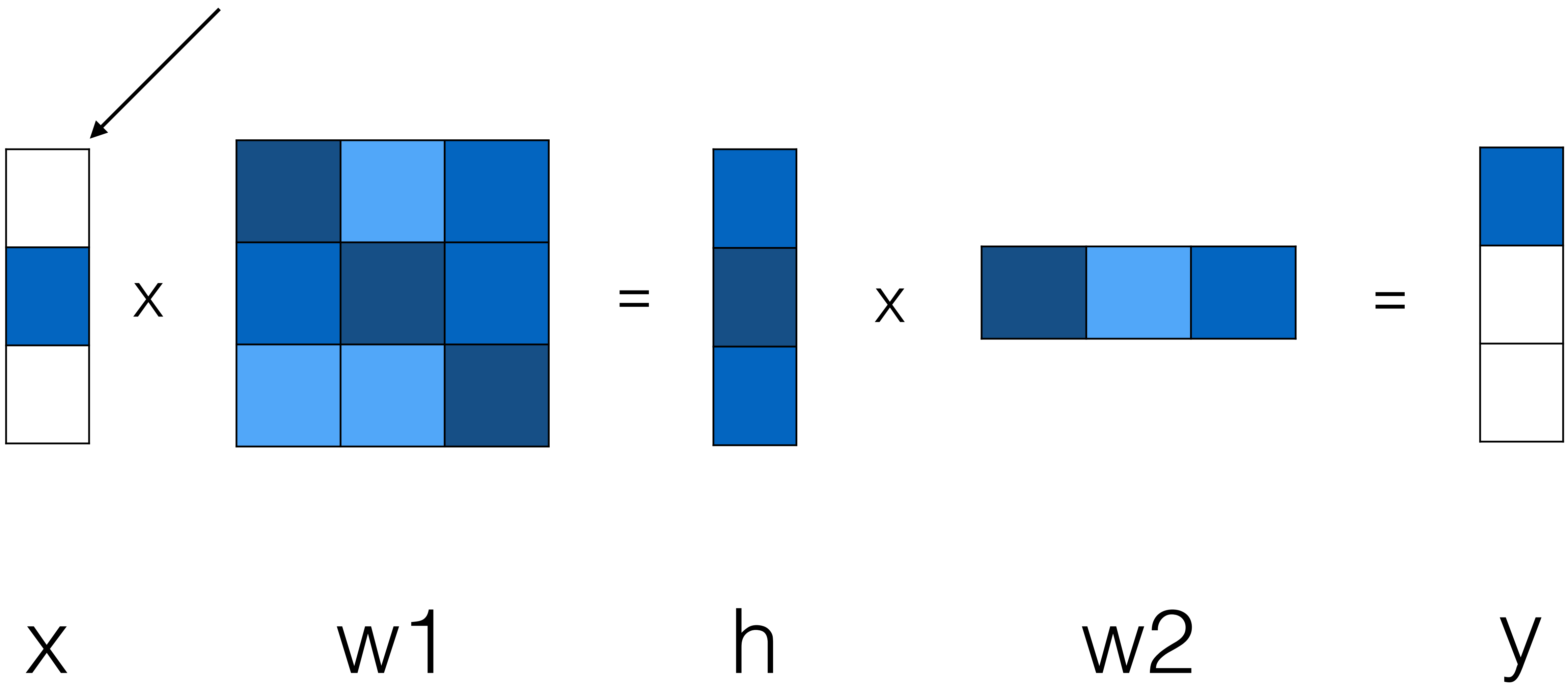
the	-0.60	-0.39	0.70	0.00
congress	-0.48	0.50	-0.12	-0.71
parliament	-0.43	-0.58	-0.69	0.00
US	-0.48	0.50	-0.12	0.71
UK	0.02	0.79	0.02	-0.44

×

-0.65	-0.34	-0.51	-0.34	-0.31
0.02	-0.54	0.34	-0.54	0.56
-0.42	0.02	0.79	0.02	-0.44
-0.63	0.27	0.00	0.37	0.63
-0.04	0.73	0.00	-0.68	0.04

Word Embeddings

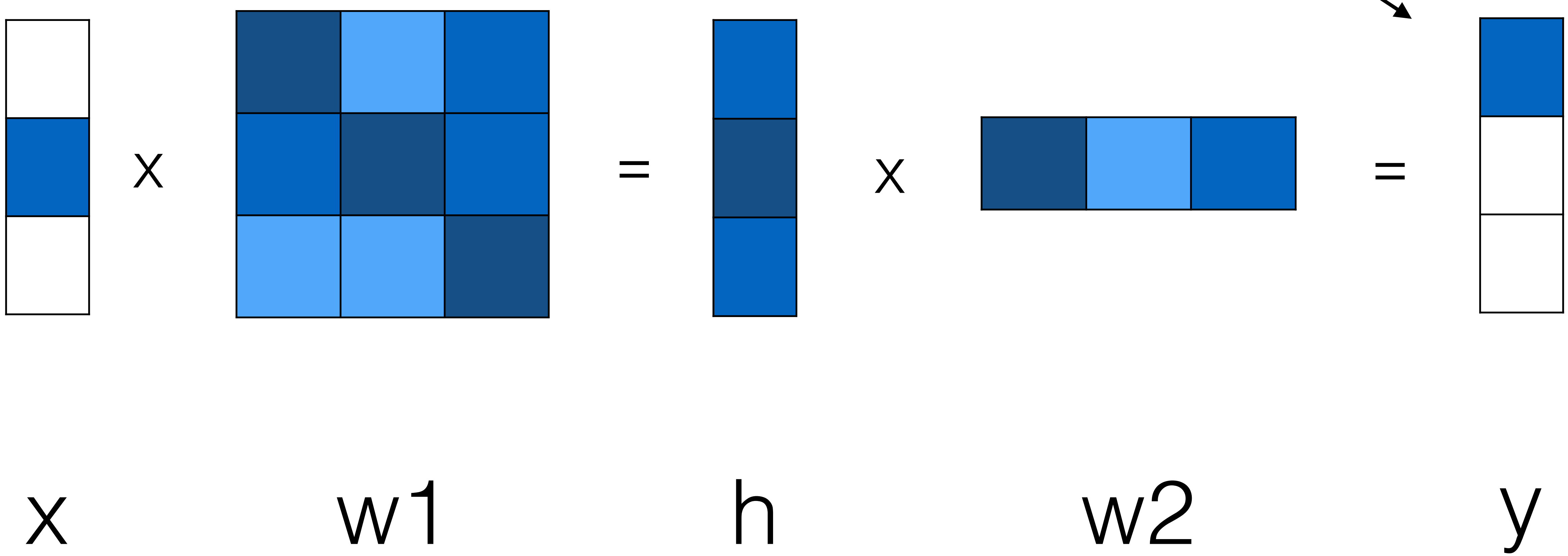
one hot encoding, i.e.: the word "congress"



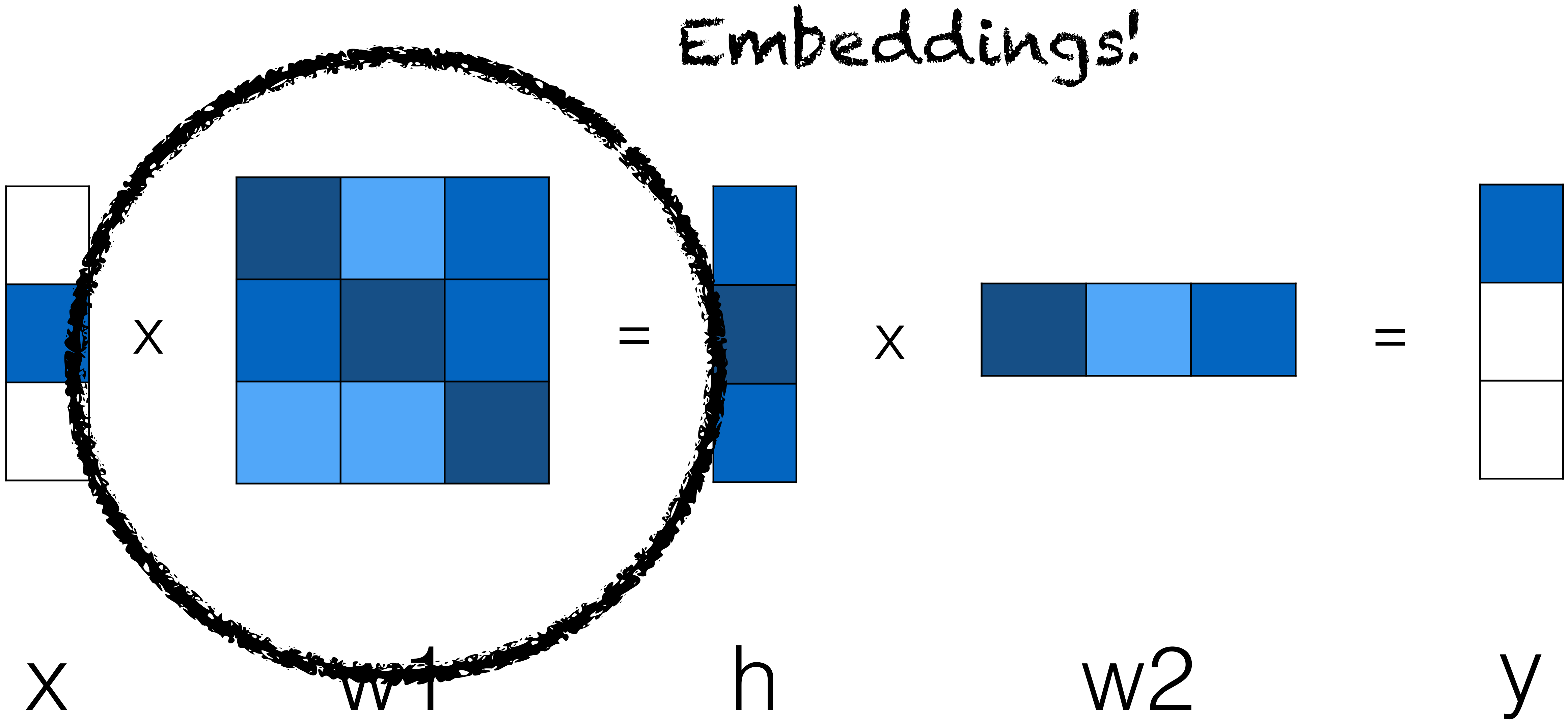
Word Embeddings

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

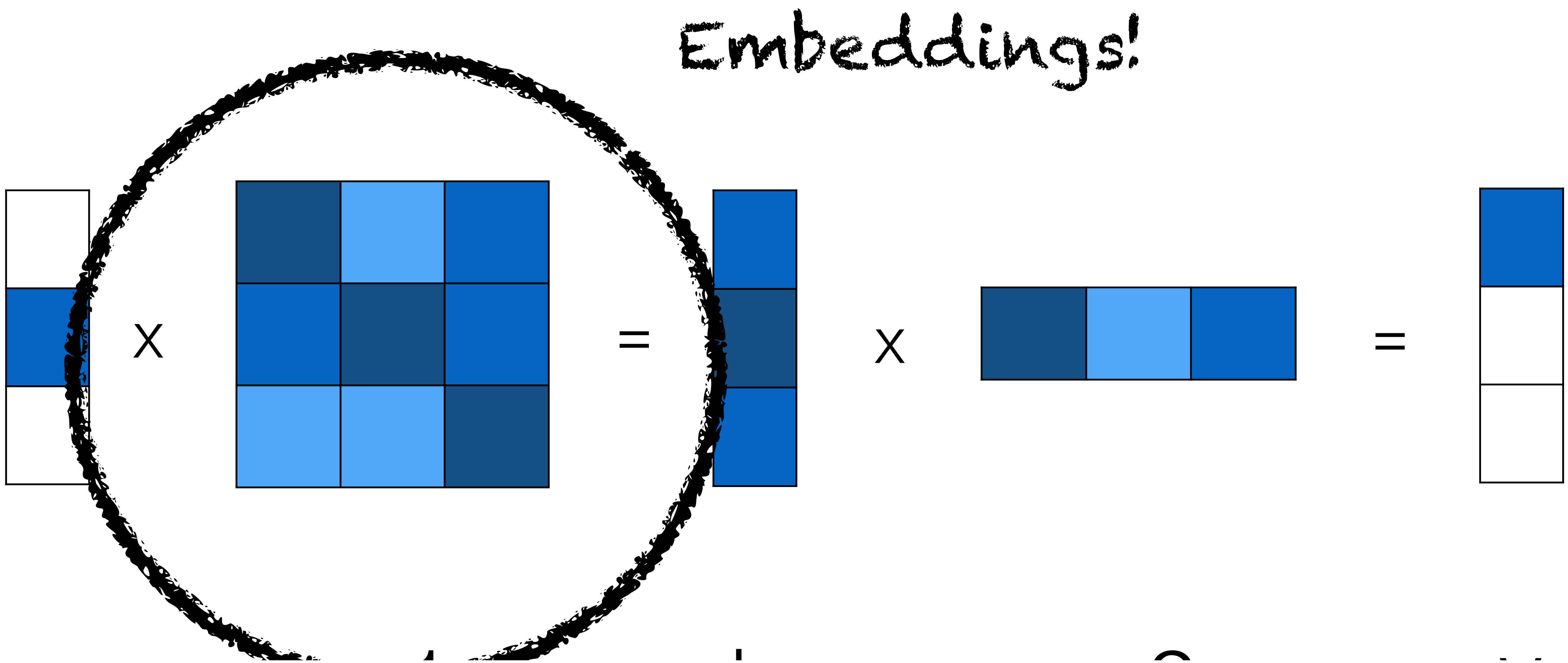
the word "stagnated"



Word Embeddings

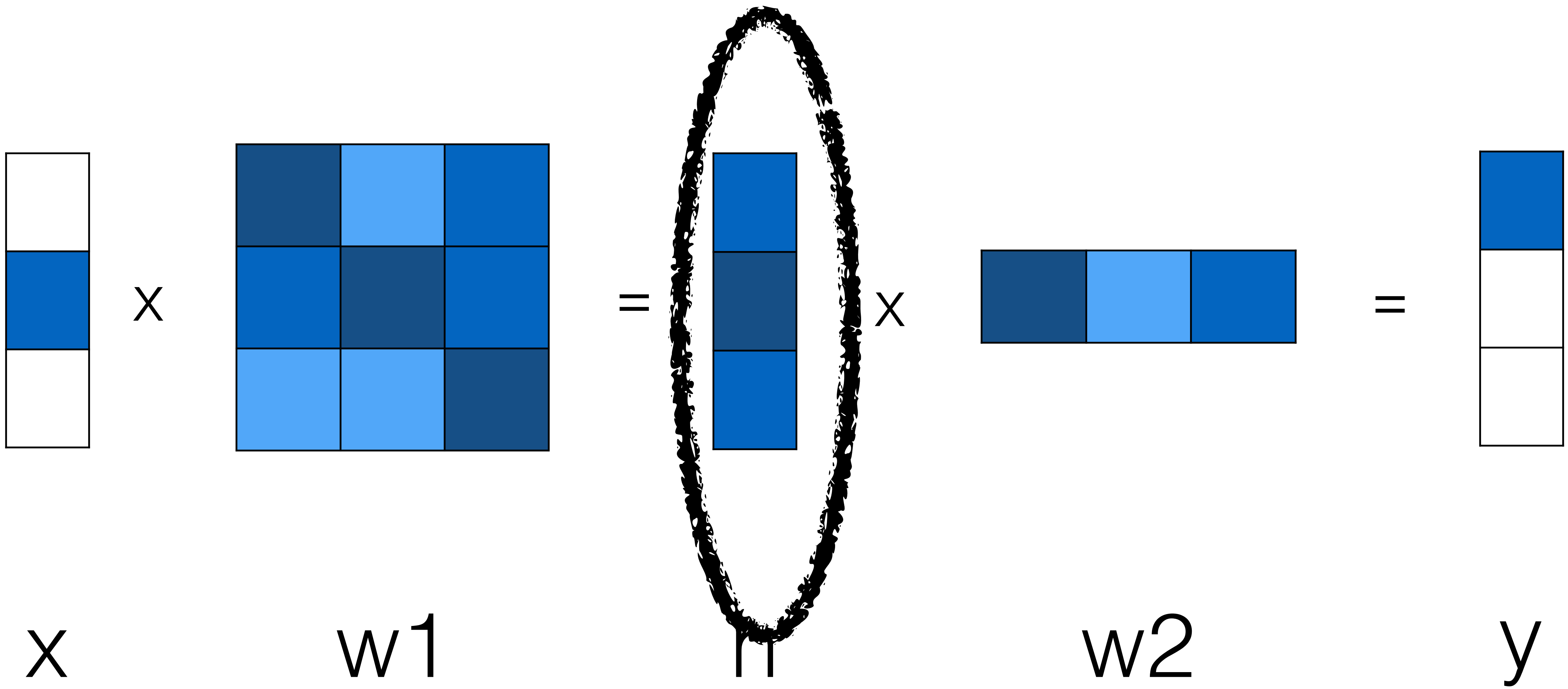


Word Embeddings



Actually the same as the MF embeddings
(assuming a linear network...)
Levy and Goldberg (2014)

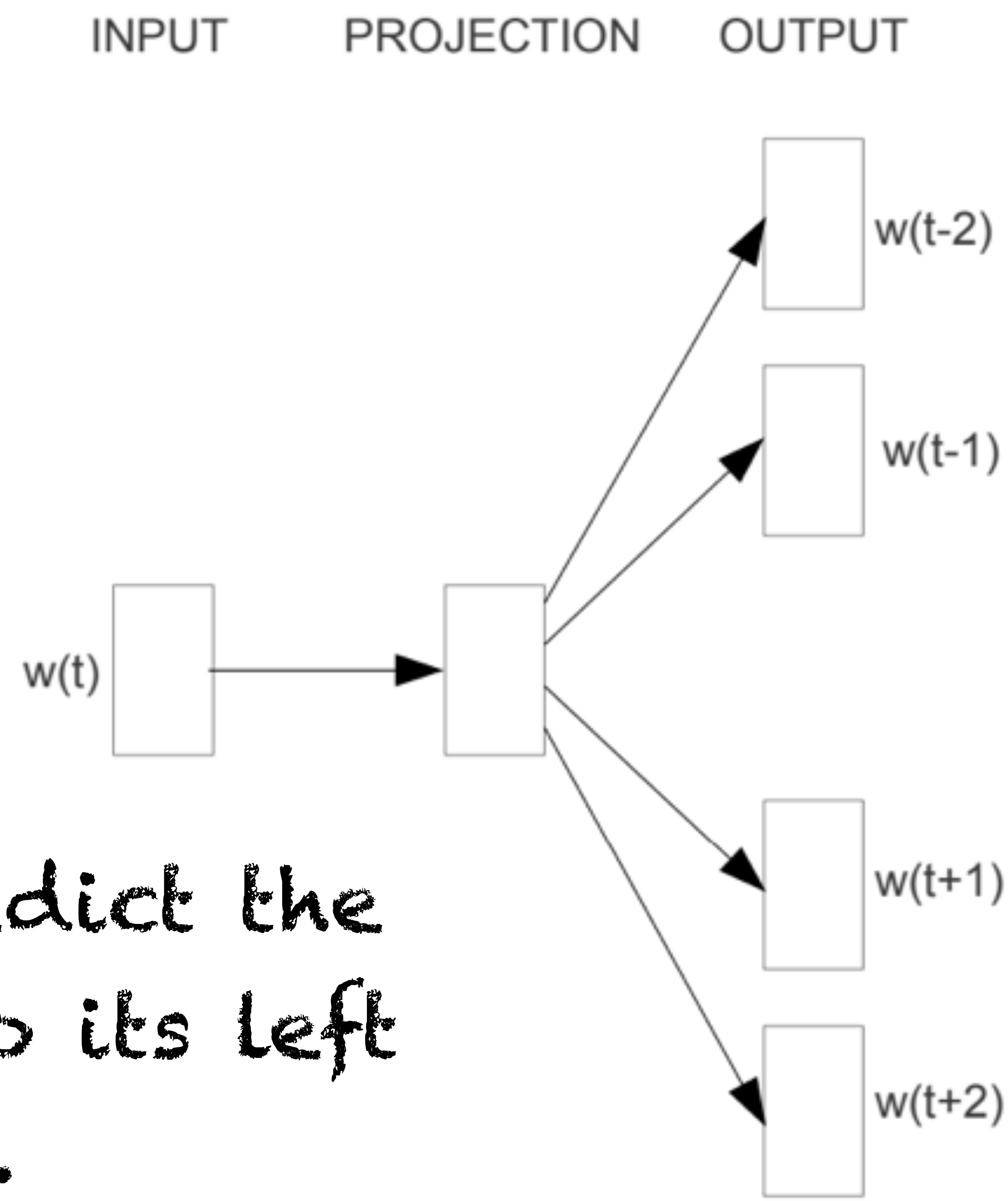
Word Embeddings



Allows parameter sharing over similar words.

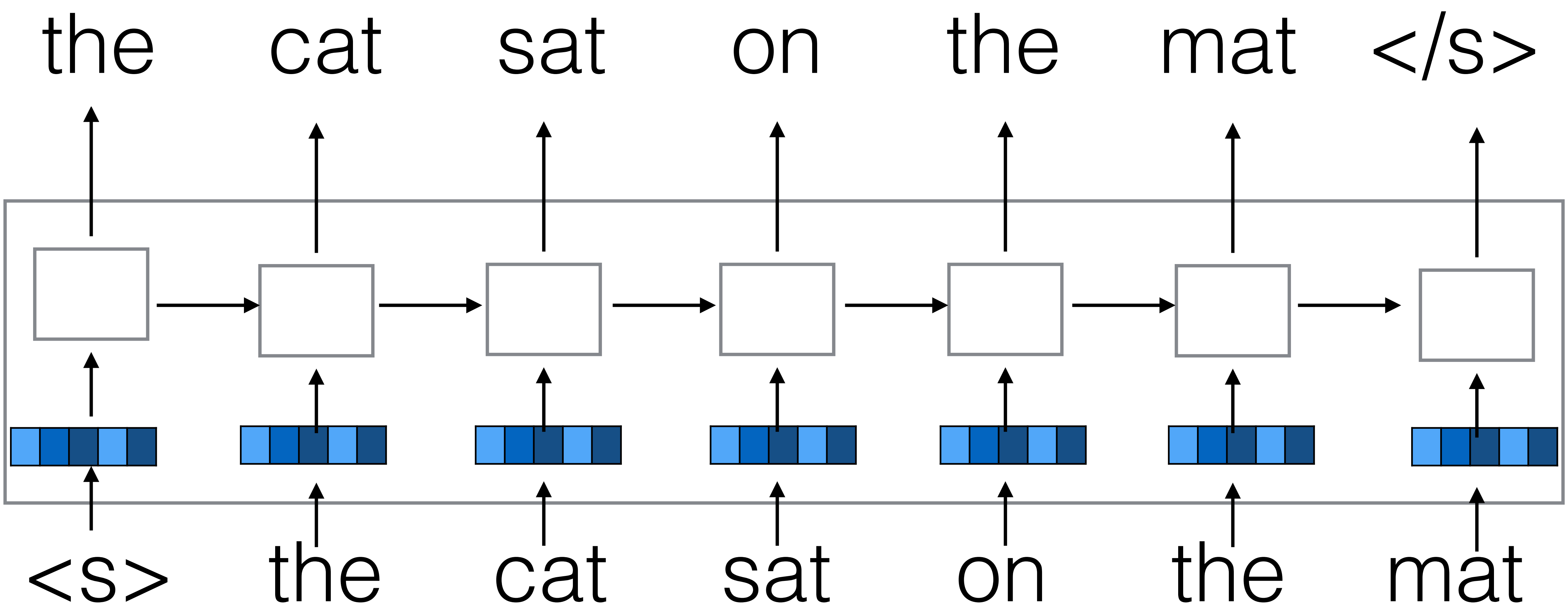
Word Embeddings

For your homework: Skipgram

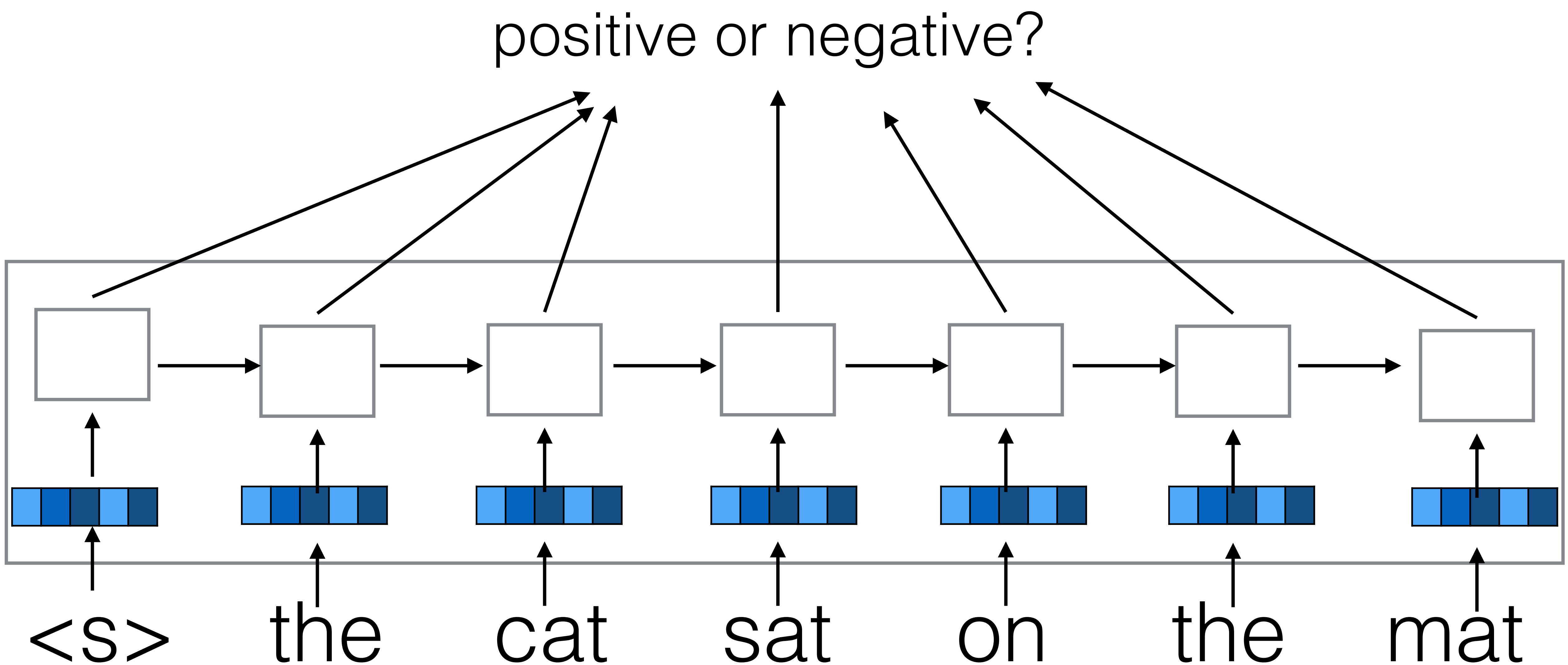


Given a word, predict the words that come to its left and right...

Word Embeddings



Word Embeddings



So why haven’t NNs solved everything?

So why haven’t NNs solved everything?

- Mostly require (massive!) supervised learning. Better use of deep RL/unsupervised pretraining?

So why haven't NNs solved everything?

- Mostly require (massive!) supervised learning. Better use of deep RL/unsupervised pretraining?

- Feature engineering has been replaced with architecture engineering/hyperparameter hacking. Metalearning?

So why haven't NNs solved everything?

- Mostly require (massive!) supervised learning. Better use of deep RL/unsupervised pretraining?

- Feature engineering has been replaced with architecture engineering/hyperparameter hacking. Metalearning?

- End-to-end-training hurts generalizability. Inductive biases on the hypothesis space?

So why haven't NNs solved everything?

- Mostly require (massive!) supervised learning. Better use of deep RL/unsupervised pretraining?
- Feature engineering has been replaced with architecture engineering/hyperparameter hacking. Metalearning?
- 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

I have a thing. Should I deep learn it?

I have a thing. Should I deep learn it?

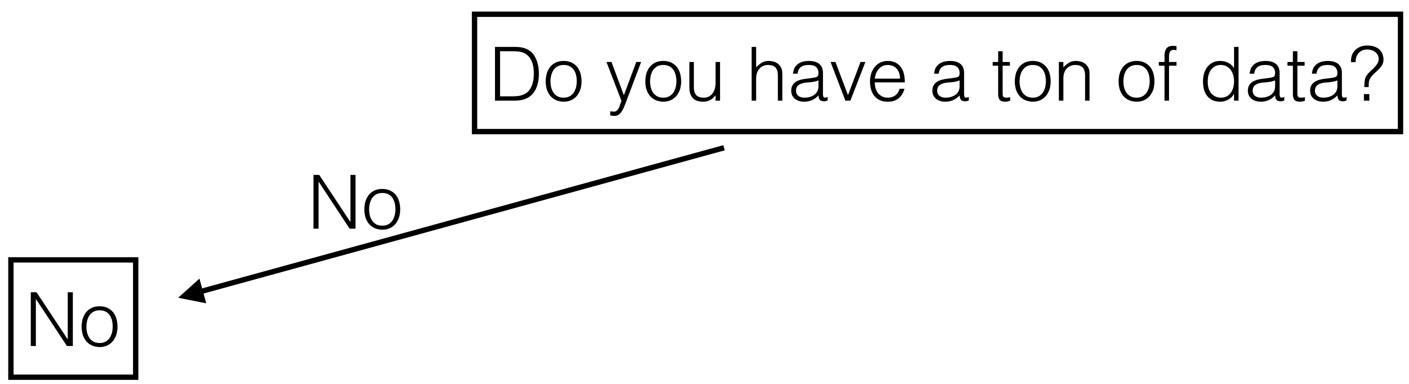
$\boxed{\text{Do you have a ton of data?}}$

I have a thing. Should I deep learn it?

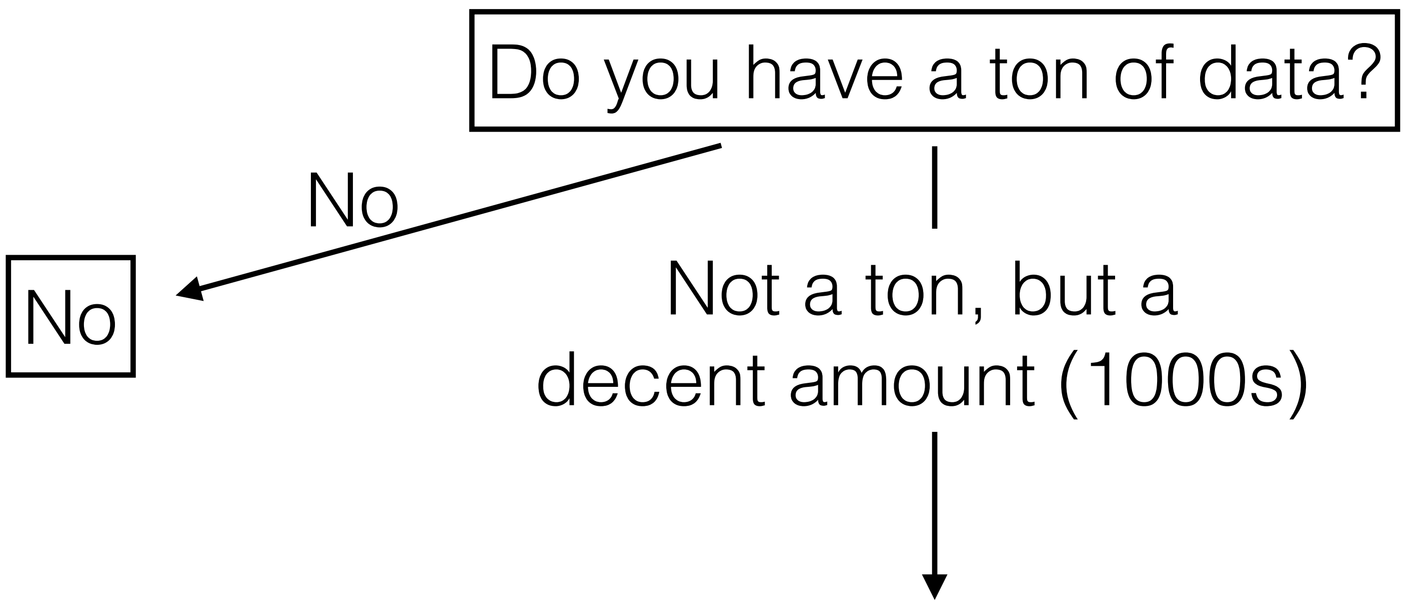
Do you have a ton of data?

No

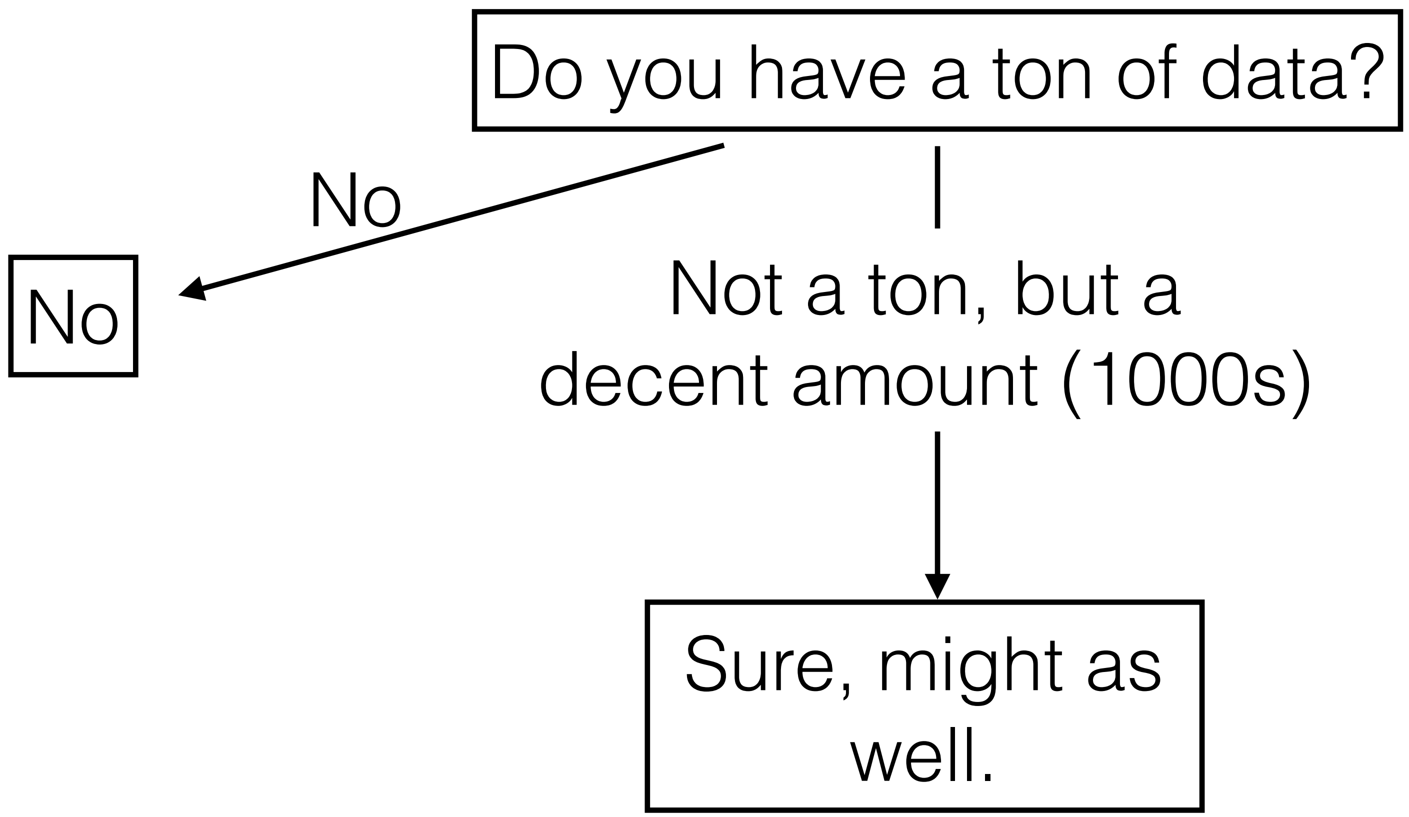
I have a thing. Should I deep learn it?



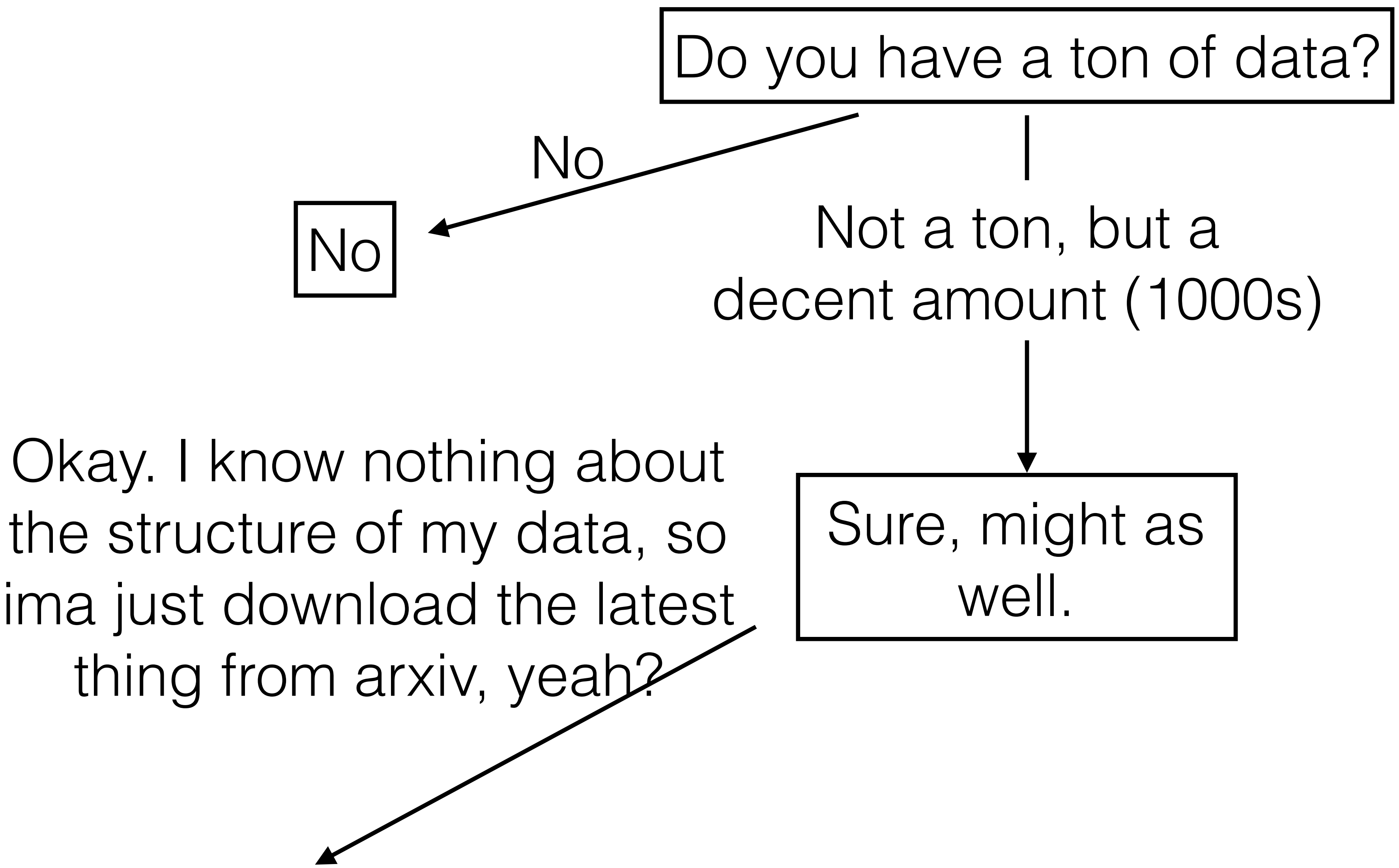
I have a thing. Should I deep learn it?



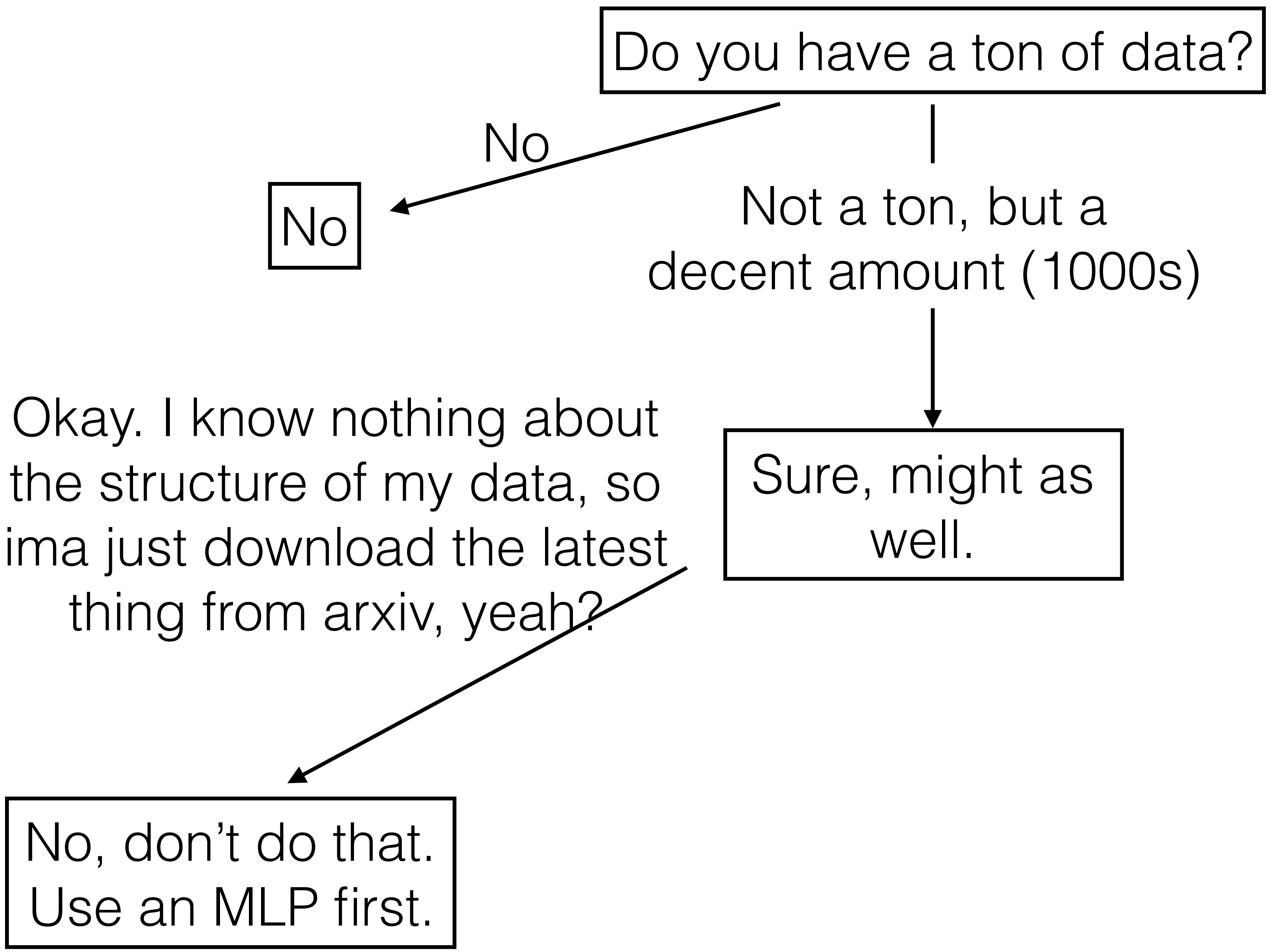
I have a thing. Should I deep learn it?



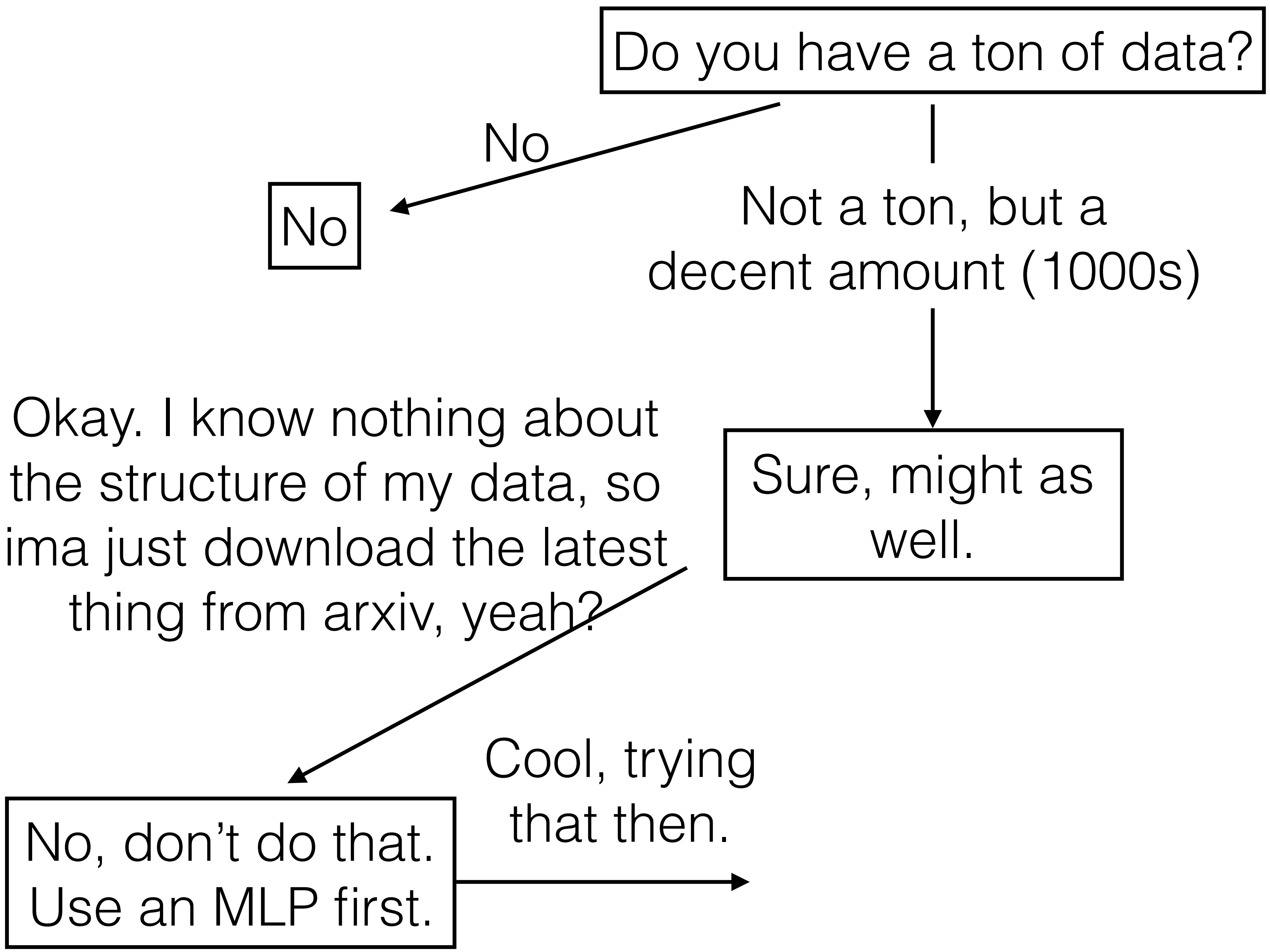
I have a thing. Should I deep learn it?



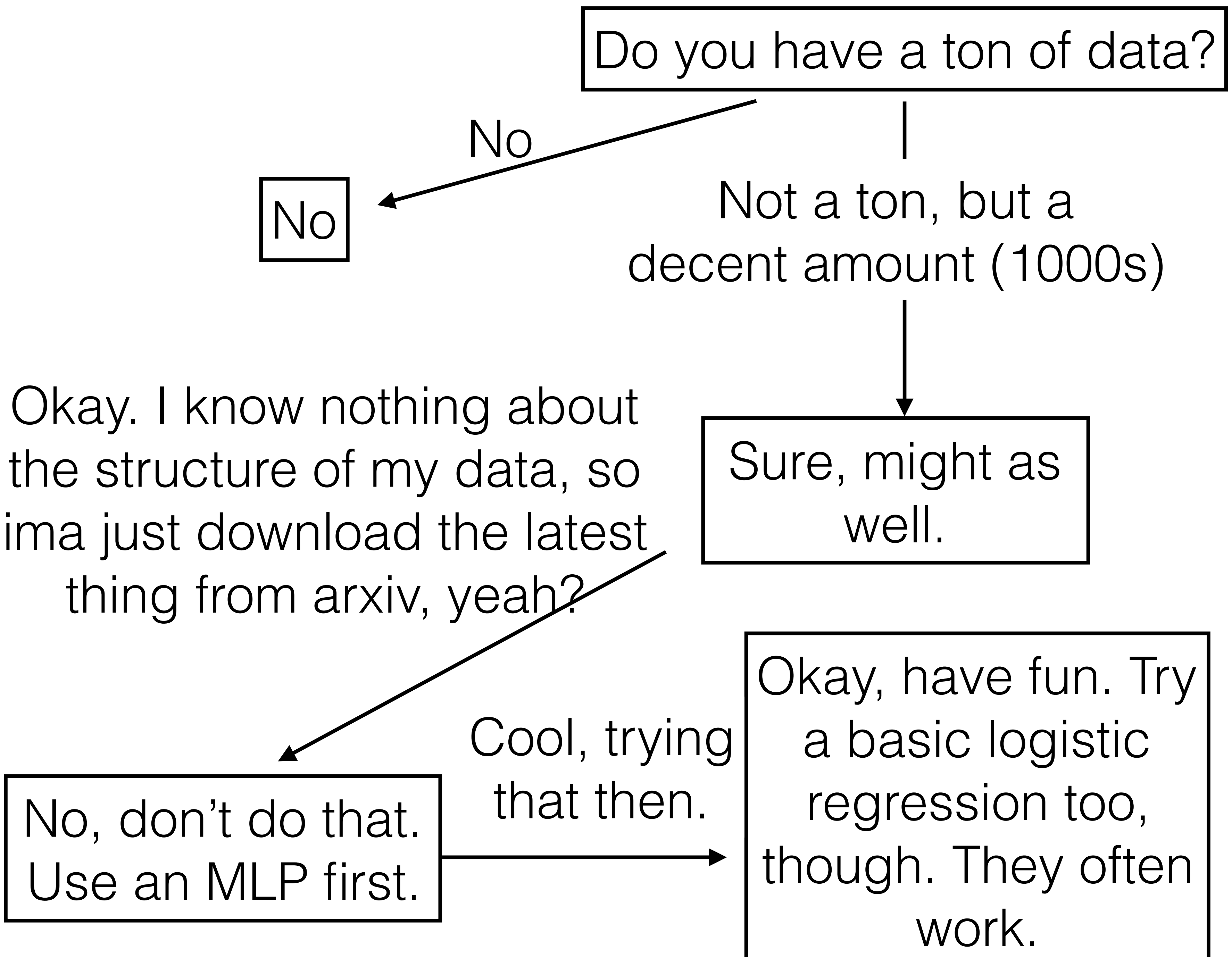
I have a thing. Should I deep learn it?



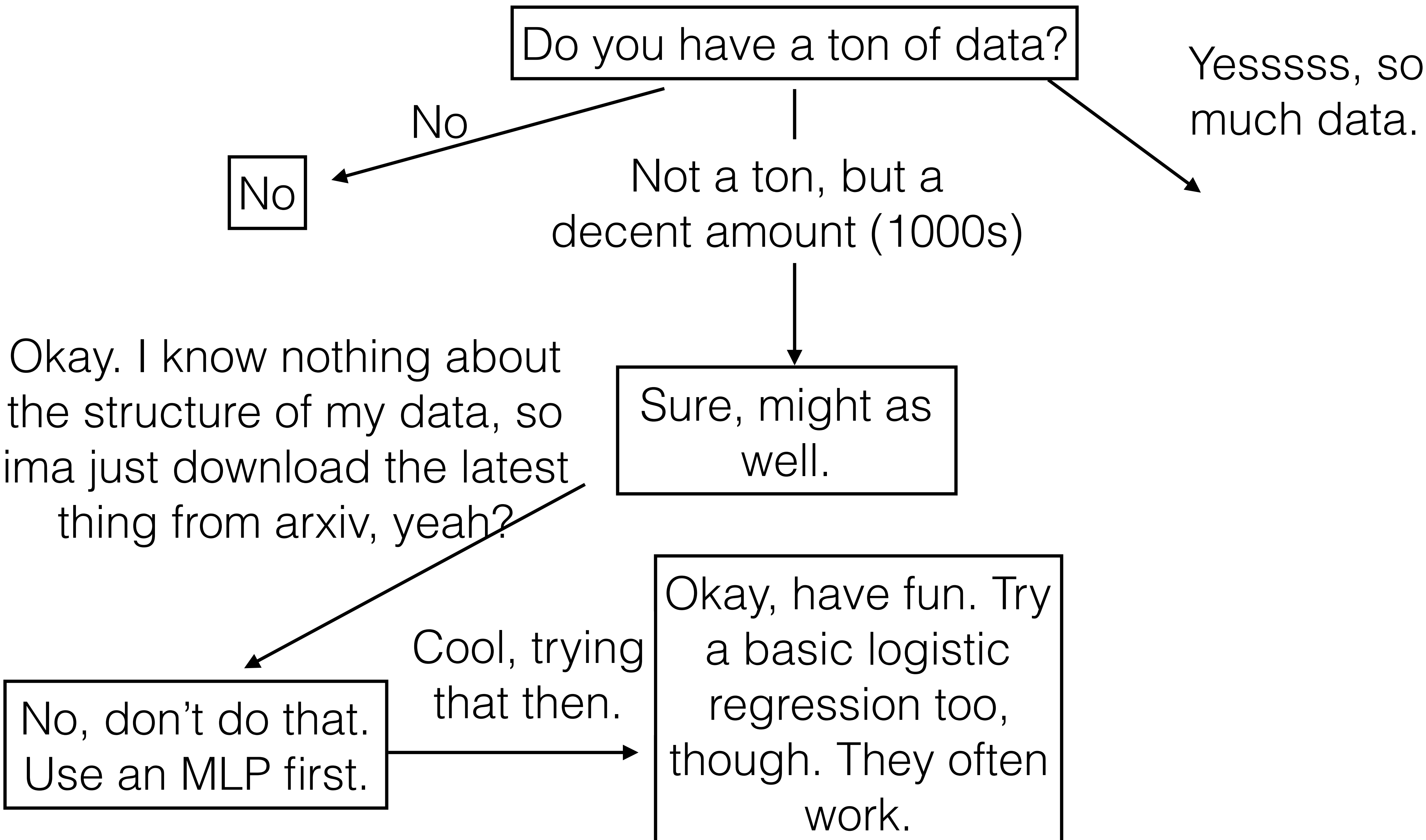
I have a thing. Should I deep learn it?



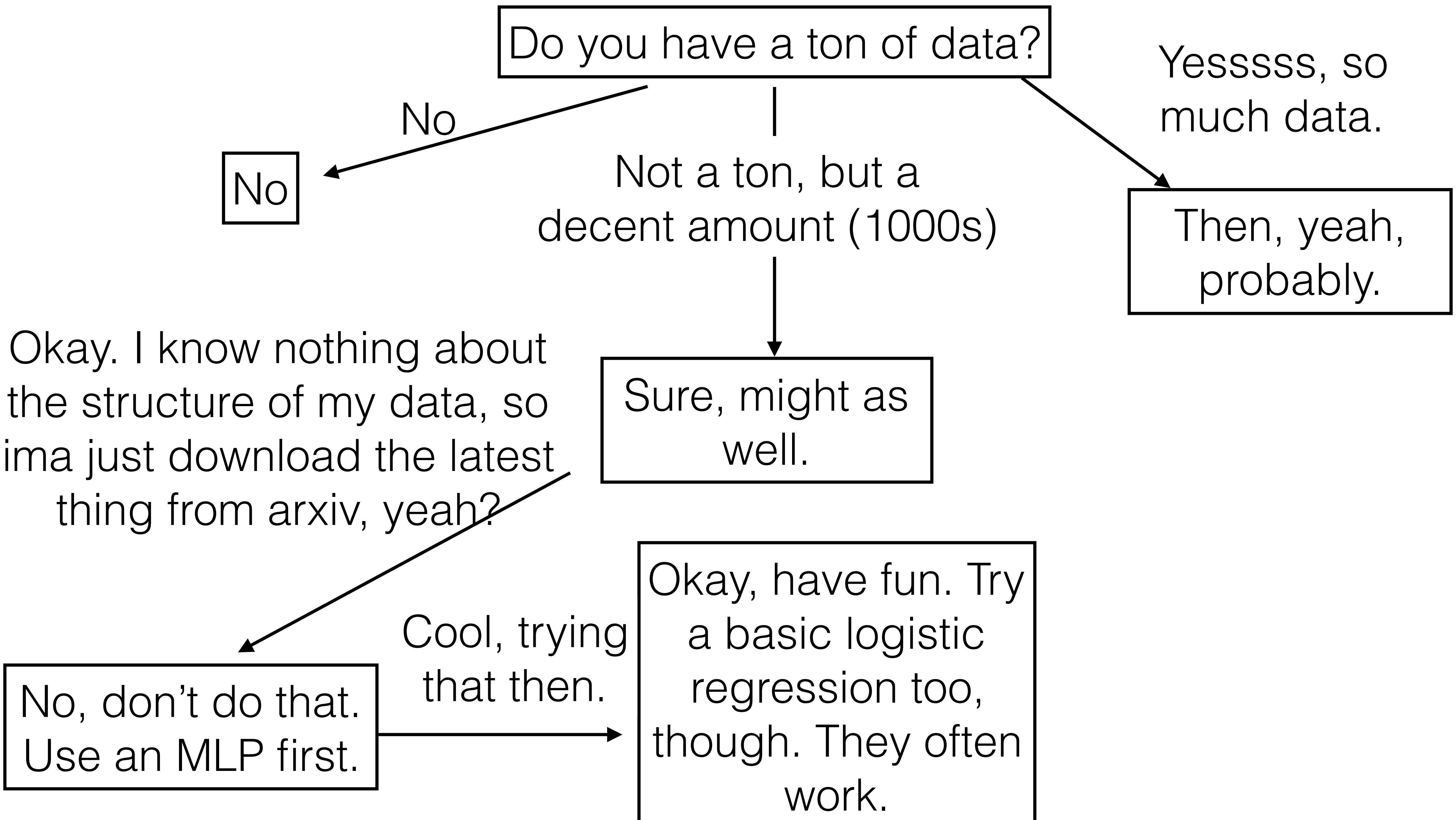
I have a thing. Should I deep learn it?



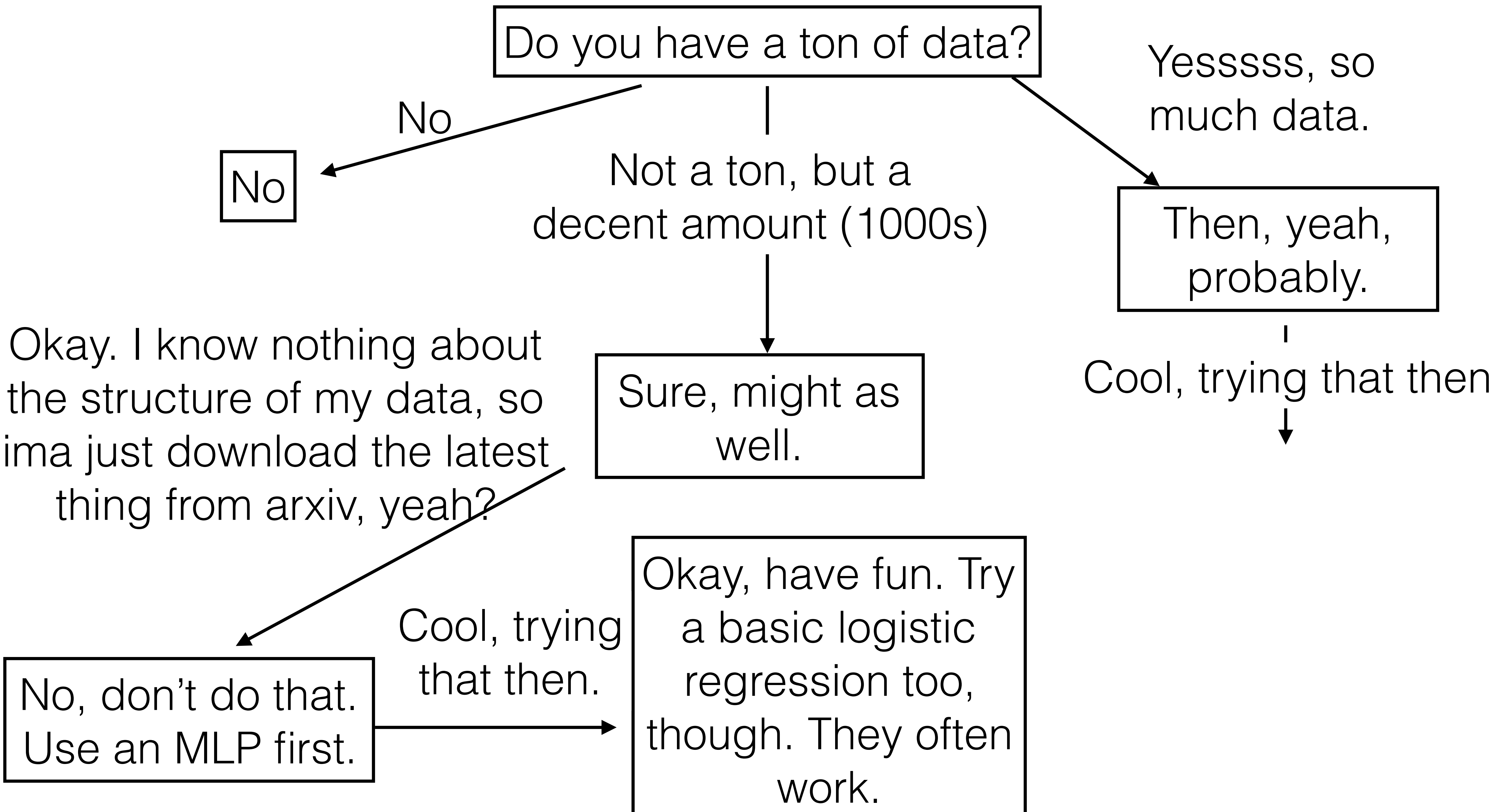
I have a thing. Should I deep learn it?



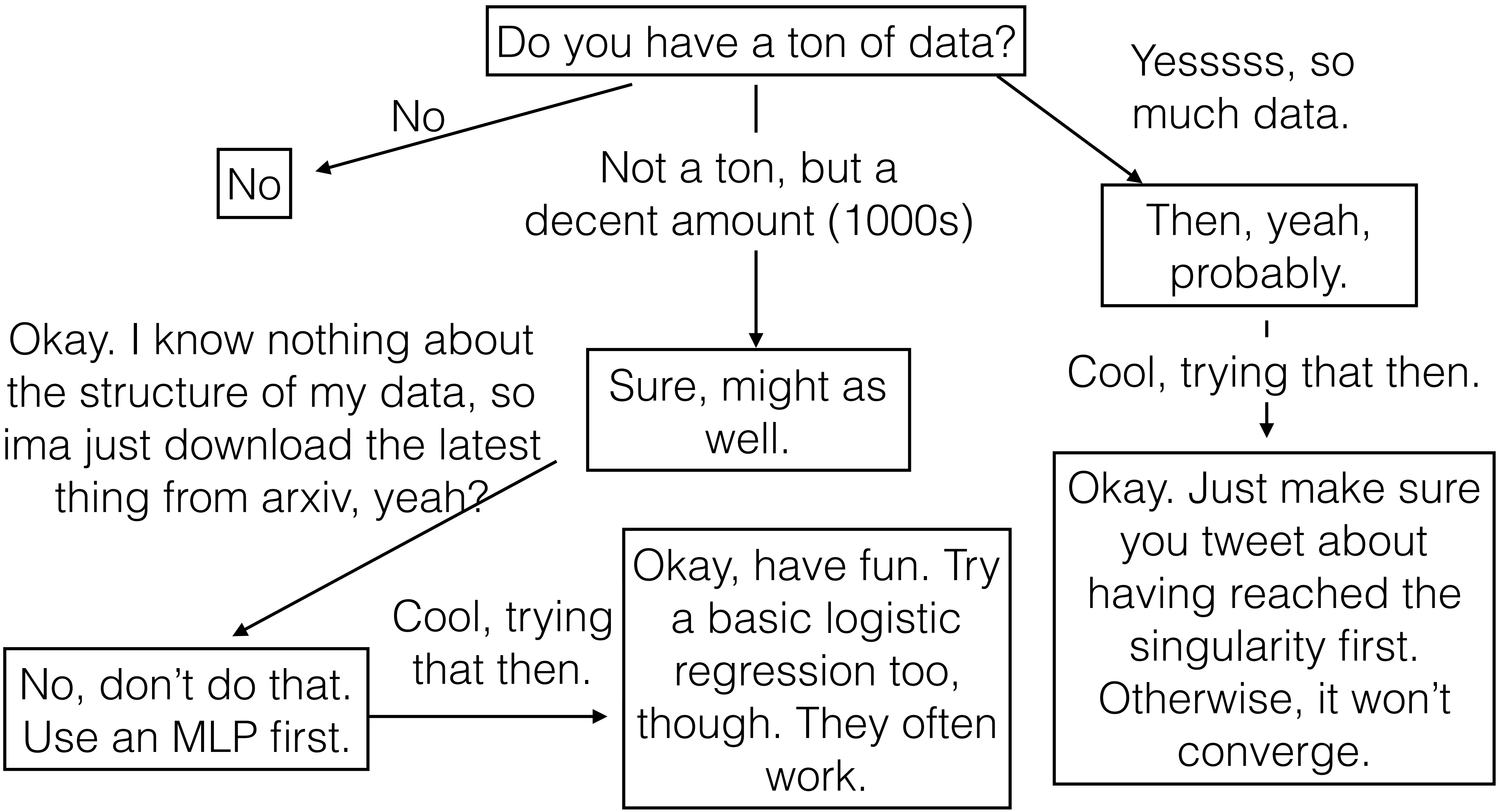
I have a thing. Should I deep learn it?



I have a thing. Should I deep learn it?



I have a thing. Should I deep learn it?



bye :)