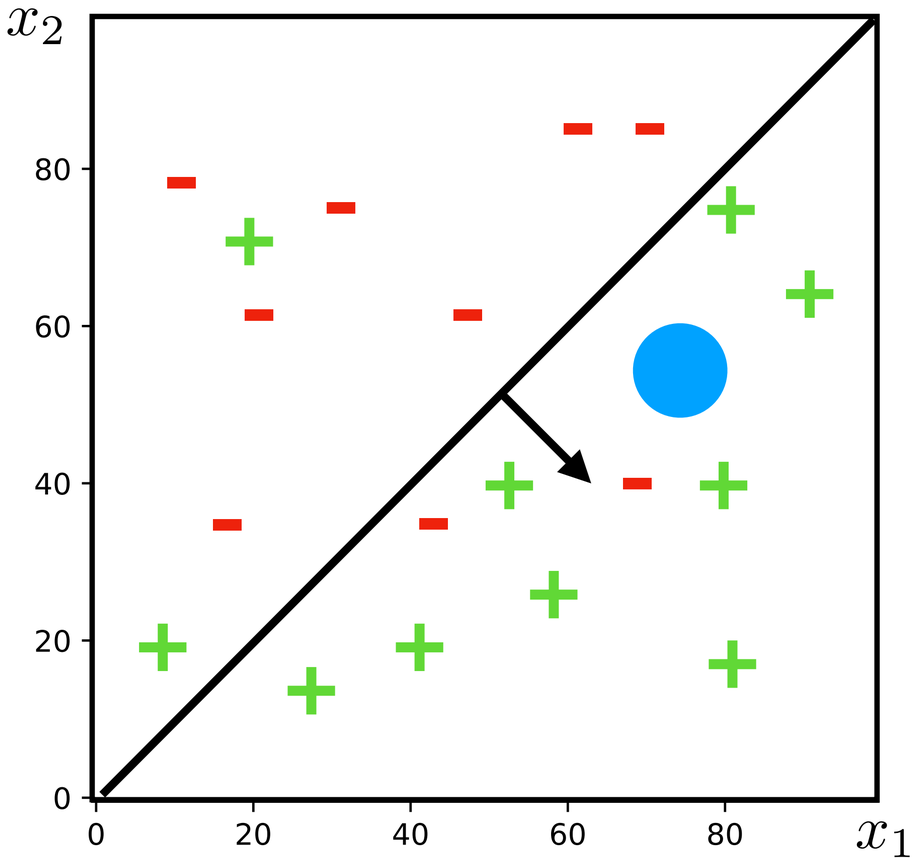
EECS 504: Neural networks

Recall: linear classifiers



"What label is this point?"

Which side of the hyperplane is **x** on?

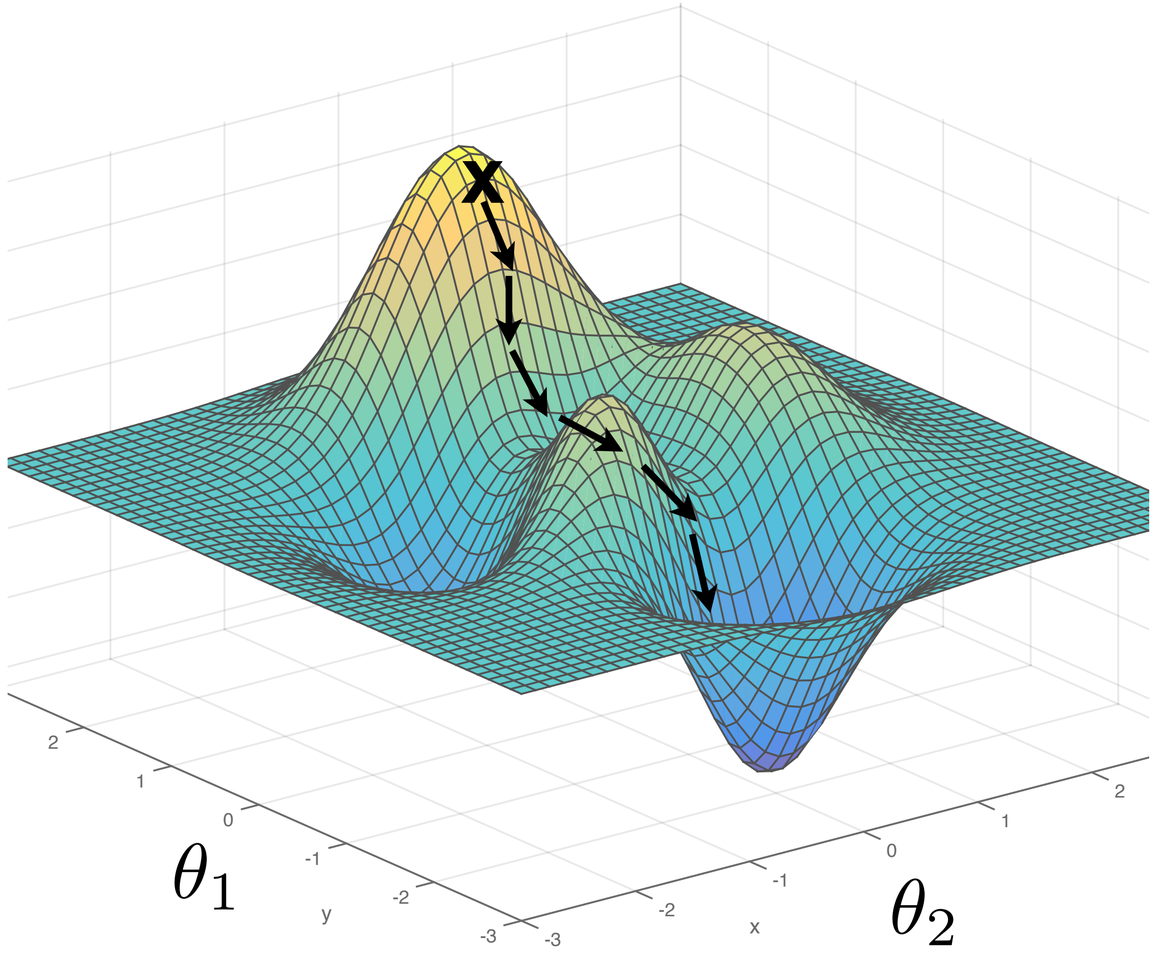
$$\hat{y} = \mathbf{x}^T \mathbf{w} + b$$

$$g(\hat{y}) = \begin{cases} 1, & \text{if} \quad \hat{y} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Example adapted from P. Isola

Gradient descent

$J(\theta)$



$$\theta^* = \arg\min_{\theta} J(\theta)$$

Source: Isola, Torralba, Freeman

Gradient descent

$$\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta) \Big|_{\theta=\theta^t}$$

What's this again?

1. Gradient of the loss w.r.t. the classifier's parameters
2. Direction of steepest descent
3. "Local" linear approximation to the function

Estimating the gradient

Idea #1: finite differences $\frac{f(x+\epsilon)-f(x)}{\epsilon}$

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J}{\theta_1} \\ \frac{\partial J}{\theta_2} \\ \cdots \\ \frac{\partial J}{\theta_d} \end{bmatrix}$$

- Easy to compute but **slow**.
- Inaccurate in some cases, unless careful
- Useful for **checking** other methods

Estimating the gradient

Idea #2: compute it analytically using calculus.

Simple example: binary labels, squared loss, and regularization on θ

$$J(\theta) = \lambda||\theta||^2 + \frac{1}{N}\sum_{i=1}^{N}(y_i - \theta^\top x_i)^2$$

$$\nabla_\theta$$

$$\nabla_\theta J(\theta) = 2\lambda\theta - \frac{1}{N}\sum_{i=1}^{N} 2(y_i - \theta^\top x_i)x_i$$

Adapted from D. Fouhey

Analyzing the gradient descent update

Recall update rule: $\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$

How does this change θ ?

"Decay" toward 0 Scalar α

$$-\nabla_\theta J(\theta) = -2\lambda\theta - \frac{1}{N}\sum_{i=1}^{N} 2(y_i - \theta^\top x_i)x_i$$

Adapted from D. Fouhey

Analyzing the gradient descent update

What happens at each example?

Scalar α

$$-\nabla_\theta J(\theta) = -2\lambda\theta - \frac{1}{N}\sum_{i=1}^{N} 2(y_i - \theta^\top x_i)x_i$$

If $y > \theta^\top x$ (too *low*): then $\theta_{t+1} = \theta + \alpha x$ for some α

Dot product before: $\theta^\top x$

Dot product after: $(\theta + \alpha x)^\top x = \theta^\top x + \alpha x^\top x$

Adapted from D. Fouhey

Computational issues with gradient descent

Batch gradient descent

Loss function:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(x_i, y_i, \theta)$$

Its gradient is the **sum of gradients** for each example:

$$\nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i, y_i, \theta)$$

Requires iterating over every training example each gradient step!

Can we speed this up?

Stochastic gradient descent

This is just an **average**:

$$\nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i, y_i, \theta)$$

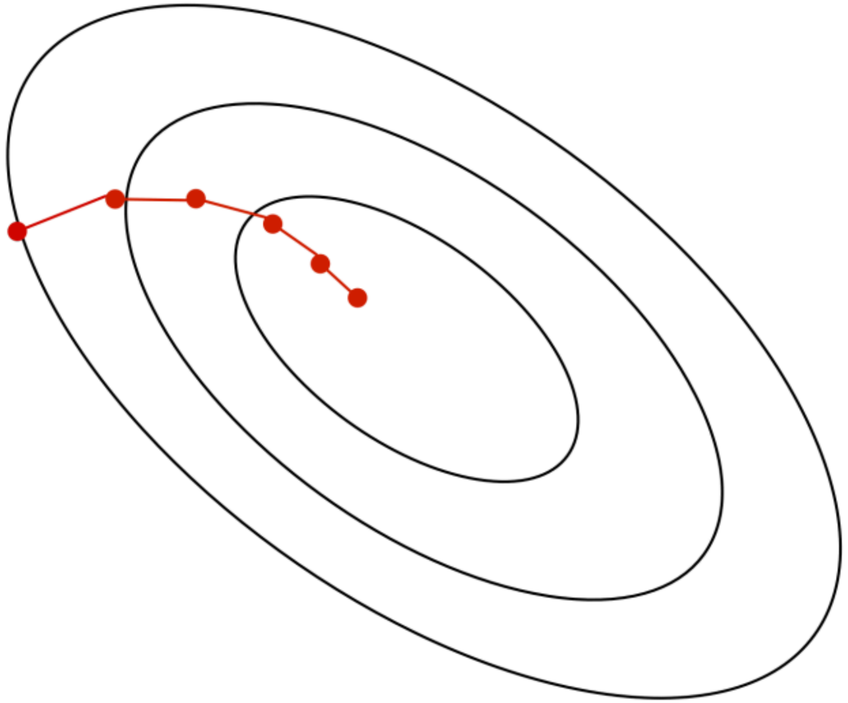
We know from statistics that we can estimate the average of a full "population" from a **sample.**

$$\nabla J(\theta) \approx \frac{1}{|B|} \sum_{i \in B} \nabla L(x_i, y_i, \theta)$$

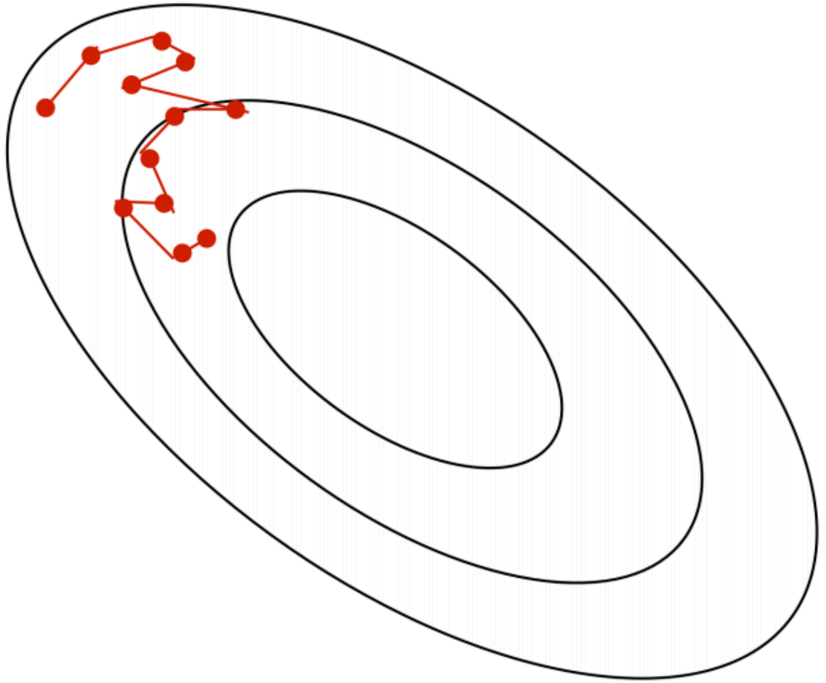
where B is a **minibatch:** a random subset of examples.

This is called **stochastic gradient descent (SGD)**.

Stochastic gradient descent



Batch gradient descent



Stochastic gradient descent

Source: R. Grosse

Stochastic gradient descent

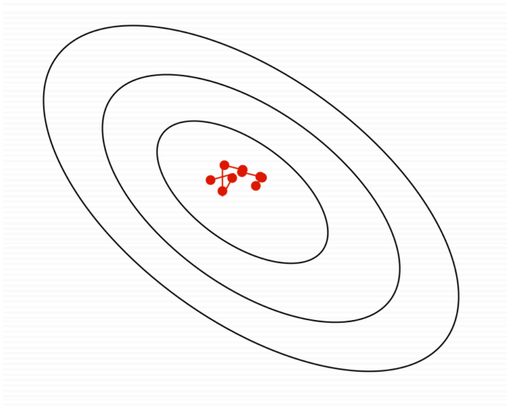
ML Hipster @ML_Hipster · Jul 20, 2012

"Oh sure, going in that direction will totally minimize the objective function" —Sarcastic Gradient Descent.

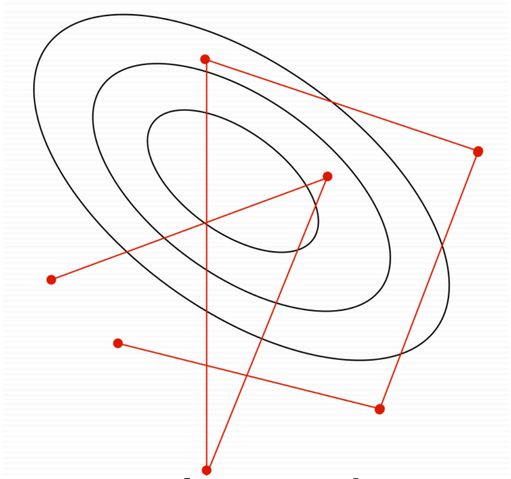
4 288 182

Learning rate

- Sensitive to the learning rate: $\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$



Small learning rate

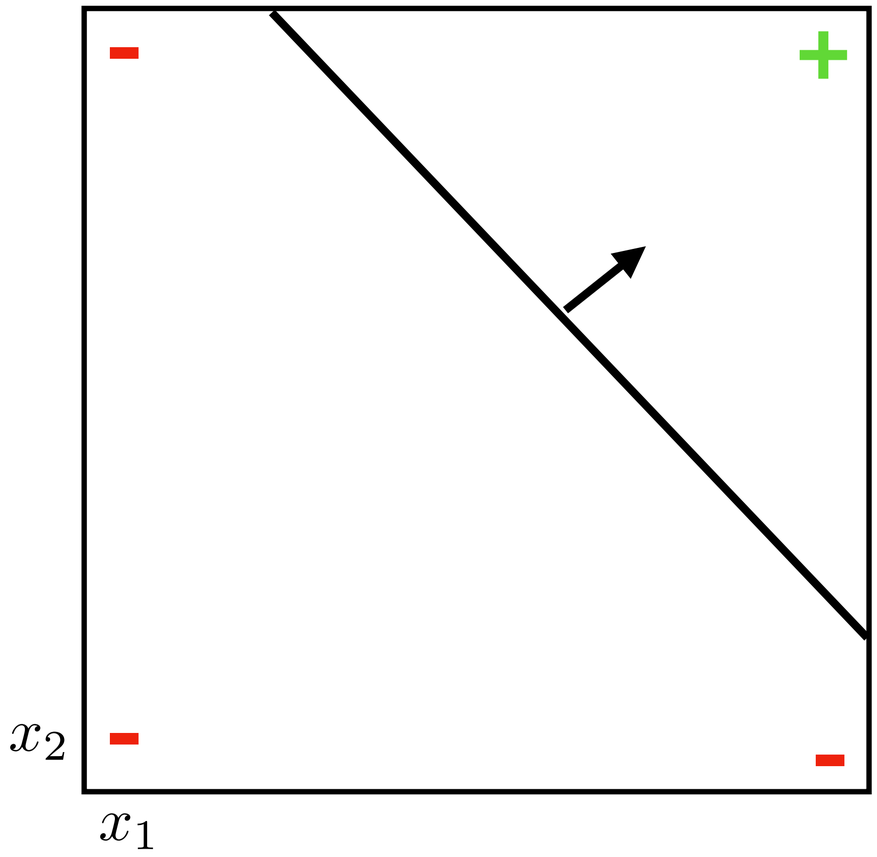


Large learning rate

- Often start with high learning rate, and decrease over time
- For example: start with $\eta_t = 0.1$, then decrease to $\eta_t = 0.01$ when training loss plateaus

Figure source: R. Grosse

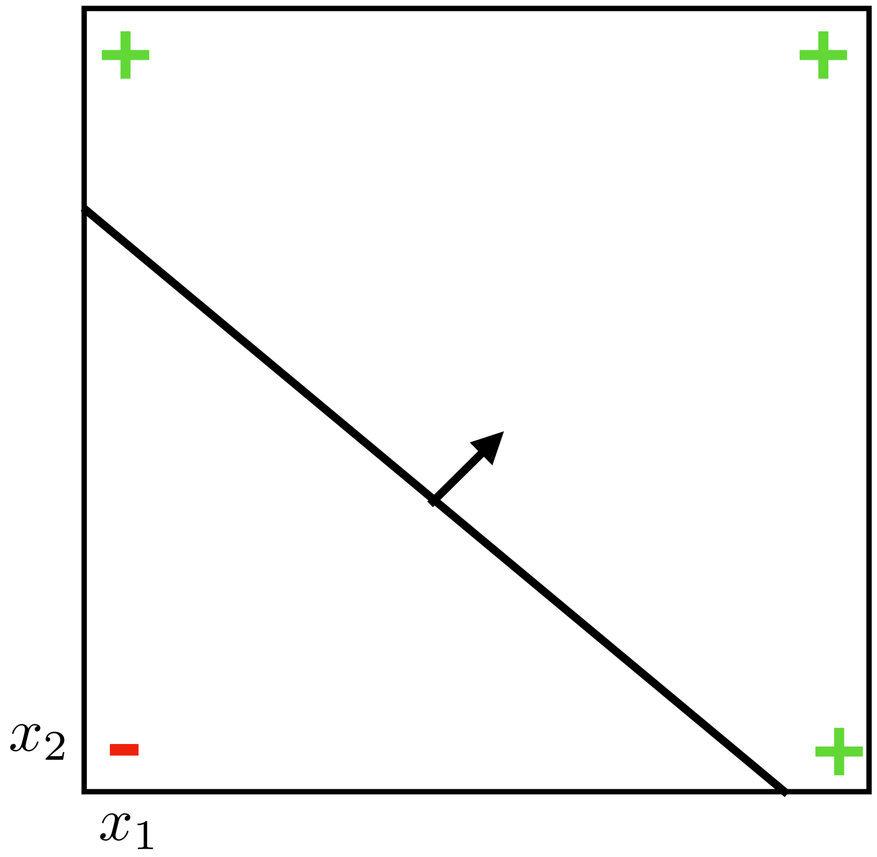
Limitations to linear classifiers



x_1 \ x_2	0	1
0	0	0
1	0	1

AND

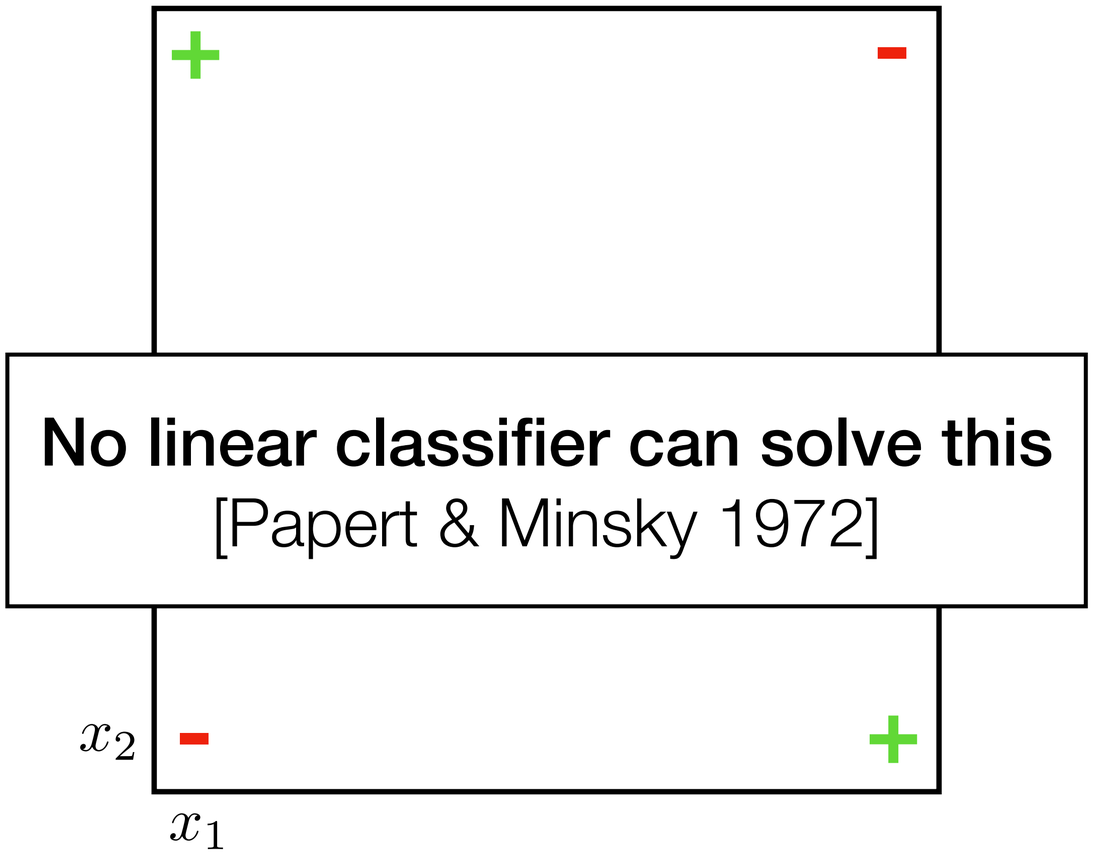
Limitations to linear classifiers



x_1 \ x_2	0	1
0	0	1
1	1	1

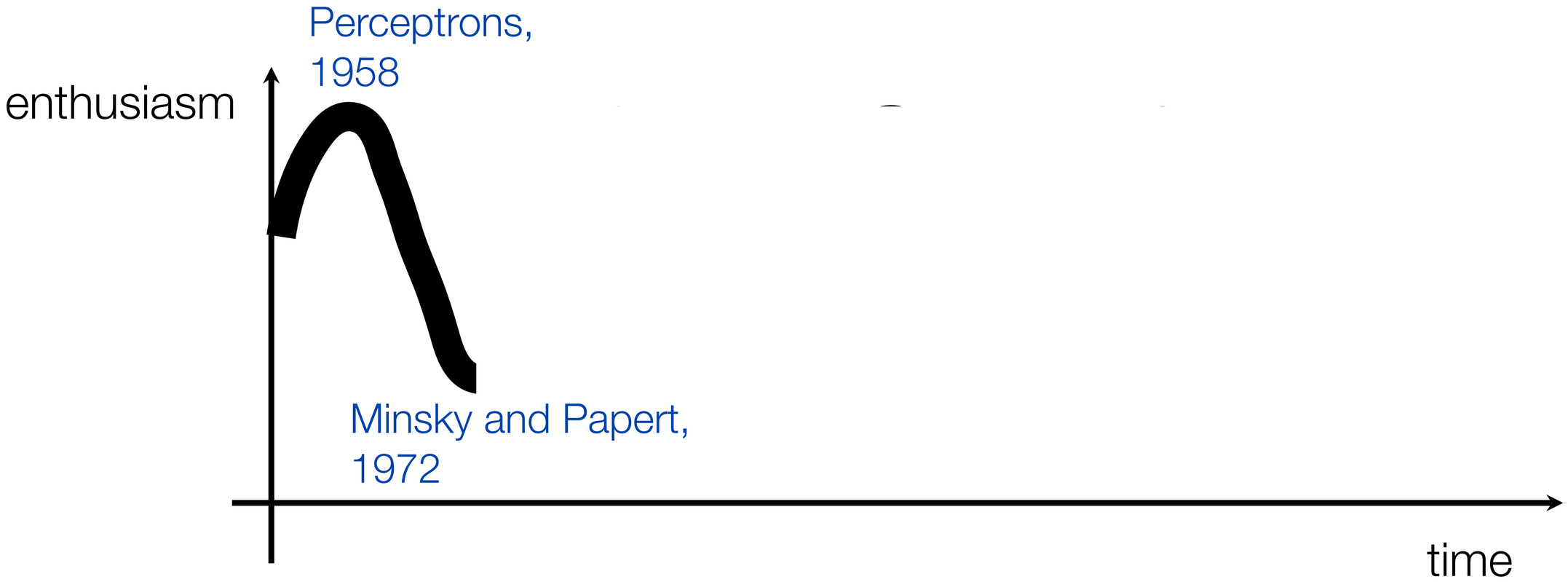
OR

Limitations to linear classifiers



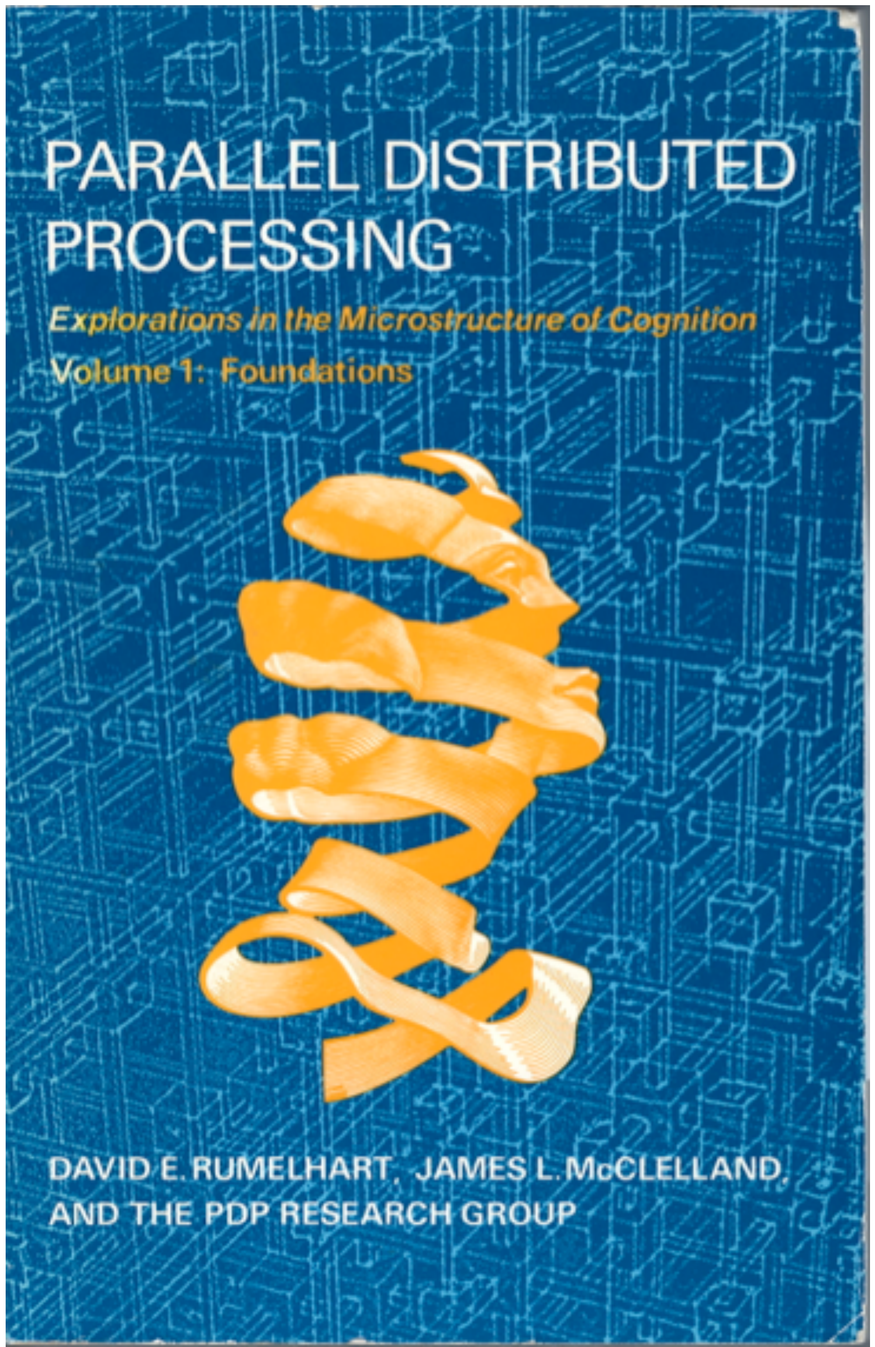
x_1 \ x_2	0	1
0	1	0
1	0	1

XOR

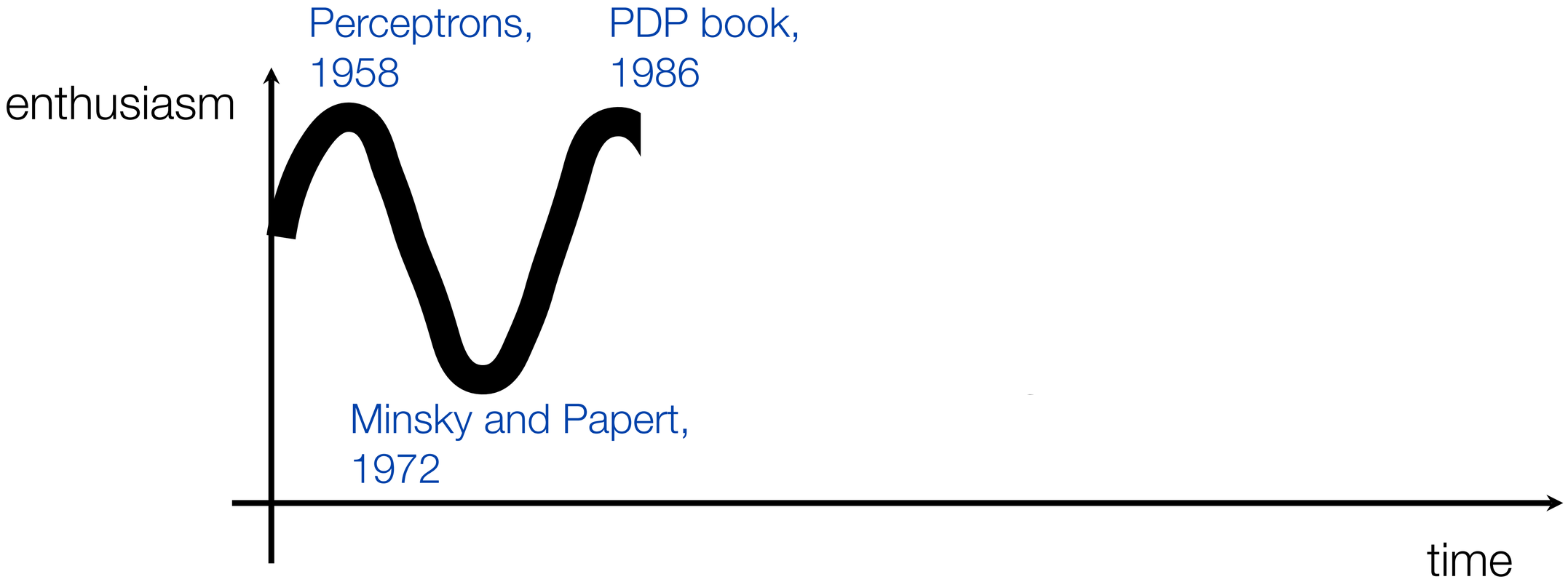


Based on slide by: Isola, Torralba, Freeman

Parallel Distributed Processing (PDP), 1986



Source: Isola, Torralba, Freeman



Source: Isola, Torralba, Freeman

LeCun convolutional neural networks

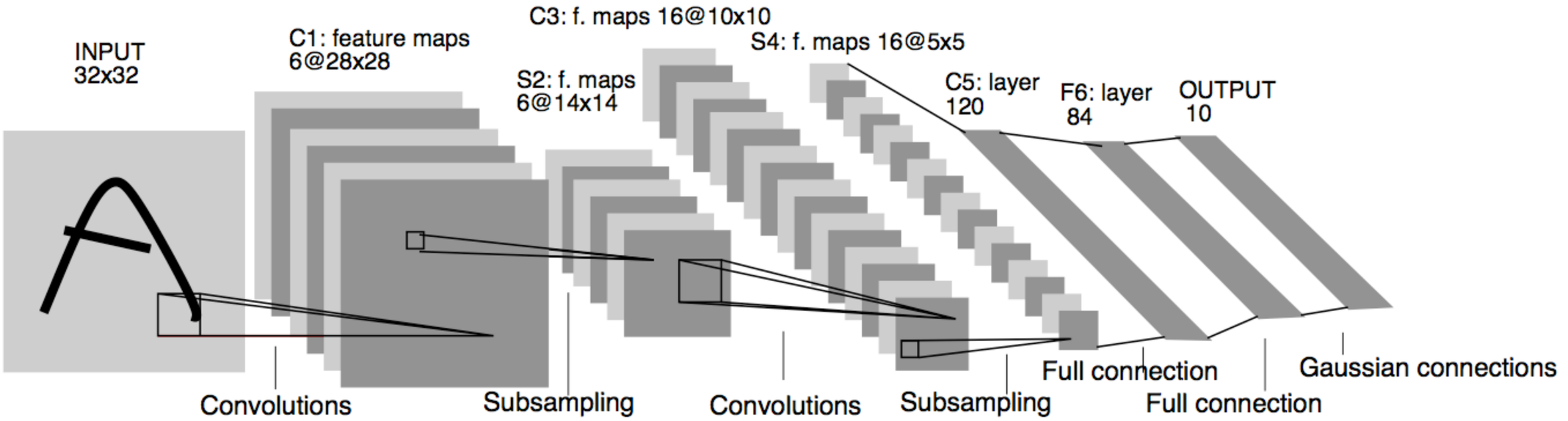


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Demos:
http://yann.lecun.com/exdb/lenet/index.html

Source: Isola, Torralba, Freeman

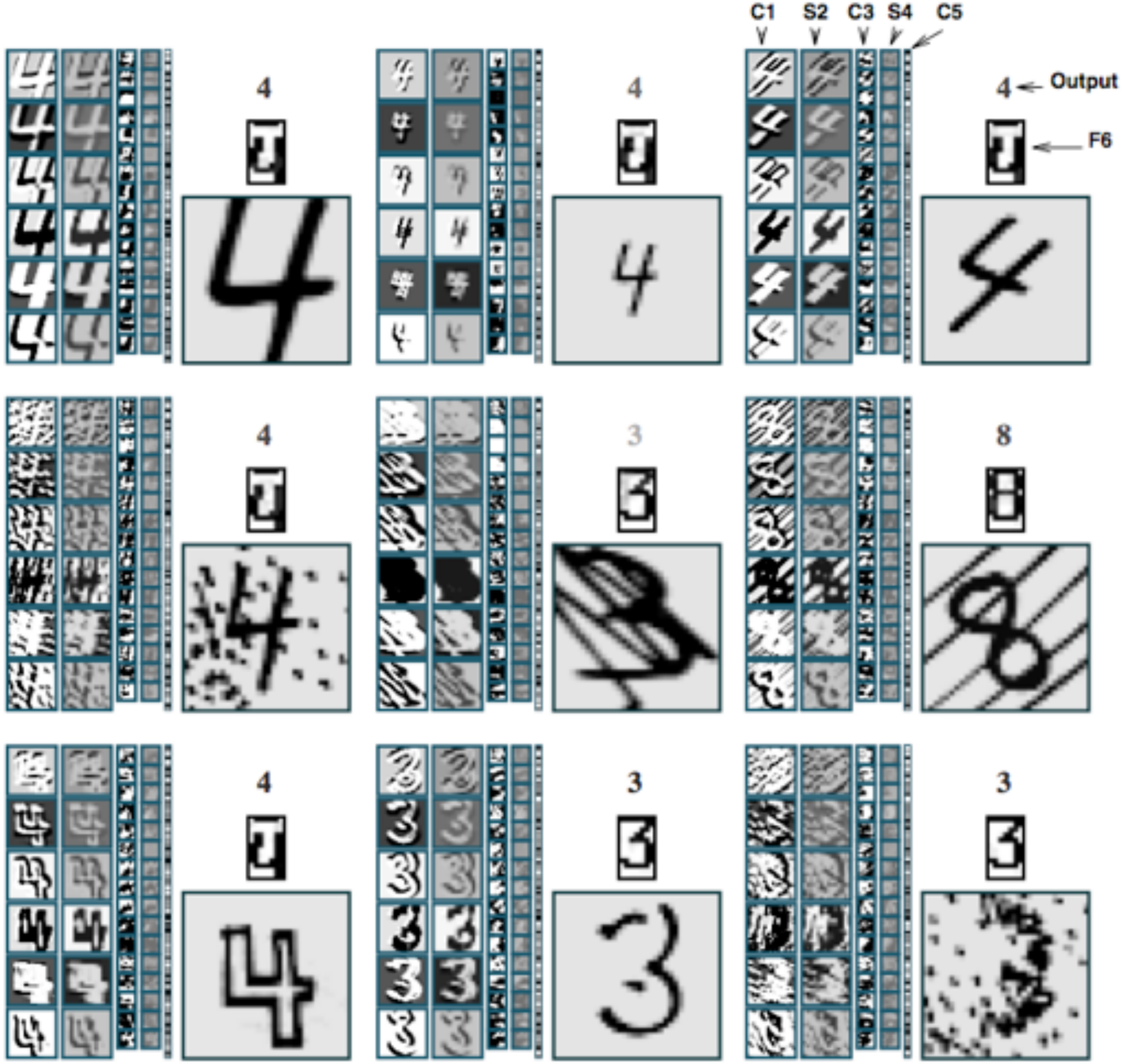
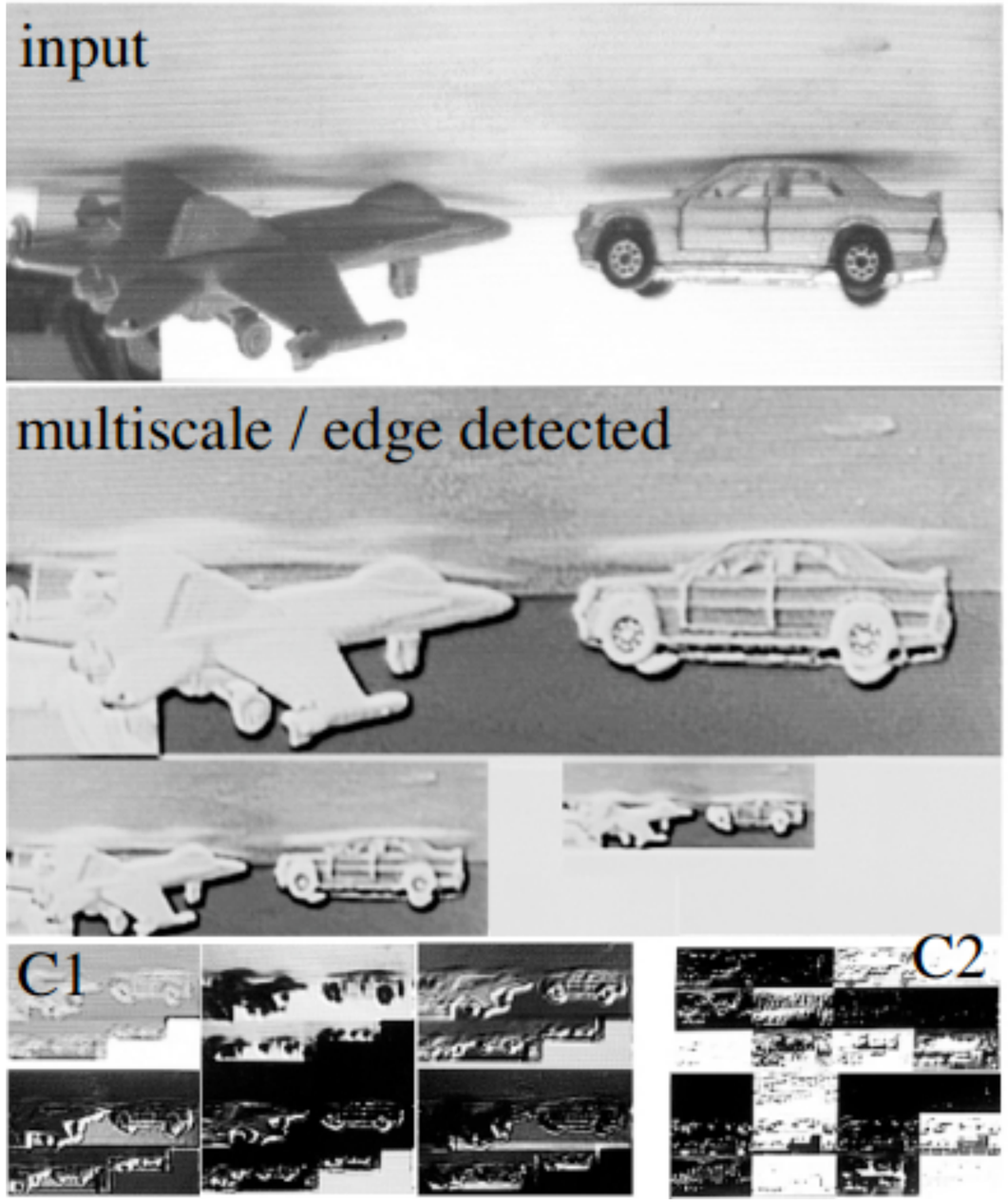


Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).

Source: Isola, Torralba, Freeman



Neural networks to recognize handwritten digits and human faces?
yes

Neural networks for tougher problems?
not really

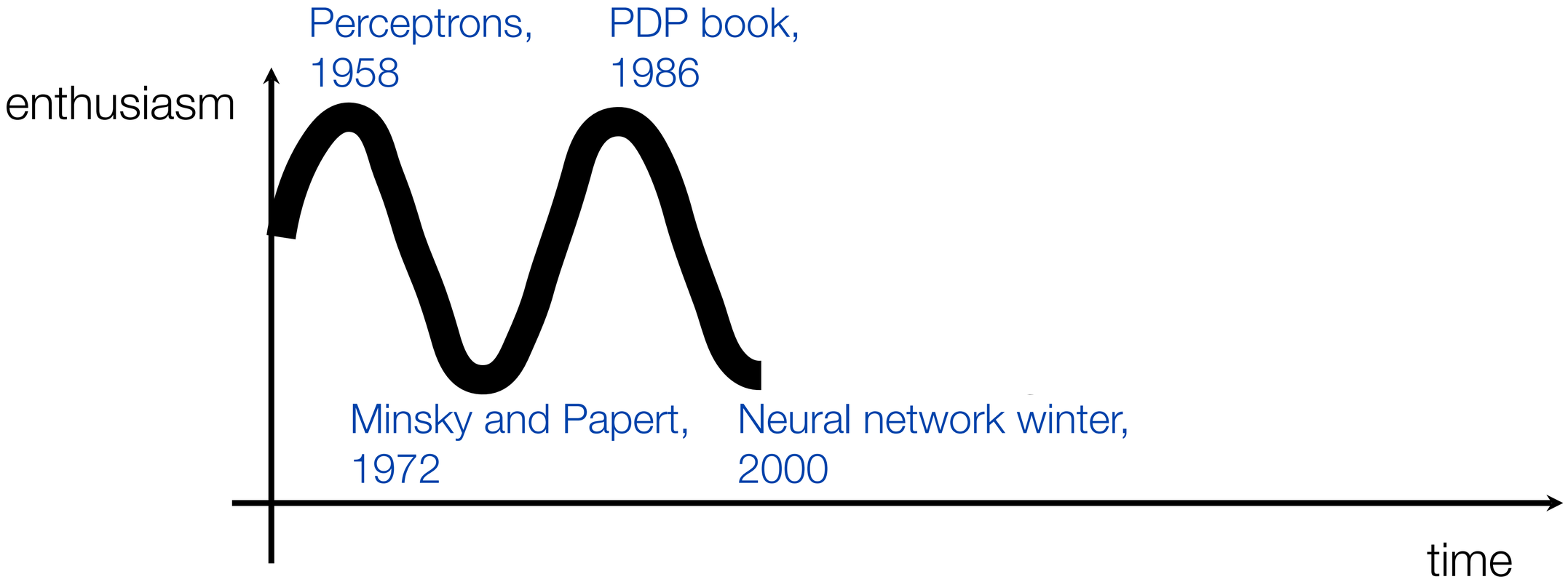
http://pub.clement.farabet.net/ecvw09.pdf

Source: Isola, Torralba, Freeman

Machine learning circa 2000

- Neural Information Processing Systems (NeurIPS), is a top conference on machine learning.
- For the 2000 conference:
  - title words predictive of paper acceptance: “Belief Propagation” and “Gaussian”.
  - title words predictive of paper rejection: “Neural” and “Network”.

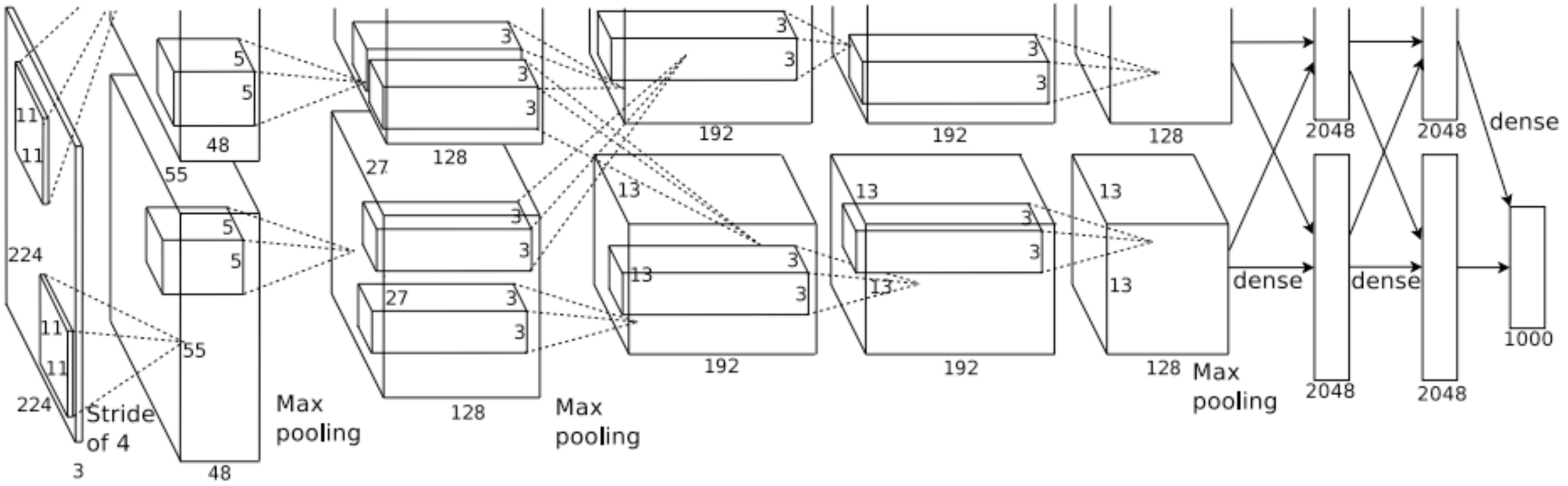
Source: Isola, Torralba, Freeman



Source: Isola, Torralba, Freeman

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

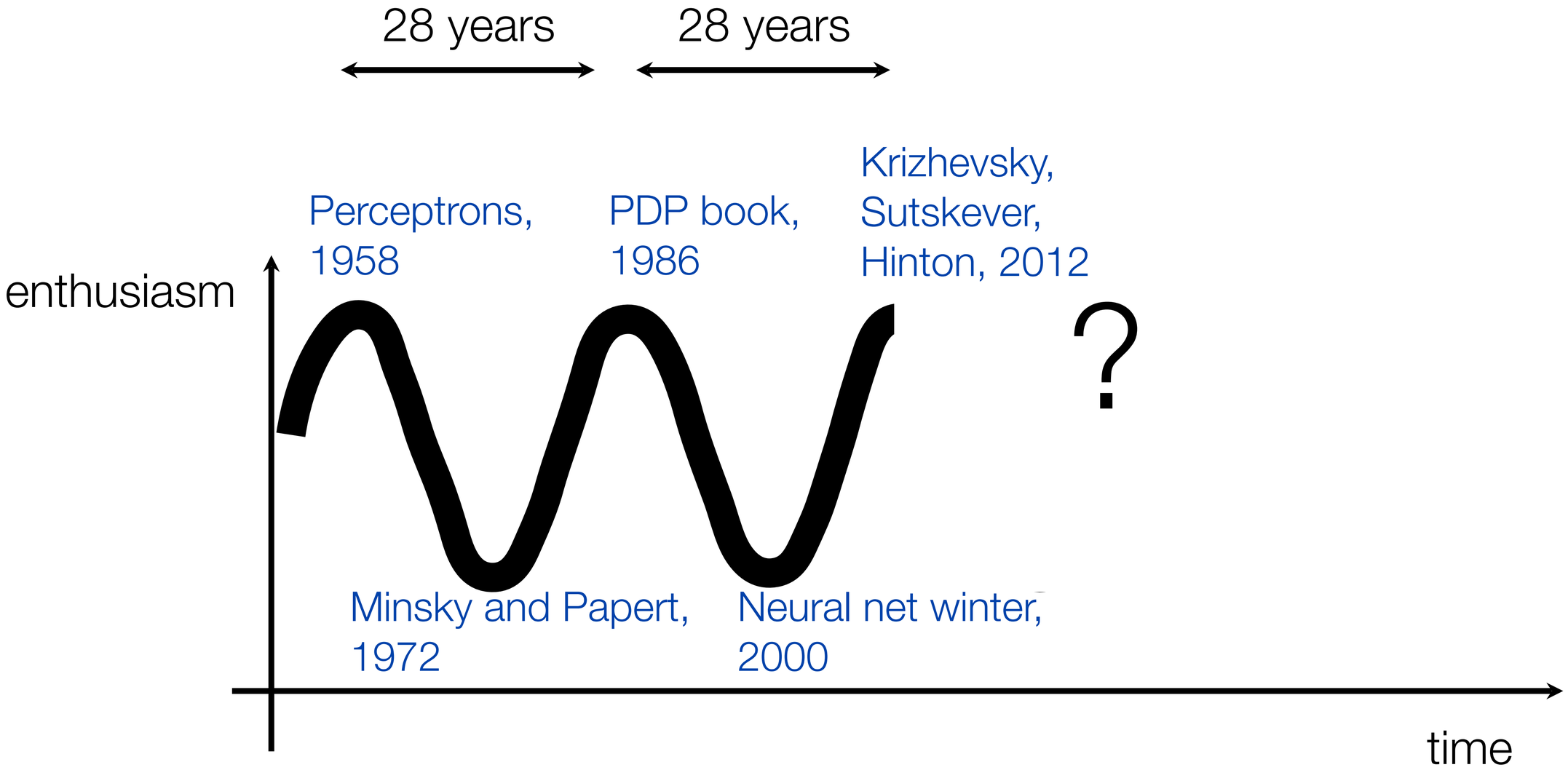
“AlexNet”



Source: Isola, Torralba, Freeman

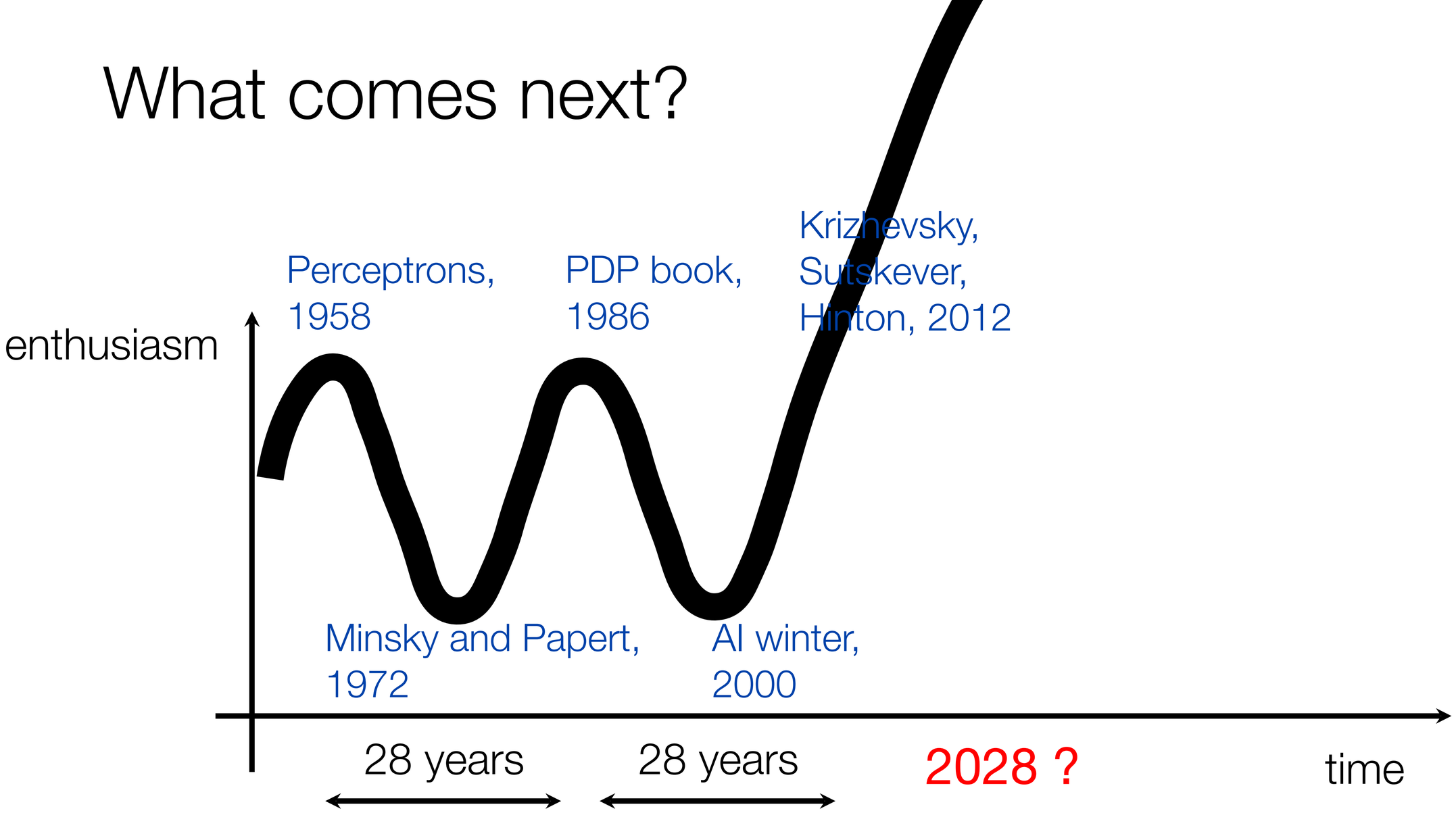
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

Source: Isola, Torralba, Freeman



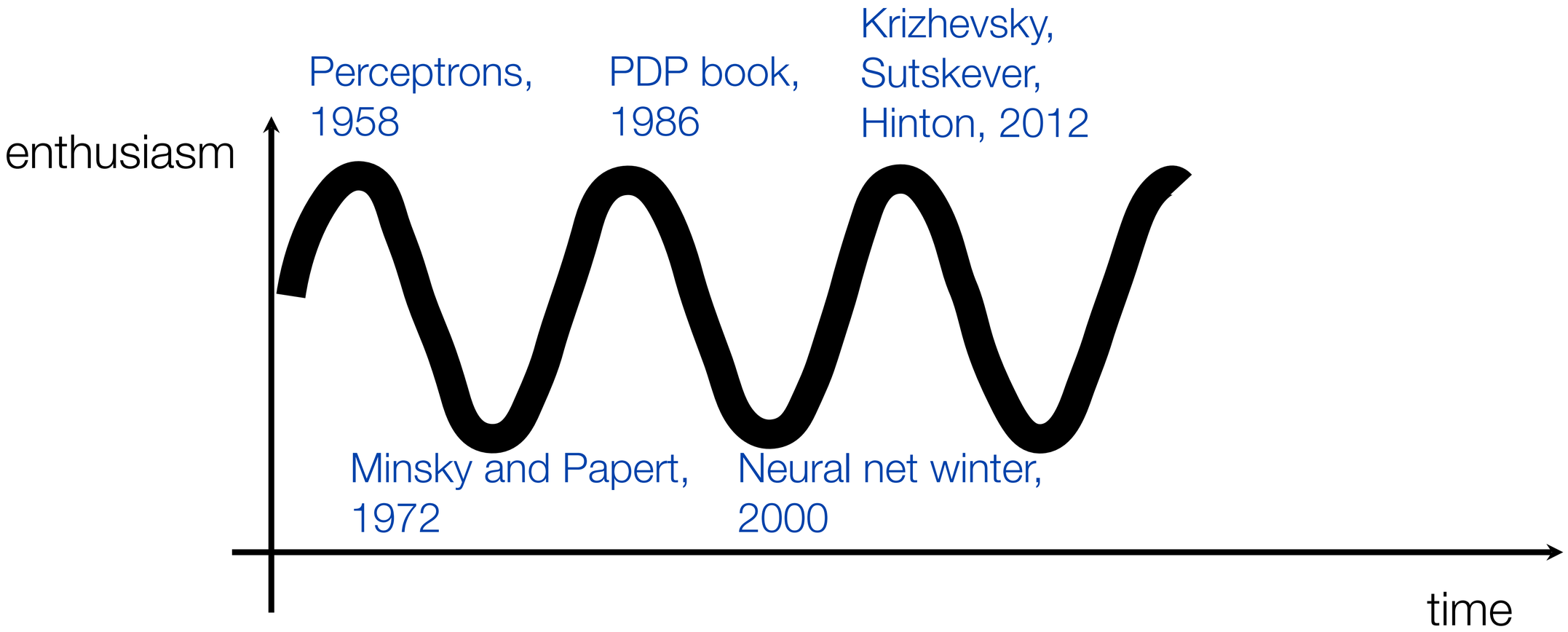
Source: Isola, Torralba, Freeman

What comes next?



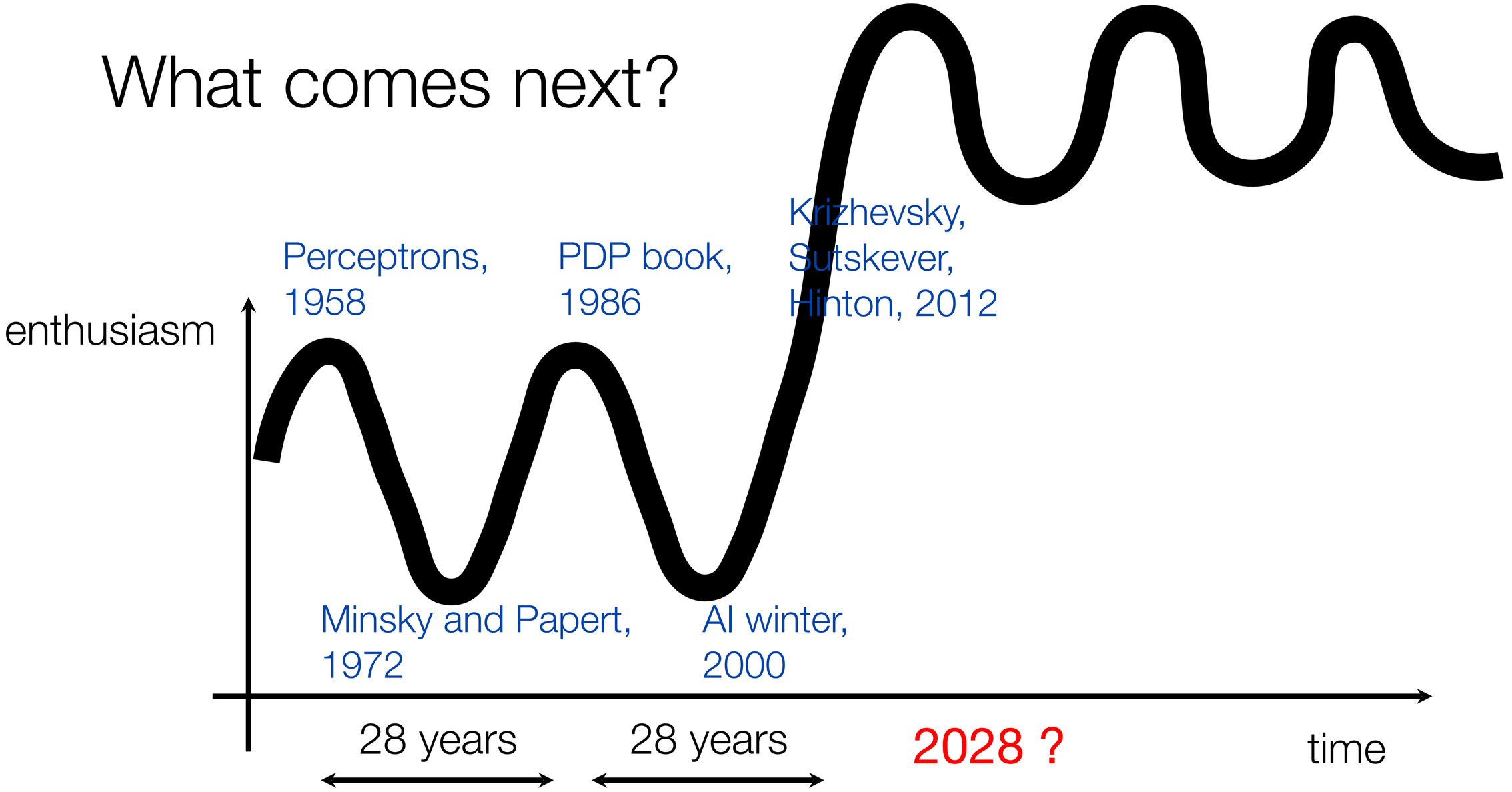
Source: Isola, Torralba, Freeman

What comes next?



Source: Isola, Torralba, Freeman

What comes next?



Source: Isola, Torralba, Freeman

Inspiration: Neurons

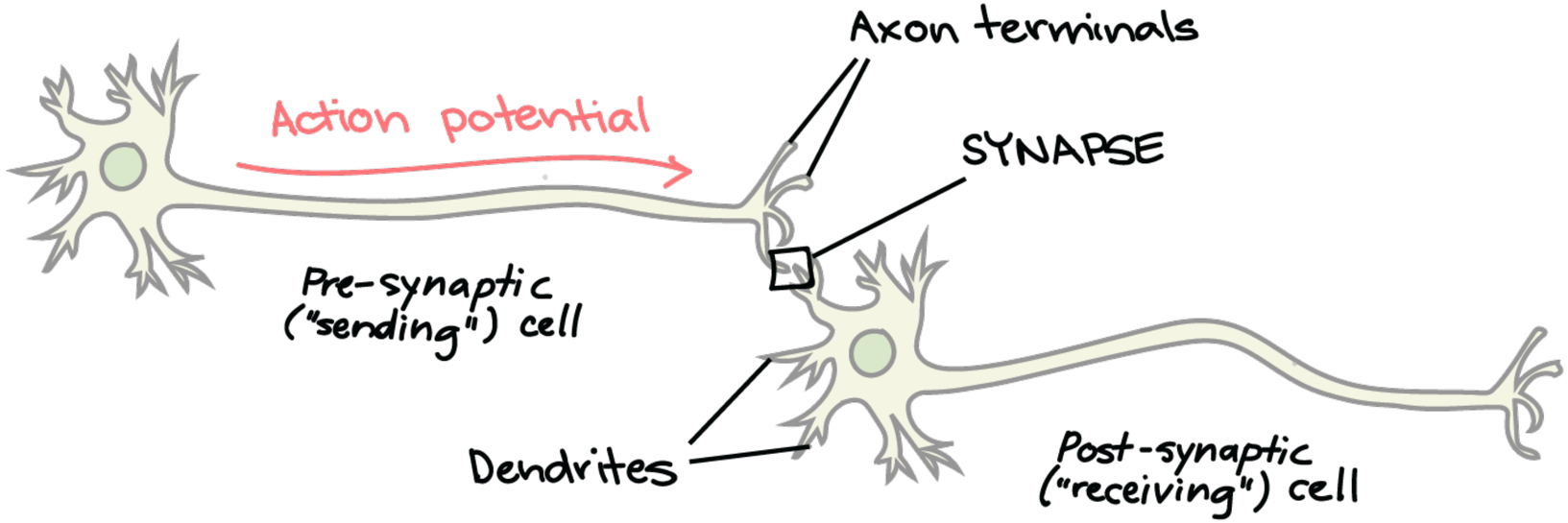
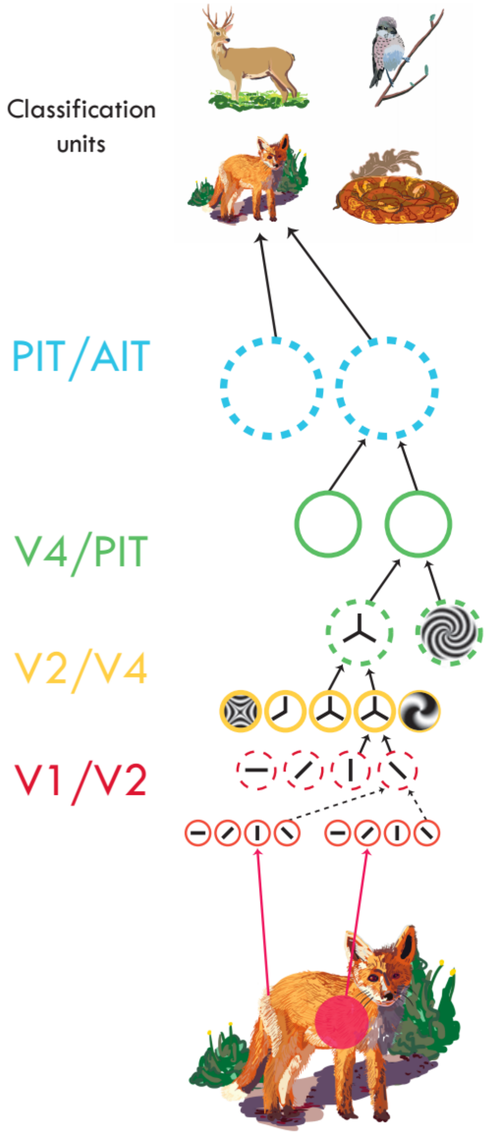


Image source: Khan academy

Inspiration: Hierarchical Representations

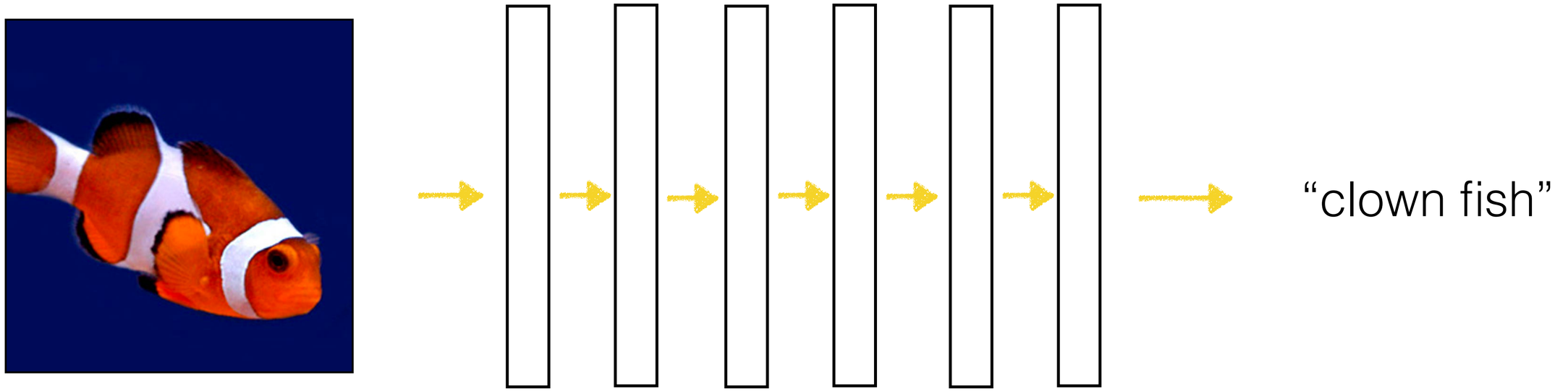


[Serre, 2014]

Best to treat as *inspiration*. The neural nets we'll talk about aren't very biologically plausible.

Source: Isola, Torralba, Freeman

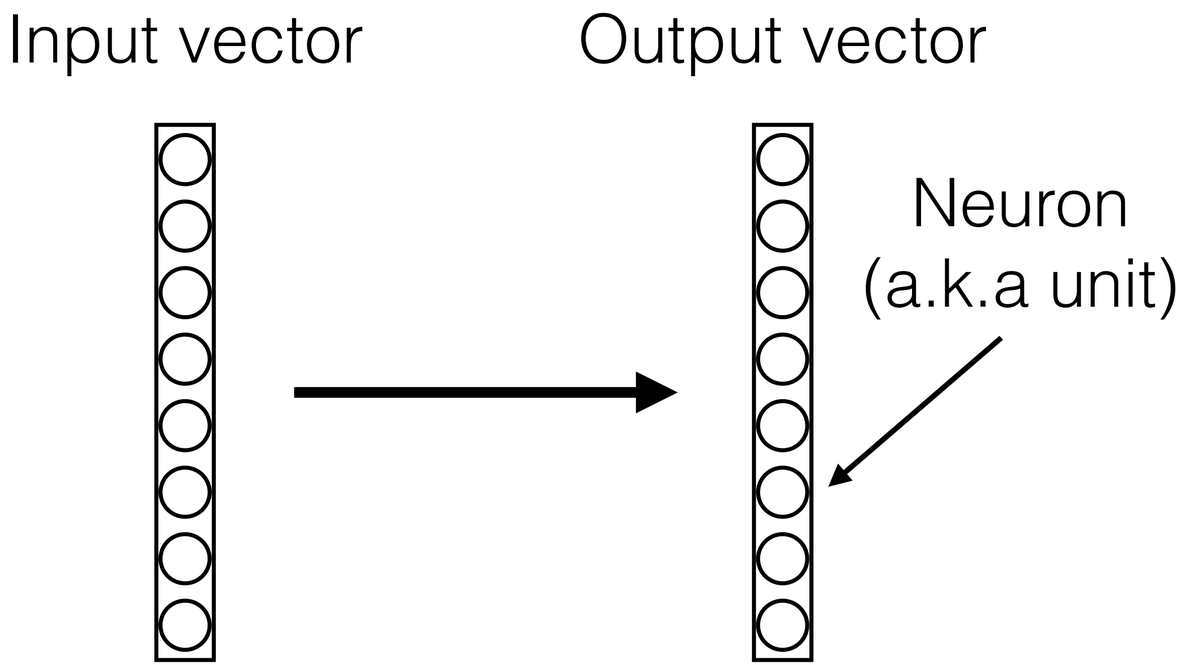
Object recognition



Neural network

Source: Isola, Torralba, Freeman

Computation in a neural net



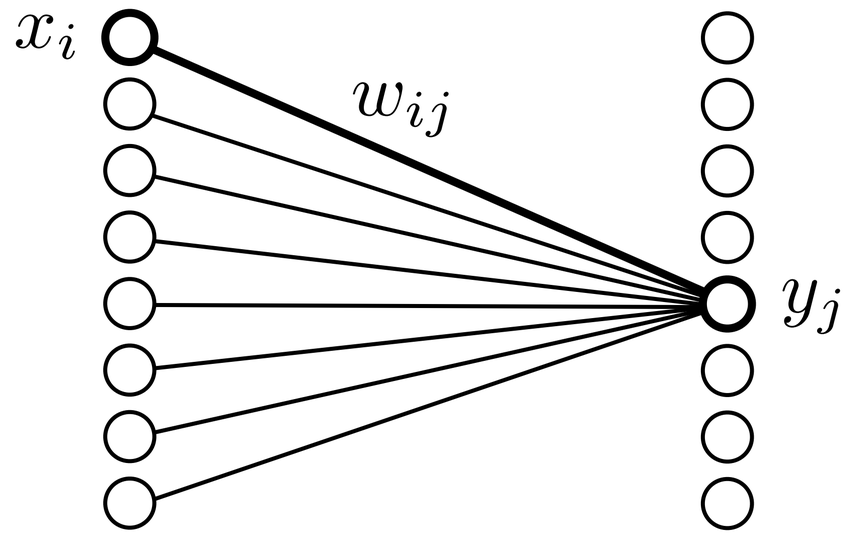
Adapted from: Isola, Torralba, Freeman

Computation in a neural net

Linear layer

Input representation

Output representation

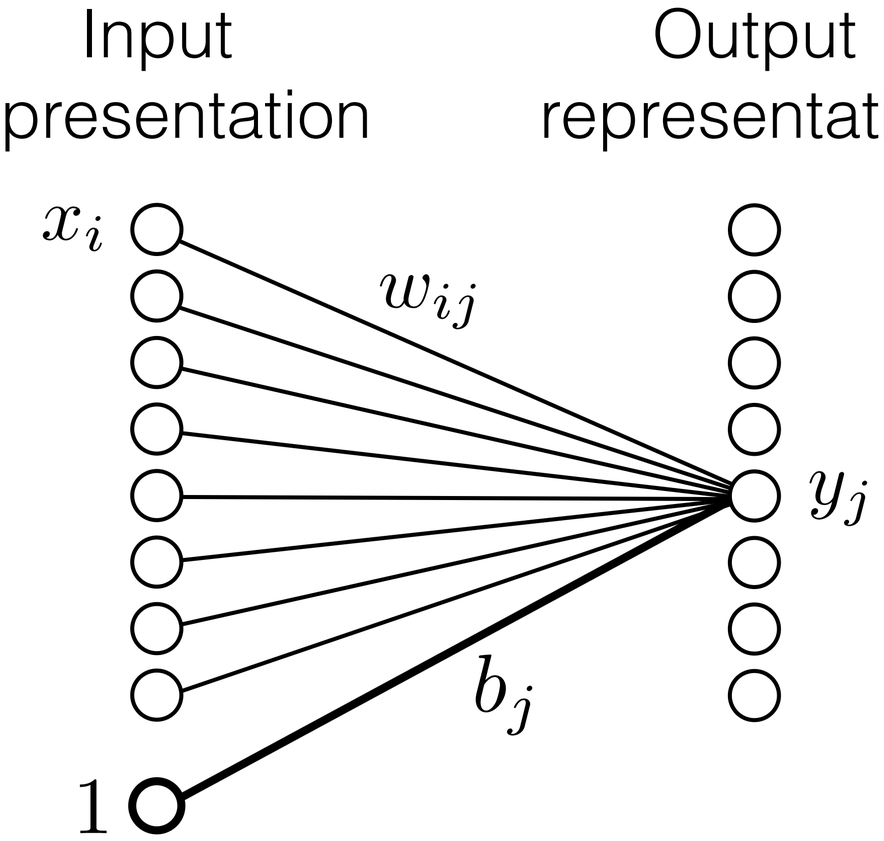


$$y_j = \sum_i w_{ij} x_i$$

Source: Isola, Torralba, Freeman

Computation in a neural net

Linear layer



$$y_j = \sum_i w_{ij} x_i + b_j$$

weights

bias

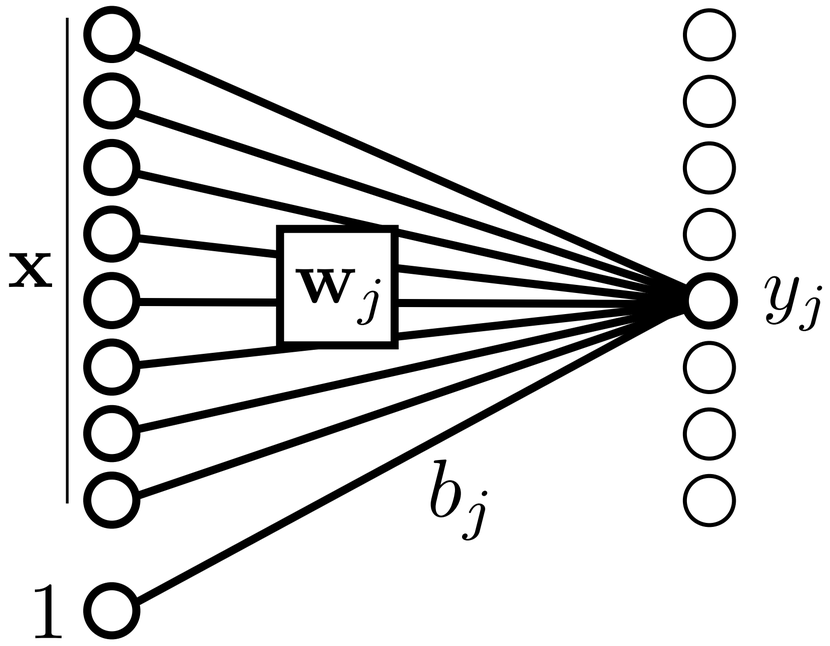
Source: Isola, Torralba, Freeman

Computation in a neural net

Linear layer

Input representation

Output representation



weights

$$y_j = \mathbf{x}^T \mathbf{w}_j + b_j$$

bias

$$\theta = \{\mathbf{W}, \mathbf{b}\}$$

parameters of the model

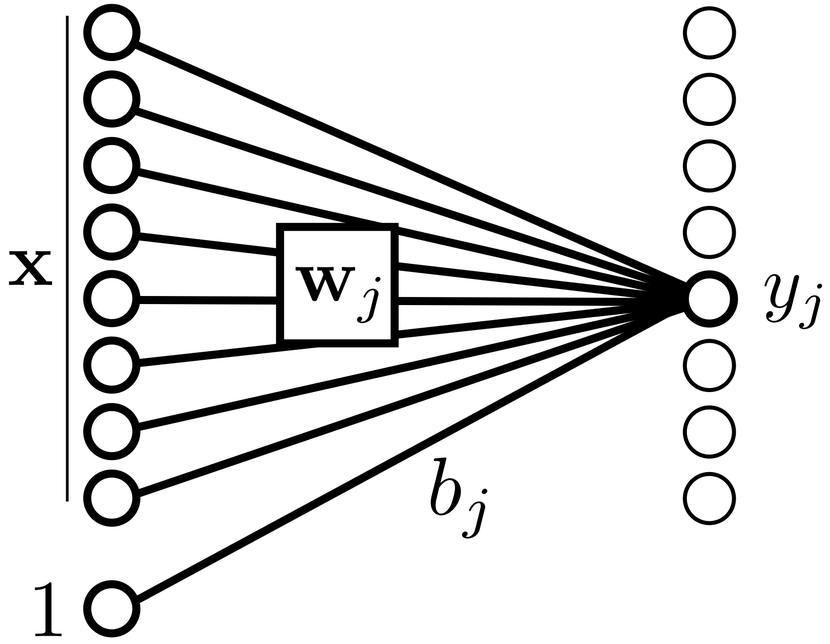
Source: Isola, Torralba, Freeman

Computation in a neural net

Linear layer

Input representation

Output representation



Full layer

weights

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

bias

$$\theta = \{\mathbf{W}, \mathbf{b}\}$$

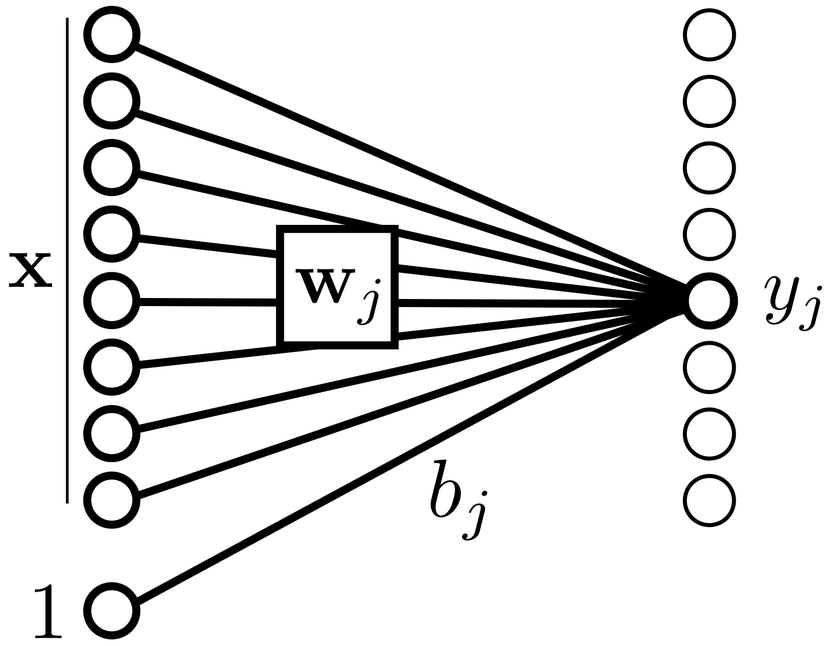
parameters of the model

Computation in a neural net

Linear layer

Input representation

Output representation



Full layer

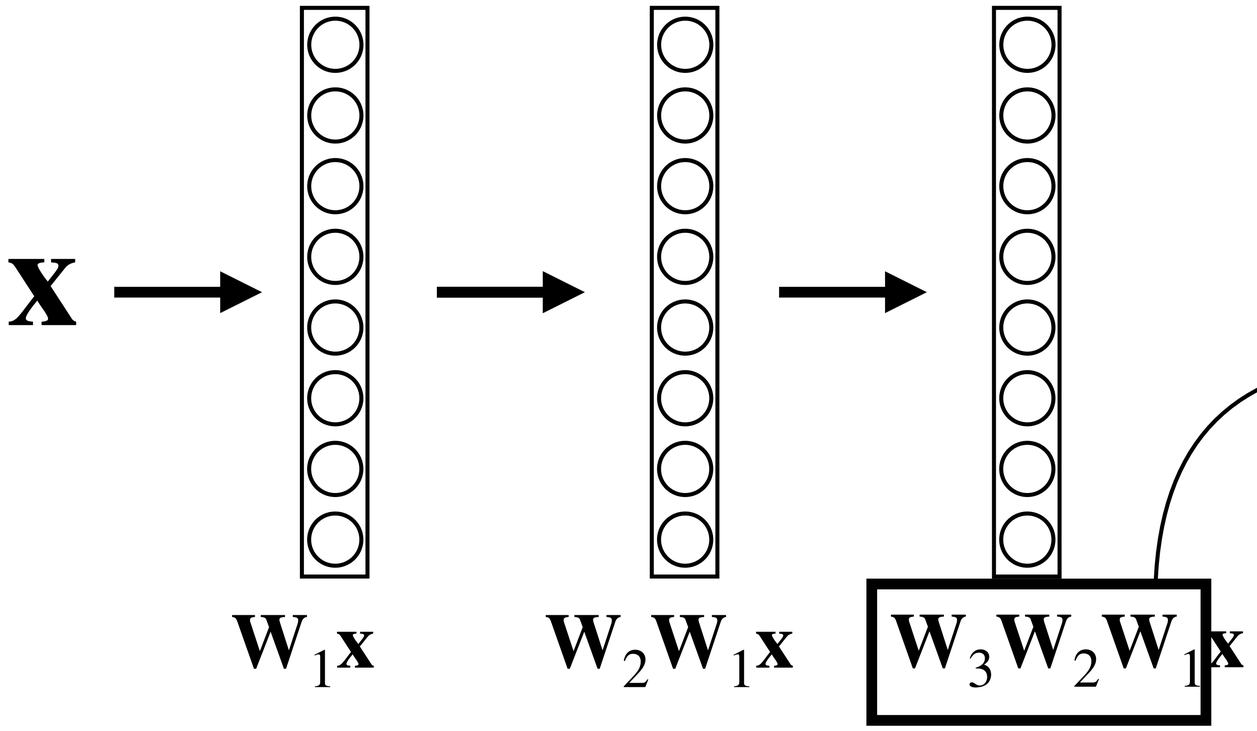
weights + bias

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

Can again simplify notation by appending a 1 to $\mathbf{x}$

What's the problem with this idea?

Consider stacking **multiple** linear layers:



Can be expressed as single linear layer!

$$\left(\prod_i \mathbf{W}_i\right)\mathbf{x} = \hat{\mathbf{W}}\mathbf{x}$$

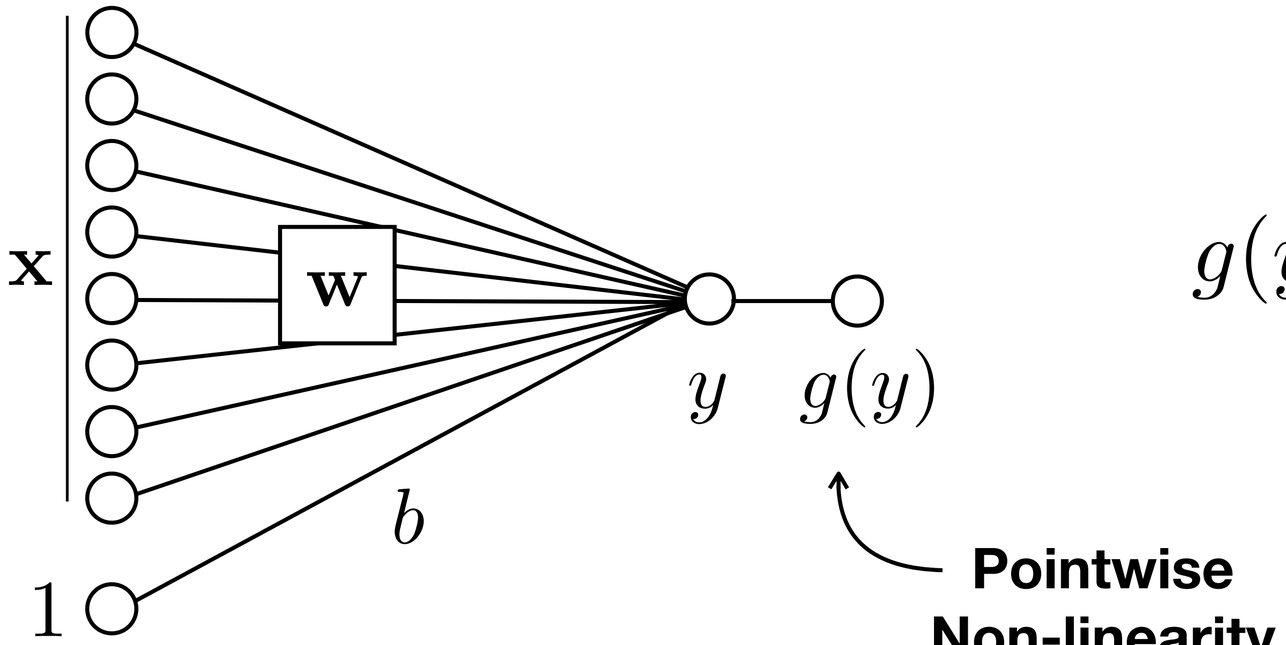
Limited power, e.g. can't solve XOR.

Solution: simple nonlinearity

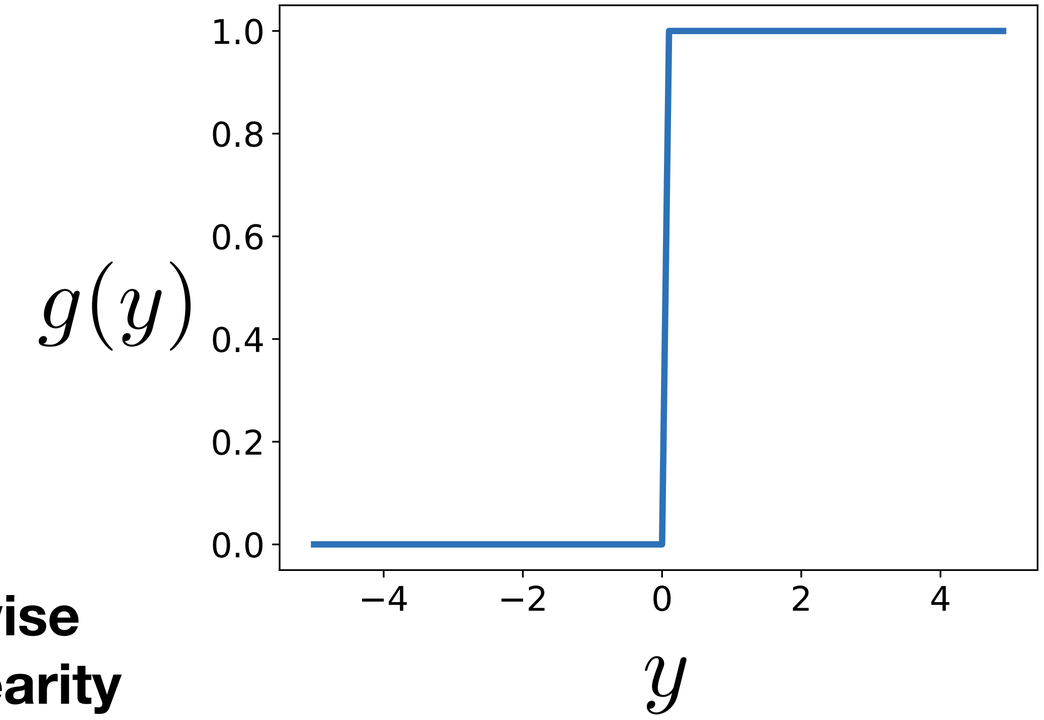
“Perceptron”

Input representation

Output representation

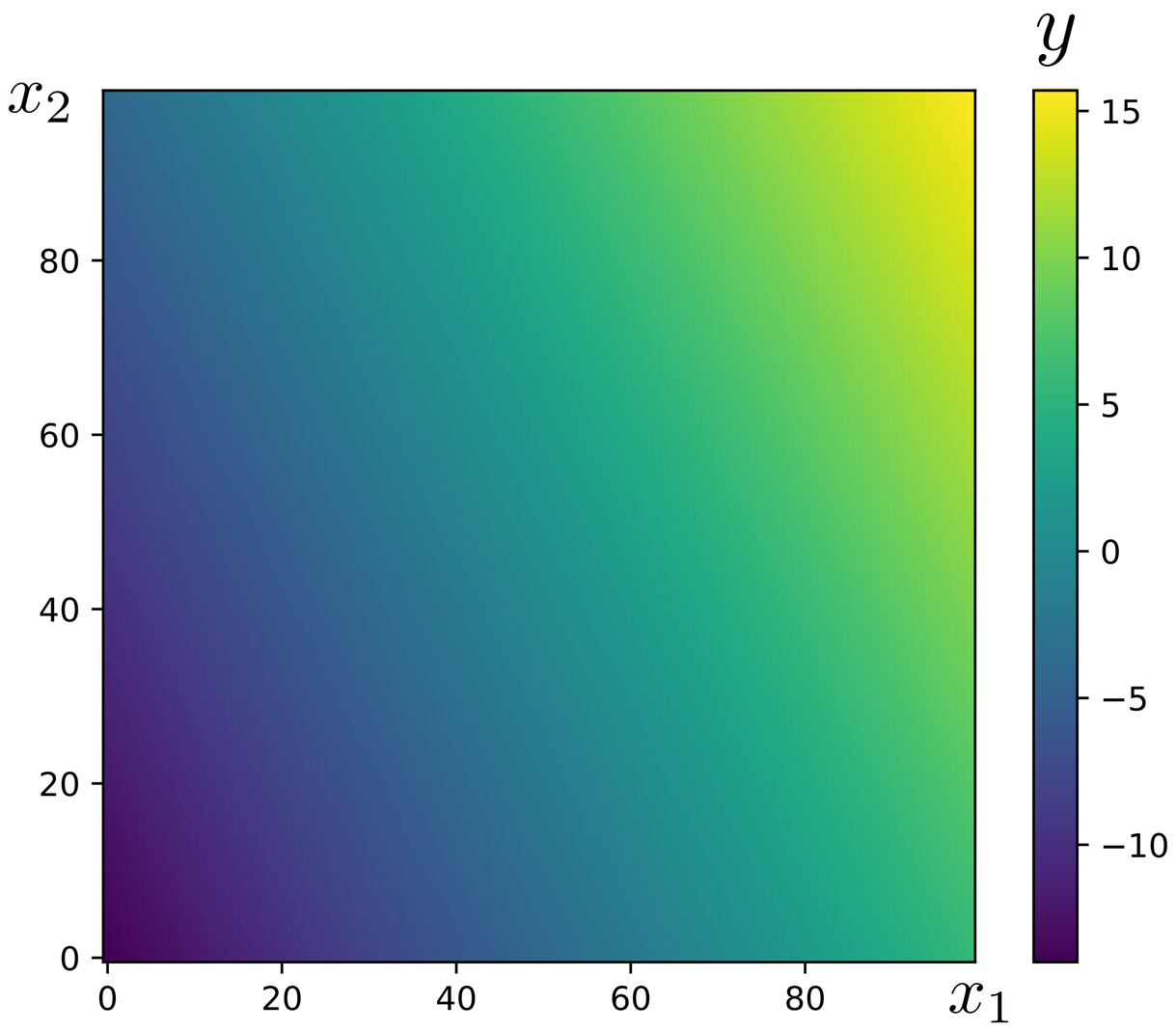


$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



Source: Isola, Torralba, Freeman

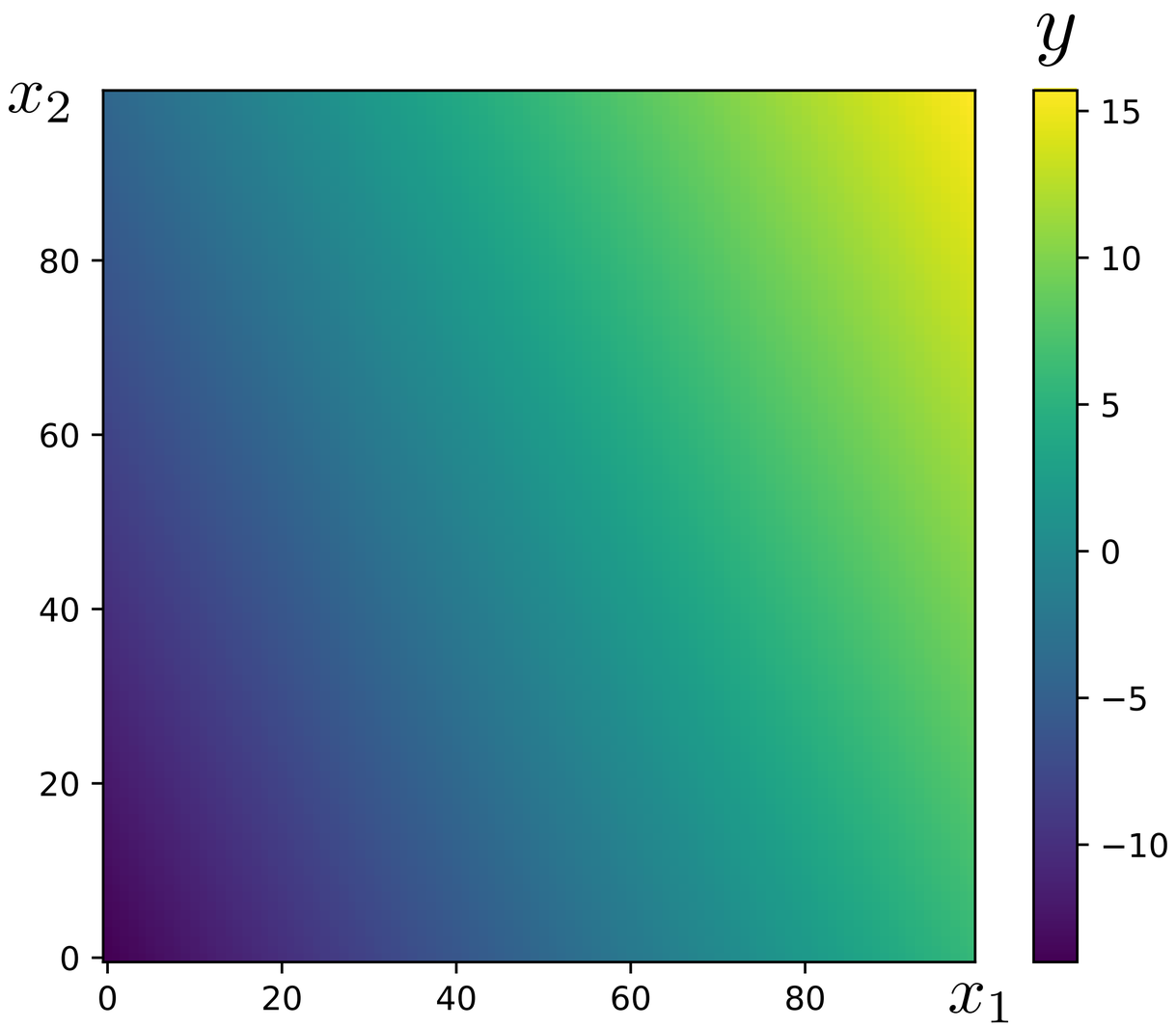
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

Source: Isola, Torralba, Freeman

Example: linear classification with a perceptron

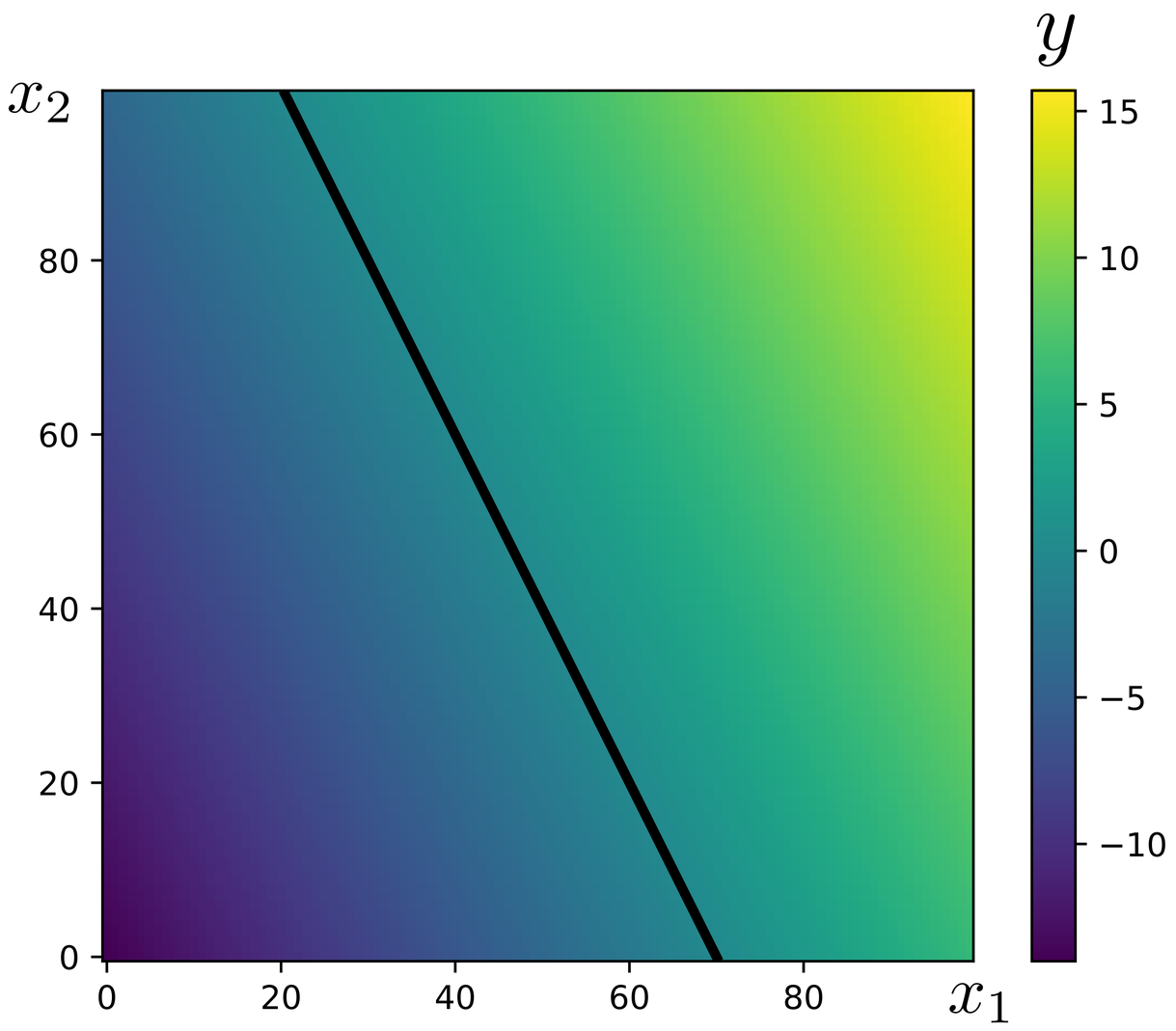


$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

Source: Isola, Torralba, Freeman

Example: linear classification with a perceptron

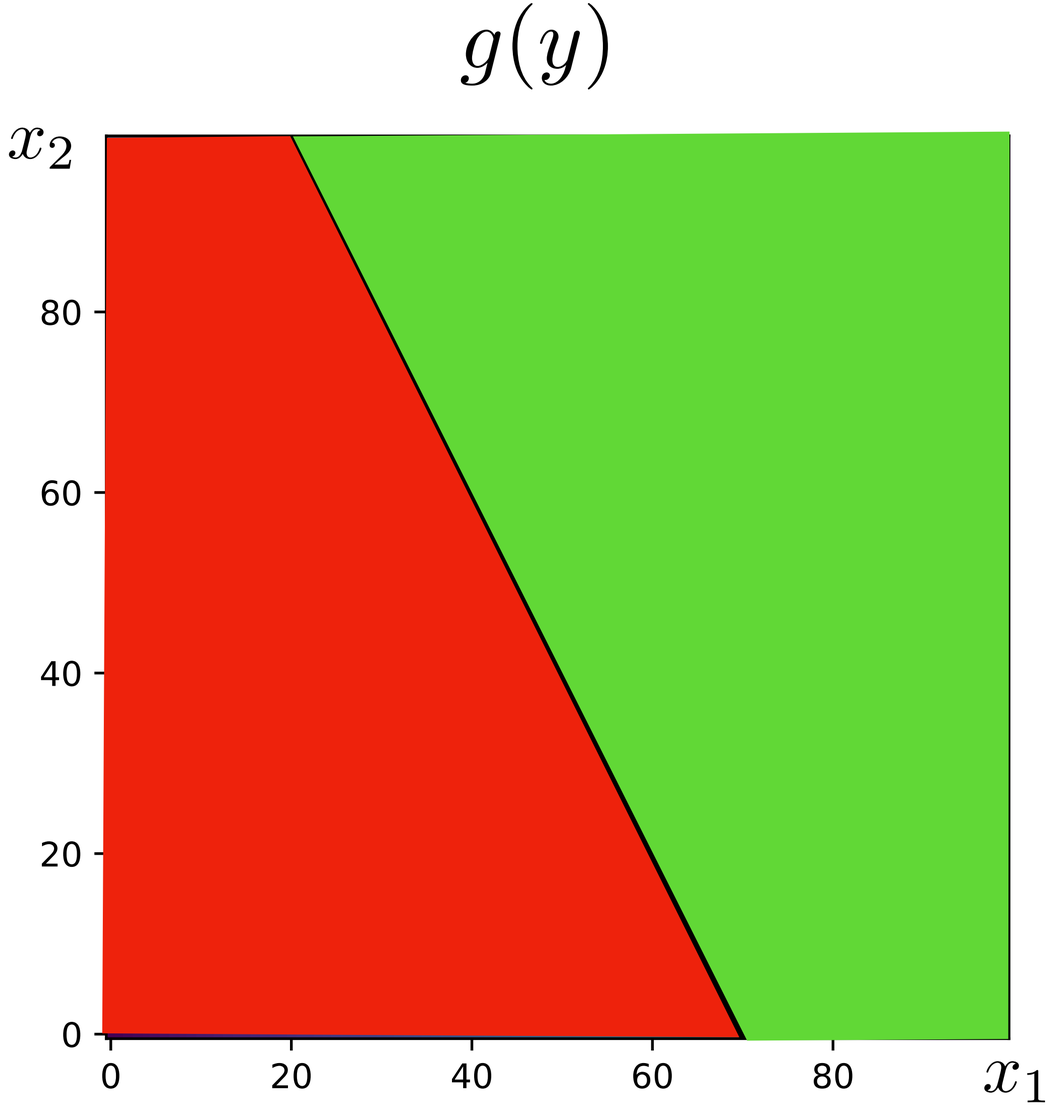


$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

Source: Isola, Torralba, Freeman

Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

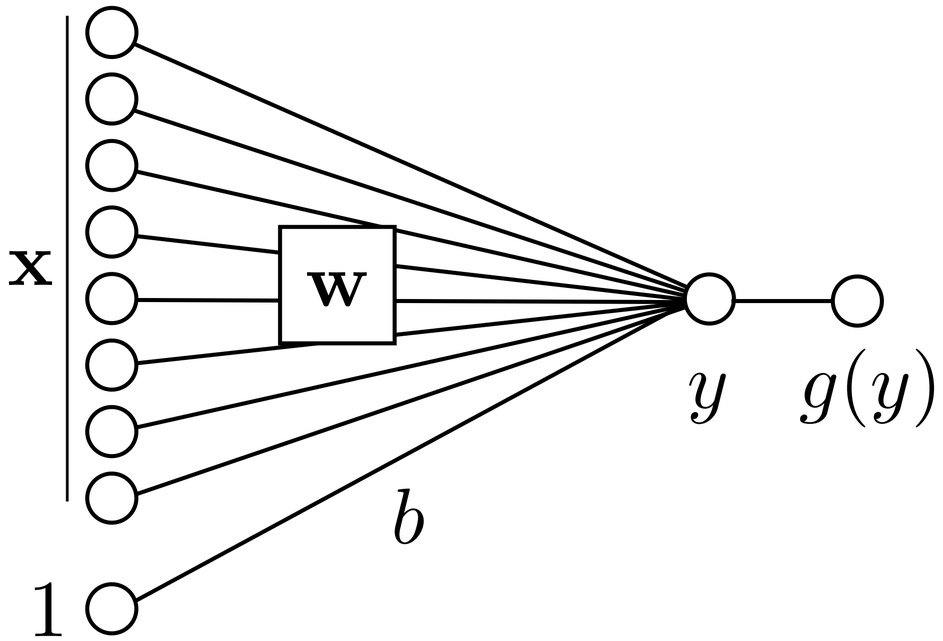
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

Source: Isola, Torralba, Freeman

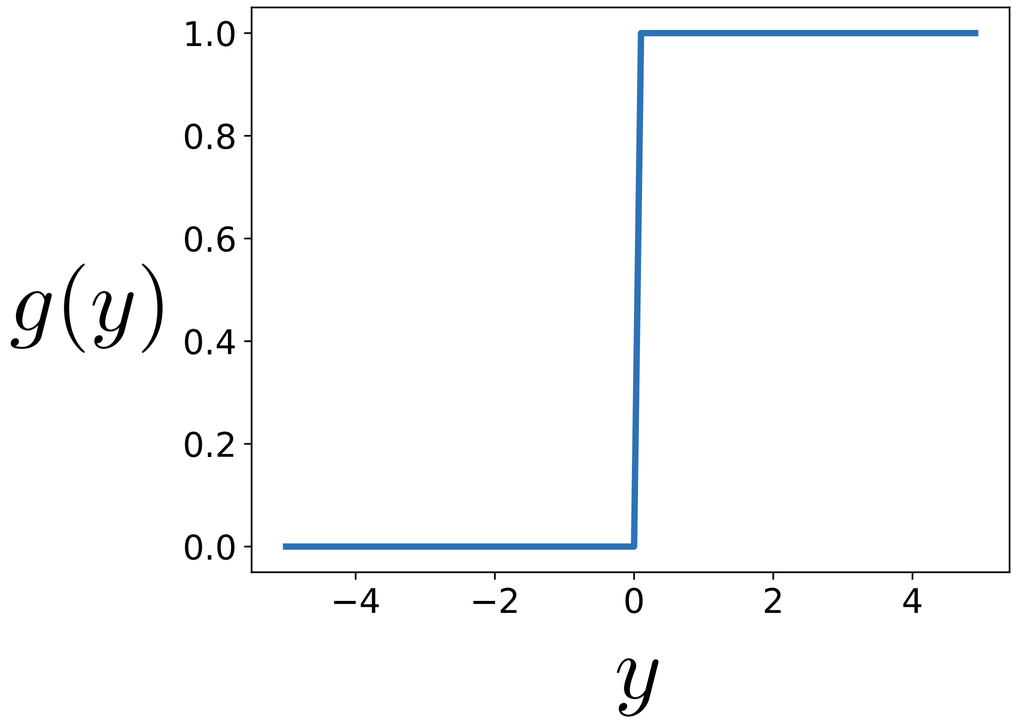
Computation in a neural net — nonlinearity

Input representation

Output representation



$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



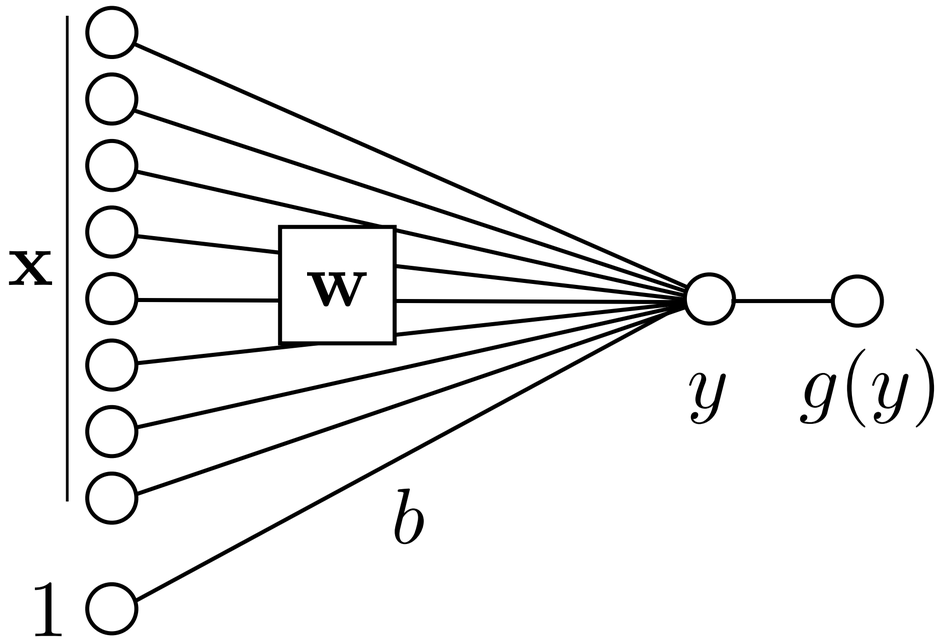
Can't use with gradient descent, $\nabla g = 0$

Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

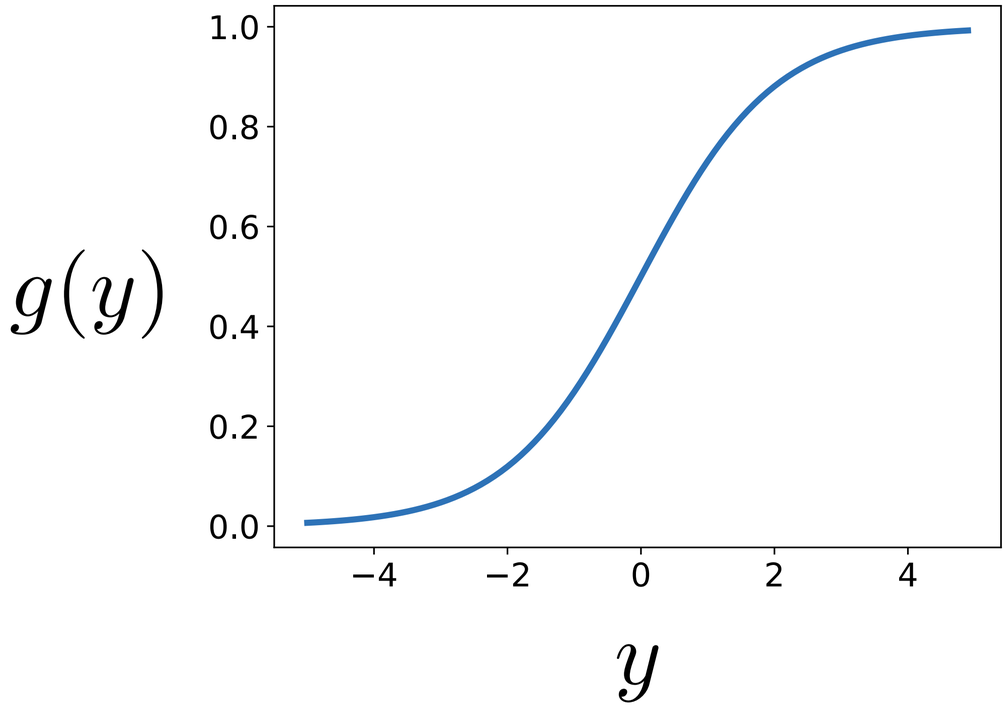
Input representation

Output representation



Sigmoid

$$g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$$



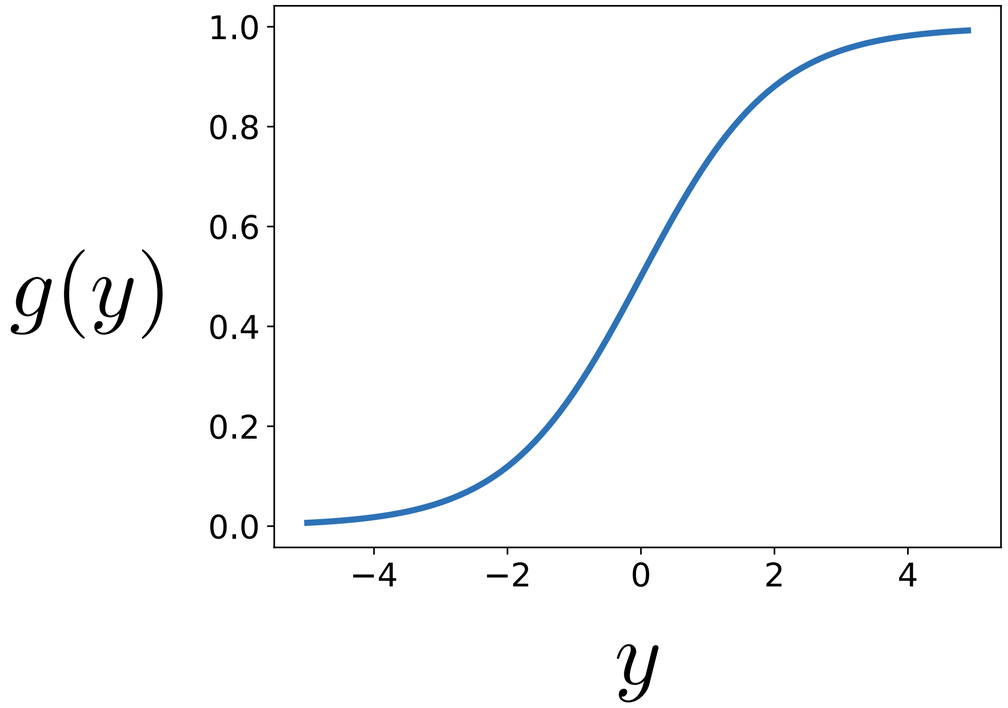
Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

- Interpretation as firing rate of neuron
- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero
- Centered at 0.5. Better in practice to use: $\tanh(y) = 2g(y) - 1$

Sigmoid

$$g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$$



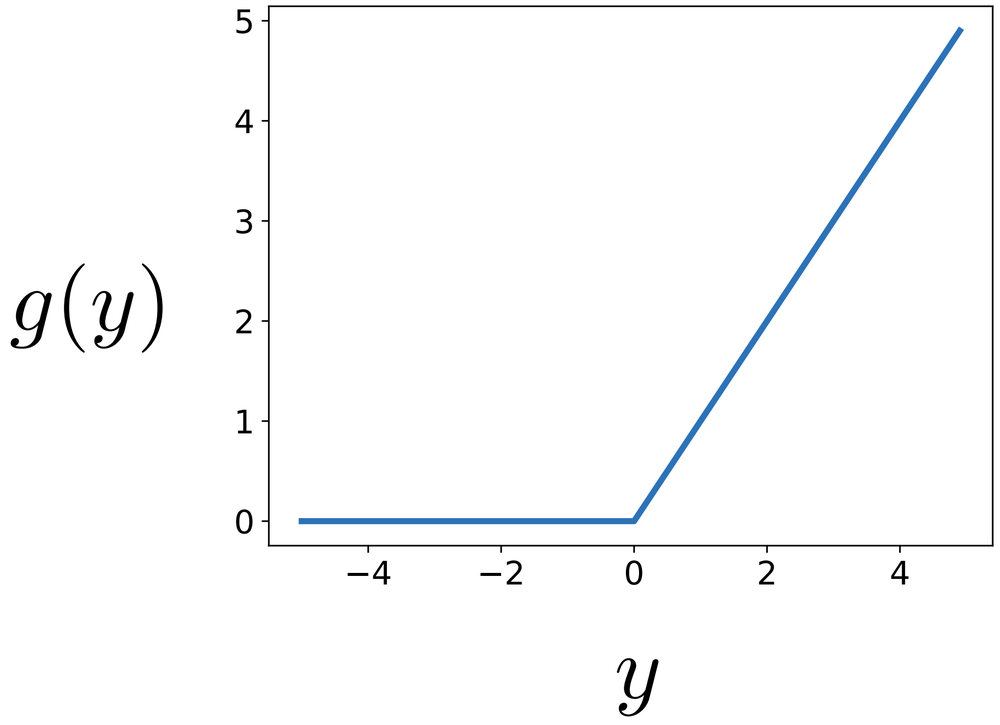
Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

- Unbounded output (on positive side)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} 0, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$
- Also seems to help convergence (see 6x speedup vs tanh in [Krizhevsky et al.])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models!

Rectified linear unit (ReLU)

$$g(y) = \max(0, y)$$



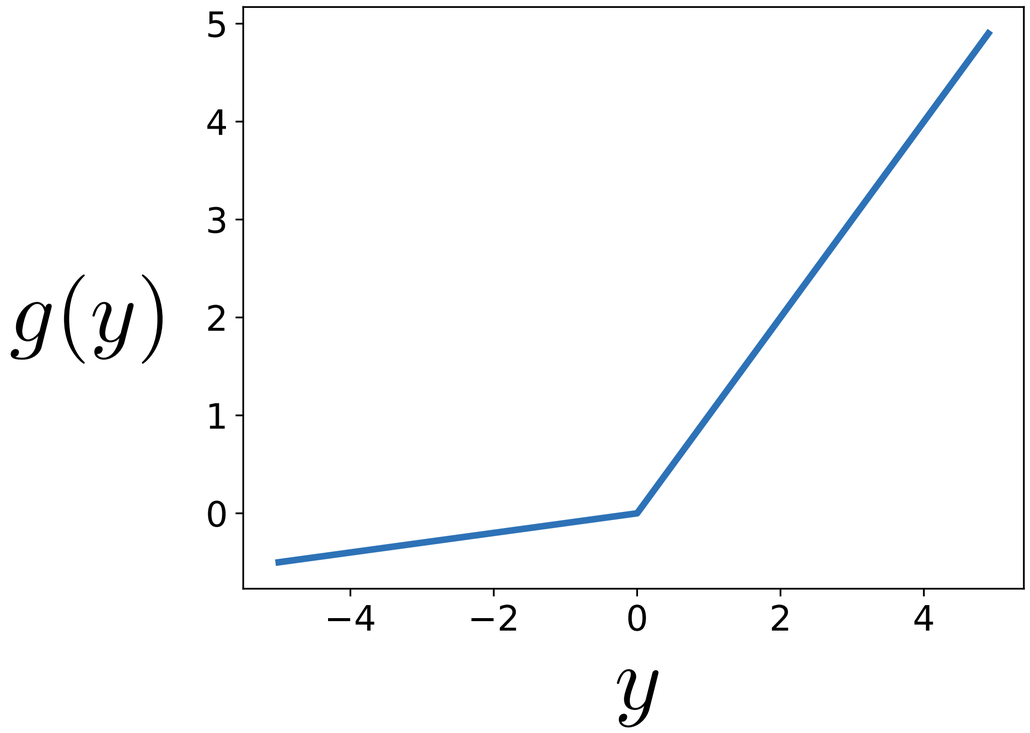
Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

- where α is small (e.g. 0.02)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} -a, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$
- Has non-zero gradients everywhere (unlike ReLU)

Leaky ReLU

$$g(y) = \begin{cases} \max(0, y), & \text{if} \quad y \geq 0 \\ a \min(0, y), & \text{if} \quad y < 0 \end{cases}$$



Source: Isola, Torralba, Freeman

Computation has a simple form

$$\mathbf{y} = \mathbf{W}^{(n)} g(\mathbf{W}^{(n-1)} \dots g(\mathbf{W}^{(3)} g(\mathbf{W}^{(2)} (g(\mathbf{W}^{(1)} \mathbf{x})))))$$

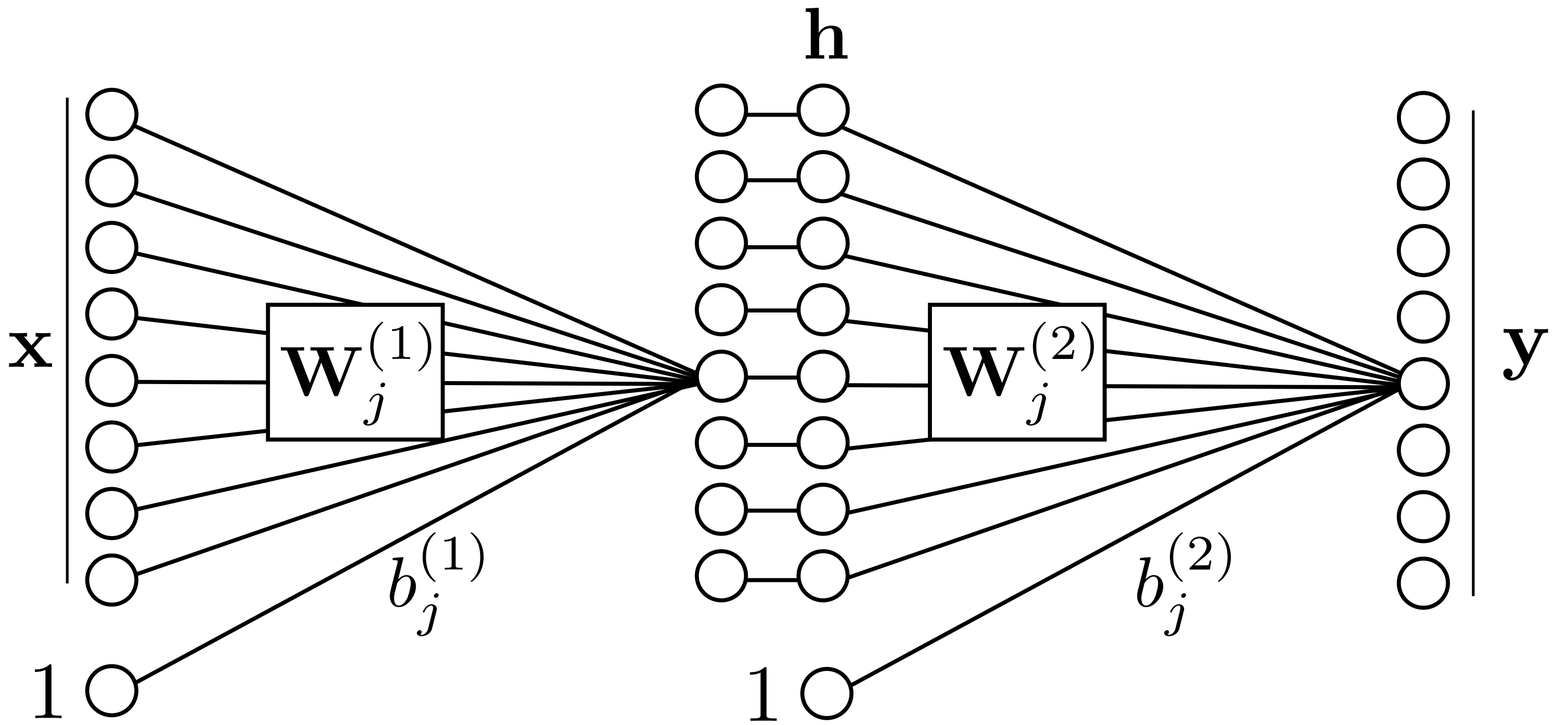
- Composition of linear functions with nonlinearities in between
- E.g. matrix multiplications with ReLU, $\mathbf{max(0,\ x)}$ afterwards
- Do a matrix multiplication, set all negative values to 0, repeat

Stacking layers

Input representation

Intermediate representation

Output representation



$\mathbf{h}$ = **“hidden units”**

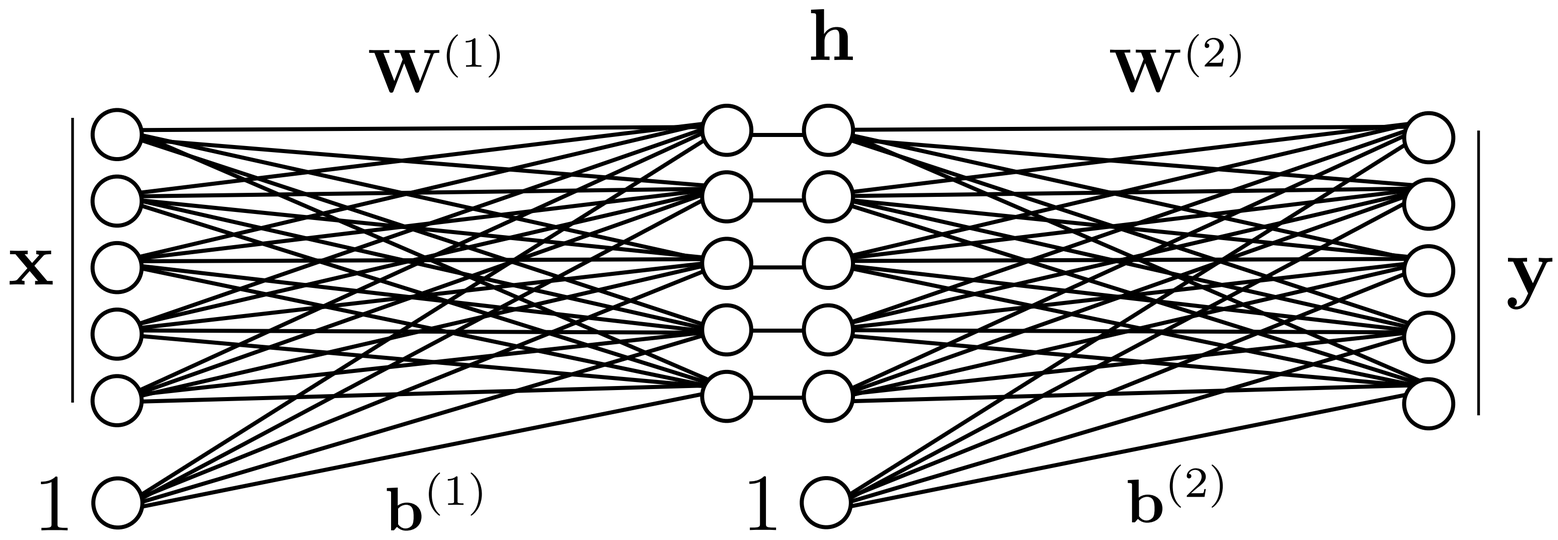
Source: Isola, Torralba, Freeman

Stacking layers

Input representation

Intermediate representation

Output representation



$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

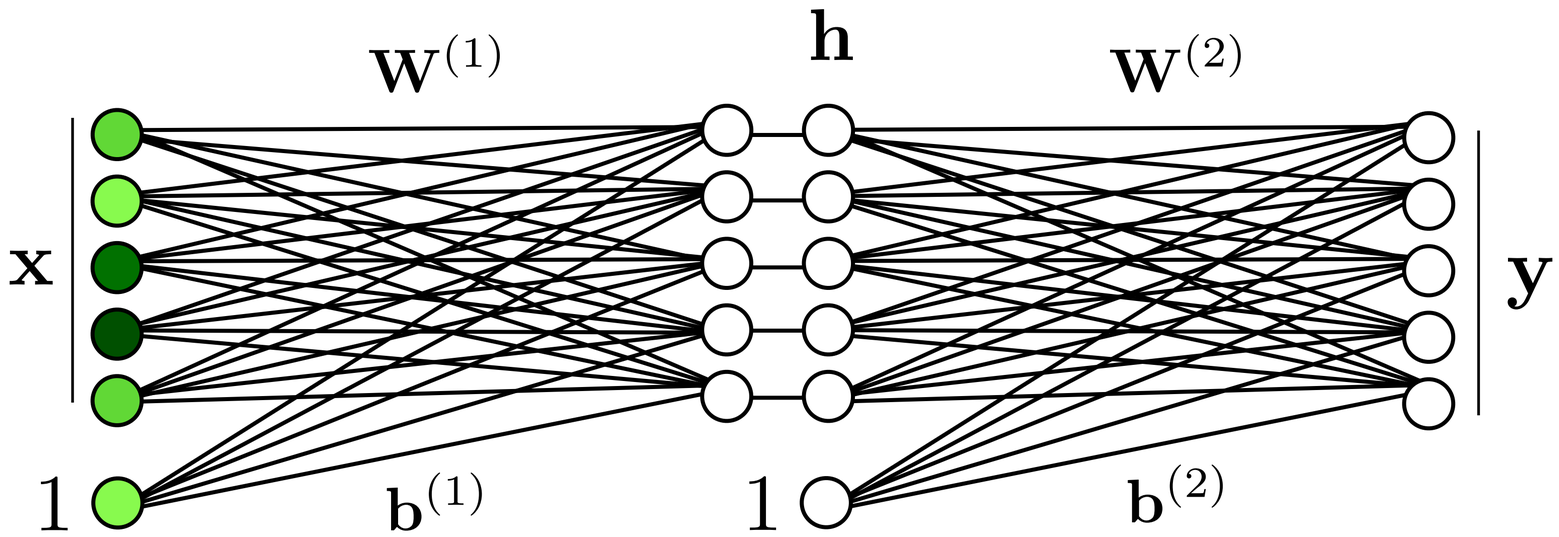
Source: Isola, Torralba, Freeman

Stacking layers

Input representation

Intermediate representation

Output representation



positive

negative

$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

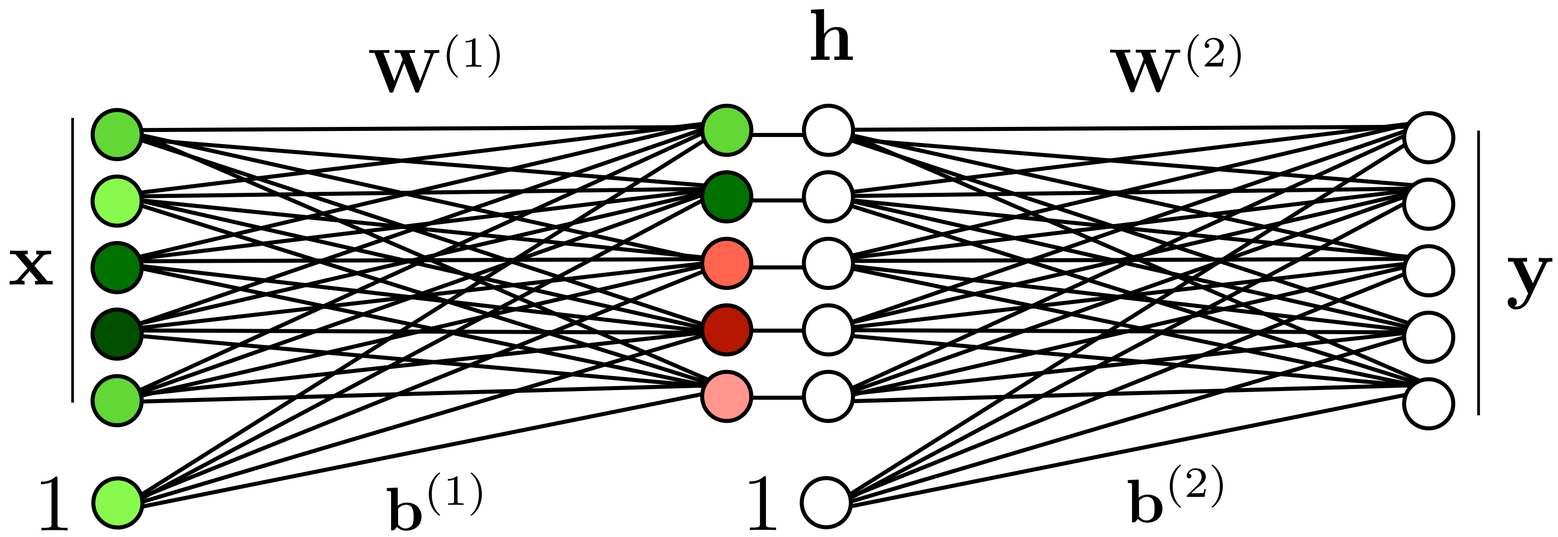
Source: Isola, Torralba, Freeman

Stacking layers

Input representation

Intermediate representation

Output representation



positive

negative

$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

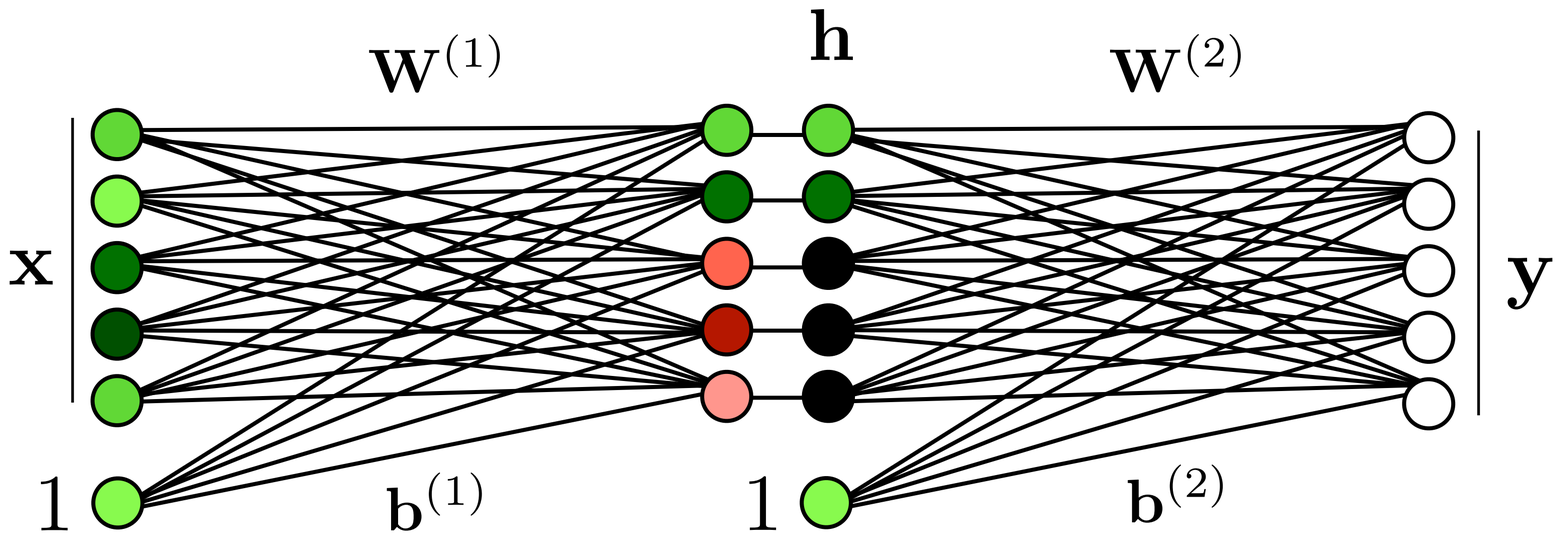
Source: Isola, Torralba, Freeman

Stacking layers

Input representation

Intermediate representation

Output representation



positive

negative

$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

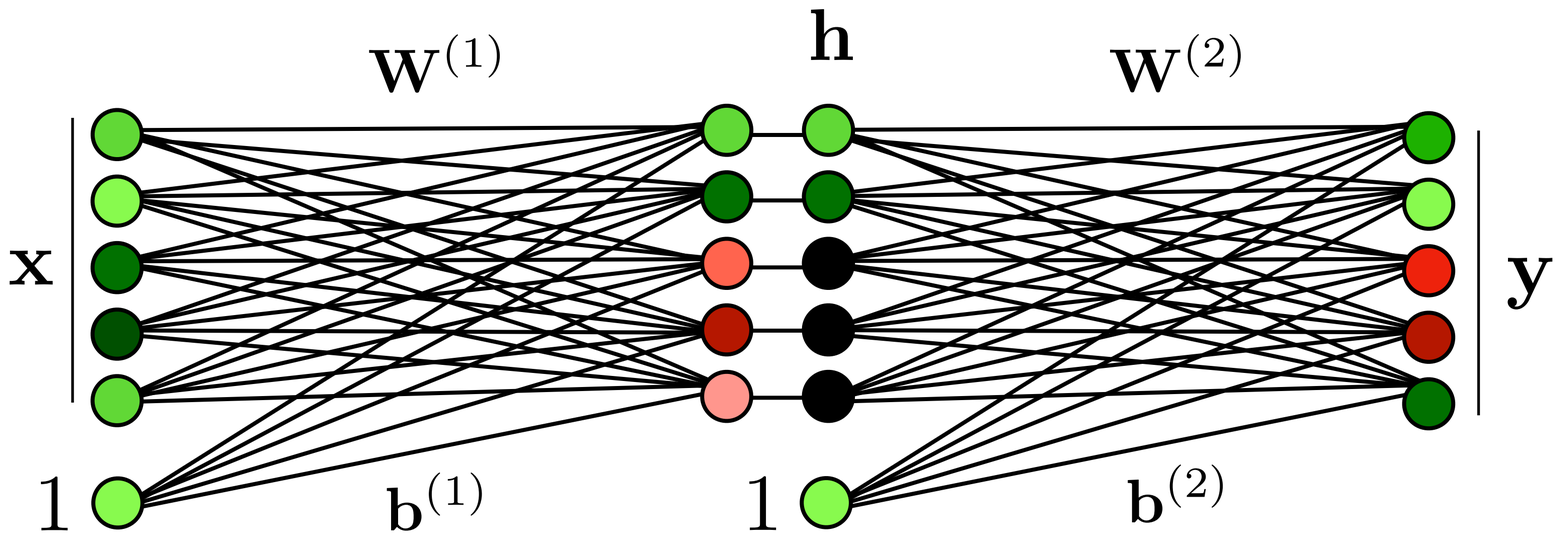
Source: Isola, Torralba, Freeman

Stacking layers

Input representation

Intermediate representation

Output representation



positive

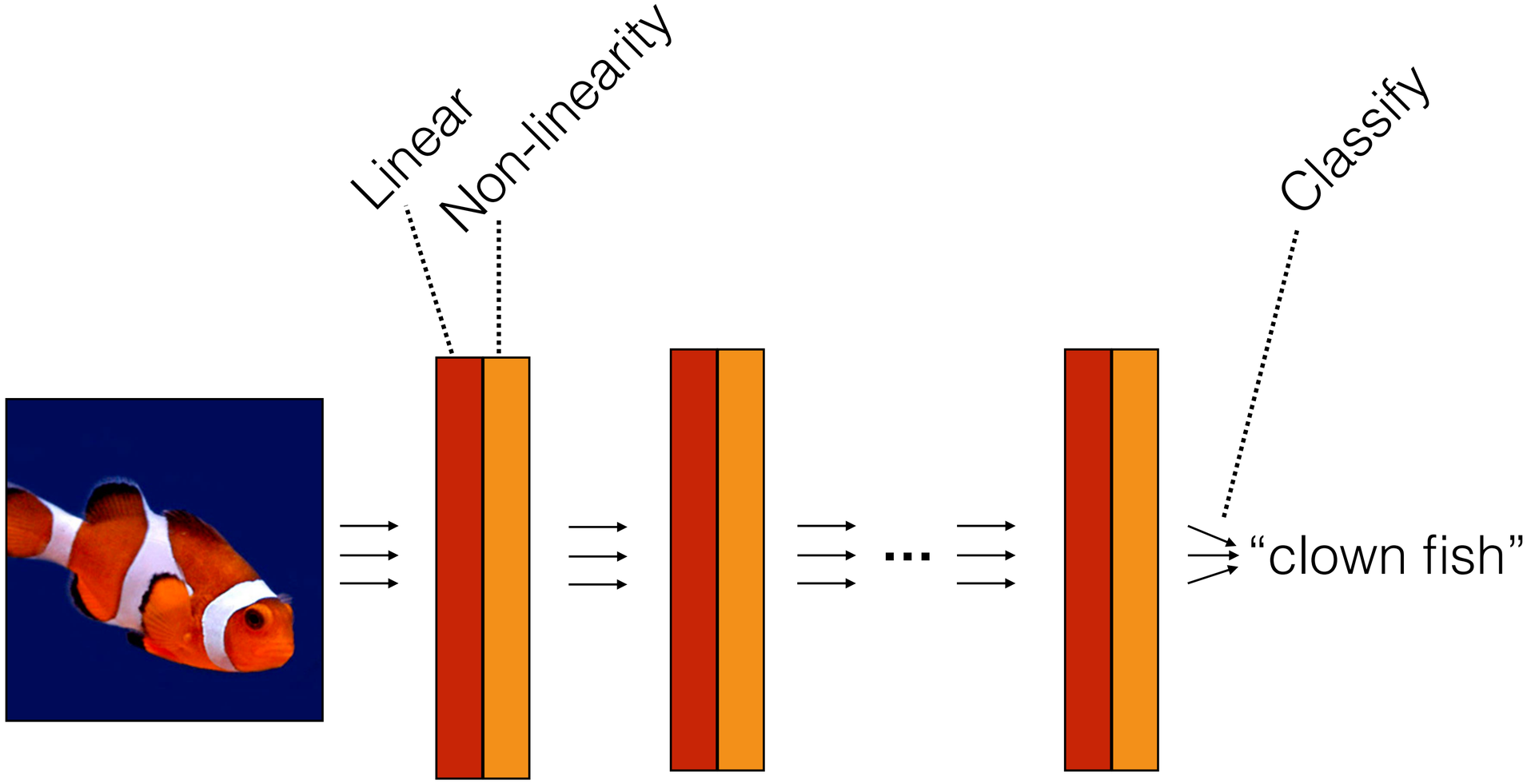
negative

$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Source: Isola, Torralba, Freeman

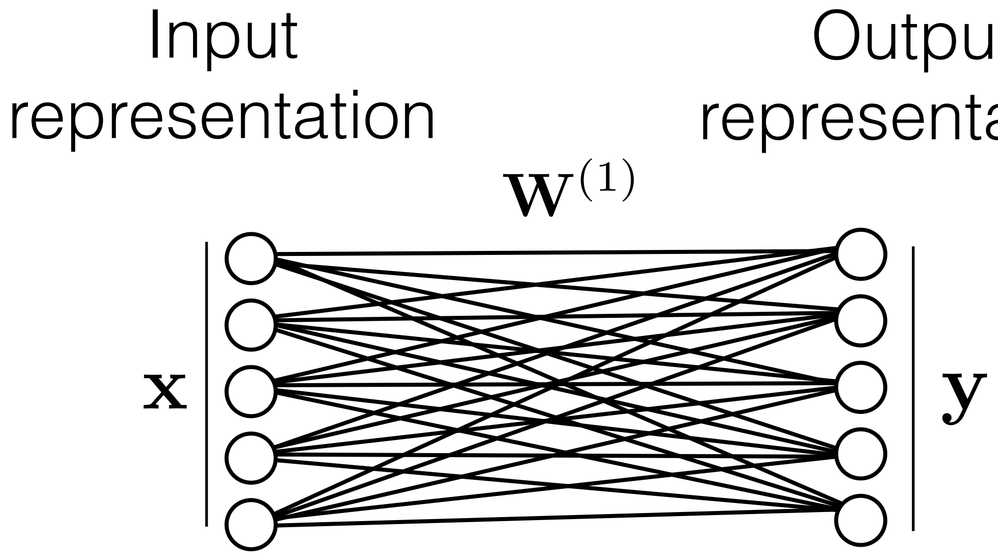
Deep nets



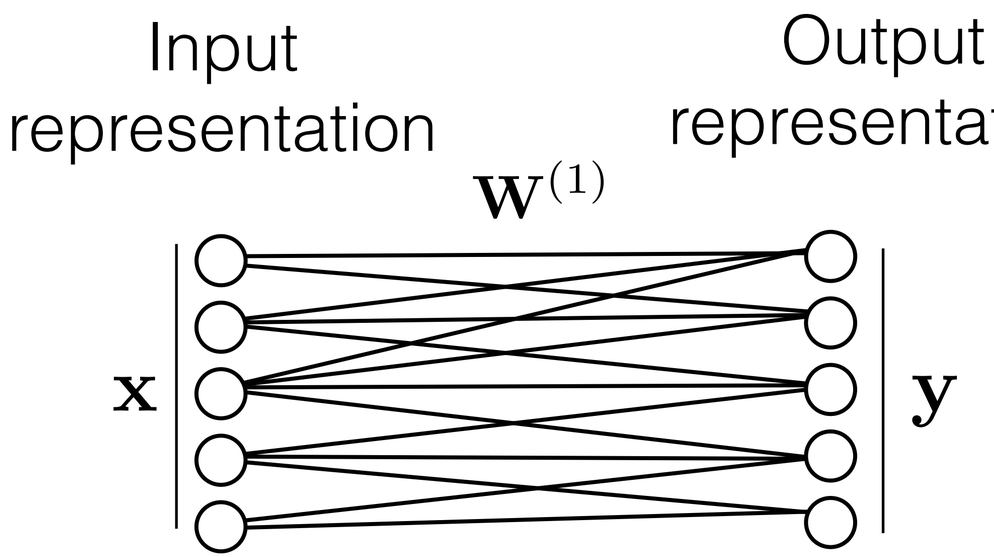
$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

Source: Isola, Torralba, Freeman

Connectivity patterns



Fully connected layer



Locally connected layer (Sparse W)

Source: Isola, Torralba, Freeman

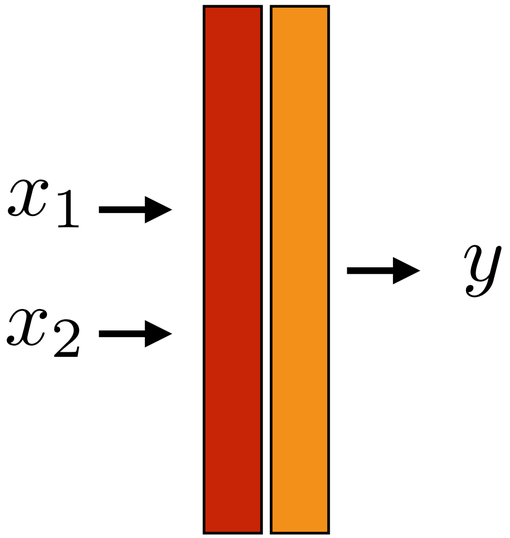
Representational power

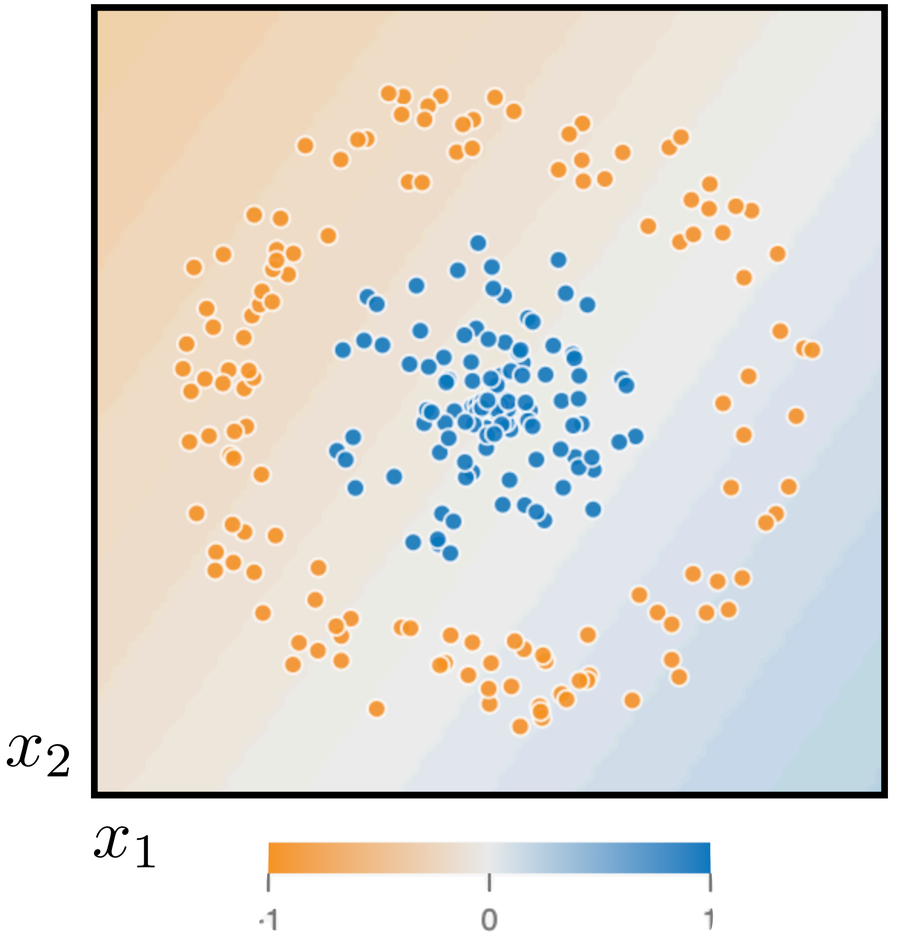
- 1 layer? Linear decision surface.
- 2+ layers? In theory, can represent any function!
  - Simple proof by M. Nielsen
    http://neuralnetworksanddeeplearning.com/chap4.html
- But issue is efficiency: very wide two layers vs narrow deep model? In practice, more layers helps.

Source: Isola, Torralba, Freeman

Example: perceptron

$$\mathbf{y} = \sigma(\mathbf{W}^{(1)}\mathbf{x})$$

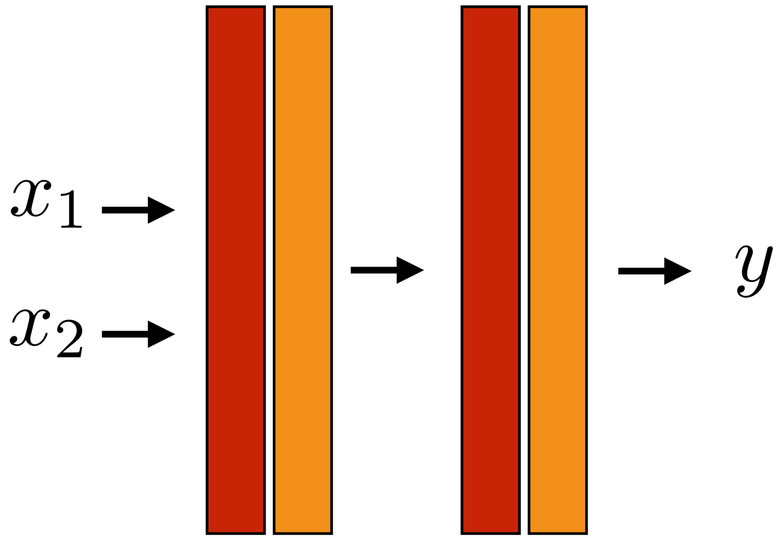


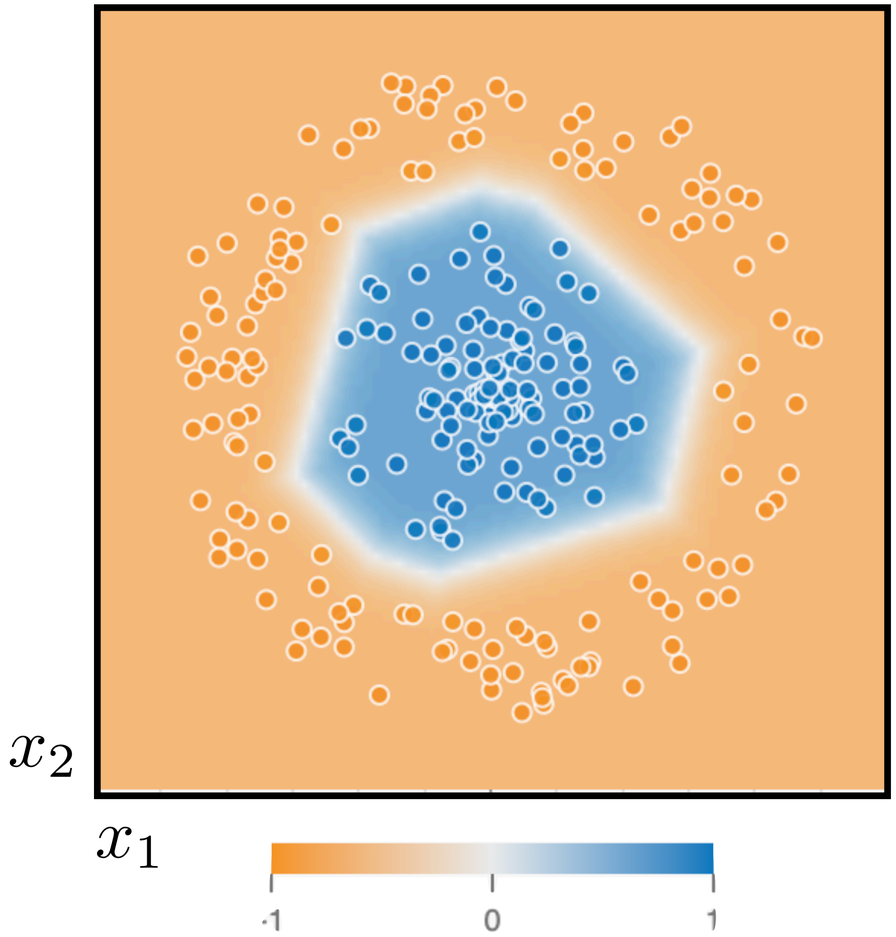


Example source: http://playground.tensorflow.org

Example: multilayer perceptron (MLP)

$$\mathbf{y} = \sigma(\mathbf{W}^{(2)} \max(0,\ \mathbf{W}^{(1)}\mathbf{x}))$$



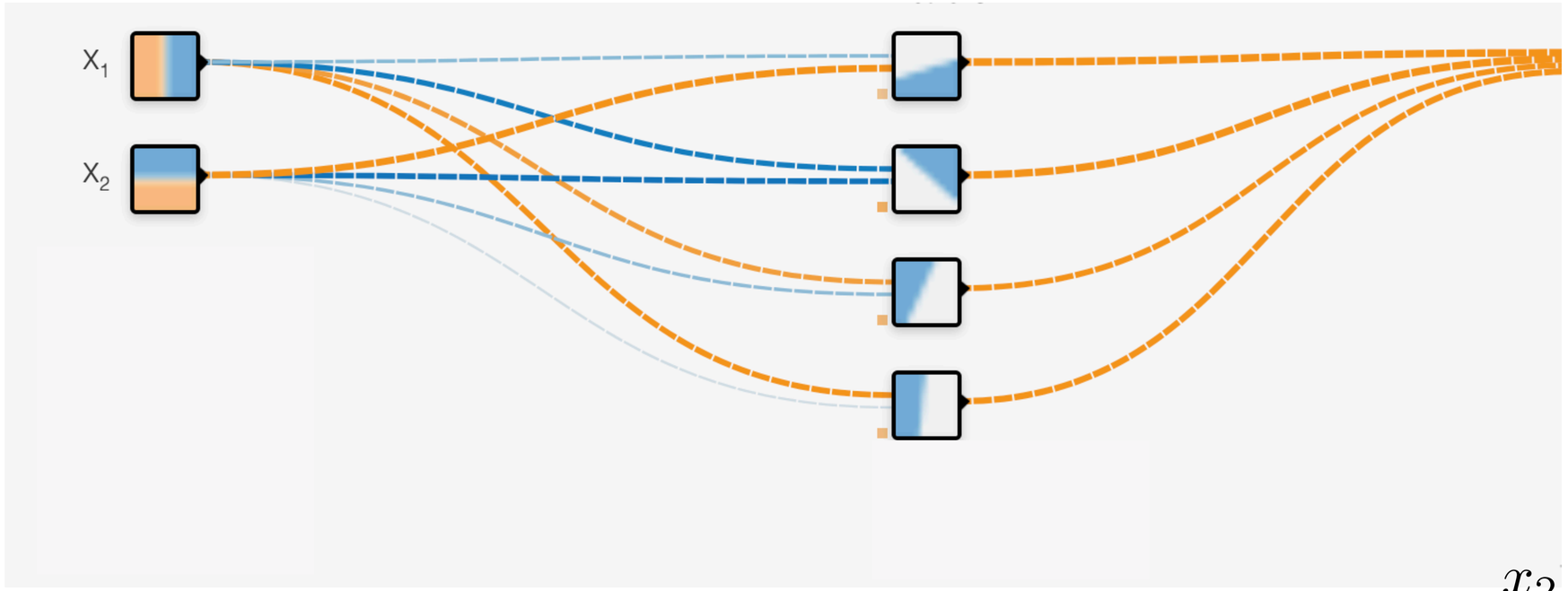


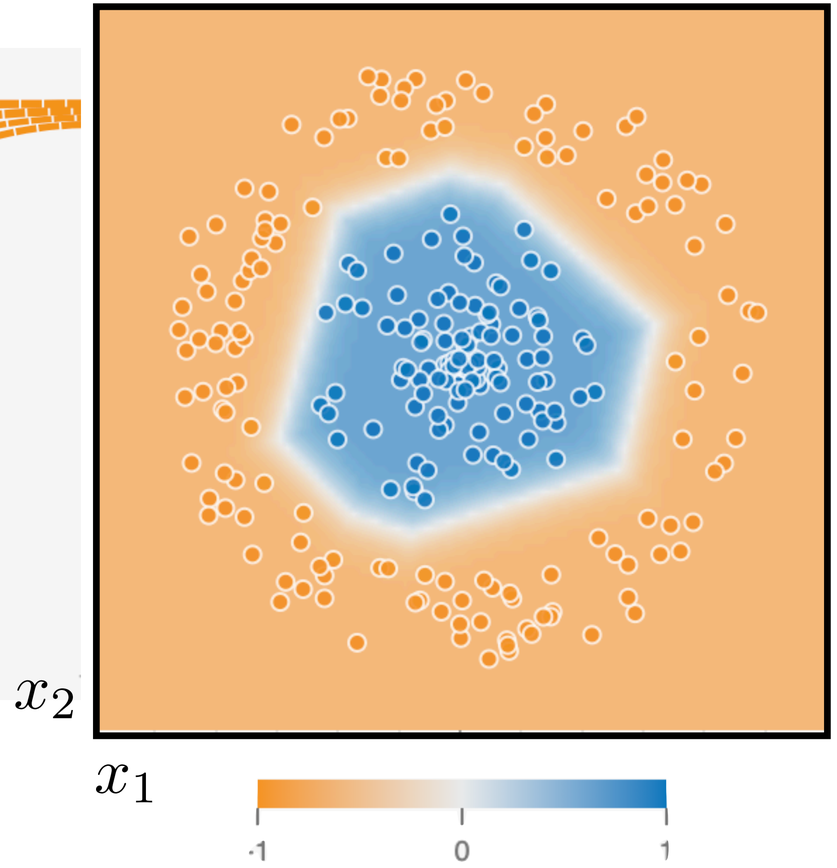
Example source: http://playground.tensorflow.org

Example: multilayer perceptron (MLP)

2D vector

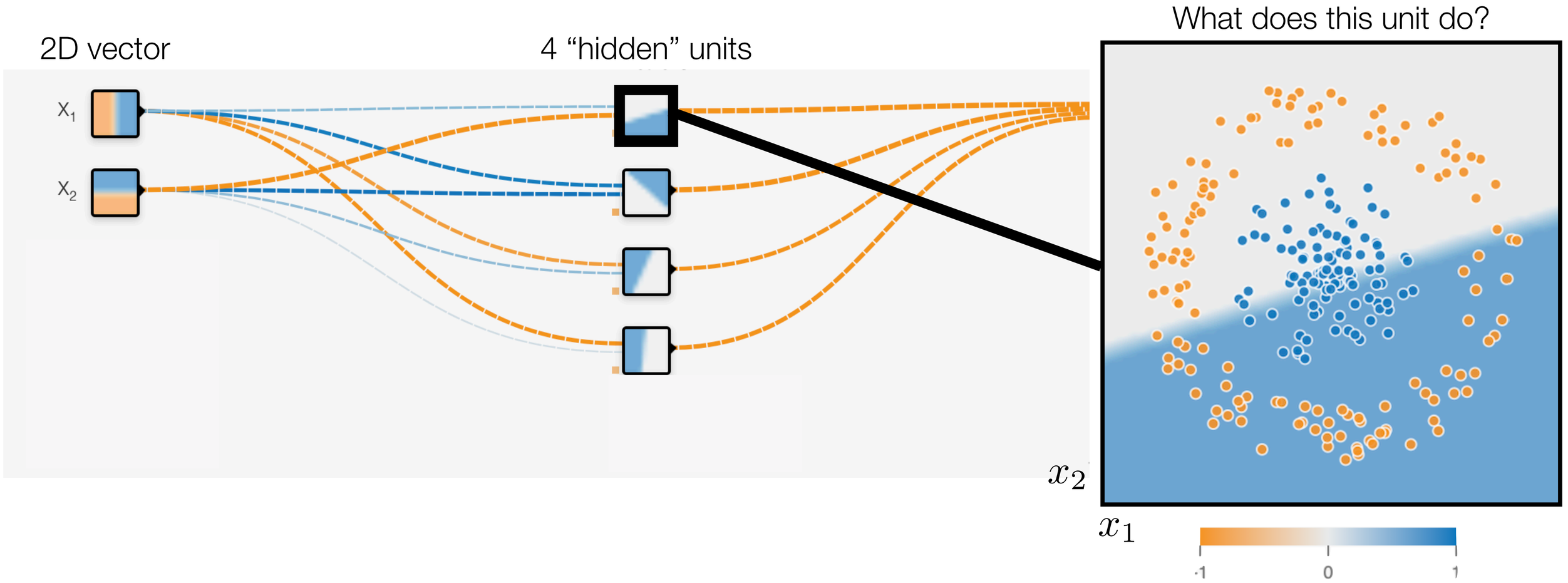
4 “hidden” units





Example source: http://playground.tensorflow.org

Example: multilayer perceptron (MLP)



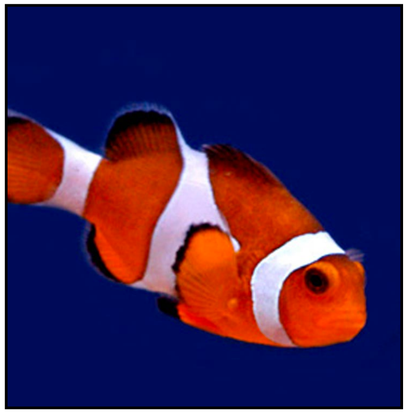
Example source: http://playground.tensorflow.org

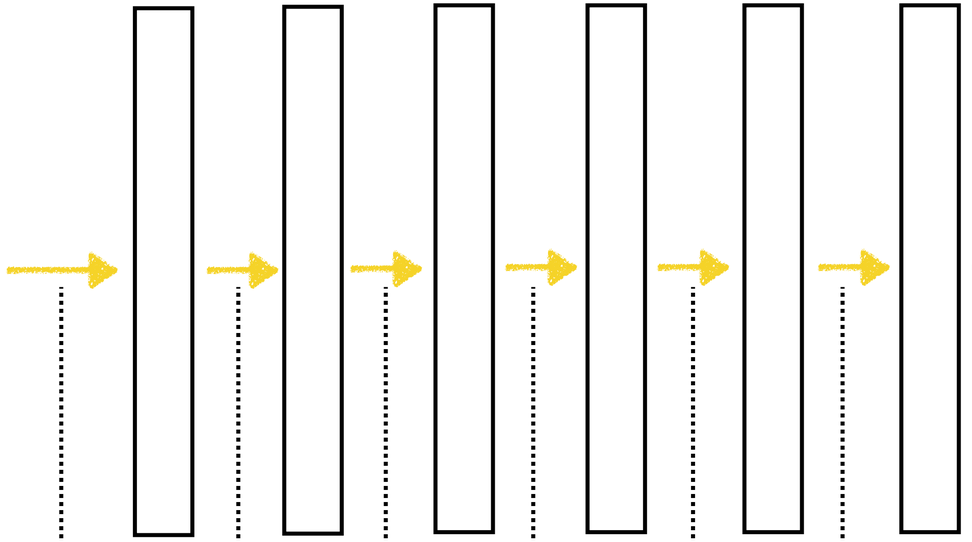
Deep learning

Learned

$\mathbf{y}_1$
"clown fish"

$\mathbf{x}_1$



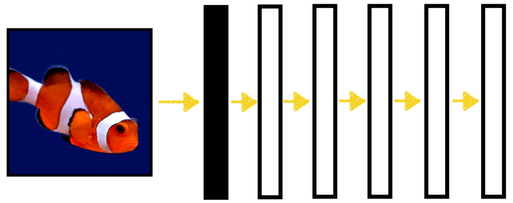


$\mathcal{L}(f_\theta(\mathbf{x}_1), \mathbf{y}_1)$

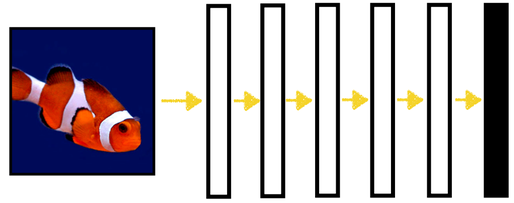
θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

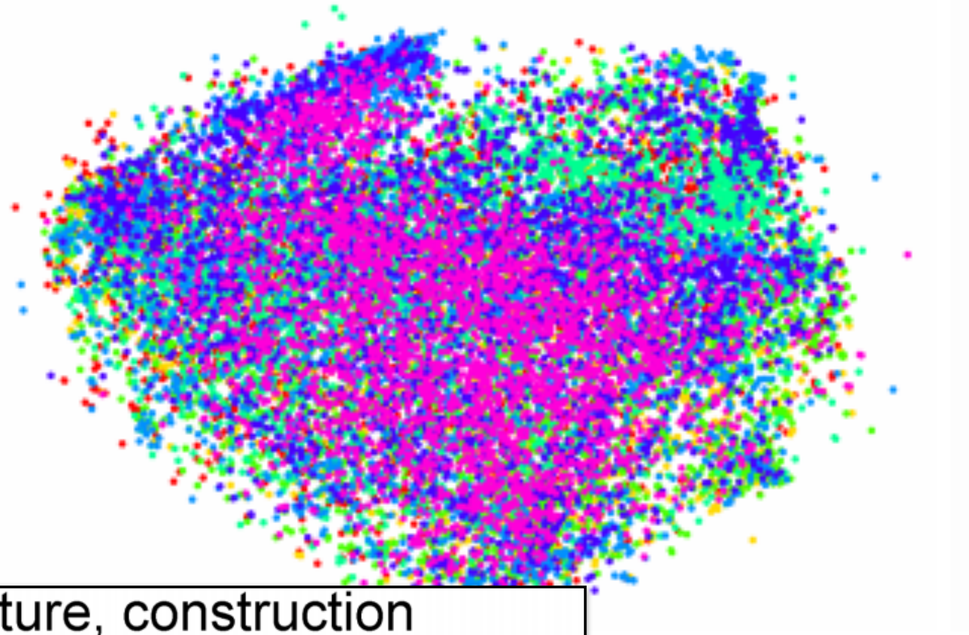
Source: Isola, Torralba, Freeman

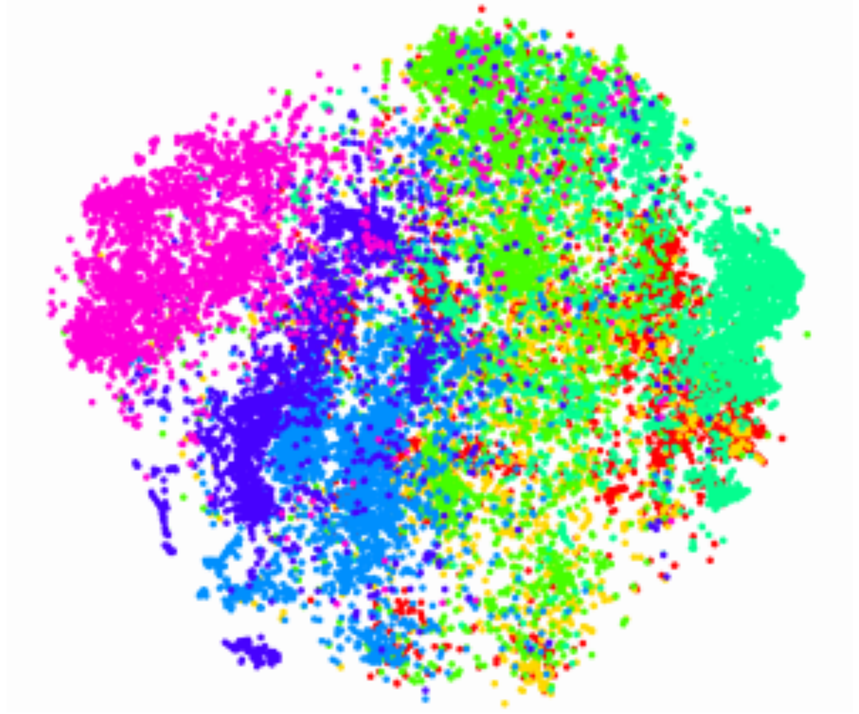


Layer 1 representation



Layer 6 representation





- structure, construction
- covering
- commodity, trade good, good
- conveyance, transport
- invertebrate
- bird
- hunting dog

[DeCAF, Donahue, Jia, et al. 2013]

[Visualization technique : t-sne, van der Maaten & Hinton, 2008]

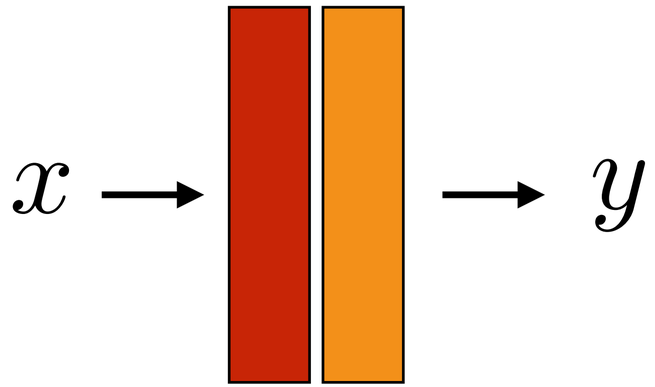
Learning parameters for neural nets

Learning parameters

Squared loss with single-variable network:

$$L = \frac{1}{2}(y - f(x))^2$$

$$L = \frac{1}{2}(y - \sigma(wx + b))^2$$



Example source: Roger Grosse

Learning parameters

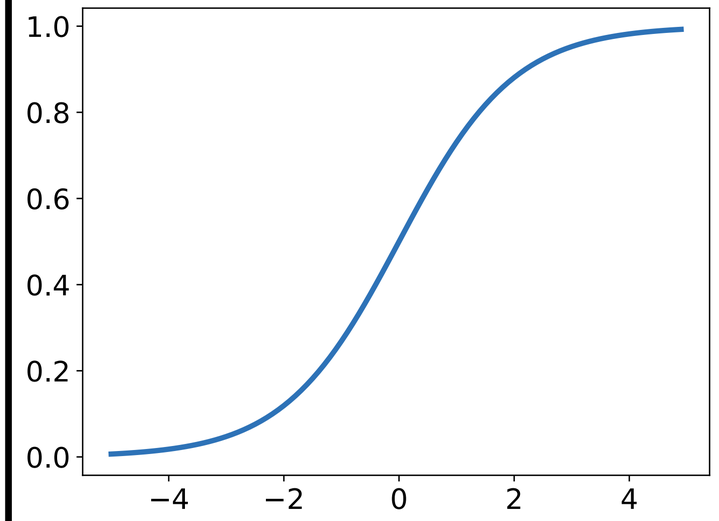
Squared loss with single-variable network:

$$L = \frac{1}{2}(y - f(x))^2$$

$$L = \frac{1}{2}(y - \sigma(wx + b))^2$$

Want: derivatives $\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial b}$

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$



Example source: Roger Grosse

Learning parameters

Writing out the layers explicitly:

$$z = wx + b$$

$$t = \sigma(z)$$

$$L = \frac{1}{2}(y - t)^2$$

How do we do this?
Use the chain rule!

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

Example source: Roger Grosse

Computing derivatives with the chain rule

$$\frac{\partial L}{\partial w} = \frac{\partial}{\partial w}\left[\frac{1}{2}(y - \sigma(wx + b))^2\right]$$

$$= (y - \sigma(wx + b))\frac{\partial}{\partial w}(y - \sigma(wx + b))$$

$$= (y - \sigma(wx + b))\sigma'(wx + b)\frac{\partial}{\partial w}(wx + b)$$

$$= (y - \sigma(wx + b))\sigma'(wx + b)x$$

$$\frac{\partial L}{\partial b} = \frac{\partial}{\partial b}\left[\frac{1}{2}(y - \sigma(wx + b))^2\right]$$

$$= (y - \sigma(wx + b))\frac{\partial}{\partial b}(y - \sigma(wx + b))$$

$$= (y - \sigma(wx + b))\sigma'(wx + b)\frac{\partial}{\partial b}(wx + b)$$

$$= (y - \sigma(wx + b))\sigma'(wx + b)$$

Example source: Roger Grosse

Limitations to this approach

- Inefficient! Lots of redundant computation
- We'll also need to extend this to multivariable functions
- **Later:** backpropagation