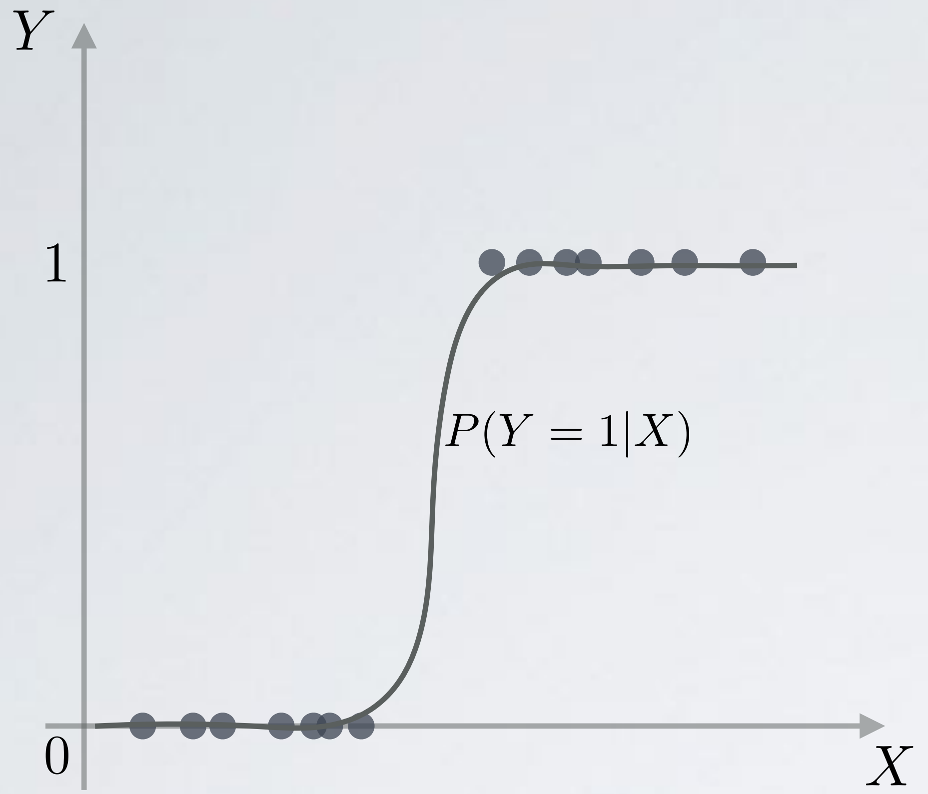
Caltech

MACHINE LEARNING & DATA MINING

CS/CNS/EE 155

Deep Learning
Part I

logistic regression



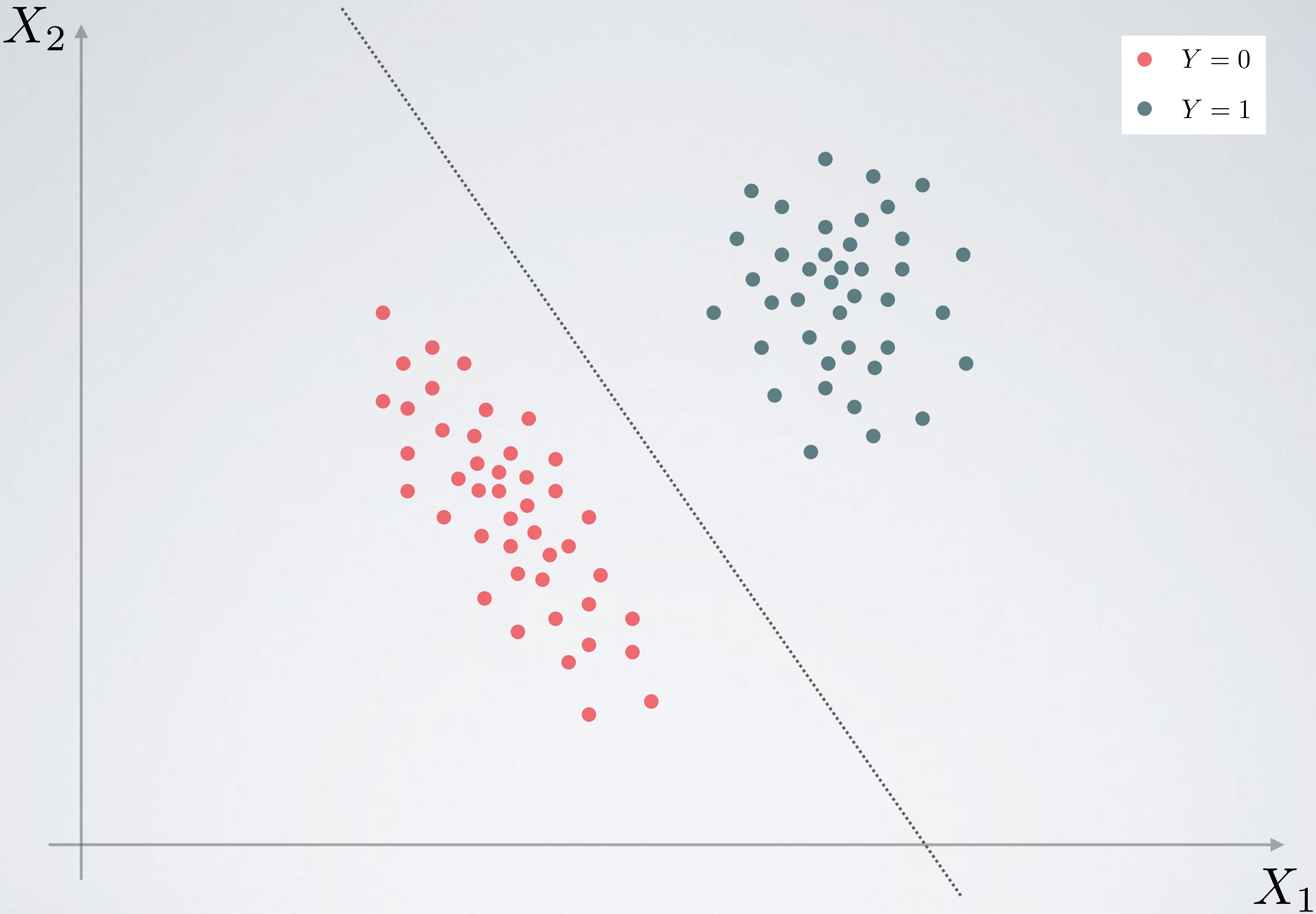
logistic function

$$\boxed{P(Y = 1|X) = \frac{1}{1 + e^{-(\mathbf{w}^\intercal \mathbf{x} + w_0)}}}$$

can fit binary labels $Y \in \{0, 1\}$

$\mathbf{w}^\intercal \mathbf{x} + w_0 = 0$ defines the boundary between the classes

in higher dimensions, this is a hyperplane



additional features in X result in additional weights in $\mathbf{w}$

if $Y \in \{0, 1\}$

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\mathbf{w}^\intercal \mathbf{x} + w_0)}}$$

use *binary cross-entropy* loss function

$$\mathcal{L} = -\sum_{i=1}^{N} \left[y^{(i)} \log(P(y_i = 1|\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - P(y_i = 1|\mathbf{x}^{(i)})) \right]$$

when $y_i = 1$ make the output close to 1

when $y_i = 0$ make the output close to 0

if $Y \in \{-1, 1\}$

$$P(y|\mathbf{x}) = \frac{1}{1 + e^{-y(\mathbf{w}^T\mathbf{x}+w_0)}}$$

use *logistic* loss function

$$\mathcal{L} = \sum_{i=1}^{N} \log(1 + e^{-y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+w_0)})$$

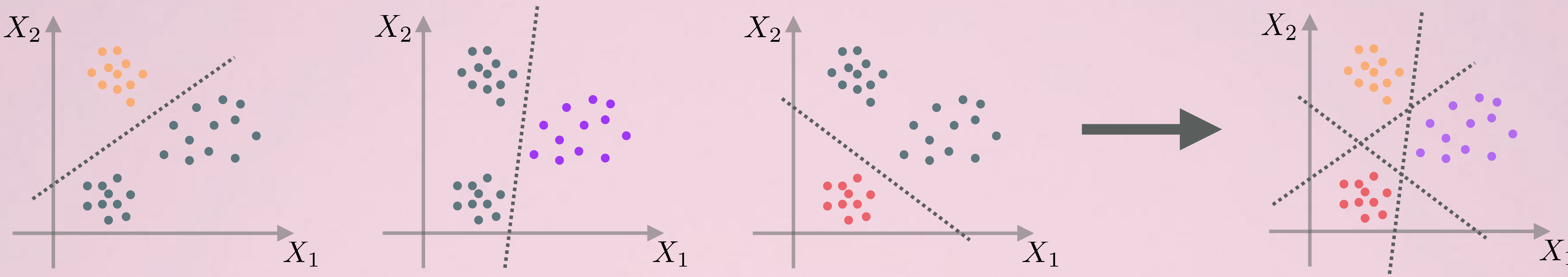
make sure $\mathbf{w}^T\mathbf{x}^{(i)} + w_0$ and $y^{(i)}$
have the same sign, and
$\mathbf{w}^T\mathbf{x}^{(i)} + w_0$ is large in magnitude

how do we extend logistic regression to handle multiple classes?

$$y \in \{1, \ldots, K\}$$

approach I

split the points into groups of one vs. rest, train model on each split

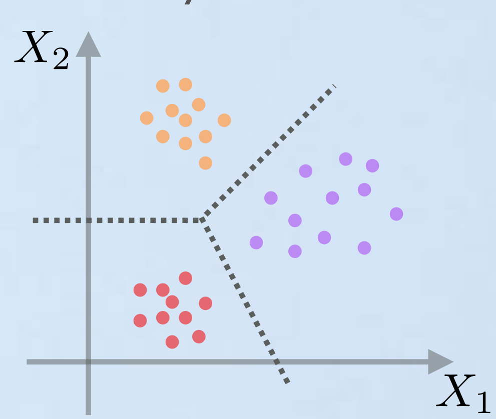


approach II

train one model on all data classes simultaneously

$$P(Y = k|X) = \frac{e^{(\mathbf{w}_k^\intercal \mathbf{x} + w_{0,k})}}{\sum_{k'=1}^{K} e^{(\mathbf{w}_{k'}^\intercal \mathbf{x} + w_{0,k'})}}$$

softmax function



multi-class logistic regression $$P(Y=k|X) = \frac{e^{(\mathbf{w}_k^\intercal \mathbf{x} + w_{0,k})}}{\sum_{k'=1}^{K} e^{(\mathbf{w}_{k'}^\intercal \mathbf{x} + w_{0,k'})}}$$

assume probabilities of the form $$P(Y=k) = \frac{1}{Z} e^{(\mathbf{w}_k^\intercal \mathbf{x} + w_{0,k})}$$

(log linear)

Z is the 'partition function,' which normalizes the probabilities

probabilities must sum to one

$$\sum_{k=1}^{K} P(Y=k) = \sum_{k=1}^{K} \frac{1}{Z} e^{(\mathbf{w}_k^\intercal \mathbf{x} + w_{0,k})} = 1$$

therefore $$Z = \sum_{k=1}^{K} e^{(\mathbf{w}_k^\intercal \mathbf{x} + w_{0,k})}$$

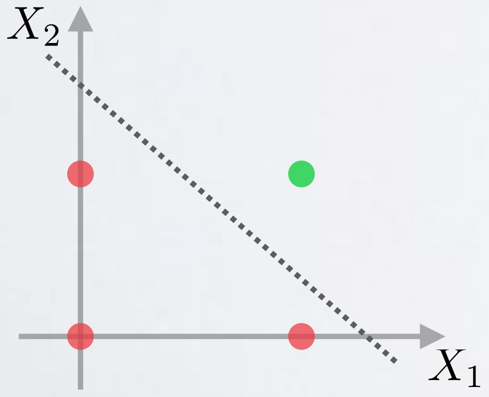
logistic regression is a <u>linear</u> classifier

linear scoring function is passed through the non-linear logistic function to give a probability output

often works well for simple data distributions

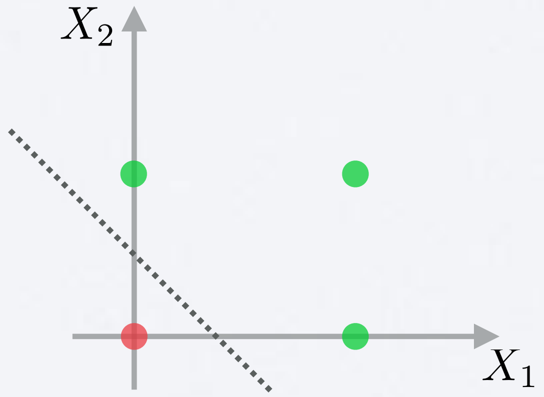
breaks down when confronted with data distributions that are not *linearly separable*

AND



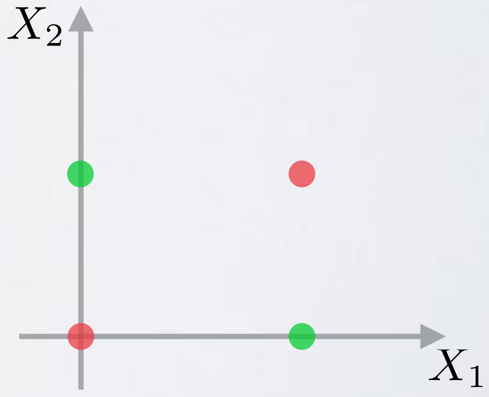
linearly separable

OR



linearly separable

XOR



not linearly separable

to tackle non-linear data distributions with a linear approach, we need to turn it into a linear problem

use a set of *non-linear features* in which the data are linearly separable

one approach:

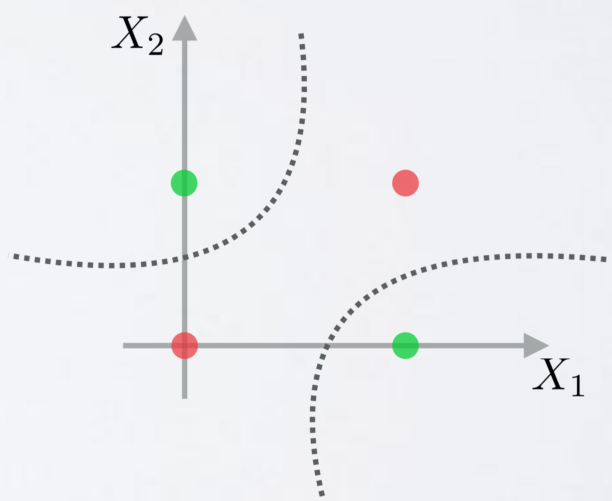
use a set of pre-defined non-linear transformations

XOR

$$X_1, X_2 \rightarrow X_1, X_2, X_1 X_2$$

linear decision boundary

hyperbolic decision boundary



to tackle non-linear data distributions with a linear approach,
we need to turn it into a linear problem

use a set of *non-linear features* in which the data are linearly separable

another approach:

logistic regression outputs a non-linear transformation,
use multiple stages of logistic regression

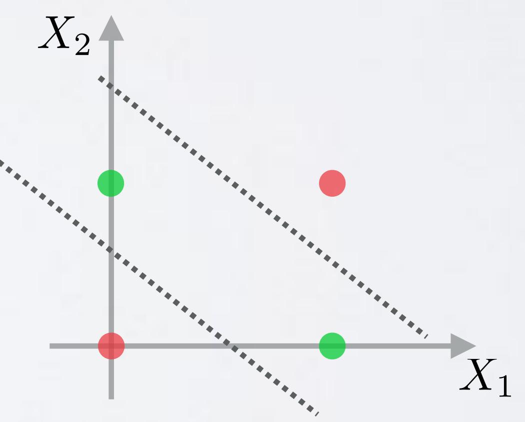
$$X_1, X_2 \rightarrow X_1 \wedge X_2, X_1 \vee X_2$$

linear decision boundary

multiple linear decision boundaries

XOR: $\neg(X_1 \wedge X_2) \wedge (X_1 \vee X_2)$

XOR



we used multiple stages of linear classifiers to create a a non-linear classifier

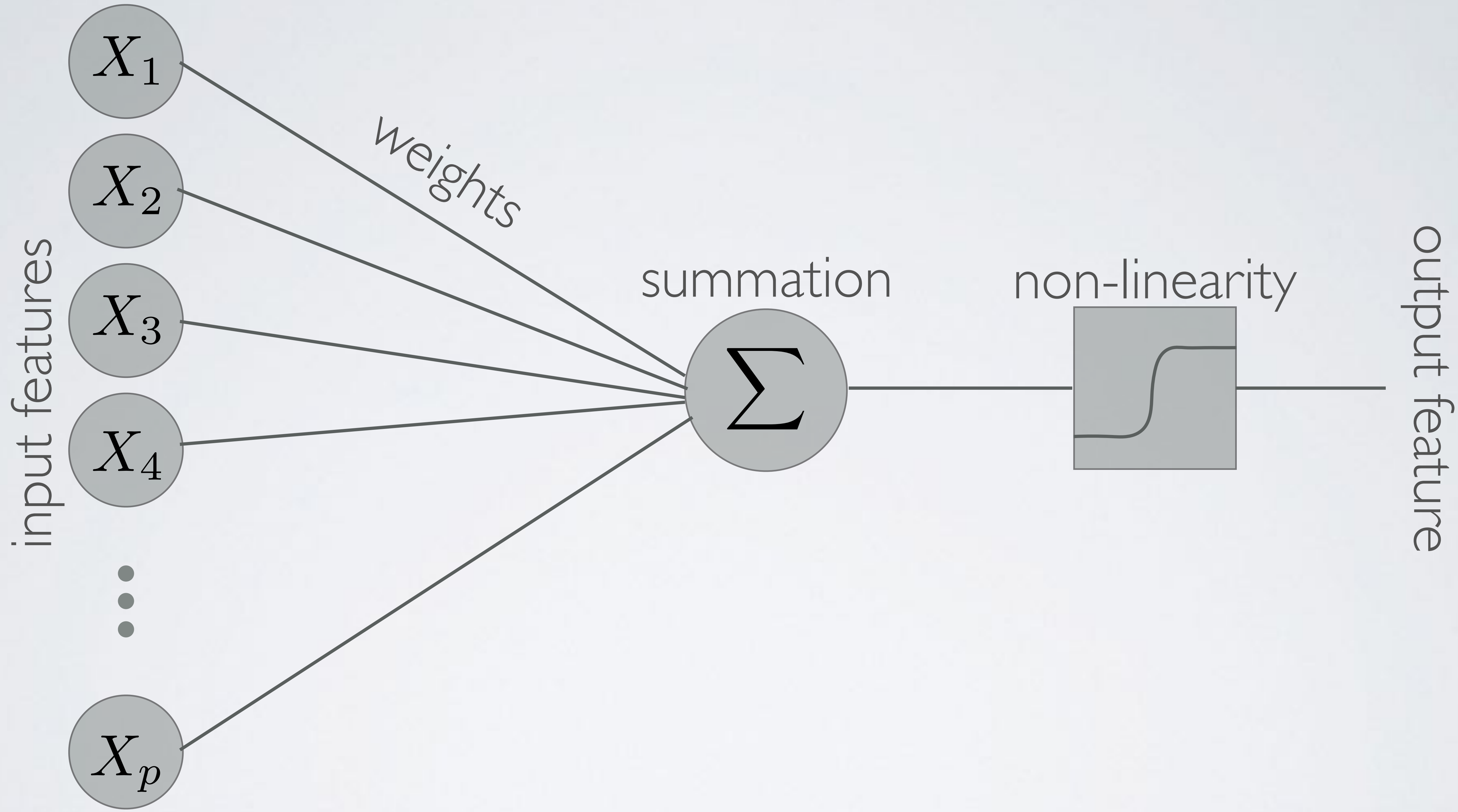
as the number of stages increases, so too does the expressive power of the resulting non-linear classifier

depth: the number of stages of processing

the point of *deep* learning

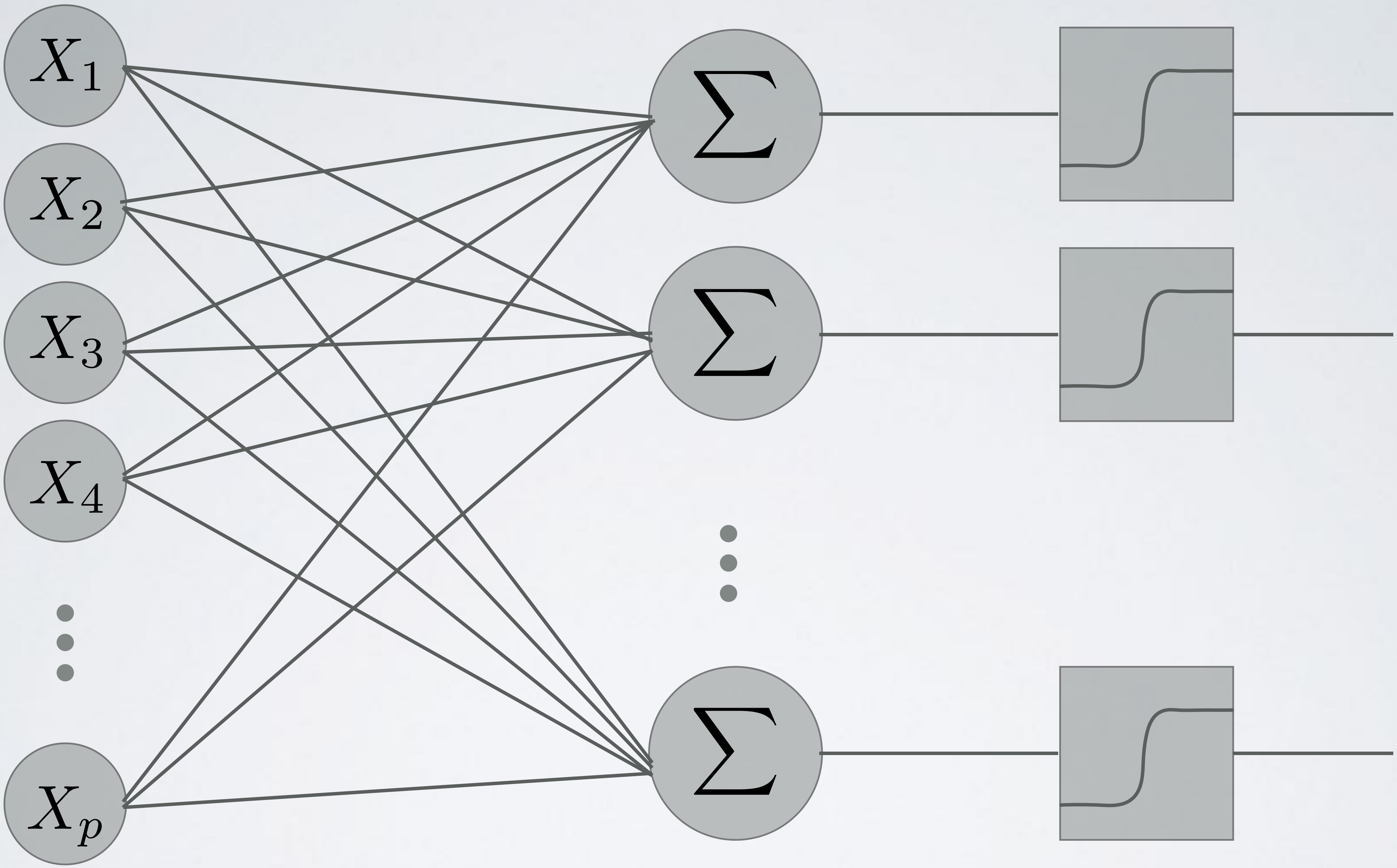
with enough stages of linear/non-linear operations (depth),
we can learn a good set of non-linear features
to linearize any non-linear problem

basic operation: logistic regression



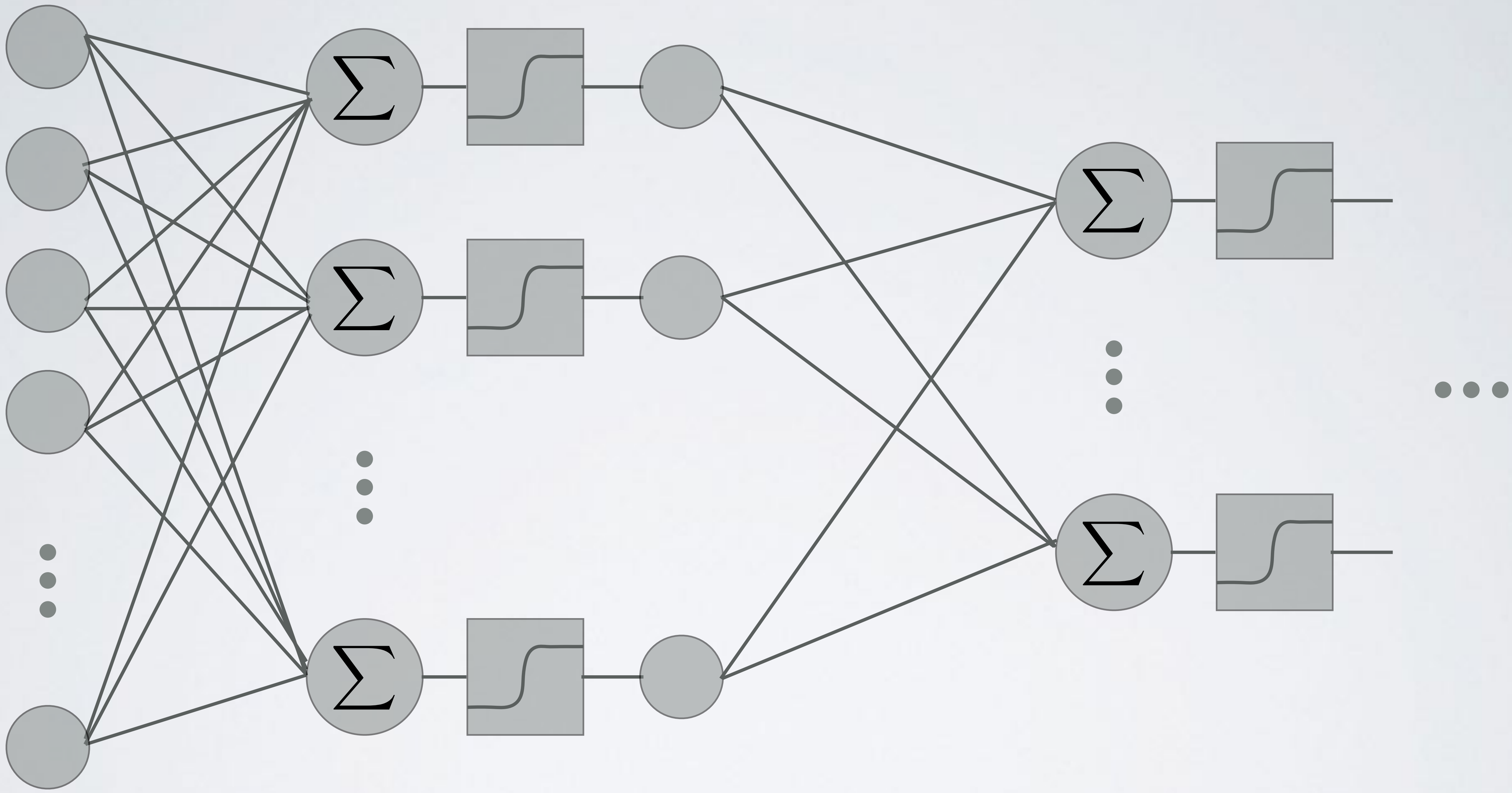
artificial neuron

multiple operations



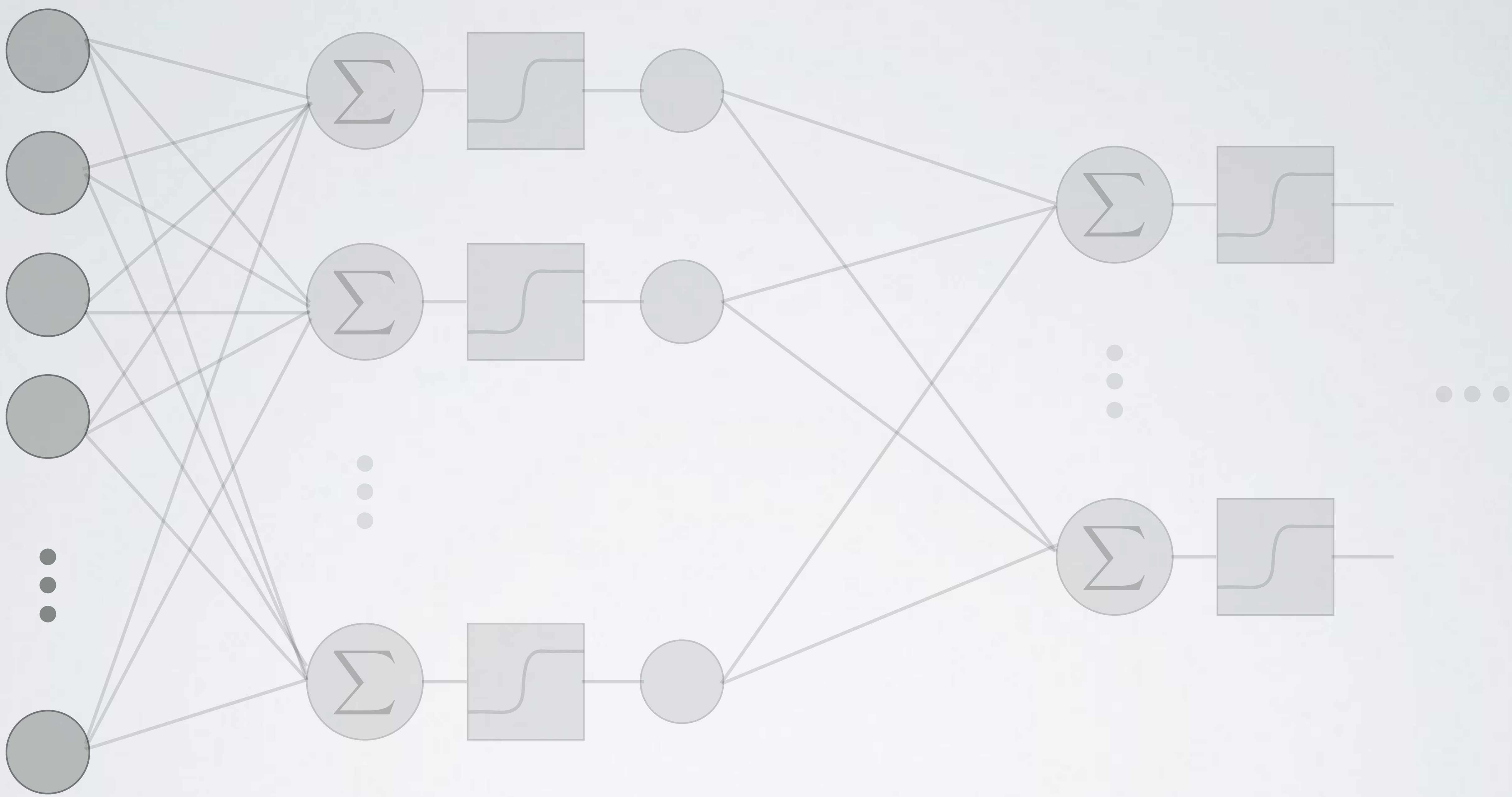
layer of artificial neurons

multiple stages of operations



artificial neural network

notation

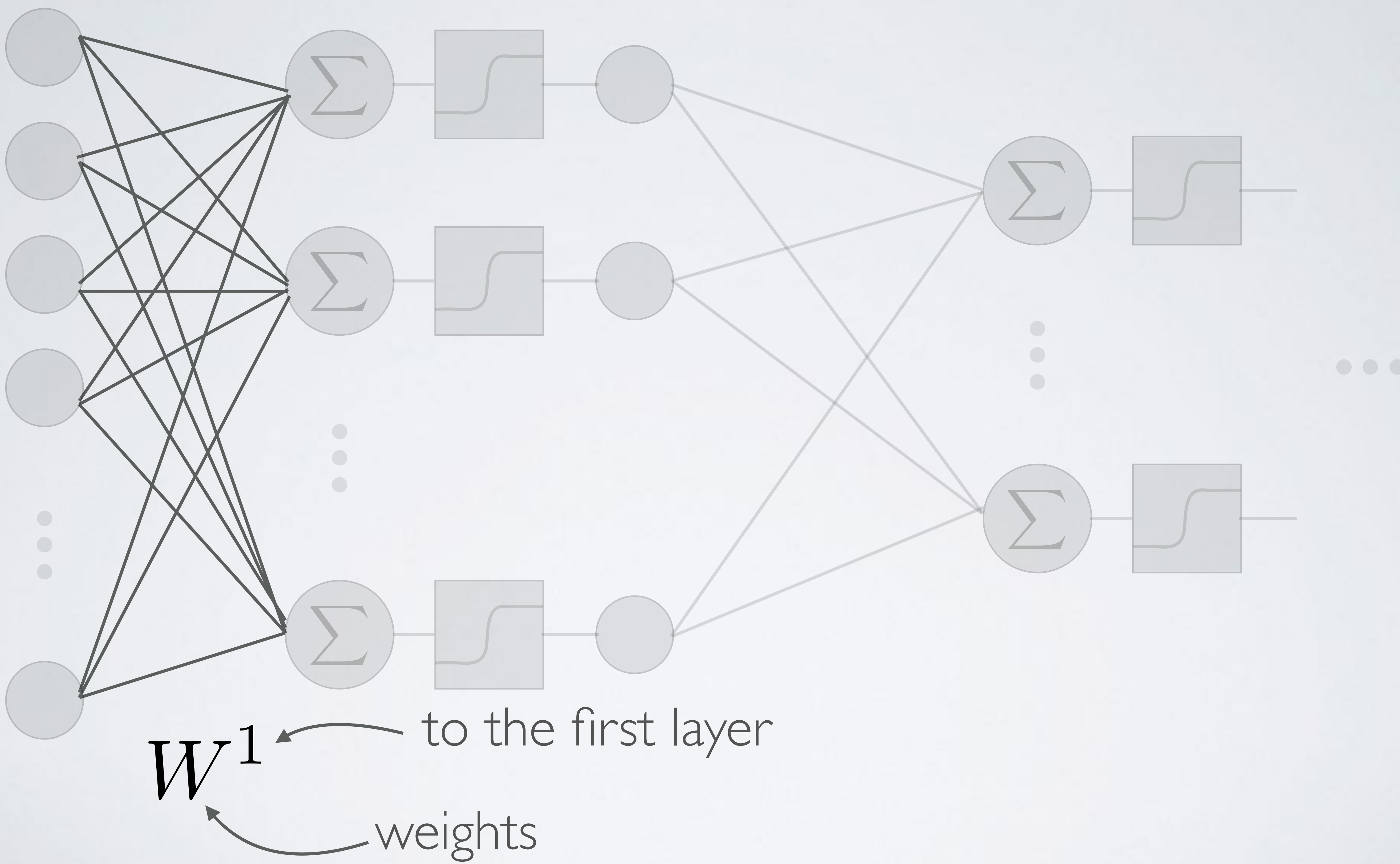


X^0

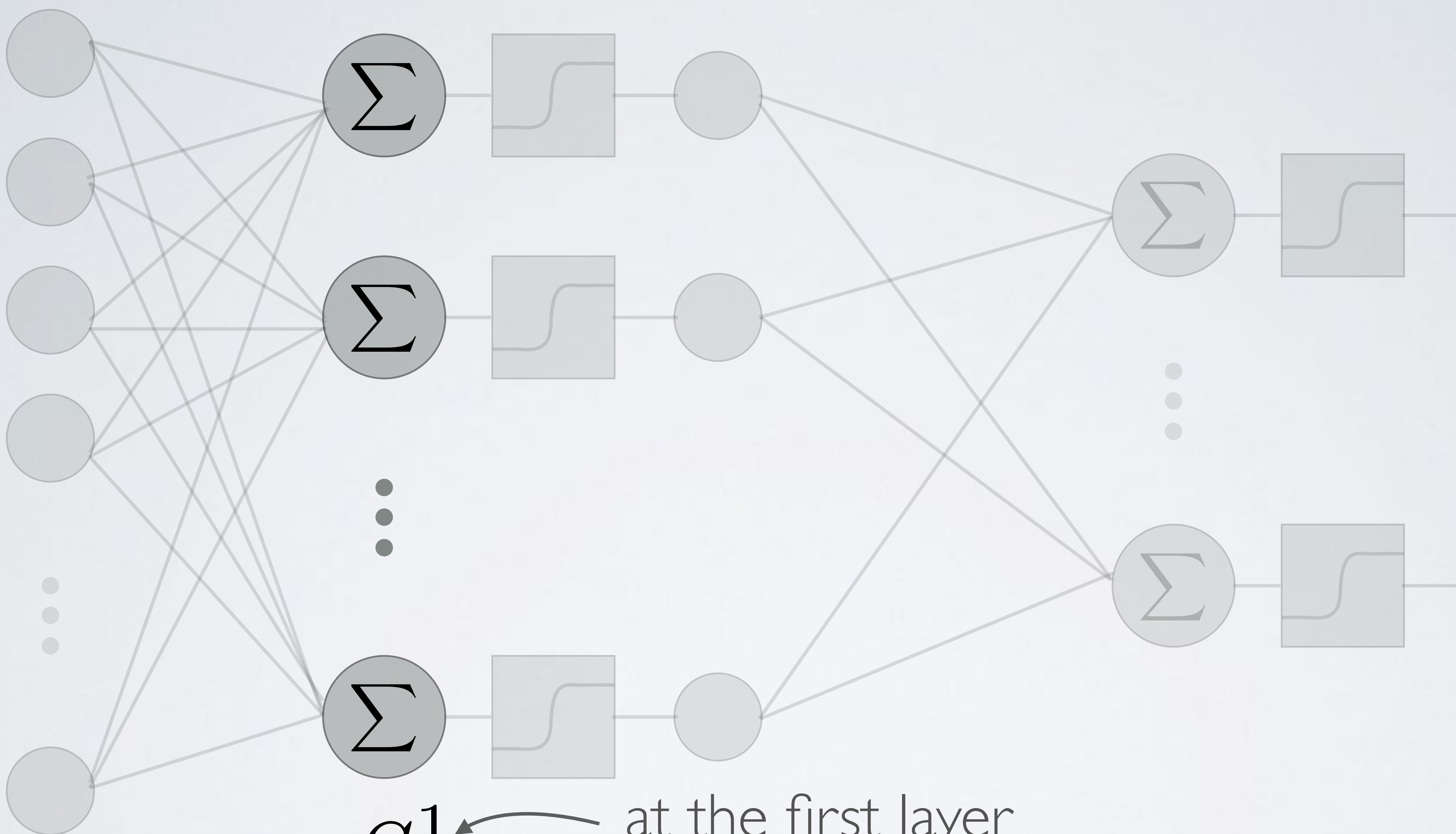
at the input layer

units

notation



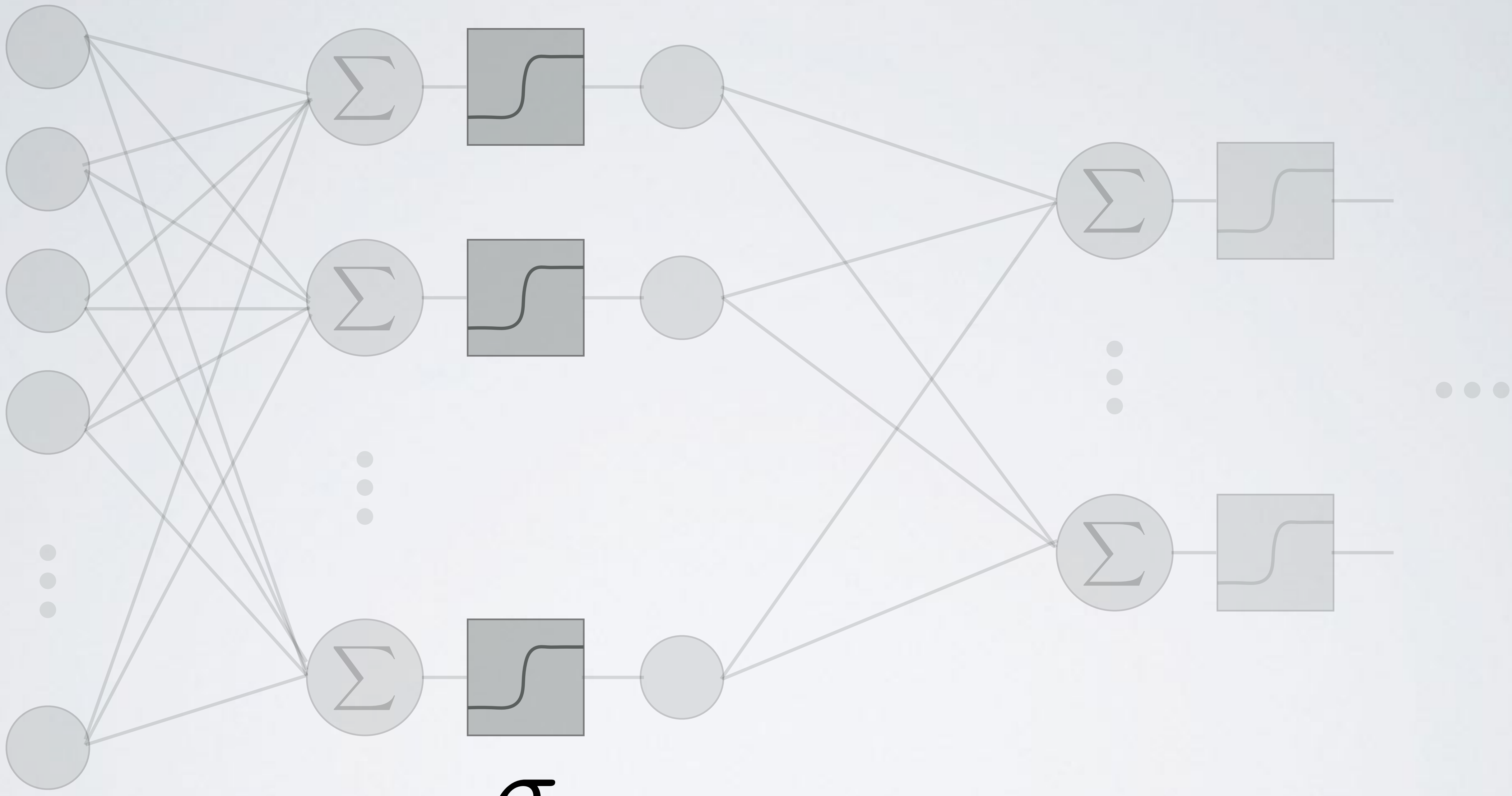
notation



S^1 at the first layer

summations

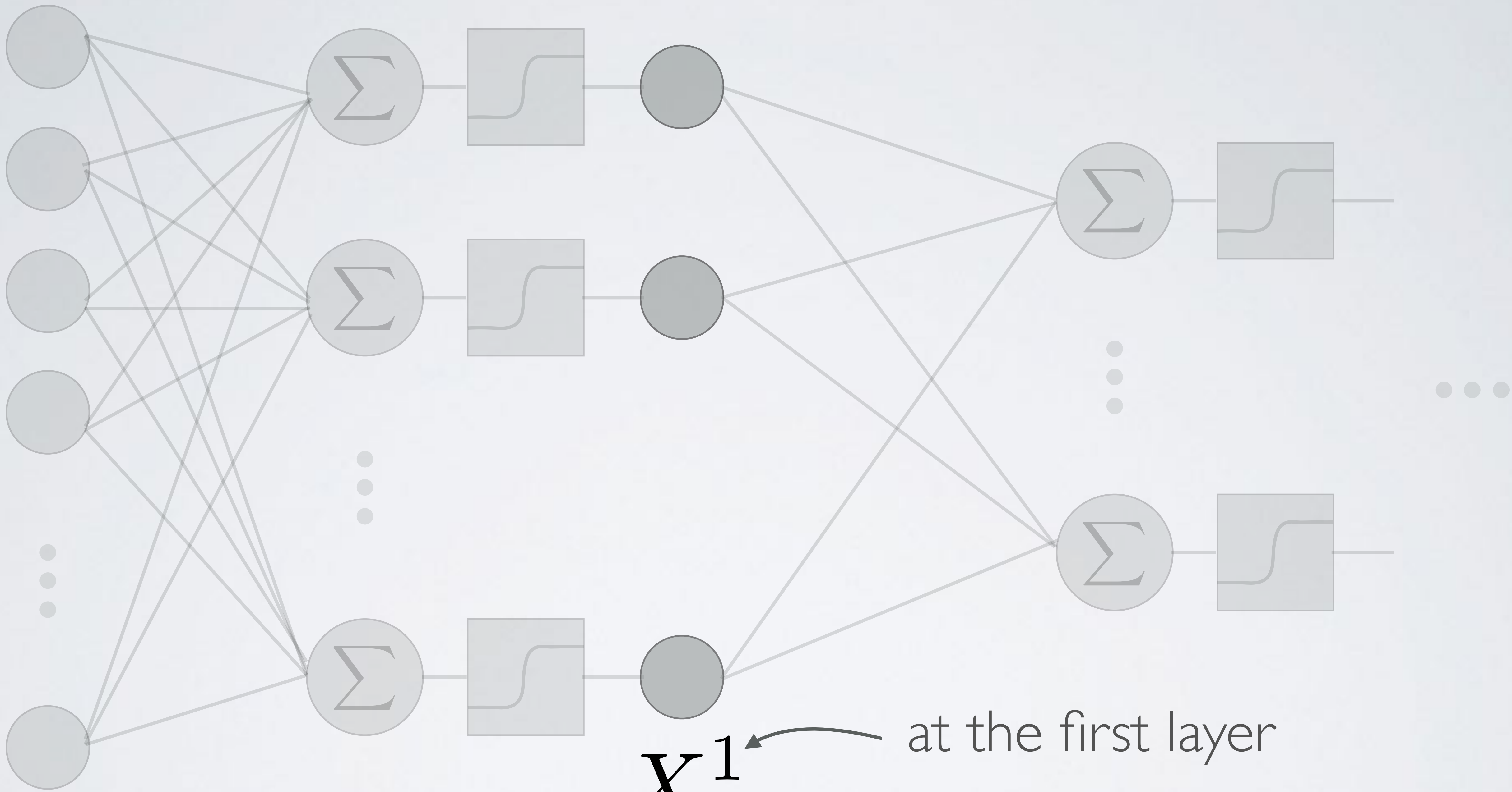
notation



σ

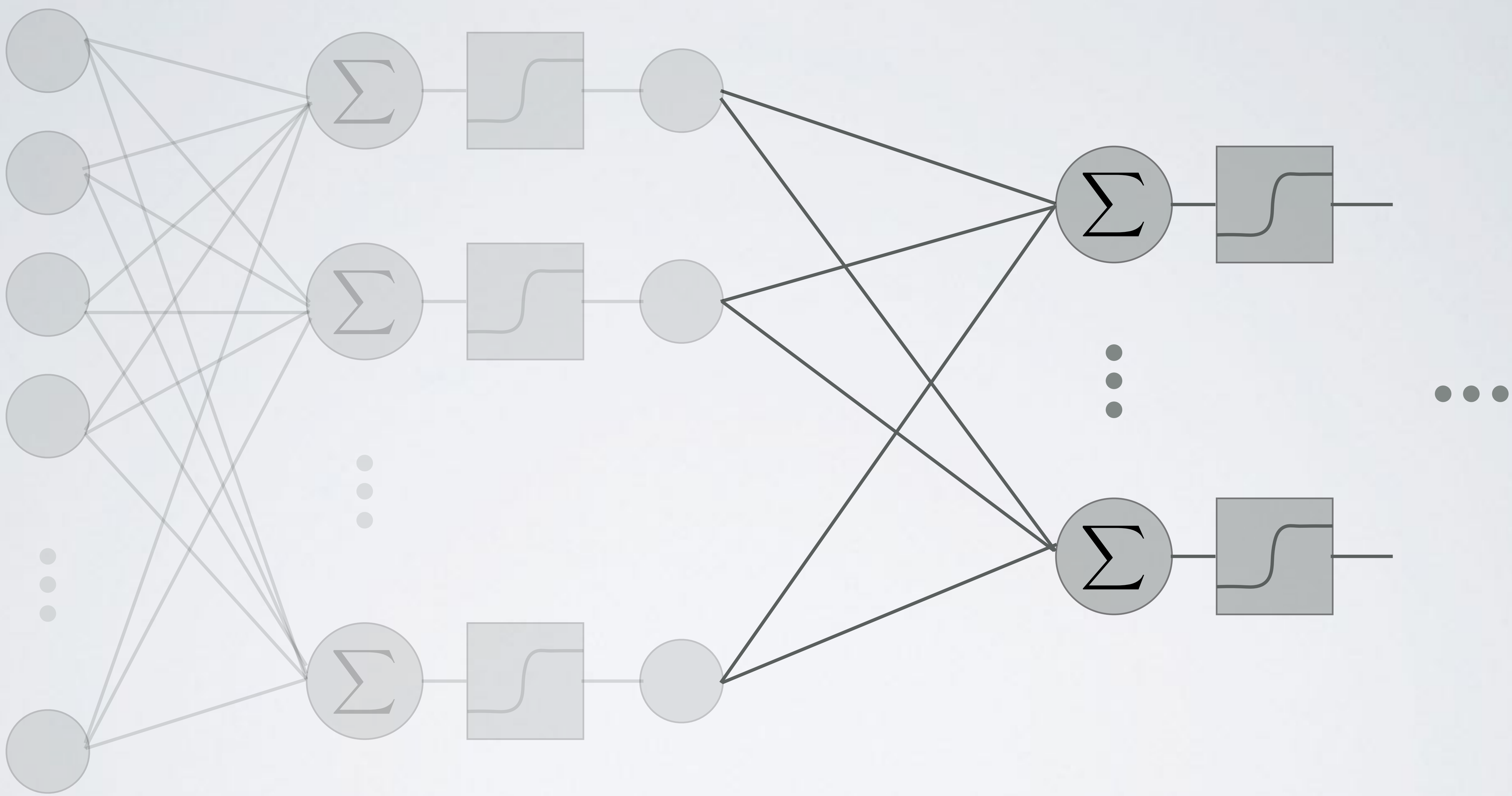
non-linearity

notation



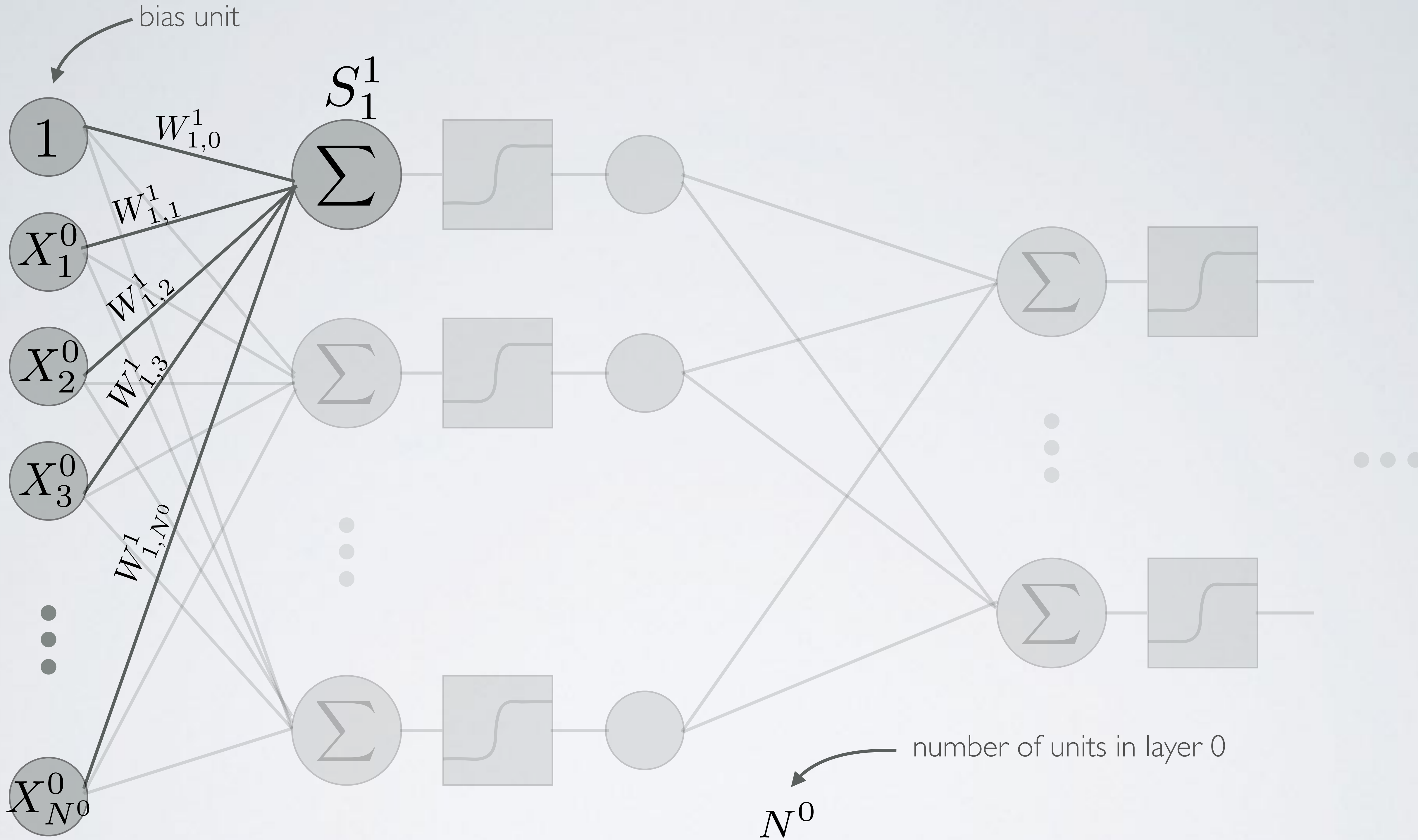
X^1

notation



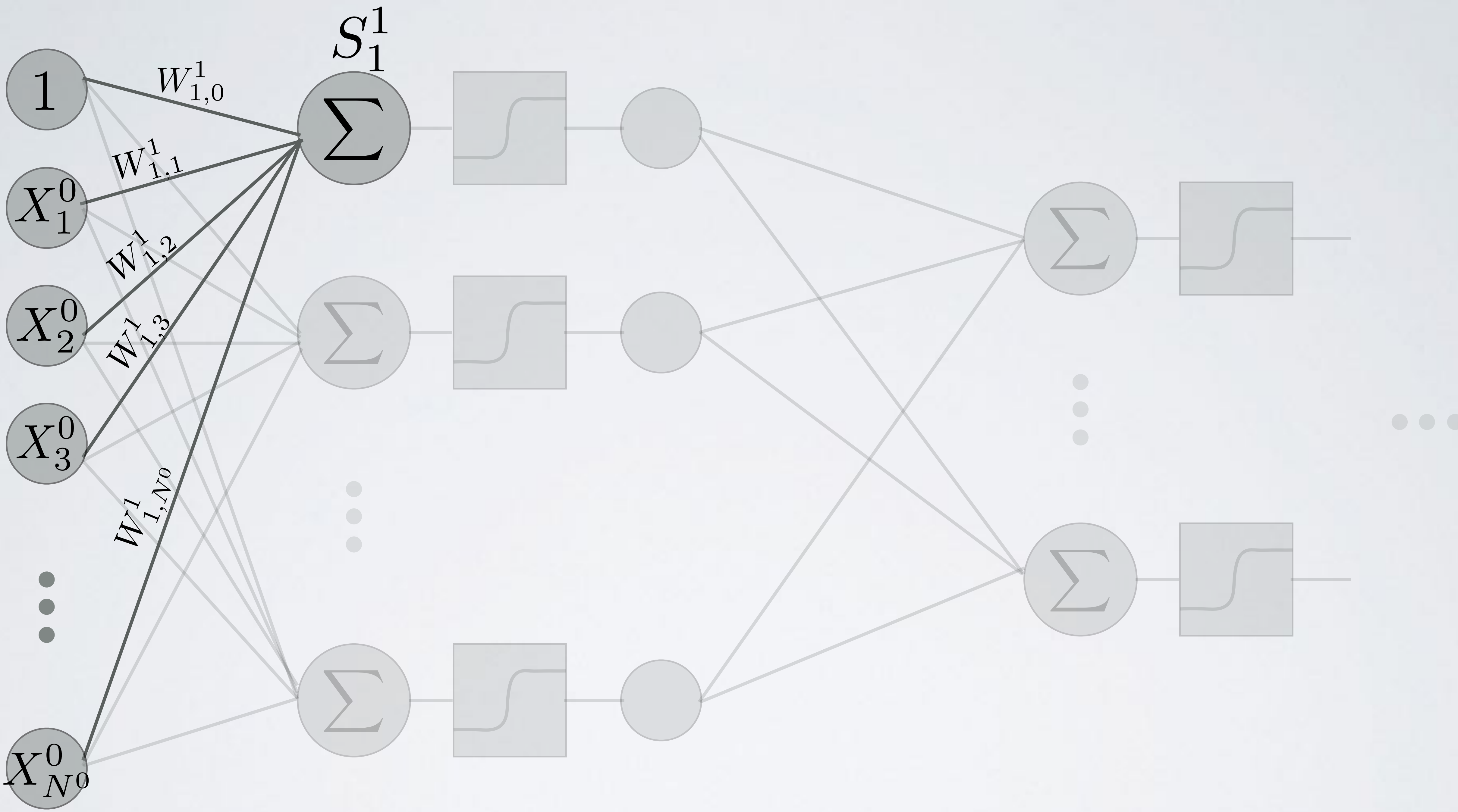
etc.

notation



$$S_1^1 = W_{1,0}^1 + \sum_{i=1}^{N^0} W_{1,i}^1 X_i^0$$

notation



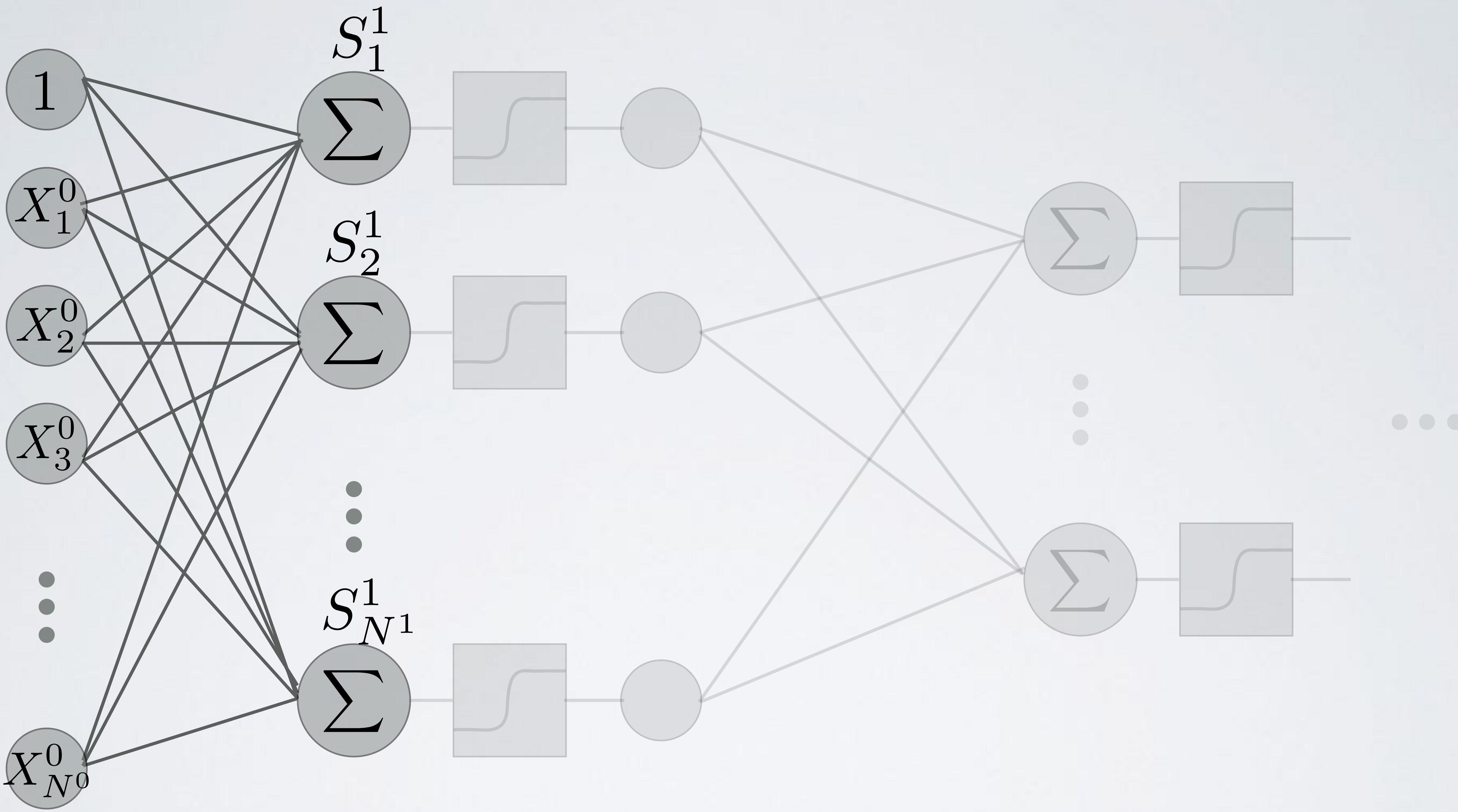
$$S_1^1 = W_1^1 X_1^0$$

vectorized form
(dot product)

absorb bias unit into X_1^0

$X_{1,0}^0 = 1$

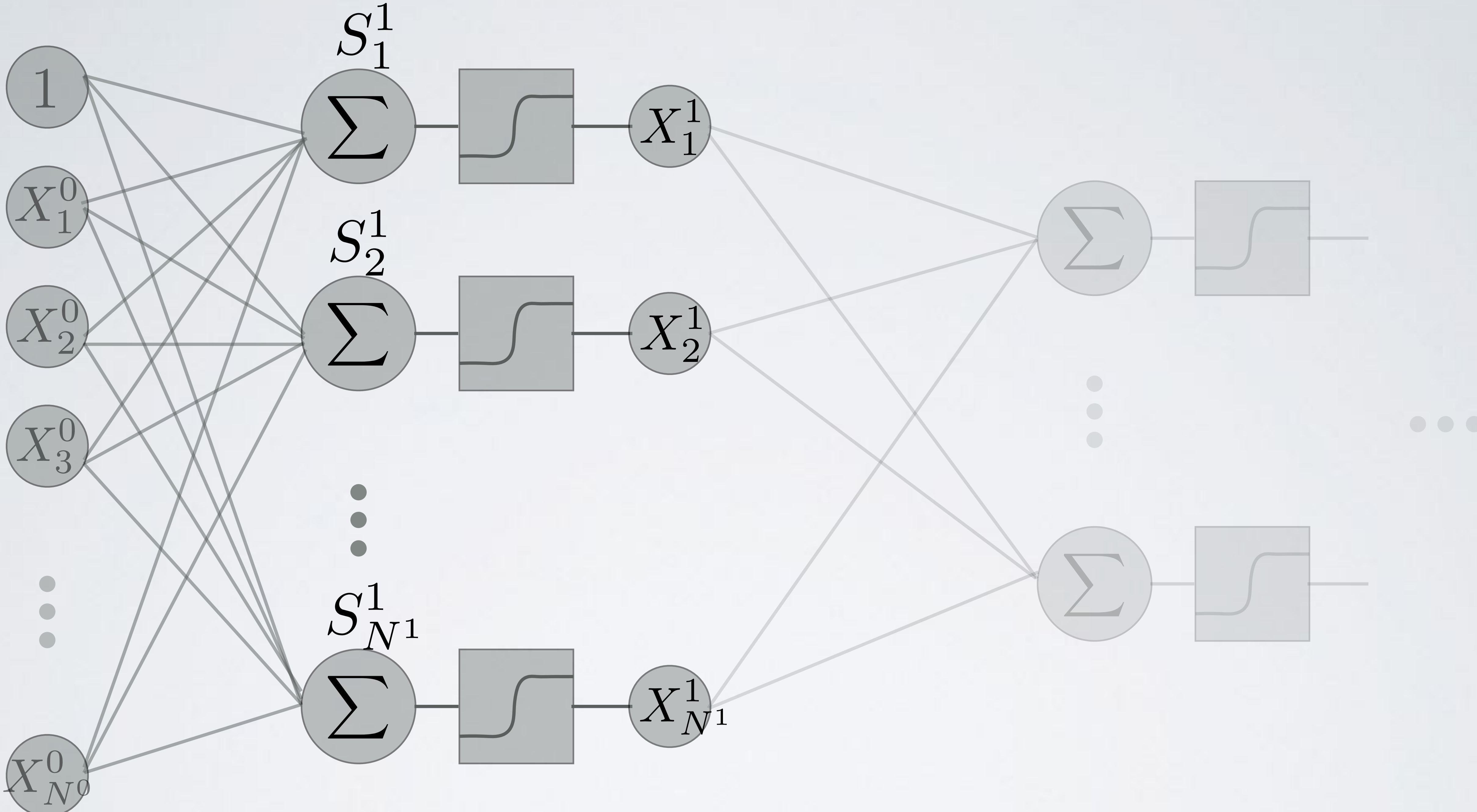
notation



$$S^1 = W^1 X^0$$

fully *vectorized* form
(matrix multiplication)

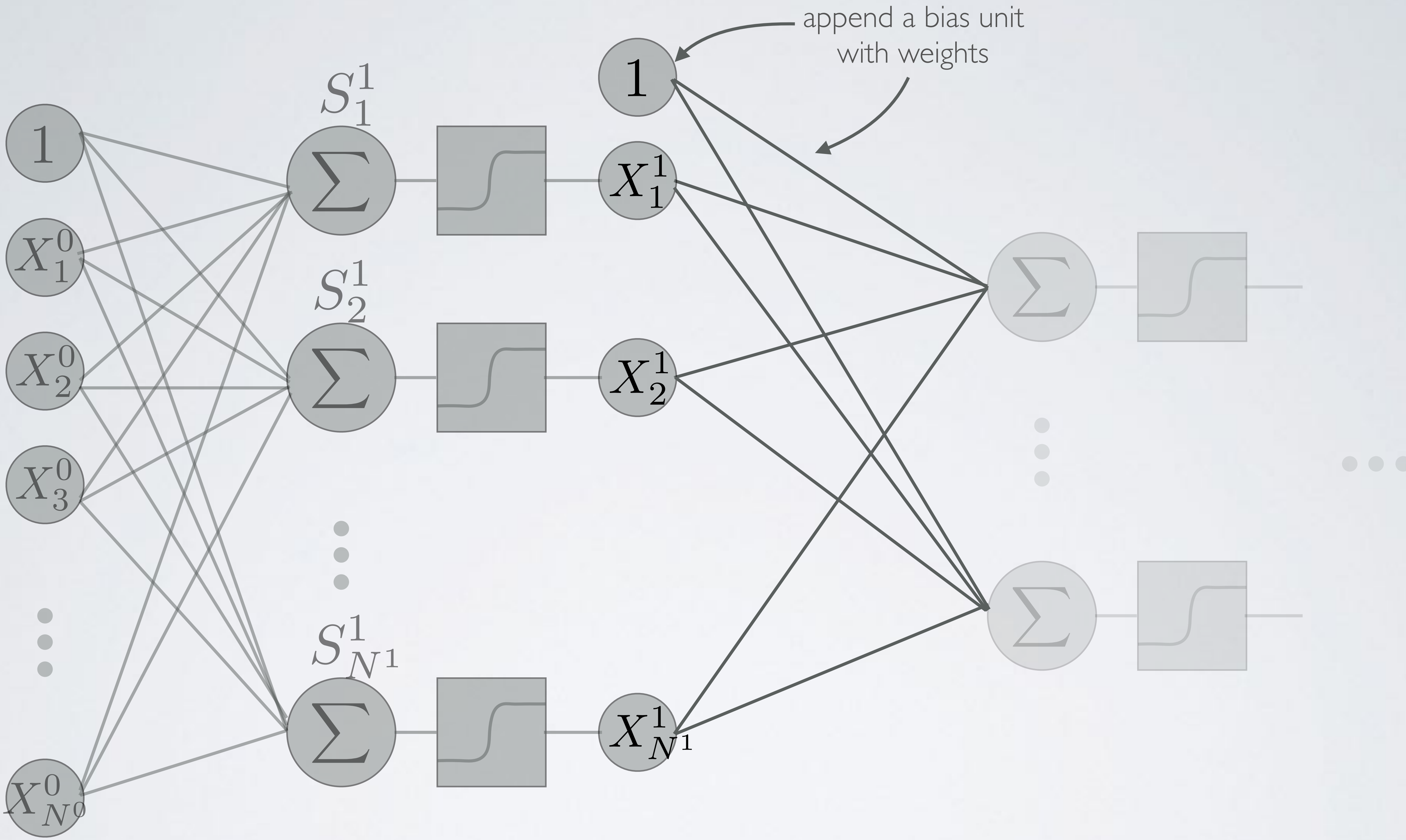
notation



$$X^1 = \sigma(S^1)$$

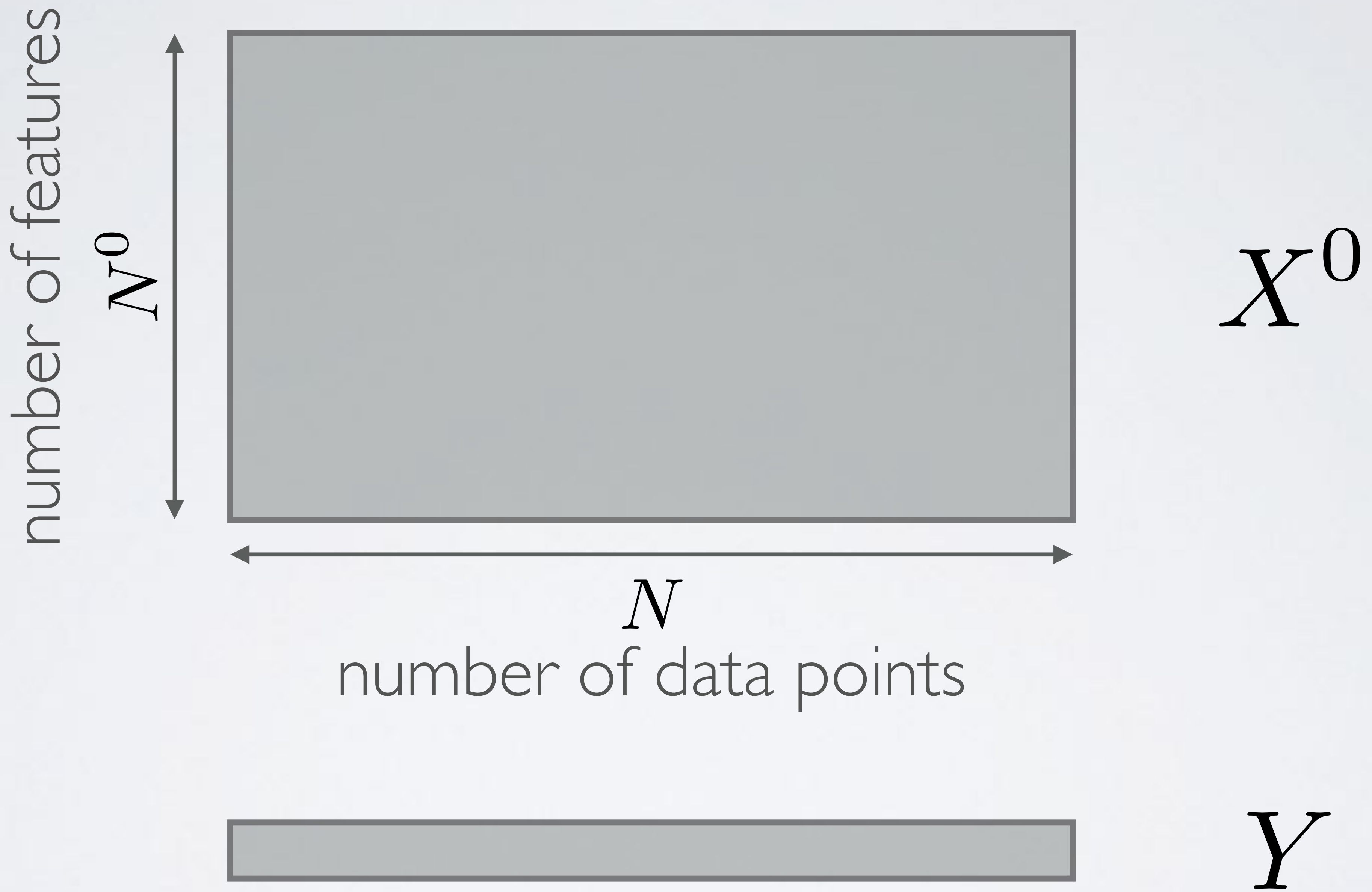
element-wise non-linearity

notation

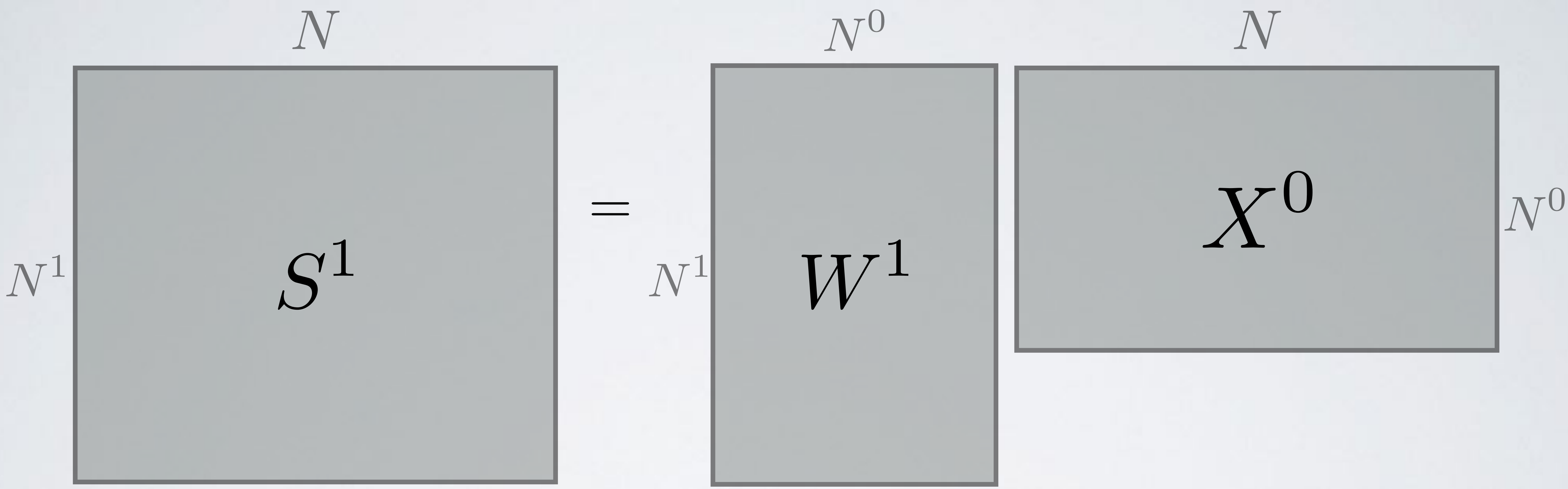


and proceed

matrix form



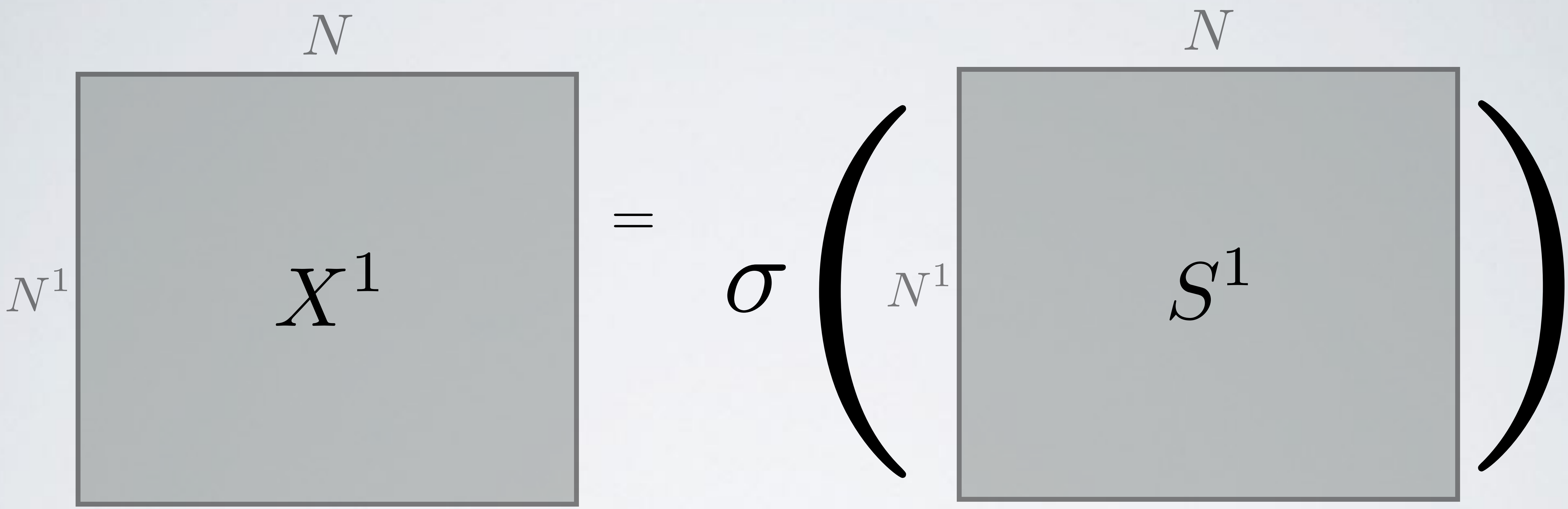
matrix form



note: appending biases to W^1 and bias units to X^0 changes

$$N^0 \to N^0 + 1$$

matrix form



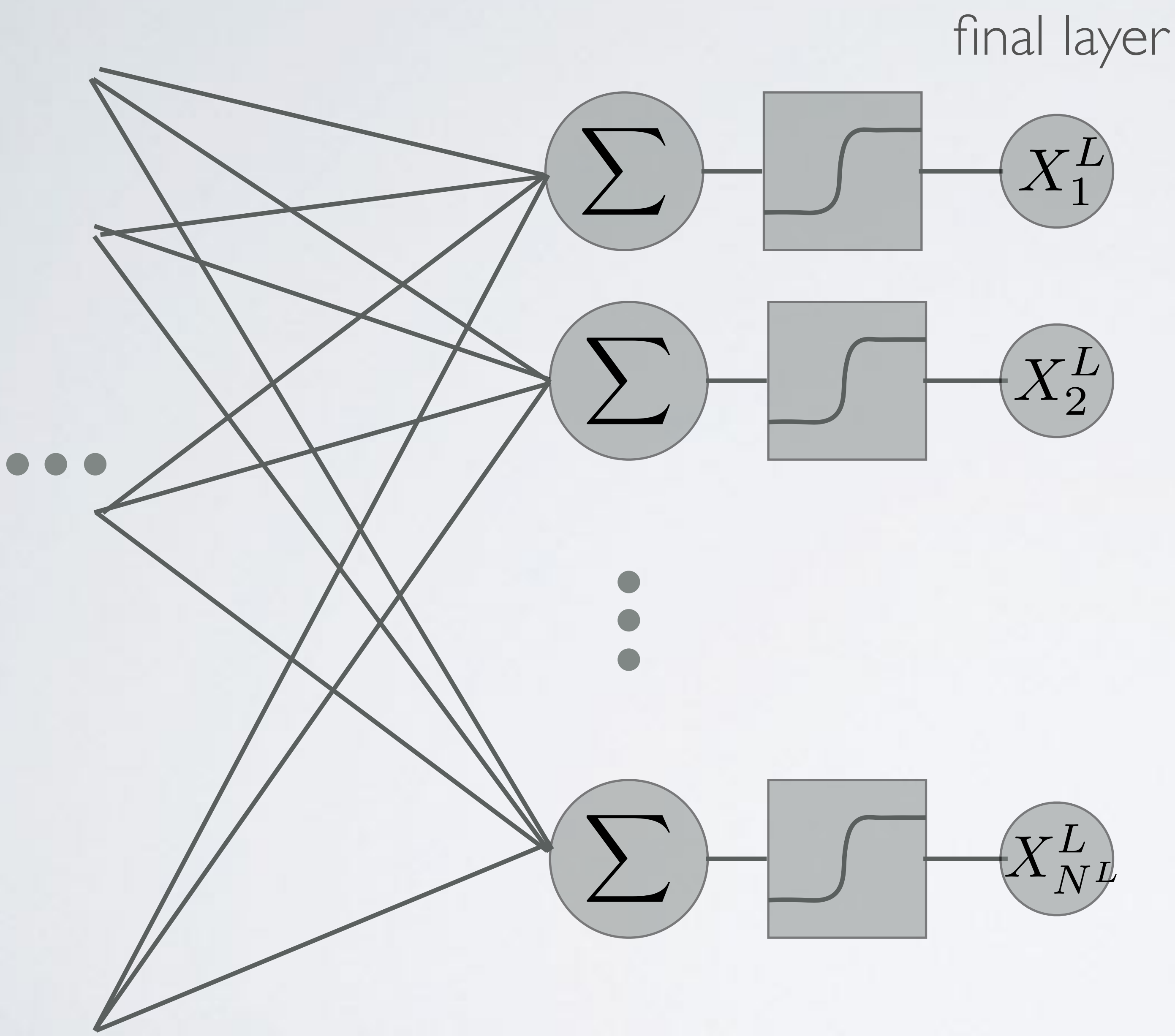
matrix form

input X^0 somewhere back in here

$$\overset{N}{N^\ell\ \boxed{X^\ell}} = \sigma\left(\boxed{W^\ell}\,\sigma\left(\boxed{W^{\ell-1}}\,\sigma\left(\boxed{W^{\ell-2}}\,\sigma\left(\ \cdots\ \right)\right)\right)\right)$$

it's just a (deep, non-linear) function!

note: bias appending omitted for clarity



train with gradient descent

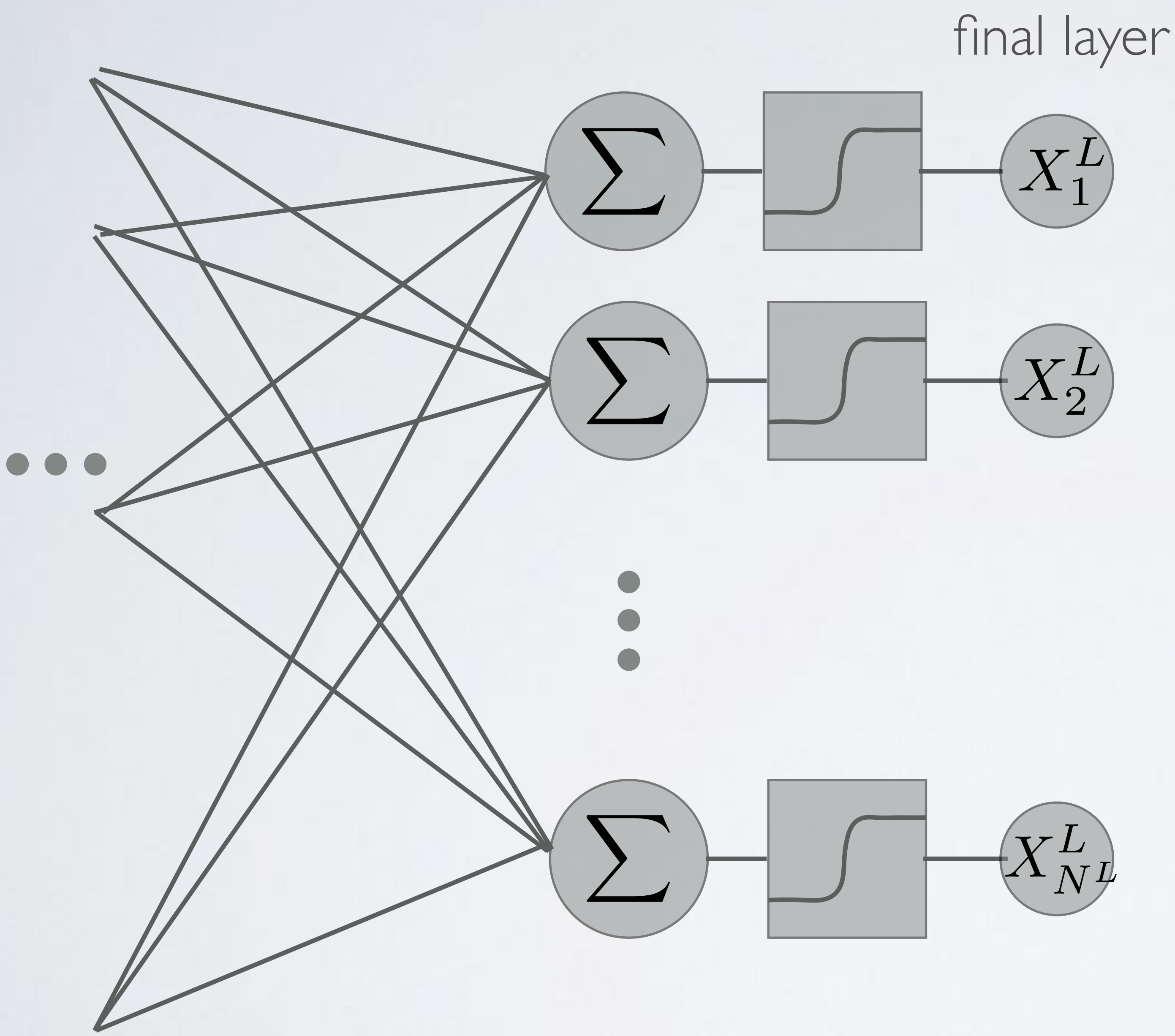
X^L is the output of the network

use $\mathcal{L}(X^L, Y)$ to compare X^L and Y

take gradients $\nabla_{W^\ell}\mathcal{L}$ of loss w.r.t. weights at each level ℓ

update the weights to decrease loss, bring X^L closer to Y

$$W^\ell \leftarrow W^\ell - \alpha \nabla_{W^\ell}\mathcal{L}$$



how do we get the gradients?
backpropagation

1. Define a loss function.

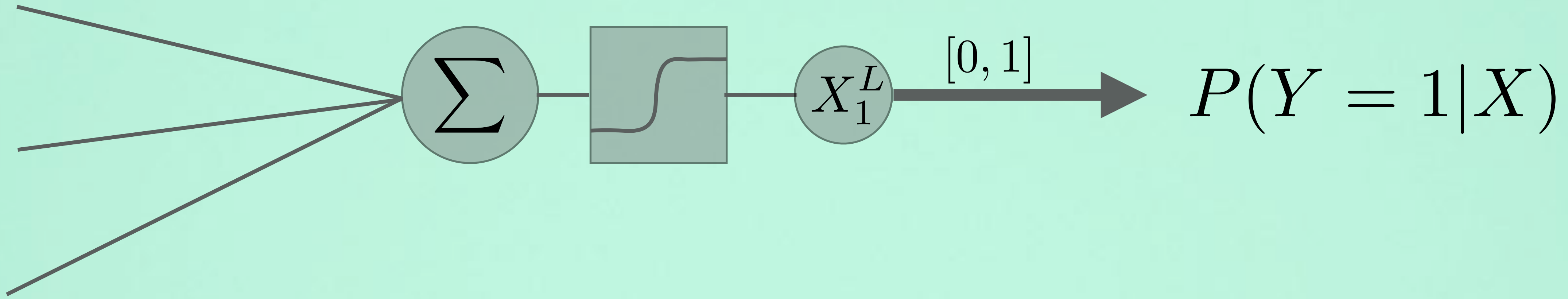
2. Use *chain rule* to recursively take derivatives at each level backward through the network.

output

what is the best form in which to present the labels?

binary labels: $Y \in \{0, 1\}$
(binary classification)

represent with a single output unit, use binary logistic regression at the last layer



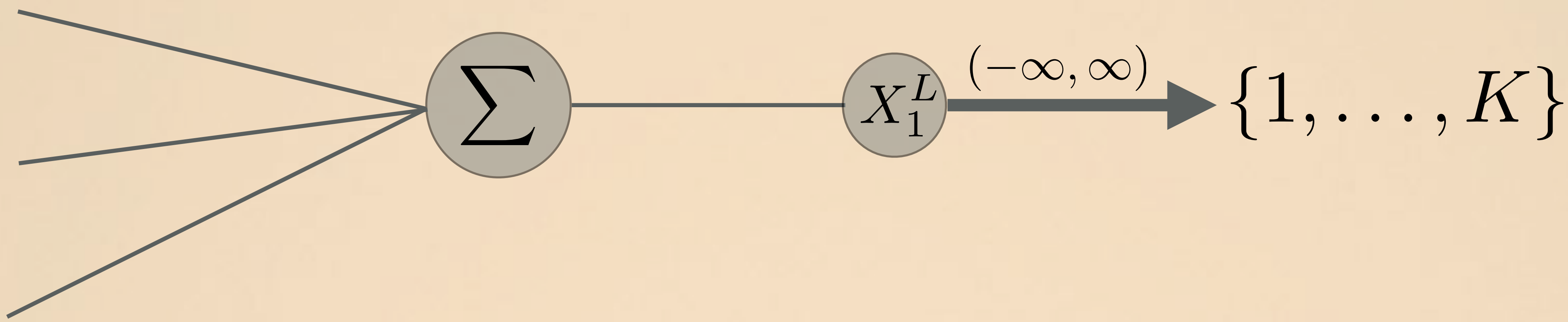
$$P(Y = 0|X) = 1 - P(Y = 1|X)$$

output

what is the best form in which to present the labels?

categorical labels: $Y \in \{1, \ldots, K\}$
(multi-class classification)

represent with a single output unit, regress to the correct class?



No, in general, classes do not have a numerical relation

class K is not 'greater' than class $K-1$

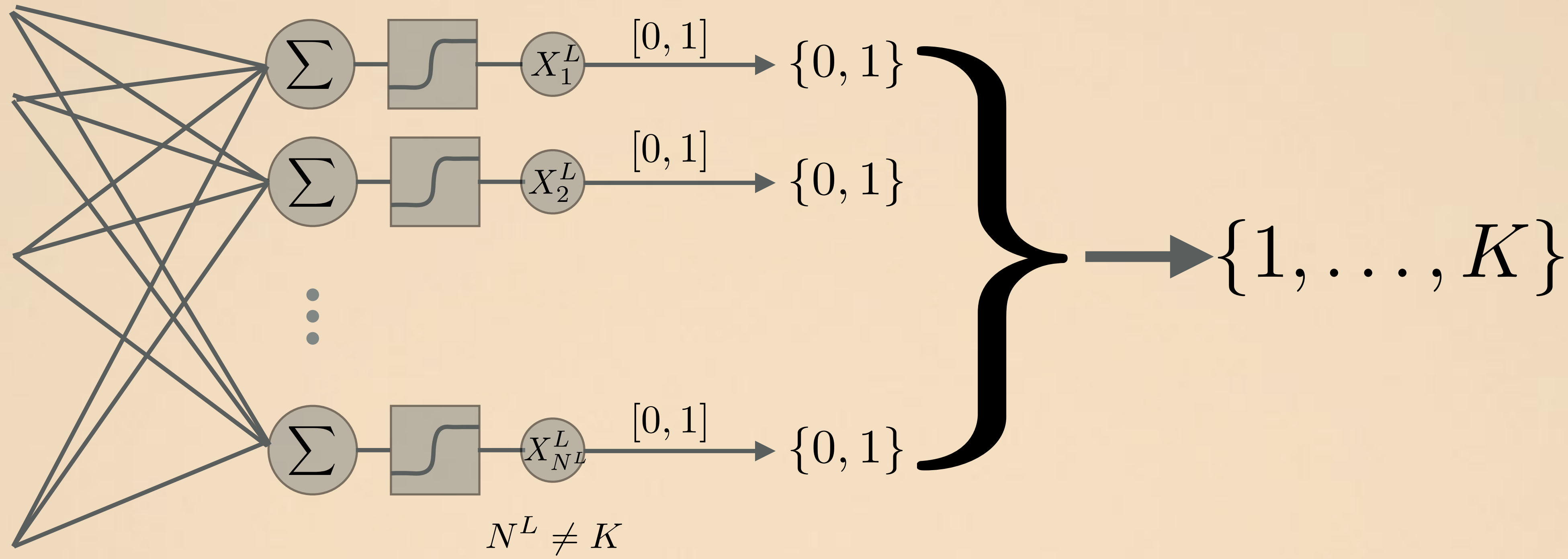
output

what is the best form in which to present the labels?

categorical labels: $Y \in \{1, \ldots, K\}$

(multi-class classification)

represent with multiple output units, regress to an encoding of the correct class (e.g. binary)



No, correct output requires coordinated effort from units

implies arbitrary similarities induced by the coding

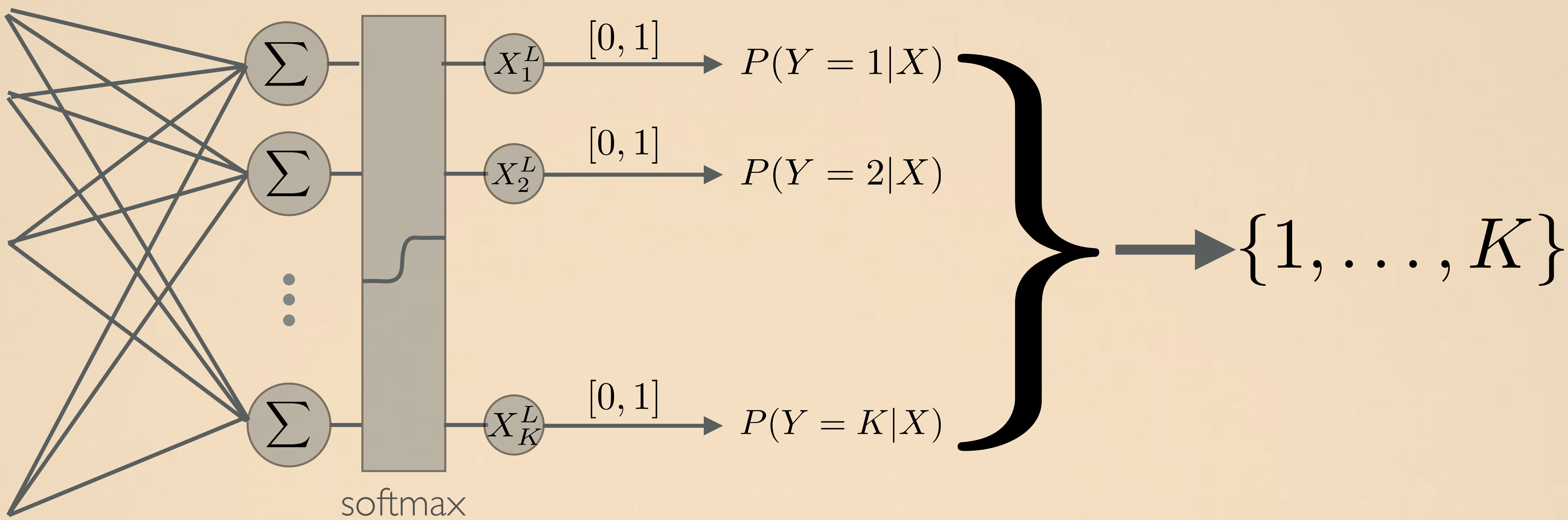
output

what is the best form in which to present the labels?

categorical labels: $Y \in \{1, \ldots, K\}$

(multi-class classification)

represent with K output units, multi-class logistic regression at the last layer



Yes

captures independence assumptions in the class structure, and is a valid output

output

categorical labels: $Y \in \{1, \ldots, K\}$
(multi-class classification)

represent with K output units,
multi-class logistic regression at the last layer

one-hot vector encoding

example $K = 5$

$$Y = 1 \quad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad Y = 2 \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad Y = 3 \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad Y = 4 \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad Y = 5 \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

loss function

probability of $\mathbf{x}$ *belonging to class* k

multi-class logistic regression

$$P(Y = k) = \frac{e^{\mathbf{w}_k^\intercal \mathbf{x}}}{\sum_{k'=1}^{K} e^{\mathbf{w}_{k'}^\intercal \mathbf{x}}}$$

softmax loss function (cross-entropy)

$$\mathcal{L}_i = -\log P(Y = y_i) = -\log \frac{e^{\mathbf{w}_{y_i}^\intercal \mathbf{x}}}{\sum_{k'=1}^{K} e^{\mathbf{w}_{k'}^\intercal \mathbf{x}}}$$

minimize the negative log probability of the correct class

recall:

$$\mathcal{L} = -\sum_{i=1}^{N} \left[y^{(i)} \log(P(y_i = 1|\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - P(y_i = 1|\mathbf{x}^{(i)})) \right]$$

softmax loss is the extension of binary case to multi-class

backpropagation

deep networks are just composed functions

$$X^L = \sigma(W^L \sigma(W^{L-1} \sigma(\ldots \sigma(W^1 X^0)))))$$

the loss $\mathcal{L}(X^L, Y)$ is a function of X^L, which is a function of each layer's weights W^ℓ

therefore, we can find $\nabla_{W^\ell} \mathcal{L}$ using chain rule

recall

$$X^\ell = \sigma(S^\ell) \qquad S^\ell = W^\ell X^{\ell-1}$$

at the output layer

$$\frac{\partial \mathcal{L}}{\partial W^L} = \frac{\partial \mathcal{L}}{\partial X^L} \frac{\partial X^L}{\partial S^L} \frac{\partial S^L}{\partial W^L}$$

backpropagation

recall

$$X^\ell = \sigma(S^\ell) \qquad\qquad S^\ell = W^\ell X^{\ell-1}$$

at the output layer

$$\frac{\partial \mathcal{L}}{\partial W^L} = \boxed{\frac{\partial \mathcal{L}}{\partial X^L}\frac{\partial X^L}{\partial S^L}}\frac{\partial S^L}{\partial W^L}$$

the first two terms are the same $\frac{\partial \mathcal{L}}{\partial S^L}$

at the layer before

$$\frac{\partial \mathcal{L}}{\partial W^{L-1}} = \boxed{\frac{\partial \mathcal{L}}{\partial X^L}\frac{\partial X^L}{\partial S^L}}\frac{\partial S^L}{\partial X^{L-1}}\frac{\partial X^{L-1}}{\partial S^{L-1}}\frac{\partial S^{L-1}}{\partial W^{L-1}}$$

backpropagation

recall

$$X^\ell = \sigma(S^\ell) \qquad\qquad S^\ell = W^\ell X^{\ell-1}$$

at the layer before

$$\frac{\partial \mathcal{L}}{\partial W^{L-1}} = \frac{\partial \mathcal{L}}{\partial X^L}\frac{\partial X^L}{\partial S^L}\frac{\partial S^L}{\partial X^{L-1}}\frac{\partial X^{L-1}}{\partial S^{L-1}}\frac{\partial S^{L-1}}{\partial W^{L-1}}$$

the first four terms are the same $\frac{\partial \mathcal{L}}{\partial S^{L-1}}$

at the layer before that

$$\frac{\partial \mathcal{L}}{\partial W^{L-2}} = \frac{\partial \mathcal{L}}{\partial X^L}\frac{\partial X^L}{\partial S^L}\frac{\partial S^L}{\partial X^{L-1}}\frac{\partial X^{L-1}}{\partial S^{L-1}}\frac{\partial S^{L-1}}{\partial X^{L-2}}\frac{\partial X^{L-2}}{\partial S^{L-2}}\frac{\partial S^{L-2}}{\partial W^{L-2}}$$

backpropagation

general overview - *dynamic programming*

compute gradient of loss w.r.t. W^L

store $\frac{\partial \mathcal{L}}{\partial S^L}$

for $\ell = L - 1, \ldots, 1$

use $\frac{\partial \mathcal{L}}{\partial S^{\ell+1}}$ to compute gradient of loss w.r.t. W^ℓ

store $\frac{\partial \mathcal{L}}{\partial S^\ell}$

backpropagation

recall

$$X^\ell = \sigma(S^\ell) \qquad\qquad S^\ell = W^\ell X^{\ell-1}$$

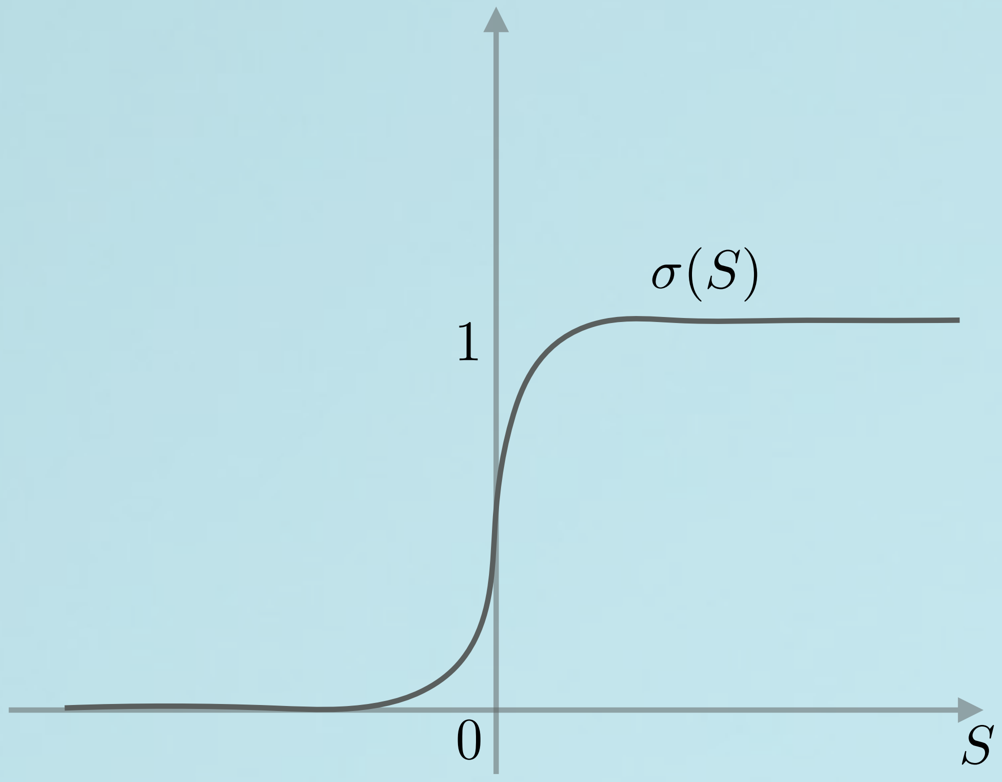
$$\frac{\partial X^\ell}{\partial S^\ell}$$

derivative of non-linearity w.r.t. input
depends on non-linearity used

$$\frac{\partial S^\ell}{\partial X^{\ell-1}} = W^\ell$$

$$\frac{\partial S^\ell}{\partial W^\ell} = (X^{\ell-1})^\intercal$$

vanishing gradients



if we use *saturating* non-linearities, the derivative of the non-linearity will always be less than one

in a deep network, there will be many of these terms multiplied together, shrinking the gradient at early layers

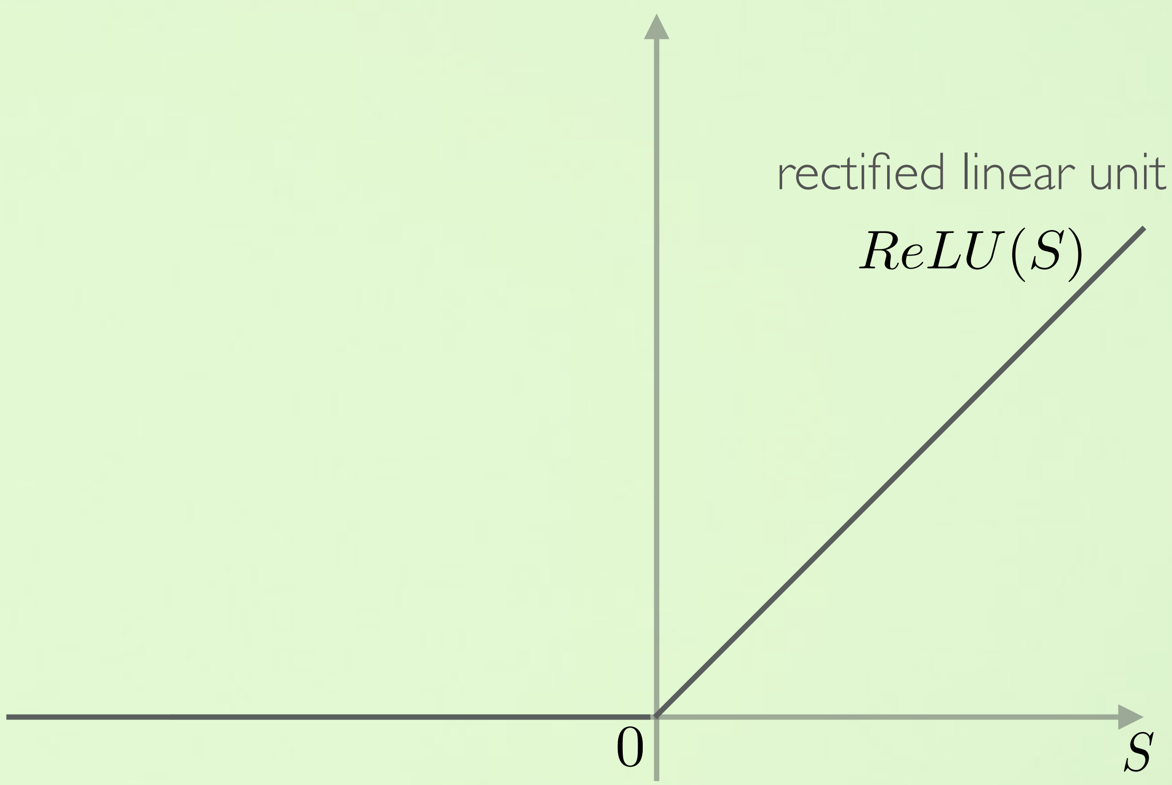
the gradient **'vanishes'**

to solve this issue, use non-saturating non-linearities

$$ReLU(S) = \max(0, S)$$

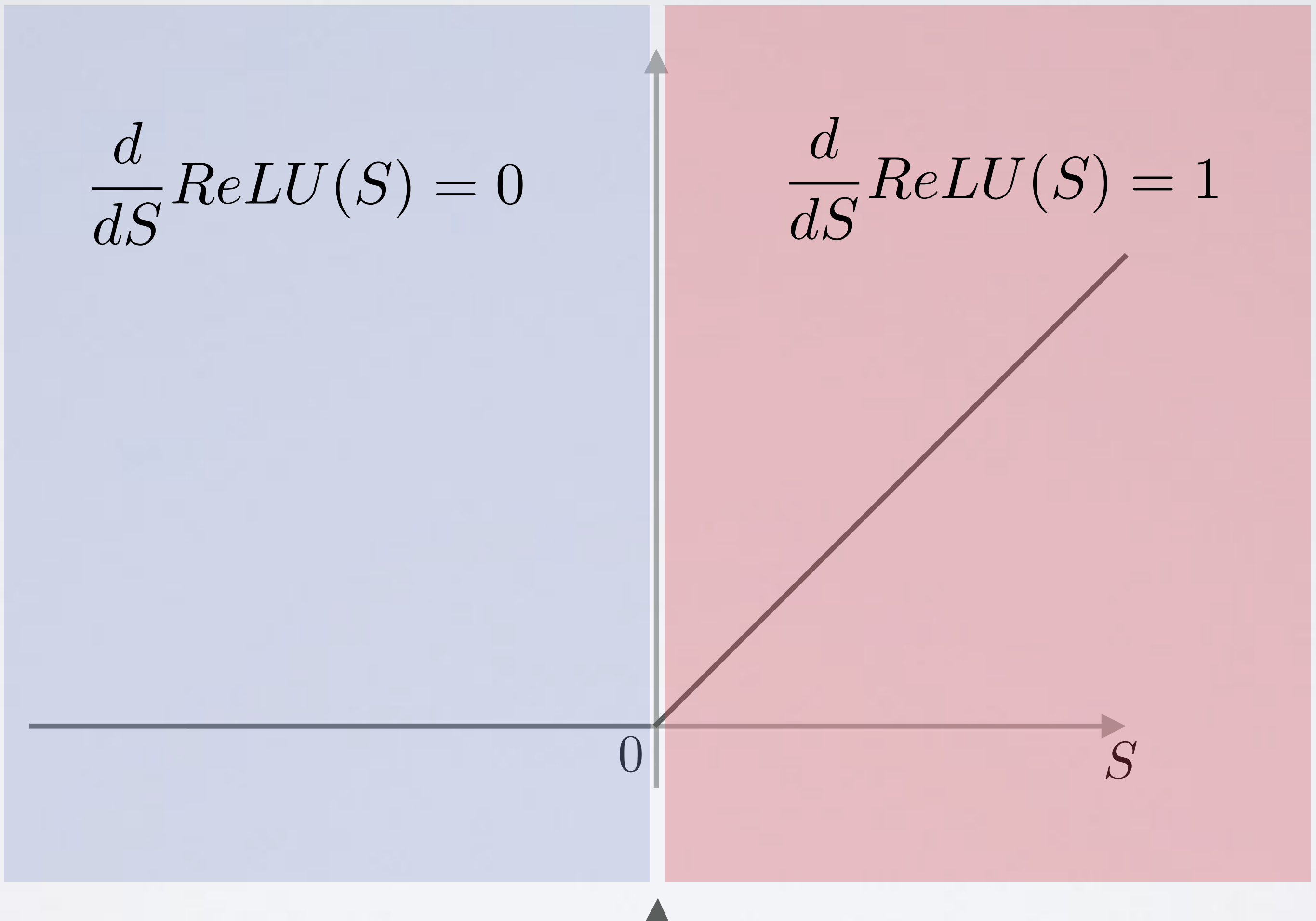
pro: keep gradient signal strong at early layers

con: partially linearizes the network, making it less expressive



Glorot, 2011

rectified linear units

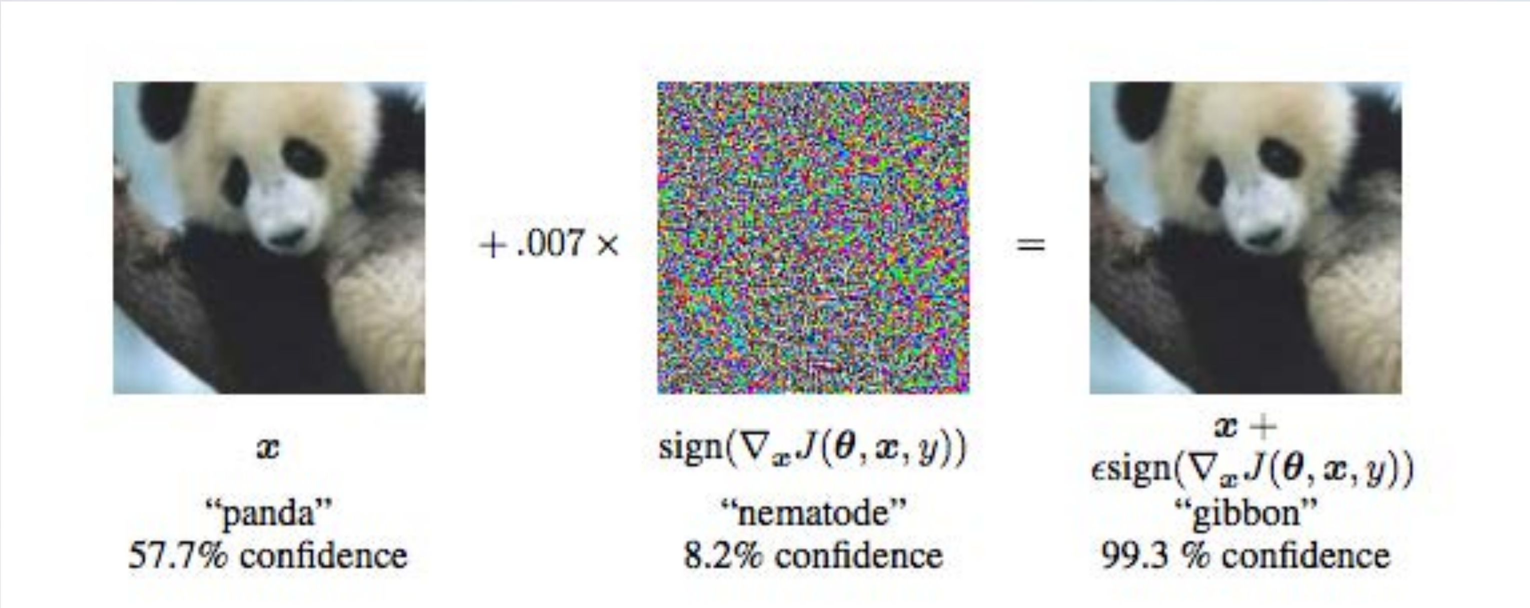


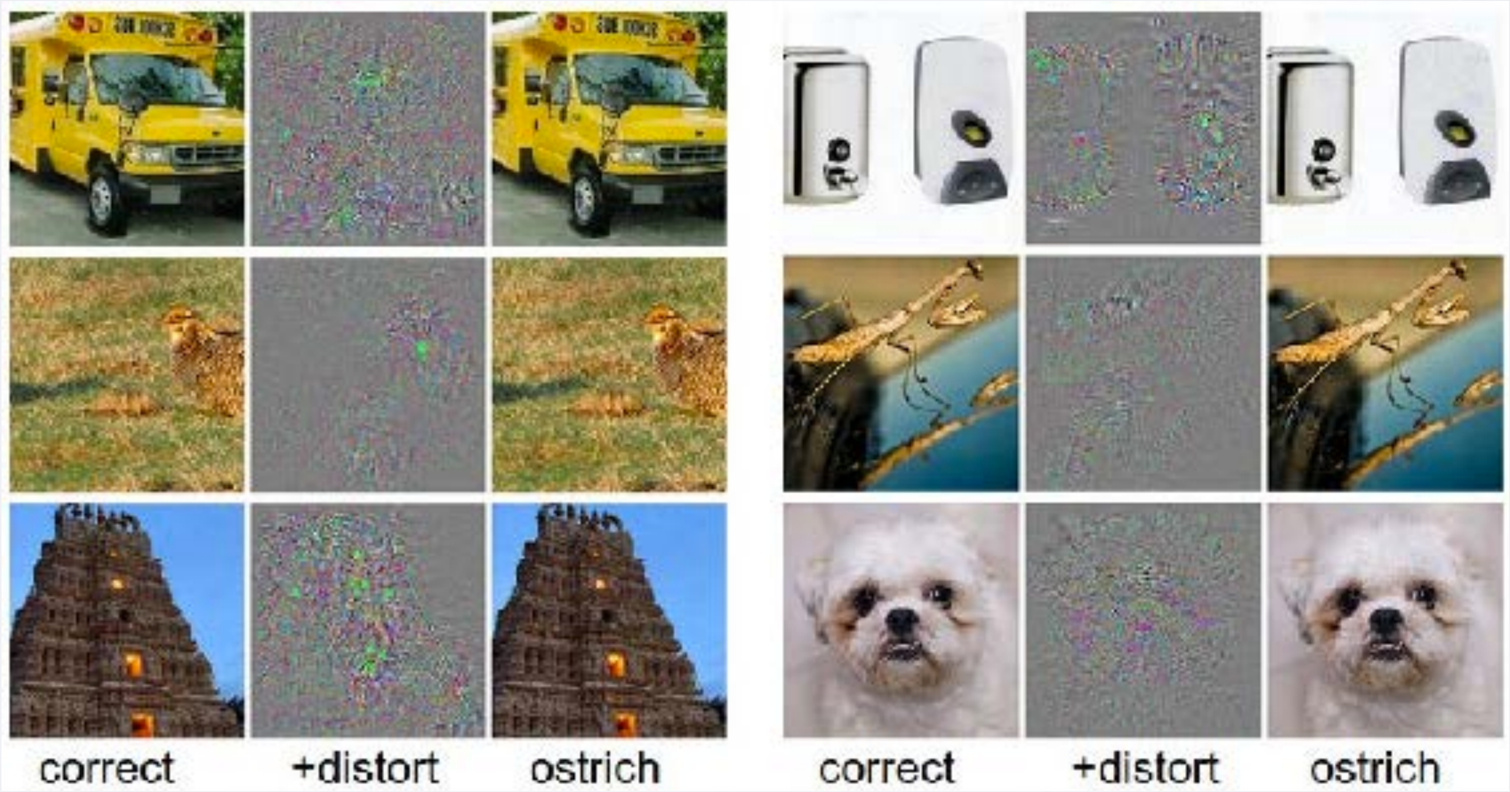
$\frac{d}{dS} ReLU(S)$ undefined

$$ReLU(S) = \max(0, S)$$

Glorot, 2011

adversarial examples





Goodfellow et al. , 2014
Szegedy et al. , 2013

regularization

we often want to punish model complexity

deep networks are a powerful model class, making it easier for them to overfit

as in other ML methods, we can regularize by putting a penalty on the weights, and adding this term to the loss function

$$\lambda \sum_{\ell=1}^{L} ||W^{\ell}||_1$$

L1 regularization

'weight sparsity'

$$\lambda \sum_{\ell=1}^{L} ||W^{\ell}||_2^2$$

L2 regularization

'weight decay'

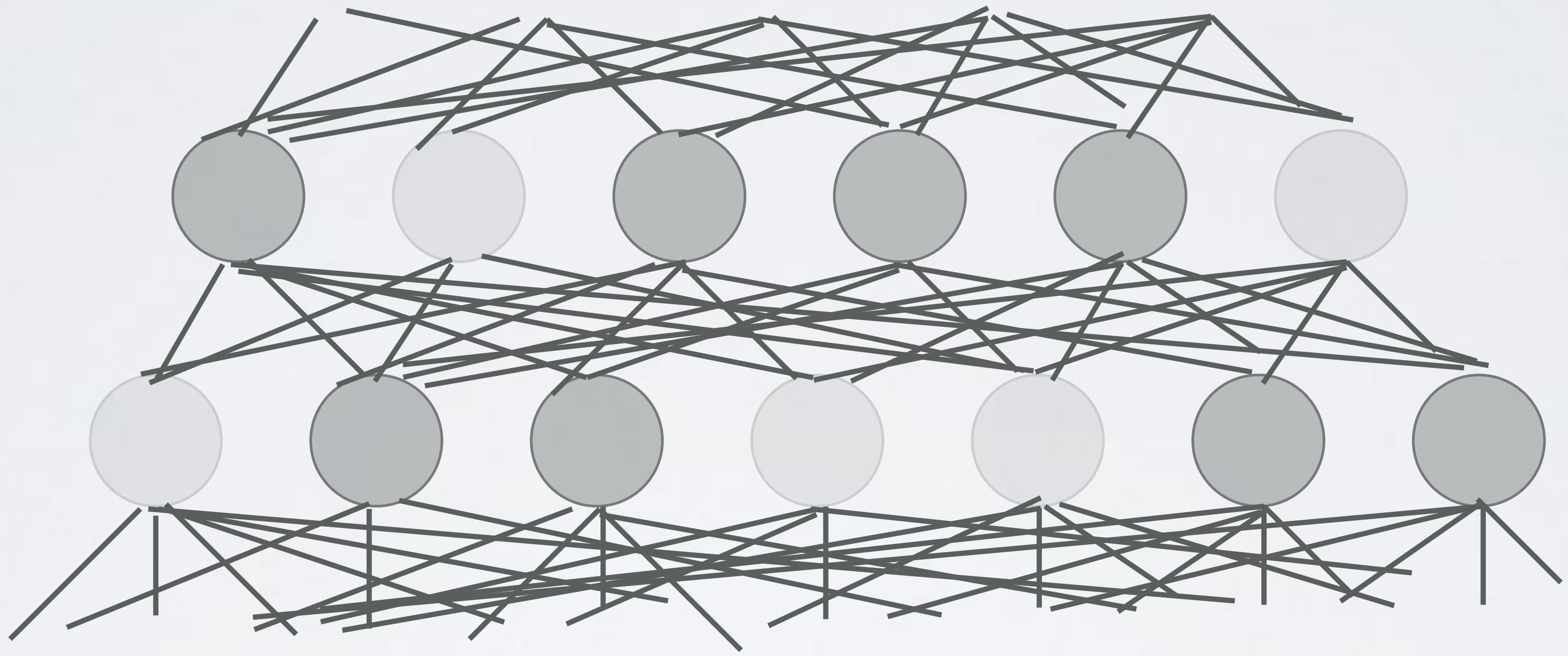
this can be achieved through L1 or L2 regularization on network's weights

other forms of regularization

dropout

often, deep networks will learn *entangled* representations, in which the internal representation depends heavily on coordinated activity from multiple units.

this is referred to as *'fragile co-adaptation,'* and often generalizes poorly



to encourage units to learn statistically independent features, randomly *dropout* a fraction of the units during each training iteration

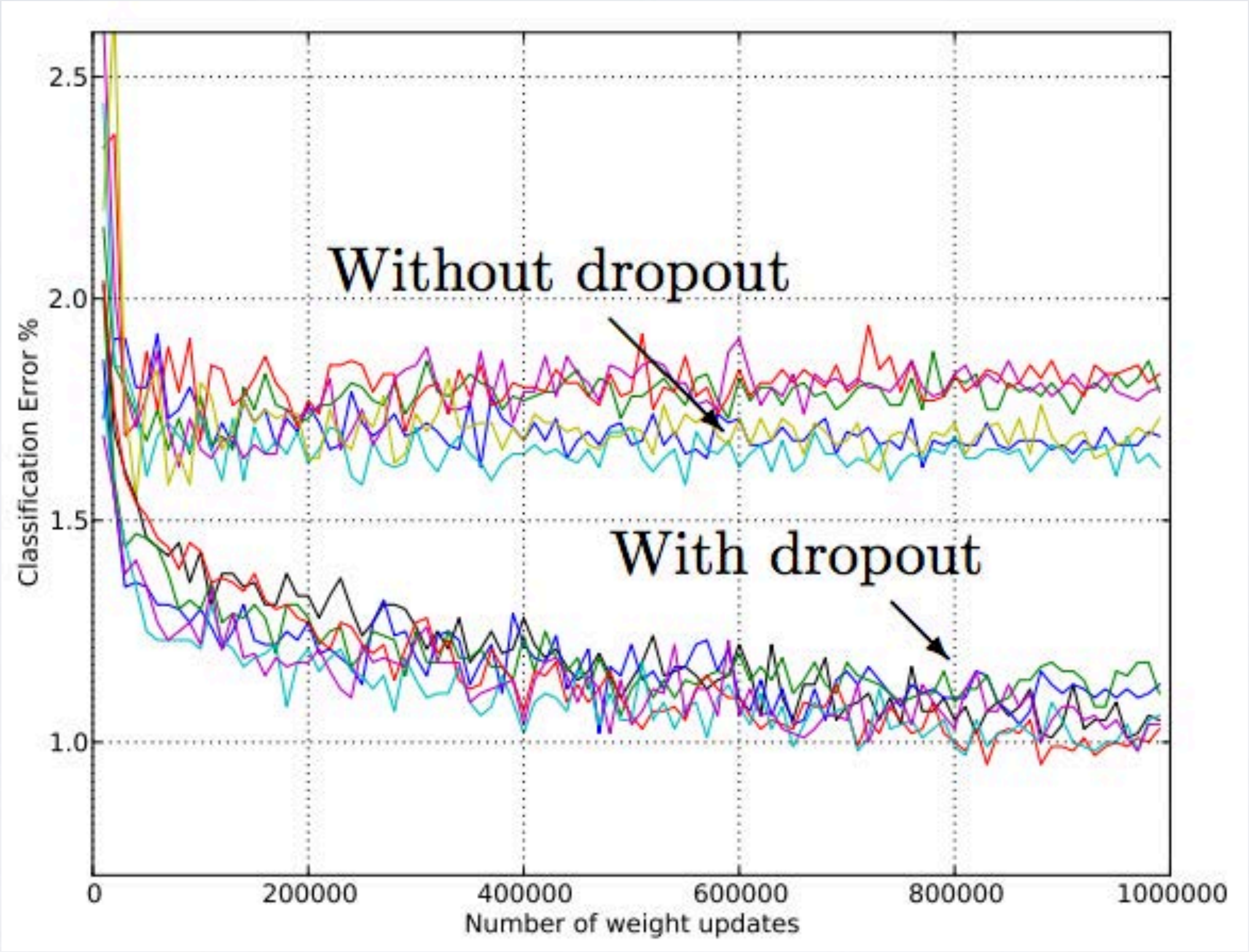
something like an 'internal ensemble' of an exponential number of different models

during test time, keep all units active

Srivastava, 2014

other forms of regularization

dropout



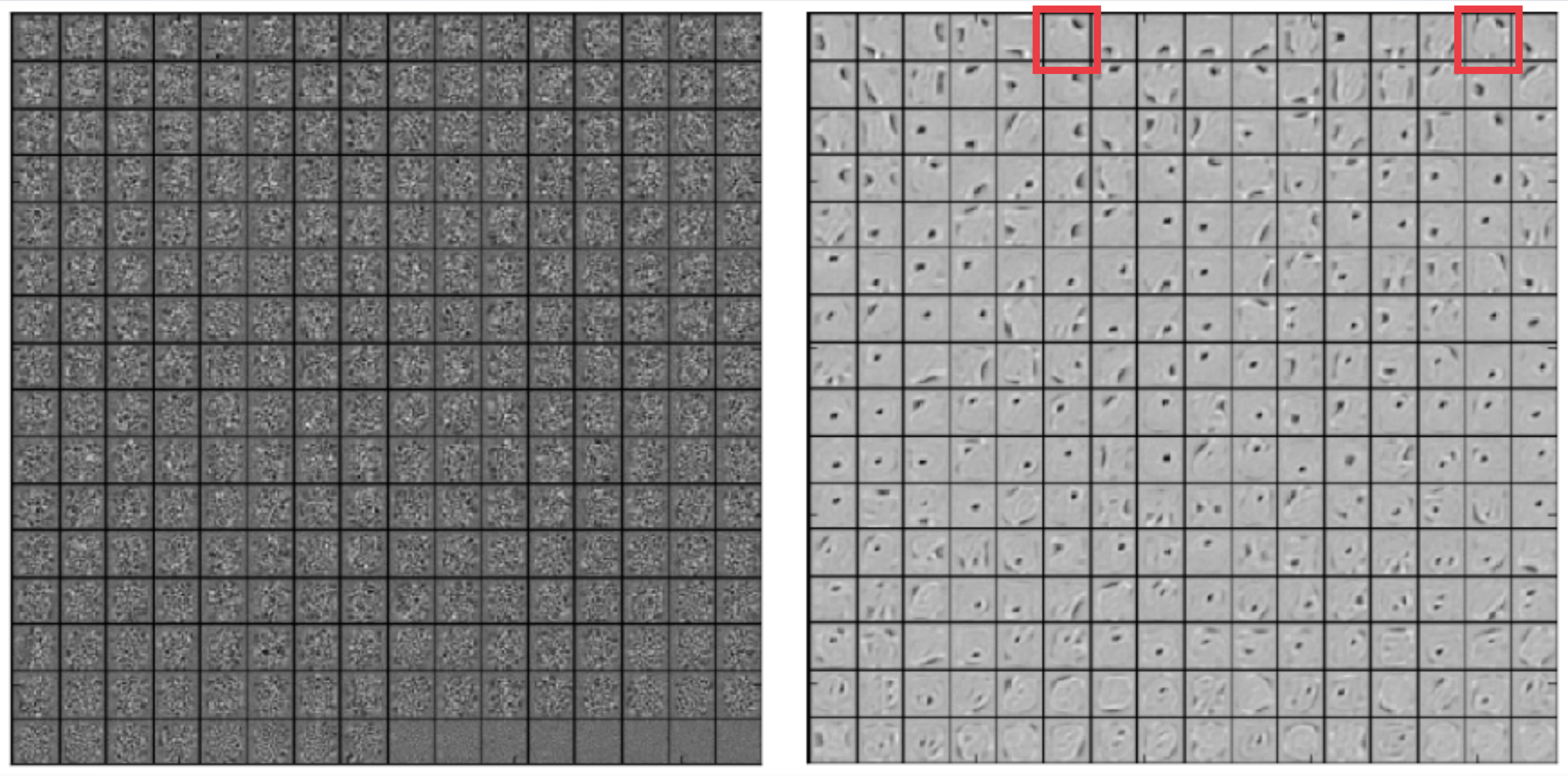
Srivastava, 2014

other forms of regularization

dropout

visualization of features learned on MNIST digits

without dropout | with dropout $p = 0.5$



redundant features

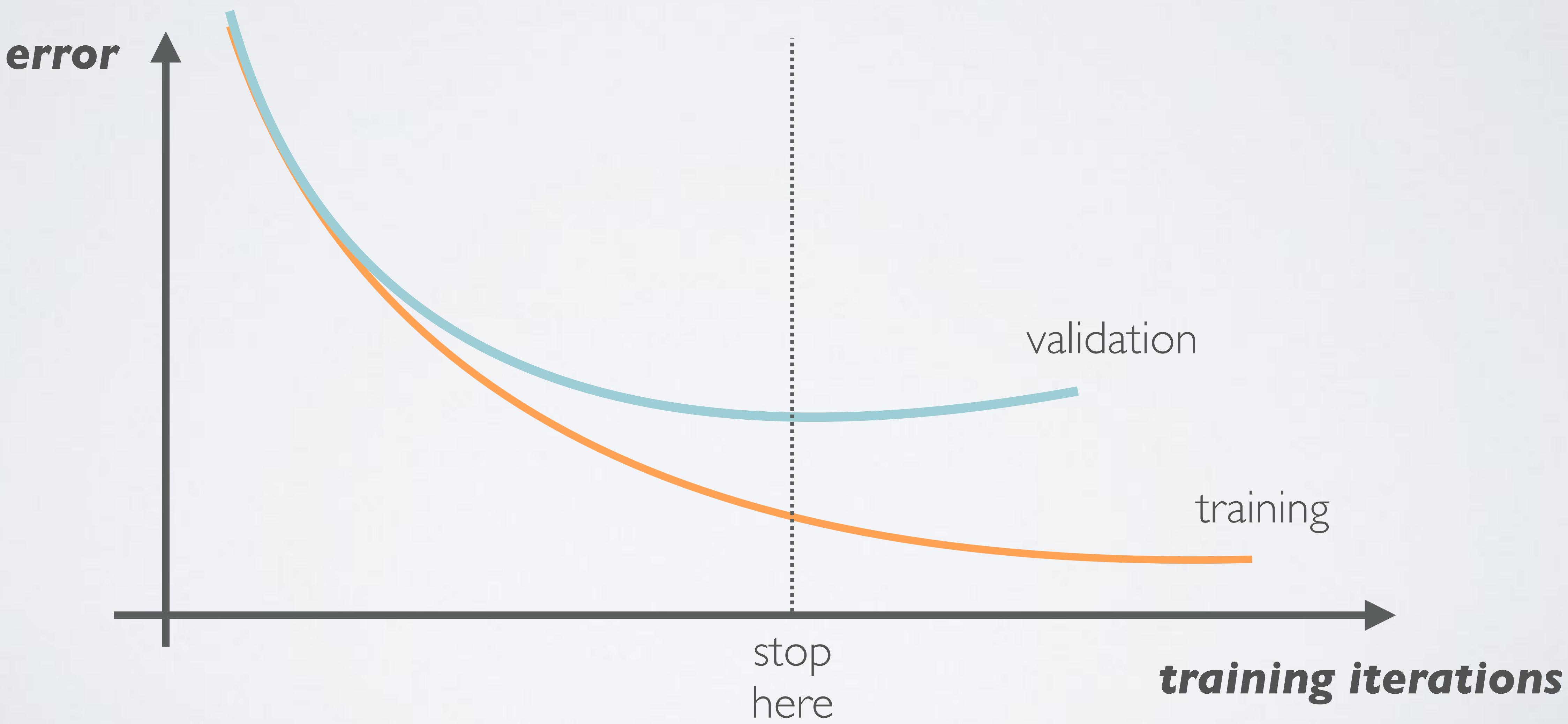
Srivastava, 2014

other forms of regularization

early stopping

stop training the model when the validation loss or error plateaus (stops decreasing)

prevents the model from overfitting to the training set



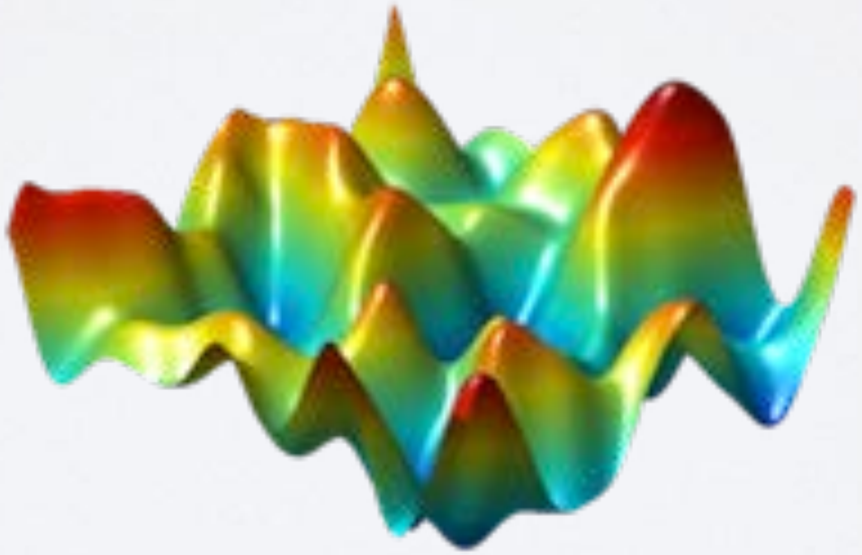
optimization

gradient descent: feed in the entire dataset, calculate gradients for each weight, update the weights, repeat until convergence

with a large model and large dataset, this will be an incredibly slow process

gradient contributions from each data example are averaged

accurate gradient, but one epoch results in a single 'step'



optimization landscape in weight space

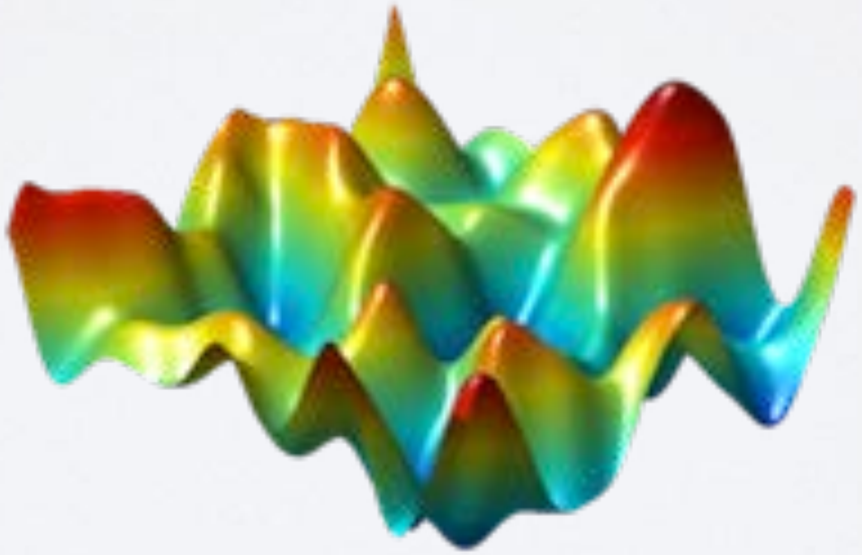
optimization

(mini-batch) stochastic gradient descent (SGD): feed in the dataset one batch at a time, calculate gradients for each weight from that batch, update the weights, repeat until convergence, randomly shuffling after each epoch

each batch provides a noisy estimate of the gradient

with an adequate batch size, this is often good enough to head in the right direction

noise in the gradient can actually help prevent getting stuck in local optima



optimization landscape in weight space

optimization

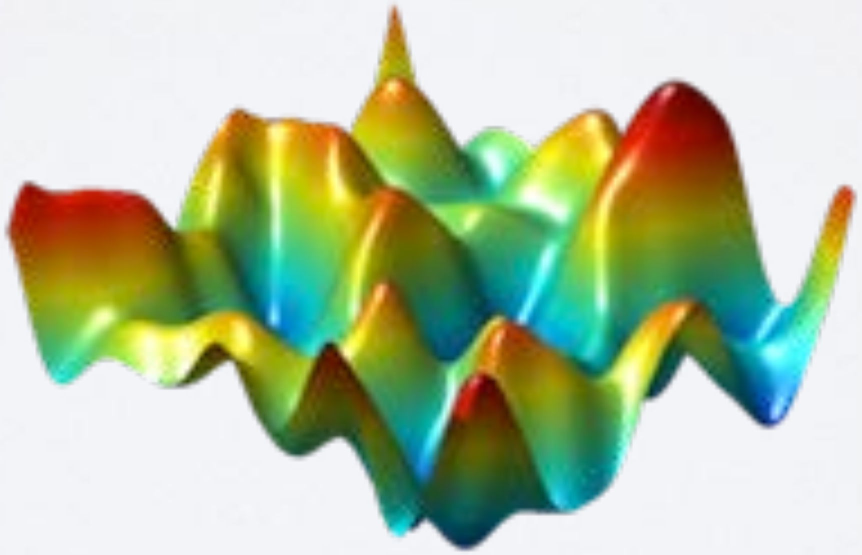
training a deep neural network involves *non-convex optimization*

convex optimization: with proper learning rate, guaranteed to converge to global optimum

non-convex optimization: no guarantees, may converge to a local optimum

should we be worried about ending up in local optima?

no, not really.
as the number of weights grows, it tends to become easier to escape these local minima.
they appear to be mostly saddle points, and most local minima are actually pretty good.



optimization landscape in weight space

what makes a deep network non-convex?

it's the non-linearities!

$$X^\ell = \sigma\left(W^\ell \, \sigma\left(W^{\ell-1} \, \sigma\left(W^{\ell-2} \, \sigma\left(\cdots \; X^0 \right)\right)\right)\right)$$

if we remove the non-linearities…

$$X^\ell = W^\ell \, W^{\ell-1} \, W^{\ell-2} \cdots X^0$$

$$\longrightarrow \quad X^\ell = W^{1 \ldots \ell} \, X^0$$

the network collapses down into a (convex) linear optimization problem

optimization techniques

vanilla stochastic gradient descent

$$W^{\ell} \leftarrow W^{\ell} - \alpha \nabla_{W^{\ell}} \mathcal{L}$$

stochastic gradient descent with *momentum*

momentum $\in (0, 1]$

$$\mu \leftarrow \beta\mu + \alpha \nabla_{W^{\ell}} \mathcal{L}$$

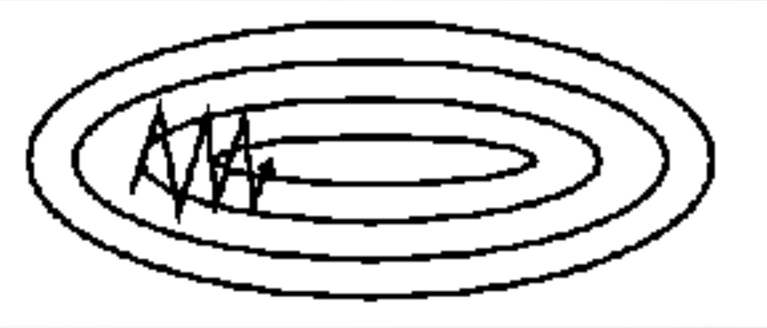
$$W^{\ell} \leftarrow W^{\ell} - \mu$$

gradient is influence by previous gradient updates

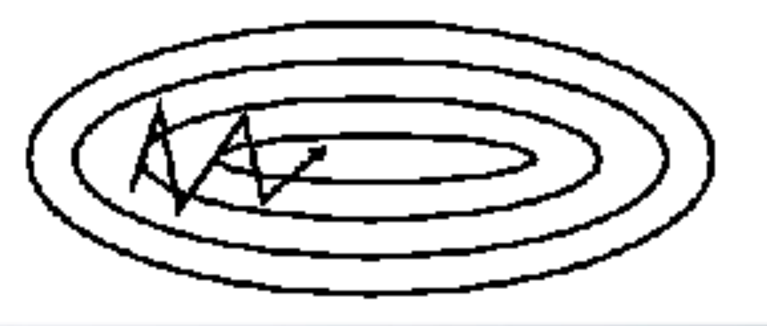
speeds up convergence immensely,
prevents optimization from bouncing back and forth too much

rule of thumb:
momentum = 0.9

vanilla SGD



with momentum



http://sebastianruder.com/optimizing-gradient-descent/index.html

optimization techniques

problem: *how do we set and anneal the learning rate?*
why have the same learning rate for all parameters?

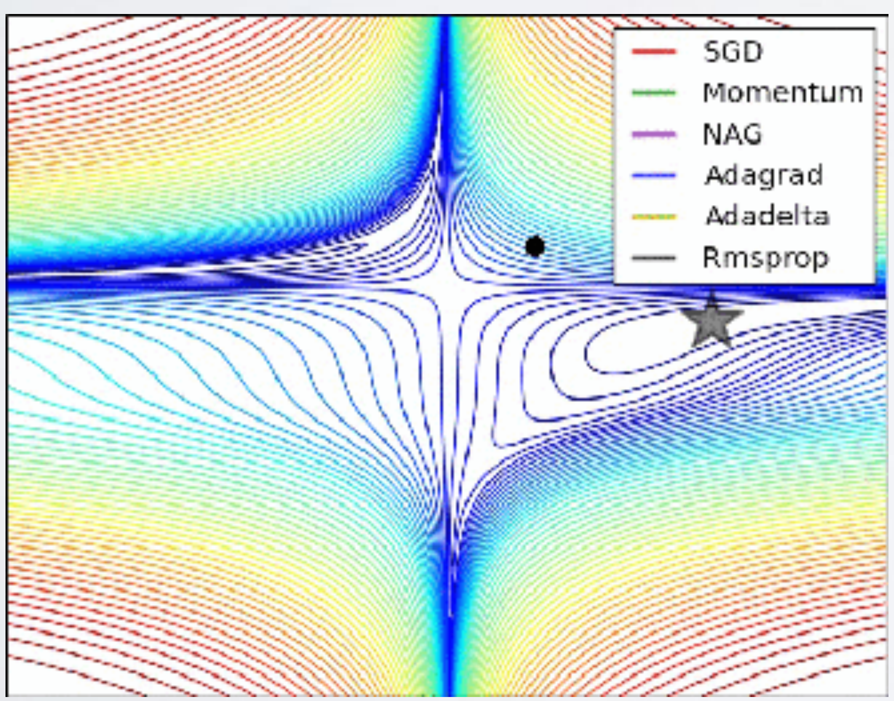
adaptive learning rate techniques

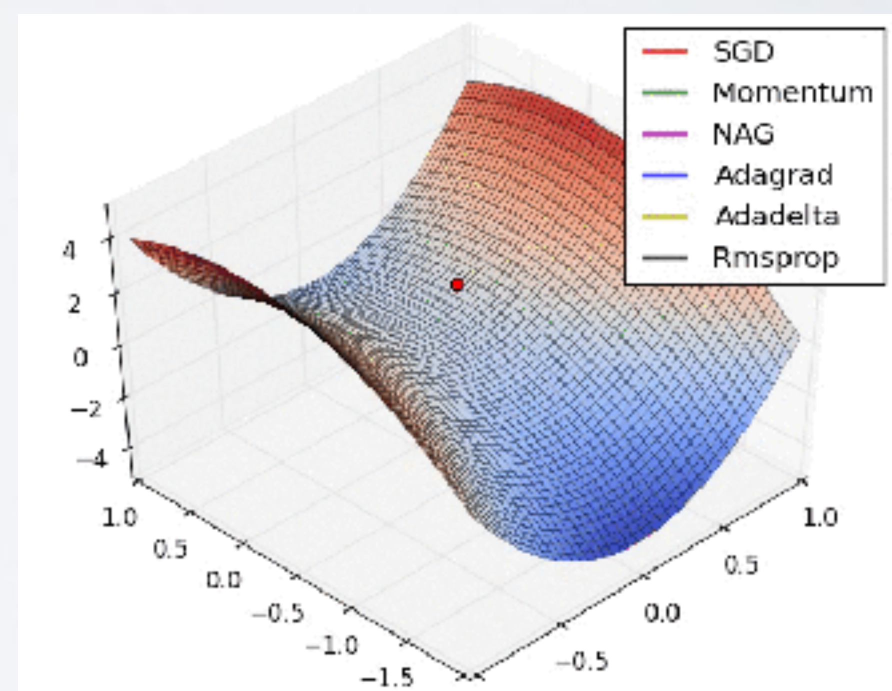
adagrad: shrink each parameter's learning rate according to magnitude of sum of past gradients

adadelta: similar to adagrad, but with exponentially decaying influence from past gradients

RMSprop: similar idea to adadelta

adam: similar to adadelta, but with additional decaying influence from previous gradients (like momentum)





http://sebastianruder.com/optimizing-gradient-descent/index.html

deep learning history (abridged)

1957 *perceptron learning algorithm*

neural net winter

1980 neocognitron

1986 backprop becomes popular

1989 convolutional neural networks

neural net winter II

2006 unsupervised pre-training of deep networks

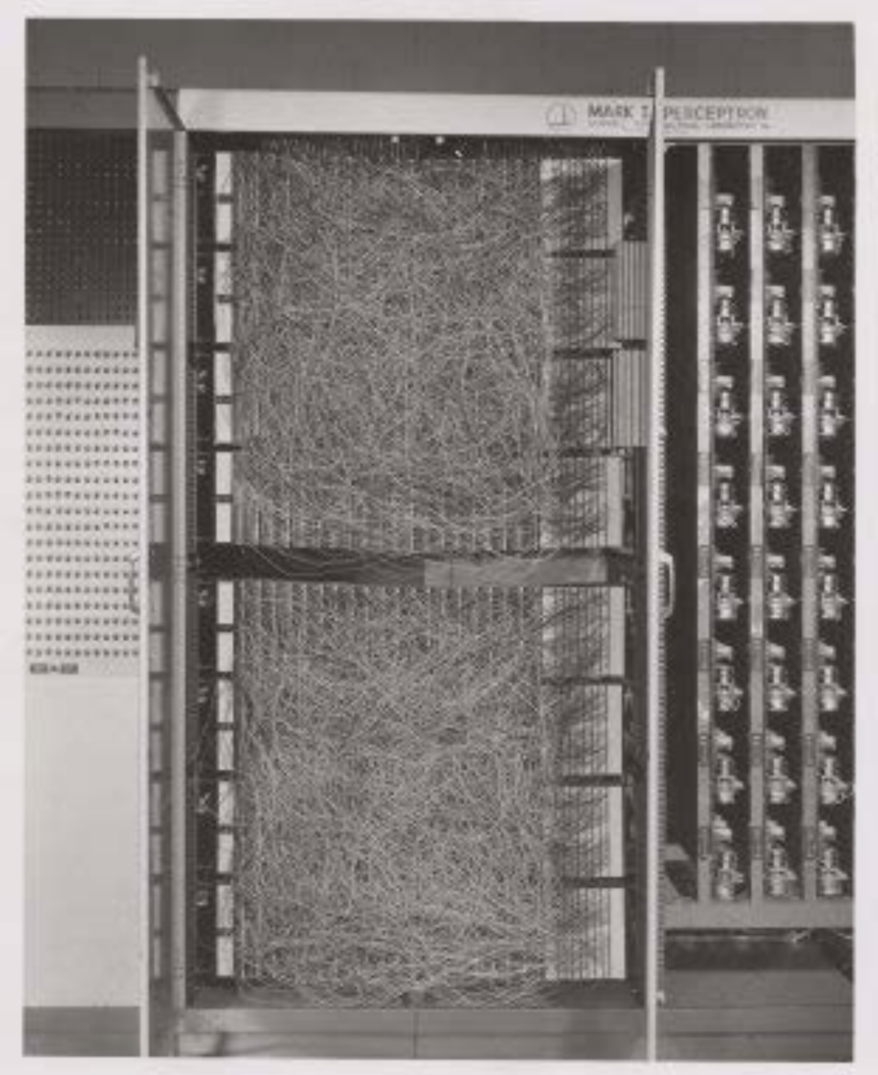
2009 use of GPUs for training deep networks

2011 unsupervised learning of cat from YouTube

2011 deep networks become state-of-the-art for speech recognition

2012 deep networks become state-of-the-art for object recognition

2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.



'Mark I Perceptron at the Cornell Aeronautical Laboratory', hardware implementation of the first Perceptron (Source: Wikipedia / Cornell Library)

deep learning history (abridged)

1957 perceptron learning algorithm

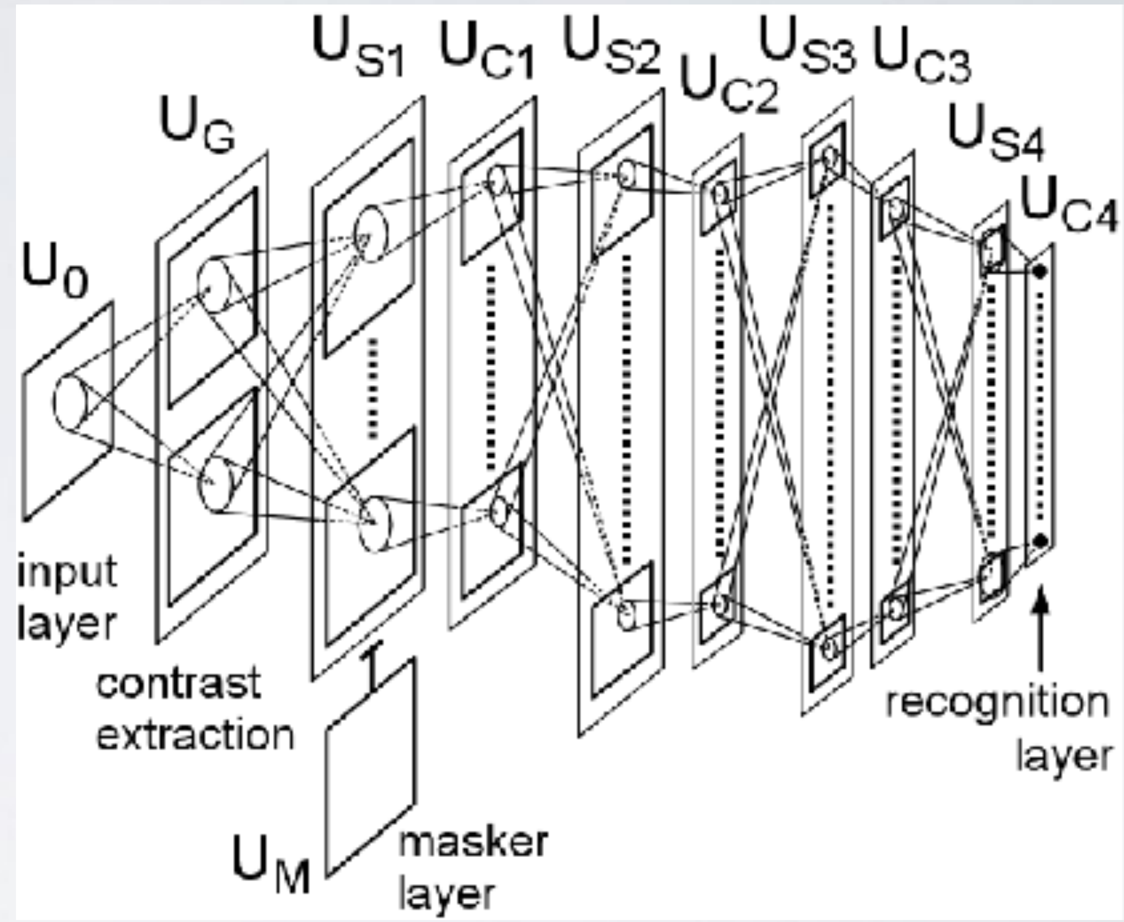
neural net winter

1980 *neocognitron*

1986 backprop becomes popular

1989 convolutional neural networks

neural net winter II



Fukushima, 1980

2006 unsupervised pre-training of deep networks

2009 use of GPUs for training deep networks

2011 unsupervised learning of cat from YouTube

2011 deep networks become state-of-the-art for speech recognition

2012 deep networks become state-of-the-art for object recognition

2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.

deep learning history (abridged)

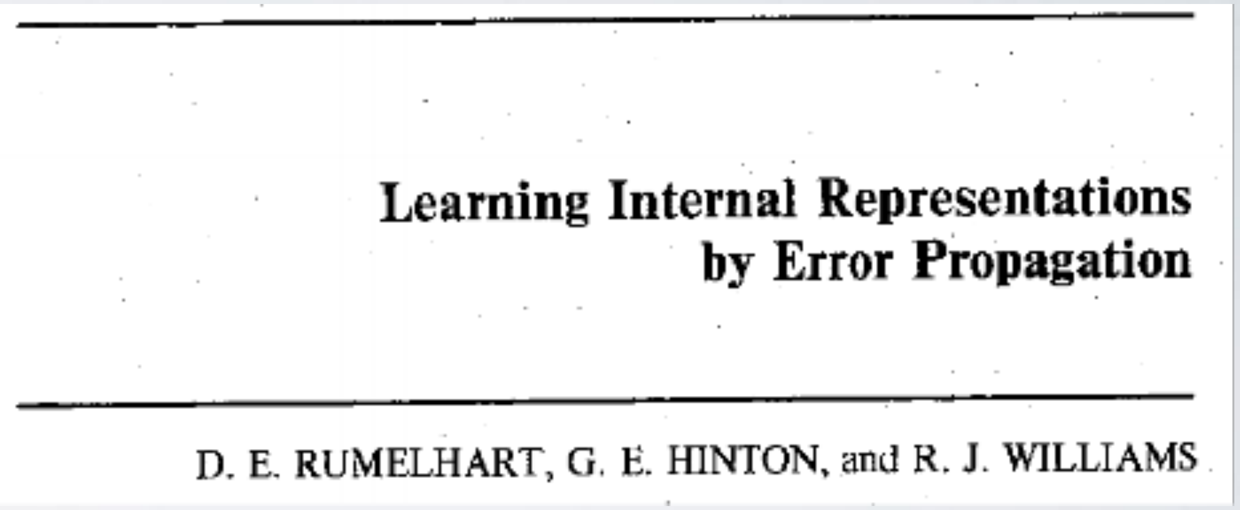
1957 perceptron learning algorithm

neural net winter

1980 neocognitron

1986 *backprop becomes popular*

1989 convolutional neural networks



Learning Internal Representations by Error Propagation

D. E. RUMELHART, G. E. HINTON, and R. J. WILLIAMS

neural net winter II

2006 unsupervised pre-training of deep networks

2009 use of GPUs for training deep networks

2011 unsupervised learning of cat from YouTube

2011 deep networks become state-of-the-art for speech recognition

2012 deep networks become state-of-the-art for object recognition

2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.

deep learning history (abridged)

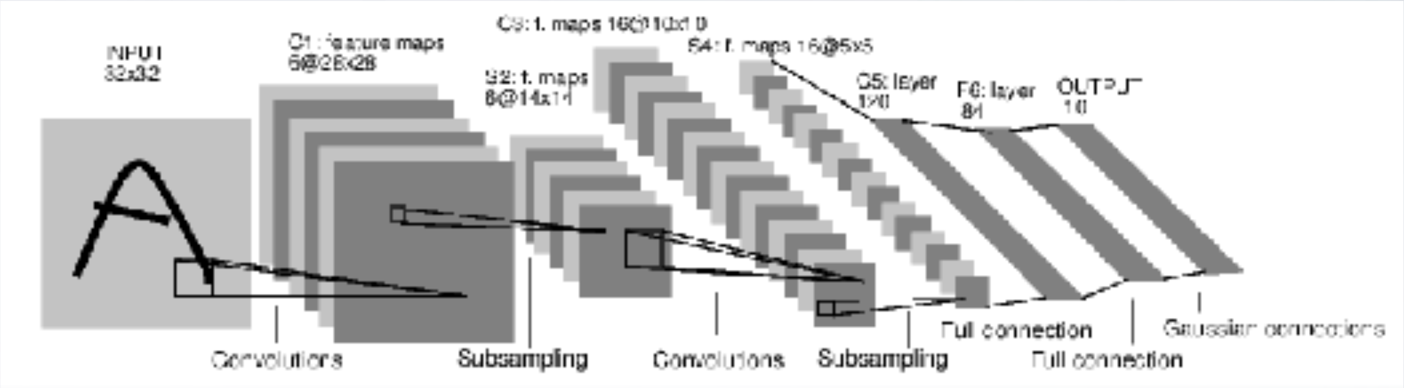
1957 perceptron learning algorithm

neural net winter

1980 neocognitron

1986 backprop becomes popular

1989 *convolutional neural networks*



LeCun, 1998

neural net winter II

2006 unsupervised pre-training of deep networks

2009 use of GPUs for training deep networks

2011 unsupervised learning of cat from YouTube

2011 deep networks become state-of-the-art for speech recognition

2012 deep networks become state-of-the-art for object recognition

2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.

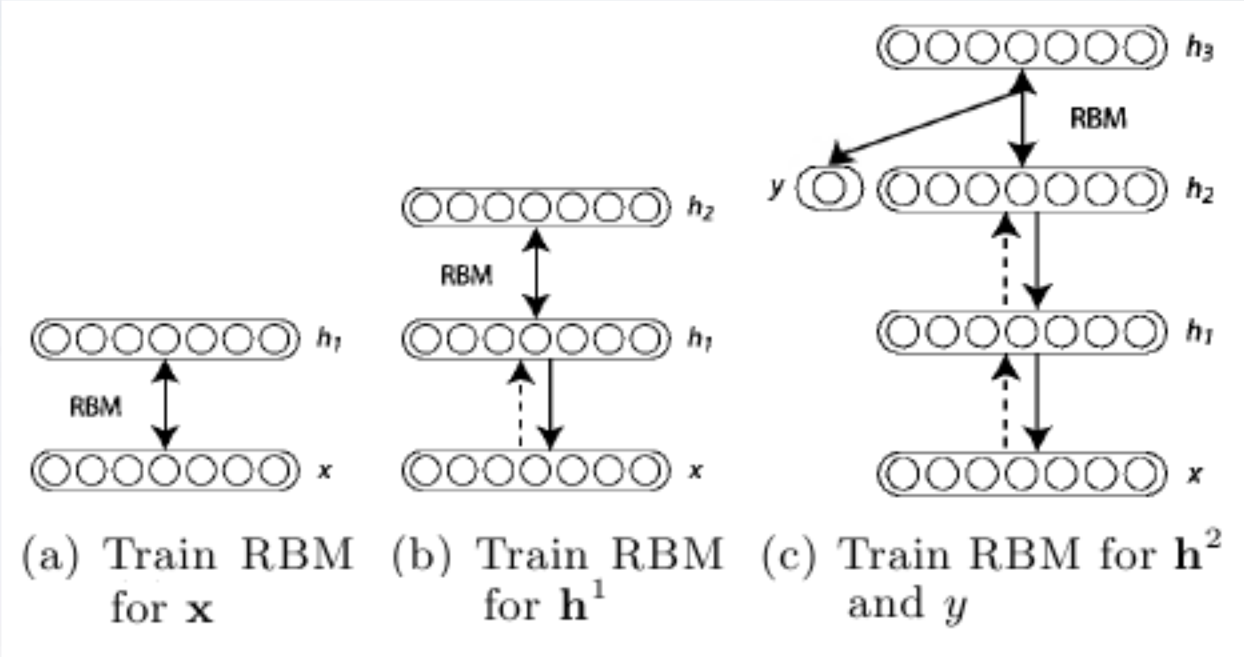
deep learning history (abridged)

1957 perceptron learning algorithm

neural net winter

1980 neocognitron
1986 backprop becomes popular
1989 convolutional neural networks

neural net winter II



Teh et al., 2006

2006 *unsupervised pre-training of deep networks*
2009 use of GPUs for training deep networks
2011 unsupervised learning of cat from YouTube
2011 deep networks become state-of-the-art for speech recognition
2012 deep networks become state-of-the-art for object recognition
2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.

deep learning history (abridged)

1957 perceptron learning algorithm

neural net winter

1980 neocognitron

1986 backprop becomes popular

1989 convolutional neural networks

NVIDIA

neural net winter II

2006 unsupervised pre-training of deep networks

2009 *use of GPUs for training deep networks*

2011 unsupervised learning of cat from YouTube

2011 deep networks become state-of-the-art for speech recognition

2012 deep networks become state-of-the-art for object recognition

2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.

deep learning history (abridged)

1957 perceptron learning algorithm

neural net winter

1980 neocognitron

1986 backprop becomes popular

1989 convolutional neural networks

neural net winter II

cat filter learned from unsupervised learning,
https://googleblog.blogspot.com/2012/06/using-large-scale-brain-simulations-for.html

2006 unsupervised pre-training of deep networks

2009 use of GPUs for training deep networks

2011 *unsupervised learning of cat from YouTube*

2011 deep networks become state-of-the-art for speech recognition

2012 deep networks become state-of-the-art for object recognition

2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.

deep learning history (abridged)

1957 perceptron learning algorithm

neural net winter

1980 neocognitron

1986 backprop becomes popular

1989 convolutional neural networks

Deep Neural Networks for Acoustic Modeling in Speech Recognition

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury

neural net winter II

2006 unsupervised pre-training of deep networks

2009 use of GPUs for training deep networks

2011 unsupervised learning of cat from YouTube

2011 *deep networks become state-of-the-art for speech recognition*

2012 deep networks become state-of-the-art for object recognition

2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.

deep learning history (abridged)

1957 perceptron learning algorithm

neural net winter

1980 neocognitron

1986 backprop becomes popular

1989 convolutional neural networks

neural net winter II

Krizhevsky, et al., 2012

2006 unsupervised pre-training of deep networks

2009 use of GPUs for training deep networks

2011 unsupervised learning of cat from YouTube

2011 deep networks become state-of-the-art for speech recognition

2012 *deep networks become state-of-the-art for object recognition*

2012- deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.

deep learning history (abridged)

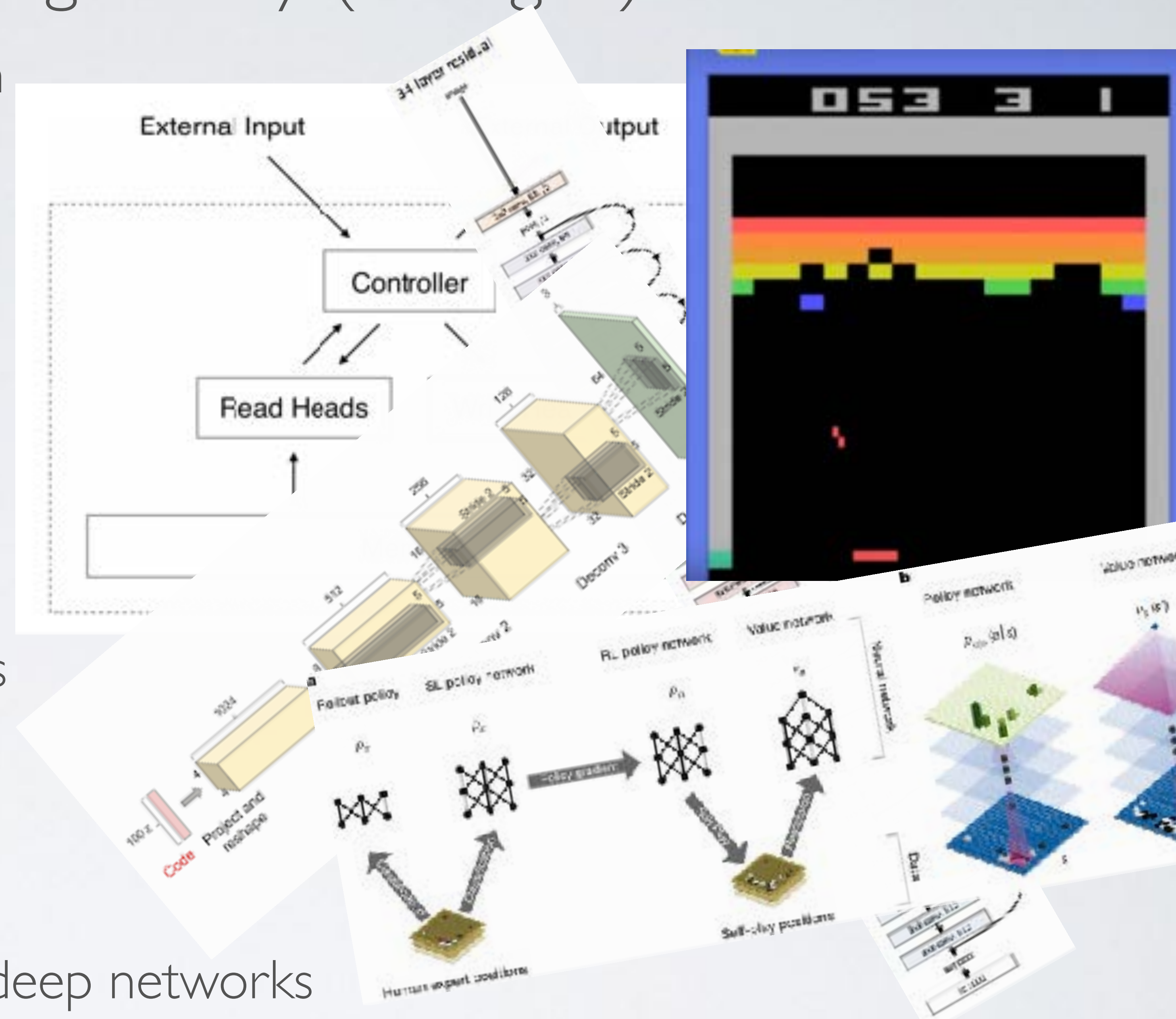
1957 perceptron learning algorithm

neural net winter

1980 neocognitron
1986 backprop becomes popular
1989 convolutional neural networks

neural net winter II

2006 unsupervised pre-training of deep networks
2009 use of GPUs for training deep networks
2011 unsupervised learning of cat from YouTube
2011 deep networks become state-of-the-art for speech recognition
2012 deep networks become state-of-the-art for object recognition
2012- *deep learning boom: ResNets, Neural Turing Machine, deep generative models, deep reinforcement learning, etc.*



He et al., 2015
Graves et al., 2014
Salimans et al., 2016
Minh et al., 2015
Silver et al., 2016
…

recent neural network boom

deep learning is <u>hot</u> right now

what started this boom?

hardware

data

driving factors

software

TensorFlow

theano

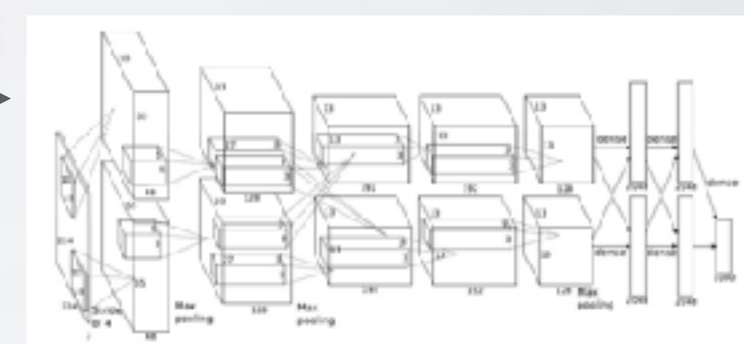
Caffe

K

torch

results

research discoveries



recent neural network boom

who started this boom?

geoff hinton

yann lecun

yoshua bengio

'the big three'

…and others

recent neural network boom

who started this boom?

yann lecun

Citation indices	All	Since 2010
Citations	29582	17815
h-index	77	59
i10-index	179	141

geoff hinton

Citation indices	All	Since 2010
Citations	117128	47516
h-index	113	86
i10-index	273	200

yoshua bengio

Citation indices	All	Since 2010
Citations	32736	25285
h-index	73	65
i10-index	245	200

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-4/

neural network winters

research in neural networks has followed a boom-bust cycle

technological breakthrough

↓

increased excitement, research

↓

A.I. hype
we've finally solved intelligence!
general A.I. is only 5-10 years away!

↓

research doesn't live up to expectations

↓

funding dries up

↓

most researchers move on to other pursuits

neural network winters

we're obviously in a boom period.
<u>is this time any different?</u>

yes and no

predicting the future of A.I. research progress is notoriously difficult

it's *possible* that deep learning research will hit a wall,
where either it becomes too computationally expensive
for most individuals to do groundbreaking research
or a new, better approach comes along

however, deep learning is now commercially successful.
most large tech companies profit from it.
so as long as it remains the state-of-the-art approach,
there will be (funding for) basic research.

Google facebook Microsoft amazon

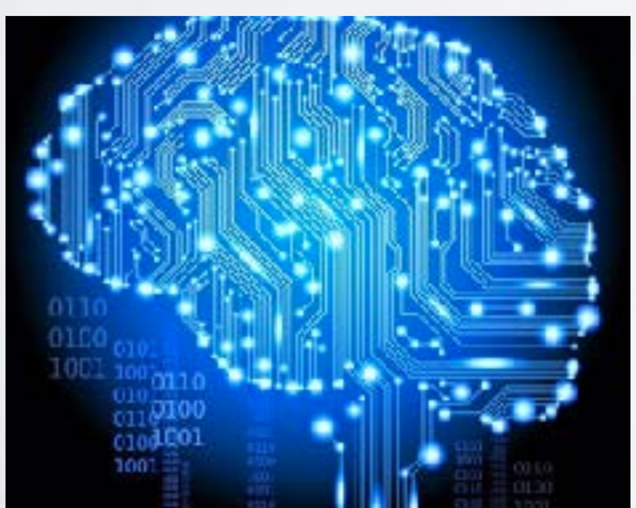
beware the hype

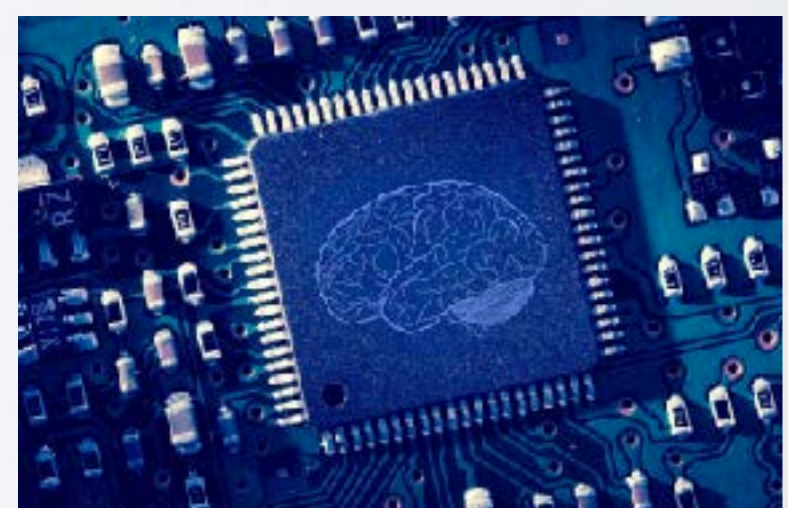
just because deep learning is booming,
it doesn't mean human-level A.I. is 'just around the corner'

saying that something is 5-10 years away is almost
always just pure *speculation*

if someone tries to tell you that deep learning works
off of the same principles as the brain,
tell them that they don't know what they're talking about

cool graphics, but highly inaccurate





does deep learning live up to the hype?

deep learning is nothing more than a passing phase

deep learning is the dawn of general A. I.

non-believers | believers

non-believers	believers
we don't have a great intuition for how or why it works	consistently beats all other methods on vision, NLP benchmarks
models are often uninterpretable a.k.a 'black box'	learn your features instead of hand-coding them
doesn't actually work like the brain	it's biologically inspired!

where does deep learning fit in?

deep learning is a useful method for approximating complicated, hierarchical functions.
this makes it well-suited for many A. I. tasks

but ultimately it's just a tool for linearizing non-linear data.
it doesn't *replace* other machine learning techniques, rather, it *enhances* them

still much work to be done in understanding these models
-what do they learn?
-how do we train them more efficiently?
-architectural principles?

better methods will be developed eventually, but they will almost certainly involve
-hierarchies
-learned features
-many parameters

when to use deep learning?

try a simple method first

deep learning requires *compute* and *data*
unless you have both, deep learning won't work

deep learning works by hierarchically sectioning non-linear surfaces
i.e. deep learning works best on non-linear data with hierarchical structure

why not other methods?

deep learning's power is in its *depth*

most other methods are not capable of depth
or if they are, they are difficult to train

next lecture

convolutional neural networks