Statistical Physics of Machine Learning
The Basic Notions

Tankut Can, Wave Ngampruetikorn & David Schwab

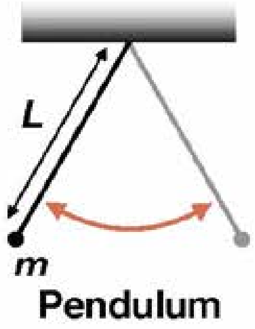
ICTP virtual school 2021

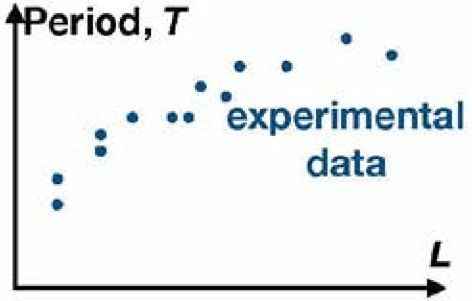
Machine Learning for Condensed Matter

THE GRADUATE CENTER CITY UNIVERSITY OF NEW YORK

Center for the Physics of Biological Function

Two fundamentally different data-analysis paradigms
Estimation vs Prediction





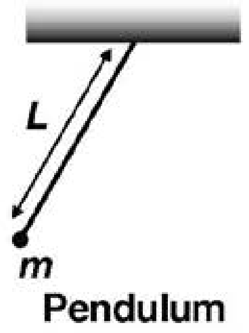
Parameter Estimation

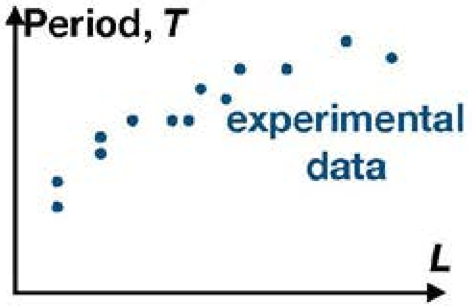
Knowing physics, what is the gravitational acceleration g *in the formula* $T = 2\pi\sqrt{L/g}$*?*

Prediction

Knowing no physics, what are the periods of new pendulums in new experiments?

Machine learning focuses on *prediction*





Pre-data science wants the most accurate estimates of model parameters from data

- Good models require insights into data generating process (e.g., physics)

- Model parameters usually have meanings

ML wants models that can accurately make predictions on *new* data

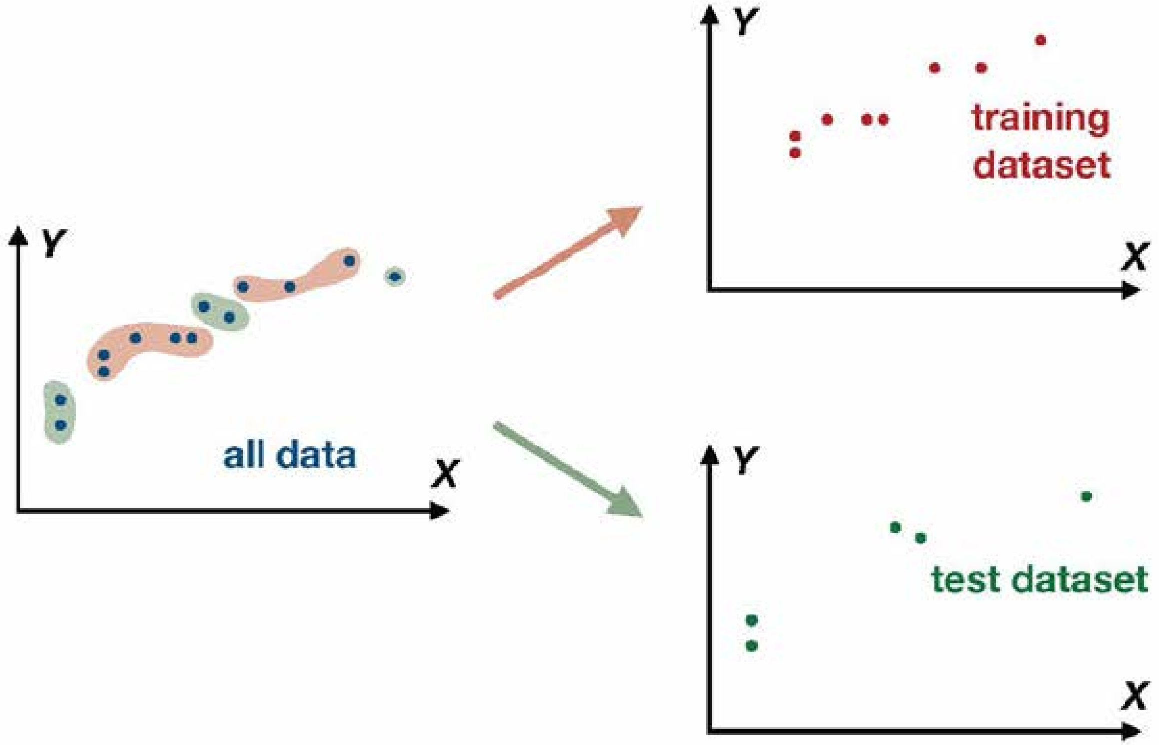
- Model parameters usually do not have obvious meanings on their own

- # of data points ~ # of parameters

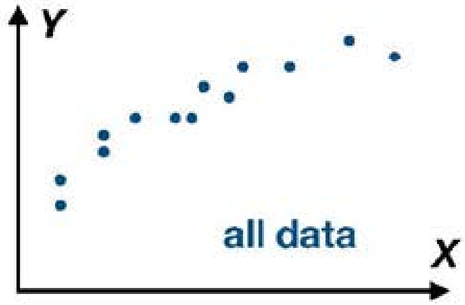
- Models can have millions of

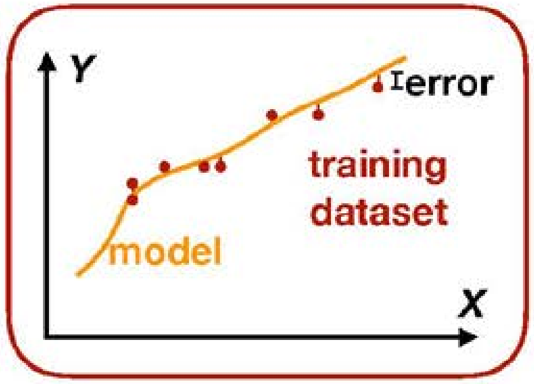
How to measure *prediction performance*?

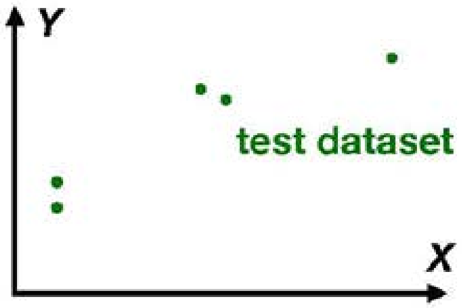
1. Randomly divide data into training and test datasets



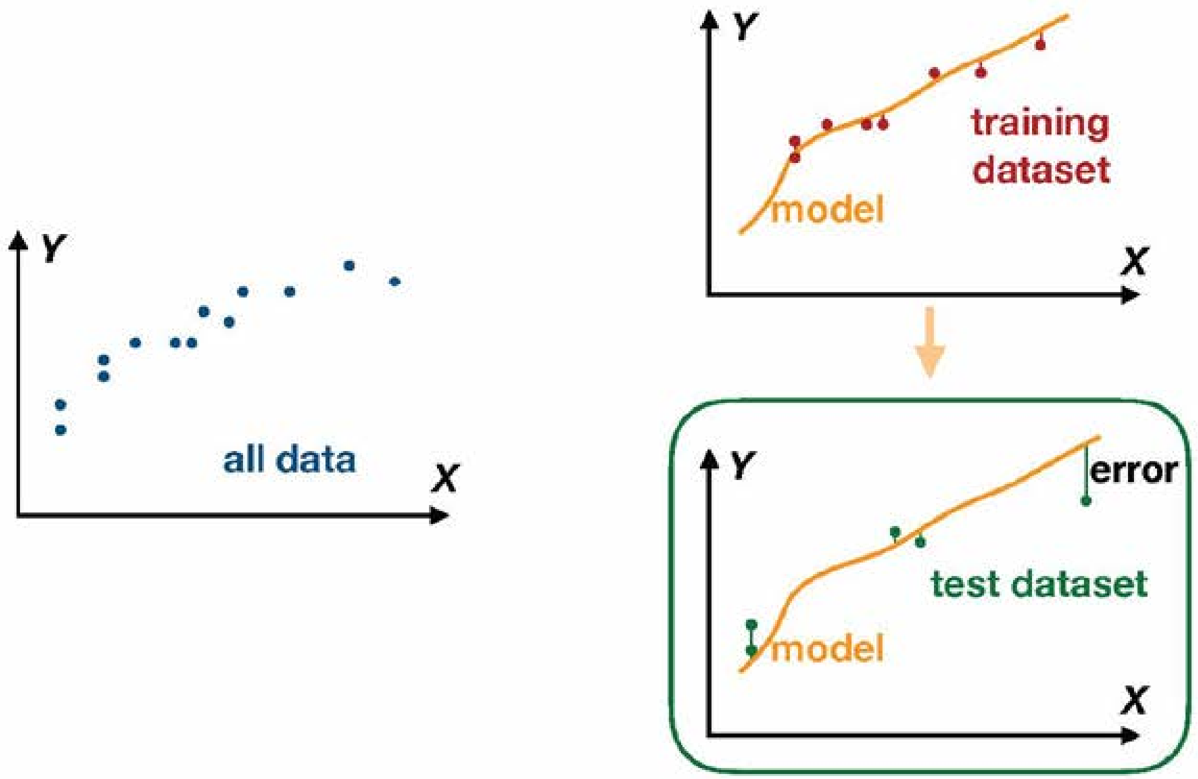
How to measure *prediction performance*?
2. Fit model by minimizing error on training dataset only



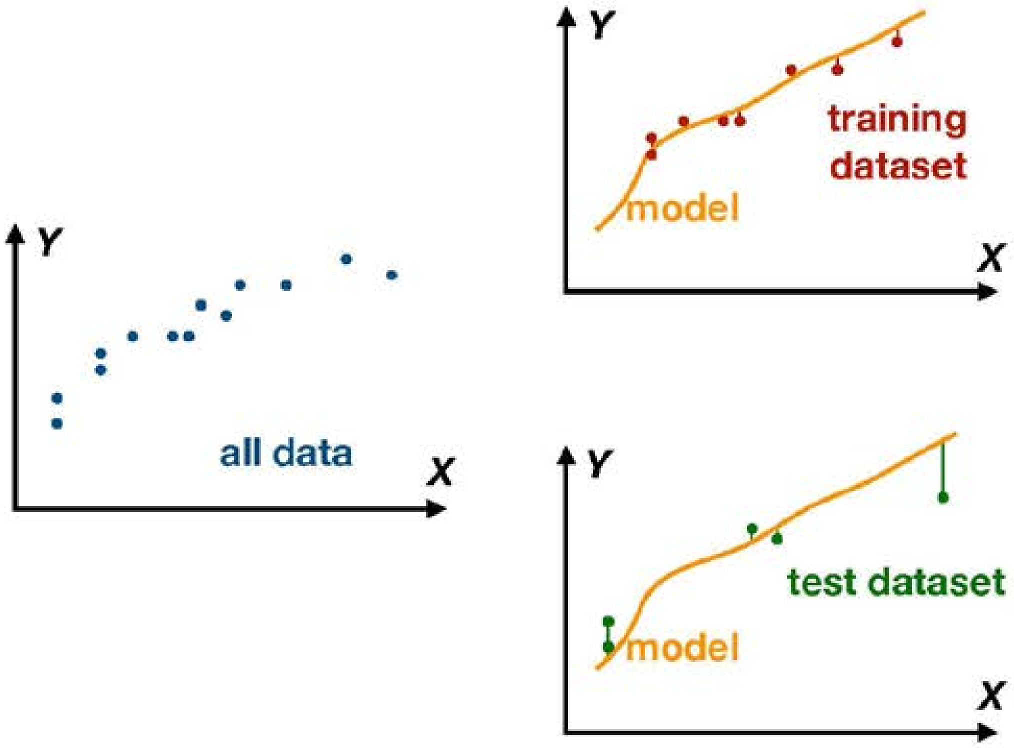




How to measure *prediction performance*?
3. Compute model error on test dataset



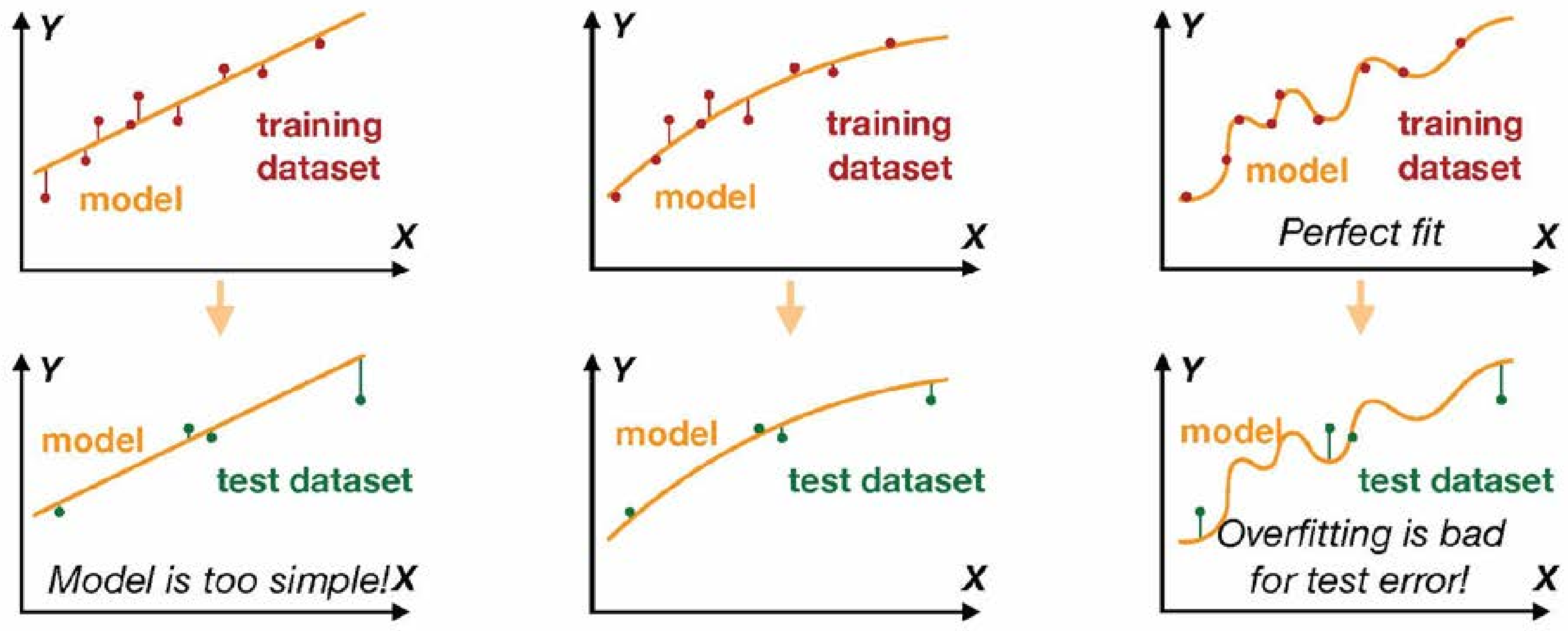
Error on test dataset is an unbiased estimate for *generalization error* (expected error on new data)



1. Randomly divide data into **training dataset** and **test dataset**
2. Fit (train) **model** by minimizing error on **training dataset** only
3. Use error on **test set** as a measure for **prediction performance**

ML goal is not to fit existing data well, but to make accurate predictions on new data

Overfitting is bad for prediction performance



Too simple model
big training error & big test error

More complex model
lower training & test errors

Too complex model
very low training error but big test error

Consider typical model fitting

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots\}$$

Let's compute averaged test error

$$
\begin{aligned}
\langle \text{Test error} \rangle &= \left\langle \sum_x (\overbrace{f(x) + \varepsilon}^{y} - f(x;\hat{\theta}_{\mathcal{D}}))^2 \right\rangle_{\mathcal{D},\varepsilon} \\
&= \left\langle \sum_x \left[\varepsilon^2 + 2\varepsilon(f(x) - f(x;\hat{\theta}_{\mathcal{D}})) + (f(x) - f(x;\hat{\theta}_{\mathcal{D}}))^2\right] \right\rangle_{\mathcal{D},\varepsilon} \\
&= \sum_x \left[\boxed{\langle\varepsilon^2\rangle_\varepsilon + 2\langle\varepsilon\rangle_\varepsilon\langle f(x) - f(x;\hat{\theta}_{\mathcal{D}})\rangle_{\mathcal{D}}} + \left\langle (f(x) - f(x;\hat{\theta}_{\mathcal{D}}))^2 \right\rangle_{\mathcal{D}}\right] \\
&= \sum_x \sigma_\varepsilon^2 + \sum_x \left\langle (f(x) - f(x;\hat{\theta}_{\mathcal{D}}))^2 \right\rangle_{\mathcal{D}}
\end{aligned}
$$

Bias-variance decomposition

$$
\begin{aligned}
\left\langle (f(x) - f(x;\hat{\theta}_{\mathcal{D}}))^2 \right\rangle_{\mathcal{D}} &= \left\langle \left[f(x) - \langle f(x;\hat{\theta}_{\mathcal{D}})\rangle + \langle f(x;\hat{\theta}_{\mathcal{D}})\rangle - f(x;\hat{\theta}_{\mathcal{D}}) \right]^2 \right\rangle \\
&= \left\langle \left[f(x) - \langle f(x;\hat{\theta}_{\mathcal{D}})\rangle \right]^2 \right\rangle + \left\langle \left[f(x;\hat{\theta}_{\mathcal{D}}) - \langle f(x;\hat{\theta}_{\mathcal{D}})\rangle \right]^2 \right\rangle \\
&\quad - 2 \left\langle \left[f(x) - \langle f(x;\hat{\theta}_{\mathcal{D}})\rangle \right] \left[f(x;\hat{\theta}_{\mathcal{D}}) - \langle f(x;\hat{\theta}_{\mathcal{D}})\rangle \right] \right\rangle \\
&= \left[f(x) - \langle f(x;\hat{\theta}_{\mathcal{D}})\rangle \right]^2 + \left\langle \left[f(x;\hat{\theta}_{\mathcal{D}}) - \langle f(x;\hat{\theta}_{\mathcal{D}})\rangle \right]^2 \right\rangle
\end{aligned}
$$

Bias-variance tradeoff

$$\langle \text{Test error} \rangle = \sum_x \sigma_\varepsilon^2 + \sum_x \left[f(x) - \langle f(x;\hat{\theta}_{\mathcal{D}}) \rangle_{\mathcal{D}} \right]^2 + \sum_x \left\langle \left[f(x;\hat{\theta}_{\mathcal{D}}) - \langle f(x;\hat{\theta}_{\mathcal{D}}) \rangle_{\mathcal{D}} \right]^2 \right\rangle_{\mathcal{D}}$$

Primer on Deep Neural Networks (DNNs)

- Benchmarks: What tasks do we want to perform?
- Architectures: What do DNNs look like?
- Learning: How do we train DNNs?
- Failure Modes: What can go wrong?

Benchmark Datasets for Image Classification

CIFAR-10

https://www.cs.toronto.edu/~kriz/cifar.html

50,000 training images
10,000 test images

32x32 color images

best: 99.7% acc.

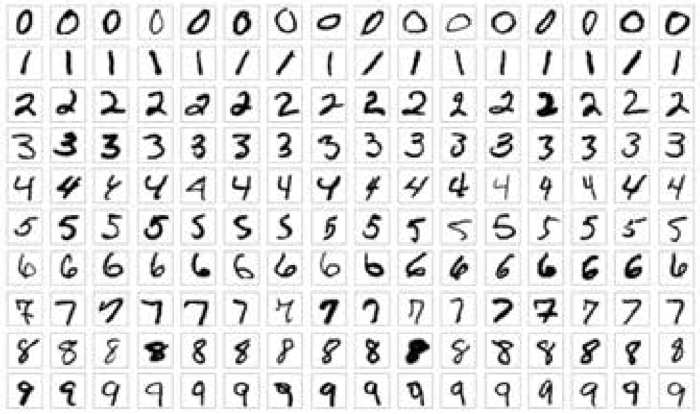
ImageNet image-net.org

1.2 million training images
50,000 validation
150,000 test images

22,000 categories
256x256 color images

best: 85.8% acc.

MNIST Handwritten Digits



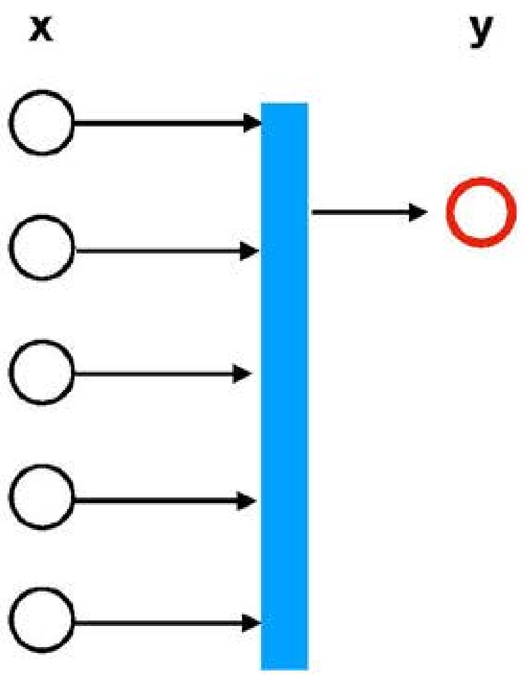
yann.lecun.com/exdb/mnist/

60,000 training images
10,000 test images

28x28 b/w images

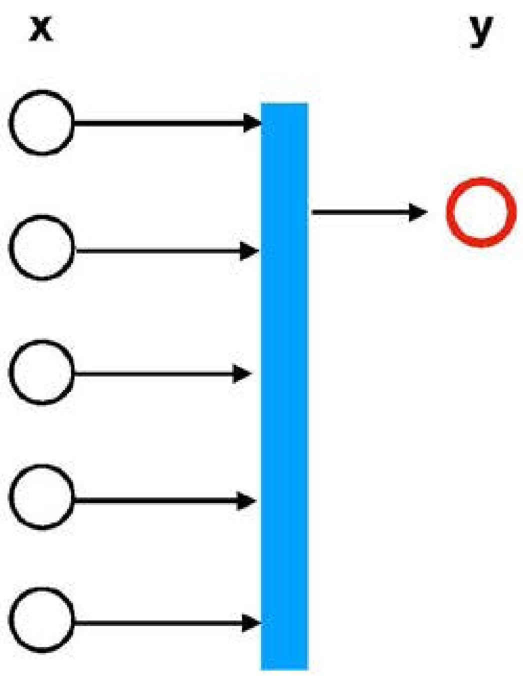
best: 99.84% acc.

Deep Learning Architectures: Fully-Connected Layer



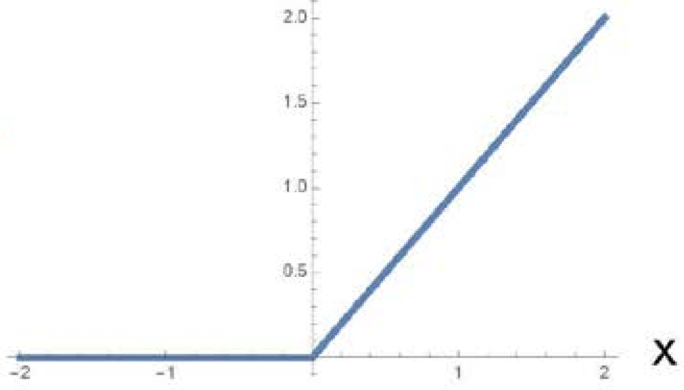
$$y_1 = \sigma\left(\sum_j \underline{W_{1j}} x_j\right)$$

Deep Learning Architectures: Fully-Connected Layer

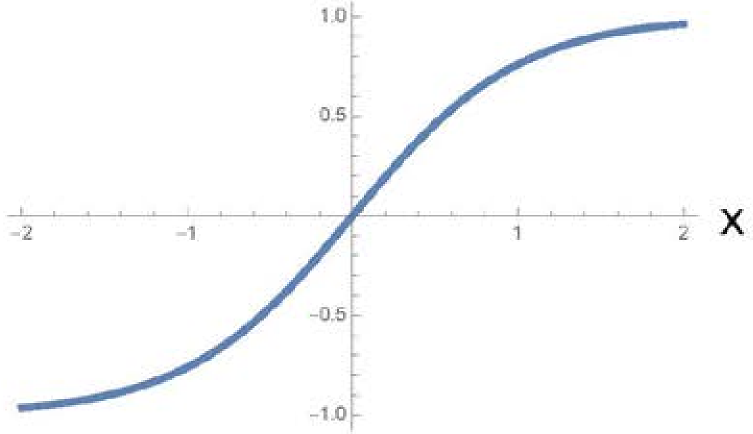


$$y_1 = \sigma\left(\sum_j \underline{W_{1j}} x_j\right)$$

$\sigma(x)$

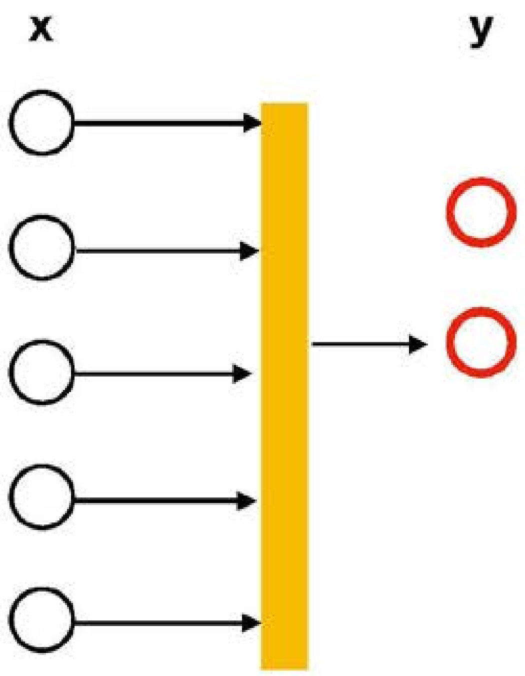


ReLU



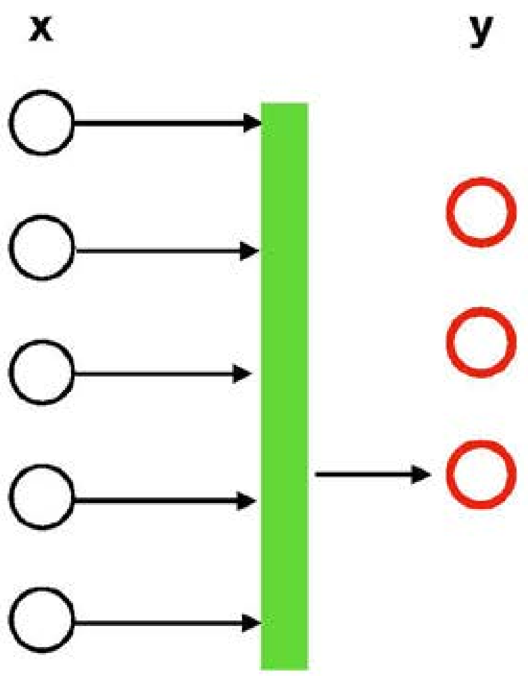
tanh(x)

Deep Learning Architectures: Fully-Connected Layer



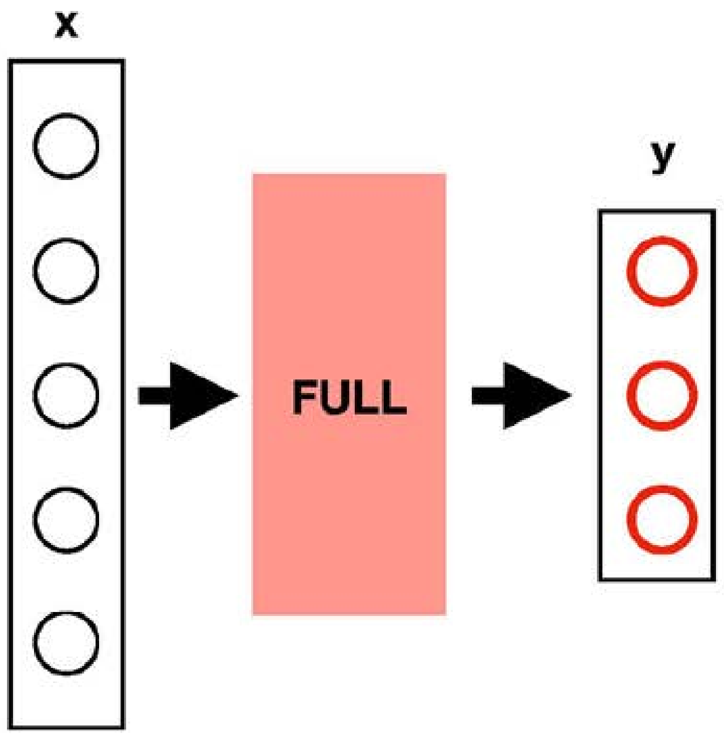
$$y_2 = \sigma\left(\sum_j \underline{W_{2j}} x_j\right)$$

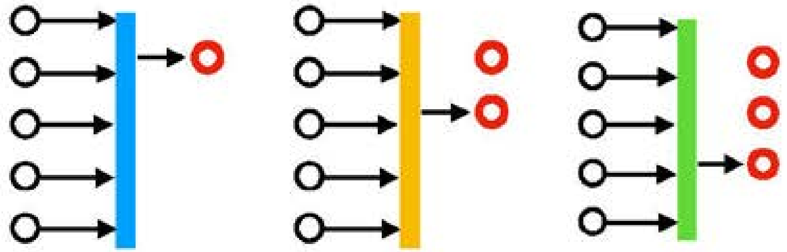
Deep Learning Architectures: Fully-Connected Layer



$$y_3 = \sigma\left(\sum_j \underline{W_{3j}} x_j\right)$$

Deep Learning Architectures: Fully-Connected Layer





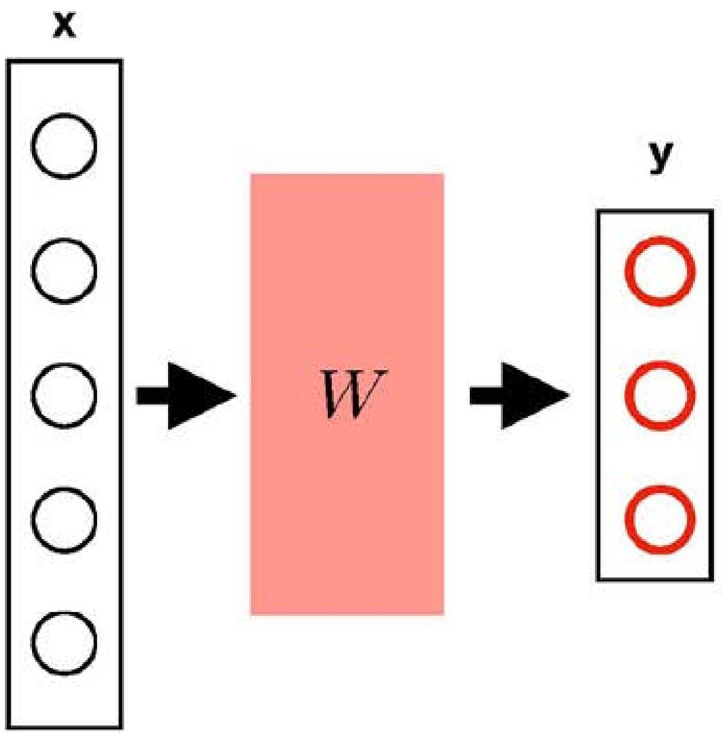
$$\mathbf{y} = \sigma\left(W\mathbf{x}\right)$$

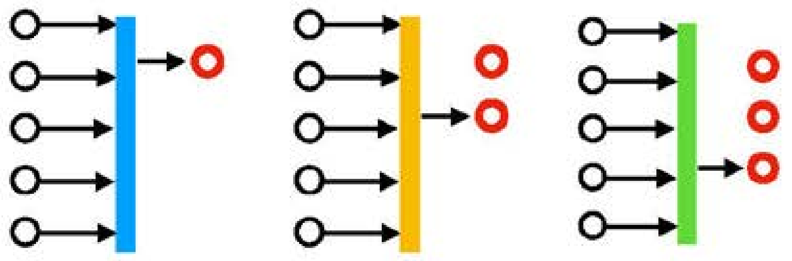
vector notation
W is a matrix.
nonlinearity is applied element-wise

- Parameters in a Fully-Connected Layer:

$$= \begin{bmatrix} --- & W_{1j} & --- \\ --- & W_{2j} & --- \\ --- & W_{3j} & --- \end{bmatrix}$$

Deep Learning Architectures: Fully-Connected Layer





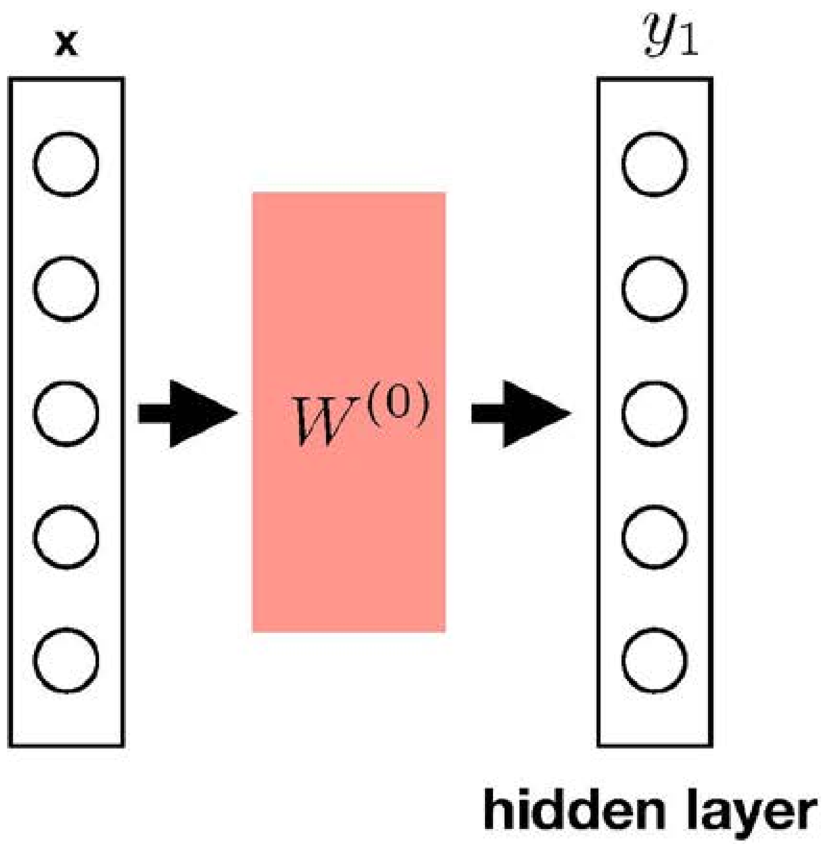
$$\mathbf{y} = \sigma\left(W\mathbf{x}\right)$$

vector notation
W is a matrix.
nonlinearity is applied element-wise

- Parameters in a Fully-Connected Layer:

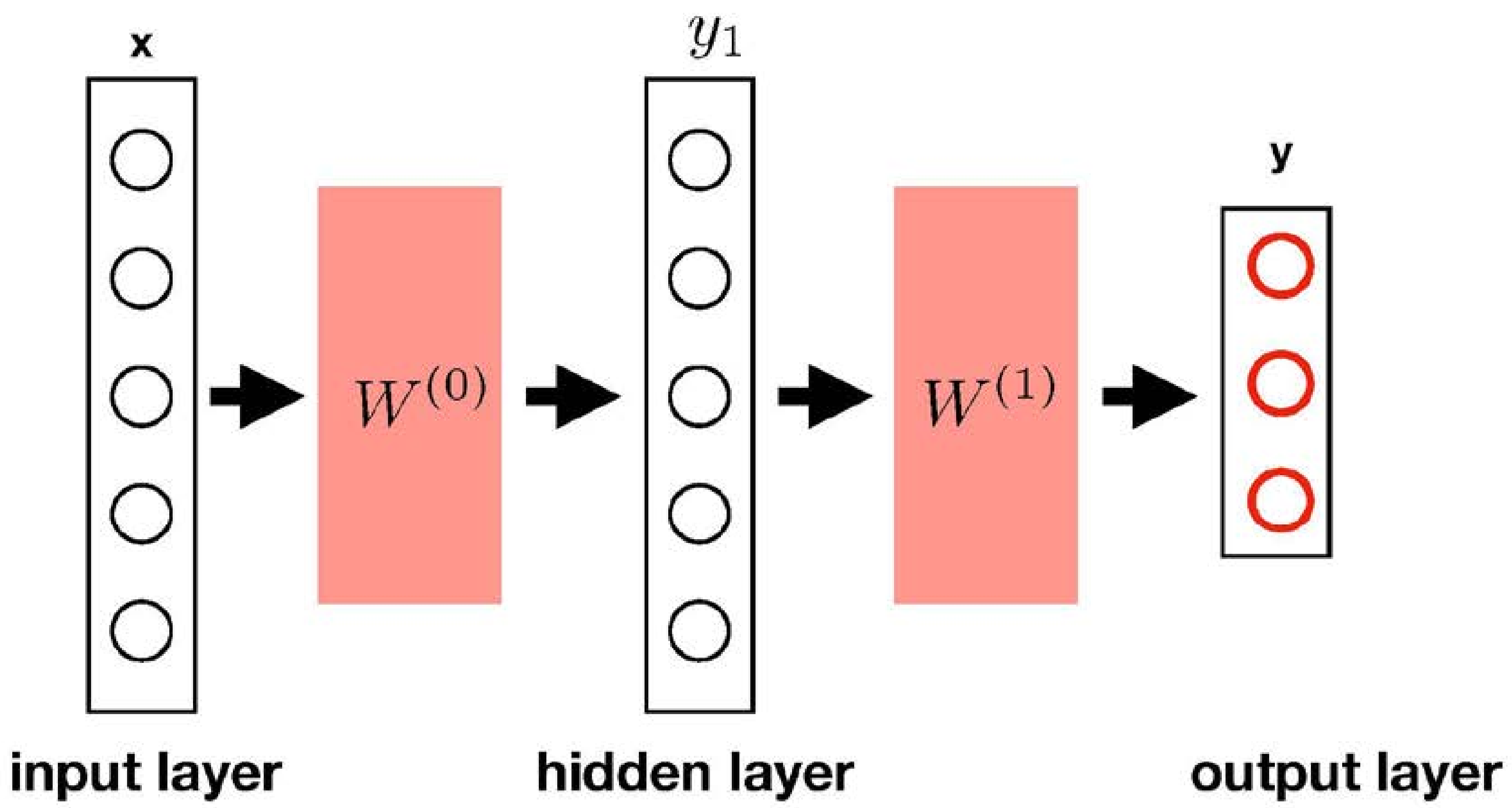
$$= \begin{bmatrix} --- & W_{1j} & --- \\ --- & W_{2j} & --- \\ --- & W_{3j} & --- \end{bmatrix}$$

Deep Learning Architectures: 1 hidden layer



$$y^{(1)} = \sigma(W^{(0)}x)$$

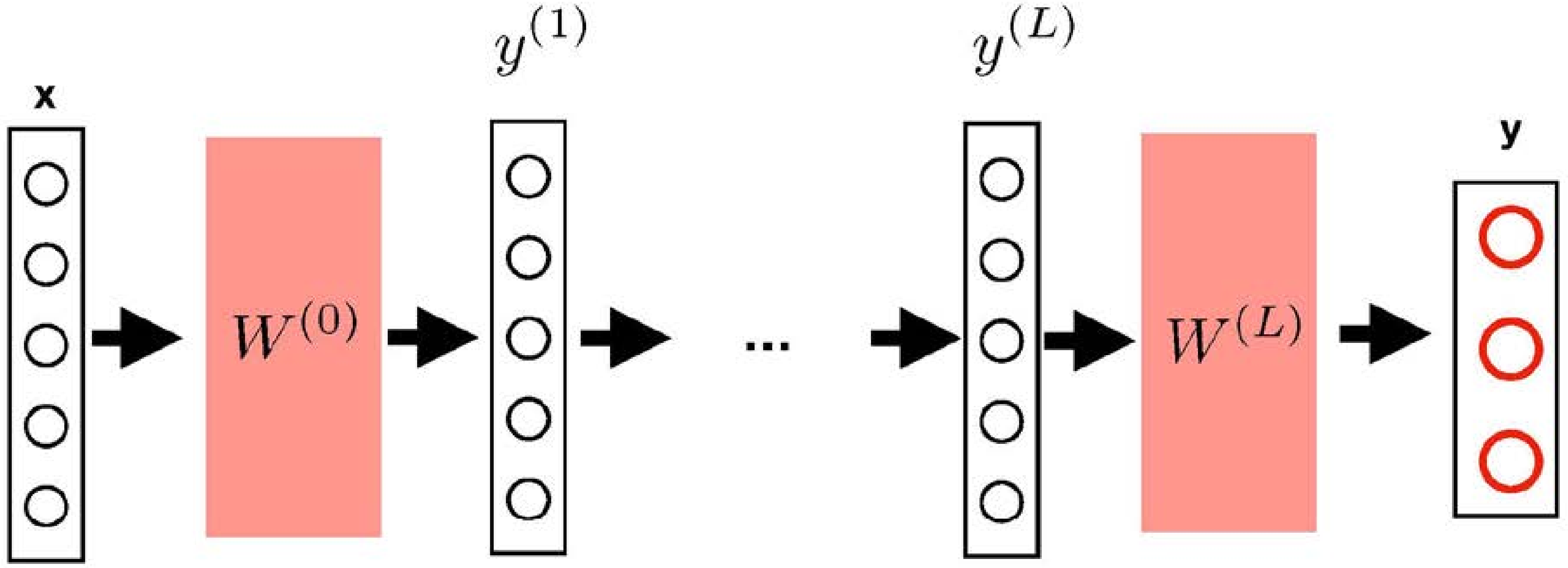
Deep Learning Architectures: 1 hidden layer



$y^{(1)} = \sigma(W^{(0)}x)$

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

Deep Learning Architectures: L hidden layers

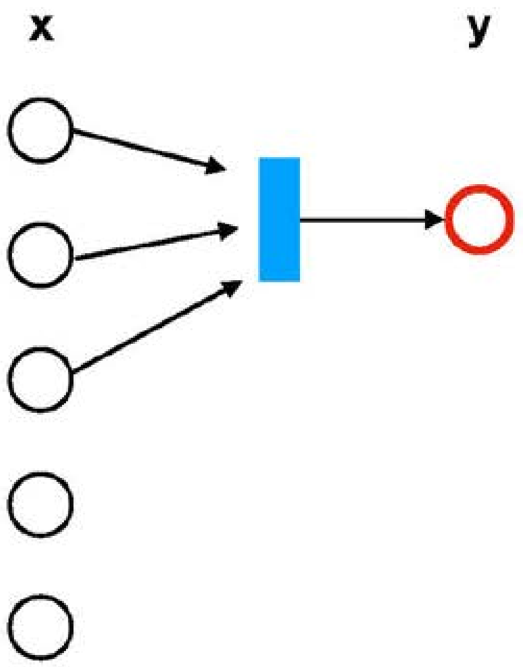


L hidden layers

$$y^{(l+1)} = \sigma(W^{(l)} y^{(l)} + b^{(l)}) \qquad y^{(0)} = x \qquad y^{(L+1)} = y$$

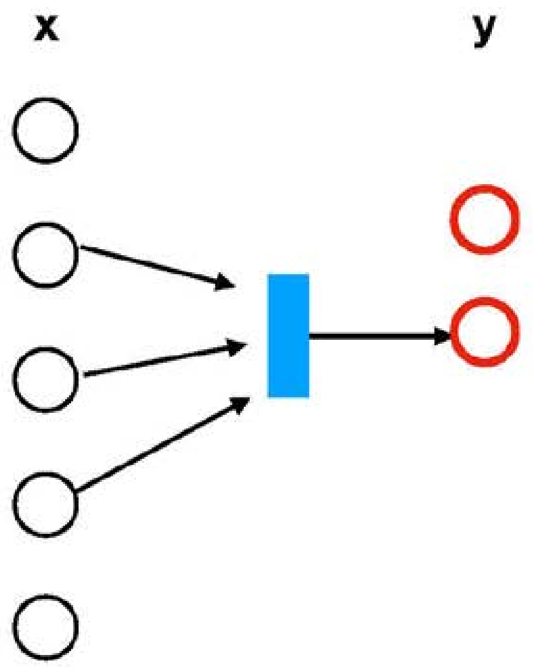
deep net with fully-connected layers is known as a
multilayer perceptron (MLP)

Deep Learning Architectures: Convolutional Layer



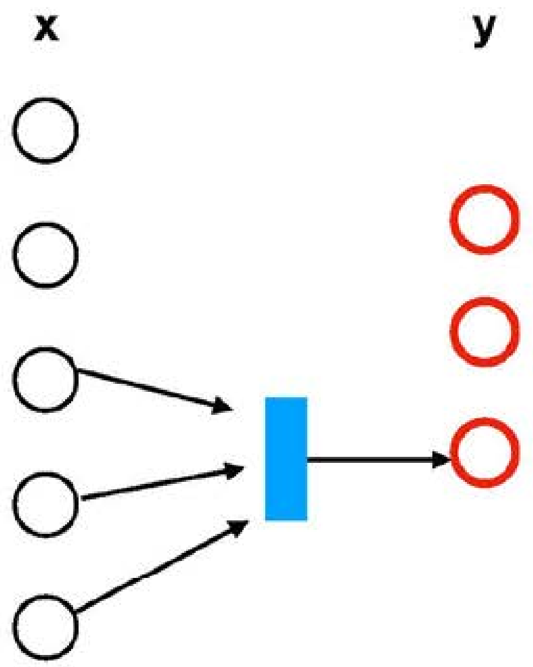
$$y_1 = \sigma\left(\sum_{j=1}^{3} W_j x_j\right)$$

Deep Learning Architectures: Convolutional Layer



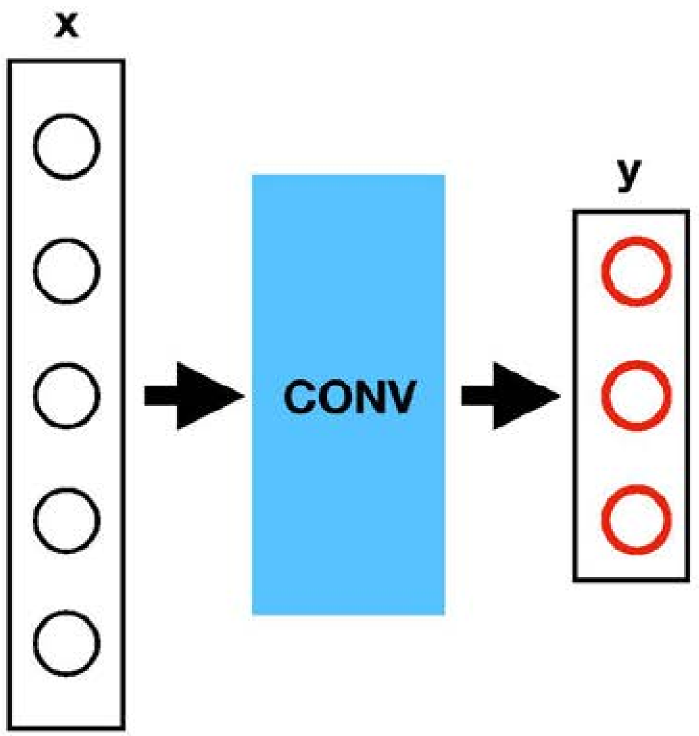
$$y_2 = \sigma\left(\sum_{j=1}^{3} W_j x_{j+1}\right)$$

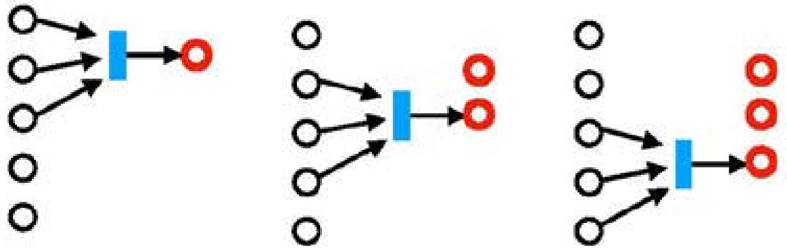
Deep Learning Architectures: Convolutional Layer



$$y_3 = \sigma\left(\sum_{j=1}^{3} W_j x_{j+2}\right)$$

Deep Learning Architectures: Convolutional Layer





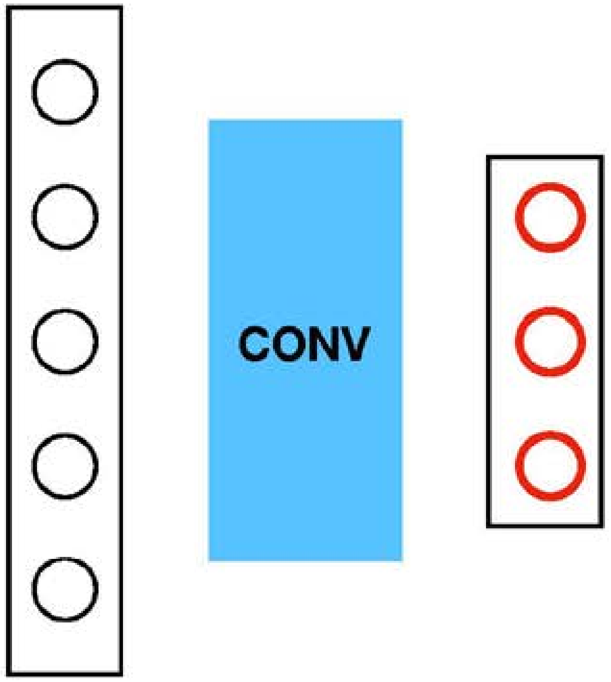
$$y_i = \sigma\left(\sum_{j=1}^{3} W_j x_{j+i-1}\right)$$

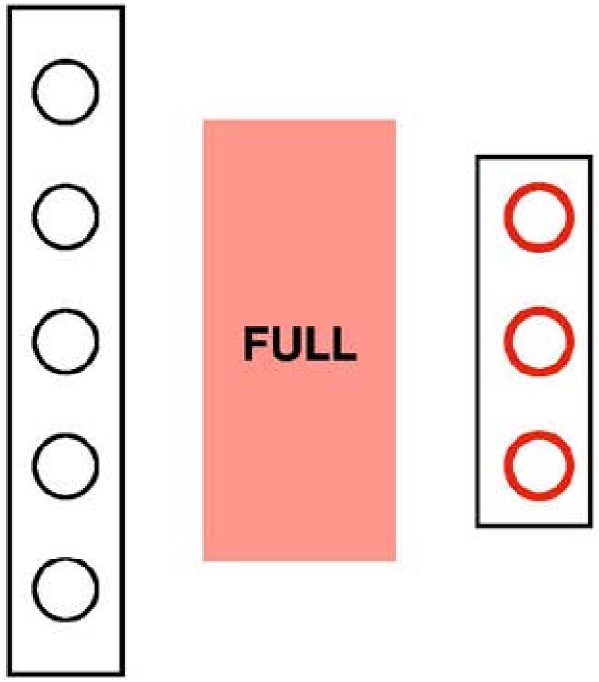
- Parameters in a convolution layer: the "convolutional filter"

$$\blacksquare = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix}$$

- Hyperparameter: filter size (=3 in this example)

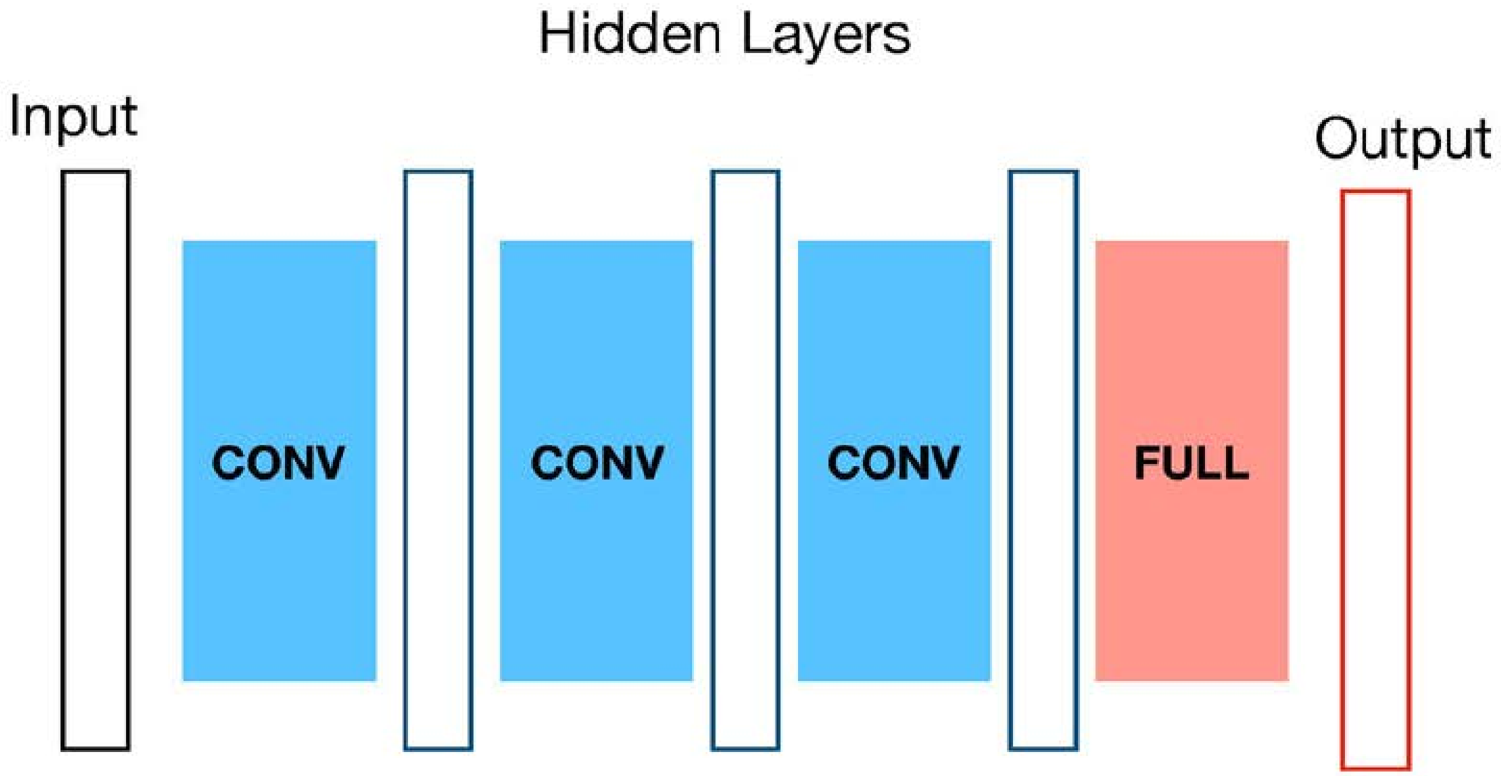
Deep Learning Architectures





- Convnets also have some additional layers: Max Pooling and average pooling. These basic ingredients are slightly adjusted to deal with images (2D array of pixels).

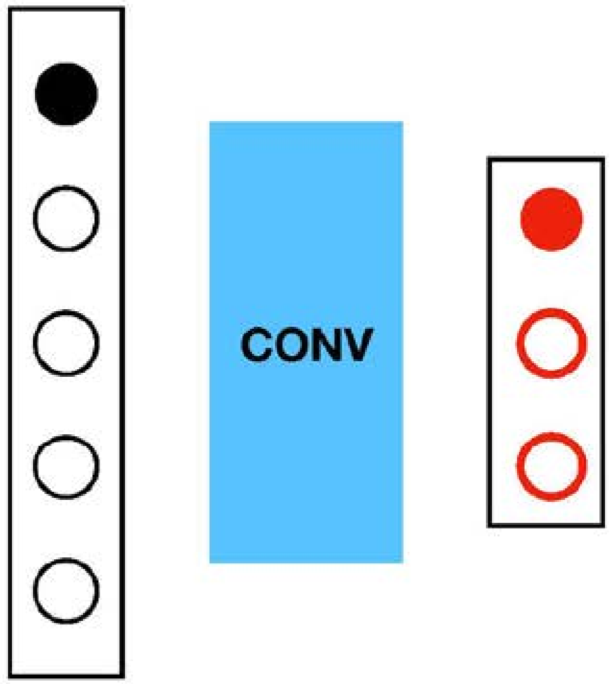
Deep Learning Architectures



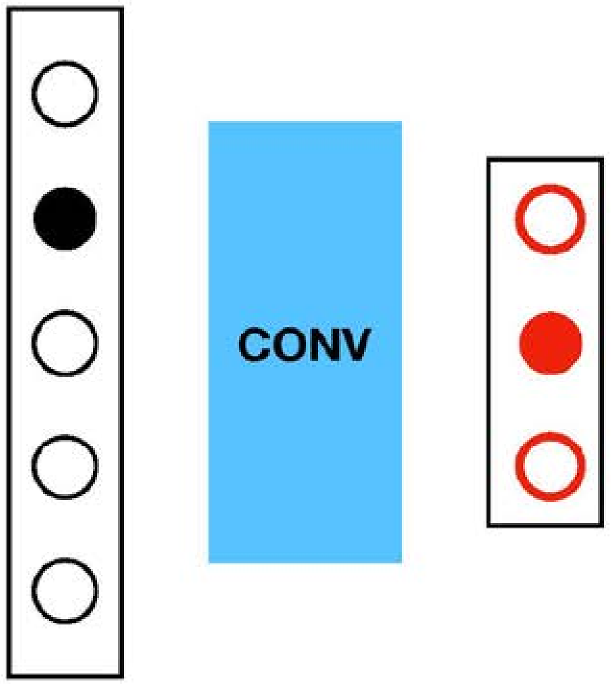
- Putting all of these layers together, end up with a parameterized function which takes an input (e.g. image), and gives an output (e.g. vector of probabilities)
- Architectural choices informed by structure of the data, trainability, practical considerations (storage), etc.

Example: Convolutional layers can track patterns that have been translated

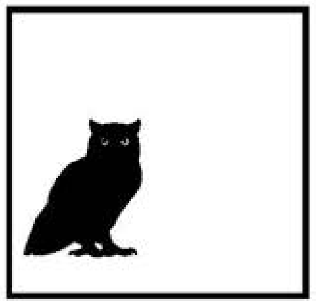
If



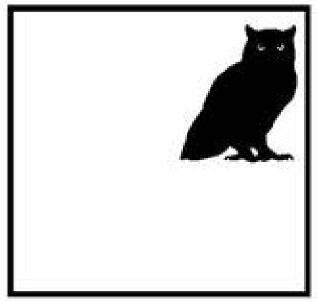
Then



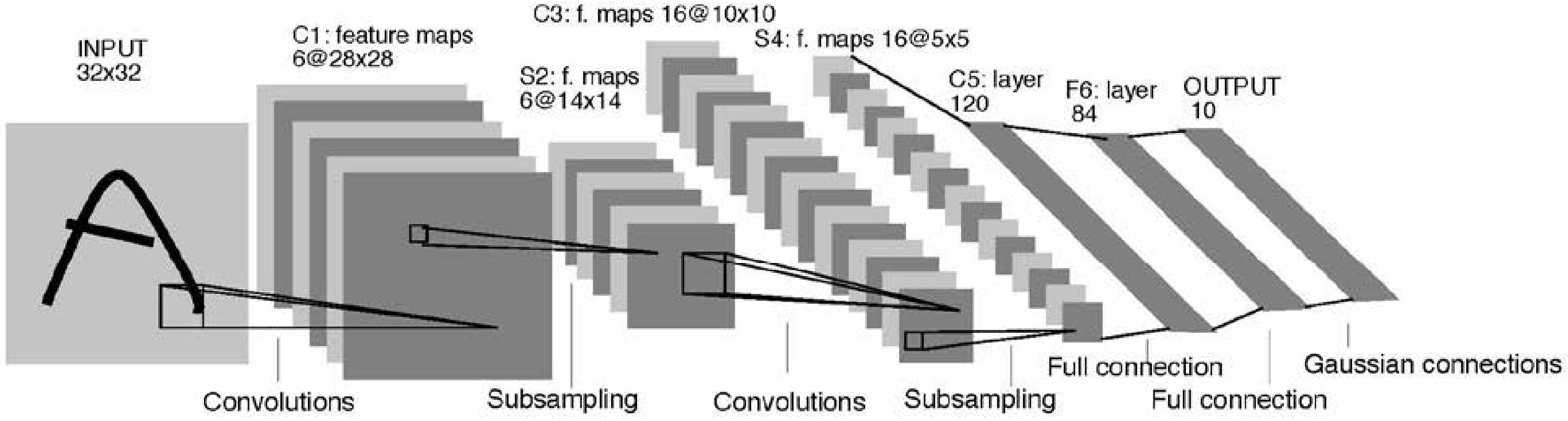
This is useful for images:



gives same output as



Example: LeNet



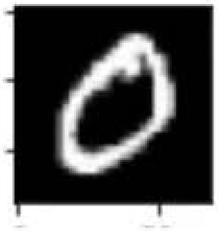
LeCun et al. 1998

Training DNNs: Supervised Learning

Training Data (x_a, y_a) $a = 1, ..., N$

example: MNIST

$x =$ $y = 5$

$x =$  $y = 0$

vectorized array of grayscale pixels [-1,1]

Training DNNs: Supervised Learning

Training Data (x_a, y_a) $a = 1, ..., N$

Loss Function (aka training error, empirical risk, objective function)

$$E = \sum_{a=1}^{N} ||y(x_a; \underset{\uparrow}{W}) - y_a||^2$$

parameters of the neural network

Loss on a single training sample $E_a = ||y(x_a; W) - y_a||^2$

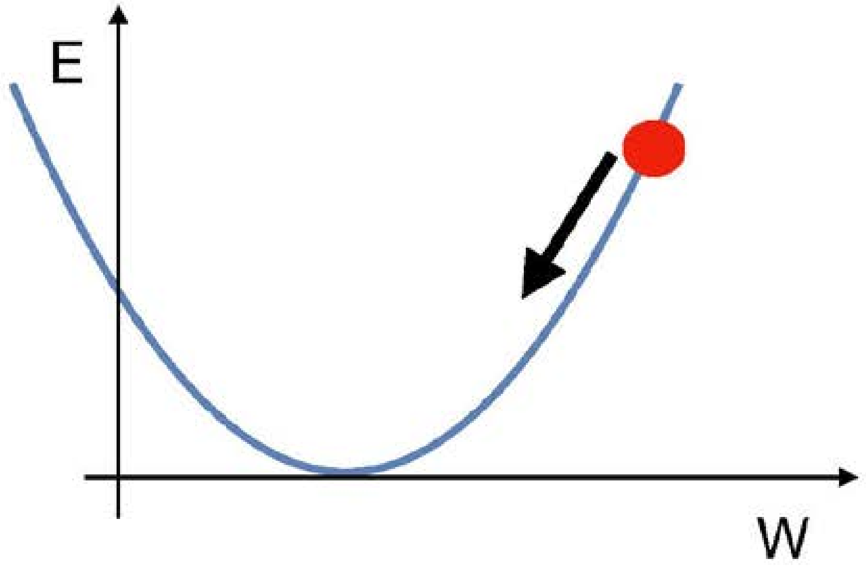
Training DNNs: Gradient Descent

Loss Function (aka training error, empirical risk, objective function)

$$E = \sum_{a=1}^{N} ||y(x_a; W) - y_a||^2$$ Training Data (x_a, y_a)

Loss on a single training sample $E_a = ||y(x_a; W) - y_a||^2$

Loss Surface



Gradient Descent takes you to a local minimum

$$\Delta W = -\eta \frac{dE}{dW}$$

$$\Delta W = -\eta \left(\frac{1}{N} \sum_{a=1}^{N} \frac{dE_a}{dW} \right)$$

Backpropagation to compute gradients

Example: Multilayer perceptron

$$y^{(l+1)} = \sigma(z^{(l)}), \quad z^{(l)} = W^{(l)} y^{(l)}$$

Depends on layer weights in this compositional manner

$$y = \sigma(W^{(L)} \sigma(W^{(L-1)} \ldots \sigma(\overbrace{W^{(l)} y^{(l)}}^{z^{(l)}})))$$

Gradient of loss obtained by chain rule

$$\frac{\partial E}{\partial W_{ij}^{(l)}} = \frac{\partial E}{\partial z_i^{(l)}} y_j^{(l)}$$

Backpropagation to compute gradients

$$\frac{\partial E}{\partial z_k^{(l)}} = \frac{\partial E}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial z_k^{(l)}}$$

$$= \frac{\partial E}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial y_{k''}^{(L)}} \frac{\partial y_{k''}^{(L)}}{\partial z_k^{(l)}}$$

$$= \frac{\partial E}{\partial y_{k'}} \left(\frac{\partial y}{\partial y^{(L)}} \frac{\partial y^{(L)}}{\partial y^{(L-1)}} \cdots \frac{\partial y^{(l+2)}}{\partial y^{(l+1)}} \right)_{k'k''} \frac{\partial y_{k''}^{(l+1)}}{\partial z_k^{(l)}}$$

$$= \frac{\partial E}{\partial y_{k'}} \left(\frac{\partial y}{\partial y^{(L)}} \frac{\partial y^{(L)}}{\partial y^{(L-1)}} \cdots \frac{\partial y^{(l+2)}}{\partial y^{(l+1)}} \right)_{k'k} \sigma'(z_k)$$

Gradient of loss is product of matrices (Jacobian):

$$\frac{\partial y_k^{(l+1)}}{\partial y_{k'}^{(l)}} = \sigma'(z_k^{(l)}) W_{kk'}$$

Backpropagation to compute gradients: exploding/vanishing gradients problem

The product of many matrices tend to vanish or explode, depending on the largest eigenvalue

$$\left|\left|\frac{\partial y^{(l+1)}}{\partial y^{(l)}}\right|\right| < 1$$ Gradients tend to vanish for deep nets

$$\left|\left|\frac{\partial y^{(l+1)}}{\partial y^{(l)}}\right|\right| > 1$$ Gradients tend to blow-up for deep nets

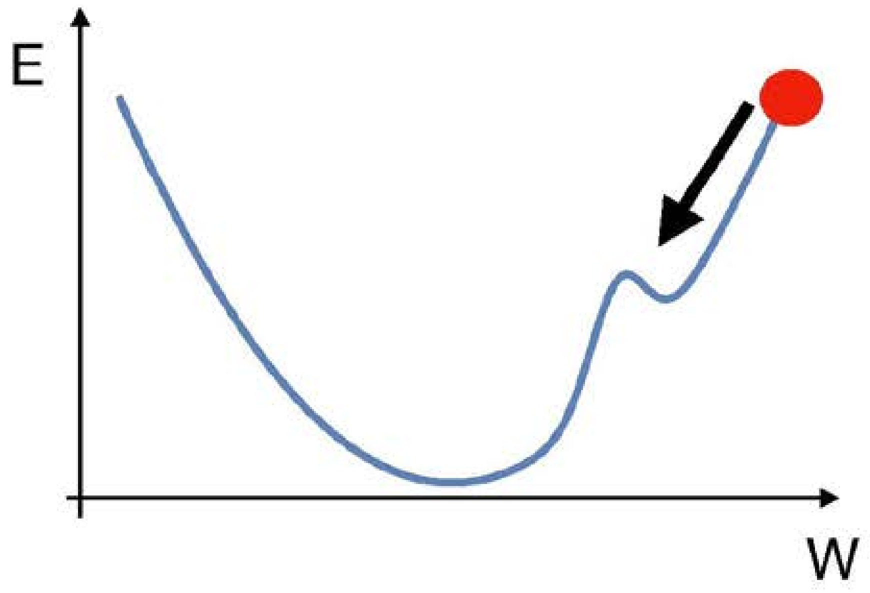
Lots of tricks to help with this issue, and thus make training possible

- Batch Norm
- Careful initialization
- Skip connections (Resnets)

Training DNNs: Stochastic Gradient Descent

Training sets can be too large to efficiently compute full gradient

Loss Function can be complicated, and might have bad local minima



Solution: introduce noise by computing gradient on a mini-batch

Training DNNs: Stochastic Gradient Descent

1) Choose Subset of B Training examples

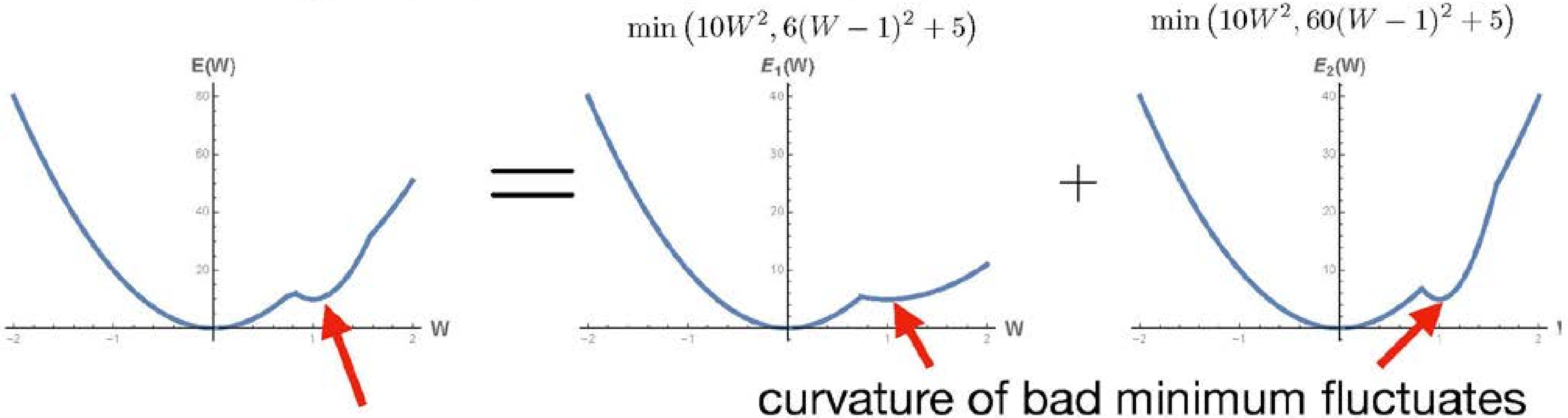
2) Update weights using gradient on loss on this subset

$$\Delta W = -\eta \left(\frac{1}{B} \sum_{a_1=1}^{B} \frac{dE_{a_1}}{dW} \right)$$

3) Choose a different subset of training examples (which you haven't seen), and repeat (2).

This is much more computationally efficient, and has the advantage that the samples of the gradient from different “batches” introduces noise that appears to kick the training out of bad local minima.

Toy Model for Stochastic Gradient Descent



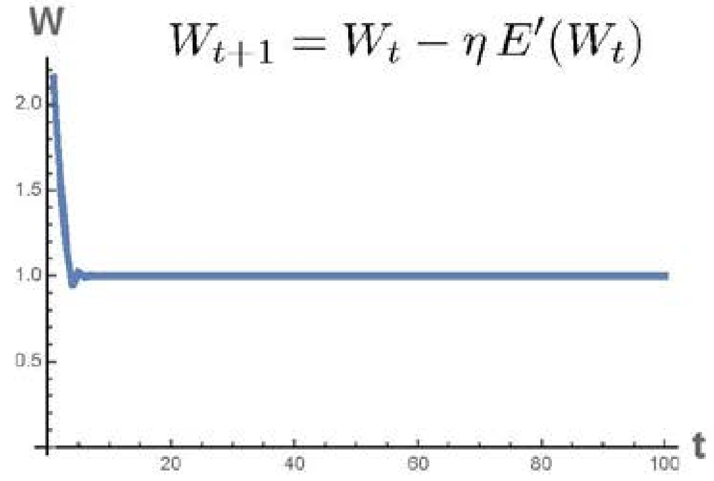
curvature of bad minimum fluctuates between samples

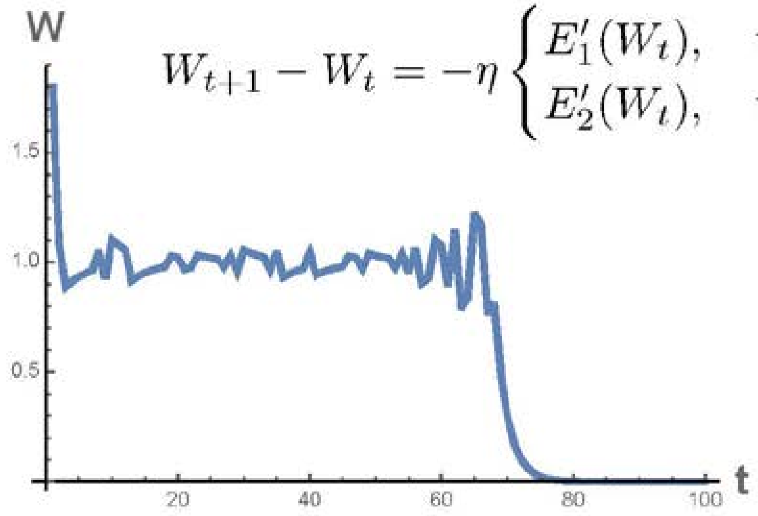
Gradient descent gets stuck at a bad local minimum at W = 1

Stochastic gradient descent escapes the bad minimum (for the right learning rate)

$$W_{t+1} = W_t - \eta\, E'(W_t)$$

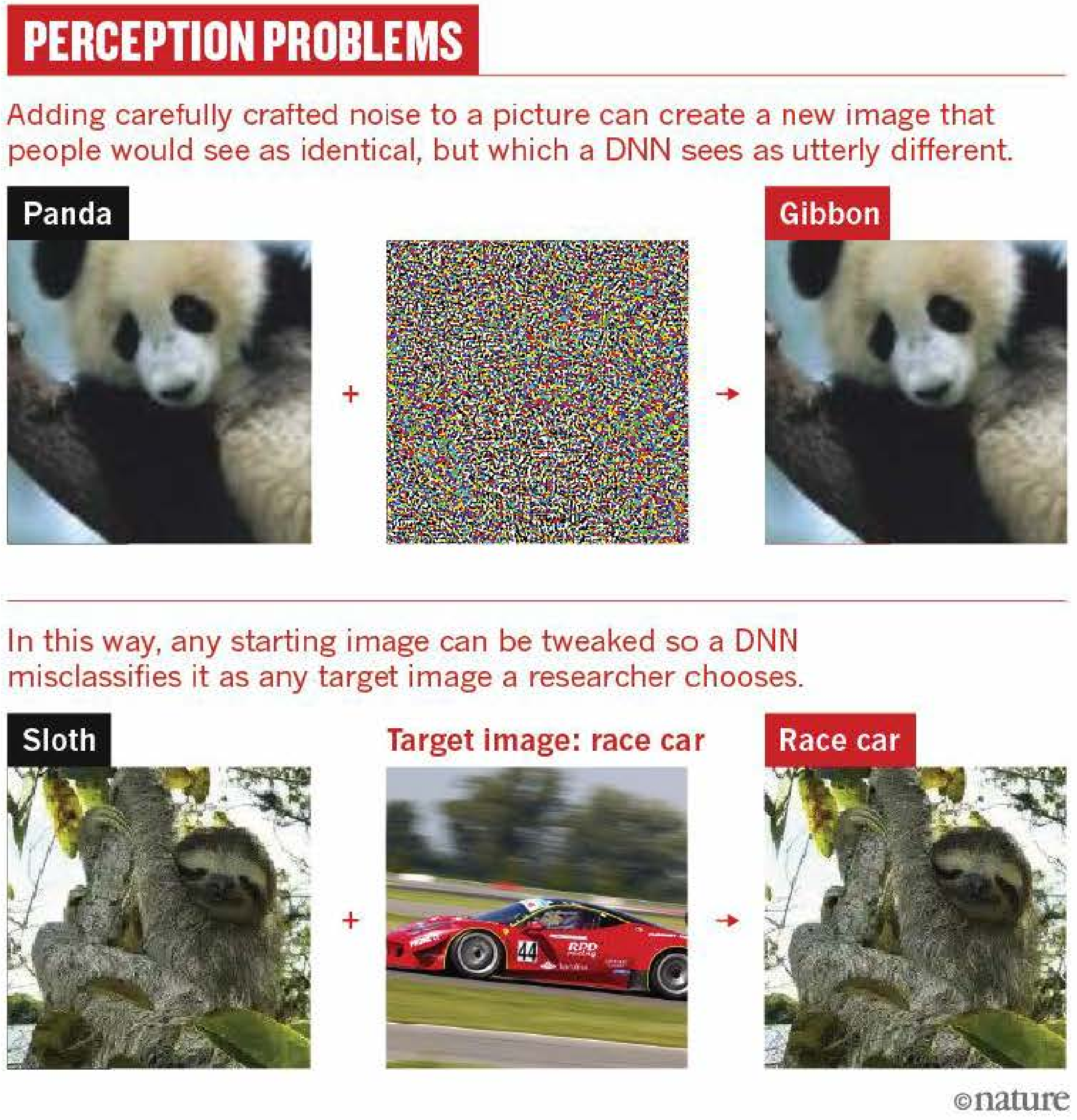
$$W_{t+1} - W_t = -\eta \begin{cases} E_1'(W_t), & \text{w/ prob.} = 1/2 \\ E_2'(W_t), & \text{w/ prob.} = 1/2 \end{cases}$$





see also Wu, Ma, Weinan E, "How SGD selects the global minima.." 2018

Fooling DNNs: Adversarial Examples


PERCEPTION PROBLEMS

Adding carefully crafted noise to a picture can create a new image that people would see as identical, but which a DNN sees as utterly different.

In this way, any starting image can be tweaked so a DNN misclassifies it as any target image a researcher chooses.

D. Heaven, 'Why deep-learning AIs are so easy to fool', Nature 574, 163-166 (2019)

see also Adversarial Examples are not Bugs, they are Features: https://gradientscience.org/adv/

Additional References

Introductory books and review articles on machine learning and neural networks:

- Bishop - Pattern Recognition and Machine Learning
- MacKay - Information Theory, Inference, and Learning Algorithms (available free online)
- Mehta et al. A high-bias, low-variance introduction to ML for physicists (https://arxiv.org/abs/1803.08823)

A good review on the connections between statistical physics, dynamical systems, and deep learning, with lots of references:

Bahri et al. “Statistical Mechanics of Deep Learning”, Annual Review of CMP 2020