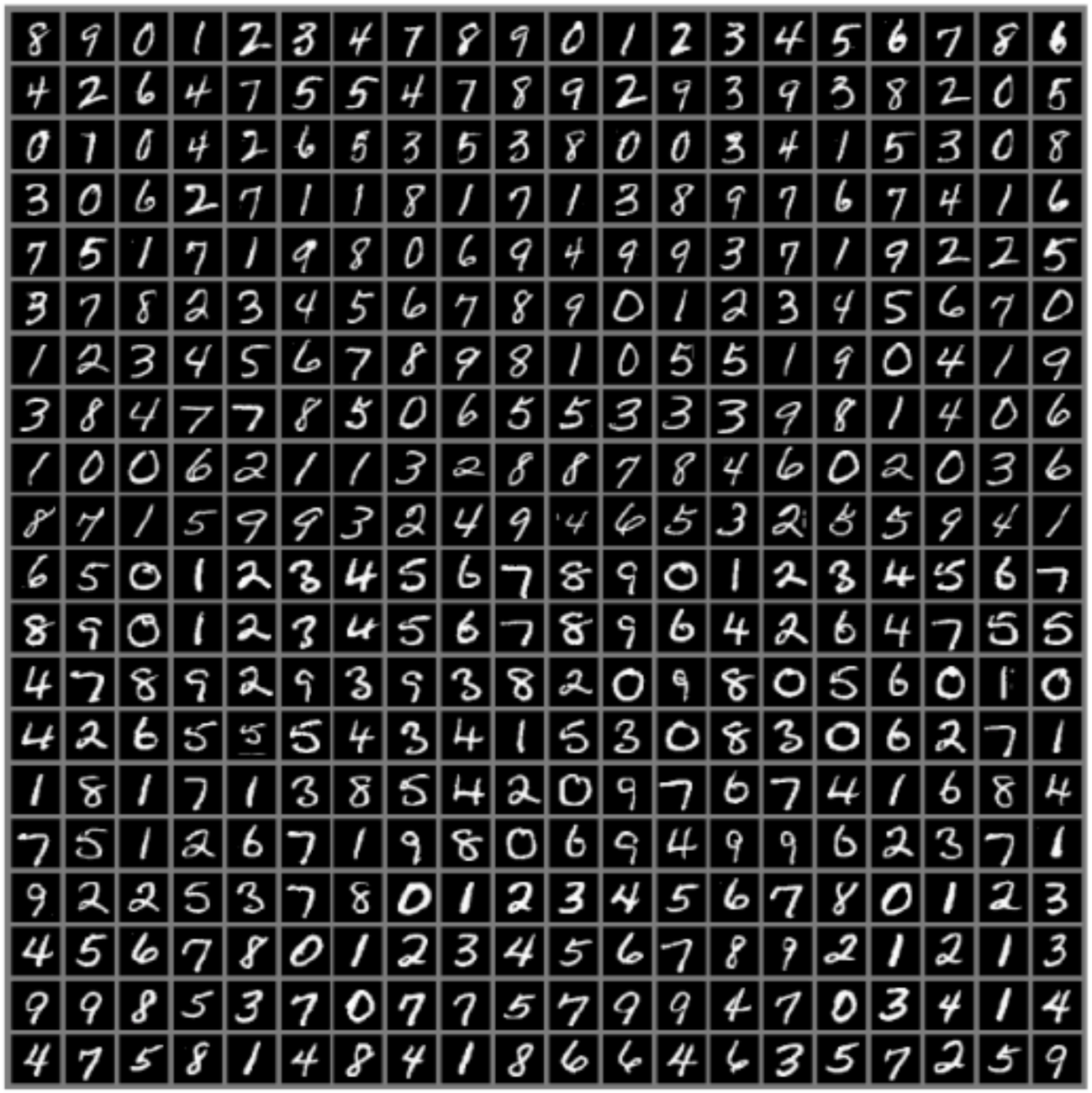
Deep Learning

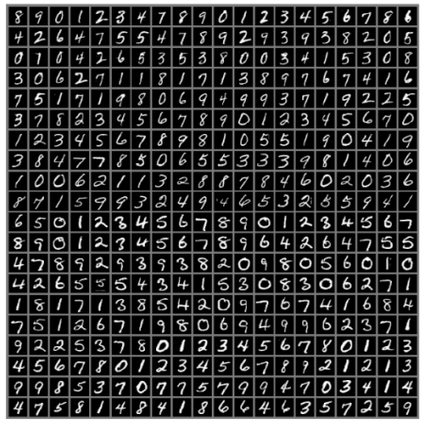
François Rousseau

Lecture #4

Lab sessions (3 & 4): MNIST



Implementation vs theory



1. DATA

```
from keras.models import Sequential
from keras.layers import Dense

# y = softmax (Wx+b)
model = Sequential()
model.add(Dense(num_classes, activation='softmax', input_shape=(784,)))

model.summary()
```

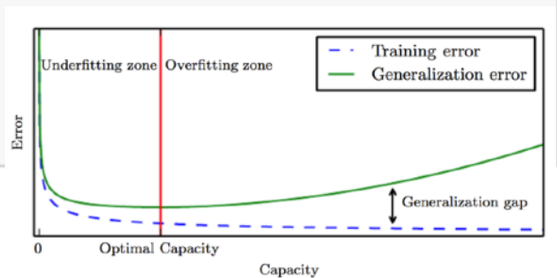
3. NETWORK

```
from keras.optimizers import RMSprop

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

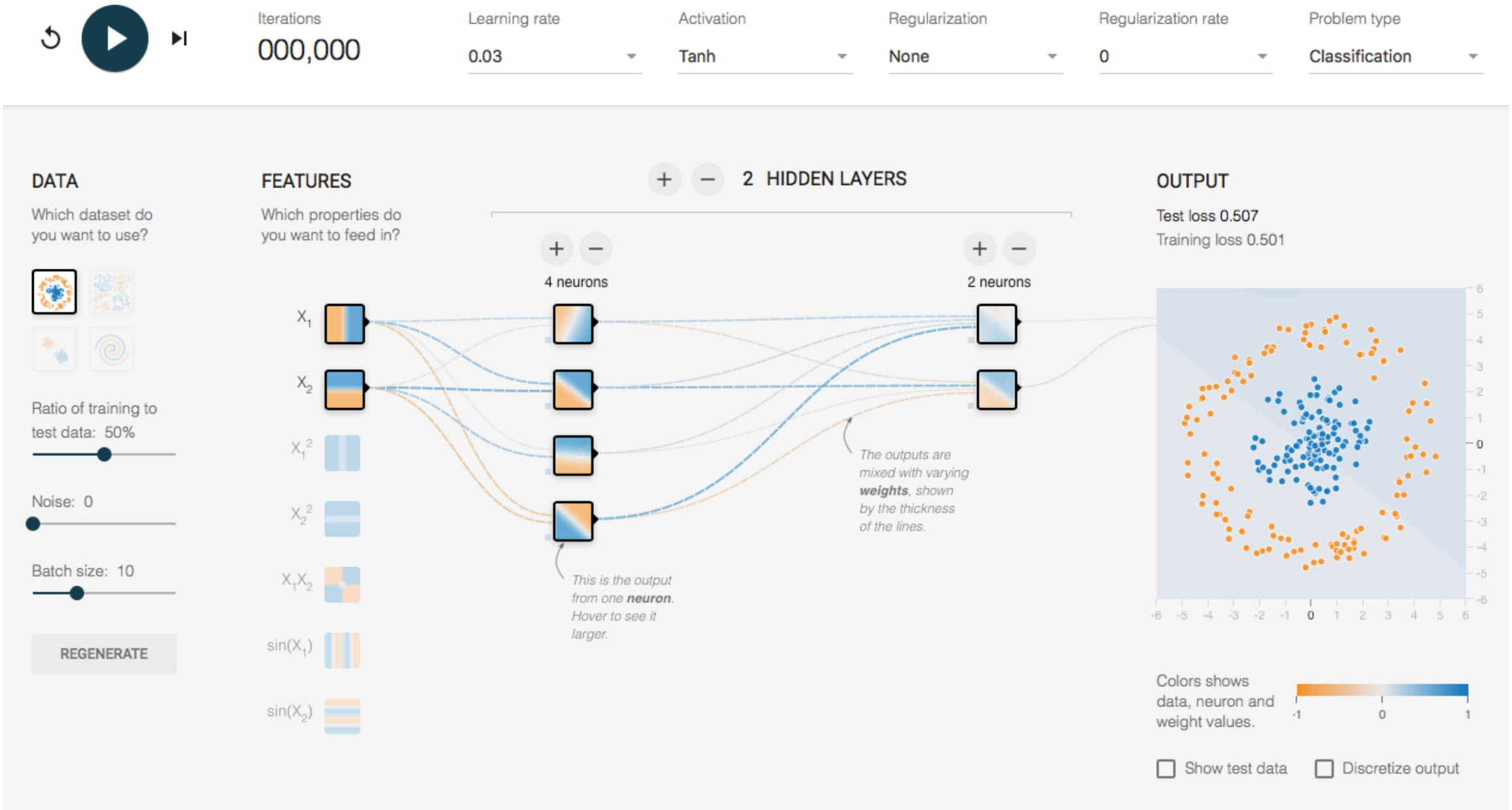
2. LOSS FUNCTION

```
batch_size = 128
epochs = 20

history = model.fit(x_train, z_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, z_test))
```

4. LEARNING

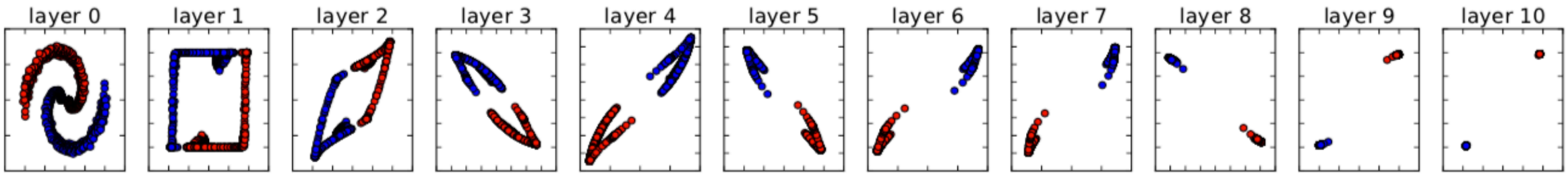


Practicing Deep Learning

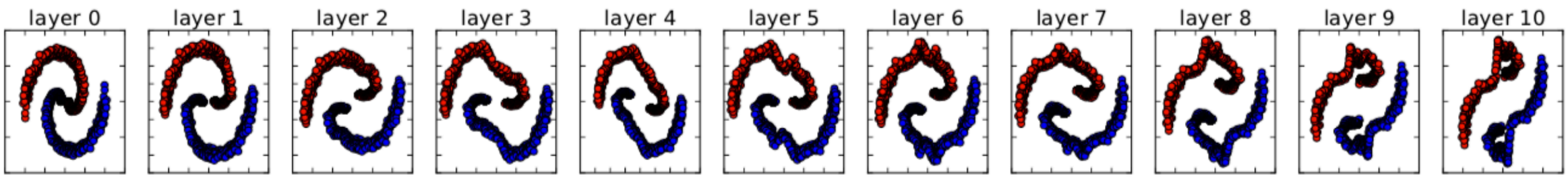


Tensorflow playground

Deep Learning & Representation Learning



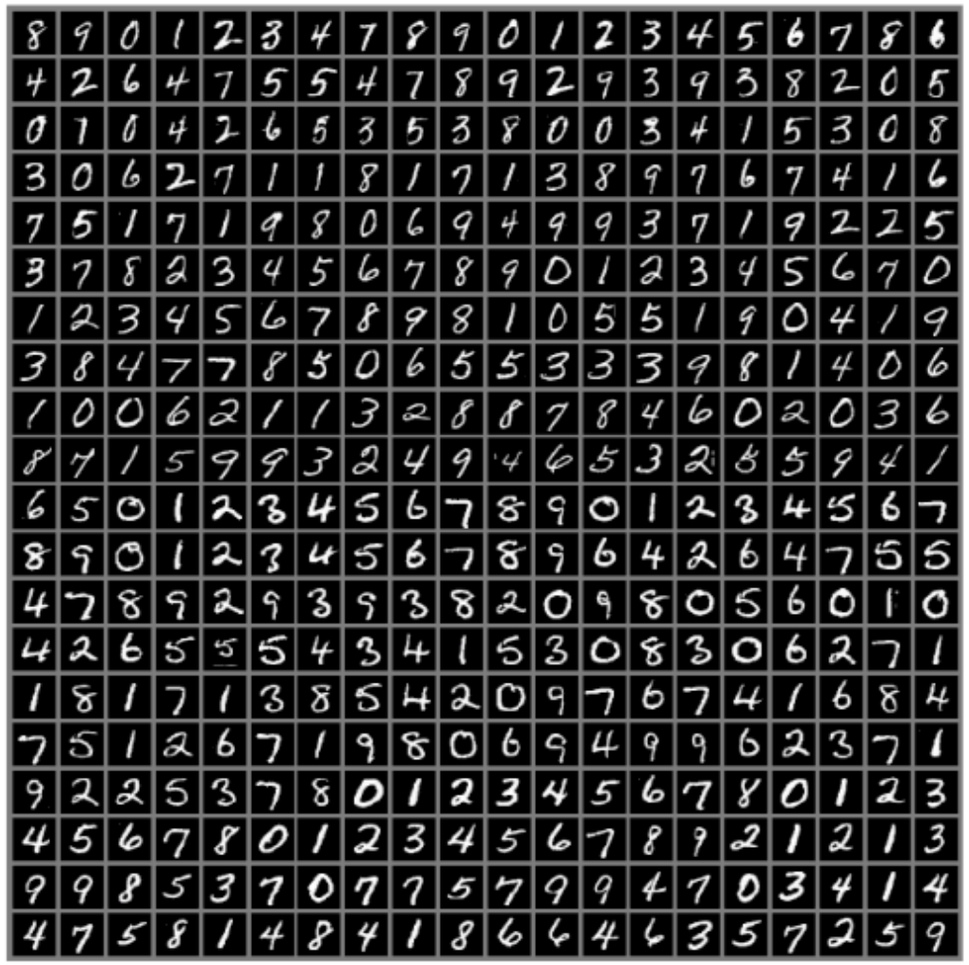
(a) A $\mathcal{C}^0$ network with sharply changing layer-wise particle trajectories.

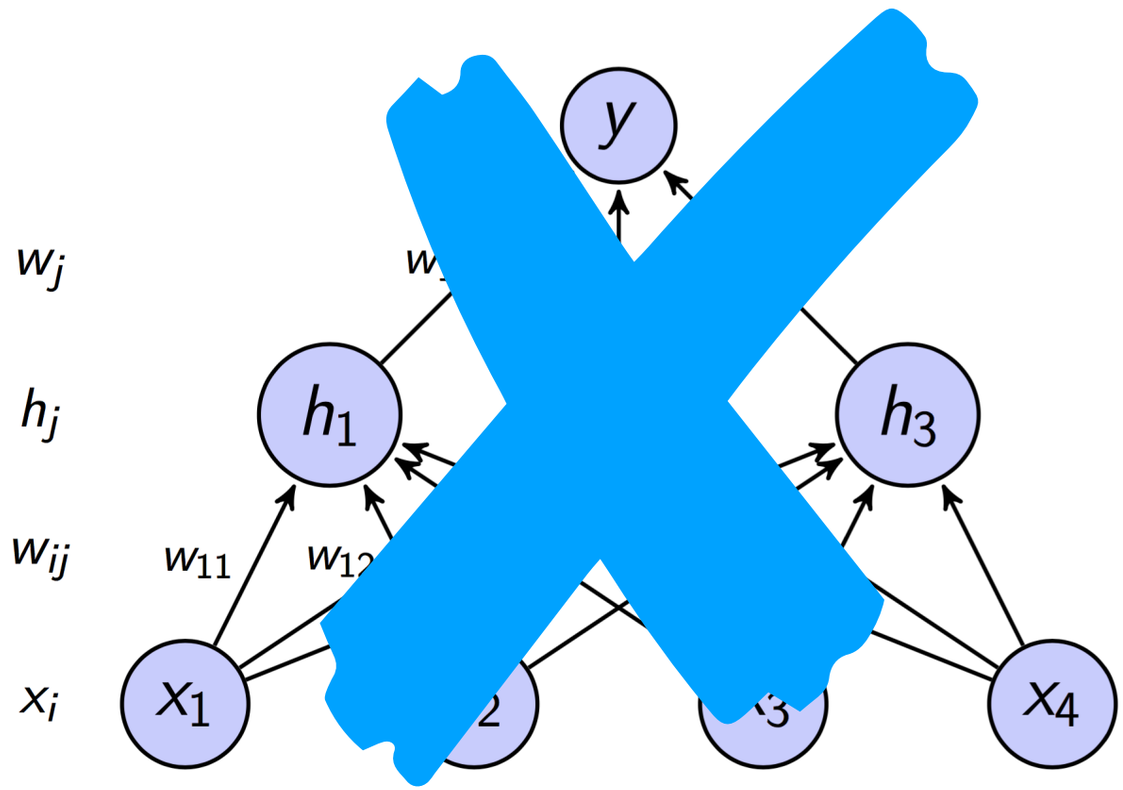


(b) A $\mathcal{C}^1$ network with smooth layer-wise particle trajectories.

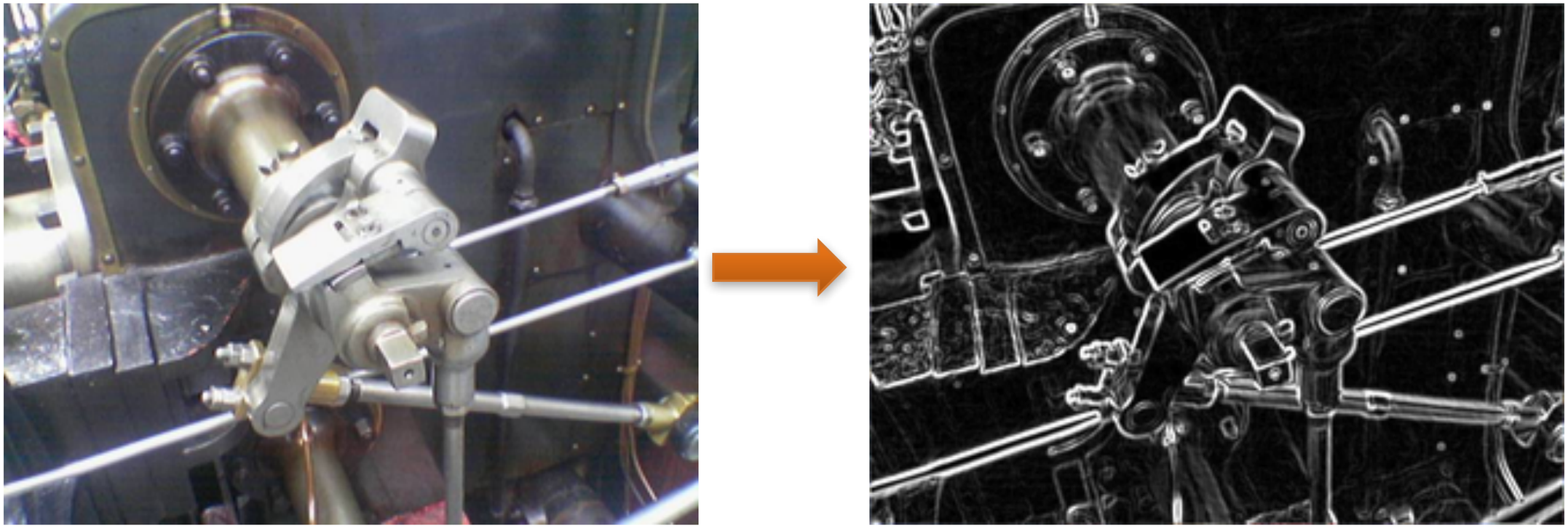
[Hauser et al. 2017]

Dealing with images





Convolutional Layers

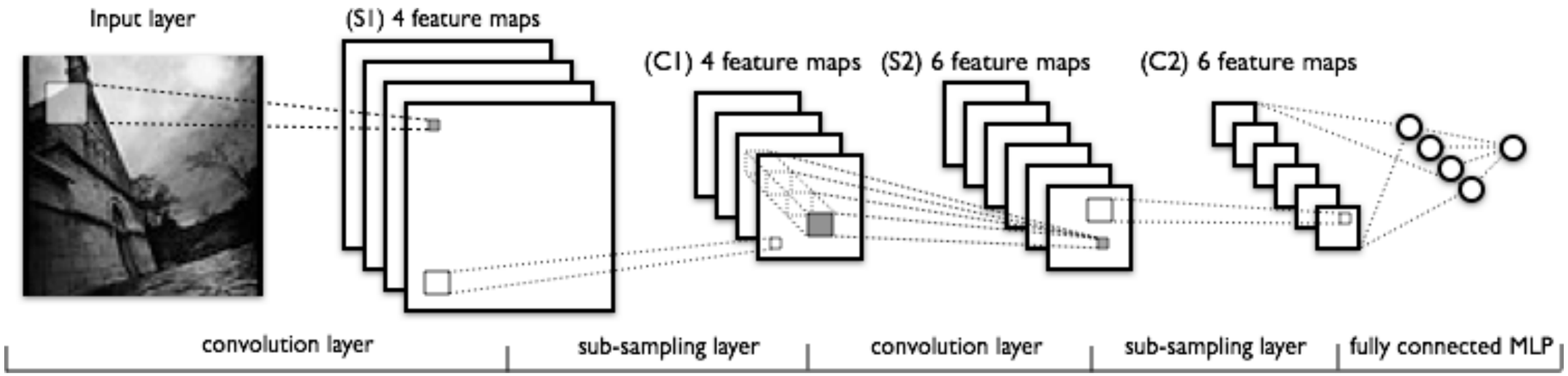


$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

Example from Wikipedia

Convolutional Neural Networks



Convolution:

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(x-t) \cdot g(t)\,\mathrm{d}t = \int_{-\infty}^{+\infty} f(t) \cdot g(x-t)\,\mathrm{d}t$$

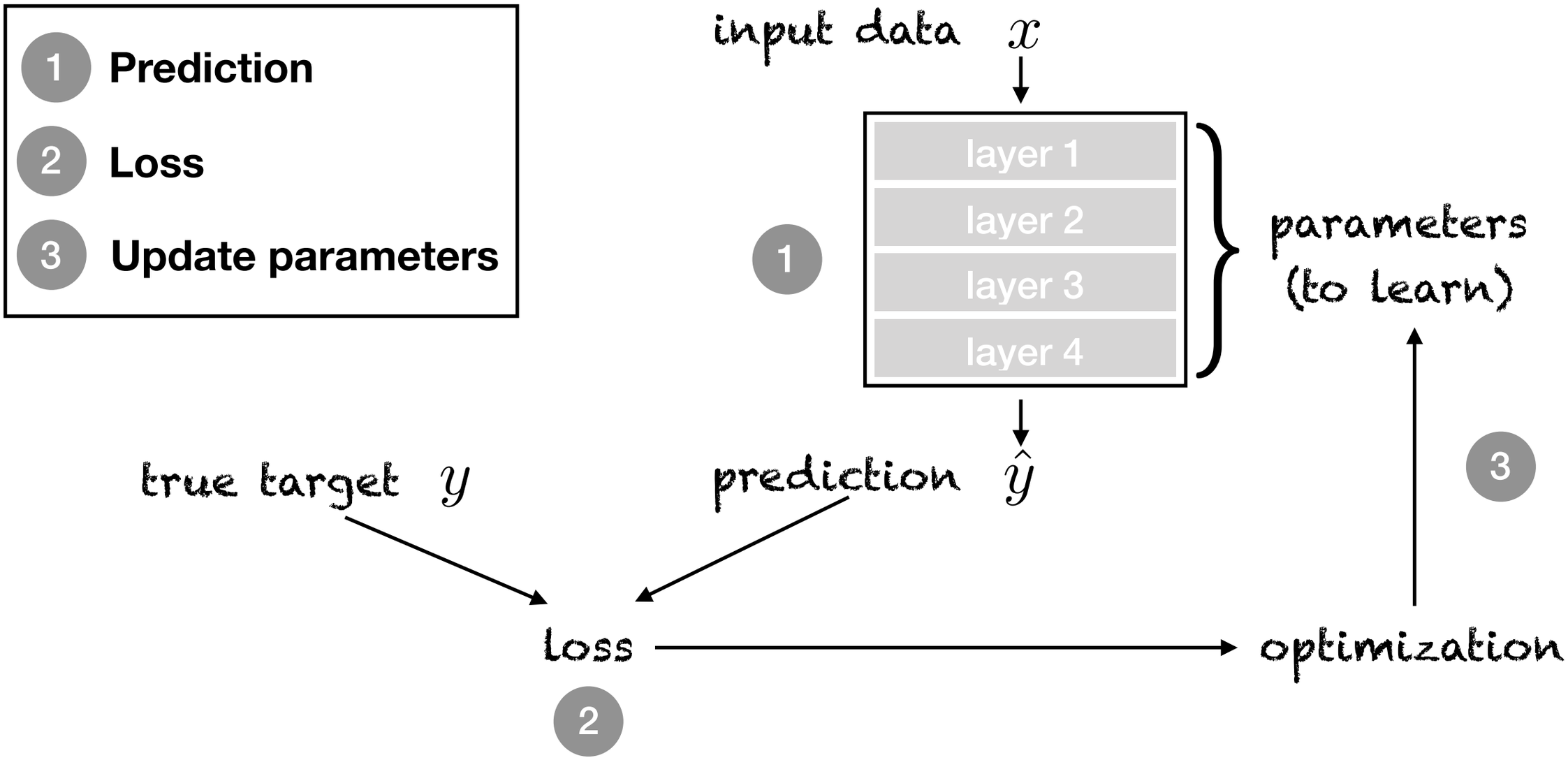
Optimization

[Chapter 8, Goodfellow et al.]

Optimization in deep learning:

finding the parameters of a neural network that significantly reduce a cost function.

1. **Prediction**
2. **Loss**
3. **Update parameters**



Gradient-based approach

- Gradient:

$$\frac{\partial J(\theta)}{\partial \theta} = \lim_{\delta\theta \to 0} \frac{J(\theta + \delta\theta) - J(\theta)}{\delta\theta}$$

- Gradient descent:

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

$$\theta^{k+1} = \theta^k - \epsilon_k \frac{\partial J(\theta^k)}{\partial \theta^k}$$

Learning rate

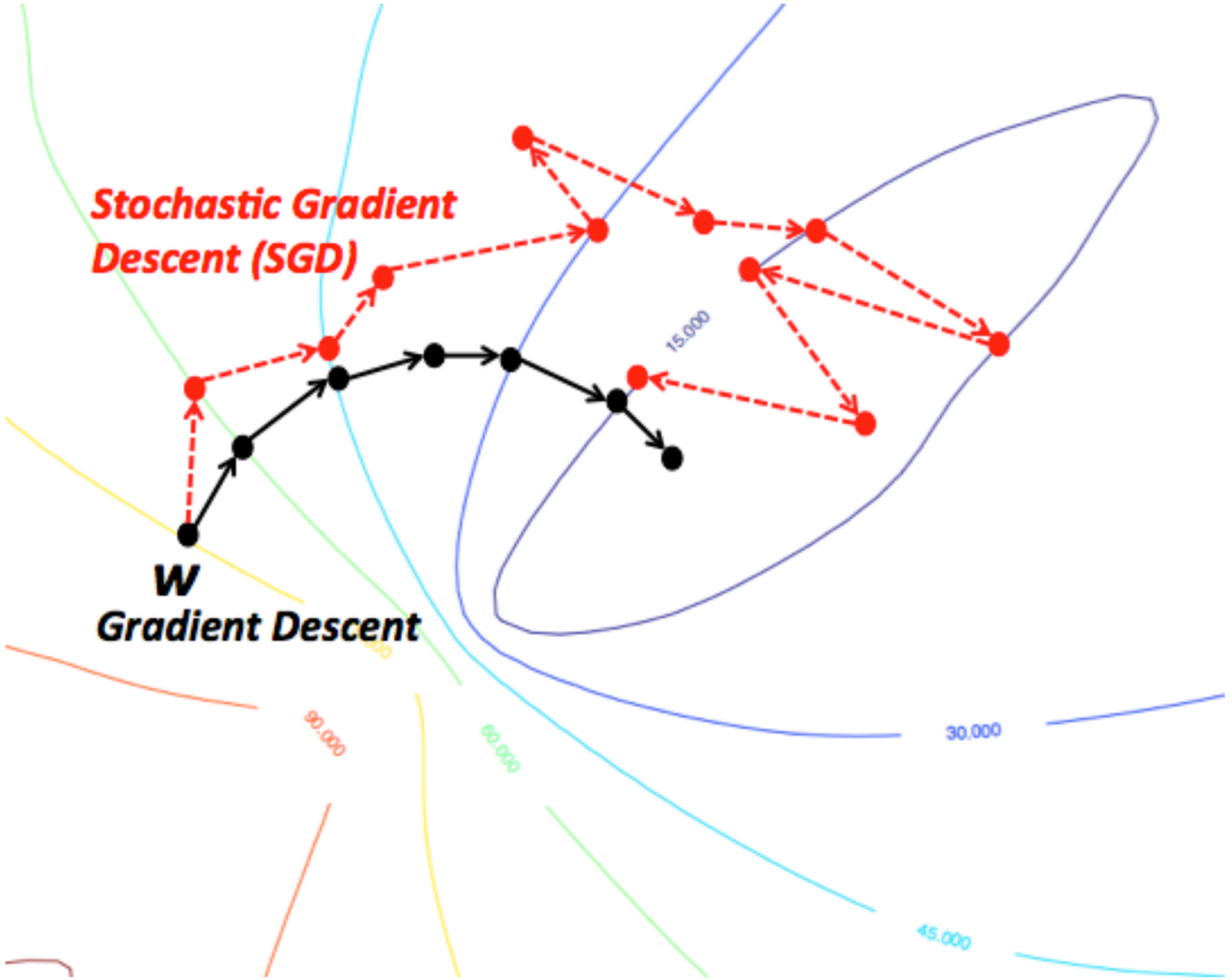
Gradient-based approach

- Stochastic gradient descent (i.i.d examples):

$$\theta^{k+1} = \theta^k - \epsilon_k \frac{\partial J(\theta^k)}{\partial \theta^k}$$

  - direction is a random variable, whose the expectation is the gradient to be estimated.
  - faster than batch gradient descent
- Minibatch SGD:
  - SGD on 10 to 100 examples (mini batch)
  - less noisy estimate of the gradient

Gradient-based approach



K. Duh, 2014

Momentum

- Momentum:
  - use a moving average of the past gradients:

$$\Delta\theta^{k+1} = \alpha\Delta\theta^{k} + (1-\alpha)\frac{\partial J(\theta^{k})}{\partial\theta^{k}}$$

- Choice of the learning rate schedule:
  - small rate: slow convergence, high rate: J can increase
  - Algorithms for adaptive learning rates: Adagrad, RMSProp, Adam, …

Back-propagation

- Back-propagation is an efficient way to recursively compute the gradient:

$$\frac{\partial L}{\partial u} = \sum_i \frac{\partial L}{\partial v_i} \frac{\partial v_i}{\partial u}$$

parent of node u

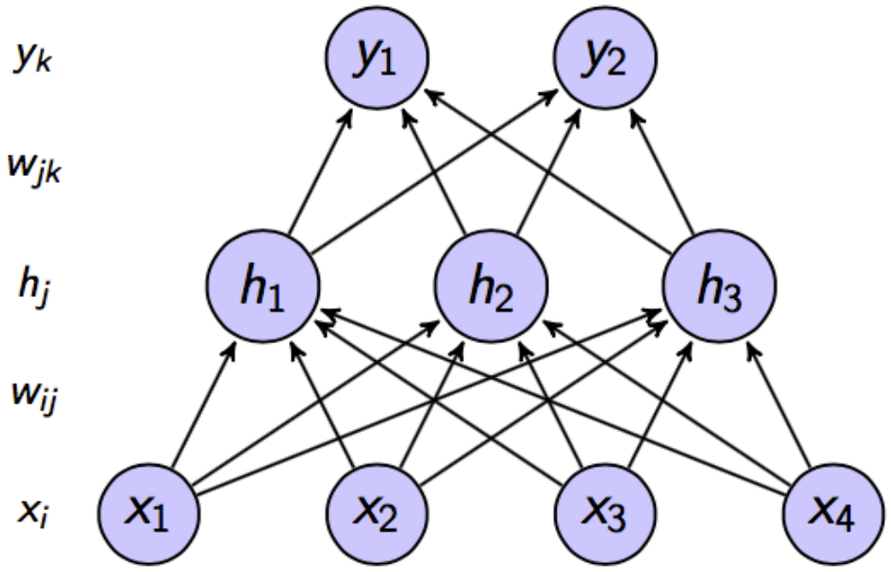
Back-propagation

$$Loss = \frac{1}{2}\sum_k (f(x_k) - y_k)^2$$

$$f(x_k) = \sigma(in_k)$$

$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial\left(\sum_j w_{jk} h_j\right)}{\partial w_{jk}} = \delta_k h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j}\frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial\left(\sum_i w_{ij} x_i\right)}{\partial w_{ij}} = \delta_j x_i$$



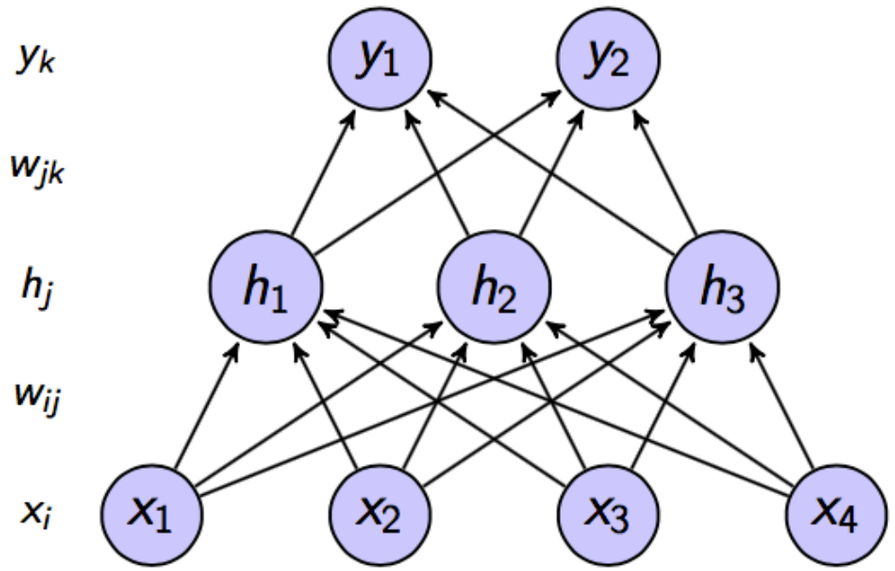
$$\delta_k = \frac{\partial}{\partial in_k}\left(\sum_k \frac{1}{2}[\sigma(in_k) - y_k]^2\right) = [\sigma(in_k) - y_k]\sigma'(in_k)$$

$$\delta_j = \sum_k \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial in_j} = \sum_k \delta_k \cdot \frac{\partial}{\partial in_j}\left(\sum_j w_{jk}\sigma(in_j)\right) = [\sum_k \delta_k w_{jk}]\sigma'(in_j)$$

Back-propagation

$$Loss = \frac{1}{2}\sum_k (f(x_k) - y_k)^2$$

$$f(x_k) = \sigma(in_k)$$



Stochastic Gradient Descent:

$$\theta^{k+1} = \theta^k - \epsilon_k \frac{\partial J(\theta^k)}{\partial \theta^k}$$

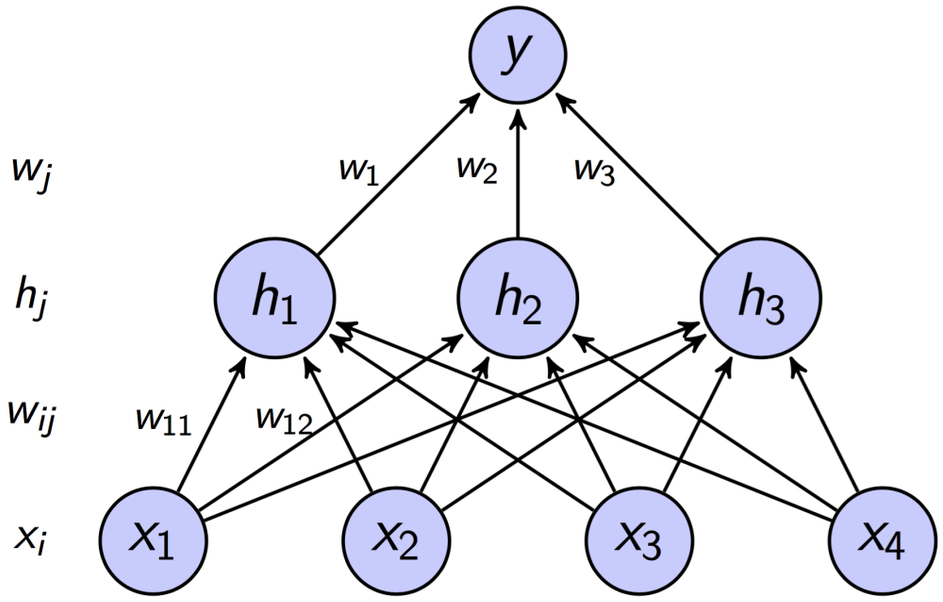
$$\frac{\partial Loss}{\partial w_{jk}} = \delta_k h_j = [\sigma(in_k) - y_k]\sigma'(in_k)h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \delta_j x_i = [\sum_k \delta_k w_{jk}]\sigma'(in_j)x_i$$

Updates involve scaled error from output and input feature

After forward pass, compute δ_k from final layer and then δ_j for previous layer.

In practice: learning




```
from keras import models
from keras import layers
```

```
network = models.Sequential()
network.add(layers.Dense(3, activation='relu', input_shape=(4,)))
network.add(layers.Dense(1, activation='softmax'))
```

```
from keras.optimizers import RMSprop

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(x_test, z_test))
```

Challenges in optimization

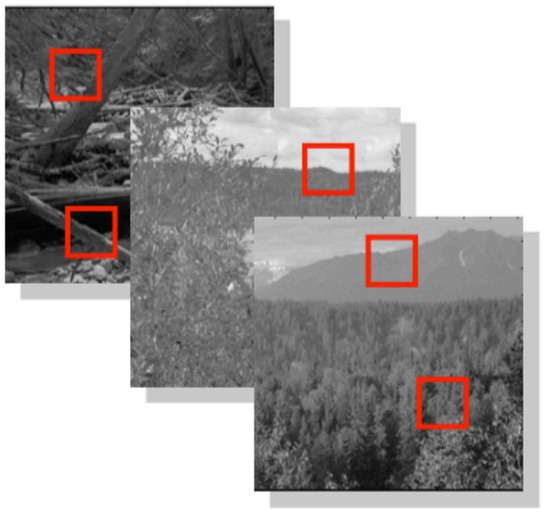
- Ill conditioning
- Local minima
- Vanishing or exploding gradients

Autoencoders

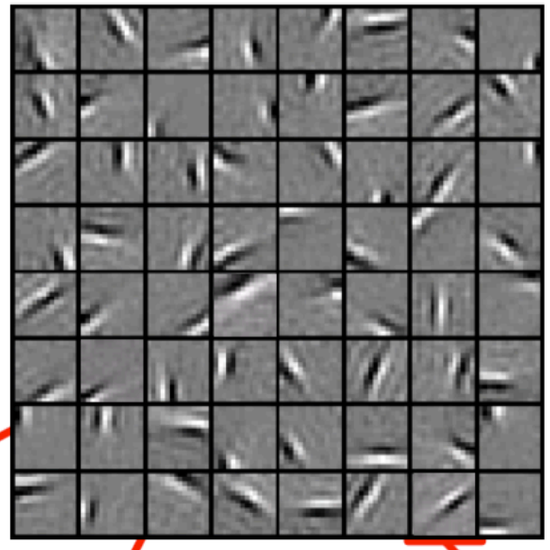
[Chapter 14, Goodfellow et al.]

Sparse Coding

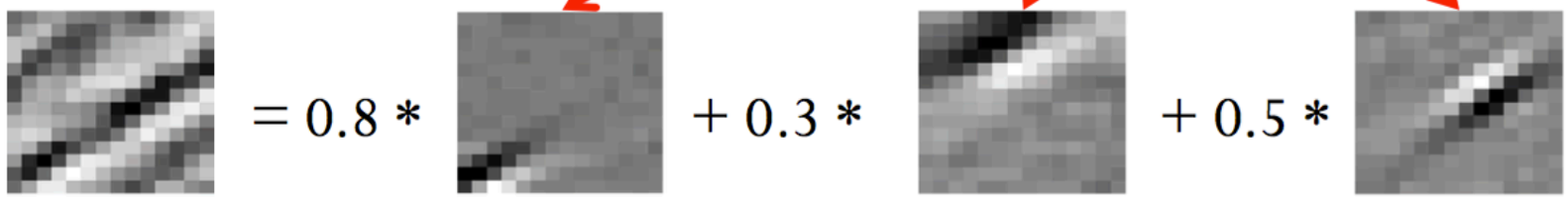
Natural Images



Learned bases: “Edges”



New example



$$x = 0.8 * \phi_{36} + 0.3 * \phi_{42} + 0.5 * \phi_{65}$$

[0, 0, … **0.8**, …, **0.3**, …, **0.5**, …] = coefficients (feature representation)

Slide Credit: Honglak Lee

Sparse Coding

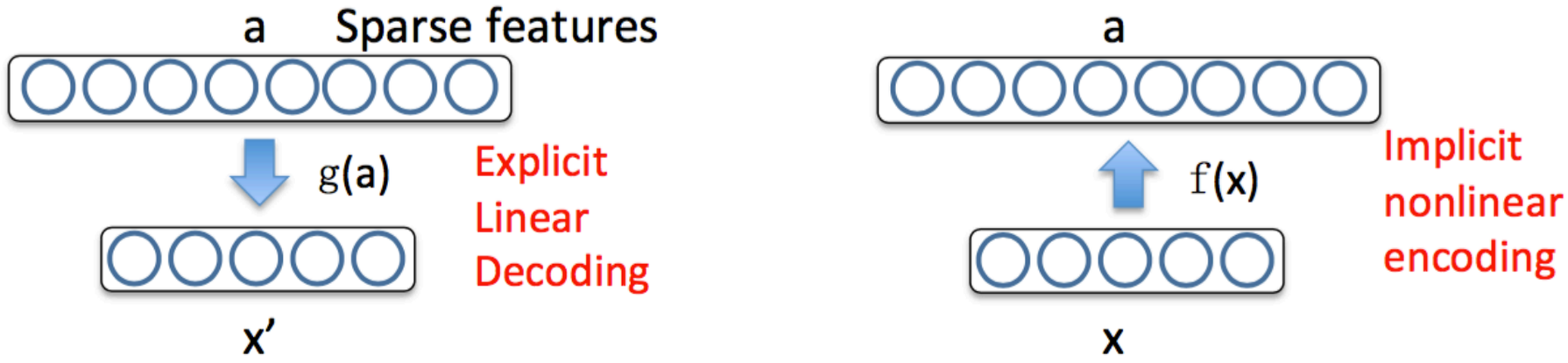
- Sparse coding: Olshausen & Field, 1996
- Developed to explain early visual processing in the brain
- Objective: given a set of input data vectors, learn a dictionary of bases:

$$x_i = \sum_k a_{ik} \Phi_k$$

- Each data vector is represented as a sparse linear combination of bases

Sparse coding

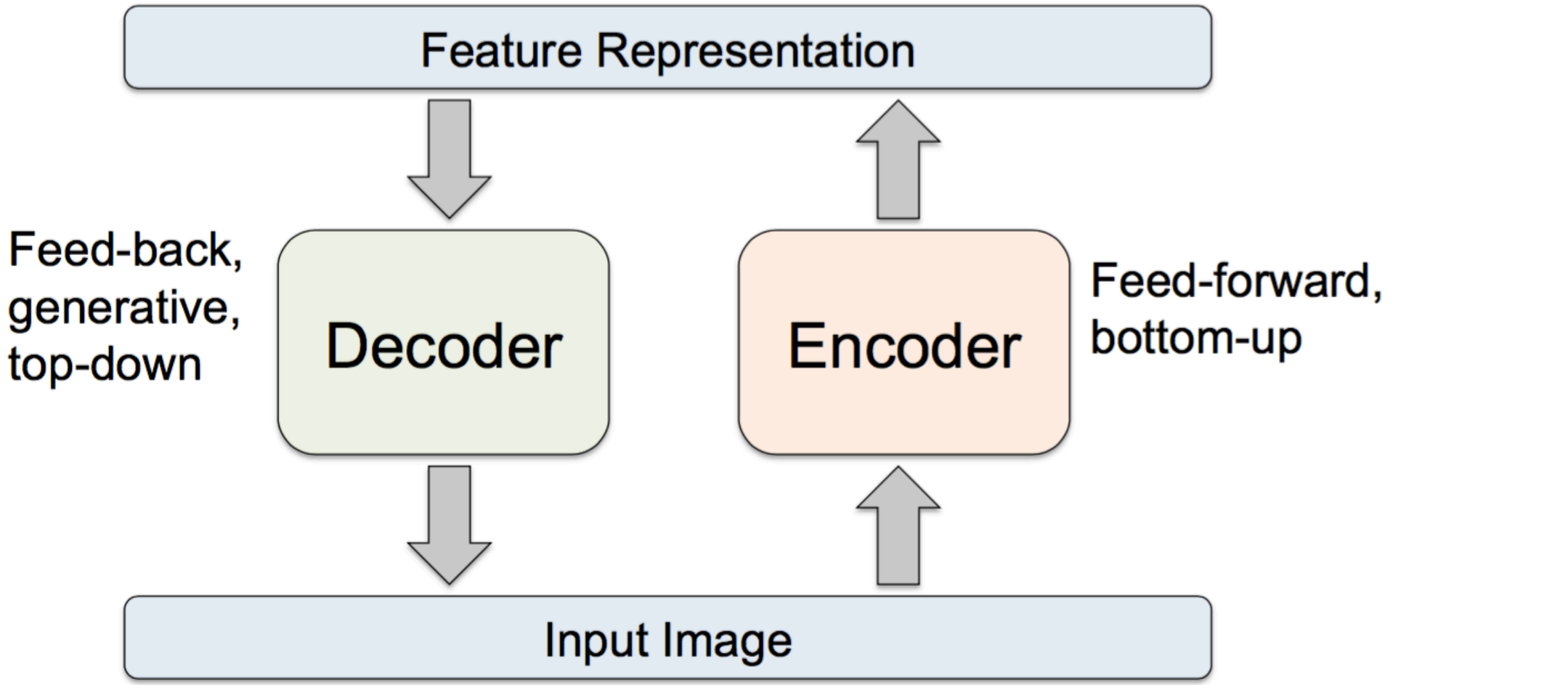
$$\min_{\mathbf{a},\boldsymbol{\phi}} \sum_{n=1}^{N} \left|\left| \mathbf{x}_n - \sum_{k=1}^{K} a_{nk}\boldsymbol{\phi}_k \right|\right|_2^2 + \lambda \sum_{n=1}^{N}\sum_{k=1}^{K} |a_{nk}|$$



- Sparse, over-complete representation **a.**
- Encoding **a** = f(**x**) is implicit and nonlinear function of **x**.
- Reconstruction (or decoding) **x'** = g(**a**) is linear and explicit.

Slide from Salakhutdinov, KDD, 2014

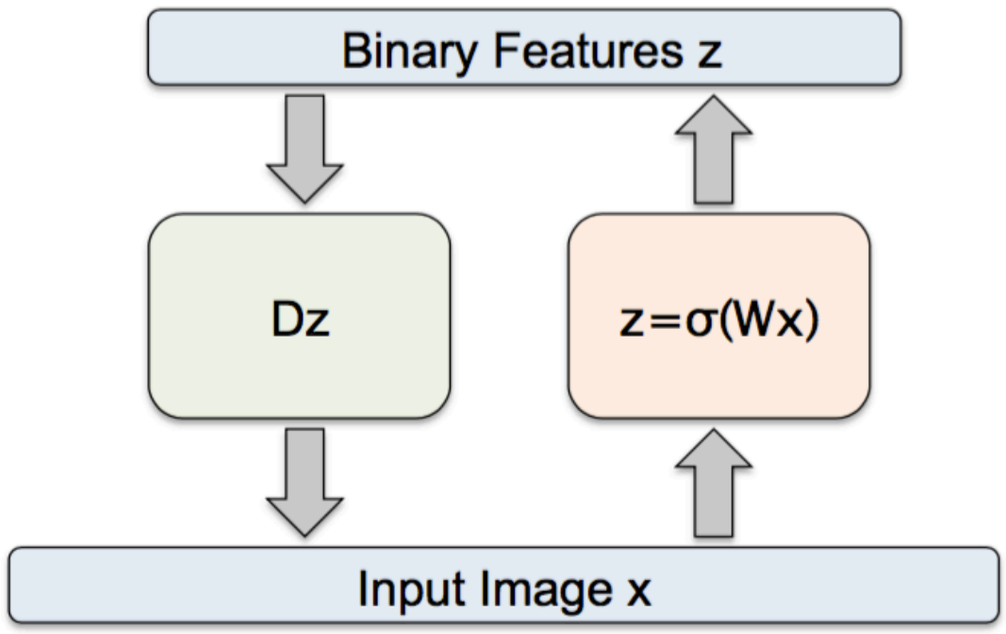
Autoencoders



- Details of what goes insider the encoder and decoder matter!
- Need constraints to avoid learning an identity.

Slide from Salakhutdinov, KDD, 2014

Autoencoders



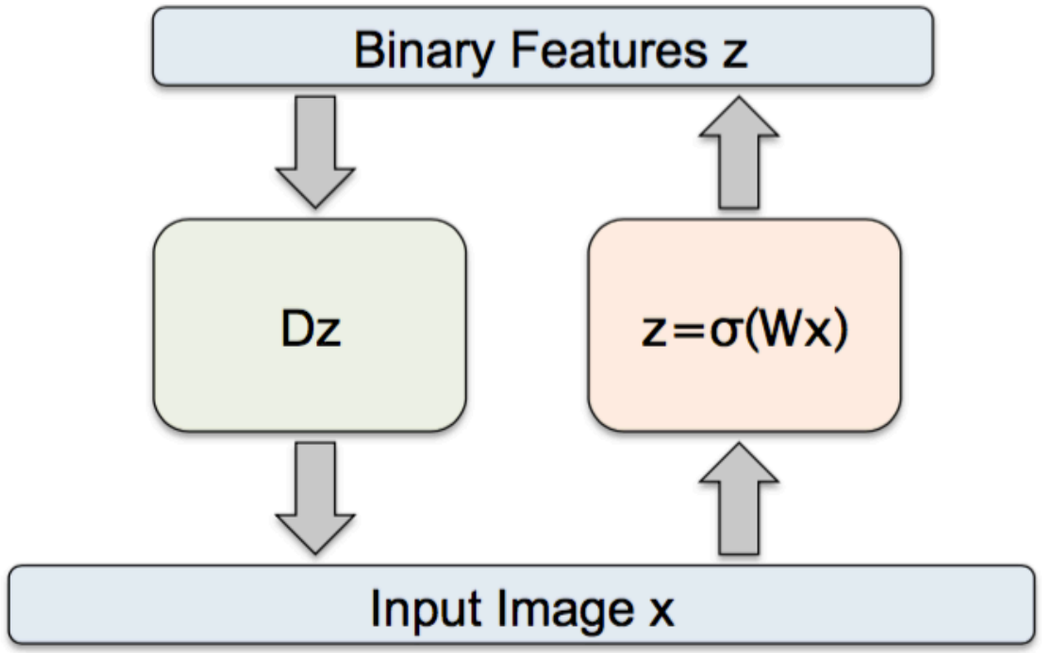
- An autoencoder with D inputs, D outputs, and K hidden units, with K<D.
- Given an input x, its reconstruction is given by:

$$y_j(\mathbf{x}, W, D) = \underbrace{\sum_{k=1}^{K} D_{jk}}_{\text{Decoder}} \underbrace{\sigma\left(\sum_{i=1}^{D} W_{ki} x_i\right)}_{\text{Encoder}}, \quad j = 1, .., D.$$

$$y_j = \sum_{k=1}^{K} D_{jk} z_k \qquad z_k = \sigma\left(\sum_{i=1}^{D} W_{ki} x_i\right)$$

Slide from Salakhutdinov, KDD, 2014

Autoencoders



• An autoencoder with D inputs, D outputs, and K hidden units, with K<D.

• We can determine the network parameters W and D by minimizing the reconstruction error:

$$E(W, D) = \frac{1}{2} \sum_{n=1}^{N} ||y(\mathbf{x}_n, W, D) - \mathbf{x}_n||^2.$$

Slide from Salakhutdinov, KDD, 2014

Autoencoder and dimensionaly reduction

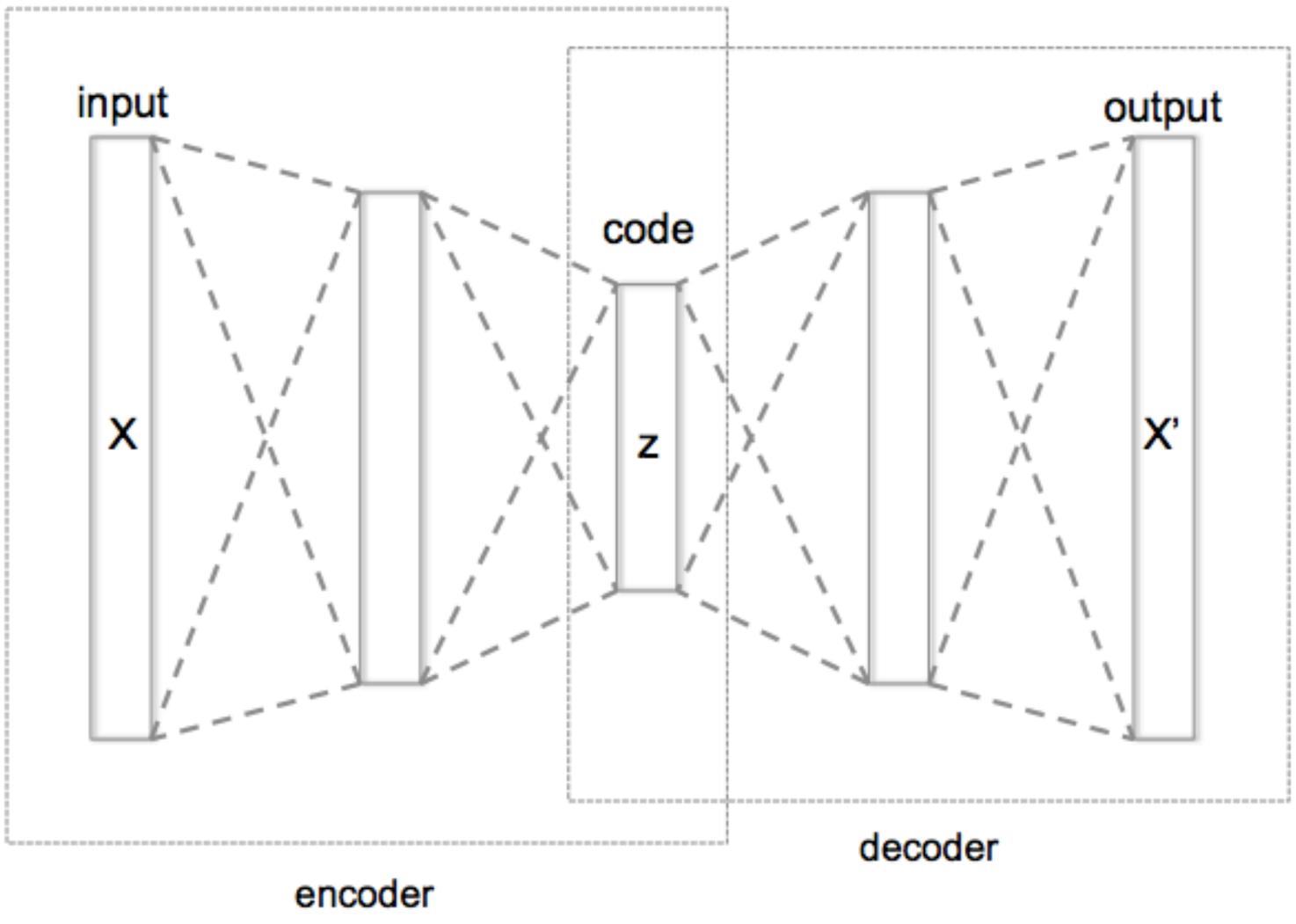
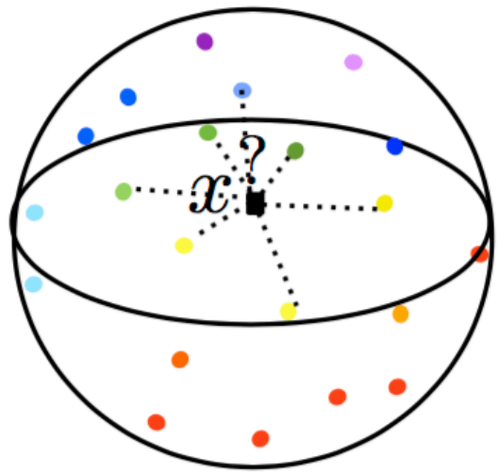


Figure from wikipedia

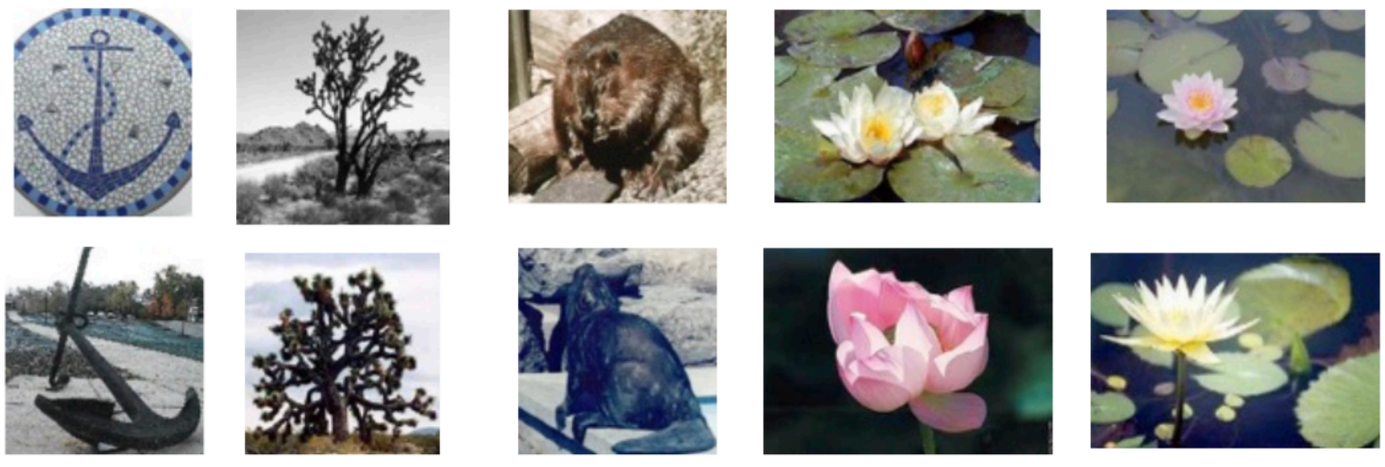
Curse of dimensionality

- $f(x)$ can be approximated from examples $\{x_i\,,\, f(x_i)\}_i$ by local interpolation if f is regular and there are close examples:



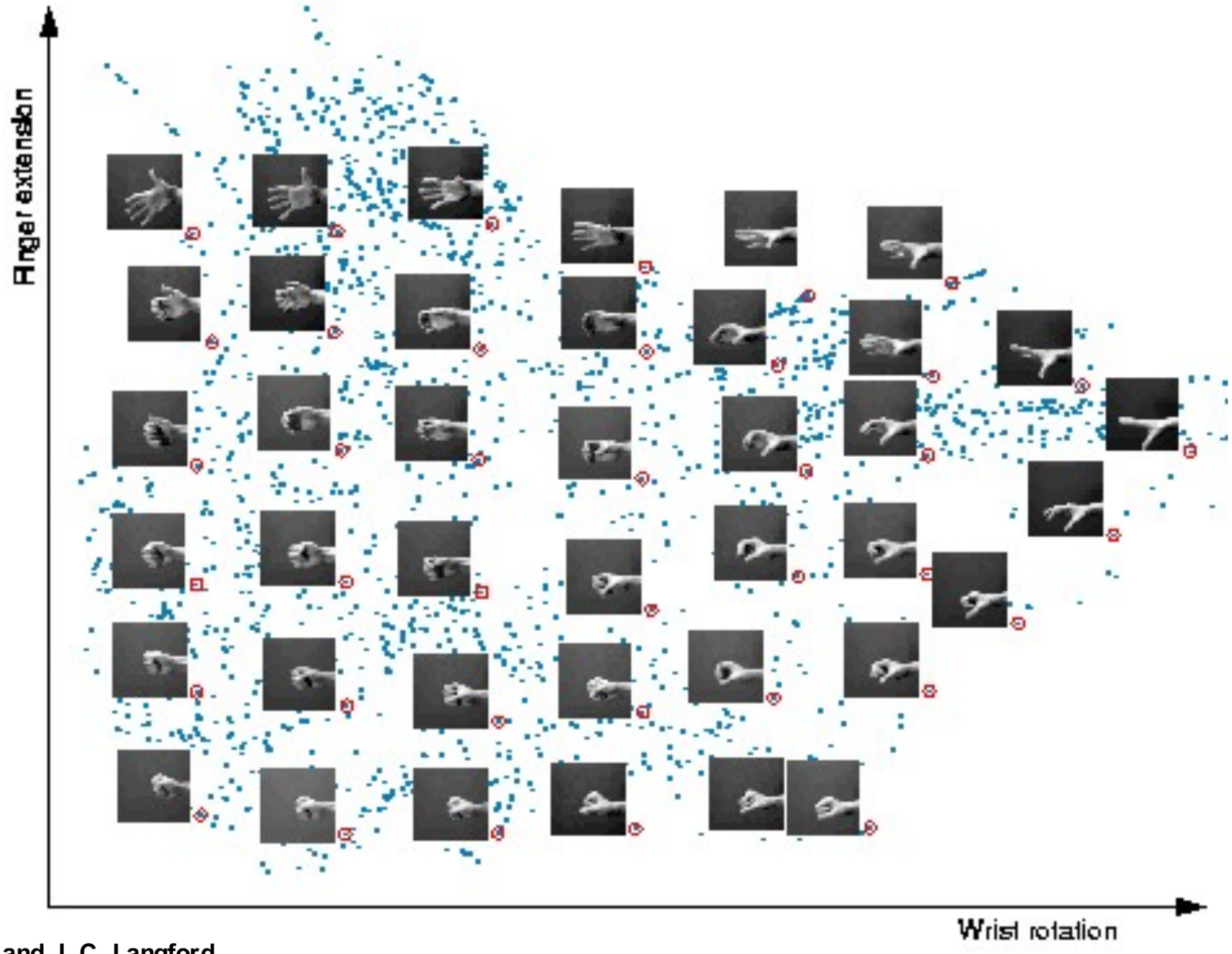
- Need ϵ^{-d} points to cover $[0,1]^d$ at a Euclidean distance ϵ

Problem: $\|x - x_i\|$ is always large



slide from S. Mallat

Dimensionality Reduction



J. B. Tenenbaum, V. de Silva and J. C. Langford
Science 290 (5500): 2319-2323, 22 December 2000

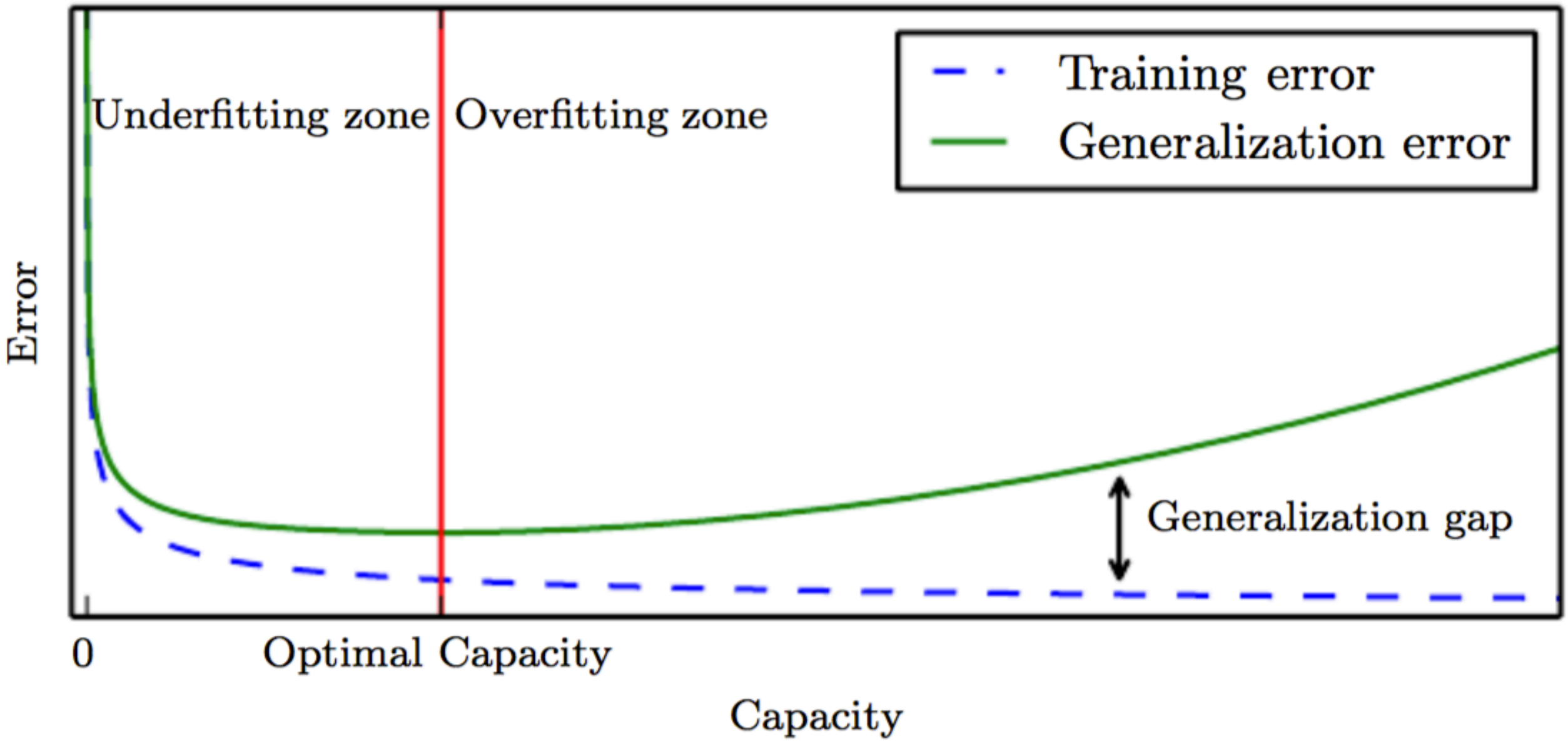
Regularization

[Chapter 7, Goodfellow et al.]

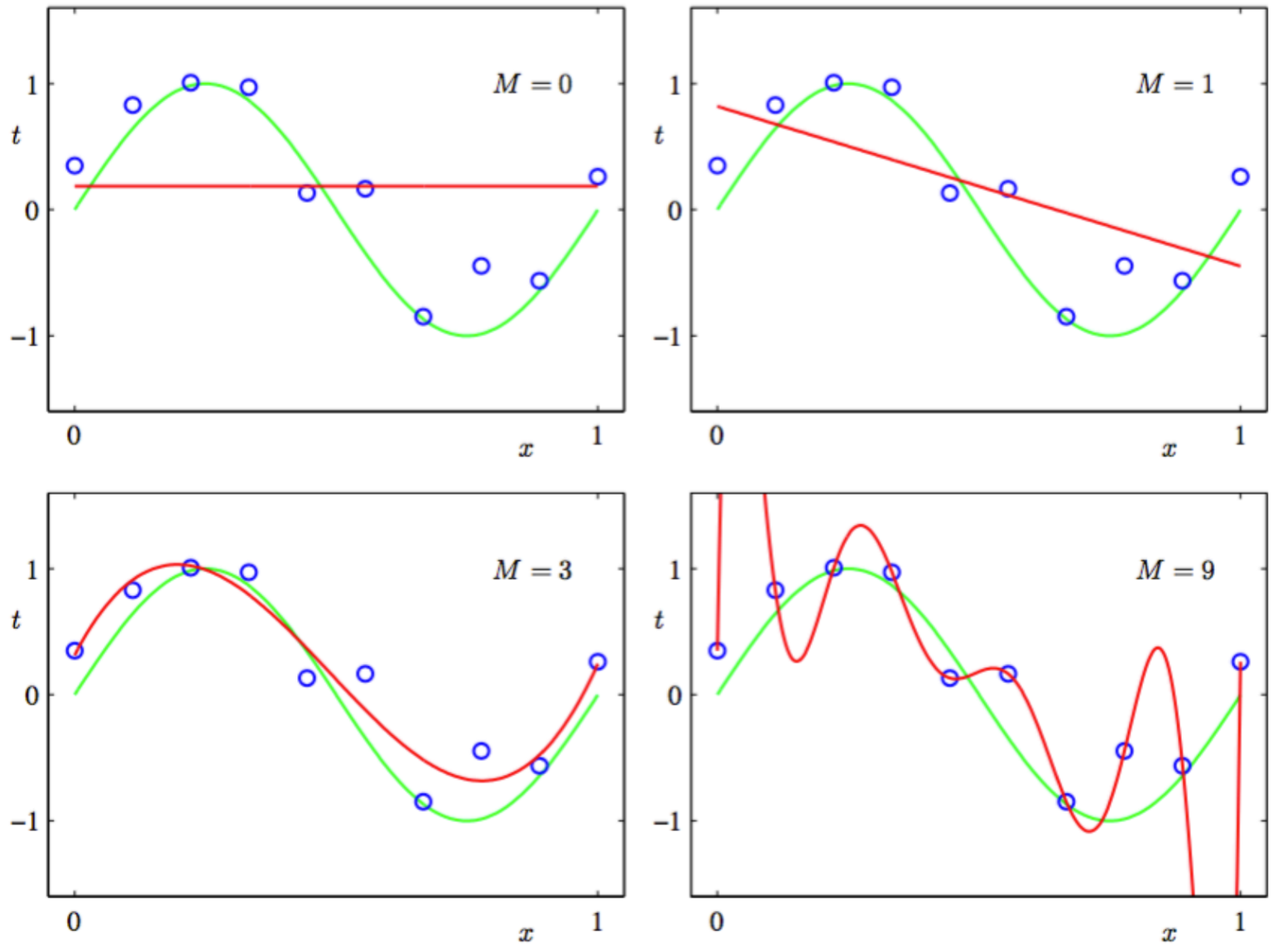
Regularization in deep learning:

reducing the generalization error but not the training error

Experiments in practice



Overfitting



Bishop, 2006

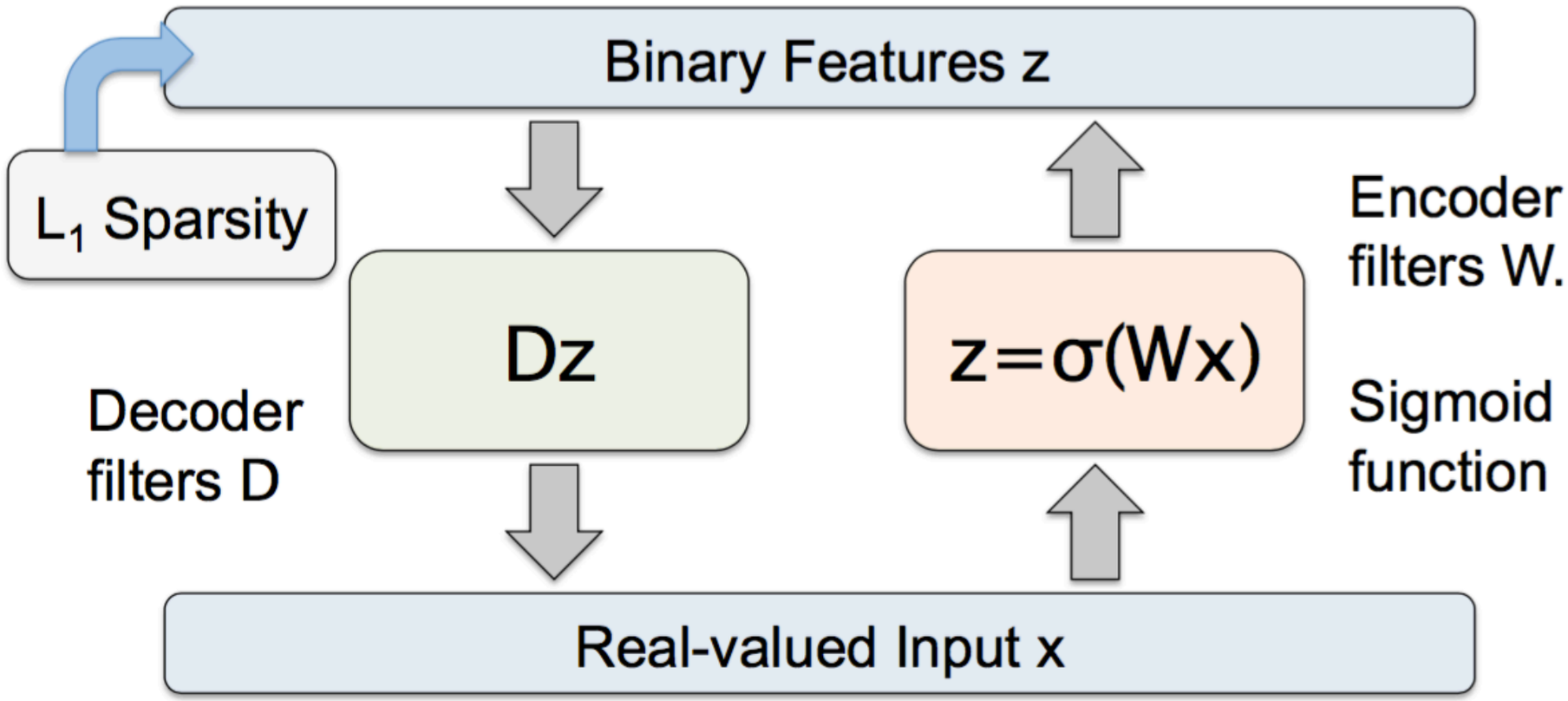
Parameter norm penalization

- Regularized objective function:

$$\tilde{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

- L² norm: $\Omega(\theta) = \frac{1}{2}||w||_2^2$
- L¹ norm: $\Omega(\theta) = ||w||_1 = \sum_i |w_i|$

Autoencoder Regularization



At training time

$$\min_{D,W,\mathbf{z}} ||D\mathbf{z} - \mathbf{x}||_2^2 + \lambda|\mathbf{z}|_1 + ||\sigma(W\mathbf{x}) - \mathbf{z}||_2^2$$

Decoder Encoder

Kavukcuoglu et al., '09

Slide from Salakhutdinov, KDD, 2014

Data augmentation

- Purpose: improving model generalization error by training on more data

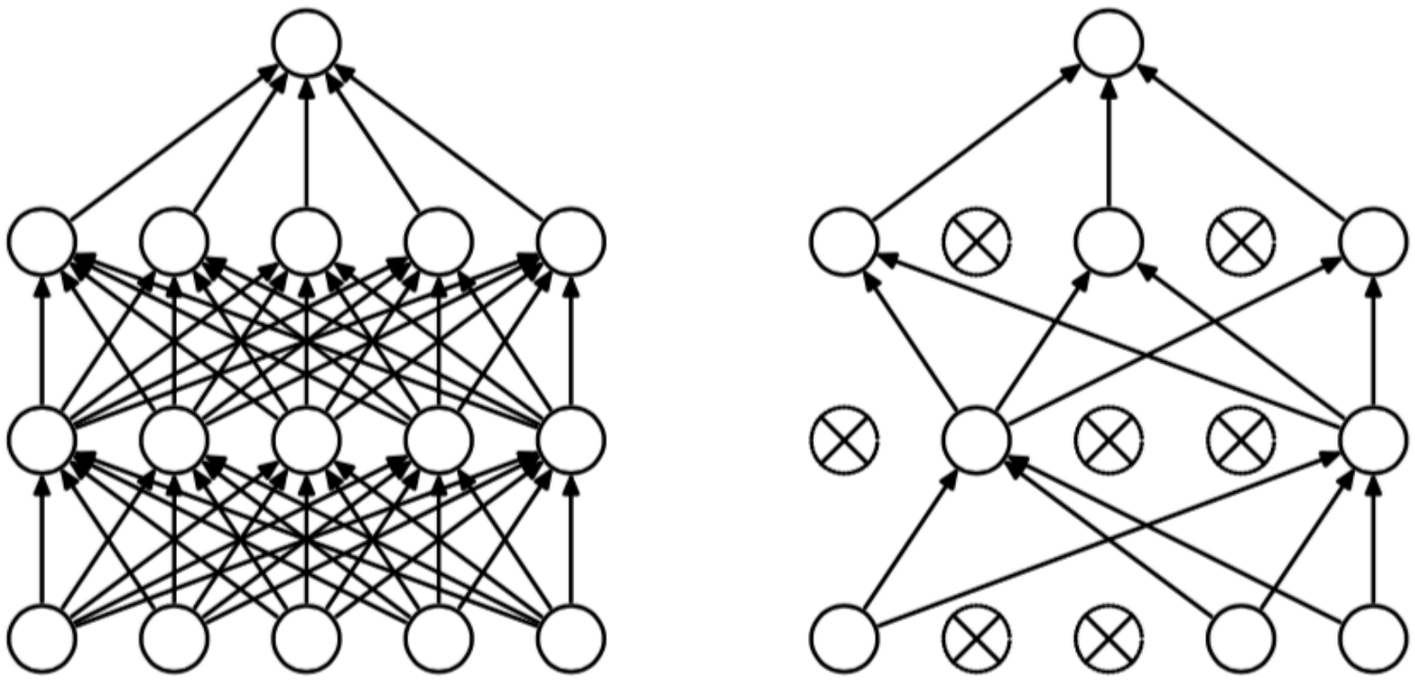
- Very efficient for object recognition

- How to:

  - apply (geometric) transformations on input data (such as translation, rotation, scaling for images).

  - noise injection

Dropout



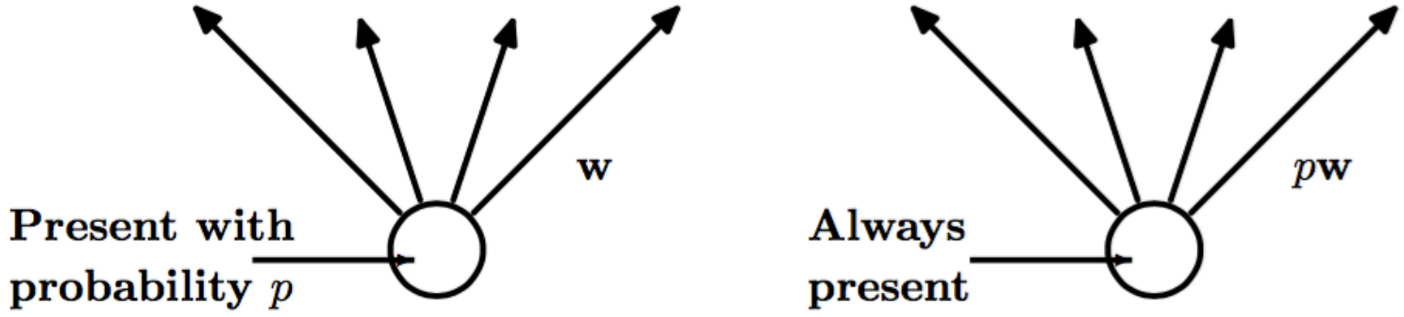
(a) Standard Neural Net

(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Srivastava et al., 2014

Dropout



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Srivastava et al., 2014

Dropout can be understood as model averaging

Dropout is a strong regularizer

Things to know

- Convolutional layers
- Sparse coding
- Autoencoders
- Representation Learning
- Curse of dimensionality
- Generative vs Discriminative
- Backpropagation
- Stochastic Gradient Descent
- Learning rate
- Regularization
- Dropout
- Data augmentation

Schedule

- 4 lectures
- 4 lab sessions
- 2 graded lab sessions
- 4 flipped classrooms
- 1 project
- 1 invited talk

Flipped classroom

- One book chapter + one video to prepare in advance (i.e. reading, taking notes, prepare two or three questions)
- MCQ at the beginning of the session (10mn)
- 20 mn lecture
- 40 mn discussion
- MCQ at the end of the session (10mn)

Hitch Hicking Guide for Deep Learning Projects

- ***Objectives***
    - Network Model Understanding (loss function, network parameters, dedicated architecture…)
    - Code understanding
    - Reproduce experiments
    - Explore new applications
- ***Expected results***
    - Python notebook explaining the network model, the code, the results…
    - Demo @ Centre Vie