MATH 4280

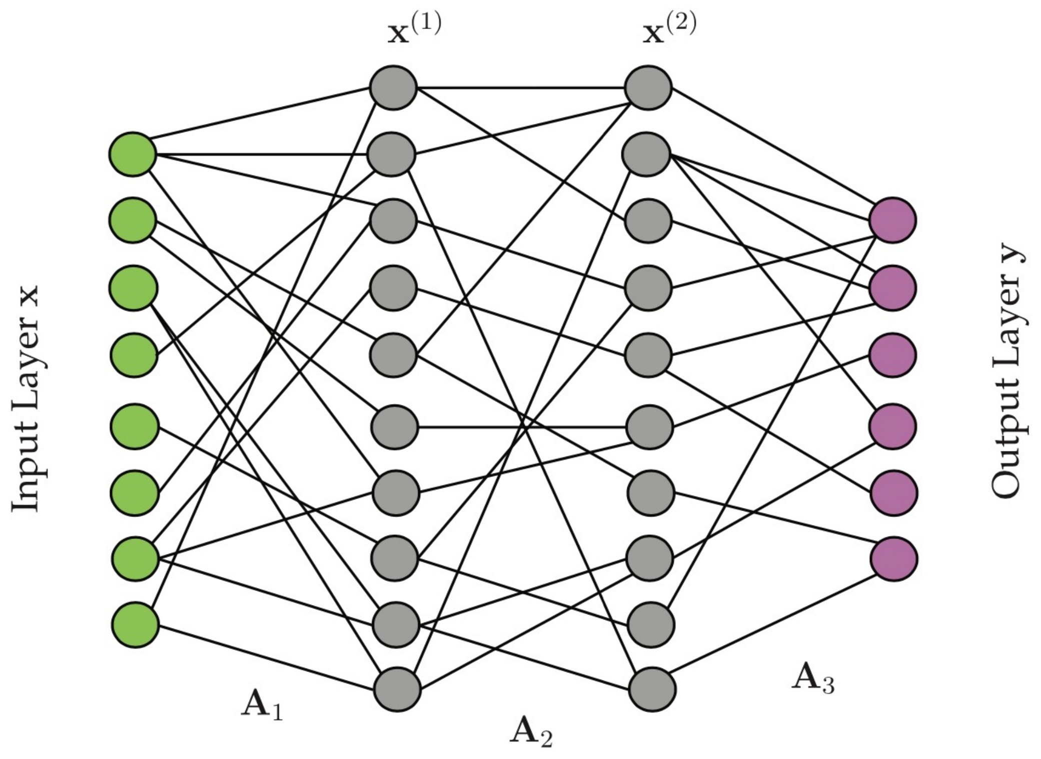
Lecture Notes 6: Neural networks

Introduction

- Neural networks (NN) were inspired by the Nobel prize winning work of Hubel and Wiesel on the primary visual cortex of cats
- Neuronal networks are organized in hierarchical layers of cells for processing visual stimulus
- The recent success of computational NN is based on two factors:
  - The continued growth of computational power
  - The exceptionally large labeled data sets

Basic ideas

- The generic architecture of a multi-layer NN is shown below
- Map the input x_j to the output y_j
- Design questions:
  - How many layers?
  - Dimension of each layer?
  - How to design the output layer?
  - Fully or partially connected layers?
  - The mapping between layers?



- We denote the various layers between input and output as $x^{(k)}$
- k is the layer number
- For linear mapping between layers, the following holds

$$\begin{aligned}\mathbf{x}^{(1)} &= \mathbf{A}_1\mathbf{x}\\ \mathbf{x}^{(2)} &= \mathbf{A}_2\mathbf{x}^{(1)}\\ \mathbf{y} &= \mathbf{A}_3\mathbf{x}^{(2)}\end{aligned}$$

which forms a compositional structure,

$$\mathbf{y} = \mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{x}$$

- In general, we can have a M layer structure

$$\mathbf{y} = \mathbf{A}_M\mathbf{A}_{M-1}\cdots\mathbf{A}_2\mathbf{A}_1\mathbf{x}$$

- This gives limited range of functional responses due to linearity

- Nonlinear mappings are used in general
- The connections between layers are given by

$$\begin{aligned}\mathbf{x}^{(1)} &= f_1(\mathbf{A}_1, \mathbf{x}) \\ \mathbf{x}^{(2)} &= f_2(\mathbf{A}_2, \mathbf{x}^{(1)}) \\ \mathbf{y} &= f_3(\mathbf{A}_3, \mathbf{x}^{(2)})\end{aligned}$$

- More generally, for a M layers network

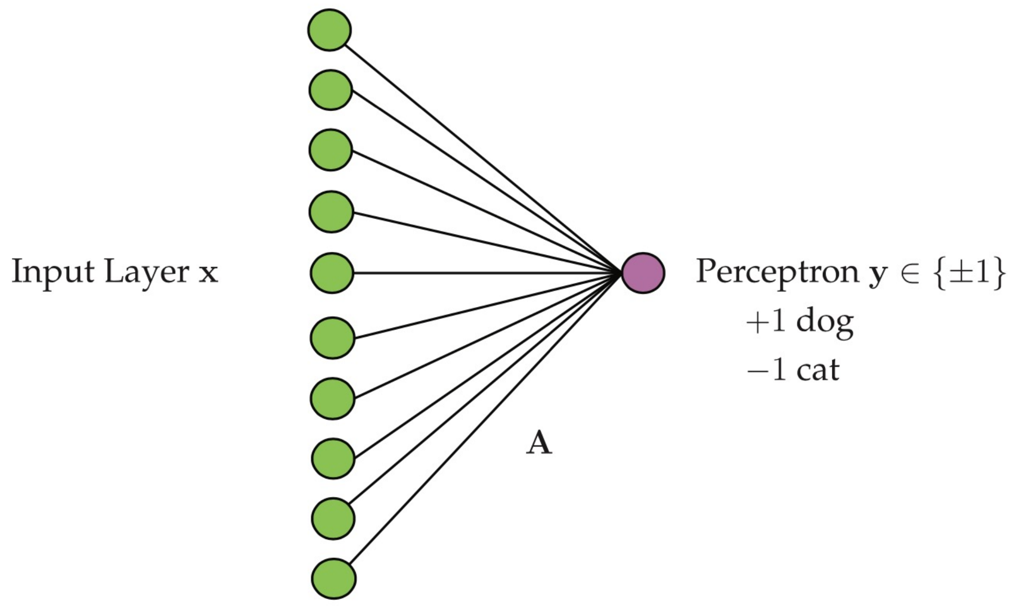
$$\mathbf{y} = f_M(\mathbf{A}_M, \cdots, f_2(\mathbf{A}_2, f_1(\mathbf{A}_1, \mathbf{x}))\cdots)$$

- This gives richer sets of functional responses

A one-layer network

- Consider the dogs and cats example, we construct a one-layer network
- We have a simple output layer

Goal: determine a mapping so that each input data x_j is labelled correctly by y_j



- We consider a linear mapping, this gives a linear system

$$\mathbf{AX} = \mathbf{Y} \;\rightarrow\; [a_1\ a_2\ \cdots\ a_n] \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_p \\ | & | & & | \end{bmatrix} = [+1\ \ +1\ \ \cdots\ -1\ \ -1]$$

- The simplest solution is the pseudo inverse of the data matrix

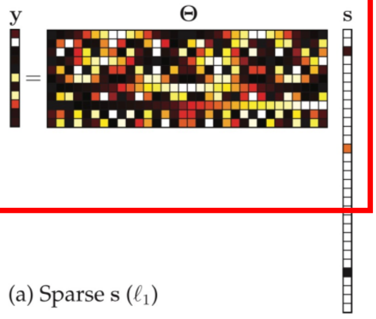
$$\mathbf{A} = \mathbf{YX}^{\dagger}$$

- One can also solve this linear system using other ways, for example, the LASSO

These are the prediction outputs

Chapter 3 Compressed sensing

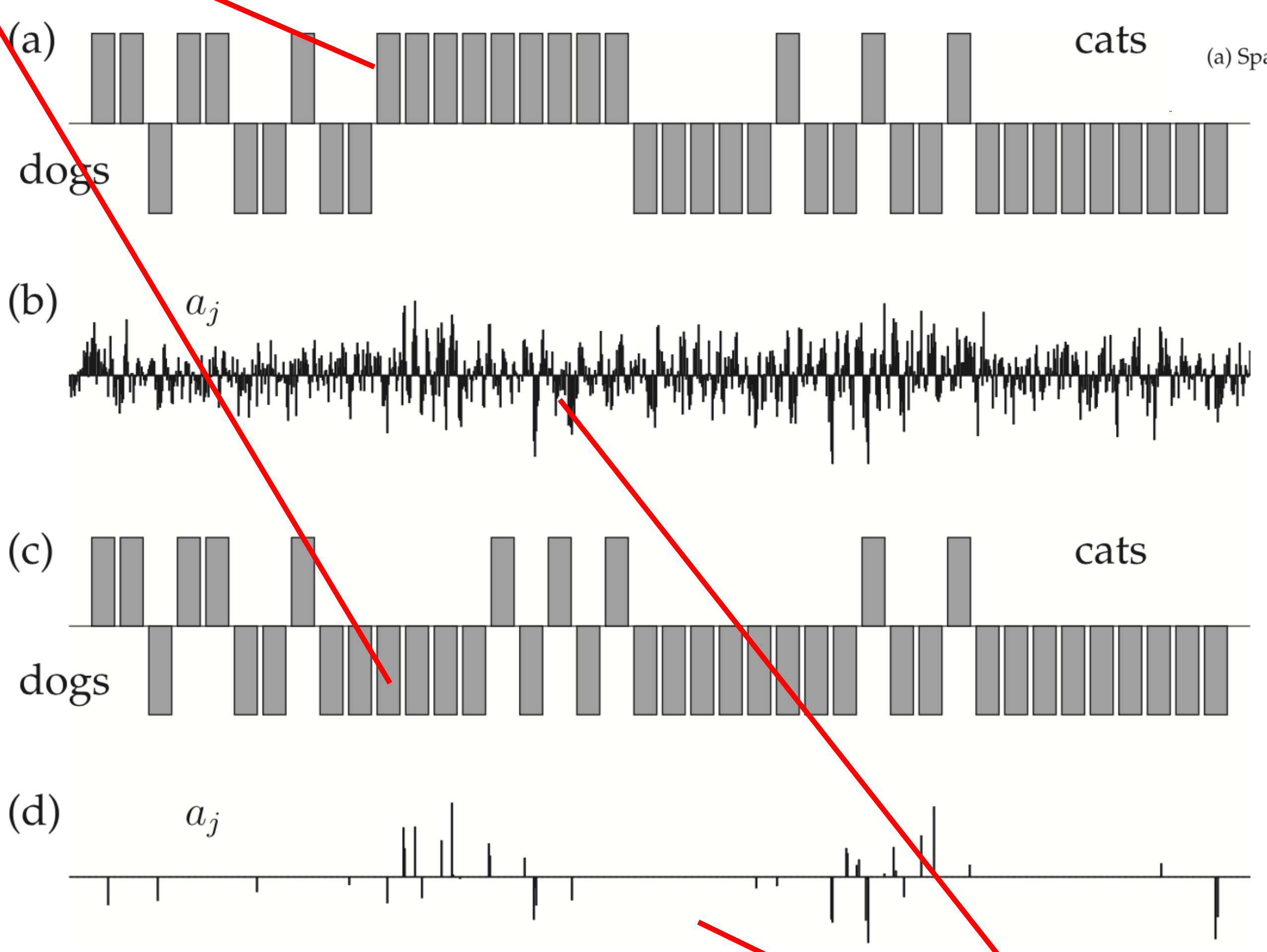
$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_1 \quad \text{subject to} \quad \mathbf{y} = \mathbf{C\Psi s}$$



(a) Sparse s (ℓ_1)

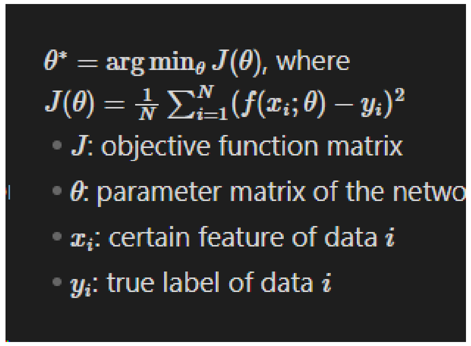
In this equation, why we want a sparse s?

WHAT is the meaning?



Using least squares

Simple machine learning



$\theta^* = \arg\min_\theta J(\theta)$, where

$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i;\theta) - y_i)^2$

- J: objective function matrix
- θ: parameter matrix of the netwo
- x_i: certain feature of data i
- y_i: true label of data i

Using LASSO (with regularization)

In this expression, we should want a sparse \theta?

$$\{\text{dog}, \text{cat}\} = \{+1, -1\}$$

Q: why here is try to plot the coefficient of A instead of x

These are the coefficients of the matrix A from 2 models.

This shows the NN is highly sparse

Activation function

- Recall that linearity limits functional responses
- We extend the mappings to nonlinear functions
- We use the representation: $\mathbf{y} = f(\mathbf{A}, \mathbf{x})$

where $f(\cdot)$ is a specified activation function, some common examples:

$$f(x) = x \quad - \quad \text{linear}$$

$$f(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \quad - \quad \text{binary step}$$

$$f(x) = \frac{1}{1 + \exp(-x)} \quad - \quad \text{logistic (soft step)}$$

$$f(x) = \tanh(x) \quad - \quad \text{TanH}$$

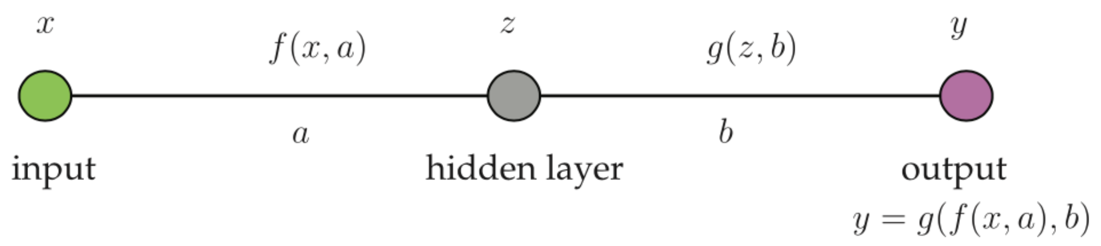
$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \quad - \quad \text{rectified linear unit (ReLU).}$$

The backpropagation algorithm

- An optimization algorithm and an objective function are needed to determine the weights in a NN
- Usually, we use a proxy instead of the true objective function, because of computational convenience
- The backpropagation algorithm (backprop) exploits the compositional nature of NN
- It gives a formulation amenable to standard gradient descent
- It is based on a simple mathematical principle: the chain rule for differentiation

An example

- Consider a single node with a hidden layer NN



- Mathematically, we have

$$y = g(z, b) = g(f(x, a), b)$$

- The functions $f(\cdot)$ and $g(\cdot)$ depends on the weights a and b
- Let y_0 be correct output, y be the NN-approximated output
- We will minimize

$$E = \frac{1}{2}(y_0 - y)^2$$

- The minimization requires

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{dy}{dz}\frac{dz}{da} = 0$$

- Observation: error backpropagate through the network
- Given the functions $f(\cdot)$ and $g(\cdot)$, the term $dy/dz \; dz/da$ can be computed explicitly
- Backprop gives the following gradient iteration

$$a_{k+1} = a_k + \delta \frac{\partial E}{\partial a_k}$$
$$b_{k+1} = b_k + \delta \frac{\partial E}{\partial b_k}$$

- Where δ is called the learning rate

- As an example, consider linear activation function

$$f(\xi, \alpha) = g(\xi, \alpha) = \alpha\xi$$

- We have

$$z = ax$$

$$y = bz$$

- The gradients become

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{dy}{dz}\frac{dz}{da} = -(y_0 - y) \cdot b \cdot x$$
$$\frac{\partial E}{\partial b} = -(y_0 - y)\frac{dy}{db} = -(y_0 - y)z = -(y_0 - y) \cdot a \cdot x$$

Remarks

- Consider a network with M hidden layers, then the derivative becomes

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{dy}{dz_m}\frac{dz_m}{dz_{m-1}}\cdots\frac{dz_2}{dz_1}\frac{dz_1}{da}$$

- Denote all the weights by w, then

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \delta\nabla E$$

where the gradient is computed by composition and chain rule (i.e. propagation) efficiently

The stochastic gradient descent algorithm

- Training a NN is expensive due to the size of the NN
- The stochastic gradient descent (SGD) algorithm gives rapid evaluation of NN weights
- The goal of the training of NN is

$$\underset{\mathbf{A}_j}{\operatorname{argmin}} E(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_M) = \underset{\mathbf{A}_j}{\operatorname{argmin}} \sum_{k=1}^{n} (f(\mathbf{x}_k, \mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_M) - \mathbf{y}_k)^2$$

- We have the following iteration to find the minimum

$$\mathbf{x}_{j+1}(\delta) = \mathbf{x}_j - \delta \nabla f(\mathbf{x}_j)$$

where δ is the learning rate (Note: find the optimal δ is expensive)

- Recall the iteration

$$\mathbf{x}_{j+1}(\delta) = \mathbf{x}_j - \delta \nabla f(\mathbf{x}_j)$$

- The above iteration is expensive due to two reasons
  - The number of weight parameters is large
  - The number of data points is large
- The SGD uses a single data point or a set of data points to approximate the gradient at each iteration
- We reformulate the problem as

$$E(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_M) = \sum_{k=1}^{n} E_k(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_M)$$

where

$$E_k(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_M) = (f_k(\mathbf{x}_k, \mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_M) - \mathbf{y}_k)^2$$

and $f_k(\cdot)$ is the fitting function evaluated at a data point

- The SGD method is described as follows
- We have the following iteration

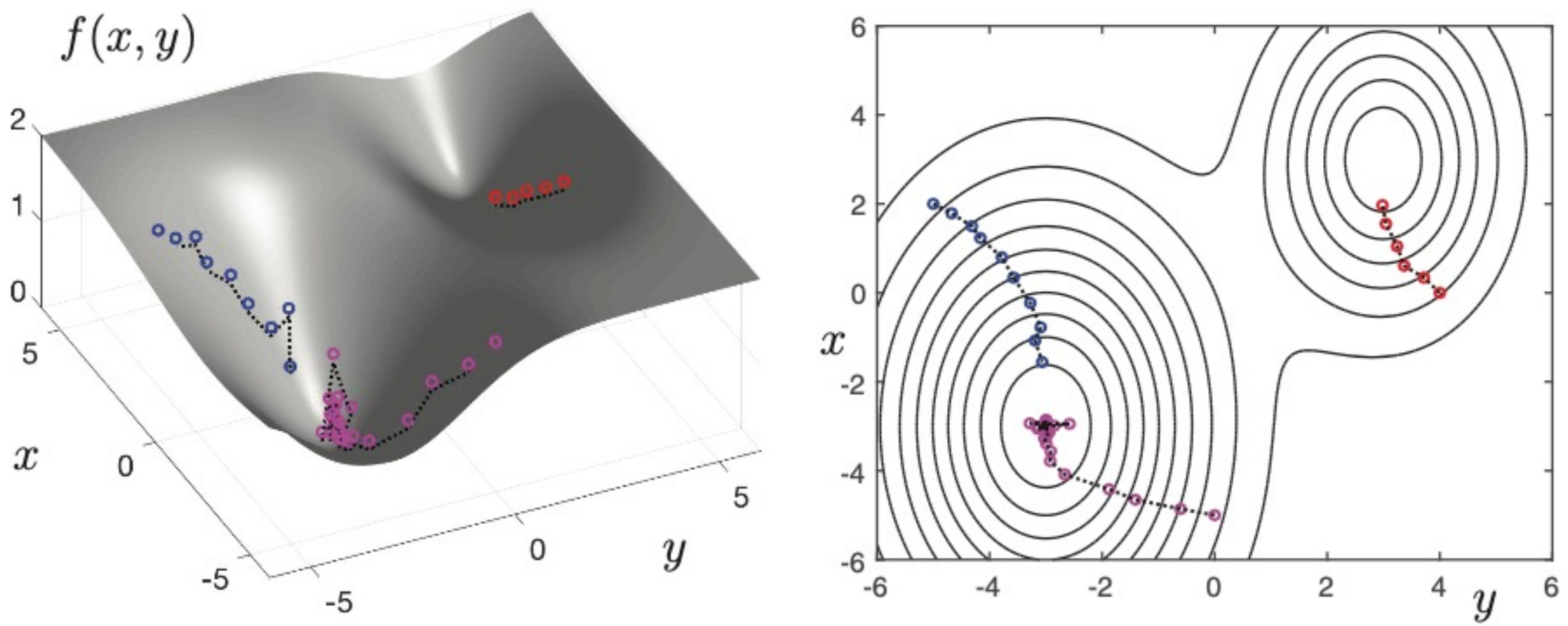
$$\mathbf{w}_{j+1}(\delta) = \mathbf{w}_j - \delta \nabla f_k(\mathbf{w}_j)$$

where w_j is the vector of all weights at the j-th iteration, and the gradient is only computed at the k-th data point

- At the next iteration, another randomly selected data point is used to compute the gradient
- One can also use a subset of points

$$\mathbf{w}_{j+1}(\delta) = \mathbf{w}_j - \delta \nabla f_K(\mathbf{w}_j)$$

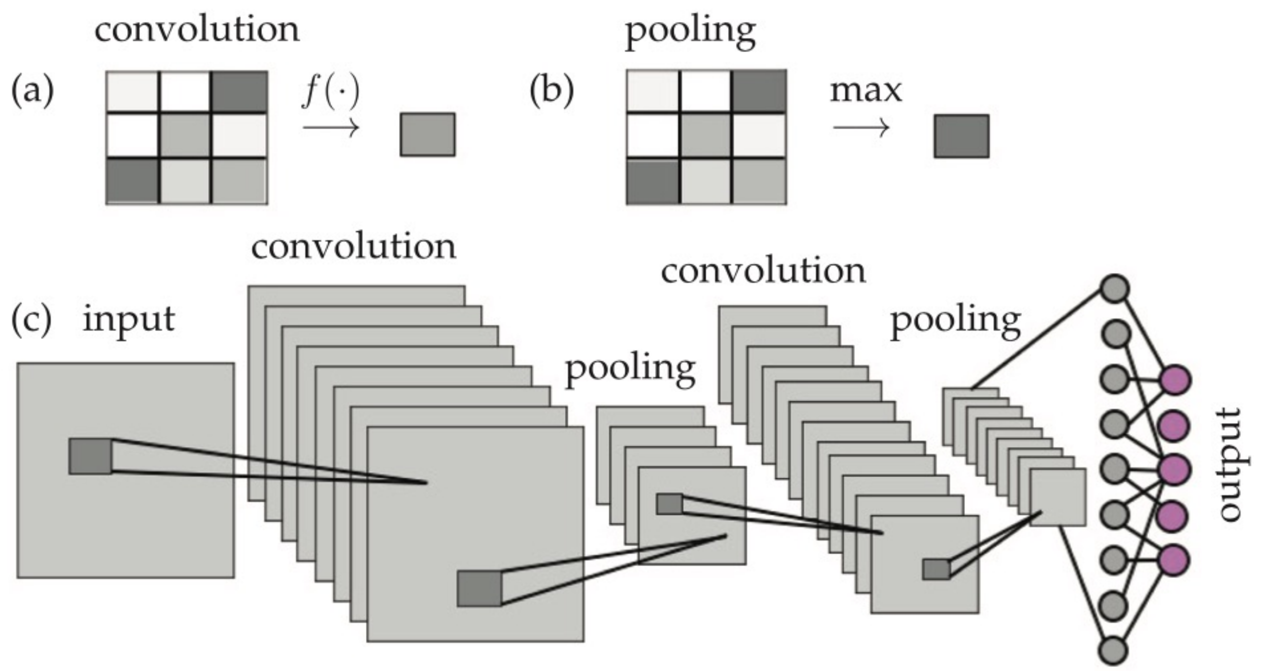
where $K \in [k_1, k_2, \cdots k_p]$ is a set of p randomly selected points



$f(x,y)$
2
1
0
5
0
-5
x
-5
0
5
y
6
4
2
0
-2
-4
-6
x
-6
-4
-2
0
2
4
6
y

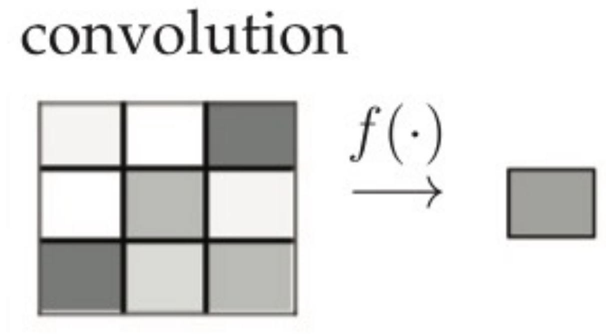
Deep convolutional neural networks

- We will explain the DCNN, which is common in practice
- There are convolutional and pooling layers



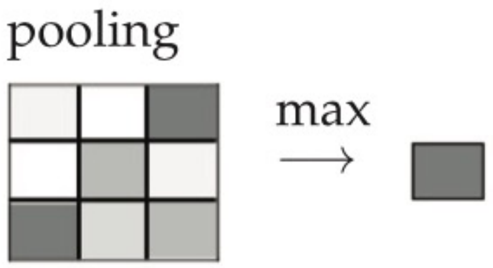
Convolutional layers

- The convolutional layers are similar to windowed transforms
- A small selection of the input space is extracted for feature engineering
- Features are created using suitable activation function



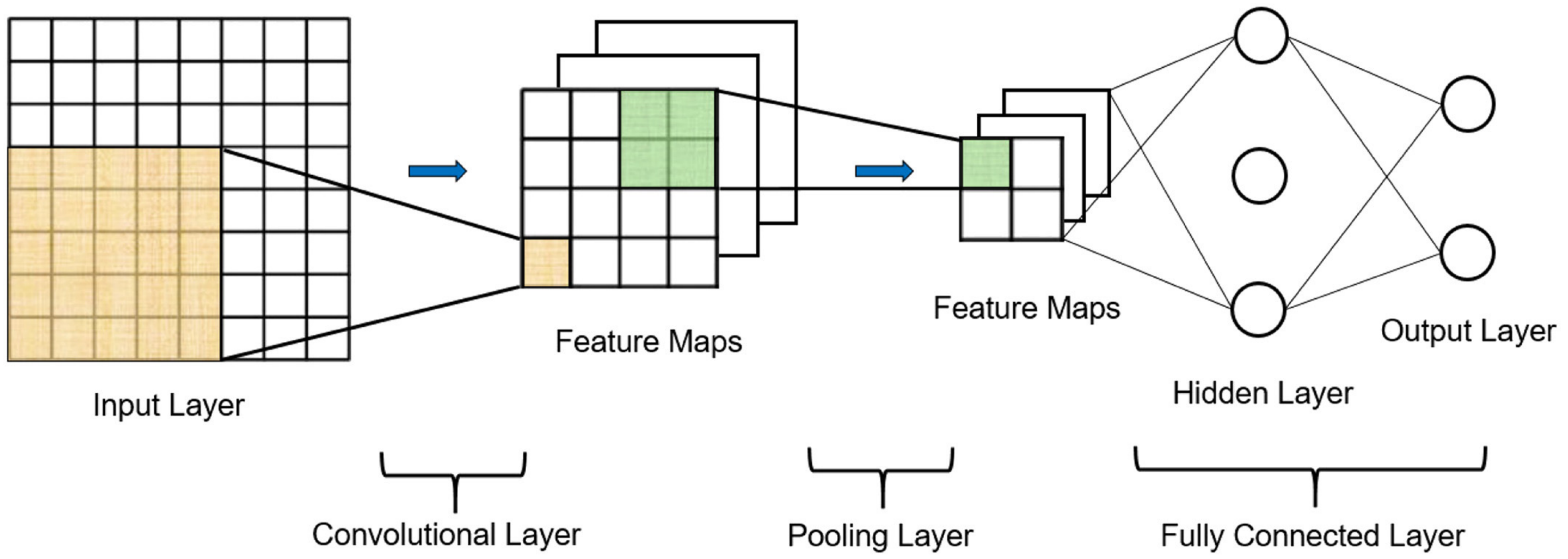
Pooling layers

- It is common to insert a pooling layer between successive convolutional layers
- It is used to reduce the number of parameters and computations in the network
- The max pooling uses the maximum value of all nodes in its convolutional window
- For example, a 3x3 window is transformed to a single number by taking the maximum value of these 9 numbers



An illustration

A fully connected layer is included to allow global connectivity



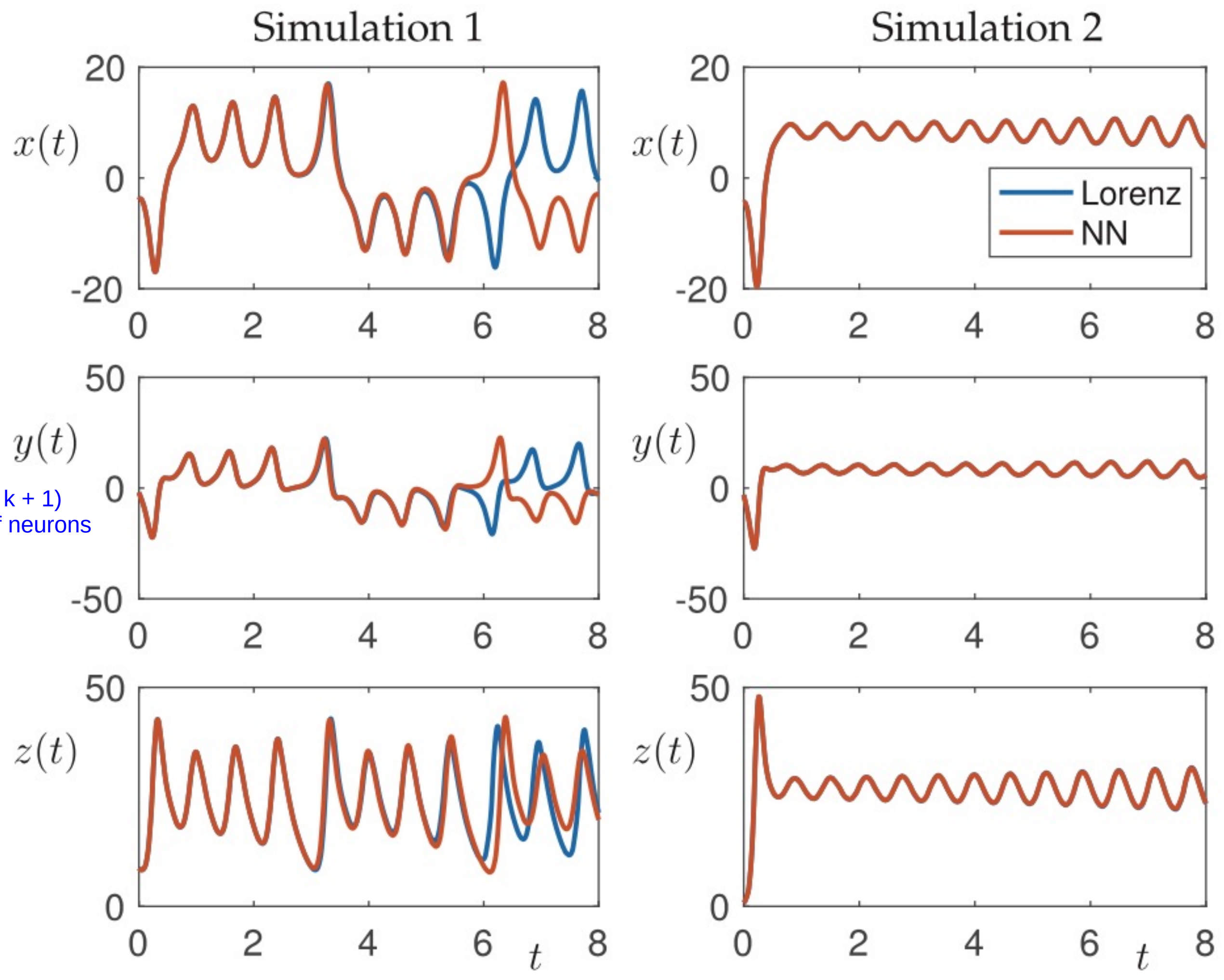
Neural networks for dynamical systems

- NN can be used to compute solutions of dynamical systems
- To illustrate the concepts, we consider the Lorenz system

$$\begin{aligned}\dot{x} &= \sigma(y - x)\\ \dot{y} &= x(\rho - z) - y\\ \dot{z} &= xy - \beta z,\end{aligned}$$

  where $\mathbf{x} = [x \;\; y \;\; z]^T$ is the state of the system
- The goal is to compute $\mathbf{x}_{k+1}$ from $\mathbf{x}_k$, where k denotes the time t_k
- We can use NN to learn this update rule

Example

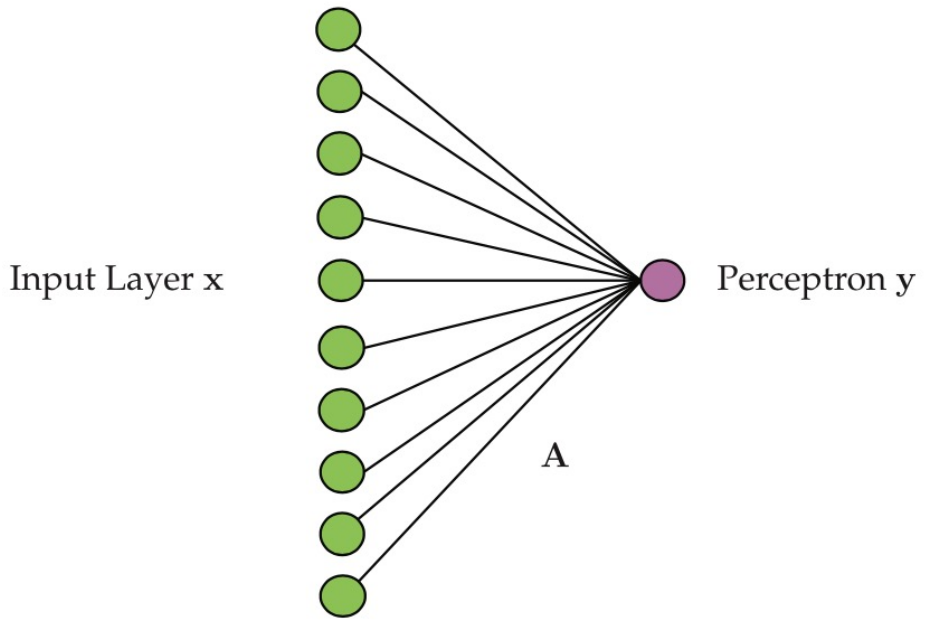


How to do this?

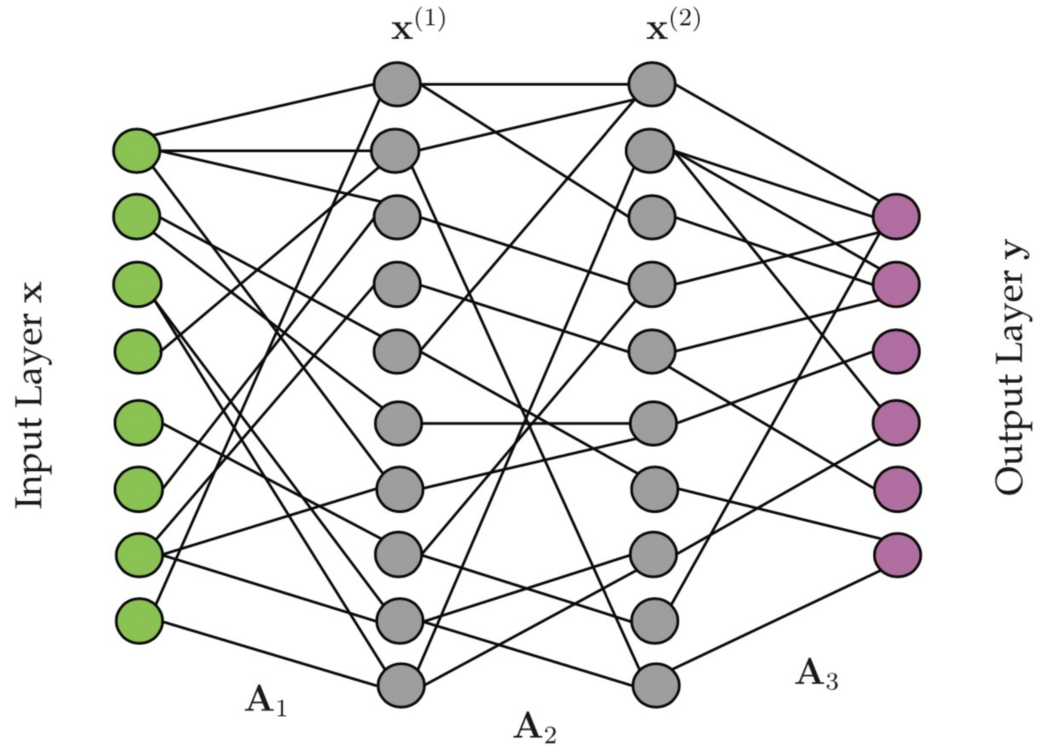
1. Construct some training data (solution at t = k, solution at t = k + 1)
2. Construct NN. Between t = k and t = k + 1, we need layers of neurons

Some NN architectures

- Perceptron
- It has one layer with a single output called the perceptron
- It is used for classification

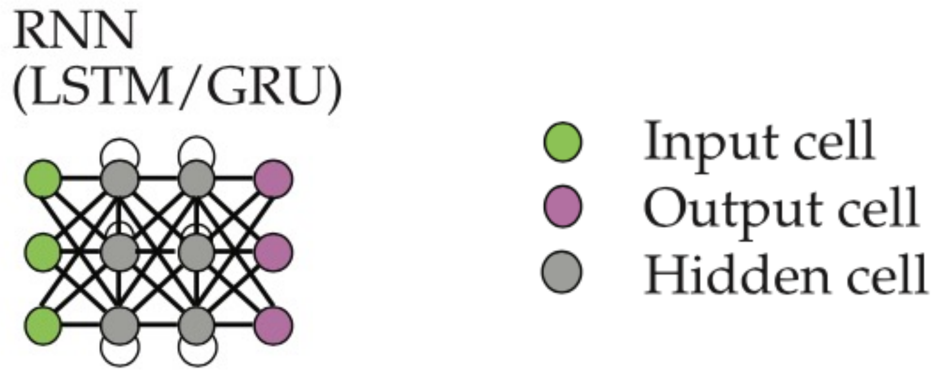


- Feed forward (FF)
- Input and output layers are connected, they do not form a cycle

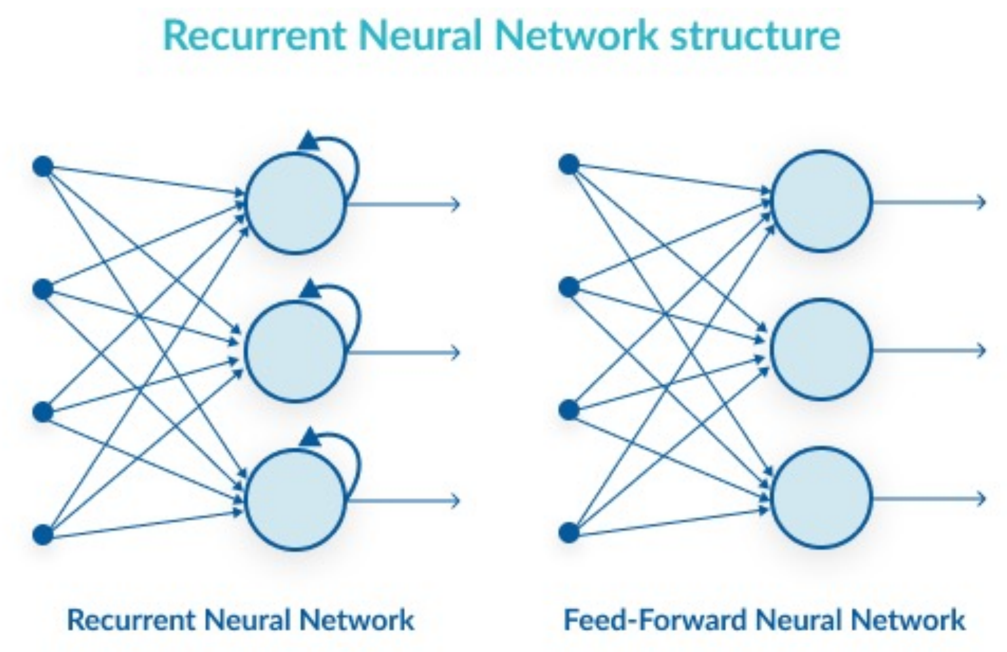


Used to model dynamical system

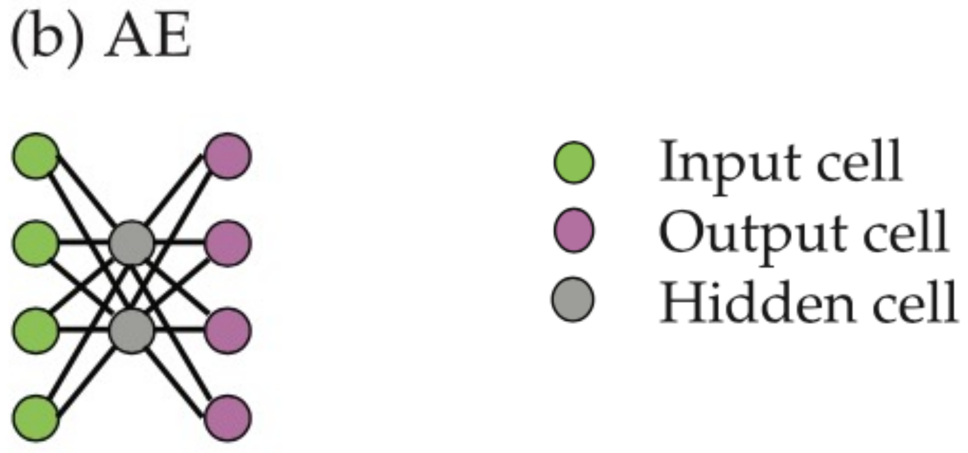
- Recurrent neural network (RNN)
- RNNs are characterized by connections between units that form a directed graph along a sequence
- It allows dynamical temporal behavior
- Importantly, each cell feeds back on itself
- It allows for time delays
- Such controlled states are referred to as gated state or gate memory



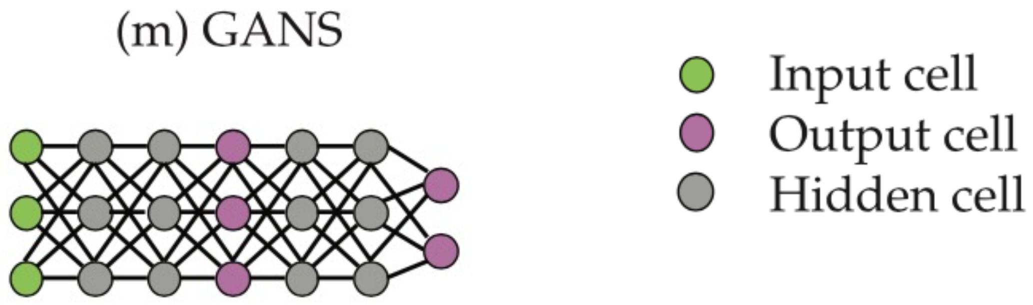
- FF vs RNN


Recurrent Neural Network structure
Recurrent Neural Network
Feed-Forward Neural Network

- Auto encoder (AE)
- The aim is to learn a representation for a set of data, for the purpose of dimensionality reduction
- The input and output cells are matched, so that AE is a nonlinear transform into and out of a new representation
- It is a generalization of linear dimensionality reduction, e.g. PCA



- Generative adversarial network (GAN)
- It trains two networks simultaneously
- One network generates content, and the other attempts to judge
- The generative network learns a map from a latent space to a particular data distribution of interest
- The discriminative network discriminates between instances from the true data and candidates produced by the generator (Identifyung true data or fake data)
- The GAN has produced interesting results in computer vision by producing synthetic data (e.g. images, movies, …)


(m) GANS
Input cell
Output cell
Hidden cell