~~Introduction to Artificial Intelligence~~

Davide Bacciu

davide.bacciu@unipi.it

IN SUPREMÆ DIGNITATIS 1343

Computational Intelligence &
Machine Learning Group

Pervasive Artificial Intelligence
Laboratory

PAI Lab

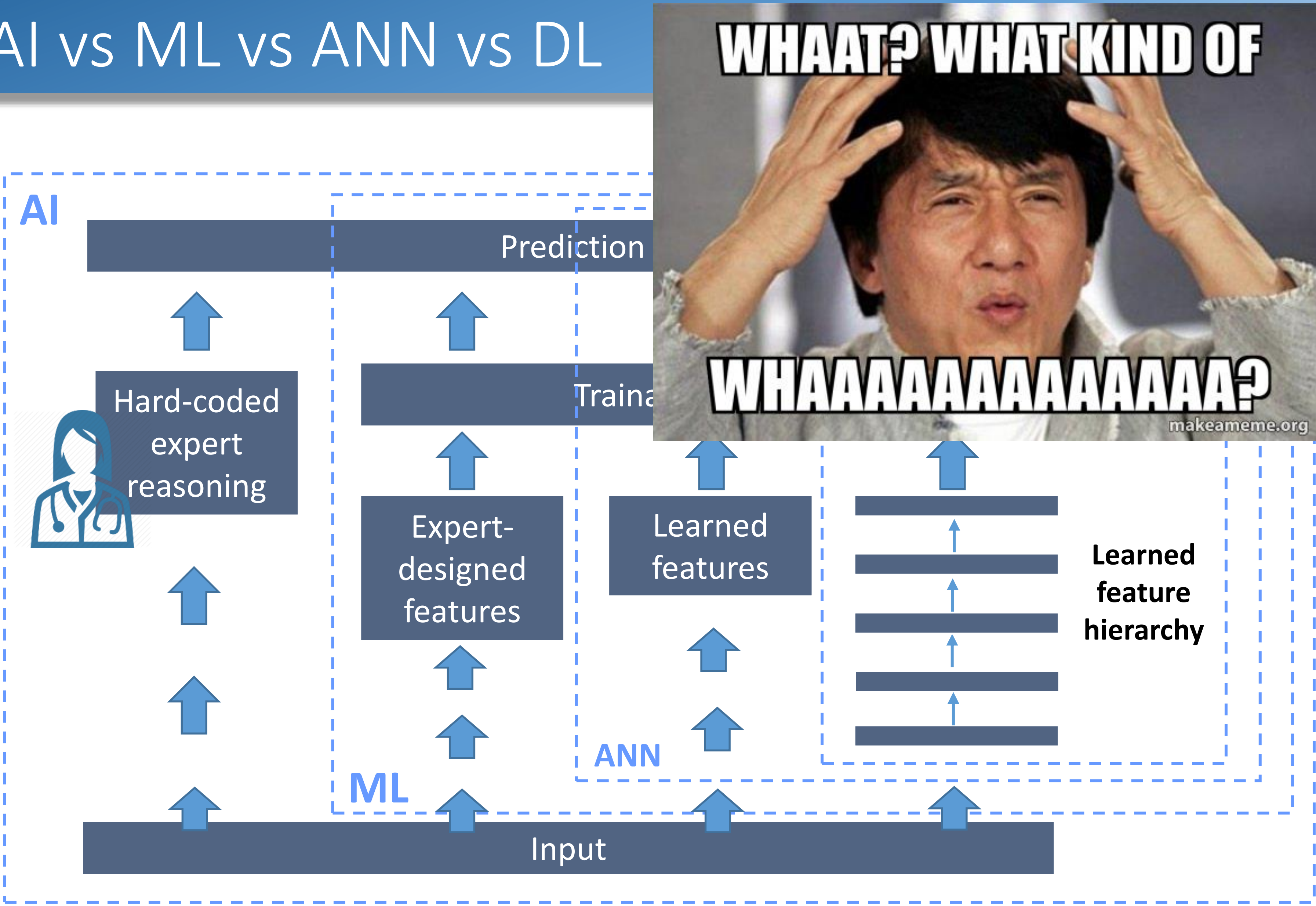
Introduction to (Deep) Neural Networks

Davide Bacciu
davide.bacciu@unipi.it

Computational Intelligence &
Machine Learning Group

Pervasive Artificial Intelligence
Laboratory

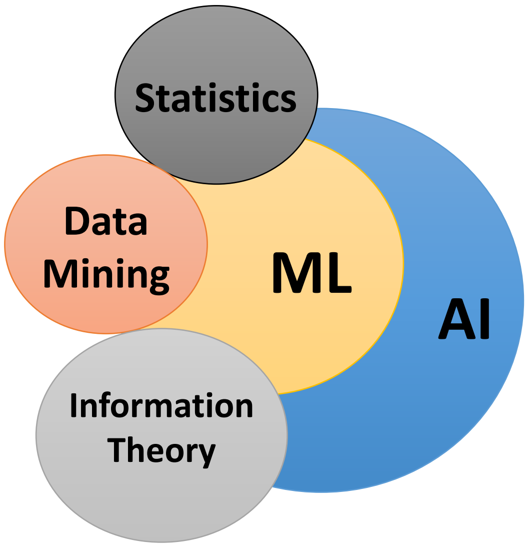
AI vs ML vs ANN vs DL



Outline

- Introduction
- Machine learning preliminaries
- Neural Networks basics
  - Neuron model
  - Architectural aspects
  - Multilayer perceptron (vectors)
- Deep learning
  - Convolutional neural networks (images)
  - Recurrent neural networks (sequences)

Machine Learning (ML)



Machine Learning is a field of artificial intelligence dealing with models and methods that allow computer to learn from data

Machine Vision

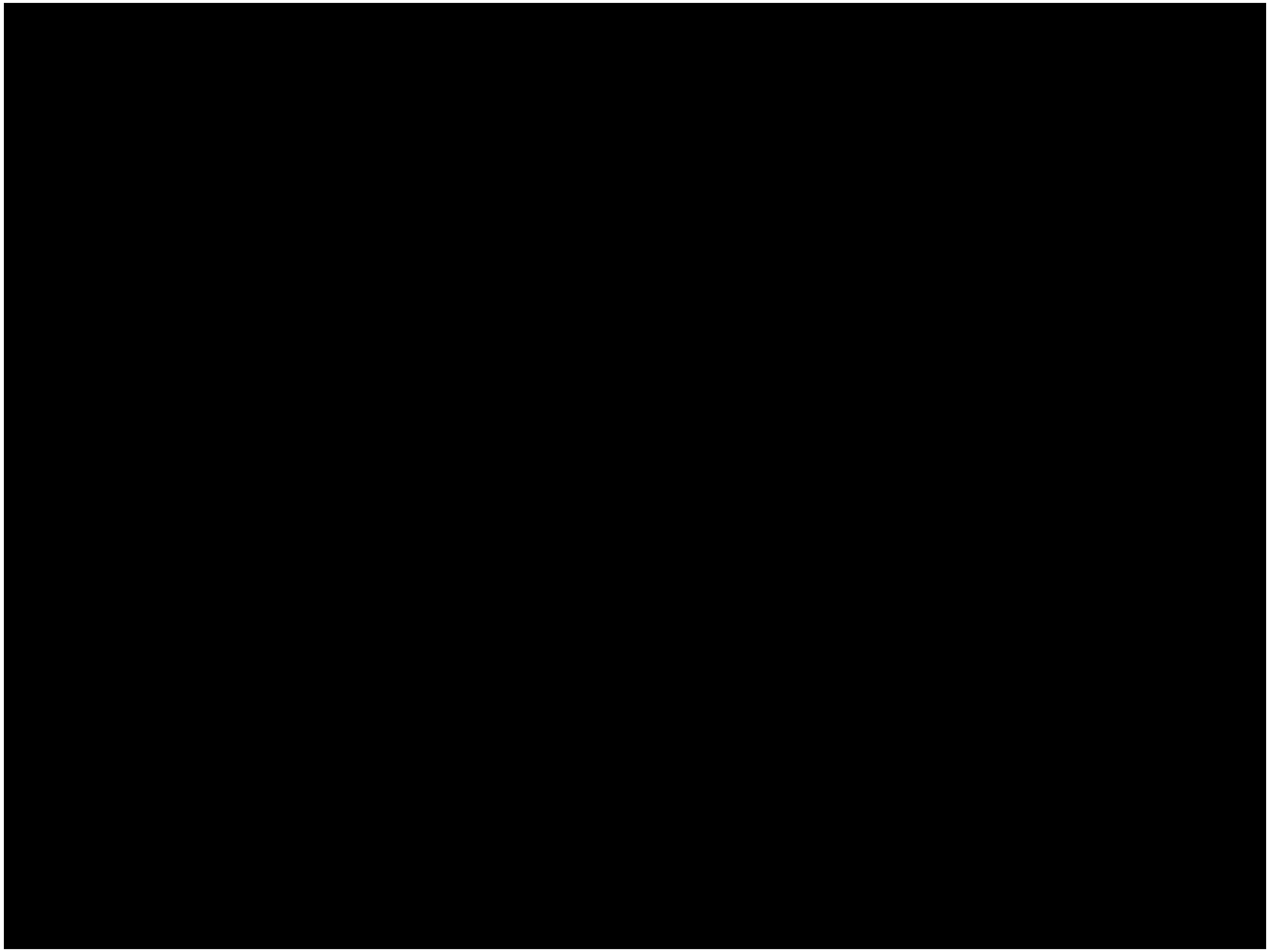
"A cat is sitting on a toilet seat"
(NeuralTalk)

...some evident open issues..

"A woman holding a teddy bear in front of a mirror"

Autonomous Driving

Deep Reinforcement Learning

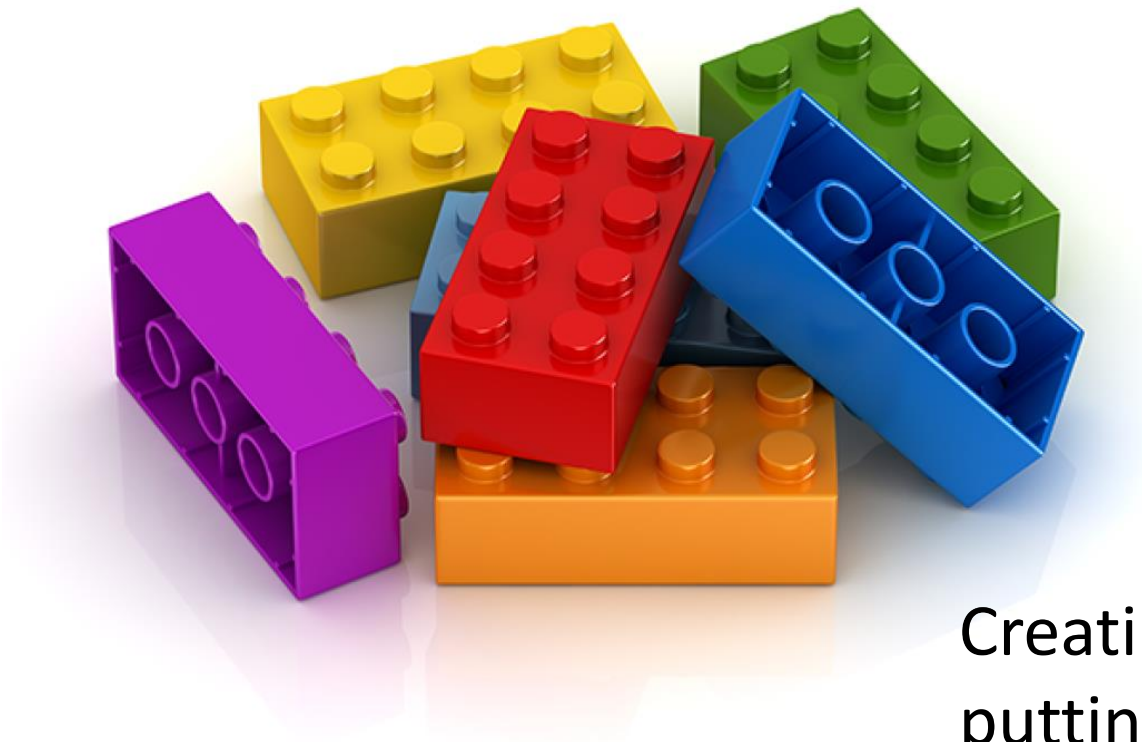


Using Machine Learning to Generate Images

Generative Adversarial Networks

Create faces of non-existing people

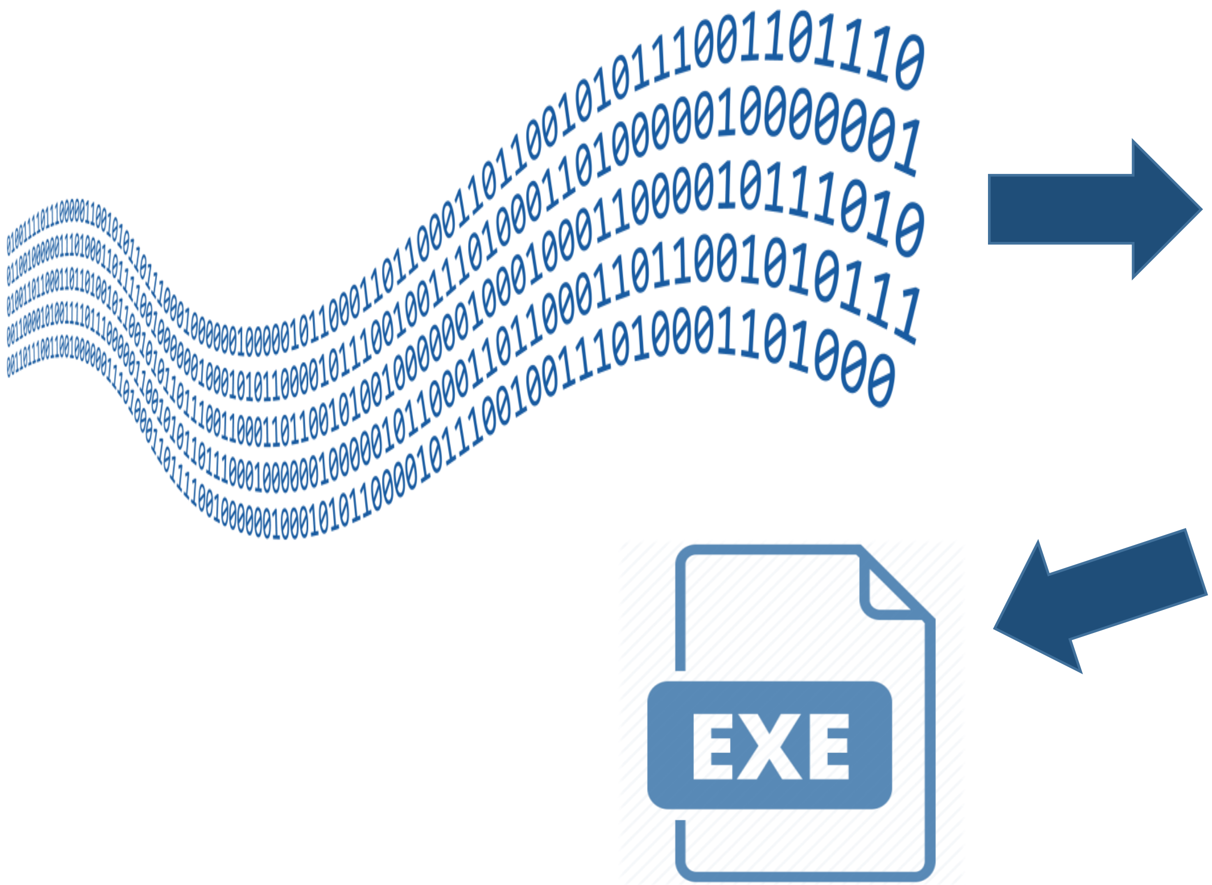
The Deep Learning Lego



Creating applications by putting together various combinations of basic types of neural networks

Differentiable Programming

Software development as a data-driven process



$$\frac{\partial P}{\partial w}$$

Python

- Support for vectorization and GPU
- Loads of useful libraries for Machine learning, Deep learning, Machine vision

The reference language for machine learning

ML preliminaries

Learning from examples

- Acquisition (inference/induction) from data (examples) of the rules, models or representations which enable the production of a desired behaviour
- The goal is not to memorize but to generalize the acquired knowledge
  - More than simply fitting the data
  - Estimating the value of function for unseen examples
- Given a set of *N* examples

$$(x_1, y_1); (x_2, y_2) \dots (x_N, y_N)$$

find a function *f*(·) such that it is a good predictor of *y* for a future input *x*

ML – Tasks & Data

Supervised Learning

Learn an unknown function predicting an output in response to an input

- Predicting credit risk given customer profile

$$(x, y)$$

Unsupervised Learning

Identification of structures, regularities associations and anomalies in the data

- Signaling anomalous transactions

$$(x)$$

Reinforcement Learning

Learning of a policy or complex behaviour while being allowed to observe only partial responses from the interaction with the environment or the user

- Autonomous agents

$$(s, a, r)$$

Empirical Error (Supervised Case)

Suppose we have a finite set

$$D = (x_1, y_1); (x_2, y_2) \ \dots \ (x_N, y_N)$$

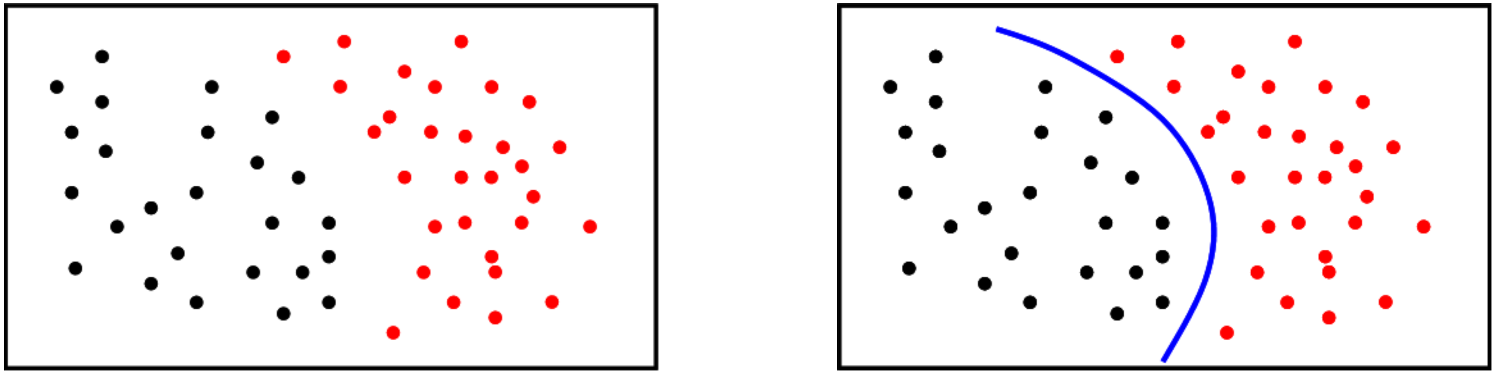
providing the target values y_i over *N* samples

The empirical (sample) error of model *M* with respect to the sample *D* is

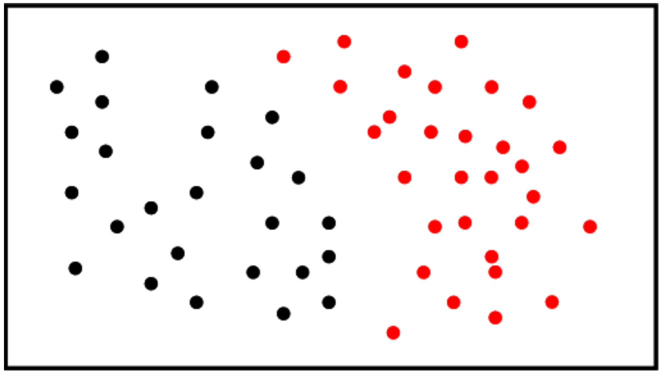
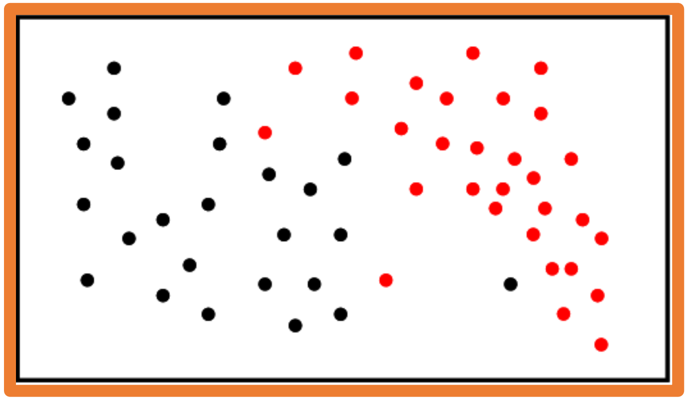
$$Err_D(M) = \sum_{(x_i, y_i)} J(M(x_i), y_i)$$

where $J(M(x_i), y_i)$ is the loss, i.e. a function measuring the discrepancy between the predicted $M(x_i)$ and the target value y_i

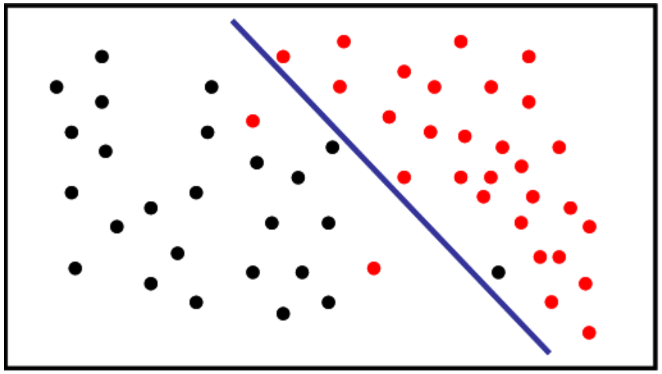
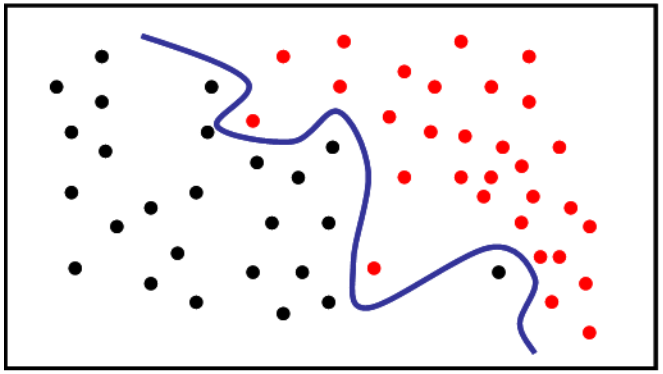
Empirical Risk & Model Complexity



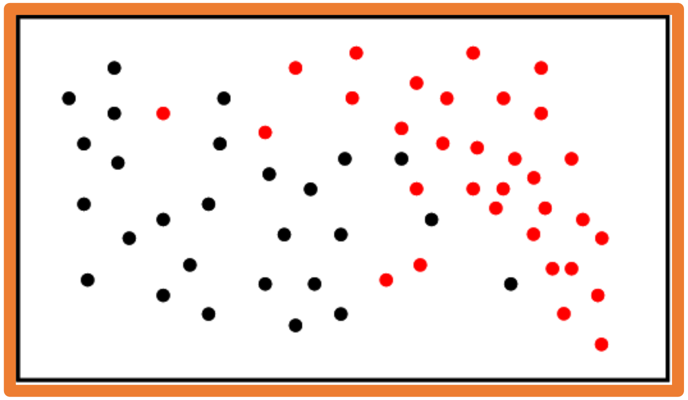
Empirical Risk & Model Complexity

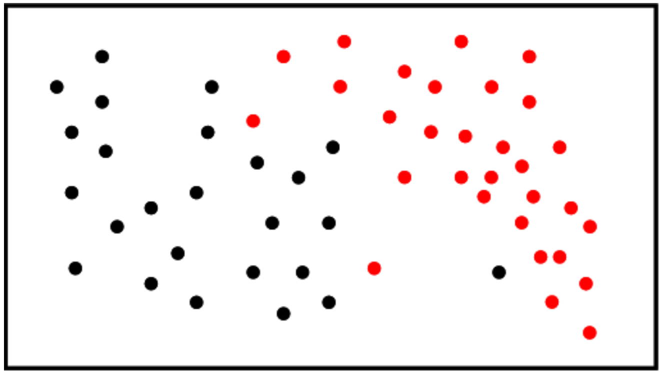



Best model now?

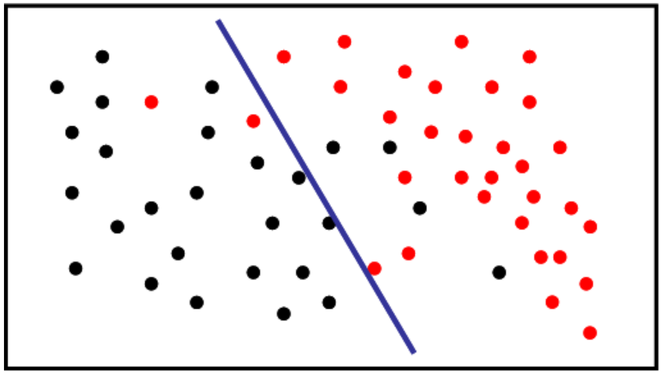



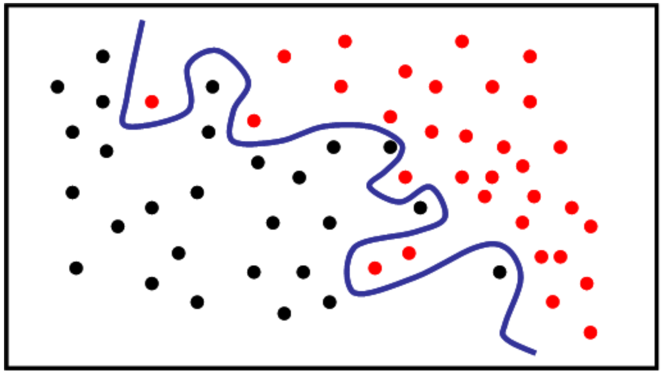
Empirical Risk & Model Complexity





Bias-Variance Dilemma





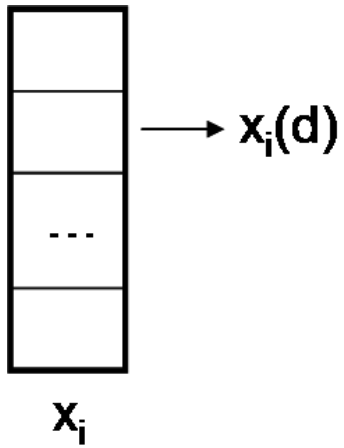
Key Ingredients of Machine Learning

- Data
- Tasks
- Learning Machinery
  - Computational model - how knowledge is represented
    - Linear regression
    - Bayesian Classifier
    - Neural Networks
  - Learning algorithm - how knowledge is adapted to the observations (examples)
    - Backpropagation
    - Expectation-Maximization
- Validation: measures of learning quality and performance

ML – Information Representation

Vectorial data

- The *i*-th input sample x_i is a *D*-dimensional numerical vector
  - Continuous, categorical or mixed values
  - Describes an individual of our world of interest, e.g. patients in a biomedical application
- The single dimensions d are called features and numerically represent an attribute of the individual
  - E.g. if x_i describes a patient, $x_i(d)$ can be his/her age
- Also output samples y_i are *D'*-dimensional numerical vectors



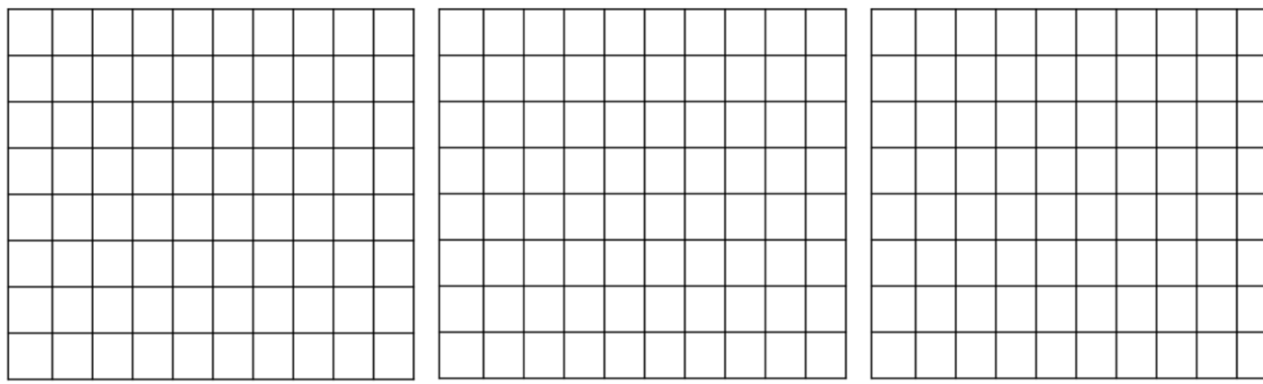
ML – Information Representation

Images

Images are matrices of pixels intensity

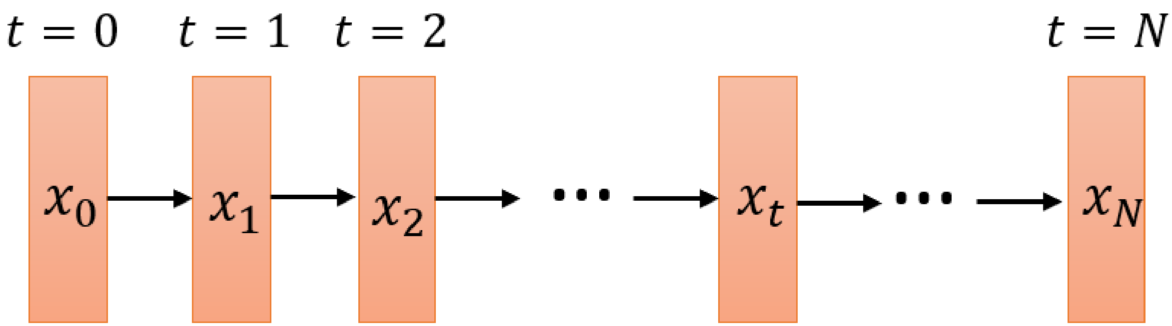
10x10x3

R G B



ML – Information Representation

Sequential data



- Variable size data characterized by sequentially dependent information
- Examples: financial timeseries, sequences of operations, natural language sentences, …
- Each element of the sequence is a vector
- In ML can be used both as input and output information

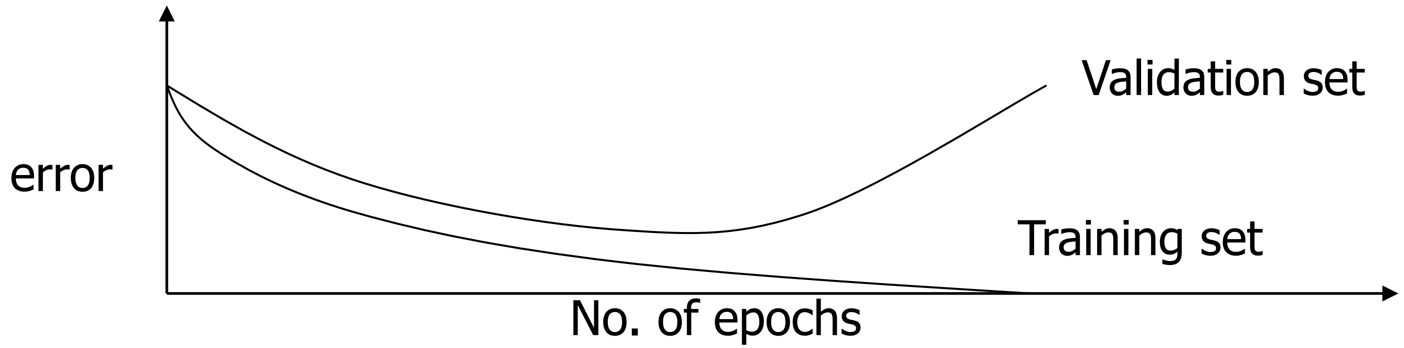
Dataset Preparation

Dataset should normally be split into three sets as follows:

- Training set – use to update the weights. Patterns in this set are repeatedly in random order. The weight update equation are applied after a certain number of patterns.
- Validation set – use to decide when to stop training only by monitoring the error and to select the best model configuration
- Test set – Use to test the performance of the neural network. It should not be used as part of the neural network development and model selection cycle

Model Selection

- Statistically sound validation techniques should be used to determine model hyperparameters
  - Non-adaptive user-chosen model parameters
  - E.g. architecture of neural networks, penalty weighting, optimization algorithm setup...
- Use validation error to select the best model configuration



Regularization

- Constrain the learning model to avoid overfitting and help improving generalization
- Add penalization terms to the error function that *punishes* the model for excessive use of resources
  - Limit the amount of parameters that are used to learn a task
  - Limit the total activation of neurons in the network

$$J' = J(y, y^*) + \lambda R(\cdot)$$

Hyperparameter to be chosen in model selection

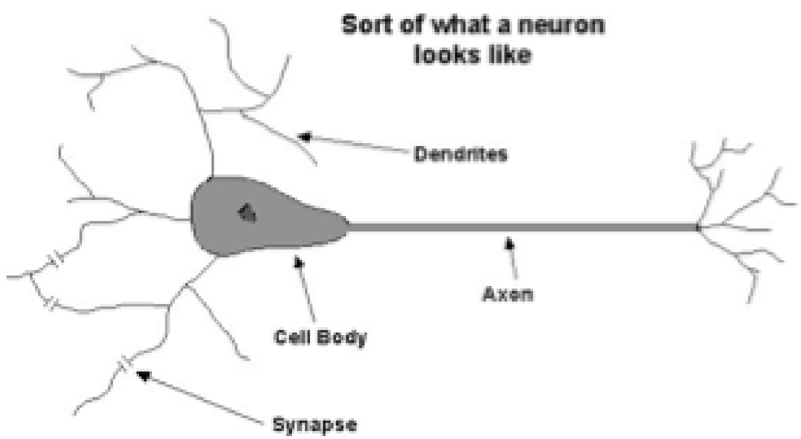
$$||A||_1 = \sum_{ij} |a_{ij}|$$

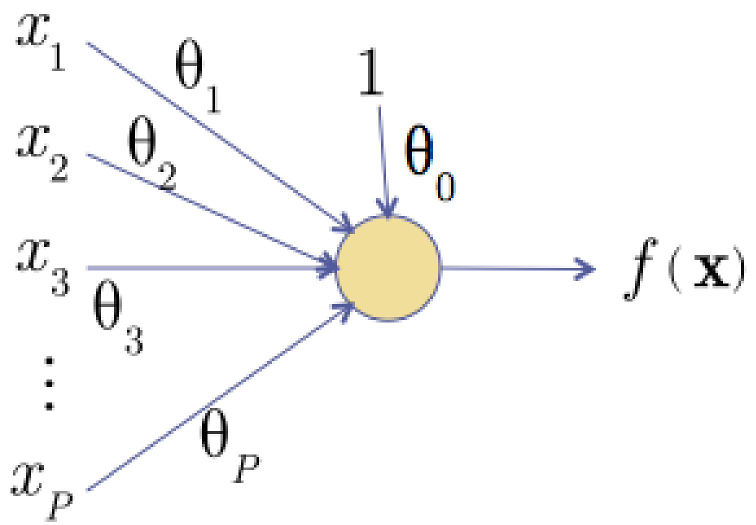
$$||A||_2 = \sqrt{\sum_{ij} a_{ij}^2}$$

Neural Networks

The Neuron Metaphor

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node





Characterizing the Artificial Neuron (I)

- Input/Output signal may be.
  - Real value.
  - Unipolar {0, 1}.
  - Bipolar {-1, +1}.
- Weight : ϑ_{ij} – strength of connection from unit **unit *j* to unit *i***
- Learning amounts to adjusting the weights ϑ_{ij} by means of an optimization algorithm aiming to minimize a cost function

Characterizing the Artificial Neuron (II)

- The bias *b* is a constant that can be written as $\vartheta_{i0}x_0$ with $x_0 = 1$ and $\vartheta_{i0} =$ b such that

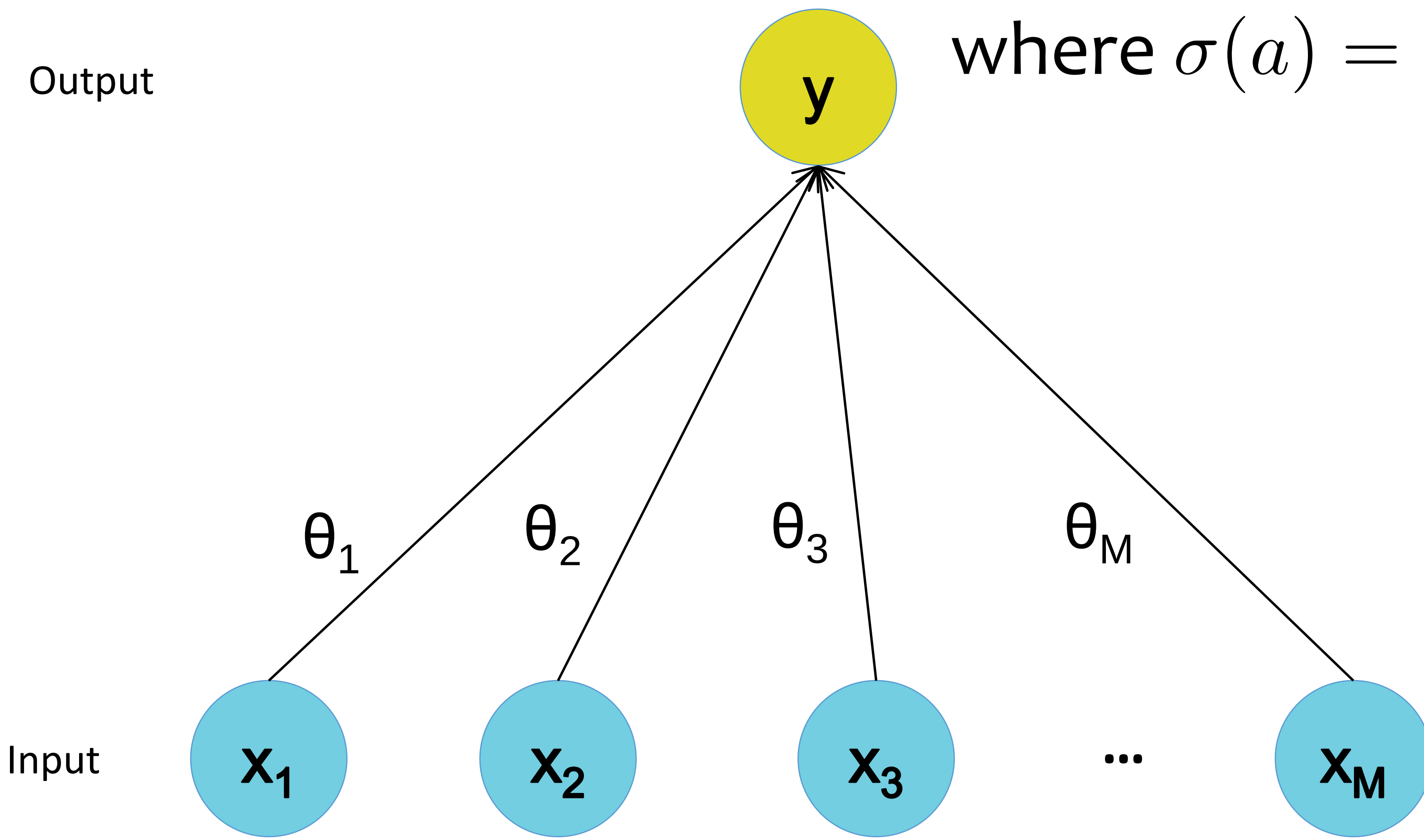
$$net_i = \sum_{j=0}^{n} \vartheta_{ij} x_j$$

- The function $f(net_i(x))$ is the unit's activation function. In the simplest case, f is the identity function, and the unit's output is just its net input. This is called a *linear unit*

A Simple Linear Neuron

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = a$

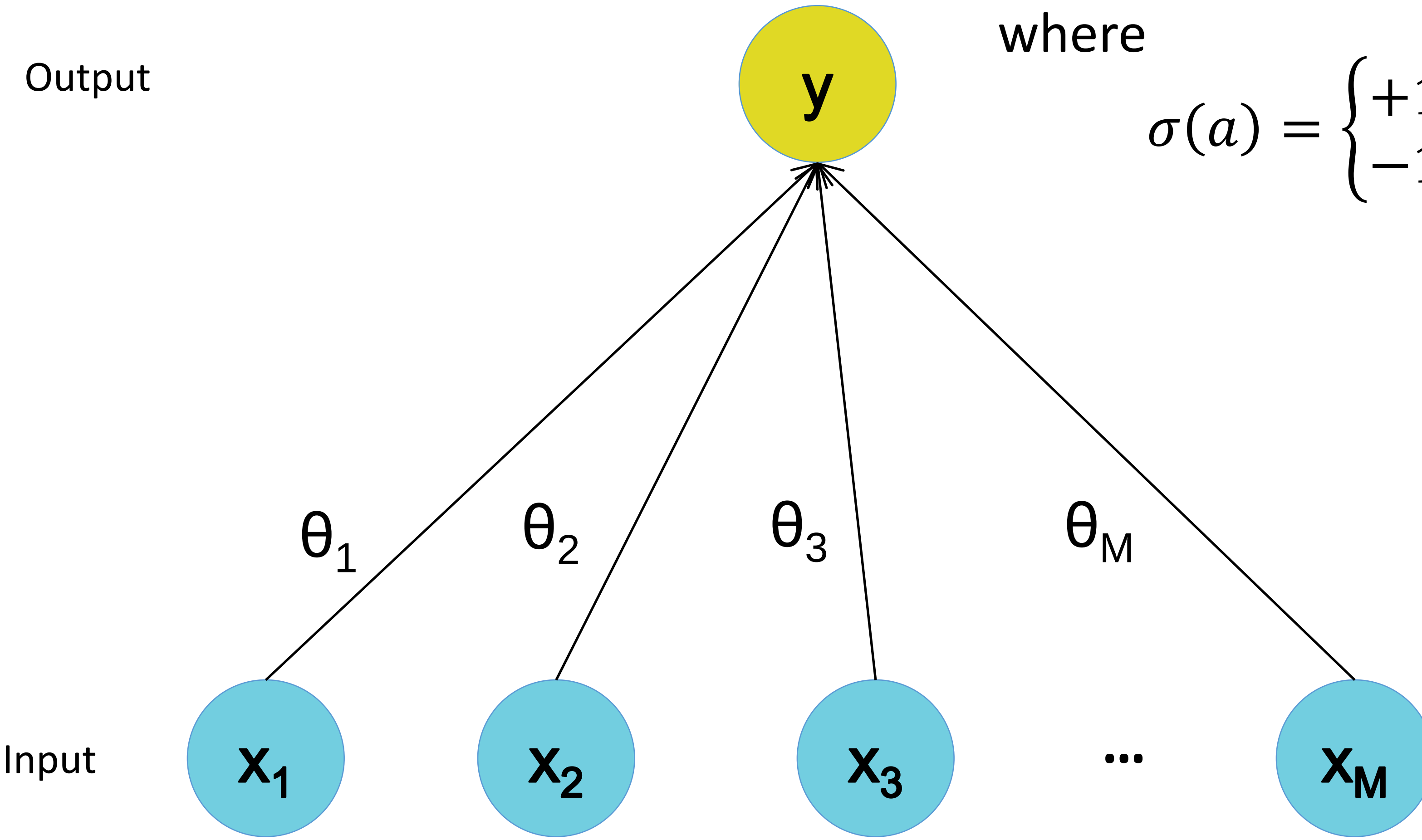


Linear Threshold Unit (a.k.a. Perceptron)

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where

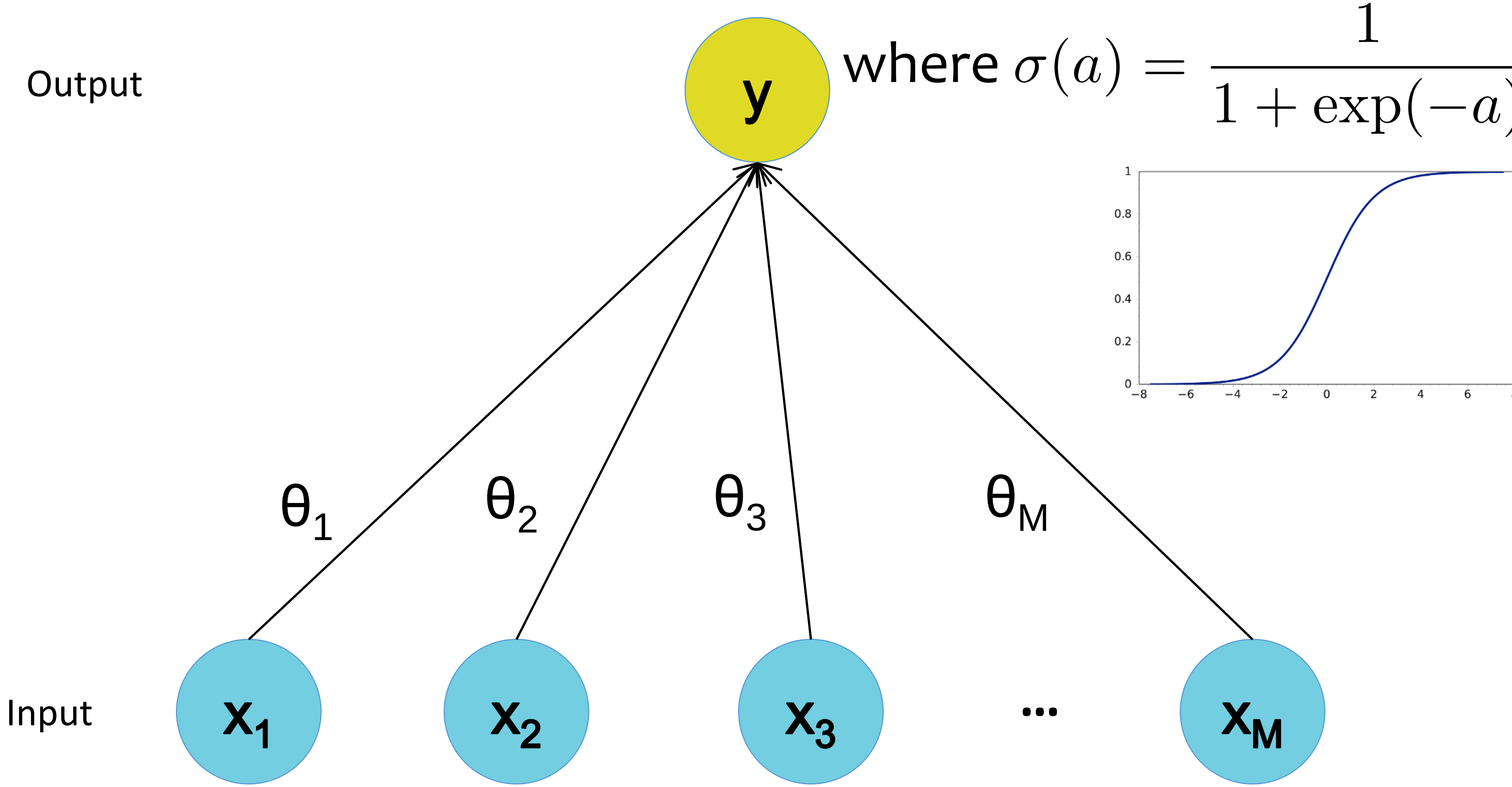
$$\sigma(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$



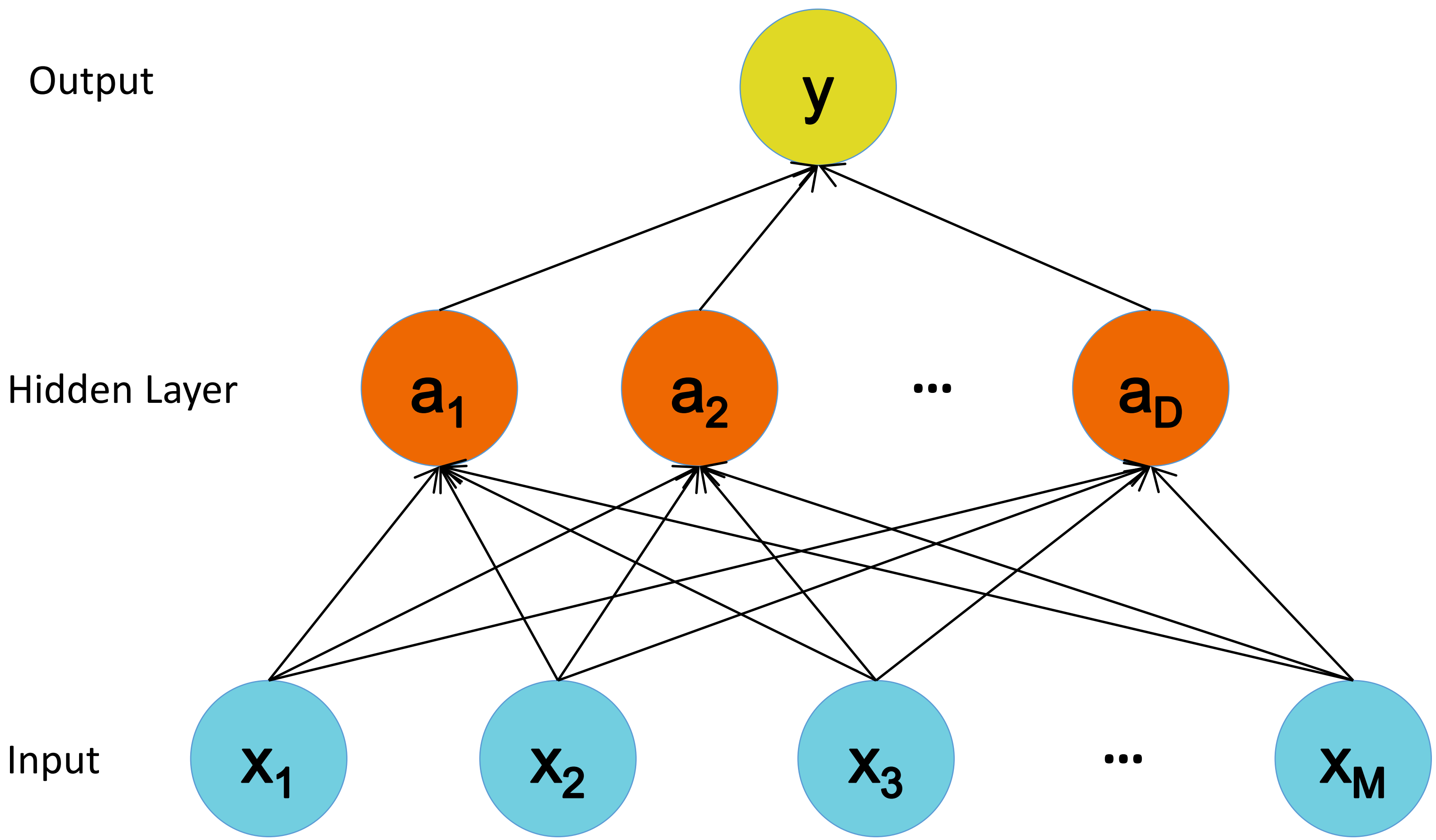
The Logistic Neuron

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

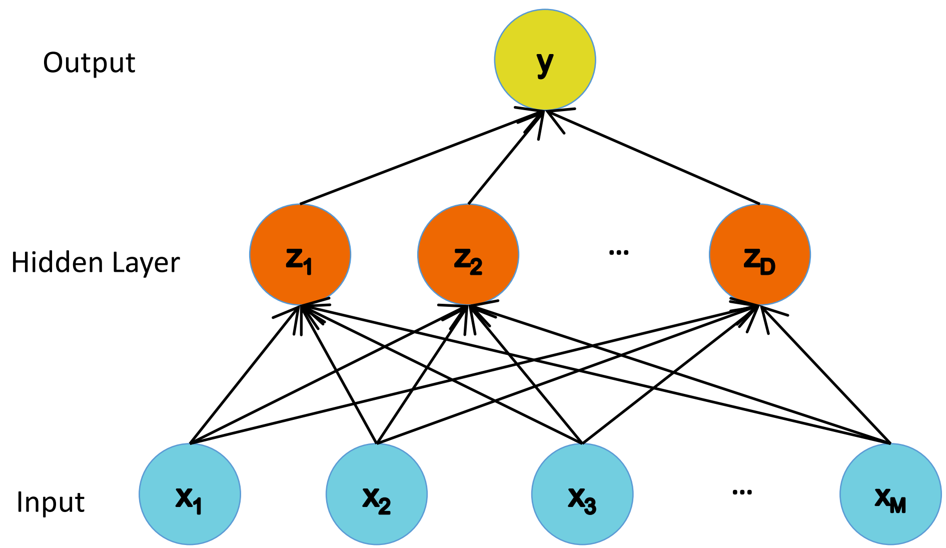
where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



Multilayer Perceptron



Multilayer Perceptron



(E) **Output (sigmoid)**
$$y = \frac{1}{1+\exp(-b)}$$

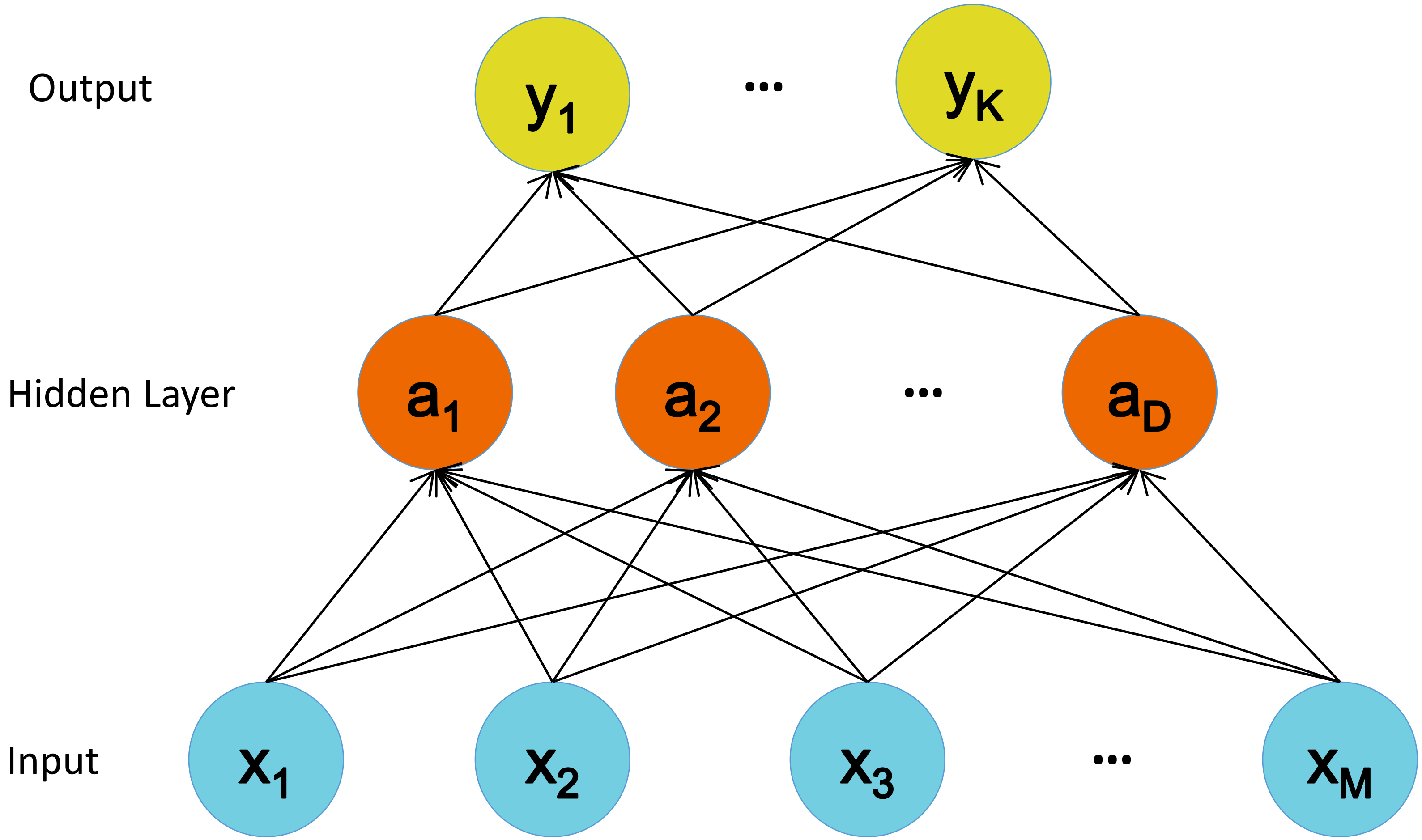
(D) **Output (linear)**
$$b = \sum_{j=0}^{D} \beta_j z_j$$

(C) **Hidden (sigmoid)**
$$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$$

(B) **Hidden (linear)**
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$$

(A) **Input**
Given $x_i, \ \forall i$

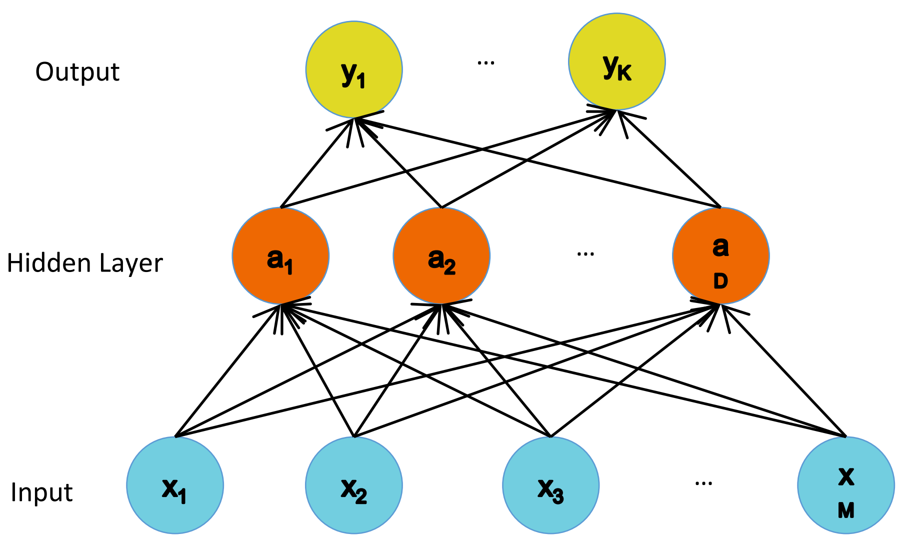
Multiple-Multiclass Outputs

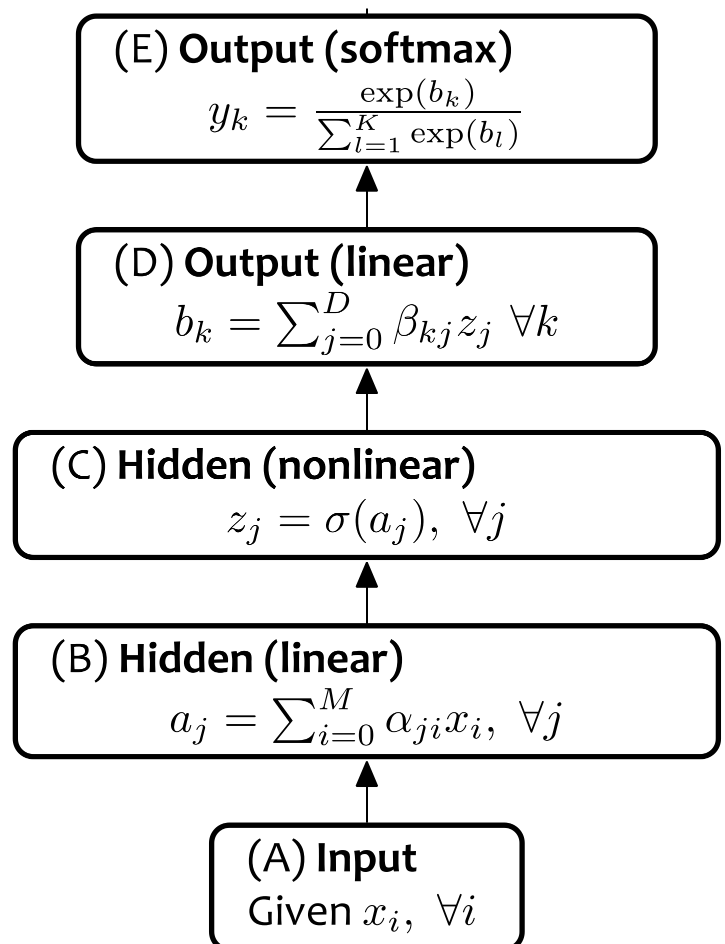


Multi-Class Output

Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$$





Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

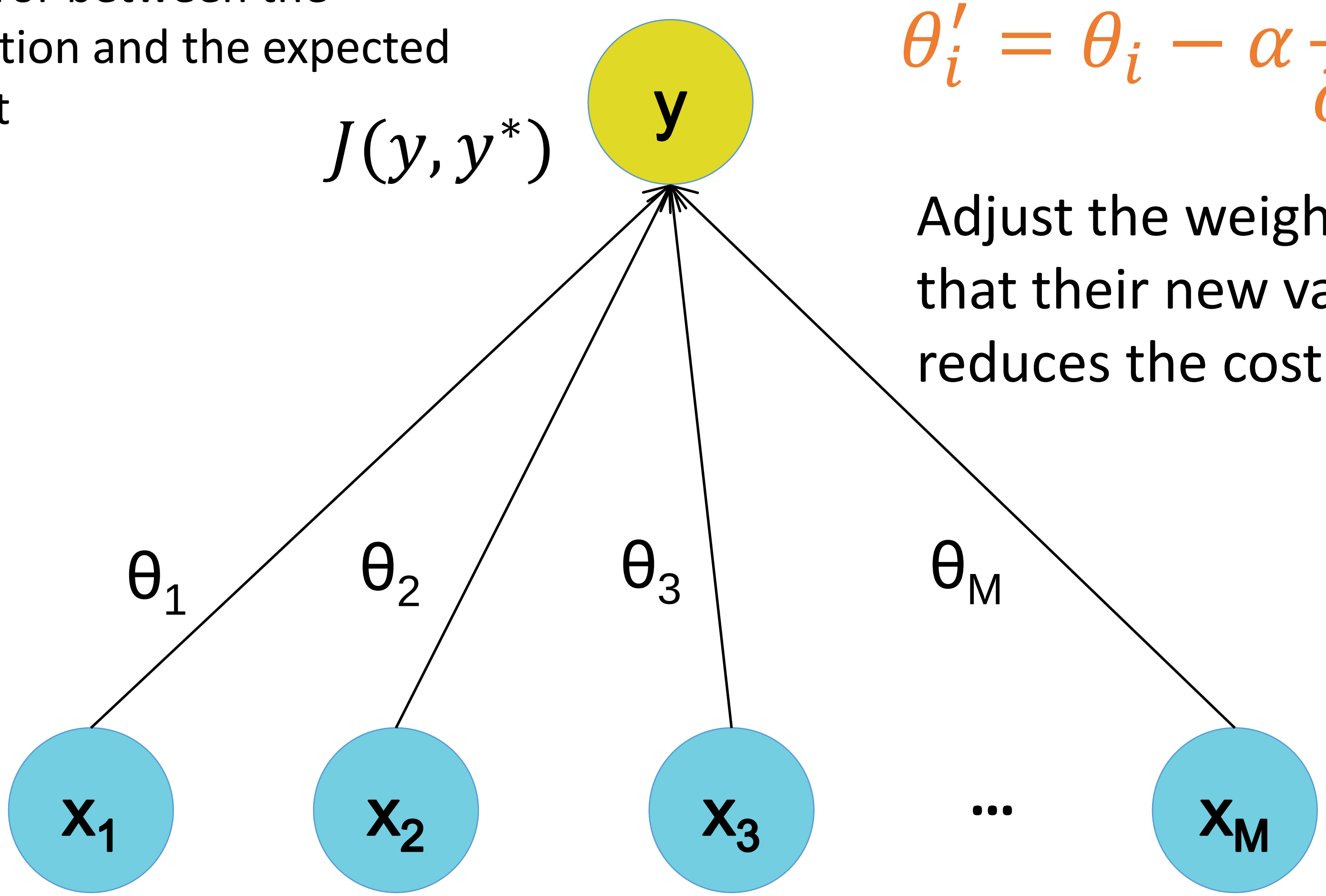
1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function for each layer
4. Loss function
5. Connectivity patterns
6. Weight sharing

…

Training NNs – Cost minimization

Compute a cost function, e.g. the error between the prediction and the expected output

$$\theta_i' = \theta_i - \alpha \frac{\partial J}{\partial \theta_i}$$

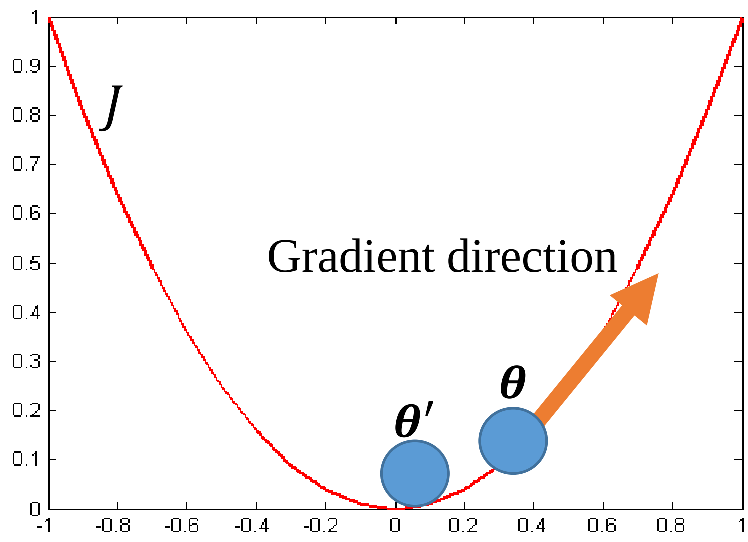


Adjust the weights so that their new value reduces the cost J

Gradient Descent

Weights are updated in the opposite direction of the gradient of the loss function

$$\theta_i' = \theta_i - \alpha \frac{\partial J}{\partial \theta_i}$$

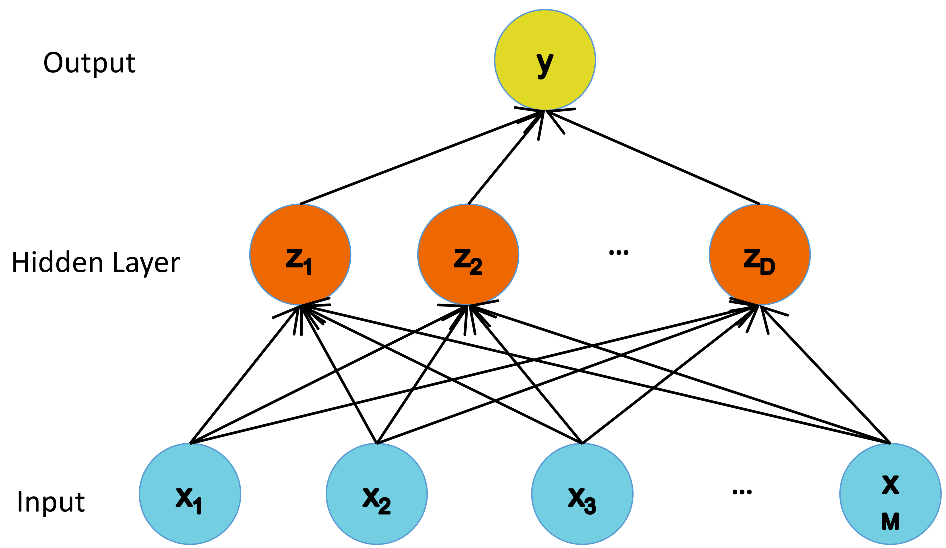


Gradient direction is the direction of uphill of the error function.

By taking the negative we are going downhill

Hopefully to a minimum of the error

Training Multilayer NNs



(E) Output (sigmoid)
$y = \frac{1}{1+\exp(-b)}$

(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

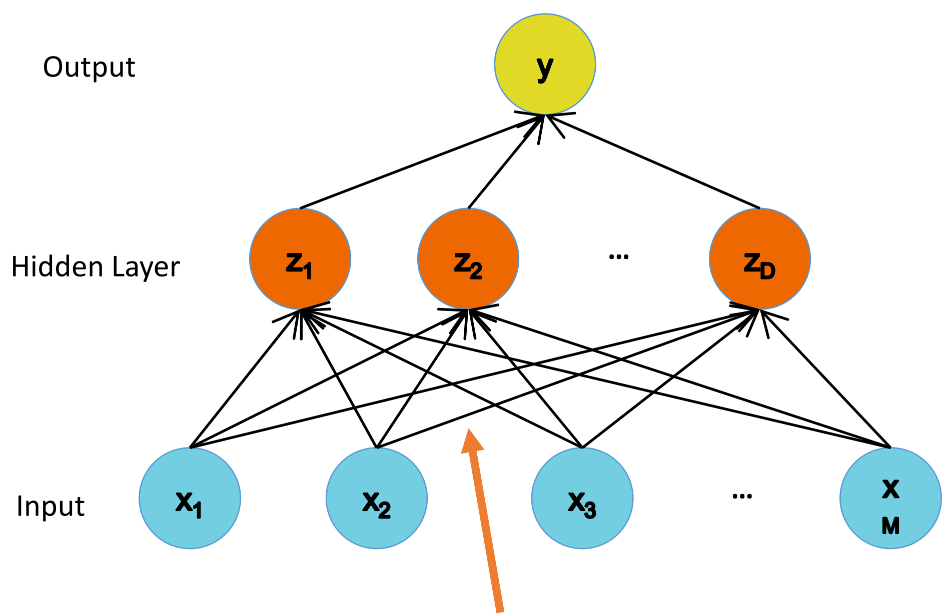
(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) Input
Given $x_i, \ \forall i$

Training Multilayer NNs

$$J(y, y^*)$$



How do we update these weights given the loss is available only at the output unit?

(F) Loss
$J = \frac{1}{2}(y - y^*)^2$

(E) Output (sigmoid)
$y = \frac{1}{1+\exp(-b)}$

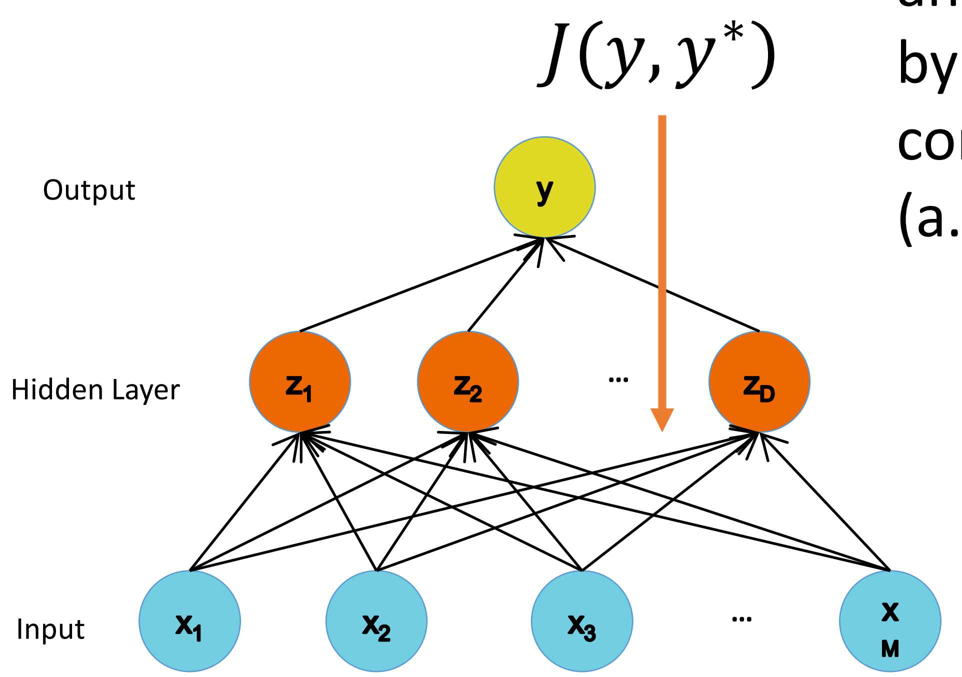
(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \;\; \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \;\; \forall j$

(A) Input
Given $x_i, \;\; \forall i$

Error Backpropagation



Error is computed at the output and propagated back to the input by chain rule to compute the contribution of each weight (a.k.a. derivative) to the loss

A 2-step process

1. Forward pass - Compute the network output (model.predict())
2. Backward pass – Compute the loss function gradients and update (model.fit())

Convergence Criteria

- Learning is obtained by repeatedly supplying training data and adjusting by backpropagation
  - Typically 1 training set presentation = 1 epoch
- We need a stopping criteria to define convergence
  - Euclidean norm of the gradient vector reaches a sufficiently small value
  - Absolute rate of change in the average squared error per epoch is sufficiently small
  - Validation for generalization performance : stop when generalization performance reaches a peak

Neural Network in 1 Slide

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

– Penalty (optional)

$$\lambda R(\cdot)$$

3. Define goal:

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

4. Train with SGD:

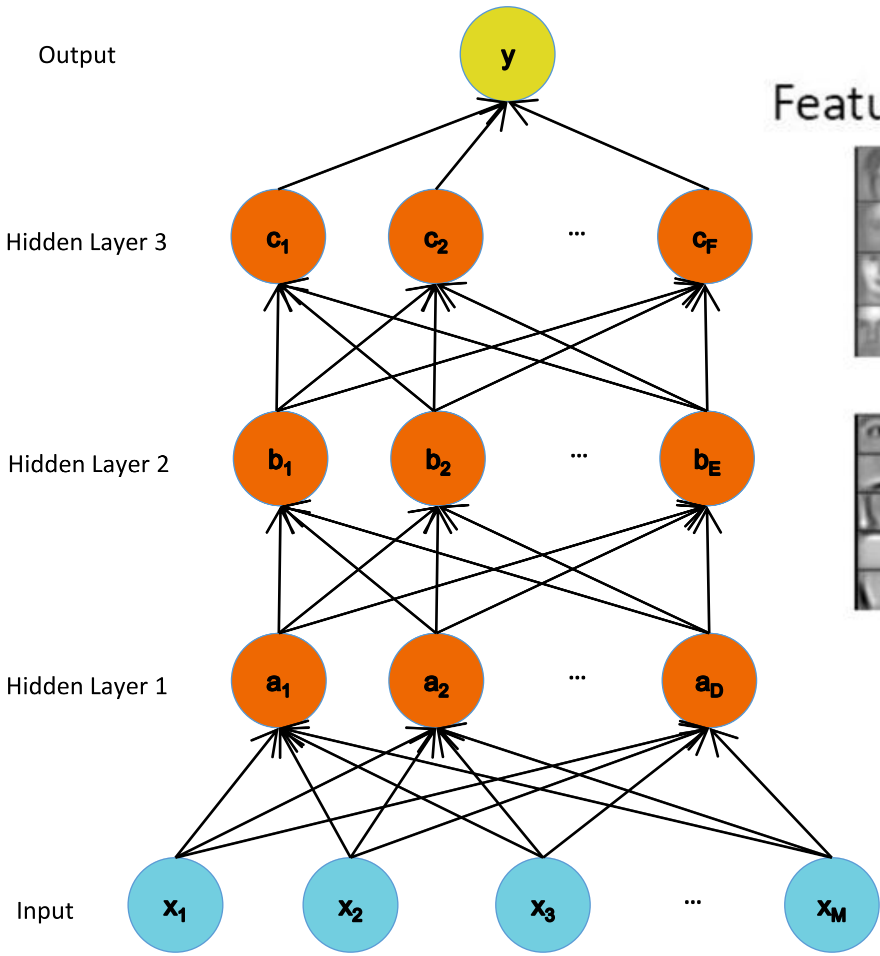
(take small steps opposite the gradient)

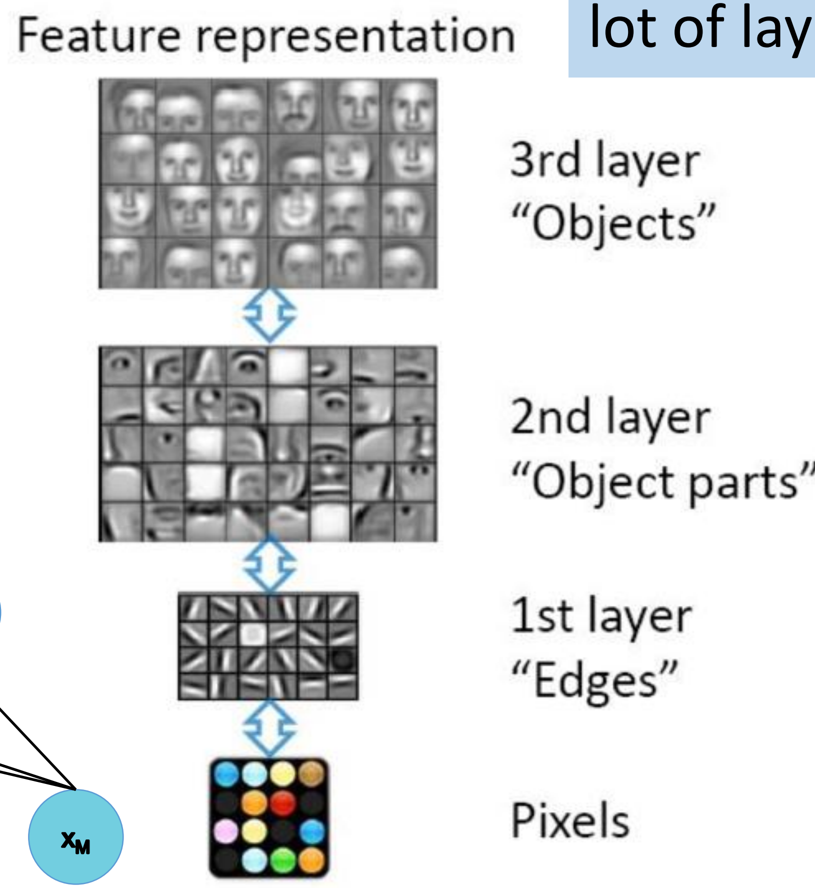
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Deep Neural Networks

Representation Learning

Deep learning is way more than having neural networks with a lot of layers



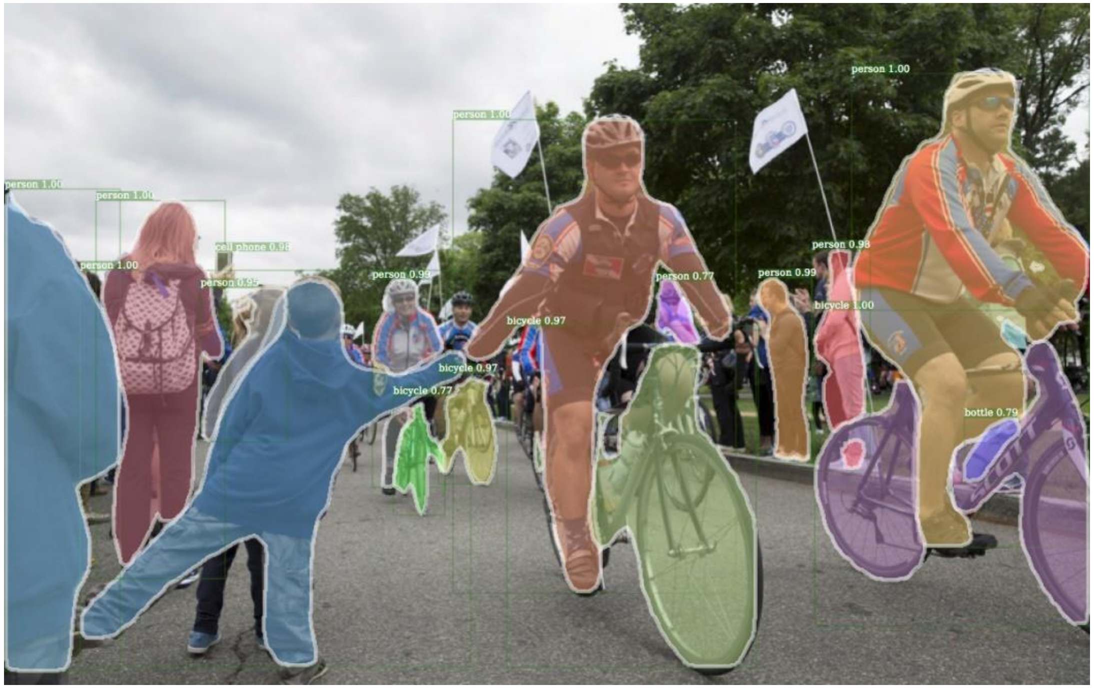


Example from Honglak Lee (NIPS 2010)

Convolutional Neural Networks

Introduction

Convolutional Neural Networks



Dense Vector Multiplication

Processing images: the dense way

32x32x3 image

Reshape it into a vector

An input-sized weight vector for each hidden neuron

100x3072

$\boldsymbol{x}$

3072

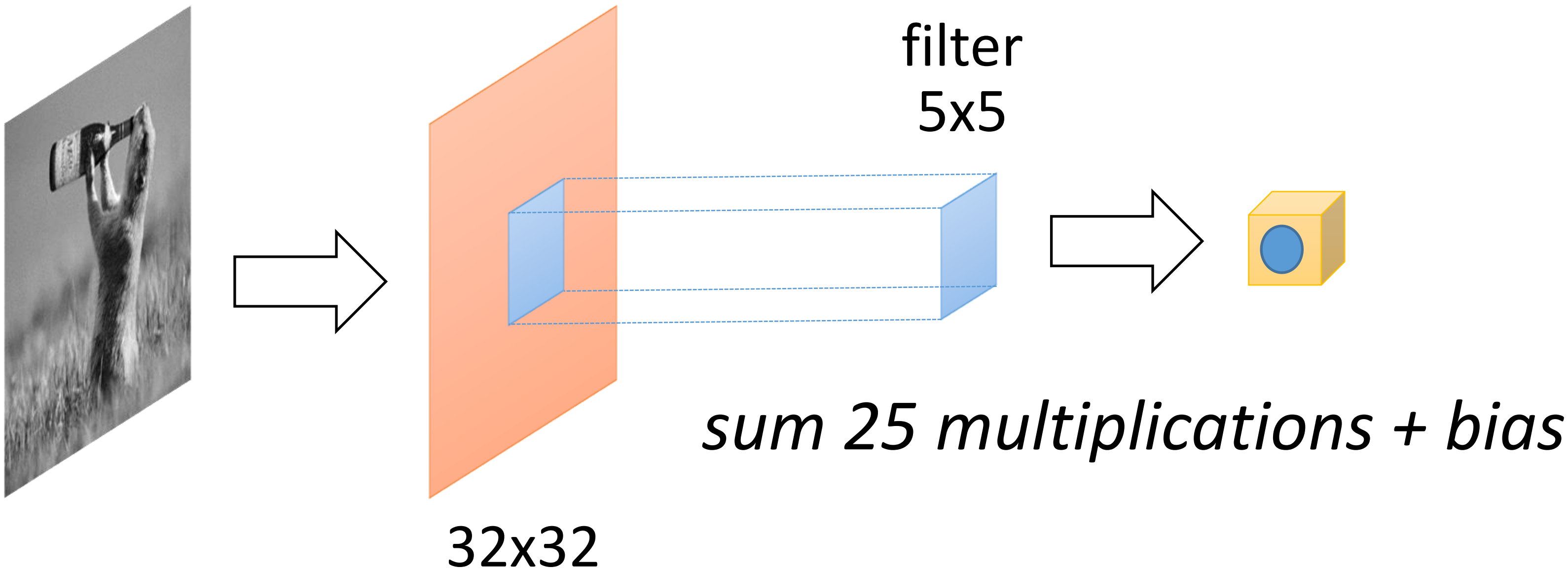
$\boldsymbol{W}$

$\boldsymbol{W}\boldsymbol{x}^{T}$

Each element contains the activation of 1 neuron

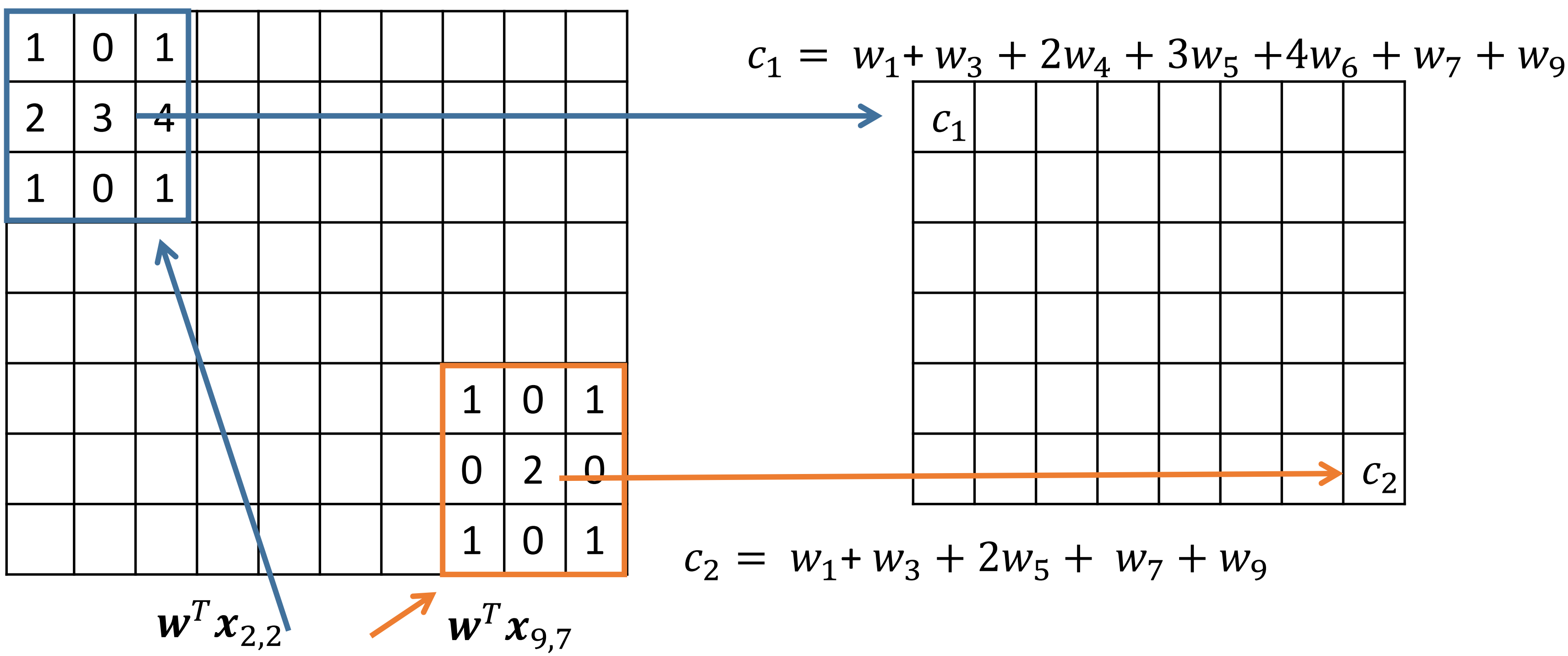
100

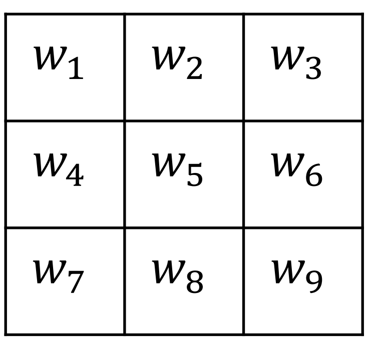
Convolution Operator



Matrix input preserving
spatial structure

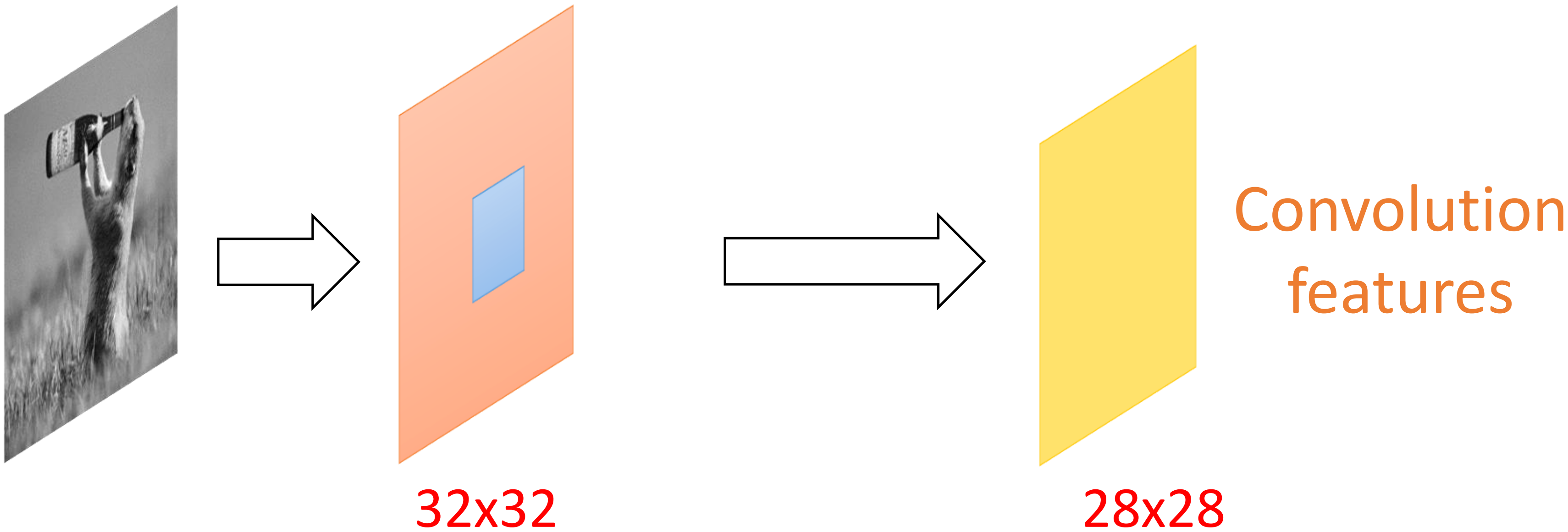
Adaptive Convolution





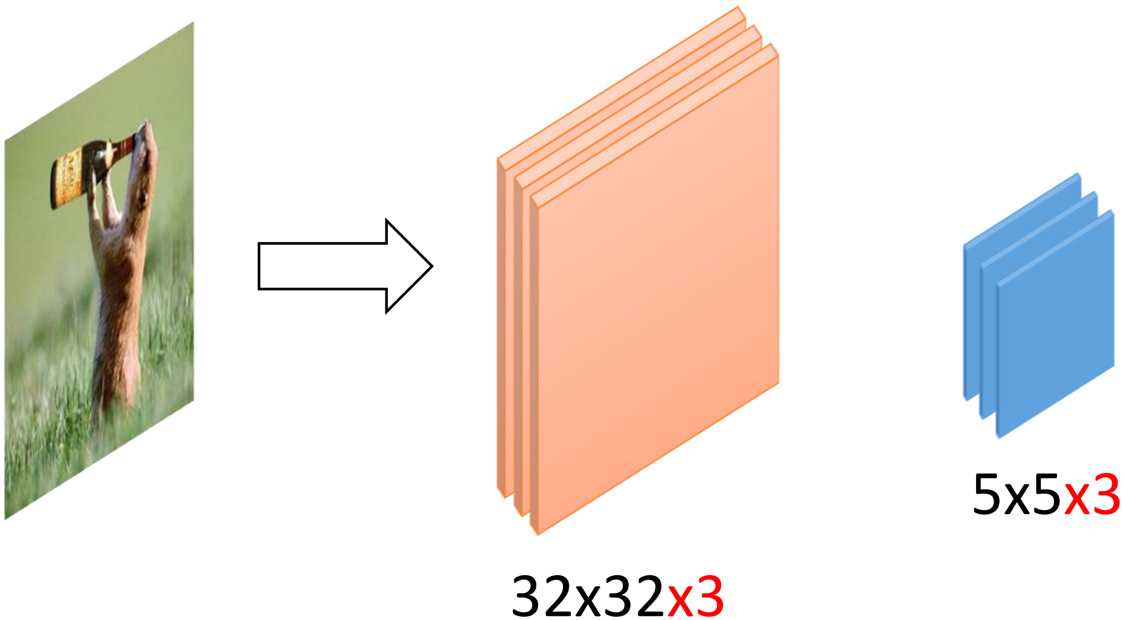
Convolutional filter (kernel) with (adaptive) weights w_i

Convolutional Features



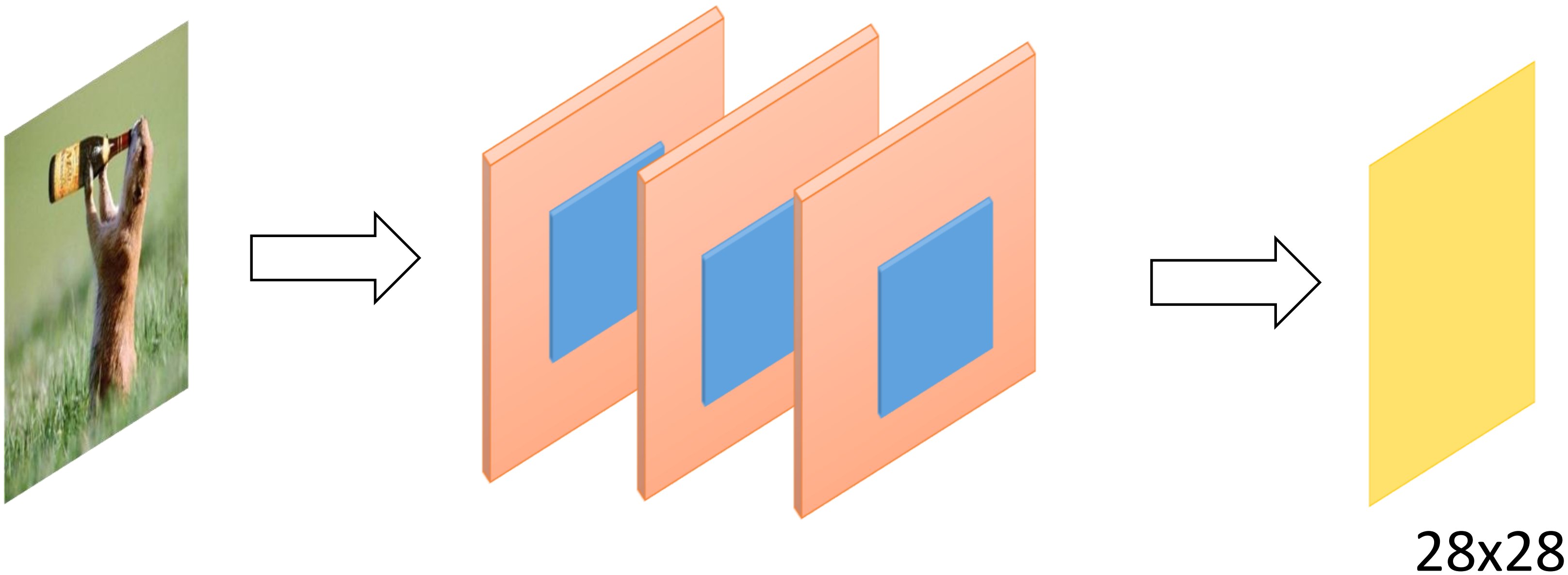
Slide the filter on the image computing elementwise products and summing up

Multi-Channel Convolution



Convolution filter has a number of slices equal to the number of image channels

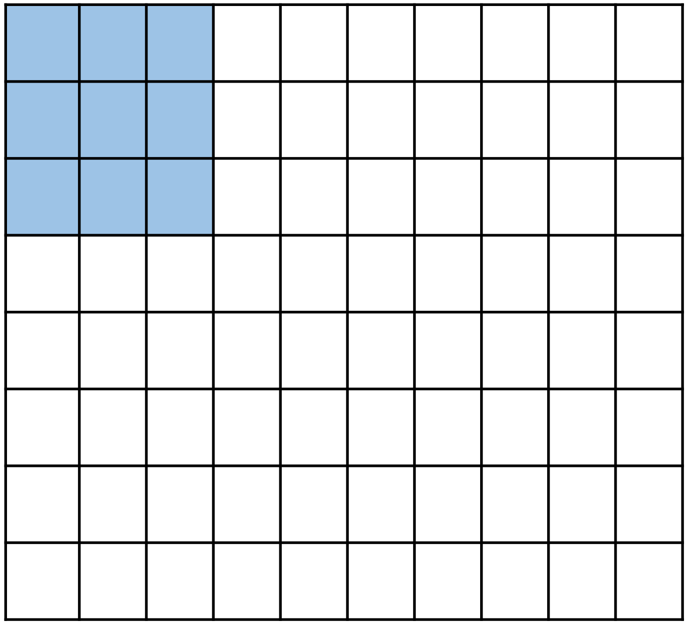
Multi-Channel Convolution



All channels are typically convolved together

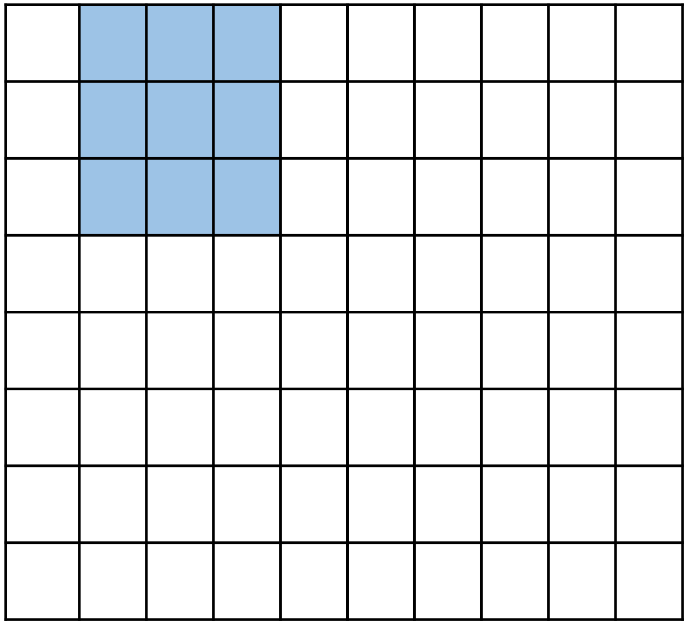
- They are summed-up in the convolution
- The convolution map stays bi-dimensional

Stride



- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1

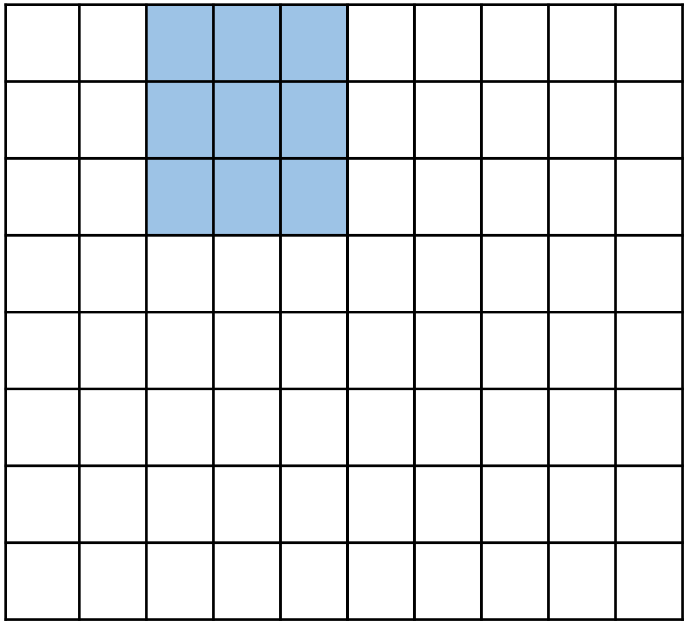
Stride



stride = 1

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1

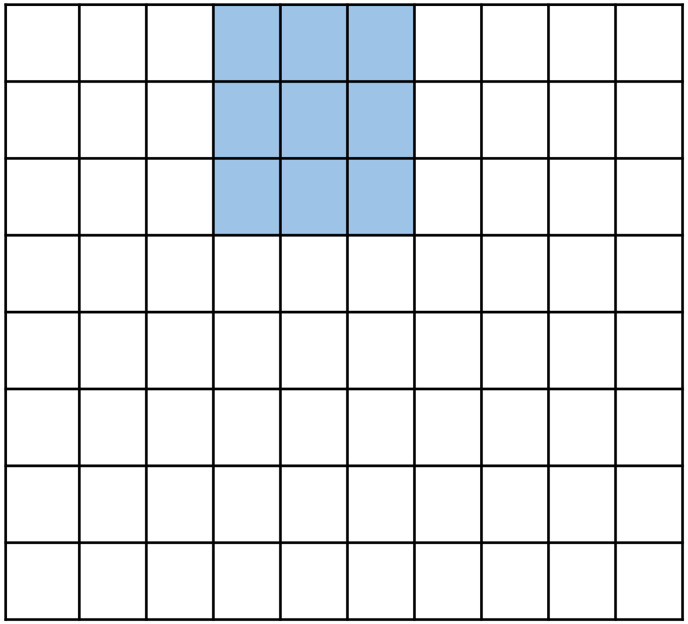
Stride



stride = 1

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1

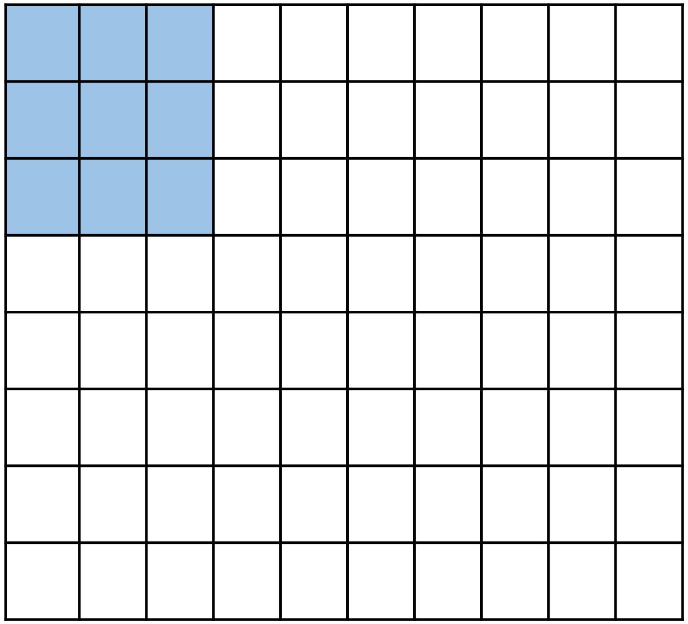
Stride



stride = 1

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1

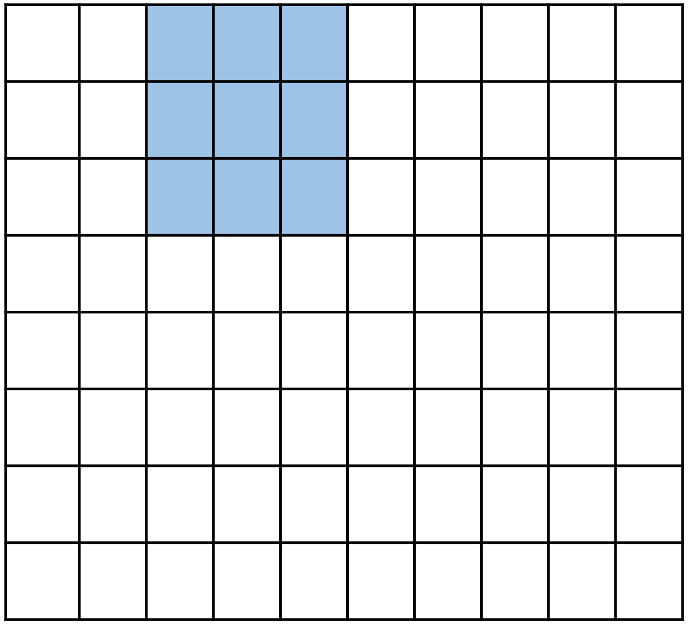
Stride



stride = 2

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter

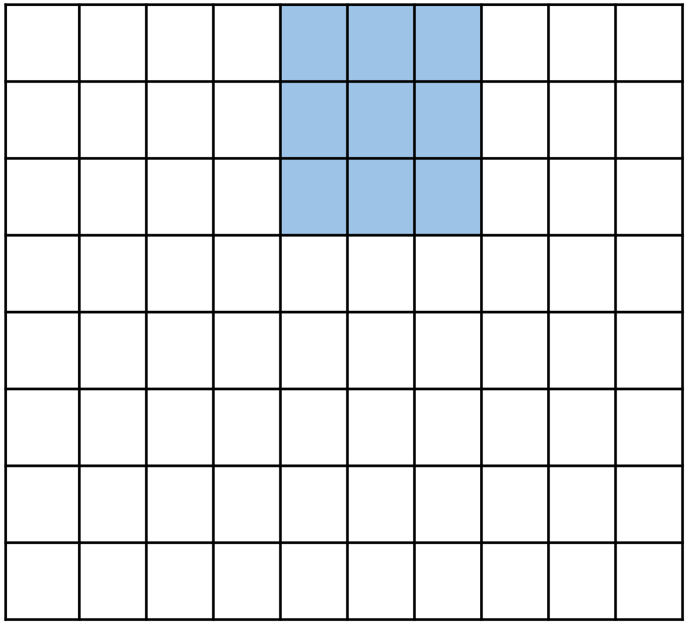
Stride



stride = 2

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter

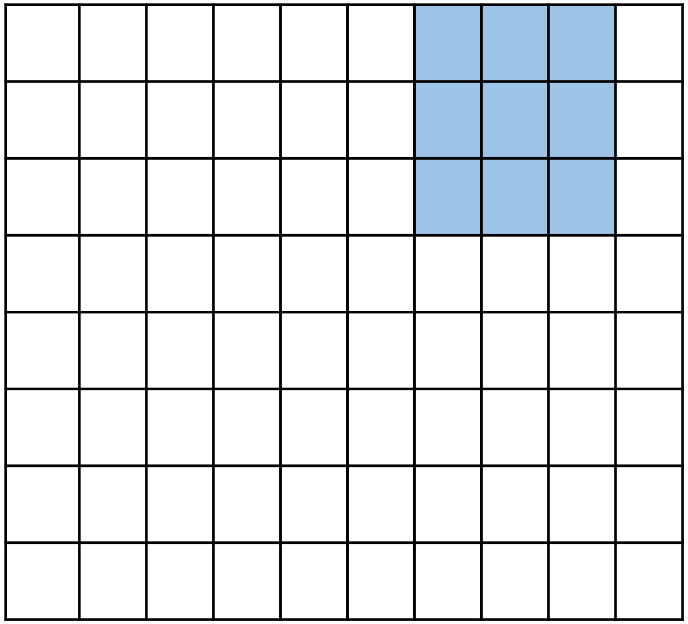
Stride



stride = 2

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter

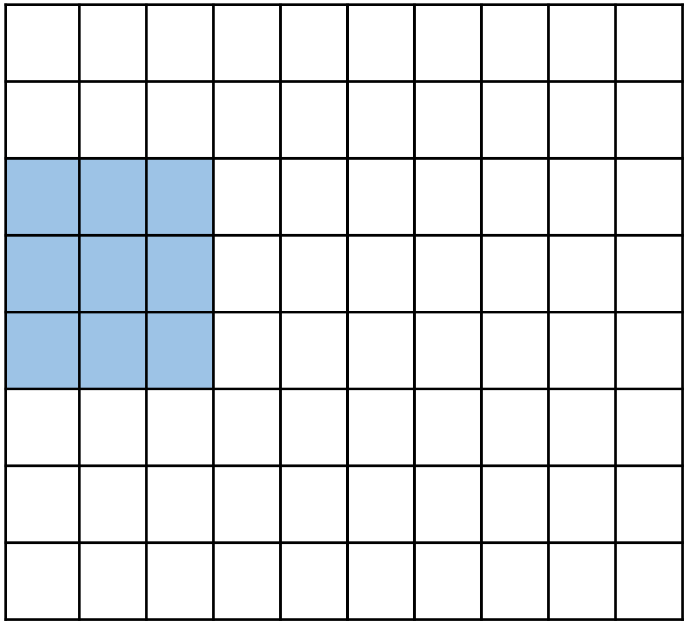
Stride



stride = 2

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter

Stride



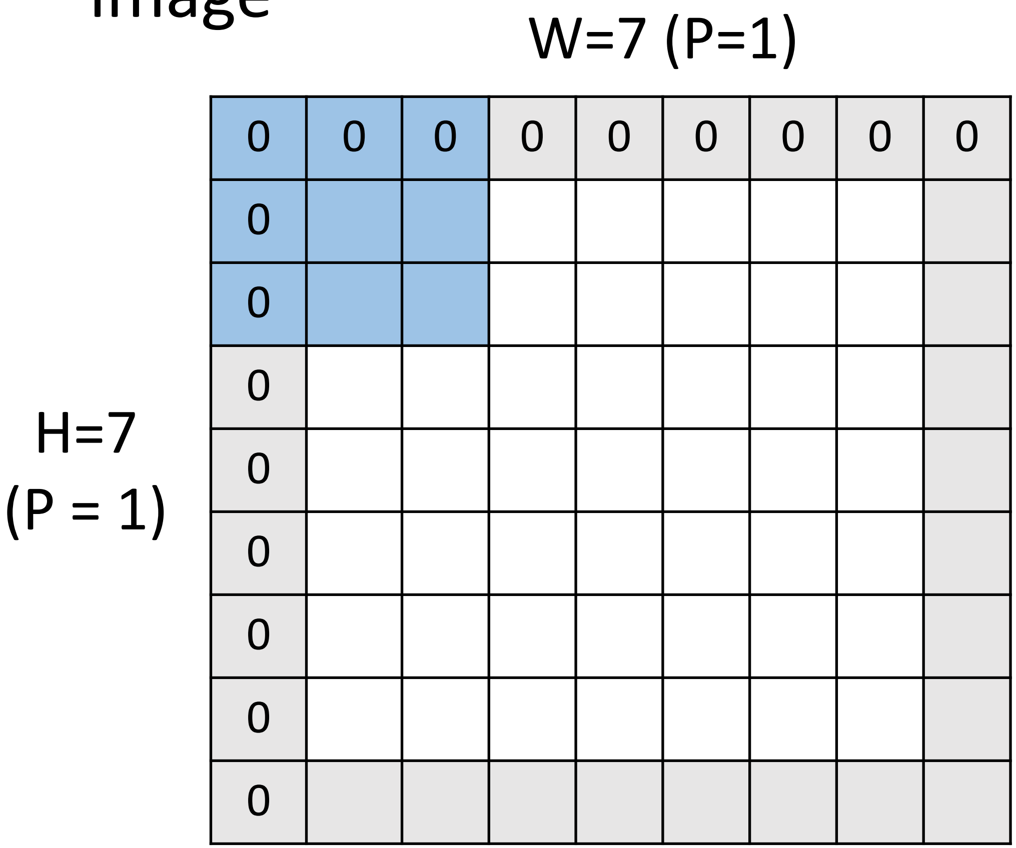
stride = 2

Works in both directions!

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter
- Stride reduces the number of multiplications
  - Subsamples the image

Zero Padding

Add columns and rows of zeros to the border of the image

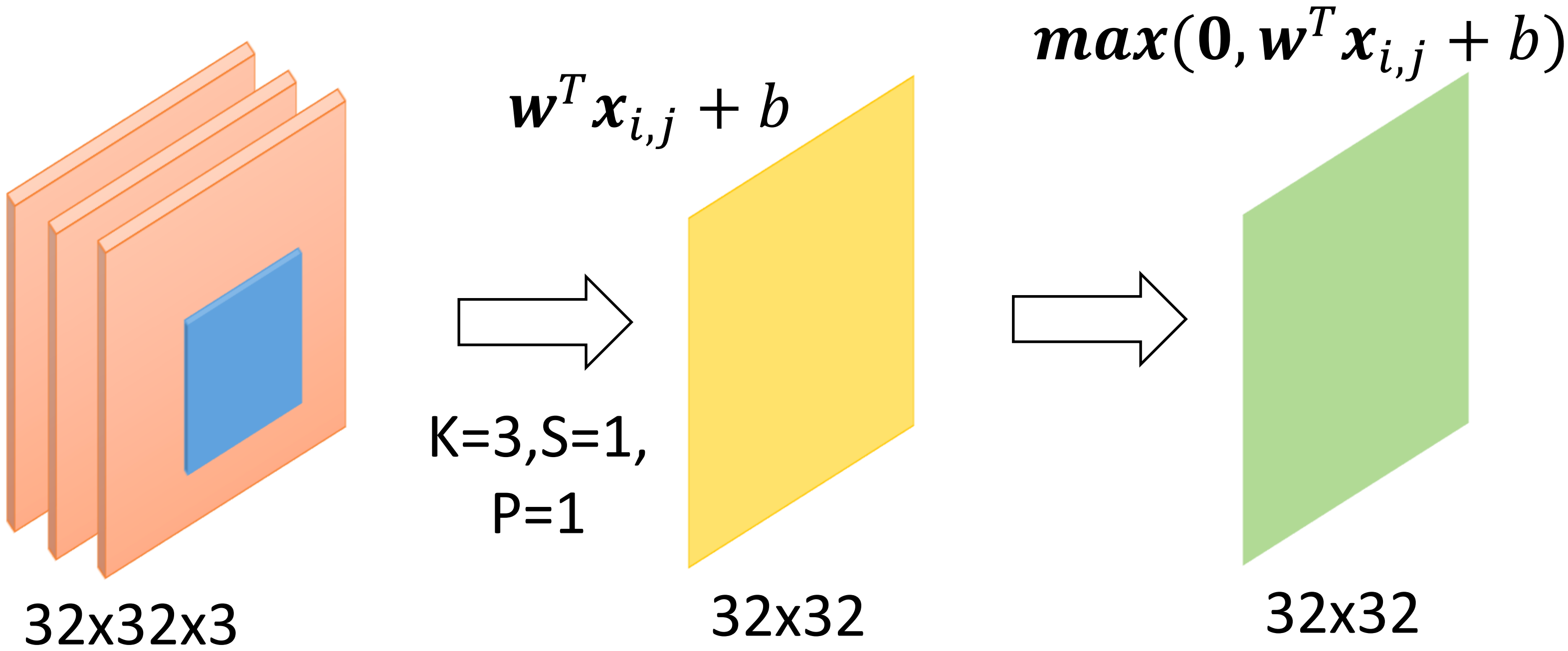


Zero padding serves to retain the original size of image

$$P = \frac{K-1}{2}$$

Pad as necessary to perform convolutions with a given stride S

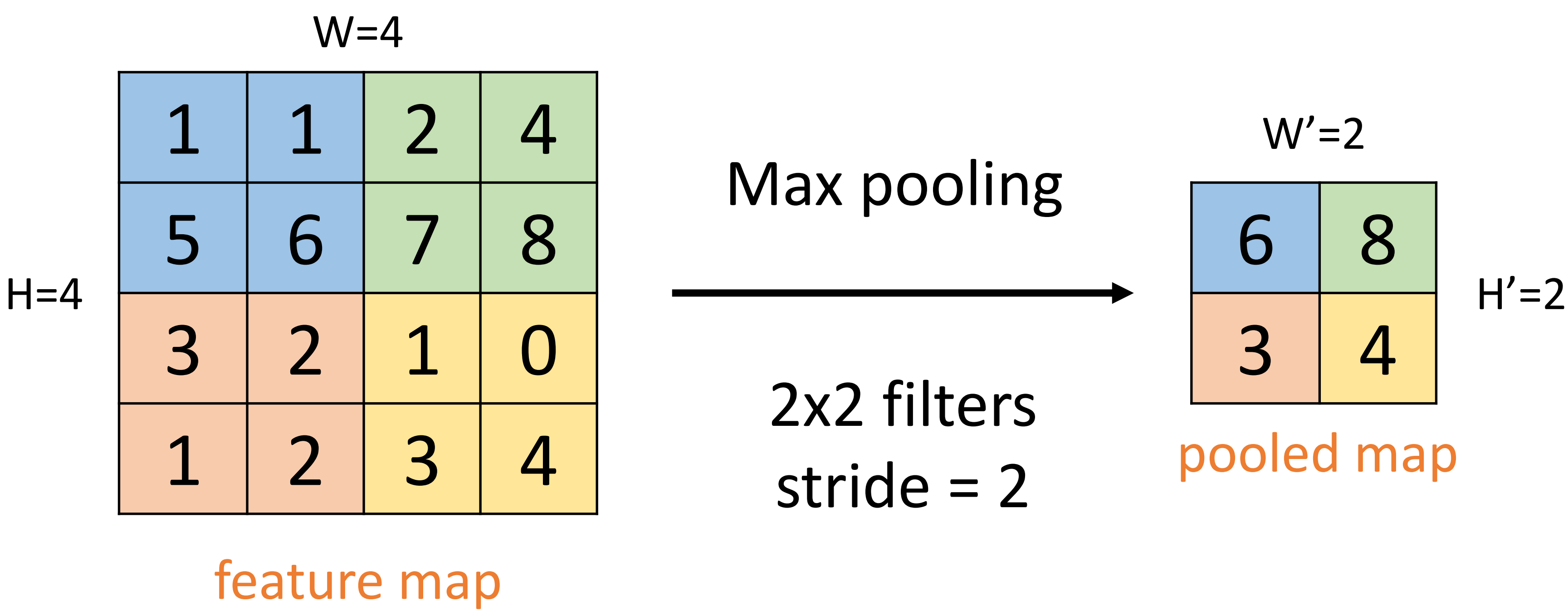
Feature Map Transformation



- Convolution is a linear operator
- Apply an element-wise nonlinearity to obtain a transformed feature map

Pooling

- Operates on the feature map to make the representation
  - Smaller (subsampling)
  - Robust to (some) transformations



Specifying CNN in Code (Keras)

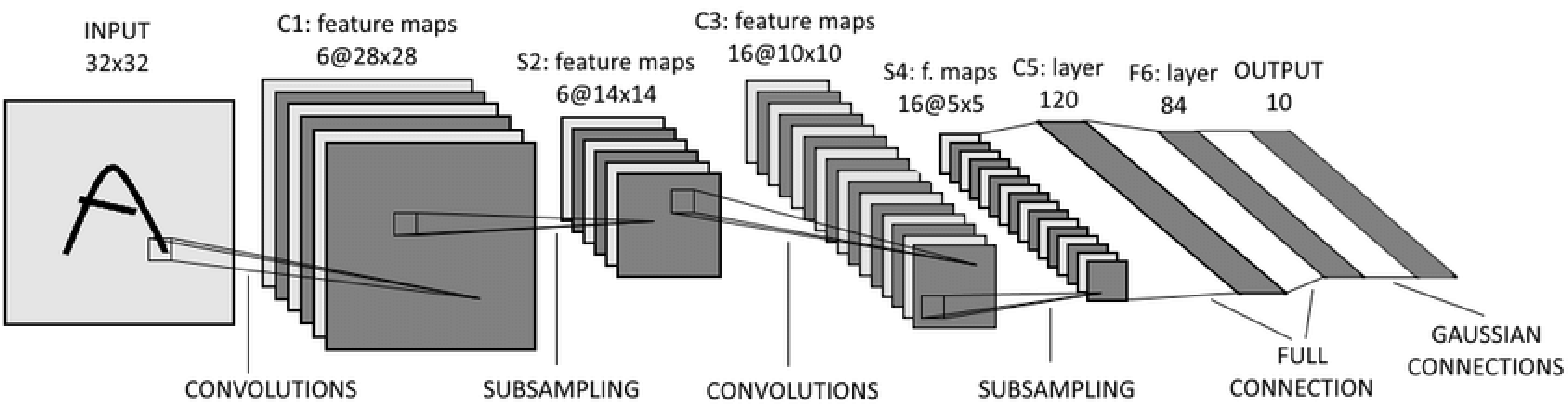
Number of convolution filters D_k

Define input size (only first hidden layer)

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (5, 5)))
model.add(Activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

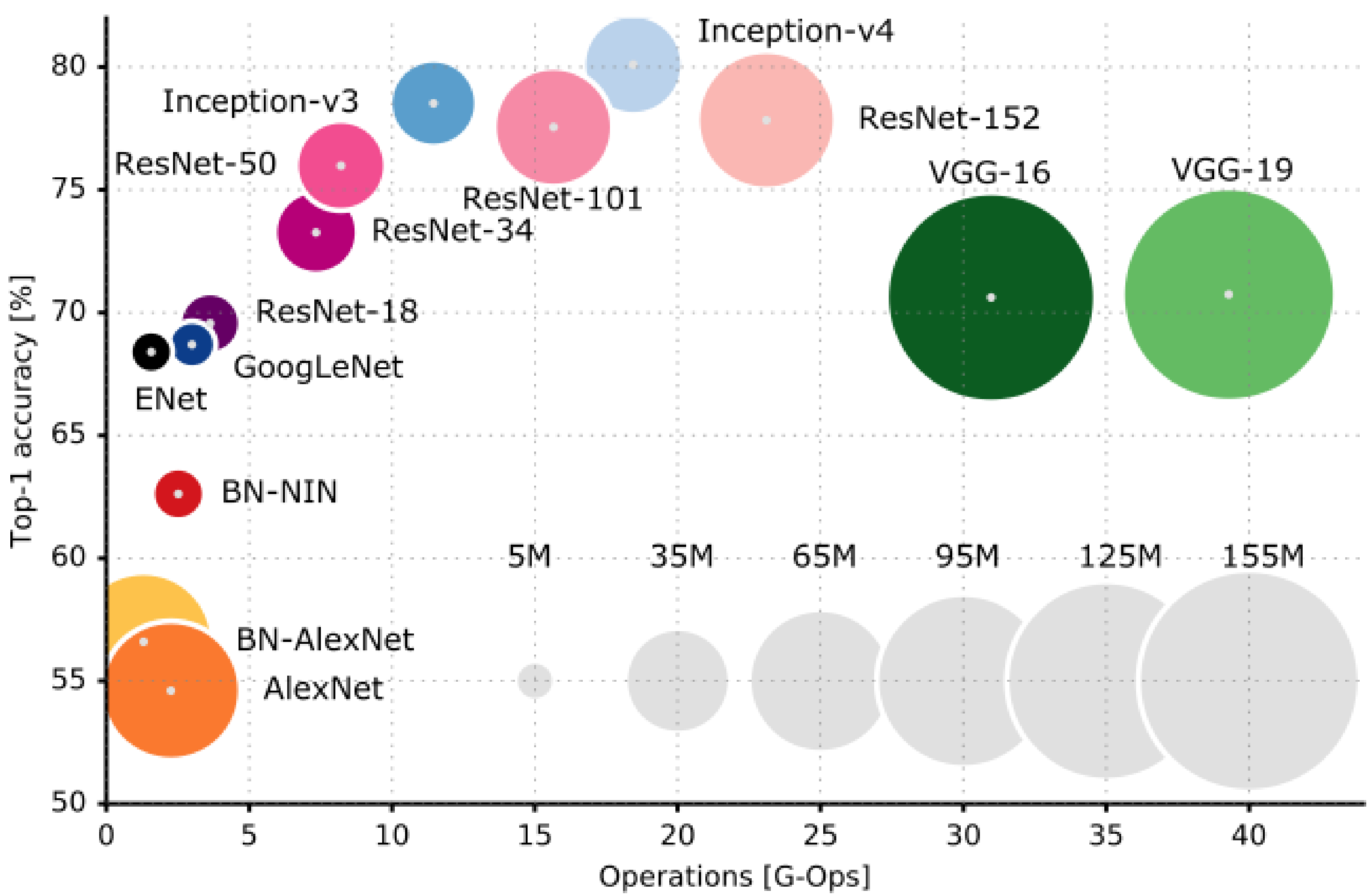
Does for you all the calculations to determine the final size to the dense layer (in most frameworks, you have to supply it)

LeNet-5 (1989)



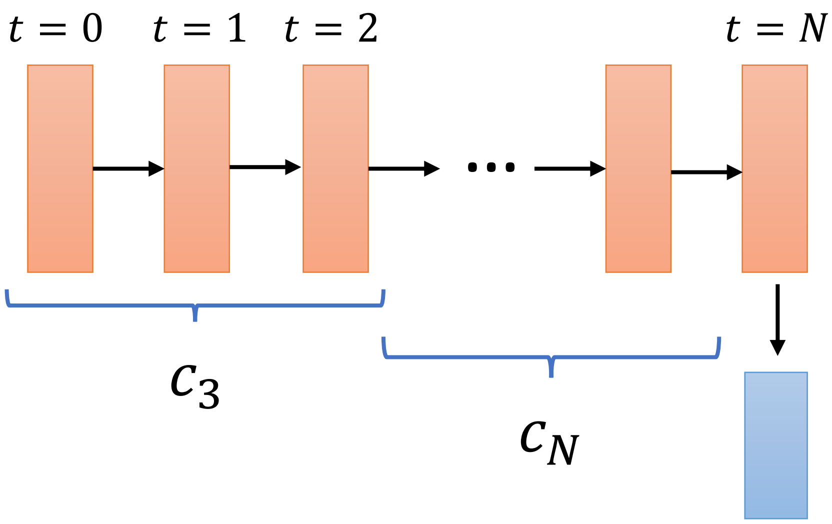
- Grayscale images
- Filters are 5x5 with stride 1 (sigmoid nonlinearity)
- Pooling is 2x2 with stride 2
- No zero padding

CNN Architecture Evolution



Recurrent Neural Networks

Dealing with Sequences in NN



Variable size data describing sequentially dependent information

Neural models need to capture dynamic context c_t to perform predictions

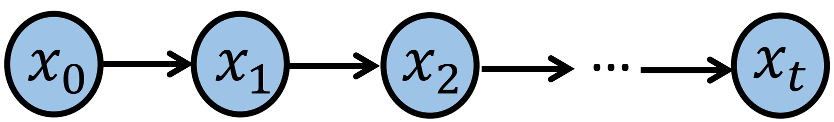
- Recurrent Neural Network
  - Fully adaptive (Elman, SRN, …)
  - Randomized approaches (Reservoir Computing)
  - Gated recurrent networks

Unfolding RNN (Forward Pass)

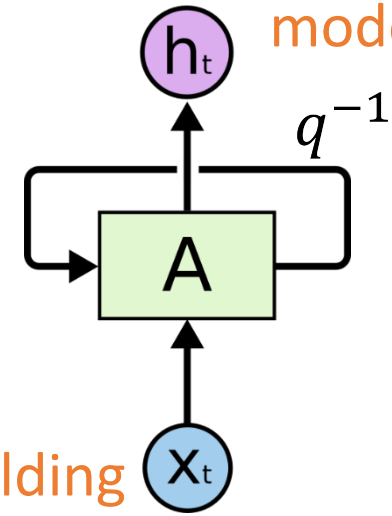
Graphics credit @ colah.github.io

By now you should be familiar with the concept of model unfolding/unrolling on the data

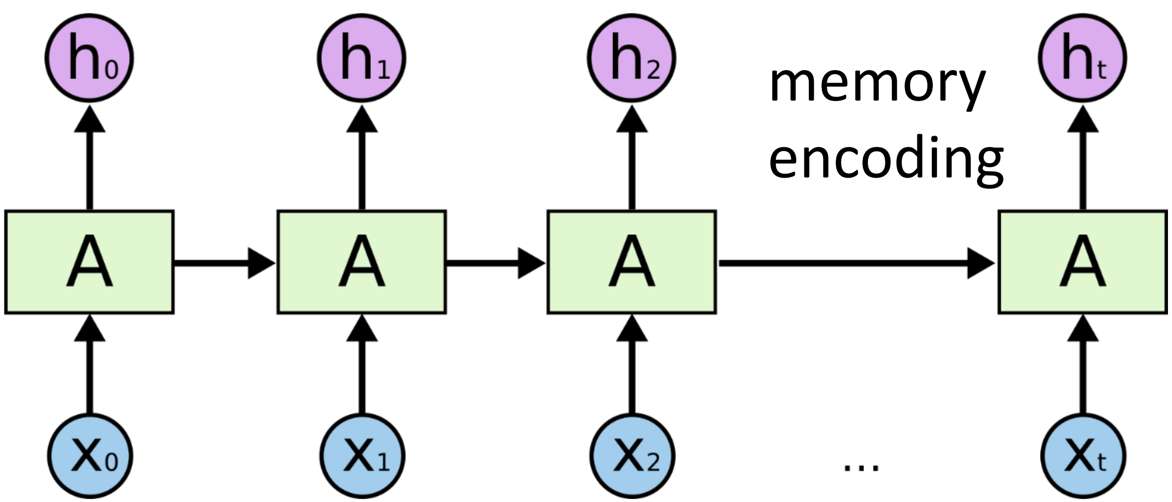
data



model

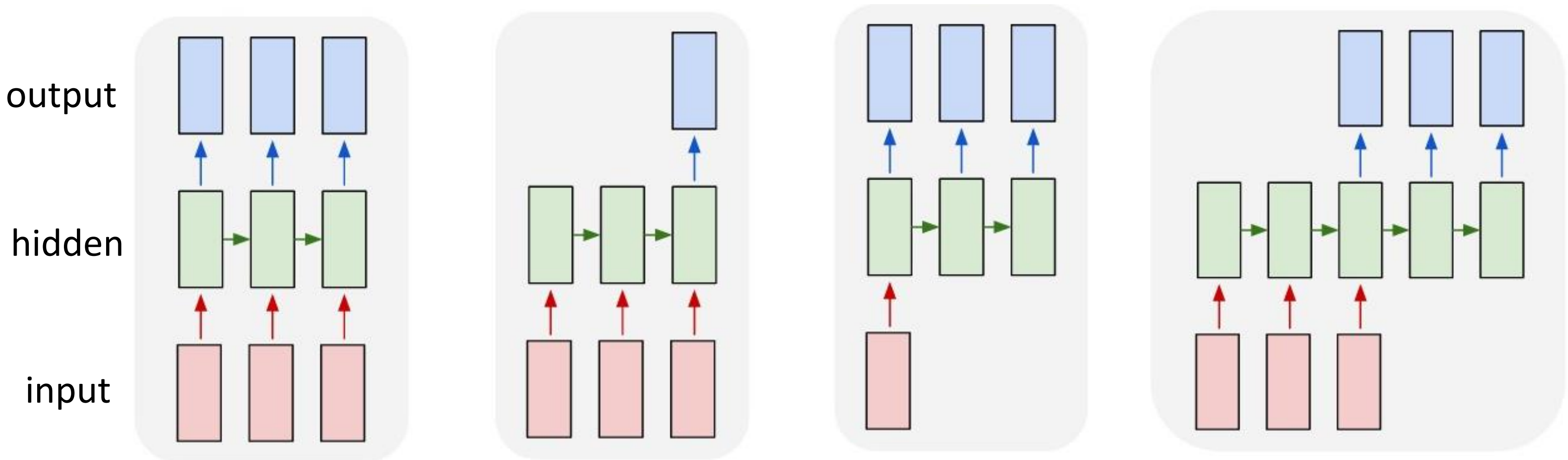


unfolding



Map an arbitrary length sequence $x_0..x_t$ to fixed-length encoding $\boldsymbol{h}_t$

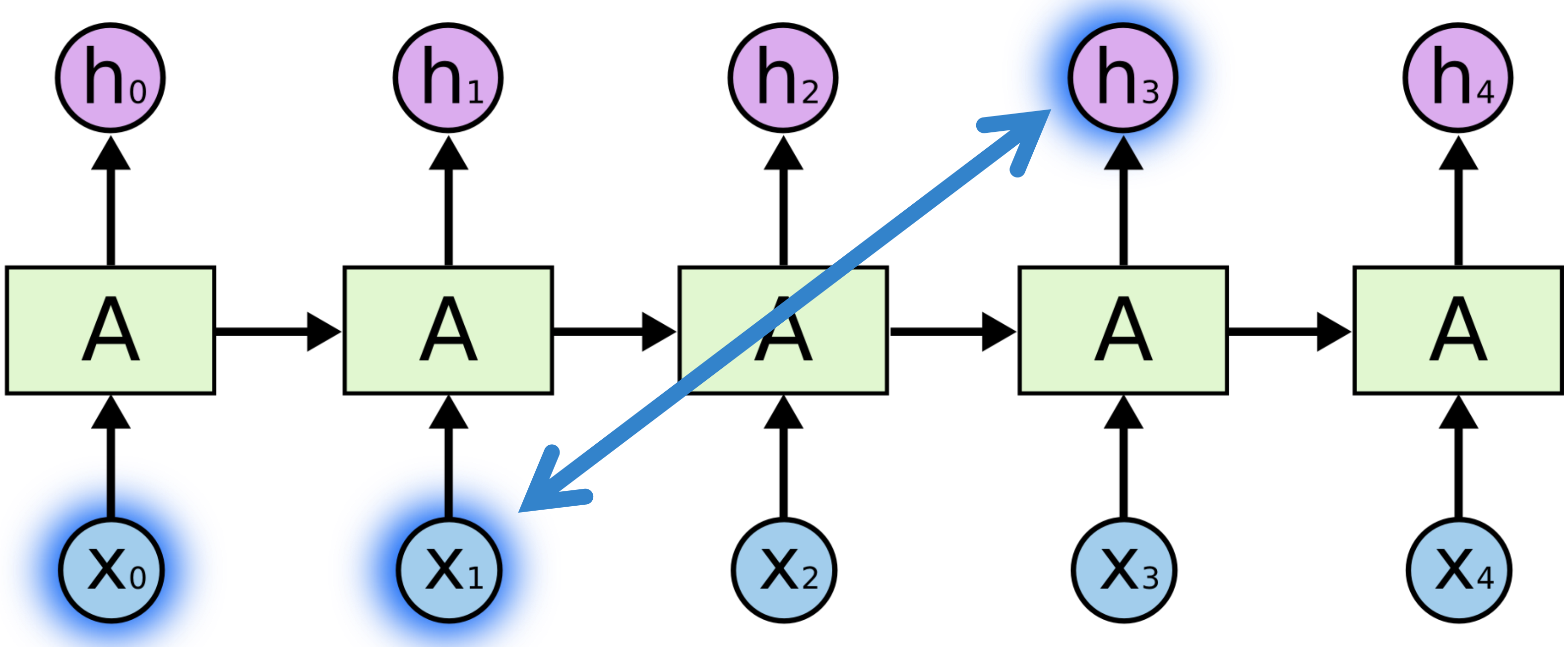
Supervised Recurrent Tasks



element to element sequence to item item to sequence sequence to sequence

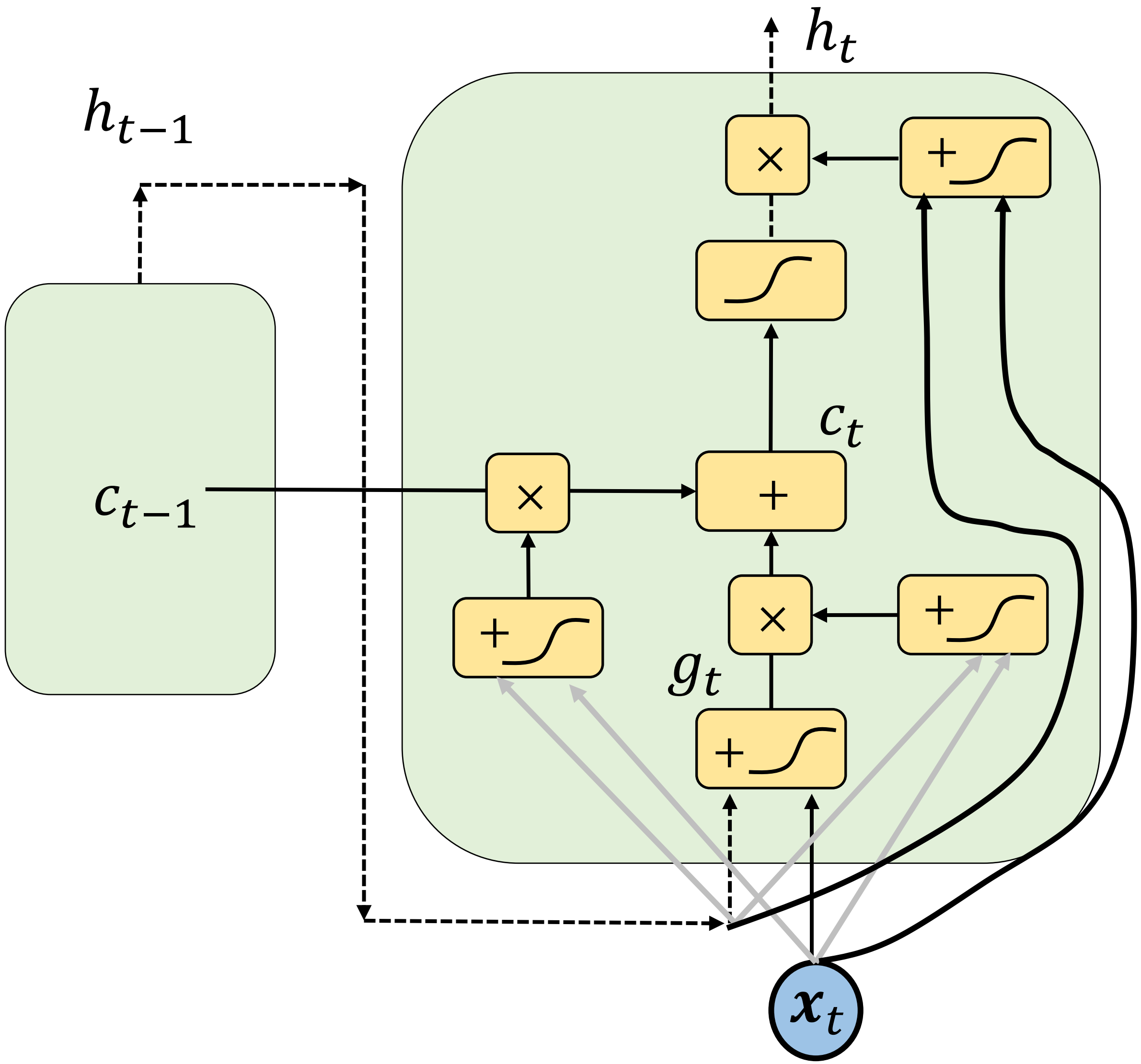
Graphics credit @ karpathy.github.io

Learning to Encode Input History



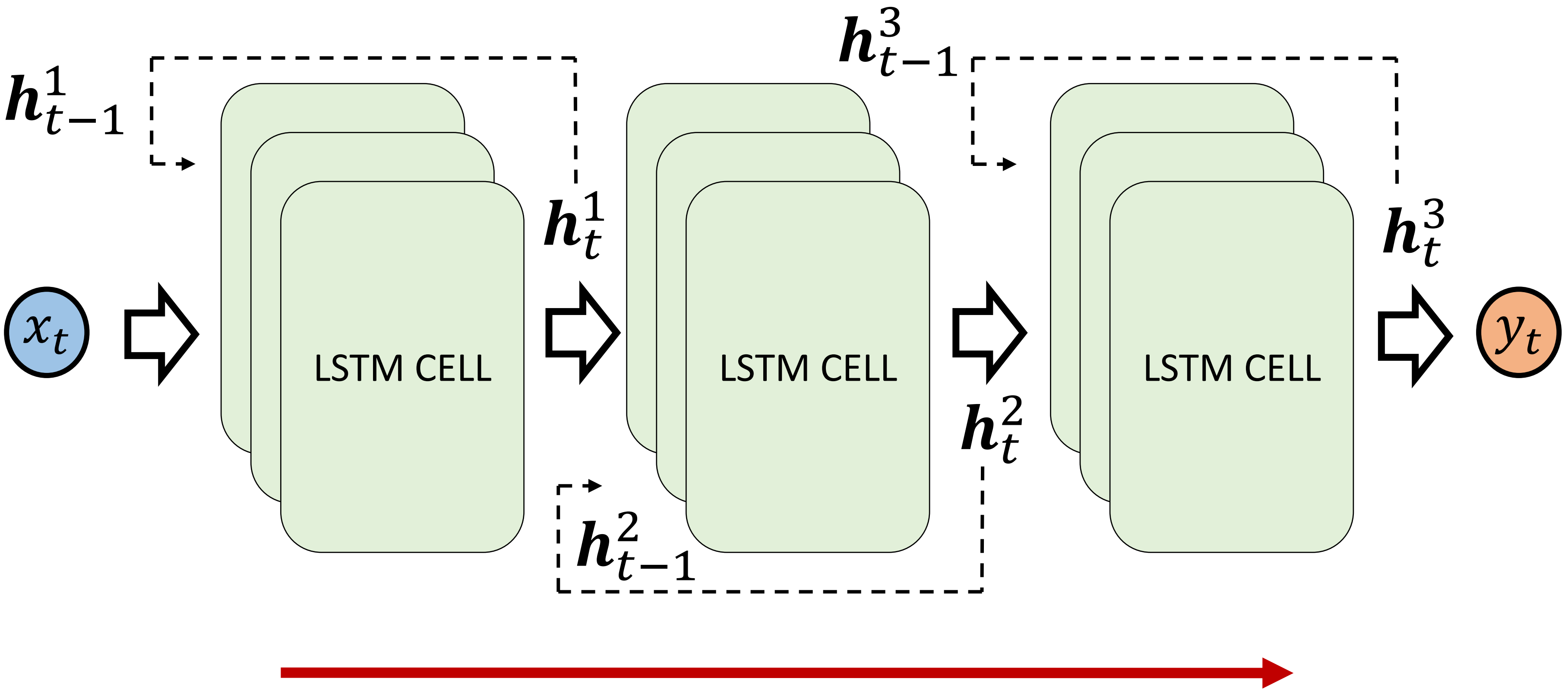
Hidden state $\boldsymbol{h}_t$ summarizes information on the history of the input signal up to time t

Long Short Term Memory – The Cell



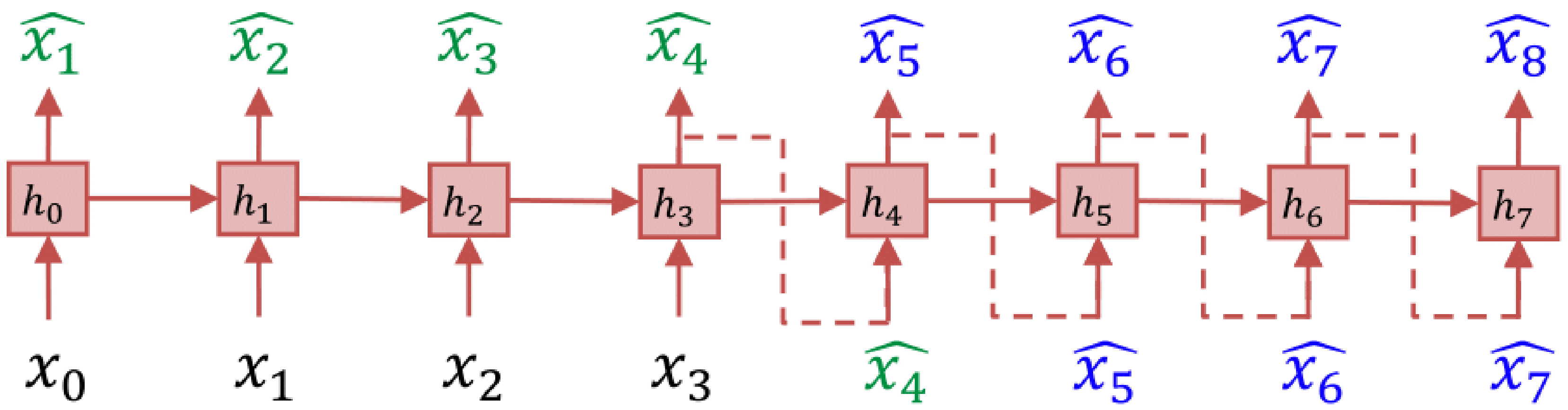
Using gates to control memory access

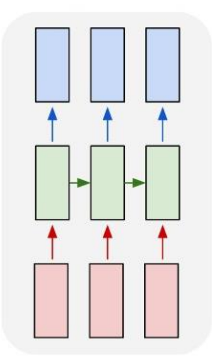
Deep LSTM



LSTM layers extract information at increasing levels of abstraction (enlarging context)

Predicting the future with RNNs





Element-to-element

: *RNN Cell*

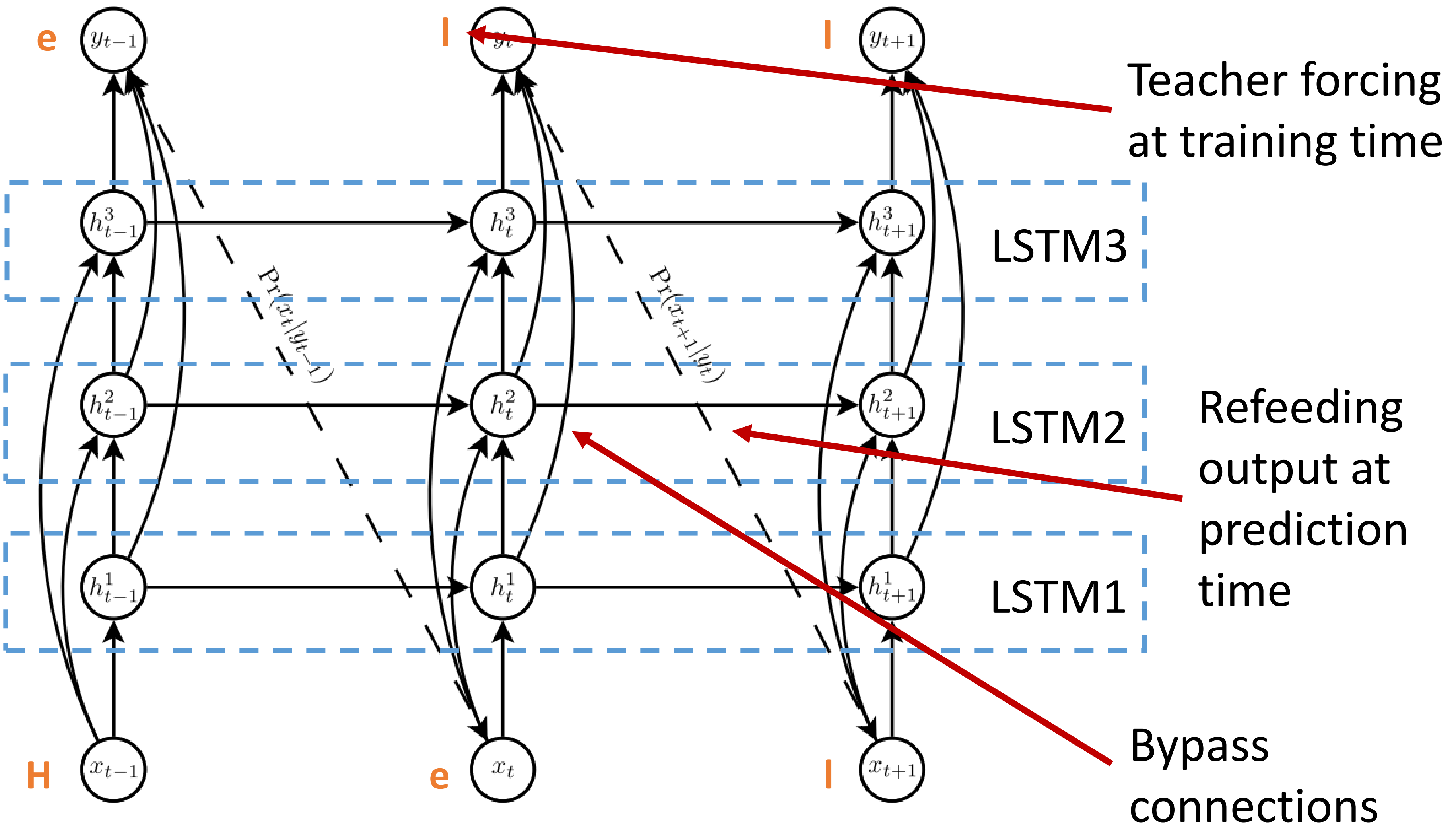
x_i: *ground-truth* $(0 \le i < 4)$

$\widehat{x_i}$: *1-step prediction* $(1 \le i < 5)$

$\widehat{x_i}$: *multi-step prediction* $(5 \le i < 9)$

h_i: *hidden state* $(0 \le i < 9)$

Generative Use of LSTM/GRU



A. Graves, Generating Sequences With Recurrent Neural Networks, 2013

Character Generation Fun

Shakespeare

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Character Generation Fun

Linux Kernel Code

```
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Generate Sad Jokes

A 3-LSTM layers neural network to generate English jokes character by character

Why did the boy stop his homework?
Because they're bunny boo!

What do you get if you cross a famous California little boy with an elephant for players?
Market holes.

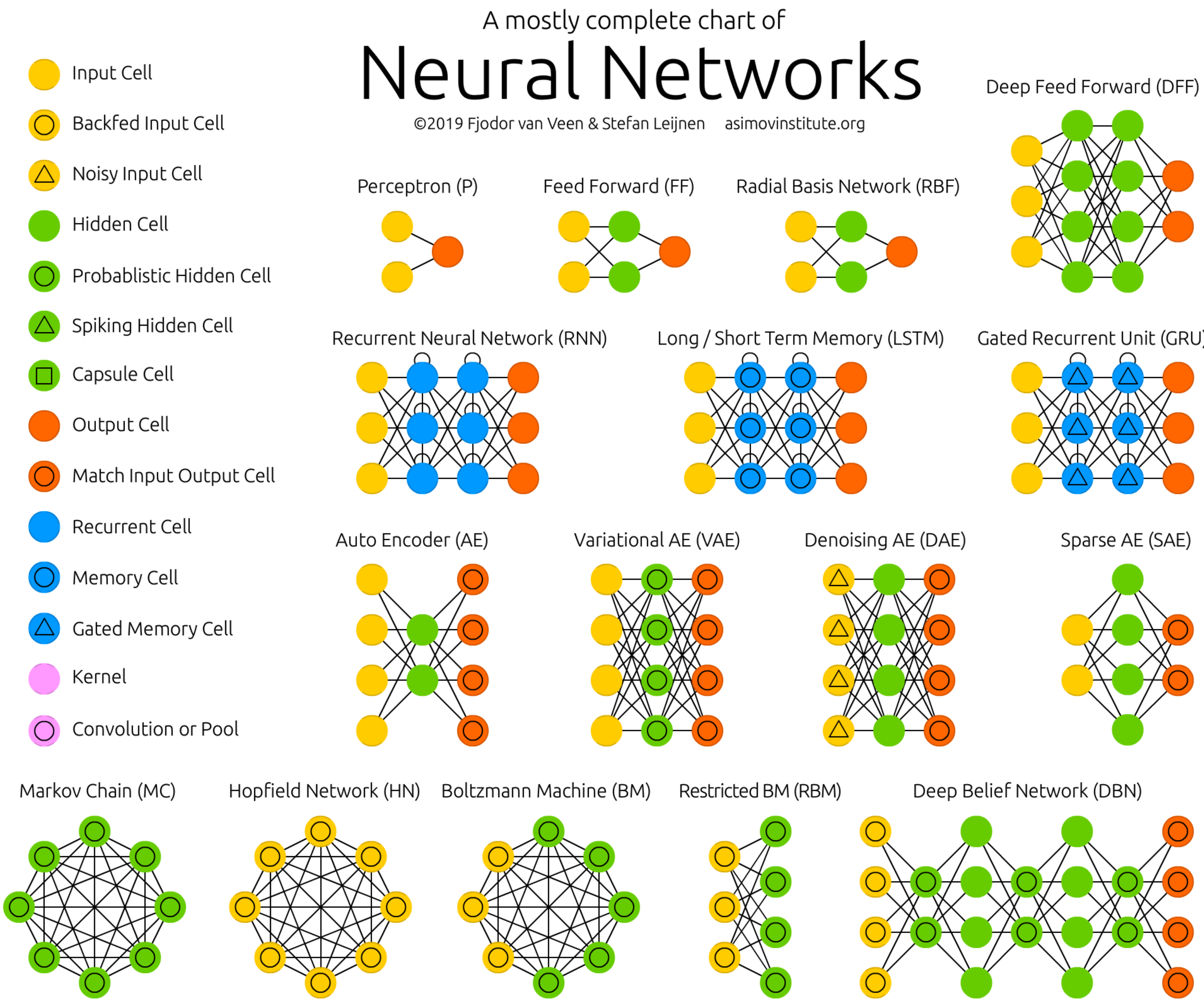
Q: Why did the death penis learn string?
A: Because he wanted to have some roasts case!

Wrap-Up

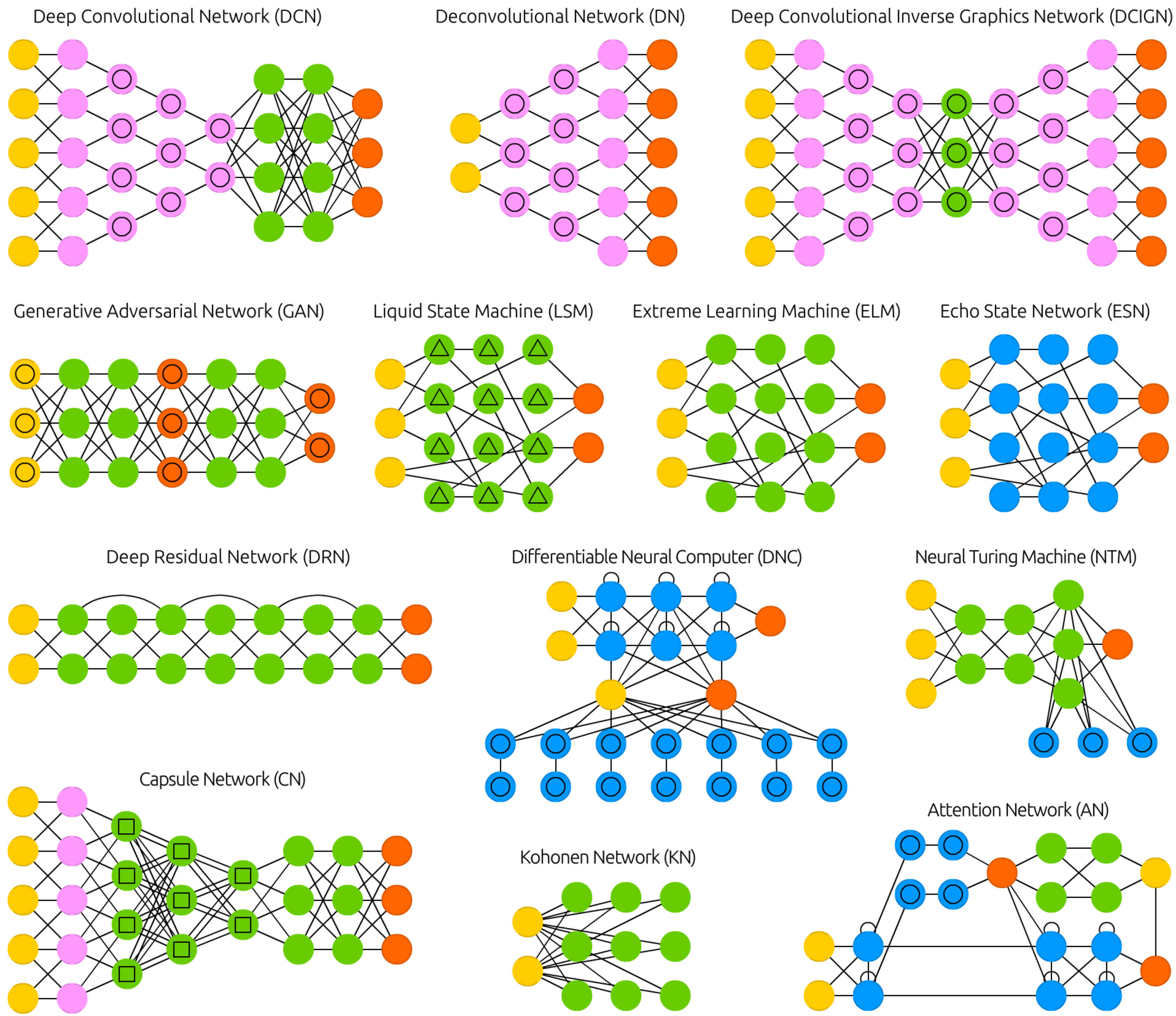
Things to Remember

- Vectorial data: feedforward neural networks
- Image data: convolutional neural networks
- Sequential data: recurrent neural networks
- Need to chose:
  - Activation and loss functions
  - Optimization algorithms
- Model selection
  - Train-valid-test
  - Data preprocessing
  - Regularization

Part A


A mostly complete chart of
Neural Networks
©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org
Input Cell
Backfed Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Capsule Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)

Part B



Actually, this is largely a subset of the existing architectures

OMG Do I have to understand how to code all this?

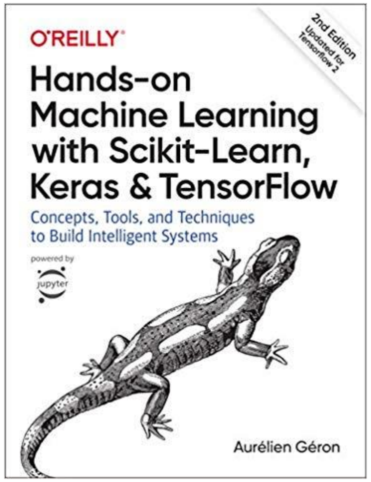
Luckily for you no…

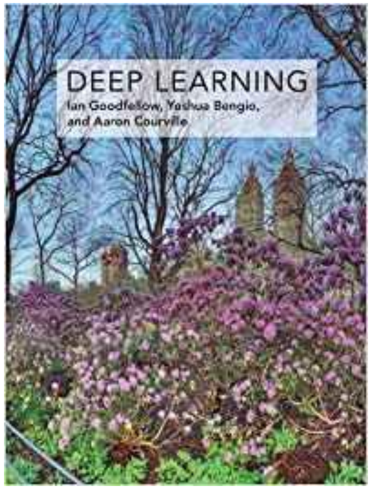
Keras

References

A practical handbook to start wrestling with Machine Learning models (2nd ed)

- 1st edition content is outdated on the NN part!





The reference book for deep learning models

- Also freely available online