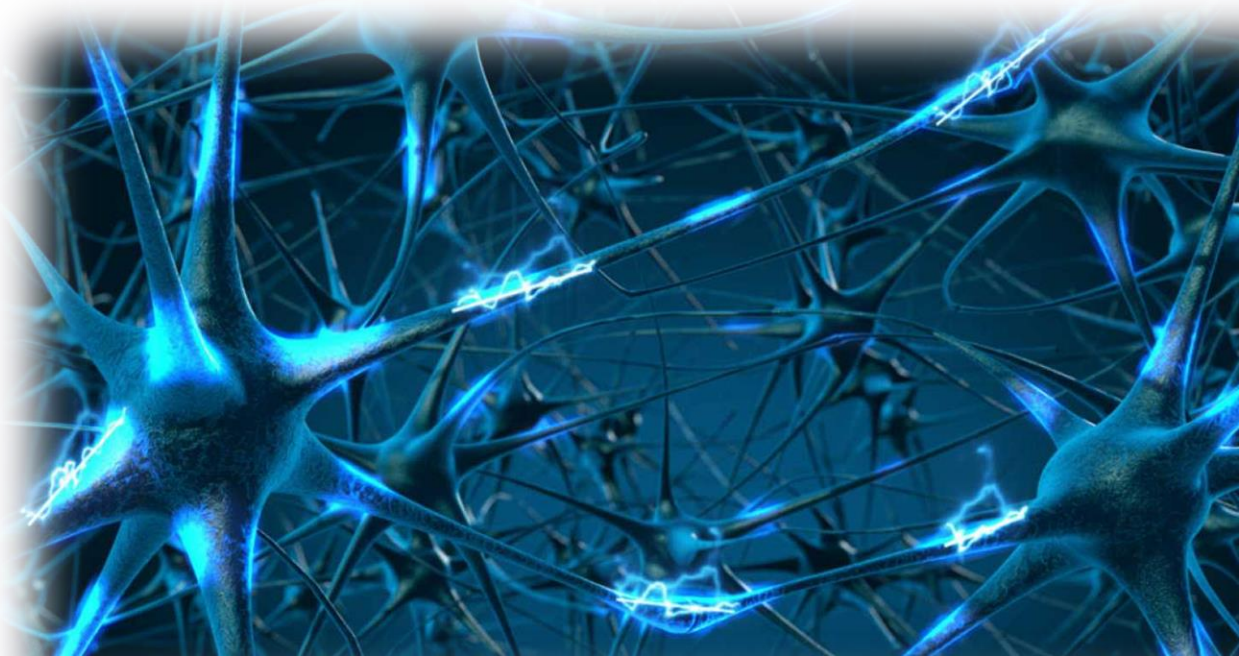
INTELLIGENZA ARTIFICIALE

INTRODUZIONE ALLE RETI NEURALI

Corsi di Laurea in Informatica, Ing. Gestionale, Ing. Informatica, Ing. di Internet

(a.a. 2022-2023)

Roberto Basili



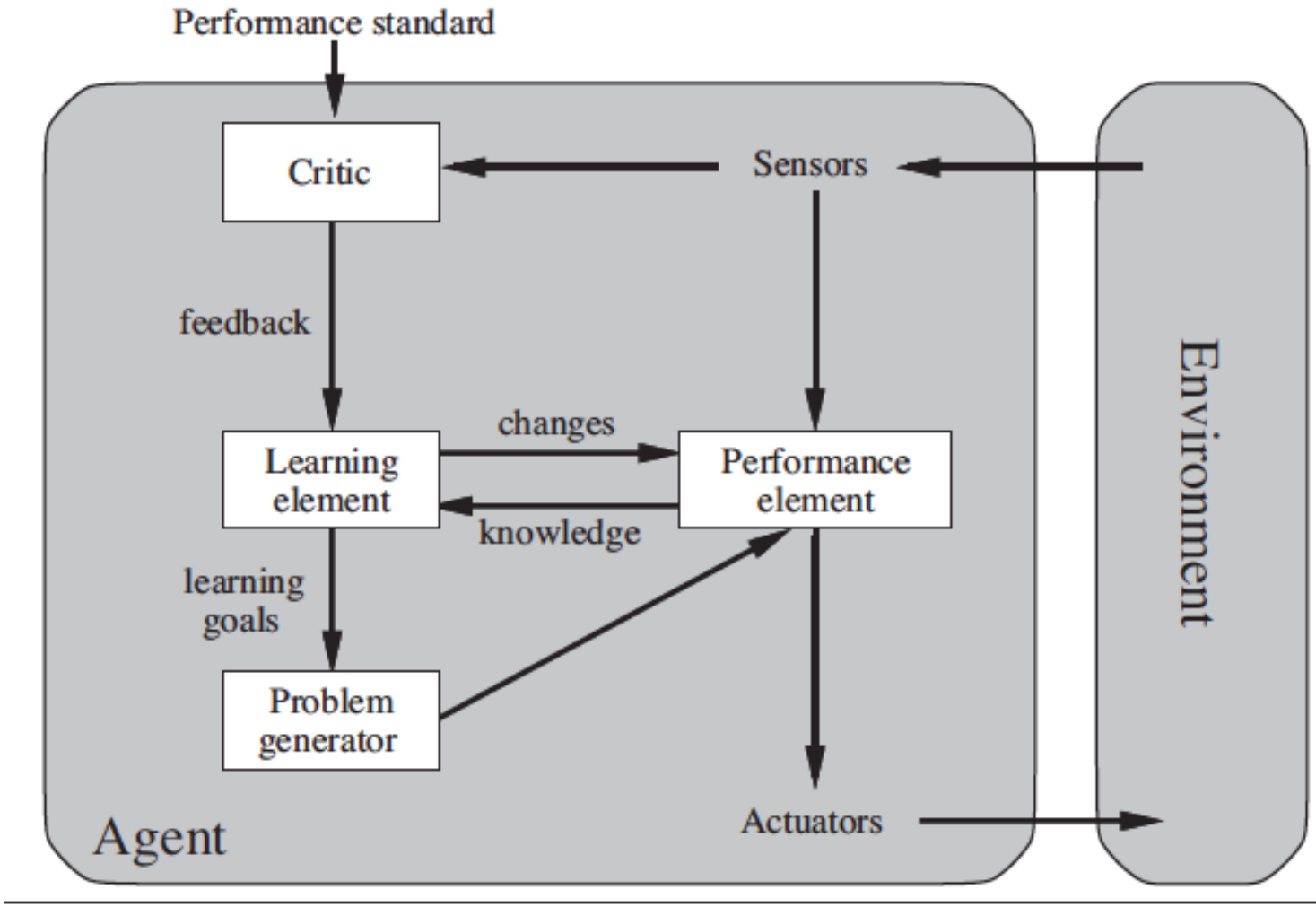
Overview (AIMA chpt. 18.6.1,18.6.3-4, 18.7)

- Recap
  - Agents & machine learning
  - Learning from examples:
    - The process of model selection
- Artificial Neural Networks
  - The linear regression problem: potential and limitations
  - Multilayer Perceptrons
  - Training MLPs
- NN Applications: image recognition
- NN Applicatuions: NLP
- Advanced Architectures

Introduction to machine learning

- Introduction to machine learning, recap
  - Task definition, Learning methodology, Learning issues: representing hypothesis
  - The Model Selection process
  - Learning paradigms
    - Supervised learning
    - Unsupervised learning
    - Reinforcement learning

AIMA learning architecture



Machine learning: definition

- *A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E* [Mitchell]
- Problem definition for a learning agent
  - Task T
  - Performance measure P
  - Experience E

Designing a learning system

1. Choosing the training experience
   - Examples of best moves, games outcome …
2. Choosing the target function
   - board-move, board-value, …
3. Choosing a representation for the target function
   - linear function with weights (hypothesis space)
4. Choosing a learning algorithm for approximating the target function
   - A method for parameter estimation

Inductive learning

- Simplest form: learn a function from examples

f is the target function

An example is a pair (*x*, *f(x)*)

Problem: find a hypothesis *h*
such that $h \approx f$
given a training set of examples

(This is a highly simplified model of real learning:
- Ignores prior knowledge
- Assumes examples are given)

Model Selection: cross validation

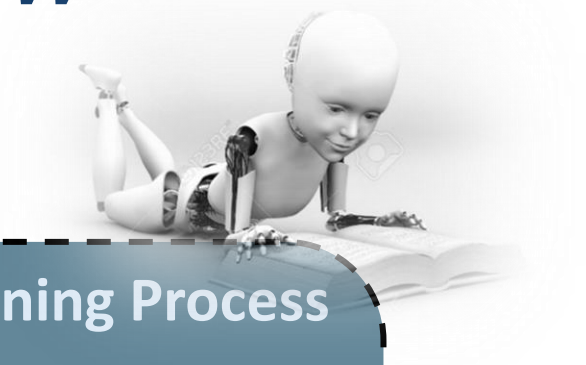
```
function MODEL-SELECTION(Learner, examples, k) returns a (hypothesis, error rate) pair

  err ← an array, indexed by size, storing validation-set error rates
  training_set, test_set ← a partition of examples into two sets
  for size = 1 to ∞ do
      err[size] ← CROSS-VALIDATION(Learner, size, training_set, k)
      if err is starting to increase significantly then
          best_size ← the value of size with minimum err[size]
          h ← Learner(best_size, training_set)
          return h, ERROR-RATE(h, test_set)

function CROSS-VALIDATION(Learner, size, examples, k) returns error rate

  N ← the number of examples
  errs ← 0
  for i = 1 to k do
     validation_set ← examples[(i − 1) × N/k:i × N/k]
     training_set ← examples − validation_set
     h ← Learner(size, training_set)
     errs ← errs + ERROR-RATE(h, validation_set)
  return errs / k        // average error rate on validation sets, across k-fold cross-validation
```

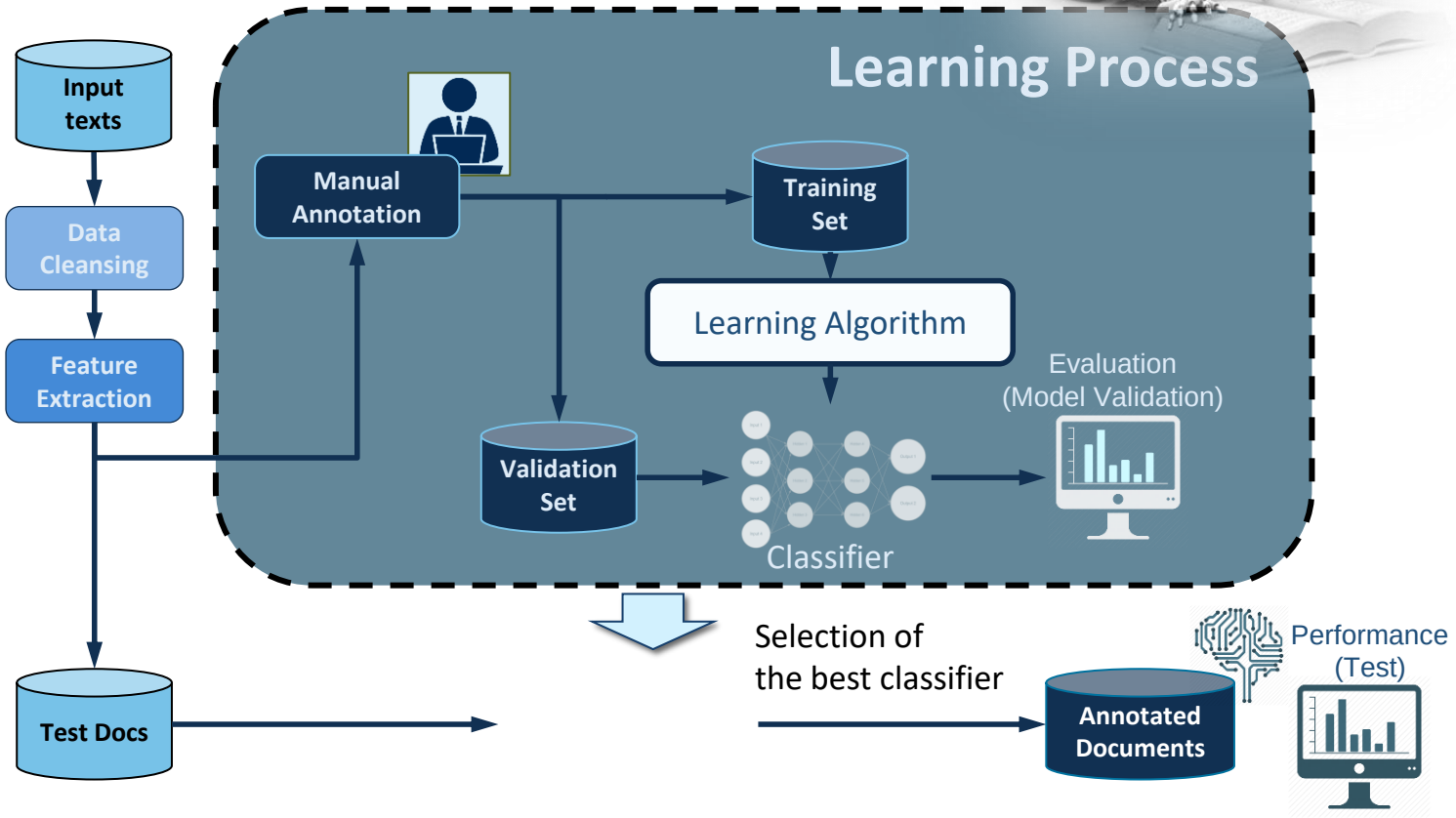
Model Selection: cross validation

```
function MODEL-SELECTION(Learner, examples, k) returns a (hypothesis, error rate) pair

  err ← an array, indexed by size, storing validation-set error rates
  training_set, test_set ← a partition of examples into two sets
  for size = 1 to ∞ do
      err[size] ← CROSS-VALIDATION(Learner, size, training_set, k)
      if err is starting to increase significantly then
          best_size ← the value of size with minimum err[size]
          h ← Learner(best_size, training_set)
          return h, ERROR-RATE(h, test_set)

function CROSS-VALIDATION(Learner, size, examples, k) returns error rate

  N ← the number of examples
  errs ← 0
  for i = 1 to k do
     validation_set ← examples[(i − 1) × N/k:i × N/k]
     training_set ← examples − validation_set
     h ← Learner(size, training_set)
     errs ← errs + ERROR-RATE(h, validation_set)
  return errs / k        // average error rate on validation sets, across k-fold cross-validation
```

Figure 19.8 An algorithm to select the model that has the lowest validation error. It builds models of increasing complexity, and choosing the one with best empirical error rate, *err*, on the validation data set. *Learner*(*size*, *examples*) returns a hypothesis whose complexity is set by the parameter *size*, and which is trained on *examples*. In CROSS-VALIDATION, each iteration of the **for** loop selects a different slice of the *examples* as the validation set, and keeps the other examples as the training set. It then returns the average validation set error over all the folds. Once we have determined which value of the *size* parameter is best, MODEL-SELECTION returns the model (i.e., learner/hypothesis) of that size, trained on all the training examples, along with its error rate on the held-out test examples.

Machine Learning workflow





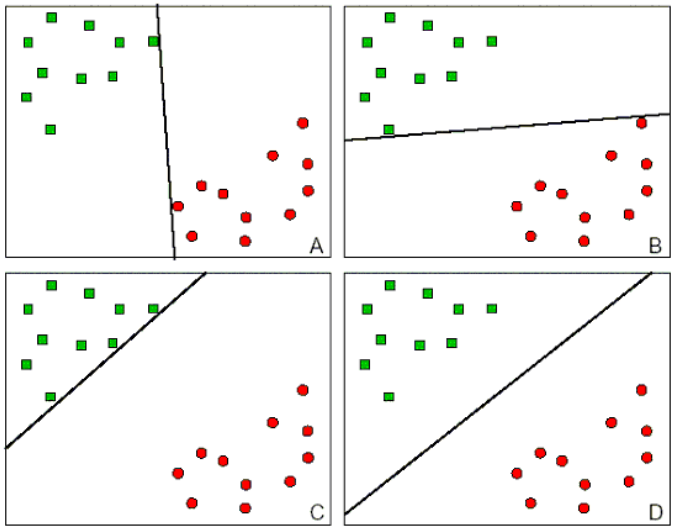
METODI DI MACHINE LEARNING

Tipi di Apprendimento

- Apprendimento Supervisionato
- Apprendimento Unsupervised
- Weakly Supervised Learning
- Reinforcement Learning

Metodi di ML: Classi di funzioni

- **Approcci discriminativi**
  - Lineari
  - h(x) = sign(**W** · **x** + b)



  - Approcci probabilistici
    - Stima delle probabilità $p(\mathcal{C}_k|\mathbf{x})$ attraverso un training set
    - Modello generativo ed uso della inversione Bayesiana

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}.$$

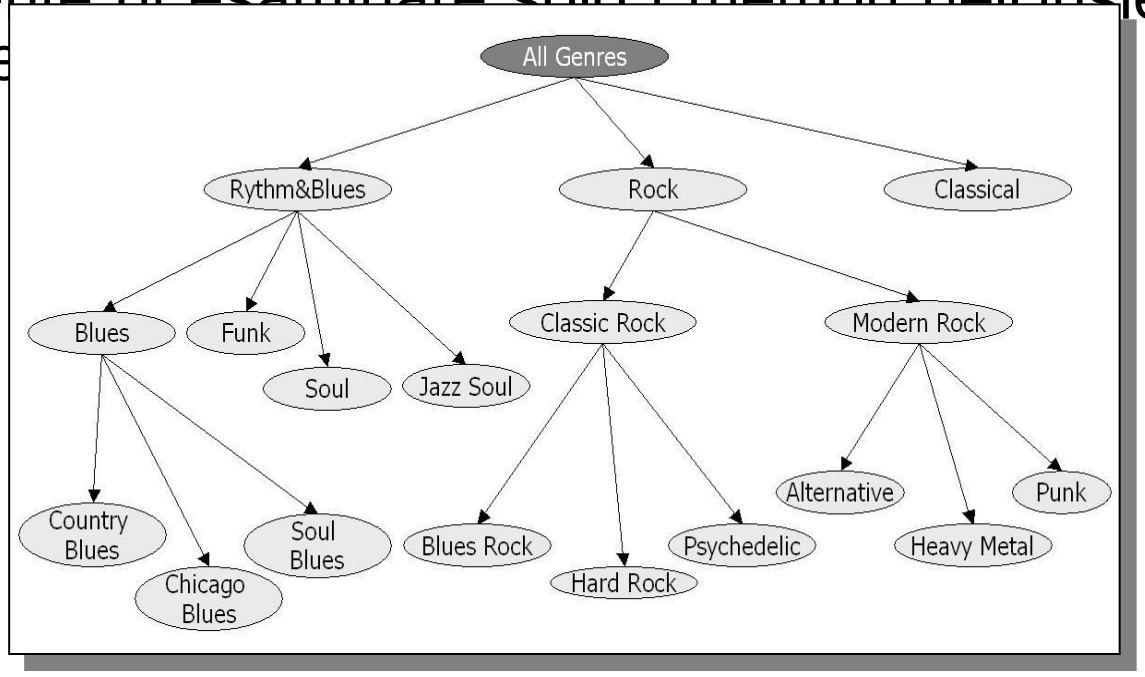
Apprendimento senza supervisione

- In assenza di un oracolo o di conoscenze sul task esistono ancora molti modi di migliorare le proprie prestazioni, ad es.
  - Migliorando il proprio modello del mondo (acquisizione/*discovery* della conoscenza)
  - Migliorando le proprie prestazioni computazionali (ottimizzazione)

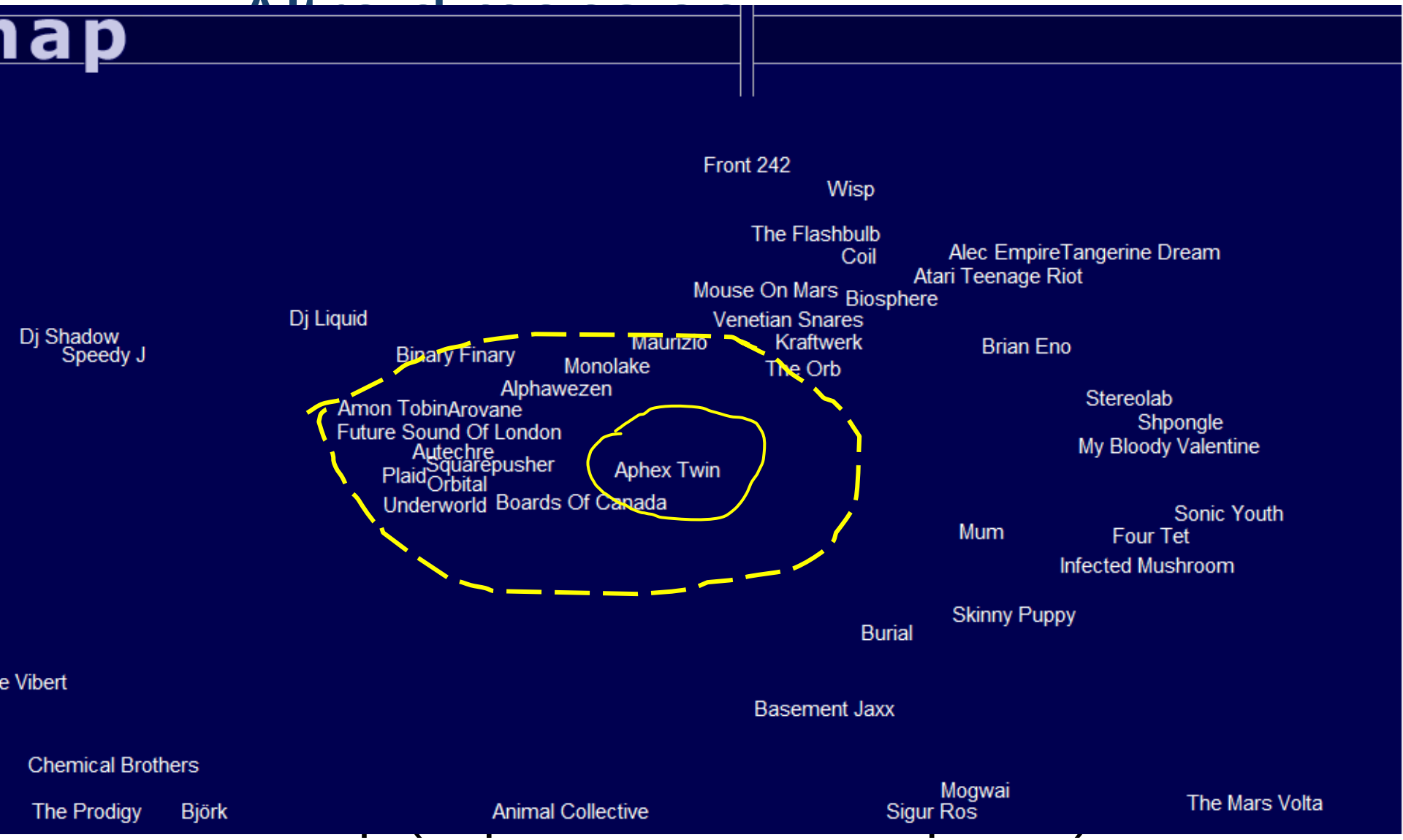
Apprendimento senza supervisione

Al termine del processo di acquisizione il sistema dispone di un sistema di classi e relazioni indotti che migliora la sua interazione futura con l'ambiente operativo (ad es. l'utente)

- Es
  - u
    g
    s
    o
  - Il miglioramento avviene quindi almeno rispetto agli algoritmi di ricerca: la organizzazione gerarchica consente di esaminare solo i membri dell'insieme in alcune



nap


Front 242
Wisp
The Flashbulb
Coil
Alec Empire
Tangerine Dream
Atari Teenage Riot
Mouse On Mars
Biosphere
Dj Liquid
Venetian Snares
Dj Shadow
Speedy J
Maurizio
Kraftwerk
Brian Eno
Binary Finary
Monolake
The Orb
Alphawezen
Stereolab
Amon Tobin
Arovane
Shpongle
Future Sound Of London
My Bloody Valentine
Autechre
Squarepusher
Plaid
Orbital
Aphex Twin
Underworld
Boards Of Canada
Sonic Youth
Mum
Four Tet
Infected Mushroom
Skinny Puppy
Burial
e Vibert
Basement Jaxx
Chemical Brothers
Mogwai
The Prodigy
Björk
Animal Collective
Sigur Ros
The Mars Volta

RETI NEURALI

Linear Classification

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$$

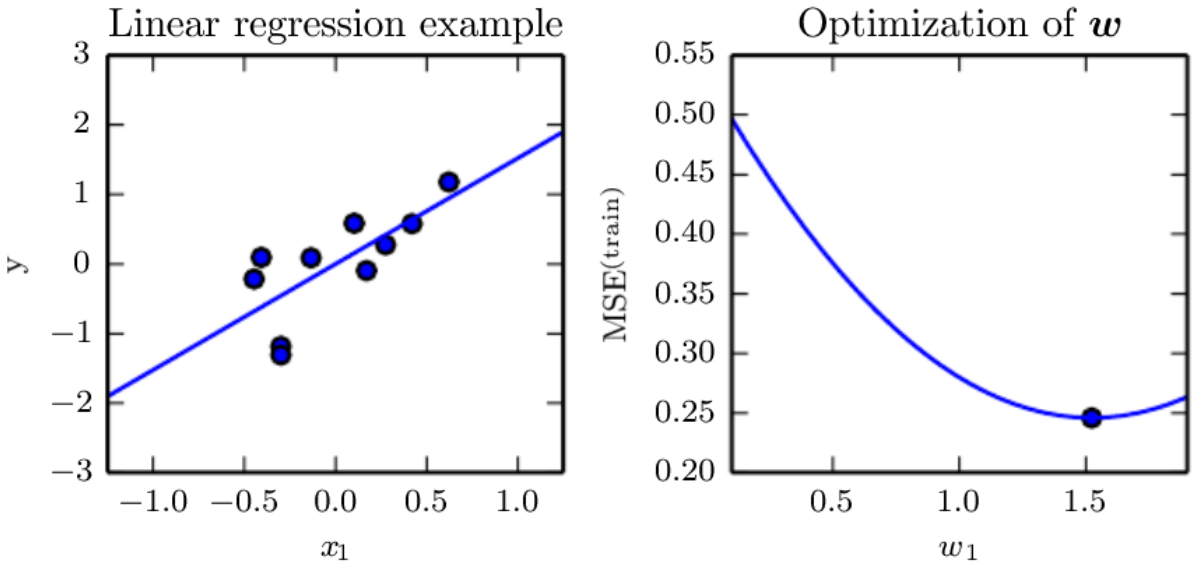


Figure 5.1: A linear regression problem, with a training set consisting of ten data points, each containing one feature. Because there is only one feature, the weight vector $\boldsymbol{w}$ contains only a single parameter to learn, w_1. *(Left)*Observe that linear regression learns to set w_1 such that the line $y = w_1 x$ comes as close as possible to passing through all the training points. *(Right)*The plotted point indicates the value of w_1 found by the normal equations, which we can see minimizes the mean squared error on the training set.

Linear Classification (2)

- Limitations:
  - Data separability (poor over non linear correlations)
  - Hard constraints not suitable for error-prone data
  - Lack of expressiveness: not so many parameters to make proper use of several variables and their possible interactions

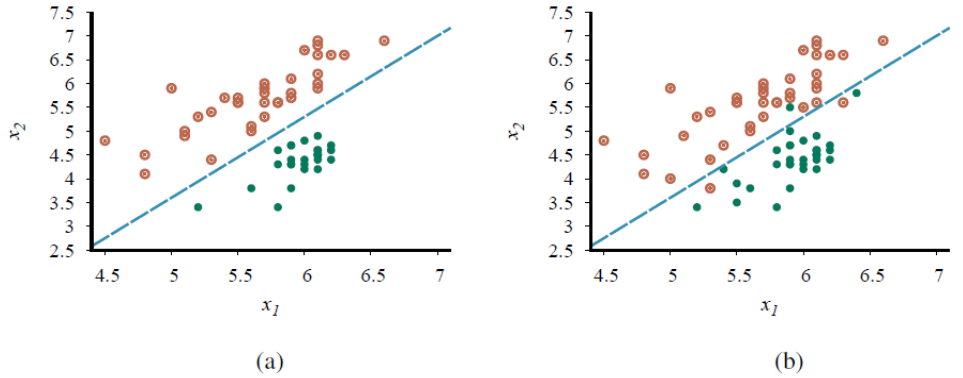
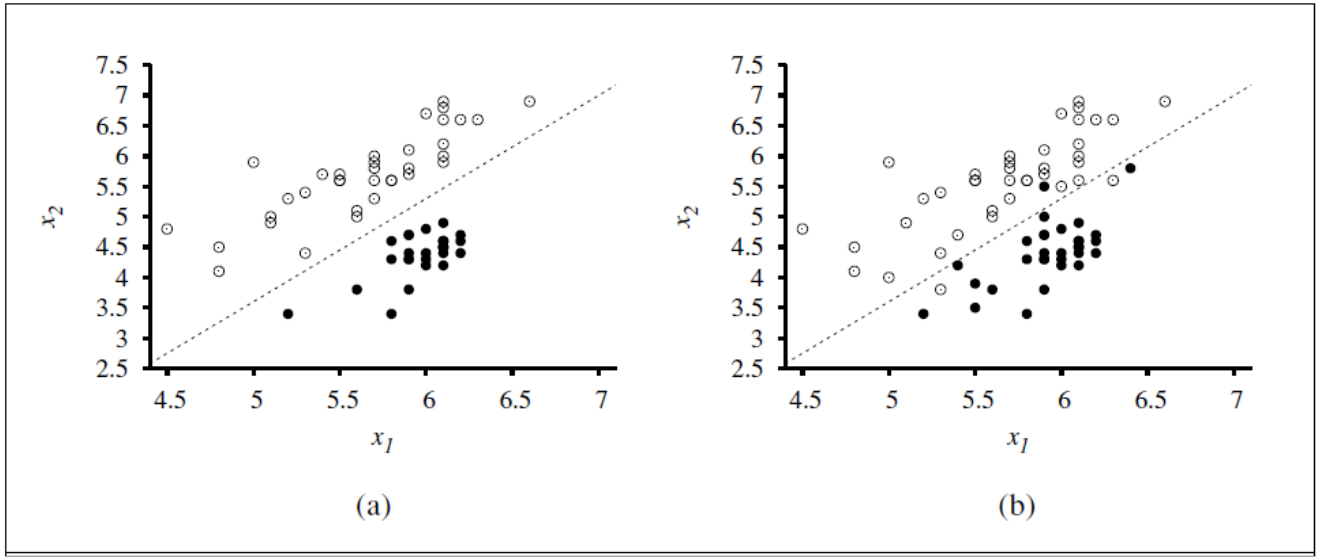


Figure 19.15 (a) Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (open orange circles) and nuclear explosions (green circles) occurring between 1982 and 1990 in Asia and the Middle East (Kebeasy *et al.*, 1998). Also shown is a decision boundary between the classes. (b) The same domain with more data points. The earthquakes and explosions are no longer linearly separable.

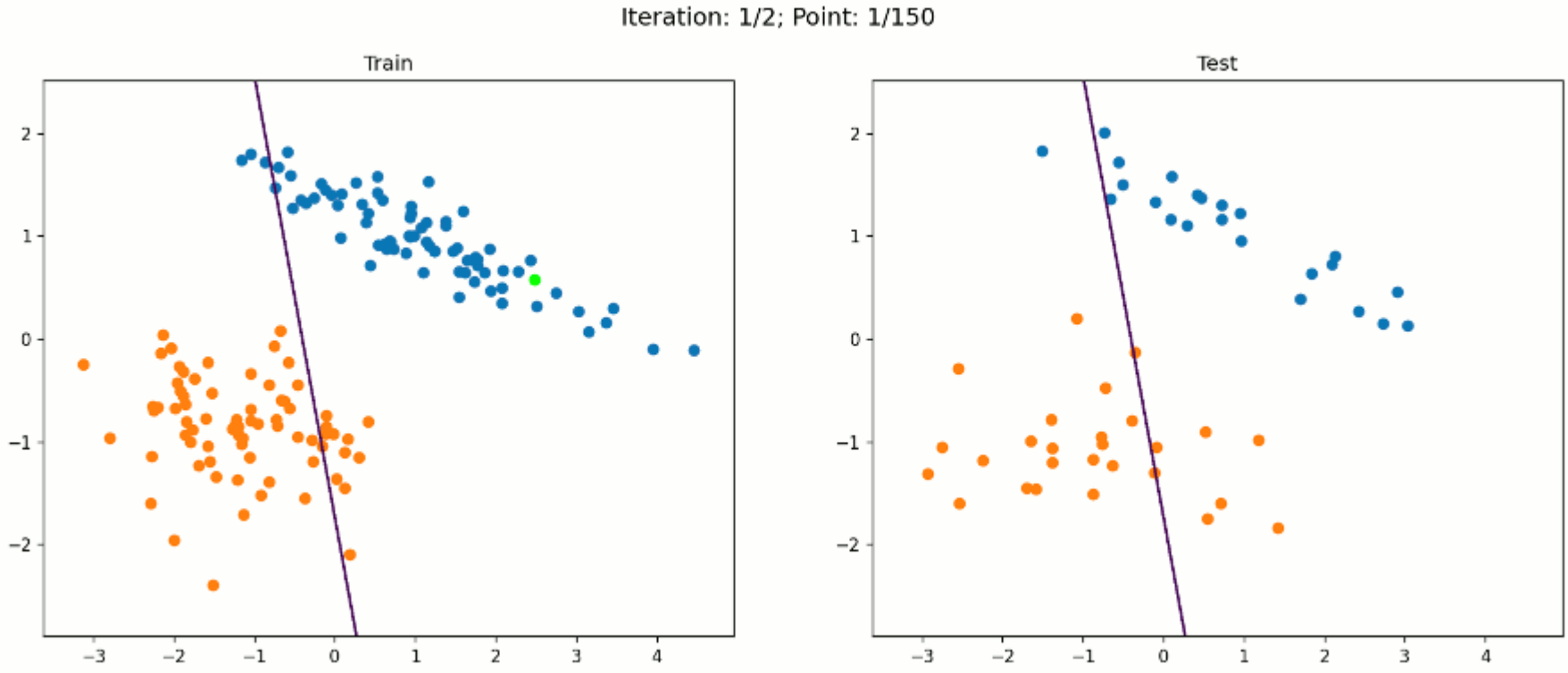
Linear Classification (3): an example



- Learned Model: $x_2 = 1.7x_1 - 4.9 \quad \text{or} \quad -4.9 + 1.7x_1 - x_2 = 0\,.$
- Separability Rule: $-4.9 + 1.7x_1 - x_2 > 0$

- Generalizing: $h_{\mathbf{w}}(\mathbf{x}) = 1$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$ and 0 otherwise.
- How to achieve it?: $w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i$

… Adjusting weights



Learning Linear Classifier

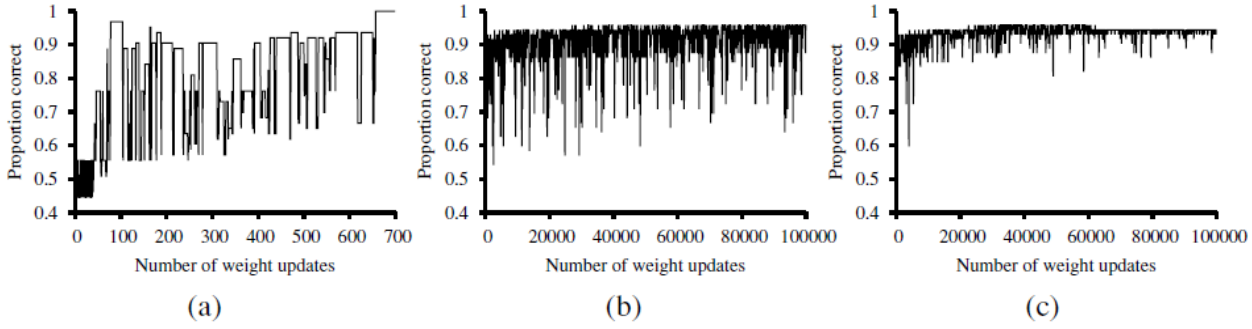


Figure 18.16 (a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data in Figure 18.15(a). (b) The same plot for the noisy, non-separable data in Figure 18.15(b); note the change in scale of the x-axis. (c) The same plot as in (b), with a learning rate schedule $\alpha(t) = 1000/(1000 + t)$.

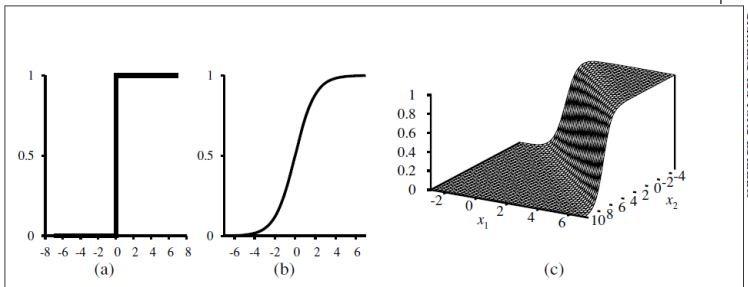


Figure 18.17 (a) The hard threshold function $Threshold(z)$ with 0/1 output. Note that the function is nondifferentiable at $z = 0$. (b) The logistic function, $Logistic(z) = \frac{1}{1+e^{-z}}$, also known as the sigmoid function. (c) Plot of a logistic regression hypothesis $h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})$ for the data shown in Figure 18.15(b).

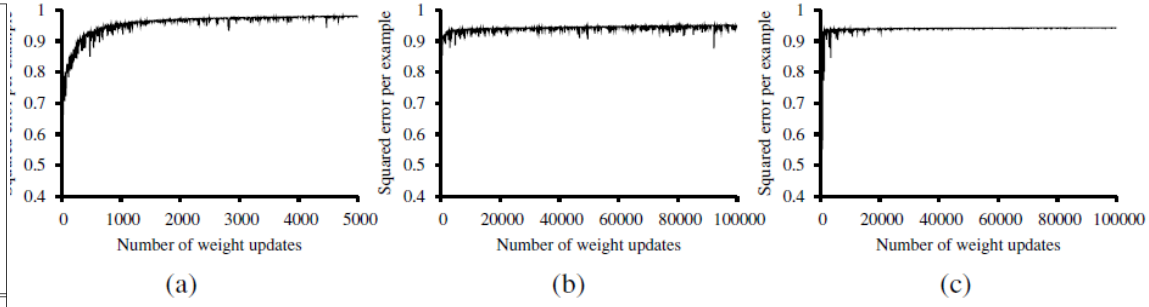
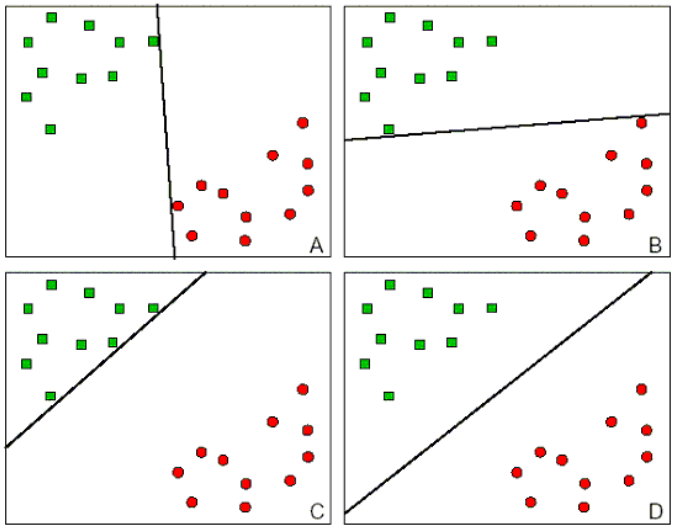
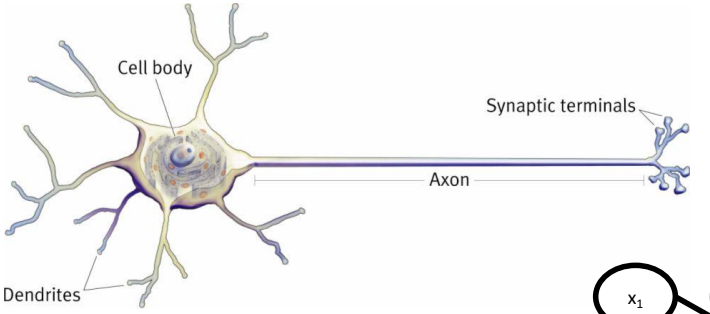


Figure 18.18 Repeat of the experiments in Figure 18.16 using logistic regression and squared error. The plot in (a) covers 5000 iterations rather than 1000, while (b) and (c) use the same scale.

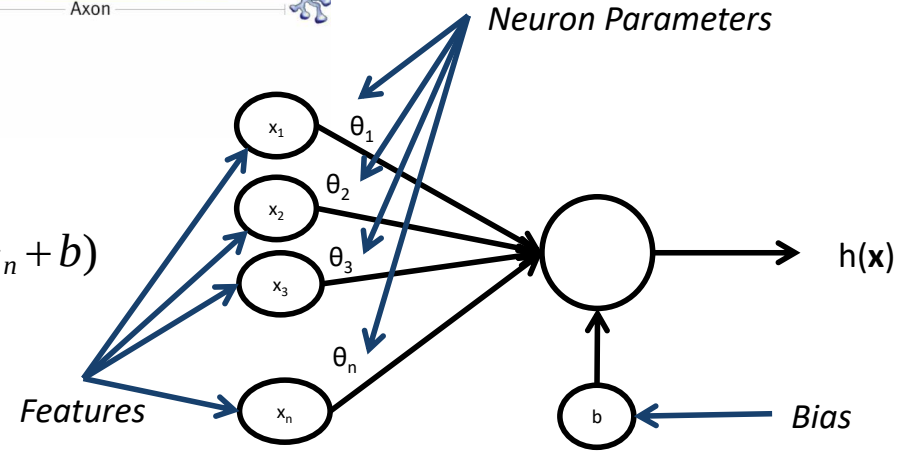
Perceptron (Rosenblatt, 1958)

- Linear Classifier mimicking a neuron





$$h(\vec{x}) = g(\sum_n \theta_n x_n + b)$$



Learning the XOR function

- The problem: $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$

XOR is not linearly separable

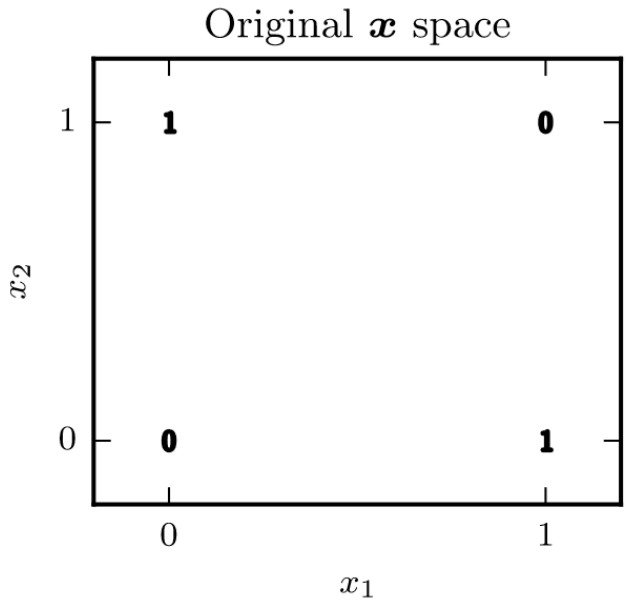
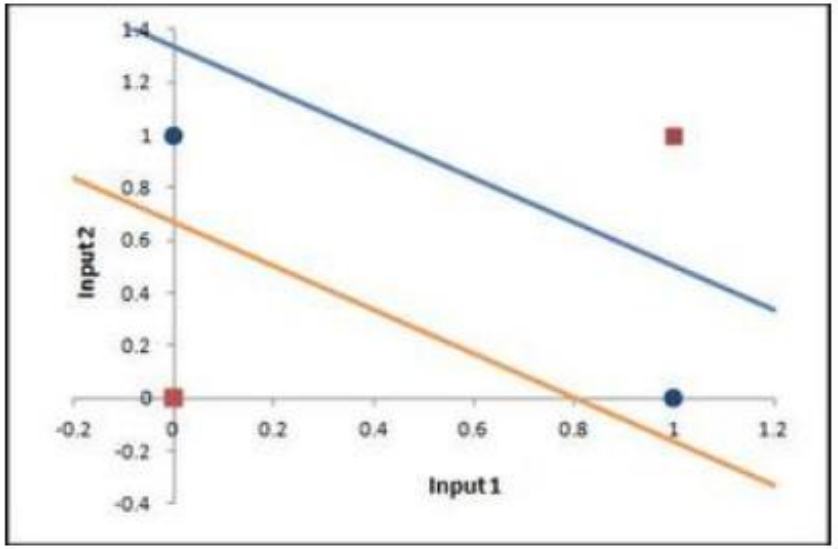


Figure 6.1, left

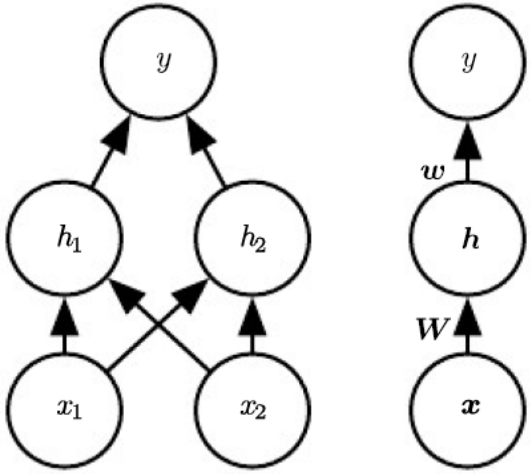
(Goodfellow 2017)

A simple MLP: the XOR function

Input1	Input2	Output
0	0	0
1	0	1
0	1	1
1	1	0



A MLP for the XOR problem



We can now specify our complete network as

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

Figure 6.2: An example of a feedforward network, drawn in two different styles. Specifically, this is the feedforward network we use to solve the XOR example. It has a single hidden layer containing two units. *(Left)* In this style, we draw every unit as a node in the graph. This style is explicit and unambiguous, but for networks larger than this example, it can consume too much space. *(Right)* In this style, we draw a node in the graph for each entire vector representing a layer's activations. This style is much more compact. Sometimes we annotate the edges in this graph with the name of the parameters that describe the relationship between two layers. Here, we indicate that a matrix $\boldsymbol{W}$ describes the mapping from $\boldsymbol{x}$ to $\boldsymbol{h}$, and a vector $\boldsymbol{w}$ describes the mapping from $\boldsymbol{h}$ to y. We typically omit the intercept parameters associated with each layer when labeling this kind of drawing.

The solution

We can then specify a solution to the XOR problem. Let

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \tag{6.4}$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \tag{6.5}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \tag{6.6}$$

and $b = 0$.

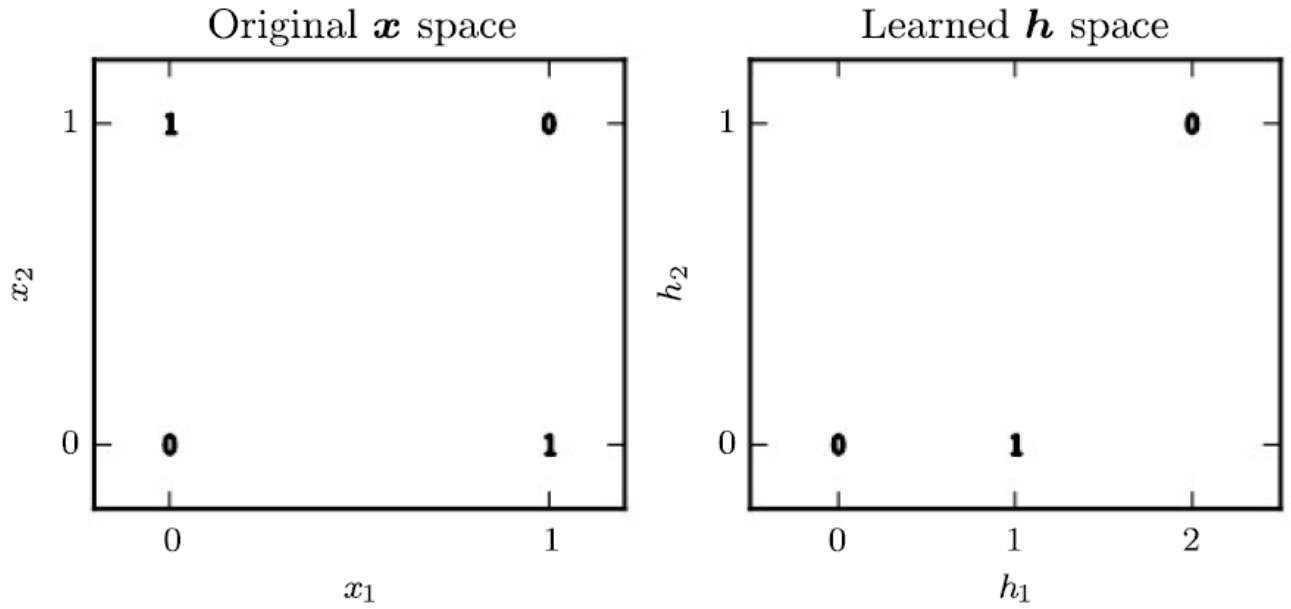
$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} \quad \boldsymbol{XW} + \boldsymbol{c} \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \quad \boldsymbol{w}^\top \max\{0, \boldsymbol{XW} + \boldsymbol{c}\} + b. \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

We can now specify our complete network as

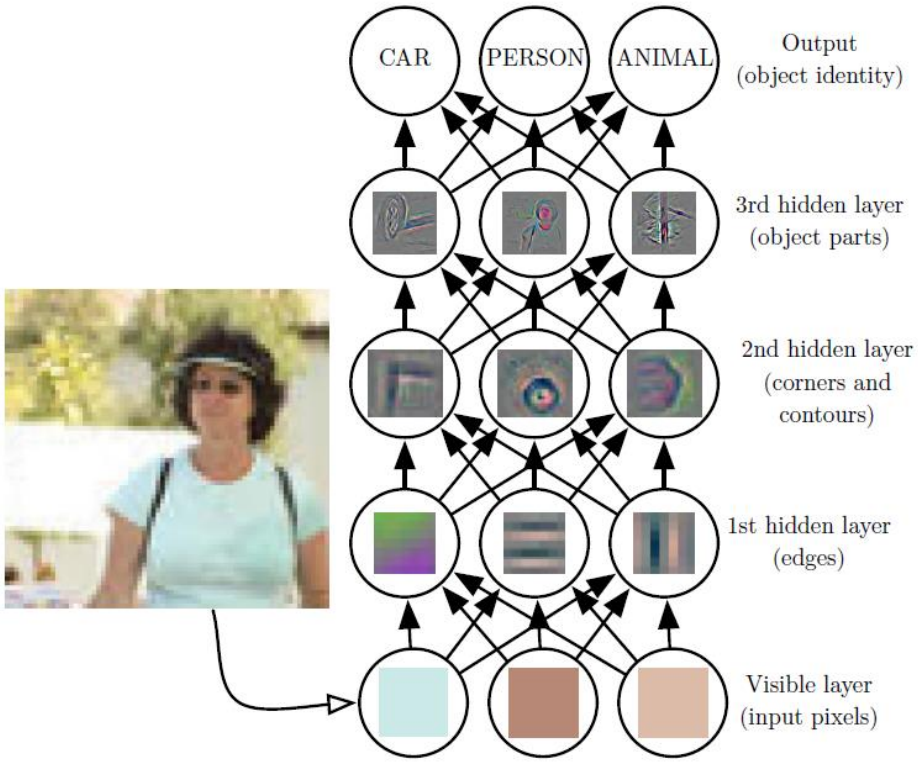
$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The new representation space



Representation and Learning



Zeiler and Fergus (2014)

Networks and Information Flow

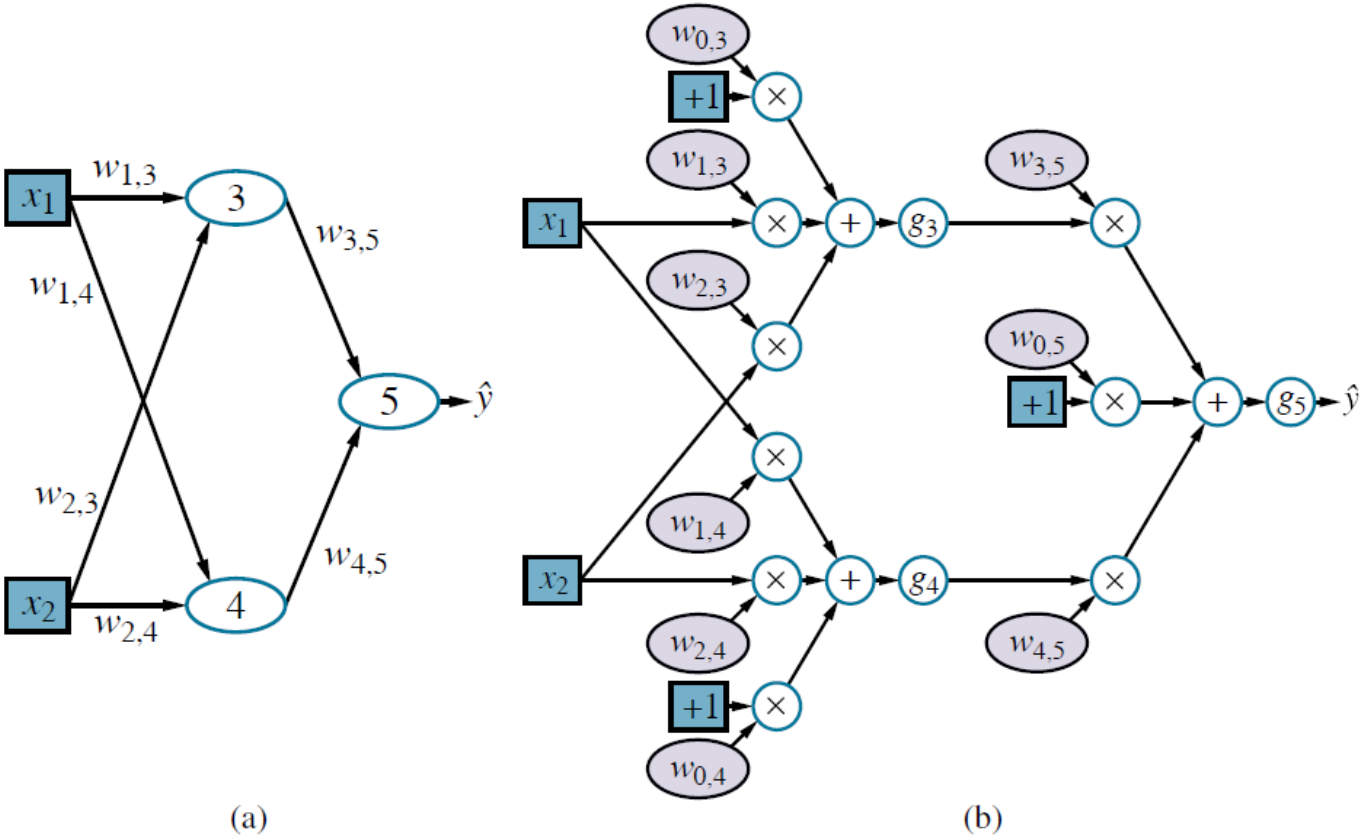
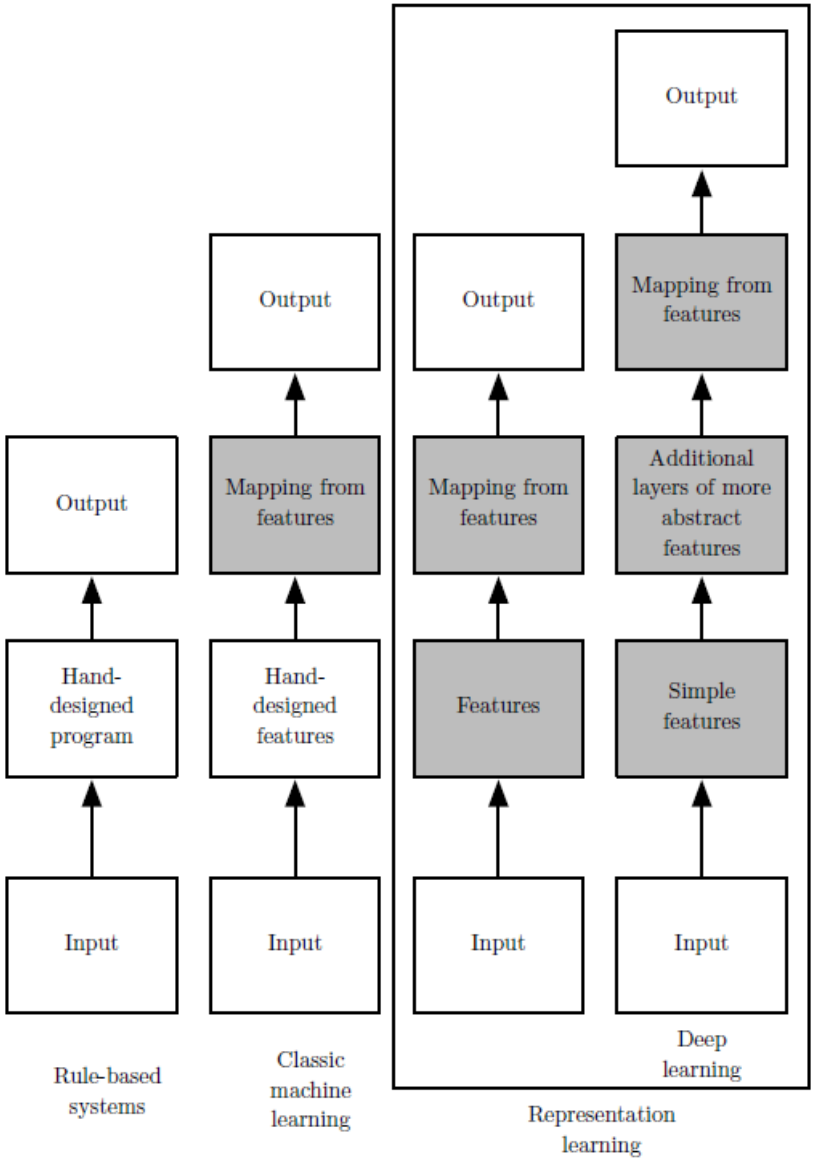


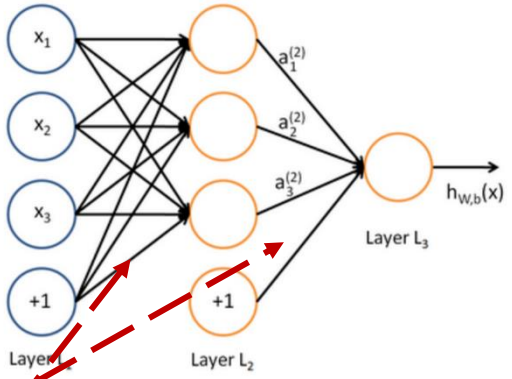
Figure 22.3 (a) A neural network with two inputs, one hidden layer of two units, and one output unit. Not shown are the dummy inputs and their associated weights. (b) The network in (a) unpacked into its full computation graph.

Learning Multiple Components



from Goodfellow et al., DL MIT book

Adding Layers …



- From simple linear laws …

$$h(\vec{x}) = g(\vec{x};\vec{\theta},b) = g(\sum_n \theta_n x_n + b)$$

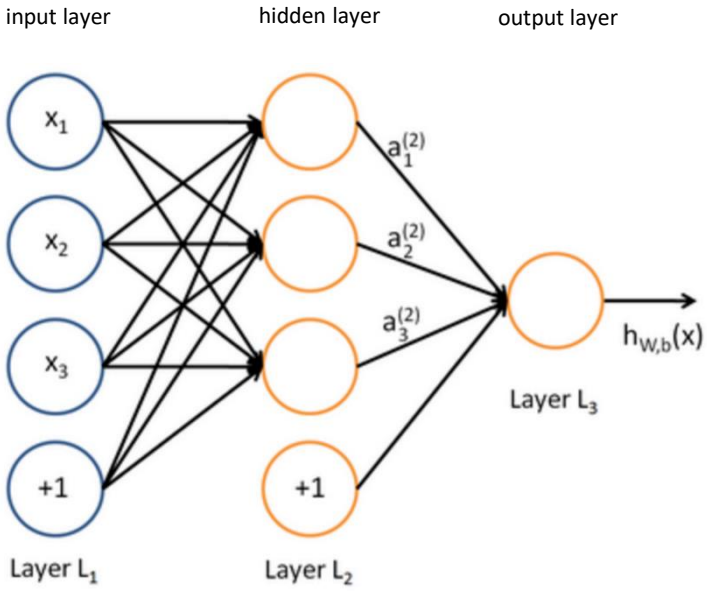
- to feedforward structures. It can be made dependent on a sequence of functions $g^{(1)}$ and $g^{(2)}$, …, $g^{(k)}$ that give rise to a structured hypothesis:

$$h(\vec{x}) = g^{(2)}(g^{(1)}(\vec{x};\vec{\theta}^{(1)},b^{(1)});\vec{\theta}^{(2)},b^{(2)}) =$$
$$= W^{(2)}g^{(2)}(g^{(1)}(W^{(1)}\cdot\vec{x}+b^{(1)})+b^{(2)}$$

- Hidden layers $h^{(1)}(\vec{x}) = g^{(1)}(W^{(1)}\vec{x}+b^{(1)})$

Artificial Neural Networks

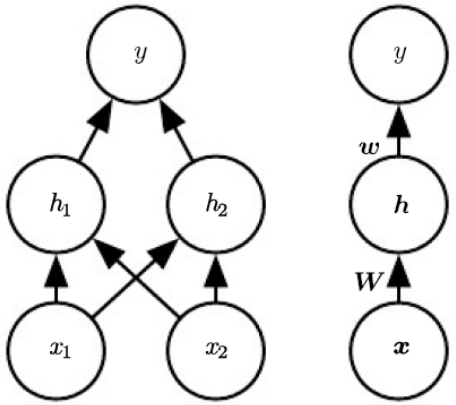
- Each circle represent a **neuron** (or unit)
  - In Figure: 3 **input**, 3 **hidden** and 1 **output**
- n_l=3 is the **number of layers**
- s_l denotes the **number of units in layer *l***
- Layers:
  - Layer *l* is denoted as L_l
  - Layer *l* and *l+1* are connected by a matrix $W^{(l)}$ of parameters
    - $W^{(l)}_{i,j}$ connects neuron *j* in layer *l* with neuron *i* in layer *l+1*
- $b^{(l)}_i$ is the *bias* associated to neuron *l* in layer *l+1*



$$h(\vec{x}) = g^{(2)}(g^{(1)}(\vec{x};\vec{\theta}^{(1)},b^{(1)});\vec{\theta}^{(2)},b^{(2)}) =$$
$$= W^{(2)}g^{(2)}(g^{(1)}(W^{(1)}\cdot\vec{x}+b^{(1)})+b^{(2)})$$

$$h^{(1)}(\vec{x}) = g^{(1)}(W^{(1)}\vec{x}+b^{(1)})$$

Training MLPs: Back-propagation



- How are parameters of the two-layer network, i.e. *W, w* and *c, b* defined?
- This is the role of the training algorithm for which:

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b \approx f^*(\boldsymbol{x}).$$

- The learning process in MLPs is based on two notions:
  - The optimization local to individual neurons
  - The adjustments to the overall network by propagation backwards from the output (where the error manifests) through all the hidden layers.

Classification and Learning

- Forward classification vs. backpropagation learning

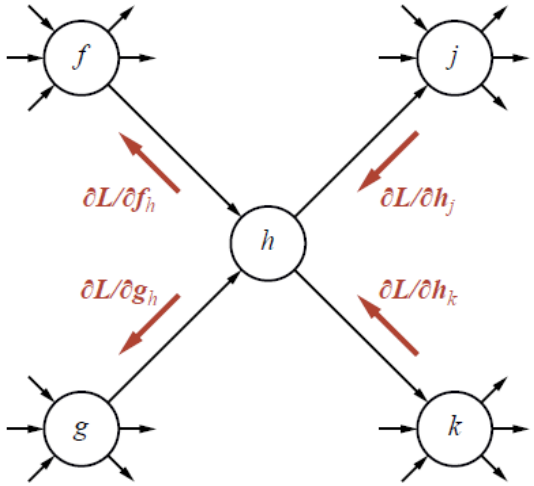
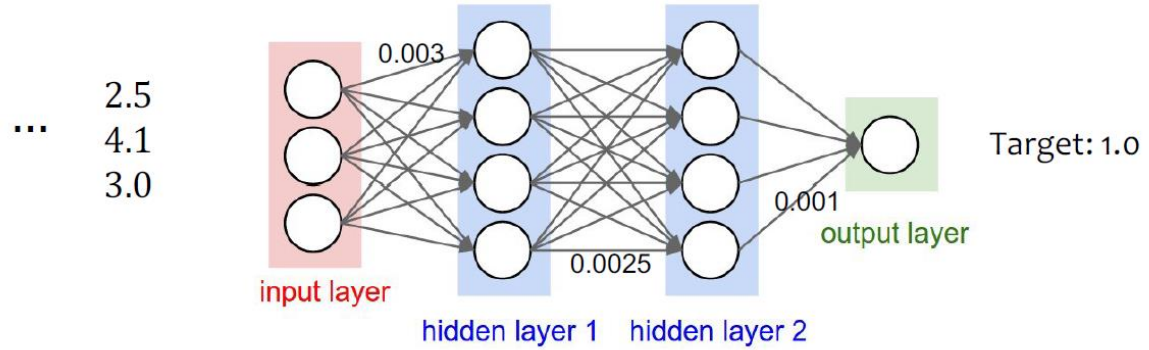
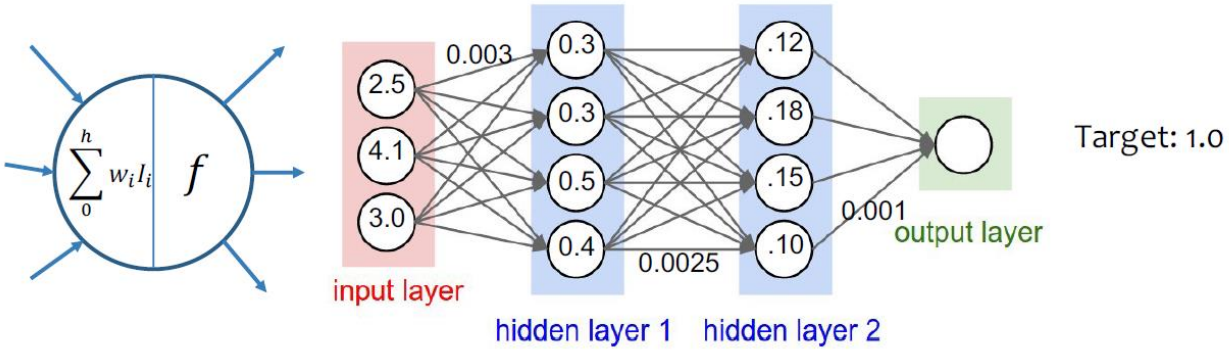


Figure 22.6 Illustration of the back-propagation of gradient information in an arbitrary computation graph. The forward computation of the output of the network proceeds from left to right, while the back-propagation of gradients proceeds from right to left.

Forward Steps

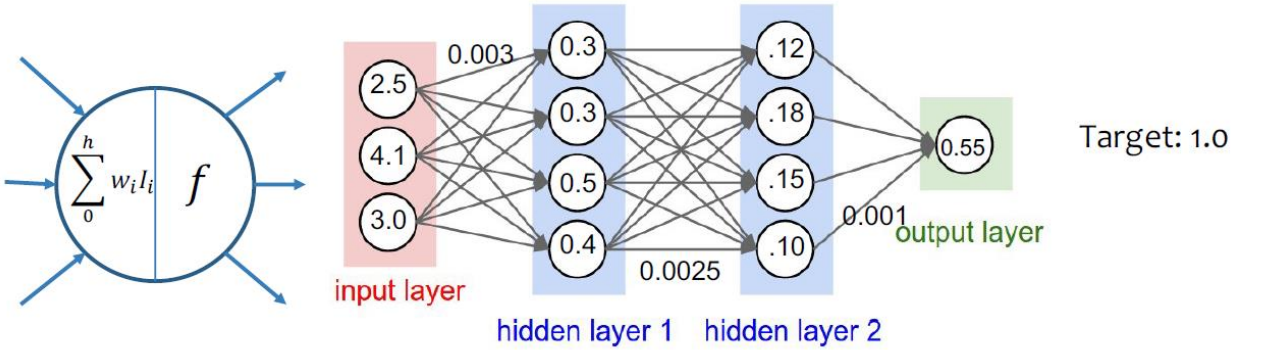


Weight Initialization and Loading the Data

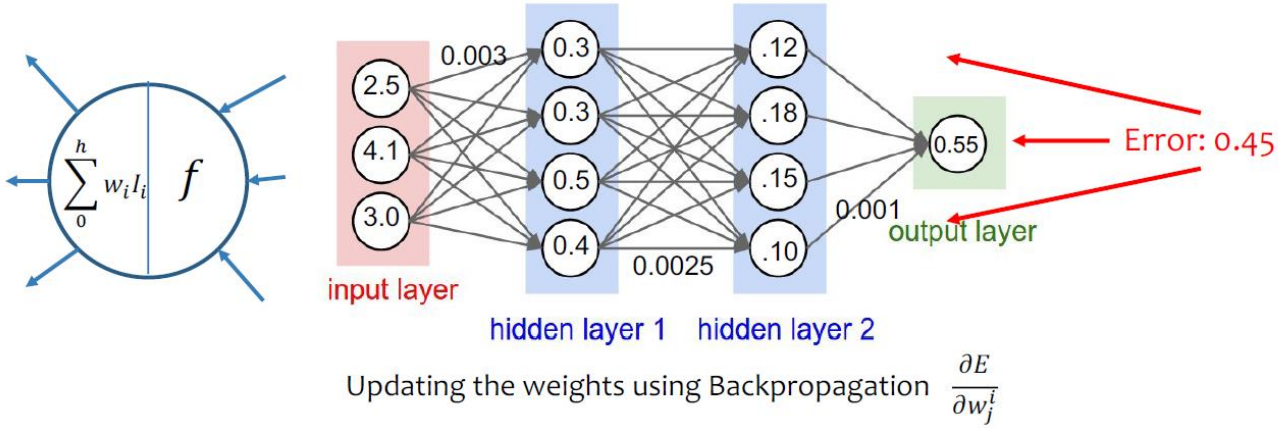


Going forward (Regularization, Dropout, Batch Normalization, …)

… and Backward steps



$Loss\ Function: (T - O) \Rightarrow Loss: 0.45$



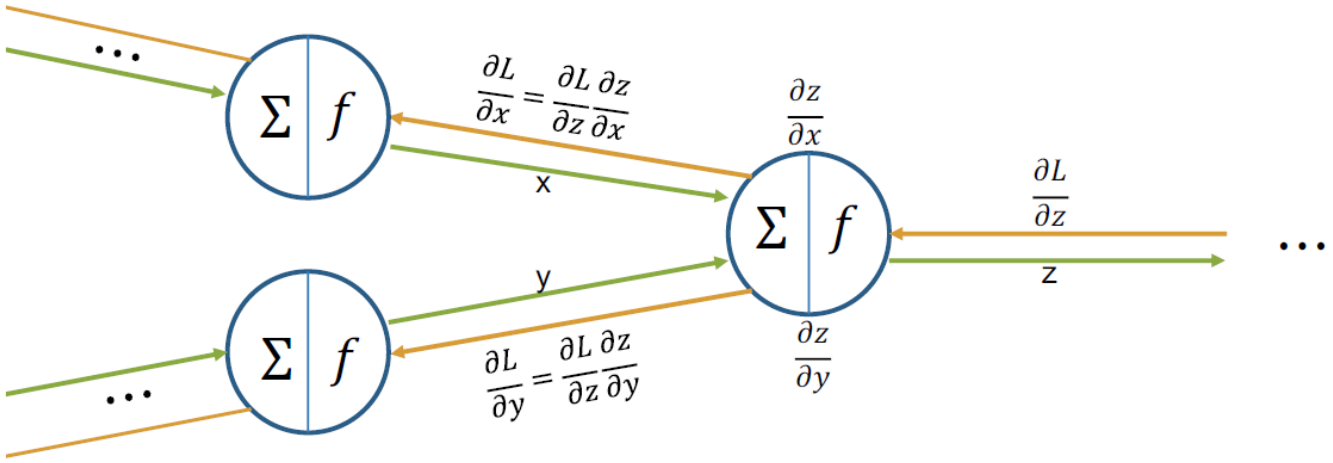
Updating the weights using Backpropagation $\frac{\partial E}{\partial w_j^i}$

How to induce the hypothesis *h* from examples

- Learn the parameters *θ* and *b*
- To find these we look at the past data (i.e. training data) optimizing an objective function
- Objective function: the error we make on the training data
  - the sum of differences between the decision function h and the label y
  - also called Loss Function or Cost Function

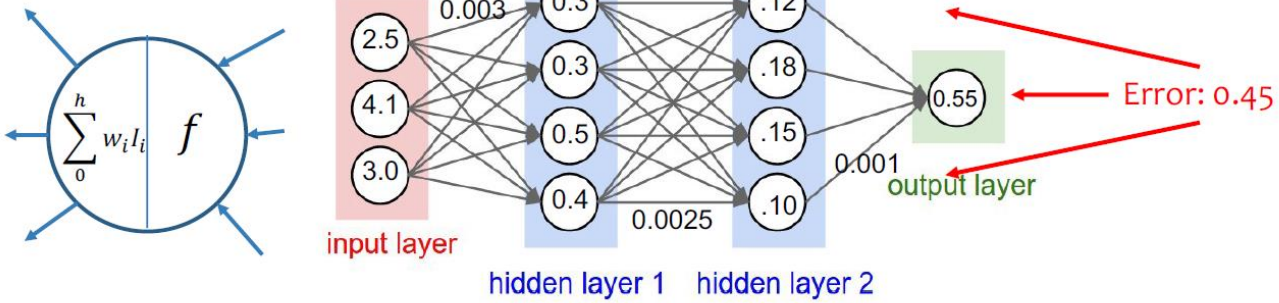
$$J(\theta, b) = \sum_{i=1}^{m} (h(x^{(i)}; \theta, b) - y^{(i)})^2$$

BP as Local Search

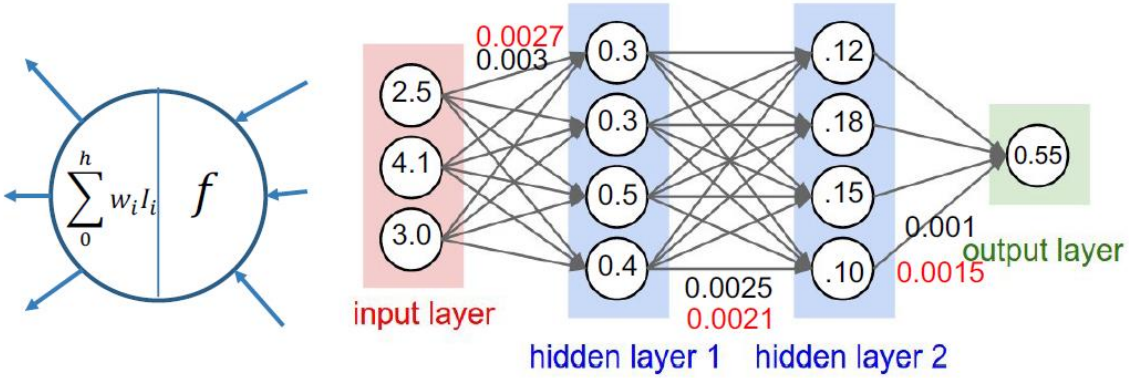


Backpropagation

Reweighting through GD



Updating the weights using Backpropagation $\frac{\partial E}{\partial w_j^i}$



By Choosing the Optimizer, new weights will be computed $w_{new} = w - \eta \frac{\partial E}{\partial w}$

A general training procedure: Stochastic Gradient Descent

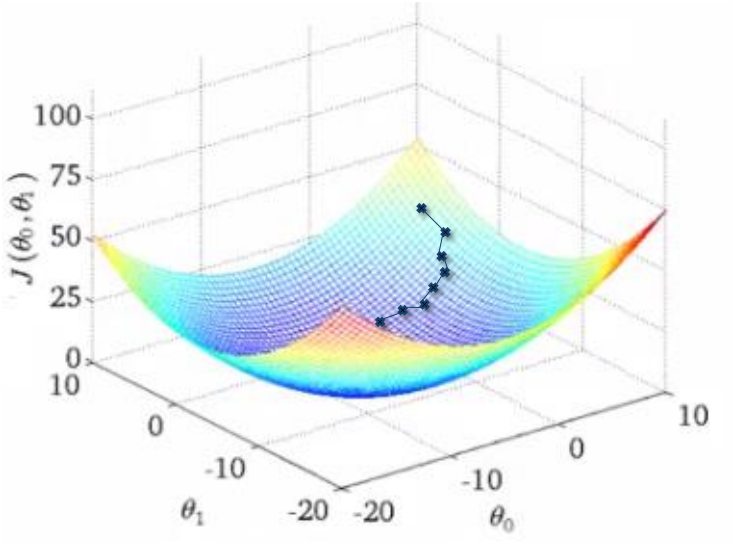
- Optimizing J means minimizing it
  - it measures the errors we make on the training data.
- We can iterate over examples and update the parameters in the direction of smaller costs
  - we aim at finding the minimum of that function
- Concretely,

$$\theta_1 = \theta_1 - \alpha \Delta \theta_1$$
$$\theta_2 = \theta_2 - \alpha \Delta \theta_2$$
$$b = b - \alpha \Delta b$$

- α is a meta-parameter, the learning rate
- Δ are the partial derivatives of the cost function wrt each parameter

Why SGD?

- Weights are updated using the partial derivatives
- Derivative pushes down the cost following the steepest descent path on the error curve



SGD procedure

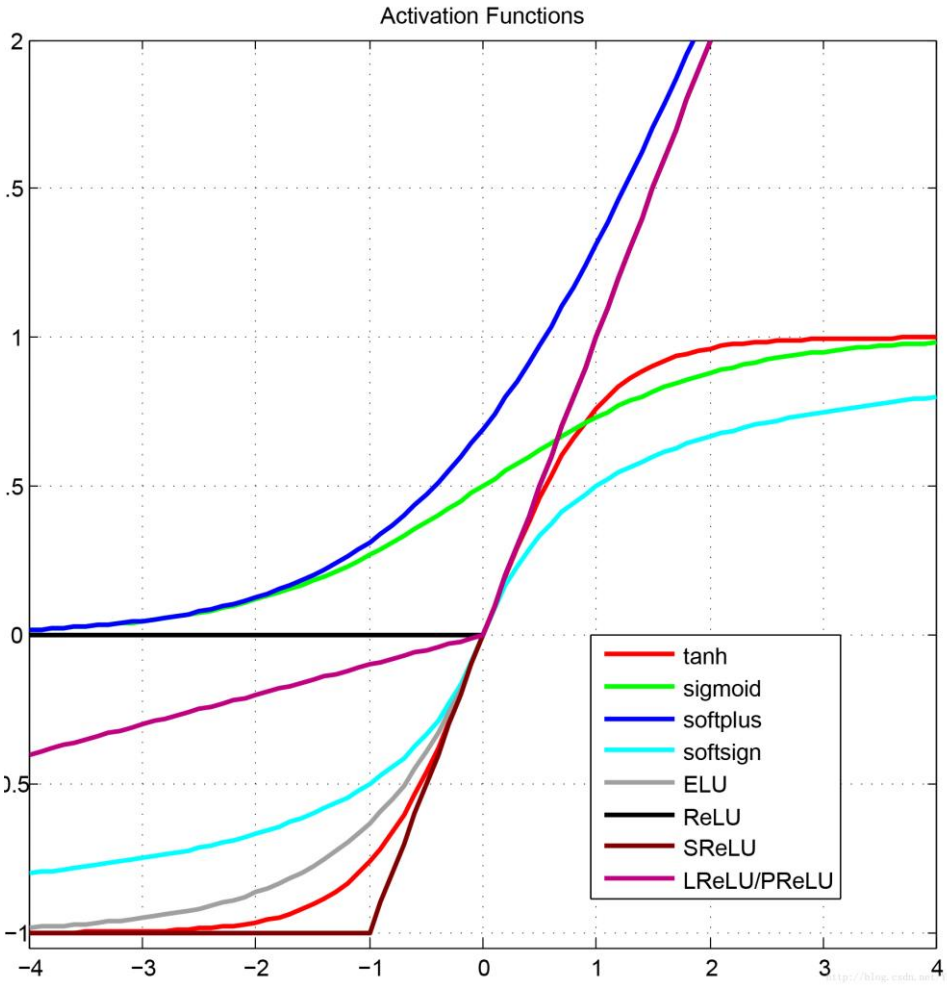
- Choose an initial random values for θ and b
- Choose a learning rate
- Repeat until stop criterion is met:
  - Pick a random training example x(i)
  - Update the parameters with

$$\theta_1 = \theta_1 - \alpha \Delta \theta_1$$
$$\theta_2 = \theta_2 - \alpha \Delta \theta_2$$
$$b = b - \alpha \Delta b$$

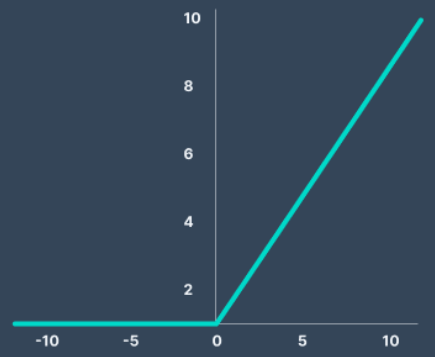
- We can stop
  - when the parameters do not change or,
  - the number of iteration exceeds a certain upper bound

… a bit of math …

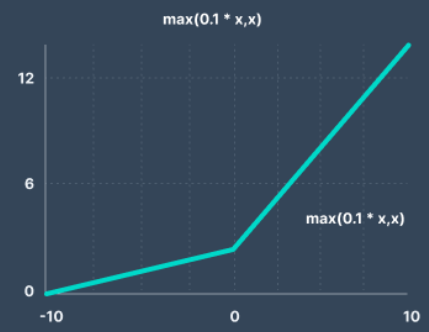
Perceptron and non-linear activation functions



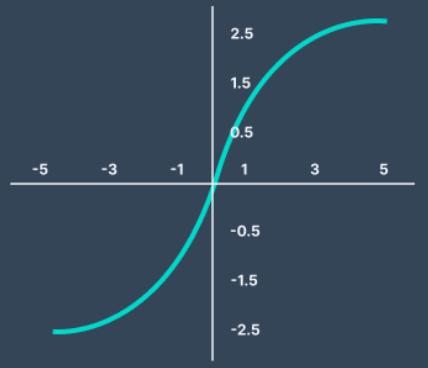
ReLU



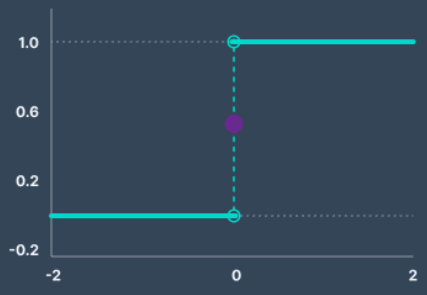
Leaky ReLU



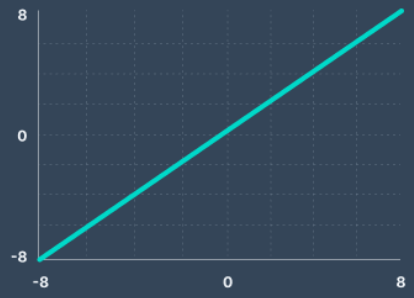
Tanh



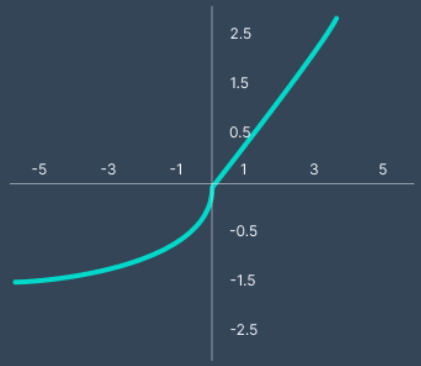
Binary Step Function



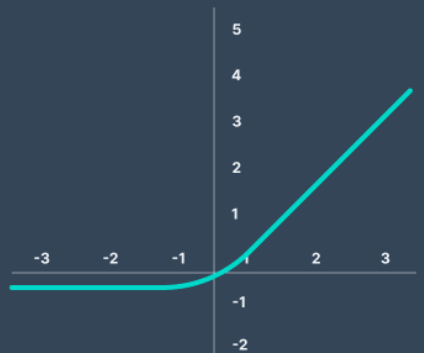
Linear



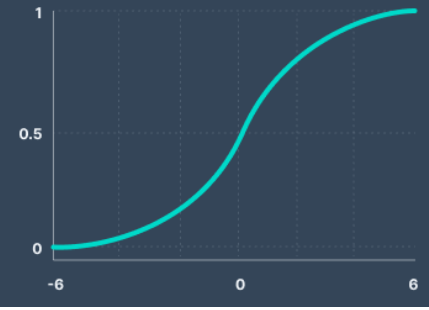
SELU



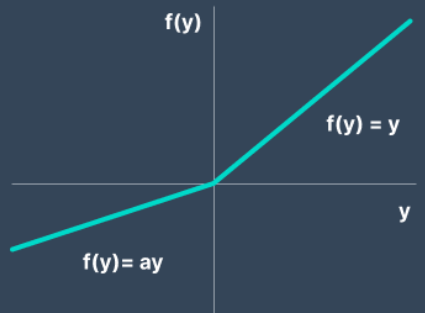
ELU



Sigmoid / Logistic



Parametric ReLU



How to induce h from examples

- Learn the parameters θ and b
- To find these we look at the past data (i.e. training data) optimizing an objective function
- **Objective function**: the error we make on the training data
  - the sum of differences between the decision function h and the label y
  - also called **Loss Function** or **Cost Function**

$$J(\theta, b) = \sum_{i=1}^{m} (h(x^{(i)}; \theta, b) - y^{(i)})^2$$

A general training procedure: Stochastic Gradient Descent

- Optimizing J means **minimizing** it
  - it measures the errors we make on the training data.
- We can iterate over examples and update the parameters in the direction of smaller costs
  - we aim at finding the minimum of that function
- Concretely,

$$\theta_1 = \theta_1 - \alpha \Delta \theta_1$$

$$\theta_2 = \theta_2 - \alpha \Delta \theta_2$$

$$b = b - \alpha \Delta b$$

- α is a meta-parameter, the learning rate
- Δ are the partial derivatives of the cost function *wrt* each parameter

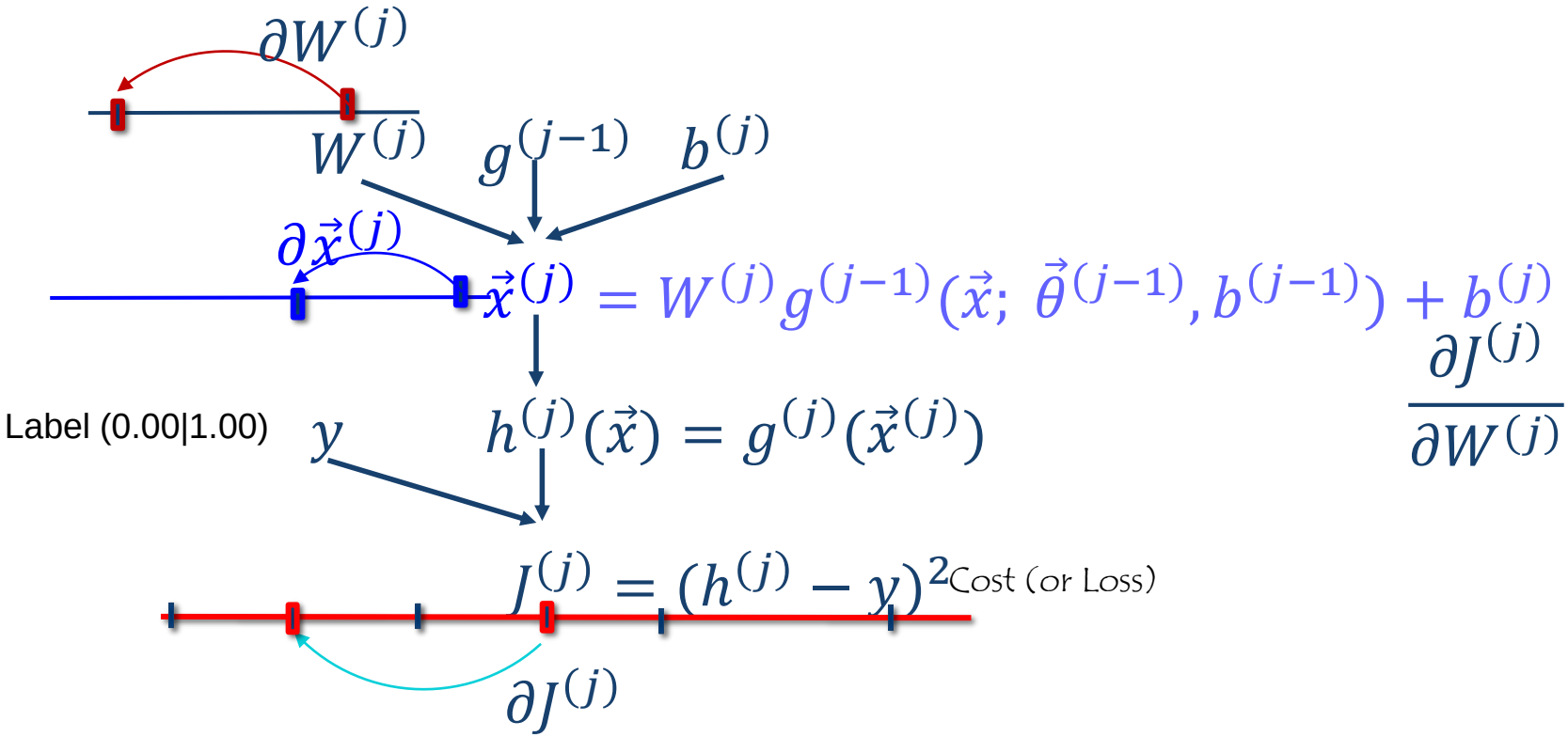
Optimizing *J*

- From the network

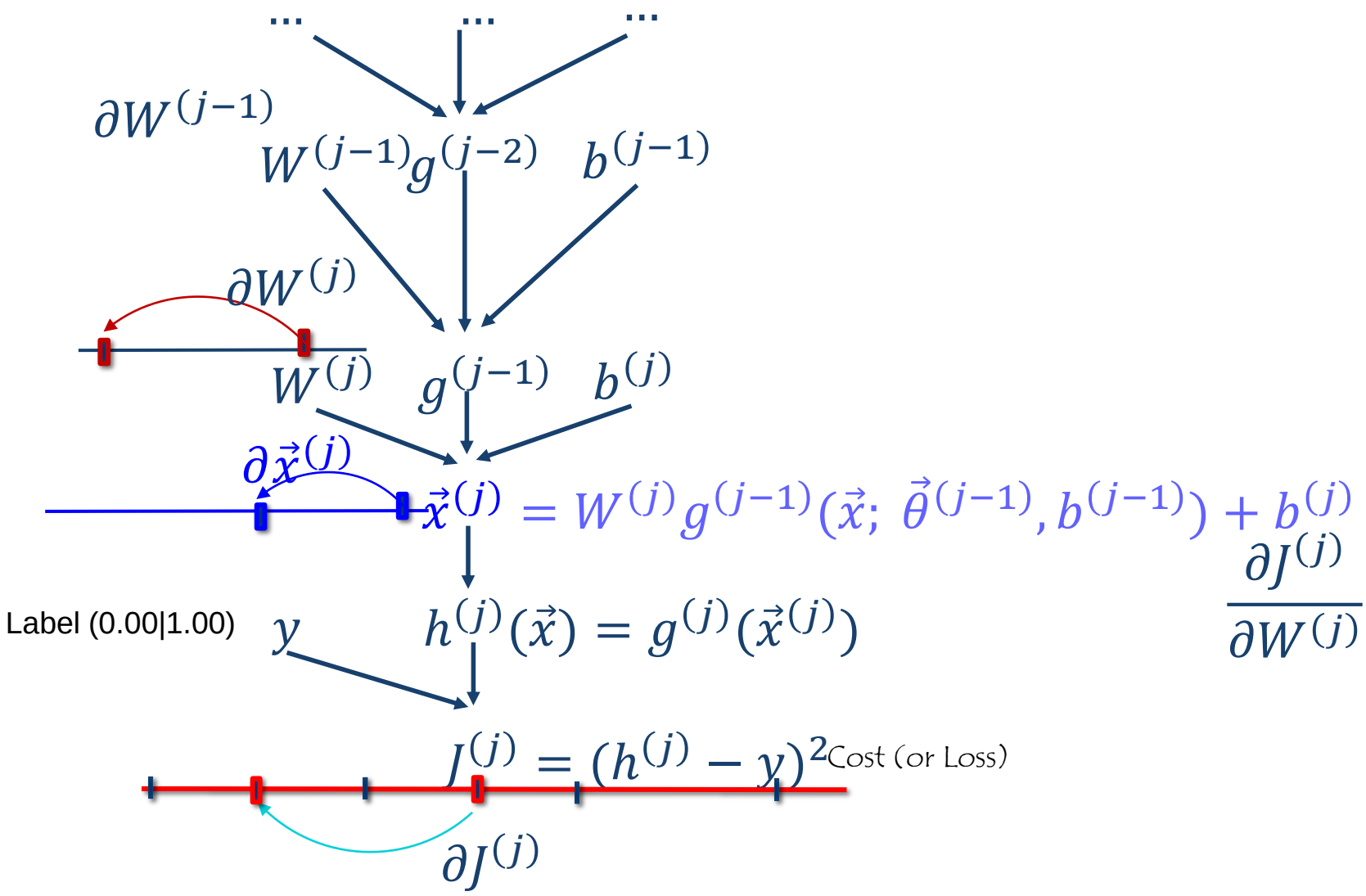
$$h(\vec{x}) = g^{(k)}\left(g^{(k-1)}(\ldots g^{(1)}(\vec{x};\vec{\theta}^{(1)},b^{(1)});\ldots);\vec{\theta}^{(k-1)},b^{(k-1)});\vec{\theta}^{(k)},b^{(k)}\right)=$$
$$= g^{(k)}\left(W^{(k)}g^{(k-1)}(W^{(k-1)}\ldots.g^{(1)}(W^{(1)}\vec{x}+\ b^{(1)})\ldots+b^{(k-1)})+b^{(k)}\right)$$

- and *j*-th layers equation:

$$h^{(j)}(\vec{x}) = g^{(j)}\left(W^{(j)}g^{(j-1)}(\vec{x};\ \vec{\theta}^{(j-1)},\ \ b^{(j-1)})+b^{(j)}\right) \qquad j=2,\ldots.,k-1$$

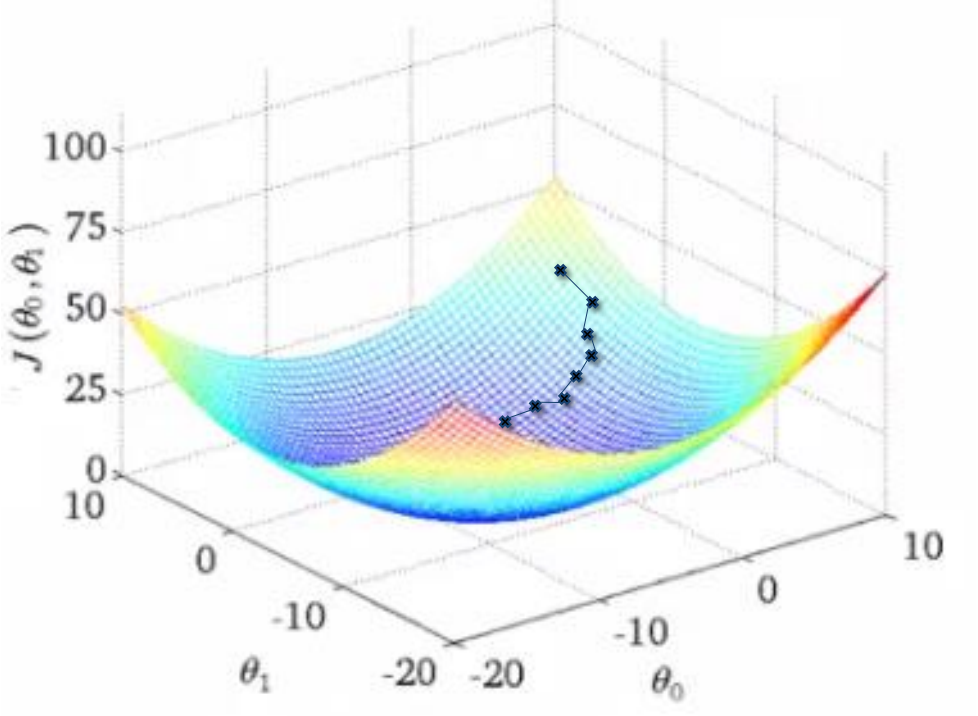


Optimizing J … backwards



Why SGD?

- Weights are updated using the partial derivatives
- Derivative pushes down the cost following the steepest descent path on the error curve



SGD procedure

- Choose an initial random values for θ and b
- Choose a learning rate
- Repeat until stop criterion is met:
  - Pick a random training example $x^{(i)}$
  - Update the parameters with

$$\theta_1 = \theta_1 - \alpha \Delta \theta_1$$
$$\theta_2 = \theta_2 - \alpha \Delta \theta_2$$
$$b = b - \alpha \Delta b$$

- We can stop **WHEN**
  - when the **parameters do not change** or,
  - the **number of iteration exceeds a certain upper bound**

Cost Function Derivative

- In order to update the parameters in SGD, we need to compute the **partial derivatives** *wrt* the learnable parameters.
- Remember the chain rule:
  - if J is a function of a given $z(x)$, then the derivative of J *wrt* x is: $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial x}$
- Thus (in $R^{2)}$, we need to compute
  - for the *i*-th example $x^{(i)}$

$$\Delta\vartheta_1 = \frac{\partial}{\partial\theta_1}(h(x^{(i)};\theta,b) - y^{(i)})^2$$

$$\Delta\vartheta_2 = \frac{\partial}{\partial\theta_2}(h(x^{(i)};\theta,b) - y^{(i)})^2$$

$$\Delta b = \frac{\partial}{\partial b}(h(x^{(i)};\theta,b) - y^{(i)})^2$$

Cost Function Derivatives

$$\Delta\theta_1 = \frac{\partial}{\partial\theta_1}(h(x^{(i)};\theta,b) - y^{(i)})^2 =$$

$$= 2((h(x^{(i)};\theta,b) - y^{(i)})\frac{\partial}{\partial\theta_1}(h(x^{(i)};\theta,b))$$

$$= 2(g(\theta^T x^{(i)} + b) - y^{(i)})\frac{\partial}{\partial\theta_1}(g(\theta^T x^{(i)} + b))$$

We have that:

$$\frac{\partial}{\partial\theta_1}(g(\theta^T x + b)) = \frac{\partial g(\theta^T x + b)}{\partial(\theta^T x + b)}\frac{\partial(\theta^T x + b)}{\partial\theta_1}$$

$$(1 - g(\theta^T x + b))g(\theta^T x + b)\frac{\partial(\theta_1 x_1 + \theta_2 x_2 + b)}{\partial\theta_1}$$

$$= (1 - g(\theta^T x + b))g(\theta^T x + b)x_1$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial g}{\partial z} = (1 - g(z))g(z)$$

$$s(x) = \frac{1}{1+e^{-x}} \quad \textit{then} \quad \frac{\vartheta s}{\vartheta x} = (1-s(x))s(x)$$

$$\frac{d}{dx}s(x) = \frac{d}{dx}((1+e^{-x})^{-1})$$

$$\frac{d}{dx}s(x) = -1((1+e^{-x})^{(-1-1)})\frac{d}{dx}(1+e^{-x})$$

$$\frac{d}{dx}s(x) = -1((1+e^{-x})^{(-2)})(\frac{d}{dx}(1)+\frac{d}{dx}(e^{-x}))$$

$$\frac{d}{dx}s(x) = -1((1+e^{-x})^{(-2)})(0+e^{-x}(\frac{d}{dx}(-x)))$$

$$\frac{d}{dx}s(x) = -1((1+e^{-x})^{(-2)})(e^{-x})(-1)$$

$$\frac{d}{dx}s(x) = ((1+e^{-x})^{(-2)})(e^{-x})$$

$$\frac{d}{dx}s(x) = \frac{1}{(1+e^{-x})^2}(e^{-x})$$

$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})}\frac{1}{(1+e^{-x})}$$

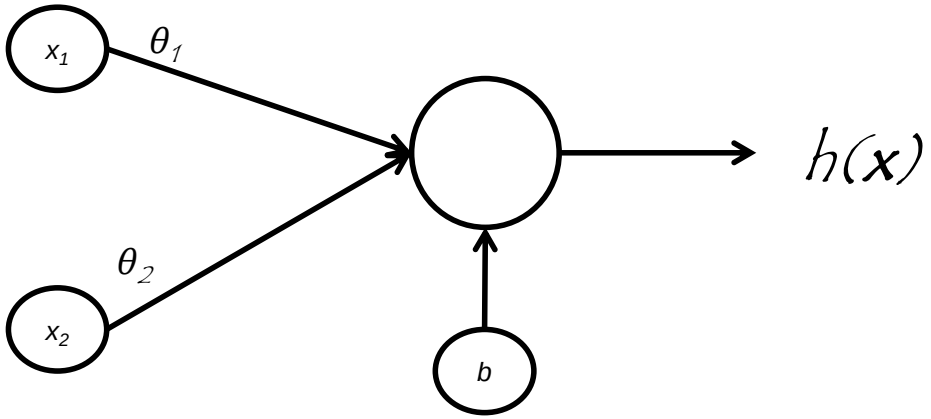
Cost Function Derivatives

Then,

$$\Delta\theta_1 = 2[(g(\theta^T x^{(i)} + b) - y^{(i)})][(1 - g(\theta^T x^{(i)} + b))g(\theta^T x^{(i)} + b)x^{(i)}{}_1]$$

and we can do the same for θ_2

$$\Delta\theta_2 = 2[(g(\theta^T x^{(i)} + b) - y^{(i)})][(1 - g(\theta^T x^{(i)} + b))g(\theta^T x^{(i)} + b)x^{(i)}{}_2]$$



Cost Function Derivatives for *b*

- For the b parameter, the same steps apply:

$$\Delta b = \frac{\partial}{\partial b}(h(x^{(i)};\theta,b) - y^{(i)})^2 =$$

$$= 2((h(x^{(i)};\theta,b) - y^{(i)})\frac{\partial}{\partial b}(h(x^{(i)};\theta,b))$$

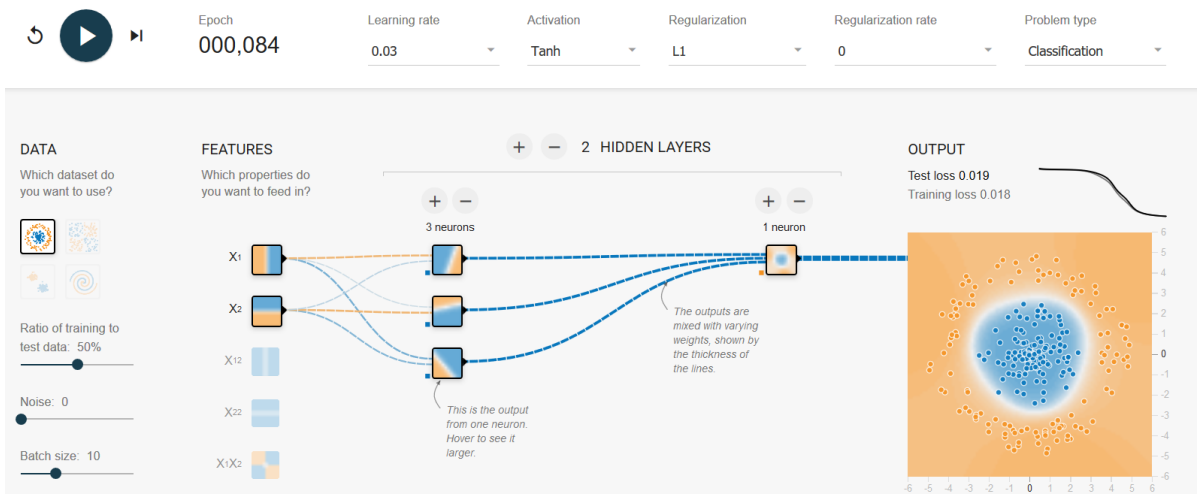
$$= 2(g(\theta^T x^{(i)} + b) - y^{(i)})\frac{\partial}{\partial b}(g(\theta^T x^{(i)} + b))$$

$$\frac{\partial}{\partial b}(g(\theta^T x + b)) = \frac{\partial g(\theta^T x + b)}{\partial(\theta^T x + b)}\frac{\partial(\theta^T x + b)}{\partial b} = (1 - g(\theta^T x + b))g(\theta^T x + b)$$

$$\Delta b = 2[(g(\theta^T x^{(i)} + b) - y^{(i)})][(1 - g(\theta^T x^{(i)} + b))g(\theta^T x^{(i)} + b)]$$

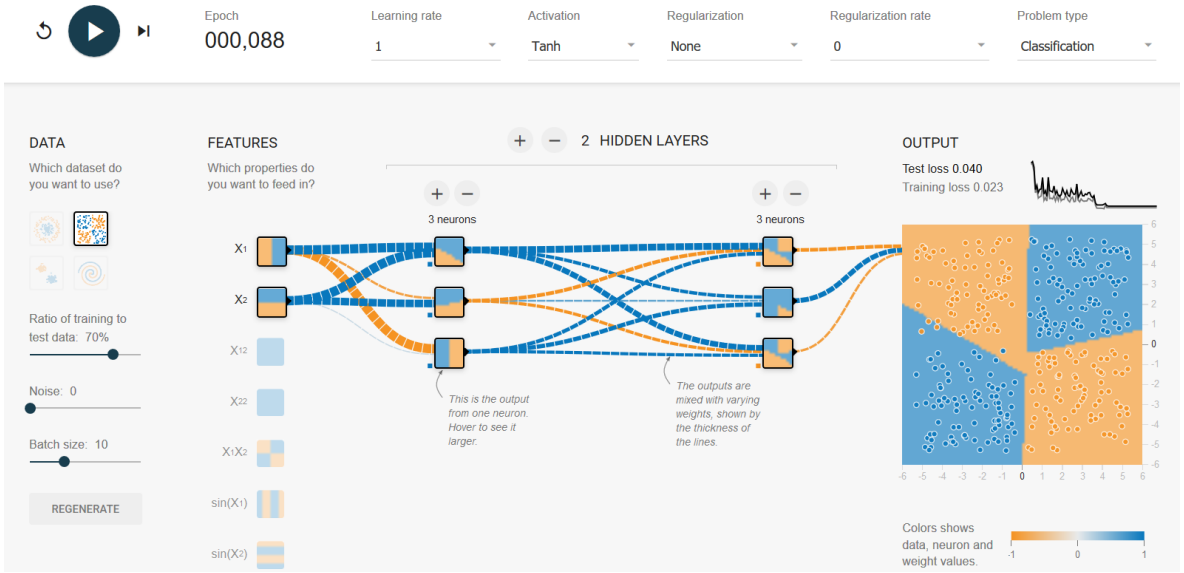
A simple demo on TensorFlow

- Look at: https://playground.tensorflow.org/



A simple demo on TensorFlow

- Look at: https://playground.tensorflow.org/

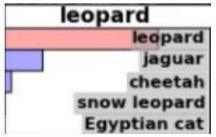


ANNS: APPLICATIONS

An Example: Neural Image Classification

Goal

Classification



ImageNet

- Over 15M labeled high resolution images
- Roughly 22K categories
- Collected from web and labeled by Amazon Mechanical Turk

http://image-net.org/

IMAGENET

14,197,122 images, 21841 synsets indexed

Home Download Challenges About

Not logged in. Login | Signup

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

https://www.image-net.org/challenges/LSVRC/

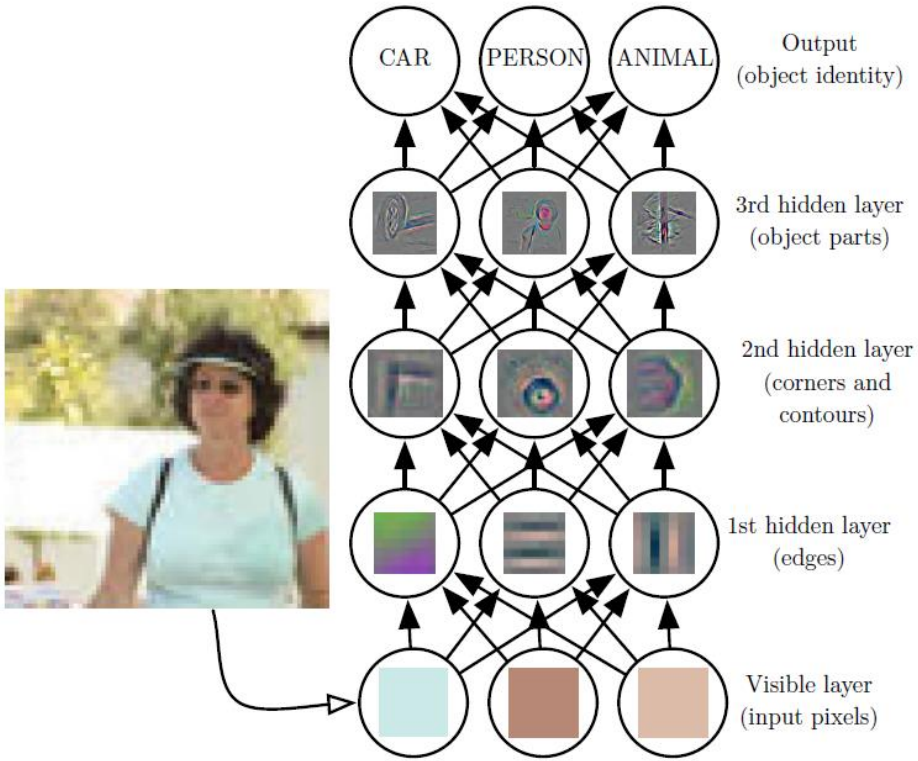
- Annual competition of image classification at large scale
- 1.2M images in 1K categories
- Classification: make 5 guesses about the image label

EntleBucher

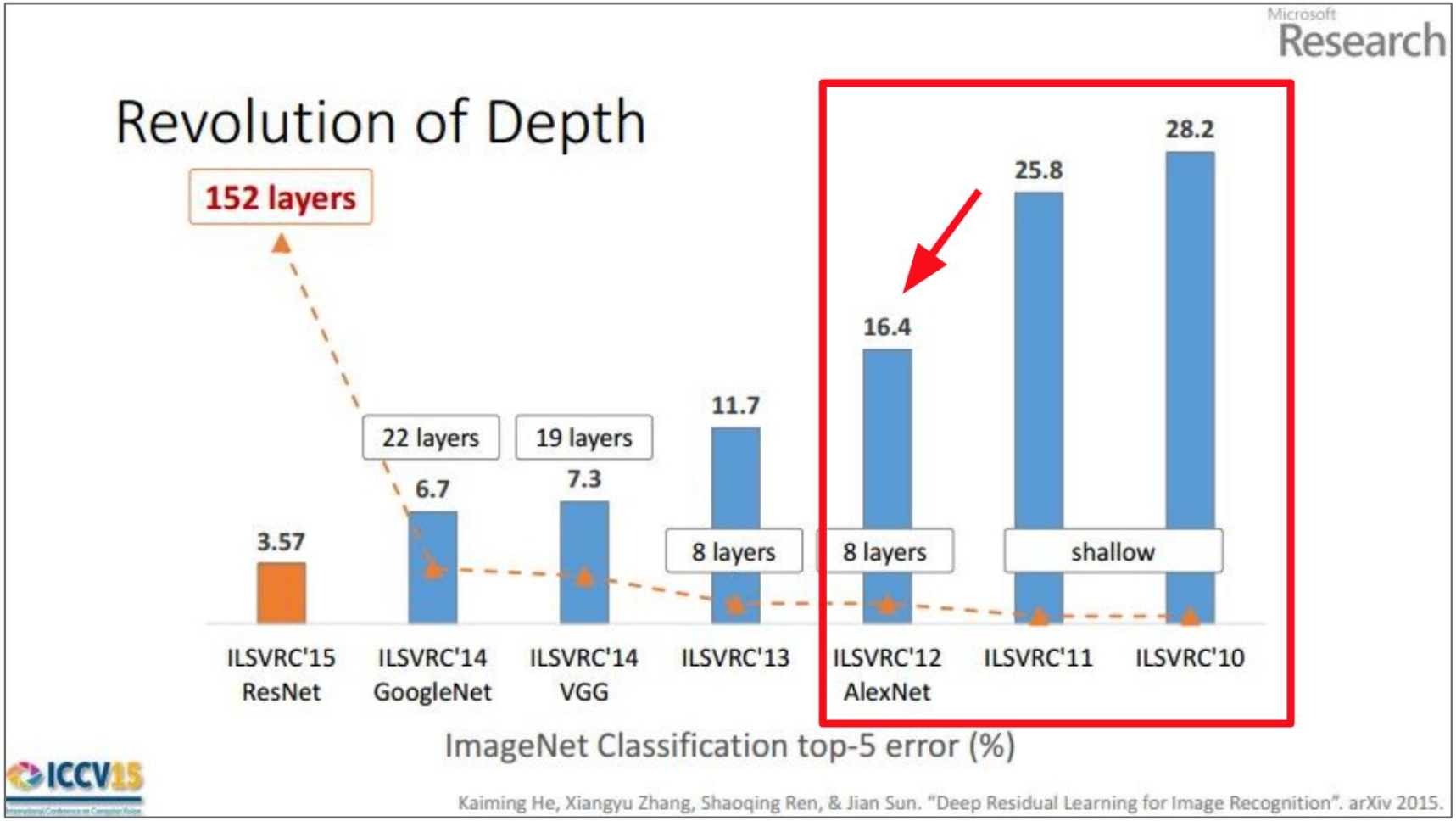
Appenzeller

Razza: Bovaro del bernese

The role of NN Depth in Representation Learning



Zeiler and Fergus (2014)



(slide from Kaiming He's recent presentation)

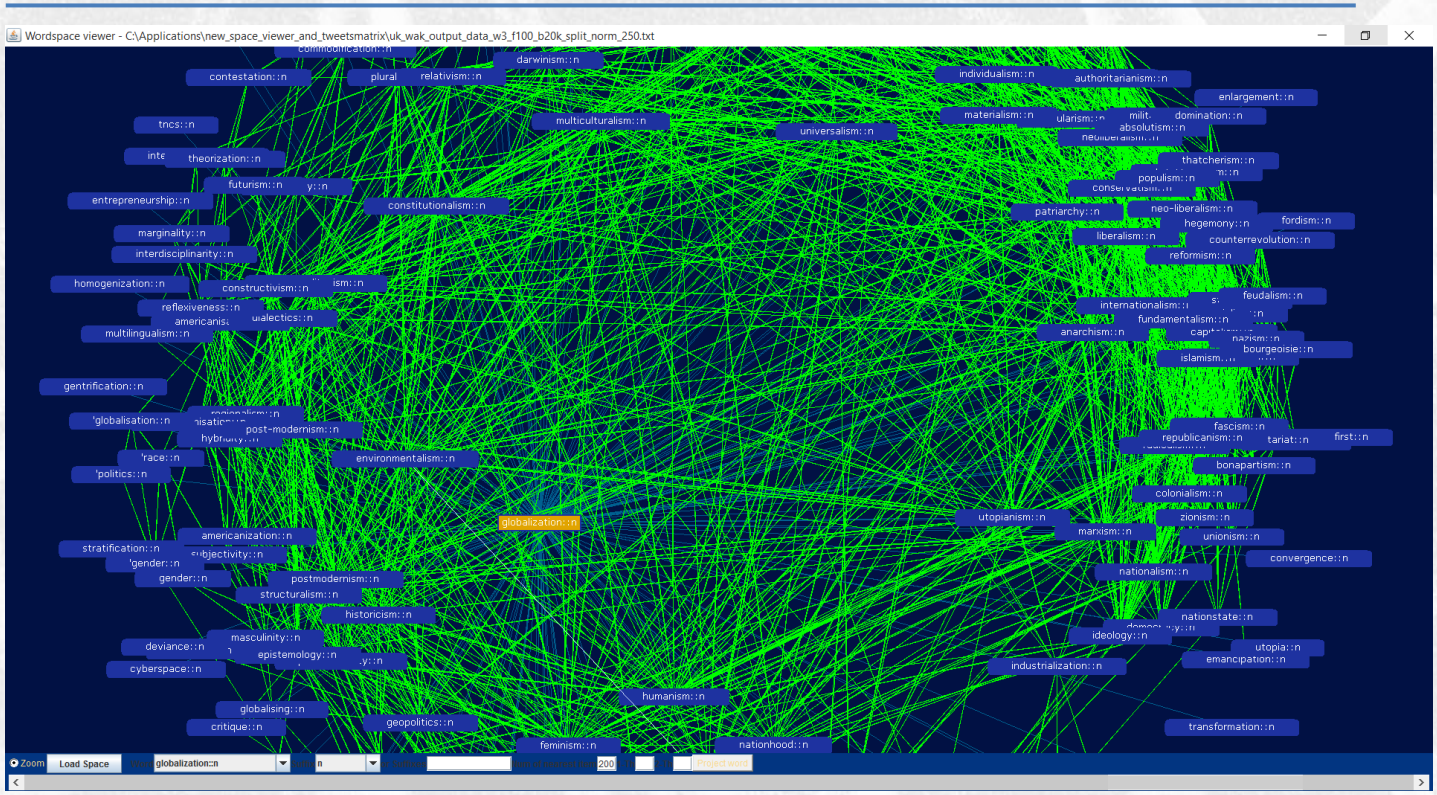
Applicazioni delle reti neurali

- Riconoscimento dei fenomeni linguistici
  - Classificazione dei testi (e.g. tweet in classi tematiche)
  - Sentiment Analysis delle Review
  - Traduzione Automatica
  - Language Models Acquisizione Automatica di Lessici Semantici da testi (non annotati)
  - Modelli vettoriali della semantica lessicale
  - Sistemi di pre-addestramento per la inizializzazione dei classificatori *supervised*
- Riconoscimento di oggetti o di pattern (ad es. emozioni dai visi) nelle immagini
- Acquisizione integrata di conoscenza da immagini, video e testi scritti
- Sistemi di Question Answering o Dialogo su immagini
  - Ad es. Chat GPT di OpenAI

Face Recognition

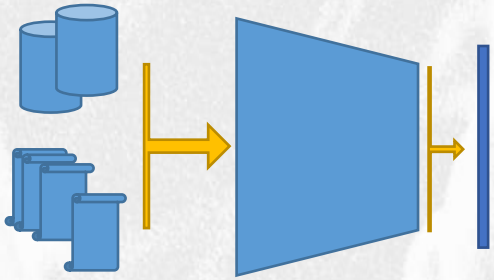
Roma, il "cervellone" che ha scovato

Knowledge Acquisition & Lexical Embeddings con le Reti Neurali



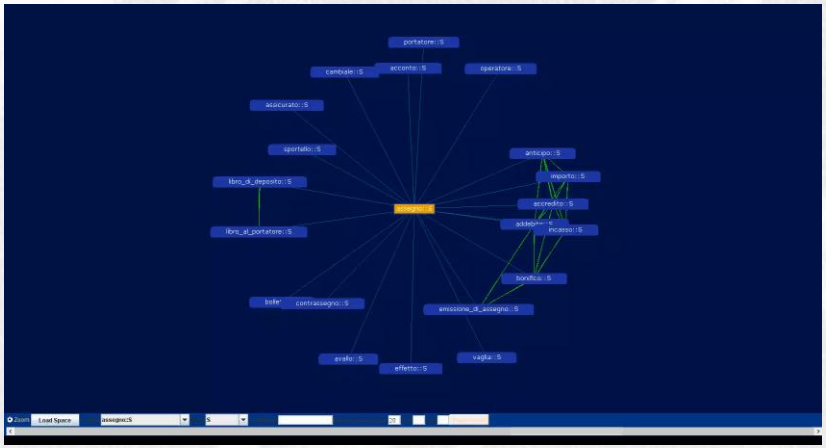
Wordspaces: Encoding & Domain Corpora

(No Use of Annotated Examples)

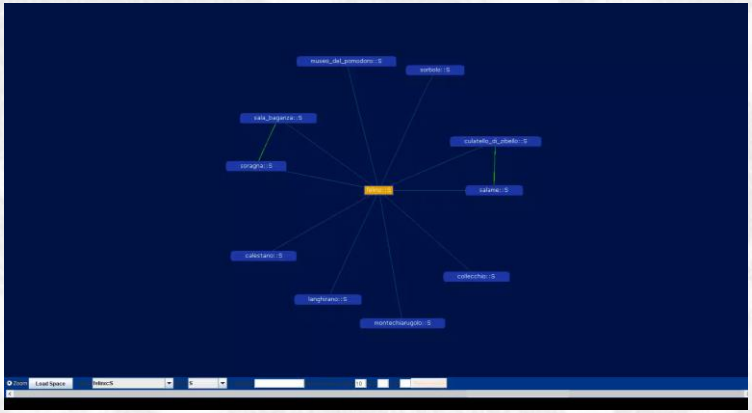


Encoding: vector-based lexicon

Monte Paschi Siena

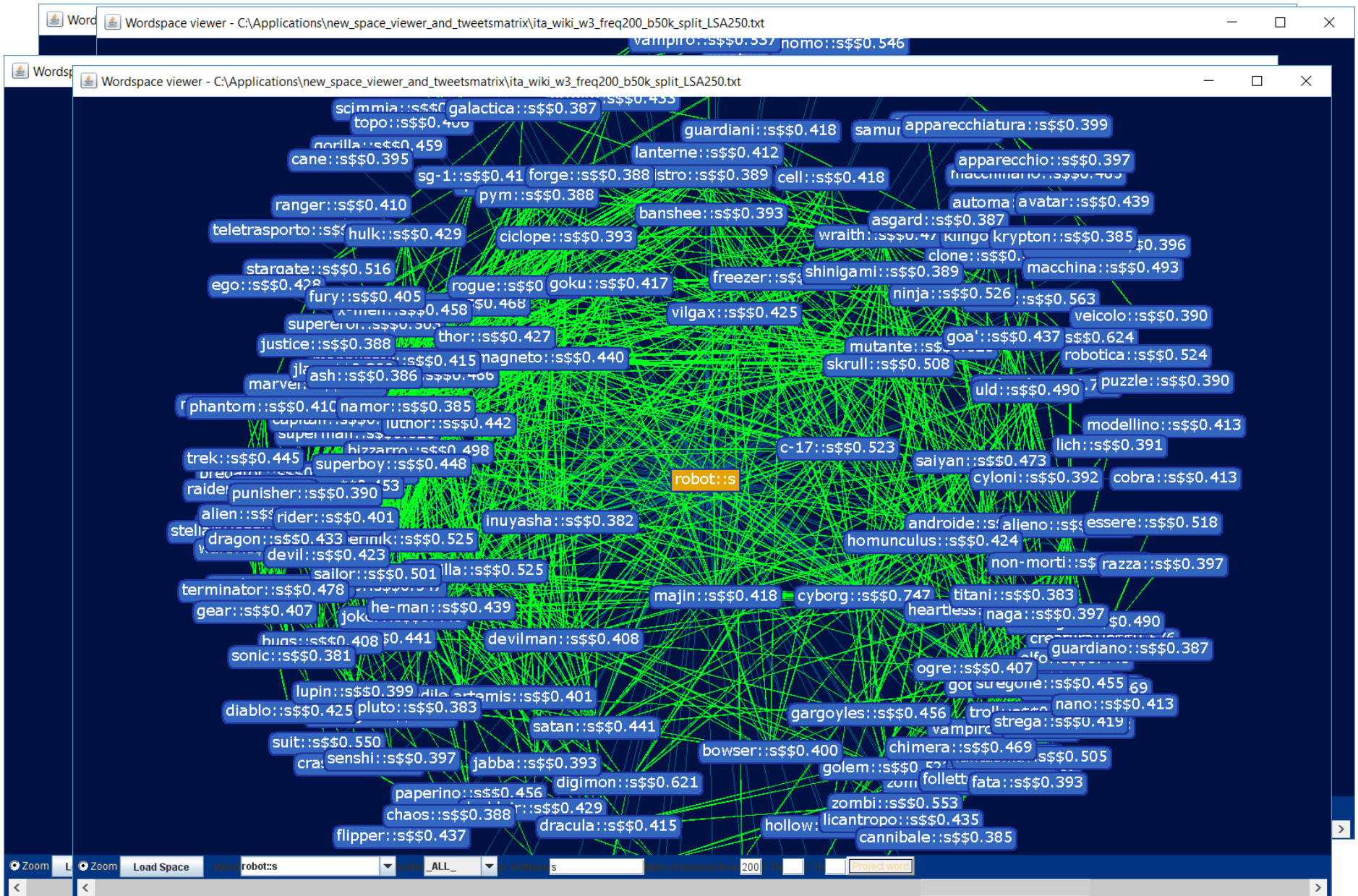


Parma



AkerSolution
(Subsea oil)

Leggere dal Web



ANNs: ADVANCED ARCHITECTURES

Recurrent Neural Networks

For example, consider the classical form of a dynamical system:

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}), \tag{10.1}$$

where $\boldsymbol{s}^{(t)}$ is called the state of the system.

Equation 10.1 is recurrent because the definition of $\boldsymbol{s}$ at time t refers back to the same definition at time $t-1$.

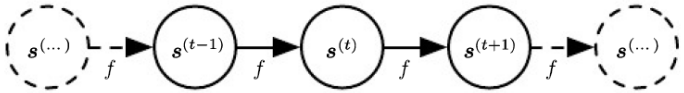
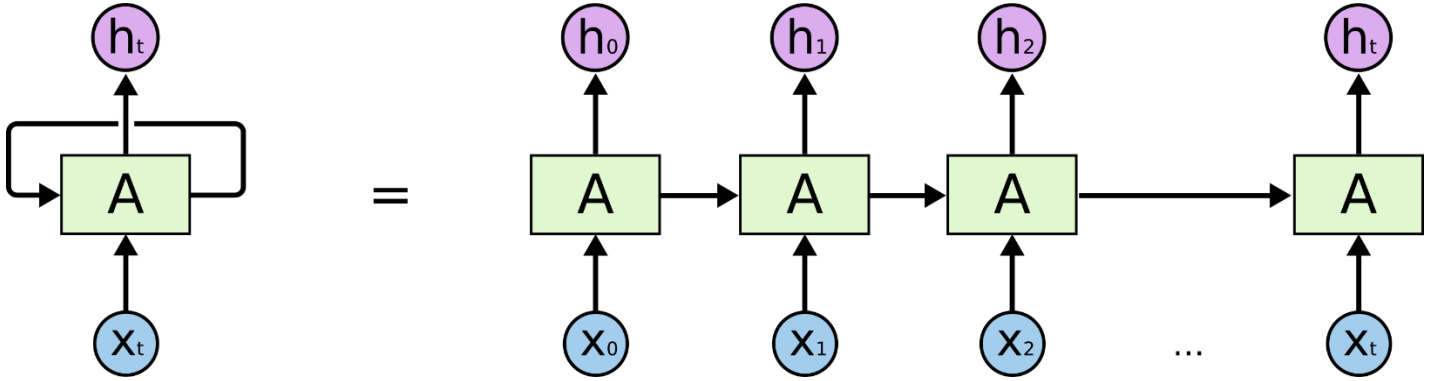


Figure 10.1: The classical dynamical system described by equation 10.1, illustrated as an unfolded computational graph. Each node represents the state at some time t, and the function f maps the state at t to the state at $t+1$. The same parameters (the same value of $\boldsymbol{\theta}$ used to parametrize f) are used for all time steps.



Types of RNNs

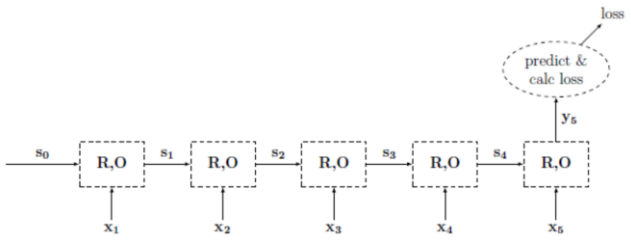


Figure 7: Acceptor RNN Training Graph.

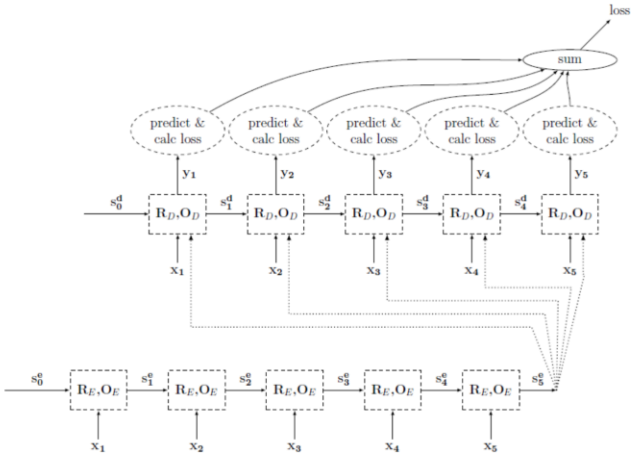


Figure 9: Encoder-Decoder RNN Training Graph.

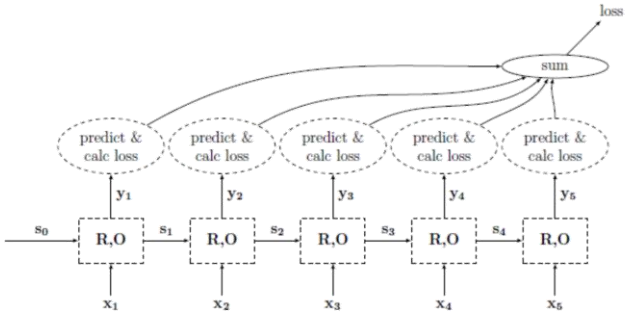


Figure 8: Transducer RNN Training Graph.

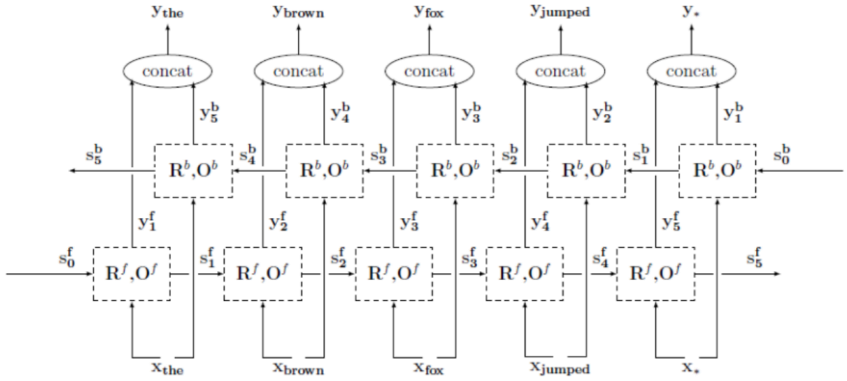


Figure 11: biRNN over the sentence “the brown fox jumped .”.

Examples: Language understanding

https://github.com/Microsoft/CNTK/wiki/Hands-On-Labs-Language-Understanding

-

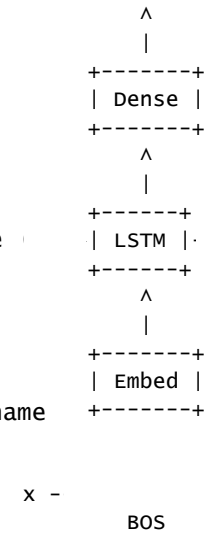
```
# BOS                   |# O
# show                  |# O
# flights               |# O
# from                  |# O
# burbank               |# B-fromloc.city_name
# to                    |# O
# st.                   |# B-toloc.city_name
# louis                 |# I-toloc.city_name
# on                    |# O
# monday                |# B-depart_date.day_name
# EOS                   |# O
```

Examples: language understanding

https://github.com/Microsoft/CNTK/wiki/Hands-On-Labs-Language-Understanding

Task: Slot tagging with an LSTM

```
19  |x 178:1 |# BOS       |y 128:1 |# O
19  |x 770:1 |# show      |y 128:1 |# O
19  |x 429:1 |# flights   |y 128:1 |# O
19  |x 444:1 |# from      |y 128:1 |# O
19  |x 272:1 |# burbank   |y 48:1  |# B-fromloc.city_name
19  |x 851:1 |# to        |y 128:1 |# O
19  |x 789:1 |# st.       |y 78:1  |# B-toloc.city_name
19  |x 564:1 |# louis     |y 125:1 |# I-toloc.city_name
19  |x 654:1 |# on        |y 128:1 |# O
19  |x 601:1 |# monday    |y 26:1  |# B-depart_date.day_name
19  |x 179:1 |# EOS       |y 128:1 |# O
```

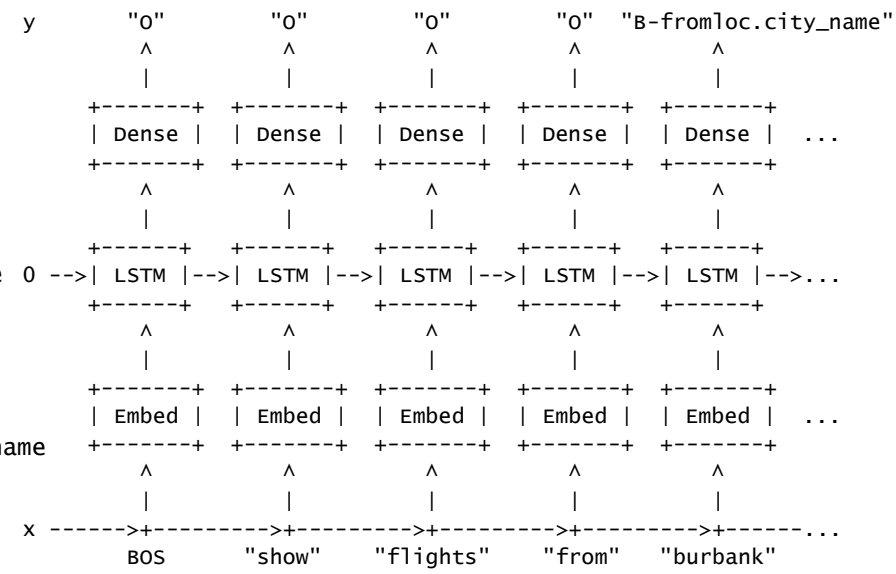


Examples: language understanding

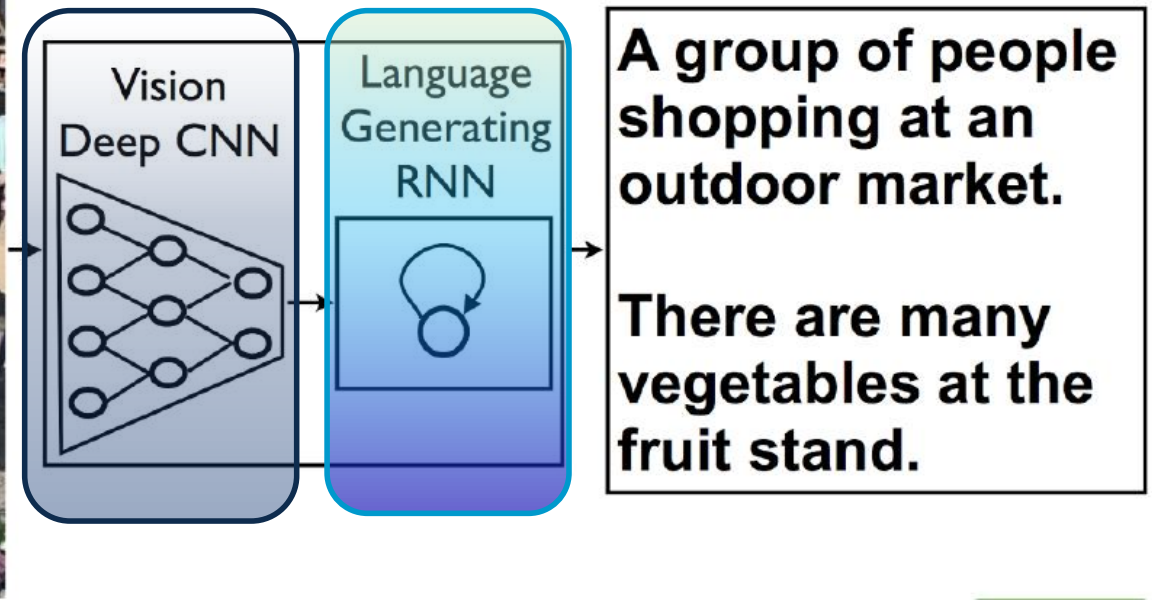
https://github.com/Microsoft/CNTK/wiki/Hands-On-Labs-Language-Understanding

Task: Slot tagging with an LSTM

```
19  |x 178:1 |# BOS       |y 128:1 |# O
19  |x 770:1 |# show      |y 128:1 |# O
19  |x 429:1 |# flights   |y 128:1 |# O
19  |x 444:1 |# from      |y 128:1 |# O
19  |x 272:1 |# burbank   |y 48:1  |# B-fromloc.city_name
19  |x 851:1 |# to        |y 128:1 |# O
19  |x 789:1 |# st.       |y 78:1  |# B-toloc.city_name
19  |x 564:1 |# louis     |y 125:1 |# I-toloc.city_name
19  |x 654:1 |# on        |y 128:1 |# O
19  |x 601:1 |# monday    |y 26:1  |# B-depart_date.day_name
19  |x 179:1 |# EOS       |y 128:1 |# O
```



Automatic image captioning: neural networks *at work*



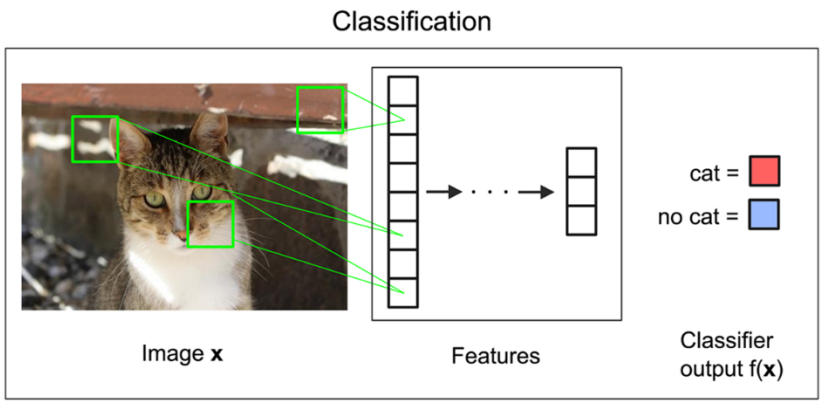
… training a neural network for italian

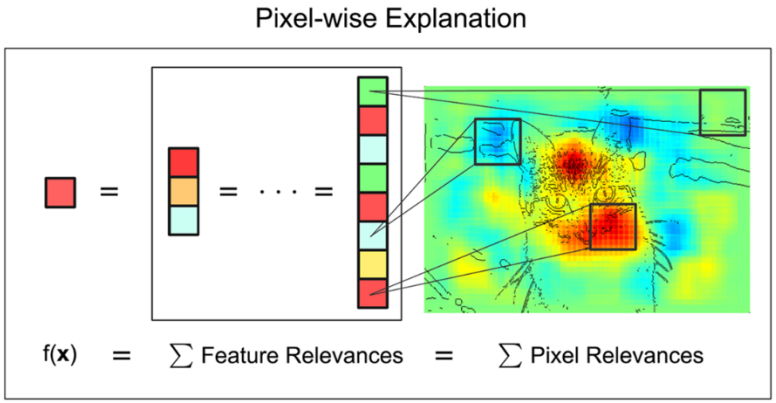
Uno scuolabus giallo parcheggiato sul lato della strada.

Un uomo che cavalca un cavallo su una strada cittadina.

Un segnale di stop che si siede su un angolo di strada.

Explainability: decisions vs. Activation states



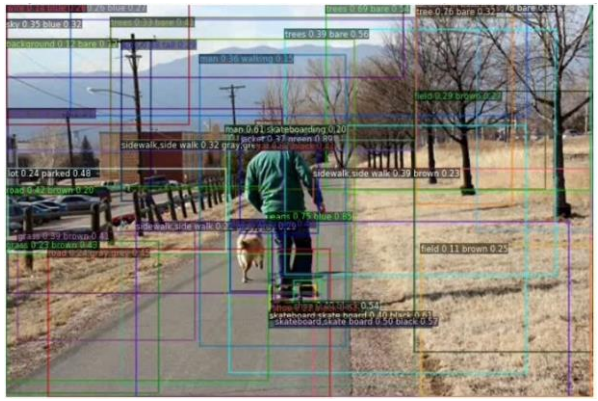


A(0.99)

A(1.00)

Visual QA

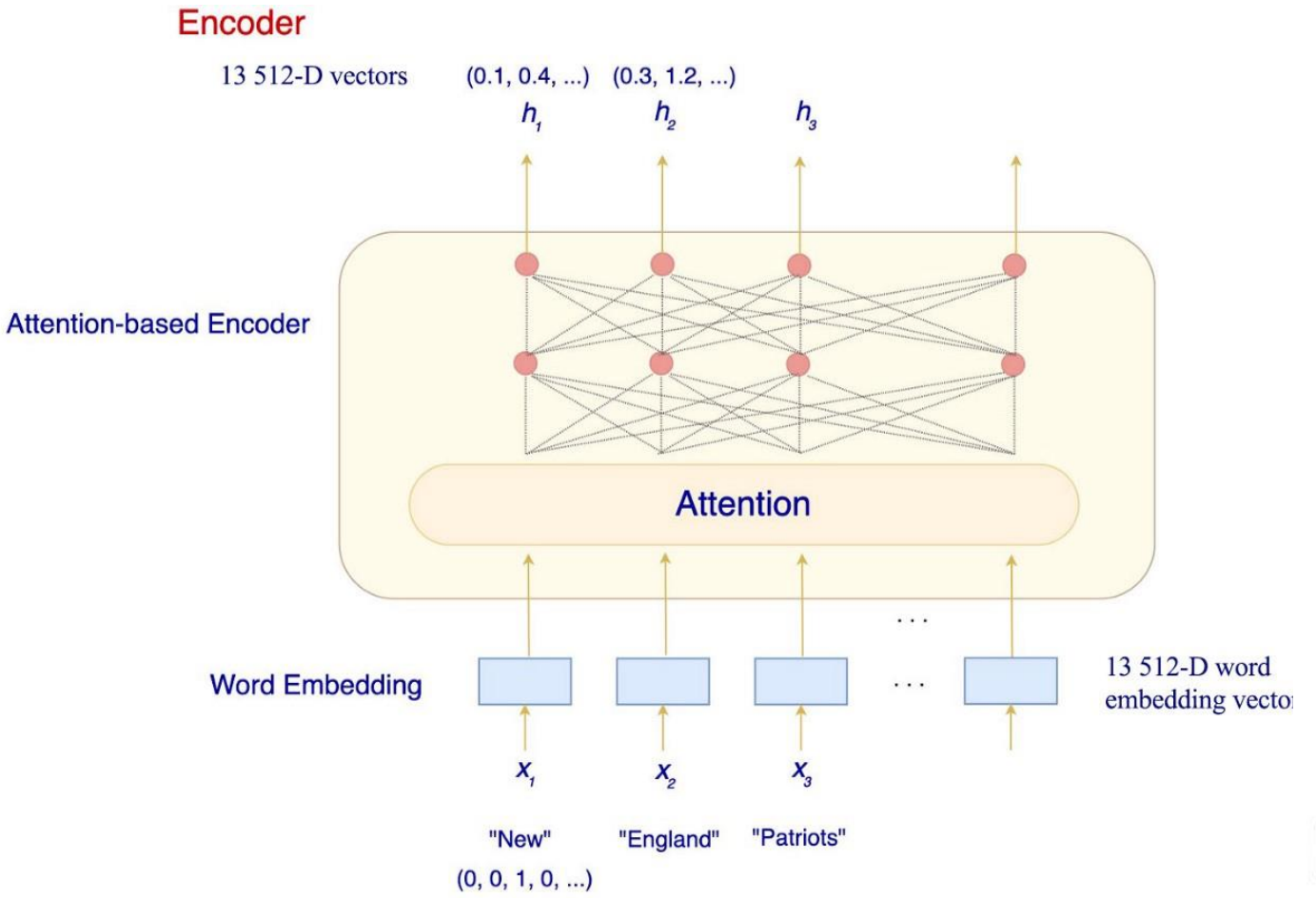
- Task:
  - Data una immagine che descrive una situazione o un evento rispondere a domande in LN verso i contenuti dell'immagine, basata sulla comprensione delle relazioni spaziali e semantiche tra gli elementi concettuali dell'immagine
- Tecnologia:
  - Convolutional Autoencoders, Recurrent neural networks and trasformer-based inferences
  - Two languages: ITA/ENG
  - ….

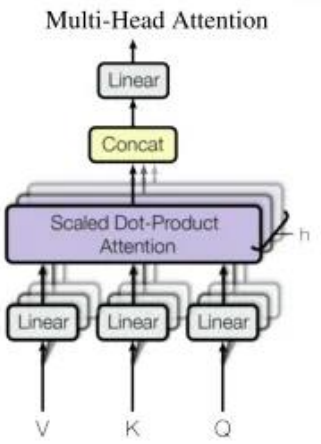


- Demo: VQA in italiano ed inglese: Demo_GQA.mp4

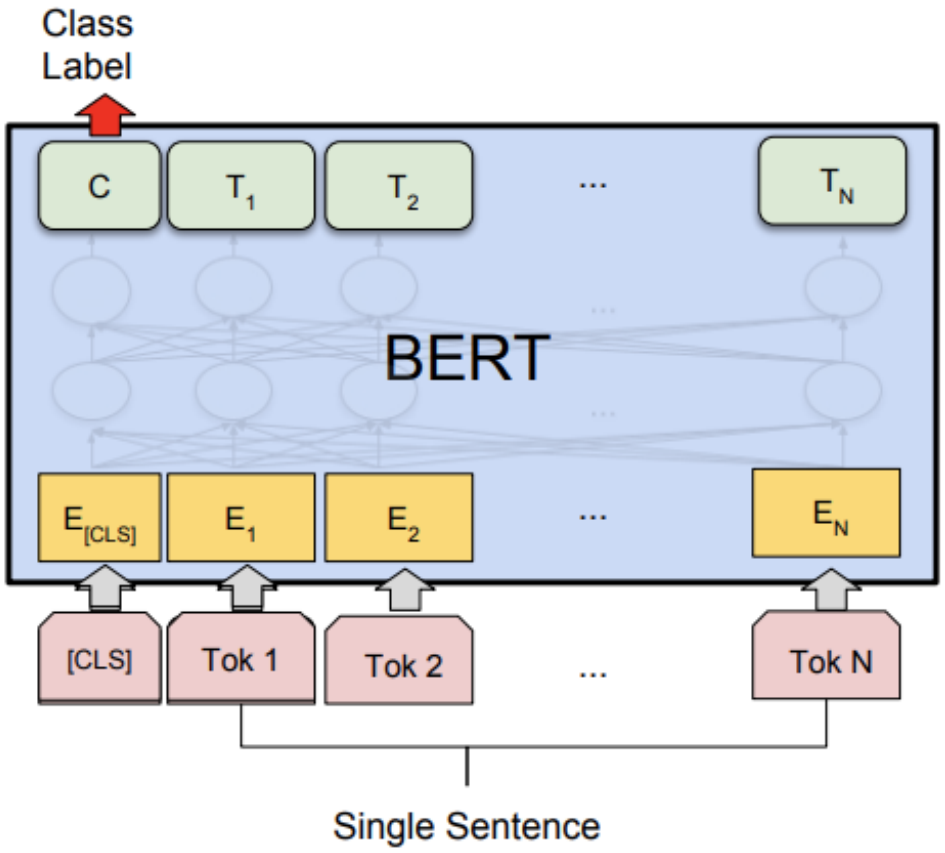
Trasformers

BERT & NLP


Encoder
13 512-D vectors
(0.1, 0.4, ...)
(0.3, 1.2, ...)
h_1
h_2
h_3
Attention-based Encoder
Attention
Word Embedding
13 512-D word embedding vecto
x_1
x_2
x_3
"New"
(0, 0, 1, 0, ...)
"England"
"Patriots"

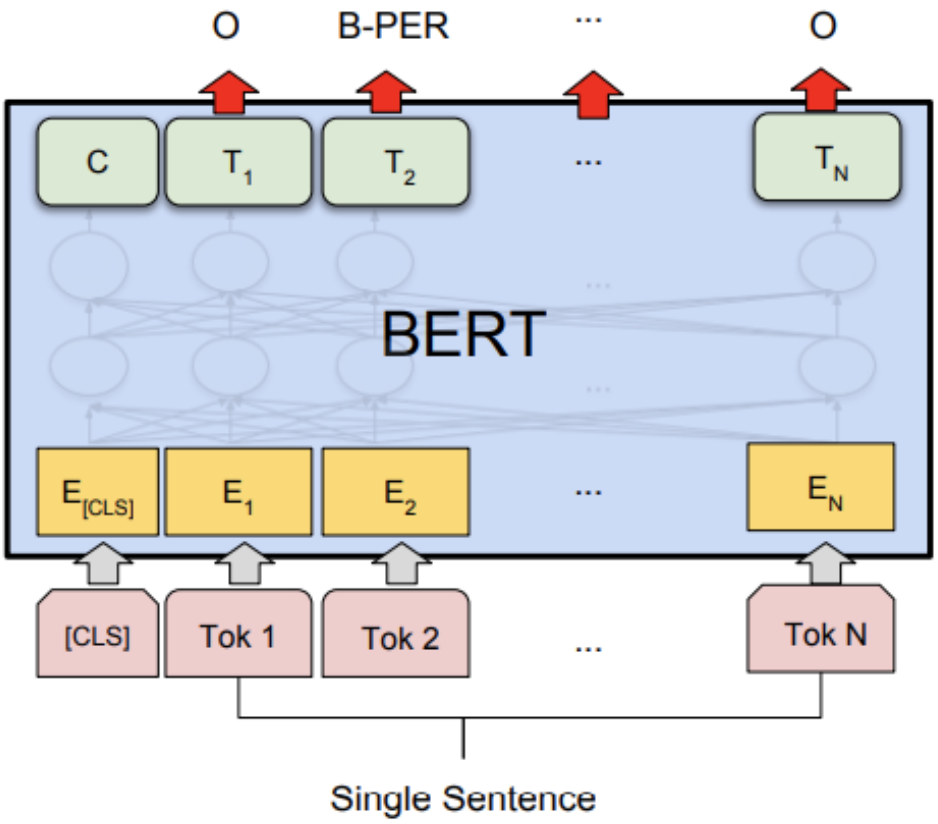

Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

BERT (Devlin et al. '18)



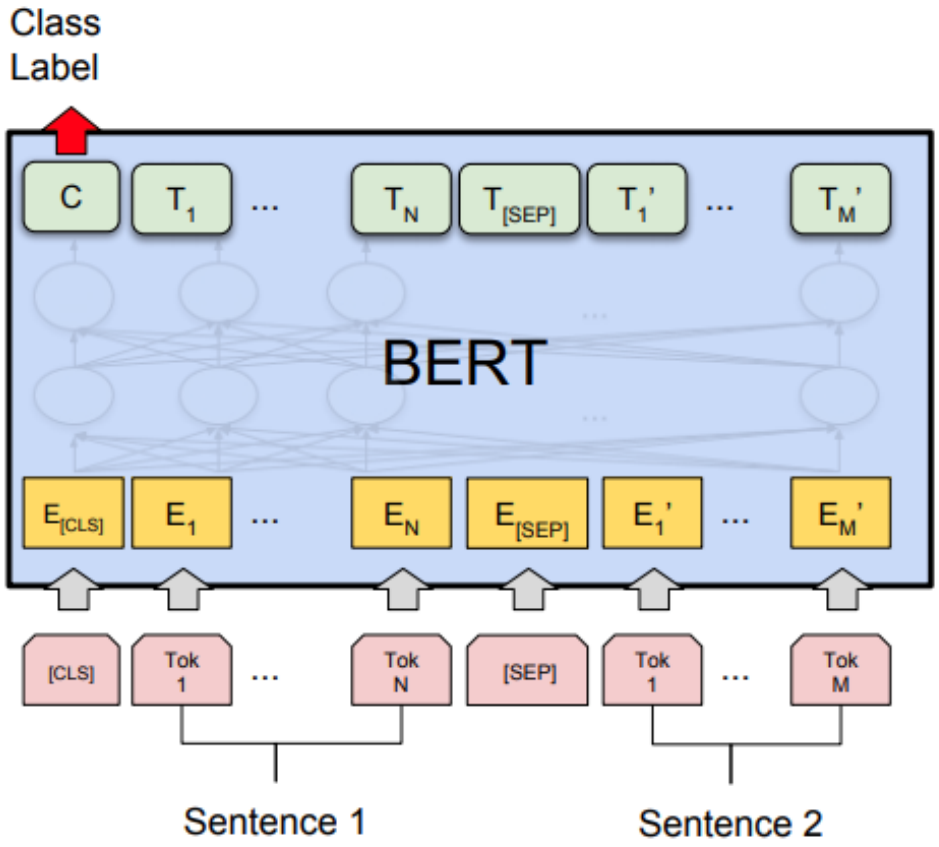
BERT for single sentence classification (Sentiment analysis, Intent Classification, etc.)

BERT (Devlin et al. '18)



BERT for Sequence Tagging Tasks (e.g., POS tagging, Named Entity Recognition, etc.)

BERT (Devlin et al. ’18)



BERT for sentence pairs classification (Paraphrase Identification, answer selection in QA, Recognizing Textual Entailment)

Relationship with other areas of AI

- Neural networks for **faster inference over large knowledge bases** wrt to logical approaches
- Neural networks for **high quality and cost-effective complex tasks**
  - Pattern recognition
  - Language and Image/Video processing
  - Complex rewriting tasks, e.g. Machine Translation
- **Learning to acquire knowledge**
  - Machine Reading for QA
  - Fast Indexing an retrieval from large document bases or Web sources
- Processing **Time** Series
  - Predictive Analuìytics tasks that depend on time
  - Sequence labeling tasks
- Complex inference on ***hybrid*** (i.e structured and unstructured) ***data***

Riferimenti Bibliografici

- *AIMA,* Chapter 18: 18.16.1, 18.3-4, 18.7

- Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press book, 2016