

Notes on Neural Networks: part 2

Alessio Micheli

micheli@di.unipi.it

2025



Dipartimento di Informatica
Università di Pisa - Italy

ML Course structure

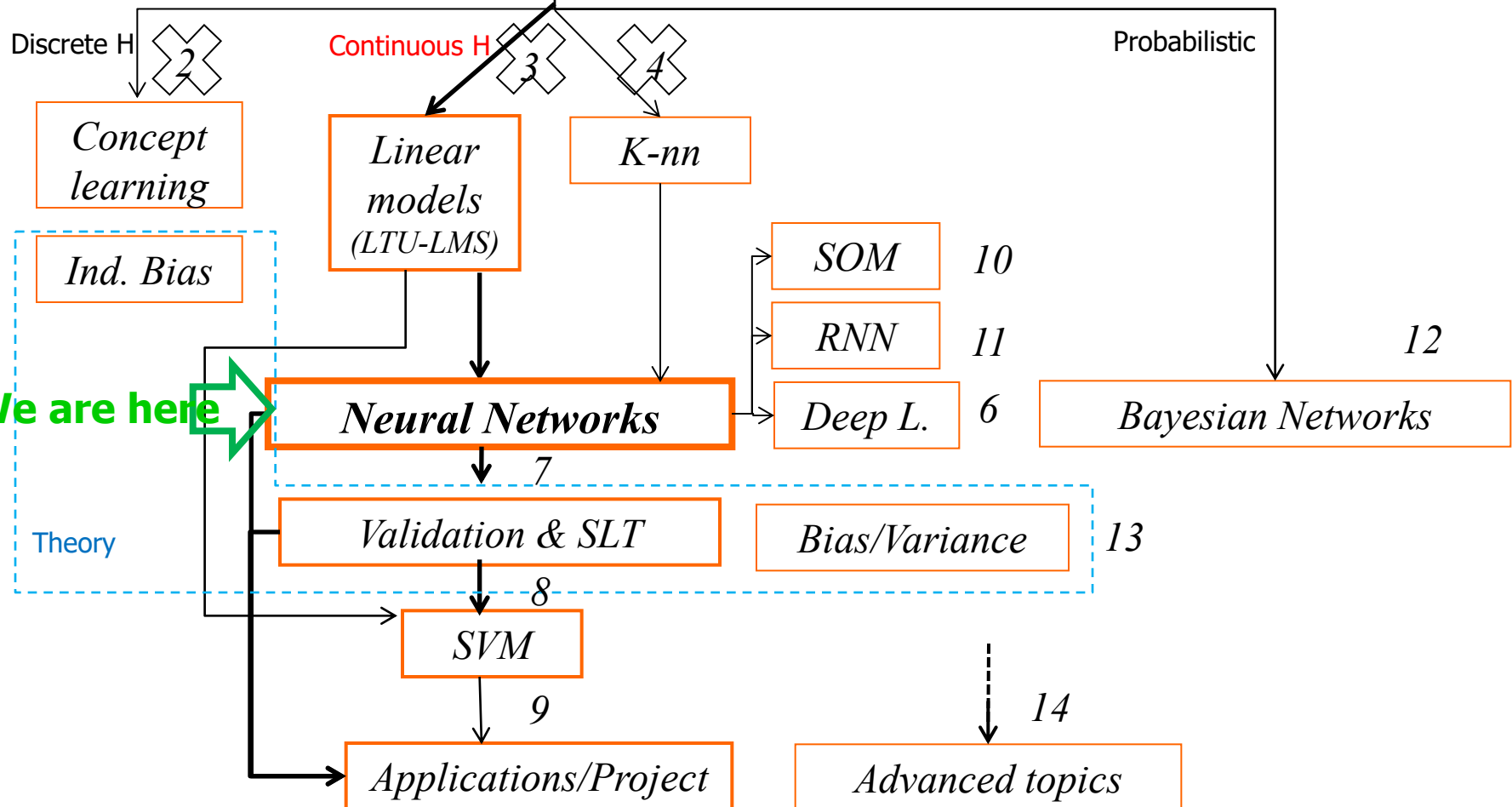
Where we are



Dip. Informatica
University of Pisa

1 **INTRO**

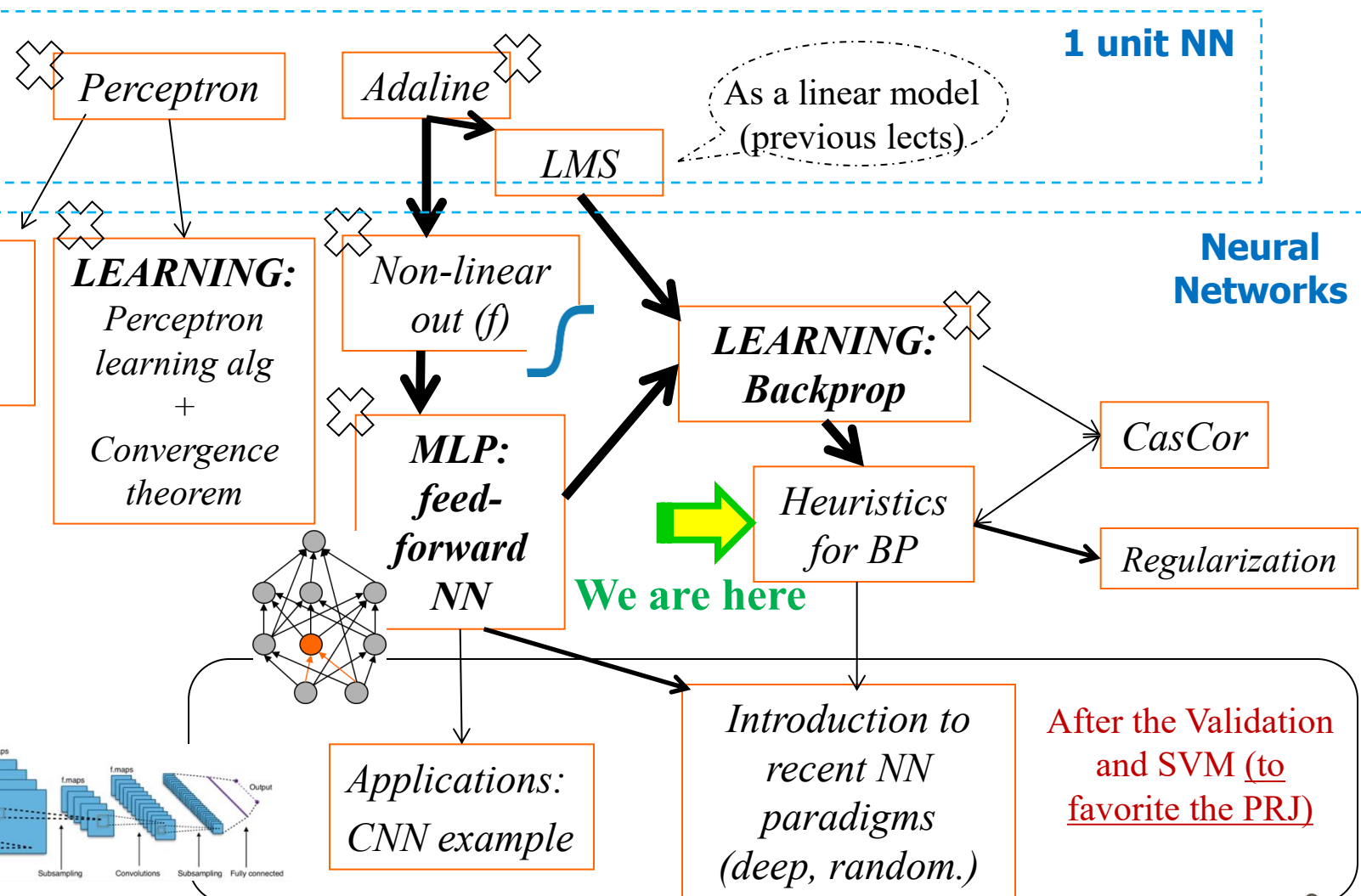
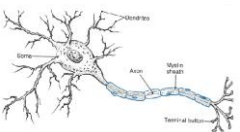
Function approximation framework
Data, Task, Model, Learn. Alg., Validation



Course structure: zoom in the **Neural Networks** box



Dip. Informatica
University of Pisa



Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, learning rate
 - Multiple Minima
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

Hyper-parameters=
values that **you** have to
set-up to run the training

Philosophy

- We don't like to introduce too much heuristic details
 - Historically: From a collection of heuristic methods (often collections of tricks originating from the contributing fields of statistics, neural networks, fuzzy logics, ...) to the construction of general conceptual frameworks ...
- They can be experimental investigated in the **project** !
- Some of them have a *theoretical ground* (guess which one!) and most of them can be applied in a more general framework (e.g. regularization)
- *Some are very common & must be used (in the prj) see the symbol → (!)*
- Remember also the symbol for tech arguments / *important topics, fast in class, because* not strictly need at a first glance, but **useful** for your successive reading (e.g. because they are easier to be understood *later*, or useful to develop the *project*, etc)



- *Avoid rules of thumb from **blogs** (unless proved in scientific papers or with an improvement empirically assessed by yourself)*

Nice interpretation

- Back-prop as a **path** trough the loss/weight space.
- The path depends on:
 - The data (assumed already given)
 - The NN model
 - The starting point (initial weight values)
 - The rate of convergence
 - The final point (stopping rule)
- They define a control for the *search* over the Hypothesis space

The Basic Alg. (**AGAIN!!!**)



Dip. Informatica
University of Pisa

Gradient descent: Given training data in the form (x, d)

- 1) Start with weight vector $\mathbf{w}_{\text{initial}}$ (small), fix η ($0 < \eta < 1$).
- 2) Compute $\Delta \mathbf{w} = -$ “gradient of $E(\mathbf{w})$ ” $= - \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ (or for each w_j)
- 3) Compute $\mathbf{w}_{\text{new}} = \mathbf{w} + \eta * \Delta \mathbf{w}$ (or for each w_j) [upgrade weights]
where η is the “step size” (learning rate) parameter
- 4) Repeat (2) until convergence or $E(\mathbf{w})$ is “sufficiently small”

- But now $\Delta \mathbf{w} = - \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ were obtained through **backpropagation** derivation/algorithm for any weights in the network
 - At step 2) to compute the Error we first apply inputs to the network computing a **forward phase** up to the outputs computation
 - then we start the **backward phase** to retro-propagate the deltas for the gradient
- See “back-prop” algorithm in (the last page of) the Back-prop pdf file
 - computing the delta for each unit in the network from the output layer to the hidden layers and then the Δw for each weight,
 - and summing over the patterns: we will see how soon!

The Basic Alg. (**AGAIN!!!**)

Gradient descent: Given training data in the form (x, d)

- 1) Start with weight vector $\mathbf{w}_{\text{initial}}$ (small), fix η ($0 < \eta < 1$).
- 2) Compute $\Delta \mathbf{w} = -$ “gradient of $E(\mathbf{w})$ ” $= - \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ (or for each w_j)
- 3) Compute $\mathbf{w}_{\text{new}} = \mathbf{w} + \eta * \Delta \mathbf{w}$ (or for each w_j) [upgrade weights]
where η is the “step size” (learning rate) parameter
- 4) Repeat (2) until convergence or $E(\mathbf{w})$ is “sufficiently small”

E.g. How to choice for the and other aspects of the training algorithm for MLP by back-prop?

Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, learning rate
 - Multiple Minima
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

Hyper-parameters=
values that **you** have to
set-up to run the training

Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, learning rate
 - Multiple Minima
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

Starting values (w_{initial} in the basic alg.)

- (!) Initialize weights by random values near zero
 - Avoid for W : all zero, high values (e.g. saturation issue), or all equals (symmetry): these can hamper the training
 - E.g. in the range $[-0.7, +0.7]$ (for standardized data, see later)
- Also: considering the Fan-in : e.g. range $\times 2/\text{fan-in}$
 - Not if fan-in is too large
 - Not for output units (else deltas start close to zero !)

Fan-in is the number of inputs to a (hidden) unit.
- Other heuristics ...orthogonal matrix, (enjoy if you like)

Very popular among practitioners: Glorot, Bengio AISTATS 2010:

 - **Basic**: 0 for biases and w random from Uniform distribution in $[-1/a, +1/a]$ with $a = \sqrt{\text{fan-in}}$ or (later results) a depending also on fan-out
 - Glorot, Bengio (**New proposed**): see inside the paper.
- (!) Note (practical for the first part of the project): a very small NN (very few units) can have a stronger dependencies w.r.t. initialization



Issues in Training NN

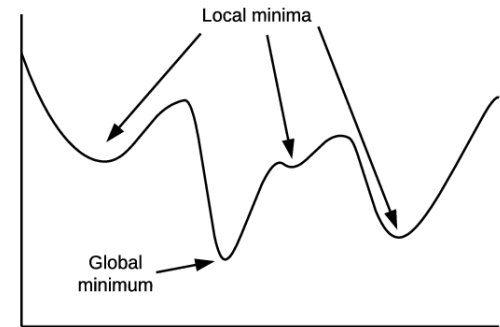
- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, learning rate
 - **Multiple Minima**
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

Multiple Minima

- Loss is not convex: many *local minima*;
Effects:



- The result depends on starting weight values, hence:
- (!)** Try a number of random starting configurations (e.g. **5-10** or more training runs or **trials**)
- Useful for the project: Take the **mean results** (mean of errors) and look to **variance** to evaluate your model and then, if you like to have only 1 response:
 - you can choose the solution giving lowest (penalized) validation error (or the median)
 - or you can take advantage of different end points by an average (committee) response: mean of the outputs/voting
 - A lecture of the course will explain why committee/ensembling approaches can be advantageous in general (later)
- Local minima stopping training with too high error is not really a big issue:
 - you can see the final training error (and restart if you are not happy)
 - a “good” local minima is often sufficient (see the next slide)



But is global minima needed?

Two general observations:

1. Indeed, in ML we don't need neither local or global minima on R_{emp} as we are searching the min of R (that we cannot compute)
 - Often we stop * in a point that has even *not* null gradient and hence it is also neither a local or global minima for the empirical (training) error.
*(we will see later how)
2. The NN build a *variable size hp space* (see at the end of NN-first part slides) → it increases VC-dim during training → the training error decreases toward zero (or global minimum) while the NN becomes too complex (high VC-dim) [we have a major problem to avoid it]
- Stopping before this condition of *overtraining* (and hence overfitting) is needed
(independently from the issue on local/global minima)

Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, learning rate
 - Multiple Minima
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)



On-line/Batch

- See batch/on-line in “Gradient descent algorithm” slide, lecture on linear models!!!

Repetita

- For **Batch** version: sum up all the gradients of each pattern over an epoch and then update the weights (Δw after each “epoch” of l patterns)
 - Memento: “*Epoch*”: an entire cycle of training patterns presentation (all the training examples)
- **Stochastic/on-line** version: upgrade w for each pattern p (by $\Delta_p w$) without wait the total sum over l :
 - It makes progress with each examples it looks at: it can be faster, but it need smaller *eta*.
- Let see the two algorithms in details:

On-line/Batch Algorithms



Dip. Informatica
University of Pisa

- See batch/on-line in “Gradient descent algorithm” slide, lecture on linear models!!!
- Steps 2 and 3 change to express an *inner* or *external* cycle on the patterns:

Batch (as before)

- 1) Fix $\mathbf{w}_{\text{initial}}$ and η
- 2) Compute $\Delta \mathbf{w} = -$ “gradient of $E(\mathbf{w})$ ” on the entire Training set (a cycle, epoch, i.e. a sum, on p is inside this step)
- 3) Compute $\mathbf{w}_{\text{new}} = \mathbf{w} + \eta * \Delta \mathbf{w}$
- 4) Repeat until convergence

On-line

- 1) Fix $\mathbf{w}_{\text{initial}}$ and η
- For each pattern p :
 - 2) Compute $\Delta_p \mathbf{w} = -$ “gradient of $E_p(\mathbf{w})$ ”
 - 3) Compute $\mathbf{w}_{\text{new}} = \mathbf{w} + \eta * \Delta_p \mathbf{w}$
- 4) Repeat until convergence*

consider also
epoch wise
repetition

$$-\frac{\partial E(\mathbf{w})}{\partial w_{tu}} = -\sum_{p=1}^l \frac{\partial E_p(\mathbf{w})}{\partial w_{tu}} = \sum_{p=1}^l \delta_{p,t} o_{p,u} \quad -\frac{\partial E_p(\mathbf{w})}{\partial w_{tu}} = \delta_{p,t} o_{p,u}$$

p = pattern, $\delta_{p,t}$: delta of unit t for pattern p $o_{p,u}$: component u , computed for the input pattern p

See later to add the $/l$ (Least **Mean** Square)

On-line/Batch First Comments



Dip. Informatica
University of Pisa

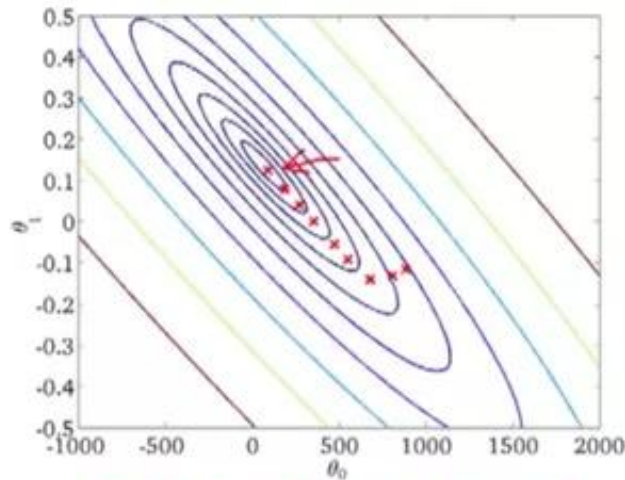
- **Batch** version: more accurate estimation of gradient
- **On-line or Stochastic** version: since the gradient for a single data point can be considered a noisy approximation to the overall gradient, this is also called **stochastic** (noisy) gradient descent.
***Zig-zag (stochastic) descent** (see figures in the next slide)*
 - It can help avoiding local minima
 - A different order of patterns in different epochs should be used (to avoid a bias/drift in the gradient descending)
 - (!) Usually random order (*shuffling* the patterns: i.e. randomly mixing their order in each epoch)
- Many **variations** exist: for instance
 - Stochastic Gradient Descent (SGD) using *minibatch* – *mb* (few training examples) for each step of gradient descent
- → see a next slide for *mb* with further comparisons

Plots stochastics vs batch

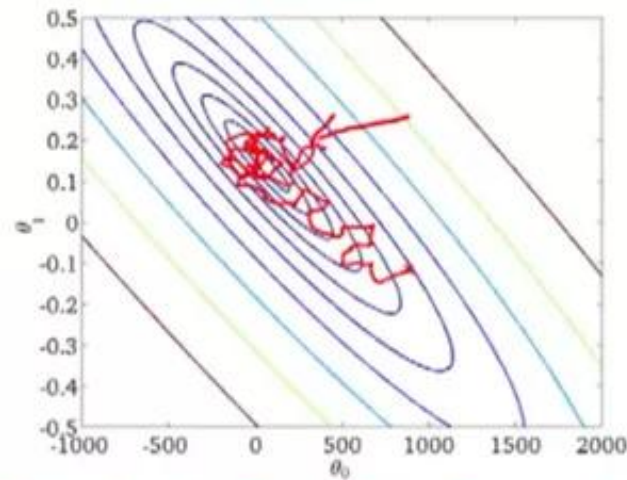
(we have already seen them)



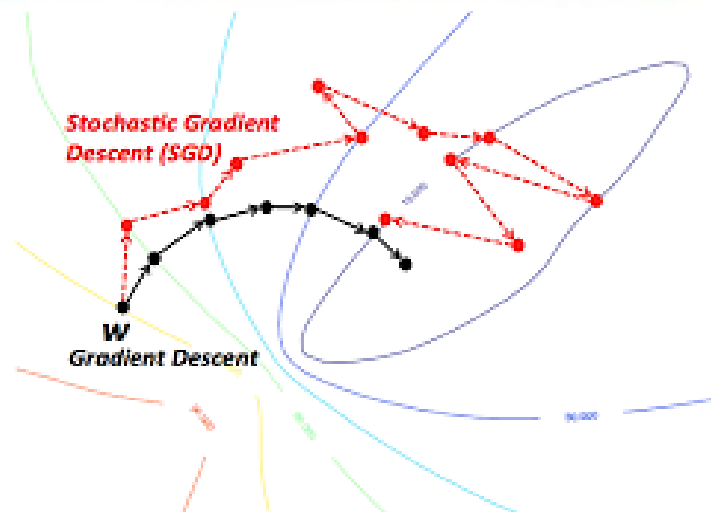
Dip. Informatica
University of Pisa



Batch Gradient Descent



Stochastic Gradient Descent

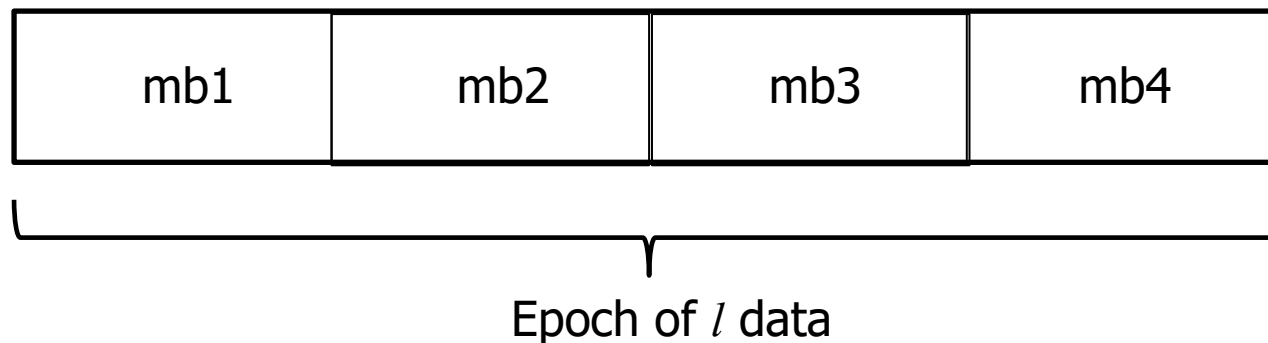


Here a case for **SGD** with high learning parameter

The Mini-Batch

In the Mini-Batch we divide each *epoch* in parts

- We sum the gradient up to ***mb*** patterns before to update the weights
- And we repeat the cycle until all the training data have been used (for an *epoch*).
- So the weight update is every **mini-batch** instead then of singular pattern (on-line) or an entire epoch (batch)





Mini-batch SGD

Mini-batch (with mb examples) to estimate the gradient

$$1 \leq mb \leq l$$

Stochastic/on-line:

- cannot exploit parallel processing of examples
- may be unstable
- + can have a regularization effect (adding noise to the learning process*)

Batch (full training set)

- + more accurate gradient estimation
- but slower! (wait all the data)

e.g. 100

+ the parallelism can be compatible with
e.g. a GPU memory

Minibatch stochastic method → aka **SGD**

- Use random sampling to implement the minibatch (unbiased estimation).
By *shuffling* (at the beginning for all the TR data or for each mini batch)
- With very large TR set or *online streaming* → even no epochs!
- Along with the advantage of gradient descent (linear cost wrt examples) SGD with mb is a key factor in extending learning to very large data sets (over **millions of examples**) because mb (and hence the cost of each SGD update) does not depend on l : *typically just mini-batch and (again) no epochs repetition*

Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, **learning rate**
 - Multiple Minima
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

And also

- Momentum
- Speed-up?

Learning rate – **eta (η)** (also w.r.t batch / on line)



Dip. Informatica
University of Pisa

Recap

- **Batch:** more accurate estimation of gradient → higher eta (η) value
- **On-line:** it can make the training faster but can be more unstable → smaller eta (η) value

More in general (on the eta values):

- High **versus** low eta values : fast (but possible unstable if too large)
versus stable (but possible too slow if too small)
- And we will see various techniques to improve the convergence speed
- Anyway, do you still ask for some values **?!?** **Typically**, search in the range [0.01-0.5] in LMS (see also later)

Practical hints on **eta** values and training behavior



Dip. Informatica
University of Pisa

- **Learning curve**, plotting errors during training, it allows you to check the behavior in the early phases of the model design for your task at hand (preliminary trials)
- (!) – Please check the **Learning Curve** while training a NN!
- We are now looking to the **curve quality** and **convergence speed**: of course, the absolute value for the training error depends also on the *model capacity* (see later for the number of units, regularization etc.) and the other hyperparameters

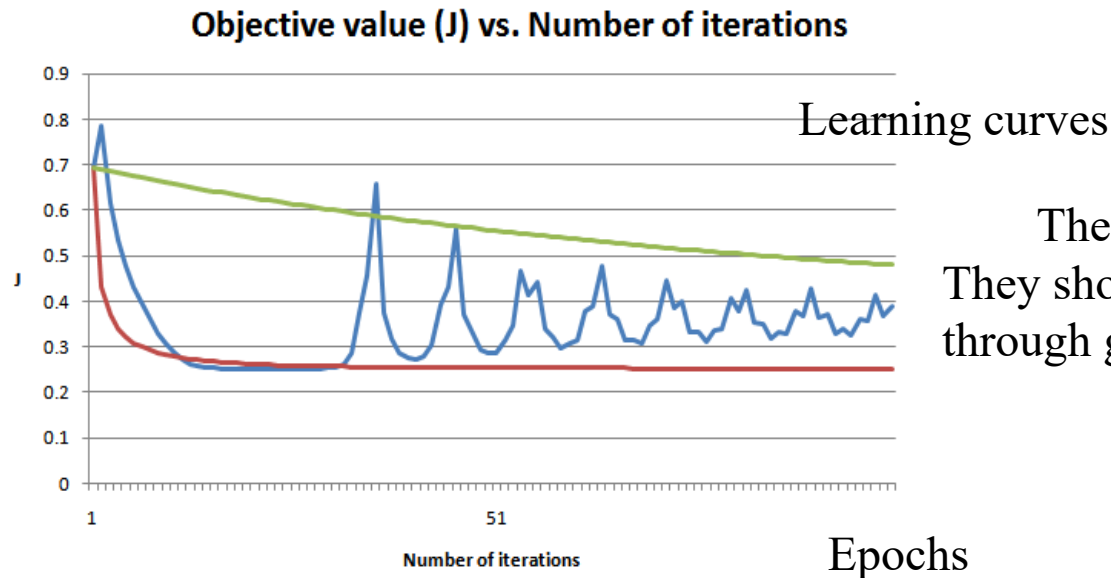
Look to the Learning Curve (**repetita**)



Dip. Informatica
University of Pisa

Which curve do you prefer?

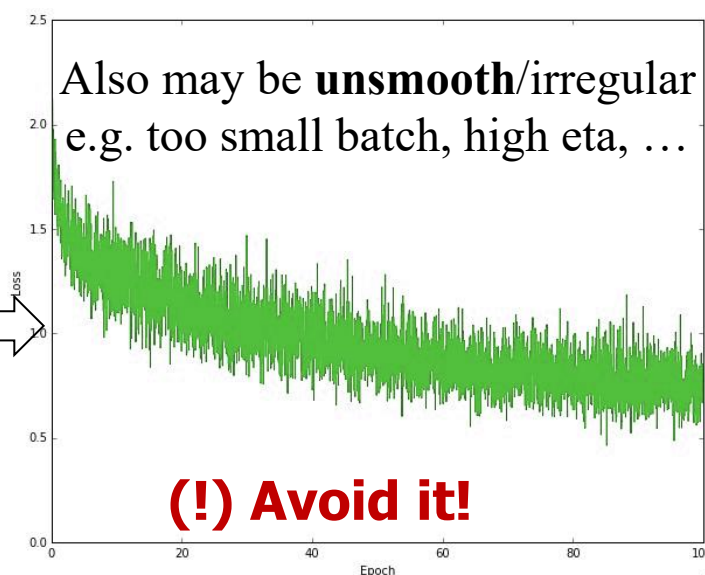
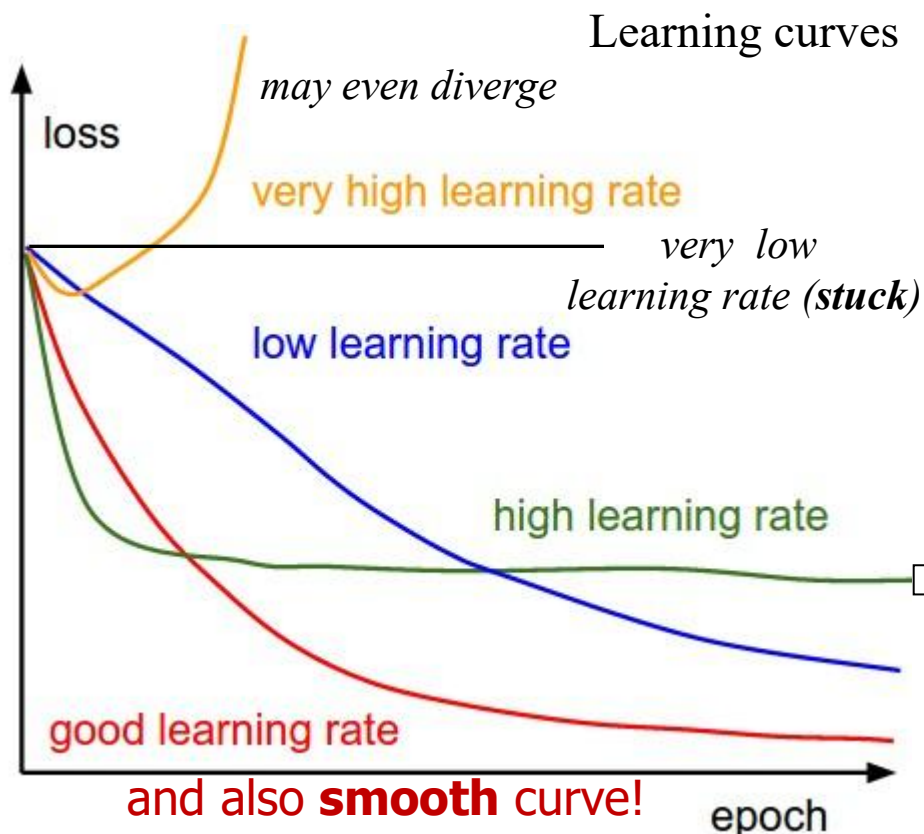
Error



These are learning curves:
They show how the error decreases
through gradient descent iterations

It means that you can decide the best approach
by looking to this plot

Examples with η



Here just looking to the optimization “failures” (on the TR set),
not the generalization errors

Learning rate – eta (η): Least Mean Square



Dip. Informatica
University of Pisa

- Practical approach: it is useful to use of the mean (average) of the gradients over the epoch: it helps to have an “uniform” approach with respect to the number of input data (*Sum of gradients/l*). Hence:

(!) – Repetita from “Gradient descent algorithm”: ***dividing by l correspond to have the Least Mean Squares***

– Note that it is equivalent to have: η/l

NOTE:

$0 < \eta < 1$

when we use LMS

(!) Note that if you compare on-line with batch using LMS (i.e. $/l$), then using **on-line** training will require much smaller eta *to be comparable!!!!*

(in order of the same factor $/l$ used to divide the gradient in the batch version)

(!) For **mini batch** (if you divide by mb in each mini-batch) you have to tune eta as well (to obtain again an eta proportional to the batch case) or simply divide eta by l also for mini-batch (exercise)

Learning rate - Improvements

In the following slides (**guide**):

1. (!) Momentum
 - Nesterov Momentum
2. Variable learning rate (initially high then decrease)
3. Adaptive learning rates (change during training and for each w)
(see also later in Deep Learning lectures)
4. In NN with many layers (see deep learning lectures):
 - It can be varied depending on the layer (higher for deep layers) or fan-in (higher for few input connections)

1. Momentum (!)

$$\Delta w_{new} = -\eta \frac{\partial E(w)}{\partial w}$$

Without momentum

(we now include η in the Δw)

$$\Delta w_{new} = -\eta \frac{\partial E(w)}{\partial w} + \alpha \Delta w_{old}$$

With momentum

$$w_{new} = w + \Delta w_{new}$$

Here η is in Δw

After you can save Δw_{new} for the next step ($\Delta w_{old} = \Delta w_{new}$)

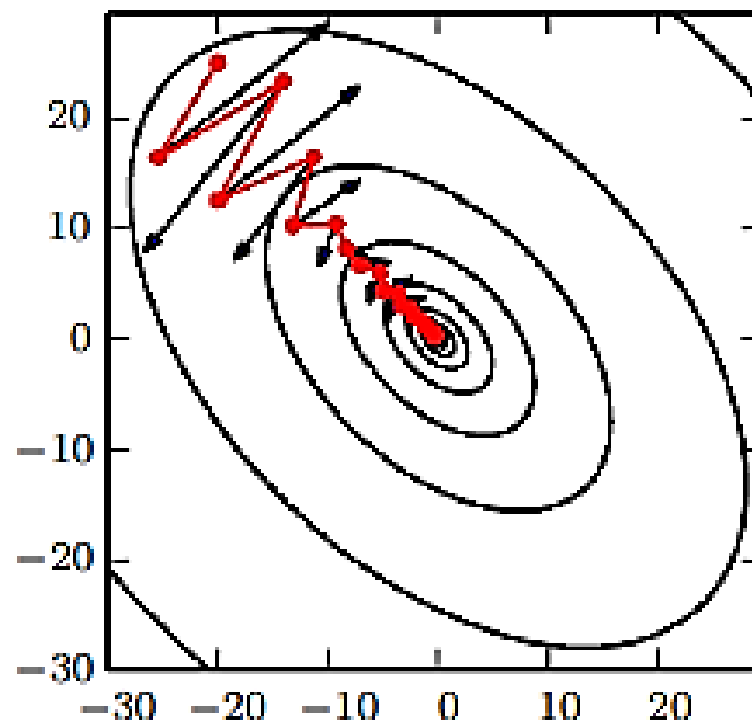
$0 < \text{momentum parameter Alfa } (\alpha) < 1$ e.g [0.5 - 0.9]

«Heavy Ball method»:

- Faster in plateaus:
same sign of grad. for consecutive iterations \rightarrow momentum increase the delta
- Damps oscillations: stabilizing effect in directions that oscillate in sign
- *Inertia effect*: Allow us to use higher eta η (learning parameter: e.g. 0.5-0.9)
- NOTE: Haykin Vol 3 eq. 4.47 " αw_{old} " should be " $\alpha \Delta w_{old}$ "

1. Momentum: Plot

- Effect on a canyon of the error surface (# poor conditioning of the Hessian matrix).
- In **red** the path with the momentum, lengthwise traversing the valley toward the minimum ...
- instead of zig-zag along the canyon walls (following the **black** directions given by the gradient)



It shows the gradient for each red point descending with the moment

1. Momentum (II)



Dip. Informatica
University of Pisa

- Momentum born with “**batch**” learning, and it is commonly assumed that helps more in batch mode
- In the on-line version can be used, in this case:
 - It considers the Δw_{old} as the Δw_{p-1} i.e. of the *previous example*
- And so, it is connected to a *moving average of the past gradients* concept to smooth out the stochastic gradient samples (*inside the epoch*)
[e.g. see it in the form $\overline{\text{grad}} = \alpha \overline{\text{grad}} + (1 - \alpha) \text{grad}_{\text{now}}$]
 - It smooths the gradient over different examples (a different meaning w.r.t. the batch version that uses the delta computed for same batch of examples, among epochs)
 - You can use it multiplying by eta (as usual) to obtain the Δw
 - See if really needed (as any other heuristic)
- For Minibatch: again, moving average of past gradients over the progressive minibatches (that have different examples, not the step before for the same examples as for batch).

1. Nesterov Momentum



Dip. Informatica
University of Pisa

A variant:

- Evaluate the gradient after the momentum is applied
 - First apply the momentum ($\underline{w} = w + \alpha \Delta w_{old}$)
 - Evaluate the *new gradient* at this interim point (i.e. with such \underline{w})
 - Compute and apply the DeltaW (to the original w) as before (summing the momentum and the *new gradient*)
- Shown to improve the rate of convergence *for the batch mode* (not for the stochastic case!)
- Inspired by Nesterov's accelerated gradient (with nice formal properties in the convex case)*
- See DL book 8.3.3, popular in SW libraries such is *Caffe* (DL software)

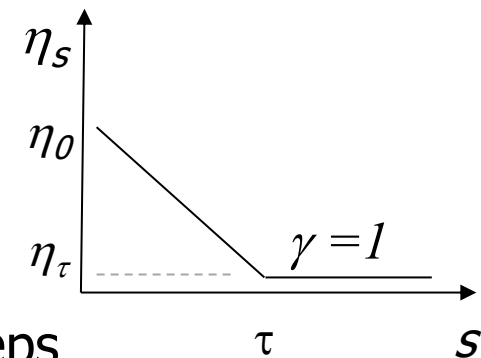
2. Variable Learning rate and Mini Batch



Dip. Informatica
University of Pisa

- Using mini-batch \rightarrow the gradient does not decrease to zero close to a minimum (as the exact gradient can do)
- Hence fixed learning rate should be avoided:
- For instance, we can **decay linearly** eta for each step until iteration tau (τ), using $\gamma = (\text{step } s) / \tau$:

$$\eta_s = (1 - \gamma)\eta_0 + \gamma\eta_\tau$$



- Then stop and use fix (small) η ($=\eta_\tau$)
- Set-up e.g.: η_τ as $\sim 1\%$ of η_0 , τ few hundred steps
- And η_0 ? Again: no instability/ no stuck trade-off (preliminary trials or grid-search)
- See Section 8.3.1 DL book

3. Adaptive Learning Rates



Dip. Informatica
University of Pisa

- Automatically adapt the learning rate during training possibly avoiding/reducing the fine-tuning phase via hyperparameters selection
- Also, they can include even the usage of a separate **eta** for each parameter of the network

E.g. (popular):


- AdaGrad
- RMSProp (becoming quite popular for DL, e.g. caffe SW)
- Adam (also very popular, often robust with the *default value*, but warning!)
- ... *continuously evolving*
- Combined with moment etc. (many variants*)
- Popular for DL: see DL book 8.5 if interested



Issue

Is the good old basic SGD still the best approach to NN training?

Slow ?

- As any iterative approaches.... Basic SGD is generally efficient (linear in the number of free parameters) but depends on the number of training data and on the number of epochs
- Lots of other heuristics approaches (in the NN literature): 
 - *Quick-prop*: go to the minima of the local estimated convex function (assumed as a parabolic curve) (see Bibliography @ end of slides)
 - *R-prop* does not use the value of the gradient (it can vanish for deep layers) but the sign of gradient (see Bibliography @ end of slides)
 - Deep Learning need special care (see later)
- Second-order techniques, or other from CM/O4DS? See next slide!

Second order methods

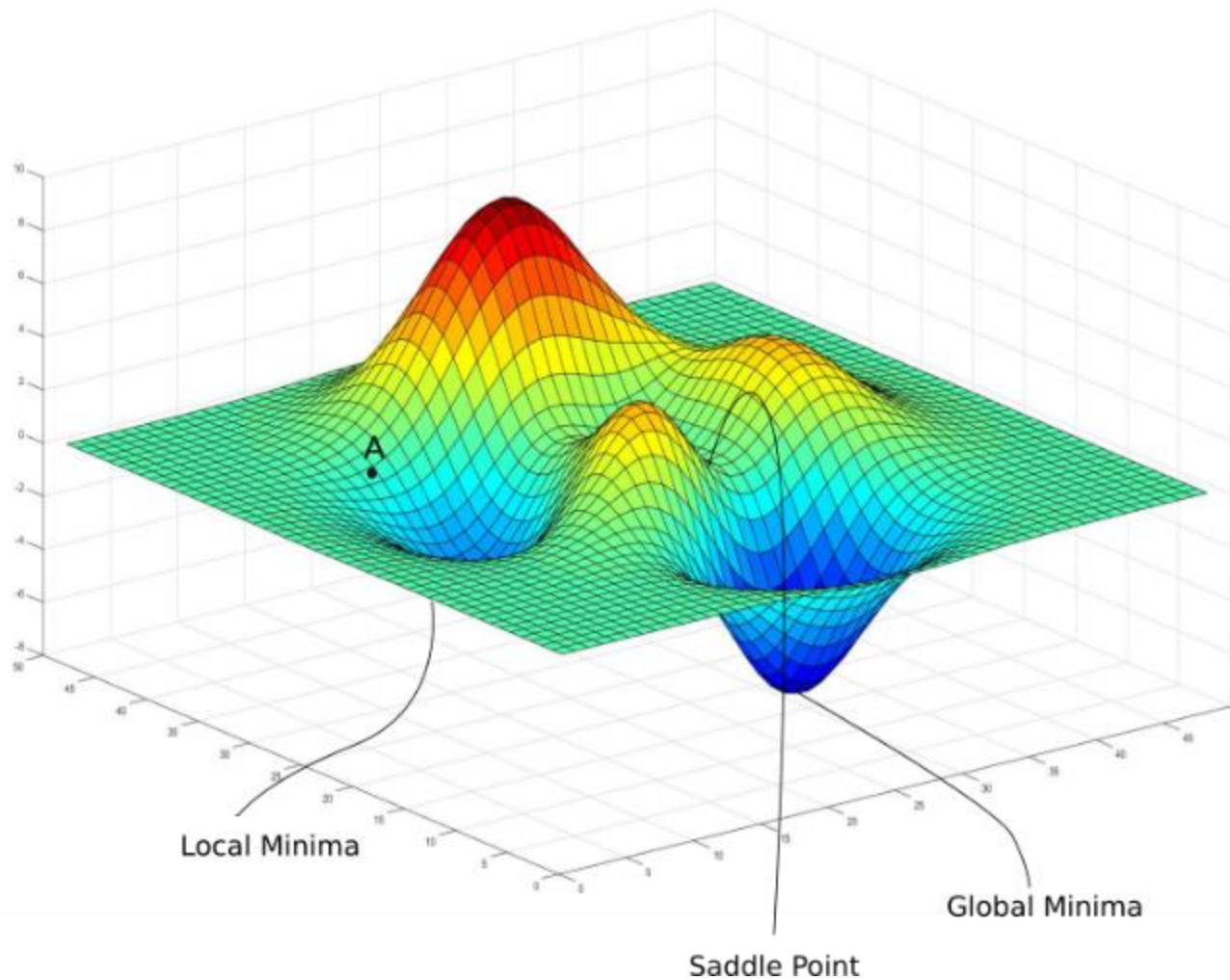
Use the *second order derivatives* (Hessian matrix): more information about the curvature of the objective function → better descent (*Newton's methods*)

- But the Hessian can be too large (i.e. computationally expensive), and see the next “saddle points” slide

In the form of *approximate* second order methods we find:

- *Approx. Newton's methods* (also *Gauss-Newton* and *Levenberg-Marquardt* approximation)
- Hessian free approaches (without computing H and H^{-1})
 - *Conjugate gradients*
 - can be nice for faster converge! (see e.g. in “Numerical Recipes in C”).
 - *Quasi-Newton methods: BFGS* (Broyden–Fletcher–Goldfarb–Shanno Algorithm)
- ...
- References: Haykin book, DL book
- Is possible to try them within a CM/ML prj? Yes, especially with ad hoc variations for NNs

Minima and Saddle Points



Means technical aspect:
Not needed at the beginning

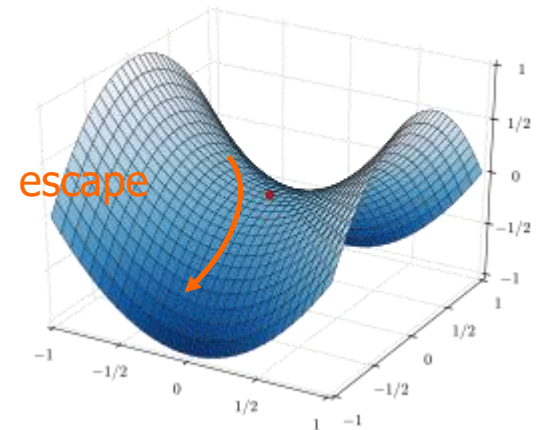
Saddle points #tech



Dip. Informatica
University of Pisa

More intriguing discussion on saddle points (see e.g. 8.2 Deep Learning book) i.e. Gradient (close to) zero but not a local minima

- Saddle points can grow exponentially (wrt to local minima) in high dimensional spaces
- Newton search for (jump to) gradient zero points – more sensible to find saddle points
- This can explain why (standard) second-order methods (looking to Hessian, e.g. *Newton's method*) have **not succeed** in replacing gradient descent for NN training
- Instead, gradient descent empirically shows to **escape** saddle points [...]
- But the length of the training path can be high for many other reason not related to local minima or saddle points (small gradient values), such as time to “circumnavigate” local maxima and so on...
- But such difficult cases are not frequent! Why? Research is on-going...!
- *Typically, we are able to find a low value of the loss quickly enough to be useful!* And especially useful to generalize! (we don't need a minimum at all, see previous slides on multiple/global minima!).





In practice?

- For optimization: **Don't worry**, the main basics approaches e.g. with **(!)**, often are enough *if you apply them correctly*
 - The **experience** with well-know techniques have a major role than sophisticated approaches *that you cannot manage*.
 - The optimization technique is relevant only if there are obstacles for training. Validation will have a major role than the optimization on the training set (we will have next lectures on this): remember, optimize on the training data is not our purpose!
 - The others are an opportunity to explore different approaches (especially using libraries)
- SGD with momentum (ad we will see regularization) is in any case a **starting point**, also as comparison if you move toward other approaches
 - You have to start from this, also for didactic reasons!

Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters**: Starting values, on-line/batch, learning rate
 - Multiple Minima
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters**: Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

Stopping criteria

- The basic is the used error ($mean\ error < E$): it is the best if you know the **tolerance of data (expert knowledge)**, but we often have not this info.
 - Also: Max (instead of mean) tolerance on data
 - For classification: Number of misclassified (error rate)
- But we often need to use “internal criteria”
 - No more relevant ***weight changes*** / ***near zero gradient*** (Euclidian norm of gradient $< \epsilon$) / No more significant ***error decreasing*** for a epoch (e.g. less than 0.1%):
NOTE: may be premature (e.g. if η is too small)! It can be applied observing k epochs (known as *patient*).
- (!) In any case stop after an excessive number of epochs (just to escape from too slow convergence),
 - but *avoid* to stop fixing an arbitrary number of epochs (*)
- *and not necessarily stop with very low training error...*(see next)
 - (*) We will discuss soon the “early stopping” and also later FAQ and typical errors for stopping (discussing the validation issues, in validation lectures)



Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, learning rate
 - Multiple Minima
 - Stopping criteria
 - **Overfitting and regularization** Much more important!
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

Stopping at the minimum?

- Typically we don't want the global minimizer of $R_{emp}(w)$, as this is likely to be an overfitting solution
- This is also a difference w.r.t. optimization (CM) methods (see the previous slide <But is global minima needed?>)
- The *control of complexity* is our main aim to achieve the best generalization capability
- For instance we need to add some *regularization*:
 - this is achieved directly through a penalty term, or indirectly by early stopping.
 - plus model selection (cross-validation) on empirical data to find the trade-off

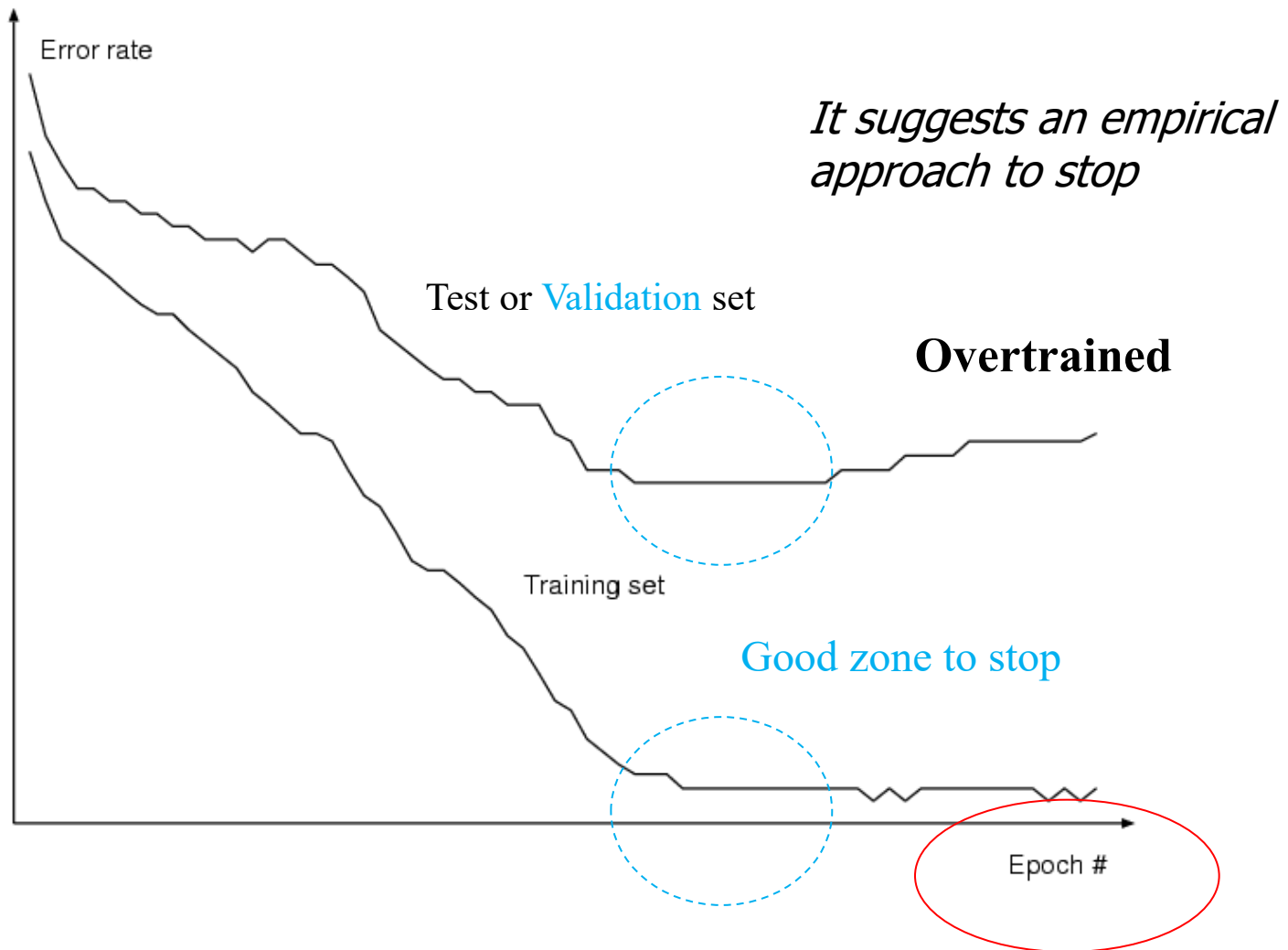
Overfitting in NN

- Start learning with small random weights (symmetry breaking)
- The mapping input→output is nearly linear:
 - *number of effective* free parameters (and VC-dim) nearly as in perceptron
- As optimization proceeds hidden units tend to *saturate*, increasing the *effective number of free parameters* (and introduce non-linearities) and hence increase *VC-dim*
- As we discussed, this is a *variable-size hypothesis space* (it changes during training)

Typical behavior of backprop: learning curve



Dip. Informatica
University of Pisa



Learning curve analysis

Recall the VC-bound plot ! (theoretical ground)

How to act?

1. *Early stopping*: use a **validation set** for determining when to stop: at the *validation error increasing*: ... quite vague*!
 - * Use more than one epoch before estimating (called "*patience*")
 - Backtracking approach
 - Note that since *effective* numb. of parameters grows during the course of training, halting training corresponds to limit the effective complexity
 - See "[Early Stopping — But When?](#)"» L. Prechelt, LNCS 2012
 - We will discuss also later about *Early Stopping* (*FAQ & typical errors*)
2. *Regularization* (on the loss: our main warhorse!)
3. *Pruning methods*: see later



Important !

2. Regularization (!)

- Training \rightarrow weight increases: We can optimize the loss considering the weight values
- A well know principled approach (related to **Tikhonov theory**, again):
- Add a **penalty term** to the error function:

$$\begin{array}{c}
 \text{Loss} \\
 \text{Loss} (w) = \sum_p (d_p - o(x_p))^2 + \lambda ||w||^2
 \end{array}
 \begin{array}{c}
 \text{Regularization/penalty term} \\
 \text{"Error" term [(M)SE]}
 \end{array}
 \begin{array}{c}
 \text{A sum over all} \\
 \text{the weights} \\
 \text{in the network} \\
 \rightarrow \sum w_i^2
 \end{array}$$

- Effect: **weight decay** (basically add $2\lambda w$ to the gradient for each weight)

$$w_{\text{new}} = w + \text{eta} * \Delta w - 2 \lambda w \quad (\text{of course } 2 \text{ can be omitted})$$

- Chose the lambda regularization parameter (generally very low value, e.g. 0.01): selected by the *model selection* phase (cross-validation)
- Note: if applied to linear model \rightarrow it is the *ridge regression* (see lect .on linear models)
- More sophisticated penalty terms have been developed (e.g. for *weight elimination*: see Haykin, you can try them)



2. Term issue: Error and Loss

Repetita



Dip. Informatica
University of Pisa



Recall what we already discussed about the objective function:

- When we have, as in the Tikhonov regularization, a penalty term in the objective function is better to clarify:
 - We can use **Loss** term for the objective function (MSE+Penalty) for model training
(coherently with the CM course, in a broader use of the term)
 - And **Error** /Risk for the “data term” (the MSE, depending on data)

$$\sum_p (d_p - o(\mathbf{x}_p))^2$$

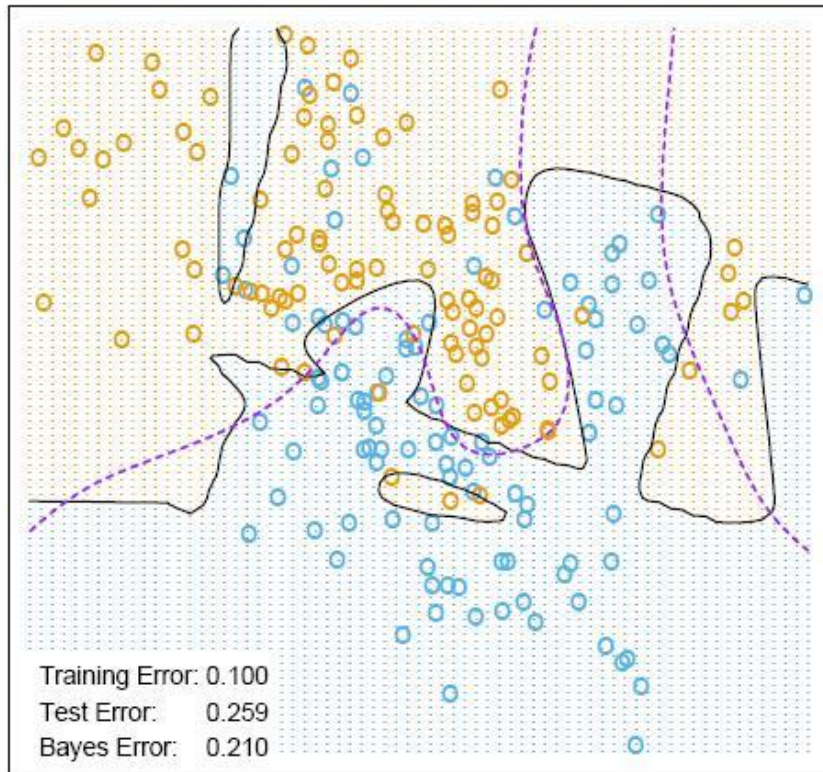
“Error” term [(M)SE]

to evaluate model error, because this is the measure useful to the user (using the model) and to be reported in the table or plots etc. (!)

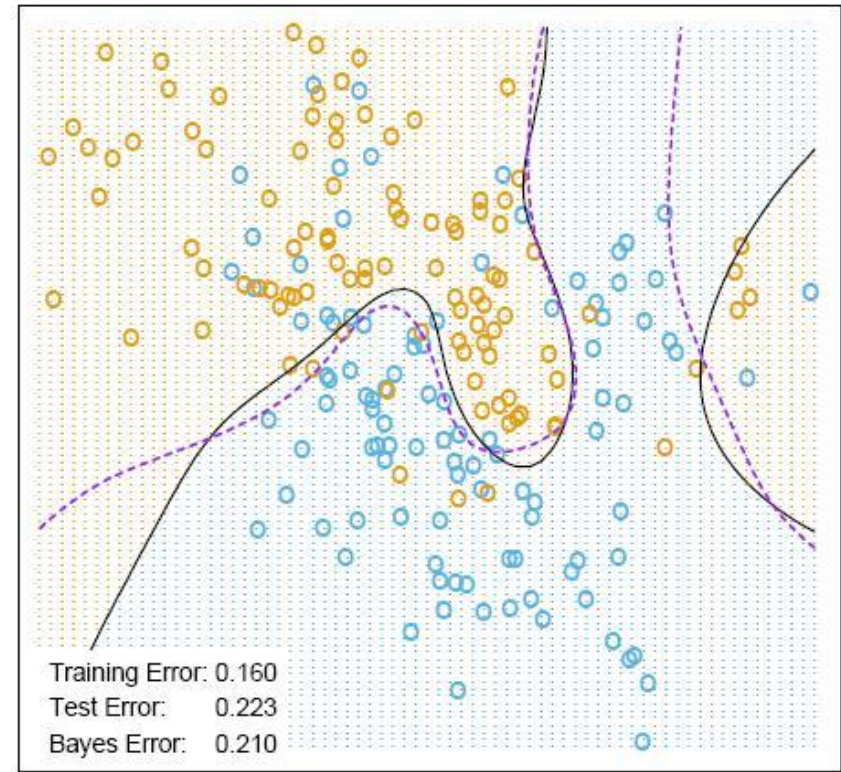
2. Effect of regularization

Weight decay with $\lambda=0.02$

Neural Network - 10 Units, No Weight Decay

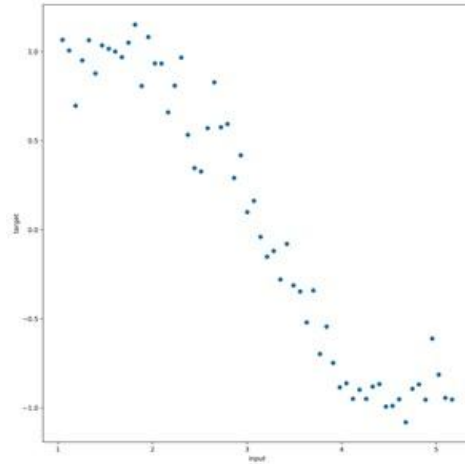


Neural Network - 10 Units, Weight Decay=0.02

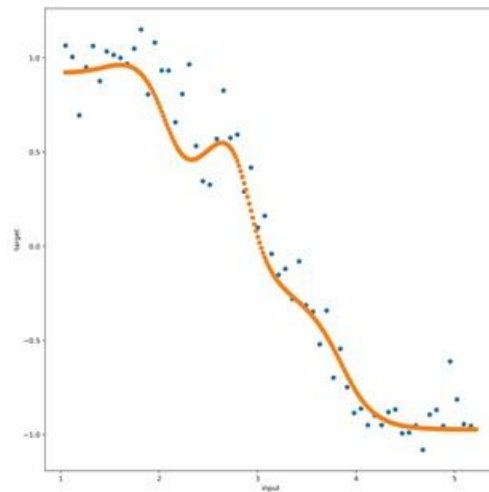
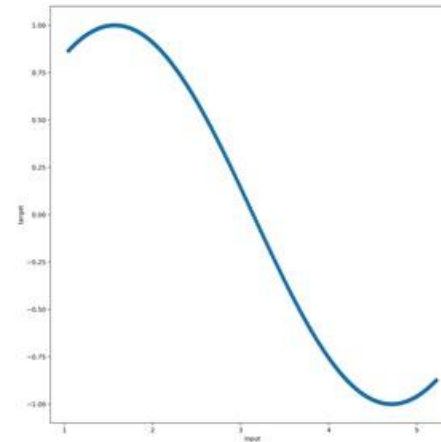


Univariate regression example

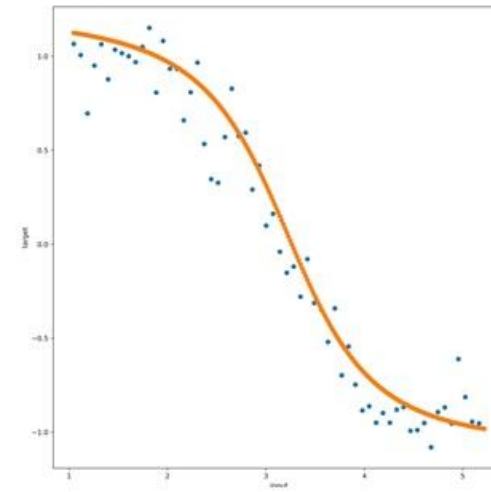
Training data



True function

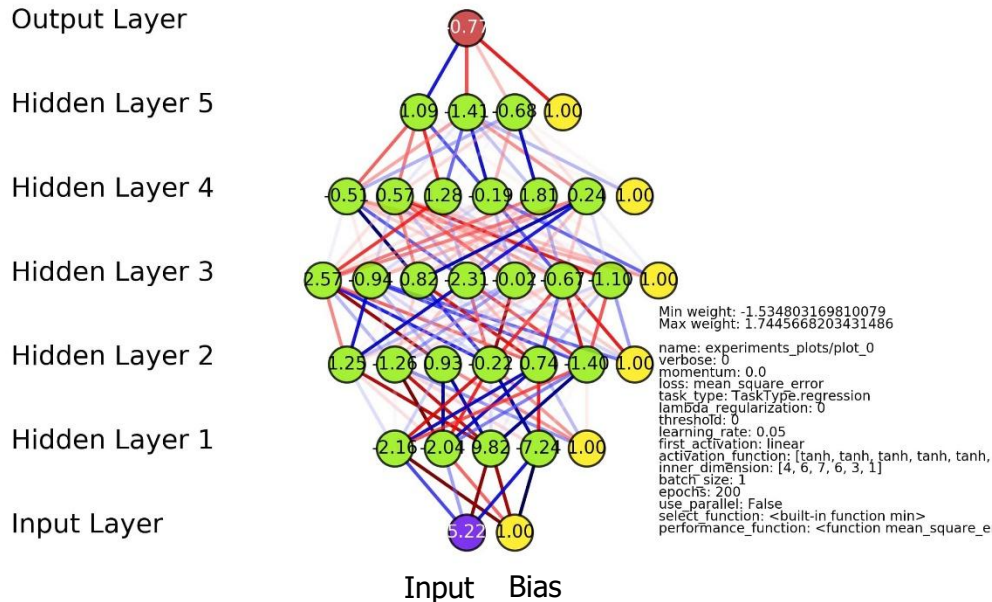


NN output $\lambda=0$



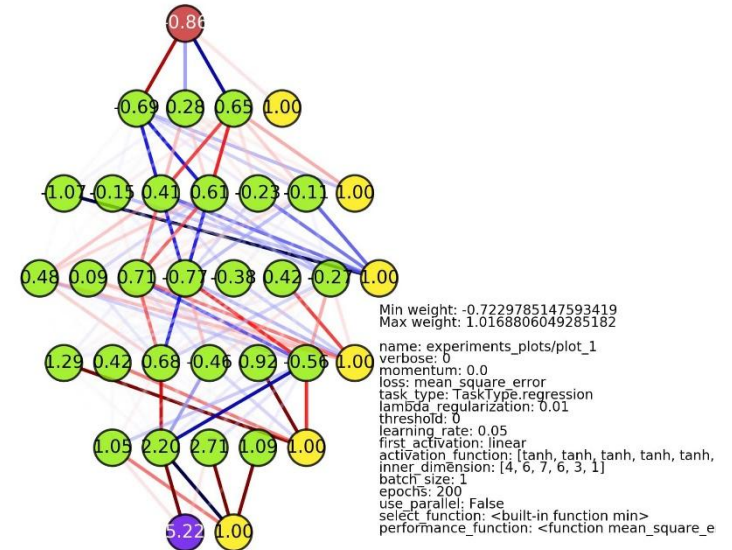
Regularized ($\lambda=0.01$)

Neural Network architecture



NN trained with $\lambda=0$:
all the weights are significantly
higher (or lower) than 0.

Neural Network architecture



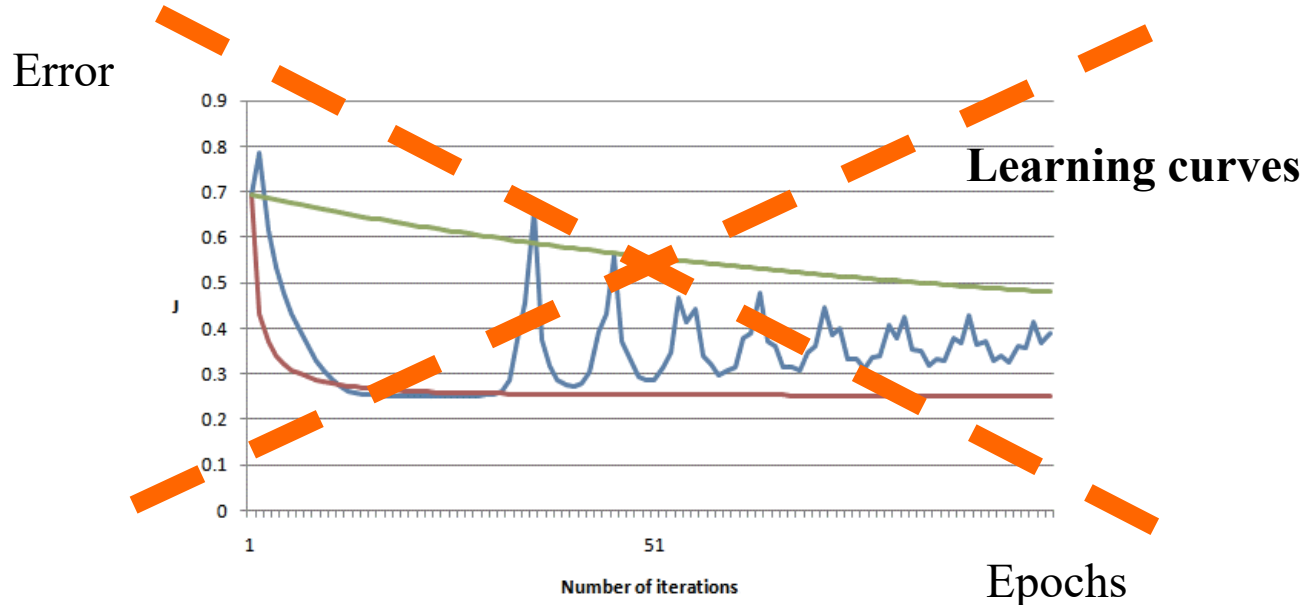
Regularized ($\lambda=0.01$):
most of the weights are close to zero
(light color \rightarrow lower in module).
Hence lower model **complexity**!

2. Regularization: Frequent misunderstandings (I)



Dip. Informatica
University of Pisa

- I. Does the regularization help the **convergence stability**?
It is important to highlight that the regularization **is not** a technique to control the *stability* of the training converge (*learning curve*) but, as you well know, it allows the control of the *complexity* of the model (measured by the VC dimension according to the SLT theory), which is related to the number of weights and values of the weights in NNs.



2. Regularization: Frequent misunderstandings (II)

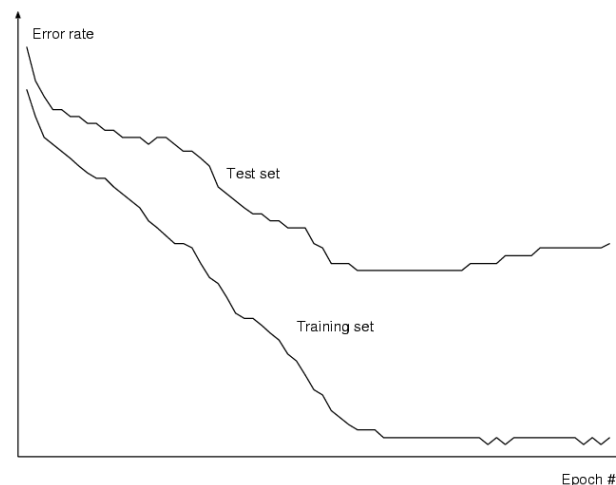


Dip. Informatica
University of Pisa



II. Better Early Stopping (**E.S.**) or Regularization (**Regul.**)?

We will discuss later, but note that the E.S. is an *empirical approach* that needs a **VL set** to decide the stopping point, which sacrifices a part of data. The Regul. in the Loss is a *principled approach*: it allows the VL curve to follows the TR curve (provided that you avoid that the VL curve is close to TR curve just due to underfitting with a too high lambda); in that case you don't need (or strictly need) of the E.S. heuristic and you can stop at training convergence. Anyway, you can also use **both**!

(and the E.S. will not enter in action if VL error does not increase)



2. Regularization: details

- Note that often the **bias** w_0 is omitted from the regularizer (because its inclusion causes the results to be not independent from target shift/scaling) or it may be included but with its own regularization coefficient (see Bishop book, Hastie et al. book) 
- Typically apply it in the batch version 
- For **on-line/mini-batch** take care of possible effects over many steps (patterns/examples): hence to compare w.r.t. batch version in a fair way do not force equal *lambda* but it would be better to use

$$\lambda \propto (mb / \# \text{total patterns } l)$$

- Of course if you choose lambda by model selection they will automatically select different lambda for on-line and batch (or any mini-batch)
- Other techniques: **Dropout** → see later

Q&A lecture set-up

- Please send to me further questions by **email** so to set up Q&A lectures

Please, **USE in the subject the TAG [ML-24-Q]**

- We can discuss
 - Self-evaluation quiz results
 - New questions to better understand some parts of the course (no questions, no Q&A lecture ;-)
- Memento
 - Special interactive classes will be used to assess activities and make a ML discussion forum (**Q&A classes**).
This makes the course a class, not just a set of records
 - Within a **cooperative** and collaborative context!

2. Putting together I (FAQ)



Dip. Informatica
University of Pisa

- Merging (1) Momentum with (2) Weight decay

$$\Delta w = -\eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} + \alpha \Delta w_{old}$$

(1) We wrote for the Momentum, but what is $E(\mathbf{w})$?
We used the MSE Error

- (2) Moving to include the weight decay (Tikhonov), we move to the *Loss*:

$$\Delta w_{tu} = -\eta \frac{\partial \text{Loss}(\mathbf{w})}{\partial w_{tu}} + \alpha \Delta w_{oldtu} \stackrel{\text{assumed}}{=} \eta \delta_t o_u - \lambda w_{tu} + \alpha \Delta w_{oldtu}$$

Where (p are omitted and) we consider the $\text{Loss} = \text{Error} + \text{Penalty}$ separating the 2 terms in the derivation and using eta only for E term (we already did it!: see the exercise in the linear model part). In this way, eta does NOT multiply lambda \rightarrow changing eta we do not change lambda and vice versa (i.e. we make **independent** the two hyperparameters and their effects, and we make easier to control them).

- In this context (because, as for the momentum, η is in the $\Delta \mathbf{w}$), we have

$$\mathbf{w}_{\text{new}} = \mathbf{w} + \Delta \mathbf{w}$$

2. Putting together II (FAQ)



Dip. Informatica
University of Pisa

Hence, $\Delta w_{tu} = \eta \delta_t o_u - \lambda w_{tu} + \alpha \Delta w_{old tu}$

- And now, does Δw include the lambda part for the momentum ?
Often yes, e.g. in the toolbox of Matlab, Caffe simulator etc.
- But we can also keep separation of the 3 hyperparameters writing:

$$\Delta w_{tu} = \eta \delta_t o_u + \alpha \Delta w_{old tu}$$

$$w_{new tu} = w_{tu} + \Delta w_{tu} - \lambda w_{tu}$$

i.e. by adding the lambda part
directly to w update
(*separating it from the
momentum memory*)

- In this form lambda, eta and alfa become ***independent*** !
- And you can better control the different role of 3 hyperparameters
- Further notes:
 - divide by l only the gradient of E (i.e. η) where you use a sum over the patterns

Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, learning rate
 - Multiple Minima
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

Number of Units

- Related to the **control of complexity** issue !!!
- Related also to the input dimension and to the size of the TR data set

In general: this is a "***model selection***" issue

- The number of units along with regularization parameter can be selected by a cross-validation, i.e. the model selection phase (on a validation set)
 - To few hidden units → underfitting; to many → overfitting (can occur)
 - *The number of units can be higher if appropriate regularization is used*
- **Constructive approaches:** the learning algorithm decides the number starting with small network and then adding new units
- **Pruning methods:** start with large networks and progressively eliminate weights or units

Constructive approaches

- *Incremental approaches*: algorithms that build a network starting with a minimal configuration and
- add new units and connections during training.
- Examples:
 - For classification: Tower, Tiling, Upstart
 - For both regression and classification: Cascade Correlation
 - **See Scott E. Fahlman and Christian Lebiere:**
The Cascade-Correlation Learning Architecture 1991 CMU-CS-90-100 Carnegie Mellon University
Also in NIPS 2, 1990, pages 524-532



: -)

Constructive approaches

- An exemplificative and effective approach :
- the **Cascade Correlation (CC)** learning algorithm
- Start with a minimum networks
- Add units if they are needed until the error is low.
- The CC alg. learns both the network weights and network topology (number of units)
 - **automatic** determination of network dimension and topology:
CC allows to deal with hypothesis spaces of flexible size, since the number of hidden units is decided by the learning algorithm;
 - training of a **single unit** for each step.

The CC algorithm

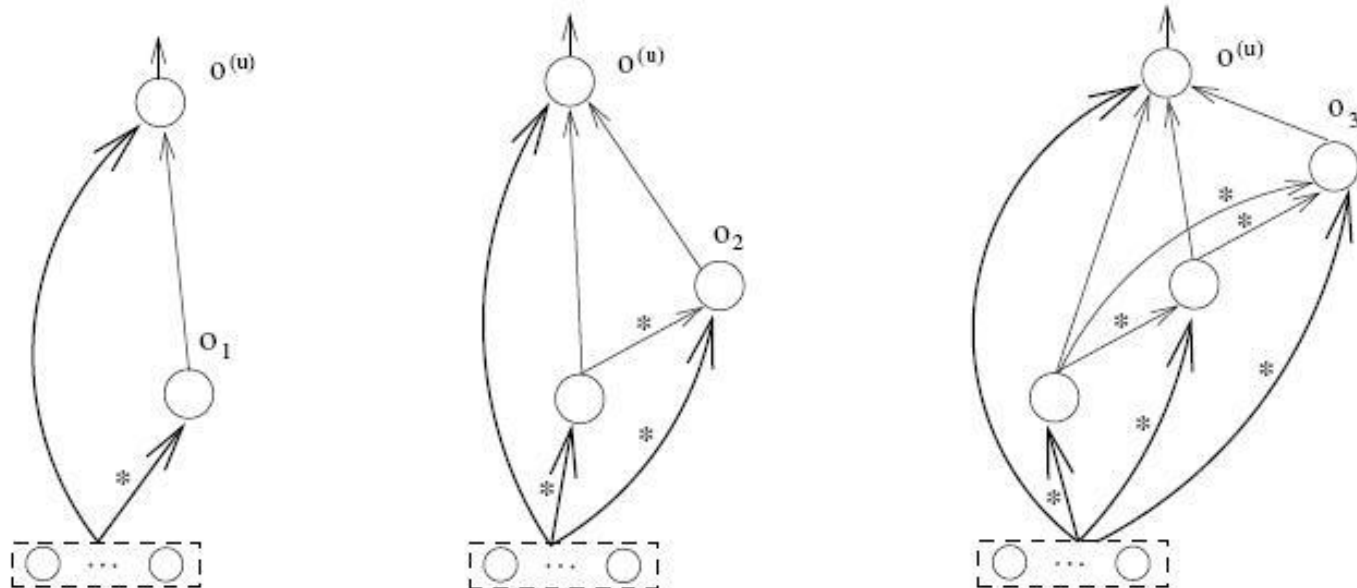
- **Start** with N0 network, a network without hidden units: Training of N0. Compute error for N0. If N0 cannot solve the problem: go to N1.
- In the N1 network a hidden unit is added such that the correlation between the output of the unit and the **residual error of network N0 is maximized** (by training its weights, see next slide).
- After training, the weights of the new unit are **frozen** (they cannot be retrained in the next steps) and the remaining weights (output layer) are retrained.
- If the obtained network N1 cannot solve the problem, new hidden units are progressively **added** which are connected with all the inputs and previously installed hidden units.
- The process continues until the residual errors of the output layer satisfy a specified **stopping** criteria (e.g the errors are below a given threshold).

The method dynamically builds up a neural network and terminates once a sufficient number of hidden units has been found to solve the given problem.

Cascade of units

CC architecture building

- The evolution of a CC network with up to 3 hidden units



© A. Micheli 2003

* = weights frozen after candidate training

Learning for CC

- Specifically, the algorithm works interleaving the minimization of the total error function (LMS), e.g. by a simple backpropagation training of the output layer, and the **maximization** of the (non-normalized) correlation, i.e. the covariance, of the new inserted hidden (candidate) unit with the residual error:

$$S = \sum_k \left| \sum_p (o_p - \underbrace{\text{mean}_p(o_p)}_{\bar{o}}) (\underbrace{E_{p,k} - \text{mean}_p(E_{p,k})}_{\bar{E}}) \right|$$

$E_{p,k} = (o_{p,k} - d_{p,k})$
 E : residual error
 with $o_{p,k}$ at the output layer
 p : pattern, k : output unit

o (in eq. for S): candidate output

- Exercise:** derivation (sketch on blackboard)

hints: $df/dx = \text{sign}(f) df/dx$; assume no effect of variations of o_p over $\bar{o} = \text{mean}_p(o_p)$

- Result:

$$\frac{\partial S}{\partial w_j} = \sum_k \text{sgn } S_k \sum_p (E_{p,k} - \text{mean}_p(E_{p,k})) f'(net_{p,h}) I_{p,j} \rightarrow \text{Input}$$

↓

h : candidate index
(not needed since we have just one candidate at time)

- Weight update: why $+ dS/dw_j$?

Notes on CC

- The role of hidden units is to reduce the residual output error
 - Each unit solves a specific sub-problem
 - Each unit becomes a permanent “feature-detector”
- POOL: Typically, since the maximization of the correlation is obtained using a gradient ascent technique on a surface with several maxima, a pool of hidden units is trained and the best one selected. This helps avoiding local maxima.
- Greedy algorithms: easy to obtain converge, may also find a minimal number of units but it may lead into overfitting (need regularization)
- Loss: Not necessarily “max S”: e.g., for regression, LMS in Prechelet 1997, Neural Networks Vol. 10 pp. 885-896

Issues in Training NN

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often over-parametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
 - **Hyperparameters:** Starting values, on-line/batch, learning rate
 - Multiple Minima
 - Stopping criteria
 - Overfitting and regularization
 - **Hyperparameters:** Number of hidden units
 - Input scaling/ Output representation
 -
- A more specific discussion for DL is in the DL book (we will see later)

Input/Output

- Preprocessing can have large effect
- For different scale values, *normalization* via :
 - Standardizations (often useful):

For each feature we obtain zero mean and standard deviation 1:
v-mean / standard dev.
 - Rescaling: $[0,1]$ range: $v-min / (max-min)$
- *Categorical inputs: see Lectures for Introduction (e.g. 1-of-k)*

A \rightarrow 100, B \rightarrow 010, C \rightarrow 001
- *Missing data*: Warning: 0 does not mean "no input" if 0 is in the input range !



Input/Output



Dip. Informatica
University of Pisa

- **Regression:** Output **linear** unit(s) (!)
 - (1 unit or more output units for multiple quantitative response)
- **Classification:** 1unit (binary) or 1-of-K with multioutput
 - Sigmoid \rightarrow choose the *threshold* to assign the class
 - *Rejection zone* (is possible, see slide on sigmoidal functions)
 - *1-of-K encoding*: to choose the winner class take the highest value among the output values
 - Often symmetric logistic (*TanH*) learns faster (see Haykin book)
 - 0.9 can be used instead of 1 (as d value) to avoid asymptotical convergence (-0.9 for -1 for TanH, or 0.1 for 0 for logistic)

Of course, target range must be in the output range of units

Regression and Classification Tasks



Dip. Informatica
University of Pisa

- **Classification:** multi output , 1-of-K encoding (each coded as 0/1 or -1/+1)
 - **Sigmoidal activation (i.e. Tanh)**
 - **For 0/1 targets:** Softmax function (sum to 1, can be interpreted as probability of the class $p(\text{class}=1|x)$!!!)

Popular
nowadays for
classification

For numerical issues:
DL book (sec 4.1 and 6.2)

Unit k
$$o_k(\mathbf{x}) = \frac{e^{(-net_k)}}{\sum_{j=1}^K e^{(-net_j)}}$$

- Also we can use an alternative error/cost function: minimize the “cross entropy” \rightarrow maximum likelihood estimations

e.g for 1 unit:
$$-\sum_{i \in TR} \{d_i \log(out(x_i)) + (1 - d_i) \log(1 - out(x_i))\}$$

If loss =0 \rightarrow no classification errors!

Ref: Bishop 1995
Also DL book (sec. 6.2)

Exercises

Open questions

(for your self-assessment and to make a good project):

- Think on the role of the different hyper-parameters
- Connect to the theory (also later after next lectures)
 - E.g. relate the optimization of the loss with penalty term to the Statistical Learning Theory (VC-bound in the introduction and next lectures) to understand the role of the *number of units*, *lambda*, *and others*, in controlling under/overfitting cases.
- Exercise: Hence, which are the most relevant for your prj (e.g. to win the blind competition)?

REPETITA: Bibliography (NN second part)



Dip. Informatica
University of Pisa

- Haykin: chapter 4
- Mitchell: chapter 4
- Hastie, Tibshirani, Friedman : cap 11
- *NEW*: some insertions from the Deep Learning book.
See details and references to sub-sections within the slides



— Specific sections:

- **Backpropagation**: Parallel Distributed Processing, Vol.1, by D. E. Rumelhart, J. L. McClelland, MIT Press: Chapter 8 by Rumelhart, Hinton, Williams (1986)
- **Cascade correlation**: S. E. Fahlman and C. Lebiere: The Cascade-Correlation Learning Architecture CMU-CS-90-100 – Carnegie Mellon University, 1991
Also in NIPS 2, 1990, pages 524-532

Others:

- **Quick-Prop**: S. E. Fahlman: An Empirical Study of Learning Speed in Back-Propagation Networks, TR CMU-CS-88-162, 1998
- **Rprop**: M. Riedmiller and Braun H. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In Proceedings of the IEEE International Conference on Neural Networks, pages pp 586–591, 1993.

FIRST APPLICATIONS

Applications (**guide slide**)

- A benchmark, anticipating the MONK example to “start” the projects
- CNN & character recognition (next lectures)
- Final lectures at the end of the course

BUT we will anticipate
other parts to favorite the project starting

PRJ: Ready to go?

Ready to start the project? **NO**,

BUT **You can start to think** about NN implementation (core of the training part, with benchmark data) or exploring first usage of the simulators; and I'm going to introduce the **Monk** benchmark.

Not ready because we have still to present

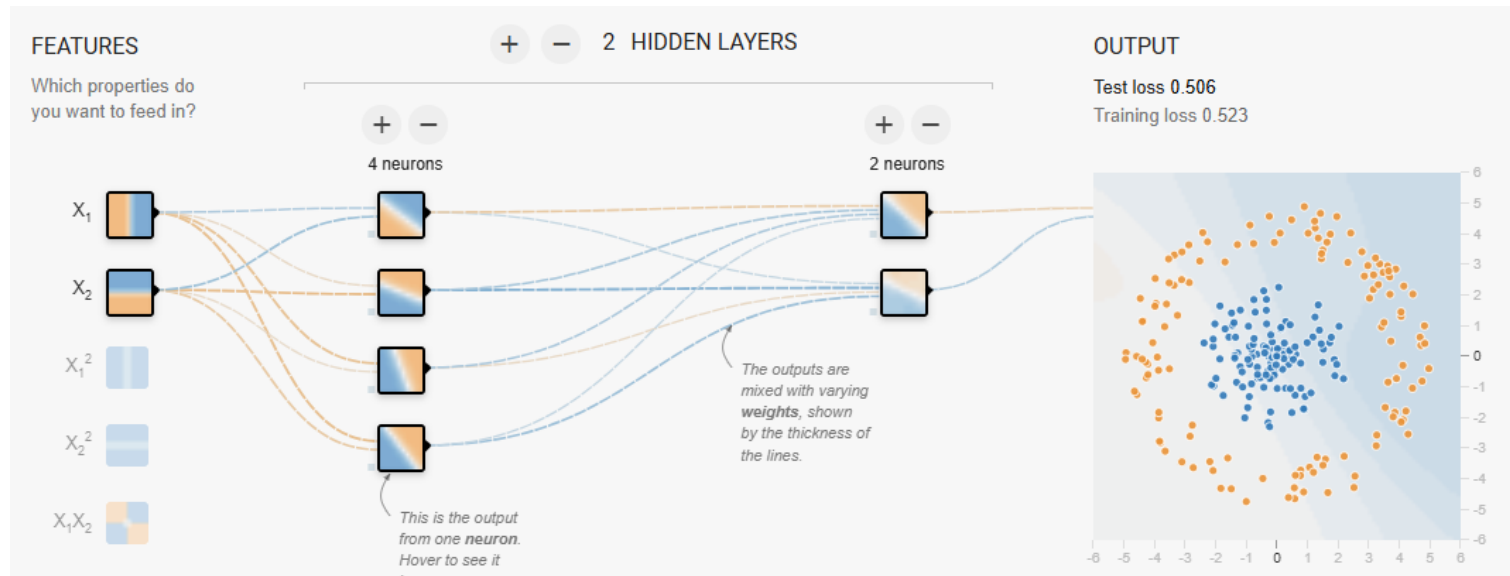
- VALIDATION (hyperparameters selection, min. 2 lectures)
- The PRJ assignment (1 lecture)

The project (type A) will have

- Backprop (gradient computation) implementation (ready)
- DeltaW rules implementation (ready)
- **Validation (TR/VL/TS) (NOT READY yet!!!)** both for type A and B

Play with a simple NN

- <https://playground.tensorflow.org/>
- Enjoy



*"Tinker With a Neural Network Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise."*

General hints: benchmark



- Difficulty of assessing implementation correctness
 - Because often a NN still works (even if hampered) with small implementation errors
- A first test (*"collaudo"*) **MONK's Problems Data Set**
 - (the results must be reported in the prj report)

<http://archive.ics.uci.edu/ml/datasets/MONK's+Problems>

- 3 tasks binary classification, small artificial data set, "not difficult" (a small NN with few units achieve a very high accuracy, up to 100%, with small time of convergence)
- There is a report with previous results using MLP (and others ML models):
 - chapter 9 <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.2363> *
- Input encoding: 6 variables with 2, 3 or 4 symbolic values each: 1-of-k → correspond to introduce 17 input units (see also sec 1.1 and check it!)
- TS includes TR, which is a bad practice, but does not change here (since 100% accuracy is 100% accuracy also on the test set !!)
- Other sources of data sets for software tests: UCI Machine Learning Repository (see later)

Other suggestions for your first trials



- That results in the following were obtained with 1 output unit for classification (using 0/1 or -1/+1 encoding both for target and outputs of course!):
 - *tanh* is suggested for the output unit
- (repetita) 1-hot encoding for the inputs (previous slide): *frequent fault!!!*
- Batch with grad/#patterns allows to converge in few hundred of epochs
- How many hidden units? Follow the manual as baseline (very few units: 2-5!!!)
 - The test works if you achieve the state of the art results with few units
- MONK can be sensible to weights initialization (due to the use of few units),
 - So, for some random initialization with high range the convergence could be slow→ try small initial random weight ranges (or using fan-in) [see lect.s before*]
- If you use on-line approaches use shuffling (patterns are ordered according to the first and second class)



NOTE!

- In the following QUALITATIVE results
 - Does not care about the values in itself, just the shape of the learning curve to see if you are close or far from a good convergence!
 - Note: student simulator results and the results from some well-know libraries are similar.
 - Note that ALL the plots are smooth in the MSE
 - We also use regularization (with a penalty term) in case of Monk3
- ❖ Good results on the MONK benchmark does not guarantee the simulator correctness
- ❖ Bad results on the MONK benchmark for sure requires revision of the code/setting

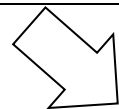
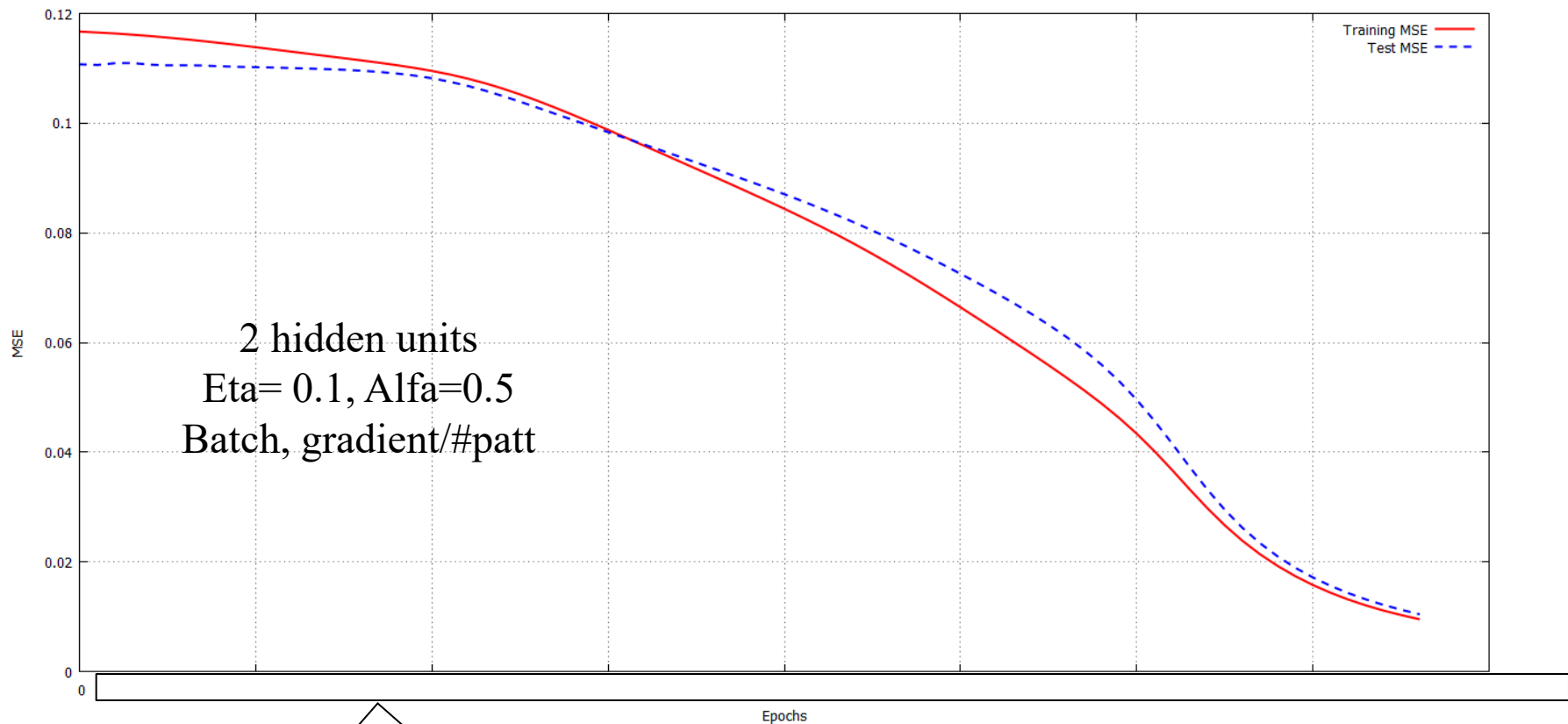
Monk2: example (MSE)

Student simulator



Dip. Informatica
University of Pisa

- MSE vs epochs



The numbers are intentionally omitted, don't care now
Depending on many factors could be e.g. 80-300/400 epochs

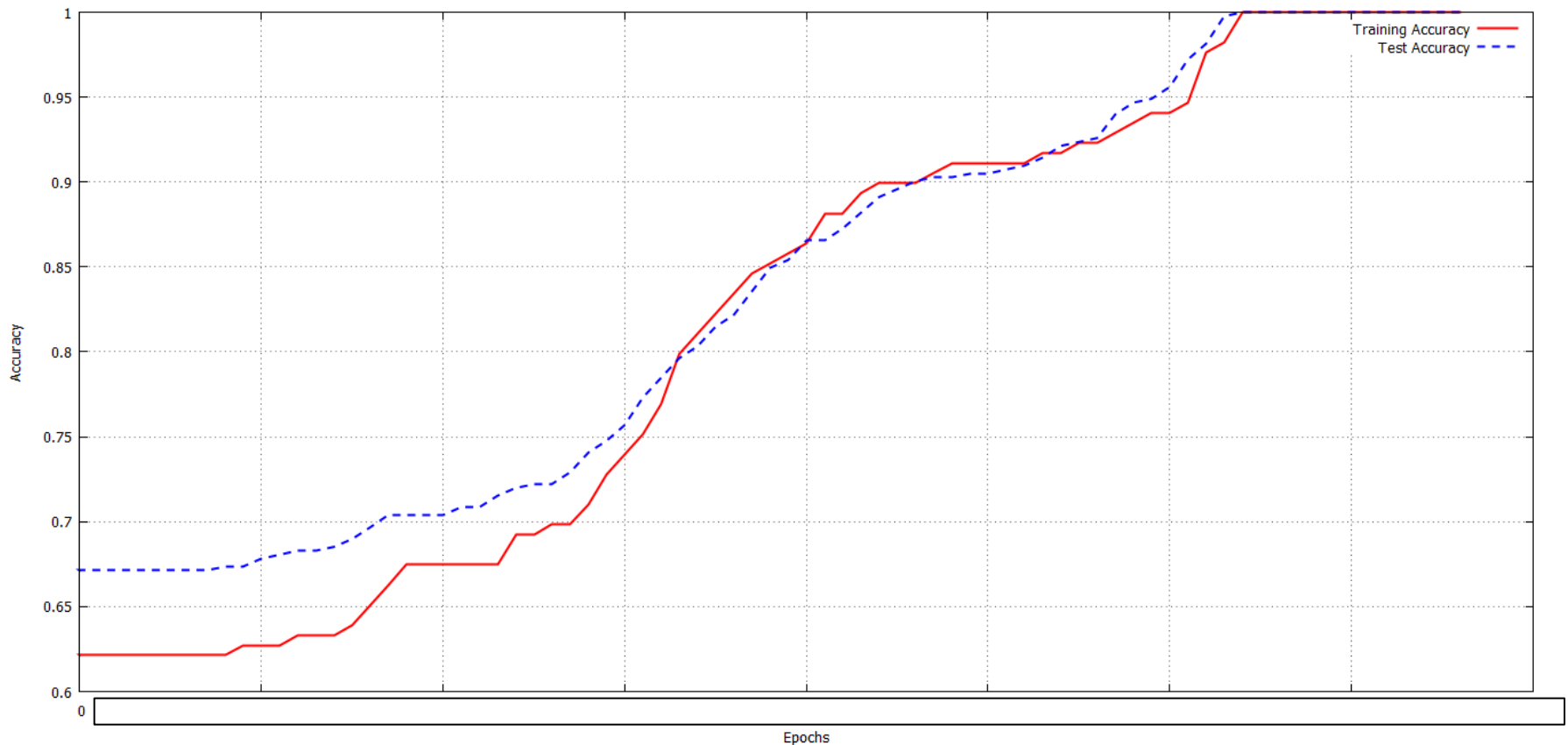
Monk2: example (Accuracy)

Student simulator



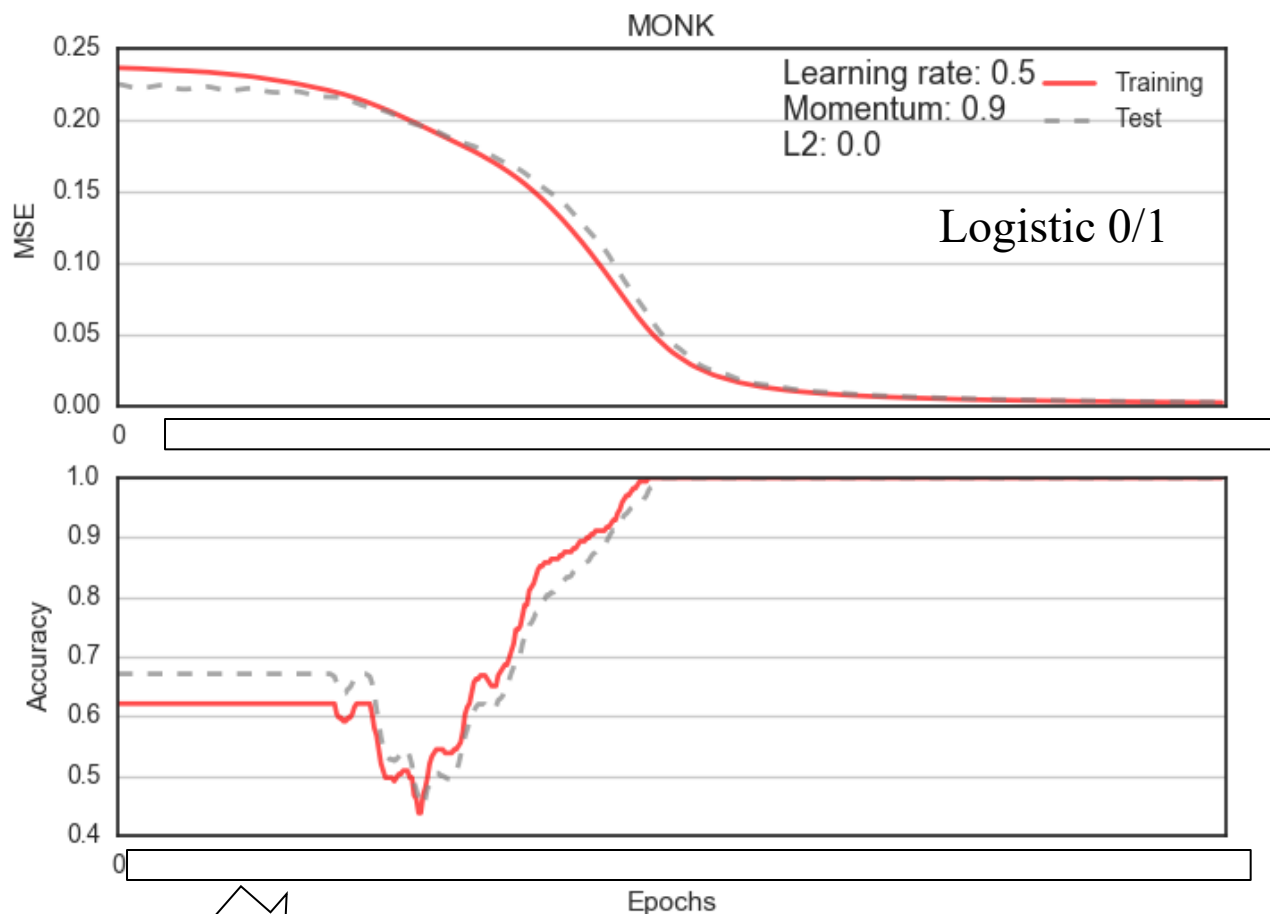
Dip. Informatica
University of Pisa

- Accuracy vs epochs. 100% accuracy is achieved on test! (which is *unusual* in general)





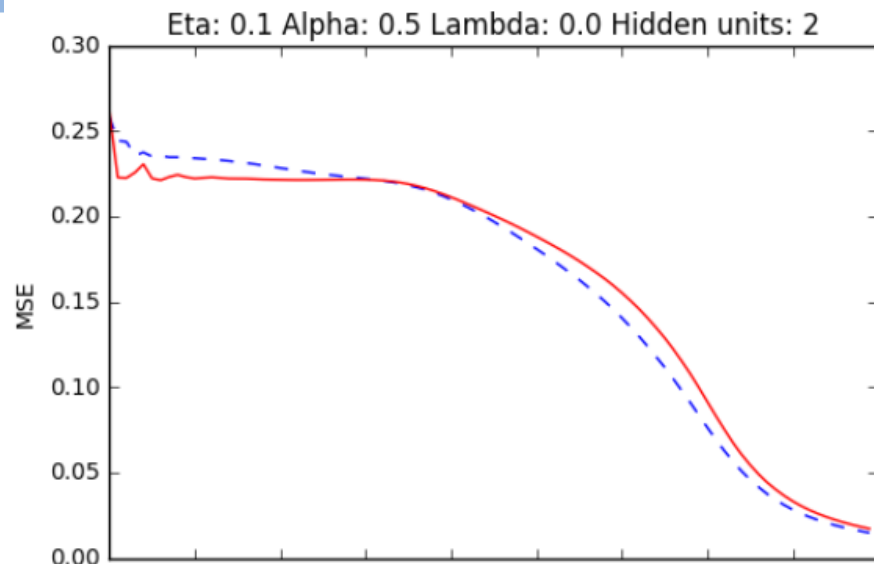
Monk2: With Theano



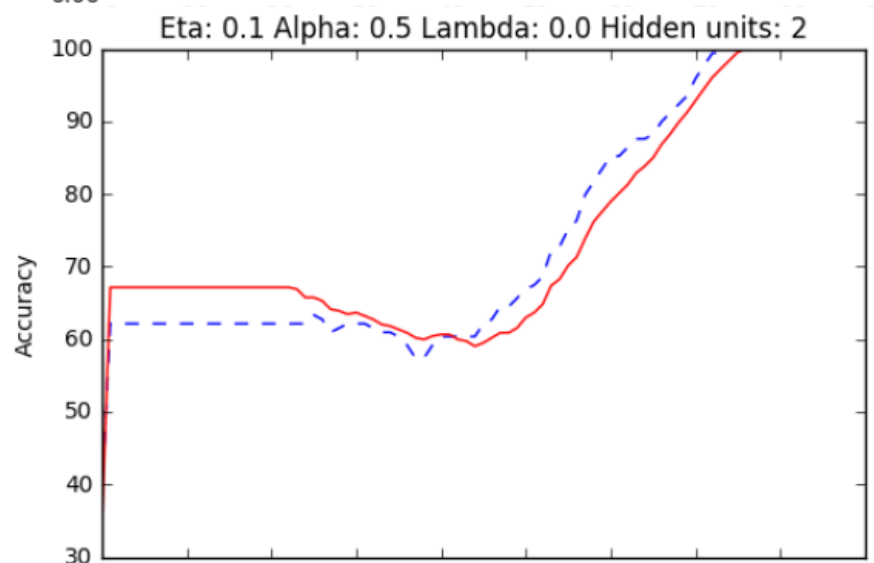
The numbers are intentionally omitted, don't care now
Depending on many factors could be e.g. 80-300/400 epochs



Monk2: Student simulator

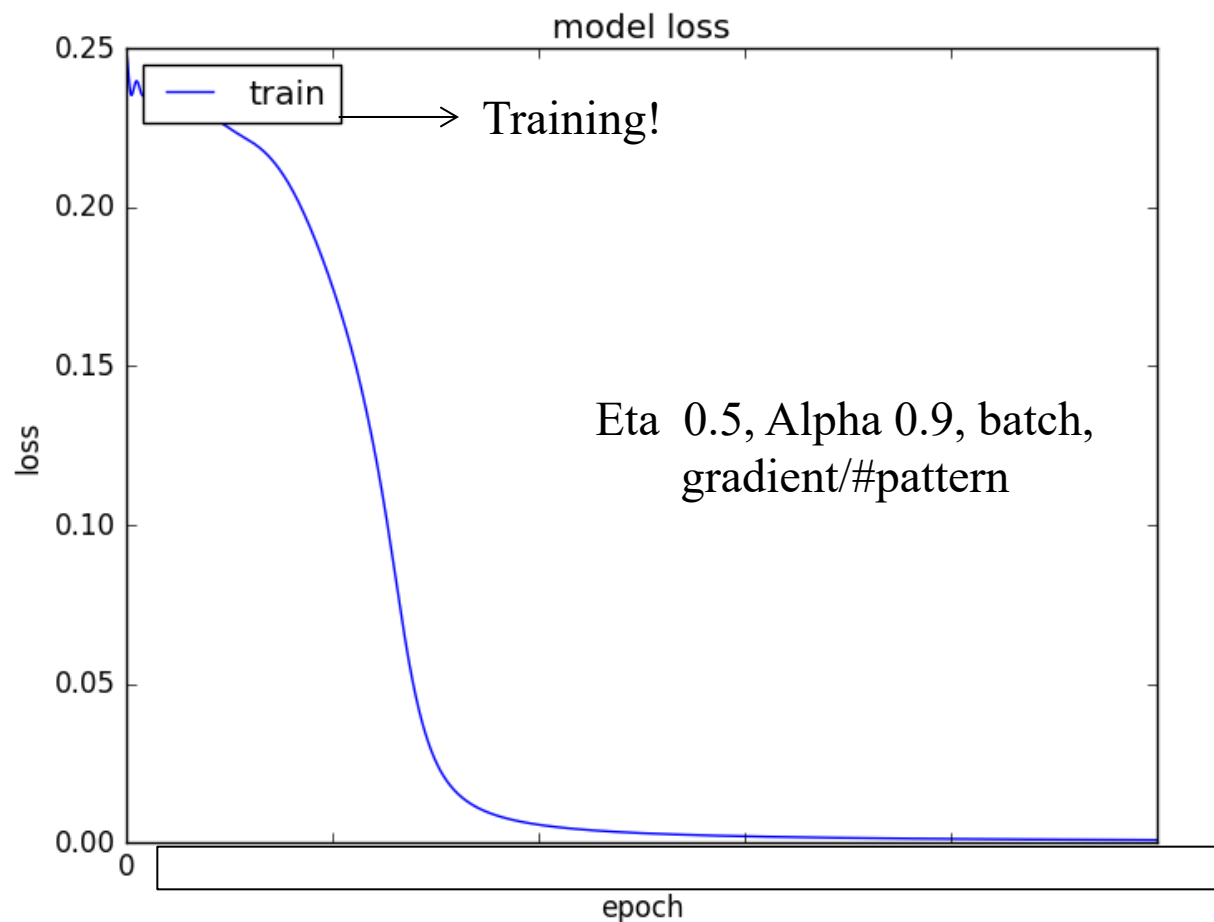


Red is test
Blue training





Monk2: Keras



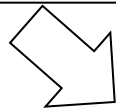
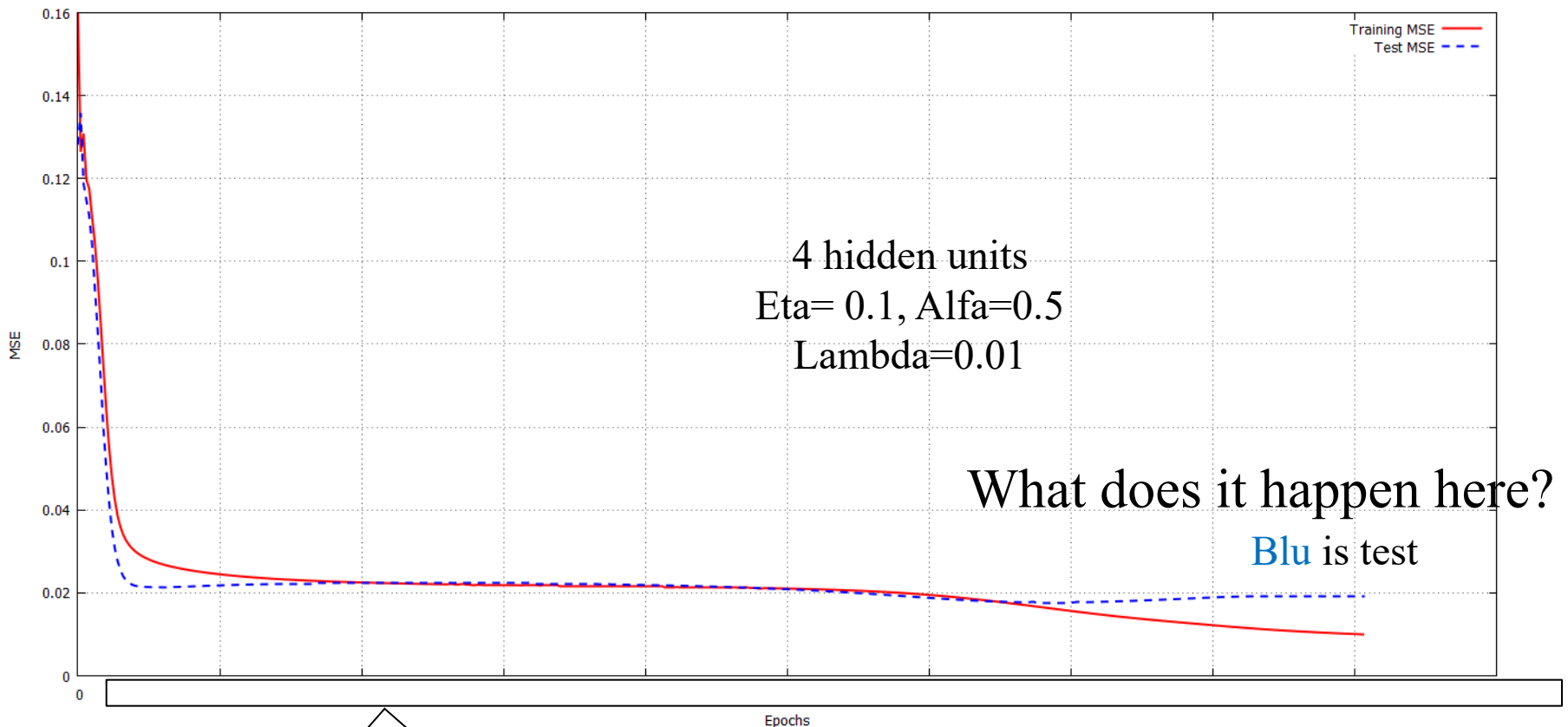
Monk3: example (with regulariz.)

Student simulator



Dip. Informatica
University of Pisa

- MSE vs epochs (monk3 can occur in overfitting, test is **not** at 100%, but improve with lambda)

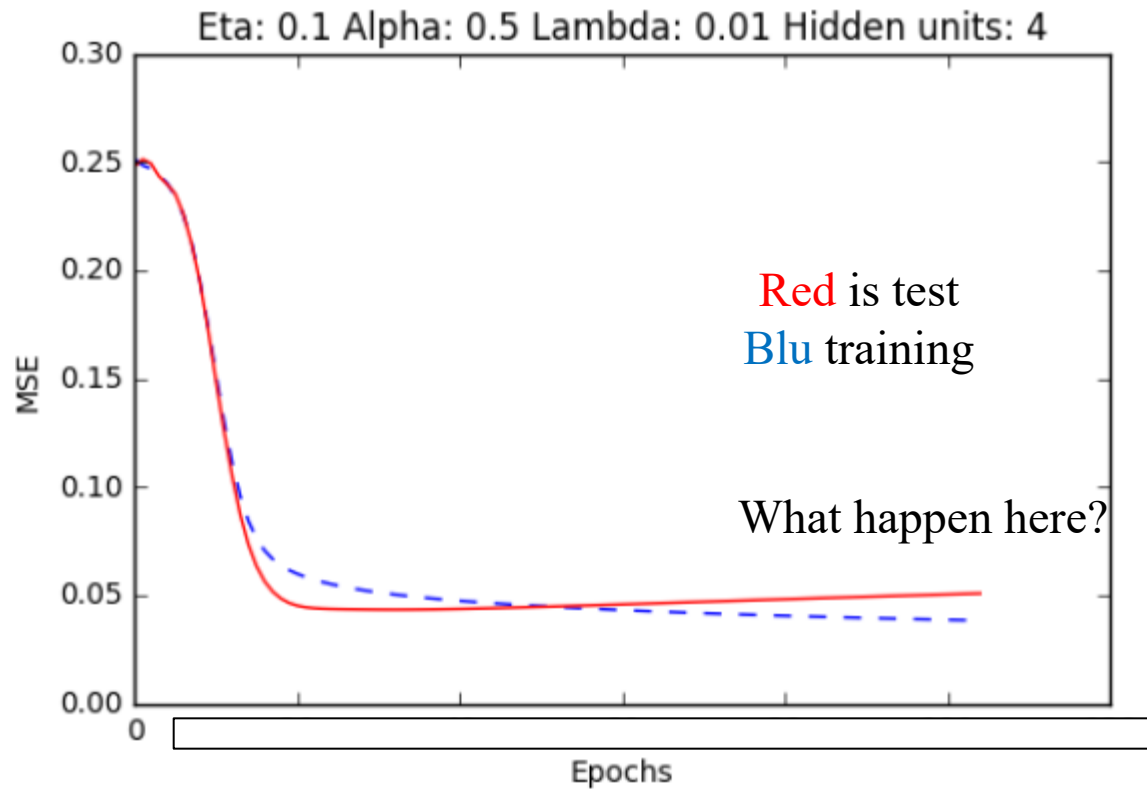


The numbers are intentionally omitted, don't care now
Depending on many factors could be 100-300 epochs

Monk3: a different student simulator



Dip. Informatica
University of Pisa



(See more in the PRJ slides) **Data Sets: other benchmark instances**



Dip. Informatica
University of Pisa

You can also select a benchmark useful to your case (e.g. to work with many data, many features etc.) from a public data sets (with hundreds of data sets), e.g.

UCI Machine Learning Repository: Data Sets

<https://archive.ics.uci.edu/datasets>



Examples (NOT requested for the PRJ)

DataSet Name	N. Instances	N. Attributes	Attribute (type)	N. Classes
Adult/Census	48842	14	Categ./Intero	2
Mushroom	8124	22	Categorical	2
(WBC) BreastCancer Wisconsin (Diagnostic)	569	32	Real	2
Nursery	12960	8	Categ.	5 (4)
Spambase	4601	57	Real	2
Zoo	101	17	Categ.	7

Other practical hints

- You can start *to think about* NN implementation (the core) or to a first exploration/usage of simulators
- I will communicate soon also the name of the *course assistants*
- To check your code (backpropagation part) you can use this VERY INFORMAL example (suggested by a student, not verified):

<http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

Warning: in that exercise the bias (w_0) was omitted from the analysis, indeed it is need to develop the example also for the bias

- Or use any “reliable” library as an “oracle” (we will introduce them later)
 - **Keras (NN)** (running on top of TensorFlow, Theano, et al., see bibliography), **Scikit-learn**, Caffe, Torch, **PyTorch (NN)**, R, **Matlab**
We will discuss many more later
- NOTE: We will provide in the next lectures the guide for the *model validation*, that are fundamentals for any applications!

Textbook and material (4) (Repetita from Lect. 1)



Dip. Informatica
University of Pisa

For software libraries:

- Books that describe *Keras / Scikit-Learn*



- "Deep Learning with Python" (F. Chollet)
- "Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow" (A. Geron).

- Using *PyTorch*, NumPy/MXNet, JAX, and TensorFlow

```
@d2l.add_to_class(LinearRegressionScratch)
def loss(self, y_hat, y):
    l = (y_hat - y) ** 2 / 2
    return l.mean()
```



- "Dive into Deep Learning" (A. Zhang, Z. Lipton, M. Li, and A. Smola) (2023). Cambridge University Press. Freely available from the website.

- We will have an introduction to the main libraries later (by the course assistants)



Many technical books/blogs exists, take care of the scientific quality!

PRJ: a preview!

- In the following: few slides for some main FAQs, (mostly repetita from the introduction of the course) but we will have a ***specific lecture on the project later!***

Assessment methods

(REPETITA from introduction)



Dip. Informatica
University of Pisa

Exam:

- **Project** (Written exam)
 - Students have the **opportunity** to develop a project realizing/applying a learning system simulator (typically a simple neural network) and to validate it through benchmarks. A written report will show the results.
 - Great *opportunity* to apply the concepts by yourself
 - Great *opportunity* to show your concrete understanding and effort for the exam
 - **Deadline:** ~(around /or more than)10-14 days before the oral exam session (see the Moodle folder of your session for the exact deadline)
 - **See details in the lecture for project presentation**
 - Last years: **competition** with *blind-test*
 - which is part of the benchmark results in the prj
 - also some joint proposals with CM course
- **Oral exam** (date according to the exams sessions):
see introduction slides



The main hints



- Follow the lectures and slides as a guide, studying *progressively* during the course
 - Special interactive classes will be used to assess activities and make a ML discussion forum (**Q&A classes**). This is a class, not a set of records*.
 - Within a **cooperative** and collaborative context!
 - The **intermediate tests** are also meant to stimulate you to **stay "online"** and to have constant *self-evaluations*

- A major hint *from past students*:

1. FIRST study the course content

2. THEN apply for the project

- This help to boost efficacy and efficiency of your work

PRJ Preview: Possible Aims

A) Realize a NN model simulator and apply it (**implementation**)

- Programming language is a free choice (C++, Python are popular for ML, or even environments as Matlab or R can be considered)

B) Extensive experimental applications of existing ML/NN simulators or code made by LLM (): (**comparison**)

- You will consider different models as NN, SVM, K-NN, ...
- Simulator/s is/are a free choice (a list will be discussed within prj lecture)

C) Contact me (e.g. PhD students etc)

A) or B): participate to the “**ML cup**” competition.


Details for A) B) [C)] will be discussed in the Prj lecture

PRJ Preview: other FAQs:

REPETITA



Dip. Informatica
University of Pisa

- **Groups:** you will participate in groups of 2/3 people.
 - Please, search immediately a partner or parterners
 - See FAQ section in the Moodle to search a partner
 - You can use the link to the shared file (see the FAQ)
 - **Repetita: Deliverables:** ~around / or more than 10-14 working days in advance w.r.t. to the oral exam: BUT the exact and strict deadline will be indicated in the Moodle folder for each session.
 - ...questions? 
- Details will be discussed later in the Prj lecture

DISCUSSION ON NN

NN historical notes

- See Haykin (end of chapter 1) and Russel, Norvig book
- **E.g.** McCulloch and Pitts (1943), Hebb (1949), Minsky (1954 and 1969), von Neumann (1956), Kohonen (1972 and 1982), Rosenblatt (1958), Widrow and Hoff (1960), Grosseberg (ART 1980), Hopfield (*) (1982), Rumelhart, Hinton, Williams, Werbos, Parker, LeCun and others (1975-1985 for backprop), Poggio and Girosi (RBF-theory 1990), Vapnik (90s' SVM), ... deep learning (see later, Hinton (*)(*), Bengio (*), LeCun (*) et al. 2007 and ahead) ... and many others!
(*) Nobel prize 2024; (*) Turing award 2018
- The field continues to grow in theory, new models and applications... *you are welcome!*

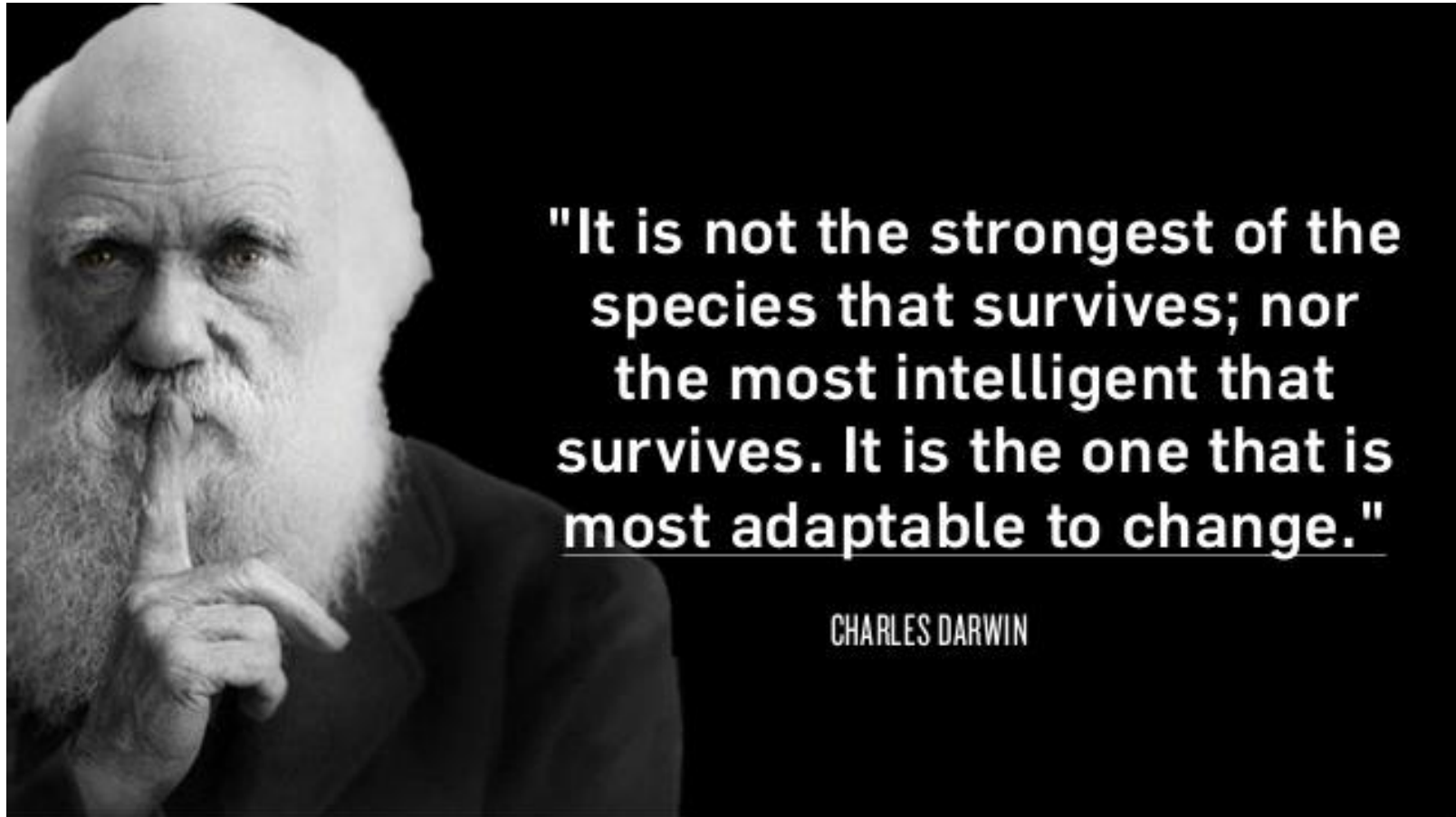
NN summary

- NN provide **flexible method** for ML
 - extend linear methods to arbitrary function estimation
 - construct an explicit hypothesis (with respect to K-NN)
- Expressive power: num. of units + architecture + training
- Hypothesis space is a rich continuous space of functions + loss function is differentiable → Training based on error minimization by gradient descent technique

NN summary (+)

- NN can learn from examples
- NN are universal approximators (Theorem of Cybenko):
flexible approaches for arbitrary functions (including non-linear) !
- NN can deal with noise and incomplete data
 - Performance degrades gracefully under adverse operating conditions
- NN can handle both continuous real and discrete data
for both regression and classification tasks
- Successful model in ML due to the flexibility in applications
- NN encompasses a wide set of models: it is a paradigm !
(in the class of subsymbolic approaches)
- E.g. also unsupervised tasks, associative memories, stochastic
approaches,

NN after more than 70 years?



Others (+)

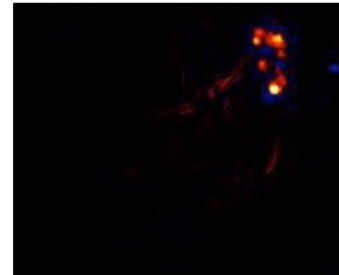
- Neural plausibility
- Distributed representation of knowledge
 - The knowledge acquired is represented compactly as weight matrices
- Fault tolerance – robustness (related also to distributed representation)
- Adapt to changes (nonstationary environment): on-line learning is possible (a.k.a. *continual learning*) → stability-plasticity dilemma

NN summary (-) critical points

- Black-box problem (inherent inability/difficulty to explain, i.e. it is difficult to interpret the acquired knowledge):
 - extraction and representation of knowledge for human experts
 - related to the current trend in the research for interpretability of ML models and the “*explainable artificial intelligence*”, **XAI**, issue

Faced by many approaches, e.g.:

- Projecting/clustering internal representation
- Computing sensitivity analysis (since 1995)
- LRP: Layer-wise Relevance Propagation [Bach et al. 2015]. Also for Deep NNs. Klaus-Robert Müller @ ICANN 2019 “*NN’s black box issue is not sense, not true anymore*” (<http://www.heatmapping.org/>) (see figures below)
- Discovering symbolic rules associated to the trained model !!!
- ...



Why is it a rooster?
Rooster crest in the heatmap
of the internal NN representation

NN summary (-) critical points (cnt)



Dip. Informatica
University of Pisa

- Architectural design :
 - Constructive/pruning methods to optimize the architecture
 - New approaches (e.g. SVM, we will see later)
- Training process dependency: training realization (we have seen many variants and possible values of the hyperparameters) has effect on the final solution
- Not the best for Missing data: standard strategies including removing data, imputations, ...

Conclusions

- When to consider NN:
 - Input is high-dimensional discrete or real-valued / classification and regression tasks
 - Possibly noisy data
 - Training time is not critical
 - Form of target function is unknown
 - Human readability of result is not critical
 - The computation of the output based on the input has to be fast

REPETITA: Bibliography (NN second part)



Dip. Informatica
University of Pisa

- Haykin: chapter 4
- Mitchell: chapter 4
- Hastie, Tibshirani, Friedman : cap 11
- *NEW*: some insertions from the Deep Learning book.
See details and references to sub-sections within the slides



— Specific sections:

- **Backpropagation**: Parallel Distributed Processing, Vol.1, by D. E. Rumelhart, J. L. McClelland, MIT Press: Chapter 8 by Rumelhart, Hinton, Williams (1986)
- **Cascade correlation**: S. E. Fahlman and C. Lebiere: The Cascade-Correlation Learning Architecture CMU-CS-90-100 – Carnegie Mellon University, 1991
Also in NIPS 2, 1990, pages 524-532

Others:

- **Quick-Prop**: S. E. Fahlman: An Empirical Study of Learning Speed in Back-Propagation Networks, TR CMU-CS-88-162, 1998
- **Rprop**: M. Riedmiller and Braun H. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In Proceedings of the IEEE International Conference on Neural Networks, pages pp 586–591, 1993.

Textbook and material (4) (Repetita from Lect. 1)



Dip. Informatica
University of Pisa

For software libraries:

- Books that describe *Keras / Scikit-Learn*



- "Deep Learning with Python" (F. Chollet)
- "Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow" (A. Geron).

- Using PyTorch, NumPy/MXNet, JAX, and TensorFlow

```
@d2l.add_to_class(LinearRegressionScratch)
def loss(self, y_hat, y):
    l = (y_hat - y) ** 2 / 2
    return l.mean()
```



- "Dive into Deep Learning" (A. Zhang, Z. Lipton, M. Li, and A. Smola) (2023). Cambridge University Press. Freely available from the website.

- We will have an introduction to the main libraries later (by the course assistants)



Many technical books/blogs exists, take care of the scientific quality!

Next for NN

Applications:

- **CNN example**

Introduction to recent NN paradigms

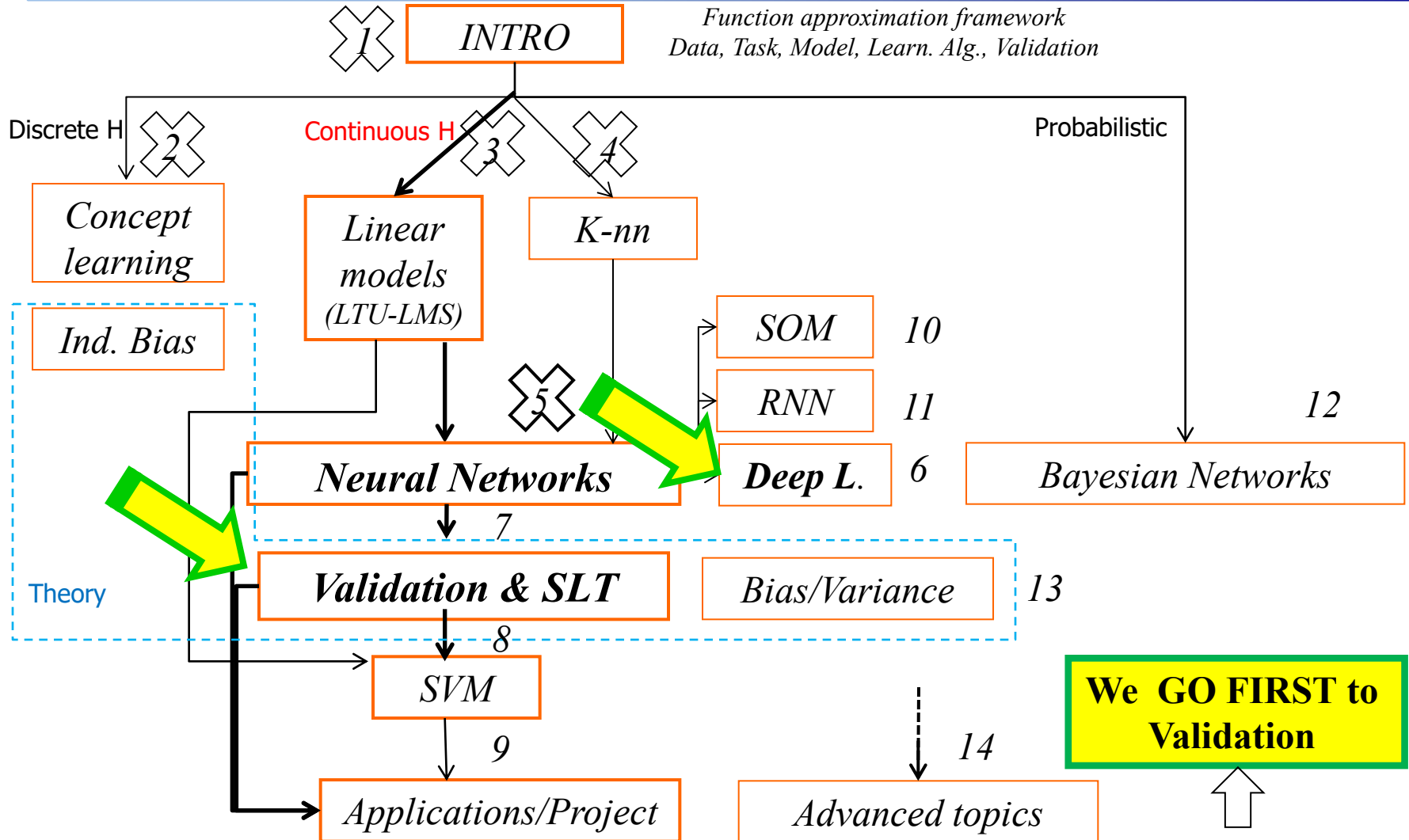
- Other architectures.....
 - **Deep learning**
 - **Random weights NN**
- But we will anticipate the **Validation part!!!**
(so to favorite prj starting)

ML Course structure

Where we go



Dip. Informatica
University of Pisa



Notes on Neural Networks: part 2

the end

Alessio Micheli

micheli@di.unipi.it



Dipartimento di Informatica
Università di Pisa - Italy