# Deep Learning in action

Basic Deep Learning concepts & methods + practical tips

Dario Garcia Gasulla
*dario.garcia@bsc.es*

# A Bit of History

The not so distant origin story

# Connectionism & Hebbian Learning

Warren McCulloch & Walter Pitts (1943):

- From neurons to complex thought
- Binary threshold activations

Howard Hebb (1949):

- “Neurons that fire together wire together”
- Weights yield *learning* and *memory*

A Logical Calculus of Ideas Immanent in Nervous Activity

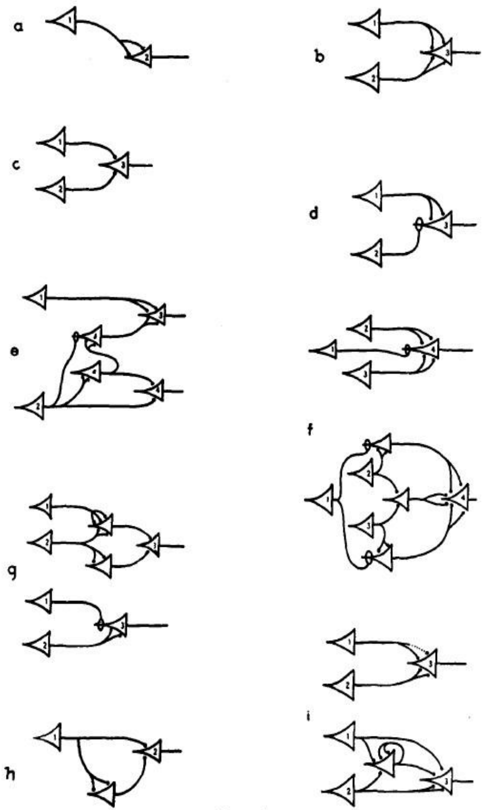



FIGURE 1

Ref: [1,31]

# Rosenblatt's Perceptron

Rosenblatt (1948): Hebb's learning + McCulloch & Pitts design

Mark I Perceptron

- 400 photosensitive receptors (sensory units)
- 512 stepping motors (association units, trainable)
- 8 output neurons (response units)

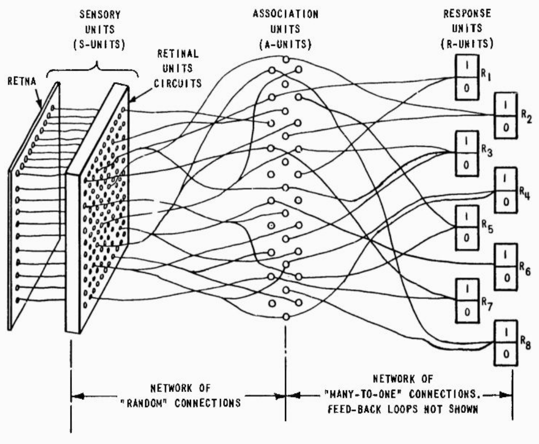
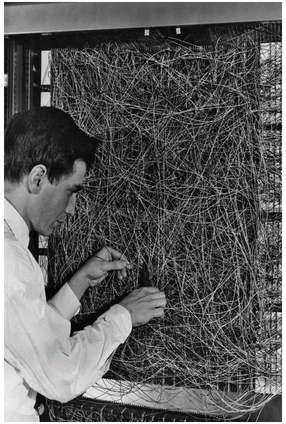


Figure I ORGANIZATION OF THE MARK I PERCEPTRON

Threshold function

$$f(x) = \begin{cases} 1 \text{ if } w \cdot x + b > 0 \\ 0 \text{ otherwise} \end{cases}$$

*w* real-valued weights
· dot product
*b* real scalar constant

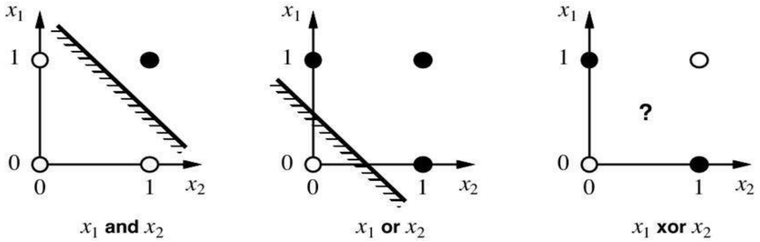


[2,3,4,5]

# Minsky & Papert: The XOR affair

The perceptron capabilities were limited (Rosenblatt)

Linear separability




"Perceptrons: an introduction to computational geometry"

(Minsky & Papert, 1969):

- Perceptron cannot learn non-linearities
- Multi-Layer networks cannot be trained

NNs abandoned until the mid 80s: ***1st AI WINTER***

Shift from connectionism to symbolism

# Backpropagation Algorithm

- How to optimize weights not directly connected to the error?
- Backpropagation algorithm:
  - Use the chain rule to find the derivative of cost with respect to any variable
- Optimization through Gradient Descent
- Used for training MLPs in (Werbos in 1974, Rumelhart et al. in 1985)

**End of NNs Winter (Begining of 2nd AI Winter)**

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

[7,8]

# Feedforward Neural Networks

# Optimization & Learning

Training through backprop (1) + gradient descent (2)

- **Forward pass**
  - Compute output for a given input
  - Error measurement (loss function)
- **Backward pass**
  - Find gradients minimizing error layer by layer (1)
  - Apply gradients (2)

# The Gradient Descent Family

**Batch GD:** Compute gradients of *all training samples* before descending

- Deterministic outcome
- Large memory cost

**Stochastic GD:** Apply gradient of *one random training sample* at a time

- Very stochastic
- Does not guarantee learning
- Poor parallelization

**Mini-batch GD:** Combine gradients of a *random subset of train samples*

- Mildly Stochastic
- Good parallelization
- Subset size aka *"Batch size"*

***WARNING**: This naming is not universally respected!*

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

# Mini-batch Training Nomenclature

Num. samples computed together: **The batch size**

One feedforward/backward cycle (one batch): **A step**

N steps (all training samples once): **An epoch**

$$N = \frac{dataset\_size}{batch\_size}$$

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

# Practical Tips I

Factors for defining the batch size (rarely)

- Instance size
- Computational efficiency
- Stochasticity
- Powers of 2

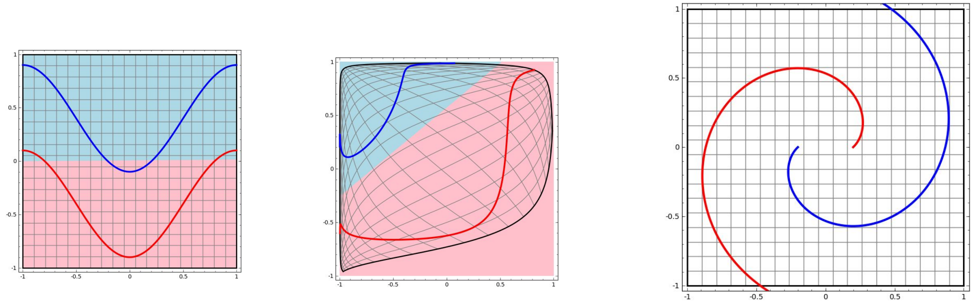
Factors for defining the number of epochs (constantly)

- Convergence
- Reliability
- Footprint

# The Manifold Hypothesis

Deep Learning is defined by the high input dimensionality

- How can we find solutions in such a vast space?
- Manifold Hypothesis:
  - The space we must navigate has a smaller dimensionality
  - Most possible inputs are unrealistic
- Each layer in a NN is a different manifold, transforming the space to facilitate the target task

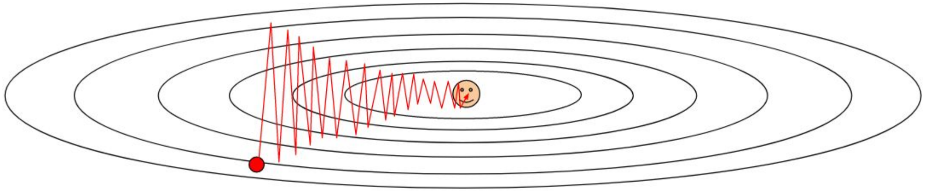


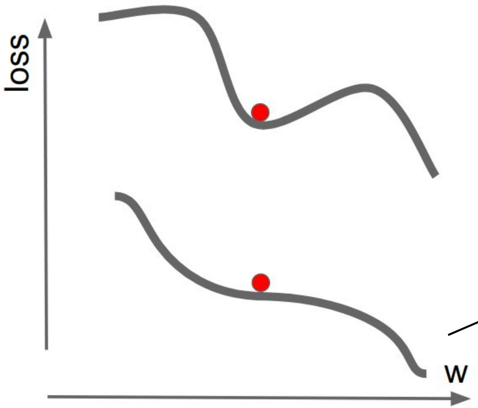
[32a,32b]

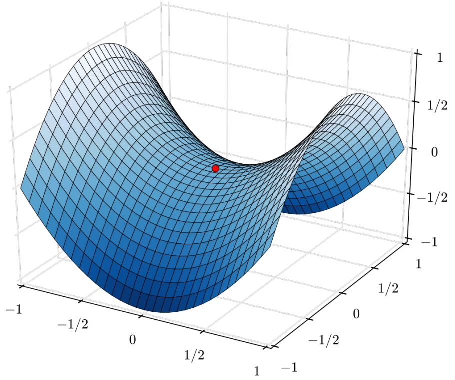
# The Art of Descending

Hard to go down in a high dimensional space

- ❖ Different loss speed among dimensions (imagine that!)
  - ■ Slow and jitter
- ❖ Local minima & saddle points
  - ■ Stuck gradient
- ❖ Mini-batches are noisy
  - ■ Back & forth





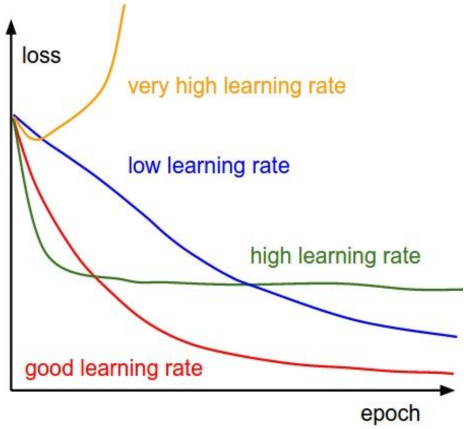



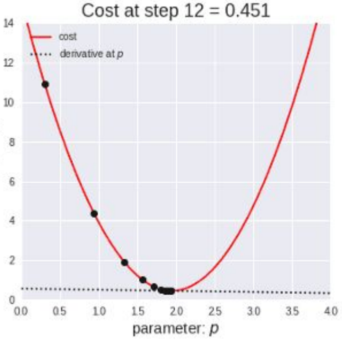
# The Speed of Descending

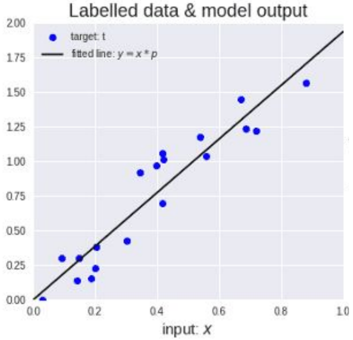
**Learning rate:** How much you move in the direction of the gradient

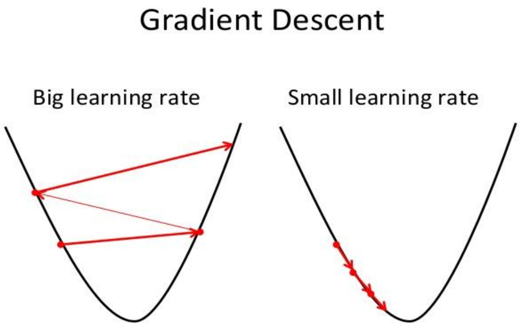
Direct effect on convergence and speed

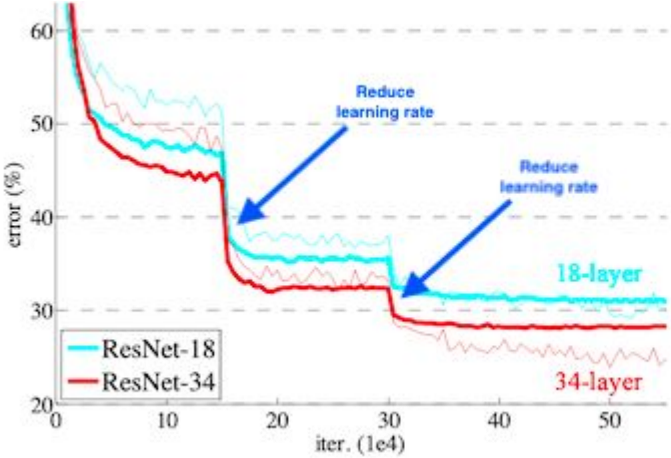
The same LR may not always be the right one!



# Practical Tips II

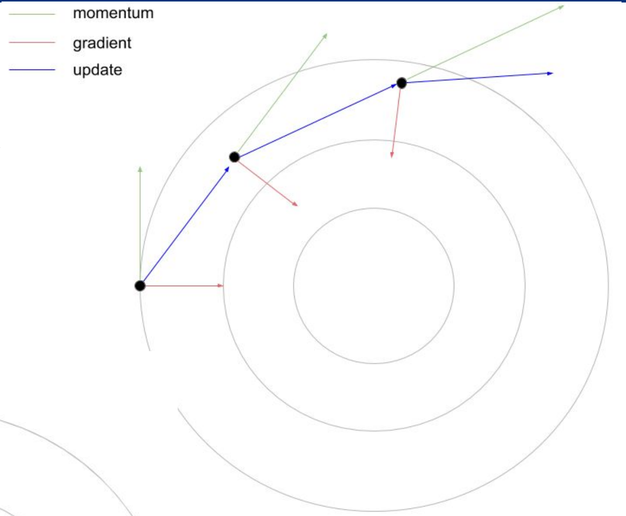
Tuning the Learning Rate

- Fix the batch size (or viceversa)
  - Theory: Double one, double the other
- Always smaller than 1
- Search by orders of magnitude
- Grid search < Random search
- In case of doubt, go small
- When stuck, reduce it

# Inertia in Optimizers

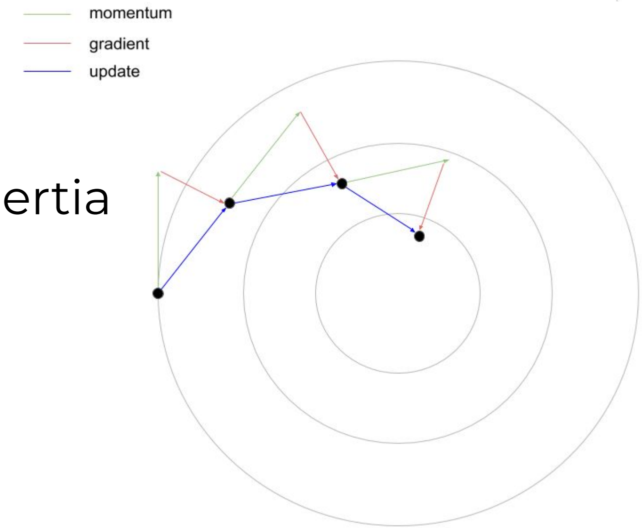
**Momentum**:

- Add a fraction of the previous gradient. *Inertia*.
- Decaying weight parameter. *Friction*.
- Faster, smoother convergence




**Nesterov**:

- Gradient computed after inertia
- See the slope ahead
- Faster convergence



# Adaptative LR Optimizers

**Adagrad**:

- Parameter-wise LR considering past updates.
- High LR for infrequent ones. Low LR for frequent ones.
- Good for sparse data. Vanishing step size due to growing history.

**Adadelta**:

- Adagrad with a decaying average over history (typically around 0.9)

**RMSprop**:

- Similar to Adadelta

**Adam**:

- Adadelta + Momentum

**Nadam**:

- Adadelta + Nesterov

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

# Practical Tips III

Optimizers

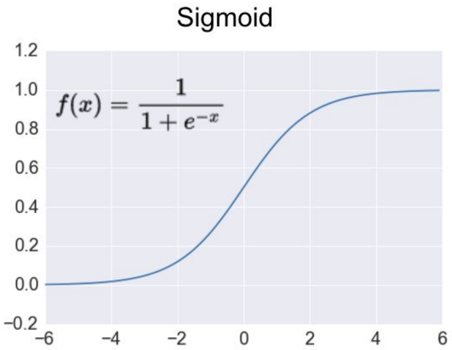
- Adam: Current popular default. Competitive with minimal tuning.
- SGD + Momentum: Great if LR is decayed properly

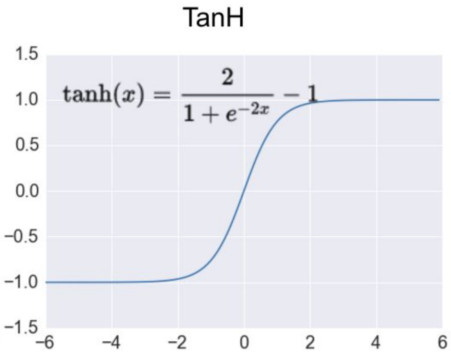
*Hyperparameters incomplete list* #1 (training)

1. Batch size
2. Number of epochs
3. Learning rate
4. Weight decay

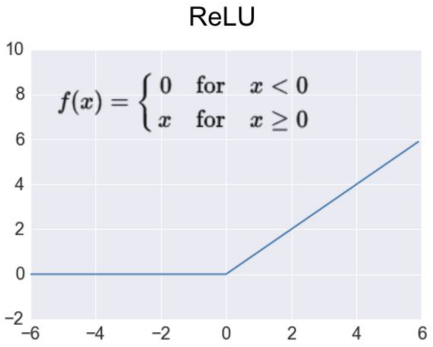
# Activation Functions

Transform the output of a layer to a given range.




- Zero gradient most of f(x): Saturates!
- Gradient is 0.25 or 1 max. Vanishes!



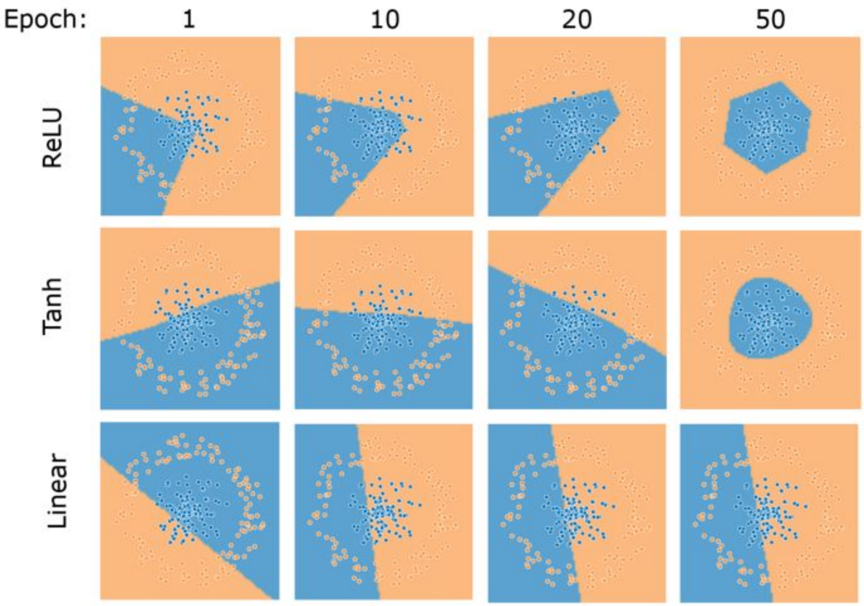

- Does not saturate
- Does not vanish
- Faster computation
- May die with high LR

ReLU is a safe choice in most cases

Undying alternatives: Leaky ReLU, PReLU, ELU, SELU, ...

# Why ReLU works

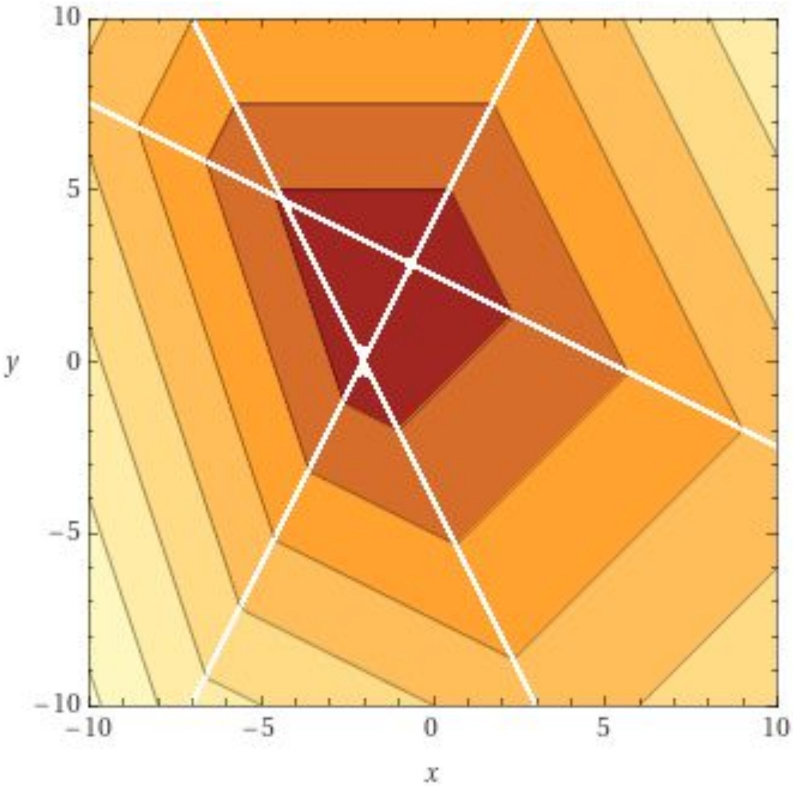
- But wait, ReLU is linear, and we need non-linearity!
- Not exactly. It's piecewise linear.
- ReLU can bend linearity
  - On one point
  - With any angle
- Just need a bunch of ReLUs

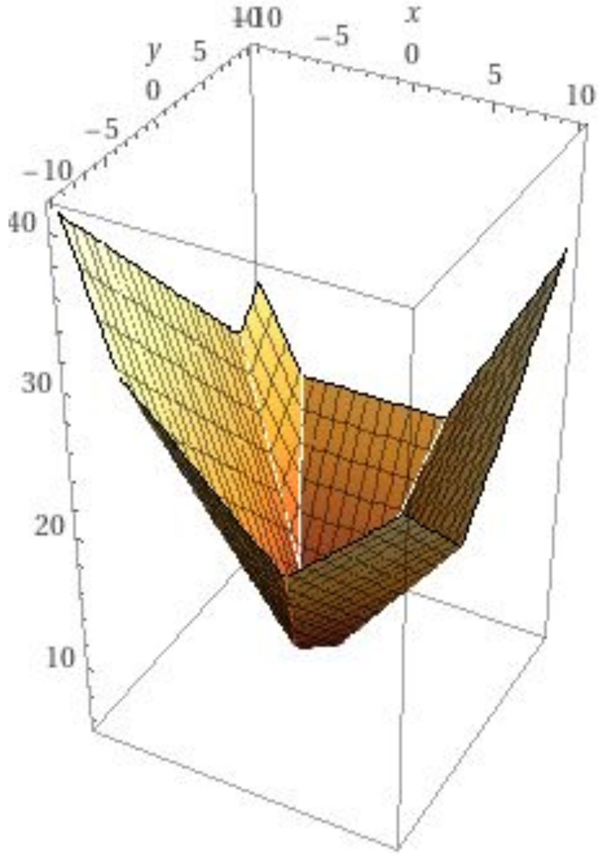



[37c]

# ReLU: Composing non-linearity

ReLU(-4-2x+y) +
ReLU(4+2x+y) +
ReLU(5-x-2y)

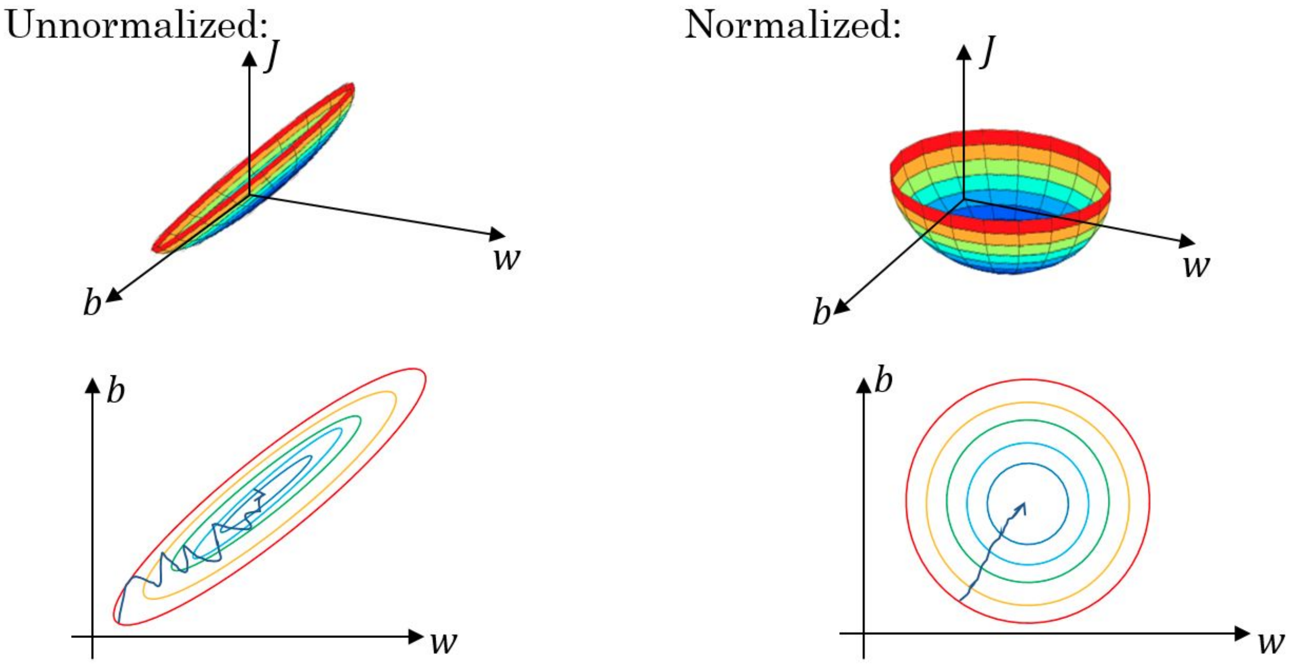



[30]

# Input Pre-processing

Make your model's life easier. She would do it for you.




Options

- Mean substraction
- Normalization
  - - mean / std
  - Image-wise?
  - Channel-wise?

Also, beware of the *imbalance*!!

# Weight Initialization

We want:

- Small numbers
- Half positive, half negative

What about bias?

- If weights are properly initialized, bias can be zero-init.

Options :

- ~~Constant values~~: No symmetry breaking.
- ~~Zeros~~: No gradient flow
- Normal distribution sample: Ok for shallow, but deviation grows with size
- Glorot/Xavier: Normalize variance by number of inputs
- He/Kaming/MSRA: Specific for ReLUs.

# Practical Tips IV

- Start with ReLUs. Explore variants as a long shot.
- Always zero-mean the data. Or normalize (init depends on it)
- If using ReLUs, *He* init. Otherwise *Glorot*.

*Hyperparameters incomplete list* #2 (initialization & preprocessing)

5. Activation function
6. Input normalization
7. Weight Initialization

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

# Regularization

Why do we need regularization?

- **Generalization:** Difference between *Machine Learning* and *Optimization*
  - We want to learn the “good” patterns
  - Neural nets are lazy. They will always go for the “easy” patterns
  - Generalization is a sweet spot that may be unreachable

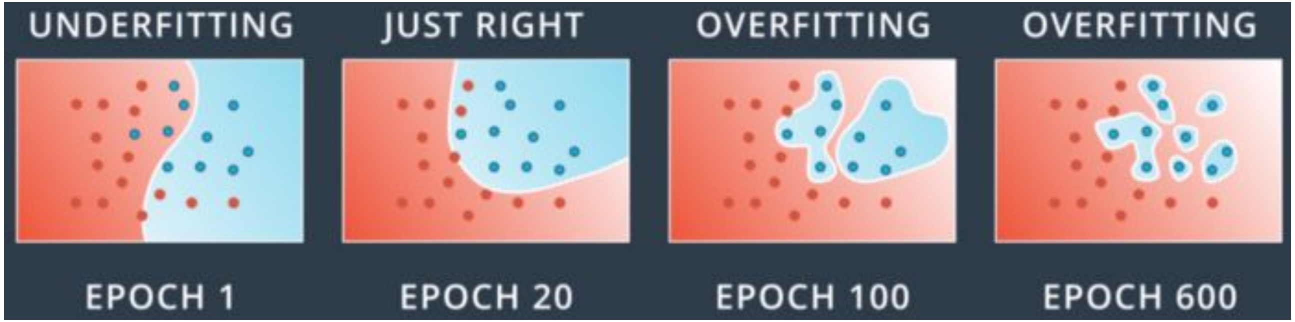
# From underfit to overfit

**Underfit**: Insufficient learning of training data patterns

**Overfit**: "Excessive" learning of training data patterns →

Key players:

- ❖ Model capacity
- ❖ Data input
- ❖ Regularizers



# Train, Test and Val

Doing a good train/val/test split is not easy! Take your time & do it right.

**Training set**

- Data used by *the model* for learning parameters
- Keep an eye for variance (spurious patterns)
- As large and varied as possible
- Use mostly as a sanity check
- Overfitting is inevitable
  - Sometimes it's desirable!

**Validation set**

- Data used by *you* for tuning hyperparameters
- Size entails reliability
- Overfitting is possible

**Test set**

- Hide under a rock
- Must be 100% independent
- Run once. Cite forever.

# Practical Tips V

Never, ever, ever

- Mix correlated data in train/val/test
- Process data in an order
  - Do shuffle with seed!
  - Reproducibility for your own sake
- Believe train results generalize
- The dataset is free of bias
- Assume balanced dataset

# Practical Tips VI

Training milestones

1. Learn, **anything**! (Train set)
   - Little capacity makes it easier
   - Rough hyperparameter estimation
   - *Goal*: Underfit
     i. Better than random

2. Learn, **everything**! (Train set)
   - Growing capacity
   - Hyperparameter refinement
   - *Goal*: Overfit

3. Learn **the right thing** (Val set)
   - Regularization
   - *Goal*: Fit

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

# Back to Regularization

Takes us from overfit to fit

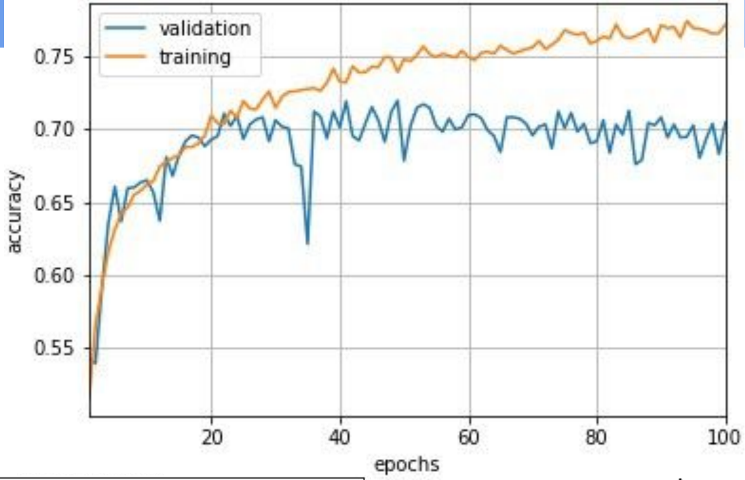
The must do ones (if overfitting):

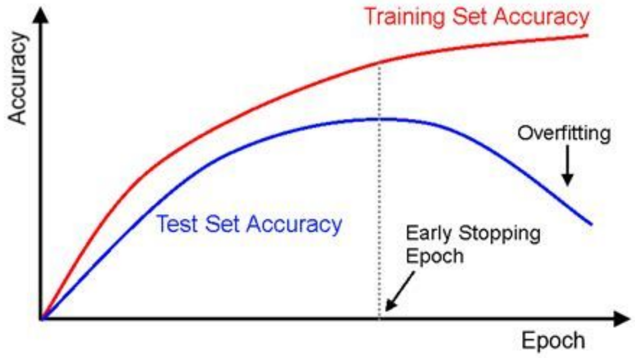
- **Early Stopping**
  - Overfitting is the end of the road
  - The guide: Validation loss/accuracy
  - Enough to understand the model (mind the footprint!)
- **Data Augmentation**
  - More data for free
  - Huge impact
  - As any data preprocessing, think thoroughly!

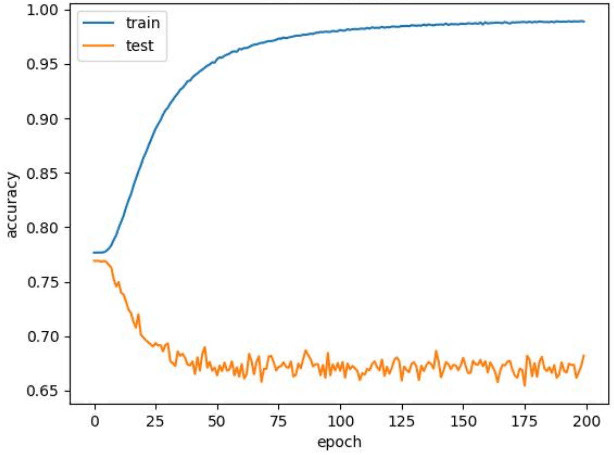
# Practical Tips VII

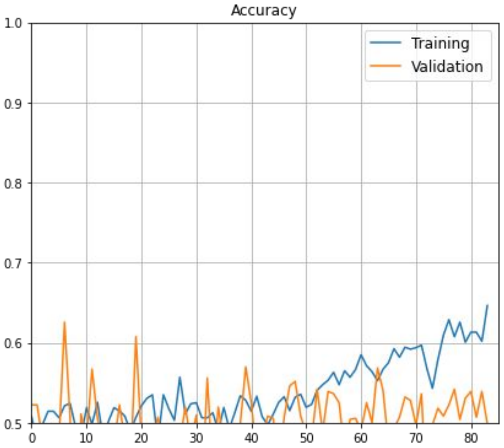
Diagnosing the curves (loss & acc.*)

- Random performance?
- General trend?




HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

# Practical Tips VIII

- Fails to converge? No trend?
  - Simplify problem/model. Decrease LR.
  - Data corrupted? Pre-processing? Weight init?

- Loss explosion? Sudden spike?
  - Problematic data instances.
  - Exploding gradients ⟹ weights.

- Loss goes down and accuracy goes down (what??)
  - Raw outcome improves, but threshold metric is not met
  - Imbalance?

**Weird curves are the worst!**

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

# Regularization through Norms

Add a regularizing term to the loss function

- **L2/L1 norm** on weights by factor (another hyperparam!)
- Makes gradients and weights smaller (L1 sparsifies inputs)
- Safe for SGD, not so for adaptive learning optimizers (e.g., Adam)
  - Uneven normalization :S

**Weight decay** is similar to L2-norm (often confused)

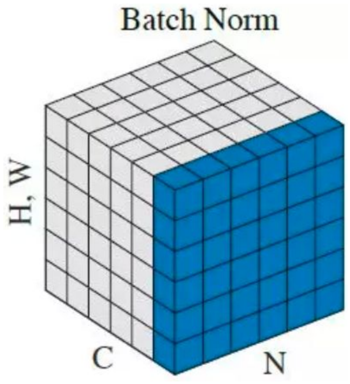
- Scaling factor on the update rule (another hyperparam!)
- Analogous to L2-norm for SGD, not for adaptative optimizers
- Theoretically safe for all (if implemented!)



[10,11,12]

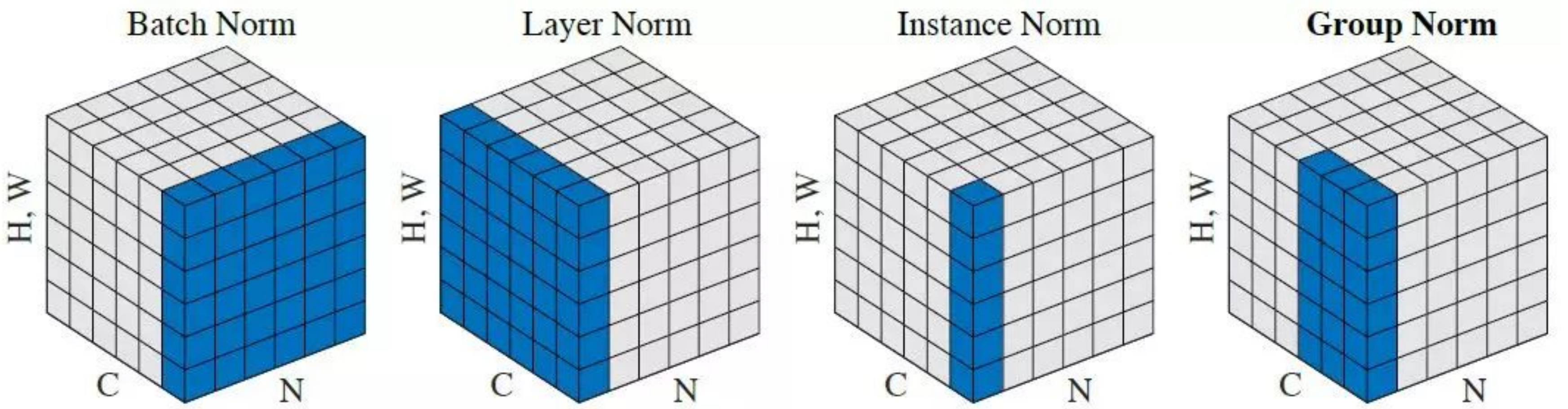
# Batch Normalization

Force the *activations* within a normal distribution




- Applied between neurons and activation functions
- Statistics computed per mini-batch (practical reasons)
- Helps with initialization and regularization
- Allows higher LR &  faster convergence
- Requires minimum & fixed batch size (does it?)

# Norms, Norms, Norms




All instances
One channel

One instance
All channels

One instance
One channel

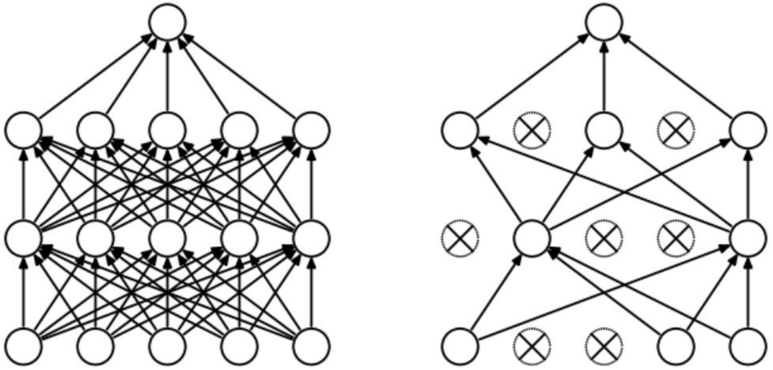
One instance
N channels

And more!!

# Dropout

Cut-off neuron inputs with a given probability

- Dropout rate (typically between 0.2 and 0.5) applied on every step
- In practice trains an ensemble of networks with “shared” params
- Inference: Use all inputs, scaled by the same probability
- Reduces co-adaptation between neurons
- Slows down training (a lot)
- Affects many other hyperparameters
- Good on FC layers. Not on Convs.



(a) Standard Neural Net (b) After applying dropout.

# Practical Tips IX

One experiment at a time

1. Analysis
   - What is wrong/improvable?
   - How can it be solved/achieved?
2. Test
   - Which alternative works better?
   - Ablation study: Alone or combined?

*Underfitting*

- ❑ Initialization
- ❑ LR, batch size
- ❑ Complexity up

*Overfitting*

- ❑ Regularization
- ❑ Complexity down

# Learning what?

Loss/Cost/Objective/Error function defines the optimization goal

- N-way Classification
  - Softmax (outs N probabilities) + Cross-Entropy (in N neurons)
- Regression
  - Mean Squared Error (in 1 neuron)
- And so many others!

# Practical Tips X

*Hyperparameters incomplete list* #3 (capacity, regularization and loss)

8. Network *capacity* (layers, neurons)
9. Early stopping policy
10. Data Augmentations
11. Normalization layers + hyperparams
12. Loss function

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

**Dario Garcia-Gasulla** (BSC)

*dario.garcia@bsc.es*

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

# References


[1] http://vordenker.de/ggphilosophy/mcculloch_a-logical-calculus.pdf
[2]http://www-public.tem-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/Rosenblatt58.pdf
[3] http://www.dtic.mil/dtic/tr/fulltext/u2/236965.pdf
[4] https://en.wikipedia.org/wiki/Perceptrons_(book)
[5]http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/
[6] https://en.wikipedia.org/wiki/Perceptrons_(book)
[7] Werbos et al. "Beyond regression:" new tools for prediction and analysis in the behavioral sciences." Ph. D. dissertation, Harvard University (1974).
[8] Rummelhart et al. "Learning Internal Representations by Error Propagation". MIT Press (1986).

# References

[9]https://towardsdatascience.com/effect-of-gradient-descent-optimizers-on-neural-net-training-d44678d27060
[10] https://arxiv.org/abs/1711.05101
[11] https://bbabenko.github.io/weight-decay/
[12] https://towardsdatascience.com/weight-decay-l2-regularization-90a9e17713cd
[12a] Santurkar S, Tsipras D, Ilyas A, Madry A. How does batch normalization help optimization? arXiv preprint arXiv:1805.11604. 2018 May 29.
[13] Veit, Andreas, Michael J. Wilber, and Serge Belongie. "Residual networks behave like ensembles of relatively shallow networks." Advances in neural information processing systems. 2016.
[14] https://thegradient.pub/semantic-segmentation/
[15] https://arxiv.org/pdf/1603.08511
[16] https://pdfs.semanticscholar.org/5c6a/0a8d993edf86846ac7c6be335fba244a59f8.pdf

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

# References


[17] https://arxiv.org/pdf/1606.00915.pdf
[18] https://arxiv.org/pdf/1610.02357.pdf
[19] https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf
[20] https://arxiv.org/abs/1603.08155
[21] https://arxiv.org/abs/1603.03417
[22] https://ai.googleblog.com/2016/10/supercharging-style-transfer.html
[23] https://arxiv.org/pdf/1903.07291.pdf
[24] http://nvidia-research-mingyuliu.com/gaugan
[25] Geirhos, Robert, et al. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness." arXiv preprint arXiv:1811.12231 (2018).

# References

[26] Beery, Sara, Grant Van Horn, and Pietro Perona. "Recognition in terra incognita." Proceedings of the European Conference on Computer Vision (ECCV). 2018.
[27] https://distill.pub/2017/feature-visualization/
[28] https://distill.pub/2018/building-blocks/
[29] Montavon, Grégoire, et al. "Layer-wise relevance propagation: an overview." Explainable AI: interpreting, explaining and visualizing deep learning. Springer, Cham, 2019. 193-209.
[30] https://medium.com/machine-intelligence-report/how-do-neural-networks-work-57d1ab5337ce
[31] Hebb, D.O. (1949), The organization of behavior, New York: Wiley

# References

[32] Dauphin, Yann N., et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." Advances in neural information processing systems. 2014.
[32a] http://www.mit.edu/~mitter/publications/121_Testing_Manifold.pdf
[32b] http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
[33] Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747 (2016).
[34] Viazovetskyi, Yuri, Vladimir Ivashkin, and Evgeny Kashin. 'StyleGAN2 Distillation for Feed-Forward Image Manipulation'. 7 March 2020. http://arxiv.org/abs/2003.03581.
[35] https://medium.com/@jonathan_hui/gan-stylegan-stylegan2-479bdf256299
[36] https://www.justinpinkney.com/making-toonify/
[37] http://chengao.vision/FGVC/files/FGVC.pdf

# References

[37b]
https://towardsdatascience.com/weight-initialization-in-deep-neural-networks-268a306540c0
[37c]
https://towardsdatascience.com/if-rectified-linear-units-are-linear-how-do-they-add-nonlinearity-40247d3e4792

**Dario Garcia-Gasulla** (BSC)

***dario.garcia@bsc.es***

HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE