Deep Learning - MAI

About

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

MAI - DL

- More width than depth
  - CNNs
  - RNNs
  - Transformers
  - Transfer Learning
  - Foundation models
  - HPC
- Some stuff somewhere else
  - Unsupervised (AEs, GANs, Diffusion)
  - RL stuff

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

MAI - DL

- Theory
  - In width
  - Presentation

- Labs (same room)
  - CNN (interview)
  - TL (interview)
  - HPC (ask Marc)

- Grade = 75% Lab + 25% Theory

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

MAI - DL

- For more details
  - https://upc-mai-dl.github.io/
  - dario.garcia@bsc.es
  - marc.casas@bsc.es (HPC)

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Deep Learning - MAI

Basic Deep Learning concepts & methods

THEORY

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

What is ~~DL~~ aka RL

François Chollet 
@fchollet

The biggest difficulty people (even fairly senior folks) seem to have in grasping that most deep learning models perform interpolation is that they think "interpolation" means "input-space L2 interpolation", i.e. linear regression 

1:42 AM · Aug 28, 2022 · Twitter Web App

"Interpolation" in the context of DL is of course *interpolation in feature space*. The name of the game is to transform your input space into one that organizes the data as a manifold where samples can be interpolated.

This is achieved, by necessity, via nonlinear transforms.

The misconception has gotten so bad that even when you point to models that are explicitly *built to learn interpolative embedding spaces*, like Transformers, some folks are like "oh no that's not interpolation, the model contains nonlinear transforms!"

Of course it does. That's how you learn the interpolative feature space. Your encoding space doesn't start out linear. So you have to learn to linearize it.

All problems you can solve with DL are interpolative problems by nature.

But there is no a priori reason to assume that such problems would be interpolative *for the Euclidean distance, in their original encoding space*.

That's because your choice of encoding space (e.g. RGB pixel space) is completely arbitrary. How you encode your data is a choice you make, not some kind of law of nature! Defaulting to the Euclidean distance is also a choice! Nothing about it is intrinsic to your problem.

The purpose of your model is to *learn* the encoding space that's intrinsic to your problem, or at least something close enough (there may be many suitable such spaces).

If such a space cannot be learned, then you have on your hands a problem for which the manifold hypothesis does not apply, and DL is simply *not* a good fit for such problems.

HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

A Bit of History

The not so distant origin story

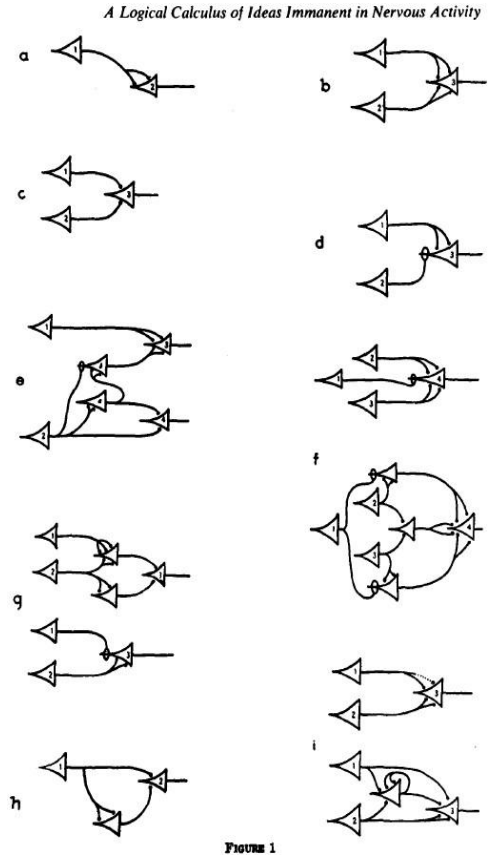
Connectionism & Hebbian Learning

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*



HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

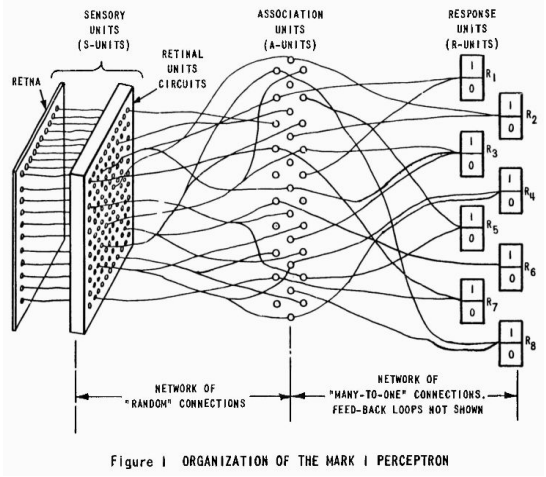
Ref: [1,31]

Rosenblatt's Perceptron

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

Mark I Perceptron

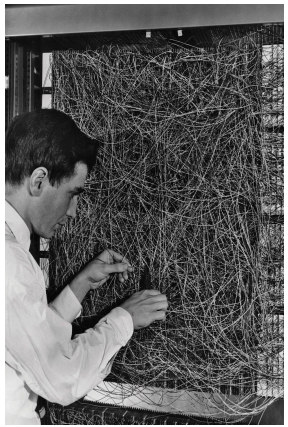
- 400 photosensitive receptors (sensory units)
- 512 stepping motors (association units, trainable)
- 8 output neurons (response units)



Threshold function

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

w real-valued weights
$\cdot$ dot product
b real scalar constant



[2,3,4,5]

Minsky & Papert: The XOR affair

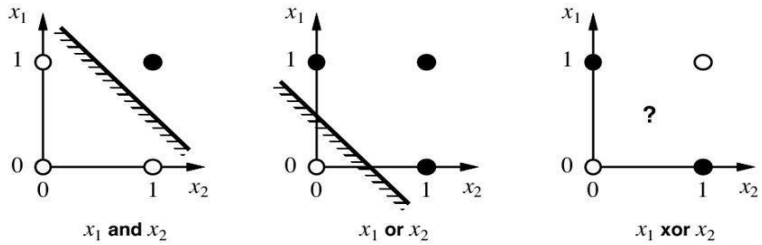
The perceptron capabilities were limited (Rosenblatt)

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 mid 80s: ***1st* AI WINTER**

Shift from connectionism to symbolism

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

[6]

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]

HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

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)

Gradient Descent

- **Backward pass**
  - Find gradients minimizing error layer by layer (1)
  - Apply gradients (2)

(1) Given a function (*loss)*, we can compute its slope (*gradient*) as a first-order derivative at the current point

(2) Move in the opposite direction of the slope, to minimize error

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!*

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**)

- For large instance size, **lower** batch sizes
- For more computational efficiency, **higher** batch sizes
- For more stochasticity, **lower** batch sizes
- Use batch sizes in the powers of 2

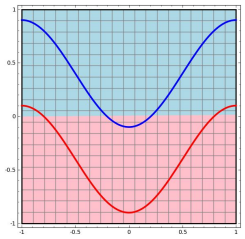
Factors for defining the number of epochs (**constantly**)

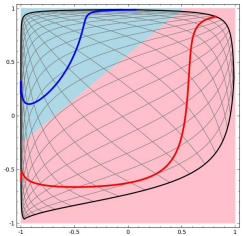
- For better convergence, **more** epochs
- For more reliability, **more** epochs
- For less footprint, **less** epochs

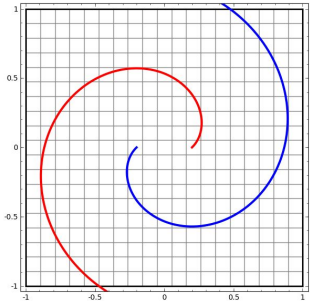
The Manifold Hypothesis

Deep Learning is defined by the high input dimensionality

- How can we find solutions in such a vast space?
- Manifold Hypothesis:
  - We can transform that into a smaller dimensionality “manifold”, in which the problem is simplified and interpolation possible.
- Each layer in a NN is a different manifold, transforming the space to facilitate the target task







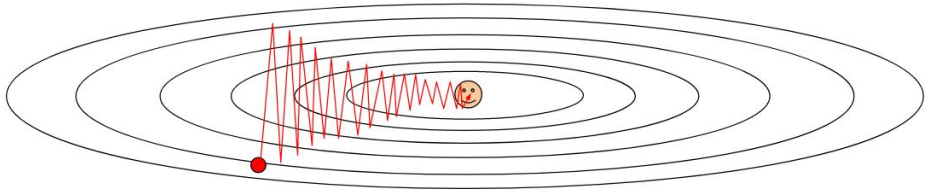
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

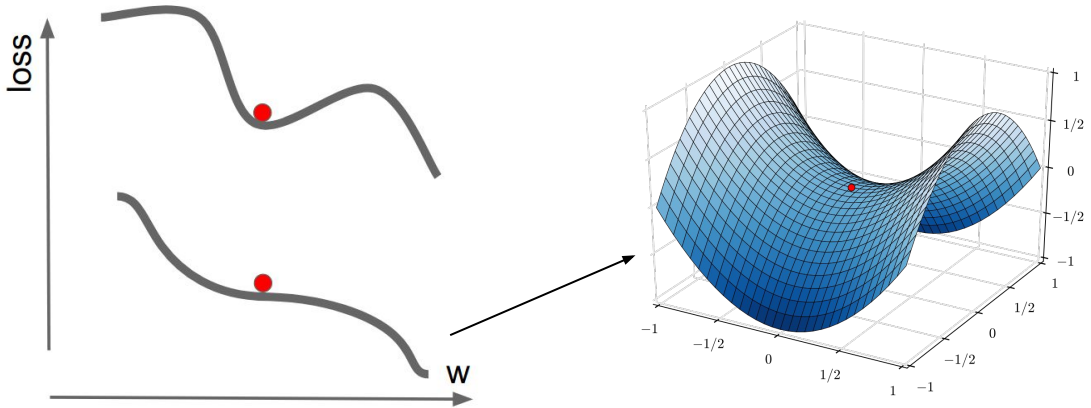
[32a,32b]

The Art of Descending

Hard to go down in a high dimensional space

- Different loss speed among dimensions (weird)
  - Slow and jitter
- Local minima & saddle points
  - Stuck gradient
- Mini-batches are noisy
  - Back & forth
  - Stochasticity & convergence





HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

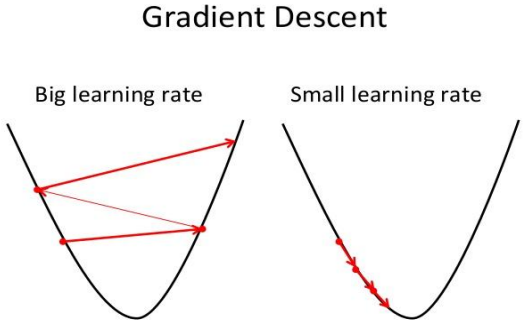
[32]

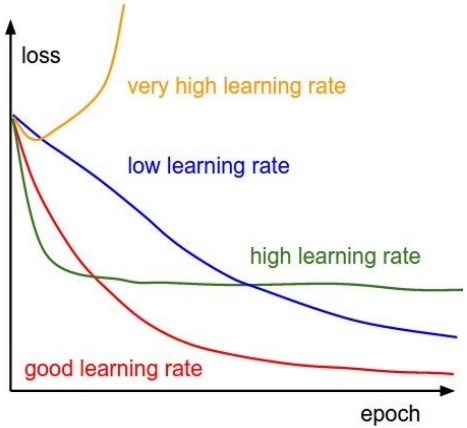
The Speed of Descending

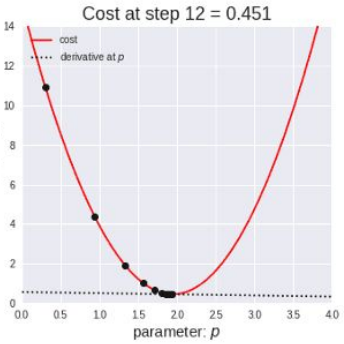
Learning rate: How much you move in the direction of the gradient

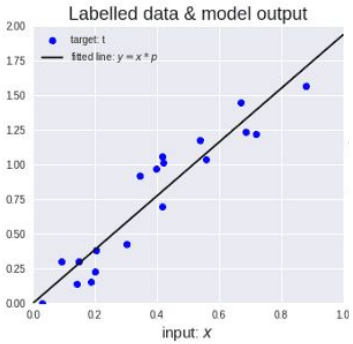
Direct effect on convergence and speed

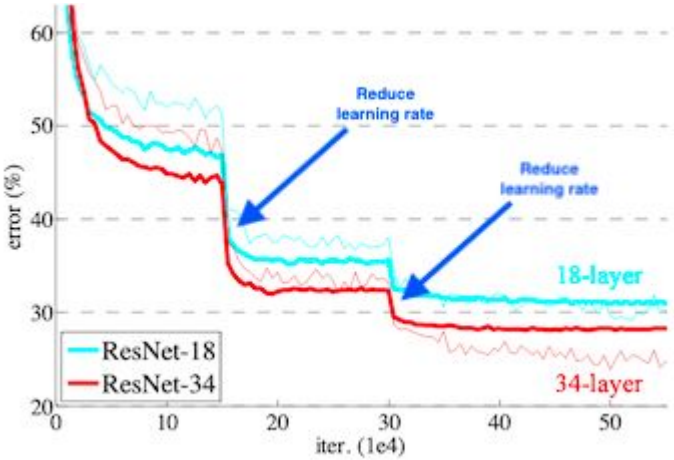
The same LR may not always be the right one











Practical Tips II

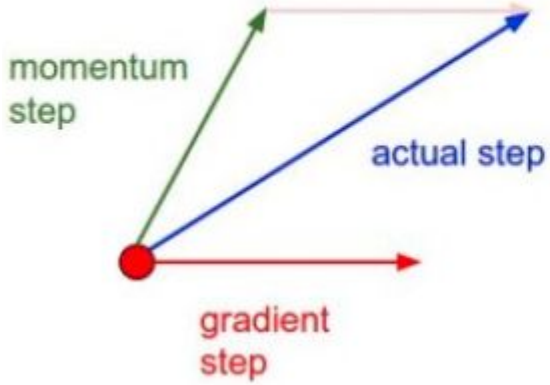
Tuning the Learning Rate

- 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

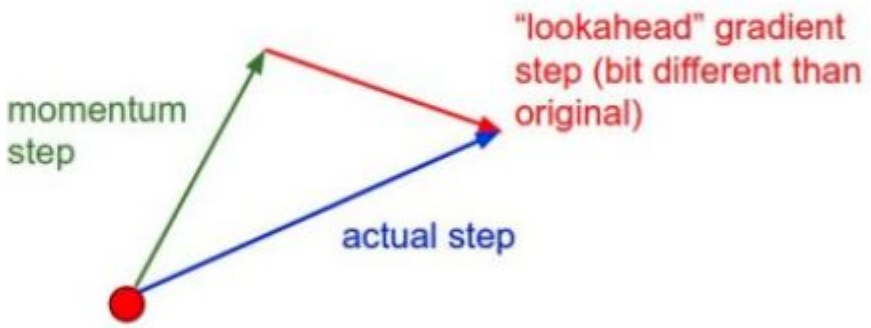
Momentum:

- Add fraction of previous gradient (*inertia*)
- Add decaying weight (*friction*)
- Faster, smoother convergence
-

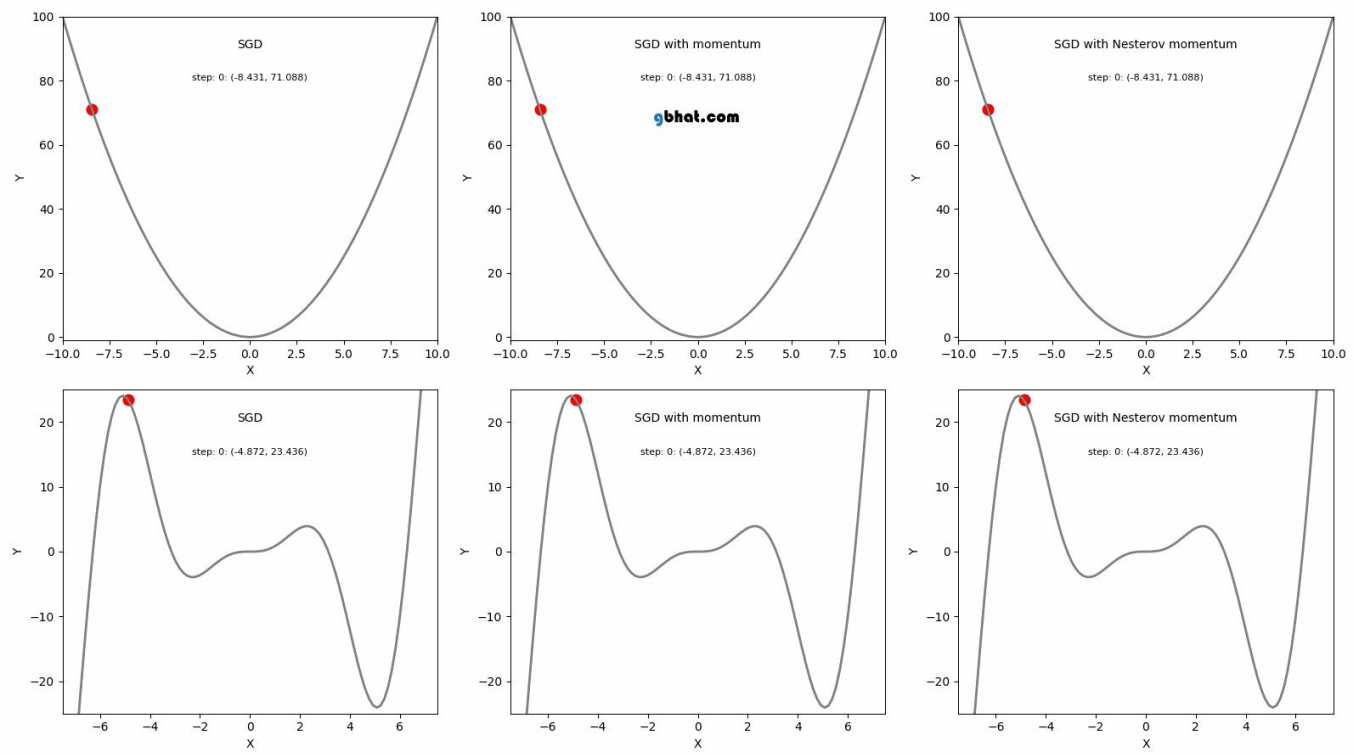


Nesterov:

- Gradient computed after inertia (see slope ahead)
- Faster convergence



Inertia in Optimizers



Adaptative LR Optimizers

Adagrad:

- Apply LR to parameter-wise gradients (**ada**ptative)
- High LR for infrequent ones. Low LR for frequent ones.
- Good for sparse data.
- Accumulates squared gradients. Scales LR by the *sqr*.

Issues:

- Requires initial global LR
- Vanishing LR. Stalls

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

[33, 48]

Adaptative LR Optimizers

Adadelta:

- Use effective LR (past param. update / current gradient)
- Max. window (decay avg.)
- Requires decay rate (0.9?)

Adam:

- Momentum (Decaying avg of past gradients, mean, beta1)
- Adadelta (Decaying avg of past *squared* gradients, variance, beta2)

Nadam:

- Nesterov + Adadelta

AMSGrad, AdaMax, AdamW, ...

[33,45,46,47,48]

Practical Tips III

Optimizers

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

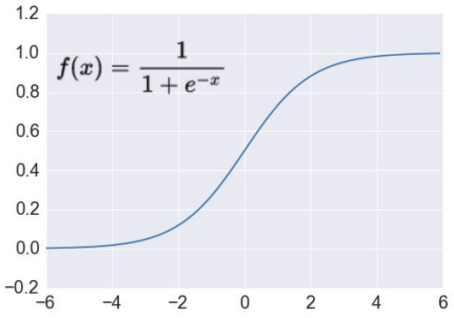
Hyperparameters incomplete list #1 (training)

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

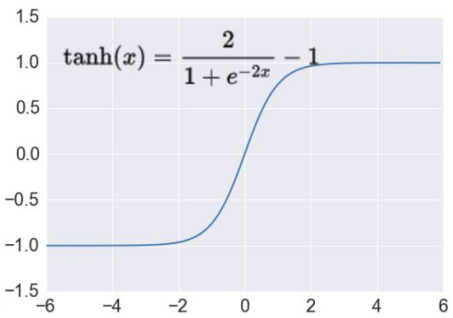
Activation Functions

Transform the output of a layer to a given range.

Sigmoid



TanH

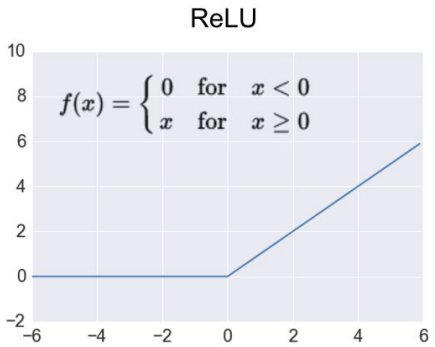


- Zero gradient most of f(x): Saturates!
- Gradient is 0.25 or 1 max. Vanishes!

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Activation Functions

Transform the output of a layer to a given range.



- Does not saturate
- Does not vanish
- Faster computation
- May die with high LR
  - No learning on negative
  - Weight init, BN, ...

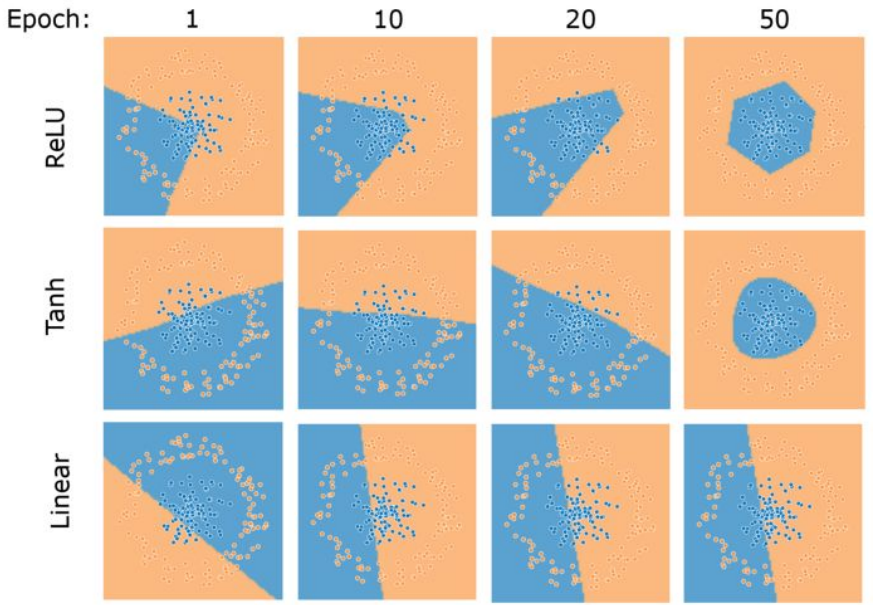
ReLU is a safe choice in most cases

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Why ReLU works

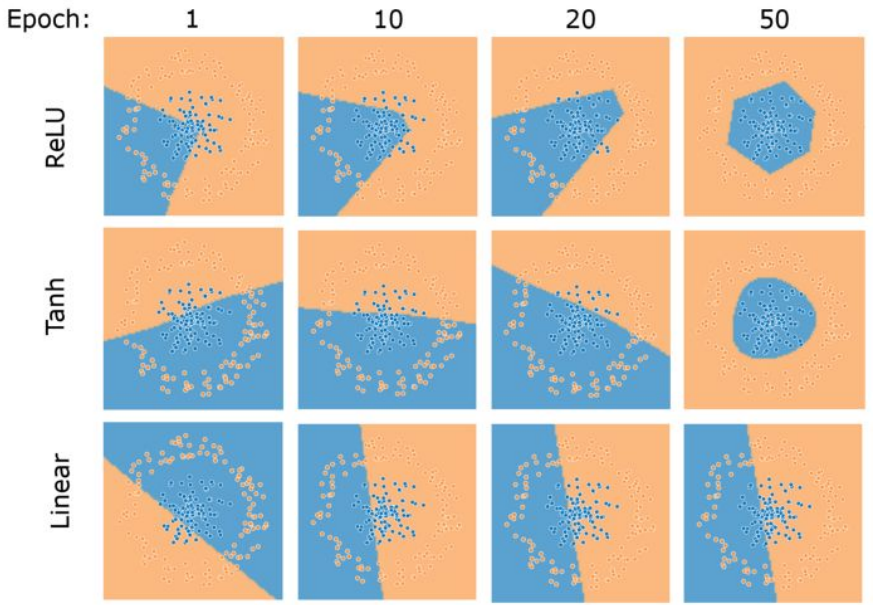
- But wait, Re**L**U is linear, and we need non-linearity!
- Not exactly. It's piecewise linear. Composed of two linear functions
- ReLU can bend linearity
  - On one point
  - With any angle
- Just need a bunch of ReLUs



[37c]

Why ReLU works

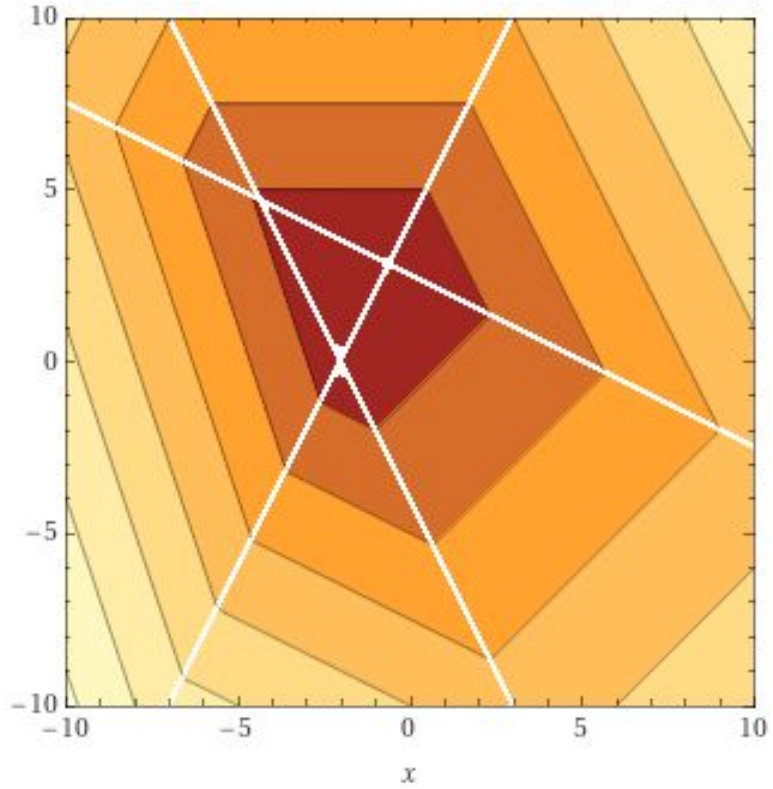
- But wait, Re**L**U is linear, and we need non-linearity!
- Not exactly. It's piecewise linear. Can compose both linear and non-linear
- ReLU can bend linearity
  - On one point
  - With any angle
- Just need a bunch of ReLUs

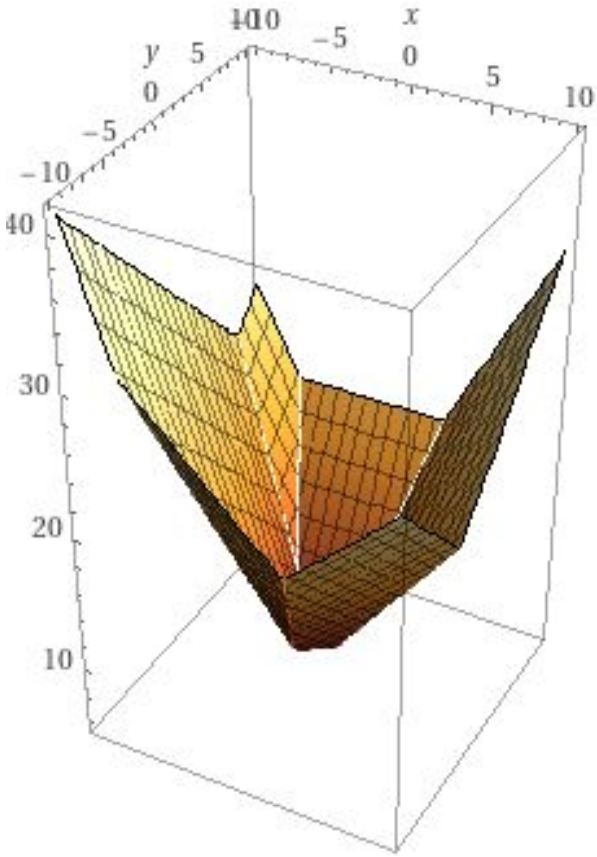


[37c]

ReLU: Composing non-linearity

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



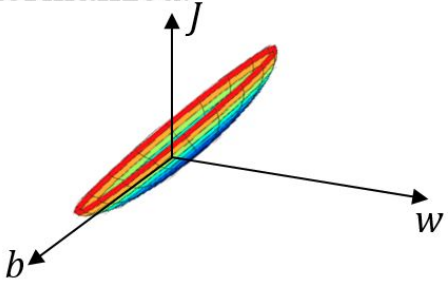


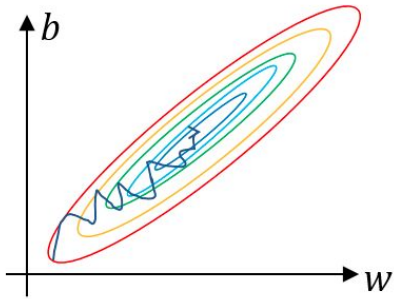
[30]

Input Pre-processing

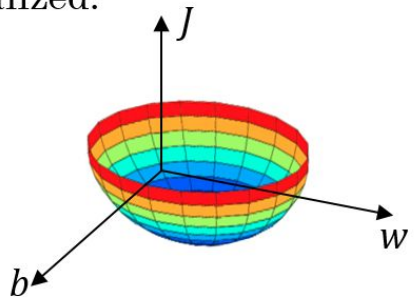
Make your model's life easier. Run flat.

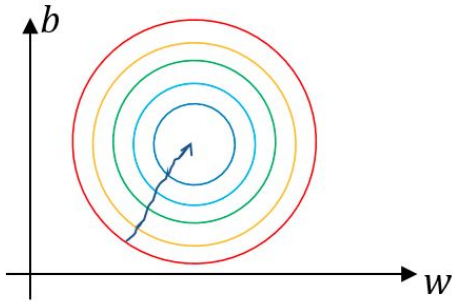
Unnormalized:





Normalized:





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 value~~: No symmetry breaking.
- ~~Zeros~~: No gradient flow
- Gaussian distribution sample: Ok for shallow, but deviation grows with size
- Glorot/Xavier: Gaussian with 0 mean. Normalize variance by number of inputs + outputs
- He/Kaming/MSRA: Gauss, 0 mean, var. wrt inputs only. Specific for ReLUs (smaller var.).

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

[37b]

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

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

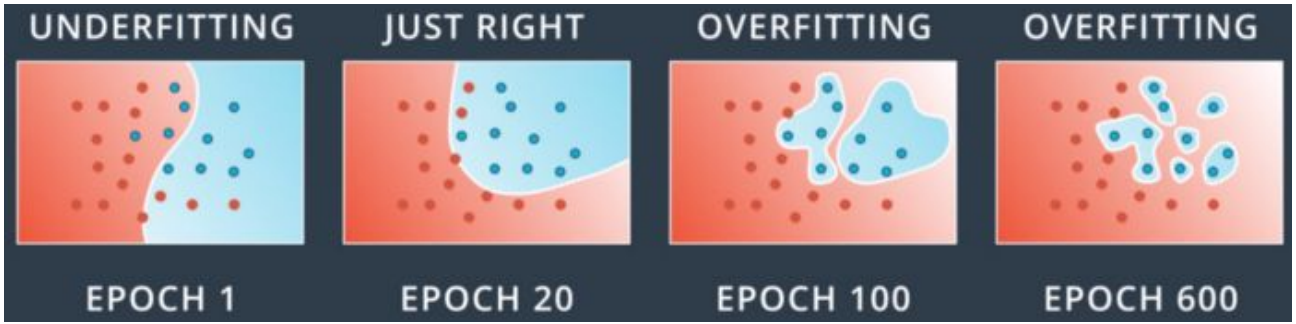
From underfit to overfit

Underfit: Insufficient learning of training data patterns

Overfit: “Excessive” learning of training data patterns →

Key players:

- Model capacity
- Data input
- Regularizers



HPAI
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Overfit in DL

- A necessary evil
- Not everything makes sense
- Ensembling

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

Back to Regularization

Takes us from overfit to fit

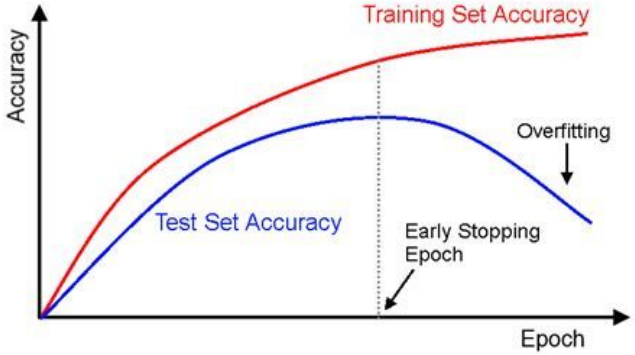
The must do ones:

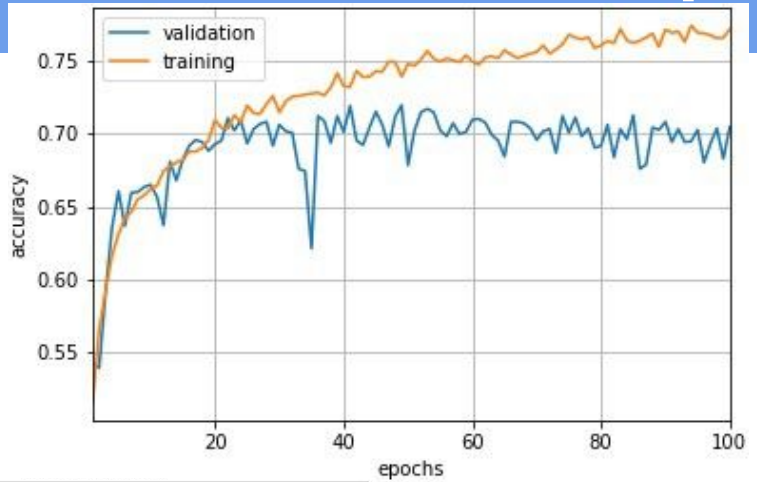
- **Early Stopping**
  - Overfitting is the end of the road
  - The guide: Validation loss/accuracy
  - Enough to understand the model (mind the footprint!)
- **Data Augmentation**
  - More data for free
  - Huge impact
  - As any data preprocessing, think thoroughly!

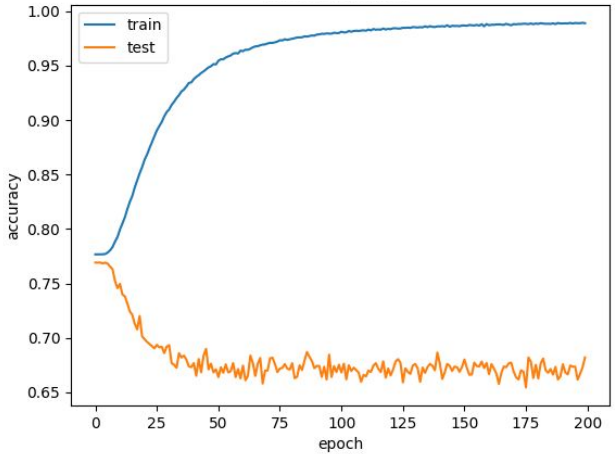
Practical Tips VII

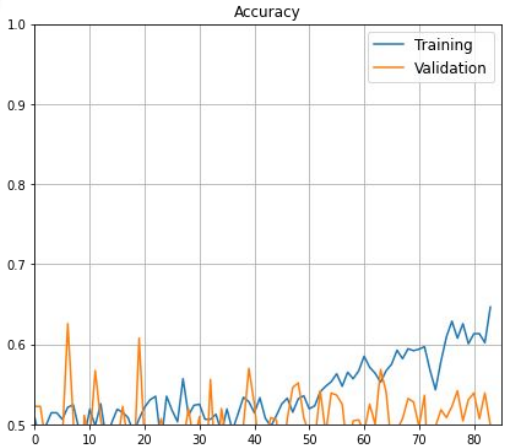
Diagnosing the curves (loss & acc.*)

- Random performance?
- General trend?









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.

Weird curves are the worst!

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Parameter Norm Penalty methods

L2/L1 norm penalize large weights by factor (alpha) added to loss

- Keeps weights small
- Changes gradients
- Alpha too large, underfit. Alpha too small, still overfit.
- Conflict with adaptive learning rates (e.g., Adam)

Weight decay add scaled weight in update step

- Independent of gradient & learning rate, decreases update
- Analogous to L2-norm for SGD, not for adaptative optimizers
- Safe for all (if implemented). AdamW (Adam with weight decay) to be sure.

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

[10,11,12,44]

Max-Norm

Limits the magnitude of the weights vector

- Constant c (another hyperparam!)
- Typically around 3 or 4
- Goes well with dropout
- May be redundant with L2-norm/weight decay

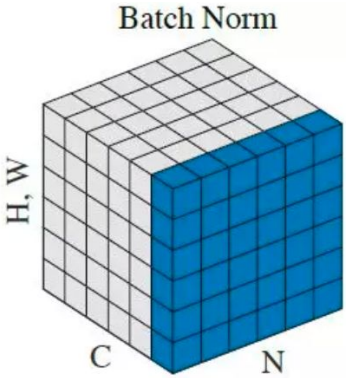
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

[10,11,12]

Batch Normalization

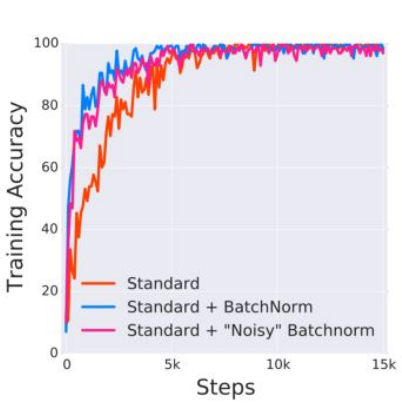
Force *activations* (samples) within a normal distribution

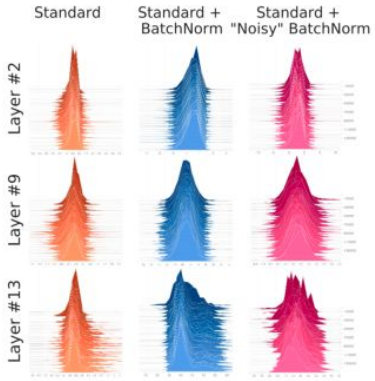
- Shift (add) & scale (mul.) for mean 1 and std dev 0
- Applied between neurons and non-linearity activation function
- Statistics computed per mini-batch (practical reasons)
- On inference, use population of mini-batch statistics
- Helps with *initialization* and regularization
- Requires minimum & fixed batch size (does it?)

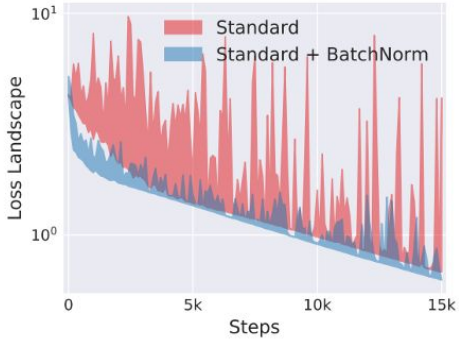


WIKI: Batch Normalization

- Reduces ~~"internal covariate shift"~~
- Smoother loss landscape
- Easier hyperparameter setting
  - Allows higher LR
- Faster convergence
- Learnable version (affine transformation)
  - Scale (gamma) and shift (beta)



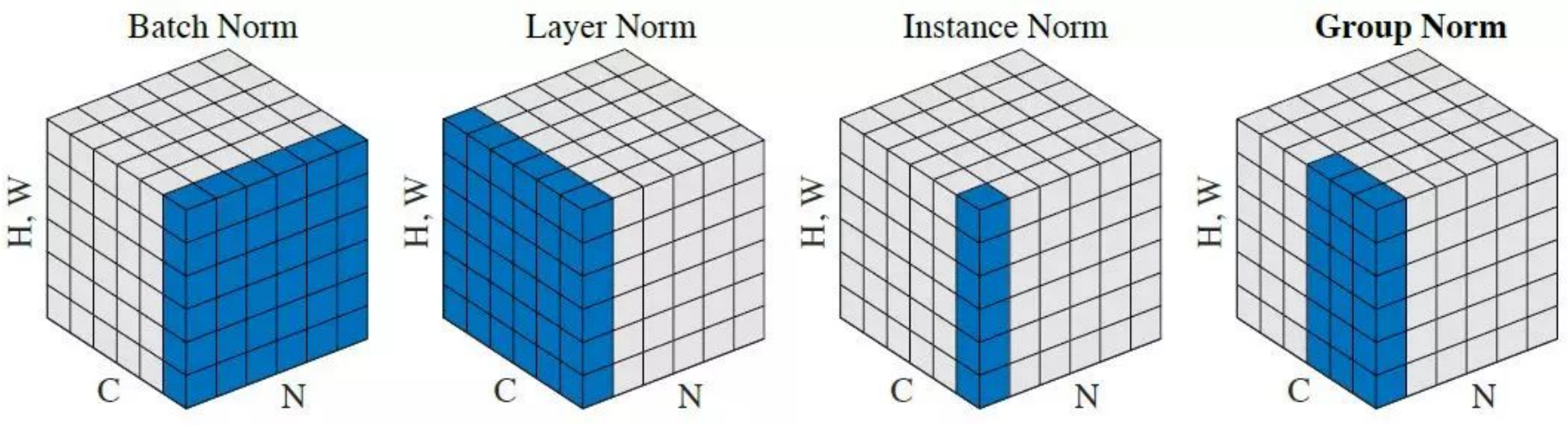




(a) loss landscape

[12a,38,39]

Norms, Norms, Norms



Batch Norm	Layer Norm	Instance Norm	Group Norm
All instances	One instance	One instance	One instance
One channel	All channels	One channel	N channels

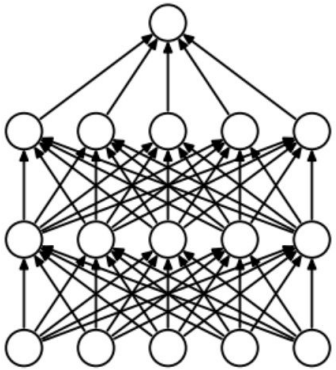
And more!!

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

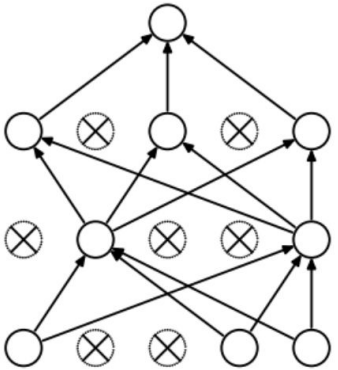
Dropout

Cut-off neuron inputs with a given probability

- Rate (typically [0.2,0.5]) on every step
- In practice trains an ensemble of nets
- Inference: Use all inputs, scaled by rate
- Reduces co-adaptation
- Slows down training (a lot)
- Affects many other hyperparameters (e.g., before batch norm or maxpool)
- Good on FC layers, generally towards the task-specific part of the net



(a) Standard Neural Net



(b) After applying dropout.

Dropout variants

- Dropconnect: Delete edges instead of neurons
- Standout: Rate based on weight (high weight -> high prob.)
- Gaussian Dropout: Faster convergence as no neuron is ever fully disconnected (Gaussian vs Bernoulli)
- Different architectures, different dropouts!

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

[41,42,43]

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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)
- Segmentation (Dice, IoU), contrastive loss (pos/neg distance)
- 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

HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

Next class:
Convolutional Neural Networks

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).

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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
[38] https://e2eml.school/batch_normalization.html
[39] https://proceedings.neurips.cc/paper/2019/file/cb3ce9b06932da6faaa7fc70d5b5d2f4-Paper.pdf
[40] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

References

[41] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, Regularization of neural networks using dropconnect

[42] L. J. Ba and B. Frey, Adaptive dropout for training deep neural networks

[43]S. Wang and C. Manning, Fast dropout training

[44] https://benihime91.github.io/blog/machinelearning/deeplearning/python3.x/tensorflow2.x/2020/10/08/adamW.html

[45] https://visharma1.medium.com/adagrad-and-adadelta-optimizer-in-depth-explanation-6d0ad2fdf22

[46] https://www.quora.com/What-is-an-intuitive-explanation-of-the-AdaDelta-Deep-Learning-optimizer

[47] https://www.youtube.com/watch?v=5KyxlN3M7HQ

[48] https://www.ruder.io/optimizing-gradient-descent/

References

[49] https://gbhat.com/machine_learning/gradient_descent_nesterov.html
[50] https://golden.com/wiki/Nesterov_momentum-YX9WPE5

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Dario Garcia-Gasulla (BSC)

dario.garcia@bsc.es

HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE