Introduction to Deep Learning

Razvan Pascanu
Research Scientist @ Google DeepMind
razp@google.com
r.pascanu@gmail.com

https://xkcd.com/1838/

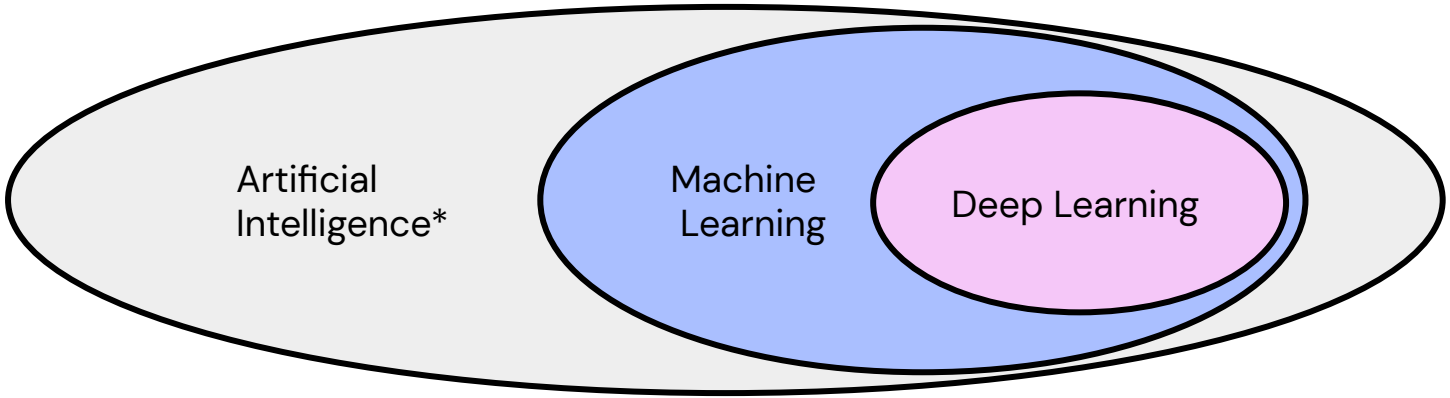
Lecture Overview (managing expectations)

- Intuitive level (a.k.a. It will be *vague* :)), so
  - ask questions on sli.do
  - ping me on slack for further details!
  - chat in person (I'm here for the entire week)
- It will be condensed
- This will introduce my current way of thinking about Deep Learning (so bias selection of topics and results)

What is Deep Learning about ?

Intuitively, machine learning (and by extension deep learning) studies how to teach machines (agents) to *solve tasks* with little to no human intervention by learning from interaction with the particular task (i.e. from data). The solution is not prescribed by the practitioner, just the learning strategy (and encoding of the solution).

Deep Learning focuses on a particular family of *parametric models* (neural networks) and (typically) a particular *family of learning algorithms* (typically gradient-based ones).



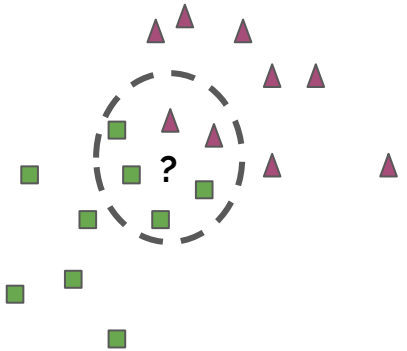
A starting point: Supervised Learning & Parametric methods

Given some data $D \subset \mathcal{X} \times \mathcal{Y}$

Find a *"good predictor"* $h : \mathcal{X} \to \mathcal{Y}$

Non-parametric
e.g.
k nearest neighbour classifier

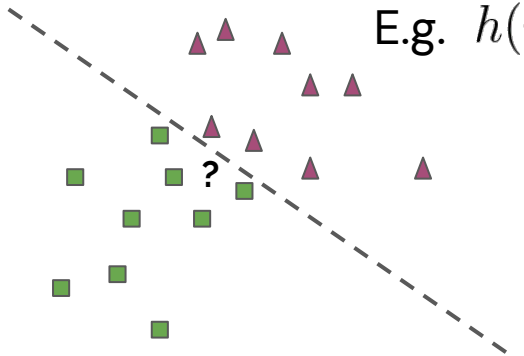
$$h : \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \times \mathcal{X} \to \mathcal{Y}$$



Parametric
e.g.
Linear models

$$h : \theta \times \mathcal{X} \to \mathcal{Y}$$

E.g. $h(w, x) = wx$



Part I : Basics

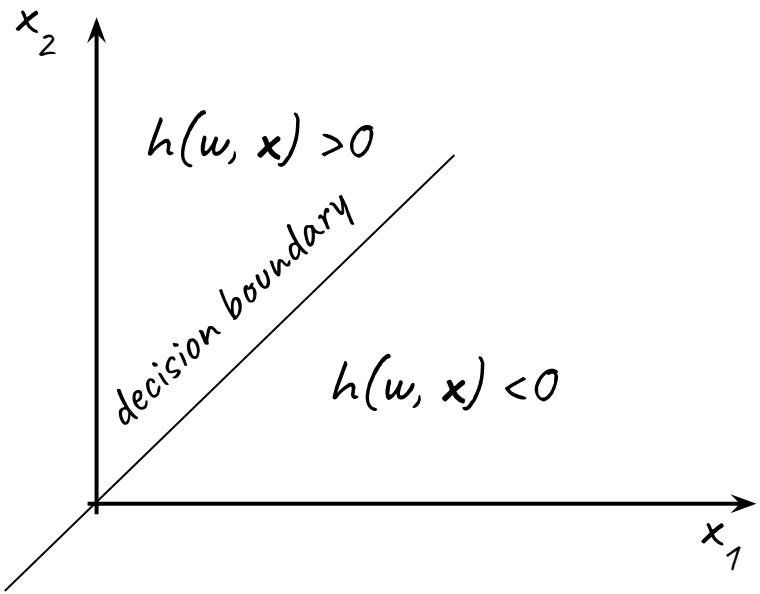
We will focus on **two** questions:

You are here → **What is a good parametrization ?**

How do we find these optimal parameters ?

What is a good parametrization ?

The linear model $h(w, x) = wx$

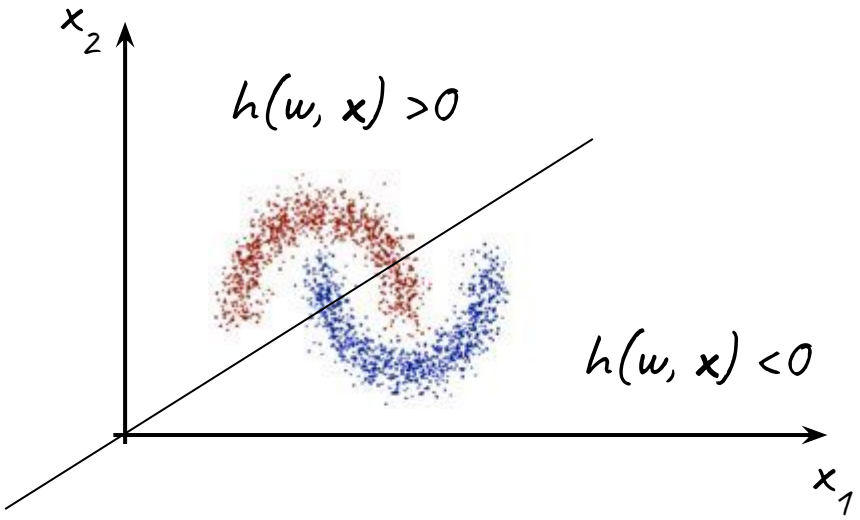


- Can straightforwardly be used for classification (and regression)
- Defines linear decision boundaries (and represents linear functions)

Intuitive, tractable to analyze, but it is not enough

What is a good parametrization ?

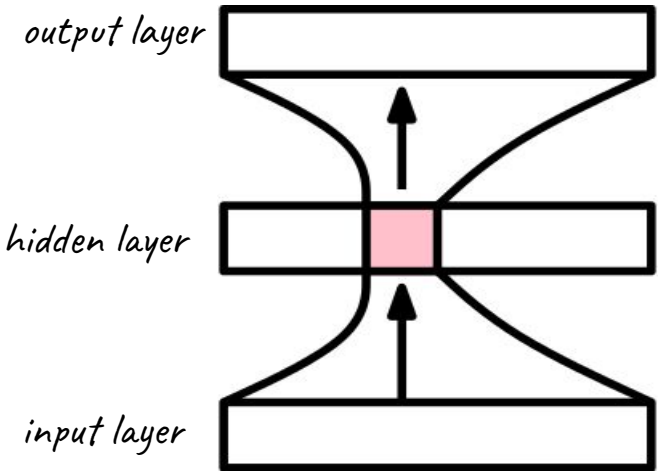
Most problems are not linearly separable.



Neural networks to the rescue! Basics:

$$h(\mathbf{x}, \theta) = \mathbf{W}_{out} ReLU(\mathbf{W}_{in}\mathbf{x} + \mathbf{b}_{in}) + \mathbf{b}_{out}$$

$$ReLU(x) = \begin{cases} x & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

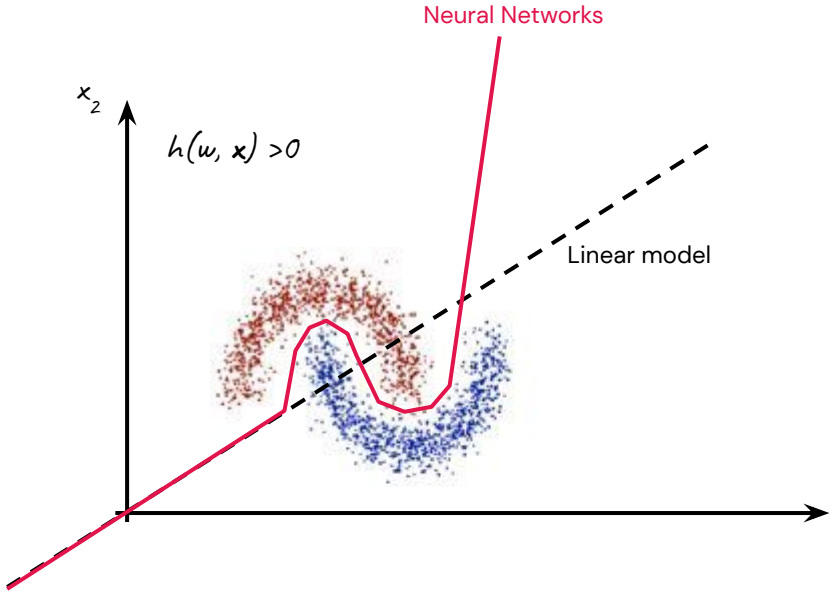


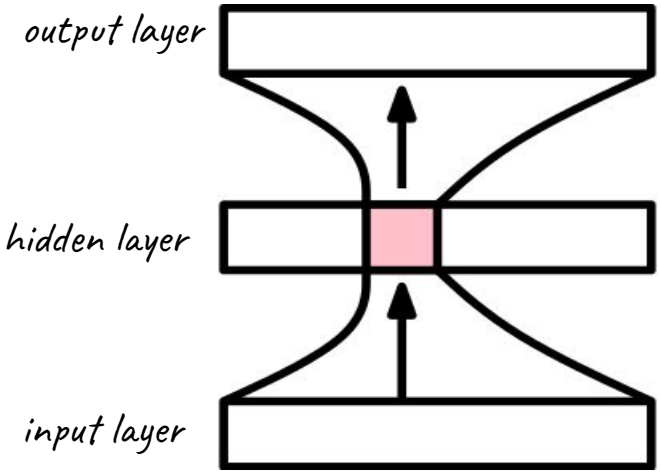
- Many non-linearities are possible

Neural networks to the rescue!

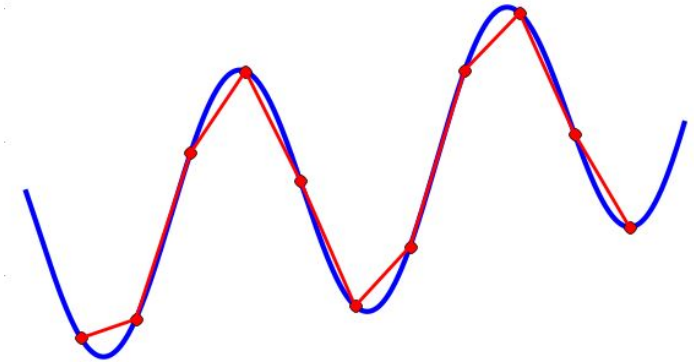
$$h(\mathbf{x}, \theta) = \mathbf{W}_{out} ReLU(\mathbf{W}_{in}\mathbf{x} + \mathbf{b}_{in}) + \mathbf{b}_{out}$$

- Other non-linearities are possible





$$h(\mathbf{x}, \theta) = \mathbf{W}_{out} ReLU(\mathbf{W}_{in}\mathbf{x} + \mathbf{b}_{in}) + \mathbf{b}_{out}$$



Universal approximation theorem
Cybenko ‘89, Hornik, Stinchcombe & White ‘89, .. Pinkus ‘99, .. as recent as Cai 2023

tl;dr : In some limit, single hidden layer neural networks can approximate arbitrarily well any nonlinear function.

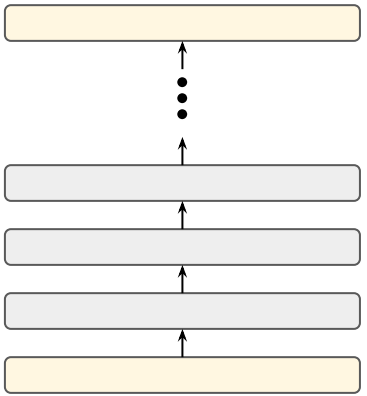
Purely an expressivity argument (i.e. there exists a neural network such that ..)

Deep in deep learning:

$$h(\mathbf{x}, \theta) = \mathbf{W}_k ReLU(\mathbf{W}_{k-1} ReLU(\mathbf{W}_{k-2} \ldots ReLU(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \ldots) + \mathbf{b}_{k-1}) + \mathbf{b}_k$$

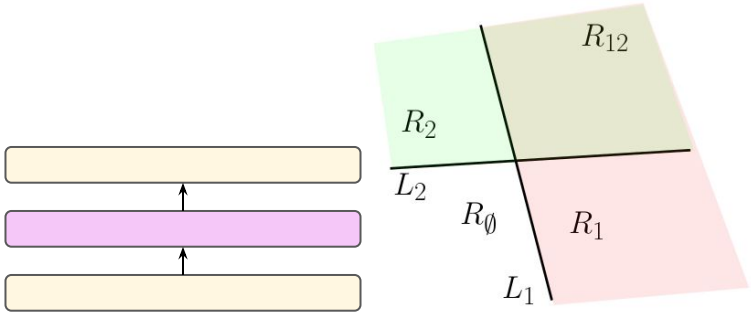
The term ***deep learning*** or ***deep neural networks*** was introduce to highlight the importance of ***depth***.

- What does depth provide? Why is this part of the secret sauce?



Single (hidden) Layer

$$\mathbf{W}_{1,[i:]}\mathbf{x} + \mathbf{b}_{1,[i]} = 0$$

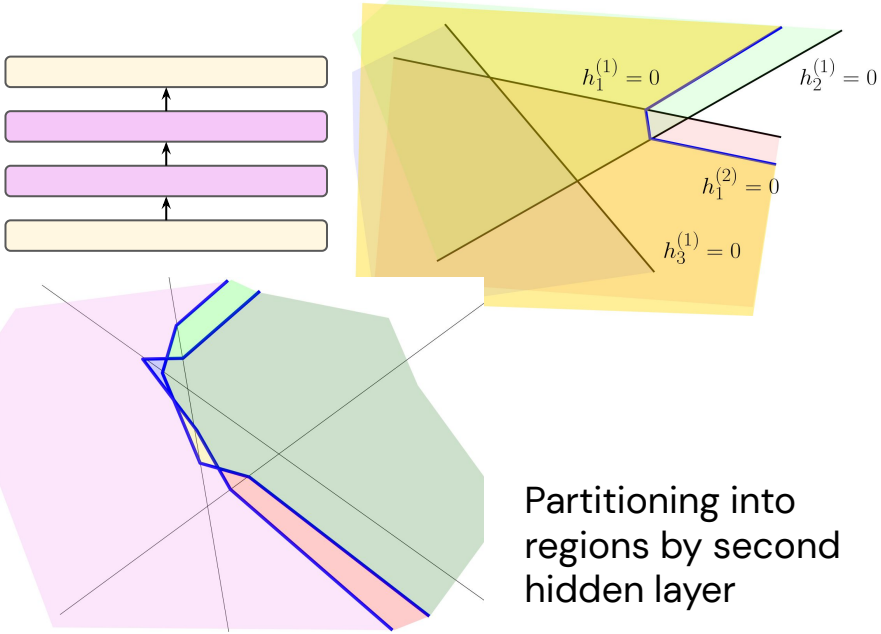


Zaslavsky theorem (1975):

$$r(\mathcal{A}_m) = \binom{m}{2} + m + 1.$$

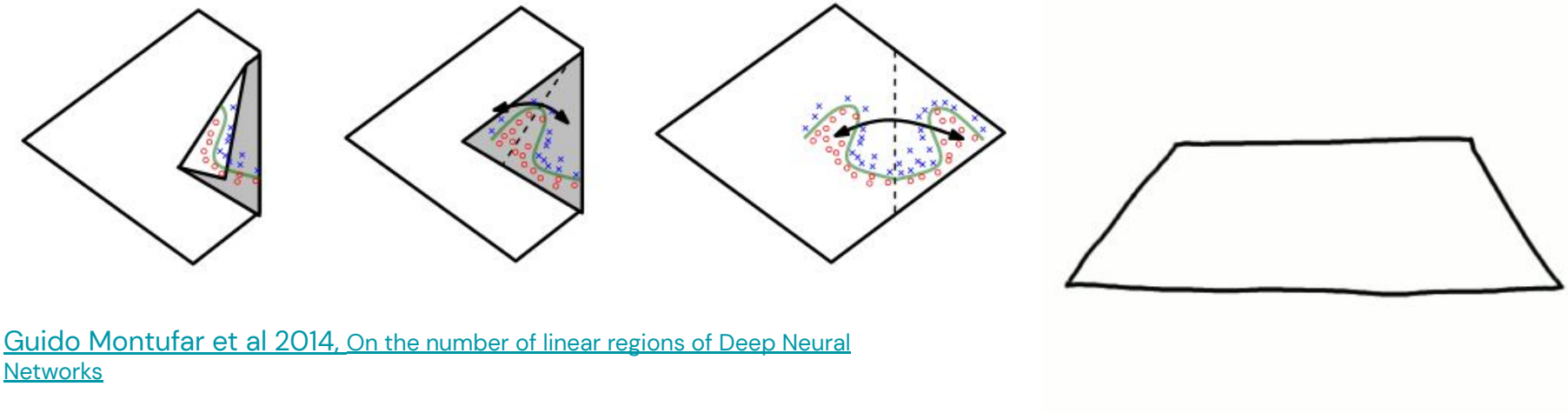
Multiple (hidden) Layers

$$\mathbf{W}_{2,[i:]}ReLU(\mathbf{W_1}\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_{2,[i]} = 0$$



Partitioning into regions by second hidden layer

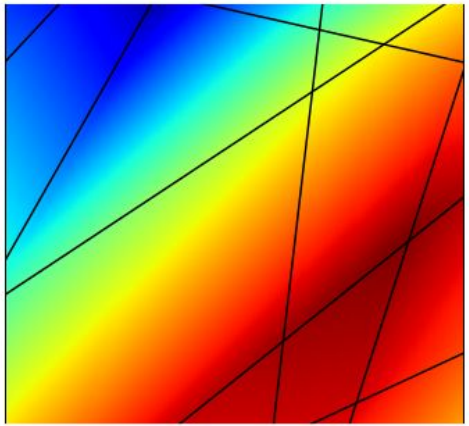
Deep Learning meets Origami

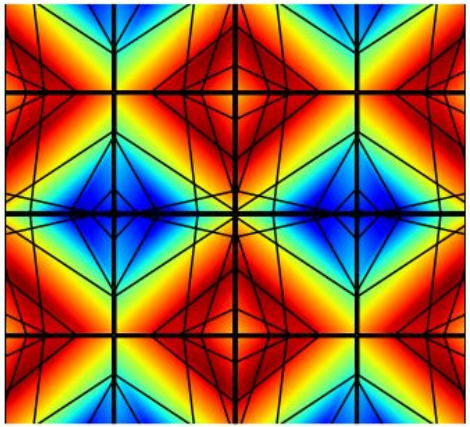


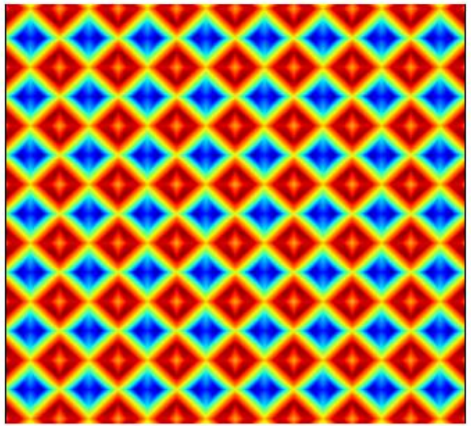
Guido Montufar et al 2014, On the number of linear regions of Deep Neural Networks

By folding the space, you gain expressivity!
(exponentially more linear regions)
without increasing the number of parameters!

- Is having exponentially more linear regions a good thing? Can this explain the success of DL?
- Is the limiting factor of previous methods expressivity?
- Do shallow model underperform because of lack of capacity?

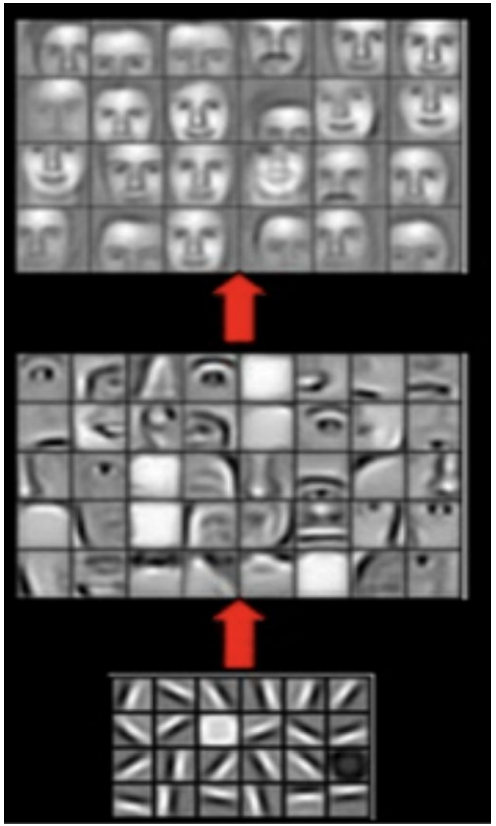






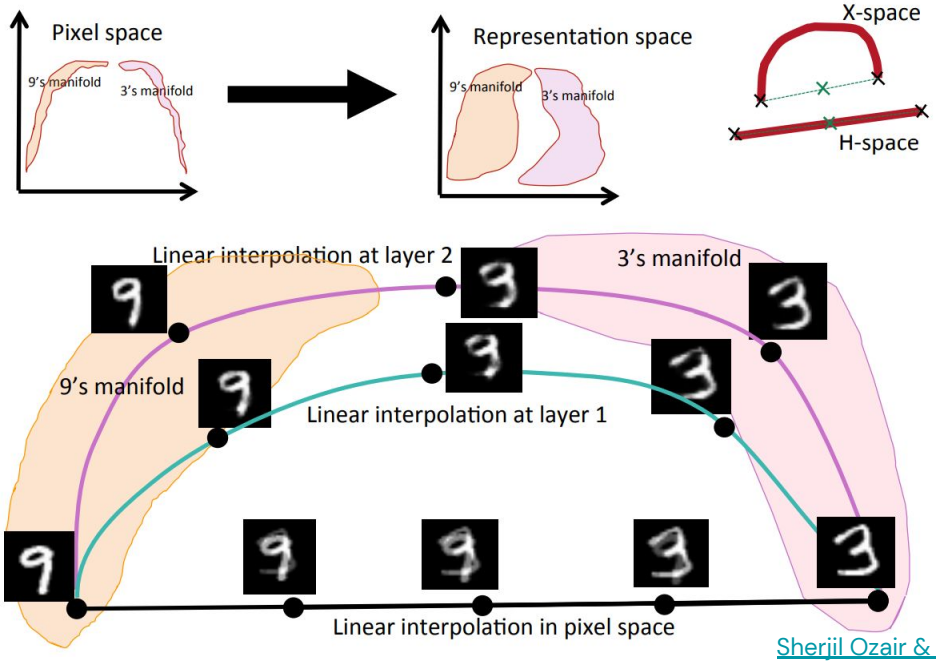
Depth acts as an inductive bias!

Do we understand what this inductive bias is?



Matt Zeiler & Rob Fergus 2013, Visualizing and Understanding Convolutional Networks

Honglak Lee et al. 2009, Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations



Sherjil Ozair & Yoshua Bengio 2014, Deep Directed Generative Autoencoders

Two arguments against the view:

- Adversarial examples show it is hard to assign semantics to units (and local changes can affect behaviour)

- Modern architectures rely heavily on skip paths that change typical view on depth (more in a few slides)

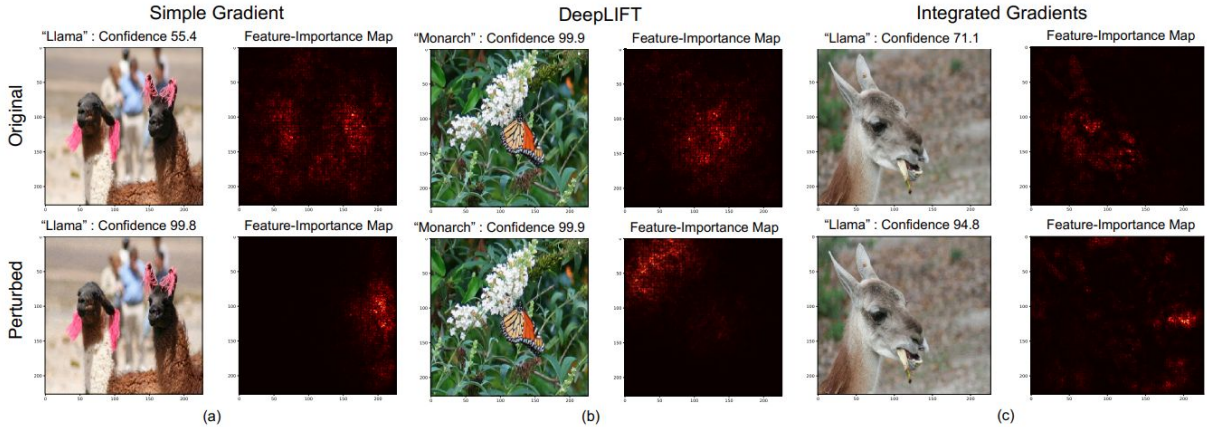
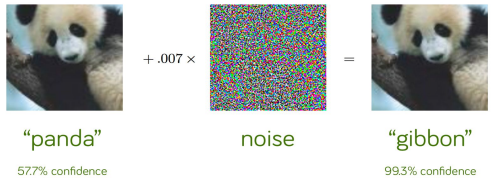


Figure 1: **Adversarial attack against feature-importance maps.** We generate feature-importance scores, also called saliency maps, using three popular interpretation methods: (a) simple gradients, (b) DeepLIFT, and (c) integrated gradients. The **top row** shows the the original images and their saliency maps and the **bottom row** shows the perturbed images (using the center attack with $\epsilon = 8$, as described in Section 2) and corresponding saliency maps. In all three images, the predicted label does not change from the perturbation; however, the saliency maps of the perturbed images shifts dramatically to features that would not be considered salient by human perception.

Ghorbani, Abid and Zou 2018 Interpretation of Neural Networks is fragile



Ian J. Goodfellow et al 2014 Explaining and Harnessing Adversarial Examples

Part I : Basics

Open-Questions: What inductive biases are we relying on?
How do we control / modify them?

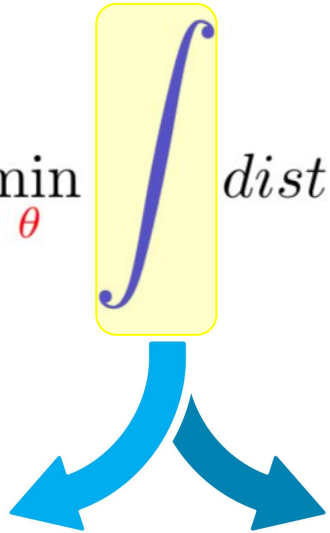
We will focus on **two** questions:

What is a good parametrization ?

You are here → **How do we find these optimal parameters ?**

How do we find the optimal parameters ?

$$\arg\min_{\theta} L(\theta) = \arg\min_{\theta} \int dist\left(h(\theta, x), \underbrace{y}_{\text{corresponding label for } x}\right) dx$$



Optimization

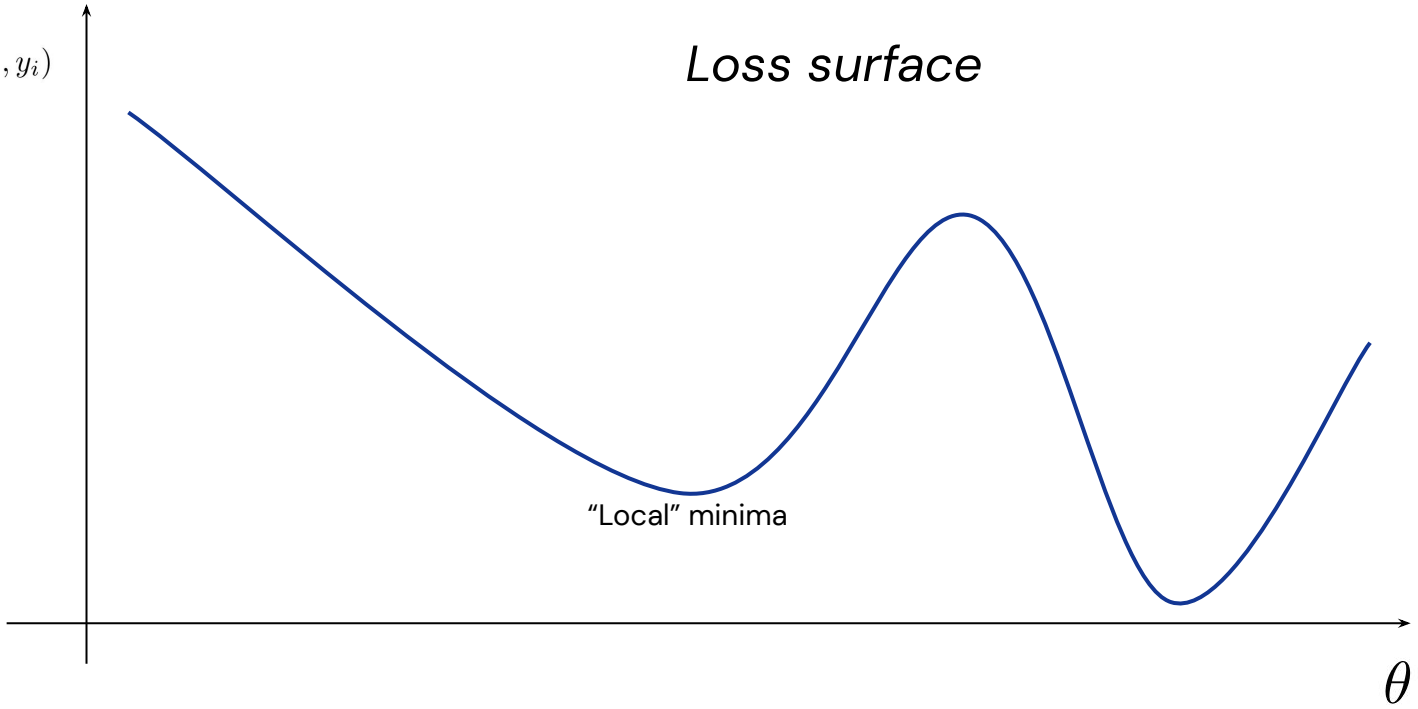
$$\int \approx \sum_{(x_i, y_i) \in \mathcal{D}}$$

Learning/Generalization

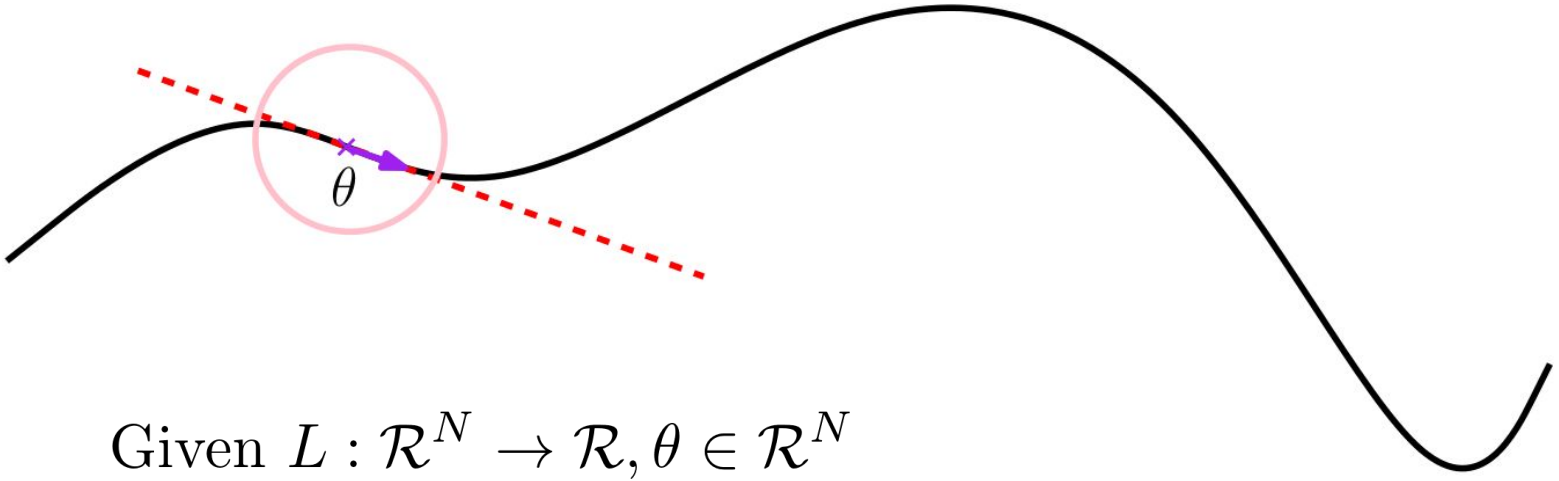
$$\int \propto \mathbb{E}_{(x_i, y_i) \sim \pi}$$

Optimization: Find the minimum of loss

$$\sum_{(x_i,y_i)\in\mathcal{D}} dist(h(\theta, x_i), y_i)$$



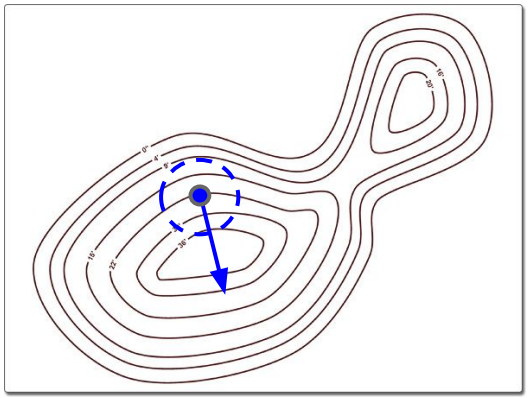
Optimization: Gradient descent



$$\text{Given } L : \mathcal{R}^N \rightarrow \mathcal{R}, \theta \in \mathcal{R}^N$$

$$\arg\min_{\Delta\theta} L(\theta + \Delta\theta) \approx \arg\min_{\Delta\theta} \left[L(\theta) + \Delta\theta \frac{\partial L}{\partial \theta} \right]$$

$$s.t. \|\Delta\theta\| \leq \epsilon$$



$$\arg\min_{\Delta\theta}\left[L(\theta)+\Delta\theta\frac{\partial L}{\partial\theta}\right]$$

$$s.t.\|\Delta\theta\|\leq\epsilon$$

$\Downarrow$ Lagrange multipliers

$$\arg\min_{\Delta\theta}\left[L(\theta)+\Delta\theta\frac{\partial L}{\partial\theta}+\eta\underbrace{\Delta\theta\Delta\theta^T}_{\approx\Delta\theta^2}\right]$$

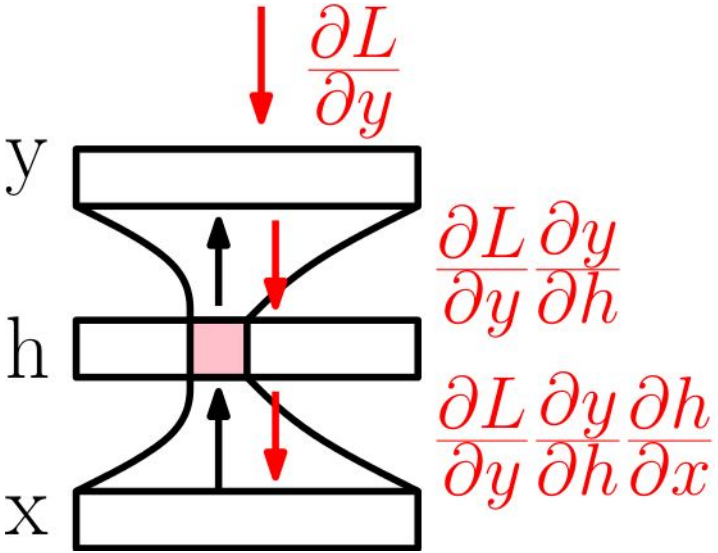
$\Downarrow$

$$\frac{\partial\left(L(\theta)+\Delta\theta\frac{\partial L}{\partial\theta}+\eta\Delta\theta\Delta\theta^T\right)}{\partial\Delta\theta}=0\Leftrightarrow\frac{\partial L}{\partial\theta}+2\eta\Delta\theta=0\Rightarrow$$

$$\Rightarrow\Delta\theta=-\underbrace{\frac{1}{2\eta}}_{\gamma}\frac{\partial L}{\partial\theta}\Rightarrow\theta_{t+1}=\theta_t-\gamma\frac{\partial L}{\partial\theta}$$

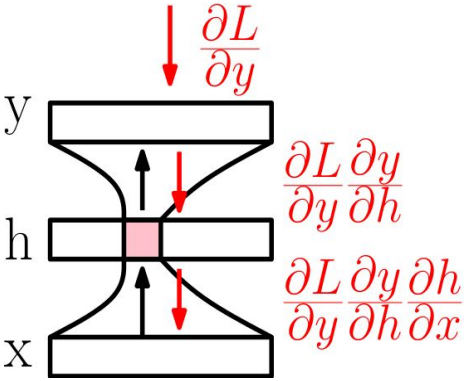
How do we compute $\frac{\partial L}{\partial \theta}$?

The celebrated ***Backpropagation Algorithm***



$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L}{\partial \theta}$$

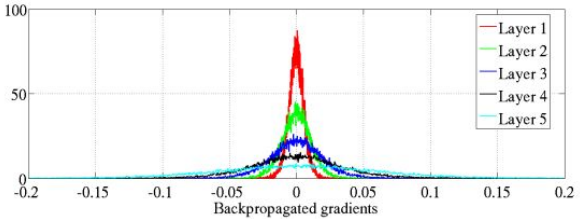
- Is this just the chain-rule?
- Why is it important to start from the output towards the input?



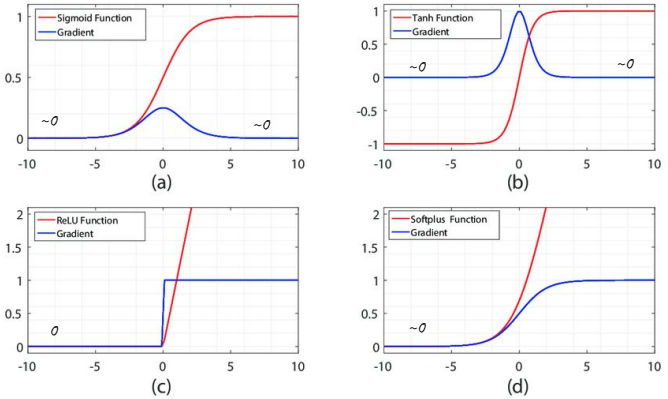
- It is always useful to **not ignore** the specific of the architecture you are playing with. **It is not just another function !**

- Still an active area of research (e.g. Gu et al. 2020; research on plasticity loss)

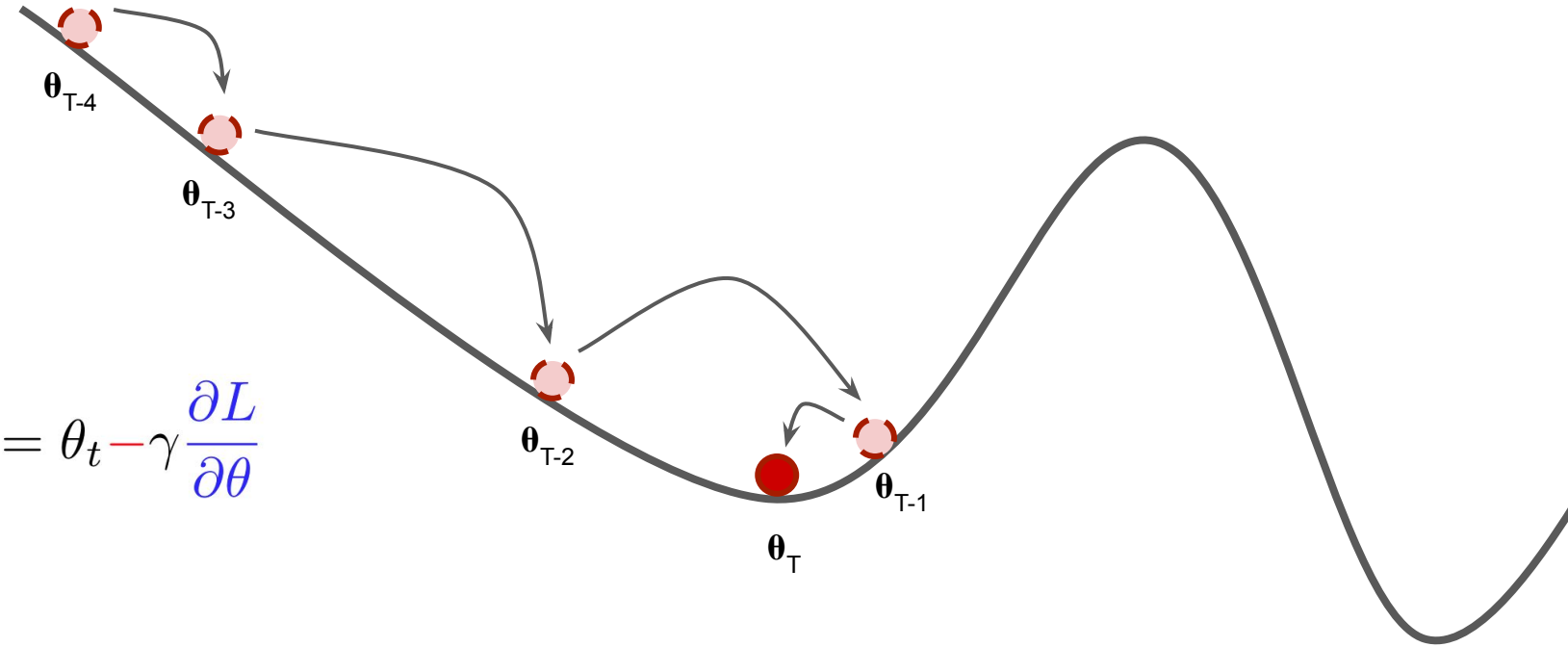
Initialization (and gradient propagation)



Glorot and Bengio AISTATS 2010

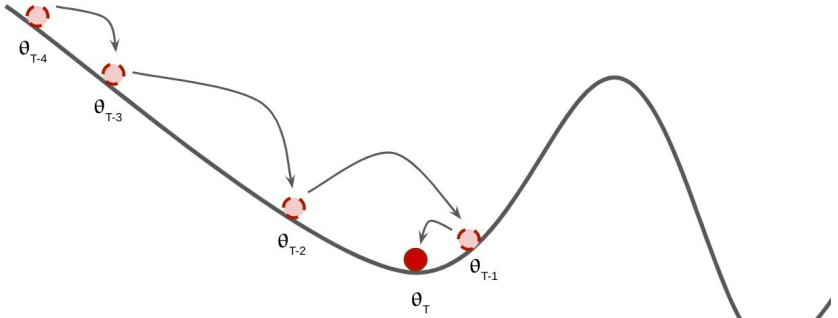


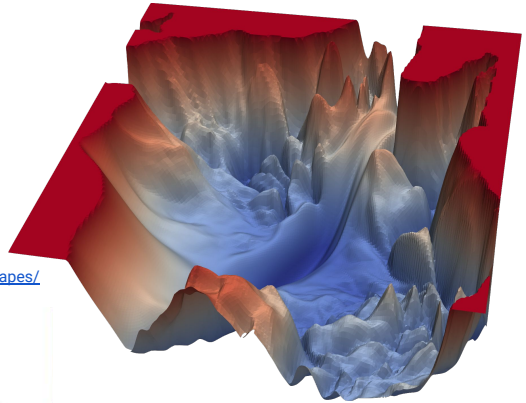
He et al. ICCV 2015 adds a correction for ReLU.



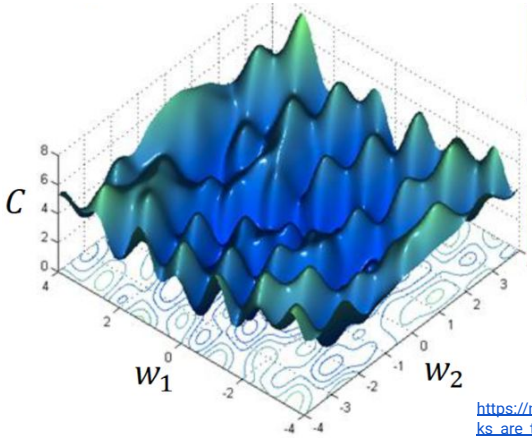
- smaller trust region: high lagrangian penalty, low learning rate, small steps
- larger trust region: low lagrangian penalty, high learning rate, large steps

Is this sufficient ?

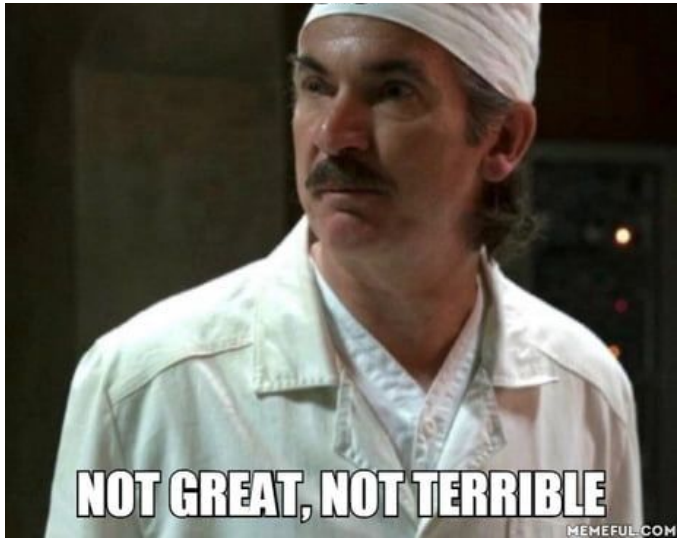




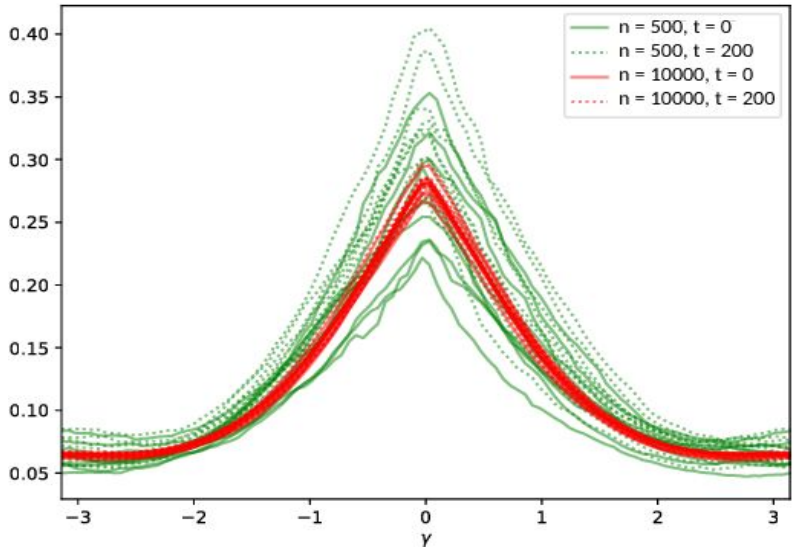
https://www.cs.umd.edu/~tomg/projects/landscapes/



https://ml4a.github.io/ml4a/how_neural_networks_are_trained/



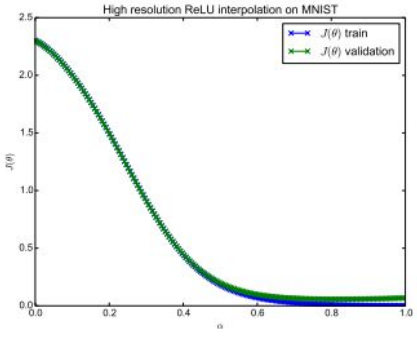
Jacot et al, Neural Tangent Kernel: Convergence and Generalization in Neural Networks



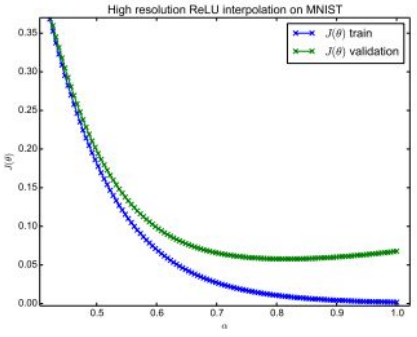
The Deep Learning Myth:

Deep Networks can be inserted almost anywhere and will just work

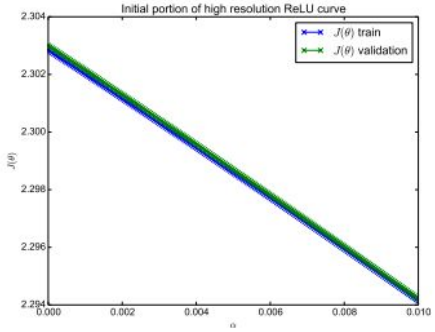
Optimization is as if the loss was convex.



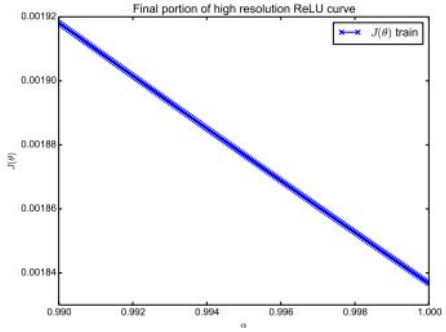
(a)



(b)



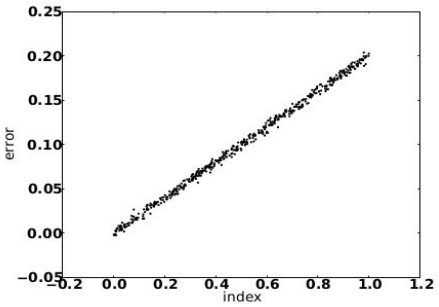
(c)

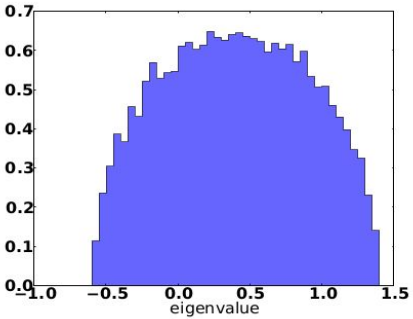


(d)

Goodfellow, Vinyals & Saxe Qualitatively Characterizing Neural Network Optimization Problems, ICLR 2015

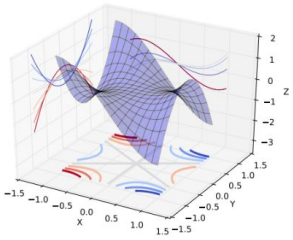
- Statistical physics (on random gaussian fields) [Bray and Dean, 2007, Fyodorov and Williams, 2007]

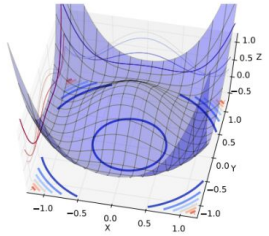


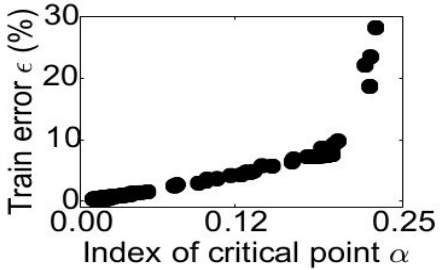


Hypothesis:

- There is a strong correlation between index and error
- It implies local minima are roughly equivalent to global minima
- Main issue in learning is dealing with saddle points





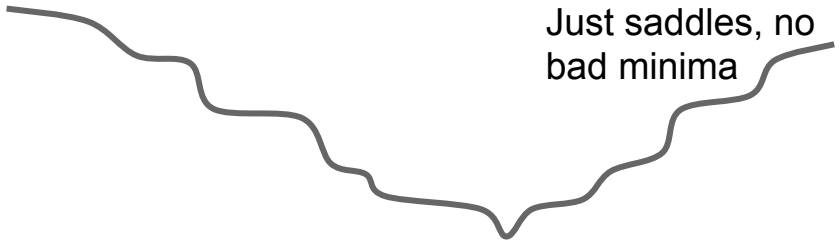


The Deep Learning Myth:

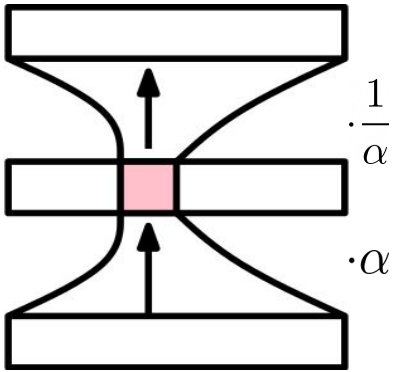
Deep Networks can be inserted almost anywhere and will just work

Optimization is as if the loss was convex.

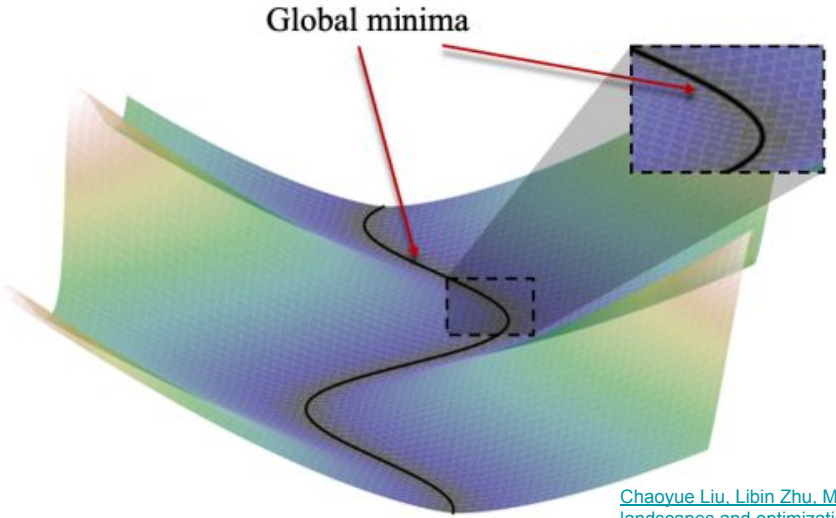
Dauphin et al, *Identifying and attacking the saddle point problem in high-dimensional nonconvex optimization, NIPS* 2014

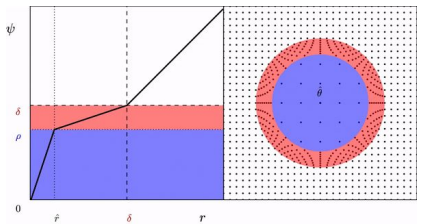


Just saddles, no bad minima



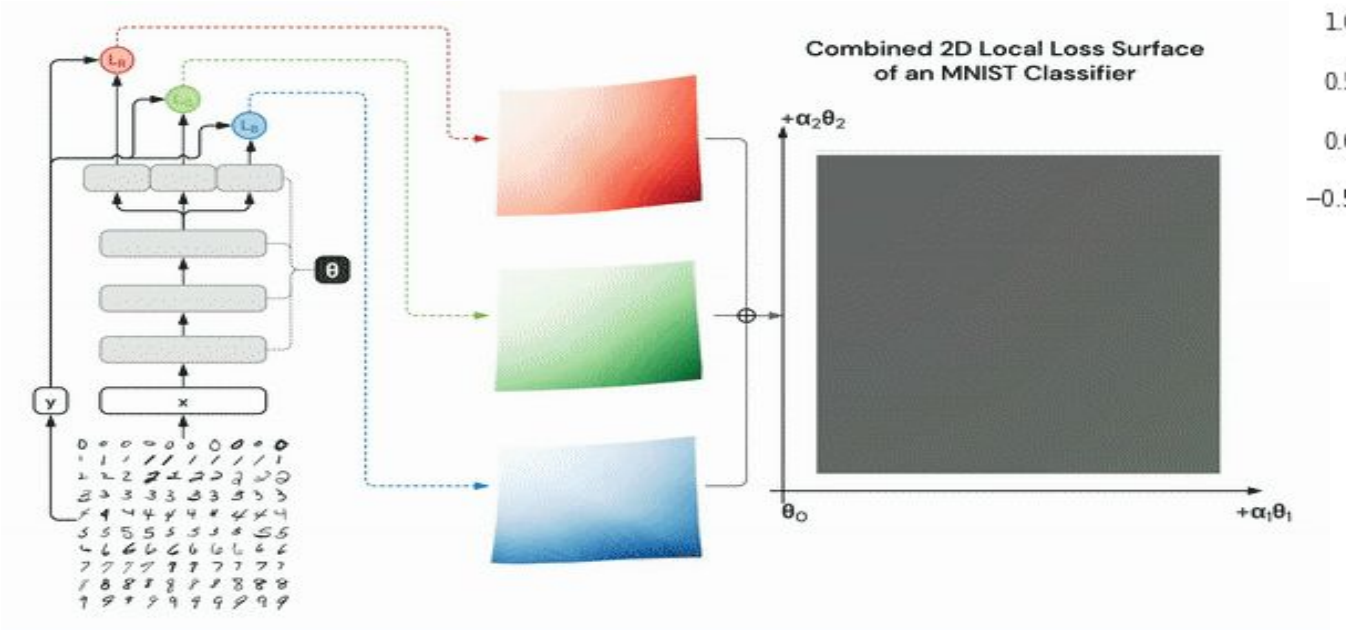
Loss function of a neural network is not even **locally convex**!



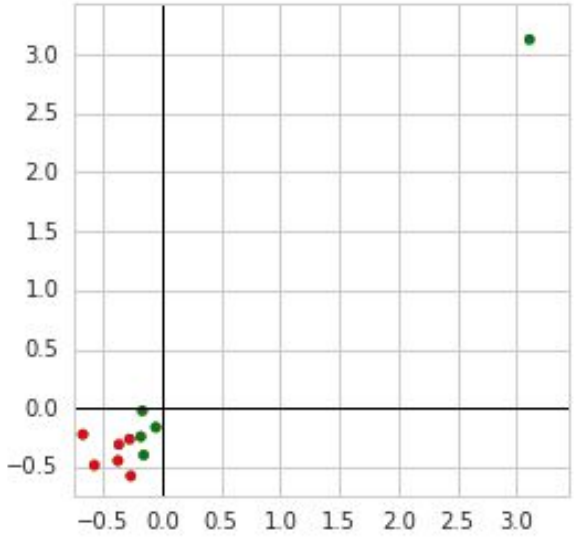


https://arxiv.org/pdf/1703.04933.pdf

Chaoyue Liu, Libin Zhu, Mikhail Belkin, Loss landscapes and optimization in over-parameterized non-linear systems and neural networks



Wojciech Czarnecki, Simon Osindero, Razvan Pascanu, Max Jaderberg,A neural network's loss surface contains every low dimensional pattern



Grzegorz Swirzcz, Wojciech Czarnecki, Razvan Pascanu, Local minima in training neural networks, 2017

Is Gradient Descent efficient ?

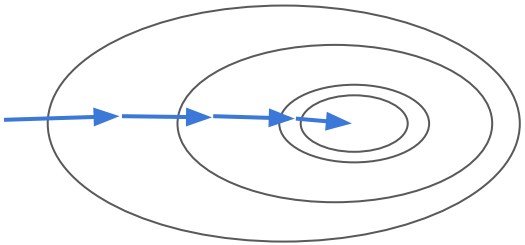
Stochastic GD vs Gradient descent: estimate true gradient by using a small subset of datapoints.

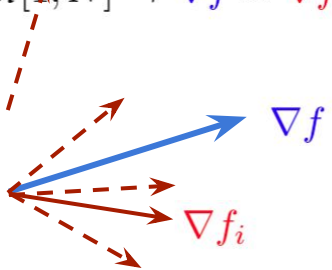
Fundamental assumptions: I.I.D. samples

$$\nabla f \overset{\text{def}}{=} \sum_i \nabla f_i$$

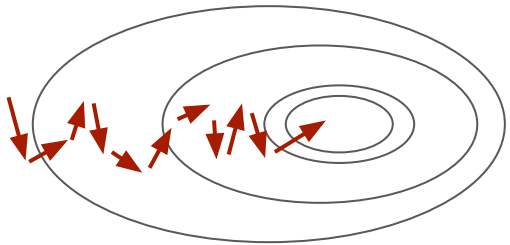
$$i \sim \mathcal{U}[1, N] \rightarrow \nabla f \approx \nabla f_i$$

Gradient Descent





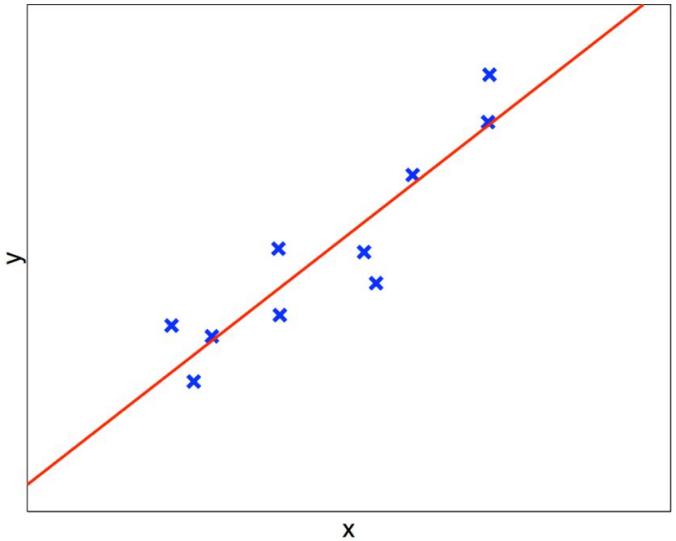
Stochastic Gradient Descent



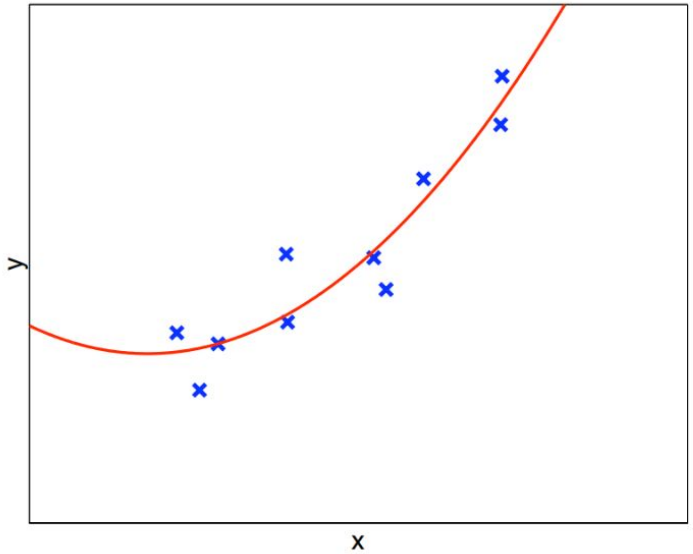
Generalization: Good prediction on new datasets

Example: Data and best linear hypothesis

$$y = 1.60x + 1.05$$

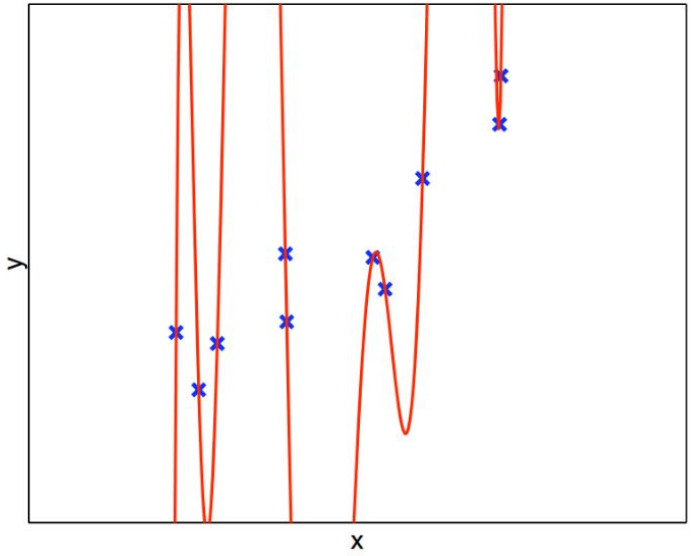


Order-2 fit



Is this a better fit to the data?

Order-9 fit



Is this a better fit to the data?

We want to be able ***to generalize***, i.e. minimize

$$\arg\min_{\theta} L(\theta) = \arg\min_{\theta} \int dist\left(h(\theta, x), \underbrace{y}_{\text{corresponding label for } x}\right) dx$$

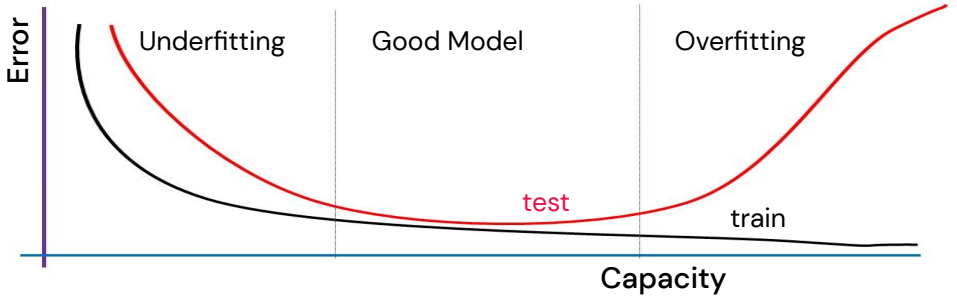
Idea: Assume a stationary distribution π from which we can sample i.i.d. $\Rightarrow$ $\int \propto \mathbb{E}_{(x_i, y_i)\sim\pi}$

We have an unbiased estimate of expectation through samples (**validation set**) i.i.d sampled from π same as the **training set**.

Not the only choice: e.g. prequential learning (*Sequential Learning of Neural Networks for prequential MDL*, Bornschein, Li & Hutter 2022; *Asymptotics of discrete MDL for online prediction*, Poland & Hutter 2005)

Learning is trying to find the minimum of an **intractable function** by minimizing some tractable approximation

- Limit model capacity to avoid overfitting
- Designer needs to pick the *right family of models*



Neural Net

Universal approximator

|W| -> ∞

Neural Net

|W| large

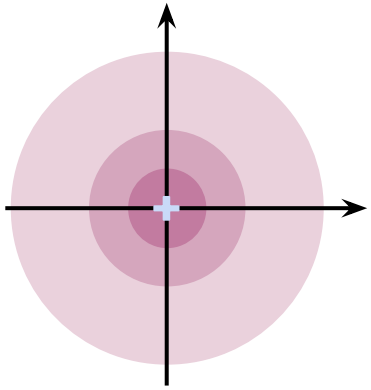
Neural Net

$h(x) = a \cdot x^2 + b \cdot x + c$

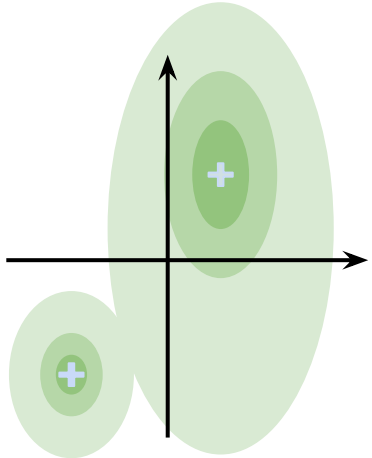
$h(x) = a \cdot x + b$

|W| small

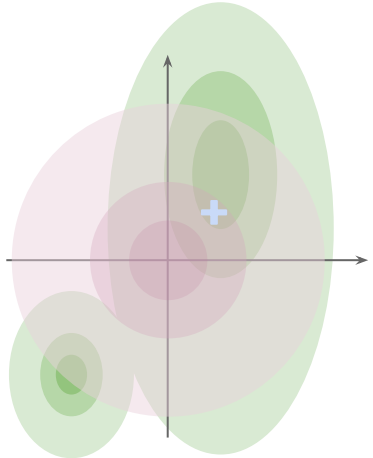
- (Additive) regularization terms. Case Study: $\mathbf{L_2}$



$$\|\theta\|^2$$

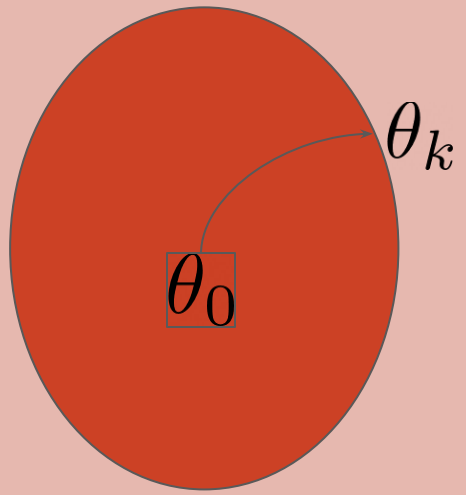


$$\sum_i [h(x_i) - y_i]^2$$



$$\gamma\|\theta\|^2 + \sum_i [h(x_i) - y_i]^2$$

Number of optimization steps = regularization



Universal approximator

Break for Q&A

Part II : Advanced Basics

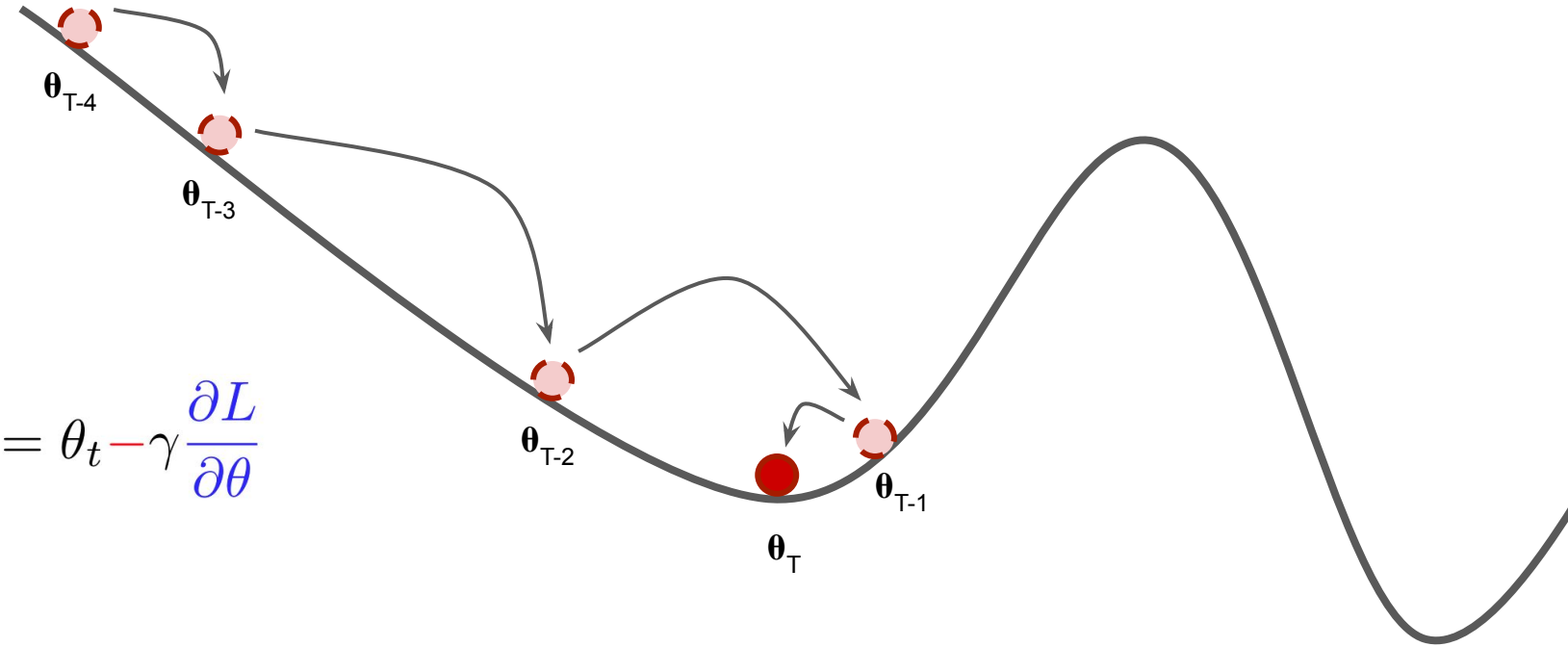
Open-Questions: What inductive biases are enforced by the learning process?
Is the DL Myth reliable?

We will focus on **two** questions:

You are here →

What is a good parametrization ?

How do we find these optimal parameters ?

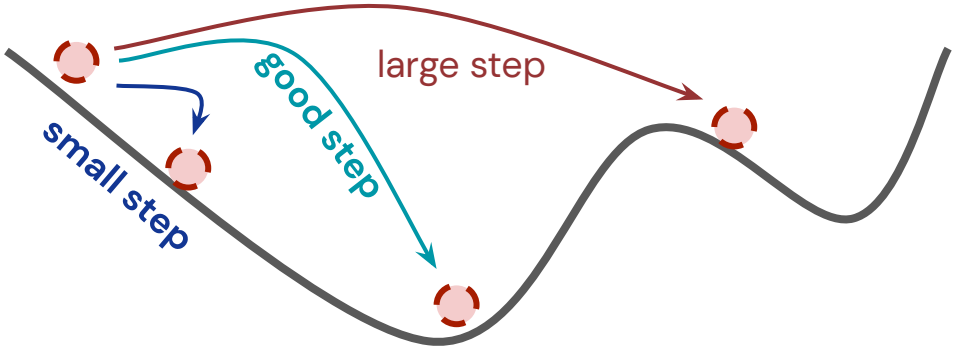


$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L}{\partial \theta}$$

- smaller trust region: high lagrangian penalty, low learning rate, small steps
- larger trust region: low lagrangian penalty, high learning rate, large steps

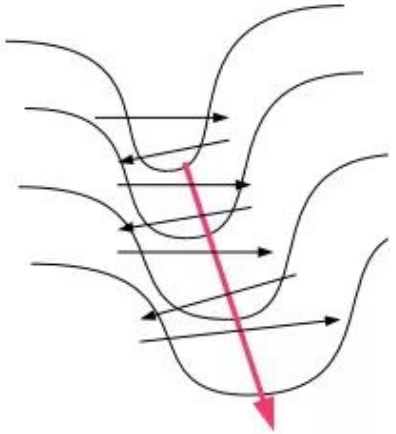
Can we do better?

Second order methods try to take into account *the slope of the loss surface*



Sloppy math:

$$\gamma \propto \frac{1}{\lim_{\epsilon \to 0} \frac{\frac{\partial L}{\partial \theta}\Big|_{\theta} - \frac{\partial L}{\partial \theta}\Big|_{\theta+\epsilon}}{\epsilon}} \propto \underbrace{\frac{\partial^2 L}{\partial \theta^2}}_{=\mathbf{H}}^{-1}$$

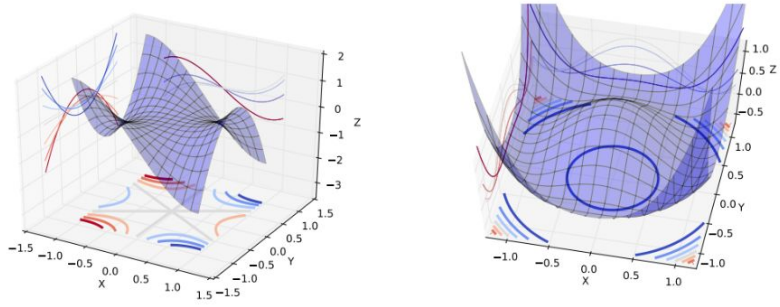


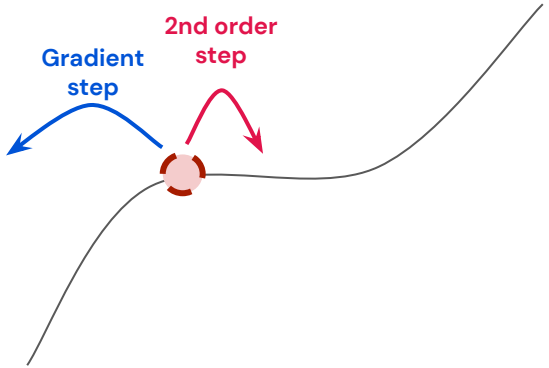
$$\theta_{t+1} = \theta_t - \mathbf{H}^{-1} \frac{\partial L}{\partial \theta}$$

Are saddles important ?

- Around saddles, the Hessian has negative eigenvalues
- Regularizing by identity can fix this (at a price): interpolating towards gradient descent

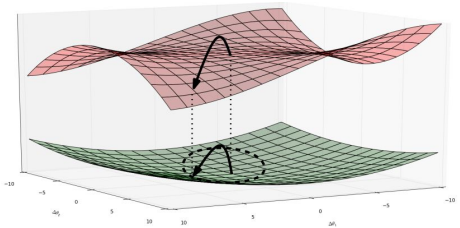
$$\theta_{t+1} = \theta_t - (\eta \mathbf{I} + \mathbf{H})^{-1} \frac{\partial L}{\partial \theta}$$





Can we do better ?

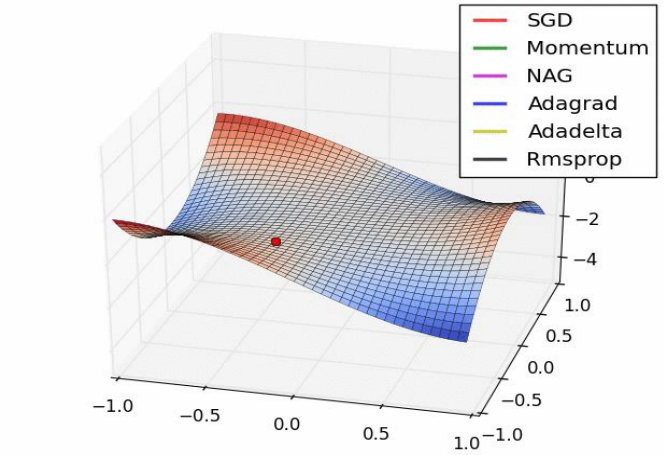
- Yes: Natural Gradient Descent



$$\arg\min_{\Delta\theta} \left[L(\theta) + \Delta\theta \frac{\partial L}{\partial \theta} \right]$$

$$\text{s.t. } KL(h(\theta)||h(\theta + \Delta\theta) \leq \epsilon$$

$$\mathbf{H} = \mathbb{E}\left[\underbrace{\left(\frac{\partial h(\theta}{\partial \theta}\right)\left(\frac{\partial h(\theta)}{\partial \theta}\right)^T}_{\mathbf{F}} \right] \approx \mathbb{E}\left[\underbrace{\left(\frac{\partial L(\theta)}{\partial \theta}\right)\left(\frac{\partial L(\theta)}{\partial \theta}\right)^T}_{\mathbf{F}_{\text{empirical}}} \right]$$

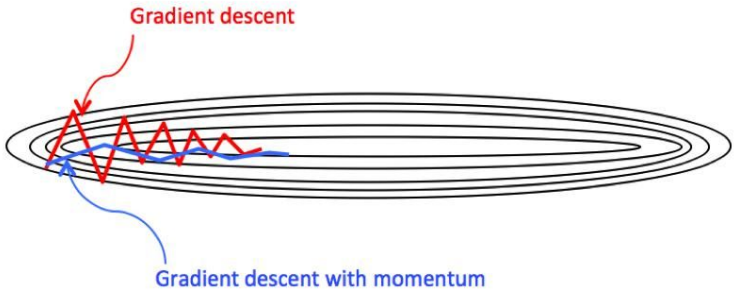


https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39

Momentum

$$V_t = \alpha V_{t-1} + (1-\alpha)\nabla f_i$$

$$\theta_t = \theta_{t-1} - \gamma V_t$$



Rmsprop / ~Adam

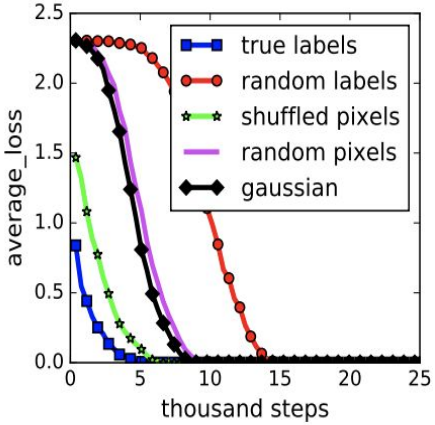
$$V_t = \alpha V_{t-1} + (1-\alpha)\nabla f_i$$

$$S_t = \beta S_{t-1} + (1-\beta)(\nabla f_i)^2$$

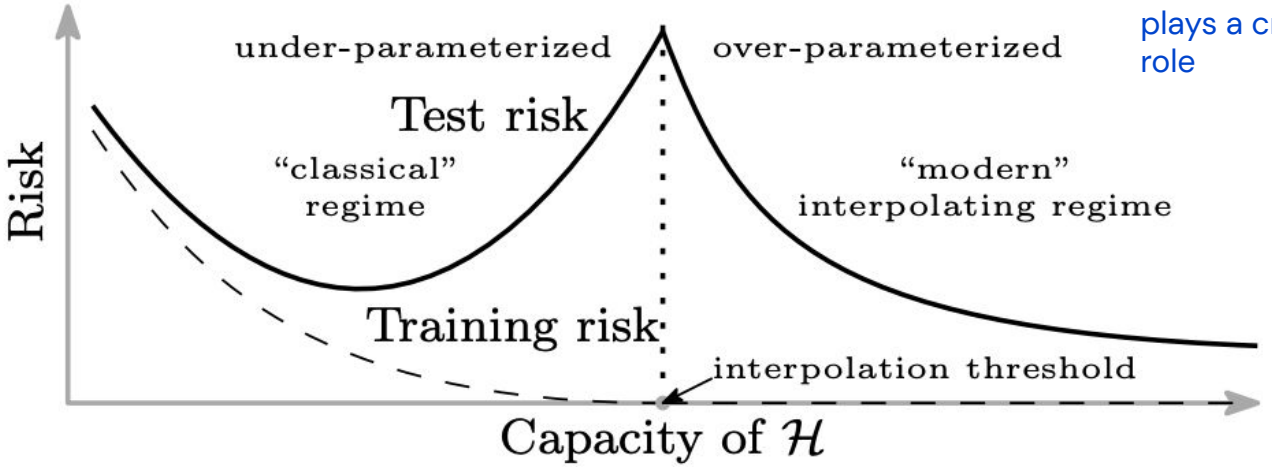
$$\theta_t = \theta_{t-1} - \gamma \frac{V_t}{\sqrt{S_t + \epsilon}}$$

Notes:

- Squared gradients are a proxy for (empirical) Fisher
- A true connection between natural gradient and Adam seems very plausible but also hard to do
- Secret recipe:
  - make better use of local information (maybe dealing with negative curvature)
  - keep computational cost low
  - try to improve generalization



Understanding Deep Learning Requires Rethinking Generalization



Initialization plays a crucial role

Model cannot achieve zero training loss

Multiple solutions with zero training loss

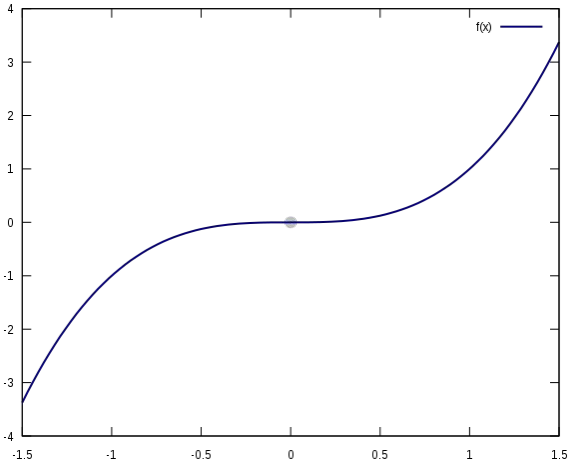
As capacity increases, the best available solution is increasingly complex

Already many solutions available, optimization selects the first solution encountered

Exactly one solution with zero training loss

Double descent curve from *Reconciling modern machine learning practice and the bias-variance trade-off* Belkin et al.
https://arxiv.org/abs/1812.11118

- Priors/regularization terms/inductive biases provide mechanisms to introduce knowledge in the learning problem
- They fundamentally restrict **or reshape** the search space for the optimal parameters
- Any step you do in creating an ML system represents some form of inductive bias. There is no prior-free machine learning systems



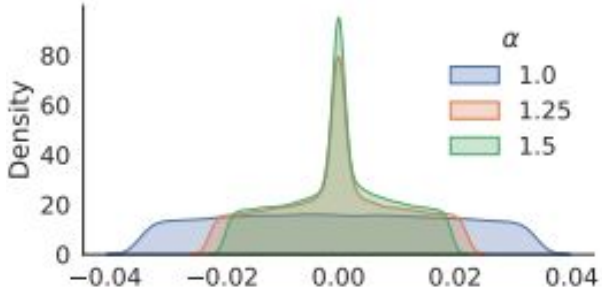
Schwarz et al., Powerpropagation: A sparsity inducing weight reparametrisation

A different way of regularizing:

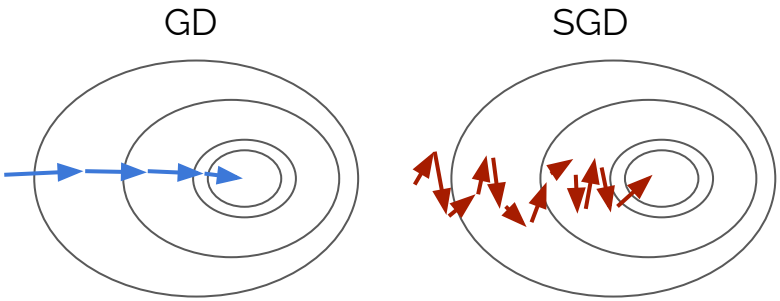
- Preserve the minima of the loss surface
- Change dynamics of GD
- Learning prefers *sparse* solutions

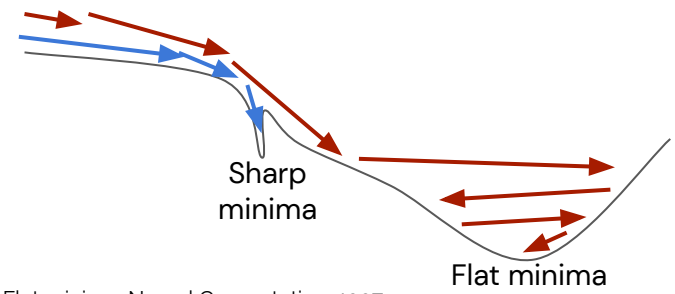
$$\Psi : \mathcal{R} \to \mathcal{R}, \Psi(x) = x|x|^{\alpha-1}$$

$$\frac{\partial \mathcal{L}(\cdot, \Psi(\phi))}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial \Psi(\phi)} \frac{\partial \Psi(\phi)}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial \Psi(\phi)} \mathrm{diag}(\alpha |\phi|^{\circ \alpha - 1}).$$



Why SGD might be better than GD and fancy optimizers? (helpful noise?)





Hochreiter & Schmidhuber, Flat minima, Neural Computation, 1997
More recent: Keskar et al. On large-batch training for Deep Learning: Generalization Gap and Sharp Minima, ICLR 2017

Foret et al, Sharpness-Aware Minimization for Efficiently Improving Generalization

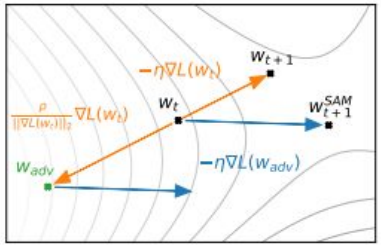
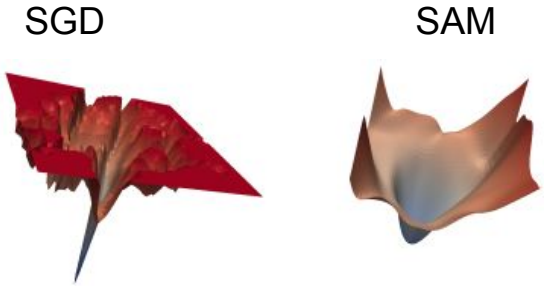
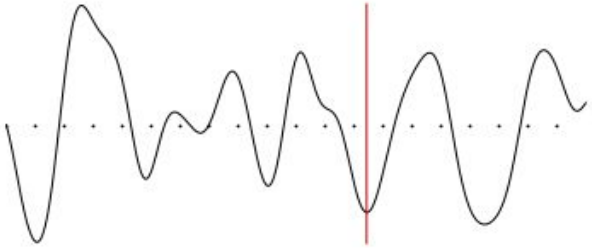


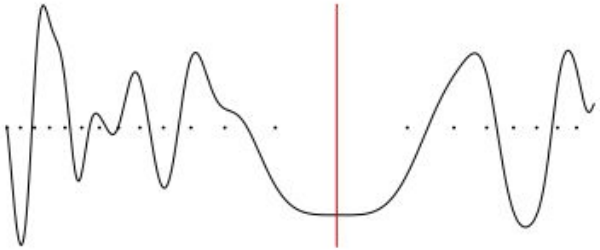
Figure 2: Schematic of the SAM parameter update.



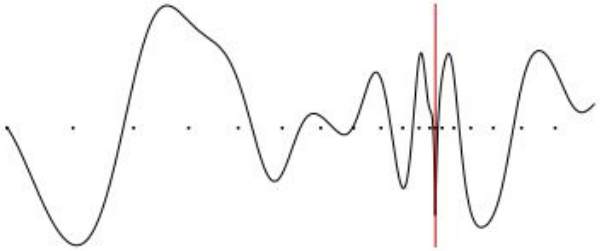
We can change flatness (largest eigenvalue) without changing the function !
We need a more robust measure of flatness
Dinh et al, 2017
https://arxiv.org/pdf/1703.04933.pdf



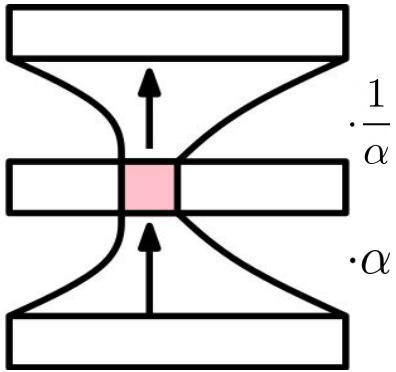
(a) Loss function with default parametrization

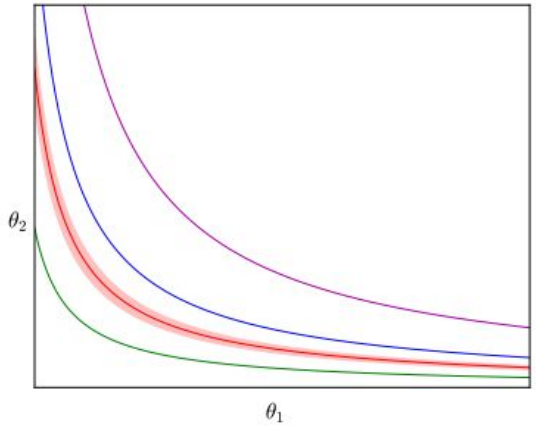


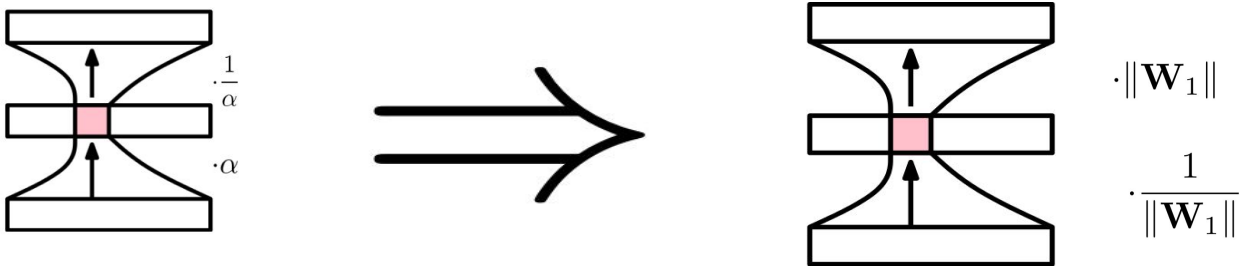
(b) Loss function with reparametrization



(c) Loss function with another reparametrization



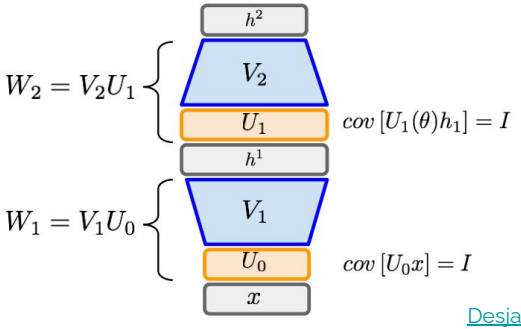


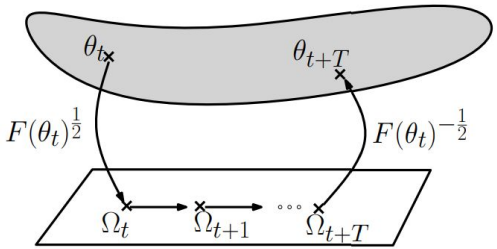


Neyshabur et al. 2015 Path-SGD: Path-Normalized Optimization in Deep Neural Networks,

$$\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial \theta}\right)\left(\frac{\partial L(\theta)}{\partial \theta}\right)^T\right] = \mathbb{E}\left[\begin{bmatrix} \ddots & & \\ & h_i(\theta)\frac{\partial L}{\partial h_i} & \\ & & \ddots \end{bmatrix}\begin{bmatrix} \ddots & & \\ & h_i(\theta)\frac{\partial L}{\partial h_i} & \\ & & \ddots \end{bmatrix}^T\right]$$

$$\mathbb{E}\left[\left(h_i(\theta)\frac{\partial L}{\partial h_i}\right)\left(h_i(\theta)\frac{\partial L}{\partial h_i}\right)^T\right] \approx \mathbb{E}\left[\underbrace{h_i(\theta)h_i(\theta)^T}_{cov(h_i)}\right]\mathbb{E}\left[\underbrace{\left(\frac{\partial L}{\partial h_i}\right)\left(\frac{\partial L}{\partial h_i}\right)^T}_{cov\left(\frac{\partial L}{\partial h_i}\right)}\right]$$

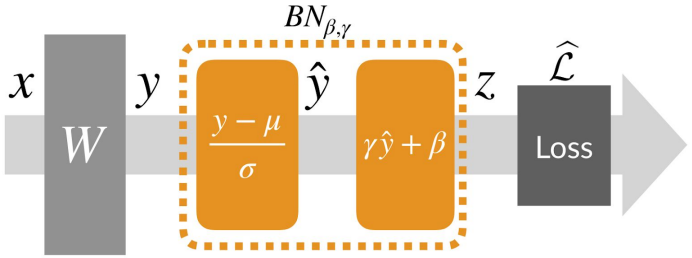




Desjardins et al. 2015, Natural Neural Networks

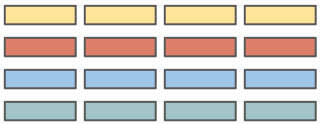
Normalization layers to improve learning

Weight Standardisation/ Layer Norm / Batch Norm



Batch size

Feature size

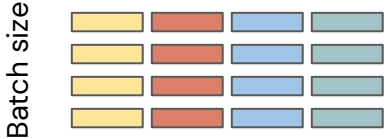


[Ioffe & Szegedy 2015, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariant Shift]

[Ba et al. 2016, Layer Normalization]

Batch size

Feature size



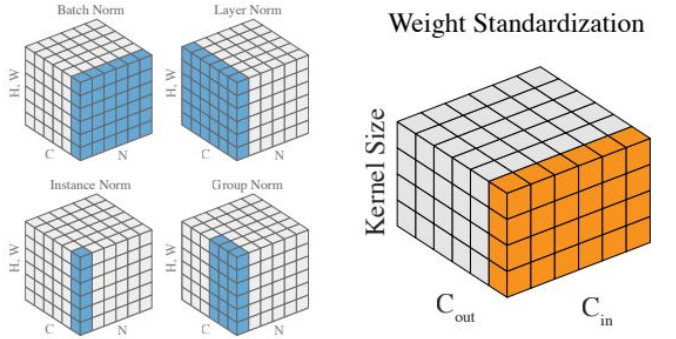


Fig. 2: Comparing normalization methods on activations (blue) and Weight Standardization (orange).

[Qiao et al 2019, Micro-Batch Training with Batch-Channel Normalization and Weight Standardization]

Re-parametrization to make learning well behaved is interesting, but of course one can stick with a traditional approach to correct gradient by curvature:

$$\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial \theta}\right)\left(\frac{\partial L(\theta)}{\partial \theta}\right)^T\right] = \mathbb{E}\left[\begin{bmatrix} \ddots & & \\ & h_i(\theta)\frac{\partial L}{\partial h_i} & \\ & & \ddots \end{bmatrix}\begin{bmatrix} \ddots & & \\ & h_i(\theta)\frac{\partial L}{\partial h_i} & \\ & & \ddots \end{bmatrix}^T\right]$$

Martens & Grosse 2015, Optimizing Neural Networks with Kronecker-factored Approximate Curvature

$$\mathbb{E}\left[\left(h_i(\theta)\frac{\partial L}{\partial h_i}\right)\left(h_i(\theta)\frac{\partial L}{\partial h_i}\right)^T\right] \approx \mathbb{E}\left[\underbrace{h_i(\theta)h_i(\theta)^T}_{cov(h_i)}\right]\mathbb{E}\left[\underbrace{\left(\frac{\partial L}{\partial h_i}\right)\left(\frac{\partial L}{\partial h_i}\right)^T}_{cov\left(\frac{\partial L}{\partial h_i}\right)}\right]$$

Woodbury matrix identity

$$\left(A+\mathbf{u}\mathbf{v}^\top\right)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^\top A^{-1}}{1+\mathbf{v}^\top A^{-1}\mathbf{u}}$$

Singh & Alistarh 2020, WoodFisher: Efficient Second-Order Approximation for Neural Network Compression

$$\Downarrow$$

$$\widehat{F}_{n+1}^{-1} = \widehat{F}_n^{-1} - \frac{\widehat{F}_n^{-1}\nabla\ell_{n+1}\nabla\ell_{n+1}^\top\widehat{F}_n^{-1}}{N+\nabla\ell_{n+1}^\top\widehat{F}_n^{-1}\nabla\ell_{n+1}}, \quad \text{where} \quad \widehat{F}_0^{-1} = \lambda^{-1}I_d$$

Why SGD might be better than GD and fancy optimizers? (helpful noise?)

Revisiting evidence say is not just noise !

Barrett & Dherin 2020 https://arxiv.org/abs/2009.11162
Smith et al 2021 https://openreview.net/forum?id=rq_Qr0c1Hyo

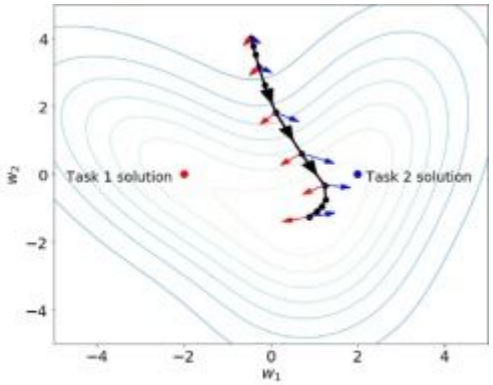
Punch line: SGD optimizes a different objective than GD

$$\tilde{L}_{GD}(\theta) = L(\omega) + \|\|\|\nabla \hat{L}(\theta)\|\|\|^2$$

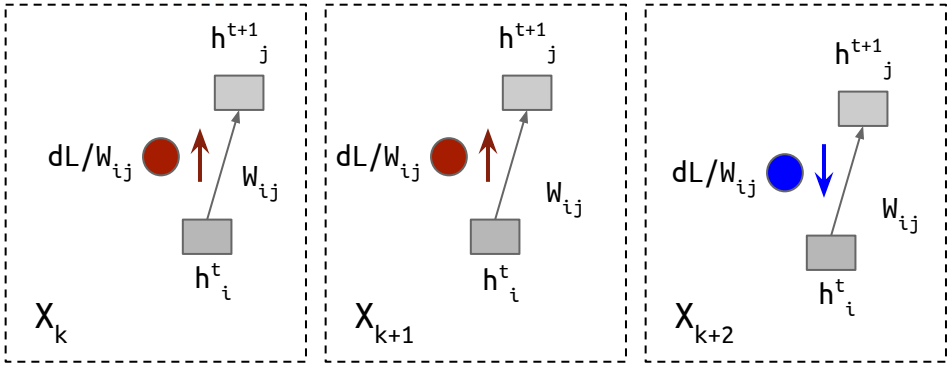
$$\tilde{L}_{SGD}(\theta) = L(\theta) + \frac{\epsilon}{4m} \sum_{k=0}^{m-1} \|\|\|\nabla \hat{L}_k(\theta)\|\|\|^2$$

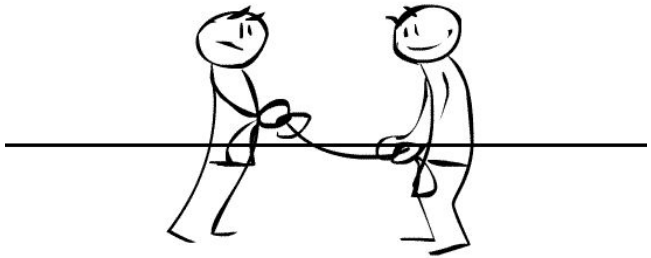
The IID assumption in Gradient Descent

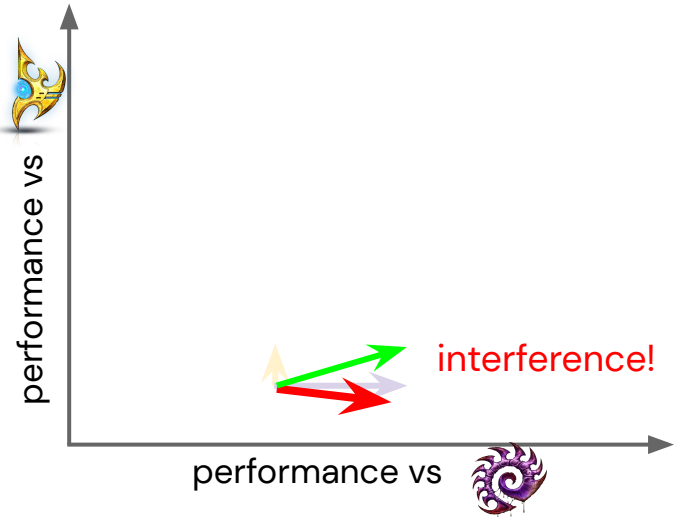
- **Independently for each data point and parameter** ask how much the loss would change if a parameter is increased or decreased in magnitude
- Modify the parameter by taking **a small step** proportional with the impact it has on the loss

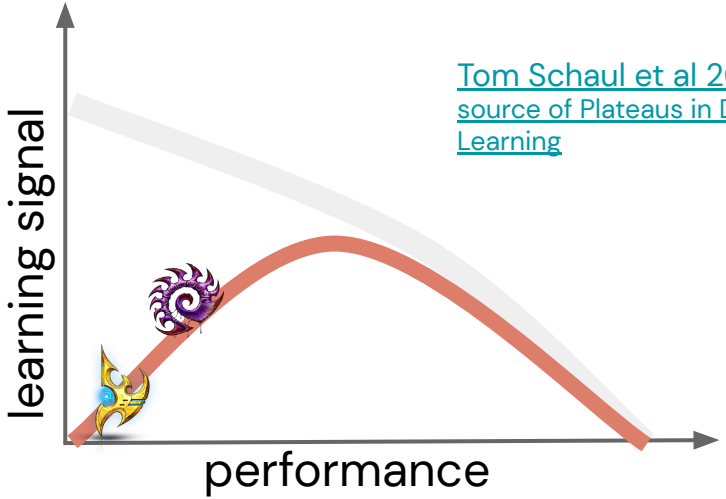


$$\frac{\partial L}{\partial W_{ij}} = \lim_{\epsilon \to 0} \frac{L(W_{ij}) - L(W_{ij} + \epsilon)}{\epsilon}$$

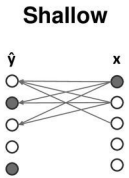


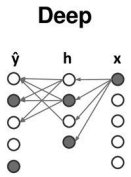


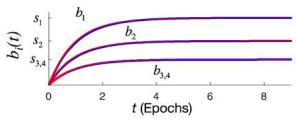


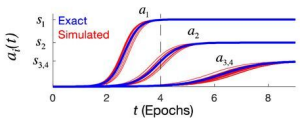


Tom Schaul et al 2019, Ray interference: a source of Plateaus in Deep Reinforcement Learning









Andrew Saxe et al 2013,Exact solutions to the nonlinear dynamics in deep linear models

Learning has **tug-of-war** dynamics to resolve credit assignment

- This requires data to be **I.I.D.**
- **No explicit knowledge composition:**
  - Catastrophic forgetting
  - Lack of forward (& backward) transfer
  - Potential issues with plasticity
    - Inability to generalize
    - Inability to learn

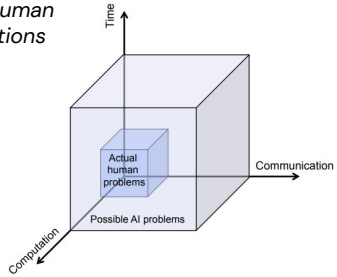
Continual Learning can be seen as a study of this topic, and how to move away from the IID assumption (no train / test set)

Strong connections to online learning

E.g.:
Hadsell et al 2020, Embracing Change: Continual Learning in Deep Neural Networks
Lyle et al 2023, Understanding plasticity in neural networks

Efficiency of learning

Thomas Griffiths, *Understanding Human Intelligence through Human Limitations*



Hypothesis (Thomas Griffiths): *The profile of human intelligence is due to its limitations*

- E.g.: AlphaGo's unnatural moves are potentially due to the lack of decomposition of the problem into tracktable subproblems [humans explicitly sacrifice performance for efficiency]

- ML systems have limitations (including computational)!, though they are different.

Hypothesis:

- We should exploit limitations (to shape the loss surface)
- One way to control the kind of solutions we learn, is to add limitations during learning or measure what we care (e.g. how efficiently we learn vs how well we do)

Further potential implications of Continual Learning on Learning/Optimization:

- learning as tracking:
  - What does optimality mean?
  - No more loss surface or curvature (at least not traditionally) or overfitting
  - Convergence rate (?)

Part II : Advanced Basics

Open-Questions: What are inductive biases of learning?
Is the DL Myth reliable?

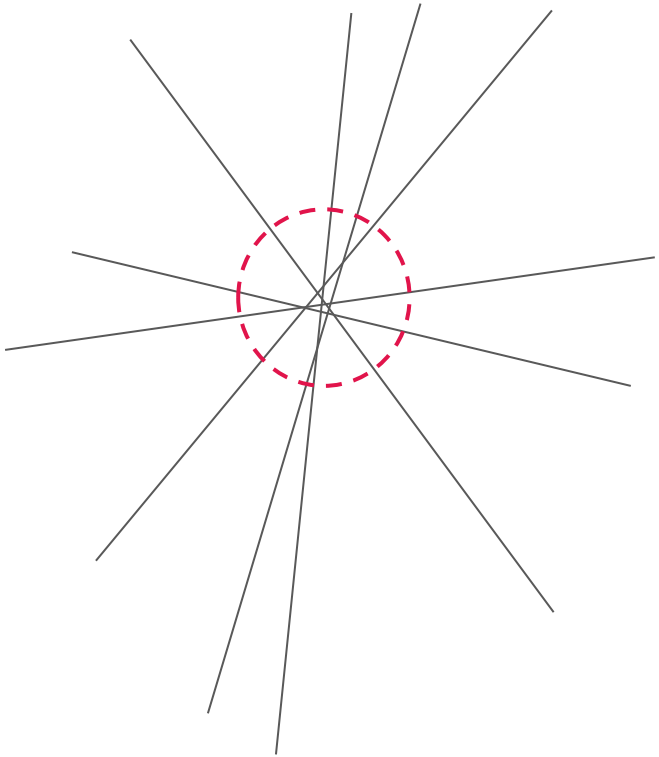
We will focus on **two** questions:

You are here → **What is a good parametrization ?**

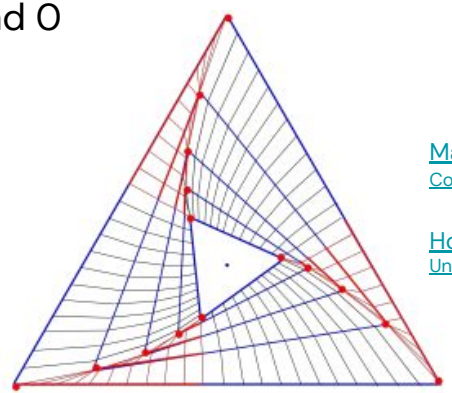
How do we find these optimal parameters ?

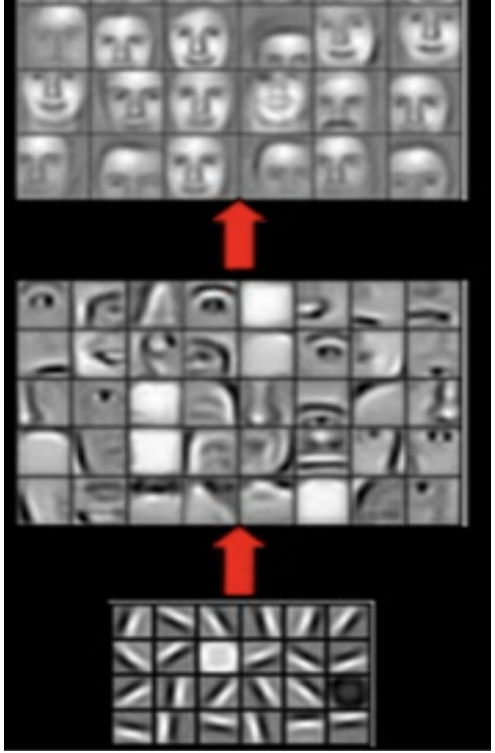
Inductive bias of initialization
Inductive bias of depth

Most regions found around 0.
Towards infinity, behaviour becomes linear



vs Convolutions -- adding regularities in linear regions around 0



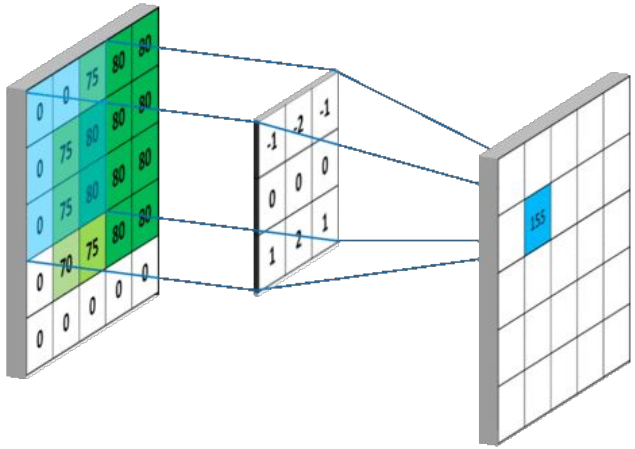


Matt Zeiler & Rob Fergus 2013, Visualizing and Understanding Convolutional Networks

Honglak Lee et al. 2009, Convolutional Deep Belief Networks for Sca Unsupervised Learning of Hierarchical Representations

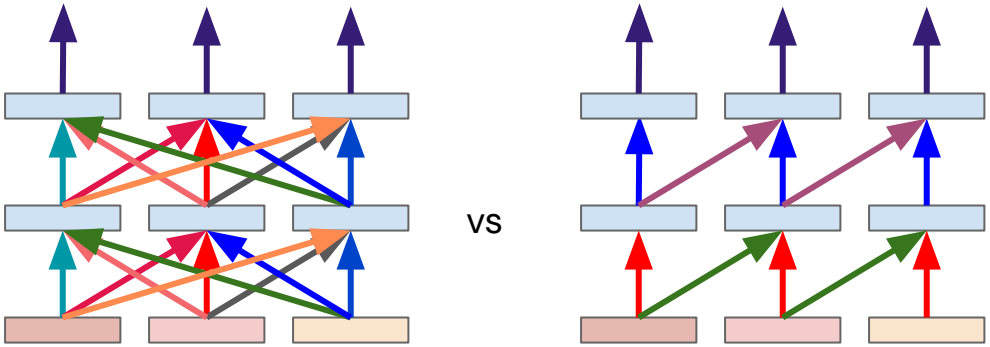
Convolutional Neural Networks

- **Structural prior:** spatial neighbourhood defines the role of a pixel
- Apply **same function** at all position
- Induces translation invariance as features are computed independent of position

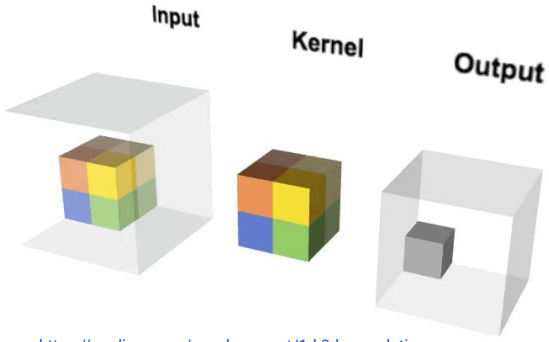


https://www.analyticsindiamag.com/convolutional-neural-network-image-classification-overview/

Can an MLP reproduce a ConvNet?

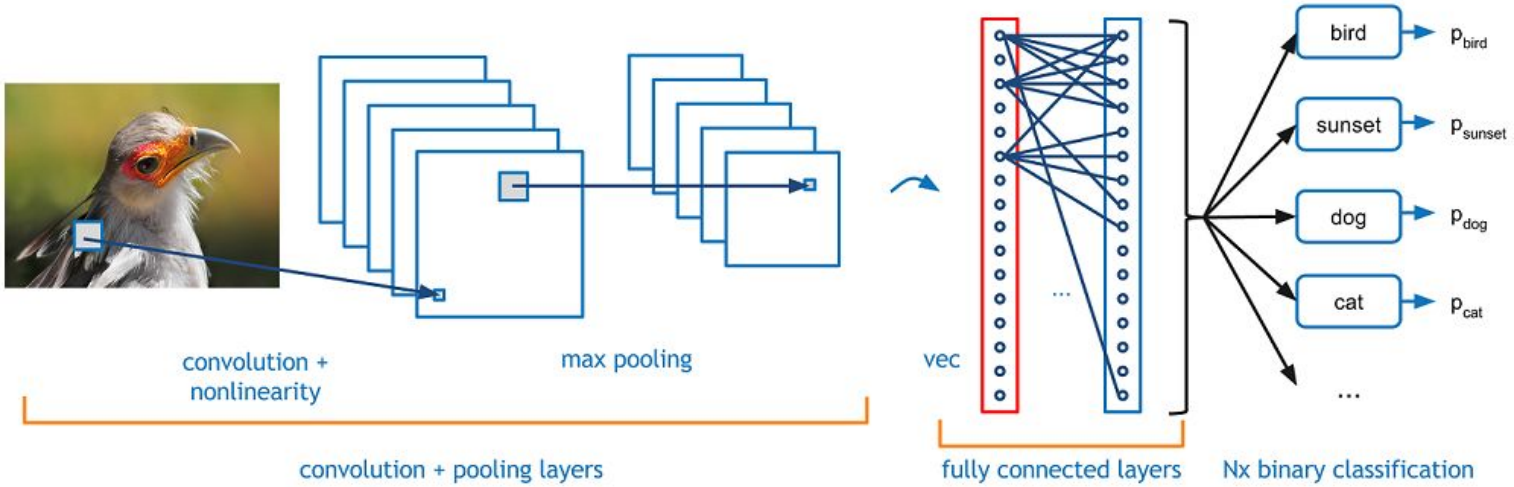


MLP vs Convolutional Network



https://medium.com/apache-mxnet/1d-3d-convolutions-explained-with-ms-excel-5f88c0f35941

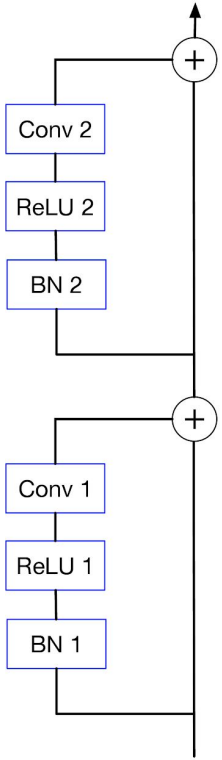
- Convolutions to compute features based on local regions
- Pooling (mostly to reduce computational cost .. not as used anymore)
- Fully connected layers (to gain global view of the input)
- Softmax to convert output in probabilities

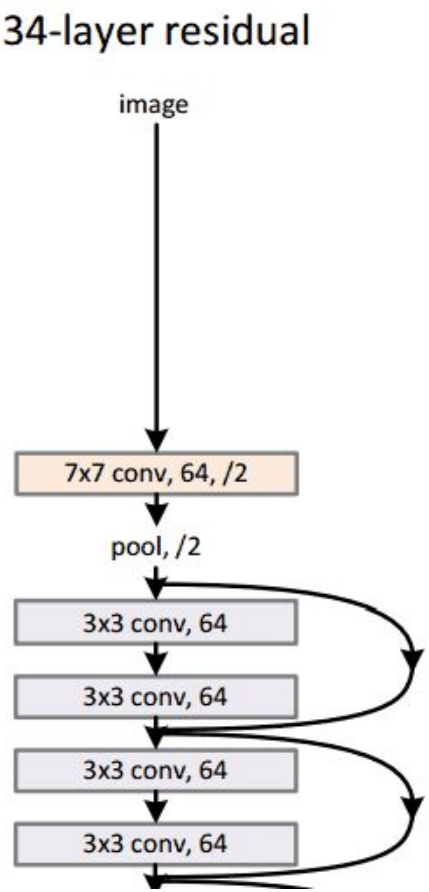


https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

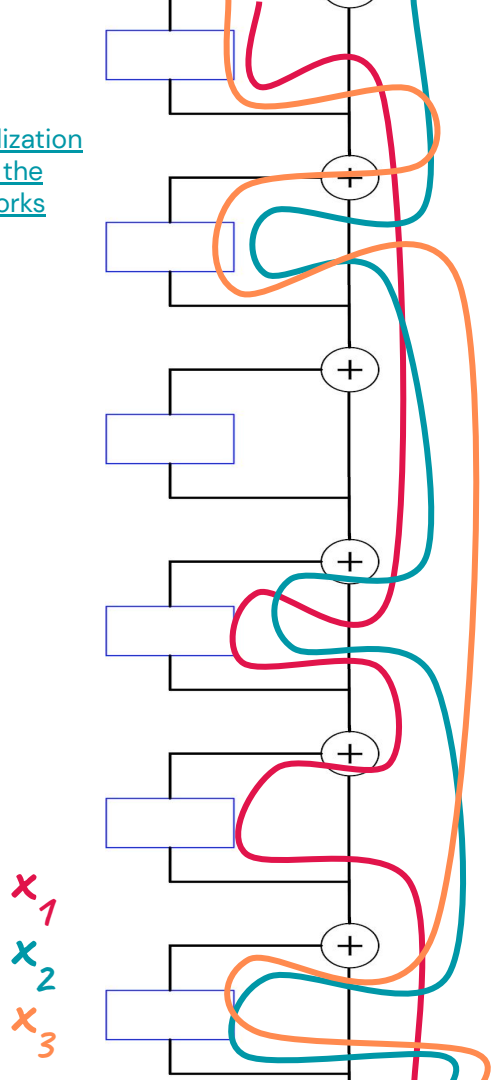
Residual Blocks

He et al. 2015



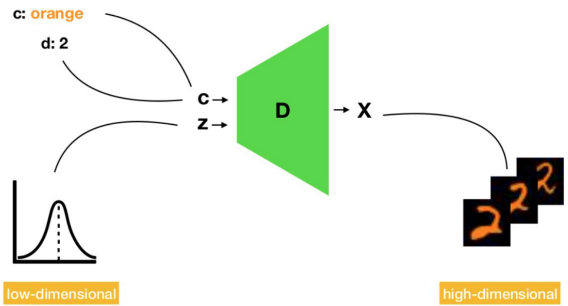


De & Smith 2020, Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks



Other tasks might require different architecture choice

Image to image translation



Famous UNet architecture:

Ronneberger et al. 2015

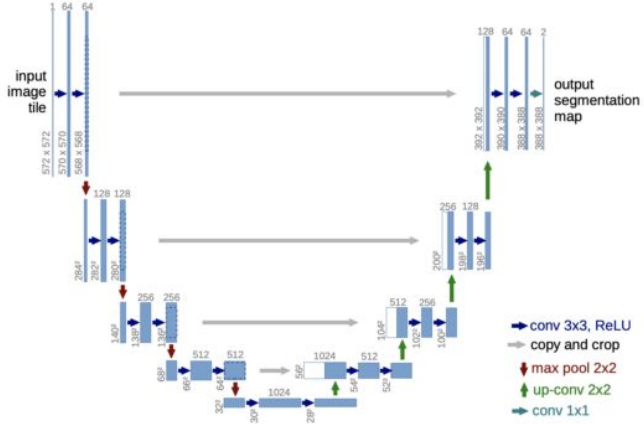


Image generation

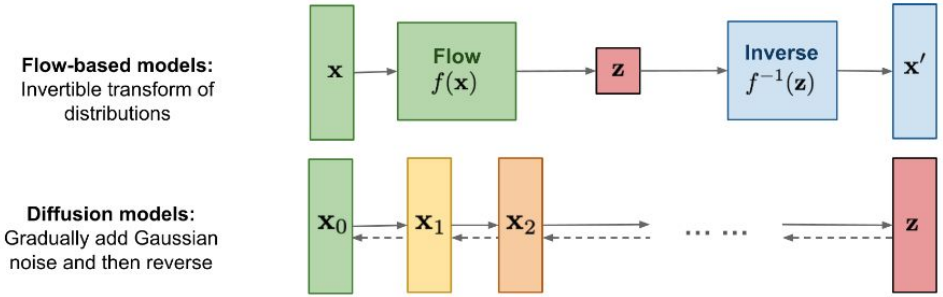
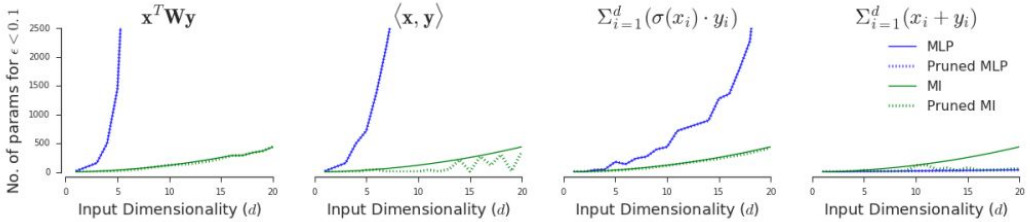
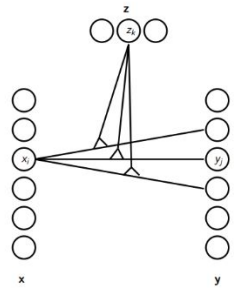


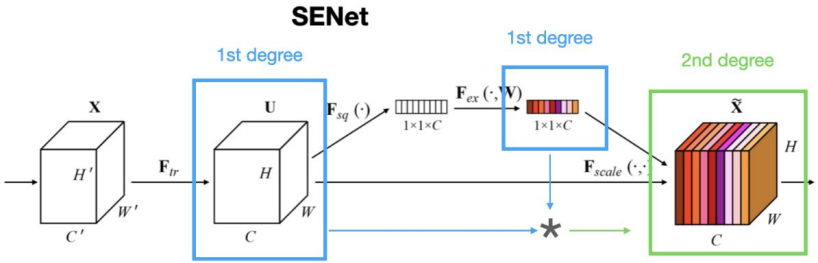
Fig. 1. Overview of different types of generative models.

Lil'log: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Polynomial neural networks (multiplicative interactions)



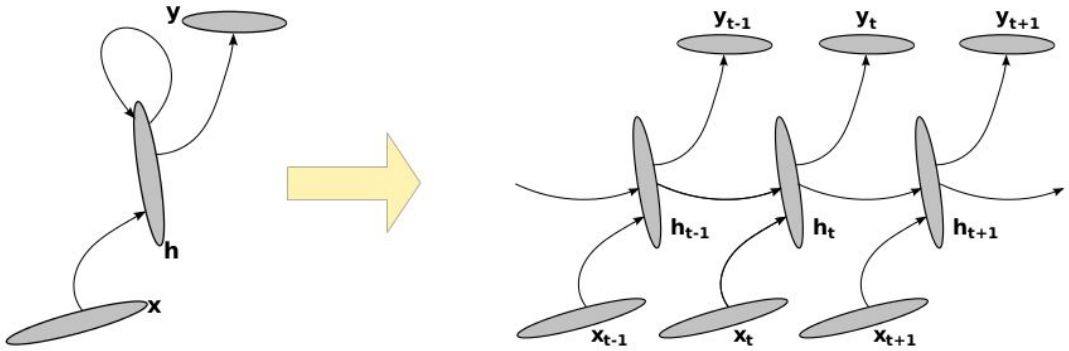


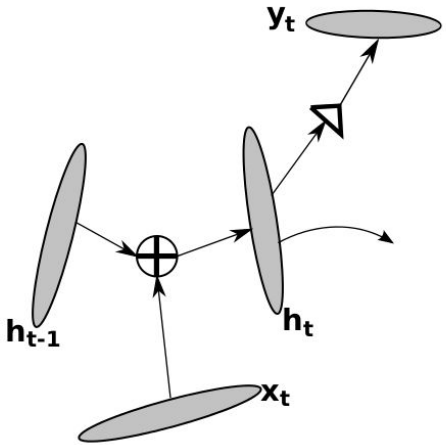


J Hu, L Shen, G Sun, *Squeeze-and-excitation networks* 2018

- Mapping Units [Hinton, 1985], "dynamic mapping" [v.d. Malsburg; 1981]
- Binocular+Motion Energy models [Adelson, Bergen; 1985], [Ozhawa, DeAngelis, Freeman; 1990], [Fleet et al., 1994].
- Sigma-Pi neural unit [Mel, Koch; 1990].
- Higher Order Botlzmann Machines / Higher Order Neural Networks [Sejnowski; 1986].
- Subspace SOM [Kohonen; 1996], topographic ICA [Hyvarinen, Hoyer; 2000] [Karklin, Lewicki;2003].
- Bilinear Models [Tenebaum and Freeman; 2008], [Ohlshaussen; 1994], [Grimes, Rao; 2005].
- Higher Order Restricted Boltzmann Machines (RBMs) [Memisevic and Hinton; 2007], [Ranzato et al; 2010].
- Gating mechanisms; LSTM [Hochreiter, Schmidhuber 1997], Multiplicative RNN [Sutskever, Martens, Hinton; 2011].

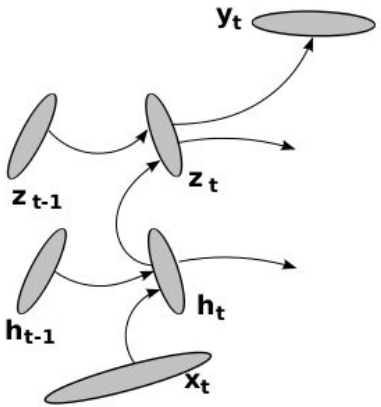
Recurrent Neural Networks



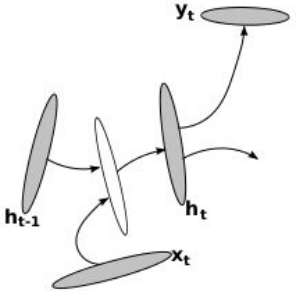


Pascanu et al. 2014

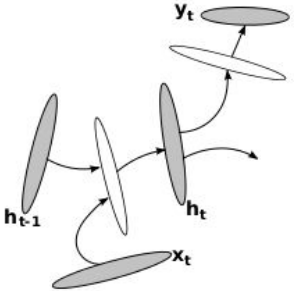
The concept of Depth for RNNs..



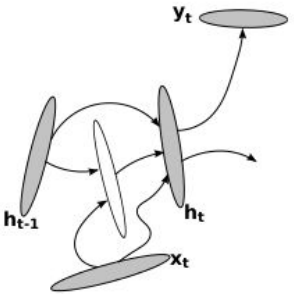
Stacked RNNs



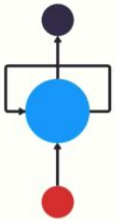
DT-RNN



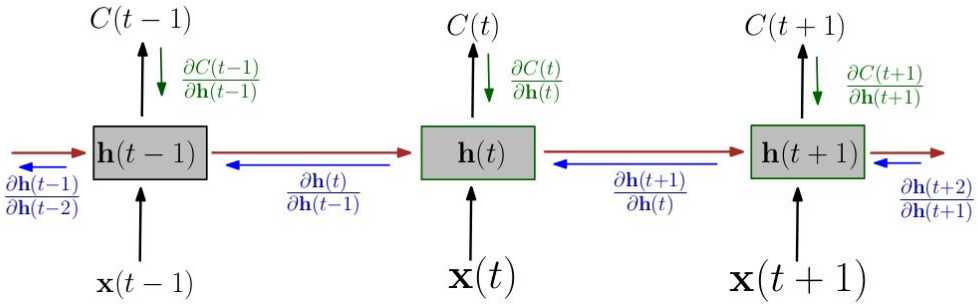
DOT-RNN



DOT(s)-RNN



Exploding Gradients



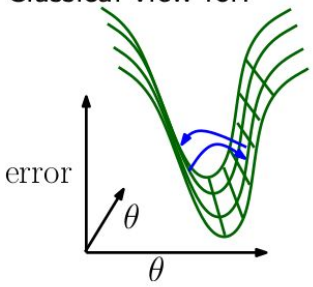
$$\frac{\partial C}{\partial \mathbf{W}} = \sum_t \frac{\partial C(t)}{\partial \mathbf{W}} = \sum_t \sum_{k=0}^{t} \frac{\partial C(t)}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(t-k)} \frac{\partial \mathbf{h}(t-k)}{\partial \mathbf{W}}$$

$$\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(t-k)} = \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}(j)}{\partial \mathbf{h}(j-1)}$$

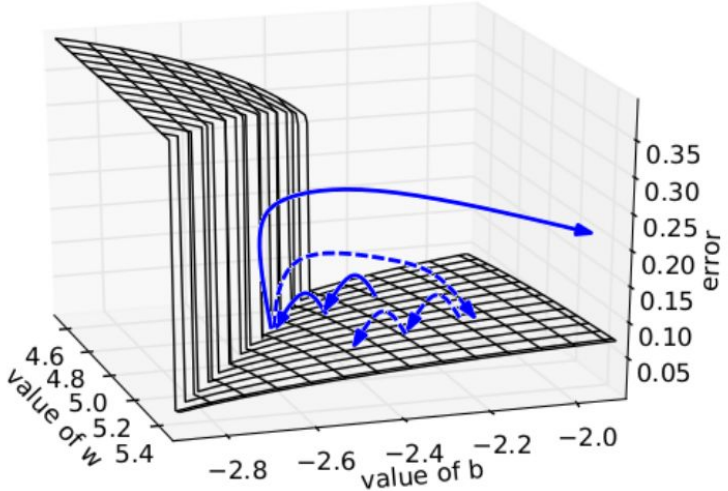
Algorithm 1 Pseudo-code for norm clipping

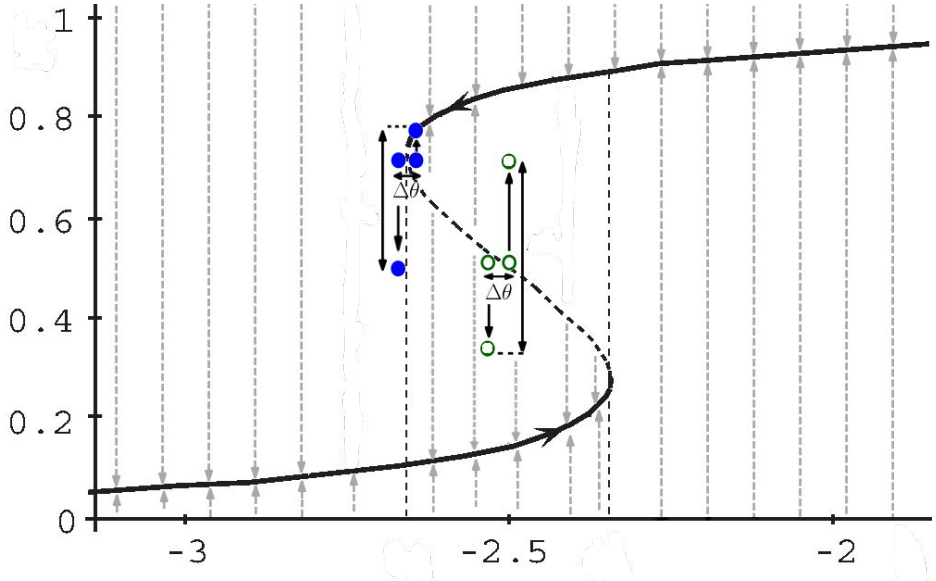
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

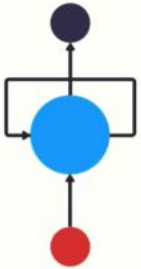
Classical view for:



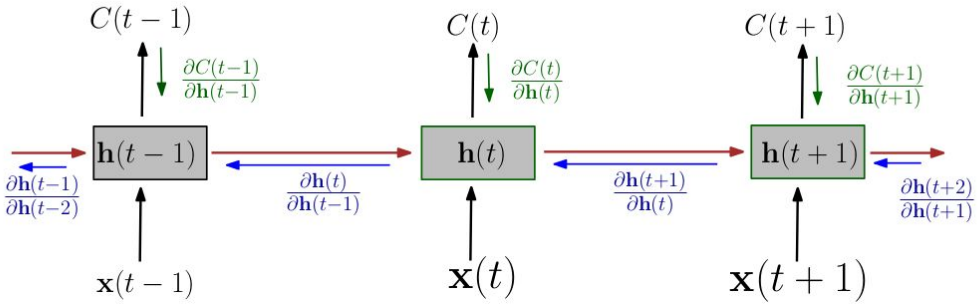
The error is $(h(50) - 0.7)^2$ for $h(t) = w\sigma(h(t-1)) + b$ with $h(0) = 0.5$







Vanishing Gradients

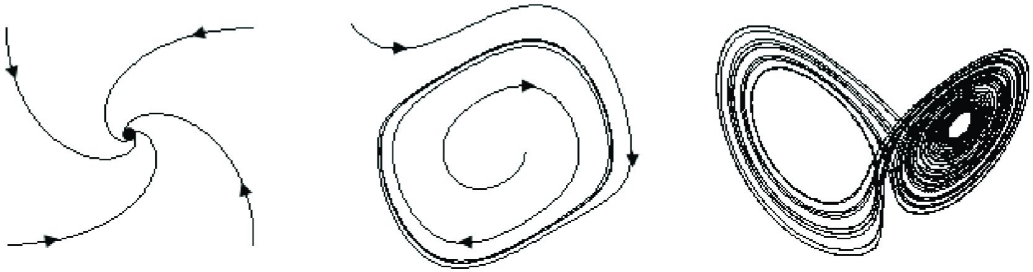


$$\frac{\partial C}{\partial \mathbf{W}} = \sum_t \frac{\partial C(t)}{\partial \mathbf{W}} = \sum_t \sum_{k=0}^{t} \frac{\partial C(t)}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(t-k)} \frac{\partial \mathbf{h}(t-k)}{\partial \mathbf{W}}$$

$$\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(t-k)} = \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}(j)}{\partial \mathbf{h}(j-1)}$$

Vanishing Gradients

$$\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(t-k)} = \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}(j)}{\partial \mathbf{h}(j-1)}$$



Fixed point attractor | Limit-cycle attractor | Strange attractor

- Weights largest eigenvalue < 1, damping regime [fixed point attractor]
- Weights close or 1, information travels through the system
- Weights largest eigenvalue > 1, potentially in a chaotic regime

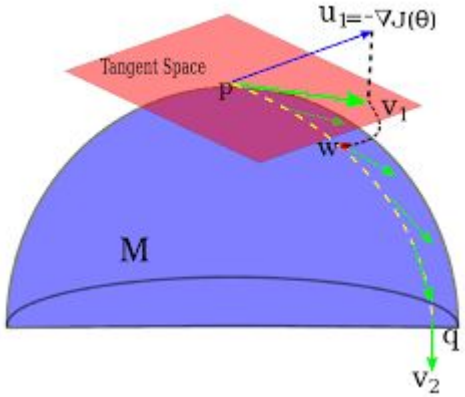
Echo State Network literature, e.g. : http://www.scholarpedia.org/article/Echo_state_network
Ilya Sutskever et al 2013, On the importance of initialization and momentum in deep learning

Saxe et al. 2014 :

- Orthogonal weights as solution for deep linear models

Henaff et al. 2016; Arjovski et al. 2016

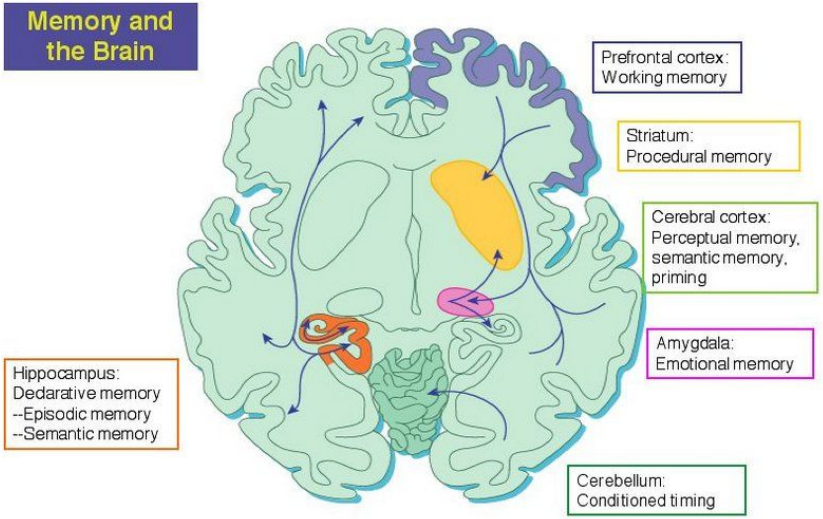
- Project recurrent weights to the space of orthogonal matrices



$$\Omega = \sum_k \Omega_k = \sum_k \left(\frac{\left\| \frac{\partial C}{\partial \mathbf{h}_{k+1}} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \right\|}{\left\| \frac{\partial C}{\partial \mathbf{h}_{k+1}} \right\|} - 1 \right)^2$$

Pascanu et al. 2012

But don't we need to forget !?
How should we think about memory ?!



[Memory: Scholarpedia]

$$\frac{\partial(x_1 + x_2 + x_3)}{\partial x_3} = 1$$

$$x_1 + x_2 + x_3 = 10$$

$$x_3 = ?$$

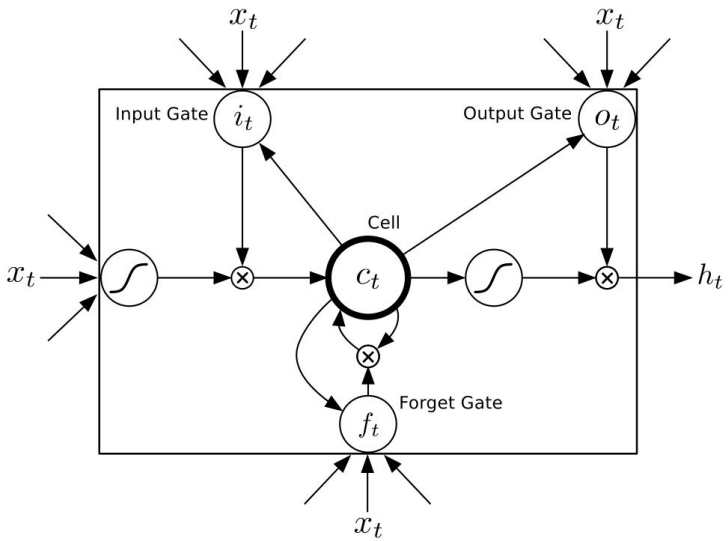
Memory can be a tricky concept
Cognitive Science:

- Episodic Memory // Long term memory
- Short term memory // Working memory
- Declarative vs Procedural vs Emotional Memory

In ML:

- Weights vs state
- What part of the input signal it stores?

$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$



Hochreiter et al. 1997
Graves 2013

The gates dilate and contracts time, similar to a low-pass filter in typical signal processing.

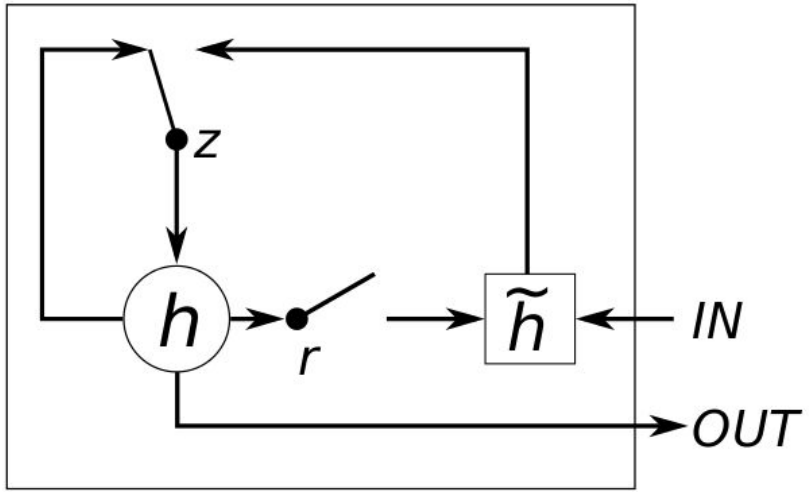
Chung et al. 2015

$$z = \sigma(W_z x_t + U_z h_{t-1})$$

$$r = \sigma(W_r x_t + U_r h_{t-1})$$

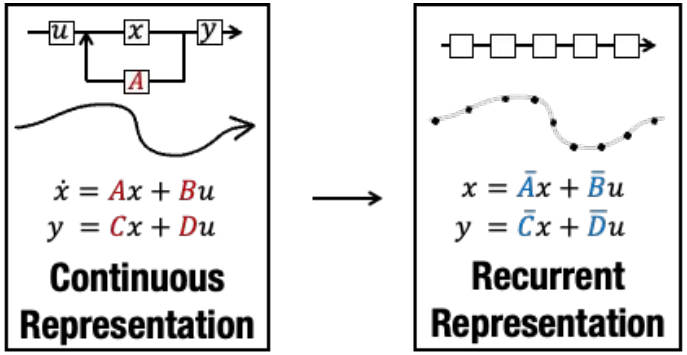
$$\tilde{h} = tanh(W_h x_t + U_h(r \circ h_{t-1}))$$

$$h_t = (1 - z) \circ h_{t-1} + z \circ \tilde{h}$$



(b) Gated Recurrent Unit

A fresh view on Recurrent Models: State Space Models



Gu et al. 2021, Efficient Modelling Long Sequences with Structured State Space Models

$$x'(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

1. Discretize $\overline{A} = I + \Delta A$
2. Recurrent "hidden state" $x_k = \overline{A}x_{k-1} + \overline{B}u_k$
3. Out projection $y_k = \overline{C}x_{k-1} + \overline{D}u_k$

Linear Recurrent Unit (LRU)

$$x_k = \mathrm{diag}(\lambda)x_{k-1} + \gamma \odot Bu_k$$

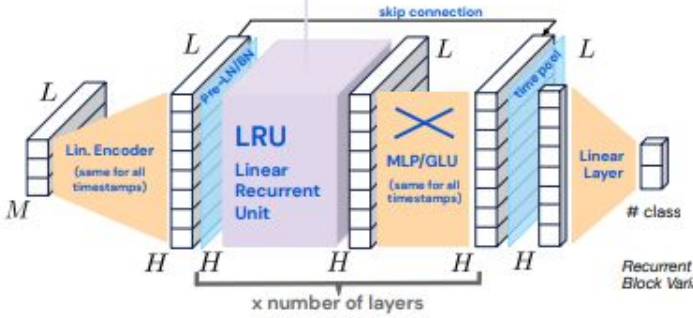
magnitude / phase

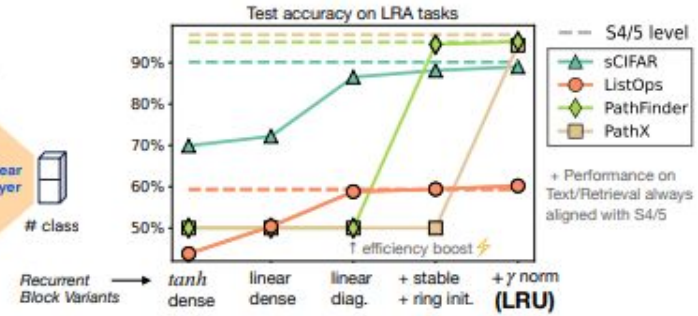
$$\lambda_j = \exp(-\exp(\nu_j^{\log}) + i\exp(\theta_j^{\log}))$$

Stable exponential parametrization

$$\gamma_j \leftarrow \left(1 - |\lambda_j|^2\right)^{1/2}$$

Normalization





Essentially a **Linear** RNN

Orvieto et al 2023, Resurrecting Recurrent Neural Networks for Long Sequences

A fresh view on Recurrent Models: State Space Models

Essentially a **Linear** RNN

Linear Recurrent Unit (LRU)

$$x_k = \mathrm{diag}(\lambda) x_{k-1} + \gamma \odot B u_k$$

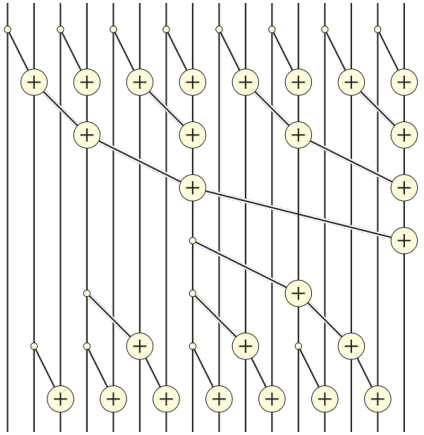
magnitude | phase

$$\lambda_j = \exp(-\exp(\nu_j^{\log}) + i \exp(\theta_j^{\log}))$$

Stable exponential parametrization

$$\gamma_j \leftarrow \left(1 - |\lambda_j|^2\right)^{1/2}$$

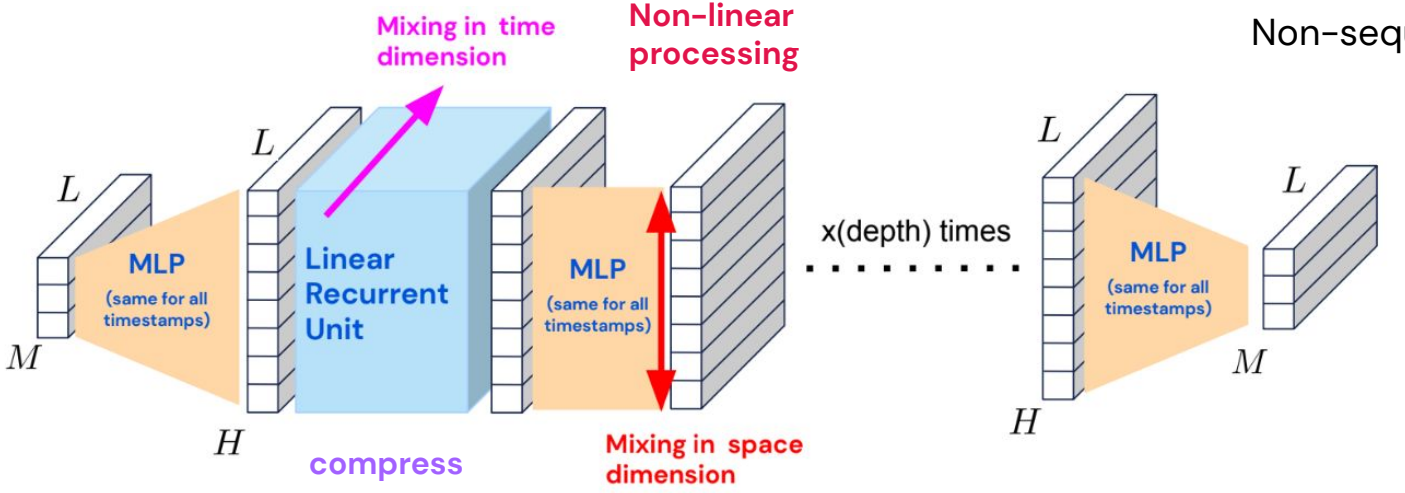
Normalization



Non-sequential evaluation

Why Linear is good?

- **Efficient computation**
- **Exact control of the system**
- **Lack of expressivity**



State Space Models: Why should we care?

	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
RWKV-4	7.4	4.38	61.20%	67.18%	76.06%	73.44%	65.51%	61.01%	37.46%	67.80%	31.22%	40.20%	88.80%	18.30%	83.68%	85.00%
Pythia	6.9	4.30	60.44%	67.98%	74.54%	72.96%	63.92%	61.01%	35.07%	66.79%	28.59%	38.00%	90.00%	15.42%	86.44%	85.00%
GPT-J	6.1	4.10	61.34%	68.31%	75.41%	74.02%	66.25%	64.09%	36.60%	66.92%	28.67%	38.20%	91.50%	16.74%	87.71%	83.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
RWKV-4 ctx8192	14.2	3.86	63.71%	70.83%	77.48%	76.06%	70.65%	63.85%	38.99%	70.24%	32.64%	41.80%	90.40%	24.58%	85.67%	85.00%
GPT-level	14.2	3.81	63.11%	70.94%	76.49%	74.97%	68.72%	65.14%	37.99%	70.77%	31.03%	39.27%	92.20%	22.37%	87.89%	82.66%
Pythia	11.8	3.89	62.38%	70.44%	75.90%	74.40%	67.38%	64.72%	36.77%	69.82%	30.74%	38.80%	91.80%	20.57%	87.58%	82.00%
GPT-NeoX	20.6	3.64	64.58%	71.94%	77.69%	76.11%	71.42%	65.98%	40.44%	72.69%	31.62%	40.20%	93.00%	25.99%	88.52%	84.00%

[Peng et al. 2023, RWKV: Reinventing RNNs for the Transformer Era]

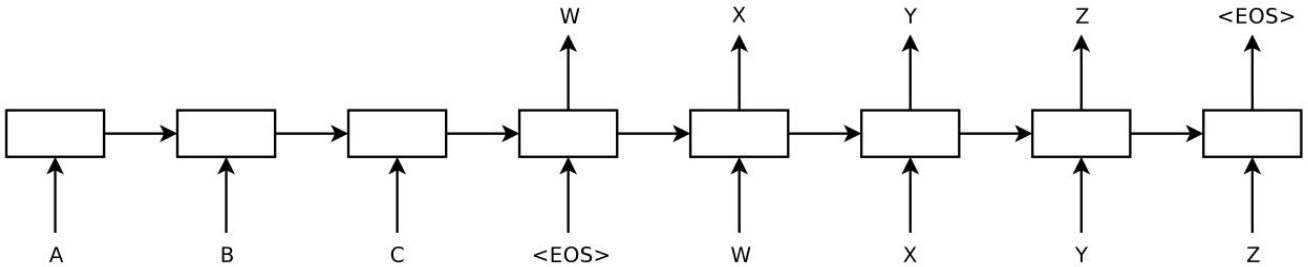
Model	Time	Space
Transformer	$O(T^2 d)$	$O(T^2 + Td)$
Reformer	$O(T \log T d)$	$O(T \log T + Td)$
Linear Transformers	$O(Td^2)$	$O(Td + d^2)$
Performer	$O(Td^2 \log d)$	$O(Td \log d + d^2 \log d)$
AFT-full	$O(T^2 d)$	$O(Td)$
MEGA	$O(cTd)$	$O(cTd)$
RWKV (ours)	$O(\mathbf{Td})$	$O(\mathbf{d})$

- RWKV is such a linear RNN
- Performs comparable to transformers (in the LLM world) at much lower computational costs
- Linear attention is a particular linear RNN as well
- Can deal with longer contexts

Towards the emergence of transformer

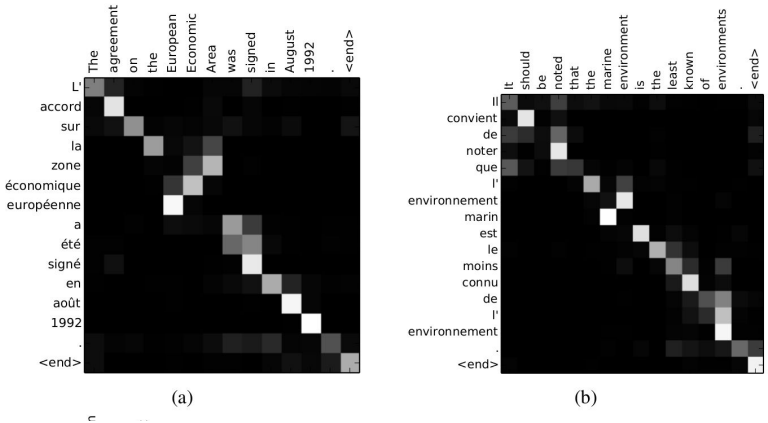
End-to-end translation systems (LSTM based):

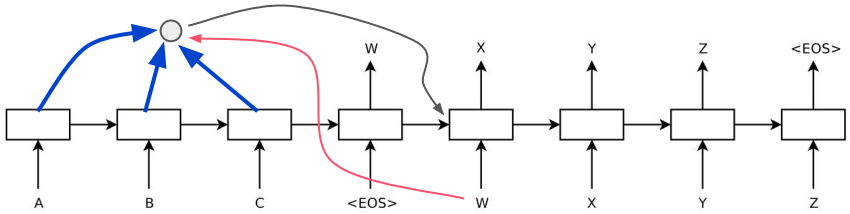
Sutskever et al. 2014

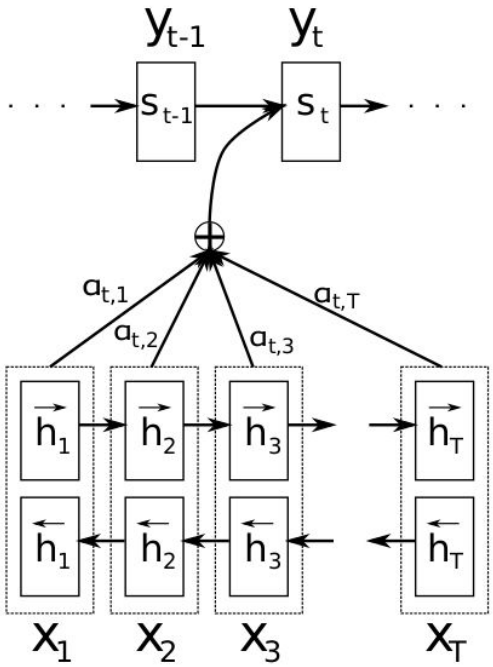


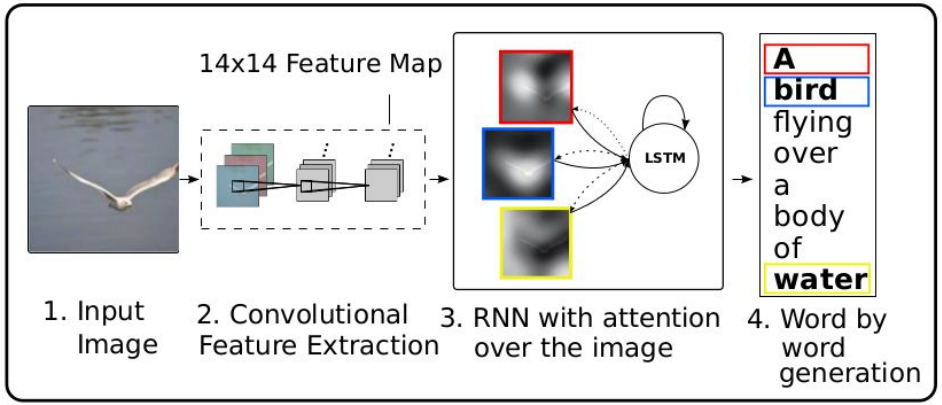
Attention mechanism

Bahdanau et al. 2015







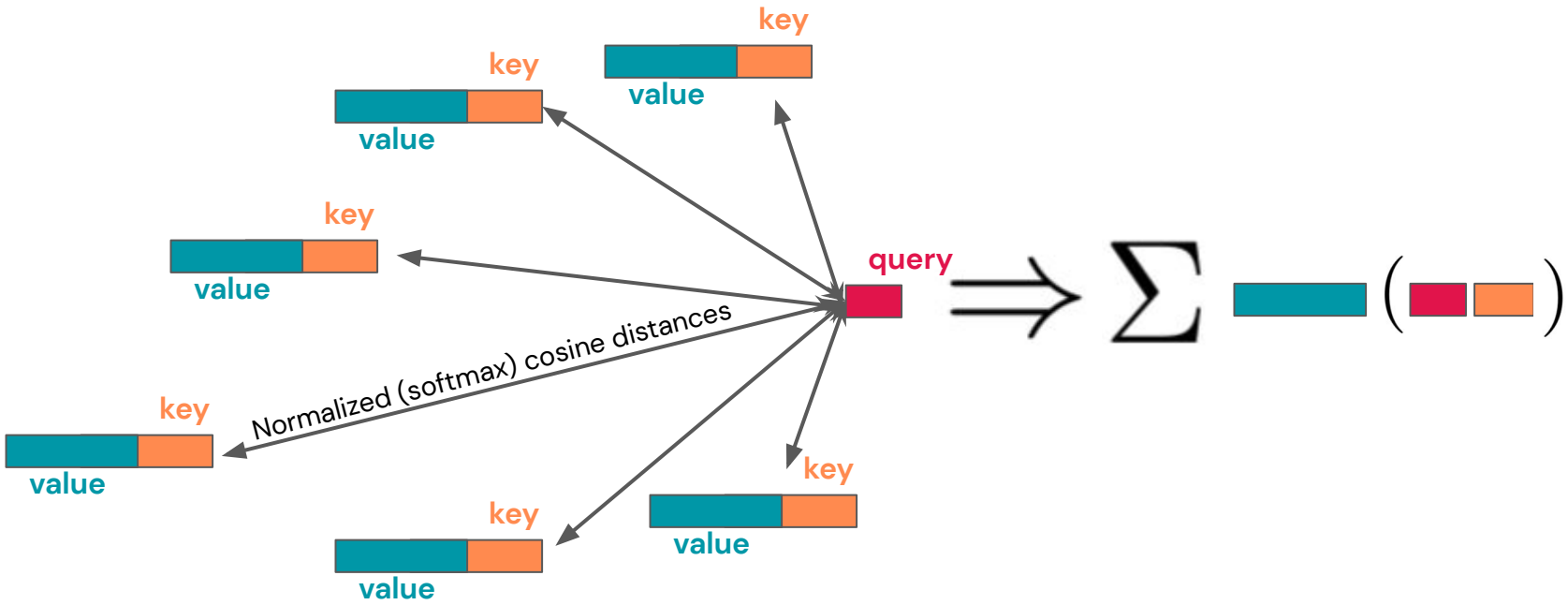


- Baidu/UCLA: Explain Images with Multimodal Recurrent Neural Networks
- Toronto: Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models
- Berkeley: Long-term Recurrent Convolutional Networks for Visual Recognition and Description
- Google: Show and Tell: A Neural Image Caption Generator
- Stanford: Deep Visual-Semantic Alignments for Generating Image Description
- UML/UT: Translating Videos to Natural Language Using Deep Recurrent Neural Networks
- Microsoft/CMU: Learning a Recurrent Visual Representation for Image Caption Generation
- Microsoft: From Captions to Visual Concepts and Back

Transformer

https://arxiv.org/abs/1706.03762

- Treat inputs as a set
- Everytime you evaluate an item, attend to all other elements

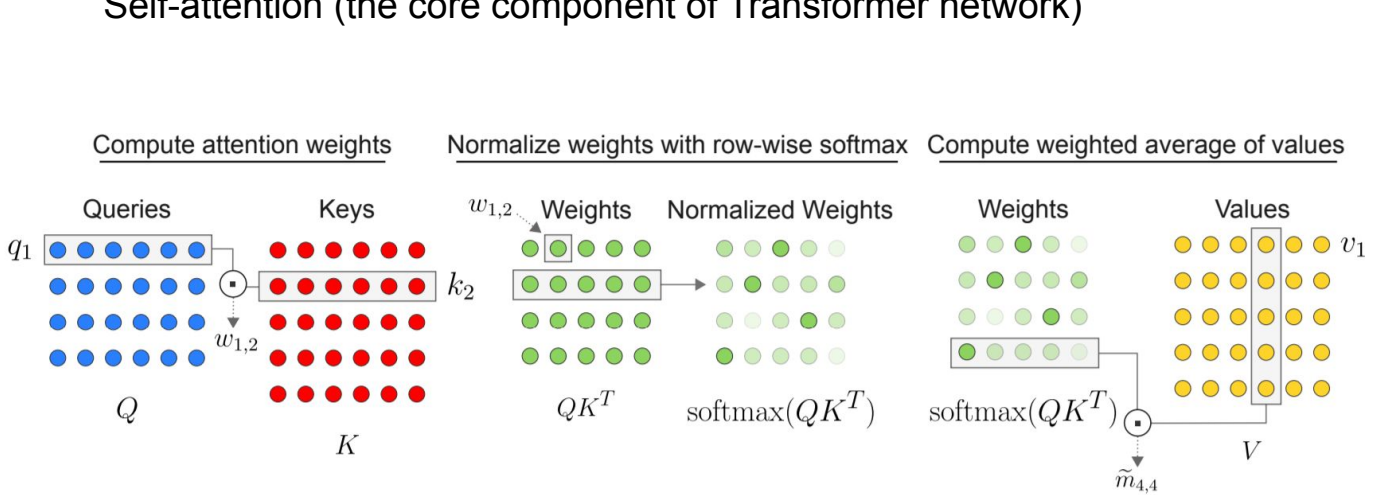


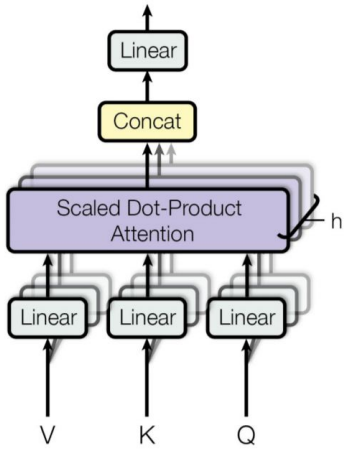
Transformer

- What about time ? - Attach positional encoding
- Why use multi-head attention?
- Why normalize using softmax?

Self-attention (the core component of Transformer network)

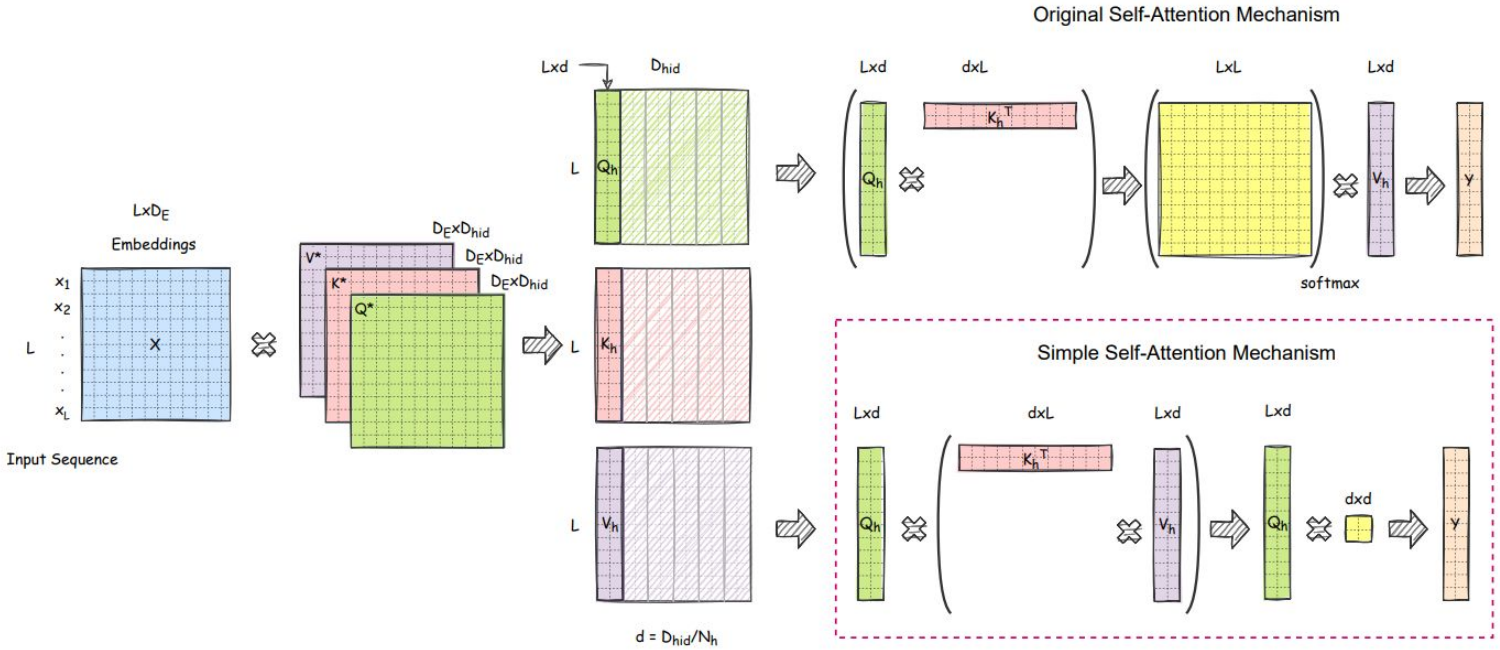
https://arxiv.org/abs/1706.03762



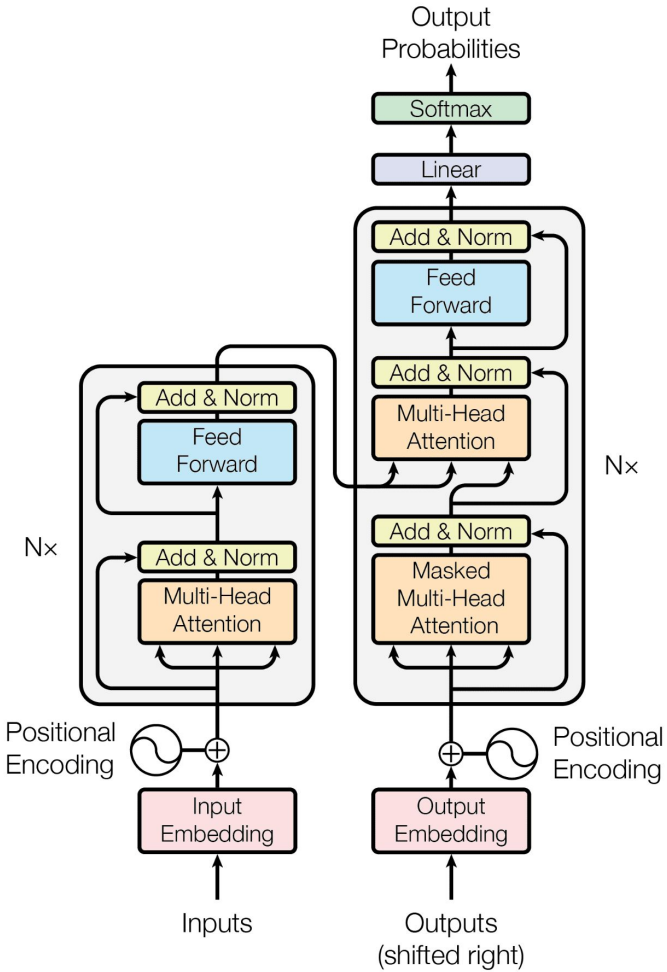


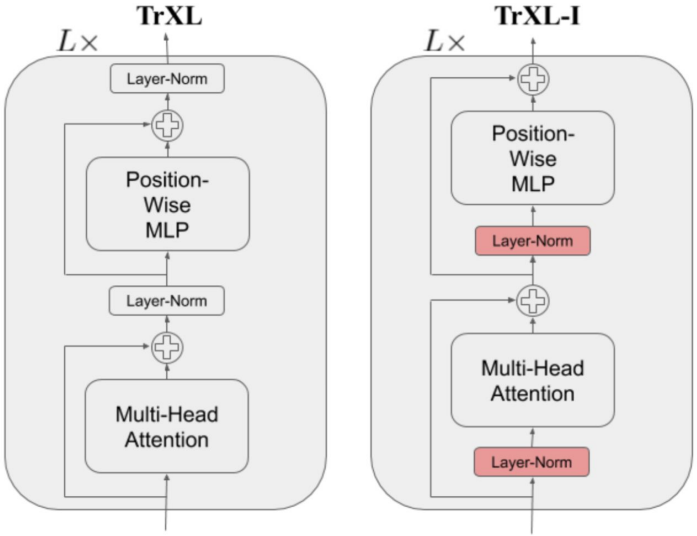
https://arxiv.org/pdf/1806.01822.pdf

Linear attention (and softmax)



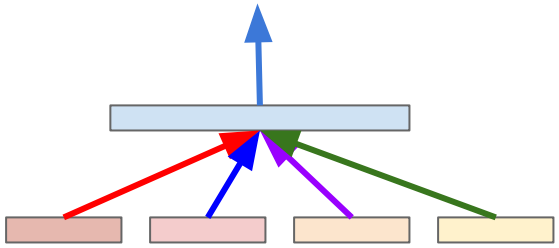
[Yorsh et al. 2021 SimpleTRON: Simple Transformer with O(N) complexity]



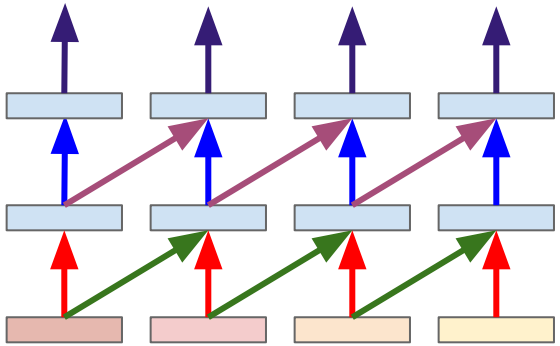


Parisotto et al. 2019, Stabilizing Transformers for Reinforcement Learning

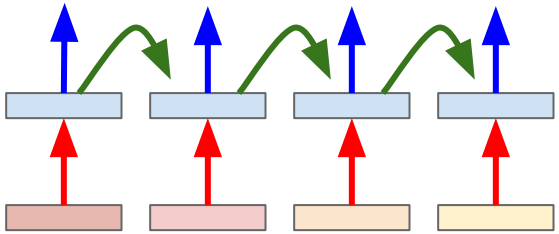
- Rethinking computation along blocks vs layers
- What type of memory does a transformer have compared to an SSM model ?



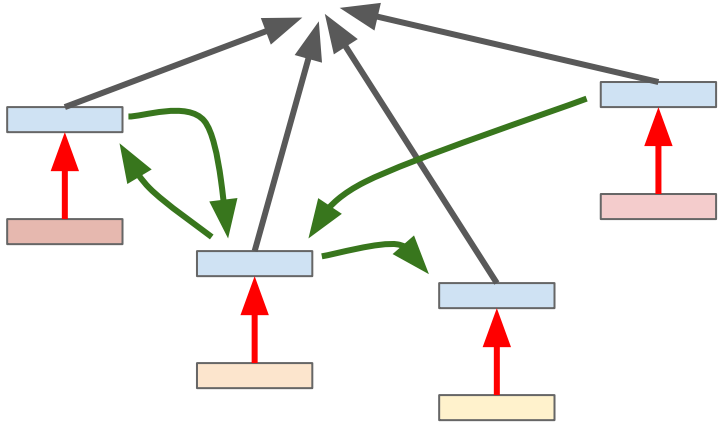
MLP



Convolutional Network



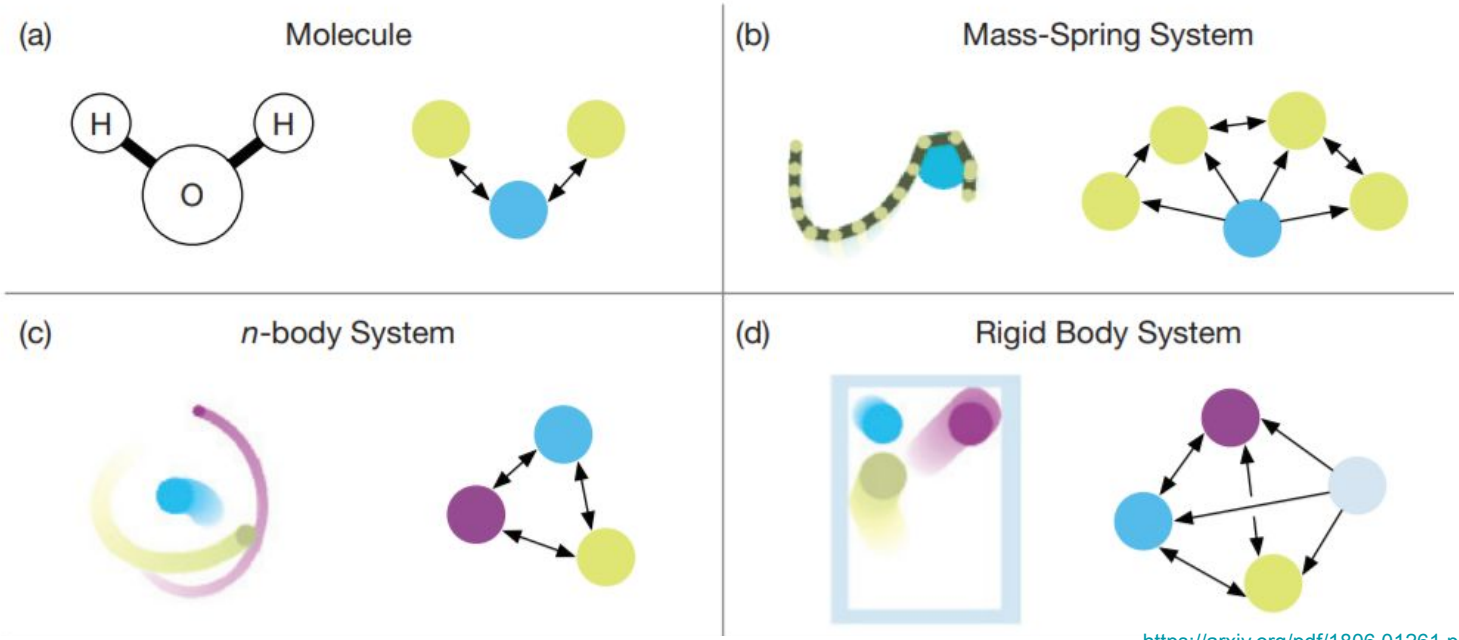
Recurrent Network



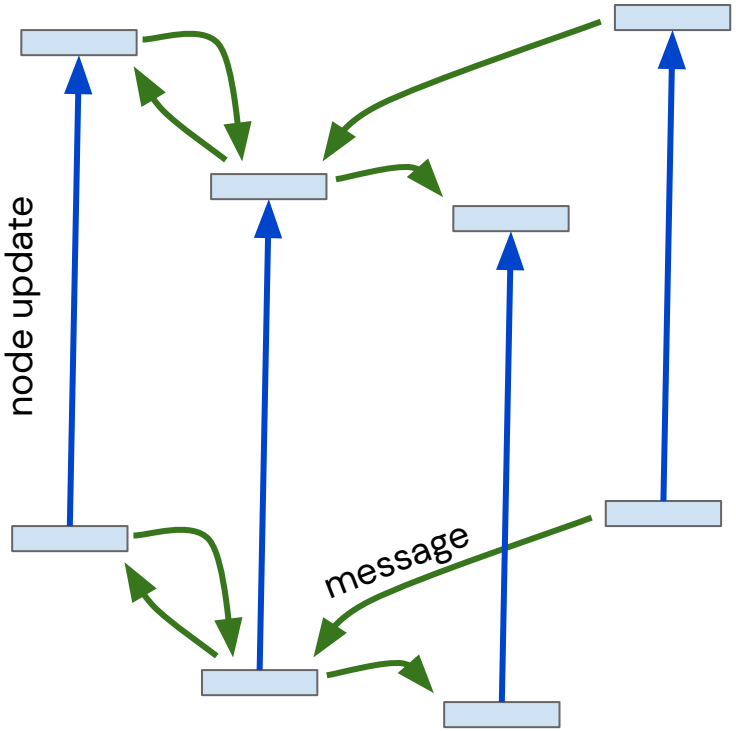
Graph Neural Networks

GraphNetworks

"... infinite use of finite means"



https://arxiv.org/pdf/1806.01261.pdf



$$\forall i, u_{i_t} = \phi\left(u_{i_{t-1}}, \bigoplus_{j \in \mathcal{N}(i)} m_{ij_{t-1}}\right)$$

$$\forall i, j \in \mathcal{N}(i), m_{ij_t} = \psi\left(u_{i_t}, u_{j_t}\right)$$

- Similar to convnets exploits locality (works over neighbourhoods) and reuse
- Similar to transformers, aggregation over a set
- Transformers can be as a special case of GNN
- This structure defines a *geometry* and GNN allows to encode invariances and knowledge about the computation that one wants to model

Conclusions

- Many fundamental topics I want to touch but couldn't:
  - Unsupervised / self-supervised learning & other paradigms
  - Meta-learning / transfer learning / hyper-parameter optimization
  - Non-paramtric learning (GPs, etc.)
  - Sparsity / compression
  - Causality / Ethics & Fairness
  - Uncertainty & probabilistic view of ML

What I hope I managed to say:

- Inductive biases: anything we do affects the solution we find
- Most choices we made have multiple side effects
- Deep Learning is far from being understood or being just another tool
- Some intuition of how many apparently different topics fit together


THANK YOU!