

Deep learning: tips and tricks for getting it right



Istituto Nazionale di Fisica Nucleare



Francesco Vaselli
tCSC Machine Learning 2024, Split, Croatia

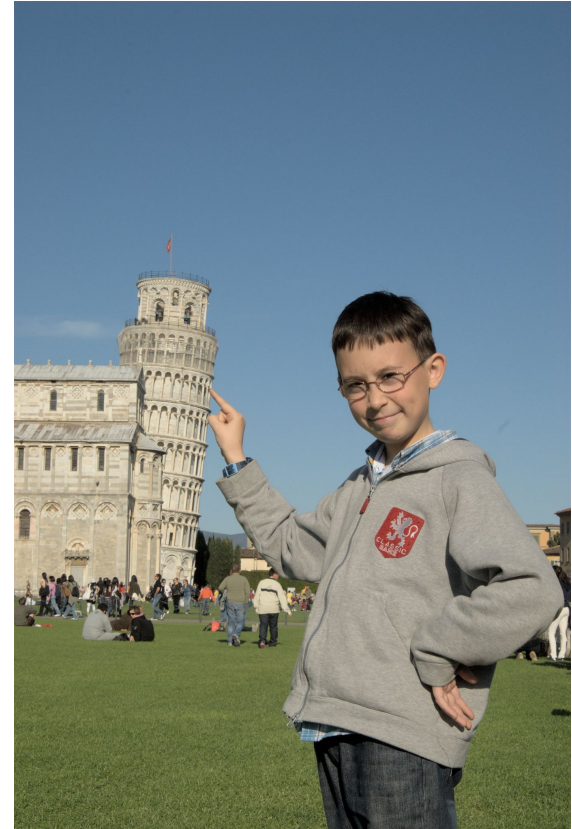
What Pisa can teach us about Deep Learning

People tried to make a very tall tower

Turns out it's not enough to stack one floor on the other if the foundations are not solid

It's the same thing with Deep Learning:

Simply stacking many neurons is not enough!



Outline

A few things you may already know, motivated and trying to understand why we do them that way

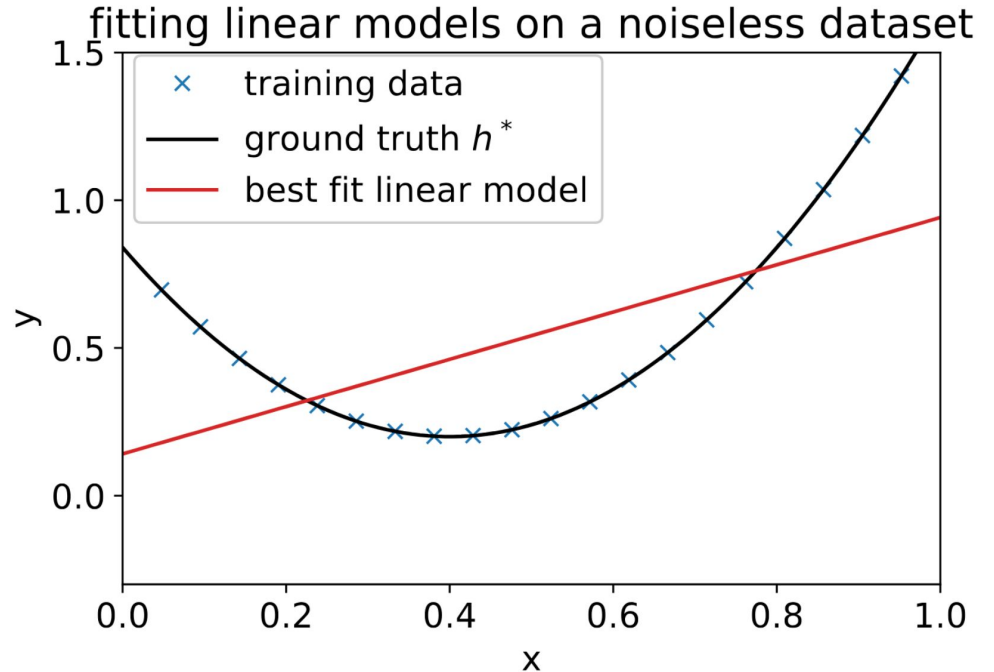
- Bias-variance tradeoff
- Basic ML recipe
- Training and Bias
- Testing and Variance
- Different architectures

Bias-variance tradeoff

Encountered in most statistical models

We have a bias when the model is too simple and it is not capturing the true relationship between x and $y = h^*(x)$

e.g. a linear model cannot reproduce a quadratic one

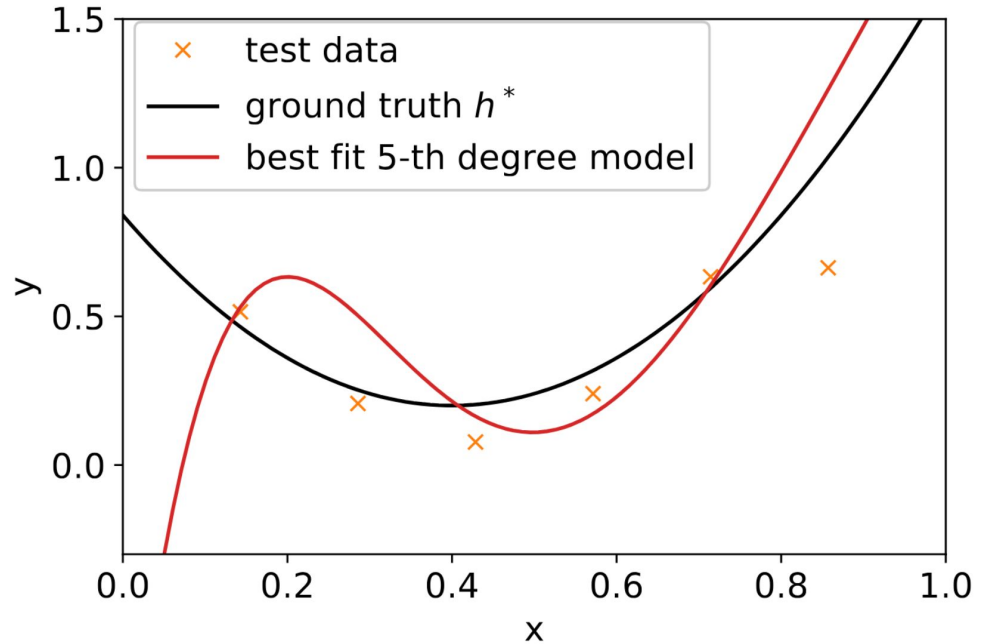


Bias-variance tradeoff

Encountered in most statistical models

When the model is too expressive, we risk fitting patterns of our small, finite training sample -- variance part of the error

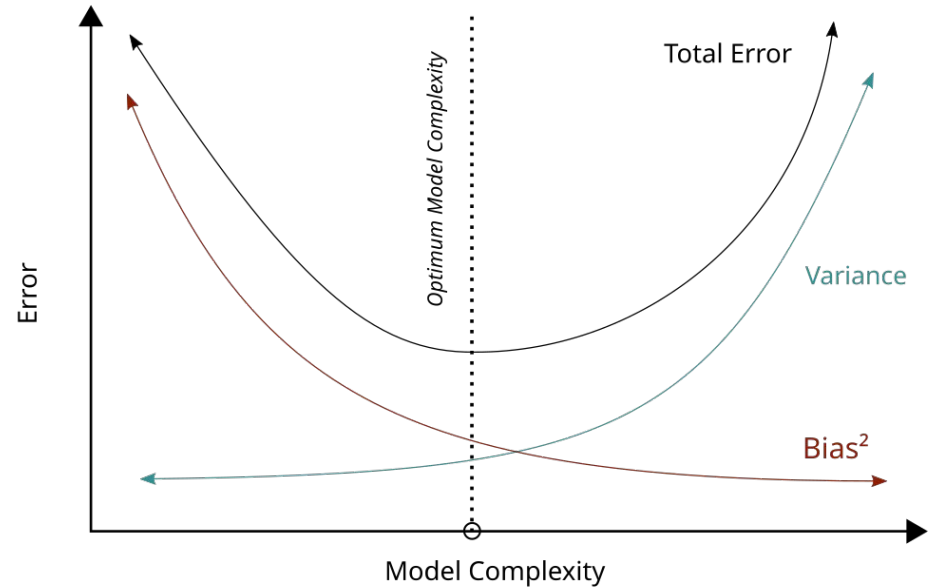
e.g. a 5th-degree polynomial will overfit a small sample from a quadratic model



Bias-variance tradeoff defines an optimal complexity

We must strike the right balance between a simple, highly biased model and a complex, variance sensitive one

Deep learning requires careful tuning and experimentation!



How do we address this? Basic ML recipe

Training dataset

How well I am modelling the
process producing the
training data

Checks for bias

How do we address this? Basic ML recipe

Training dataset

How well I am modelling the process producing the training data

Checks for bias

Test dataset

How well we generalize to previously unseen instances of the data

Checks for variance

How do we address this? Basic ML recipe

Training dataset

How well I am modelling the process producing the training data

Checks for bias

Test dataset

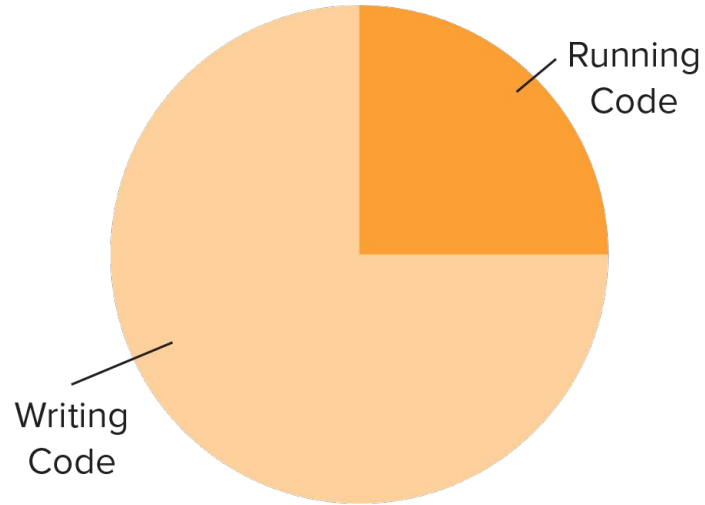
How well we generalize to previously unseen instances of the data

Checks for variance

Deploy!

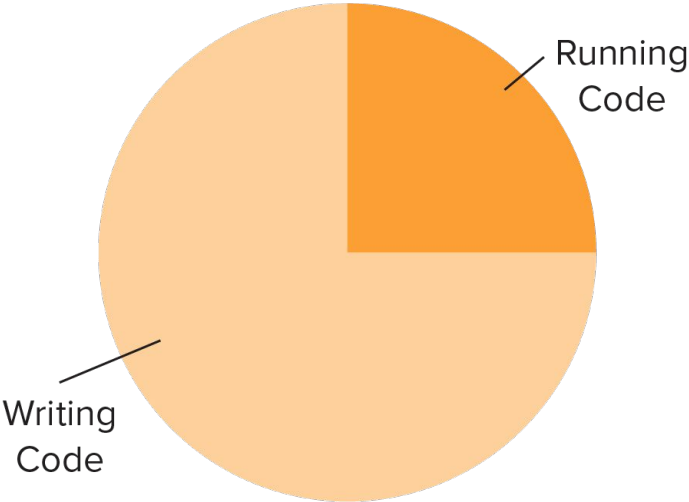
Data matters!!

How People Think ML Works

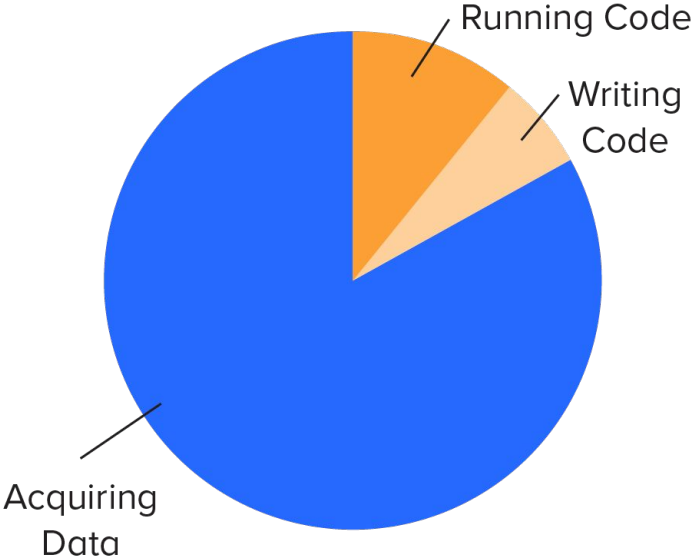


Data matters!!

How People Think ML Works



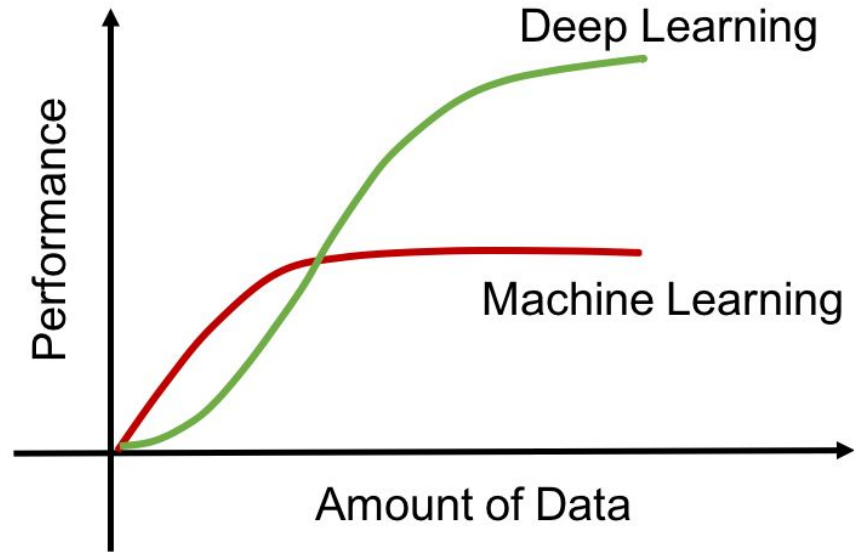
The Reality



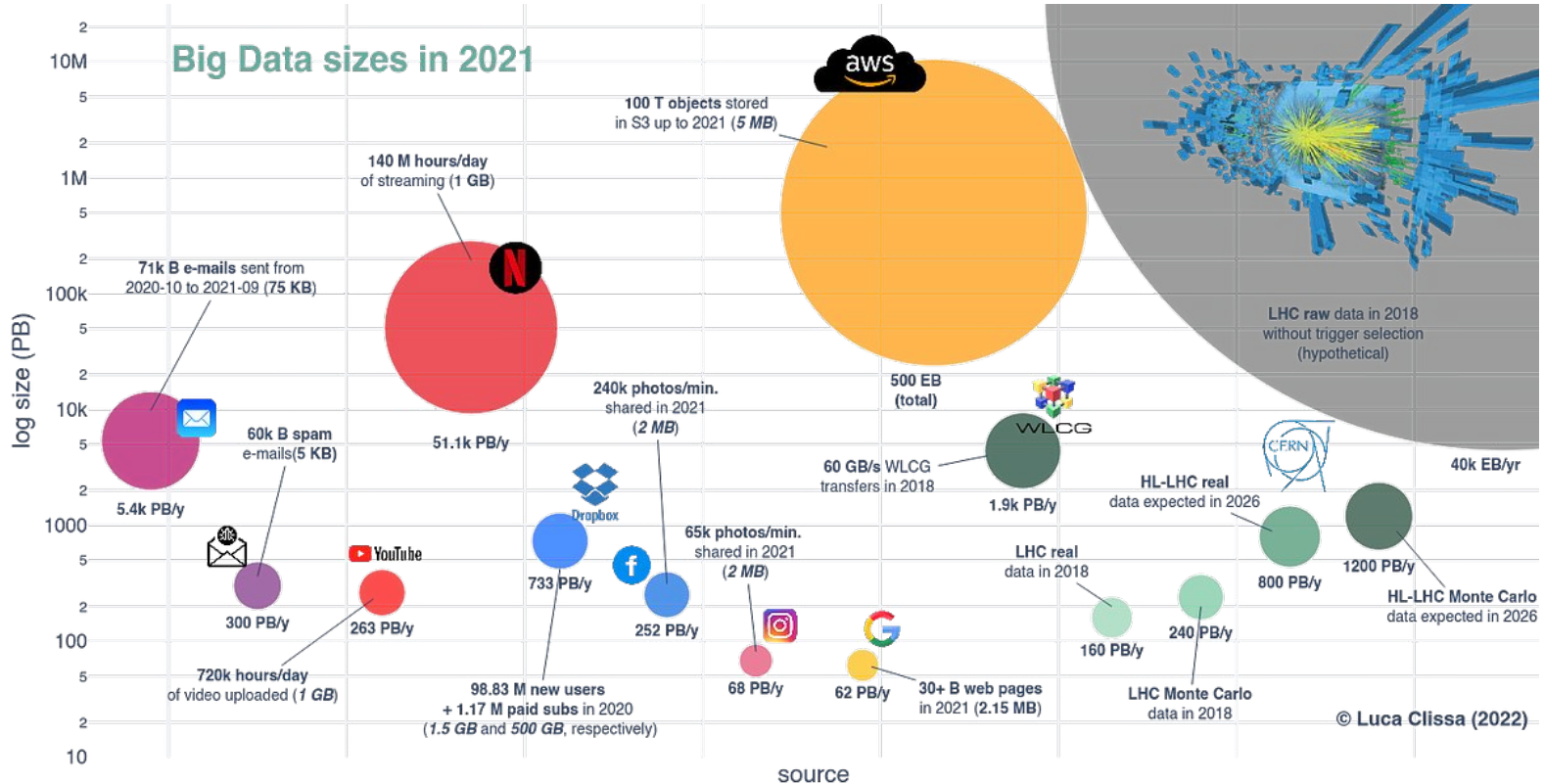
Scale drives deep learning progress!

Assuming:

1. You can fit the training set pretty well.
2. The training set performance generalizes pretty well to the test set.



Fortunately, we have an abundance of data!



We still need to be careful in how we handle them

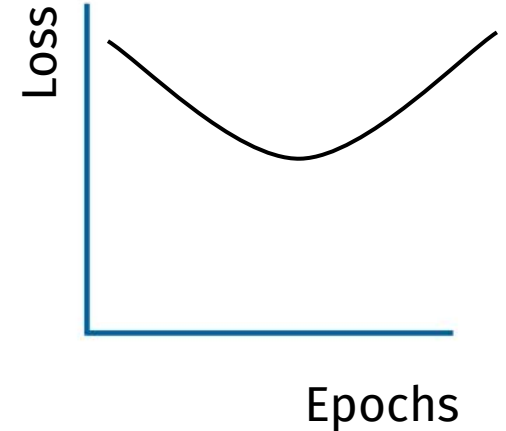
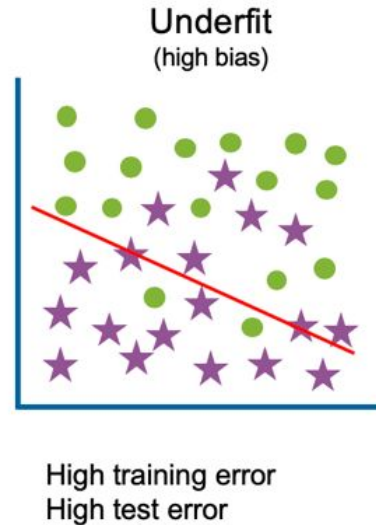
Understand your data

- Why is it relevant to your problem?
- Is some important physical information missing?
- Is the data correctly labelled?
- Is the data introducing some unwanted correlations?
- Is the data still relevant to my problem?

The art of Training

What can I do if the training performance is lacking?

It means that we are not learning the underlying statistical model



The art of Training

What can I do if the training performance is lacking?

It means that we are not learning the underlying statistical model:

- Train longer
- Bigger network (more capacity)
- Gradients spaces and learning algorithms
- Vanishing/exploding gradients

Moving around the loss space can be tricky!

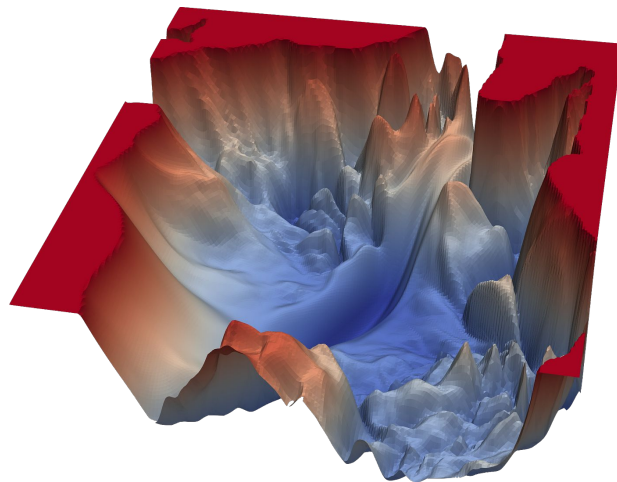
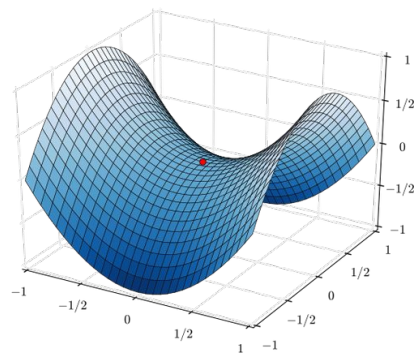
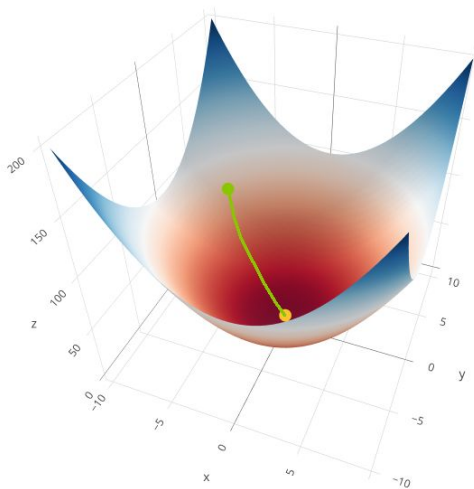


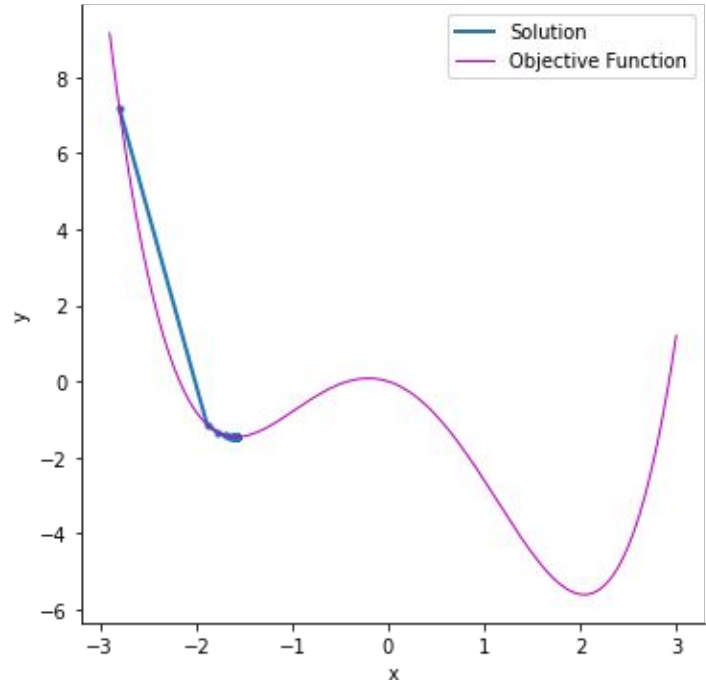
Image credit <https://www.cs.umd.edu/~tomg/projects/landscapes>

Basic gradient descent is limited

The naive stochastic/batch gradient descent has many limitations in how it navigates complex loss spaces:

- sensitive to noise in current sample/batch
- easy to get trapped in local minimum

$$\theta_i = \theta_{i-1} - \gamma * g_i$$



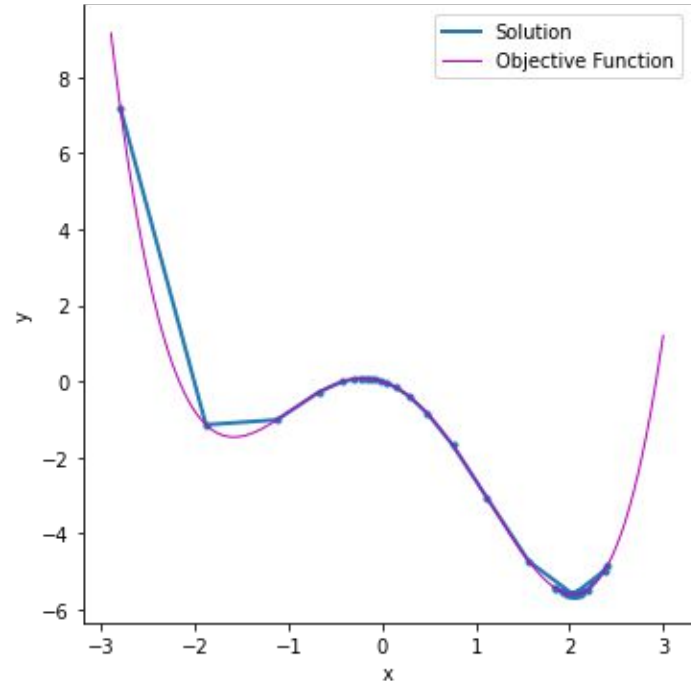
<https://optimization.cbe.cornell.edu/index.php?title=Momentum>

Momentum helps overcoming these limitations

Momentum is an extension to the algorithm that builds inertia in a search direction to overcome local minima and oscillation of noisy gradients.

$$b_i = \mu * b_{i-1} + g_i$$

$$\theta_i = \theta_{i-1} - \gamma * b_i$$

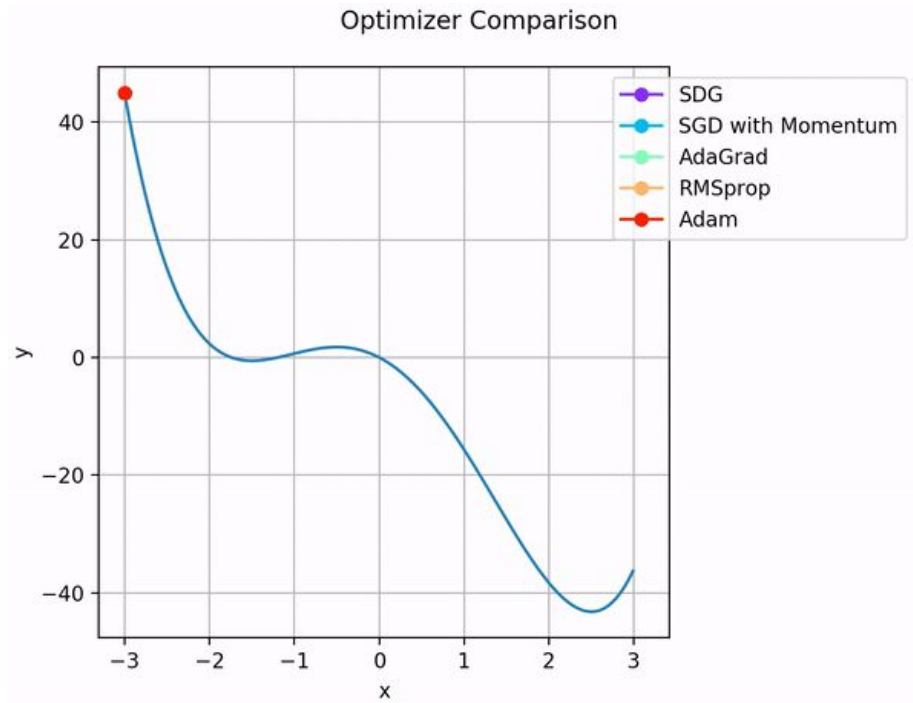


<https://optimization.cbe.cornell.edu/index.php?title=Momentum>

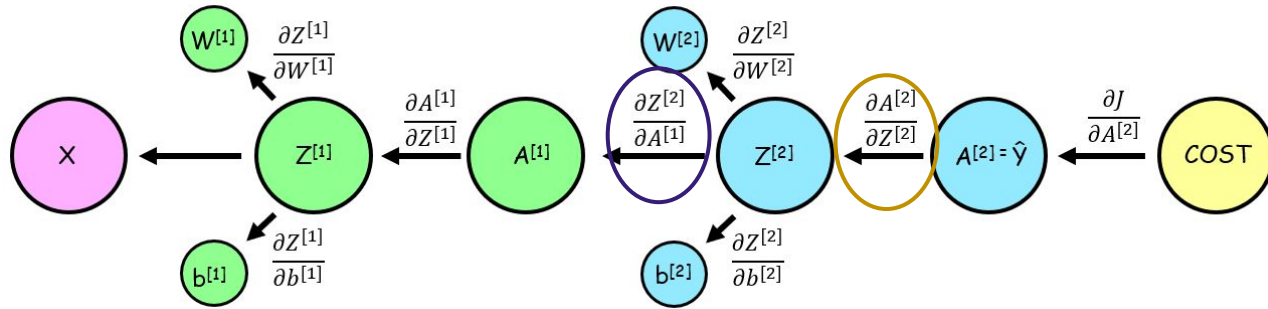
State-of-the-art: Adaptive Moment Estimation (Adam)

Adam is an adaptive learning rate algorithm:

- It uses momentum
- It dynamically adjusts the learning rate for each individual parameter within a model, rather than using a single global learning rate



Vanishing/Exploding gradients



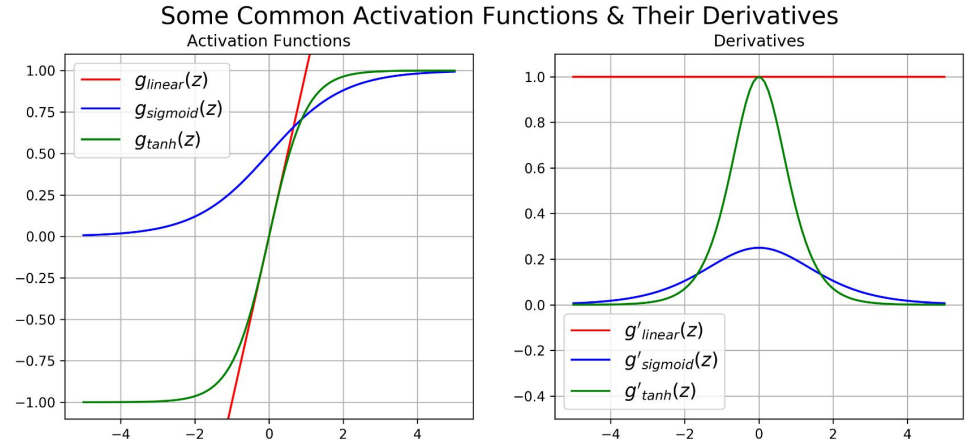
Remember that the gradients of the loss depend on:

- the derivative of the activation functions
- the derivative of the outputs (i.e. the weights)

The gradients can vanish for small derivatives

Some choices of activation functions have small derivatives

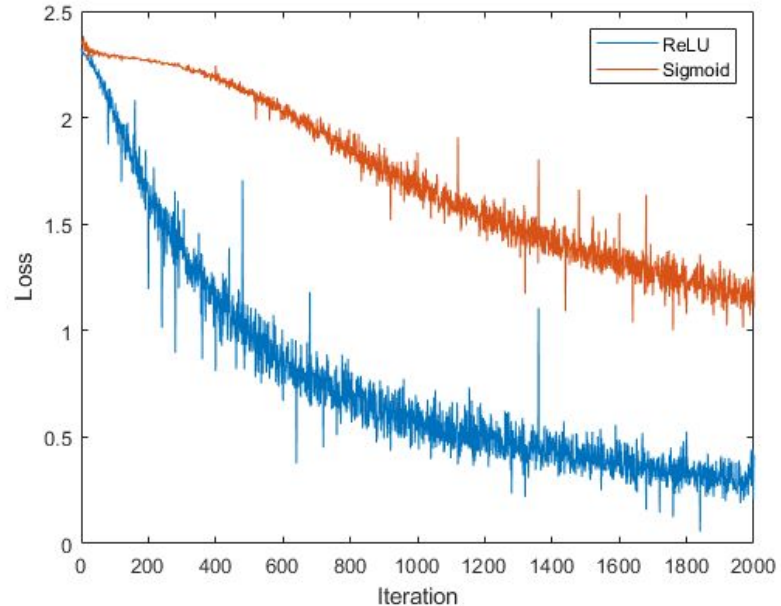
This can lead to chain multiplication of small numbers!



The gradients can vanish for small derivatives

This can lead to chain multiplication of small numbers, making the gradients of the initial layers effectively 0 and preventing learning

Mitigated through initialization and change of activation functions



The gradients can explode for large weights

The weights can have a norm $\gg 1$

Multiplication of large numbers will result in exploding gradients for first weights

Can be mitigated with regularization and clipping of weights

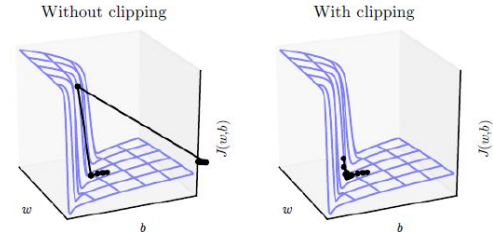
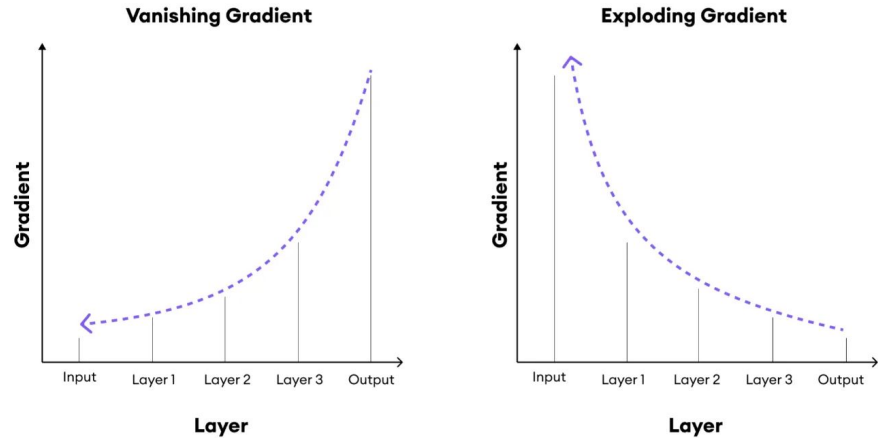


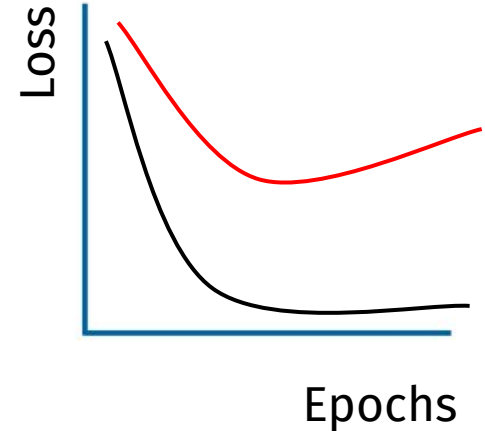
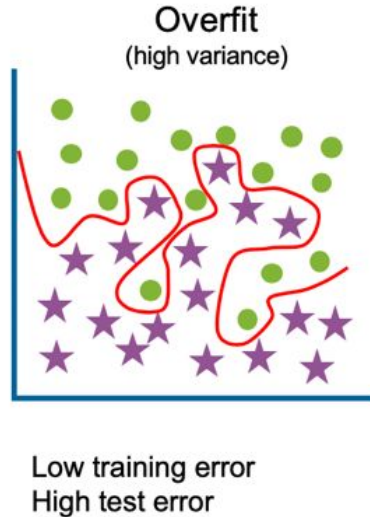
Image 1 credit
<https://www.superannotate.com/blog/activation-functions-in-neural-networks>

Image 2 credit
Deep Learning, Goodfellow et al

The art of Testing or *Regularization*

What if the testing performance is poor after training?

It means that we are modelling a specific variance of our train dataset



The art of Testing or *Regularization*

What if the testing performance is poor after training?

It means that we are modelling a specific variance of our train dataset:

- Use more training data
- Regularization techniques
- Preprocessing of data

Regularization I: Weight decay

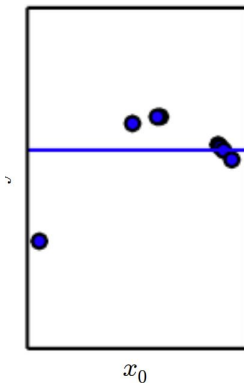
$$L_{new}(w) = L_{original}(w) + \lambda w^T w$$

We add a term in the loss function proportional to the L2-norm of the weights

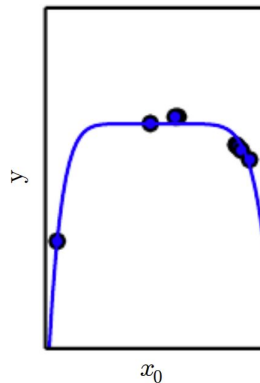
Penalizes large weights

But why do smaller weights correspond to simpler models?

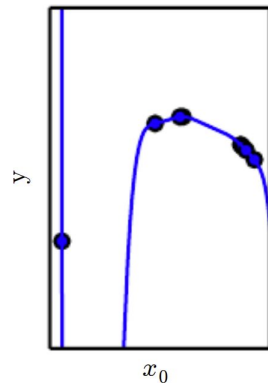
Underfitting
(Excessive λ)



Appropriate weight decay
(Medium λ)



Overfitting
($\lambda \rightarrow 0$)

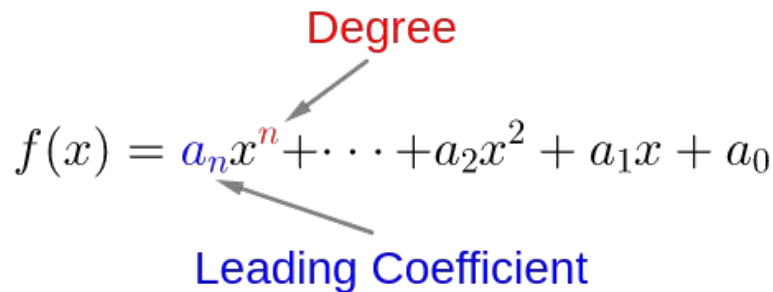


Regularization I: Weight decay

Limits model complexity and non-linearity: think of an N-layer network as a Nth degree polynomial for one input feature when the others are fixed.

$$y = f_n(W_n * \dots f_1(W * x + b) \dots)$$

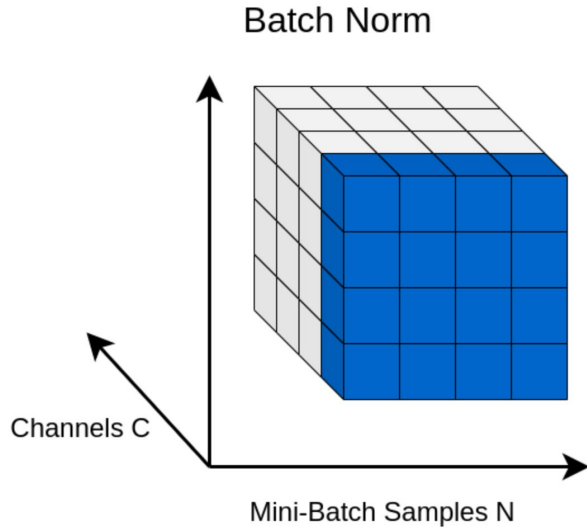
We are reducing the coefficients of such a polynomial, hence making it less expressive!



The diagram shows the polynomial equation $f(x) = a_n x^n + \dots + a_2 x^2 + a_1 x + a_0$. A red arrow points from the word "Degree" to the exponent n in the term $a_n x^n$. A blue arrow points from the words "Leading Coefficient" to the coefficient a_n in the same term.

$$f(x) = a_n x^n + \dots + a_2 x^2 + a_1 x + a_0$$

Regularization II: Batch Normalization



Increases robustness by subtraction of “batch-random” mean and variance

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

Batch Normalization may help in the following case

Train:

Dog, $y=1$



Not Dog, $y=0$



Batch Normalization may help in the following case

Train:

Dog, $y=1$



Not Dog, $y=0$

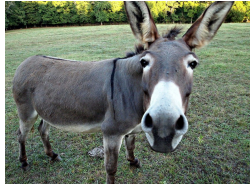


Test:

Dog, $y=1$



Not Dog, $y=0$



Batch Normalization may reduce covariance shift

Covariance shift refers to changes in the data distribution between training and testing, affecting model performance.

Batch normalization helps by normalizing input features across mini-batches, reducing internal covariate shift and stabilizing learning, thus improving generalization across varying distributions.

Train:



Test:



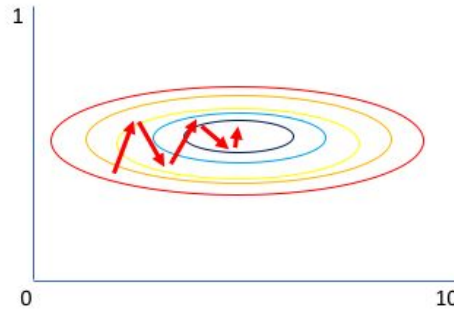
BatchNorm??



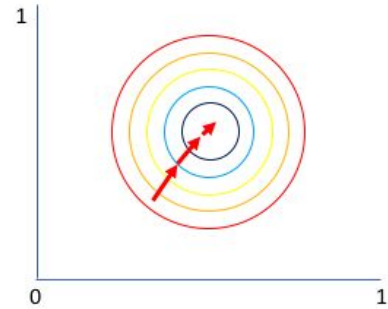
(Data) Regularization III: Normalization of Inputs

It's standard practice to normalize the input of the network to have mean 0 and variance 1

This helps to make the gradients space more regular and speed up training



Gradient of larger parameter dominates the update

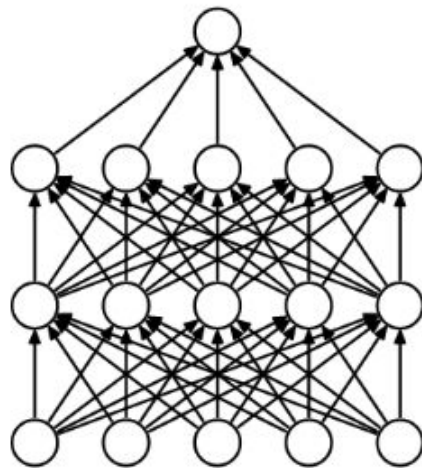


Both parameters can be updated in equal proportions

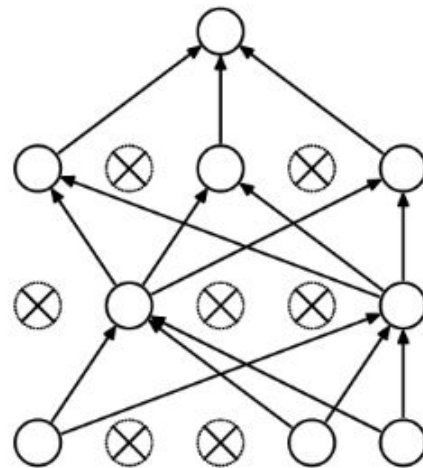
Regularization IV: Dropout

We don't want to rely on single input features, so we spread out the information through many “sub-networks”

At inference time, all of the sub-networks are activated and contribute to the output (“wisdom of the crowd”)



(a) Standard Neural Net



(b) After applying dropout.

Data preprocessing can help with generalization

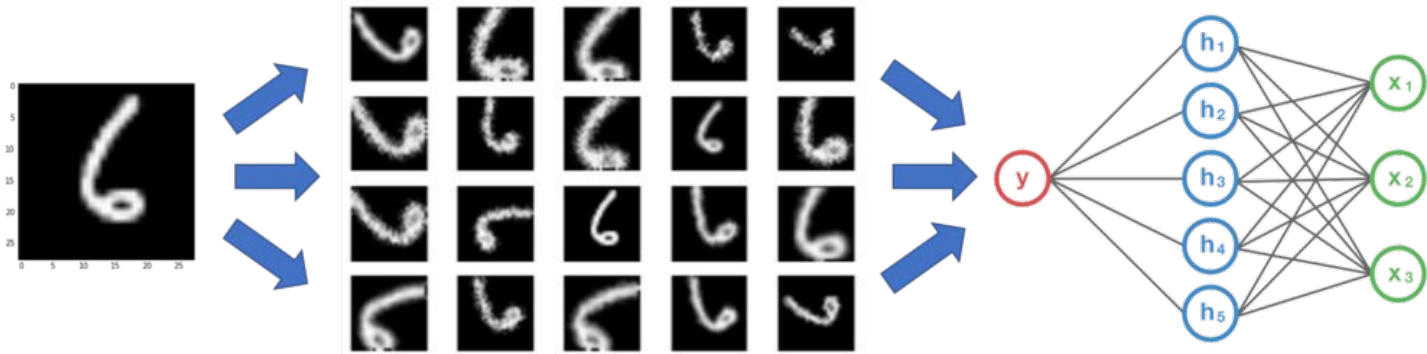
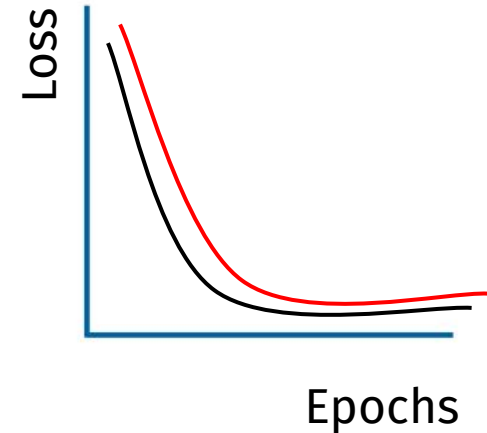
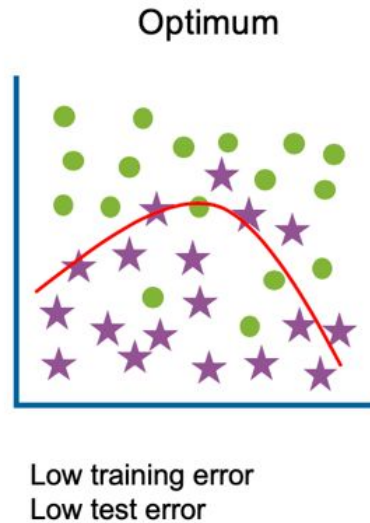


Image credits:
<https://learnai1.home.blog/2020/08/15/data-preprocessing-for-neural-networks/>

Now hopefully things work ok!

Both the train and test errors are reasonable

We can deploy our models in the wild!



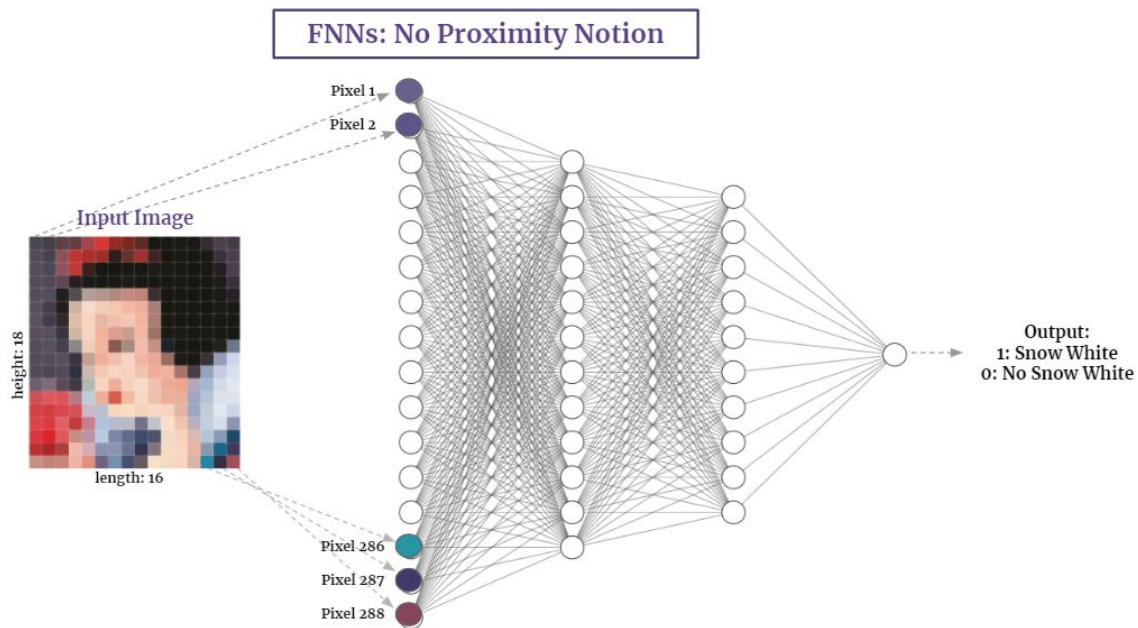
What if we were to work with images?

In a Feed-forward NN we need to *flatten* the image

For a 18*16 pixels image that's already 288 input features

For a Whatsapp image ~800*800 -> 640k inputs!!

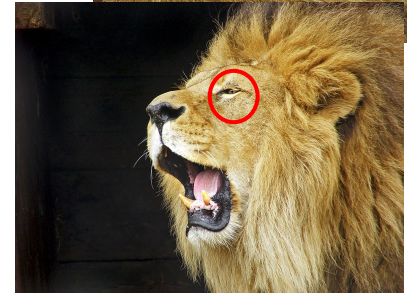
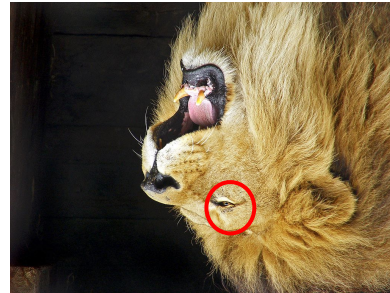
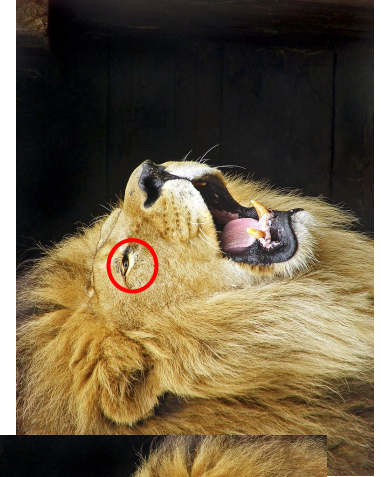
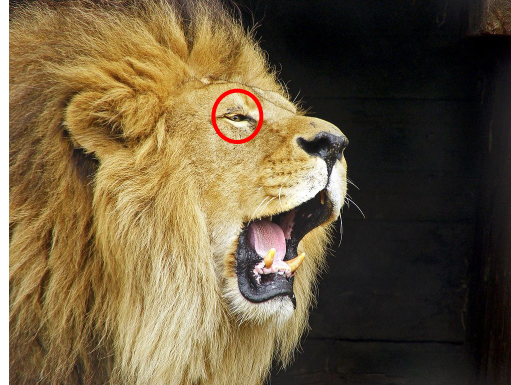
Additionally, no notion of *locality*



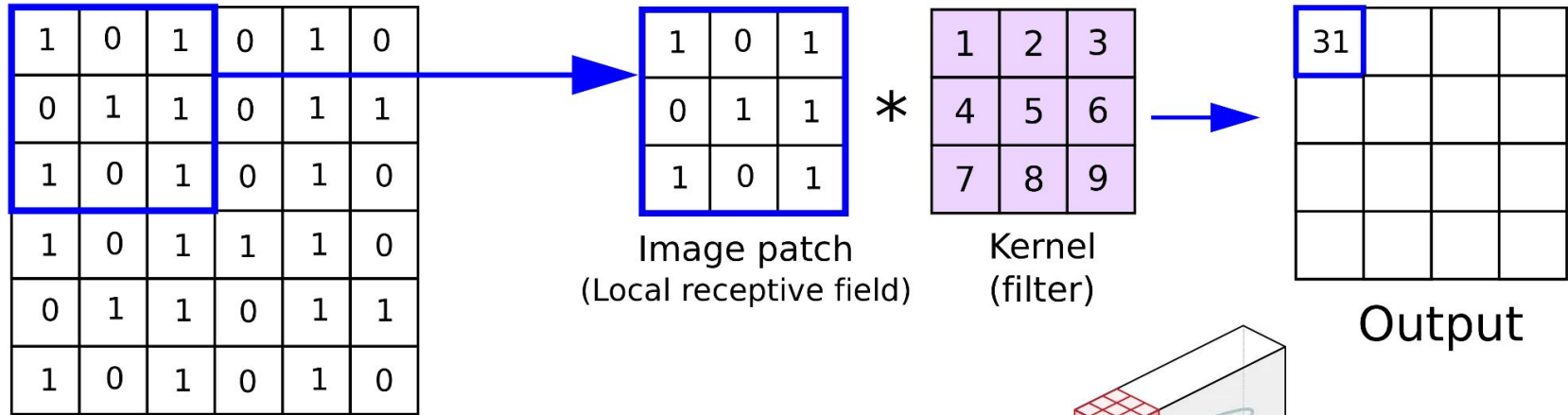
Data symmetries matter! Can we account for them?

An eye is an eye, whether or not it's top or bottom, right or left, small or big, etc.

How can we build a network that is invariant under the shift of input features?



Convolutional neural networks



Input

Image patch
(Local receptive field)

Kernel
(filter)

Output

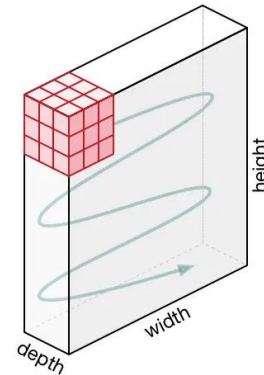


Image 1 credit
<https://wikidocs.net/164365>
Image 2 credit
<https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>

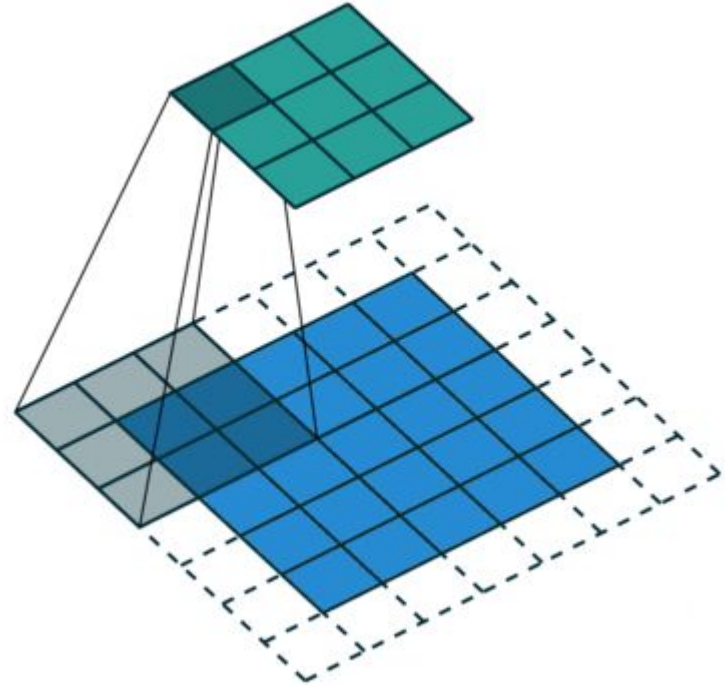
Convolutional neural networks

Essentially

Filter == Learned matrix multiplication

By going over the image, each filter can focus on the local features

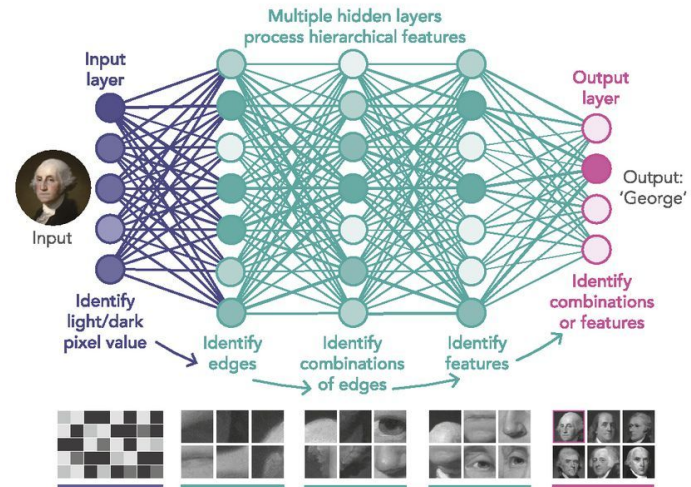
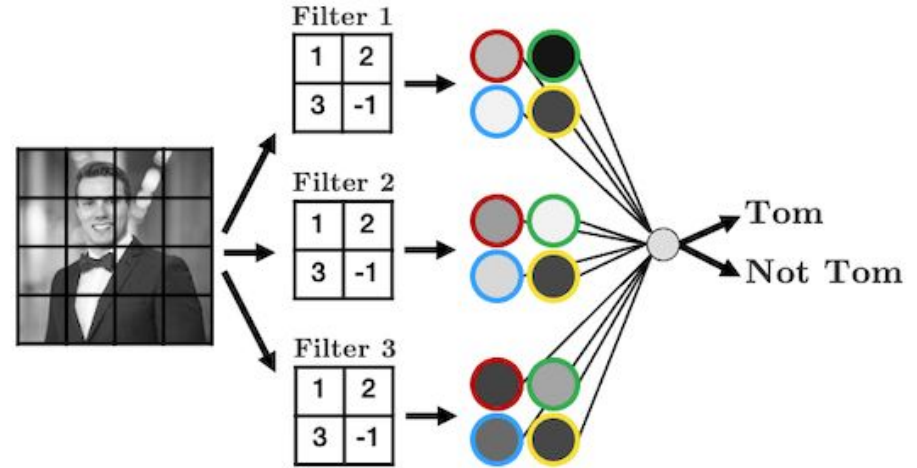
We will have filter specializing in recognizing the same element over multiple training images



Filters in action

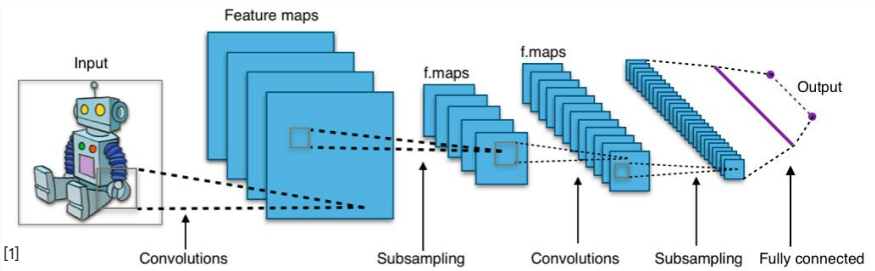
By going over the image, each filter can specialize on the extraction of single features and create an higher-level representation

The last part of the network is usually a feedforward NN which reasons on such a representation



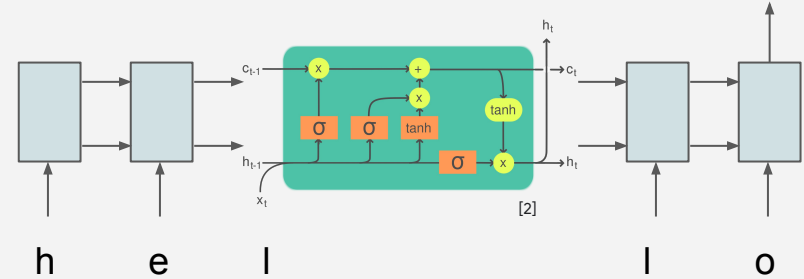
Computer Vision

Convolutional NNs (+ResNets)



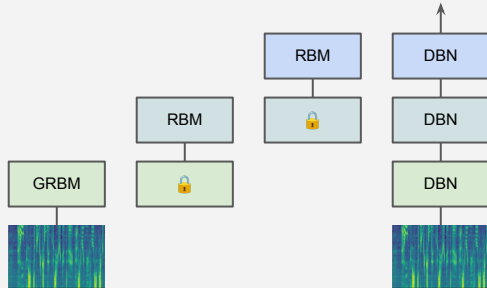
Natural Lang. Proc.

Recurrent NNs (+LSTMs)



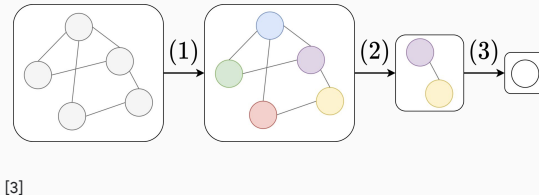
Speech

Deep Belief Nets (+non-DL)



Science

Graph NNs



RL

BC/GAIL

Algorithm 1 Generative adversarial imitation learning

- 1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
- 2: **for** $i = 0, 1, 2, \dots$ **do**
- 3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
- 4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s, a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s, a))] \quad (17)$$

- 5: Take a policy step from θ_i to θ_{i+1} , using the TRPO rule with cost function $\log(D_{w_{i+1}}(s, a))$. Specifically, take a KL-constrained natural gradient step with

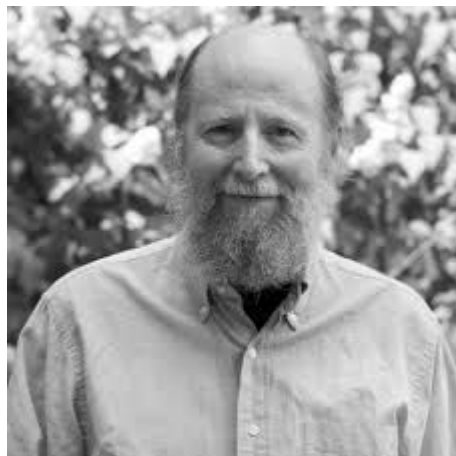
$$\hat{\mathbb{E}}_{\tau_i}[\nabla_{\theta} \log \pi_{\theta}(a|s) Q(s, a)] - \lambda \nabla_{\theta} H(\pi_{\theta}), \quad (18)$$

where $Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s, a)) | s_0 = \bar{s}, a_0 = \bar{a}]$

- 6: **end for**

The end?

A new trend in frontier AI/DL: the “bitter lesson”



The biggest lesson that can be read from 70 years of AI research is that **general methods that leverage computation are ultimately the most effective**, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of **continued exponentially falling cost per unit of computation**. [...]

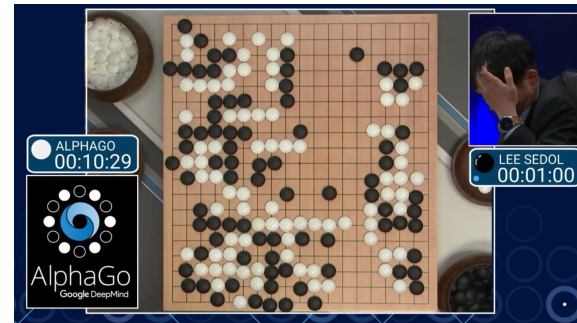
Seeking an improvement that makes a difference in the shorter term, **researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation**. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. [...]

And the **human-knowledge approach tends to complicate methods** in ways that make them less suited to taking advantage of general methods leveraging computation. [...] --- **Rich Sutton**

Two famous examples

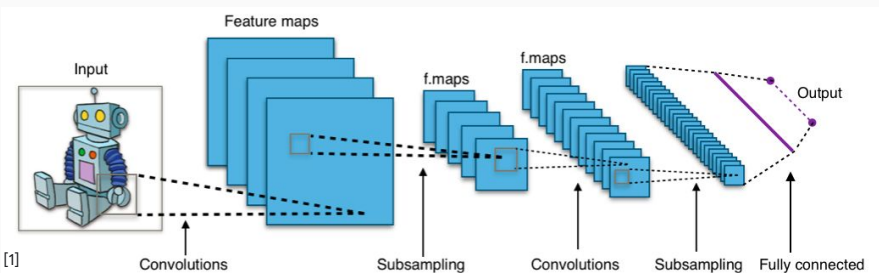
Deep Blue and AlphaGo leveraged massive, brute force search strategies and then self-play

At the time, this was looked upon with dismay by the majority of computer-chess/go researchers who had pursued methods that leveraged human understanding of the special structure of chess/go!



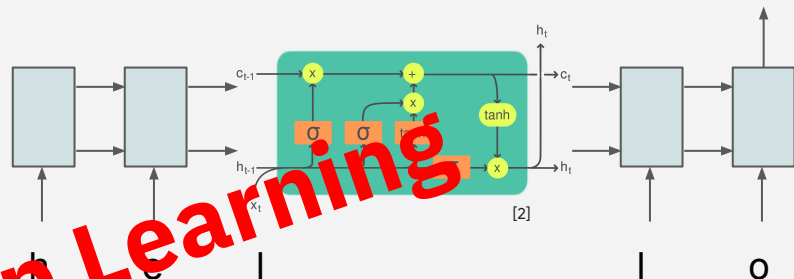
Computer Vision

Convolutional NNs (+ResNets)



Natural Lang. Proc.

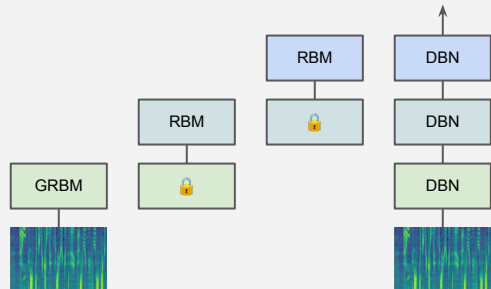
Recurrent NNs (+LSTMs)



Pre-2017 Deep Learning

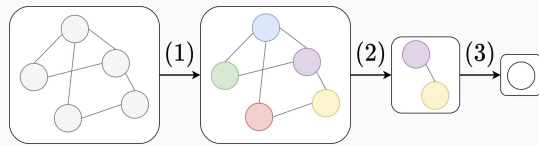
Speech

Deep Belief Nets (+non-DL)



Science

Graph NNs



RL

BC/GAIL

Algorithm 1 Generative adversarial imitation learning

- 1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
- 2: **for** $i = 0, 1, 2, \dots$ **do**
- 3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
- 4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_w \log(D_w(s, a))] + \hat{\mathbb{E}}_{\tau_E} [\nabla_w \log(1 - D_w(s, a))] \quad (17)$$

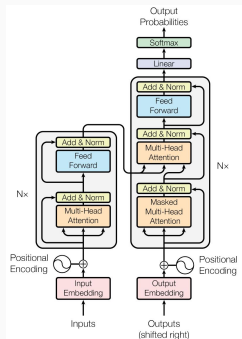
- 5: Take a policy step from θ_i to θ_{i+1} , using the TRPO rule with cost function $\log(D_{w_{i+1}}(s, a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q(s, a)] - \lambda \nabla_{\theta} H(\pi_{\theta}), \quad (18)$$

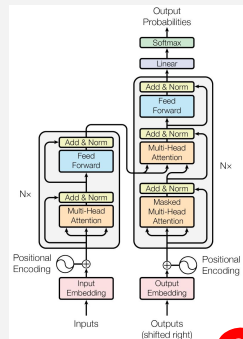
where $Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i} [\log(D_{w_{i+1}}(s, a)) | s_0 = \bar{s}, a_0 = \bar{a}]$

- 6: **end for**

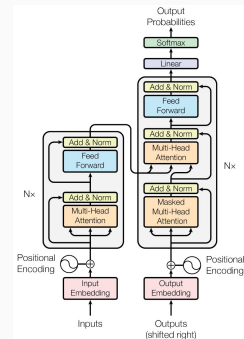
Computer Vision



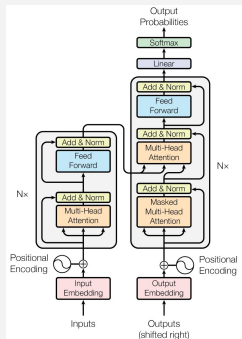
Natural Lang. Proc.



Reinf. Learning

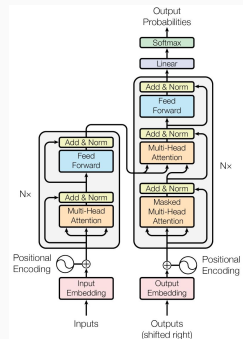


Speech

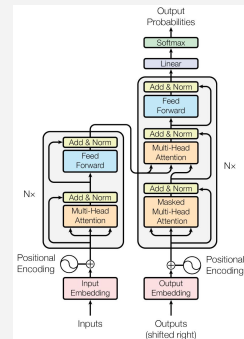


Current trends

Translation



Graphs/Science



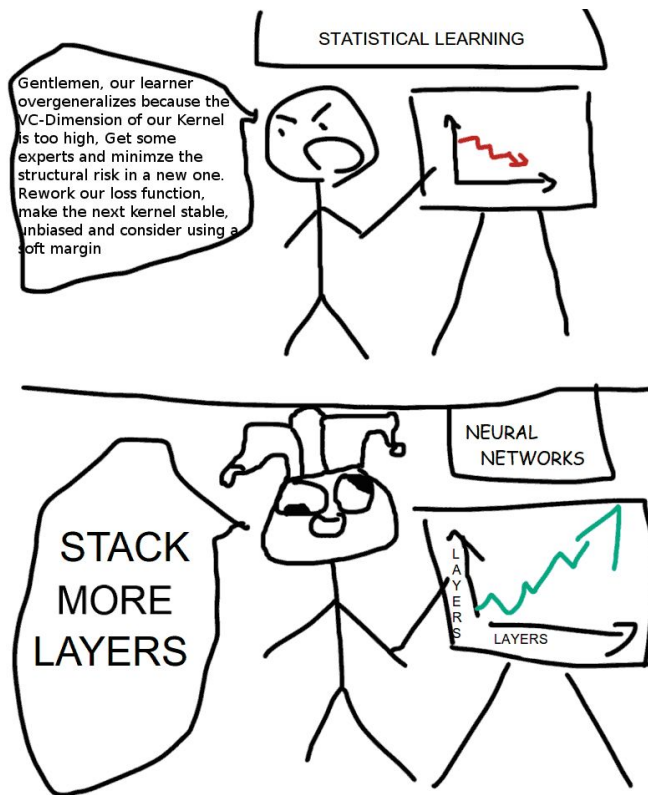
So in the end, we're back to stacking layers??

Not quite!

Everything we've seen in this lecture is the backbone of scaling Deep(er) Networks and making them converge

Human/Physical knowledge is still extremely helpful and impactful in the short/medium term both for time and scale regimes

And most of the times we are dealing with finite resources to train/deploy



Conclusions

From a simple set of linear algebra operations, we can construct incredible tools called Deep Neural Networks

Basic neurons are not enough! We need to introduce a set of algorithms and data preprocessing to make learning easier, more stable, and more generalizable

Different ways of combining neurons can result in optimized architectures for handling different data types... but who knows what the future holds!

backup

We can do the math if needed

$$\text{MSE} \triangleq \text{E} [(y - \hat{f})^2] = \text{E} [y^2 - 2y\hat{f} + \hat{f}^2] = \text{E} [y^2] - 2\text{E} [y\hat{f}] + \text{E} [\hat{f}^2]$$

$$\begin{aligned} \text{E} [y^2] &= \text{E} [(f + \varepsilon)^2] \\ &= \text{E}[f^2] + 2\text{E}[f\varepsilon] + \text{E}[\varepsilon^2] && \text{by linearity of E} \\ &= f^2 + 2f\text{E}[\varepsilon] + \text{E}[\varepsilon^2] && \text{since } f \text{ does not depend on the data} \\ &= f^2 + 2f \cdot 0 + \sigma^2 && \text{since } \varepsilon \text{ has zero mean and variance } \sigma^2 \end{aligned}$$

$$\begin{aligned} \text{E} [y\hat{f}] &= \text{E} [(f + \varepsilon)\hat{f}] \\ &= \text{E}[f\hat{f}] + \text{E}[\varepsilon\hat{f}] && \text{by linearity of E} \\ &= \text{E}[f\hat{f}] + \text{E}[\varepsilon]\text{E}[\hat{f}] && \text{since } \hat{f} \text{ and } \varepsilon \text{ are independent} \\ &= f\text{E}[\hat{f}] && \text{since } \text{E}[\varepsilon] = 0 \end{aligned} \qquad \begin{aligned} \text{MSE} &= f^2 + \sigma^2 - 2f\text{E}[\hat{f}] + \text{Var}[\hat{f}] + \text{E}[\hat{f}]^2 \\ &= (f - \text{E}[\hat{f}])^2 + \sigma^2 + \text{Var} [\hat{f}] \\ &= \text{Bias}[\hat{f}]^2 + \sigma^2 + \text{Var} [\hat{f}] \end{aligned}$$

$$\text{E} [\hat{f}^2] = \text{Var}(\hat{f}) + \text{E}[\hat{f}]^2 \quad \text{since } \text{Var}[X] \triangleq \text{E} [(X - \text{E}[X])^2] = \text{E}[X^2] - \text{E}[X]^2 \text{ for any random variable } X$$