

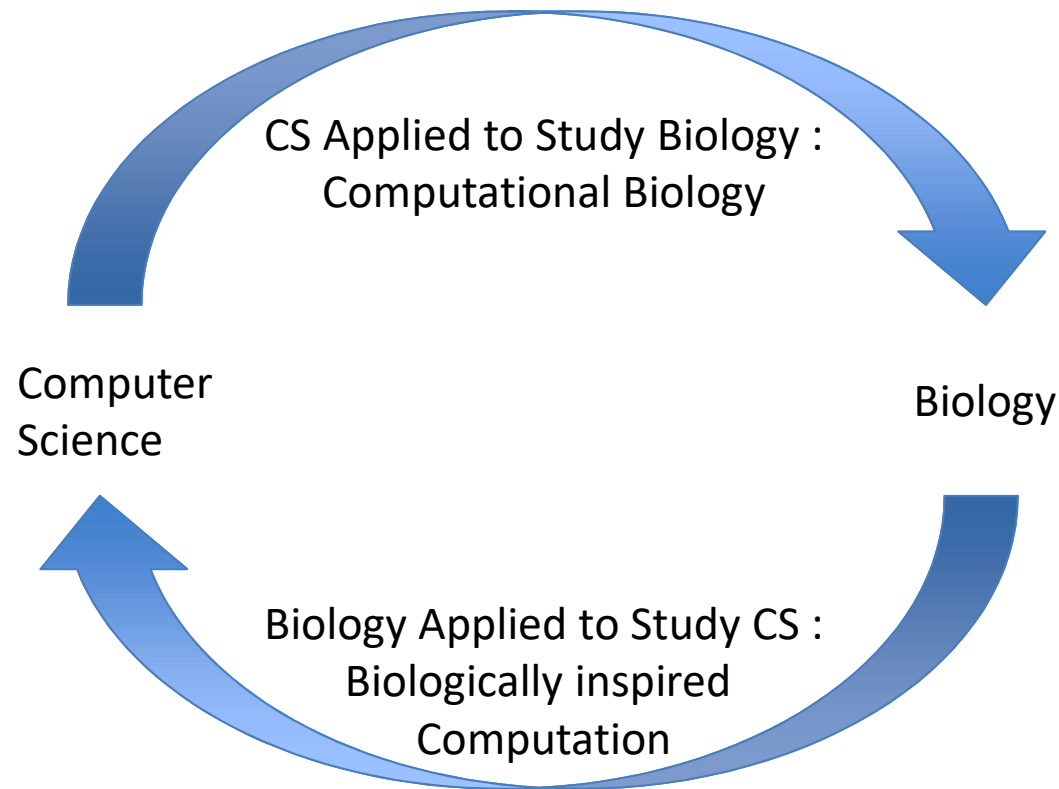
Deep learning in bioinformatics

Pradipta Ray

BIOL 6385 / BIOL 6389

(pedagogical structure based on
Vincent Vanhoucke's course)

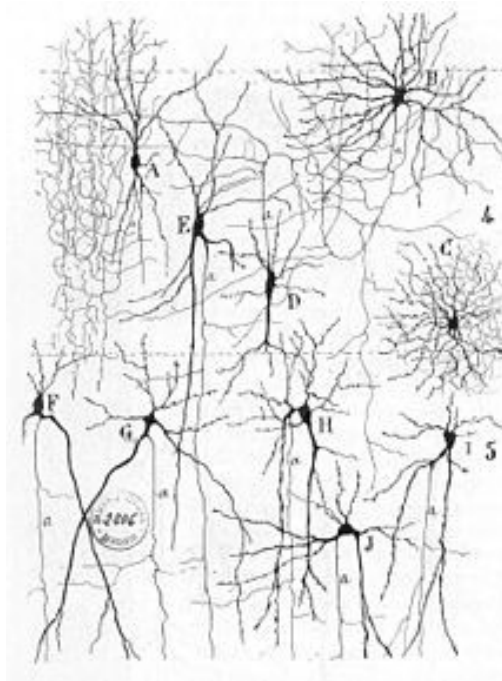
Transfer learning : Computational biology vs Biologically inspired computation



Reticular vs cellular - connectionist models of brain



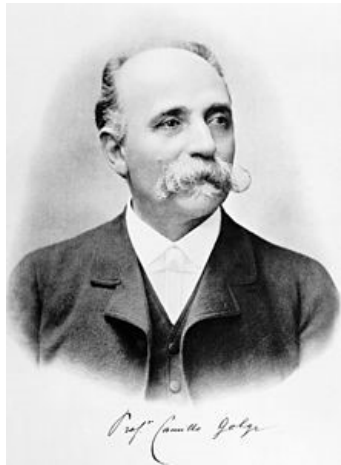
Joseph von Gerlach



The brain is composed of many cells (neurons) each with multiple connections.



Santiago Ramon y Cajal

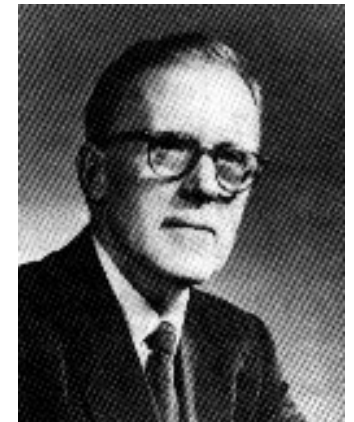


Camillo Golgi

The brain is a single, continuous network flow, with branching connections between regions

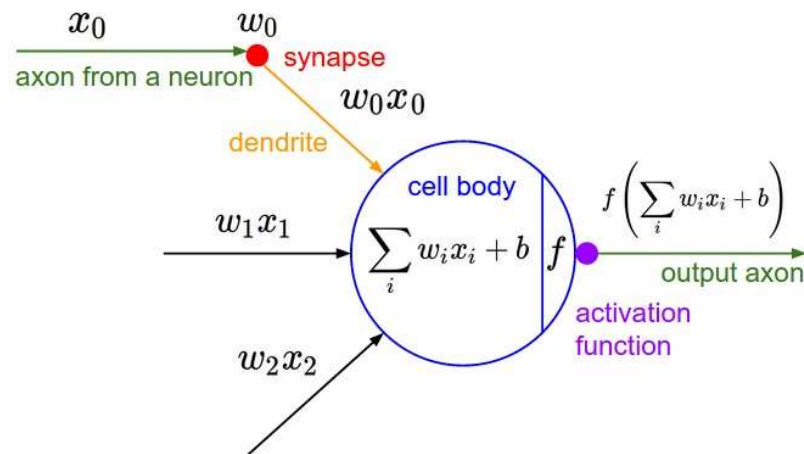
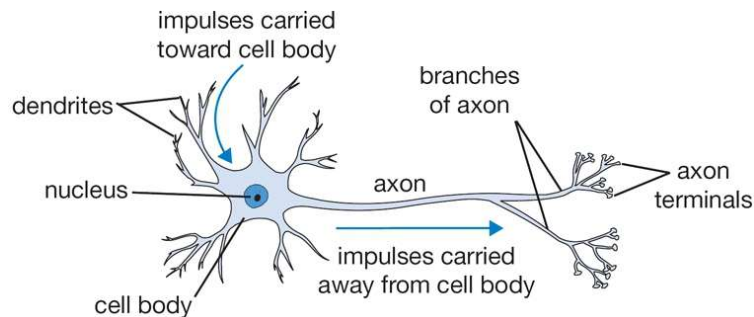
Wikipedia

Neurons that fire together wire together : electrical conductivity identified by Berger is achieved by dedicated circuits.



Donald Hebb

Perceptron

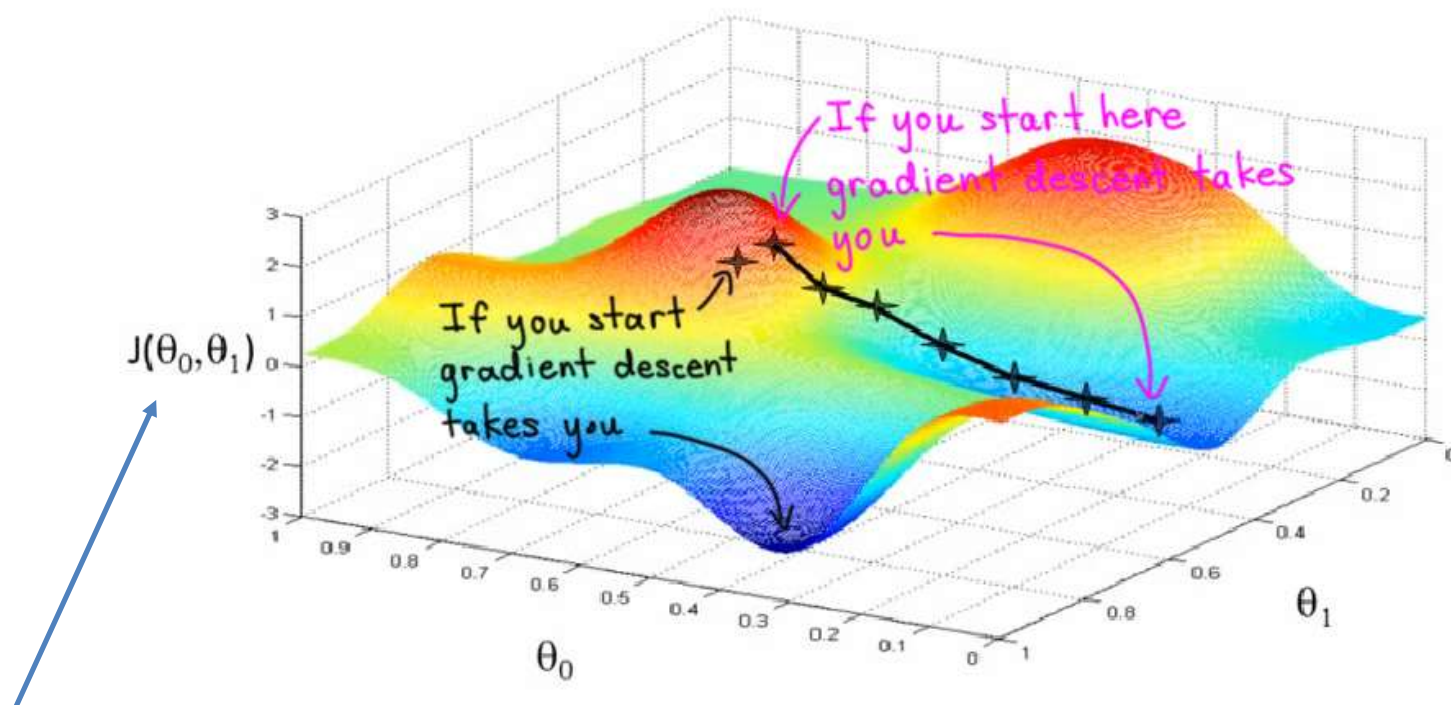


- Thresholded, weighted sum of inputs : output 0 or 1 based on superiority test
- Predictor / input variables : Many, real-valued
- Response variable : discrete (binary)

What kind of a learning problem is this ?

Estimating perceptron parameters

- Gradient descent



What function should we optimize ?

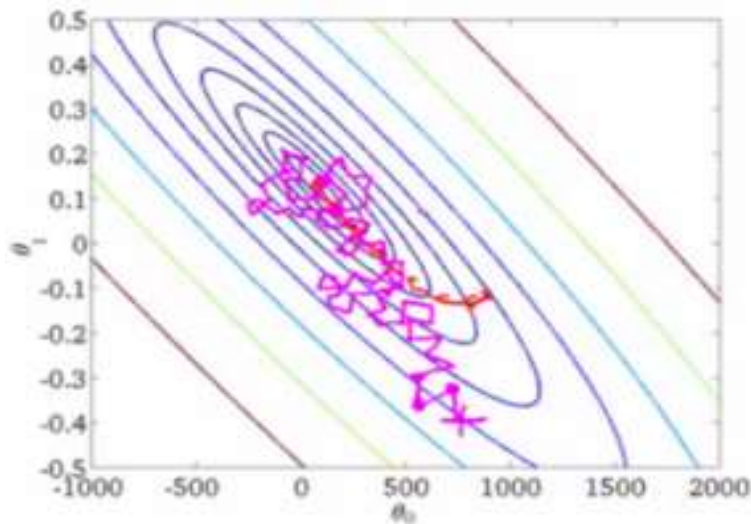
Stochastic gradient descent

Offline, batch and online learning

Adaptive learning
rate

For $j = 1, 2, \dots$

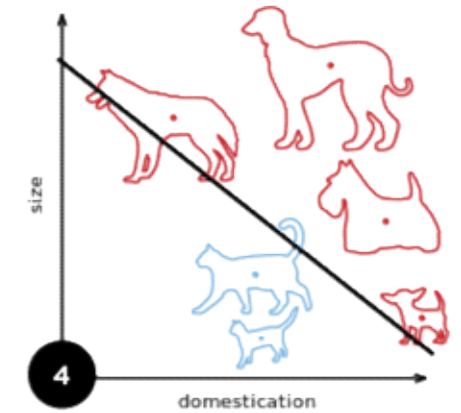
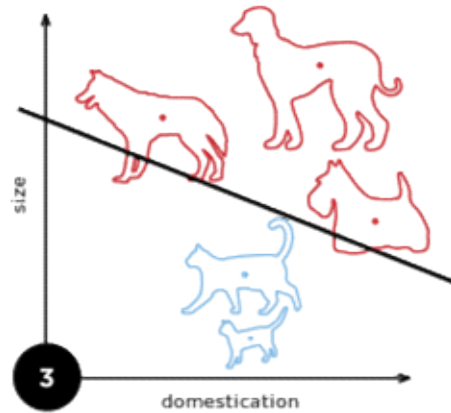
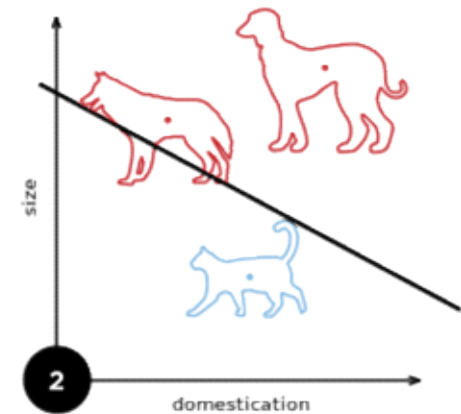
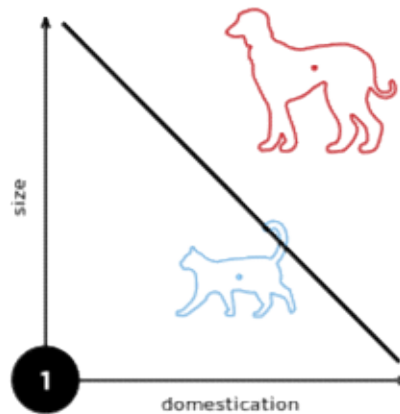
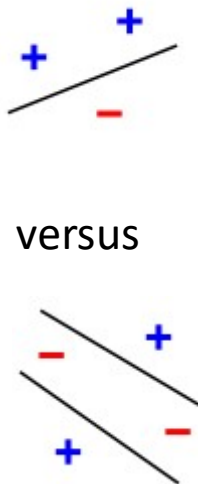
$$\mathbf{x}^{(j+1)} = \mathbf{x}^{(j)} - \gamma \nabla f_i(\mathbf{x}^{(j)})$$



Instead of using entire dataset, uses batches of data for each step

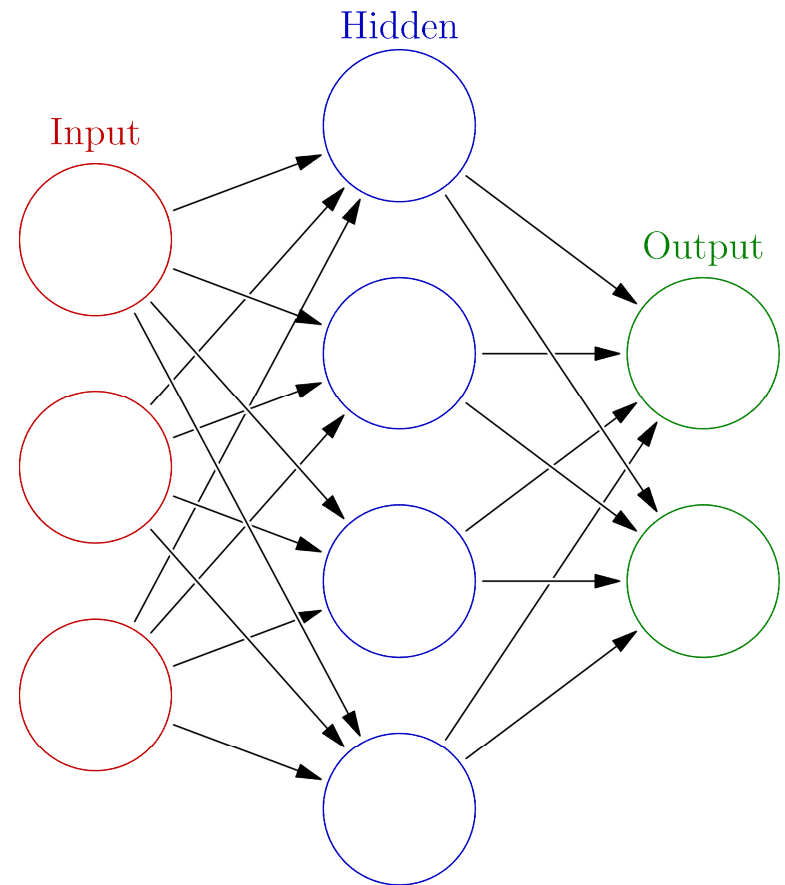
Problems with perceptrons

- Can only classify linearly separable datasets



Neural Networks

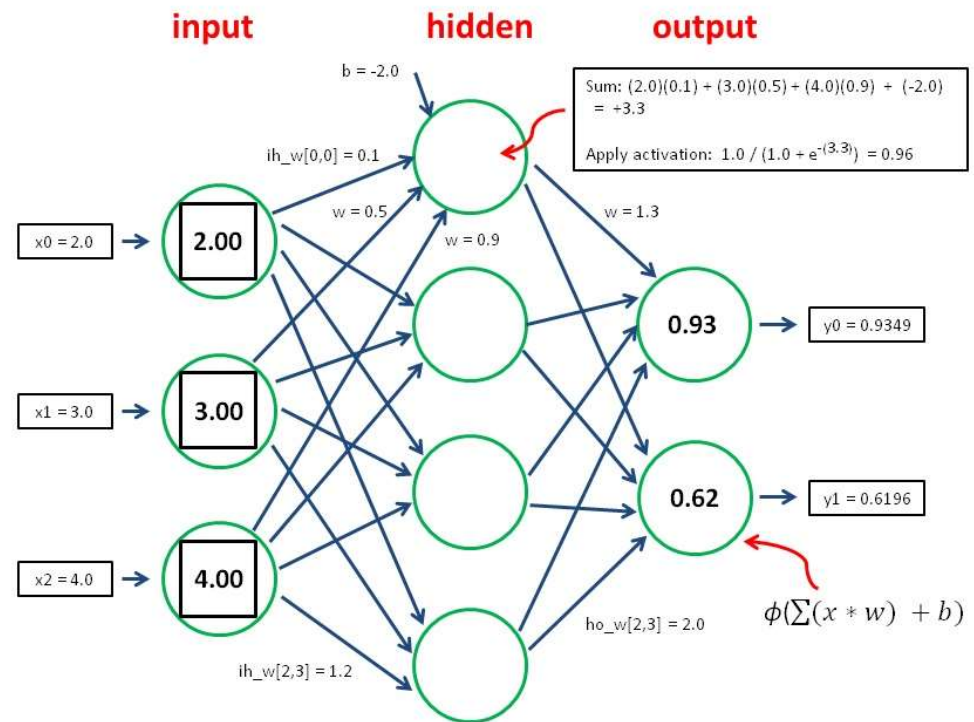
- Layered network of perceptrons
- Able to perform non-linear separation for classification
- Question : how to train it ?



A case in action

NB

- There can be multiple outputs
- The output layer need not be thresholded
- Topology :
feedforward vs recurrent



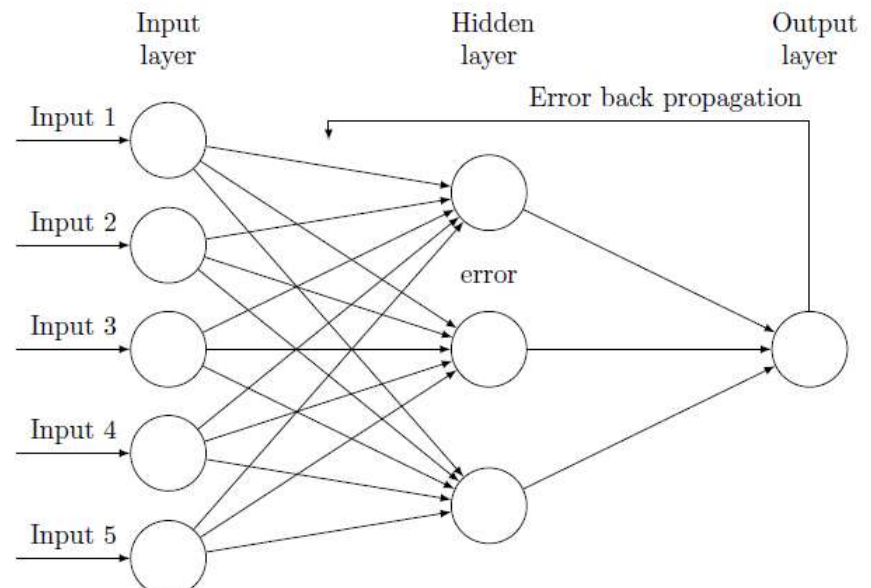
Backpropagation

- SGD through layers : how well does it work ?

Wikipedia

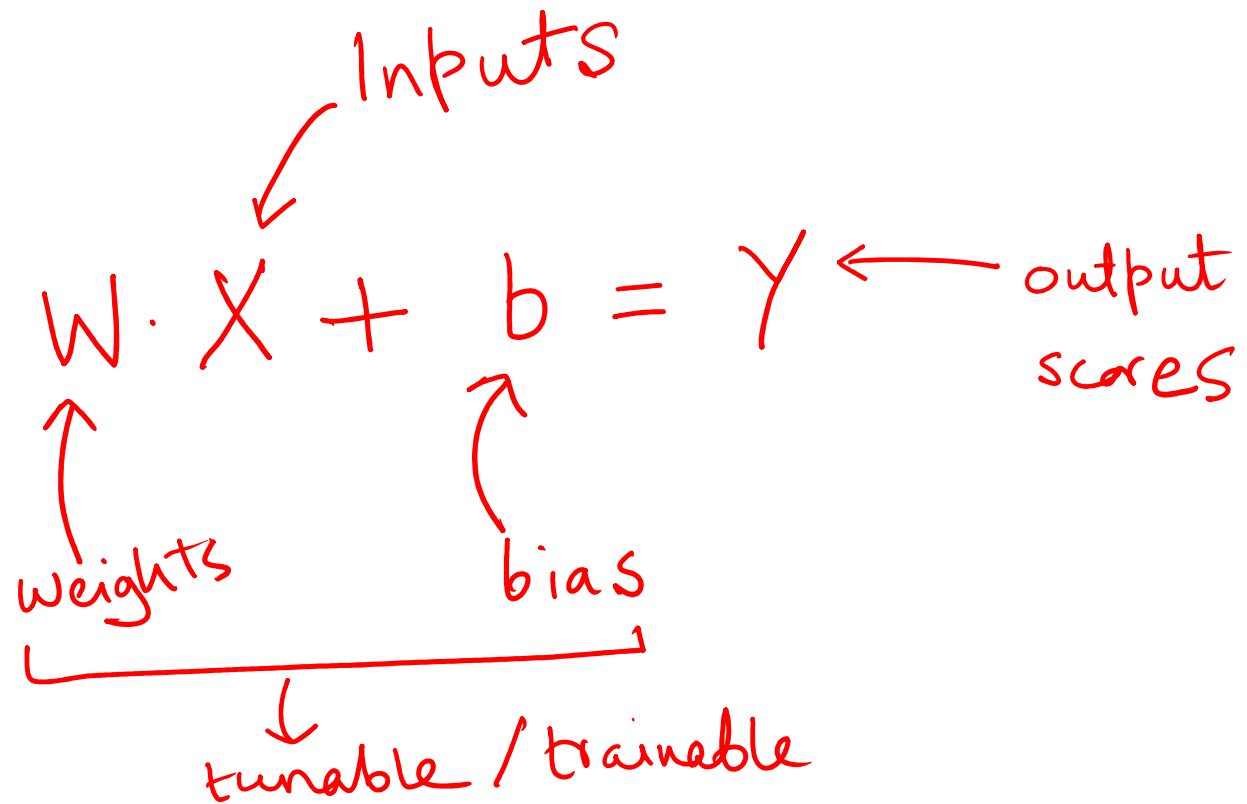
```
initialize network weights (often small random values)
do
  forEach training example named ex
    prediction = neural-net-output(network, ex) // forward pass
    actual = teacher-output(ex)
    compute error (prediction - actual) at the output units
    compute  $\Delta w_h$  for all weights from hidden layer to output layer // backward pass
    compute  $\Delta w_i$  for all weights from input layer to hidden layer // backward pass continued
    update network weights // input layer not modified by error estimate
  until all examples classified correctly or another stopping criterion satisfied
return the network
```

- Works well in theory & practice
 - “Ripple” backwards



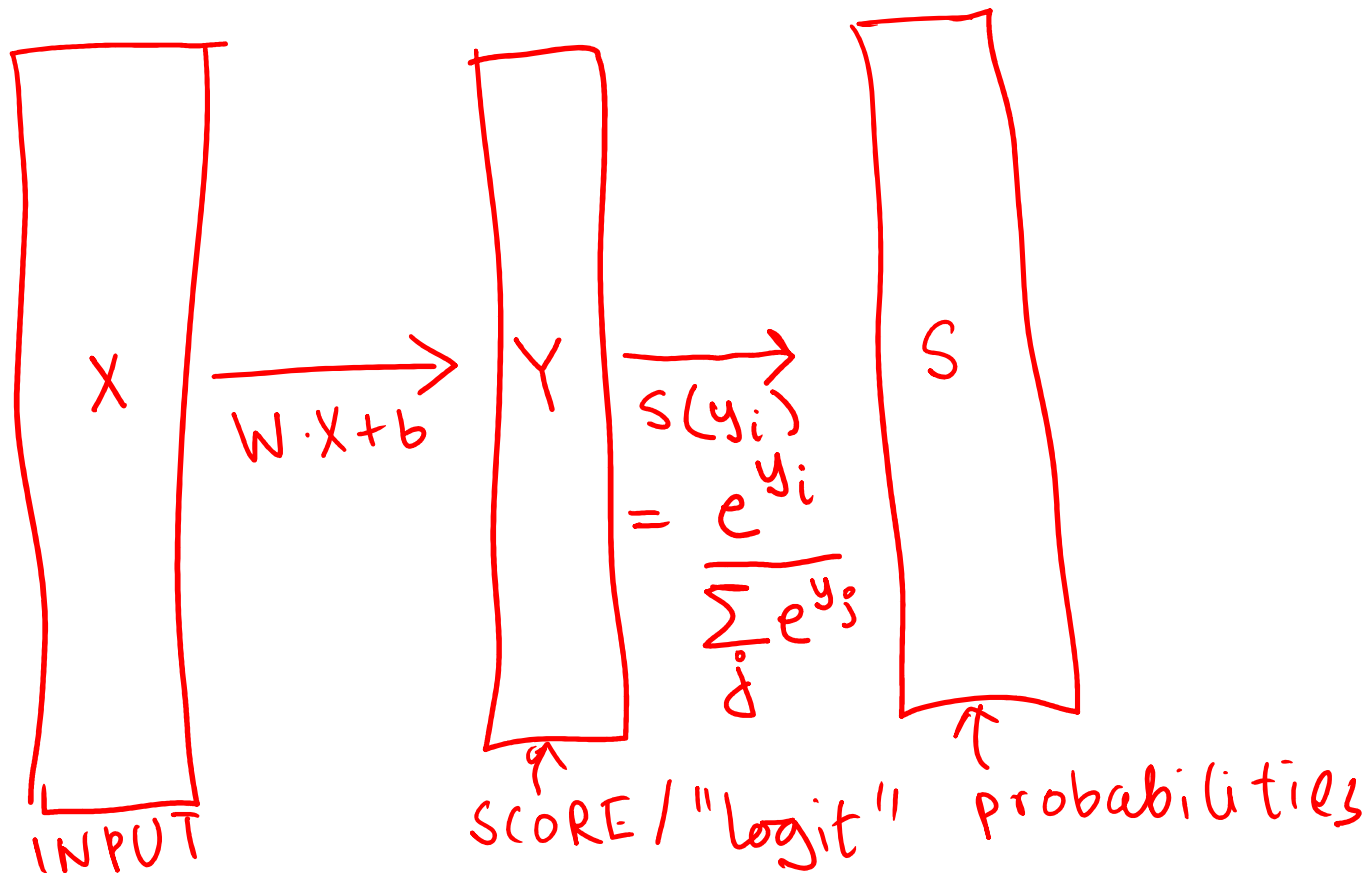
C M Hughes

Logistic classifier



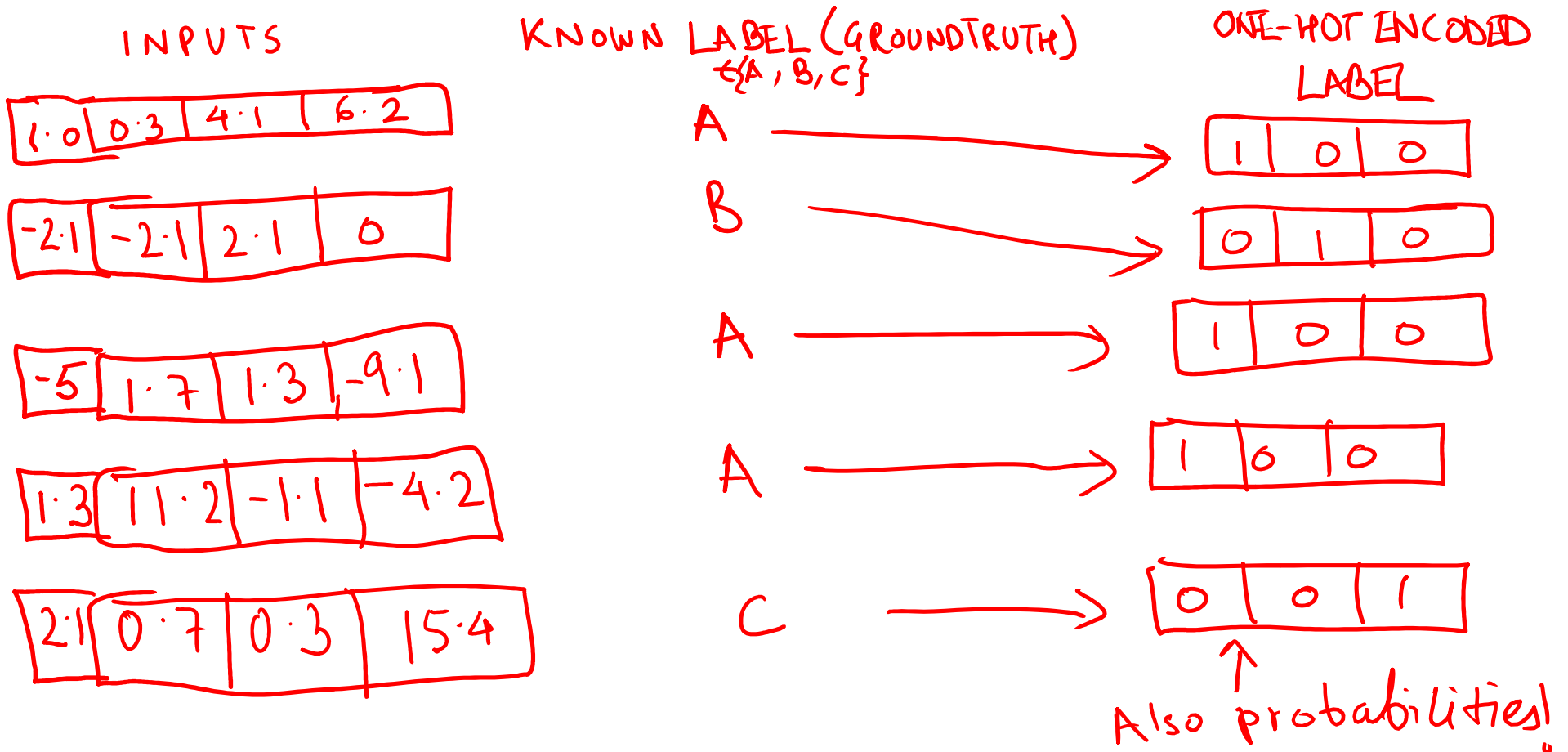
Logits to probabilities

- Converting scores to probabilities



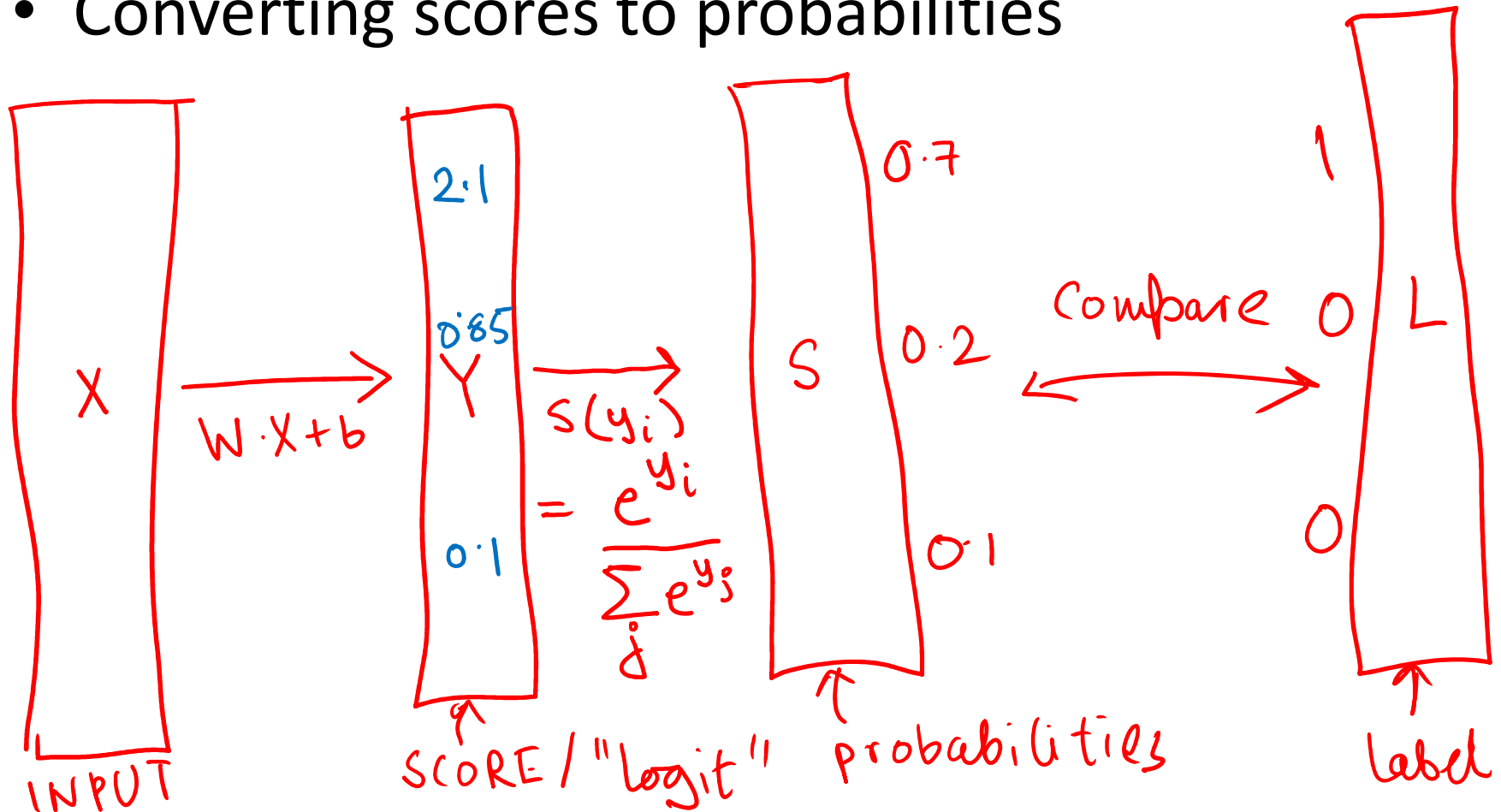
One-hot encoding

- A useful representation of the class label



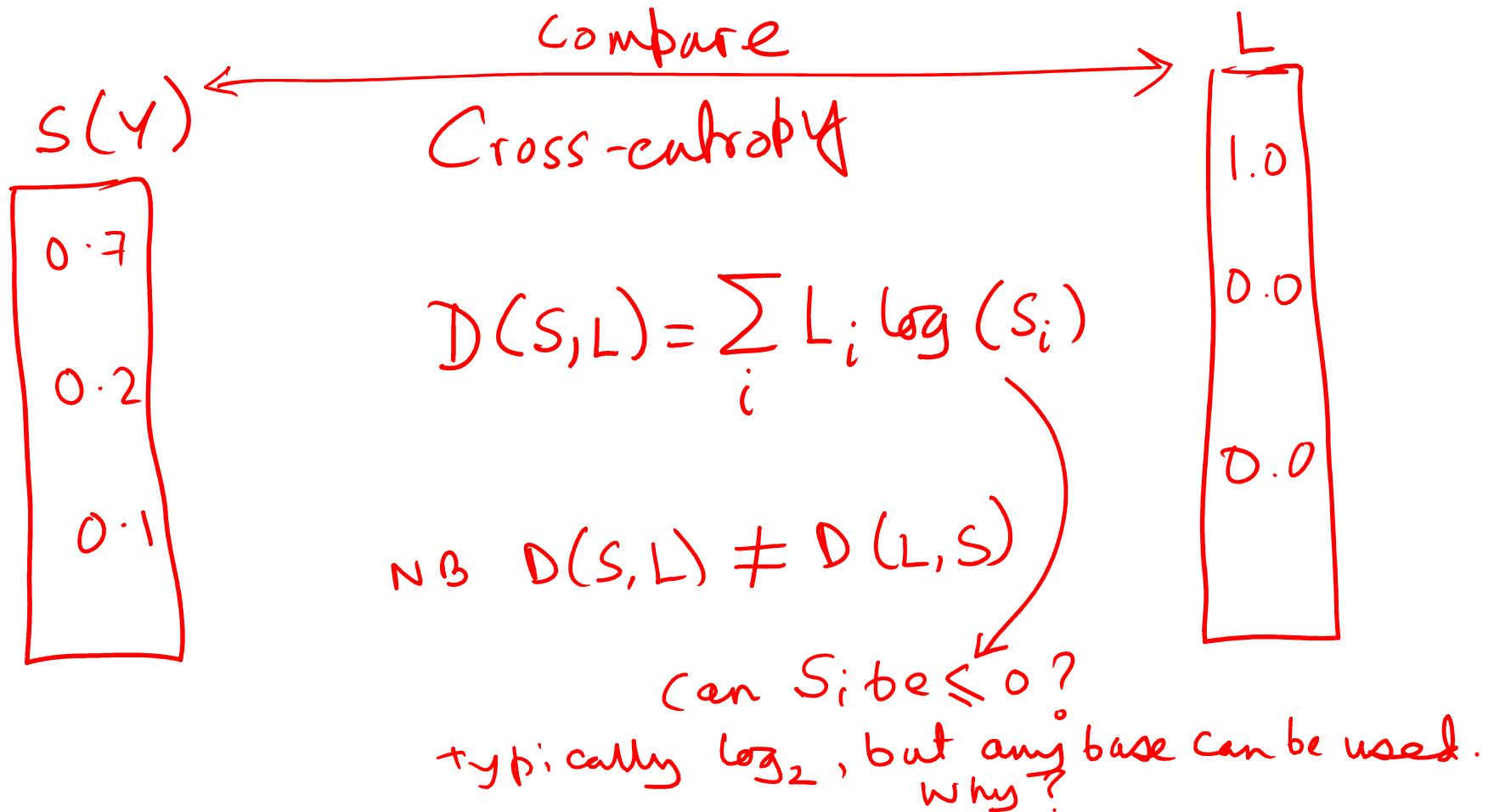
How good is the prediction ?

- Converting scores to probabilities



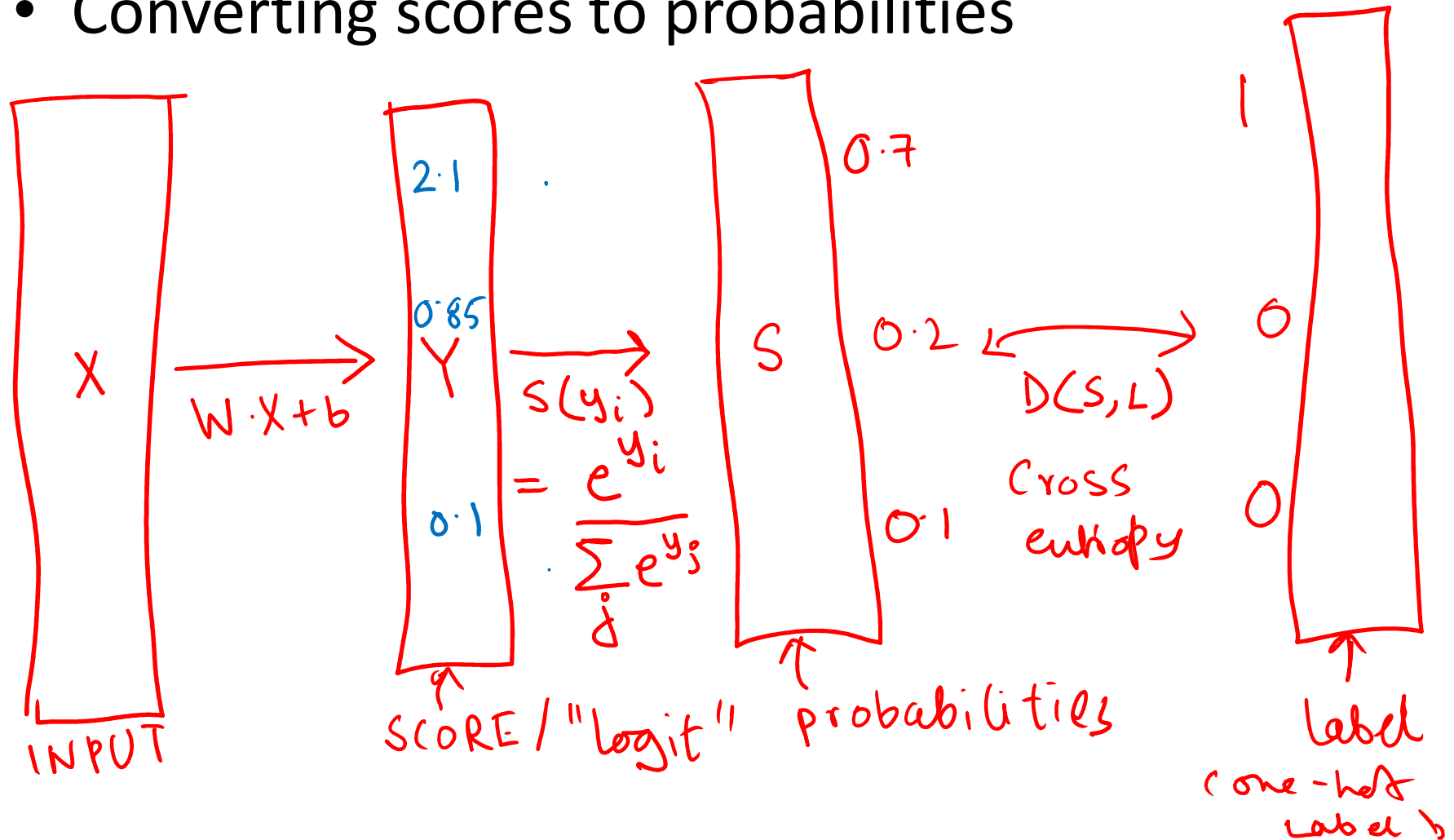
How good is my prediction ?

- Asymmetric : not a proper distance function



Putting it together : Multinomial logistic classifier

- Converting scores to probabilities



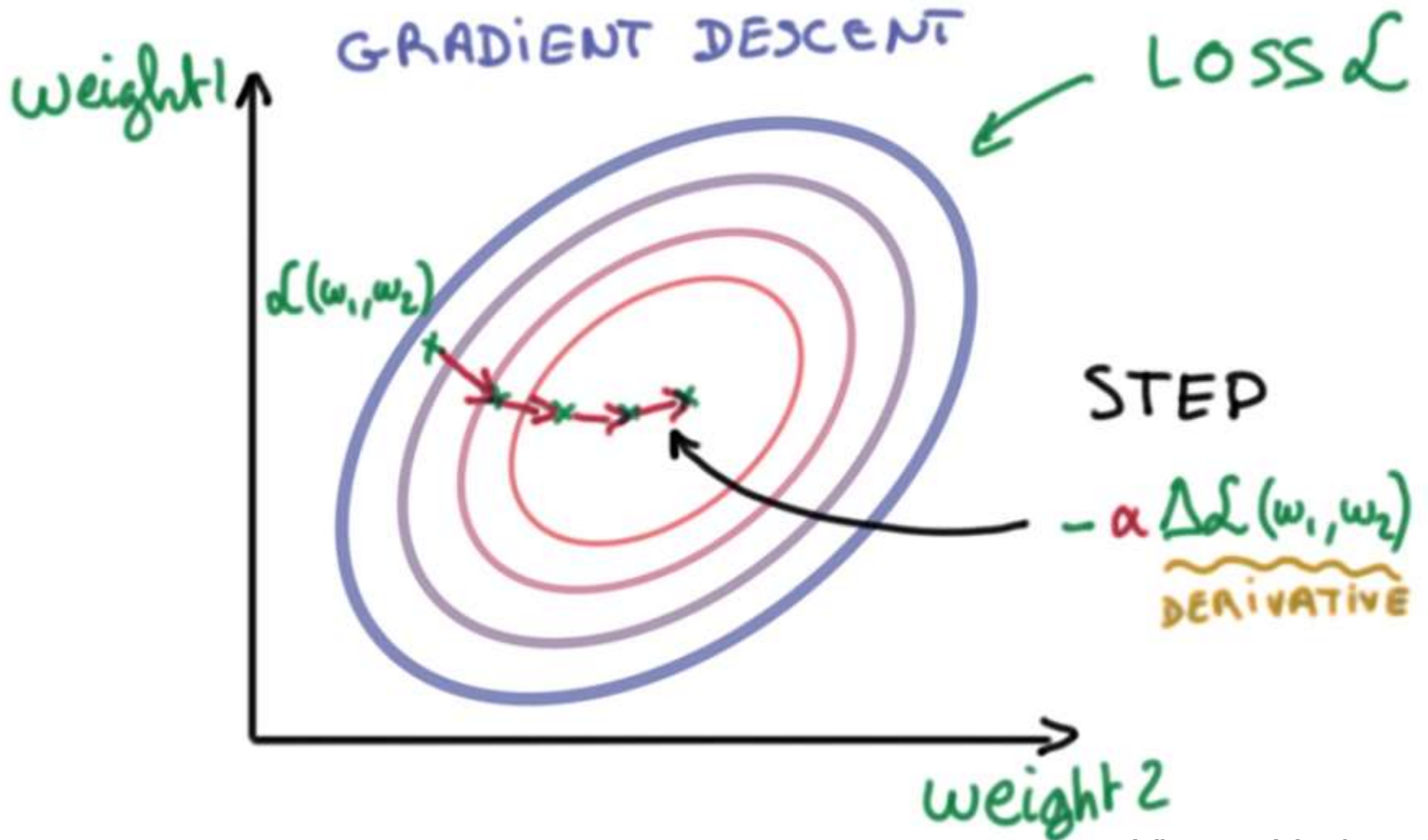
Loss function

- How well are we doing on the entire training data ?

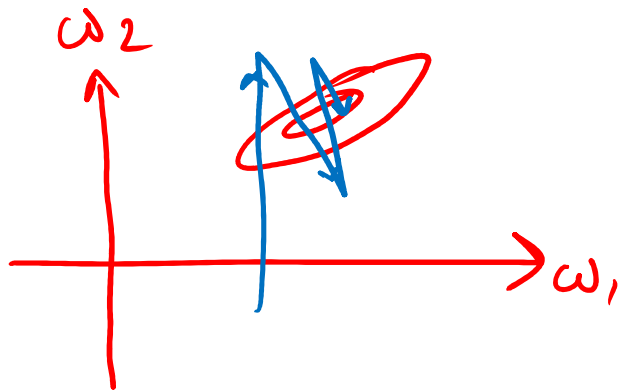
$$\mathcal{L} = \frac{1}{N} \sum_i D(s(WX_i + b), L_i)$$

↑
cross entropy averaged over
entire training data

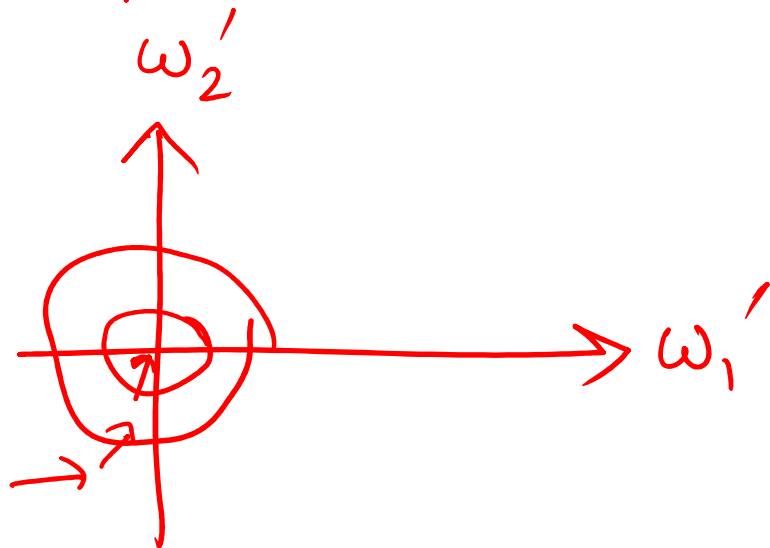
Gradient descent



Transform your variables



BADLY
CONDITIONED
WEIGHTS



WELL
CONDITIONED
WEIGHTS

$$\omega_i \rightarrow \omega_i'$$

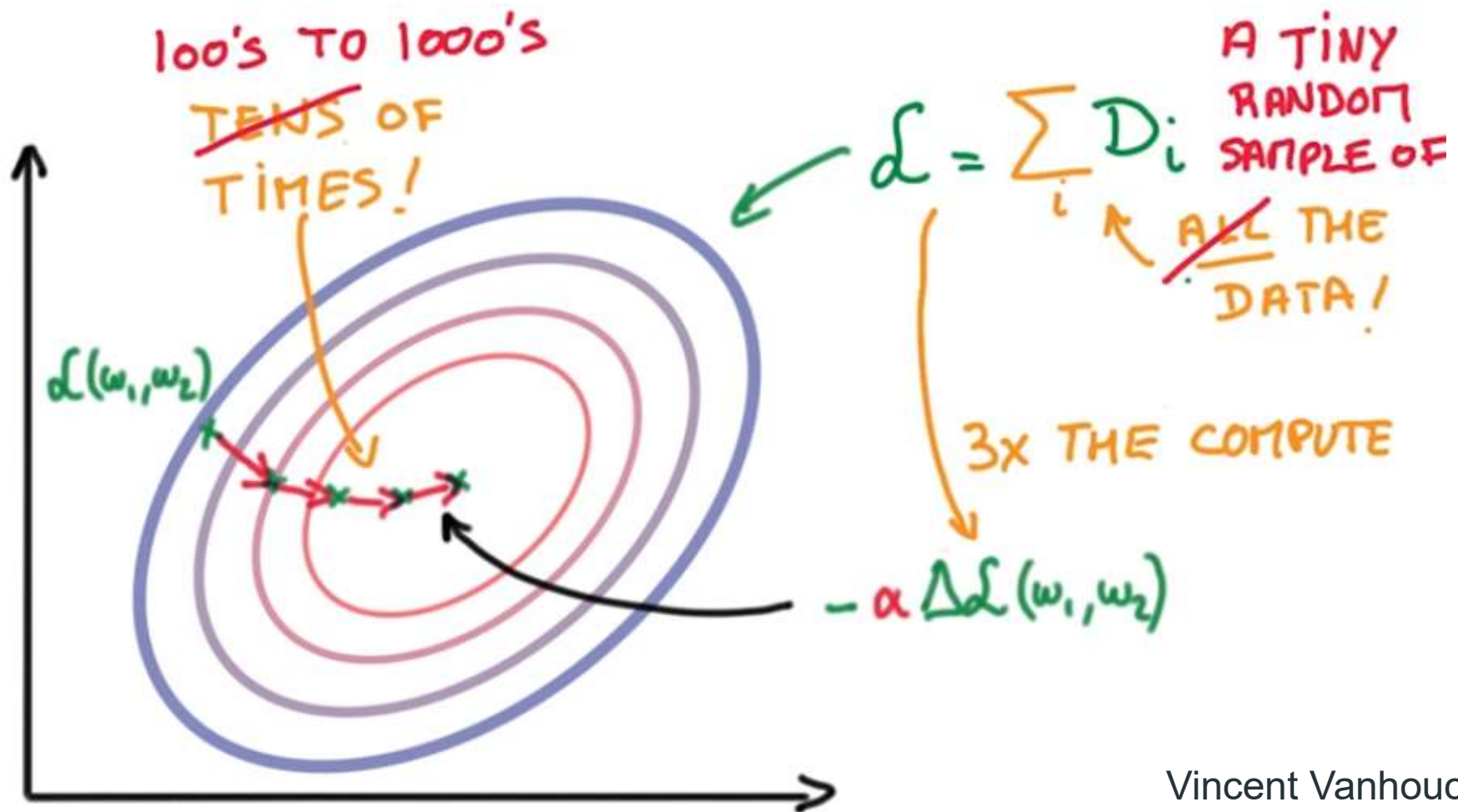
s.t.

$$\overline{\omega_i} = 0$$

$V(\omega_i)$'s are the
same

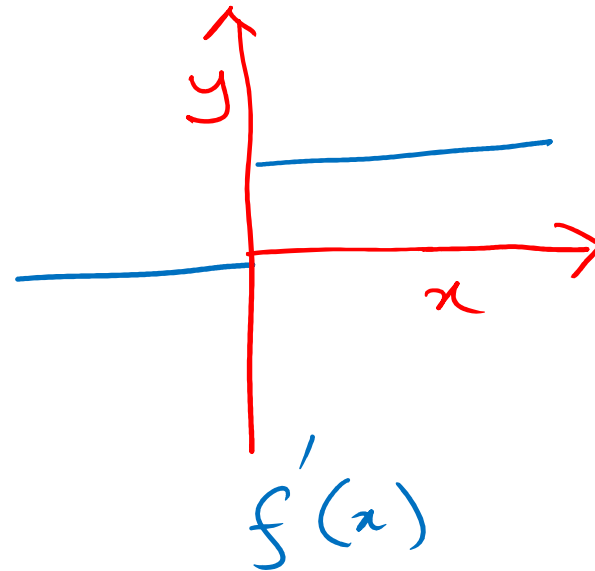
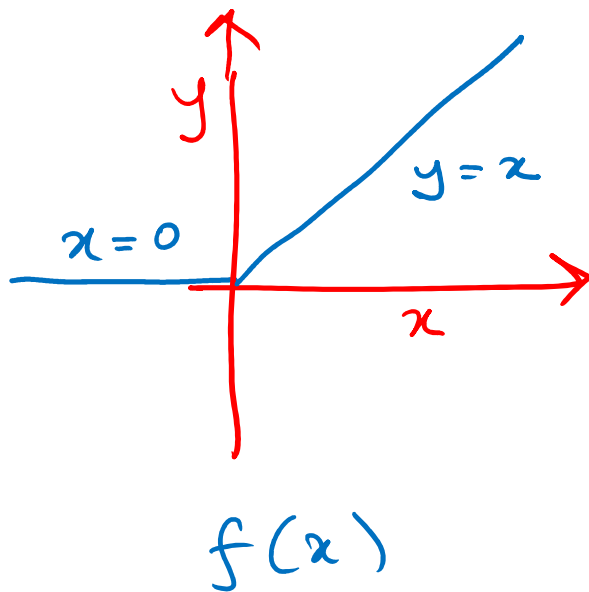
Still doesn't scale with data

- Stochastic gradient descent to the rescue



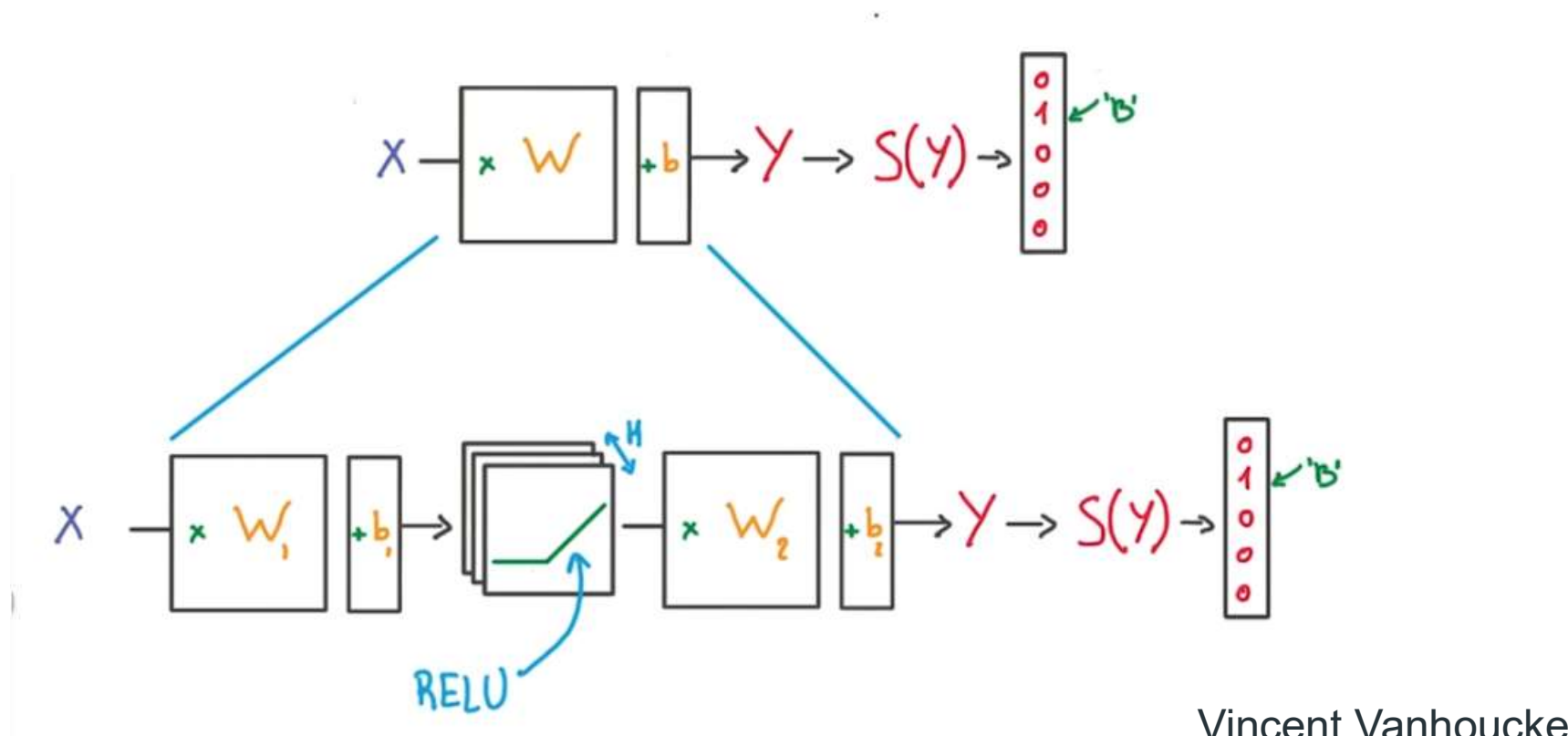
A simple way to introduce nonlinearity

- Piecewise linear function, well behaved derivative (catch : not defined at $x = 0$)



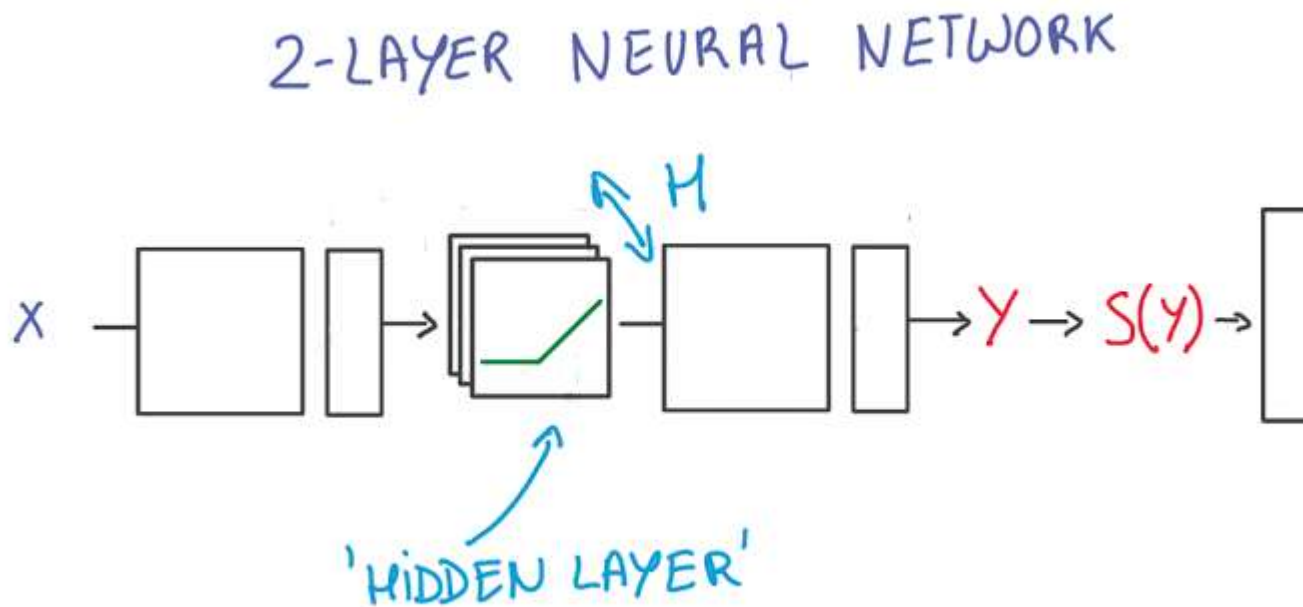
Adding nonlinearity to the system

- Introduce RELU as a layer before calculating the output : we have a neural network



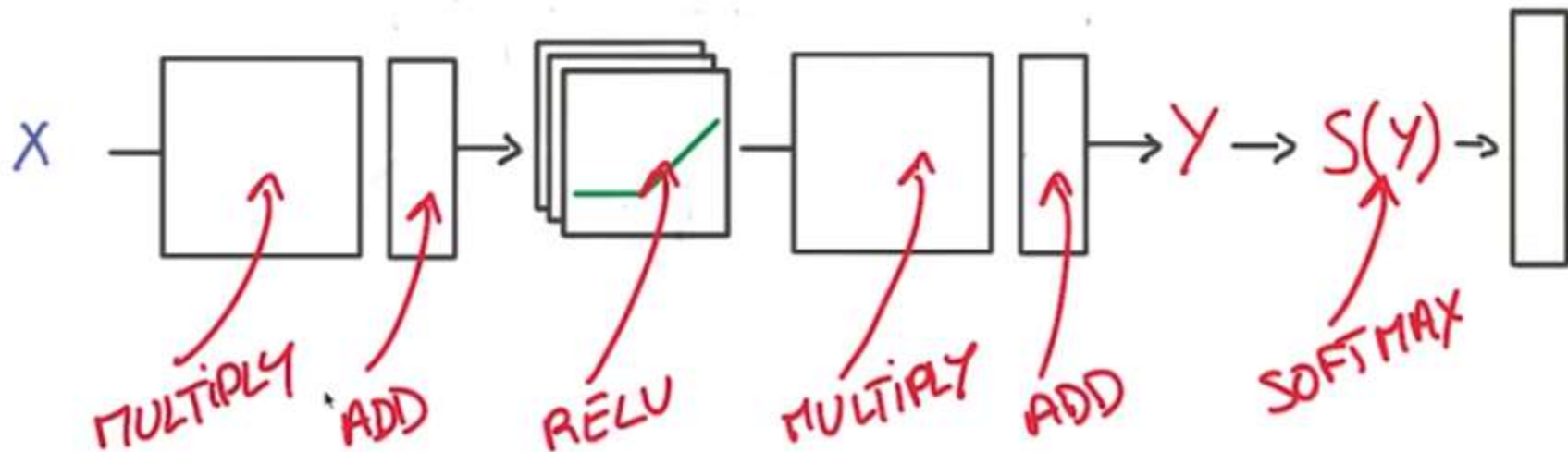
Traditional neural network

- One hidden layer



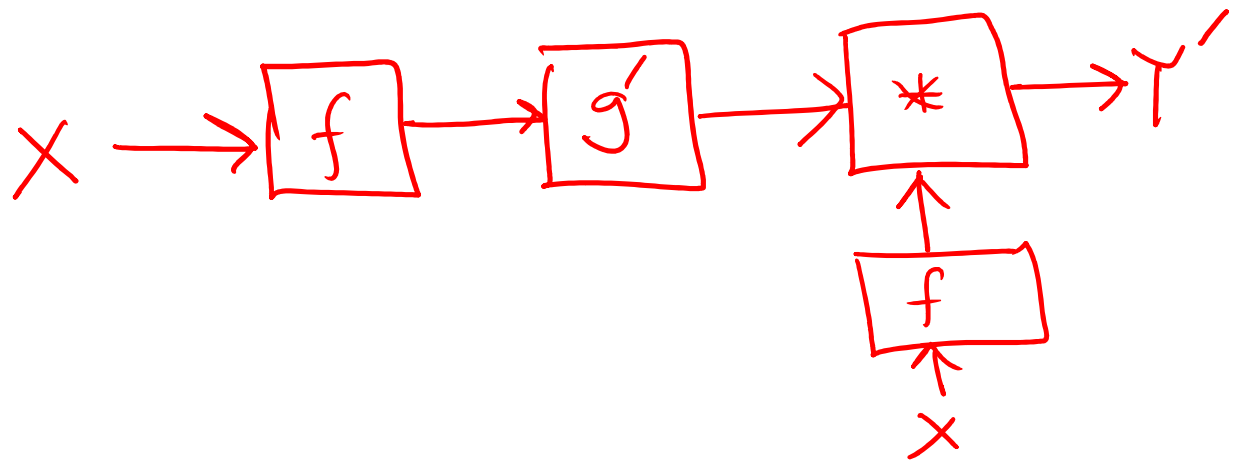
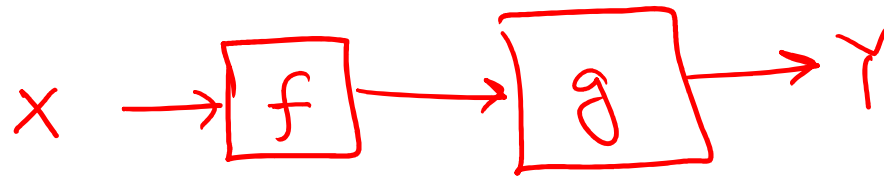
Step back

STACKING UP SIMPLE OPERATIONS



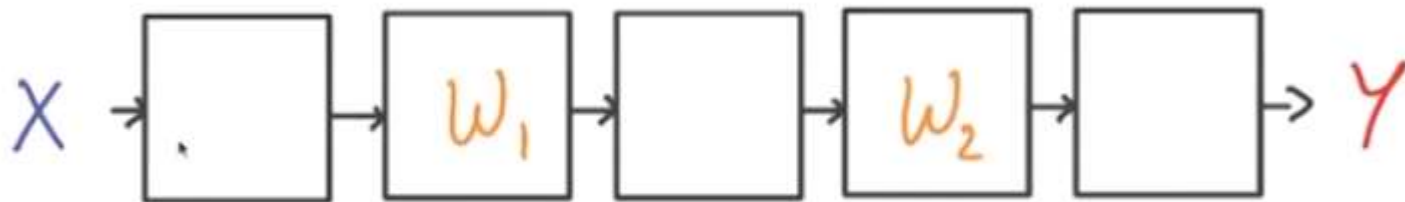
Chain rule

$$[g(f(x))]' = g'(f(x)) \times f'(x)$$



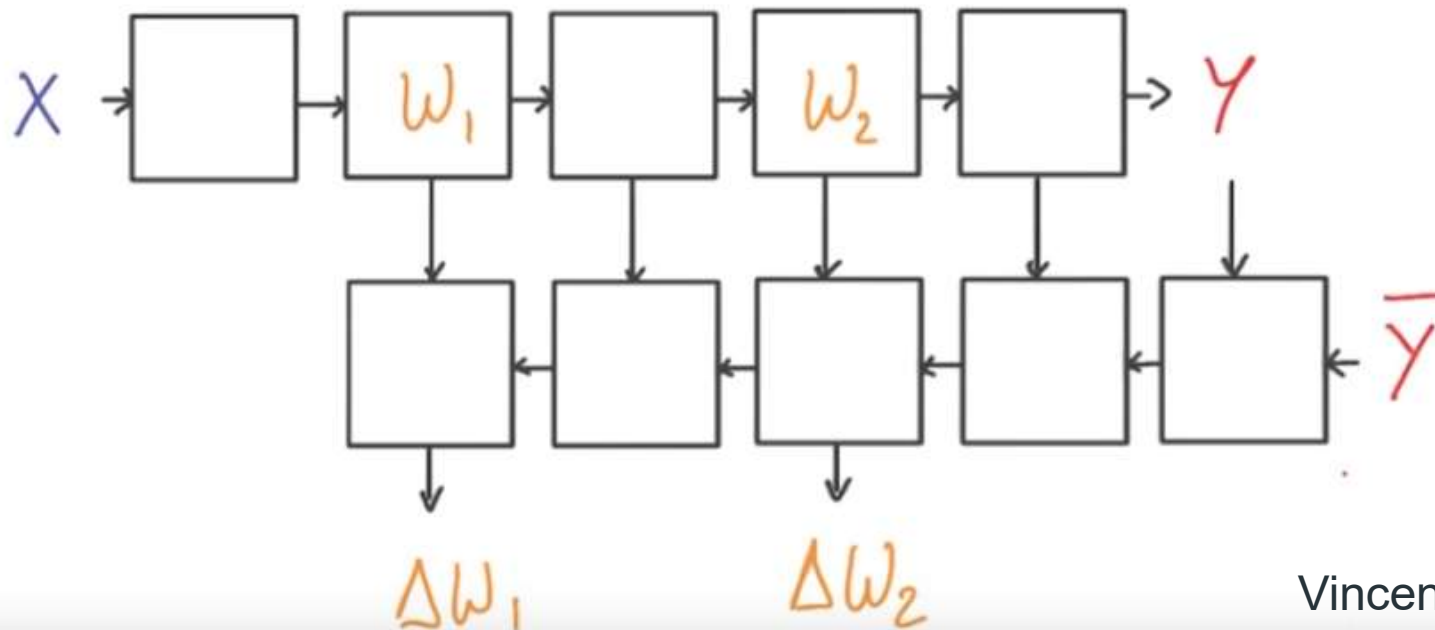
How would it work in practice ?

- Consider your framework as a pipeline of transformations transforming input X into output Y
- Some transformations are parameterized (eg. matrix multiplications) and some are not (eg. ReLu)

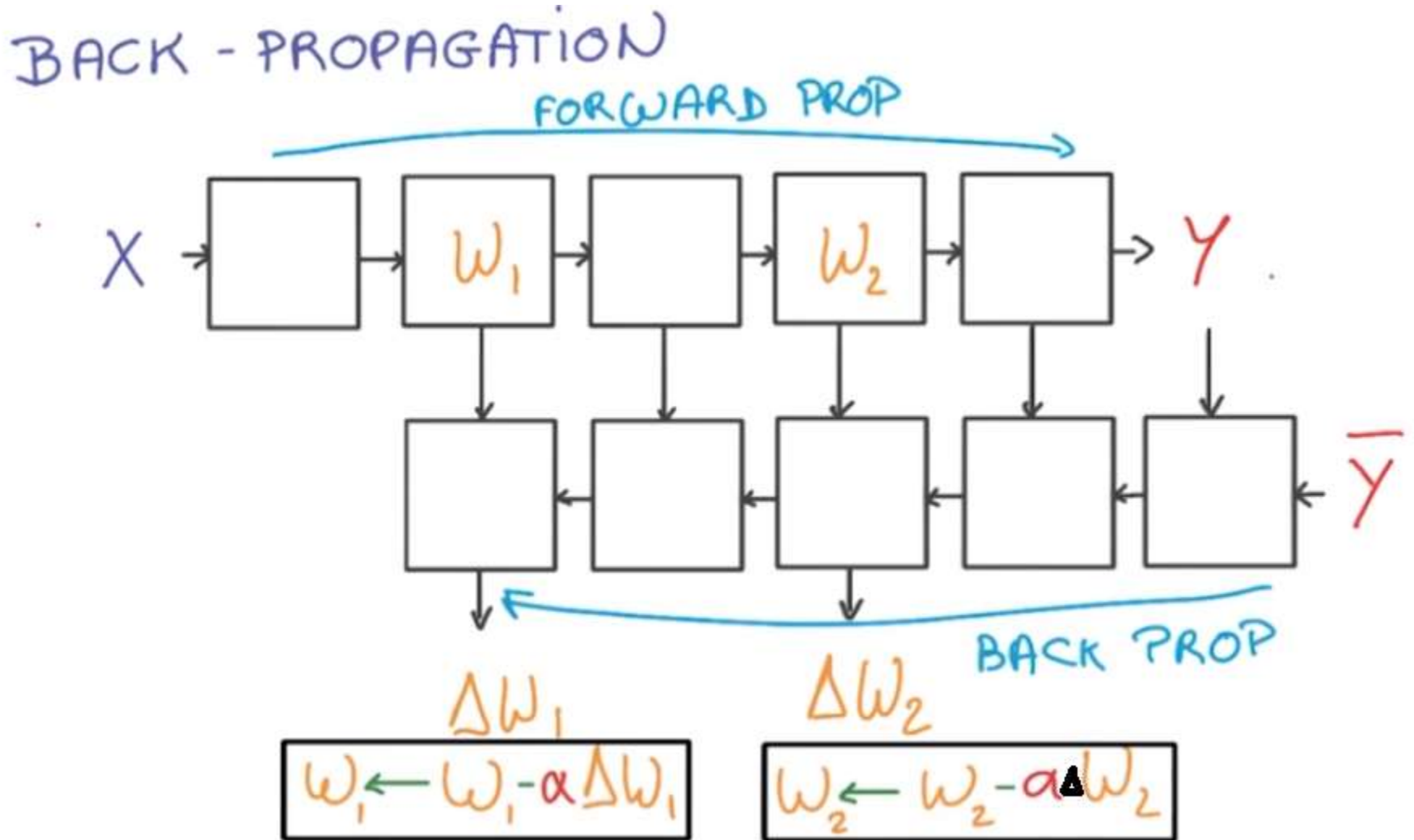


Utilizing chain rule

- Copy and shift the whole pipeline one block over
- Reverse flow of information on copied pipeline and couple the 2 pipelines as follows



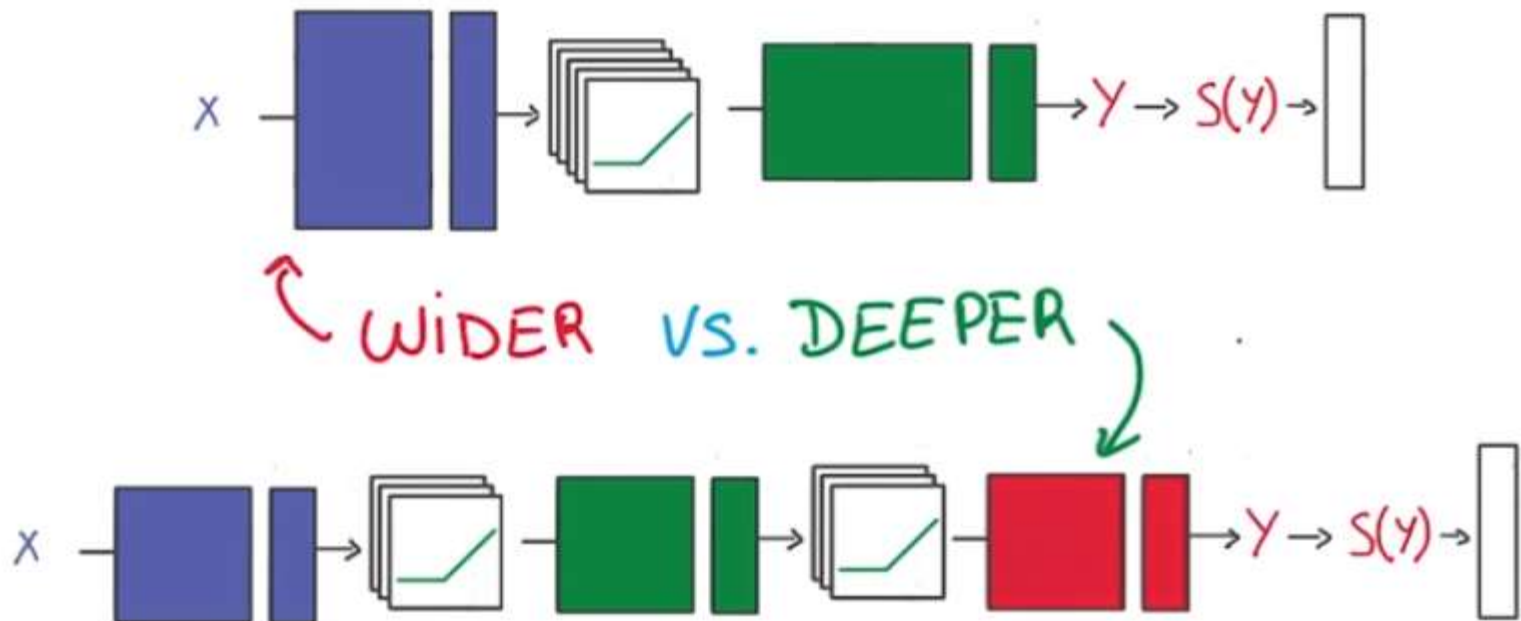
Backpropagation



Scalability : wider or deeper ?

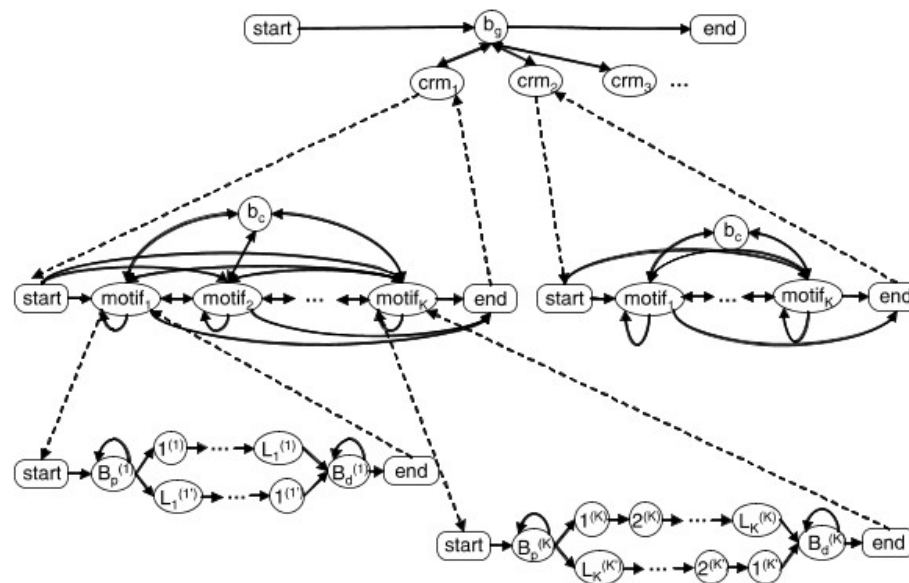
- Which is more tractable ?

DEEP NETWORKS



Why should multi-layered approaches work ?

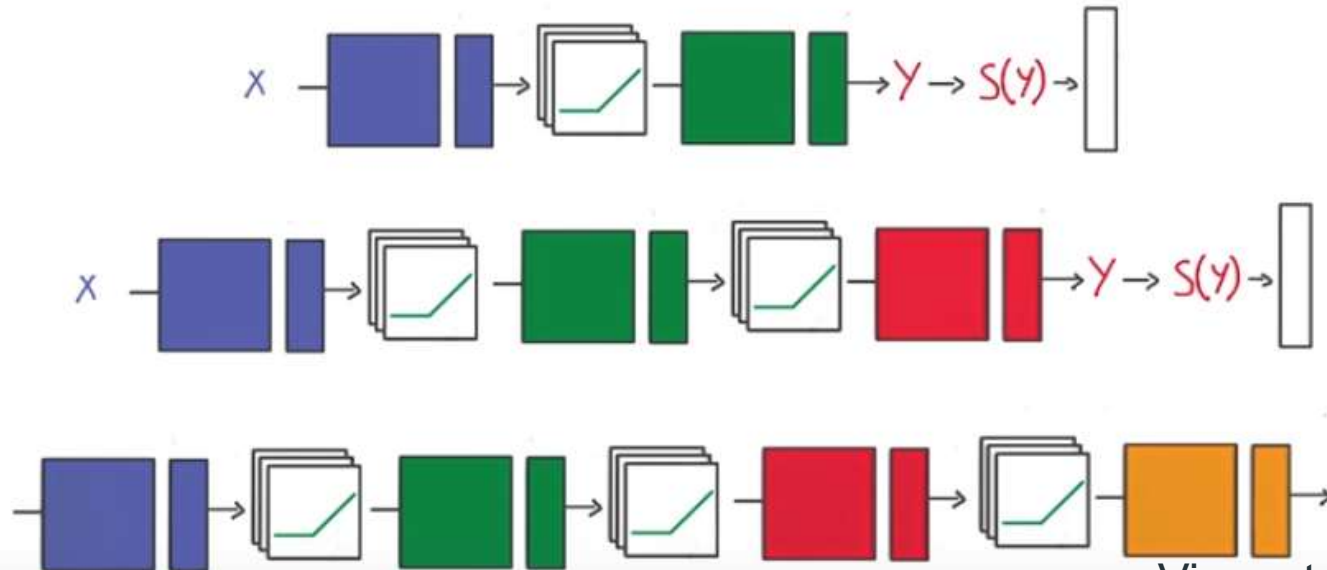
- Typically, many learning tasks are hierarchical
 - Identifying gene regulatory regions in the genome



Why were they not discovered before ?

- Lots of parameters to train : computational power and training data only recently available

DEEP NETWORKS

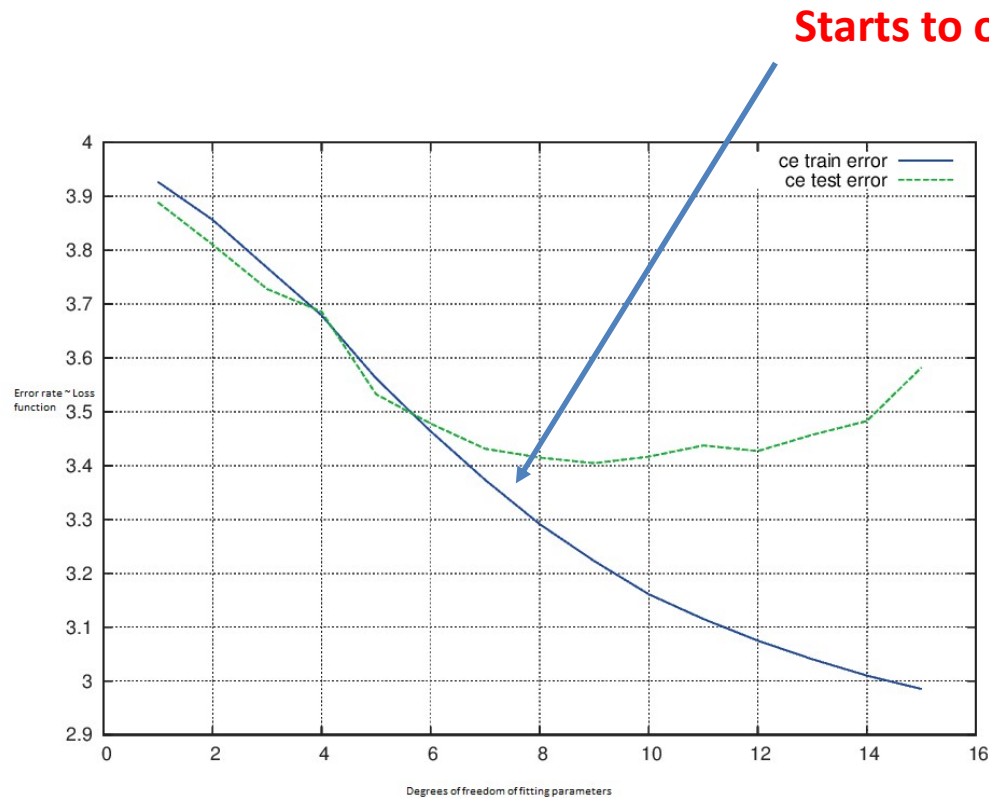


High dimensional models require Big Data

- Take an extreme case
 - More model parameters than training data points
- Perfectly mimic the response variable on the training set
- Fit to the idiosyncracies (noise) of the training set, and not to the underlying broad correlative patterns (signal) : overfitting

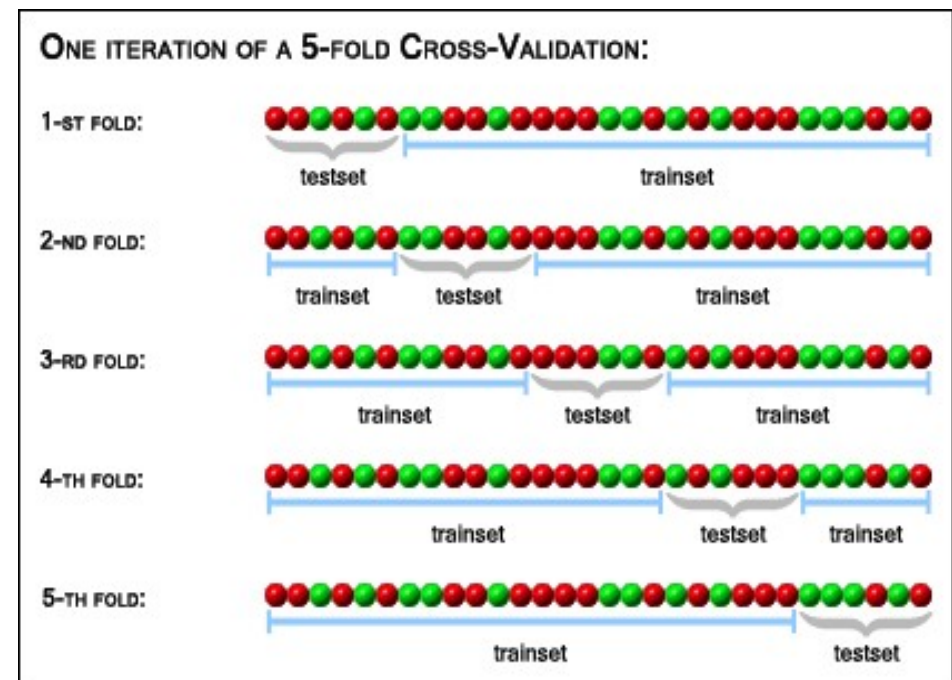
How can we recognize overfitting ?

- Differential performance on training and validation sets



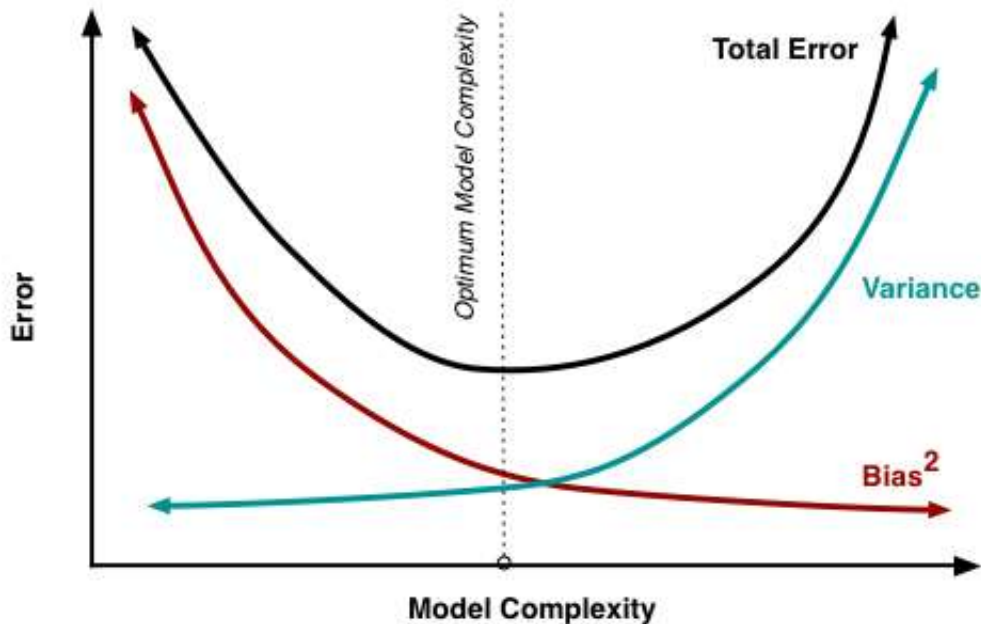
How can we avoid overfitting ?

- Model cross-trained and tested on multiple large, heterogeneous datasets carved out of the original dataset
- Requires a lot of data and a lot of computing power



The virtues of sparsity

- Fitting an optimal number of parameters can get us best results

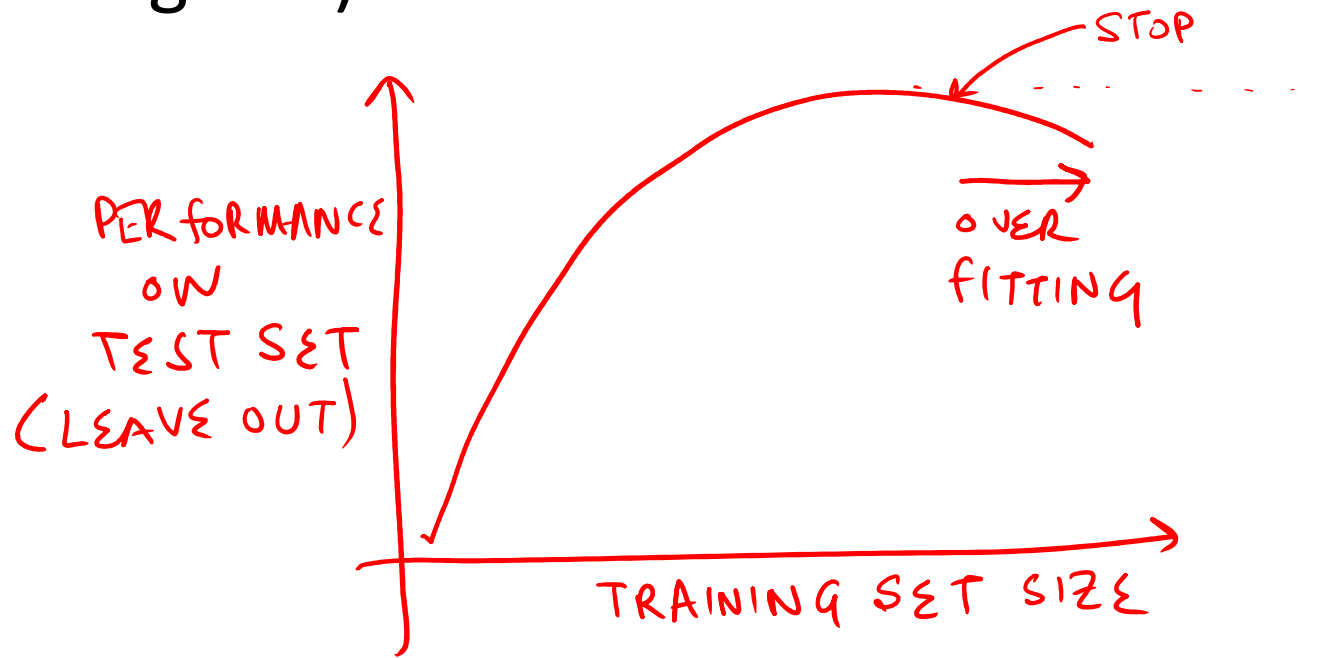


Scott Fortmann-Roe

- “Skinny jeans” problem : choose a jeans one size too large is easier to fit into than the perfect fit
- Choose a model slightly bigger than you need, and then reduce complexity

Making deep nets work

- Do not use more data than you have to : even with the right model complexity you need to guard against overfitting (overoptimizing on training set)



Making deep nets work

- Add a penalty term to the loss function to penalize complexity

$$d' = d + \beta \cdot \frac{1}{2} \|\omega\|_2^2$$

Scale factor of penalty term

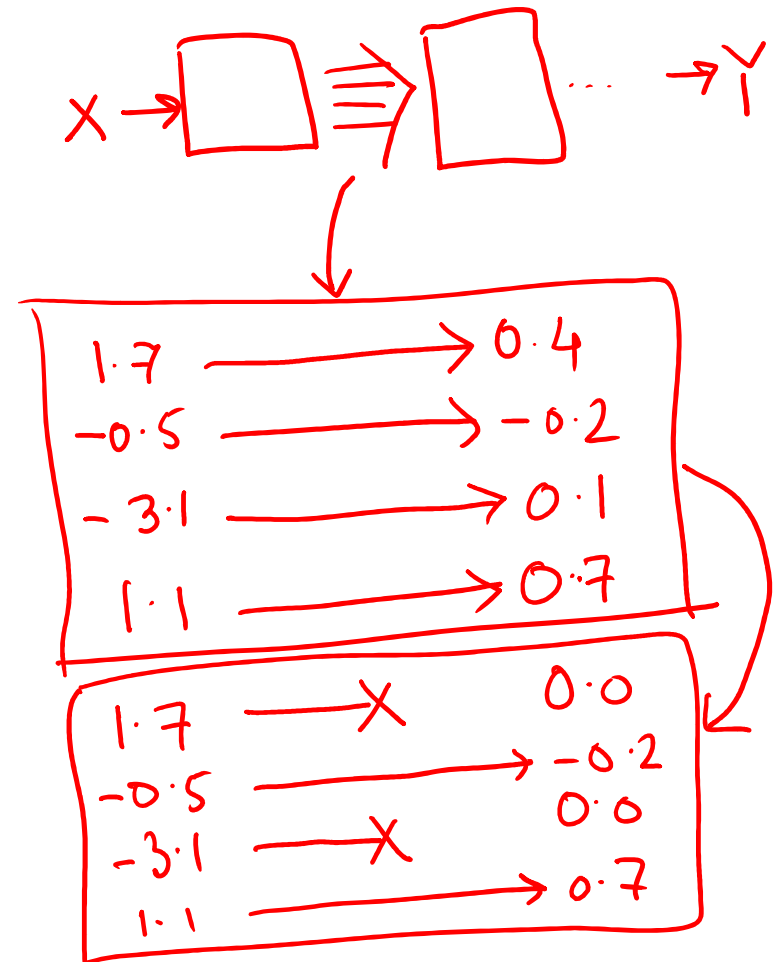
Penalty is L2 norm :
Penalizes based on
distance of vector
from origin

traditional
loss

Penalty for model
complexity (models
far from origin penalized)

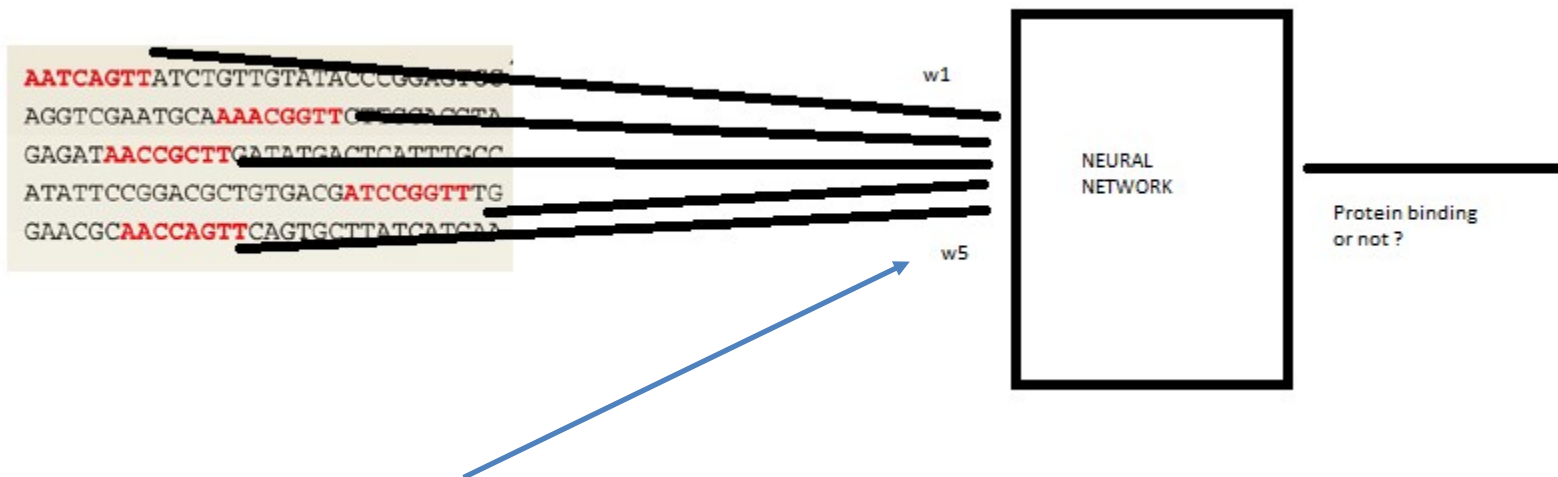
Making deep nets work

- Dropouts : “drop” (set to zero) a fraction (say, half) of the connections for each training example randomly
- Stops weights from “co-adapting” and breaks correlations between parameters



Making deep nets work

- Weight sharing : should the classifier treat some parameters equally ?



If the binding of protein to DNA is invariant of the exact genomic position of the motif, weights w_1, w_2, \dots, w_5 can be combined such that $w_1 = w_2 = \dots = w_5 = w$

Model simplification based on invariance properties !

21st century Biology : high throughput assays

- Low throughput assays : eg. qPCR – queries a trait (in this case mRNA abundance) for one or a few locations in the genome / epigenome / transcriptome / proteome
- High throughput assays : eg. RNA-seq – queries a trait (in this case mRNA abundance) for many or all locations in the genome / epigenome / transcriptome / proteome **simultaneously**

How many high throughput dimensions ?

- One
 - RNA-seq : transcriptional level across every gene in the genome

$x_1, x_2, x_3, \dots, x_N$: for large N

- Two
 - Hi-C : For every pair of sites in the genome, characterize spatial proximity between the sites (both high throughput dimensions : genome)
 - Single cell RNA-seq : For every cell and every gene, characterize the transcriptional level in that gene for that cell (high throughput dimensions : cells and transcriptome)

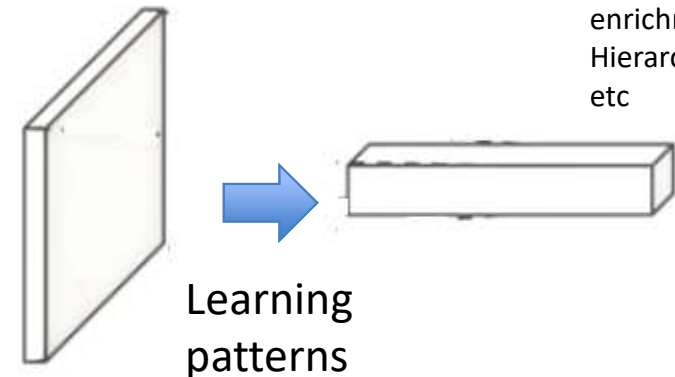
$x_{11}, x_{12}, x_{13}, \dots, x_{1N}$: for large N

$x_{11}, x_{21}, x_{31}, \dots, x_{M1}$: for large M

What do we want to learn from such datasets ?

- Are some pathways / processes enriched ?
- Is there some typical correlation with a previously unrelated trait ?
- So, a set of structured outputs is desired which is common to the entire dataset(s)

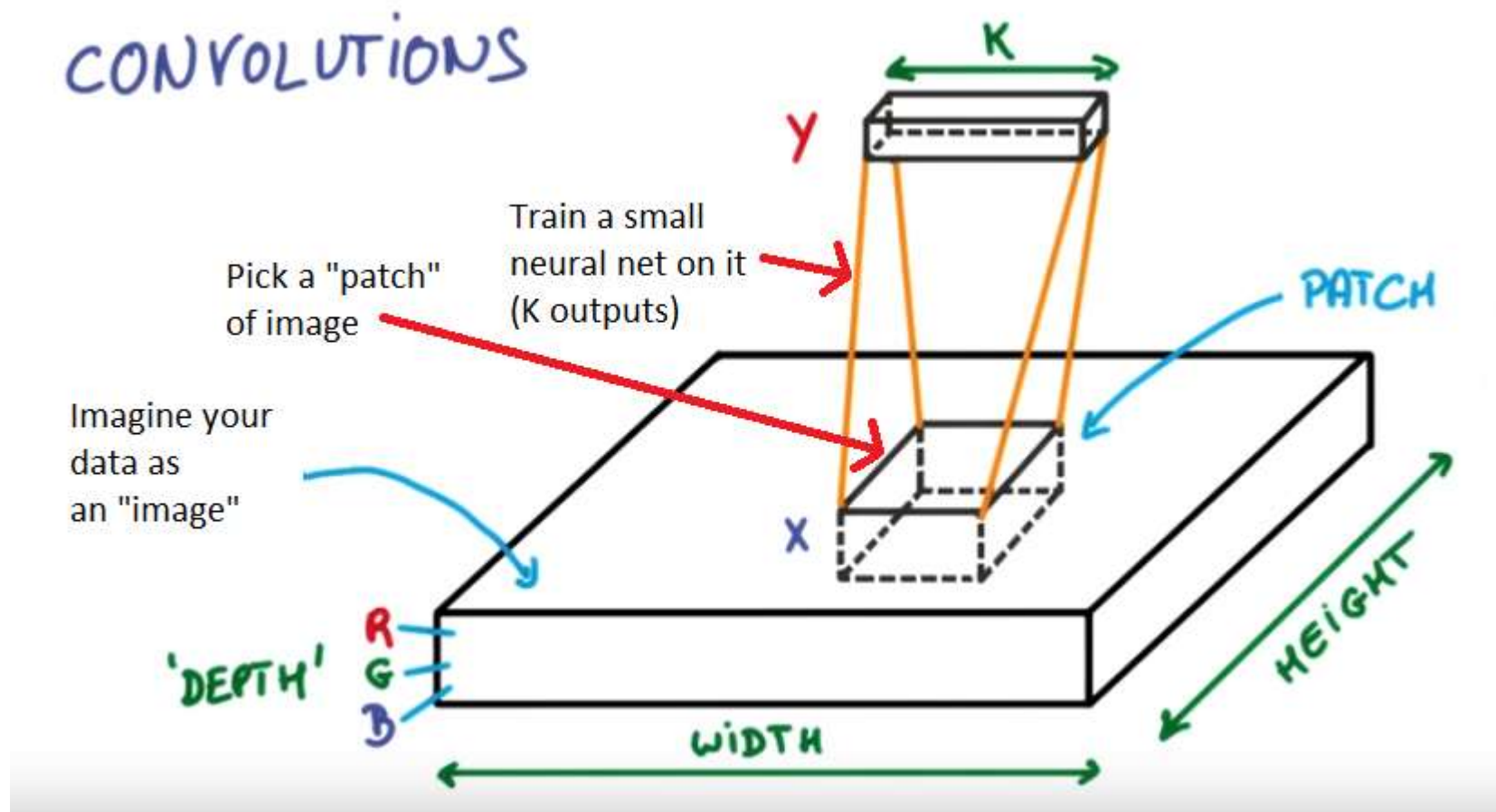
Data : Height and width – high throughput
Dimensions
Depth – low throughput dimensions (healthy
Vs normal)



Data

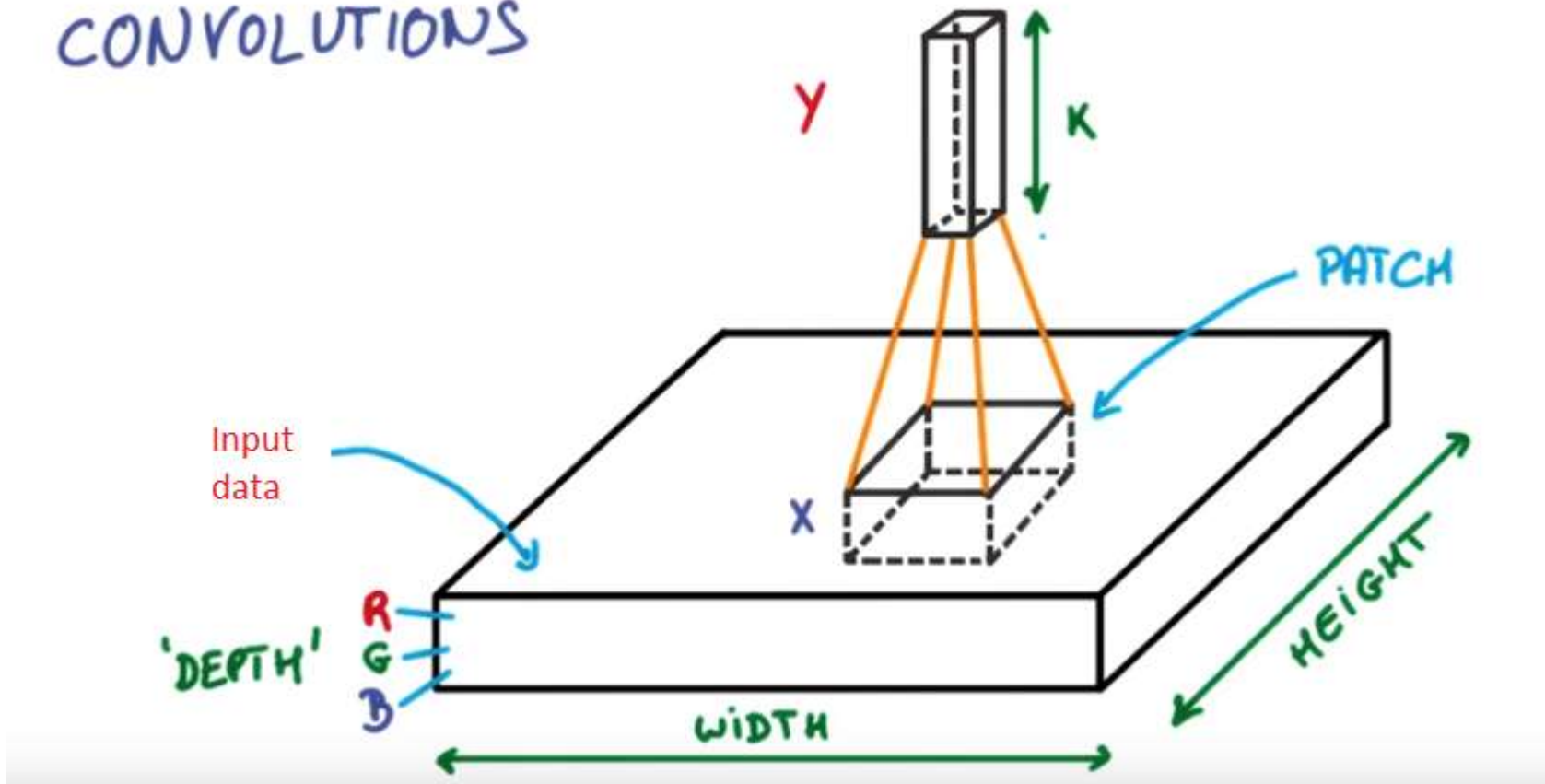
Knowledge

“Mix up” a small contiguous part of the dataset by training a neural net on it



An equivalent representation

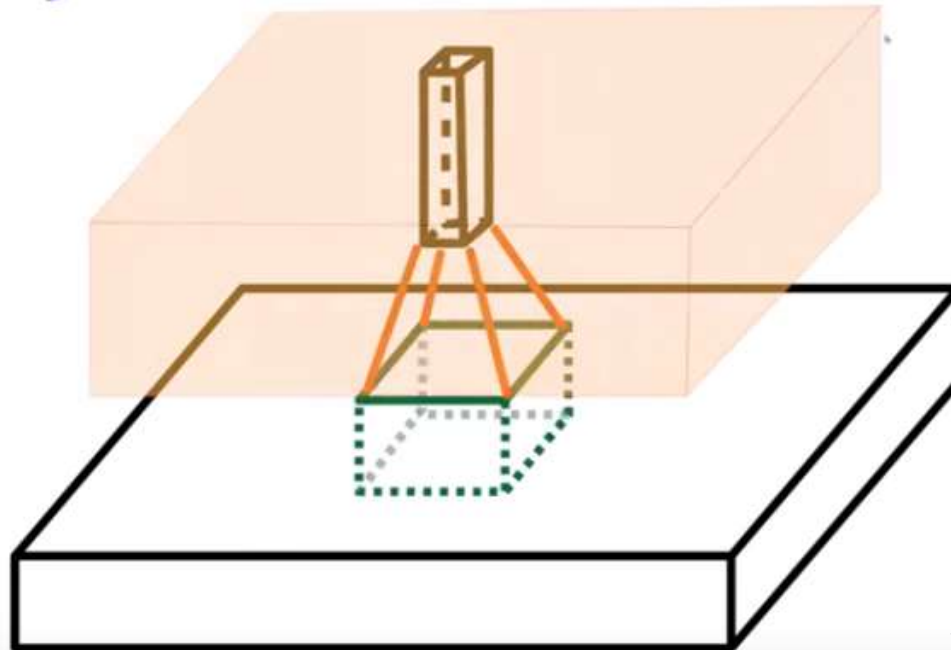
CONVOLUTIONS



Create an image from NN outputs

- Slide the NN across the image, and assemble the outputs to form an “output image”

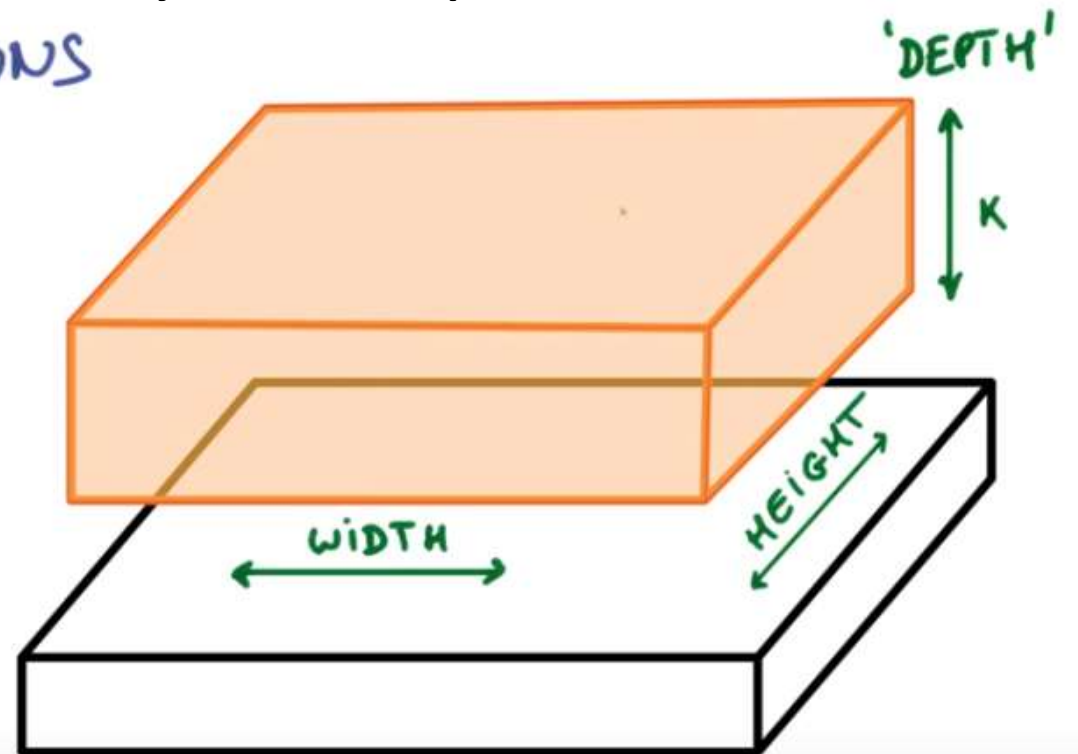
CONVOLUTIONS



What is the size of the output ?

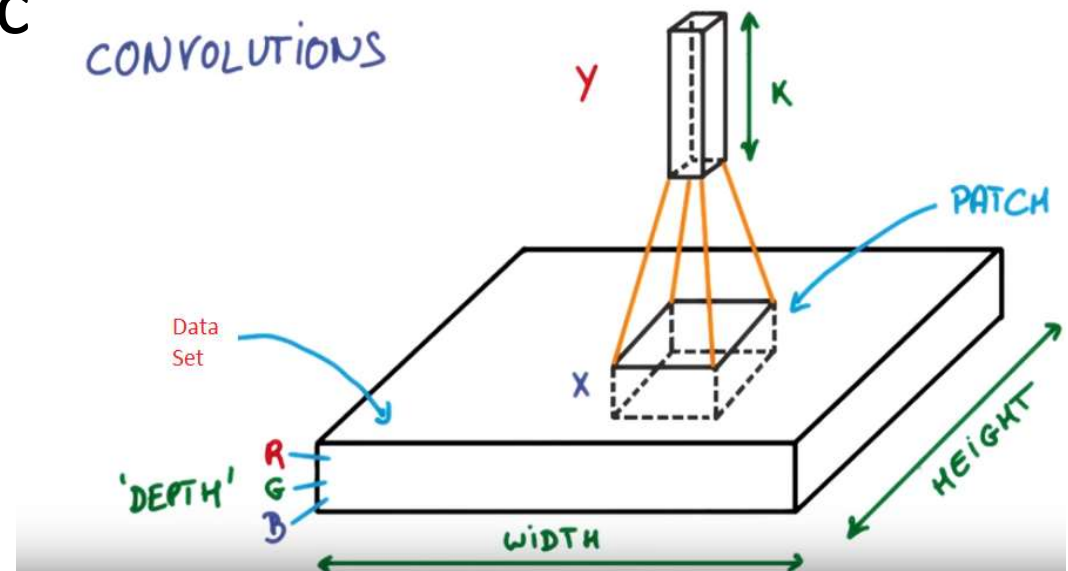
- Depth = k (output of neural net)
- Height and width = Height and width of original image scaled by the step-size of the sliding *CONVOLUTIONS*

Entire operation
= "convolution"



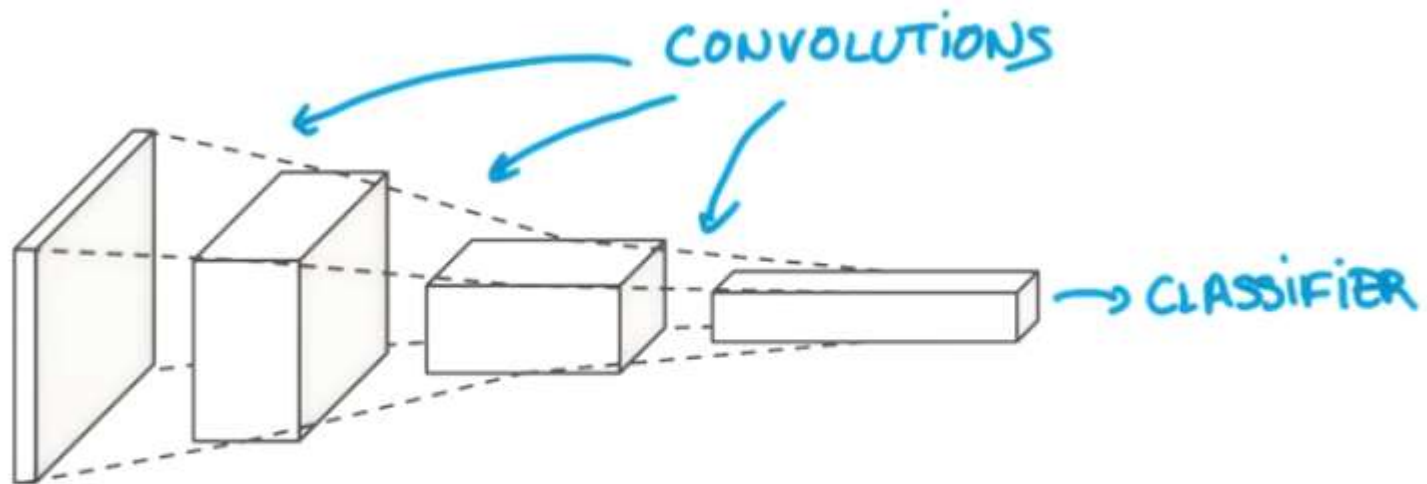
Patch size

- Patch size = 1×1 : called a 1×1 convolution
- Patch size = image size : equivalent to a layer of a traditional neural network
- Weights are shared across the image (parameters specific to a region of the image will not be inferred)



Stacking convolutions : the “deep” part of it

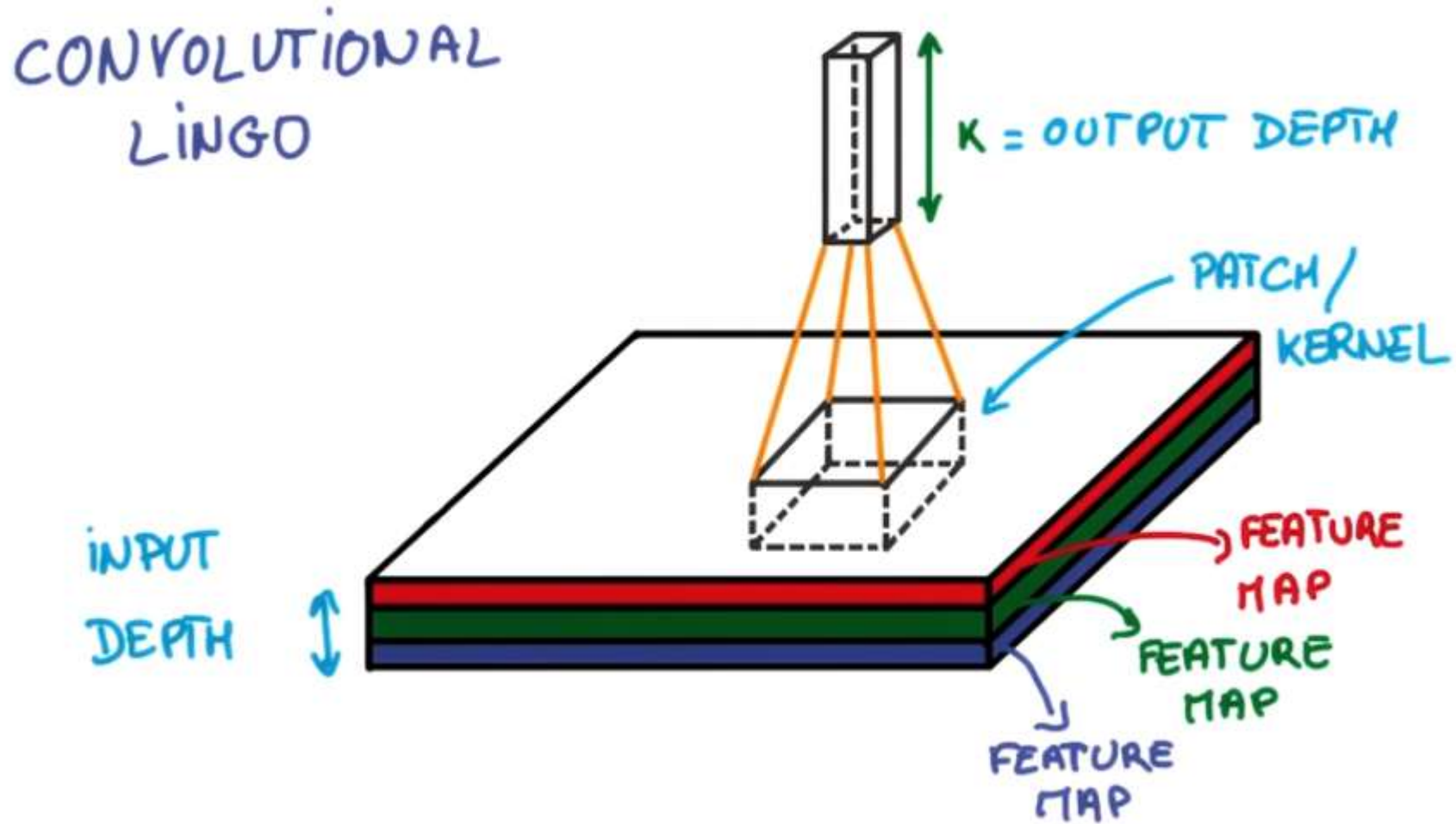
CONVOLUTIONAL PYRAMID



256x256 X 3 → 128x128x16 → 64x64x64 → 32x32x256 ...

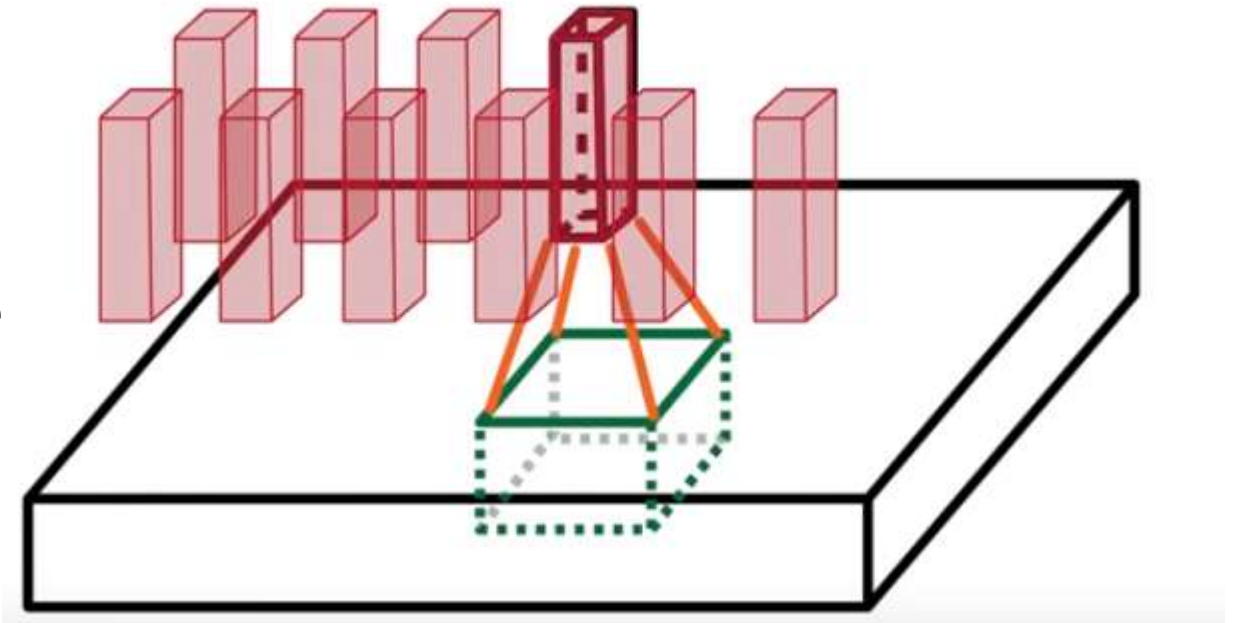
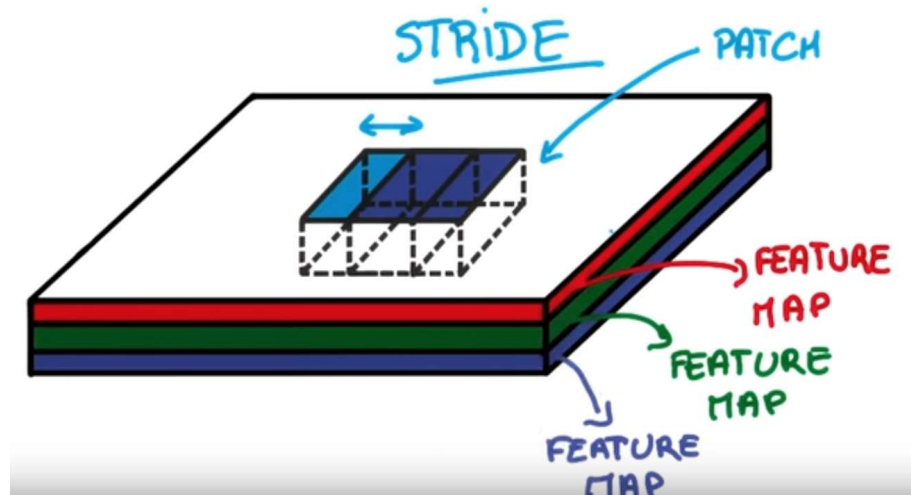
SPACE → DEPTH = SEMANTIC REPRESENTATION

Increase depth (extracting info),
decrease space (generalize) at each
step



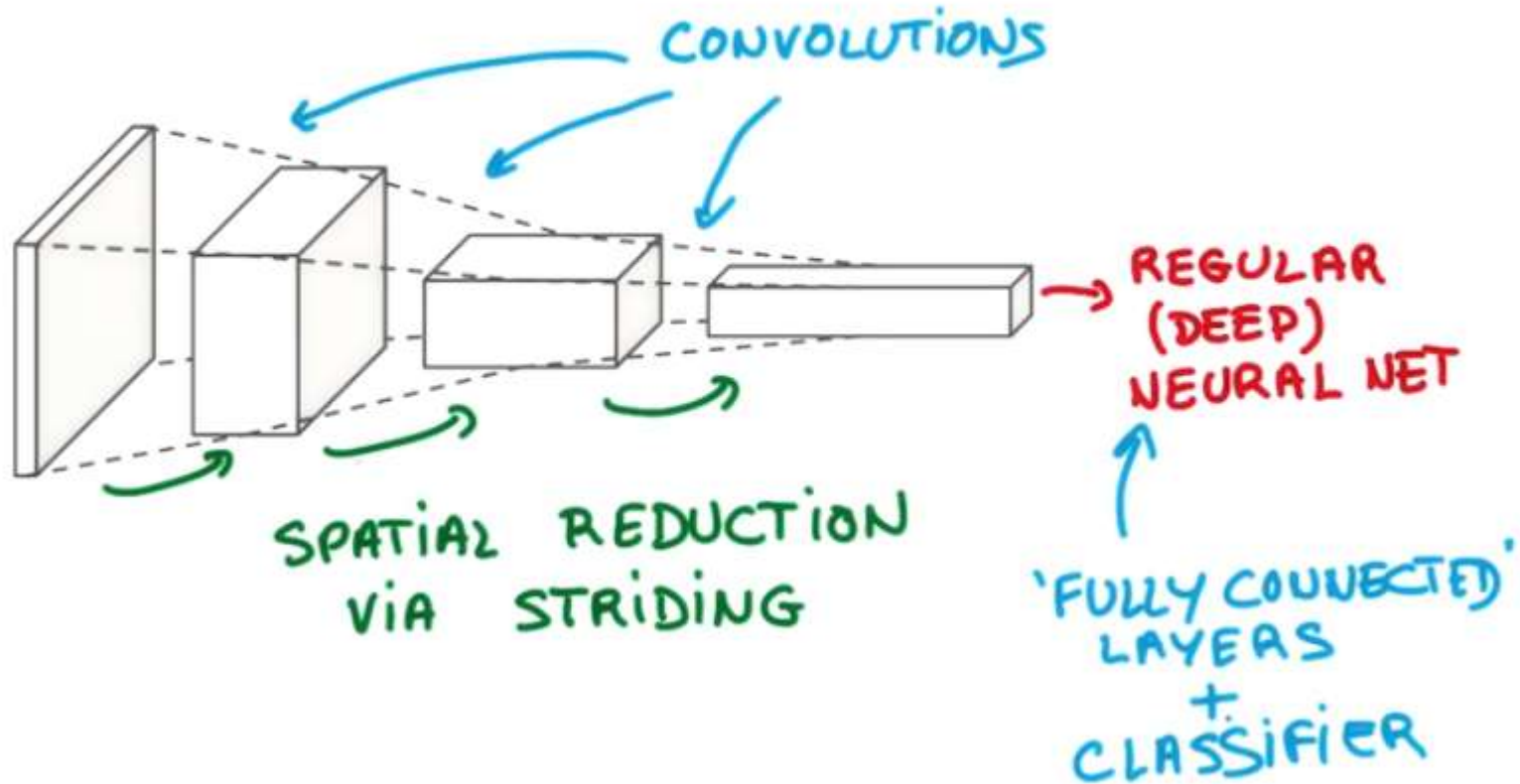
Stride

- Stride : resolution of analysis
Step size of the convolution as the NN marches across image
- Paddings on the side



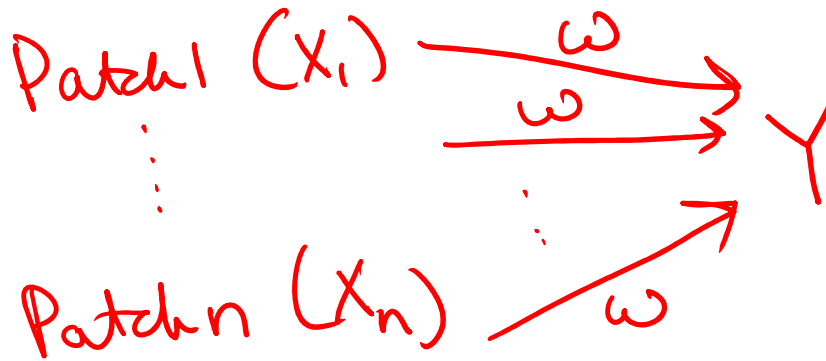
Hooking up conv-nets with regular deep nets

CONVOLUTIONAL NETWORK



How to train the neural net ?

- How sharing works : works out of the box



$$\frac{\Delta \mathcal{L}}{\Delta \omega} = \frac{\Delta \mathcal{L}}{\Delta \omega} (\text{patch}_1) + \dots + \frac{\Delta \mathcal{L}}{\Delta \omega} (\text{patch}_n)$$

↑ add gradients from every patch

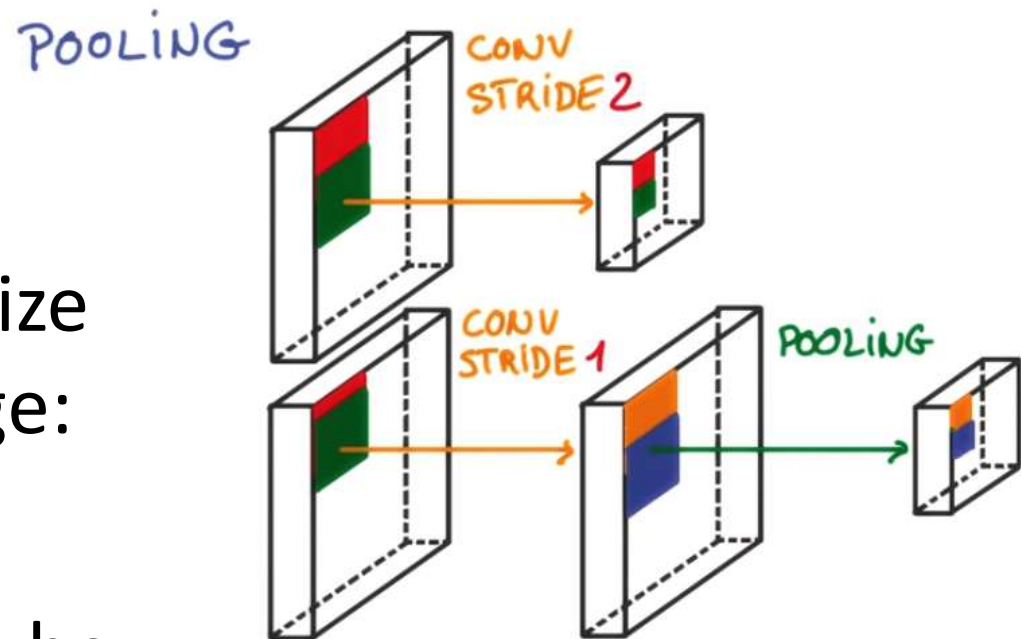
Reducing size at each step

- Increase stride / Step size
- Or, use a second stage to summarize parts of the image:

Pooling

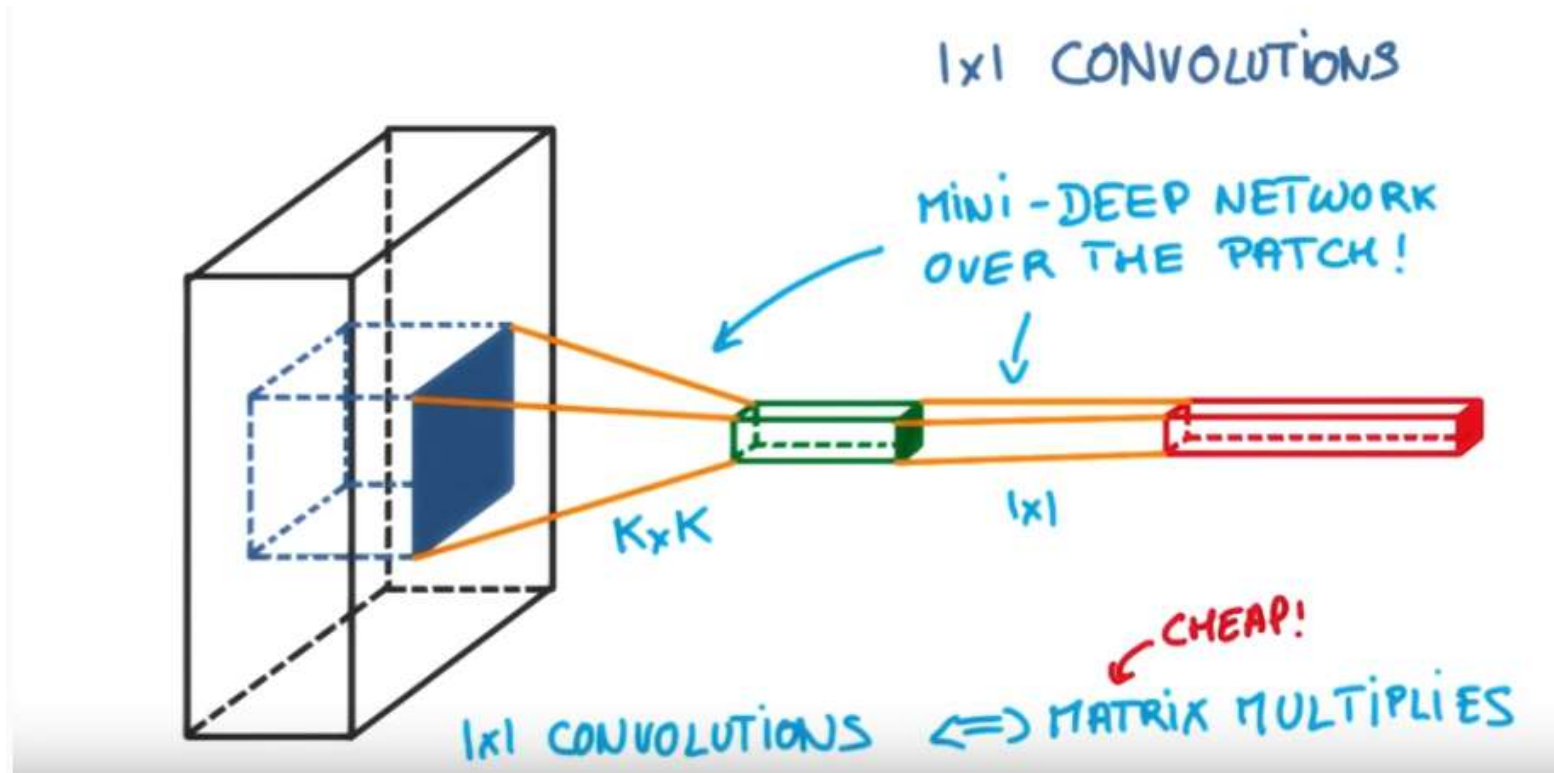
Summarization can be

Max, Average, etc



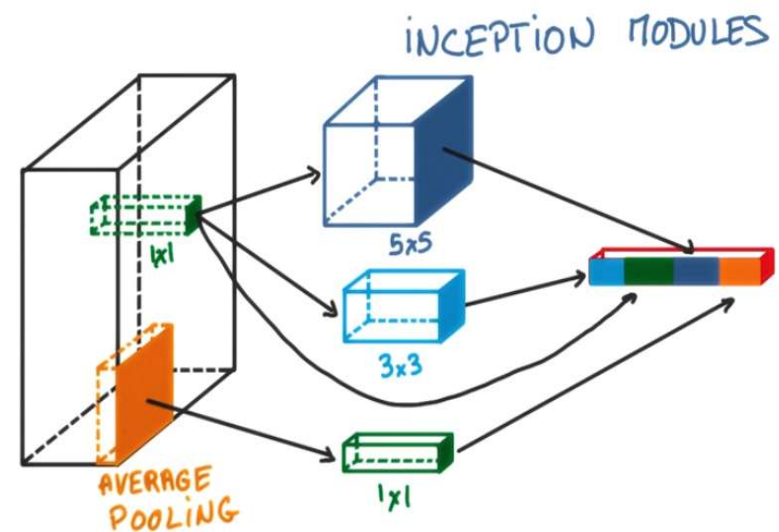
Why use 1X1 convolutions in a convolution stack ?

- Converts a linear classifier into a neural network : without too many new parameters



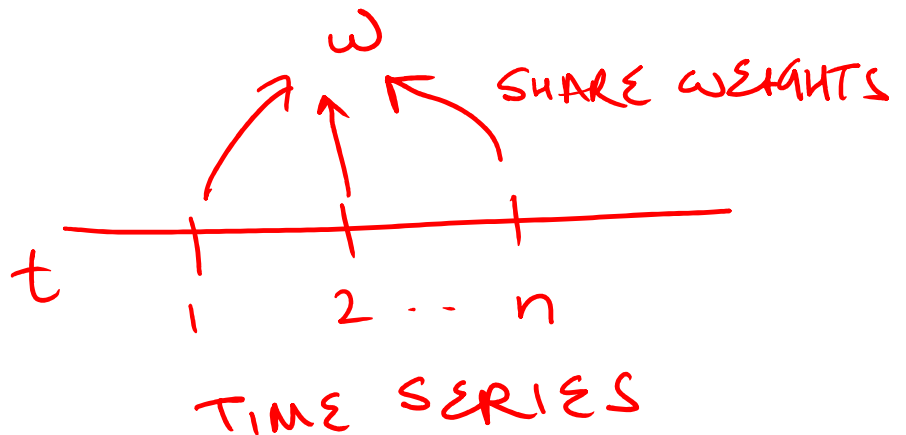
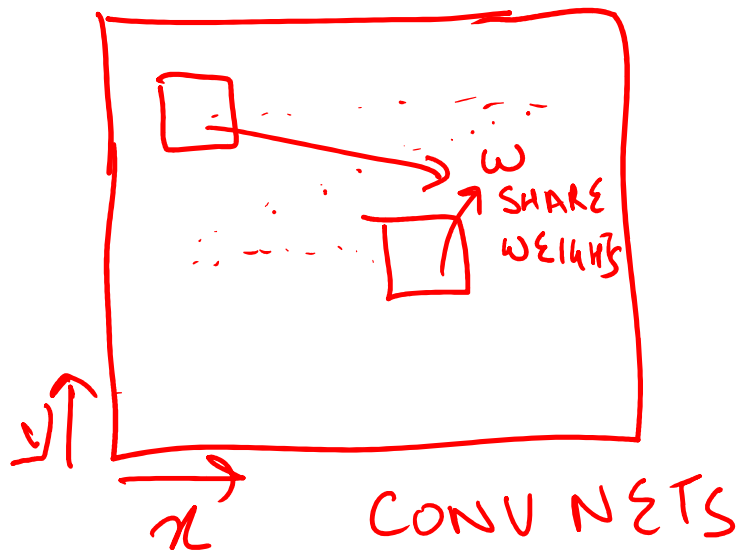
Inception modules

- At each step of the pyramid, a choice of using pooling or convolution (and of what size) ?
- You can hedge and have it all
 - Performance is better than using an individual step (different information captured by different convolutions)

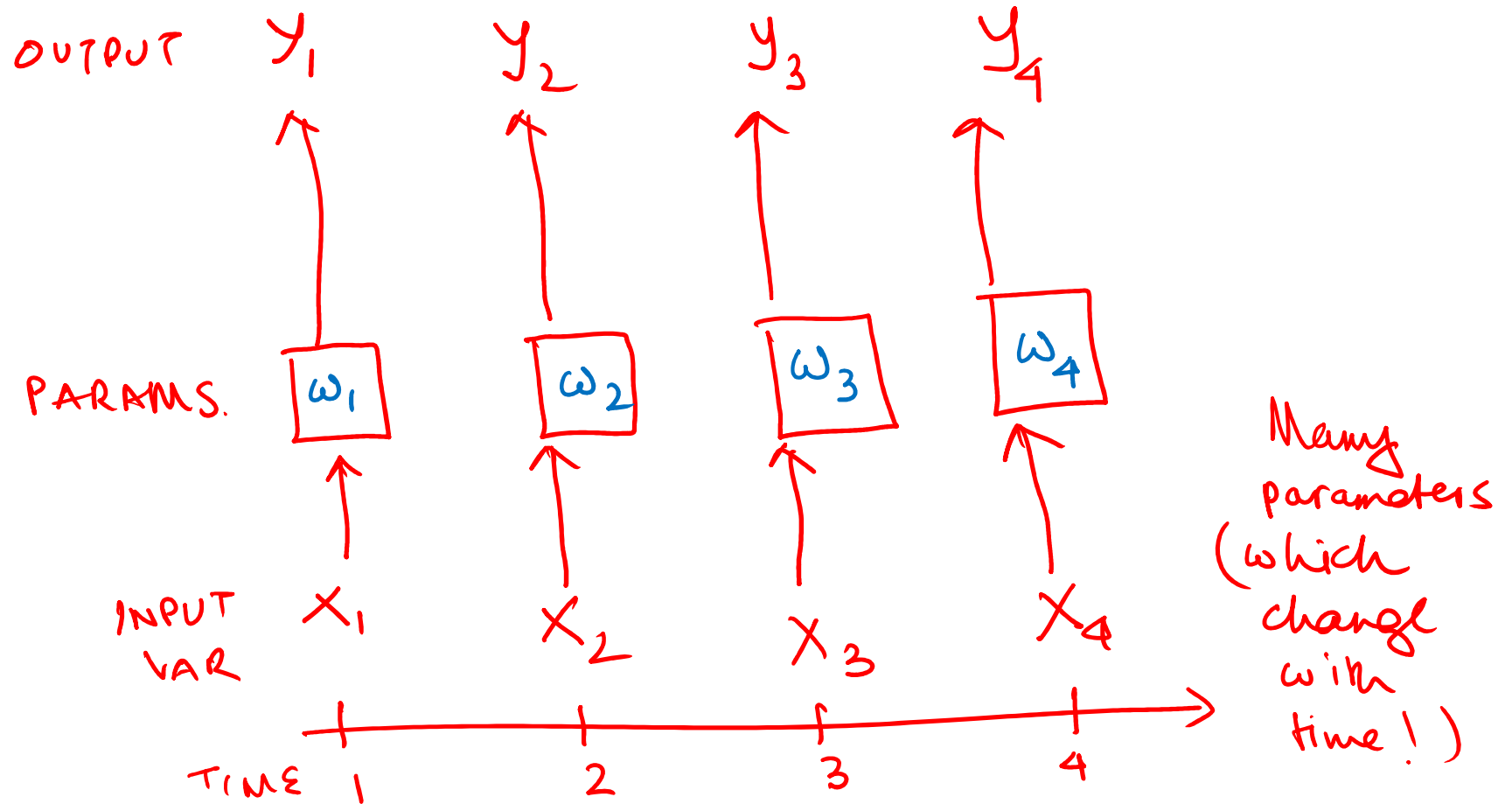


Time series data

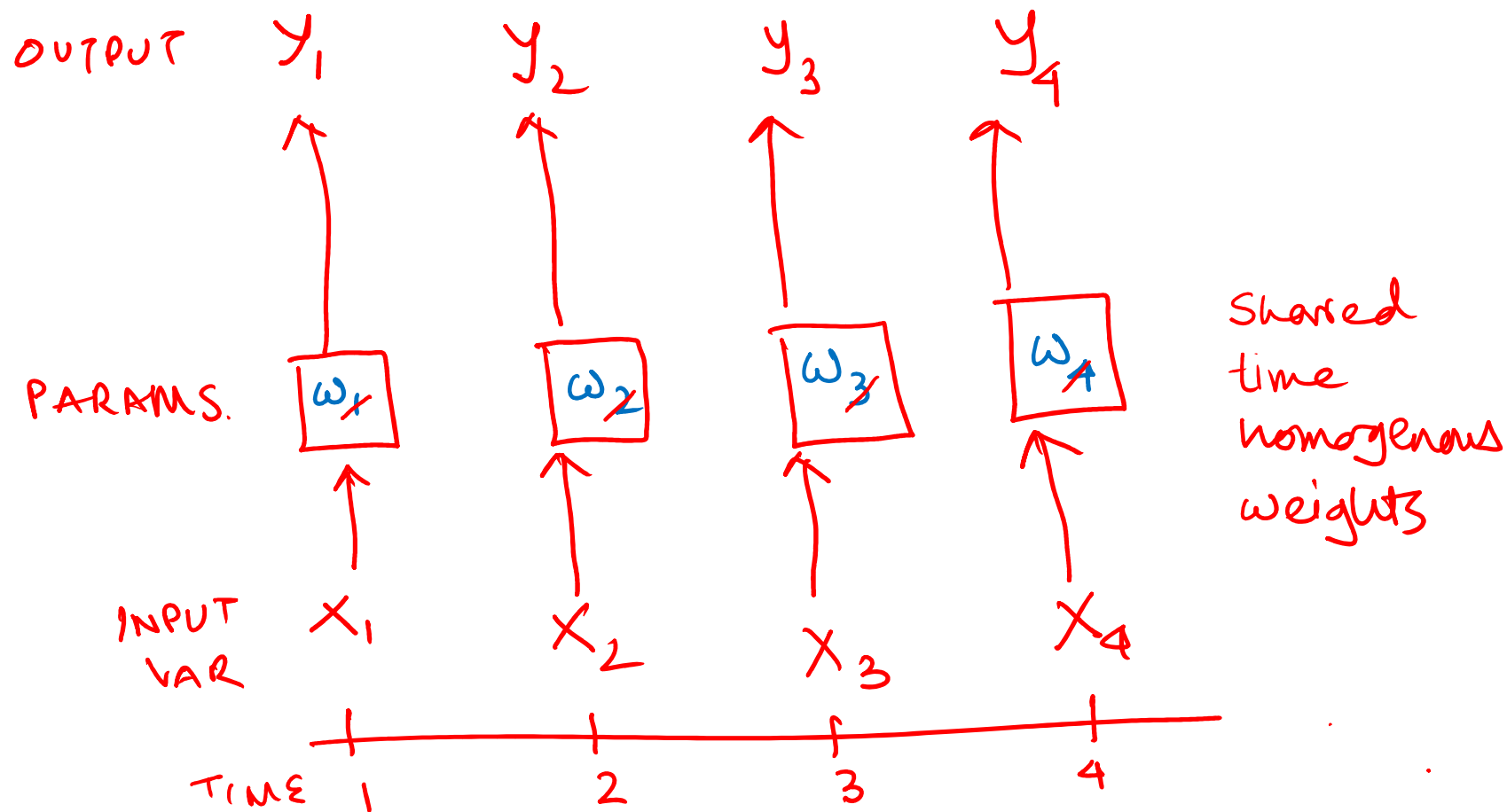
- Weights shared across patches (hence across all dimensions)
- For time homogeneous models, we can share parameters across time dimension



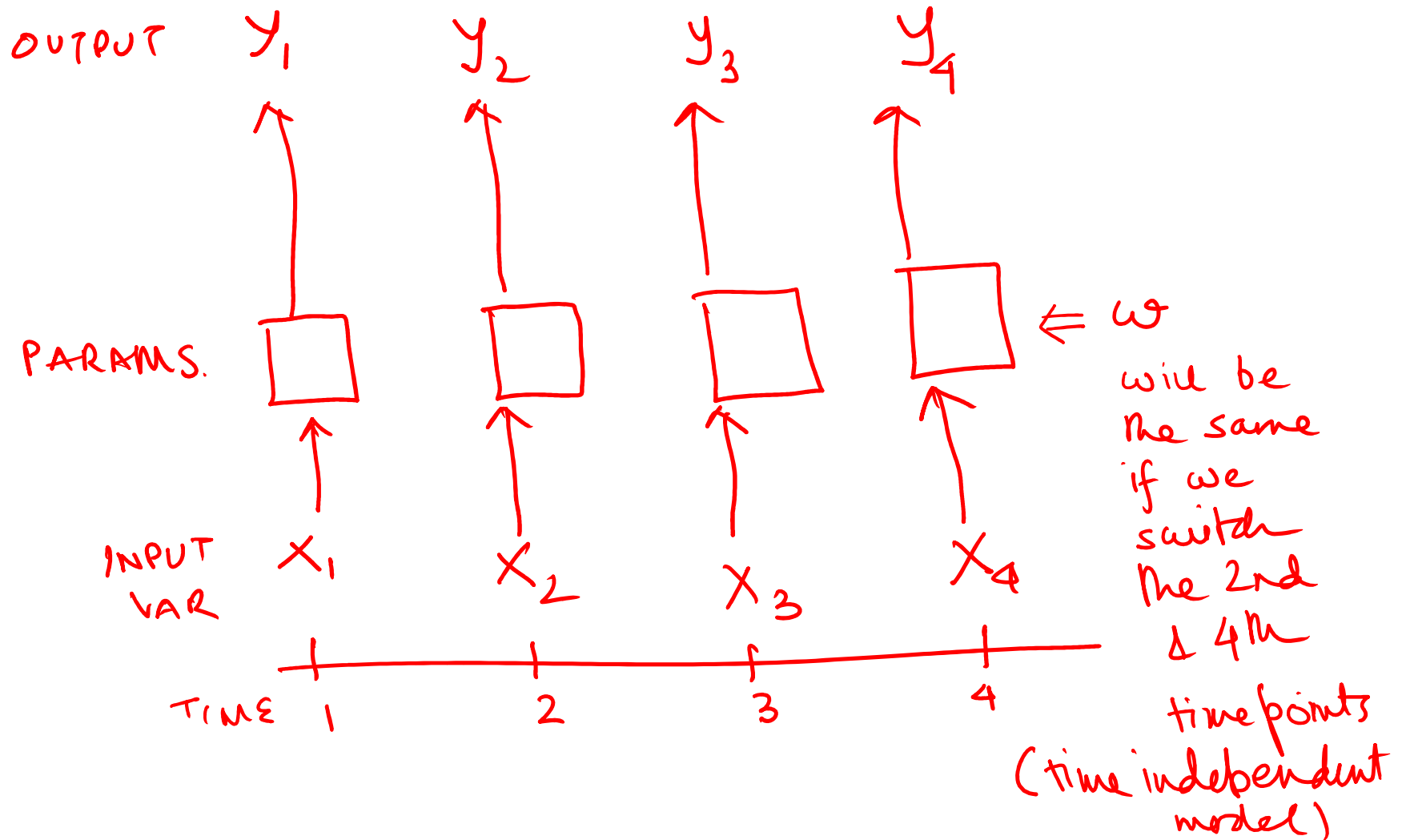
Neural networks with temporal data



Taking advantage of time homogeneity



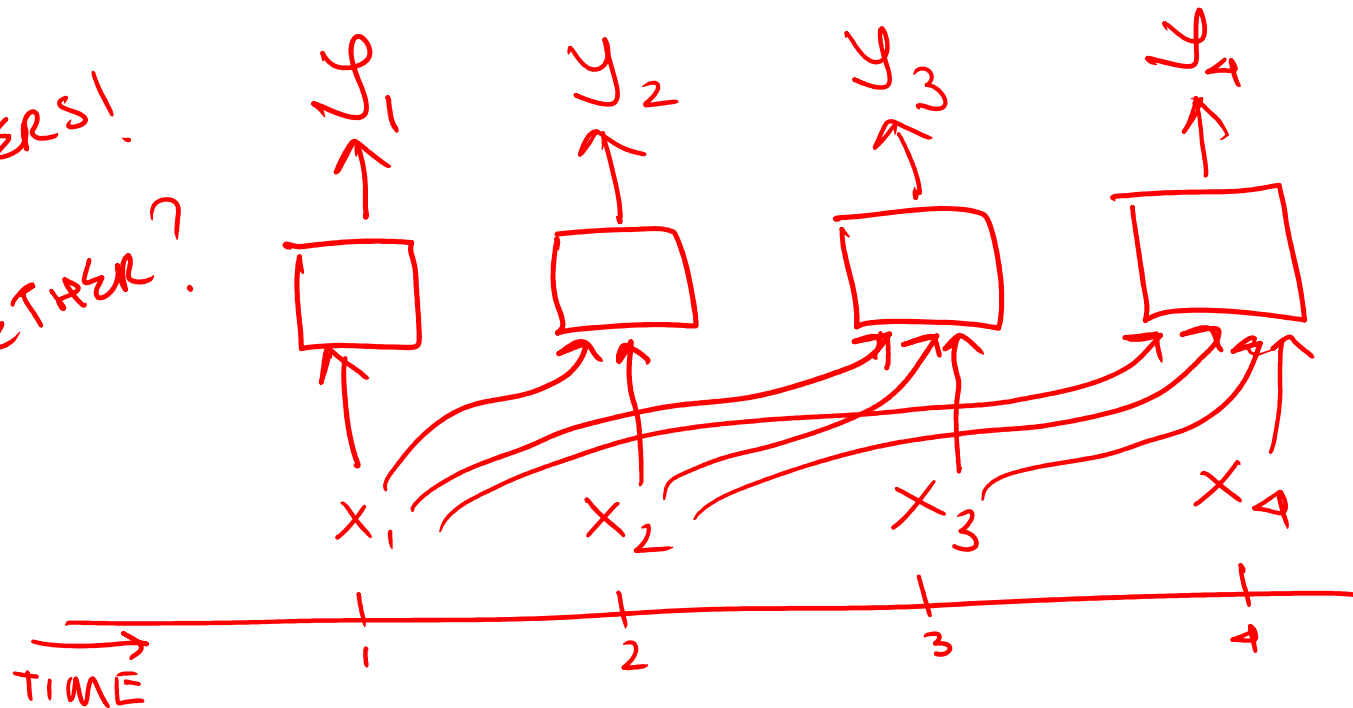
Time homogeneity and time agnosticism



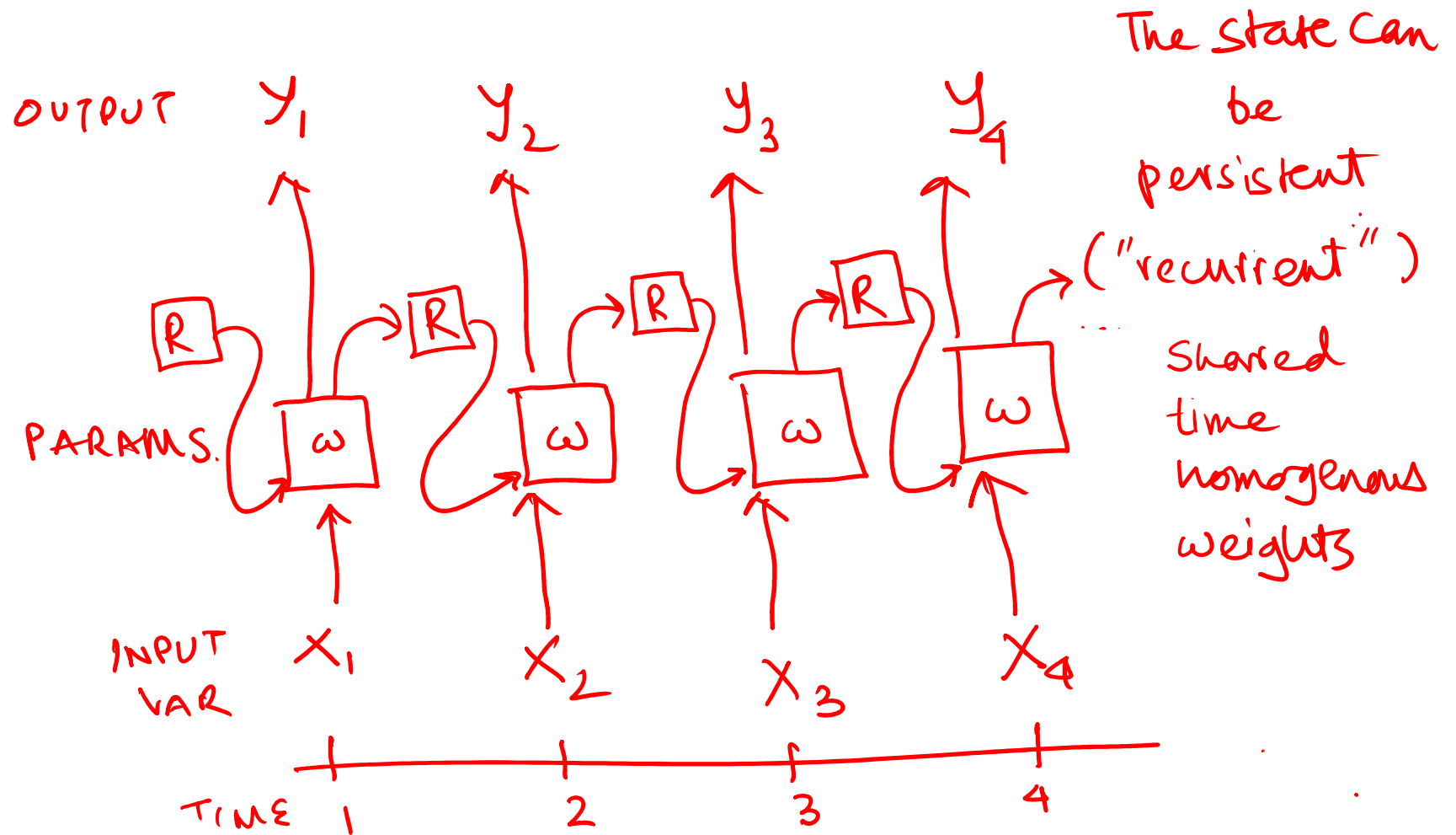
Using history

- What if all the previous timesteps can be fed in at time t ?

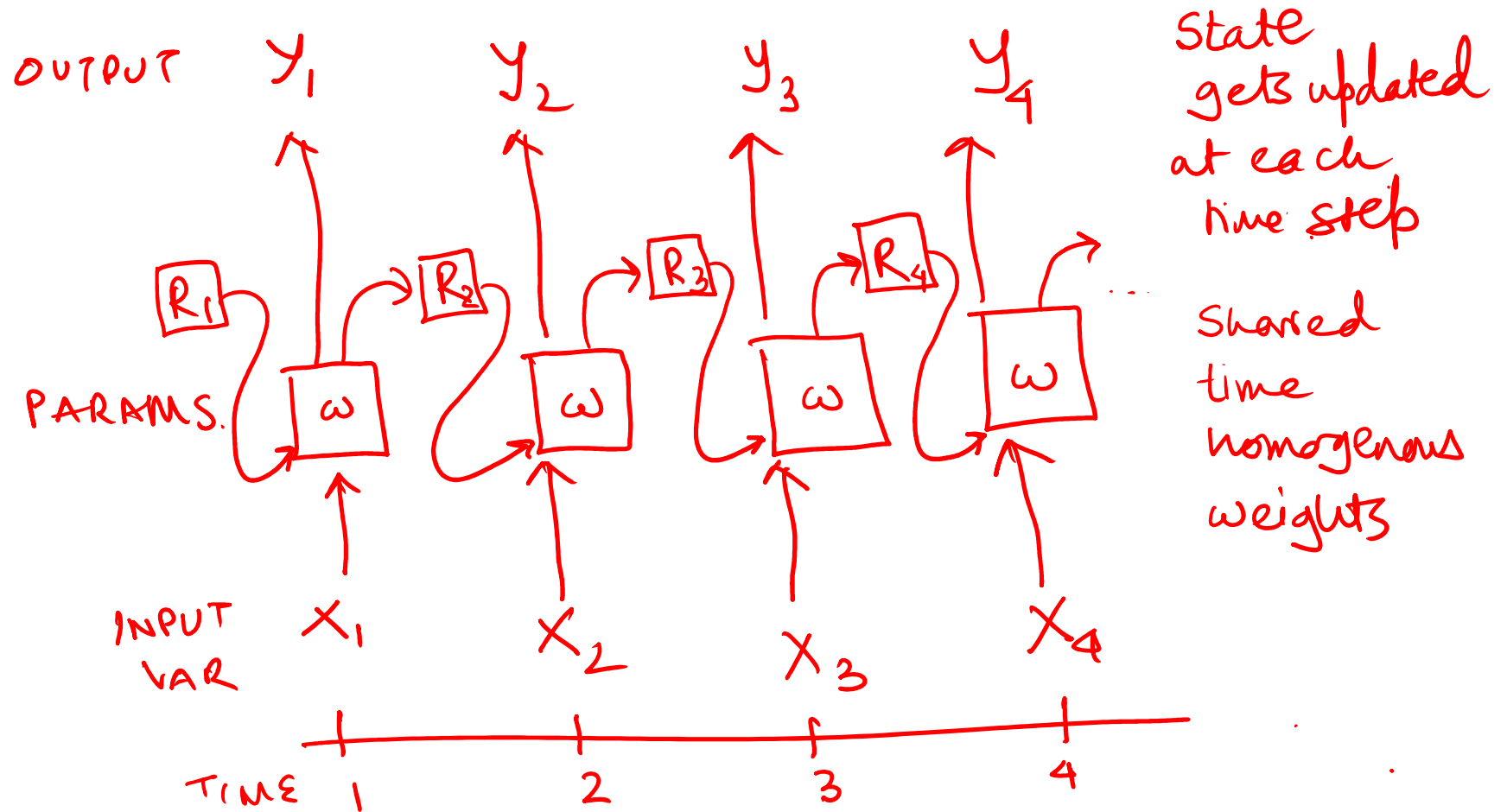
Too MANY
PARAMETERS!
WHICH TO
SHARE TOGETHER?



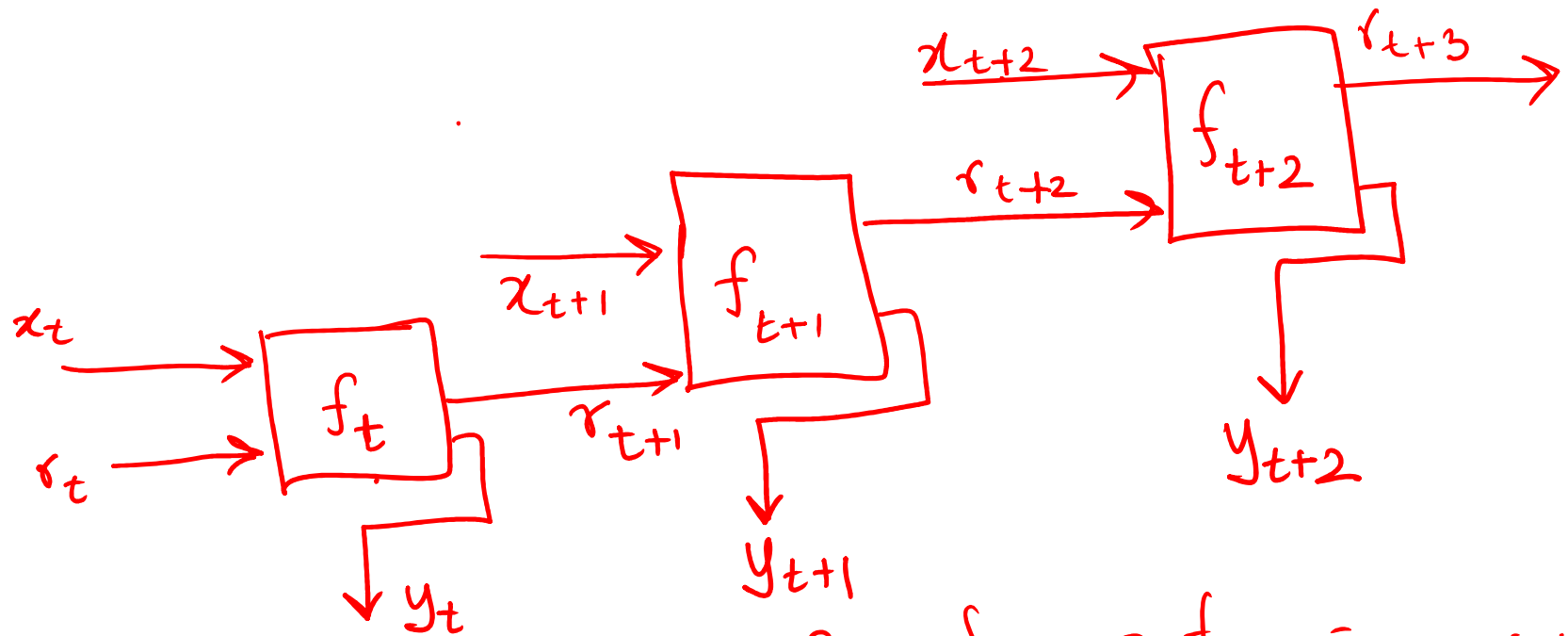
Designing an internal state



Designing an internal state



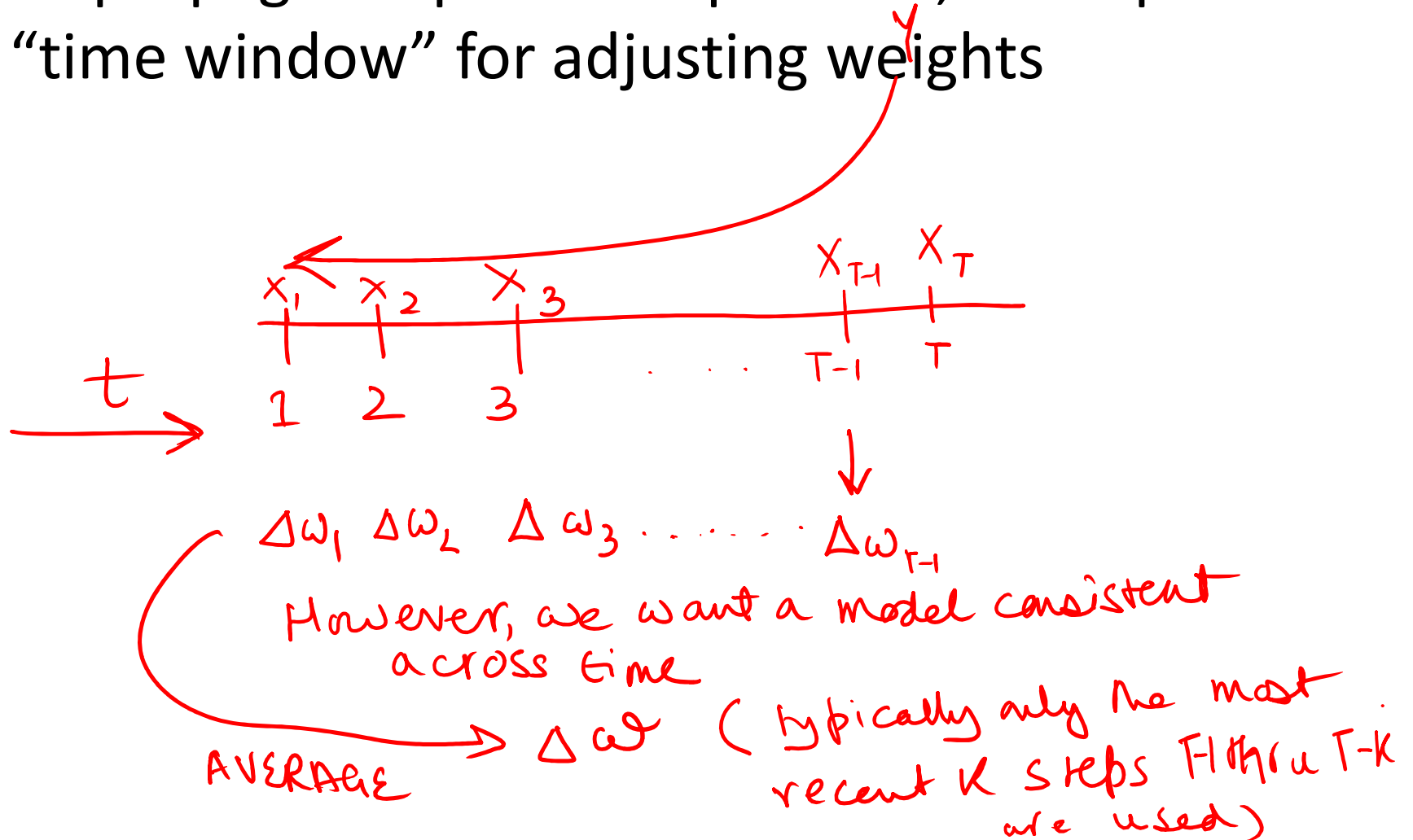
Linking input and output : unfolding in time



To share weights, $f_t = f_{t+1} = f_{t+2} = \dots$
How to perform backprop?

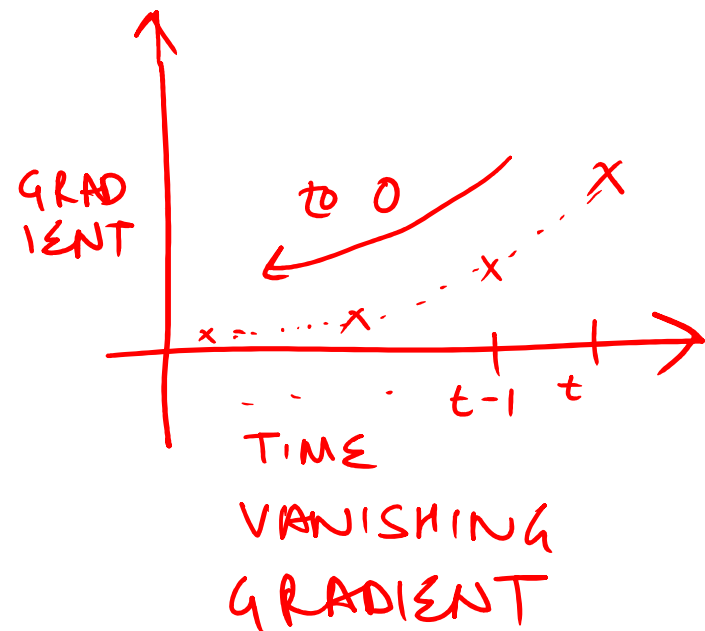
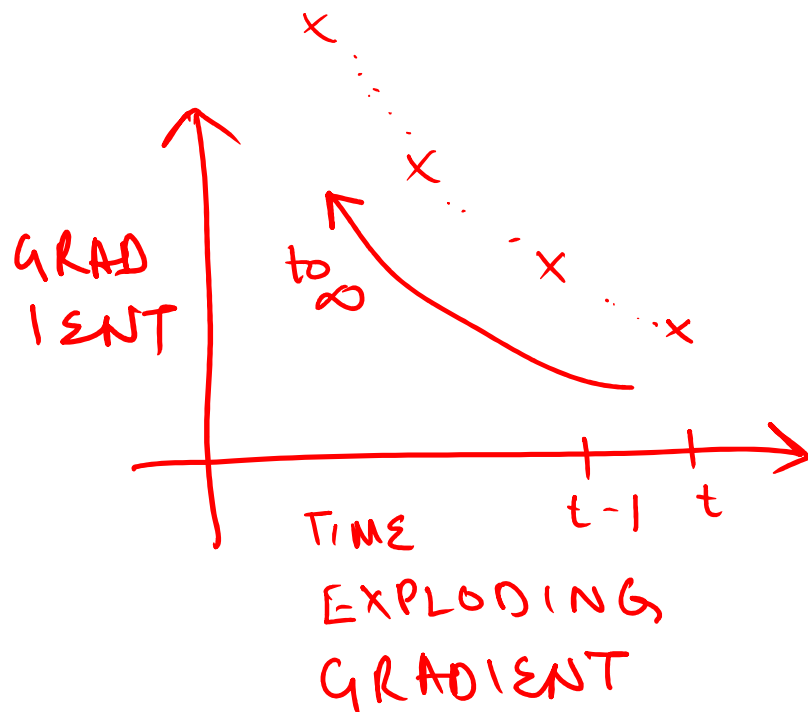
Backprop through time

- Backpropagate upto $t=0$ if possible, else upto a “time window” for adjusting weights

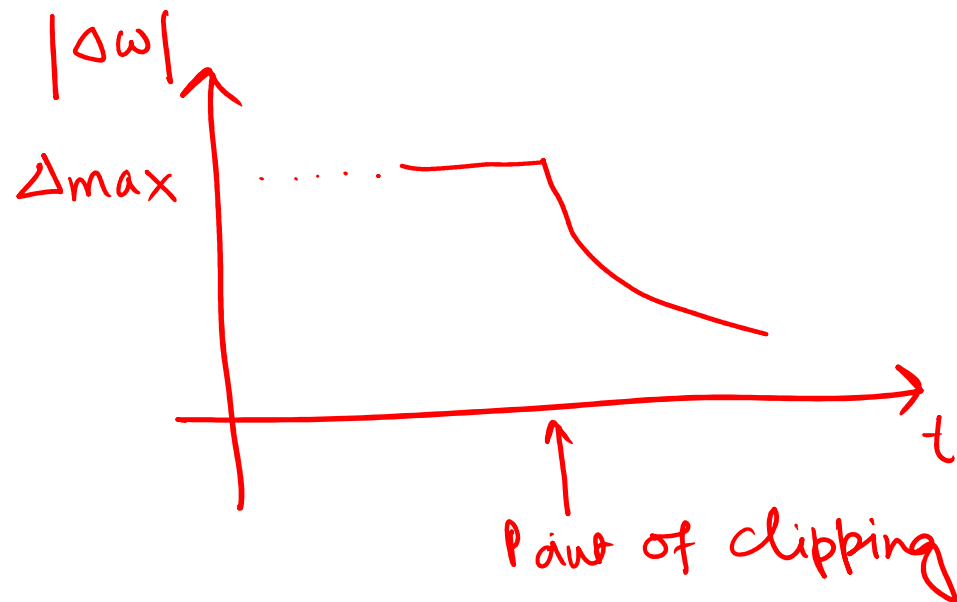


Dealing with feedback

- Standard feedback issues
 - Accentuation and dampening of weights



Exploding gradient : gradient clipping



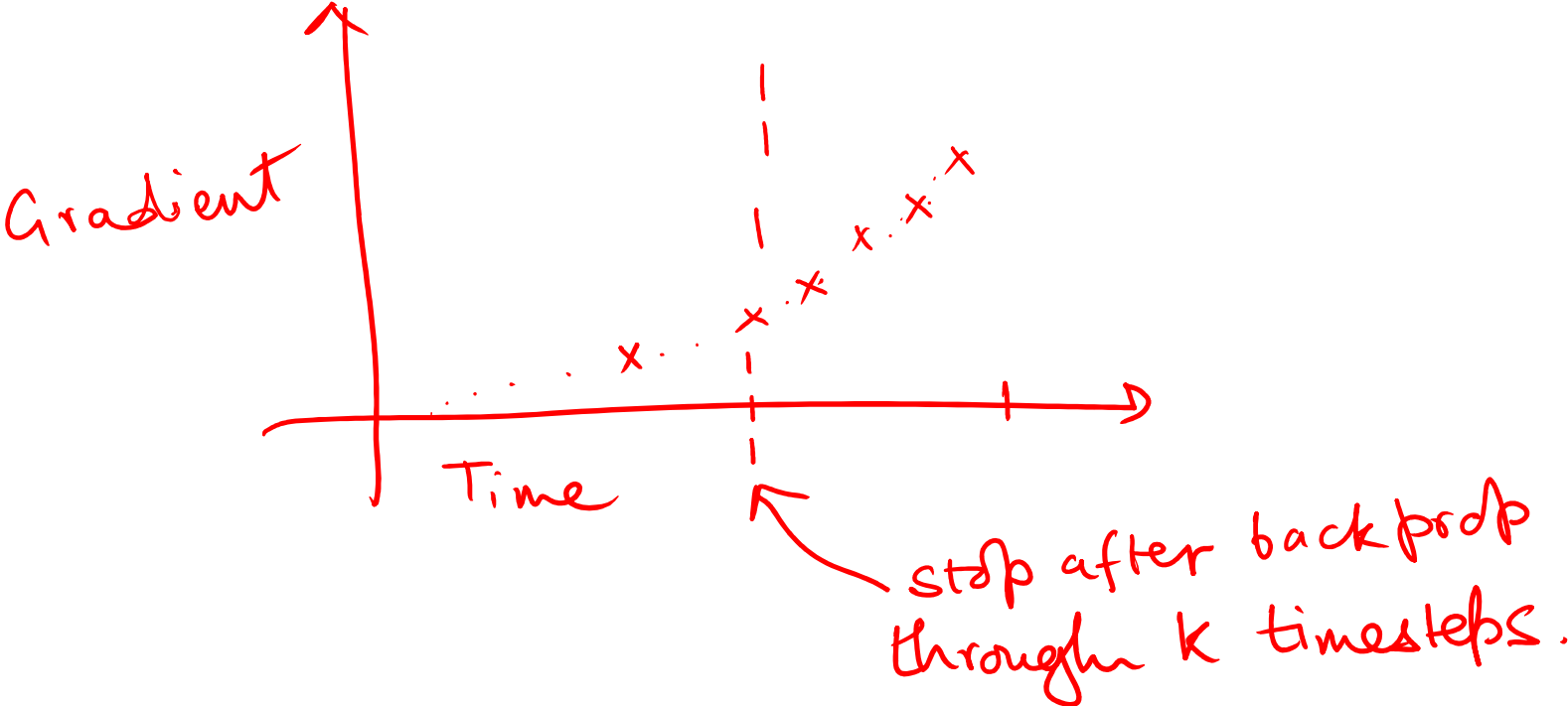
if $|\Delta\omega| > \Delta_{max}$

clipped $\Delta\omega$
 $= \text{sign}(\Delta\omega)$
 $\times \Delta_{max}$

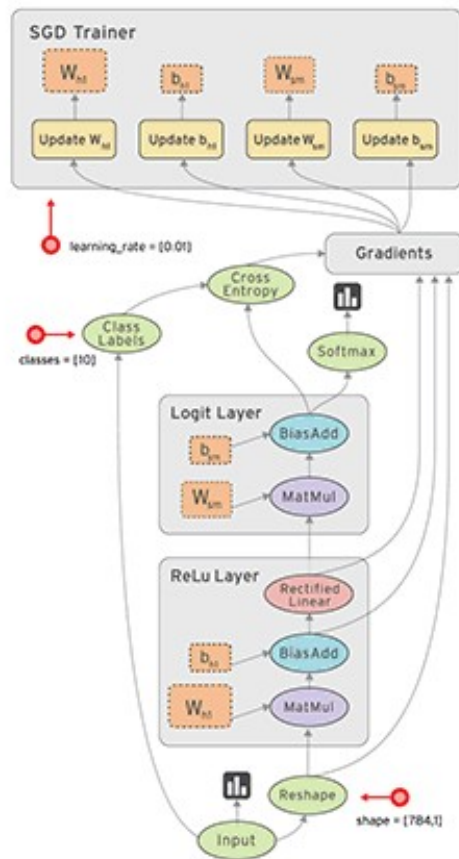
else
no clipping

Remember, $|\Delta\omega|$ is bounded, not $|\omega|$!

Vanishing gradient : limited backprop



What's out there



TensorFlow, Google's Deep Learning framework (open source)

Theano 0.8.0 documentation

next | modules | index

Welcome

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

- **tight integration with NumPy** - Use `numpy.ndarray` in Theano-compiled functions.
- **transparent use of a GPU** - Perform data-intensive calculations up to 140x faster than with CPU (float32 only)
- **efficient symbolic differentiation** - Theano does your derivatives for function with one or many inputs.
- **speed and stability optimizations** - Get the right answer for $\log(1+x)$ even when x is really tiny.
- **dynamic C code generation** - Evaluate expressions faster.
- **extensive unit-testing and self-verification** - Detect and diagnose many types of errors.

Theano has been powering large-scale computationally intensive scientific investigations since 2007. But it is also approachable enough to be used in the classroom (University of Montreal's [deep learning/machine learning](#) classes).

News

- Theano 0.8 was released 21th March 2016. Everybody is encouraged to update.
- Multi-GPU.
- We added support for [CuDNN v5](#).
- We added support for `cwmem` to speed up the GPU memory allocation.
- Theano 0.7 was released 26th March 2015. Everybody is encouraged to update.
- We support [CuDNN](#) if it is installed by the user.
- Open Machine Learning Workshop 2014 [presentation](#).
- Colin Raffel [tutorial on Theano](#).
- Ian Goodfellow did a [12h class with exercises on Theano](#).
- New technical report on Theano: [Theano: new features and speed improvements](#).
- [hPCs 2011 Tutorial](#). We included a few fixes discovered while doing the Tutorial.

You can watch a quick (20 minutes) introduction to Theano given as a talk at [SciPy 2010](#) via streaming (or downloaded) video:

[Transparent GPU Computing With Theano](#). James Bergstra, SciPy 2010, June 30, 2010.

Download

Theano is now [available on PyPI](#), and can be installed via `easy_install Theano`, `pip install Theano` or by downloading and unpacking the tarball and typing `python setup.py install`.

Those interested in bleeding-edge features should obtain the latest development version, available via:

```
git clone git://github.com/Theano/Theano.git
```

You can then place the checkout directory on your `$PYTHONPATH` or use `python setup.py develop` to install a `.pth` into your `site-packages` directory, so that when you pull updates via Git, they will be automatically reflected the "installed" version. For more information about installation and configuration, see [installing Theano](#).

Status

Build [passing](#) | [PyPI](#) [v0.8.0](#) | [Downloads](#) [402/month](#)

Citing Theano

Theano, Python toolkit (open source)

Deep Genomics
Genomics
Startup using deep
learning

DEEP LEARNING



GENOMICS

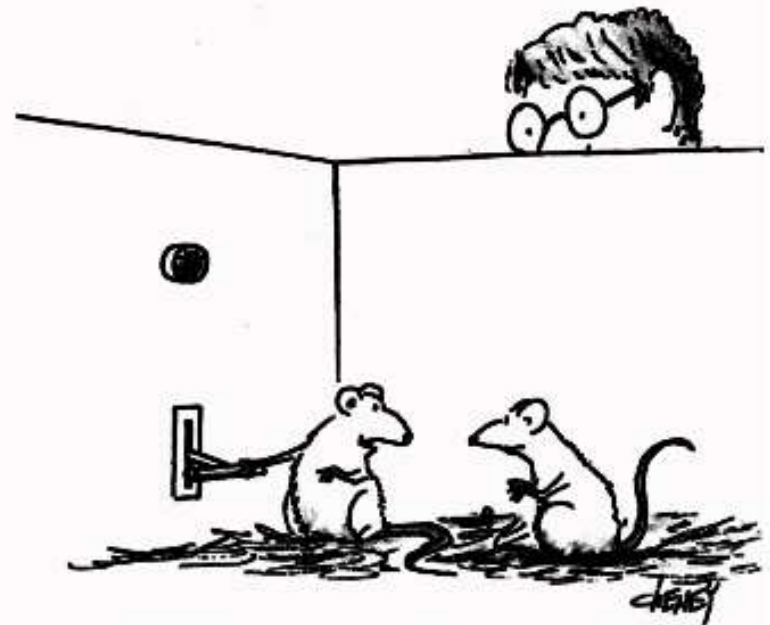


PRECISION MEDICINE



Limits of deep learning

- Correlations cannot distinguish between implication, reverse implication and equivalence : what is the cause of good performance ?
- Semantics of learnt variables are grounded in scientific context : but hard to interpret parameters learnt by deep learning
- Adversarial samples cause issues in deep nets : bad for badly behaved data
- Lack estimates of uncertainty : how to quantify quality of prediction
- Many hyperparameters and parameters to tune : many ways to go wrong



It's a rather interesting phenomenon. Every time I press this lever, that post-graduate student breathes a sigh of relief.

Terry's Lab Rat

Thanks !

- Further reading :
 - <http://deeplearning.cs.cmu.edu> : a course by Bhiksha Raj
 - <http://www.nature.com/nbt/journal/v33/n8/full/nbt.3313.html>
 - <http://www.nature.com/nbt/journal/v33/n8/full/nbt.3300.html>