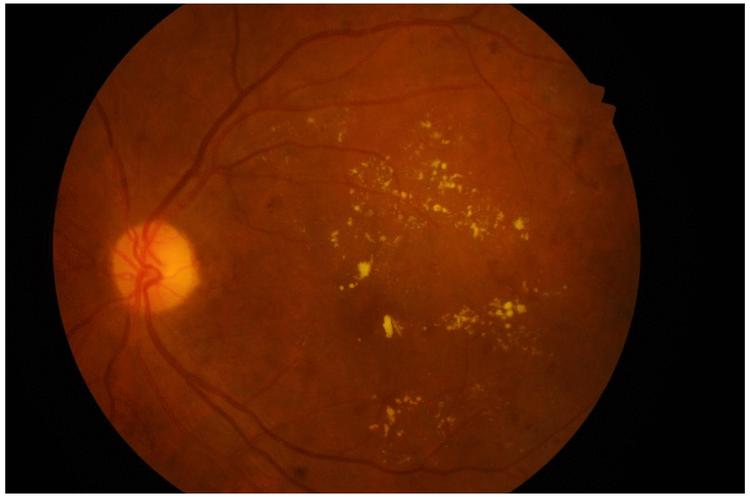
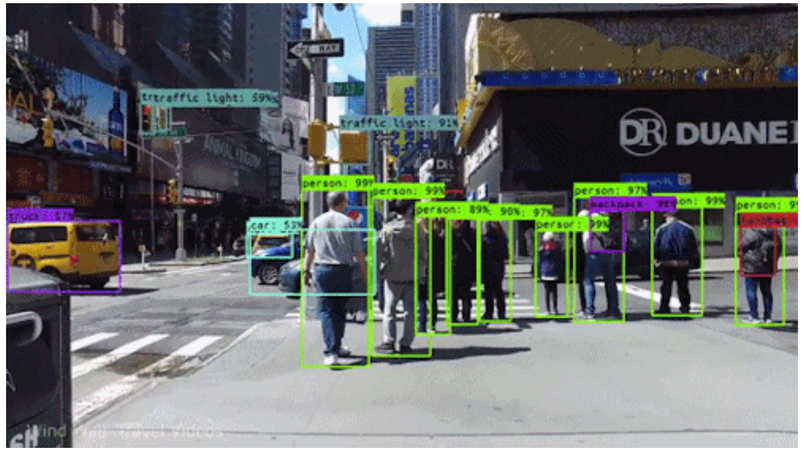
RICH ORMISTON

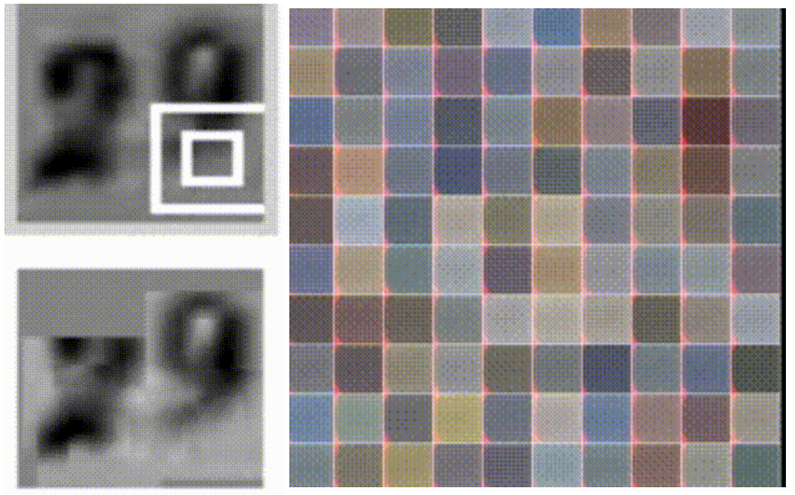
DEEP LEARNING IN GRAVITATIONAL WAVES DATA ANALYSIS

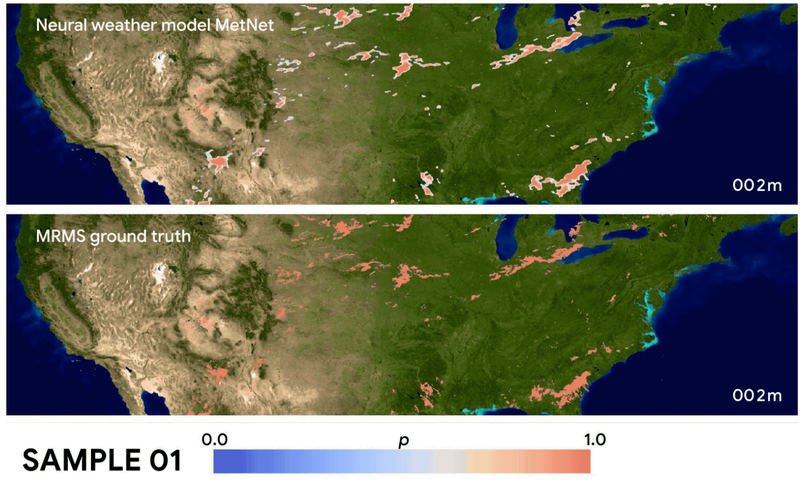
NRT Seminar at the University of Minnesota - Jan 26, 2021

MACHINE LEARNING IS EVERYWHERE

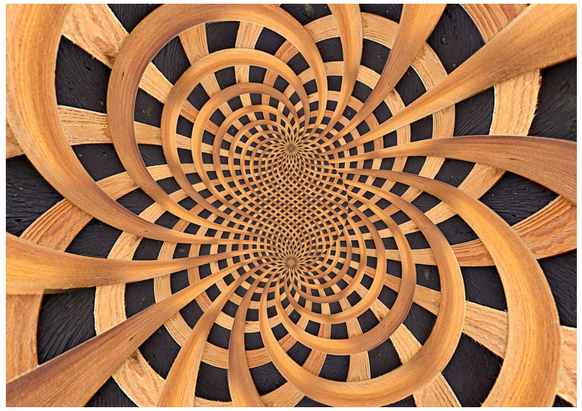






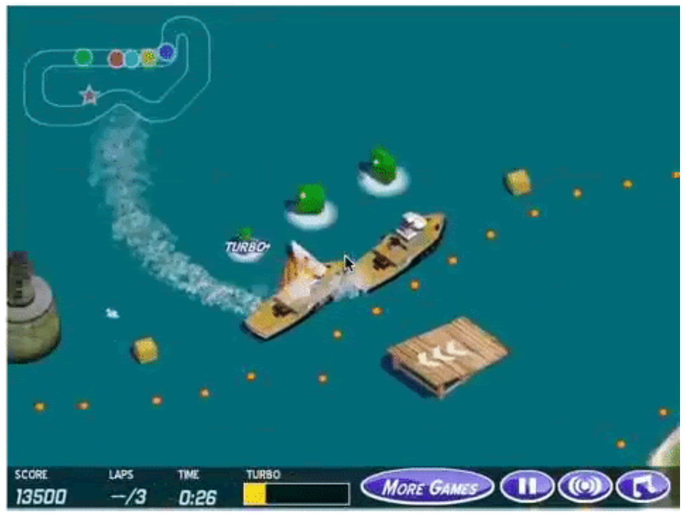


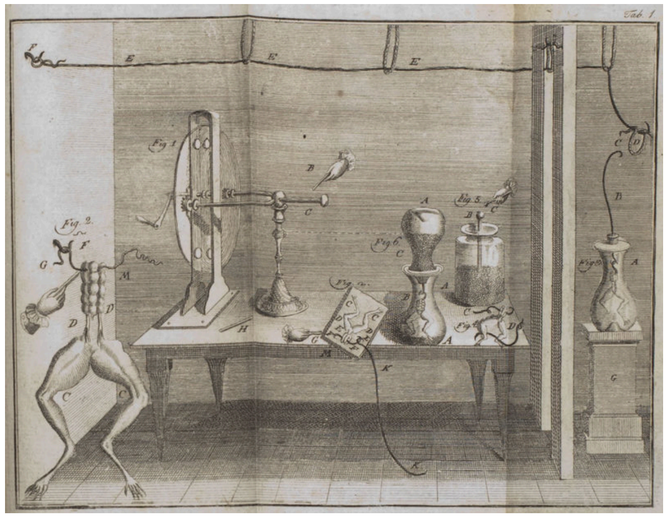
NOT ALL APPLICATIONS ARE USEFUL (YET)



+

=





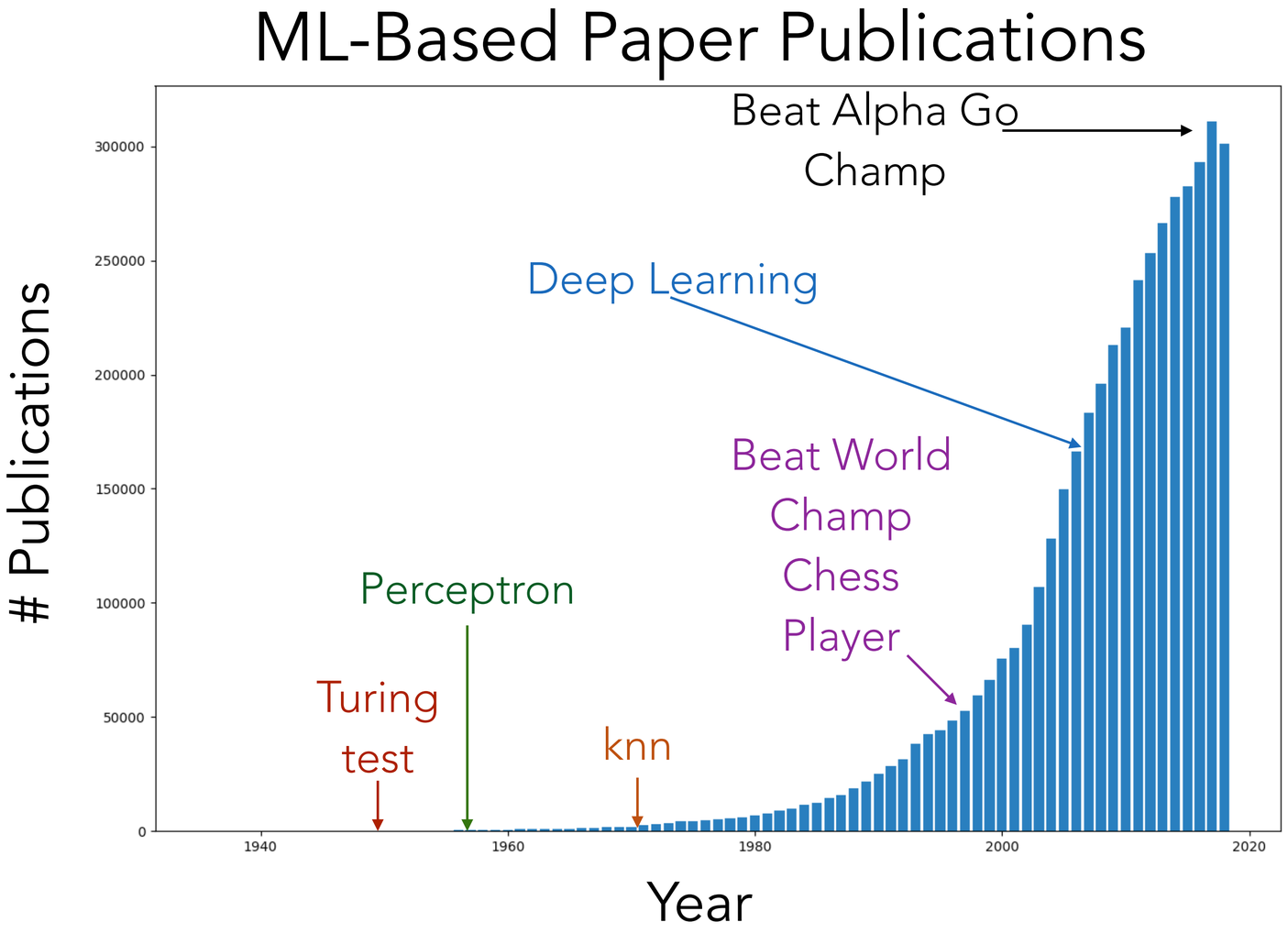
A BRIEF HISTORY OF MLAs

- Better Computers (CPU & GPU)

- Understanding of nonlinear problems

- Solving interesting problems

- **Parametrizing our ignorance**



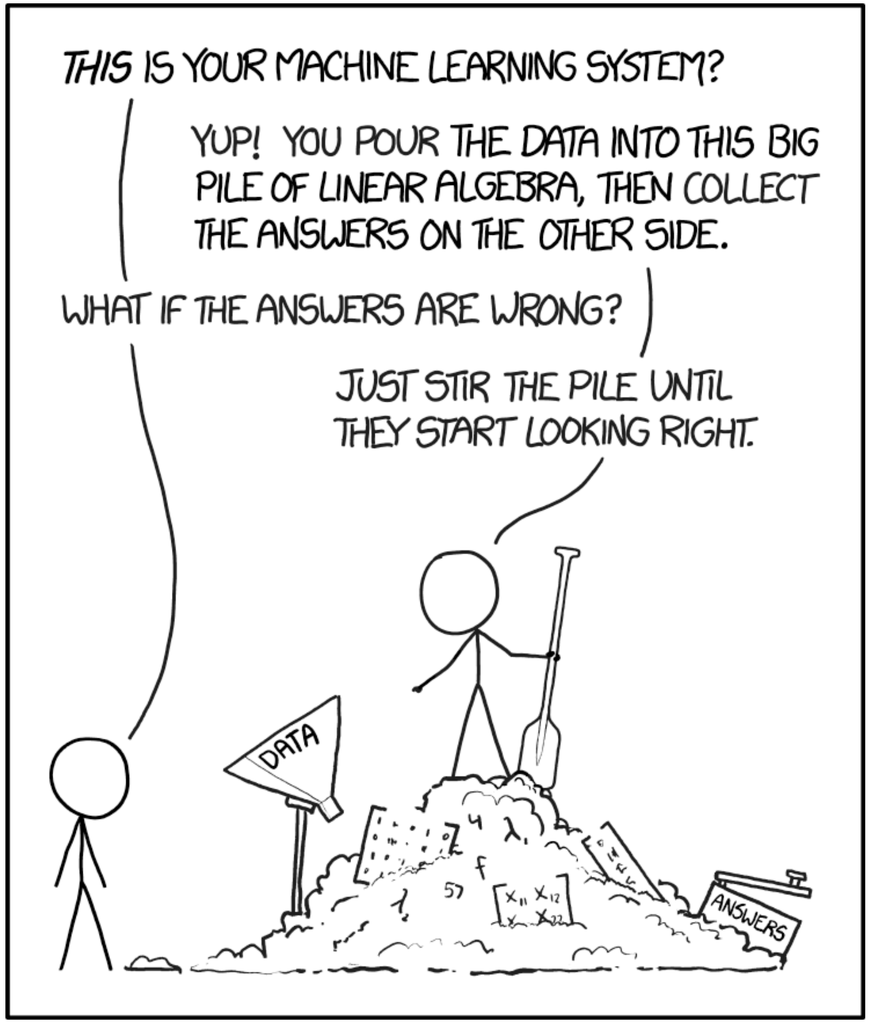
THE HARD PARTS OF DEVELOPING MLAs

- Asking good questions

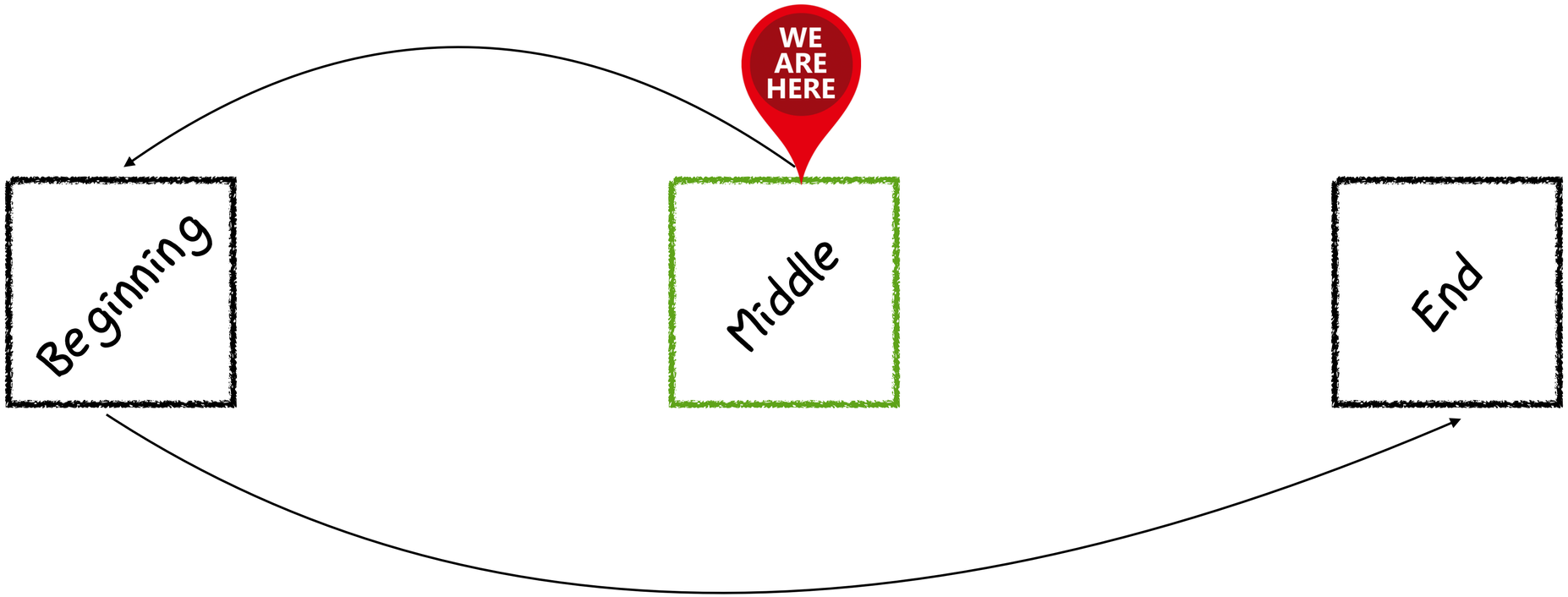
- Acquiring good data

- Understanding the requirements of the problem

HOW THEY WORK



LET'S START IN THE MIDDLE OF THE STORY...

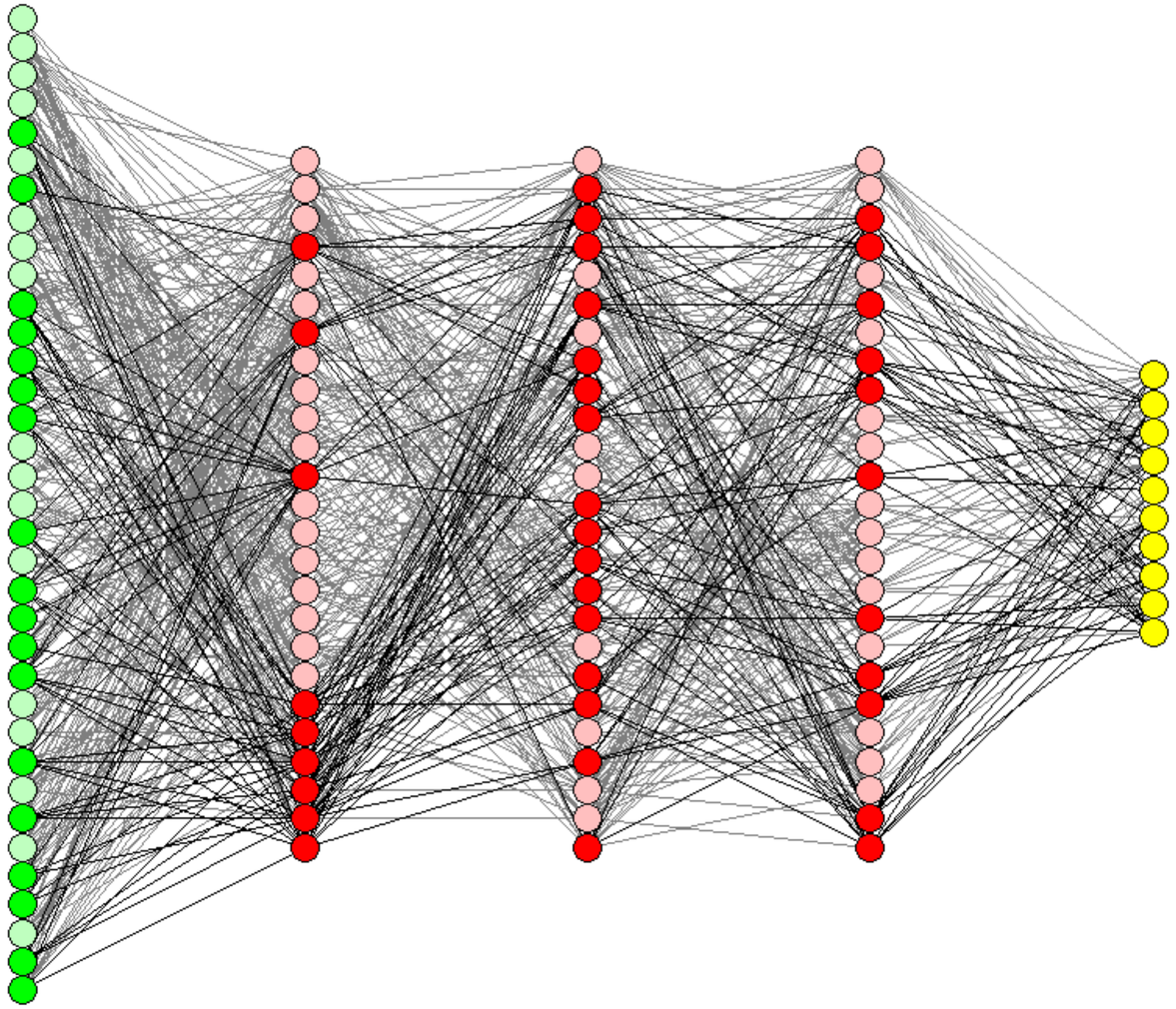


ADDITION NETWORK - 1

Let's build a network that can add 2 numbers together:

1. By hand

2. Using python



A calculator designed by a University

ADDITION NETWORK - 2

- Input two numbers (a, b)
- Output the sum: $a + b$
- Should hold true for any pair of numbers (n_1, n_2)
- **First problem**: We need to reduce a pair of numbers to a single number

ADDITION NETWORK - 3

- Input two numbers (a, b)
- Output the sum: $a + b$
- Should hold true for any pair of numbers (n_1, n_2)
- **First problem**: We need to reduce a pair of numbers to a single number

Answer: Use matrices!

ADDITION NETWORK - 4

Let's try something generic

$$\begin{pmatrix} \alpha_{11} & \alpha_{21} \\ \alpha_{12} & \alpha_{22} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a \cdot \alpha_{11} + b \cdot \alpha_{21} \\ a \cdot \alpha_{12} + b \cdot \alpha_{22} \end{pmatrix}$$

We have a summation, but this is still a vector. However, we can turn a vector into a scalar with a dot product

$$\begin{pmatrix} \beta_1 & \beta_2 \end{pmatrix} \begin{pmatrix} \alpha_{11} & \alpha_{21} \\ \alpha_{12} & \alpha_{22} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \beta_1(a \cdot \alpha_{11} + b \cdot \alpha_{21}) + \beta_2(a \cdot \alpha_{12} + b \cdot \alpha_{22})$$

ADDITION NETWORK - 5

This is close! If we manually set the matrix entries, we can get the desired result. Let

$$\alpha_{11} = \alpha_{21} = \beta_1 = 1$$

$$\alpha_{12} = \alpha_{22} = \beta_2 = 0$$

$$\rightarrow \underbrace{(1 \quad 0)\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}}_{Network}\begin{pmatrix} a \\ b \end{pmatrix} = a + b$$

We now have the answer to

$$\begin{pmatrix} a \\ b \end{pmatrix} \boxed{?} = a + b$$

ADDITION NETWORK - 5

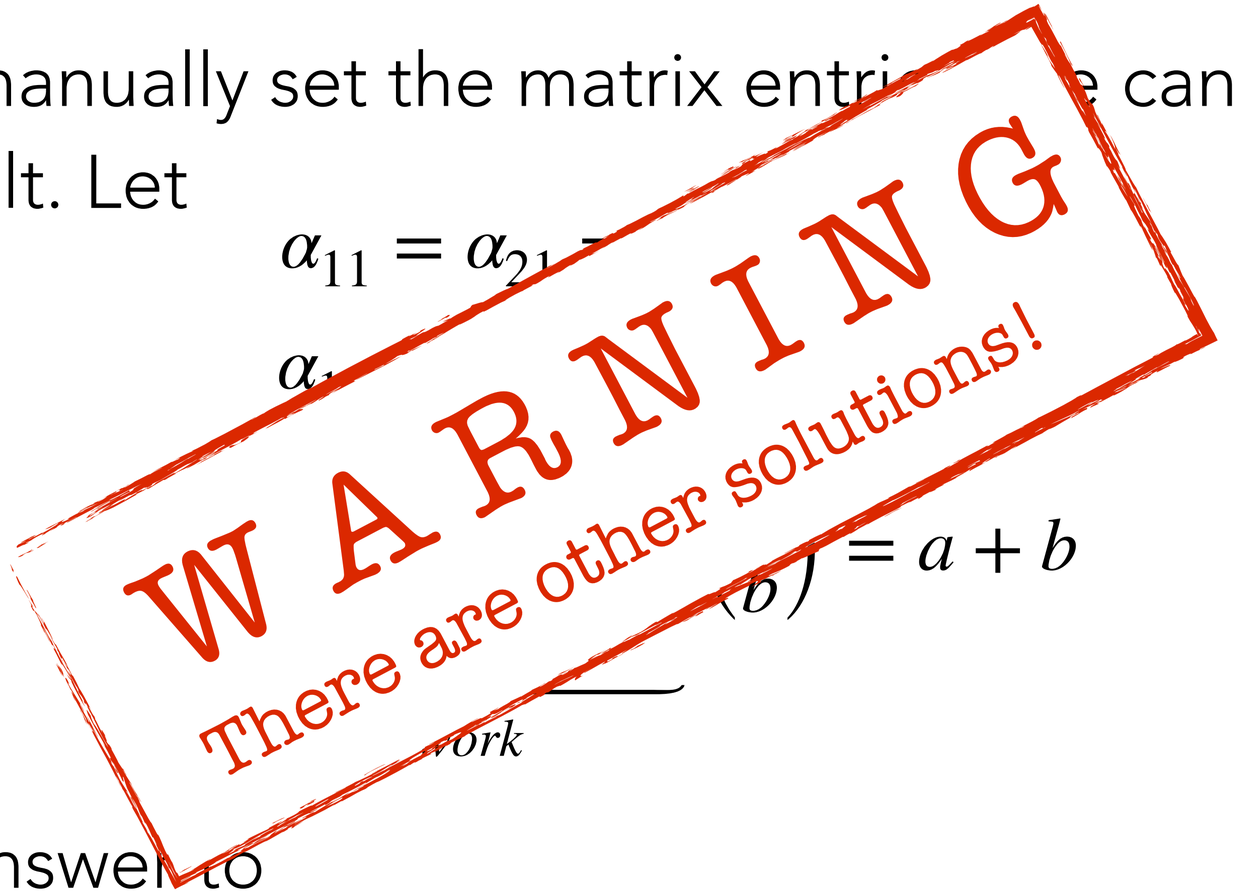
This is close! If we manually set the matrix entri[illegible]e can get the desired result. Let

$$\alpha_{11} = \alpha_{21} =$$

$$\alpha_{\cdot}$$

$$\left(b\right) = a + b$$

work



We now have the answer to

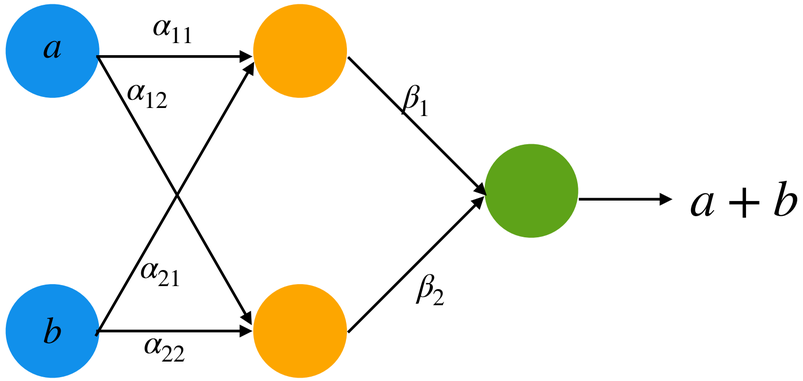
$$\binom{a}{b} \boxed{?} = a + b$$

ADDITION NETWORK - 6

We can write the general solution graphically as follows

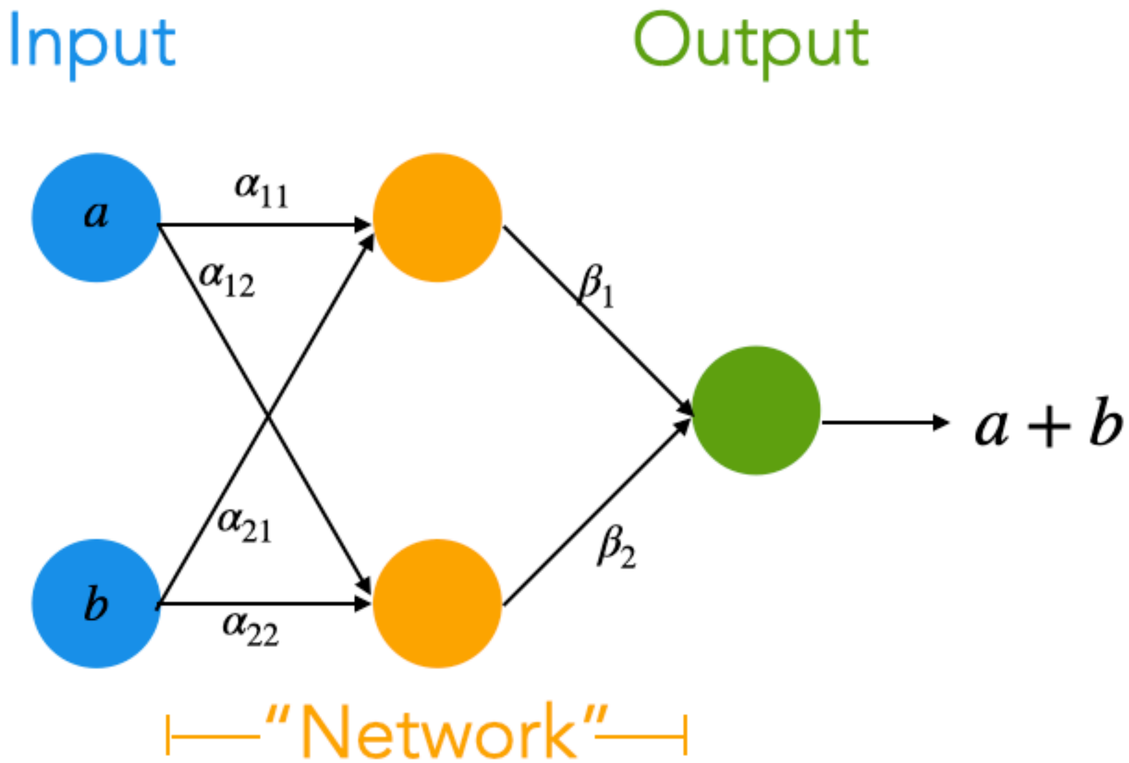
$$\begin{pmatrix} a \\ b \end{pmatrix} \boxed{?} = a + b$$

$$\begin{pmatrix} \beta_1 & \beta_2 \end{pmatrix} \begin{pmatrix} \alpha_{11} & \alpha_{21} \\ \alpha_{12} & \alpha_{22} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \beta_1(a \cdot \alpha_{11} + b \cdot \alpha_{21}) + \beta_2(a \cdot \alpha_{12} + b \cdot \alpha_{22})$$



ADDITION NETWORK - 7

We can write the general solution graphically as follows

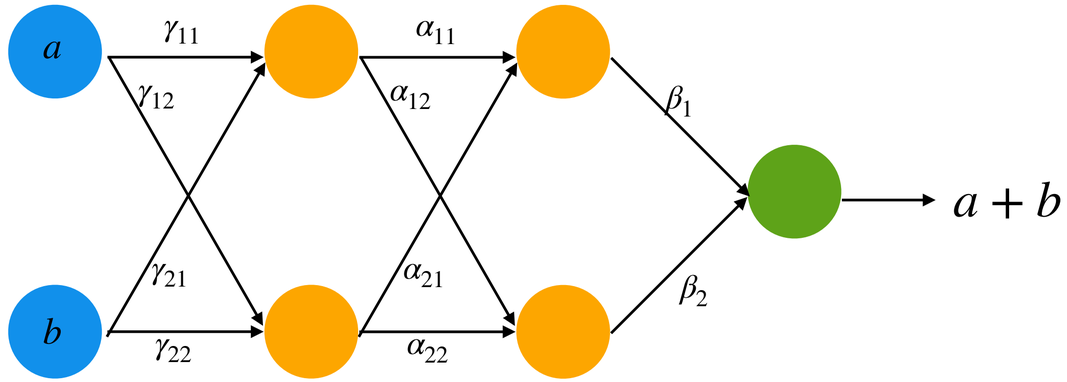


Question: is this “architecture” unique?

ADDITION NETWORK - 8

Answer: Yes! Kind of… for linear networks only, so no not really.
To see this, let's start by adding another 2x2 matrix to the "network"

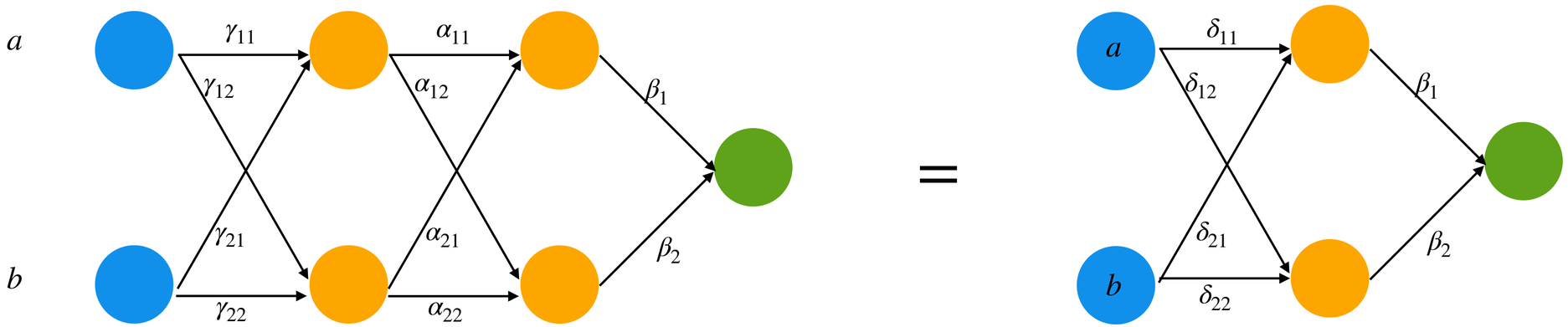
$$\begin{pmatrix}\beta_1 & \beta_2\end{pmatrix}\begin{pmatrix}\alpha_{11} & \alpha_{21}\\ \alpha_{12} & \alpha_{22}\end{pmatrix}\begin{pmatrix}\gamma_{11} & \gamma_{21}\\ \gamma_{12} & \gamma_{22}\end{pmatrix}$$



ADDITION NETWORK - 9

But the product of two rotations is just another rotation, so really we can write

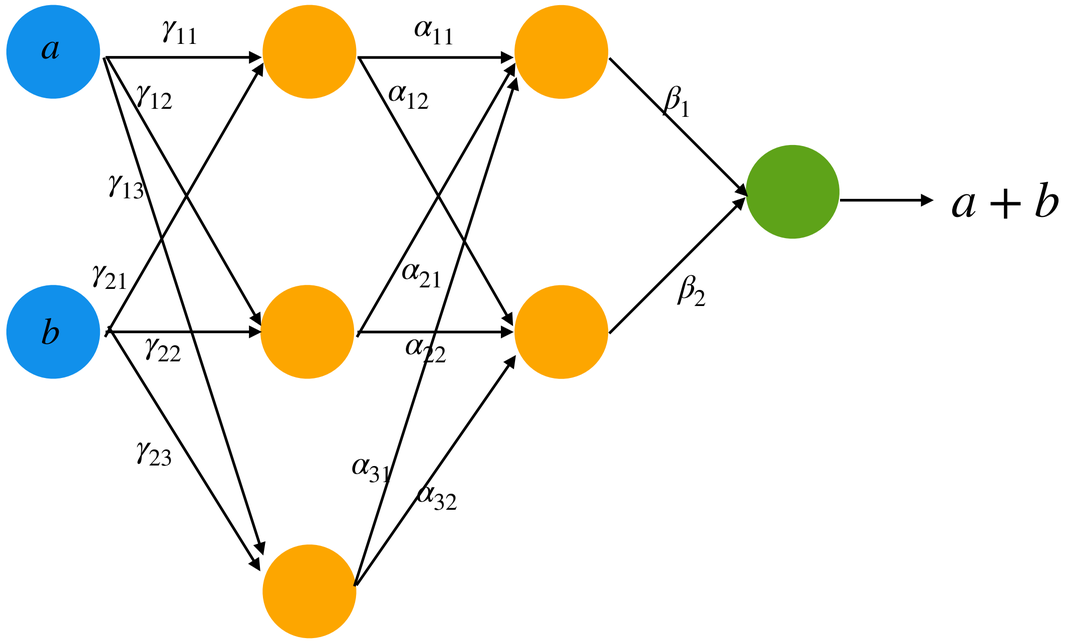
$$\begin{pmatrix} \alpha_{11} & \alpha_{21} \\ \alpha_{12} & \alpha_{22} \end{pmatrix} \begin{pmatrix} \gamma_{11} & \gamma_{21} \\ \gamma_{12} & \gamma_{22} \end{pmatrix} = \begin{pmatrix} \delta_{11} & \delta_{21} \\ \delta_{12} & \delta_{22} \end{pmatrix}$$



ADDITION NETWORK - 11

What about other matrix sizes?

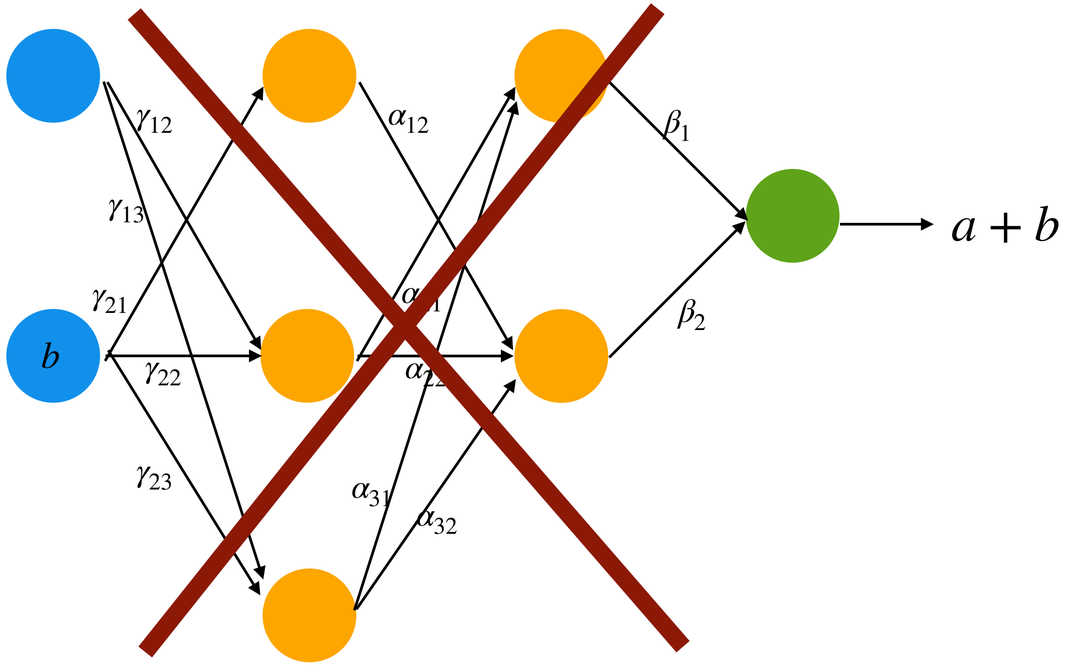
$$\begin{pmatrix}\beta_1 & \beta_2\end{pmatrix}\begin{pmatrix}\alpha_{11} & \alpha_{21} & \alpha_{31}\\ \alpha_{12} & \alpha_{22} & \alpha_{32}\end{pmatrix}\begin{pmatrix}\gamma_{11} & \gamma_{21}\\ \gamma_{12} & \gamma_{22}\\ \gamma_{13} & \gamma_{23}\end{pmatrix}$$



ADDITION NETWORK - 10

Again, no. This adds more weights that we'd have to sort out (14 instead of 6), but we are again going from a 2x2 → 1x1. There is no benefit to increasing dimensions. In general, a dimension reduction is preferable when possible.

Prune excess!



ADDITION NETWORK - 9

Therefore, although they look different, all other representations are fundamentally the same. But if you had determine every matrix value, which would you rather use?

The lessons here are:

-Reduce the complexity to the least amount needed to solve the problem
-There will probably be multiple solutions
-Neural networks are fundamentally linear algebra (kind of)

LINEAR ALGEBRA GIVES LINEAR RESULTS

Take notice that this equation is linear in a and b

$$\begin{pmatrix}\beta_1 & \beta_2\end{pmatrix}\begin{pmatrix}\alpha_{11} & \alpha_{21}\\ \alpha_{12} & \alpha_{22}\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix} = \beta_1(a\cdot\alpha_{11}+b\cdot\alpha_{21})+\beta_2(a\cdot\alpha_{12}+b\cdot\alpha_{22})$$

In other words, we could never find a 'network' to give us the answer to $a \cdot b$ or $e^{a\cdot b}$ etc. Since the world isn't very linear, we have to address this.

$$\begin{pmatrix}\beta_1 & \beta_2\end{pmatrix}\begin{pmatrix}\alpha_{11} & \alpha_{21}\\ \alpha_{12} & \alpha_{22}\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix} \rightarrow \begin{pmatrix}\beta_1 & \beta_2\end{pmatrix} f\left[\begin{pmatrix}\alpha_{11} & \alpha_{21}\\ \alpha_{12} & \alpha_{22}\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix}\right]$$

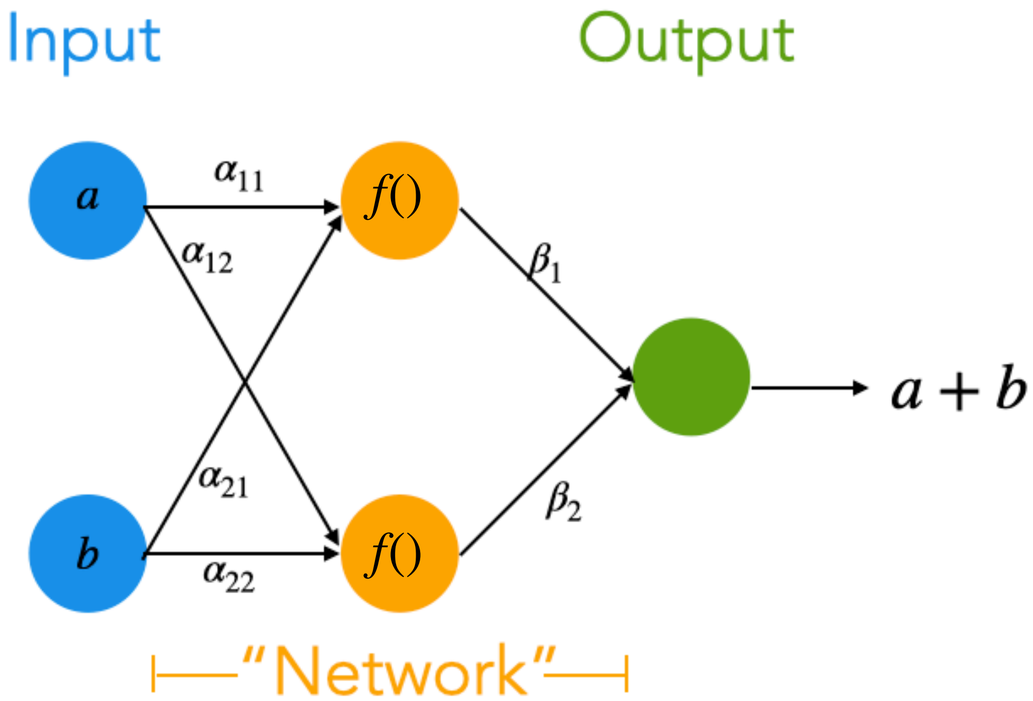
NONLINEAR ALGEBRA GIVES NONLINEAR RESULTS

The function $f()$ can be anything we want. If $f()$ is linear (i.e., $f(x) = x$) then the output will be a linear function of the input. But if $f()$ is nonlinear (i.e., $f(x) = x^2$) then the output is a *nonlinear* function of the input

$$
\begin{aligned}
(\beta_1\beta_2)\, f\left[\begin{pmatrix}\alpha_{11}\alpha_{21}\\ \alpha_{12}\alpha_{22}\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix}\right] &= (\beta_1\beta_2)\begin{pmatrix}f(a\cdot\alpha_{11}+b\cdot\alpha_{21})\\ f(a\cdot\alpha_{12}+b\cdot\alpha_{22})\end{pmatrix}\\
&= \beta_1(a\cdot\alpha_{11}+b\cdot\alpha_{21})^2+\beta_2(a\cdot\alpha_{12}+b\cdot\alpha_{22})^2\\
&= a^2(\ldots)+b^2(\ldots)+ab(\ldots)
\end{aligned}
$$

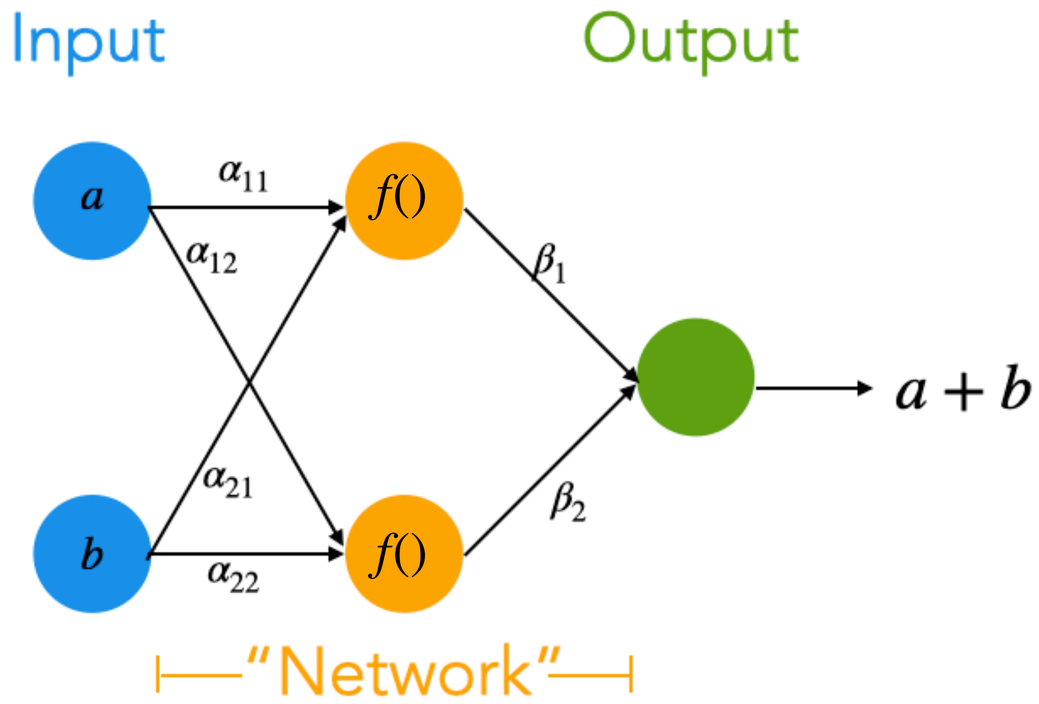
NONLINEAR ALGEBRA GIVES NONLINEAR RESULTS

Graphically, this is the same as before, but we understand that at each of the 'nodes' connecting the input to the output, there is a function being applied to the network weights



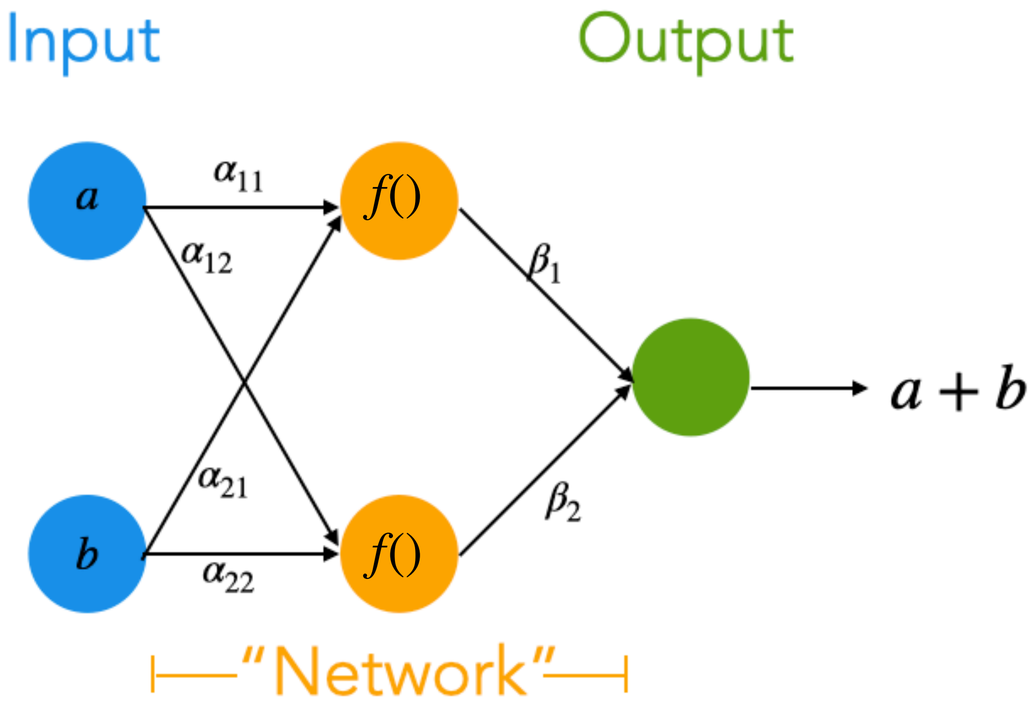
REVIEW AND TERMINOLOGY

- The orange circles that connect the input to the output are called "**hidden layers**" (the network sees them, but we do not)
- The functions we use to determine how the data mixes with the weights is called the "**activation function**"
- The matrix weights with their activation functions are the "**neurons**."
- The forward transfer of the input in this way qualifies this as a "**feed forward**" or "**dense**" network

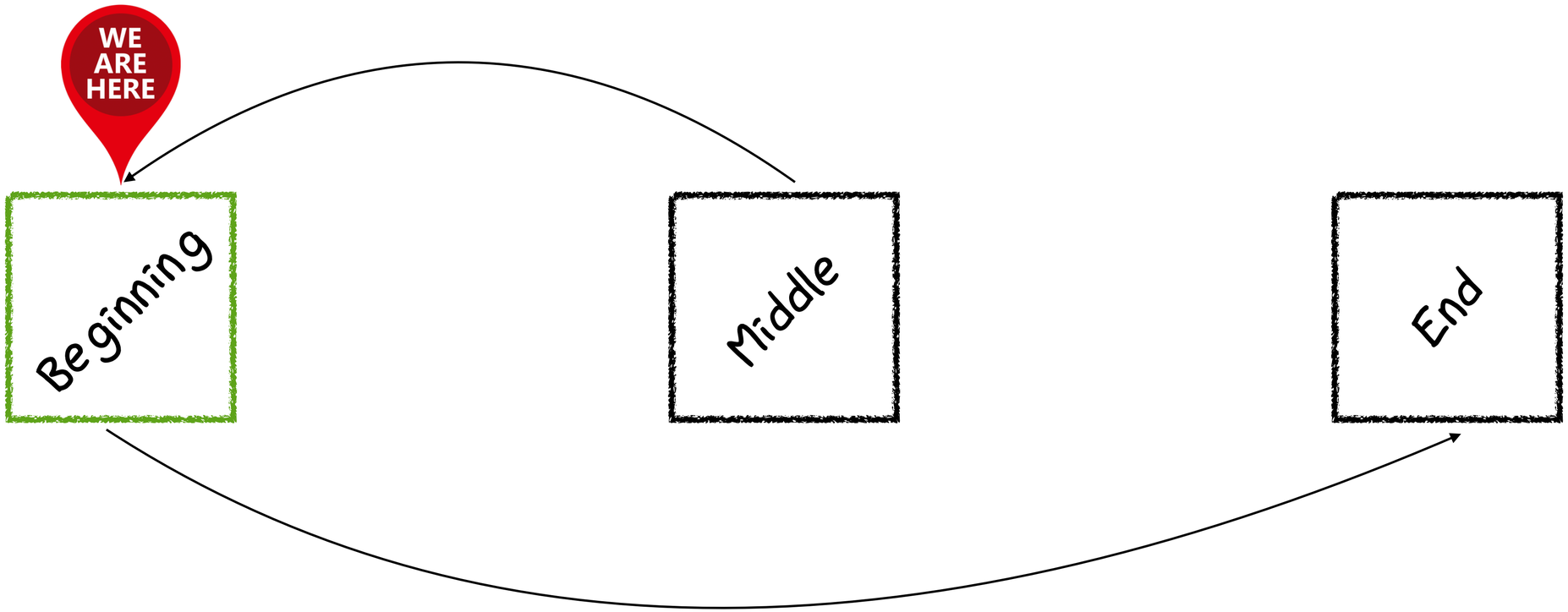


REVIEW AND TERMINOLOGY

- More layers does not necessarily mean better results - increasing the complexity increases the parameter space and training time (often $\mathcal{O}(n^3)$ or worse)
- Poorly choosing the activation function could mean never figuring out the proper weights
- We still don't know how the weights update, but let us first implement this network

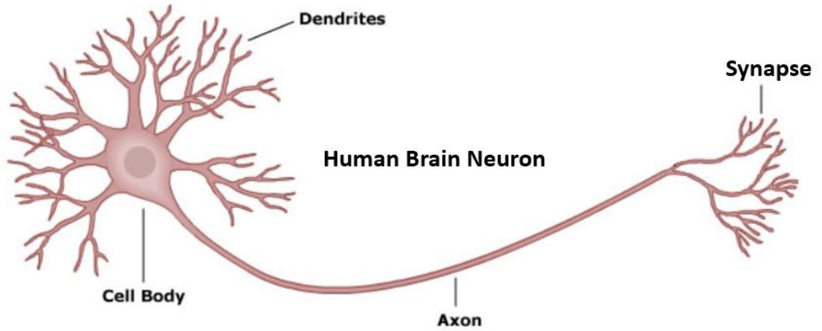


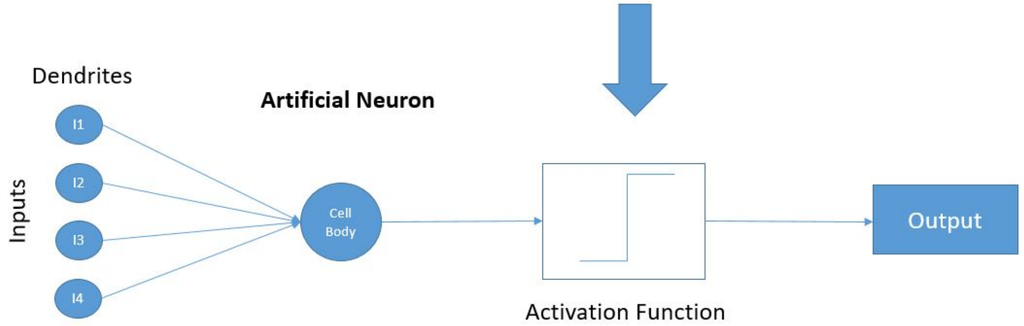
THE MATHEMATICAL ENGINE



THREE EASY STEPS

- **We** construct a network architecture (some [non] linear algebra)
- **We** supply a metric to describe how good our prediction is (or rather how bad the error is) a.k.a the “cost function”
- Stochastic gradient descent updates the weights to minimize the error (backpropagation)





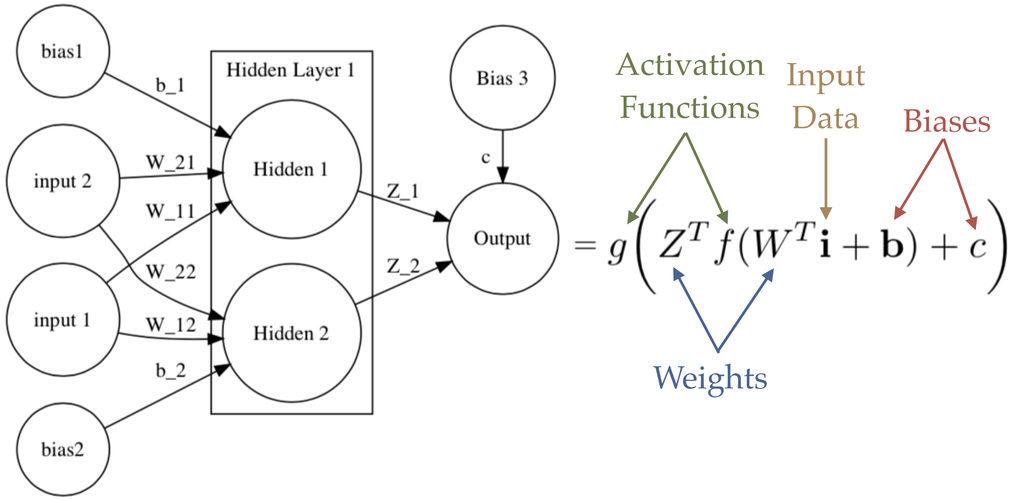
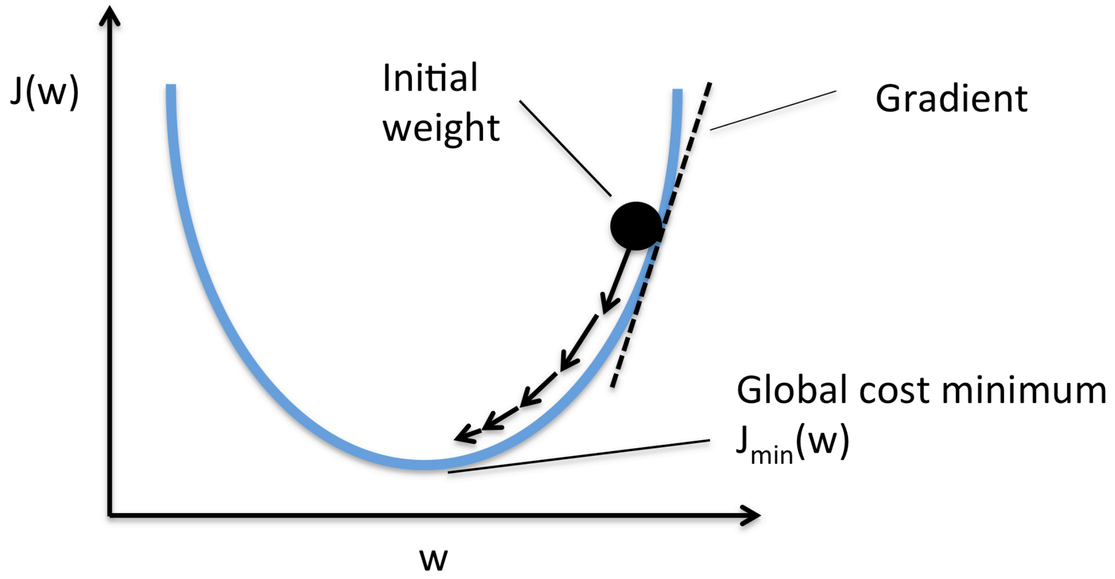


Image Credit: MPLSVPN - Moving Towards SDN and NFV Based Networks

GRADIENT DESCENT



Learning rate

Cost function

$$\mathbf{w} := \mathbf{w} - \eta \vec{\nabla} \sum_i J(w_i)$$

$$\rightarrow \mathbf{w} - \eta \vec{\nabla} \frac{1}{N} \sum_i^{N-1} (Y_i - \hat{Y}_i)^2$$

Target

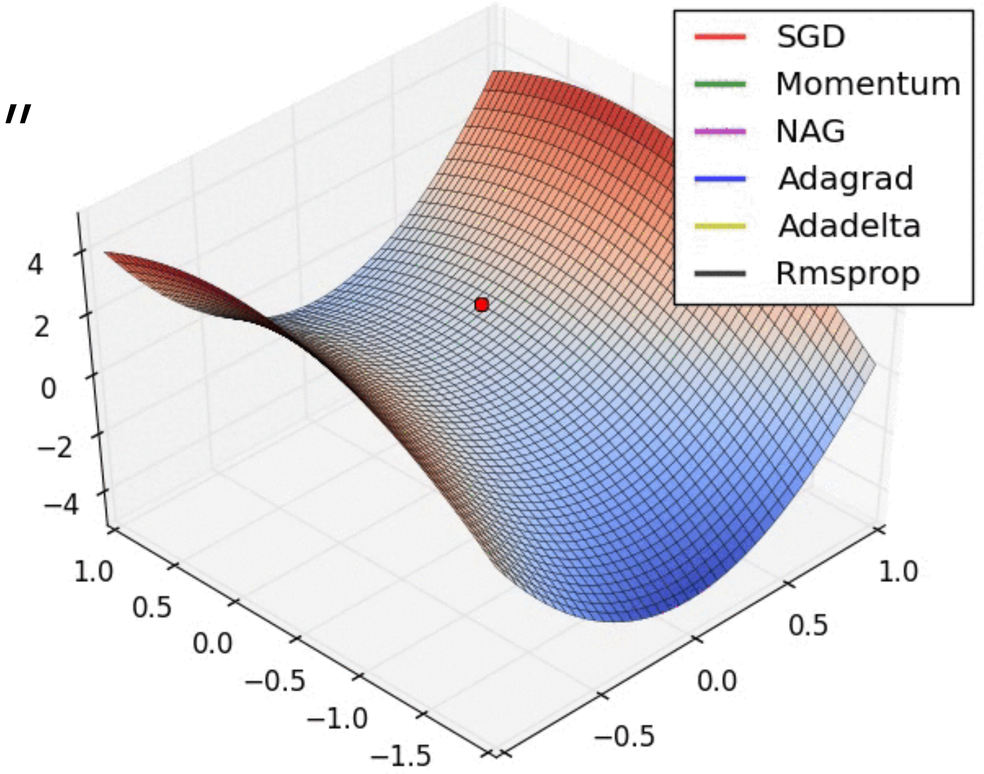
Estimate

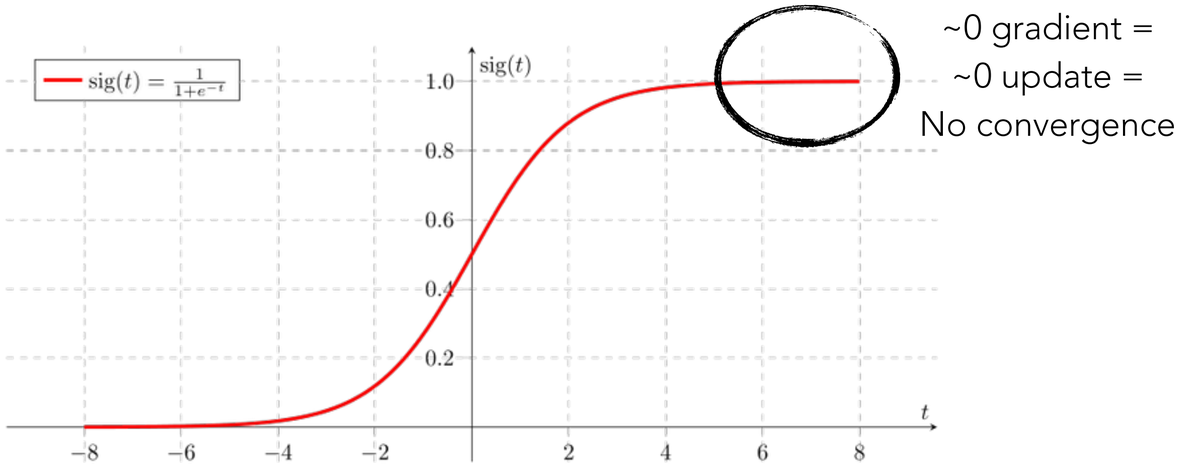
The learning rate, eta, sets the step size as represented by the arrows in the plot. Often, the learning rate is a decaying function of the epoch to improve convergence.

Easy to converge with simple parameter spaces. Extremely difficult and computationally costly with high dimensionality spaces.

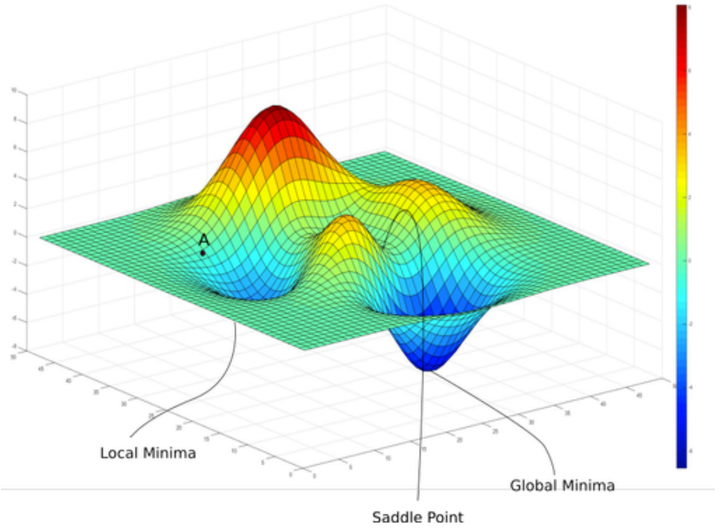
OTHER WAYS TO "DESCEND"

There are any number of ways to quantify the error. But a poor choice, or rather poor data pre-processing, can lead to network which fails to converge (vanishing gradient problem). The "plateaus" are the other solutions



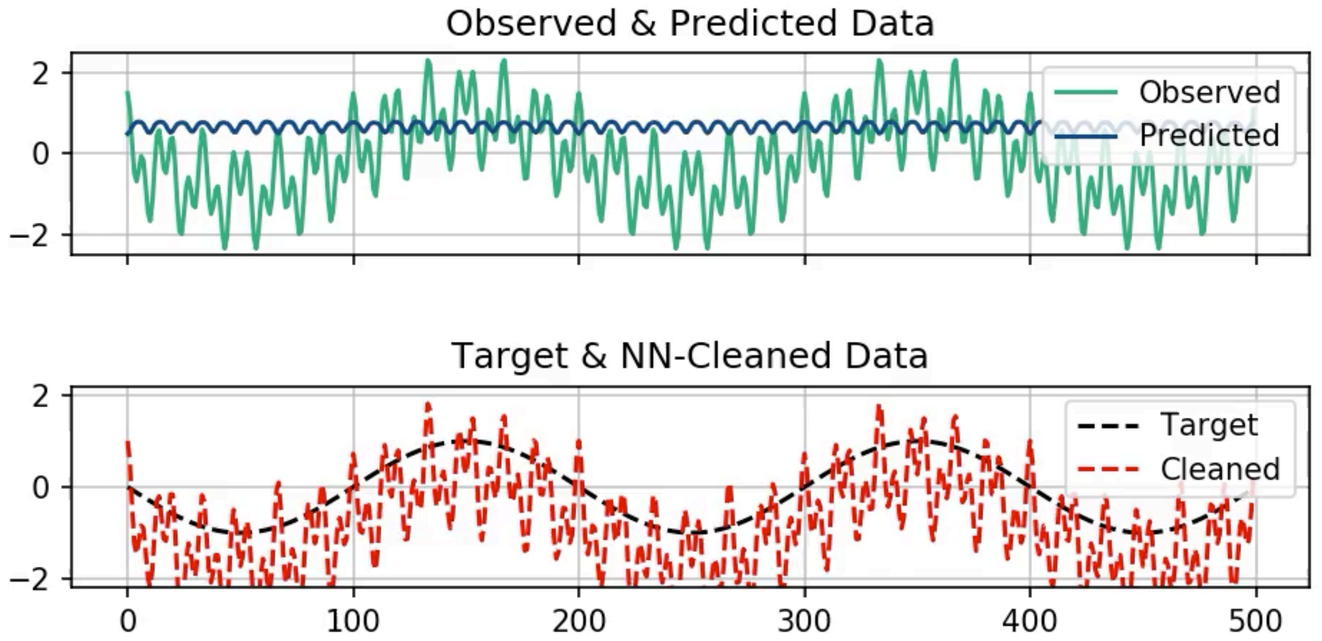


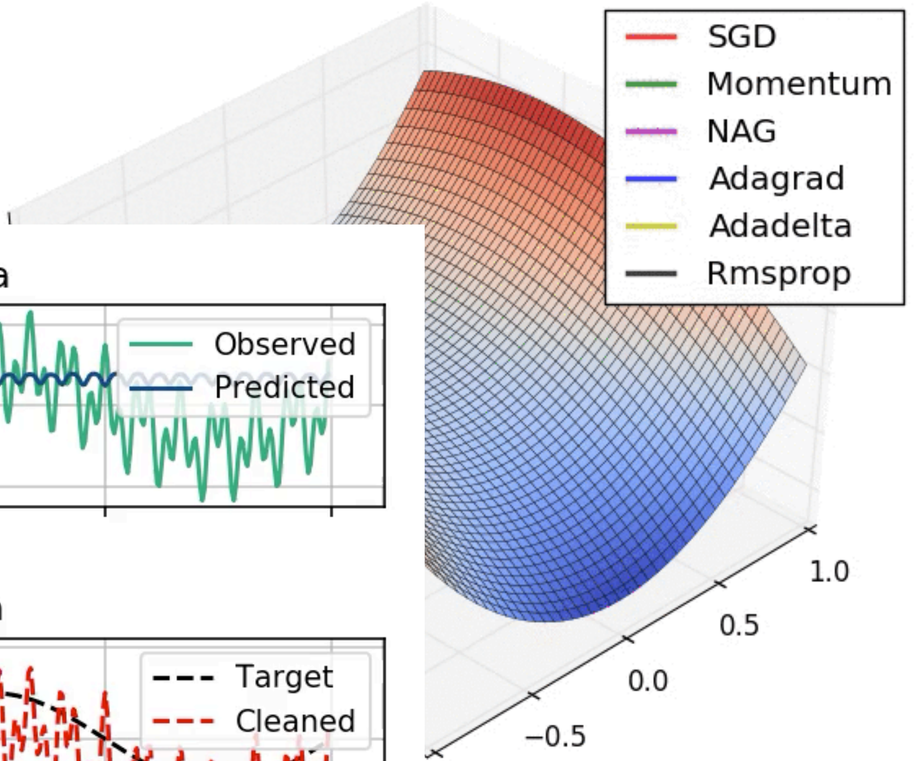
~0 gradient =
~0 update =
No convergence

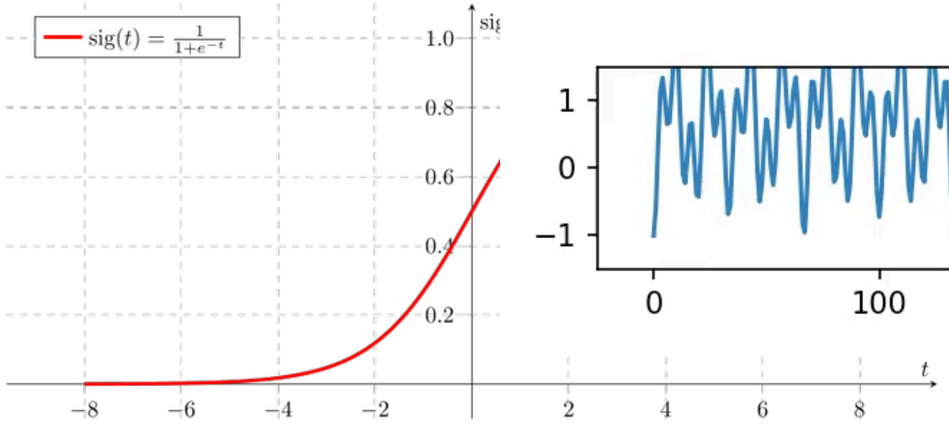


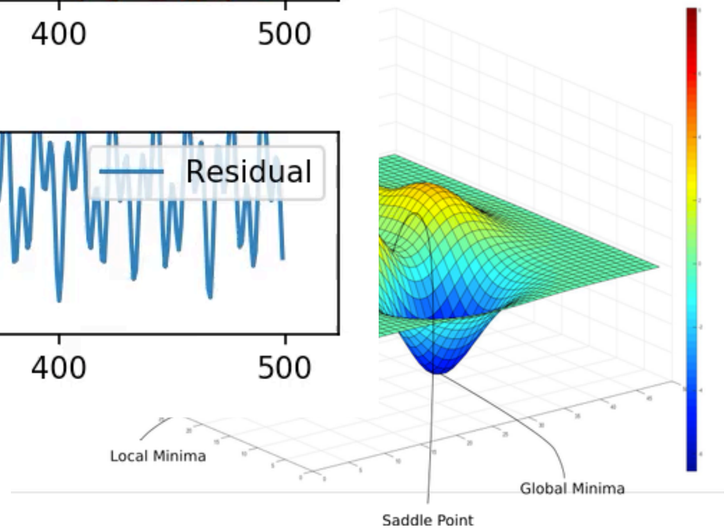
OTHER WAYS TO "DESCEND"

There are any num
quantify the error.
choice, or rather p
processing, can le
which fails to conv
gradient problem)

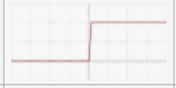
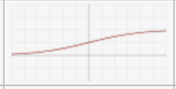
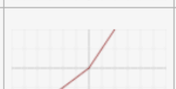


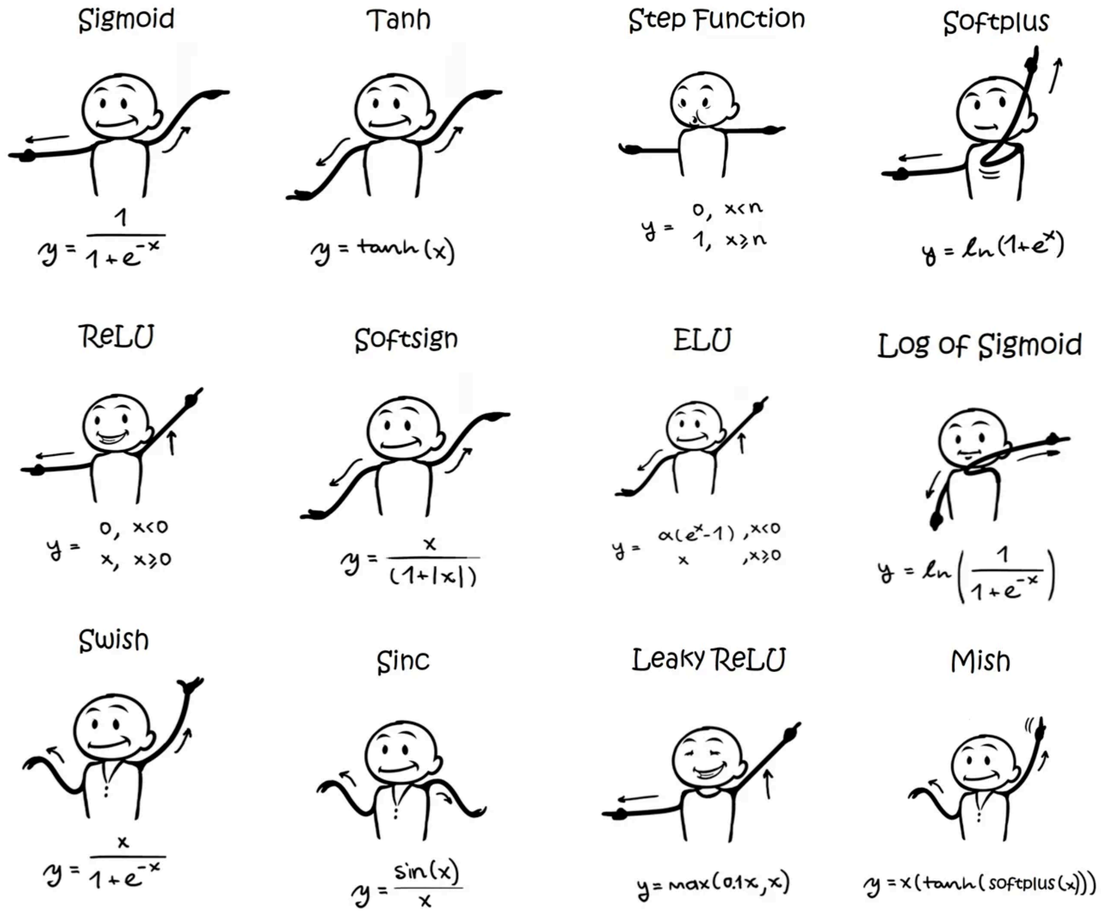






ACTIVATION FUNCTIONS
REGRESSION VS CATEGORICAL

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$



BACK PROPAGATION

How do the weights update based on the error? The chain rule!

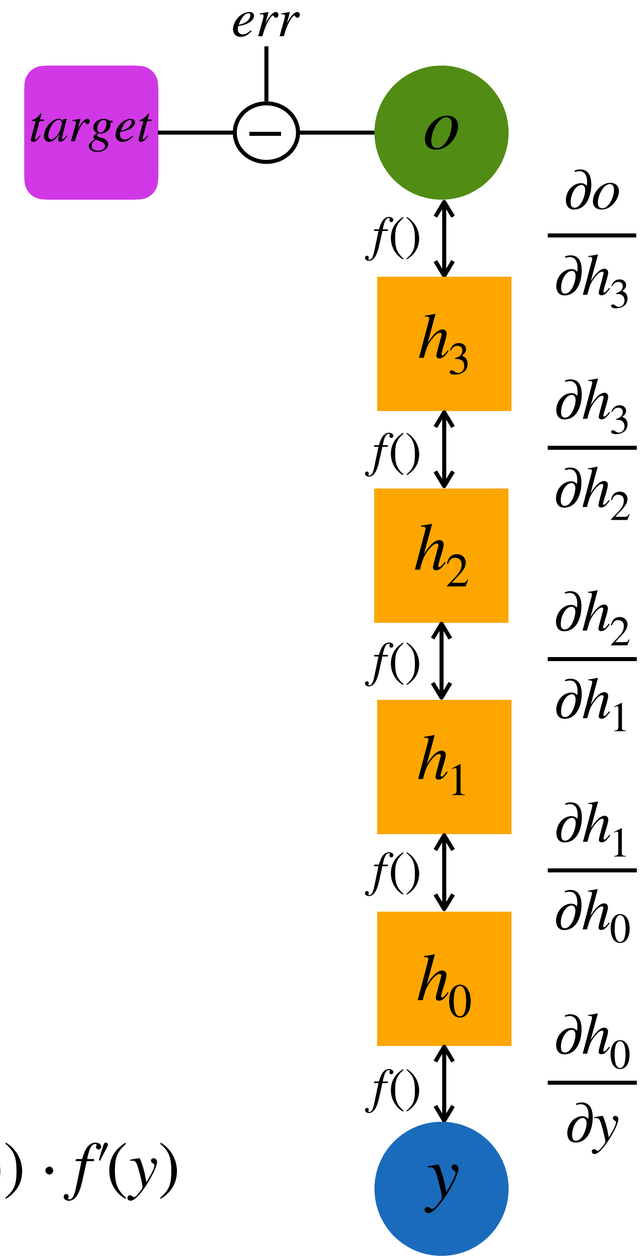
$$\frac{\partial o}{\partial y} = \frac{\partial o}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial h_0}\frac{\partial h_0}{\partial y}$$

Keep in mind we have to take the derivative of the activation functions as well. Ultimately we have to find

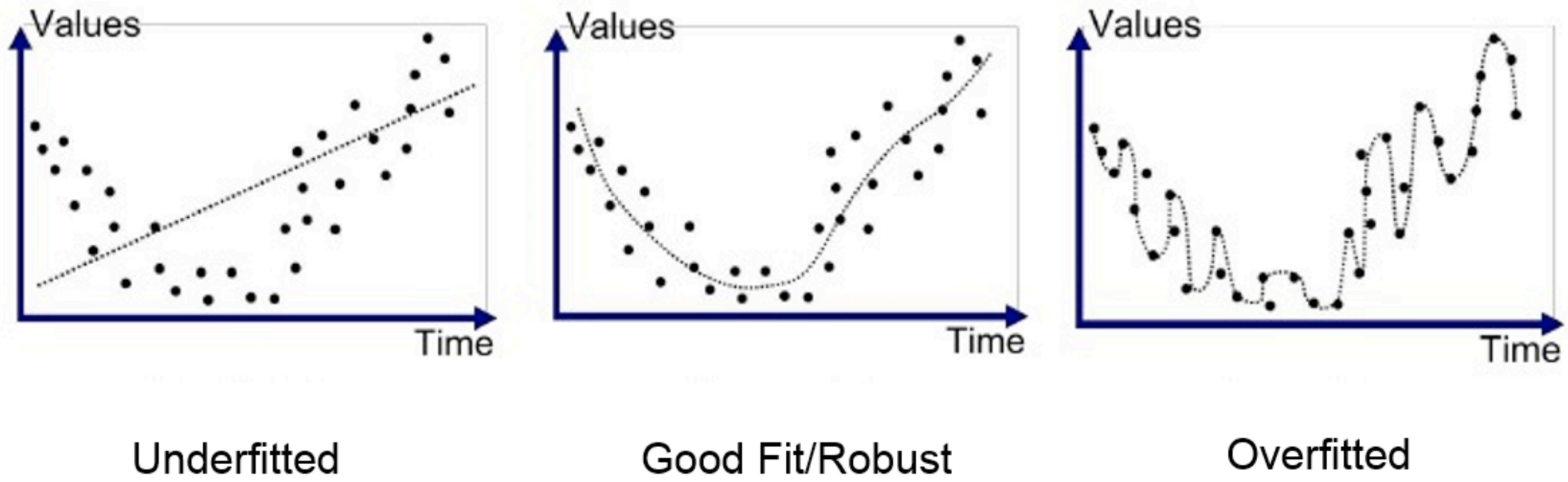
$$\rightarrow f'(h_3) \cdot f'(h_2) \cdot f'(h_1) \cdot f'(h_0) \cdot f'(y)$$

$$= f'(f(f(f(f(y))))) \cdot f'(f(f(f(y)))) \cdot f'(f(f(y))) \cdot f'(f(y)) \cdot f'(y)$$

$=$ 



UNDERFITTING AND OVERFITTING



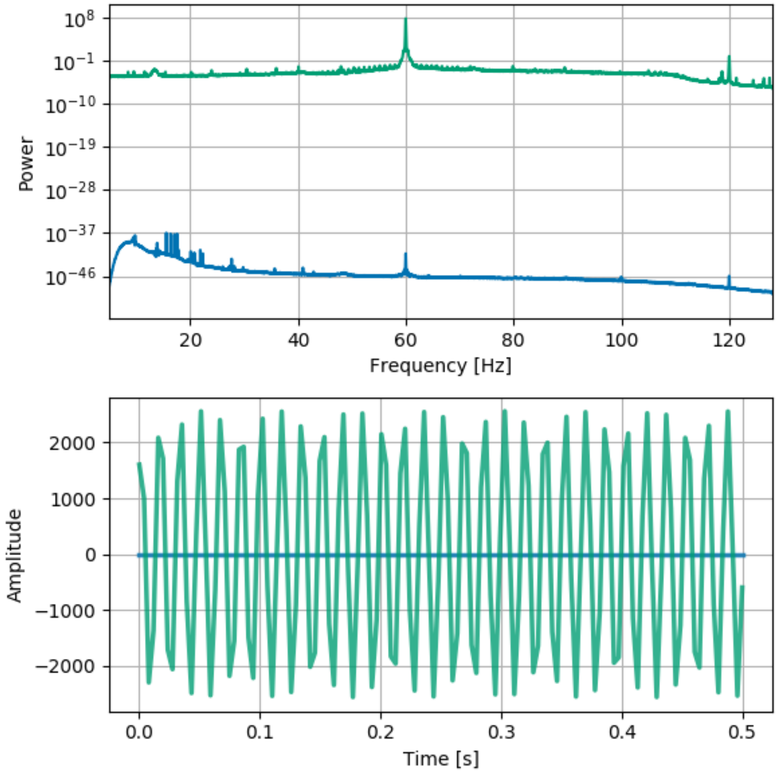
Use regularization methods and pruning

Photo from this website

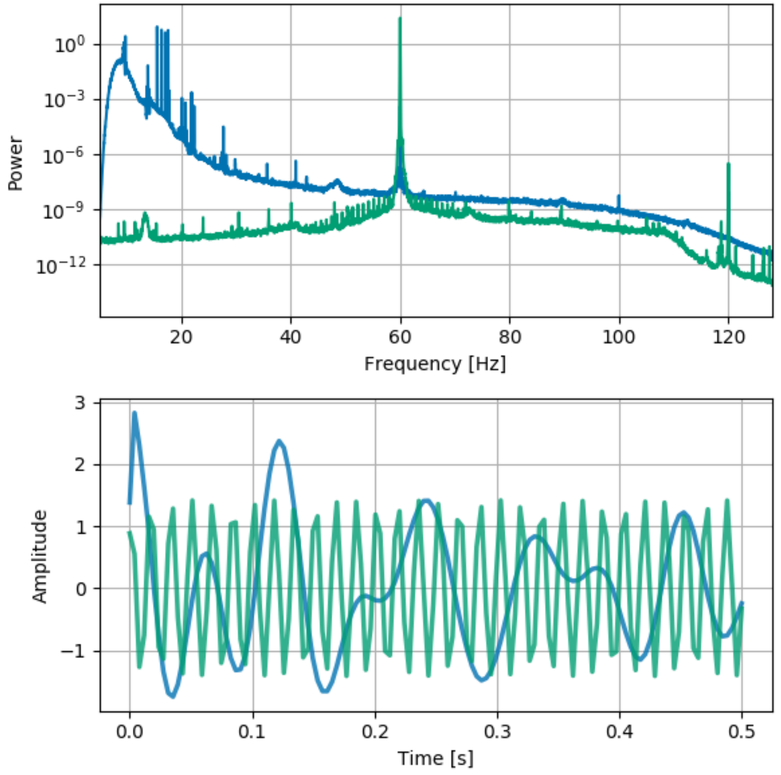
PREPROCESSING: SCALING

Also batch norm, batch renorm…

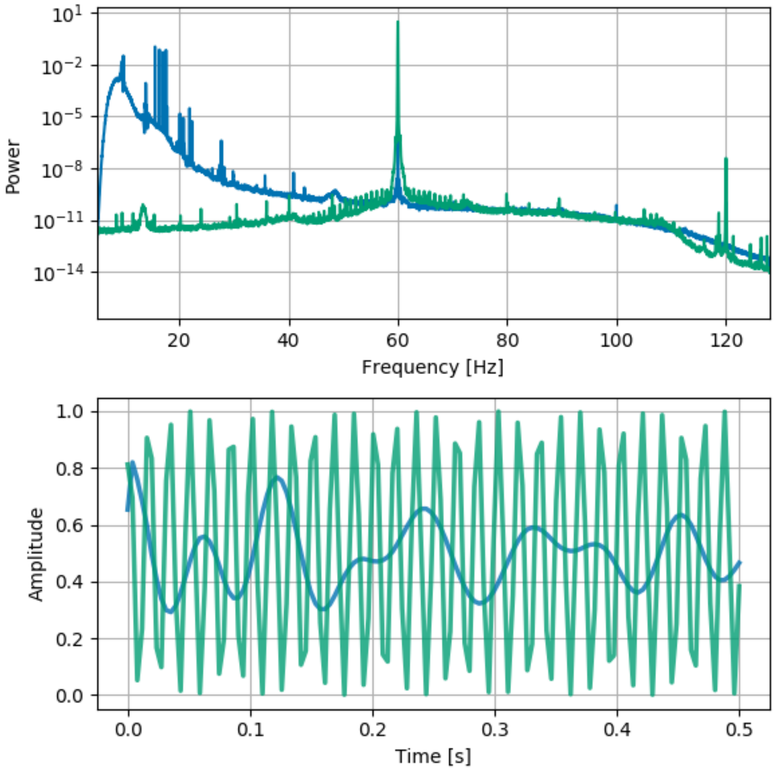
Before



Standardizing $y := \frac{y - \mu}{\sigma}$



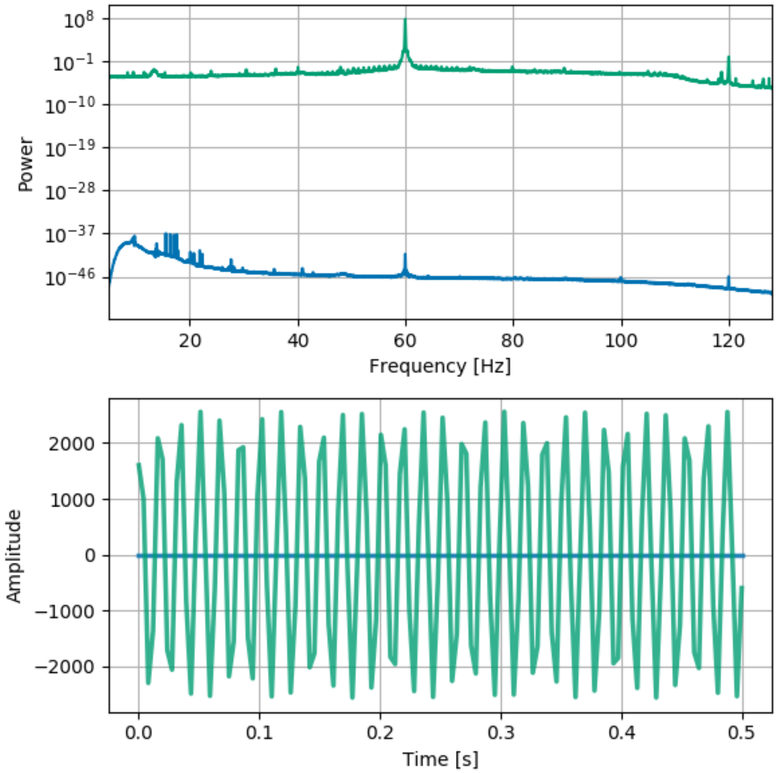
MinMax $y := \frac{y - \min(y)}{\max(y) - \min(y)}$



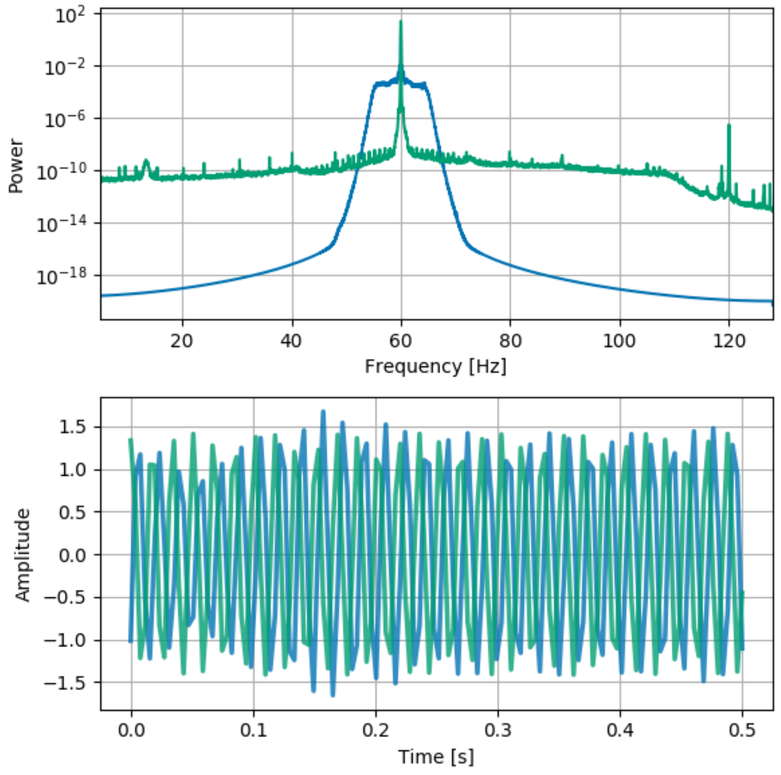
Note the y-axes! Also be aware of re-scaling at the end

PREPROCESSING: BANDPASSING (FEATURE EXTRACTION)

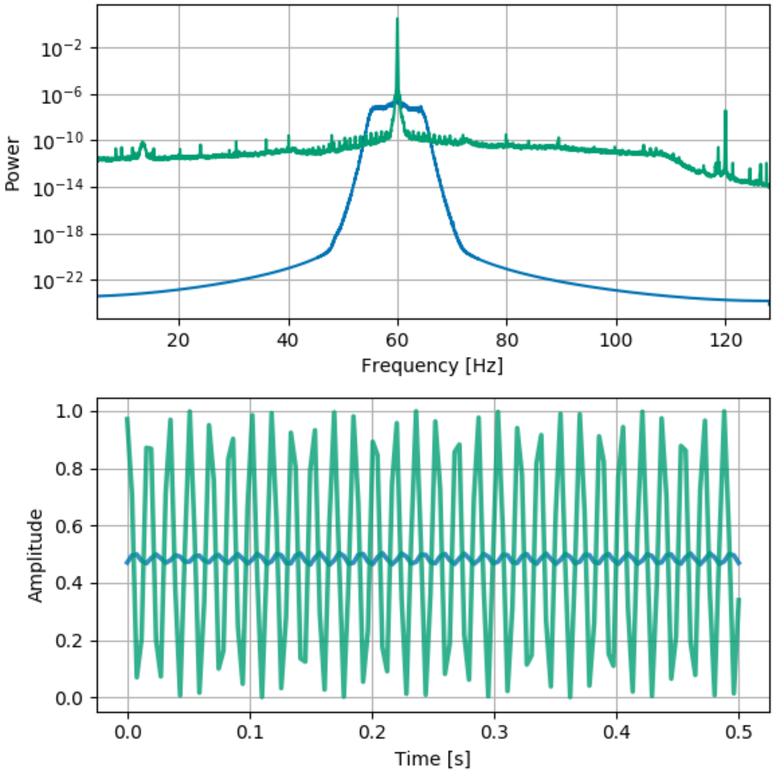
Before



Standardizing $y := \frac{y - \mu}{\sigma}$



MinMax $y := \frac{y - \min(y)}{\max(y) - \min(y)}$



(This phase delay will come back to bite us)

FEATURE IDENTIFICATION

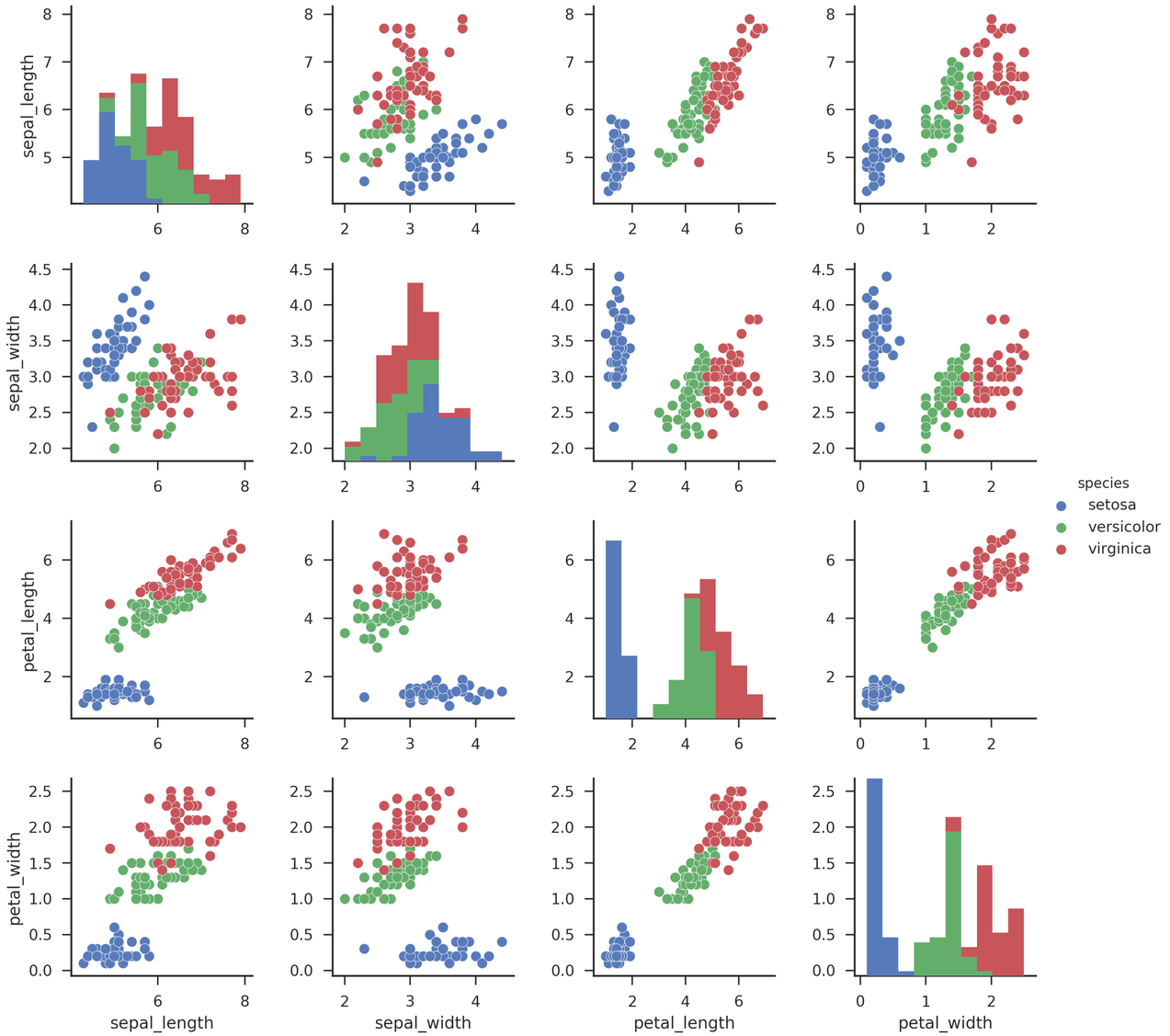
- Make sure your data contains the information you intend to understand

-Lots of data != better dataset

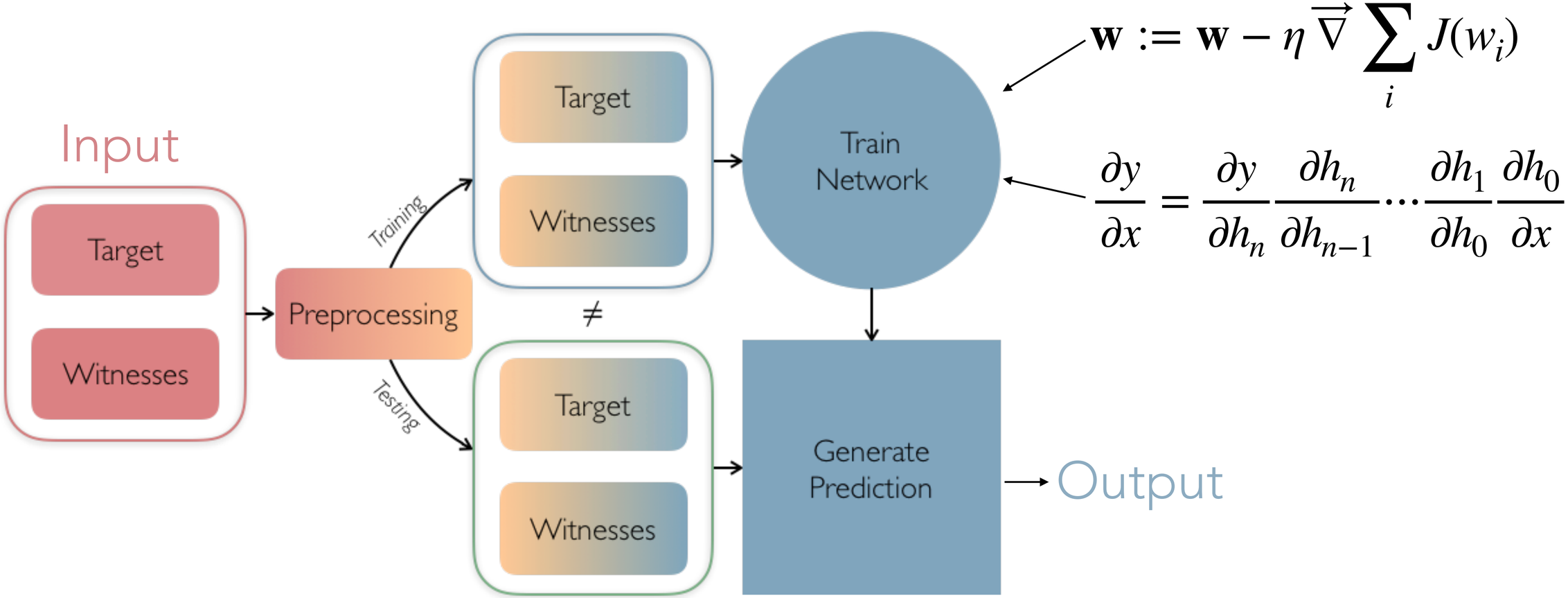
- Bad data = bad results

- Visualizations are very helpful!

Iris Dataset



GENERAL WORKFLOW



RECURSIVE NEURAL NETWORKS

WHY WE NEED THEM

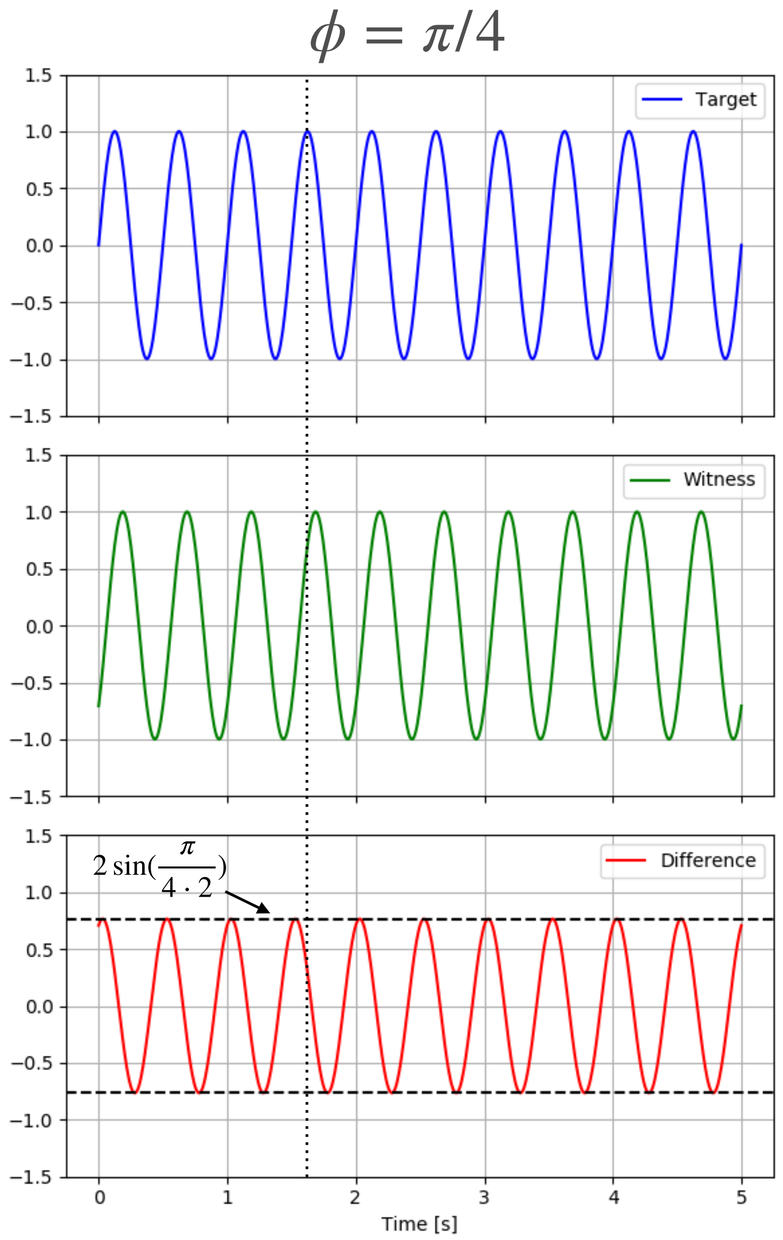
If we try to predict $\sin(2\pi ft)$ given $\sin(2\pi ft + \phi)$, 1-to-one predictions will fail because

$$\sin(2\pi ft + \phi) - \sin(2\pi ft) = 2\sin(\frac{\phi}{2})\cos(2\pi ft + \frac{\phi}{2})$$

So if $\phi \neq 0$, the RHS is not zero.

In other words, the network is always "chasing" the loss function which will oscillate with the target. For small phases, the prediction scales linearly with the phase mismatch.

The network needs to "know" where it is on the curve



WHY WE NEED THEM

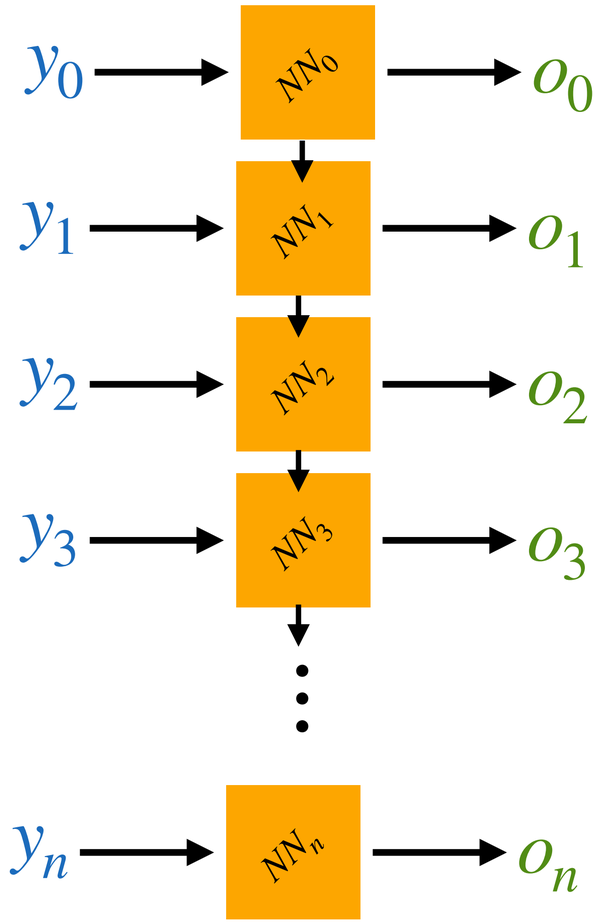
BERT

```
sentences = ['To be or not to be',
             'Be that as it may',
             'Was it true that they were ill',
             'They may be running late',
             'Cheetahs were running around']
```

- *Given a word, could you predict the rest of the sentence (don't get stuck in a loop)?*
- *'Was… were…' so the beginning of the sentence needs to be remembered!*

UNWRAPPING THE DATA ARRAY

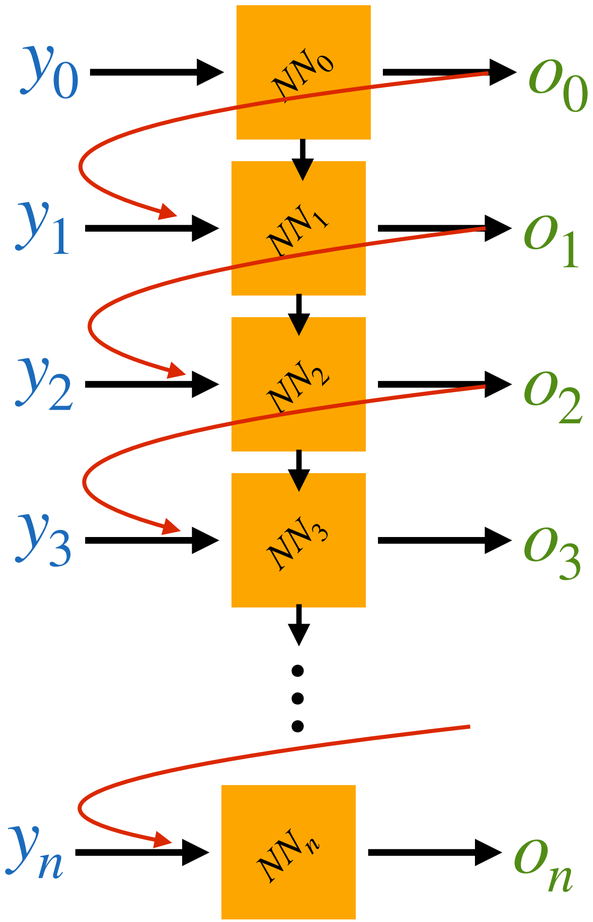
Feed forward 1-to-1



Send Predictions

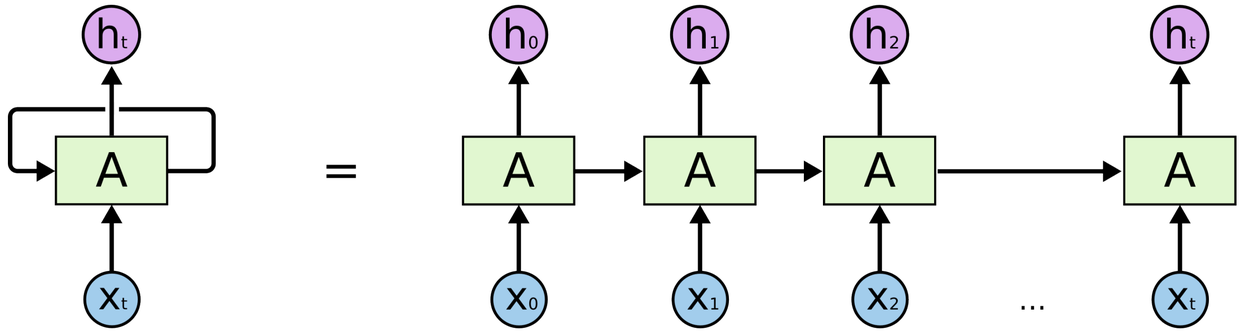
To Inputs

Basic RNN



SENDING OUTPUTS TO INPUTS

- Each input now "sees" the past and how it affected the network
- Important for evolving dependencies and forecasting
- The "cells" (labeled "A") can get rather complicated



Unfolded RNN

"A" = Cells

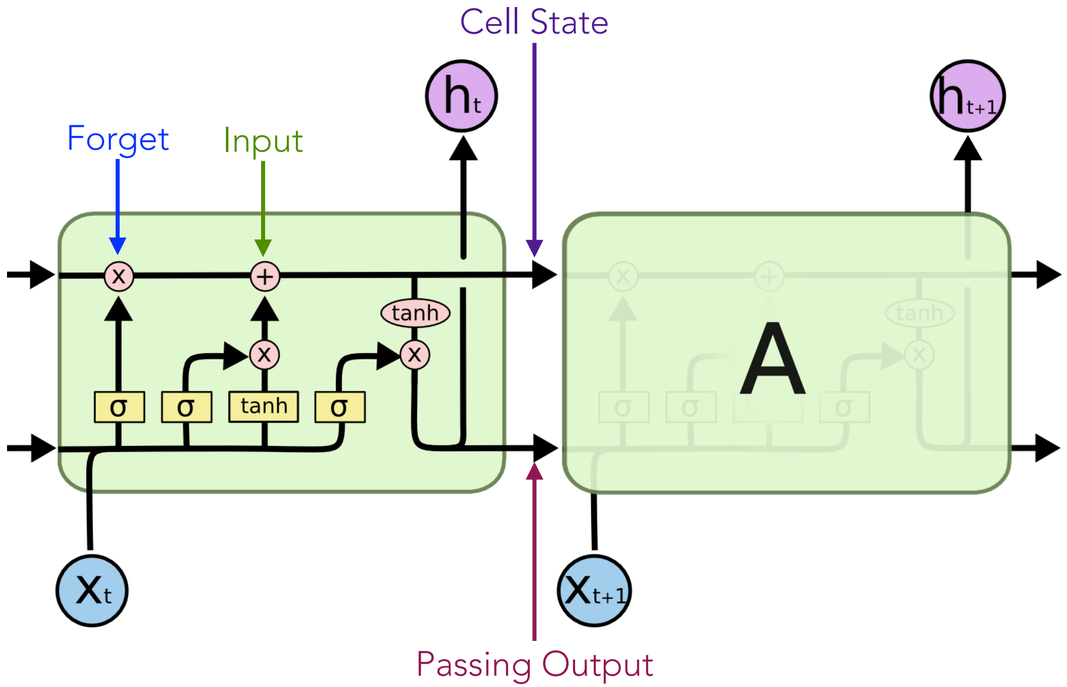
x_i = inputs

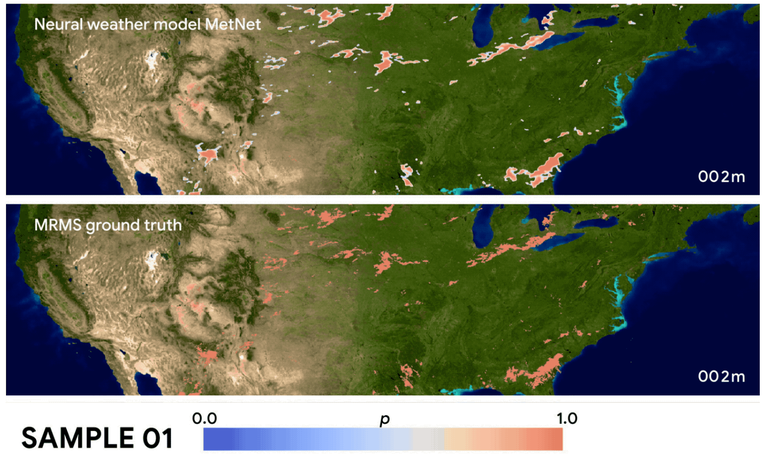
h_i = outputs (of hidden layers)

Image Credit: Chris Olah

LSTMS VS VANILLA RNNS (OR GRUS)

LSTMs (Long Short Term Memory) cells use “forget” gates which aim to de-rank transient information





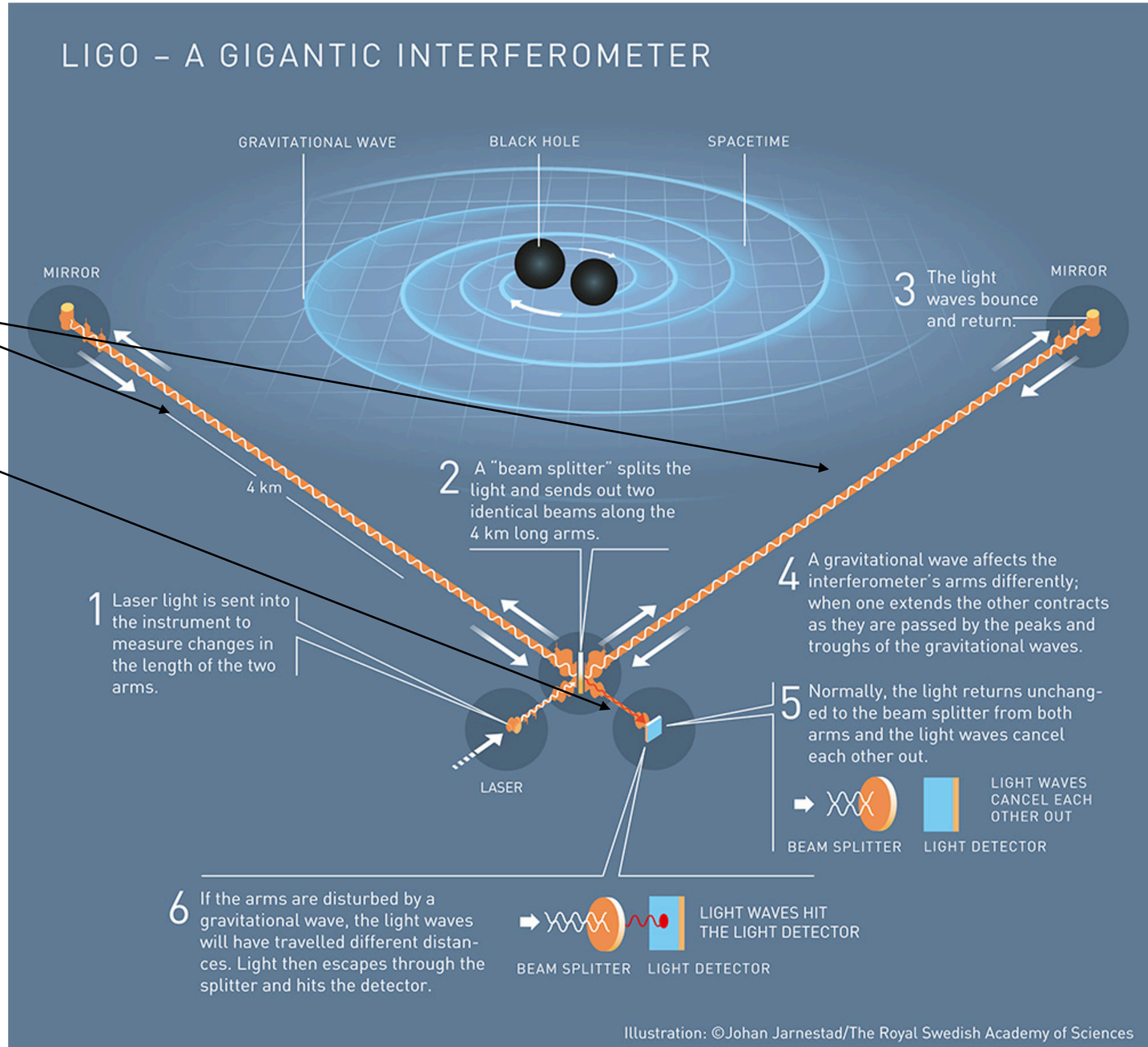
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$

Image Credit: Chris Olah

LIGO DIGRESSION

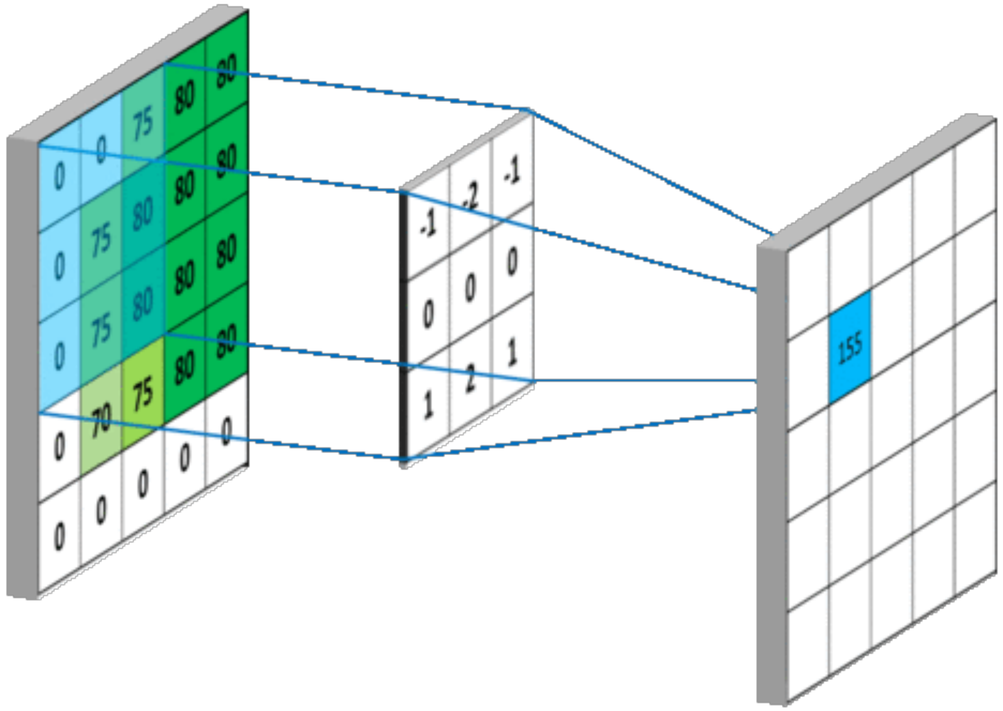
If the lengths of these ‘arms’ change, then the interference pattern of the light at the output changes.

Gravitational waves AND noise can do this though :/



CONVOLUTIONAL NEURAL NETWORKS

HOW CNNS ARE DIFFERENT (FILTER KERNELS & ABSTRACTION)



(train the filter weights)

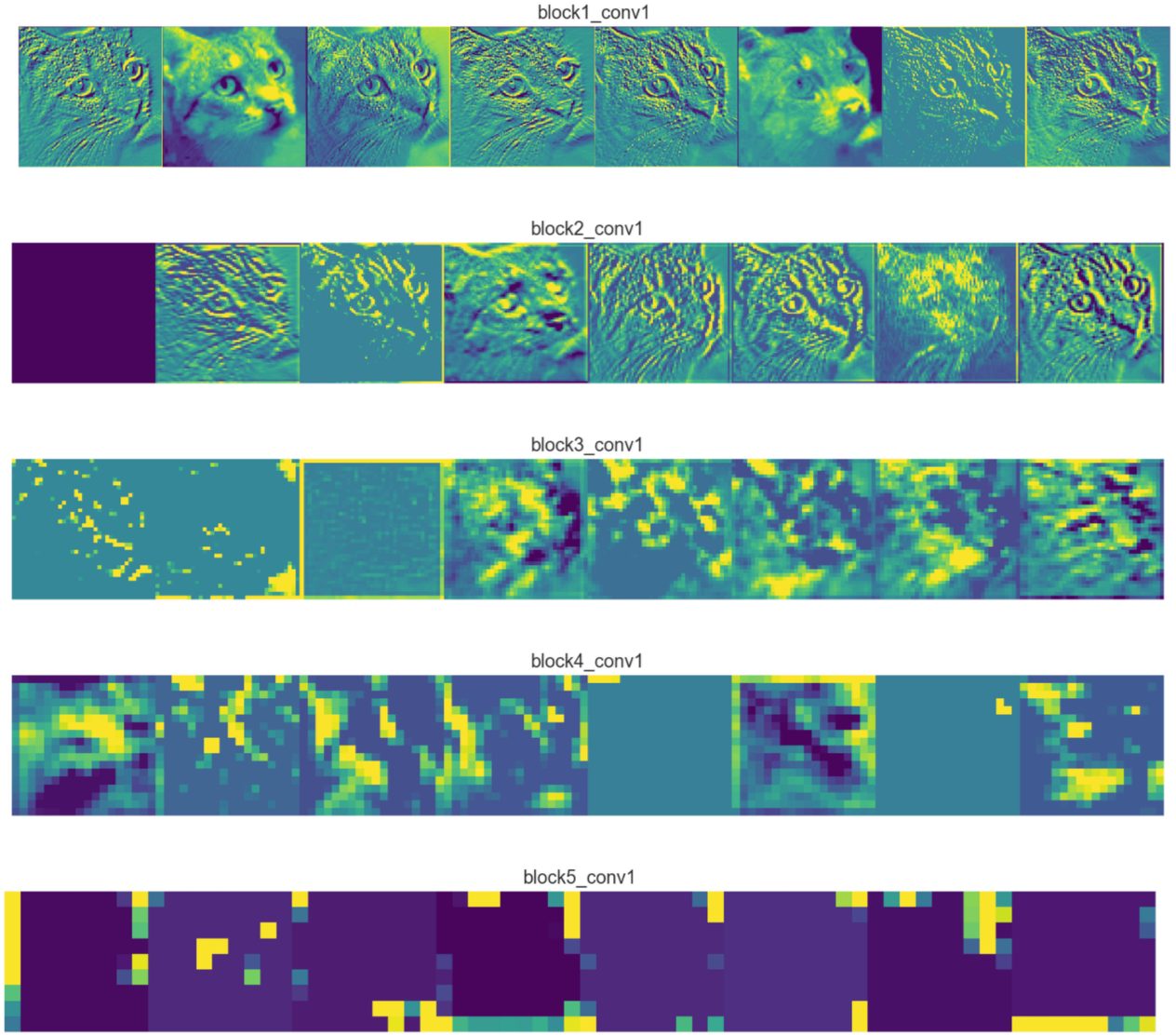


Image Credit: Towards Data Science

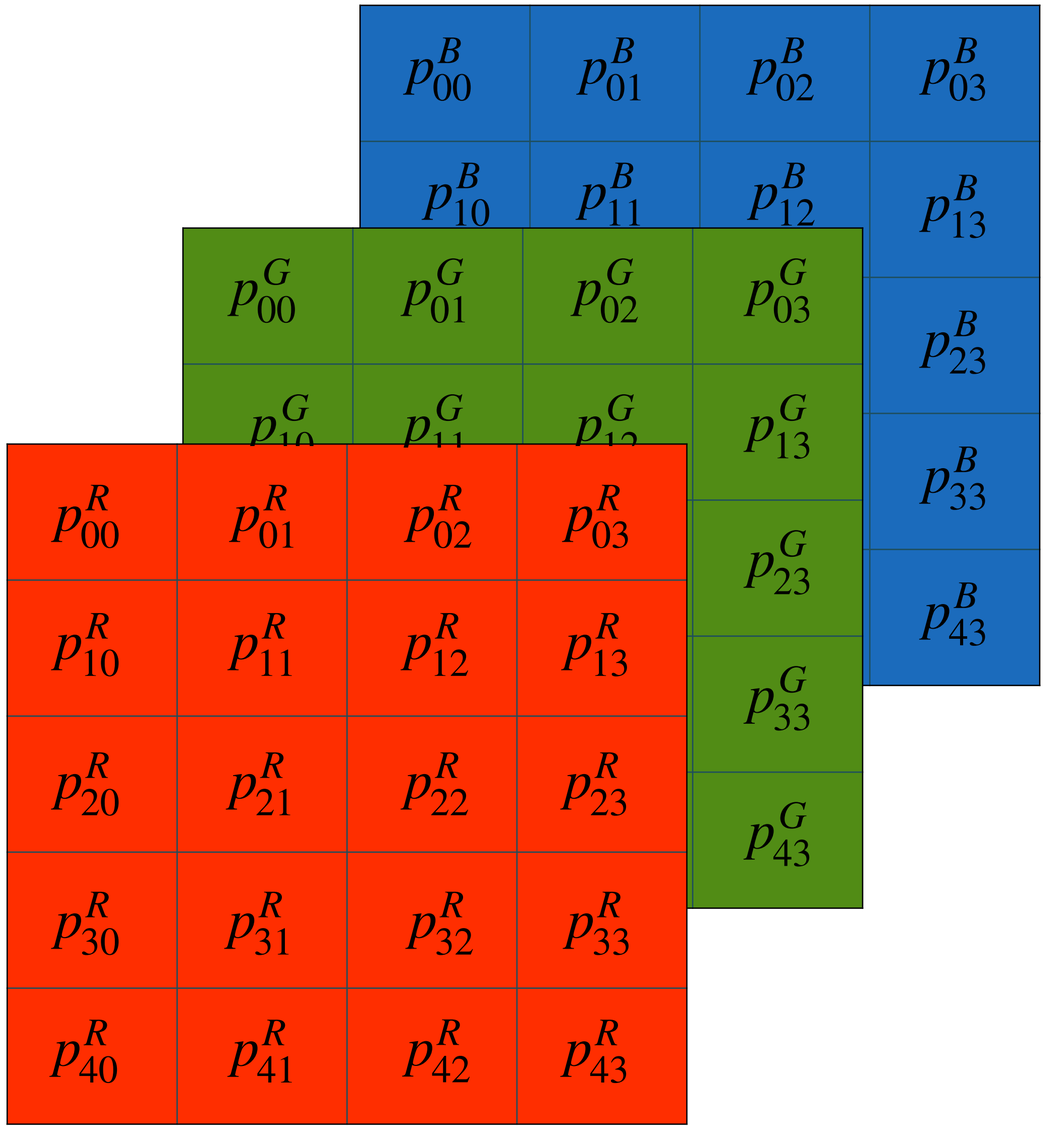
IMAGES ARE 3-DIMENSIONAL

Colored pixels are specified by 5 values (R,G,B and (i,j)) as $p^k_{i,j}$

Greyscale only requires 3 (amplitude, (i,j))

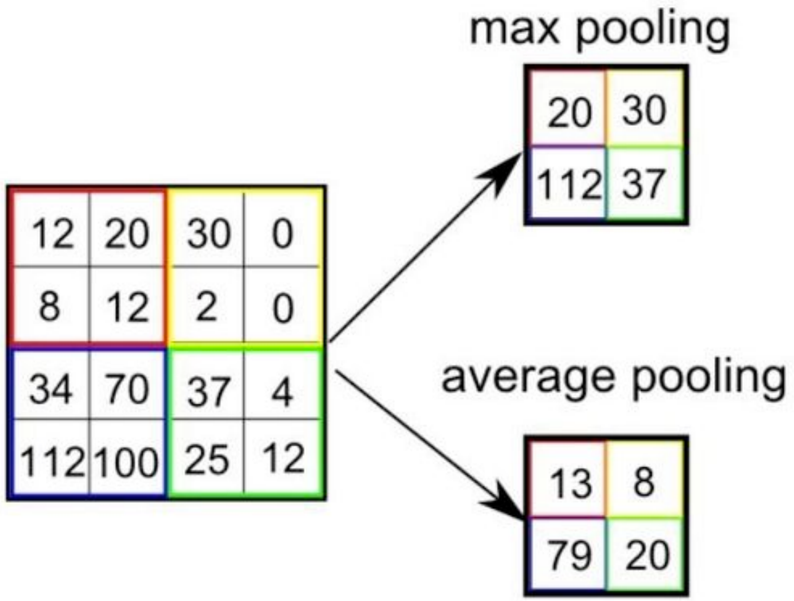
8-bit color means $2^8 = 256$ colors.

Typically min max scale to 0-1



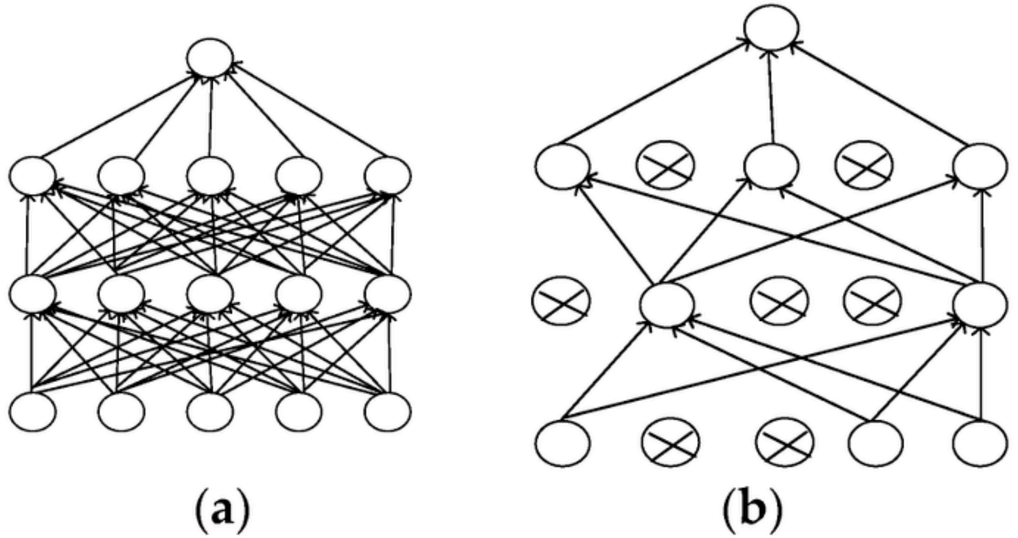
POOLING AND DROPOUT

Max & Avg Pooling



Can be 1, 2 or 3 dimensional. Minimizes dimensionality and can help training

Dropout



Decrease biased learning and take advantage of network depth

Image Credit: MDPI

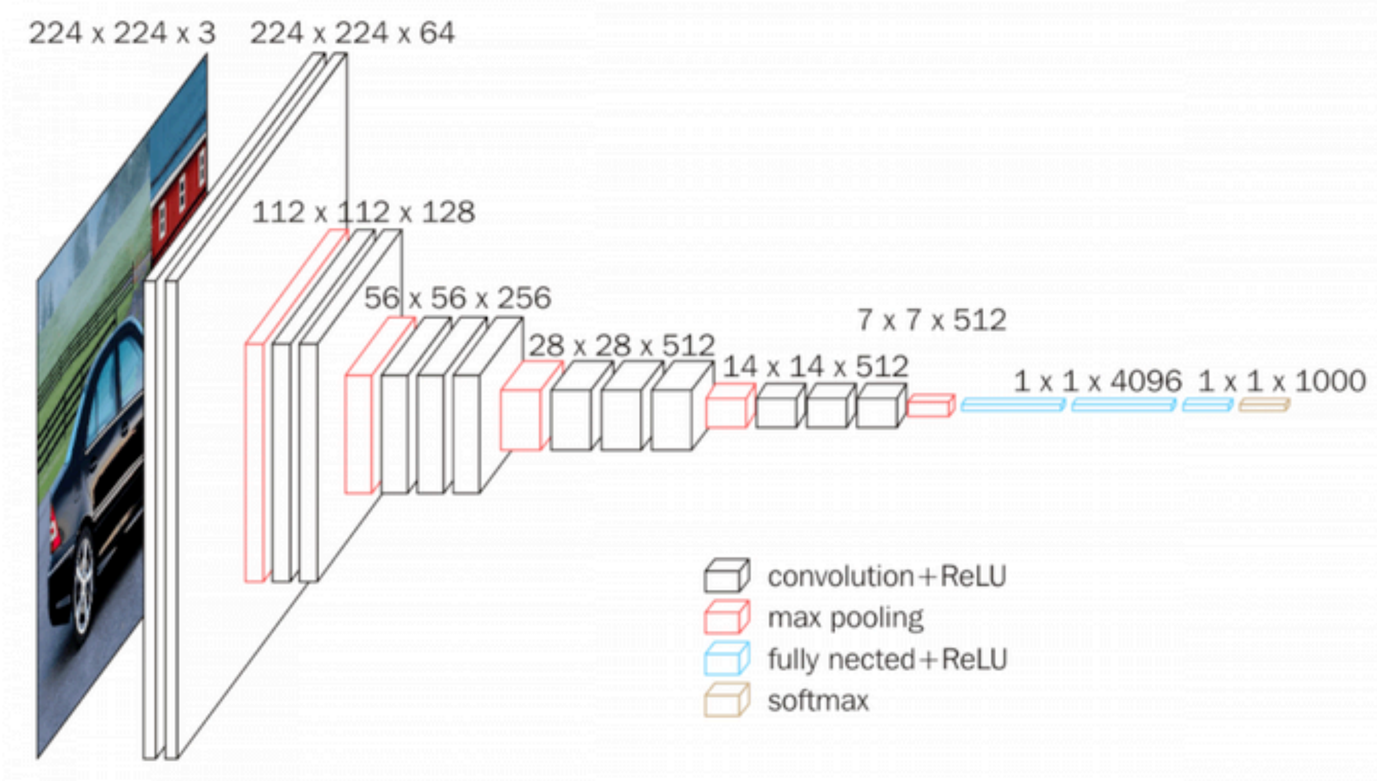
GENERAL ARCHITECTURE

Conv, ReLU, Maxpool

Some combination / repetition of these layers characterizes most “vanilla” CNNs.

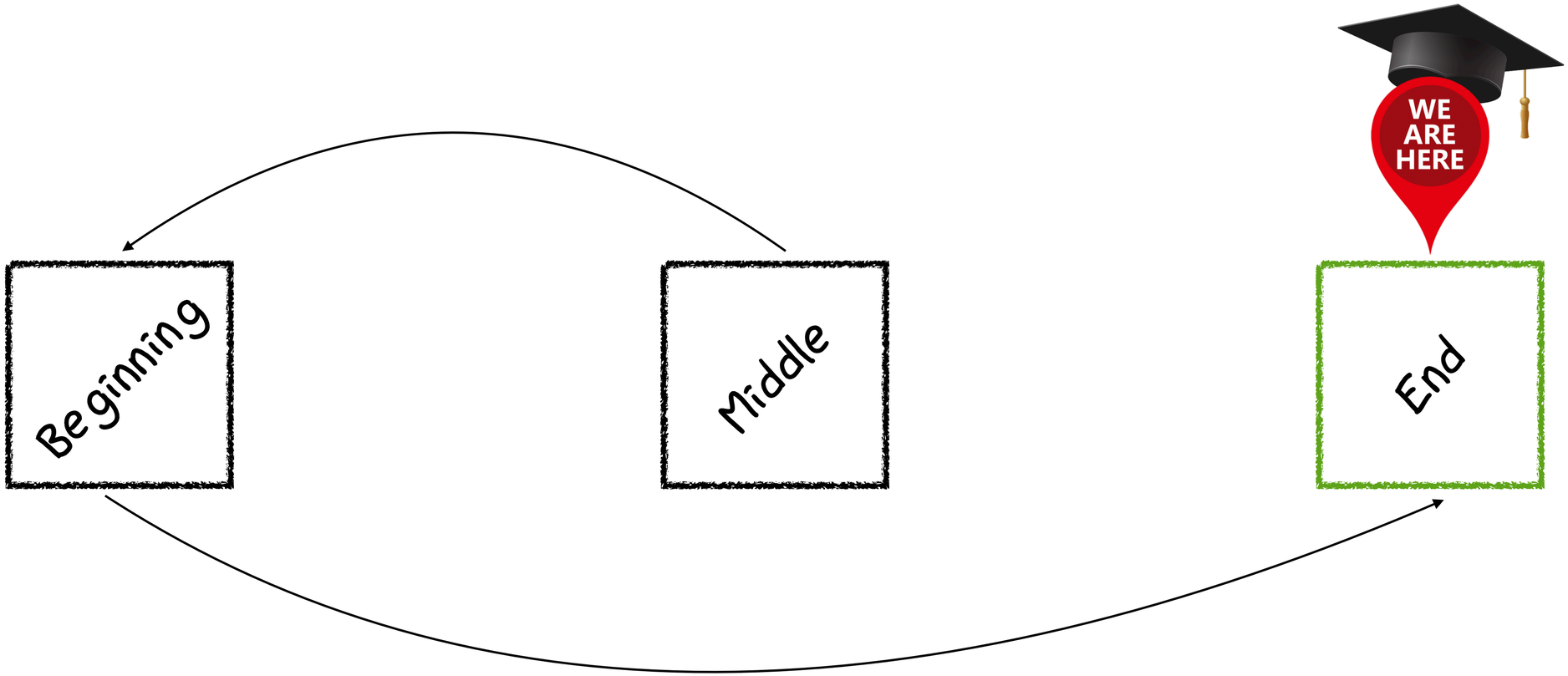
The output activation depends on the problem.

RCNNs, U-Nets and a few other are quite different



VGG Network (arxiv)

WRAPPING UP WITH THEORY



LET'S RUN SOME NETWORKS!

Github: https://git.ligo.org/rich.ormiston/nrt_seminar

```
$ git clone git@git.ligo.org:rich.ormiston/nrt_seminar.git
```

-or-

```
$ git clone https://git.ligo.org/rich.ormiston/nrt_seminar.git
```

FUN LINKS

- Tensorflow Playground
- CNN Activation Map Visualization for MNIST
- Facial Recognition Development

EXTRA SLIDES