Biologically Inspired Computation

Dr Marta Vallejo

m.vallejo@hw.ac.uk

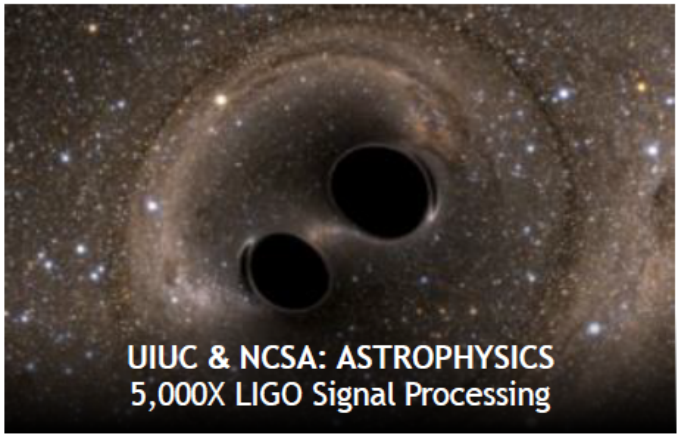
Lecture 4

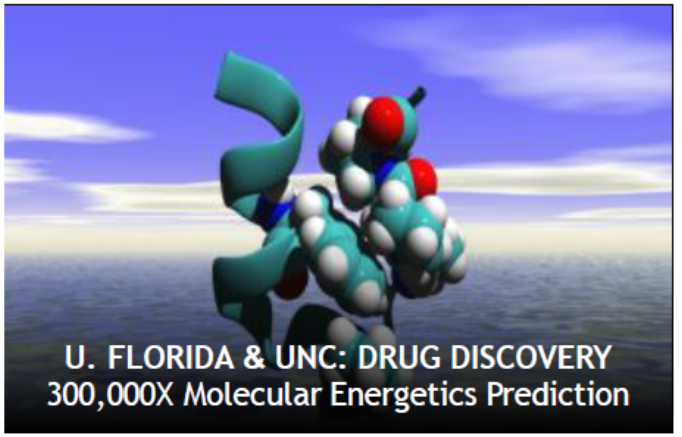
Introduction to Deep Neural Networks

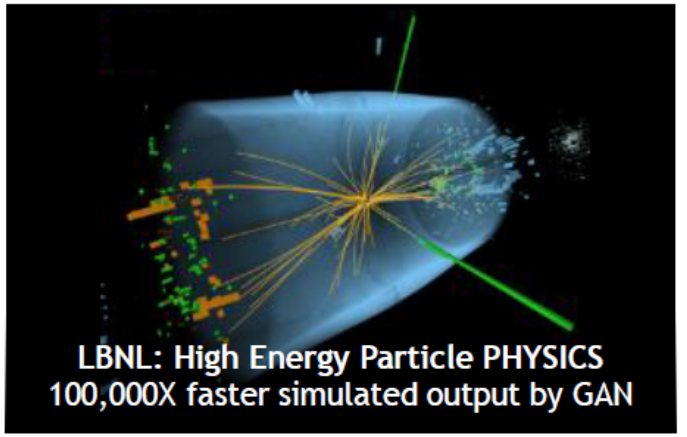
- Big Data & Deep Learning
- Scalability of ML algorithms
- Deep Learning: More Pros/Cons
  - Feature Engineering vs. DNN Learning
  - Transfer Learning
- Hierarchical Feature Representation
- Deep Learning Libraries
- Infrastructure: on-premise vs. cloud
- Hyperparameters
- Experimental Nature of Deep Learning
- Hidden Layers and number of Units
- Techniques to avoid overfitting
  - Regularisation
  - Early Stopping
  - L1 & L2 Regularisation
  - DropOut
- Activation Functions for Deep Learning
  - ReLU Activation Function
  - ReLU and Sparsity
  - Dying ReLU Problem
  - Variants of ReLU

DEEP LEARNING COMES TO HPC

Accelerates Scientific Discovery

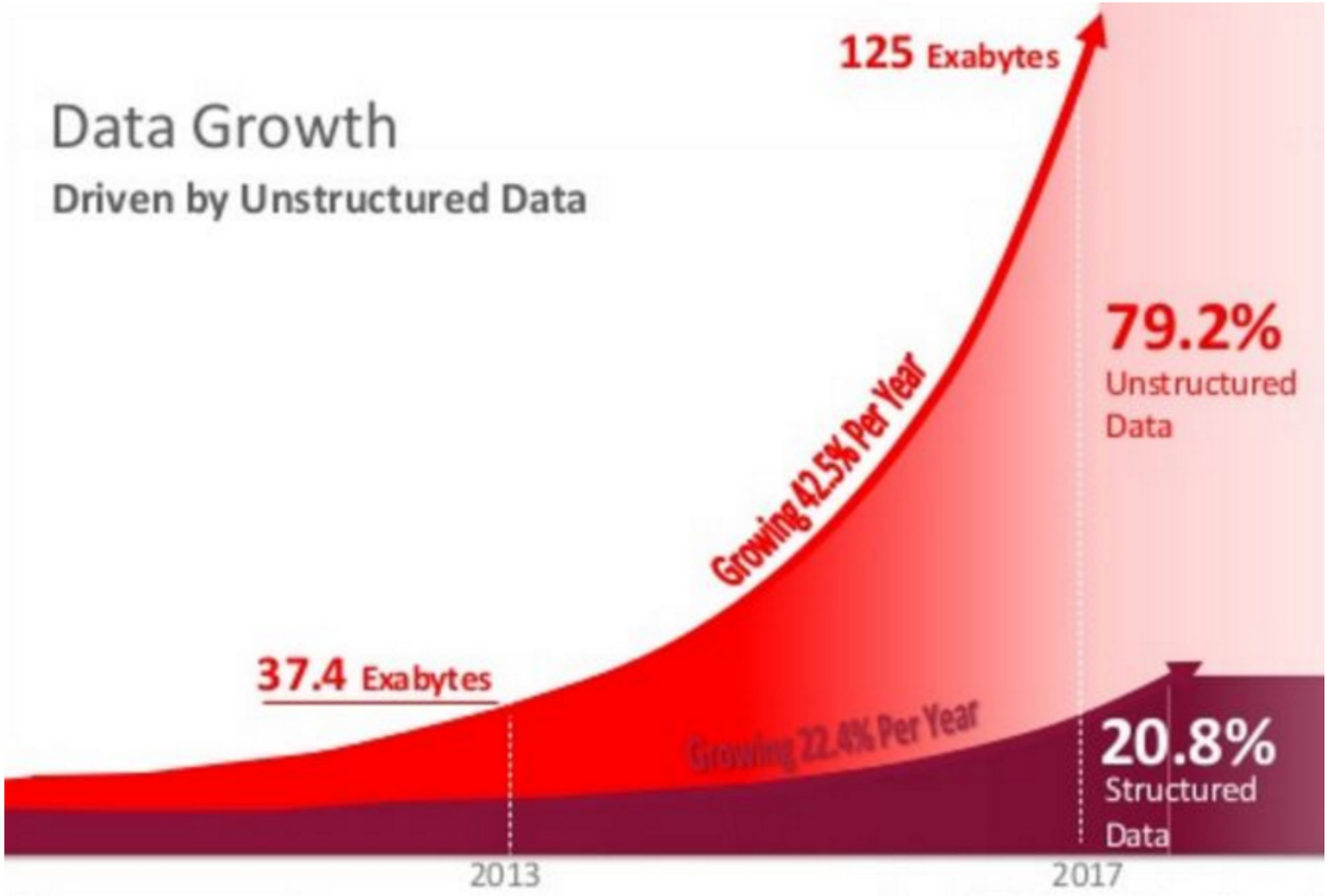






Big Data & Deep Learning

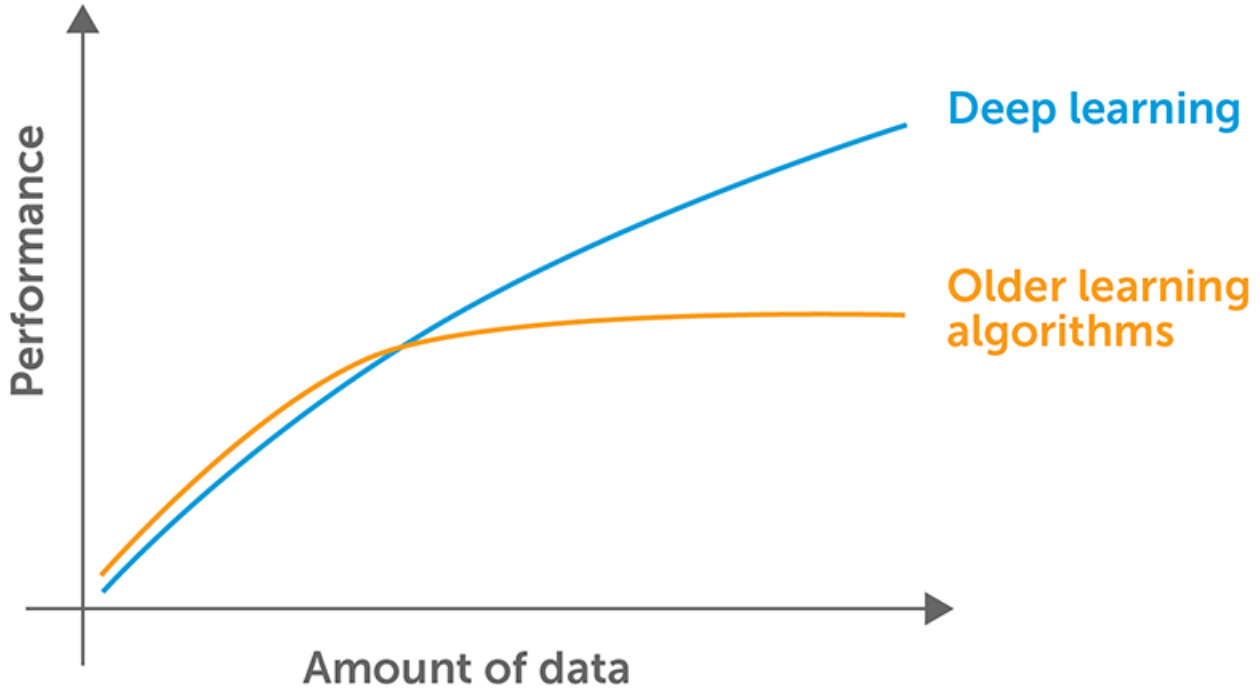
New applications and devices have caused an **exponential growth** of data. Its analysis and subsequent transformation into better understanding and prediction of different phenomena is a critical issue.



Source: ISD - 2014, Structured Data vs. Unstructured Data: The Balance of Power Continues to Shift

Scalability of ML algorithms

Most machine learning (ML) algorithms are as good as or even better than Deep Learning for small amount of data. However, traditional ML algorithms do not scale well for huge amount of data (mostly unstructured). Why? Because of algorithmic complexity.



How do data science techniques scale with amount of data?

Deep Learning performance is strongly correlated with the amount of available training data

Deep Learning: More Pros/Cons

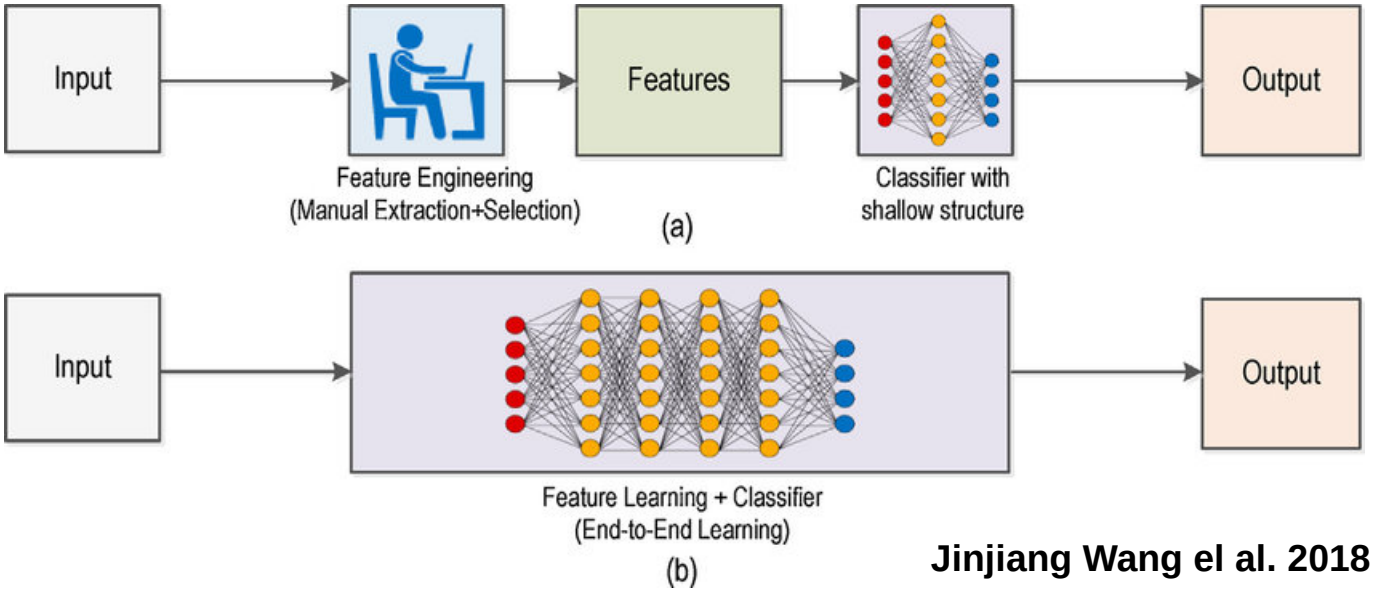
Pros:

- Automatically learn the high-level features representation. It is no longer needed expert knowledge to select the best features.
- Modularity: DNN can be composed like LEGO bricks
- Able to do Transfer Learning
- Advances made in the algorithms itself to making them run much faster than before
- Less probable to get stuck in local optima

Cons:

- Backpropagation does not work well if it is randomly initialised
- Requires huge amount of data for training
- Expensive computation power needed for training and testing
- "black box" nature: you do not know how and why your network came up with a certain output.
- If you need more control over the details of your algorithm, the duration of the development can be significant.

Feature Engineering vs. DNN Learning



Jinjiang Wang el al. 2018

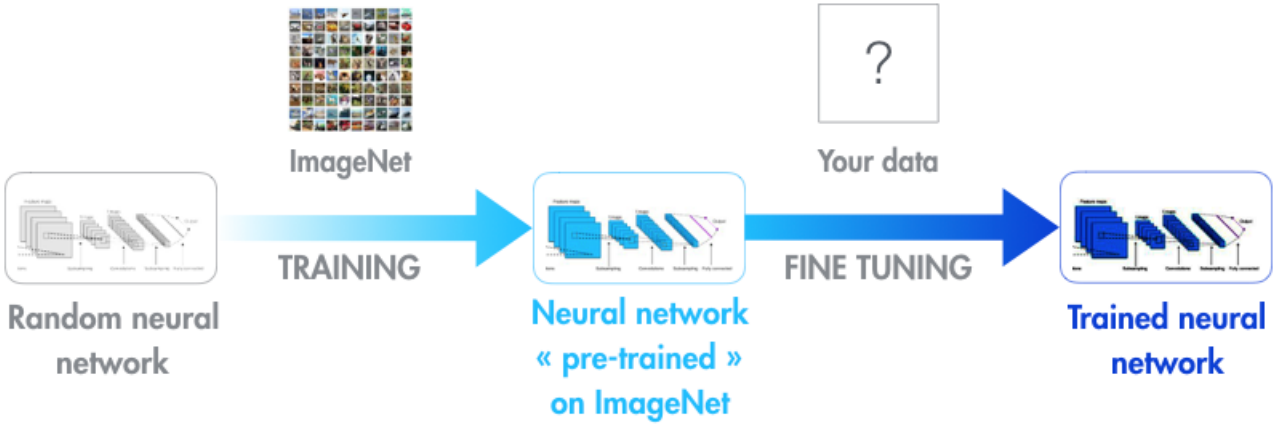
Feature engineering is the process of using domain knowledge (normally from experts) to create features that make machine learning algorithms to work. This is normally a **time consuming** process. These features would be used as our input.

In DNN we can skip this step and include the raw data as input.

Transfer Learning

Transfer learning is the possibility of using **pre-trained** architectures in Deep Learning by making small changes. Conventional machine learning models use domain specific features, making difficult to share their weights among different problems.

Instead of building a model from scratch we can use a model created by some one else to solve a similar problem as a starting point (exploit commonalities).

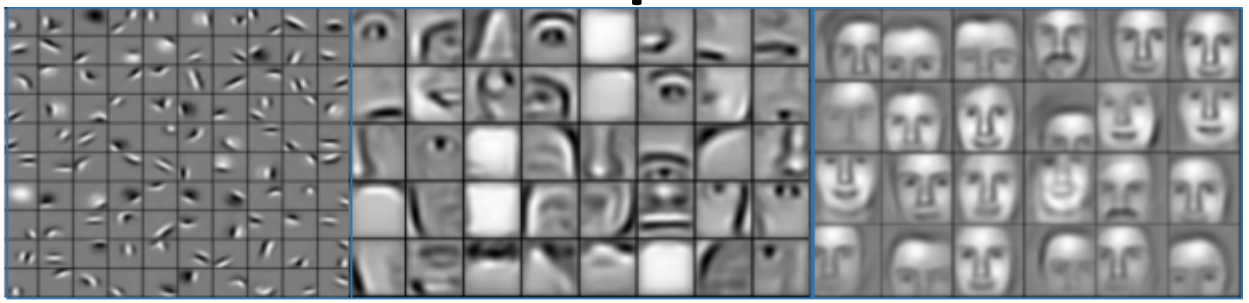


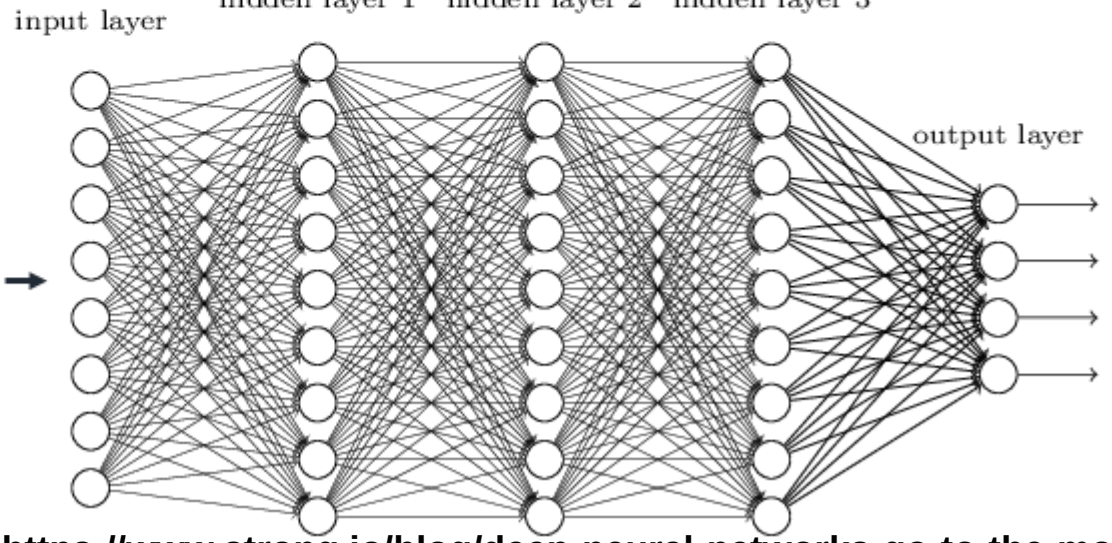
From Giles Wainrib

This is called "**domain adaptation**"

Hierarchical Feature Representation

Deep neural networks learn hierarchical feature representations





https://www.strong.io/blog/deep-neural-networks-go-to-the-movies

Each successive layer in a neural network uses features from the previous layer to learn more complex features. Layers closer to the output contains more abstract features.

Image recognition: Pixel → edge → texton → motif → part →object

Text: Character → word → word group → clause → sentence

Speech: Sample → spectral band → sound →…→ phone → phoneme → word

Deep Learning libraries

Many open source deep learning libraries are available online:

Comparison of deep learning libraries

	Core	Speed for batch* (ms)	Multi-GPU	Distributed	Strengths
Caffe	C ++	651.6	O	X	Pre-trained models supported
Neon	Python	386.8	O	X	Speed
TensorFlow	C ++	962.0	O	O	Heterogeneous distributed computing
Theano	Python	733.5	X	X	Ease of use with higher-level wrappers
Torch	Lua	506.6	O	X	Functional extensionality

Notes: Speed for batch* is based on the averaged processing times for AlexNet with batch size of 256 on a single GPU ; Caffe, Neon, Theano, Torch was utilized with cuDNN v.3 while TensorFlow was utilized with cuDNN v.2.

There are still no clear front-runners, and each library has its own strengths

Deep Learning libraries

According to benchmark test results:

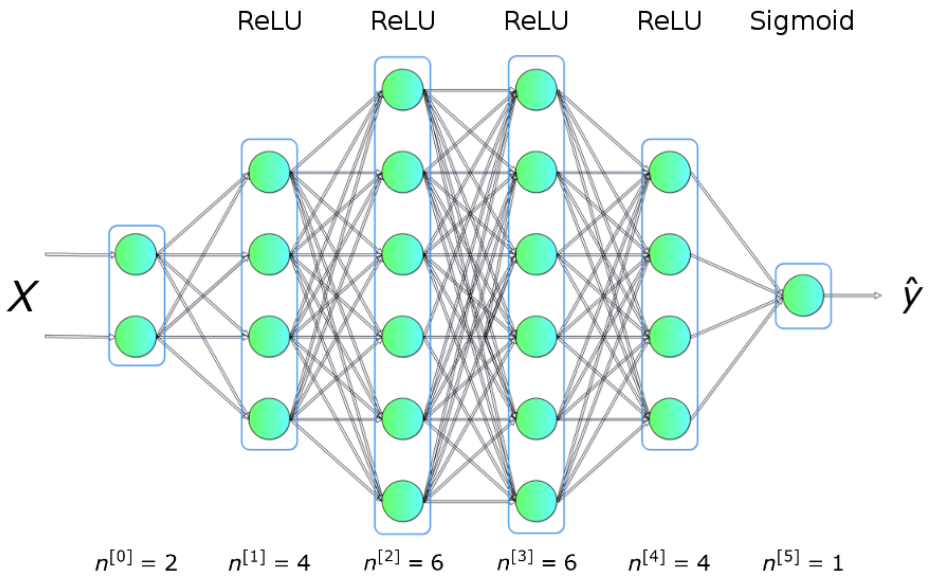
- **Neon (Python)** shows a great advantage in the processing speed.
- **Caffe (C++)** advantages in terms of pre-trained models
- **Torch (Lua)** good at functional extensionality.
- **Theano (Python)** provides a low-level library to define and optimise mathematical expressions
- **TensorFlow (C++ with a Python interface)**: currently shows limited performance but is undergoing continuous improvement. It can also take advantage of Keras.

Higher-level wrappers developed on top of Theano to provide more intuitive interfaces:

- Keras
- Lasagne
- Blocks

Deep Learning libraries: pros

- No need to deep understanding of the maths behind.
- No bugs to worry in the structure construction.
- No need of high-level programming skills.



The code for this network in Keras:

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(4, input_dim=2,activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=50, verbose=0)
```

https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba

Deep Learning Infrastructure: On-premise vs. Cloud

On-premise

- Need of specialised hardware: **Graphical Processing Units** (GPU)
- Quality of results depends on quality of the GPU
  - Can be an expensive investment
  - Can achieve fast results
- High power consumption (~200Hz)
- Need fast data transfer from local storage
- No privacy issues
- Set-up and Maintenance

Nvidia (eg. DGX systems)
Lambda Labs
AMAX
Boxx

Cloud computing paradigms

- Network delay
- For small-medium projects
- Little upfront investment
- User privacy and data security
- Speed of Hardware upgrade
- With high level of usage, this option is more expensive

Google Cloud Platform
Amazon Machine learning
Paperspace
Azure Machine Learning studio

Hyperparameters

Apart from your normal set of parameters:

$$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \dots$$

As we did previously, we need to decide other type of values for our algorithm. These choices determine drastically the **performance** of our model:

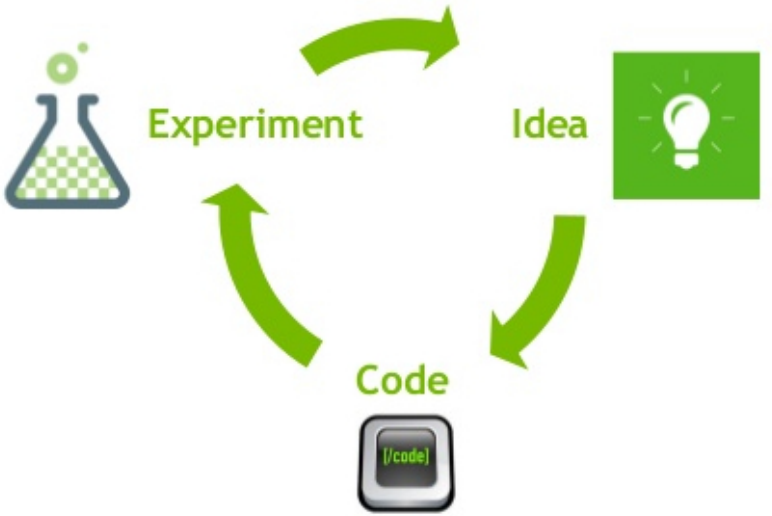
These variables are called **hyperparameters** and in this type of architecture are:

- Learning Rate α
- Number of Iterations
- Number of Hidden Layers
- Number of Hidden Units in each layer
- Choice of the activation function in each layer
- Others we did not see yet

Experimental Nature of Deep Learning

Good **generalisation** properties depend on the ratio of trainable parameters (weights (|w|), biases (|b|), hyperparameters, the size of the input data. . .

It is a very **empirical** process (trial and error)



Steps:

- Take a look to other pre-trained models if they are similar to the problem
- Otherwise, start with a small configuration and extended checking the increment in the performance.

  Tuning parameters is a dynamical process. Changes in HW and data can perturb the suitability of your current configuration

Hidden Layers and Number of Units

We can use different combination of layers: fully connected, recurrent, convoluted or pooling

You cannot analytically calculate the number of layers or the number of nodes to use per layer in an artificial neural network to address a specific real-world problem.

Many hidden units within a layer with **regularisation** techniques can increase accuracy → Just keep adding layers until the test error does not improve any more

Empirically, greater depth does seem to result in better generalisation for a wide variety of tasks

Smaller number of units may cause underfitting.

To know more about neural architectures:

https://www.automl.org/automl/literature-on-neural-architecture-search/

Techniques to Avoid Overfitting

There are different techniques used to avoid **overfitting** in DNN and thus achieve good **generalisation** performance.

Here are mentioned the most important ones:

- Add more data
- Use data augmentation: like randomly rotating images, zooming in, adding a colour filter etc. Data augmentation should be applied only to the training set and not on the validation/test set.
- Use architectures that generalise well
- Add regularisation
- Reduce architecture complexity.

As a general rule, you can start by overfitting the model and then take measures against overfitting. Note that your model is underfitting if the accuracy on the validation set is higher than the accuracy on the training set.

Regularisation

Main idea: Large number of parameters cause overfitting.

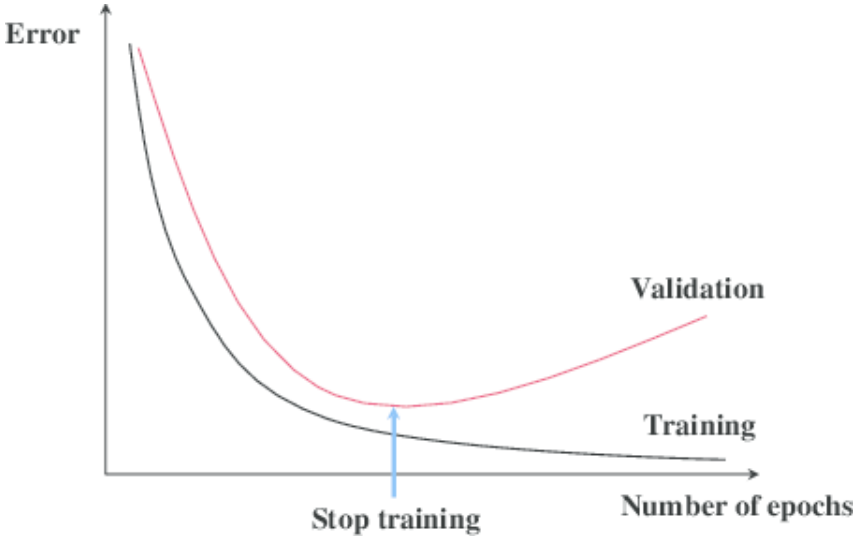
There are different regularisation techniques that can be applied to Deep Neural Networks:

- **Weight decay (L1 & l2 regularisation)**: adds a penalty term for the size of the weights.
- **Dropout**: the most common used method
- **Batch normalisation**: makes the input to each layer have zero mean and unit variance. It is inserted after dense layers and before the nonlinearity in convolutional neural networks. We will see more later

  To know more:
  https://arxiv.org/pdf/1502.03167.pdf

- **Early Stopping**: monitor the overfitting of the deep learning process.

Early Stopping



Early Stopping: a neural network is optimised using a training set. A separate test set is used to **halt training** to mitigate overfitting and improving the generalisation.

- Use the training algorithm to adjust the weights of the networks so that it gives better predictions for each and every input-target.
- Pass the entire test set to the network, make predictions, and compute the value of network test error.
- Compare the test error with the one from the previous iteration. If the error keeps decreasing, continue training; otherwise, stop training.

Early stopping also saves a significant amount of time.

L1 & L2 Regularisation

L1 is also called Lasso regression and L2 Ridge regression:

They are called **weight decay methods**. They add a penalty term to the objective loss function so that weight parameters converge to smaller absolute values.

L1: $\mathcal{L}_{new} = \mathcal{L} + \frac{\lambda}{2}\sum |W|$ **L2:** $\mathcal{L}_{new} = \mathcal{L} + \frac{\lambda}{2}\sum W^2$

where $\mathcal{L}$ Is the loss function (cross entropy, mean squared error...)

The difference between L1 and L2 is the **penalty term**: the power of w (1 in L and 2 in L2). The difference in their properties:

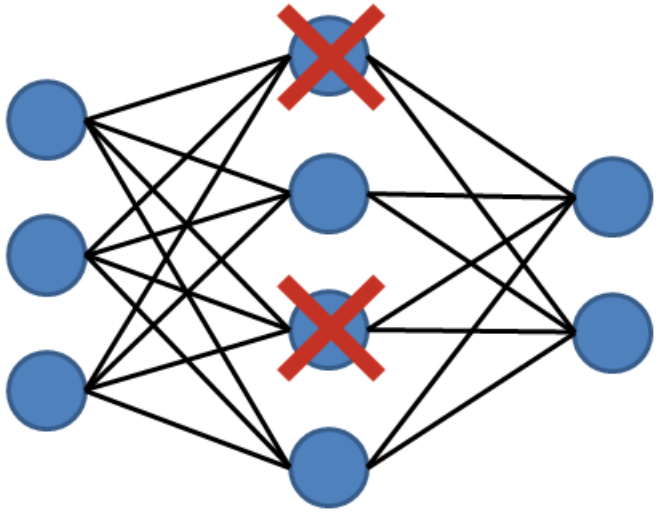
L1 does not have an analytical solution, but L2 does (can be calculated efficiently). However, L1 solutions are **sparse** (only few entries are non-zero) which allows it to be used in sparse algorithms, which makes the calculation more computationally efficient.

DropOut

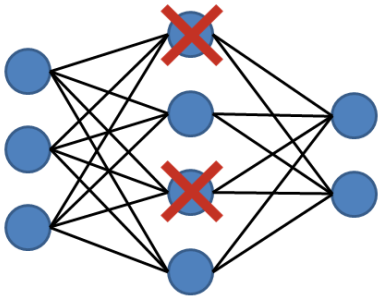
Dropout randomly removes hidden units during training and can be considered an ensemble of possible subnetworks.

Generally, it is used a small dropout value of 20%-50% of neurons with 20% providing a good starting point. A too low probability value has minimal effect and a too high value results in **under-learning**.

It is likely to get better performance when dropout is used on **larger networks**, giving the model more of an opportunity to learn independent representations.



DropOut



To enhance the capabilities of dropout, a new activation function, **maxout** and a variant of dropout for RNNs called **rnnDrop** have been proposed.

To know more about maxout:
https://cs.adelaide.edu.au/~carneiro/publications/226.pdf

And abount rnnDrop:
https://ieeexplore.ieee.org/document/7404775/

Weight decay, batch normalisation and dropout are only applied in **fully connected** (and convolutional) layers at the end of the model. Why?

Dropout causes lost of information. If the lost is done in the first layer, the lost is propagated to the whole network. Therefore, a good practice is to start with a low dropout in the first layer and then gradually increase it.

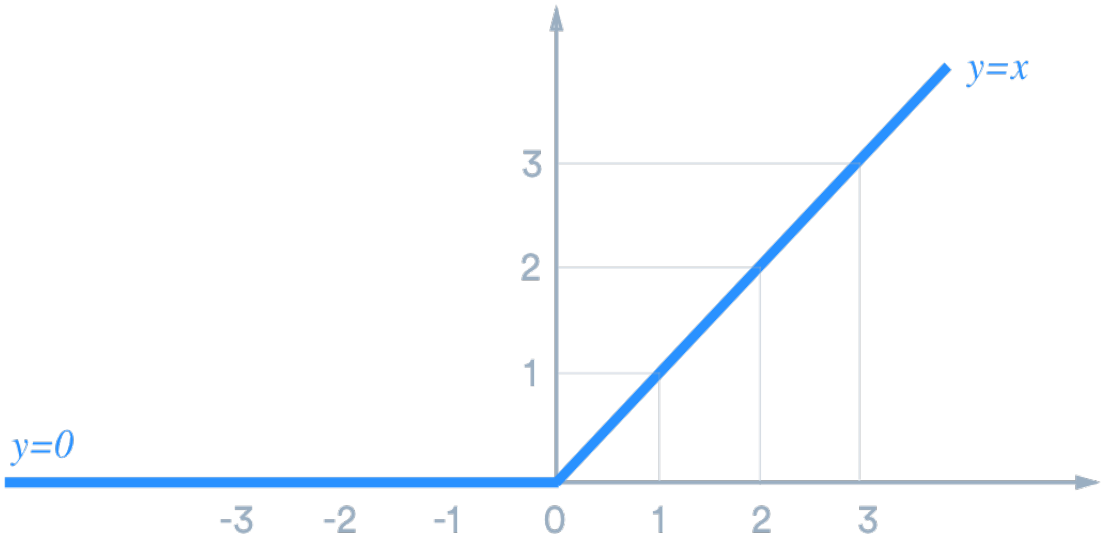
Activation Functions for Deep Learning

New activation functions were proposed to be able to deal with deep architectures. We are going to see:

- ReLU (Rectified Linear Unit) Activation Function.
- ReLU and Sparsity
- Dying ReLU problem
- Variants of ReLU

ReLU (Rectified Linear Unit) Activation Function

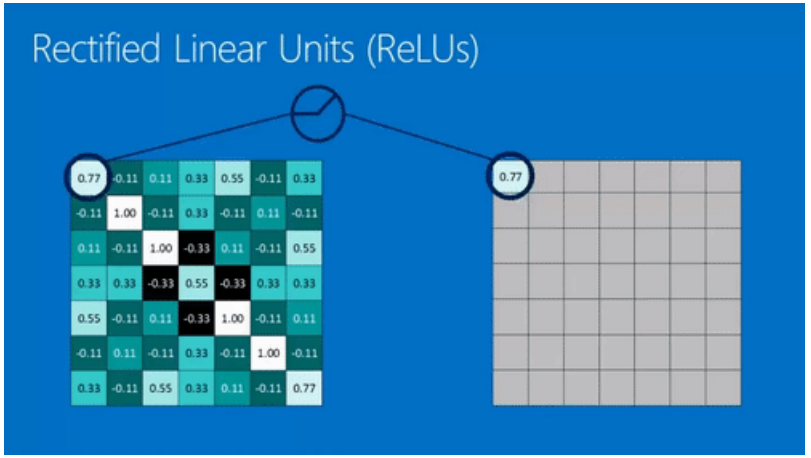
The ReLU function is the most used activation function in deep neural networks, especially in the case of Convolutional Neural Networks.



$$ReLU = max(0, z)$$

The ReLU function and its derivative both are monotonic. It is non-linear and differentiable. It has a range from 0 to infinity. It is linear (identity) for all positive values, and zero for all negative values.

ReLU (Rectified Linear Unit) Activation Function



https://gfycat.com/gifs/detail/PerfectCalculatingAxisdeer

ReLU returns a gradient zero for negative values. This make it less computational expensive than sigmoid/tanh functions.

It trains and **converges faster**. This linearity means that the slope does not plateau or "saturate," when values are large. It does not have the vanishing gradient problem suffered by other activation functions like sigmoid or tanh.

In the paper that won the 2012 ImageNet LSVRC-2012 competition "ImageNet Classification with Deep Convolutional Neural Networks" said that ReLU speed-up by 6 times over the same accuracy over tahn.

ReLU and Sparsity

Sparse networks are networks that only part of the nodes fires, the ones that deals with different aspects of the problem. Sparsity results in concise models that often have better predictive power and less overfitting/noise.

A sparse network is faster than a dense network, as there are fewer things to compute.

Sigmoid/Tanh functions fire all in a similar way. ReLU fires less, which is more efficient.

For random values given in the initialisation step: 50% of nodes have zero values and only 50% fires.

$$ReLU = max(0, z)$$

Dying ReLU Problem

Once a node become zero, it stops responding to variations in its error and it is unlikely for it to recover. Neuron dies, causing that part of the network gets passive. Over the time you may end up with a large part of your network doing nothing.

This problem is called the **dying ReLU problem**.

To be clear, as long as not all of the values interacting in a single node are negative, we can still get a slope out of ReLU. The dying problem is likely to occur when learning rate is too high or there is a large negative bias.

Lower learning rates often mitigates the problem. Otherwise modified ReLUs like **leaky ReLU** and **ELU** are also good alternatives to try. They have a slight slope in the negative range, thereby preventing the issue.

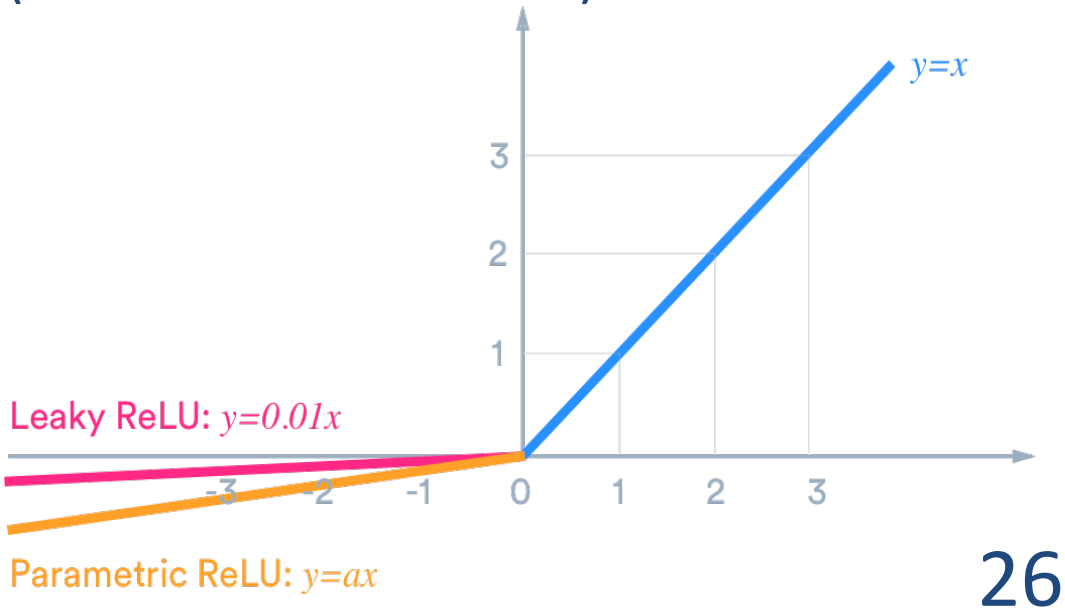
Variants of ReLU

Leaky and Parametric ReLU: these functions, with range -infinity to infinity, have a small slope for negative values, instead of only zero. The leak helps to increase the range of the ReLU function. For leaky ReLU the value of x is normally 0.01.

When x is not 0.01 then it is called Parametric ReLU. Both activation functions and their derivatives are monotonic in nature.

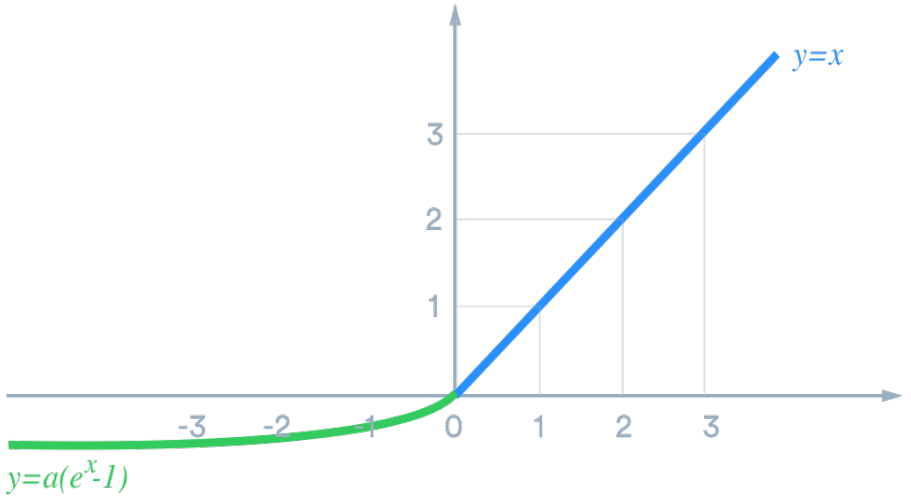
Apart from fixing the dying ReLU problem, it speeds up the training since its mean activation is close to 0 (like the tanh function).

Be aware that the result is not always consistent. Leaky ReLU is not always superior to plain ReLU, and should be considered only as an alternative.



Variants of ReLU

Exponential Linear (ELU, SELU): instead of a straight line, it uses a log curve. It saturates for large negative values, allowing them to be essentially inactive.



ReLU 6: is a ReLU capped at 6. This value 6 is an arbitrary choice. The upper bound encouraged the model to learn sparse features earlier.