Deep Learning Winter School 2018

Nina Dethlefs (Computer Science), David Benoit (Chemistry), Chris Collins and Darren Bird (Viper HPC)

UNIVERSITY OF HULL

Plan for today and tomorrow

Participants

results from survey 1

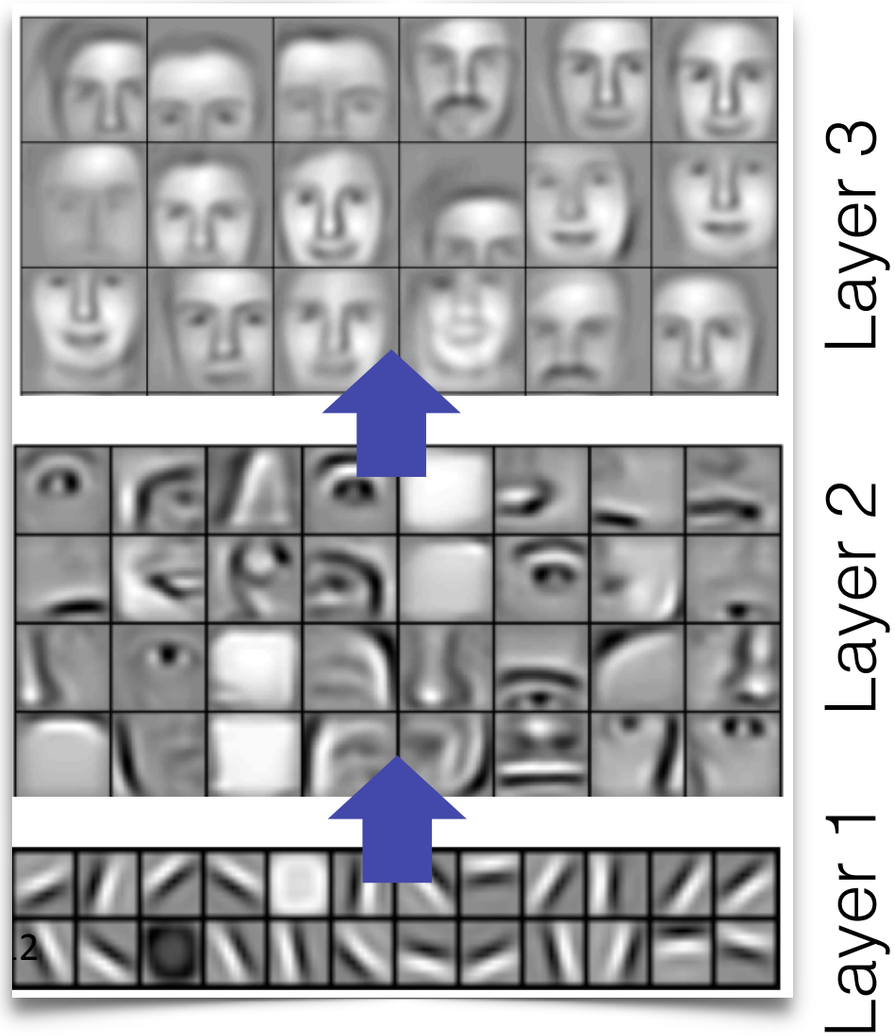
How does deep learning work?

(in general)

Deep Learning - Basic ideas

Models learn increasingly abstract representations of inputs - at different layers of abstraction - and recognise patterns in unstructured data.

Applications: self-driving cars, natural language processing, image/ video recognition, medical imaging, stock market prediction, …



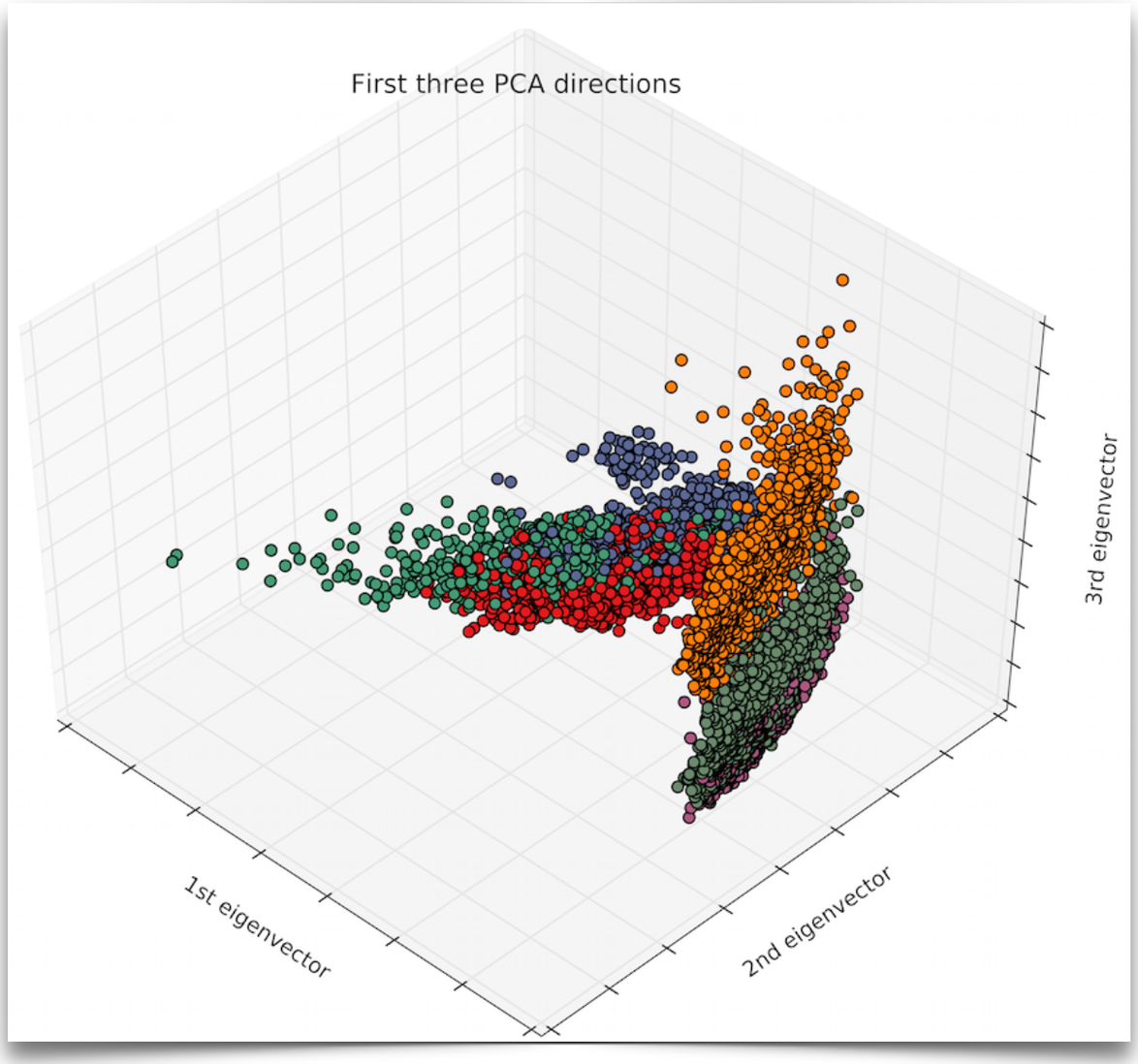
Why do deep learning?

- Discovering patterns + relationships in large, disparate, multimodal and high-dimensional datasets.

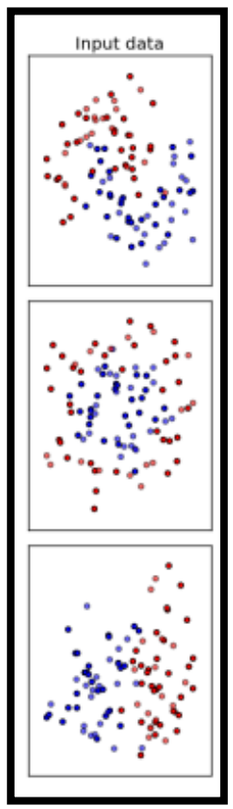
  - Models: induction, simulation, simplification, data understanding

  - Machine learning: algorithms for processing, inference + prediction

  - HPC: distributed and parallel processing and storage; portability

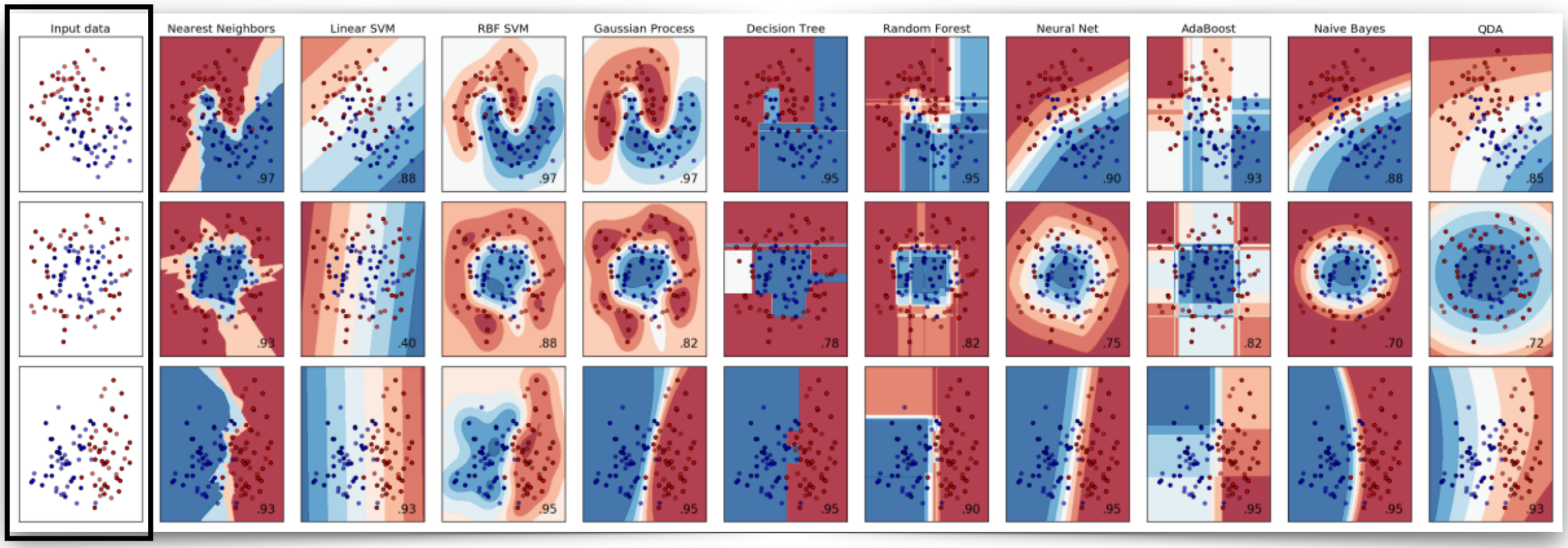


Machine learning



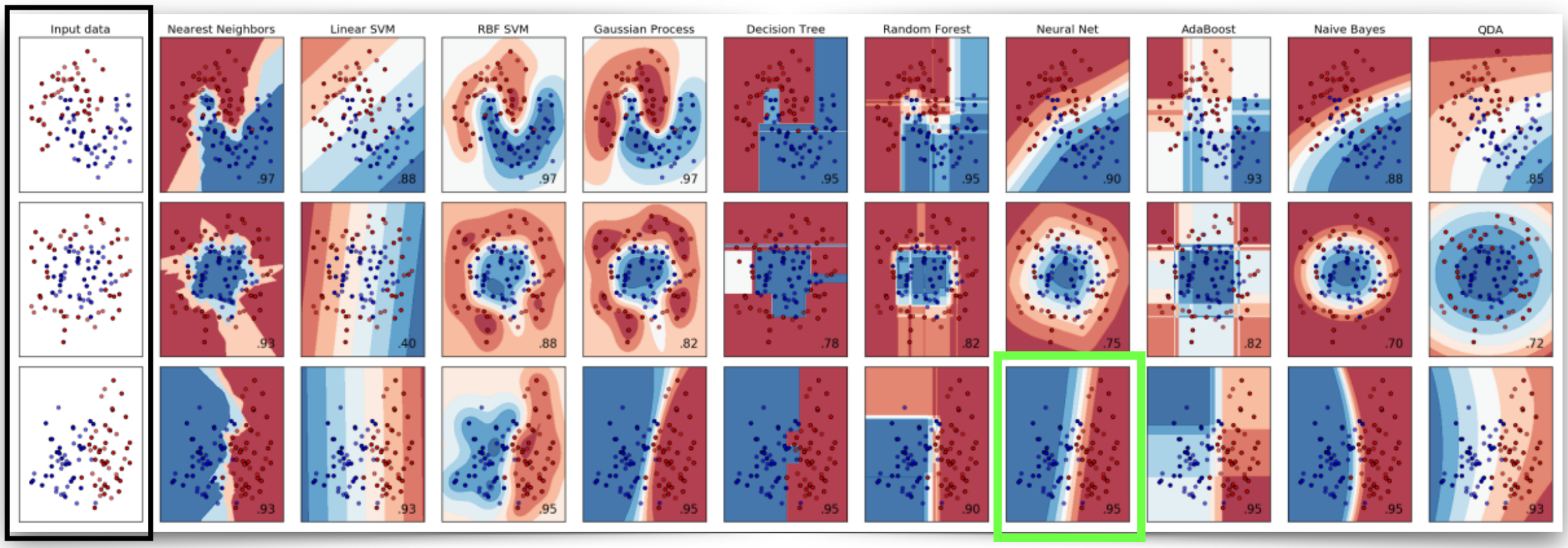
Datasets with different properties require different solutions.

Machine learning



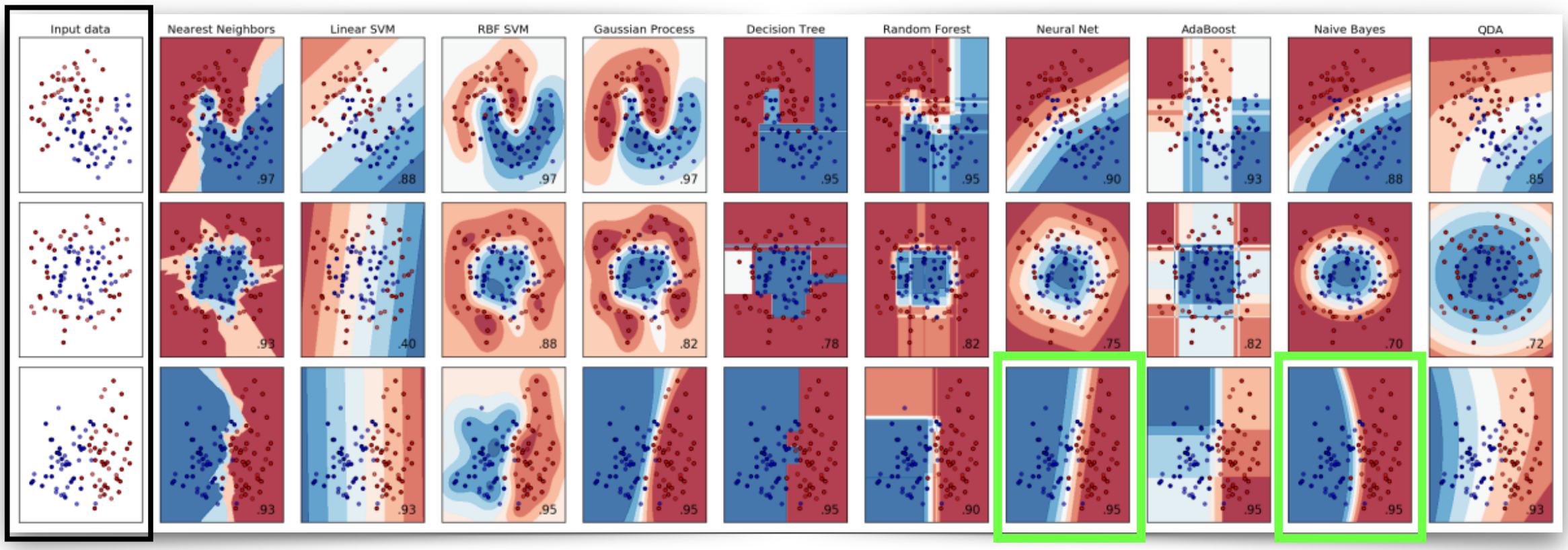
Different algorithms will provide different solutions.

Machine learning



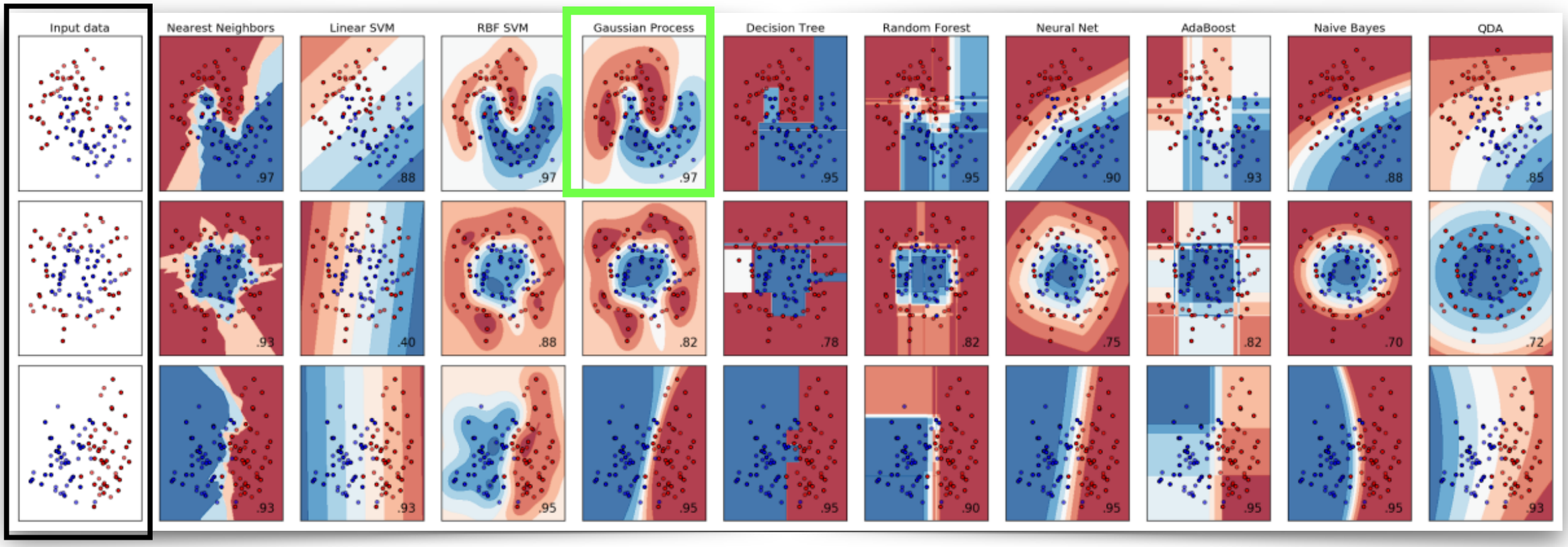
Example: neural net for linearly separable data

Machine learning



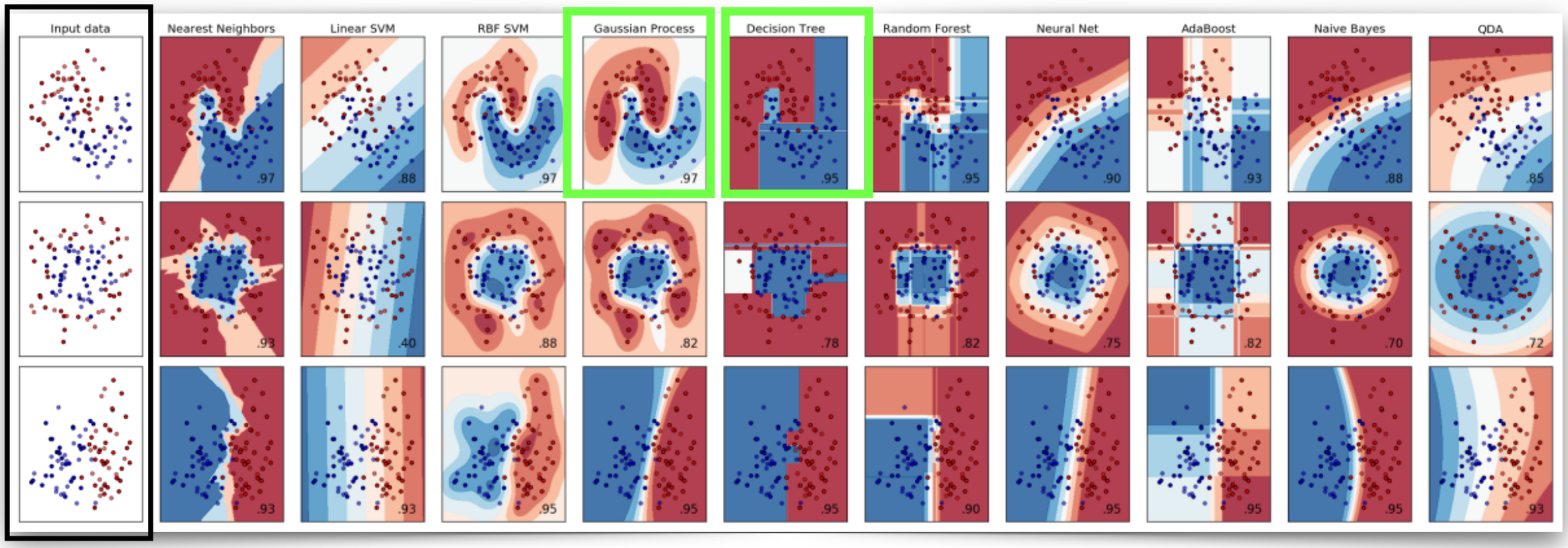
Neural network vs Naive Bayes for linear data

Machine learning



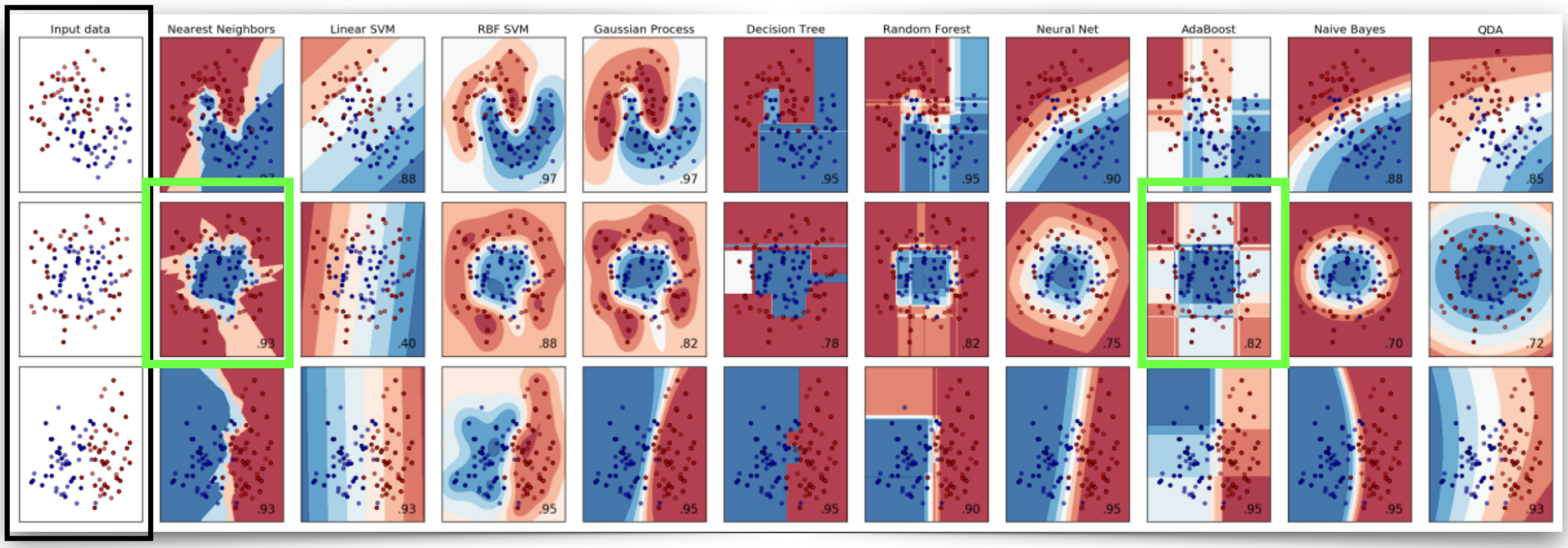
Gaussian Process for non-linearly separable data

Machine learning



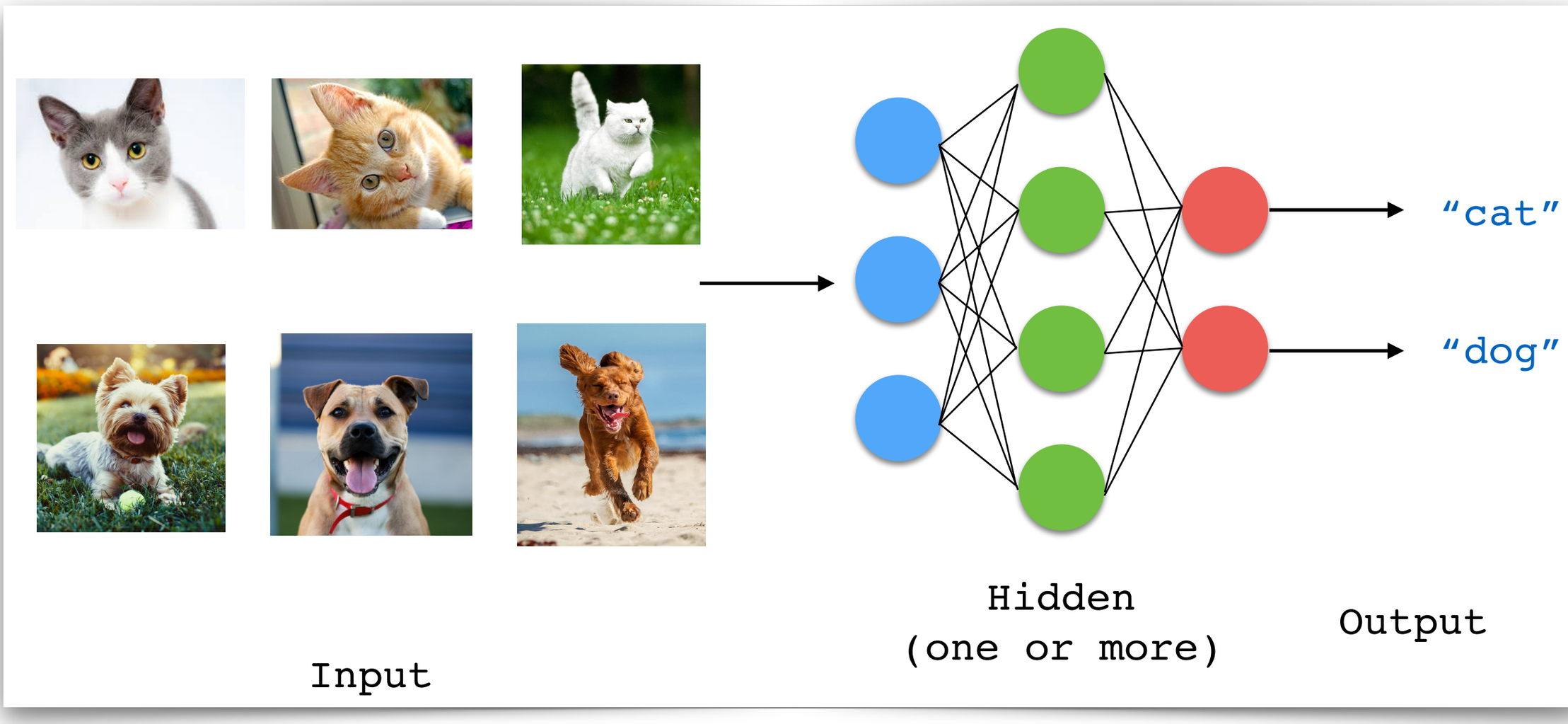
Gaussian Process vs Decision tree for non-linear data

Machine learning



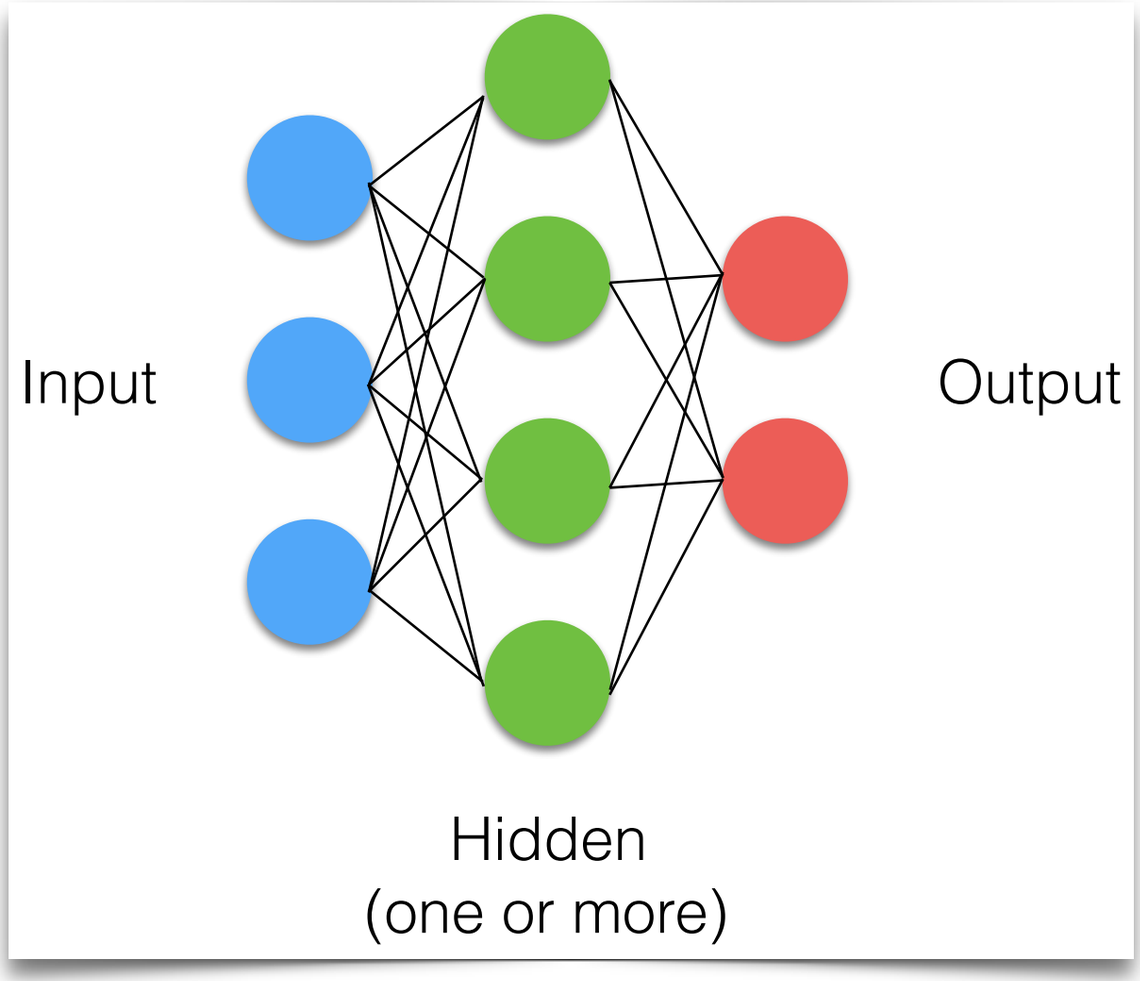
Comparison of clustering techniques

A simple neural network



A simple neural network

- Learns a mapping function from a set of inputs to a set of outputs by minimising a loss through training
- Multiple layers of abstraction
- Computationally expensive
- Technique behind most recent breakthroughs in AI



The “best” neural network

Different types of neural nets for different tasks…

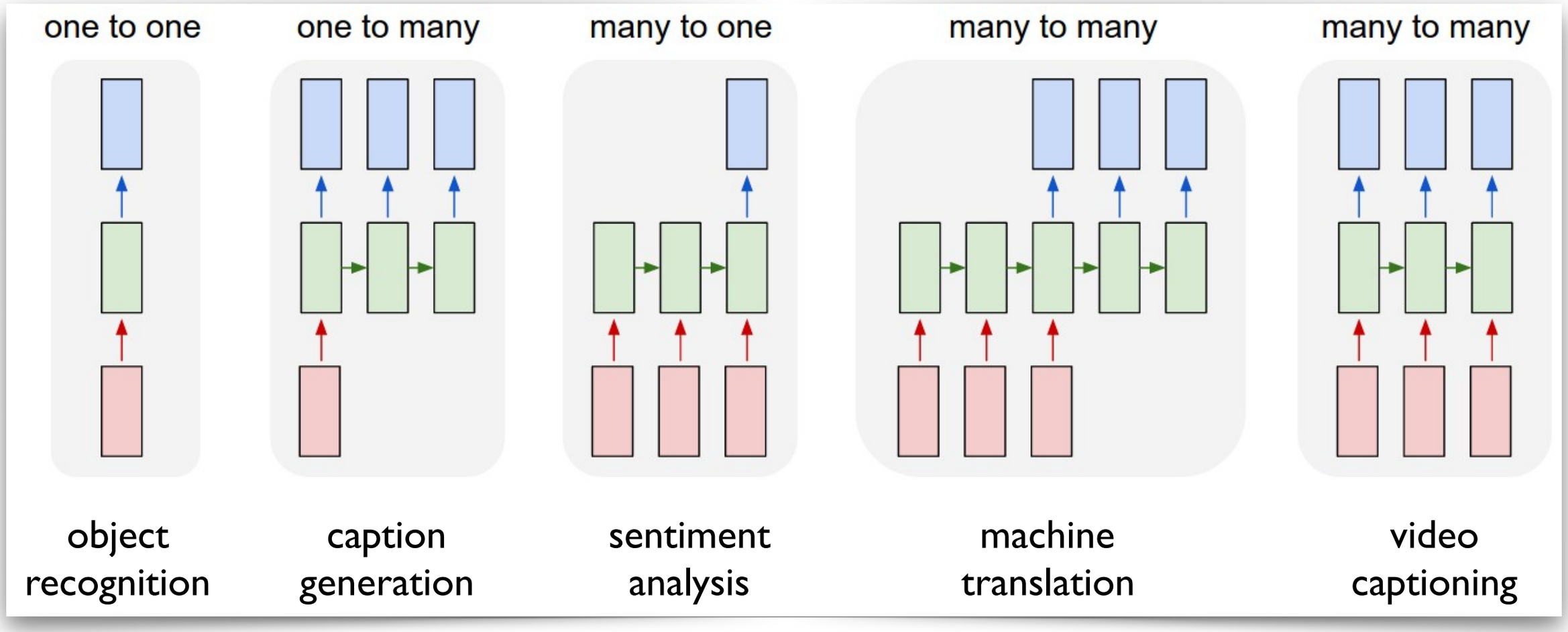
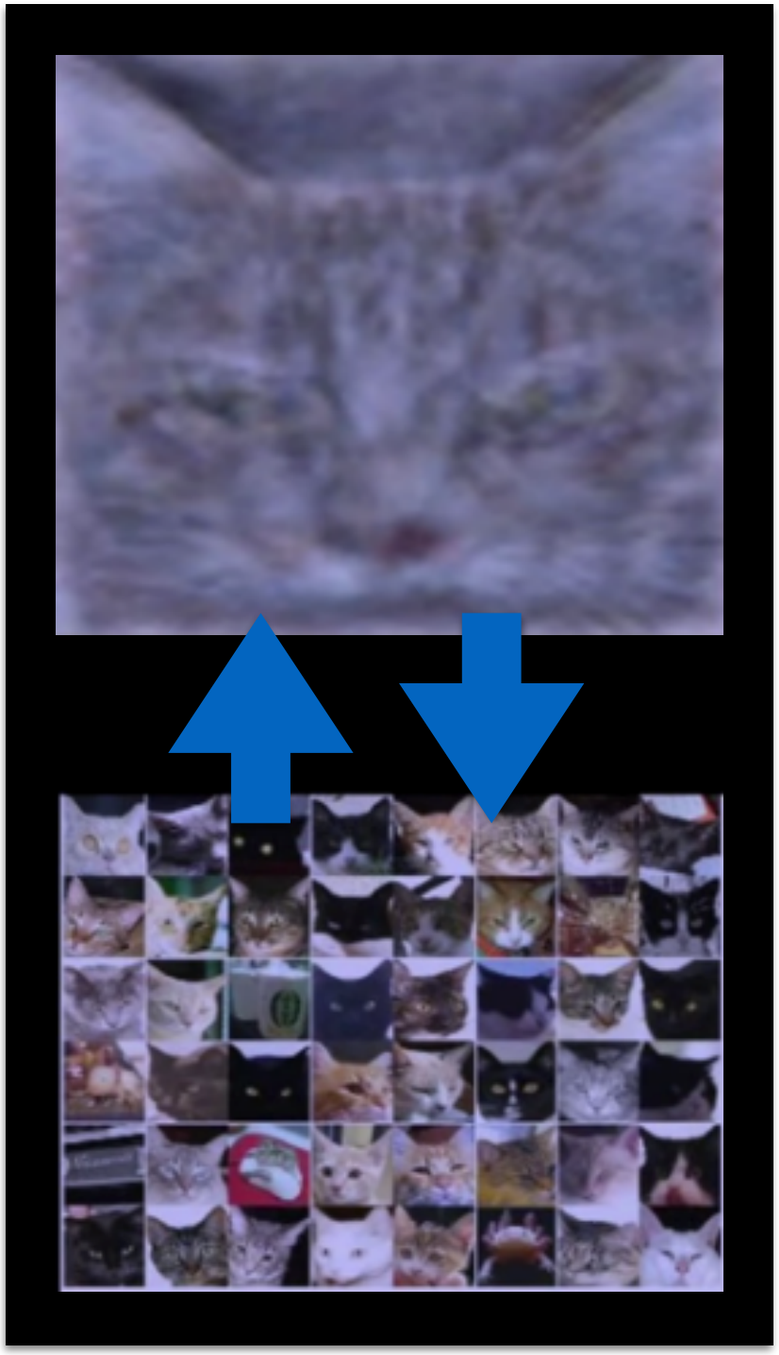


Image processing

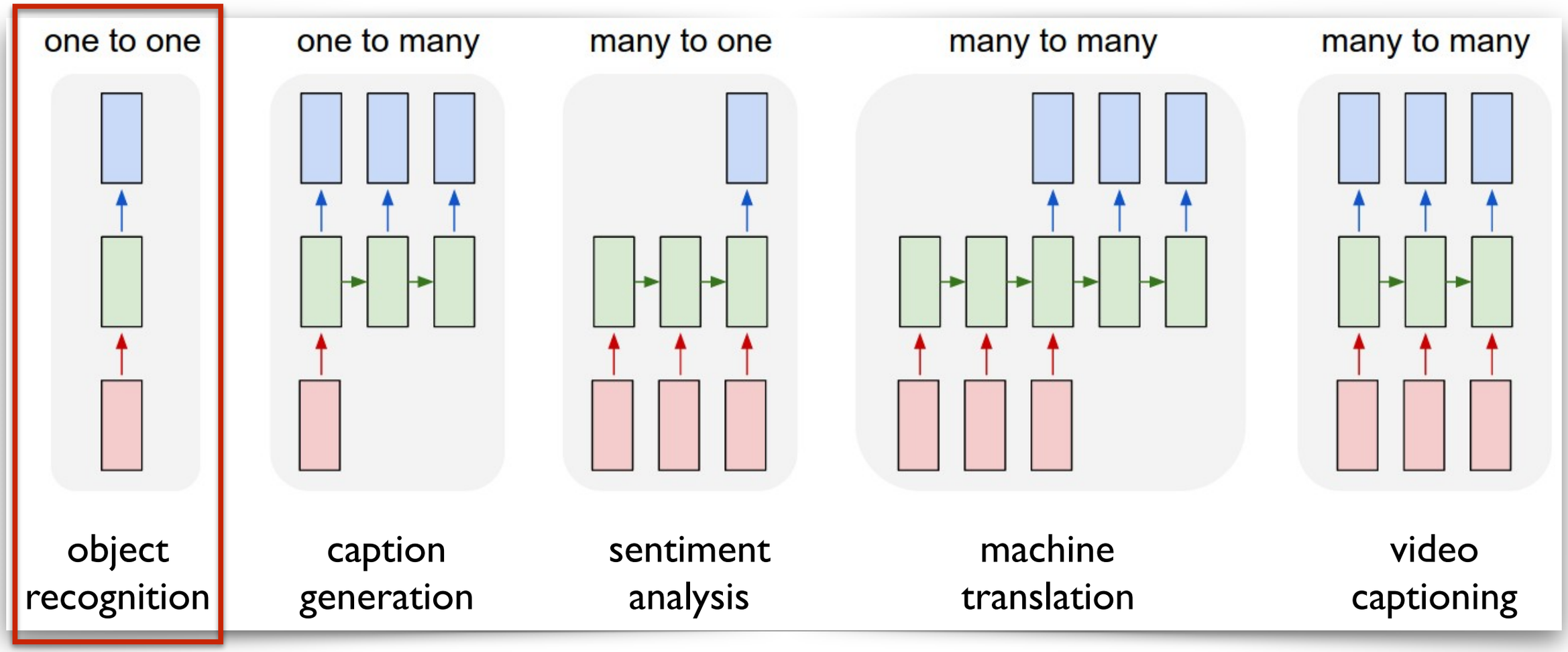
Imagine processing



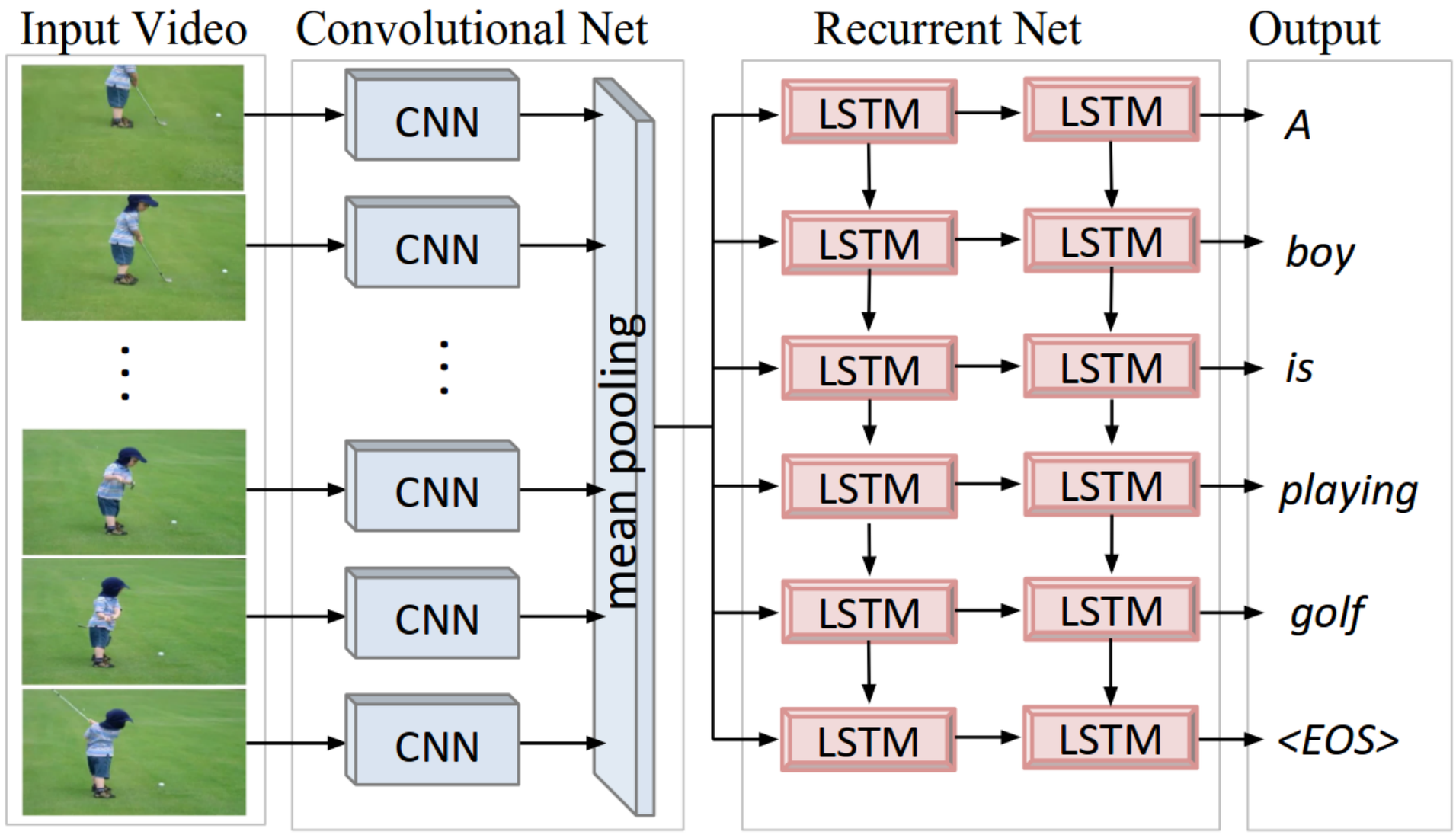
Andrew Ng, keynote.

The “best” neural network

Different types of neural nets for different tasks…



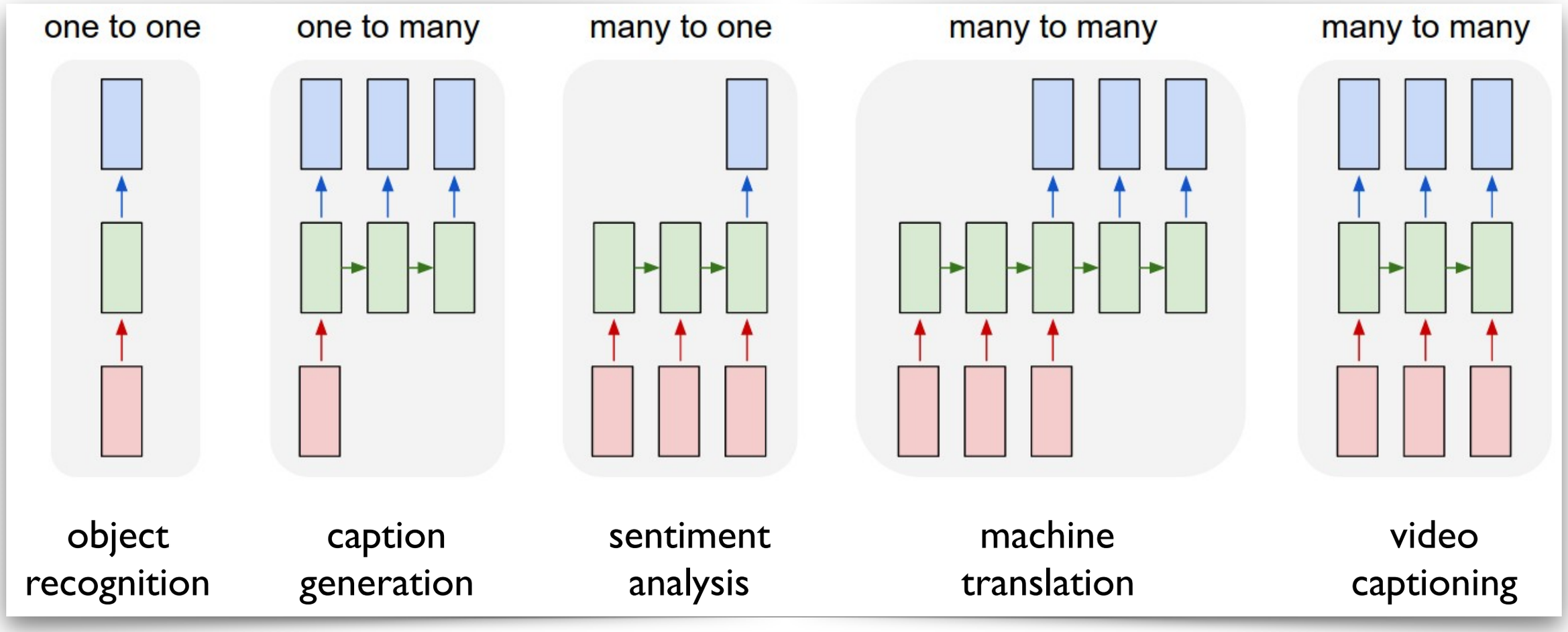
Video caption generation



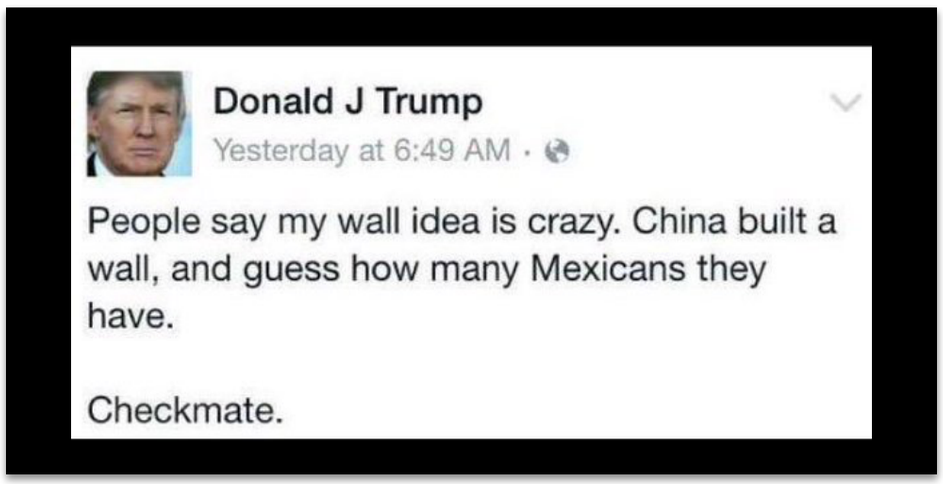
Venugopalan, et al., (2015), "Translating Videos to Natural Language Using Deep Recurrent Neural Networks", CVPR.

The “best” neural network

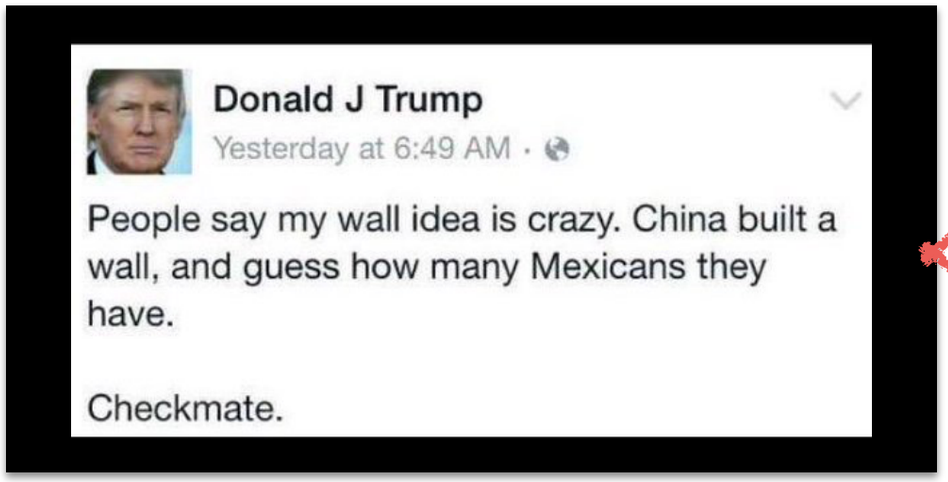
Different types of neural nets for different tasks…

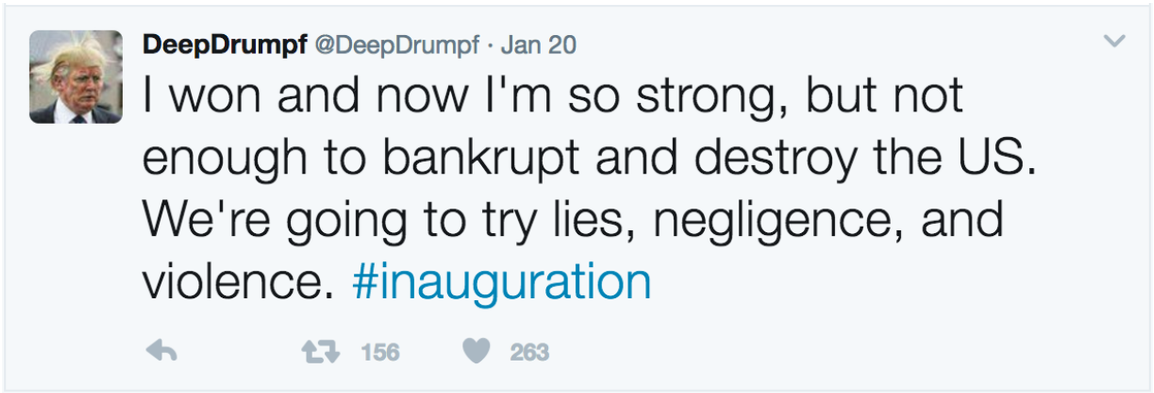


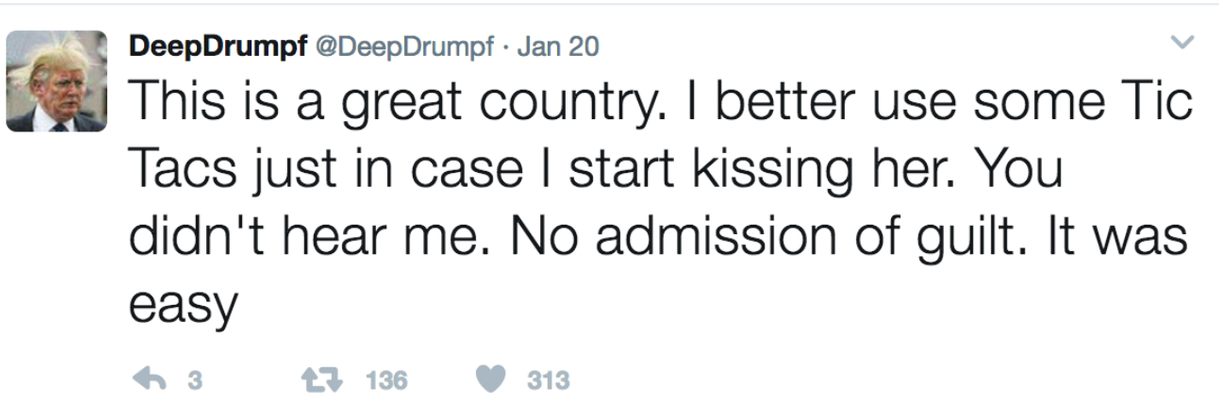
Sequence generation



Sequence generation



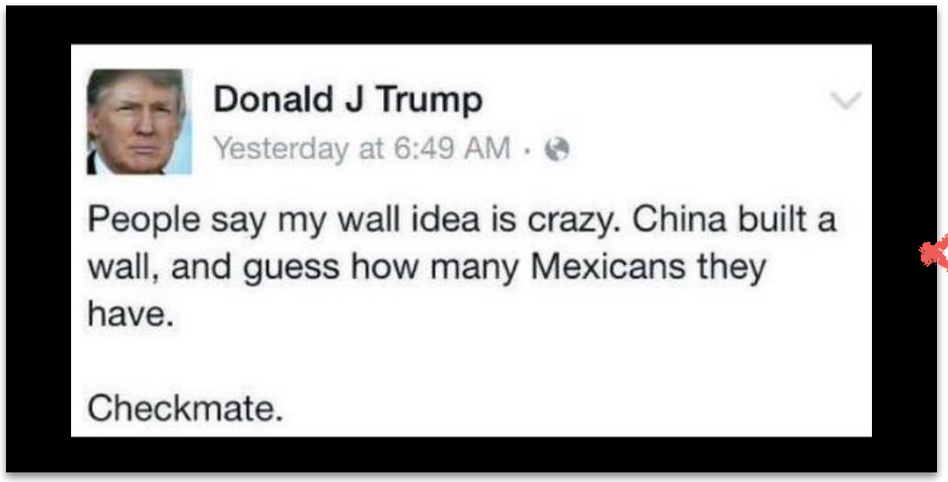


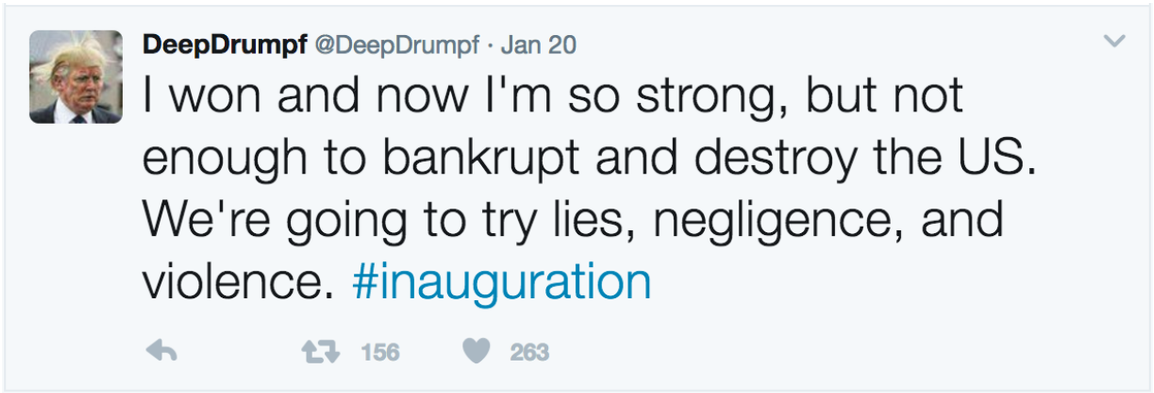


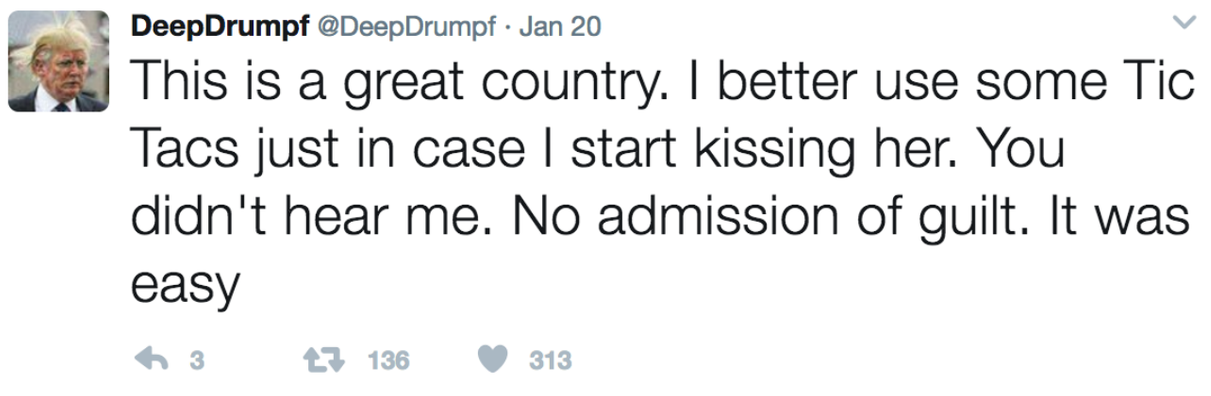
MIT's DeepDrumpf

https://twitter.com/deepdrumpf?lang=en

Sequence generation







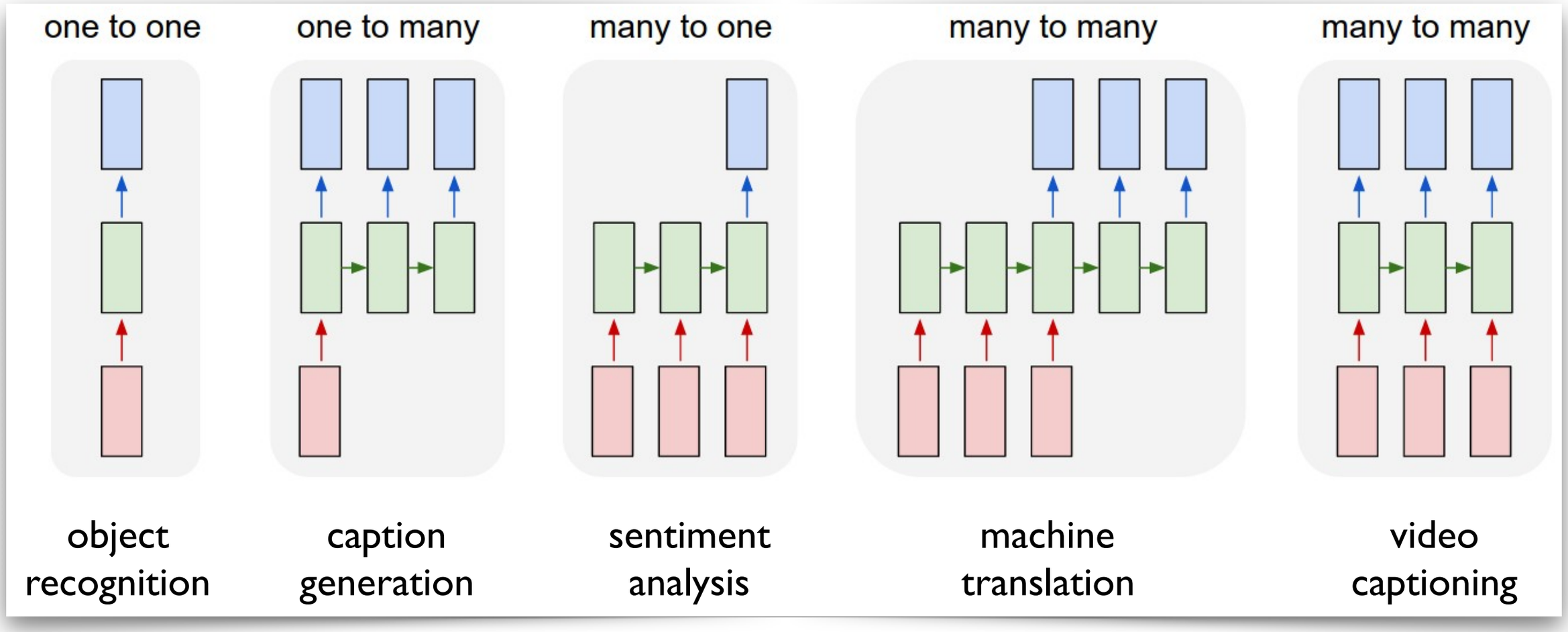
Microsoft's Tay

MIT's DeepDrumpf

https://twitter.com/deepdrumpf?lang=en

The “best” neural network

Different types of neural nets for different tasks…



Source code generation

Trained from code on GitHub

Neural net learns patterns of how humans program

Generates many (completely random) OpenCL kernels as synthetic benchmarks for GPU language compilers

CLgen

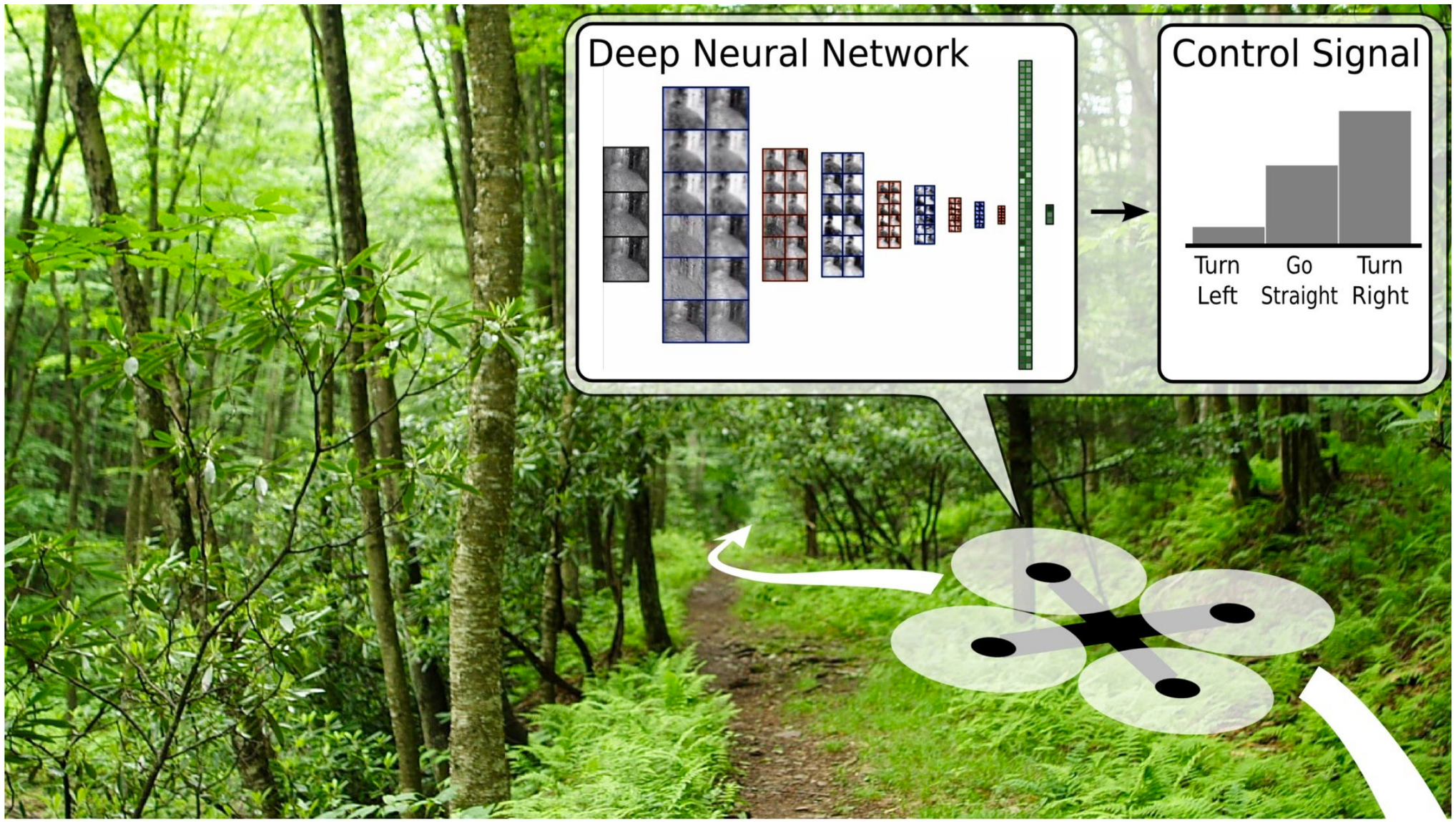
Deep Learning Program Generator.

```
1   #define DTYPE float
2   #define ALPHA(a) 3.5f * a
3   inline DTYPE ax(DTYPE x) { return ALPHA(x); }
4
5   __kernel void saxpy( /* SAXPY kernel */
6       __global DTYPE *input1,
7       __global DTYPE *input2,
8       const int nelem)
9   {
10      unsigned int idx = get_global_id(0);
11      // = ax + y
12      if (idx < nelem) {
13          input2[idx] += ax(input1[idx]); }}
```

https://github.com/ChrisCummins/clgen

C. Cummins, Synthesising Benchmarks for Predictive Modeling, CGO 2017, Best paper award.

Flying drones

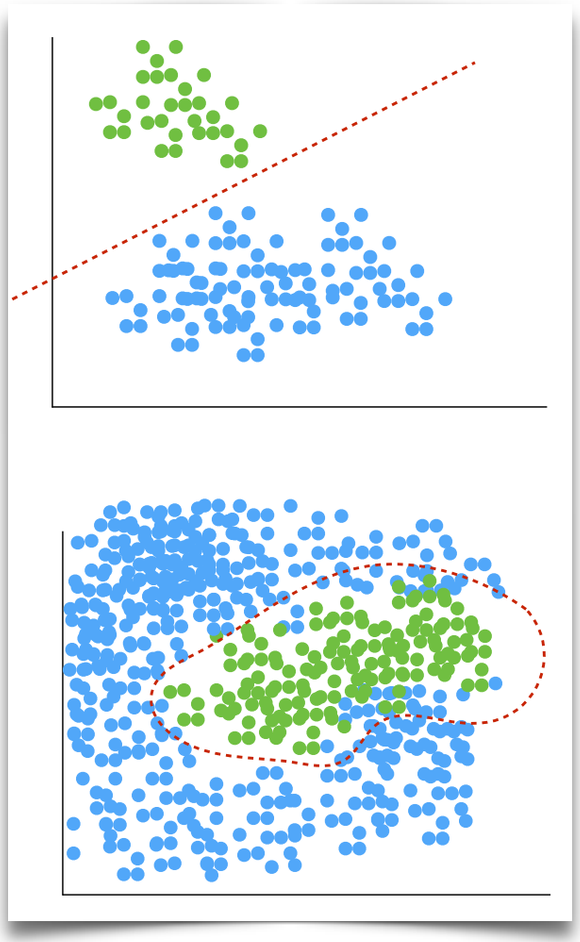


Drones learn to fly unsupervised in new environments.

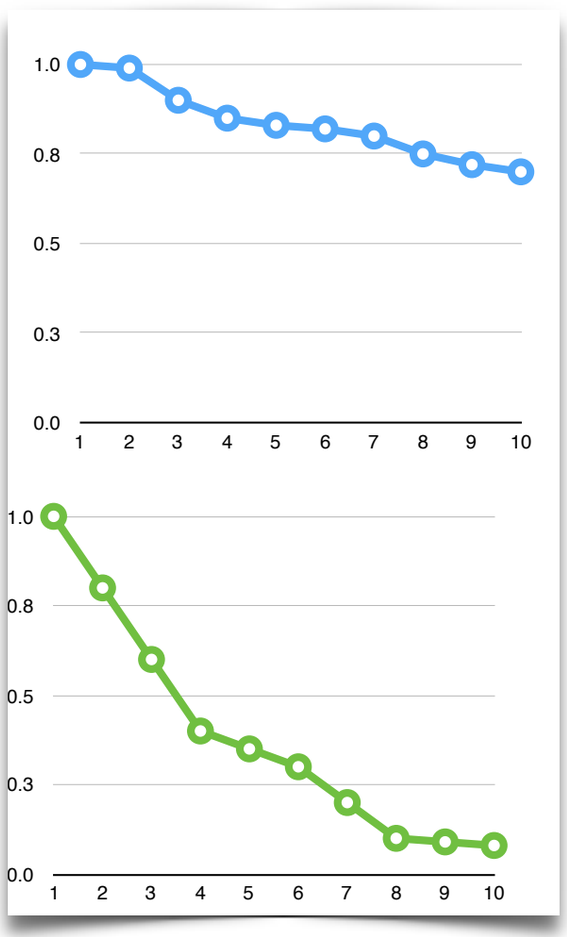
Guisti, et al., (2016), “A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots”, IEEE RA-L.

Parameter tuning

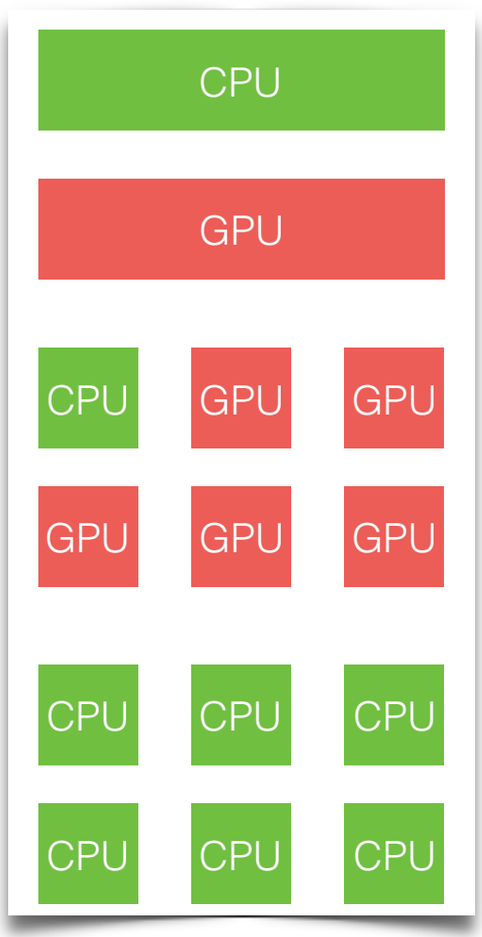
Linearity of data
(4+ parameters)



Learning rate
(10+ parameters)



Hardware
(5+ parameters)

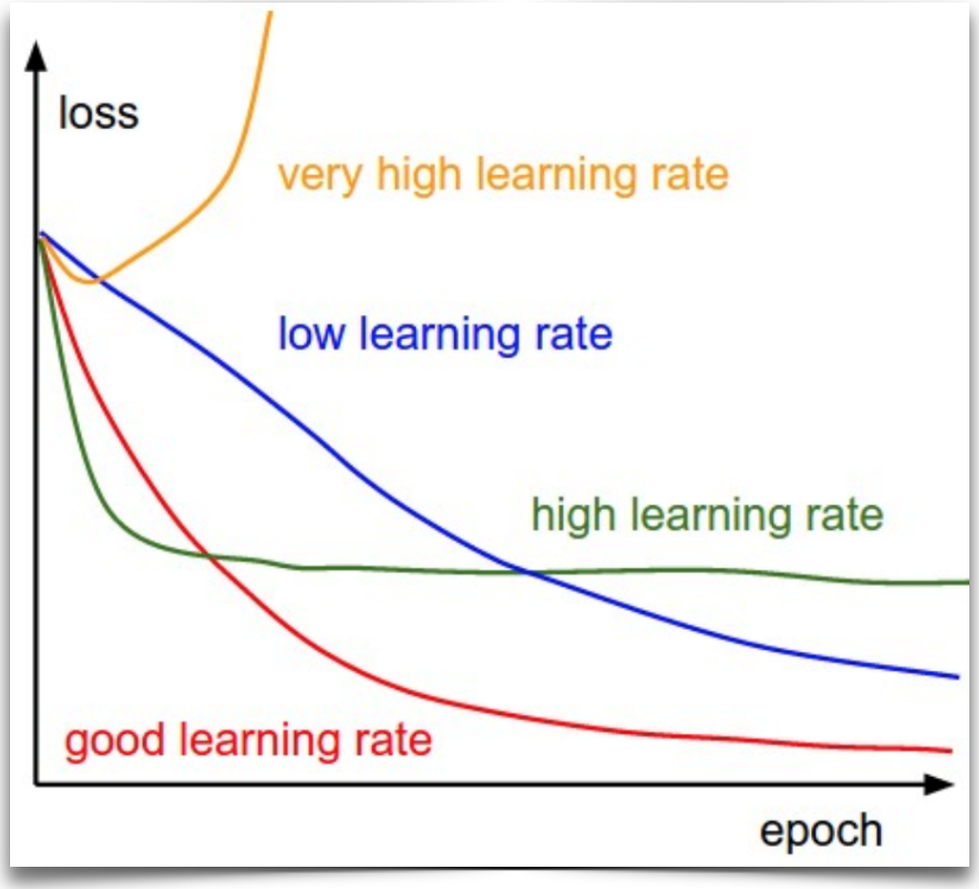


Learning rates

Learning rates need to be tuned for each learning task.

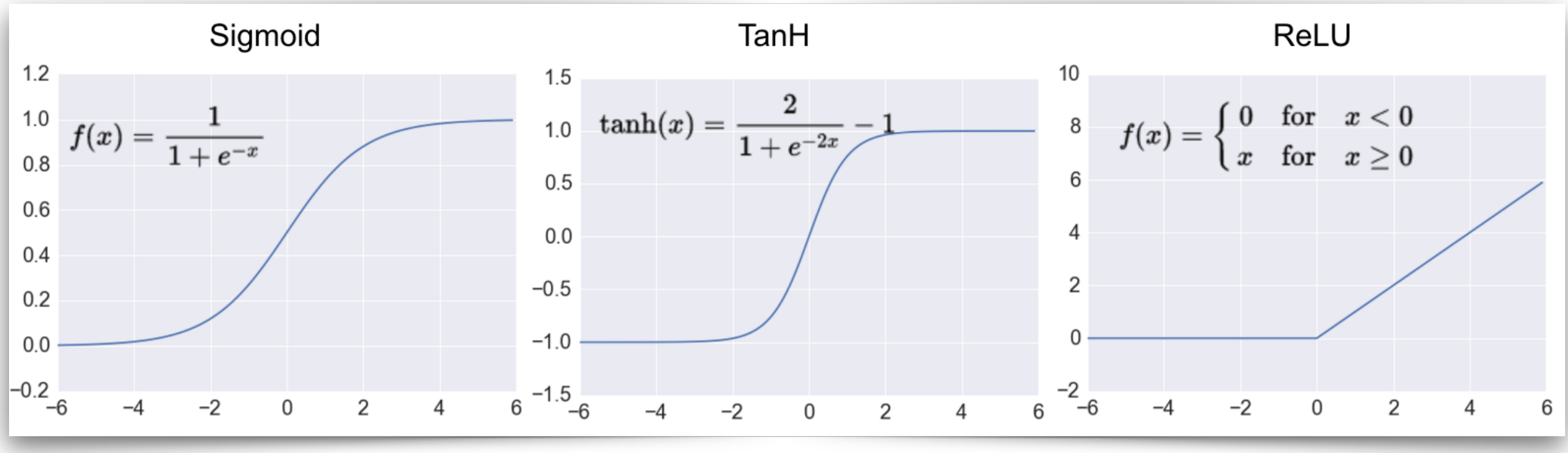
Often, we need higher learning rates with smaller datasets as small learning rates take long to learn.

But high learning rates can “bounce around” and make it difficult to reduce the loss over time.



http://cs231n.github.io/neural-networks-3/

Activation functions

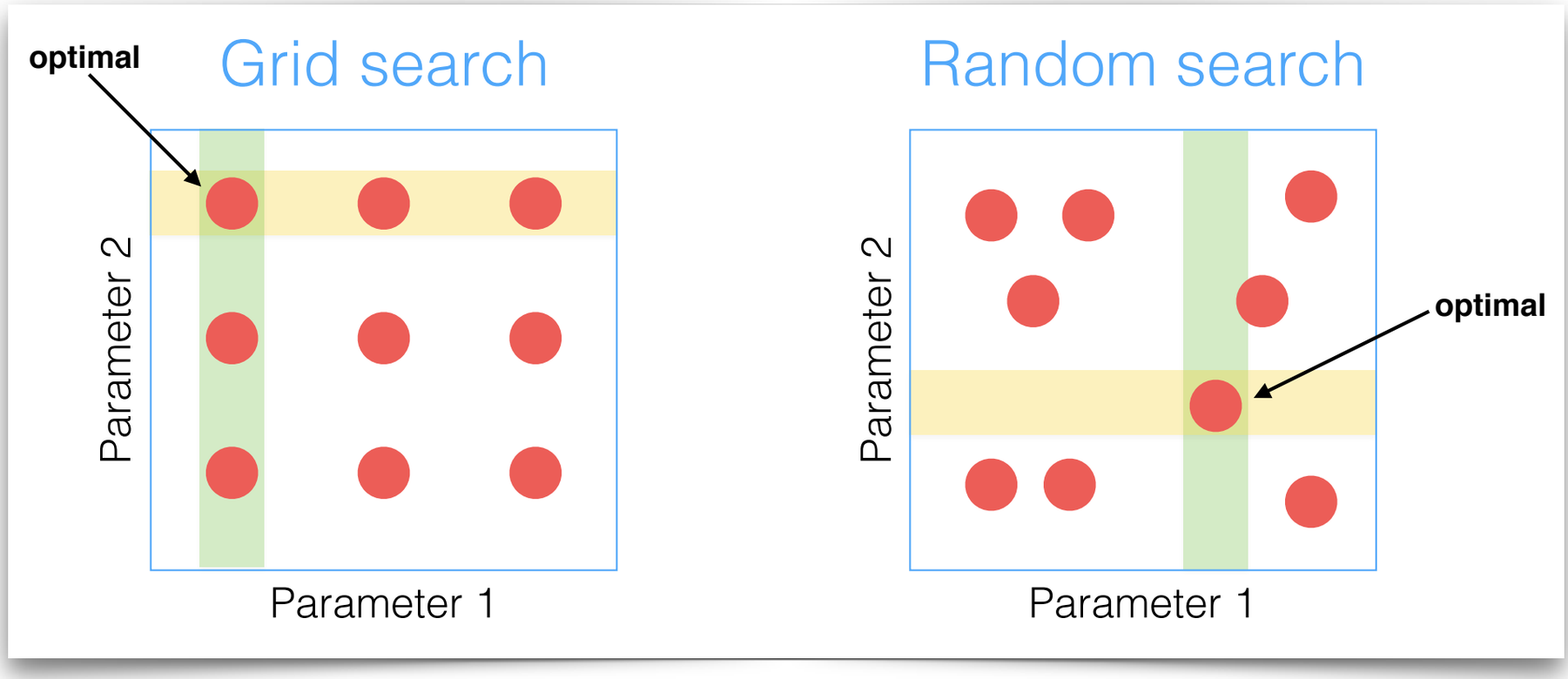


https://www.kdnuggets.com/2017/09/neural-network-foundations-explained-activation-function.html

We pass a neural networks' weights through an activation function.

These are different types of non-linear functions that squash the weights into a certain range, e.g. {0 … 1}.

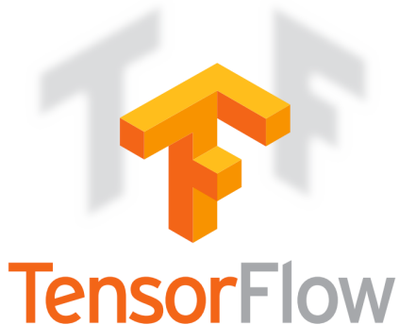
Optimising parameters



J. Bergstra et al. (2011) Algorithms for Hyper-parameter Optimization. Proceedings of NIPS.

There are algorithms that find the “optimal” combination of hyper-parameters automatically. Common choices are brute force (i.e. performing an exhaustive search) and trial and error (i.e. finding parameters by experimentation on randomly chosen search spaces.

Libraries and requirements



- Most libraries provide a mathematical framework (matrix and tensor multiplication, gradient computation, activation functions, etc.)

- But little help with: Data pre-processing, dimensionality of data, encoding and decoding, visualisation, GPU-compatibility / HPC compatibility

Keras

“Folk wisdoms” of deep learning

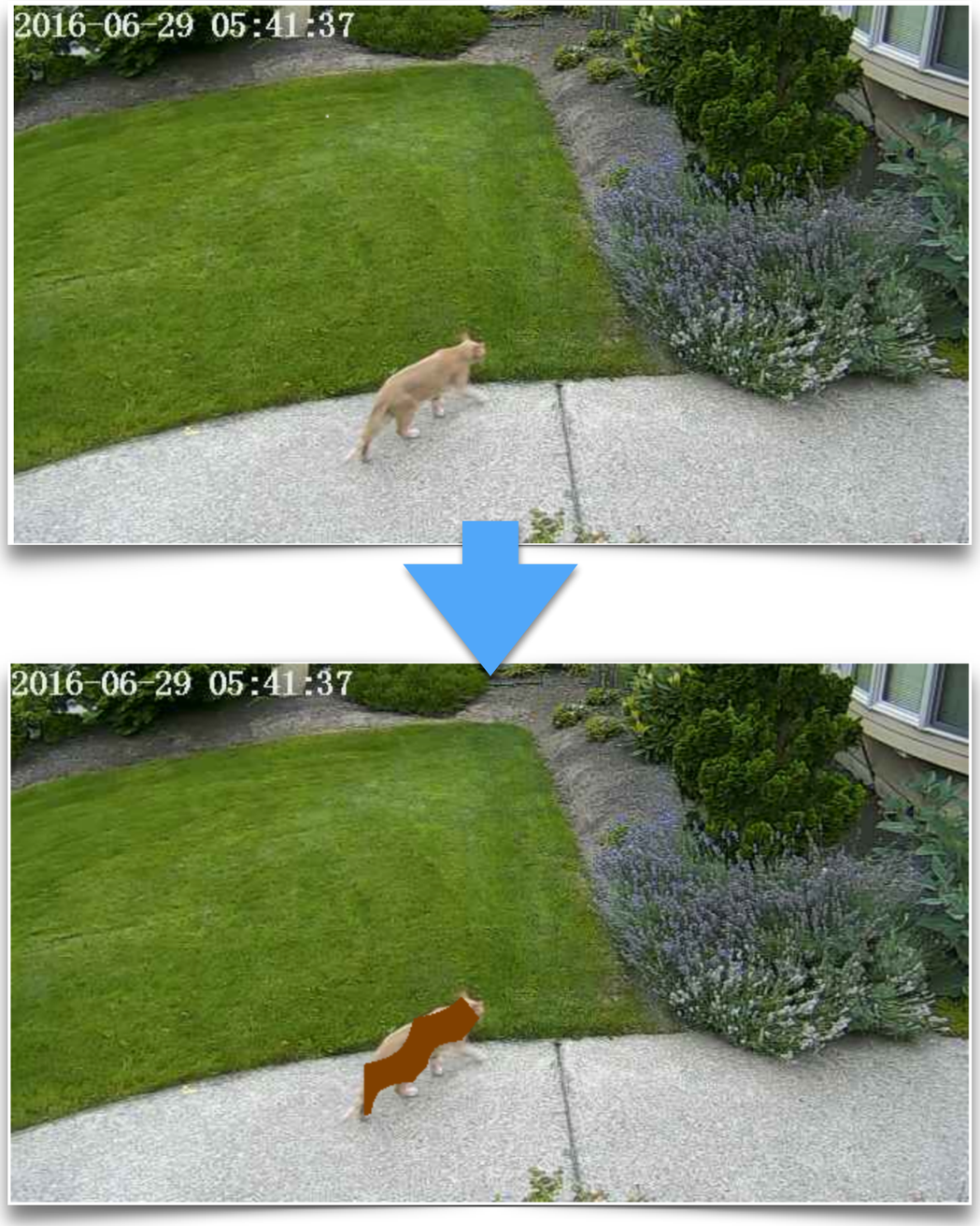
- To train good models, we need a lot of data
- Data need to be “balanced”
- Very general rules of thumb for parameter tuning
- Generally run on a GPU and / or in a distributed way
- (Don’t expect to understand what the model learnt and why)

Interim summary

- Deep learning works - given the right setup and conditions - and can be applied to many useful tasks

- Different neural networks are suitable for different tasks

- Lots of space for parameter tuning and optimisation

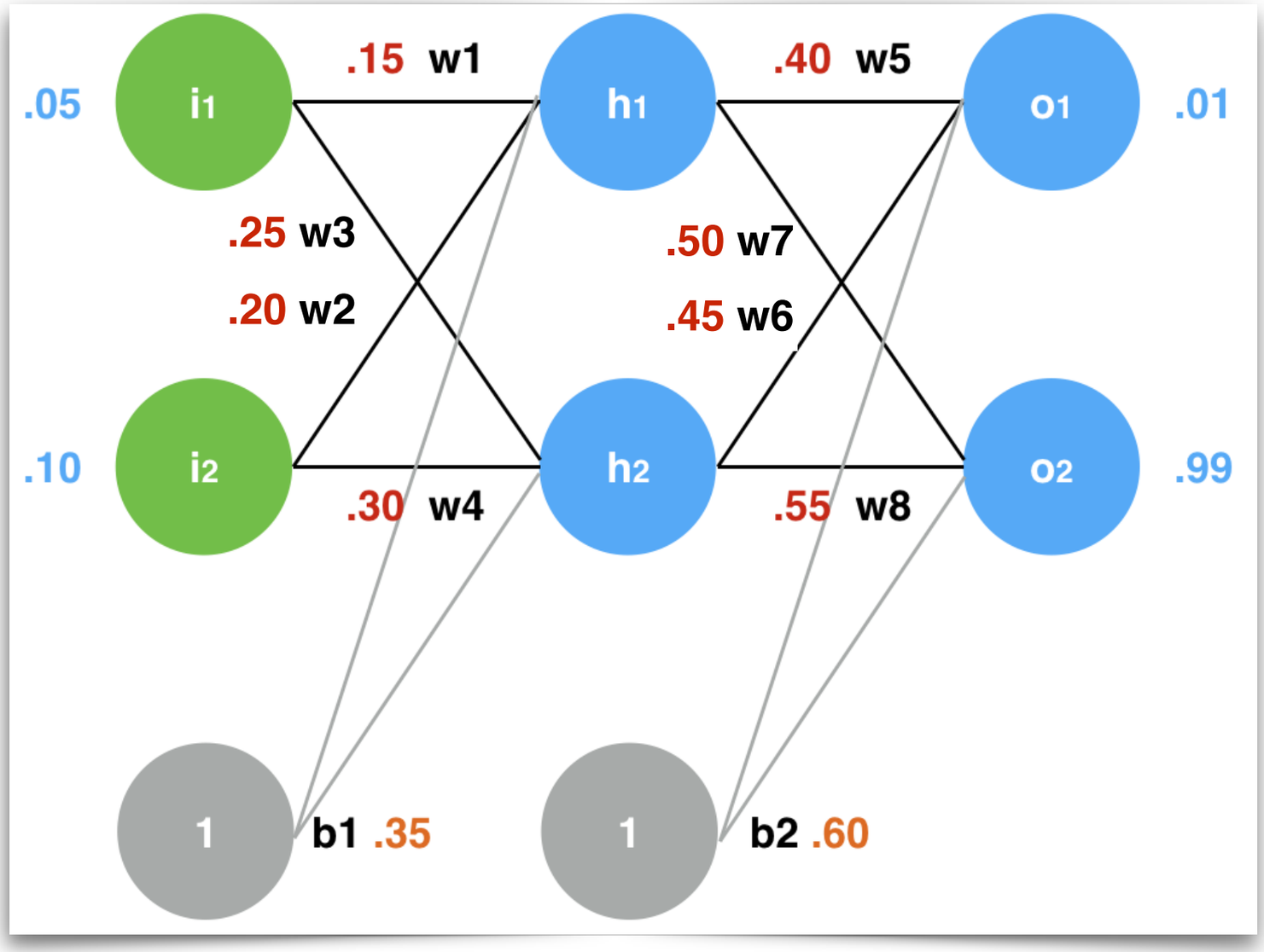


https://blogs.nvidia.com/blog/2016/07/07/deep-learning-cats-lawn/

How does deep learning work?

(exactly)

A neural net with weights



https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Backpropagation algorithm

Algorithm 1 Backpropagation algorithm (from Wikipedia).

```
1:  function computeWeights
2:      initialise network weights (often small random values)
3:      for each training exampled named ex do
4:          prediction = neural-net-output (network, ex) // forward pass
5:          actual = teacher-output (ex)
6:          compute error (prediction - actual) at the output units
7:          compute Δ_{w_h} for all weights from hidden layer to output layer // backward pass
8:          compute Δ_{w_i} for all weights from input layer to hidden layer // backward pass continued
9:          update network weights // input layer not modified by error estimate
10:     end for
11:     until all examples classified correctly or another stopping criterion is satisfied
12:     return the network
13: end function
```

The forward step

With the forward step, we want to compute the total error of the current neural and its weights. Lets compute the weight for h_1 :

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

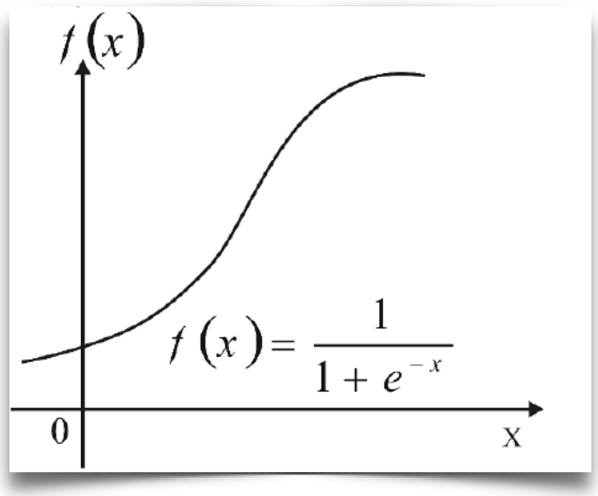
Lets do the same for net_{h2} :

The forward step

Next we need to pass our weights net_{h1} and net_{h2} through the activation function, e.g. a sigmoid function. We can call these weights out_h1 and out_h2 :

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.3774}} = 0.593269992$$

Lets do the same for out_{h2} :



We now have an internal feature representation.

The forward step

Finally, we need the weights that connect the hidden units with the outputs. We compute net_{o1} and net_{o2}:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

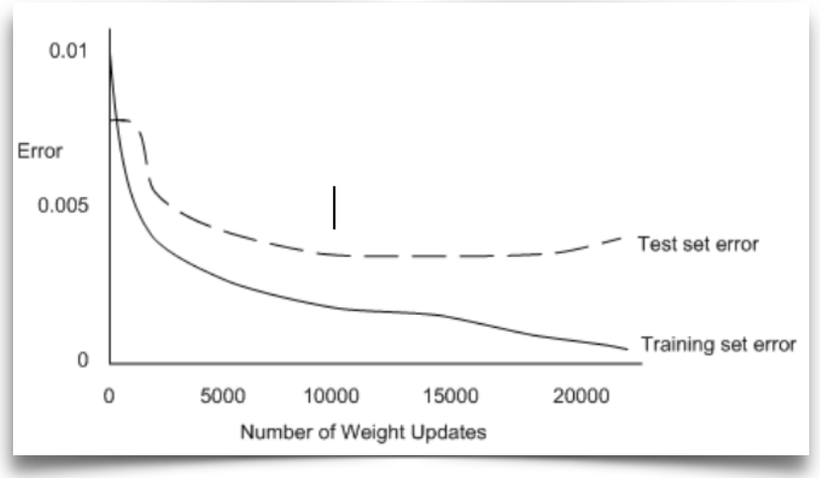
$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

Passing this through the activation function, we get:

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-1.105905967}} = 0.75136507$$

Doing this for out_{o2}, we get = 0.772928465.

The total error



Our computed weights allows to compute the total error now:

$$E_{total} = \sum_{o} \frac{1}{2}(target_o - output_o)^2$$

So, for E_{o1}:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Computing the same for E_{o2}, we get 0.023560026. Therefore:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

The softmax function

Our errors E_{o1} and E_{o2} are still just weights, not probabilities.

If we wanted probabilities, we can apply the softmax function:

$$P(out_{o_i}) = \frac{out_{o_i}}{out_{o_j} + out_{o_j}}$$

out_{o1} = 0.75136507, therefore P(o_1) =
out_{o2} = 0.772928465, therefore P(o_2) =

The backward step

So far, we only know the current error. Now we need to pass it back through the network to update the weights and learn over time.

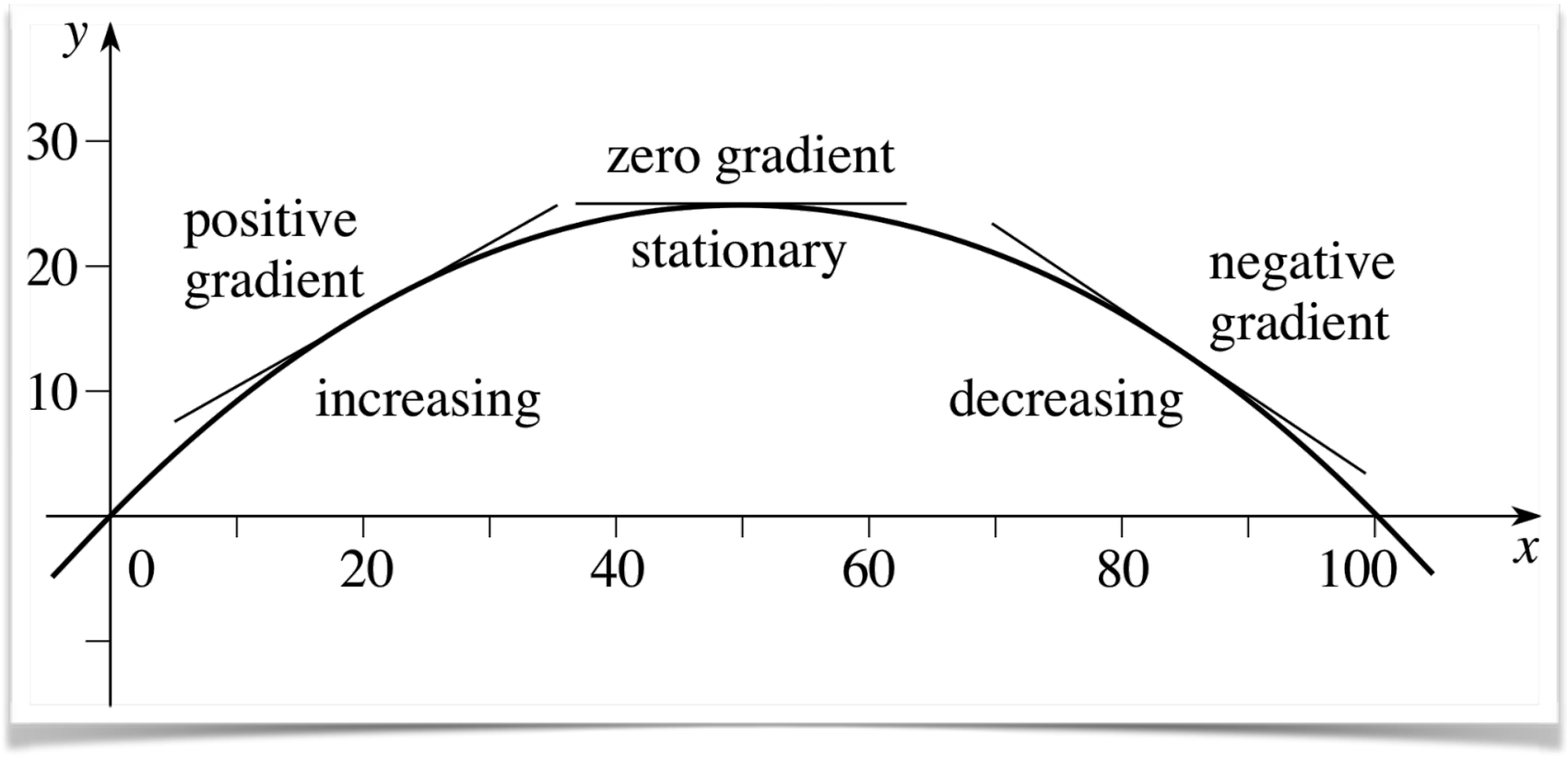
Lets start by computing the update for w_5:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$
$$= \delta_{o1} out_{h1}$$
$$\text{where } \delta_{o1} = -(target_{o1} - out_{o1}) \times out_{o1}(1 - out_{o1})$$

What we need is the "partial derivative" of E_{total} with respect to w_5; aka "the gradient with respect to w_5".

Partial derivatives

The partial derivative (or gradient) shows the change and the direction of change.



The backward step

Substituting the values in δ_{o1}, we get:

$$\delta_{o1} = (0.01 - 0.75136507) \times 0.75136507 \times (1 - 0.75136507)) = 0.13849856$$

and therefore for the partial derivative with respective to w_5:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1} = 0.13849856 \times 0.593269 = 0.0821669$$

Lets compute the partial derivative with respect to w_6.

The backward step

Lets do the same for w_7 and w_8:

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial out_{o2}} \times \frac{\partial out_{o2}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_7}$$
$$= \delta_{o2} out_{h1}$$

We need δ_{o2} = -0.0389231.

$$\frac{\partial E_{total}}{\partial w_7} = \delta_{o2} out_{h1} = -0.03898231 \times 0.593269 = -0.022602537$$

The partial derivative with respect to w_8 = -0.022740239.

The backward step

To update the neural net's weights, we do:

$$w_5^+ = w_5 - \eta \times \frac{\partial E_{total}}{\partial w_5} = 0.40 - 0.5 \times 0.082167041 = 0.35891648$$

$$w_6^+ = w_6 - \eta \times \frac{\partial E_{total}}{\partial w_6} = 0.45 - 0.5 \times 0.082666763 = 0.408666186$$

$$w_7^+ = w_7 - \eta \times \frac{\partial E_{total}}{\partial w_7} = 0.50 - 0.5 \times -0.022602537 = 0.511301269$$

$$w_8^+ = w_8 - \eta \times \frac{\partial E_{total}}{\partial w_8} = 0.55 - 0.5 \times -0.022602537 = 0.561370119$$

where η is the learning rate, e.g. 0.5

The backward step

There is a similar process for updating the weights $w1$ … $w4$.
You can find all the details in the backpropagation tutorial.

Our total error after the first example was 0.2984. With our updated weights, we get 0.2910 for our second example.

This might not seem like much of a reduction but after a sufficient amount of training epochs, the error will go down and our predictions will become better.

Algorithms and models

- Backpropagation is useful to understand when working with deep learning

- It represents the backbone of deep learning algorithms — across individual models

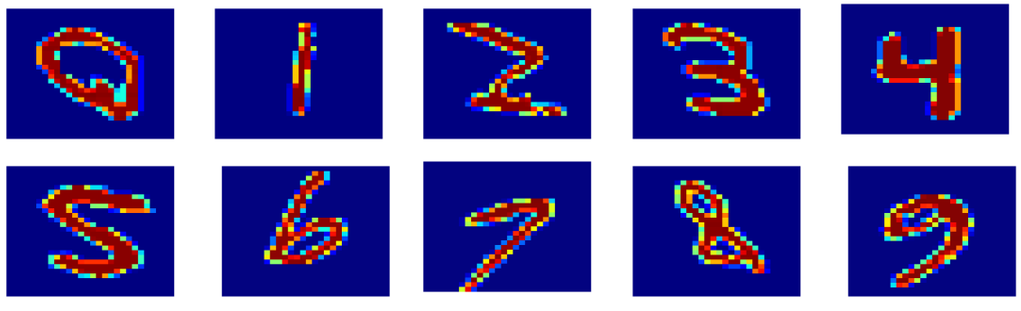
- Algorithms such as Stochastic gradient descent, Adam, AdaDelta, RMSprop and other use it

Later: hand-written digit recognition

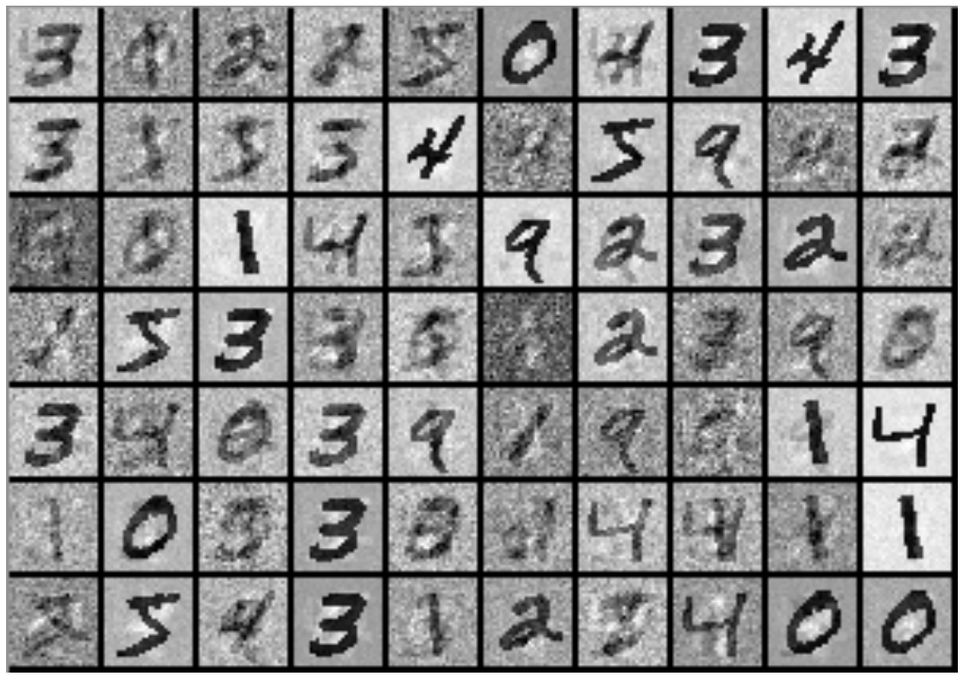
Hand-written digit recognition

MNIST dataset (freely available)

- 10 discrete outputs in range [0-9]
- Images are represented as 28*28 numeric pixel matrices
- 60K training examples and 10K test examples - sufficient size

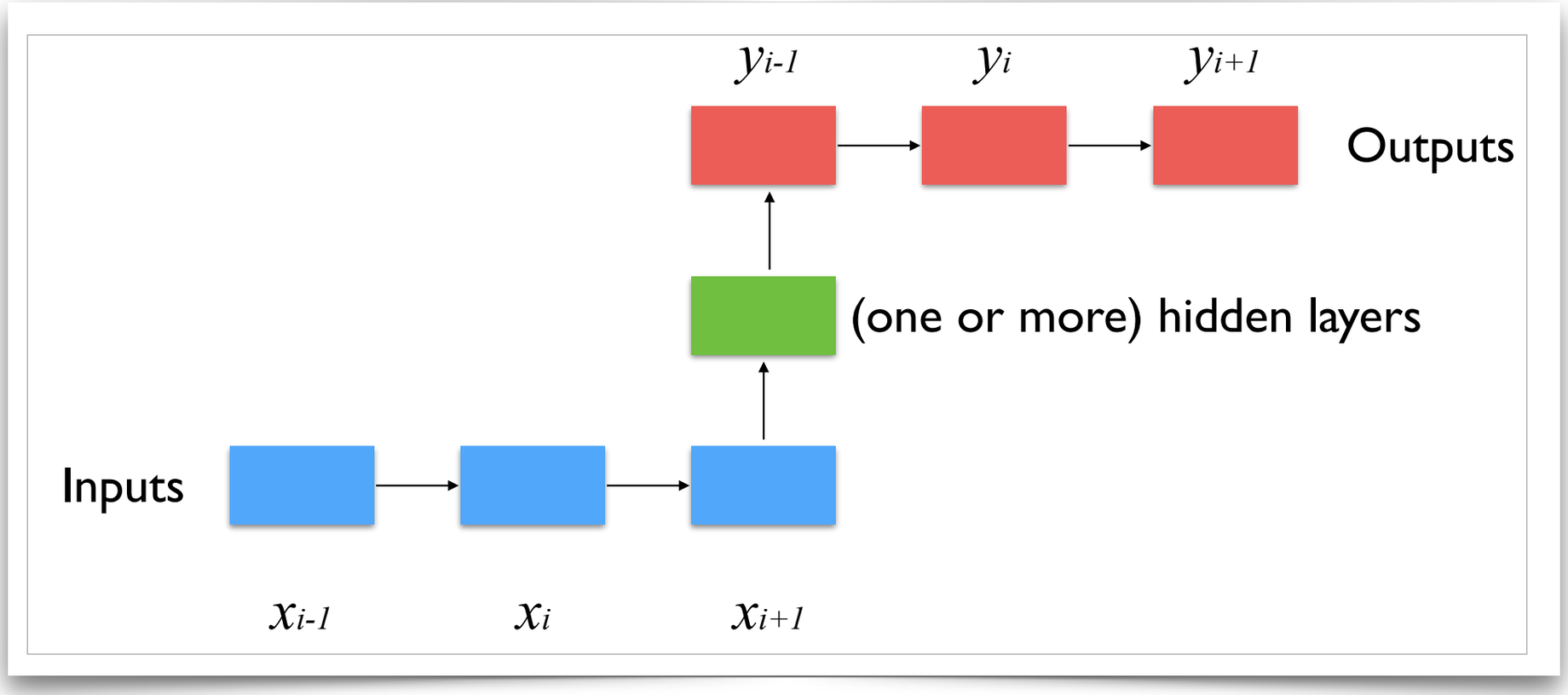


Reconstruction from original



Reconstruction from learnt weights

Recurrent neural nets



A sequence-to-sequence recurrent neural network

Convolutional neural nets

A convolutional neural network with convolutional, max pooling and fully-connect dense layer.

Libraries and tools

- Various libraries for deep learning available: best known are TensorFlow, Theano, Keras, Caffe, Torch.

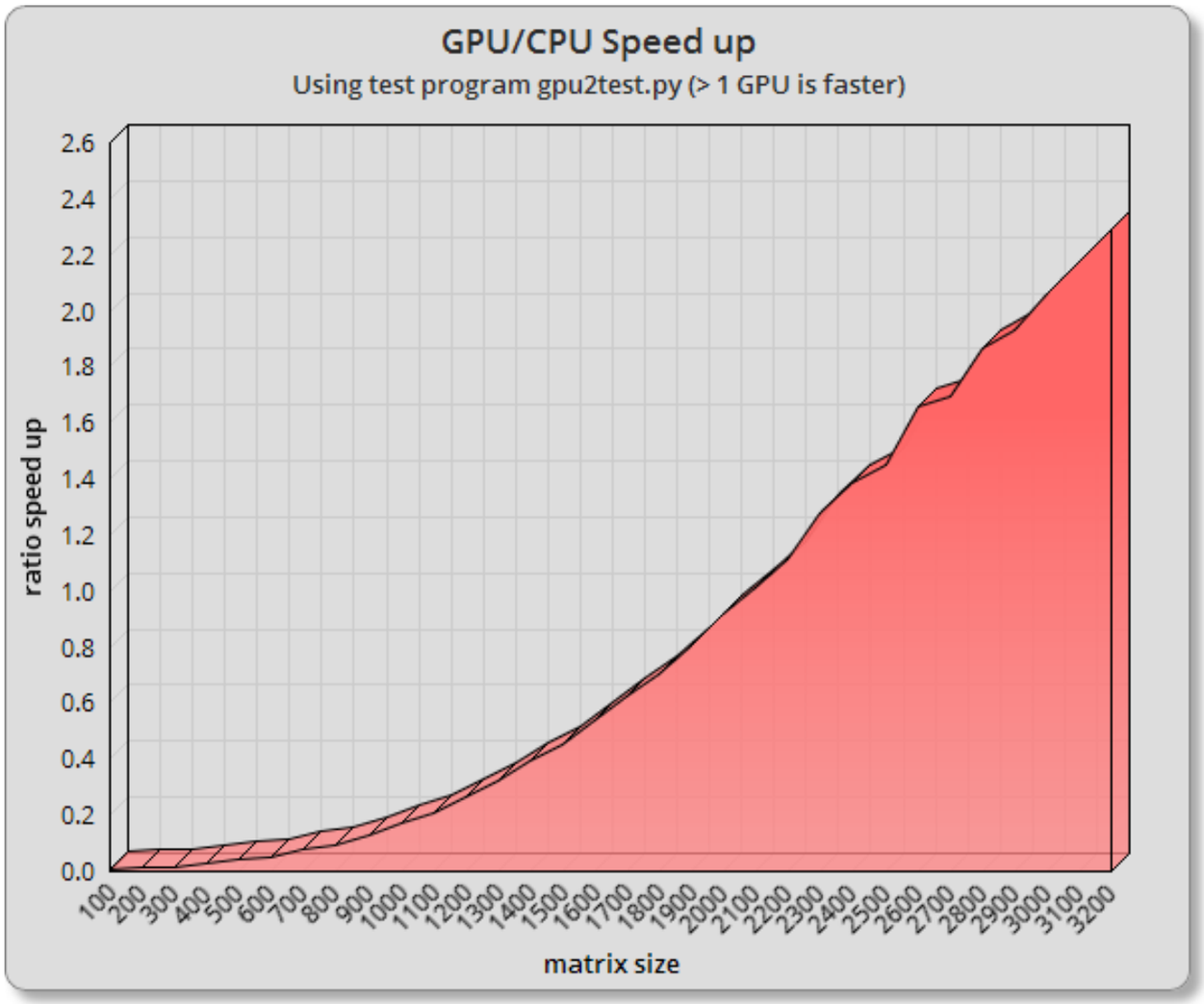
- Very good Python libraries:
  - numpy = mathematical functions and matrix operations
  - sklearn = machine learning
  - matplotlib = plots and visualisation
  - pandas = data manipulation and analysis

HPC and GPUs

- Deep learning involves matrix multiplication, so possible speed-ups with GPUs

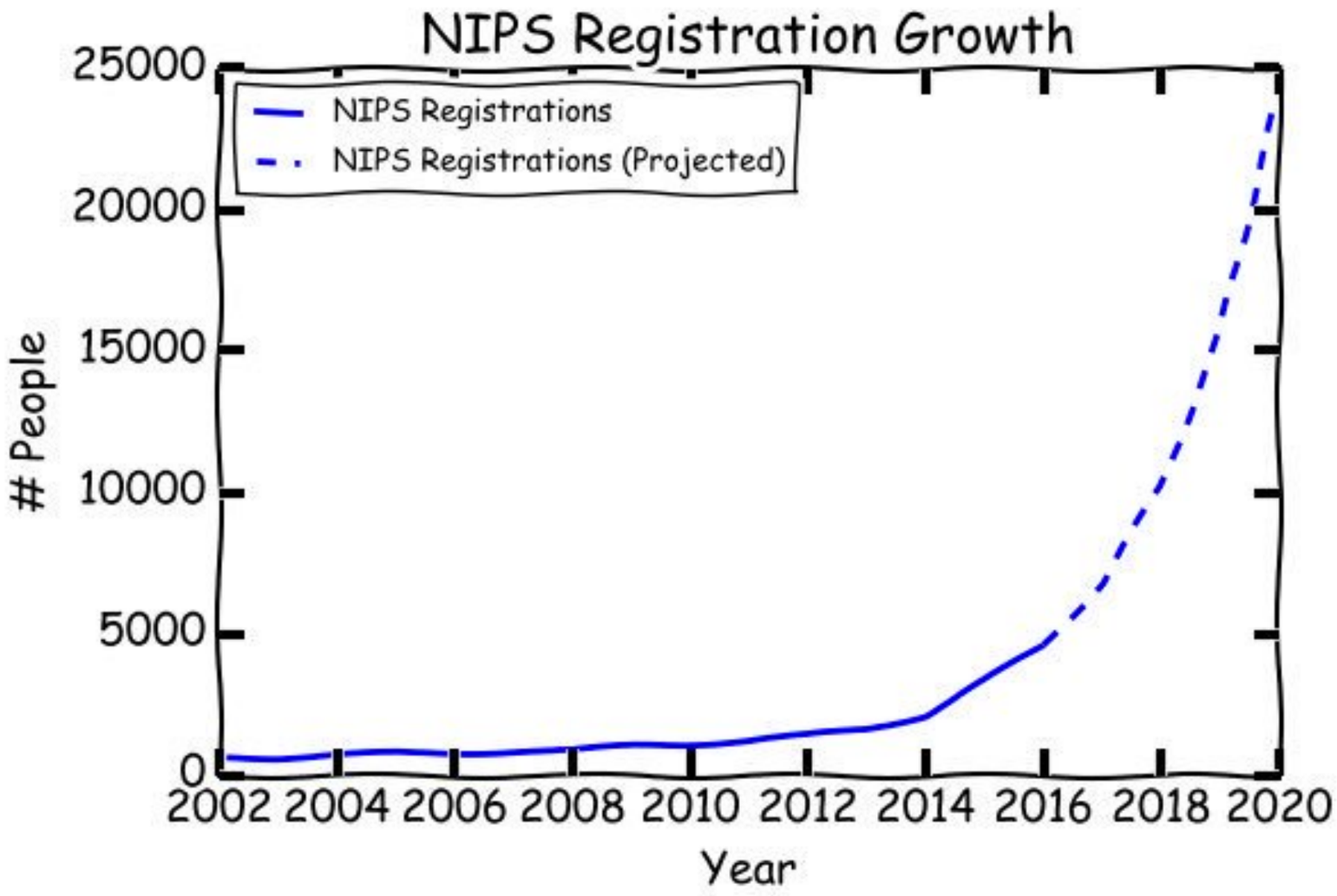
- Nvidia, Intel, Google make specific hardware for deep learning

- Depends on matrix size, though - GPUs are not faster by default



from Darren Bird

Why now?



NIPS (Neural Information Processing Systems) is the leading conference on deep learning and neural networks.

Why now?

- Deep learning is not new - it's been around since the 1960s but was unsuccessful before 2006.

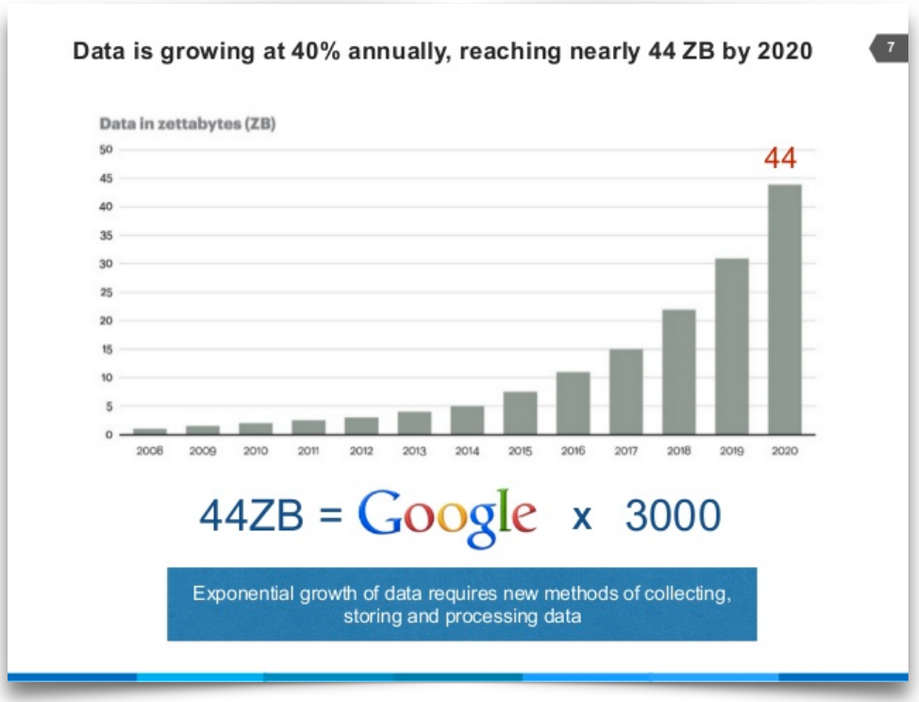
- What has changed in terms of algorithms?
  - New methods for unsupervised pre-training
  - More efficient parameter estimation methods
  - Better understanding of model regularisation

Why now?

- Deep learning is not new - it's been around since the 1960s but was unsuccessful before 2006.

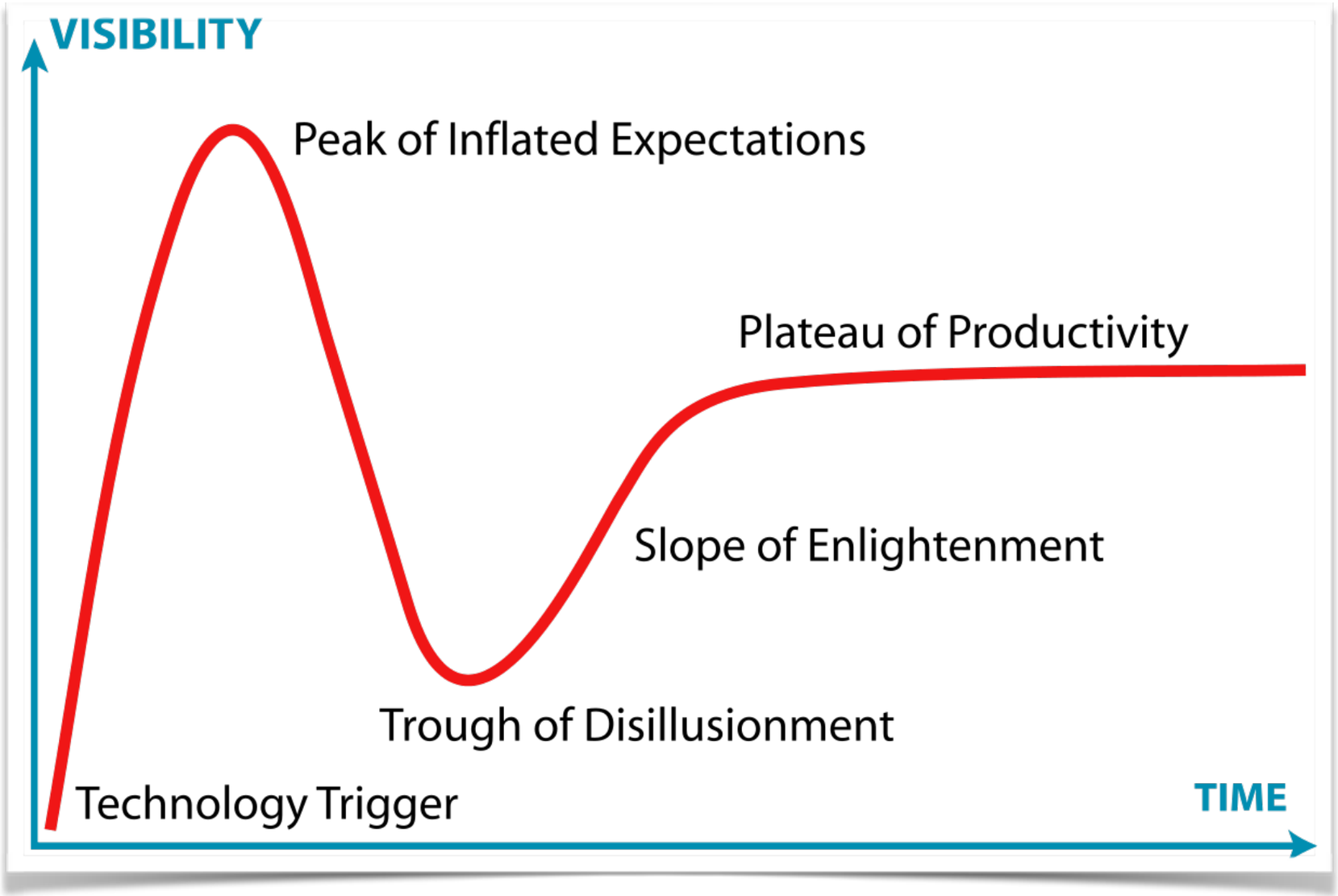
- What has changed in terms of hardware / resources?

- Better hardware and compute power
  - GPUs and large-scale CPU clusters
  - More data available and easier to store and process



https://itsvit.com/blog/what-does-big-data-mean-for-your-business/

Why now?

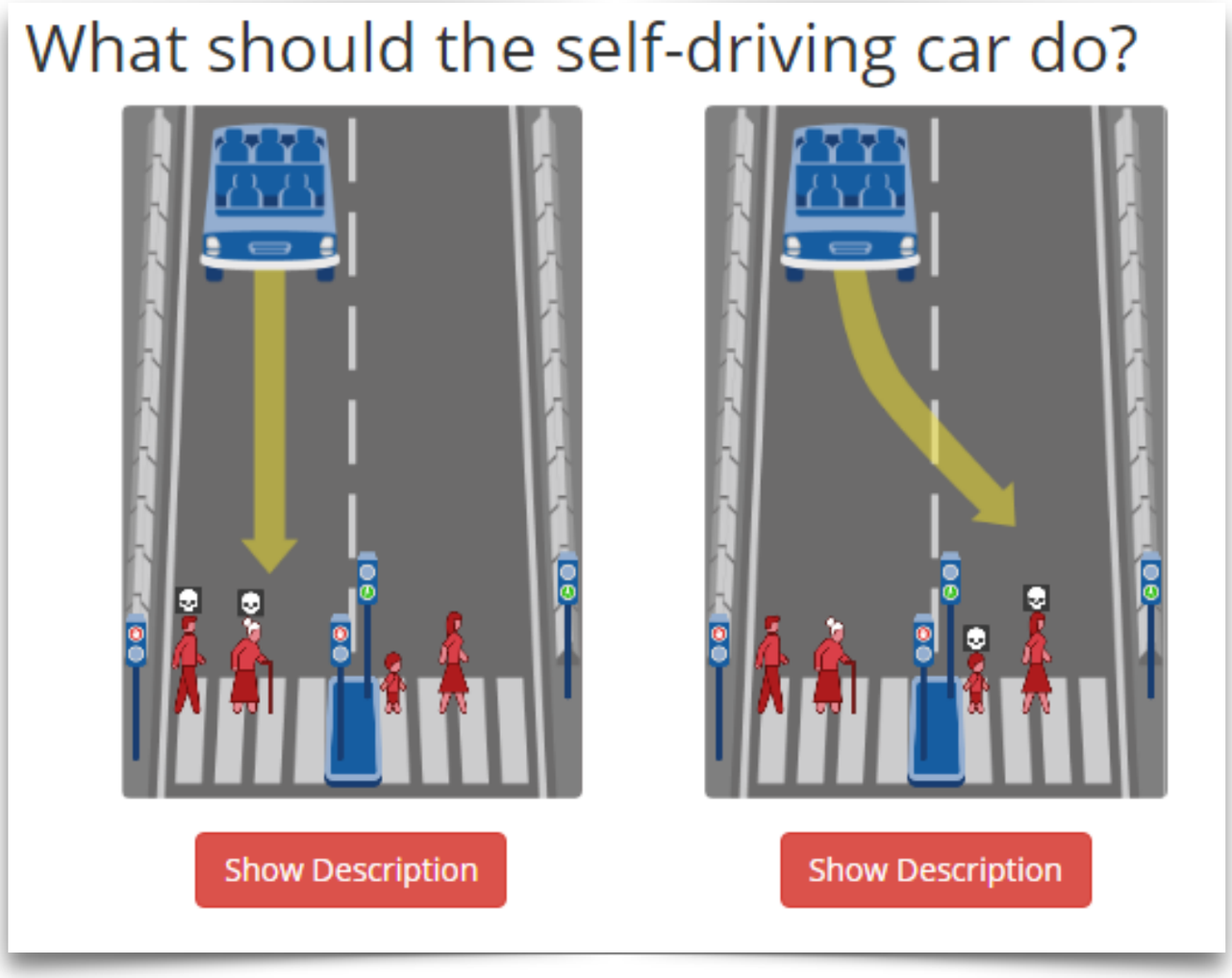


https://en.wikipedia.org/wiki/Hype_cycle

1980 **1990/2000s** **2006** **???**

New problems

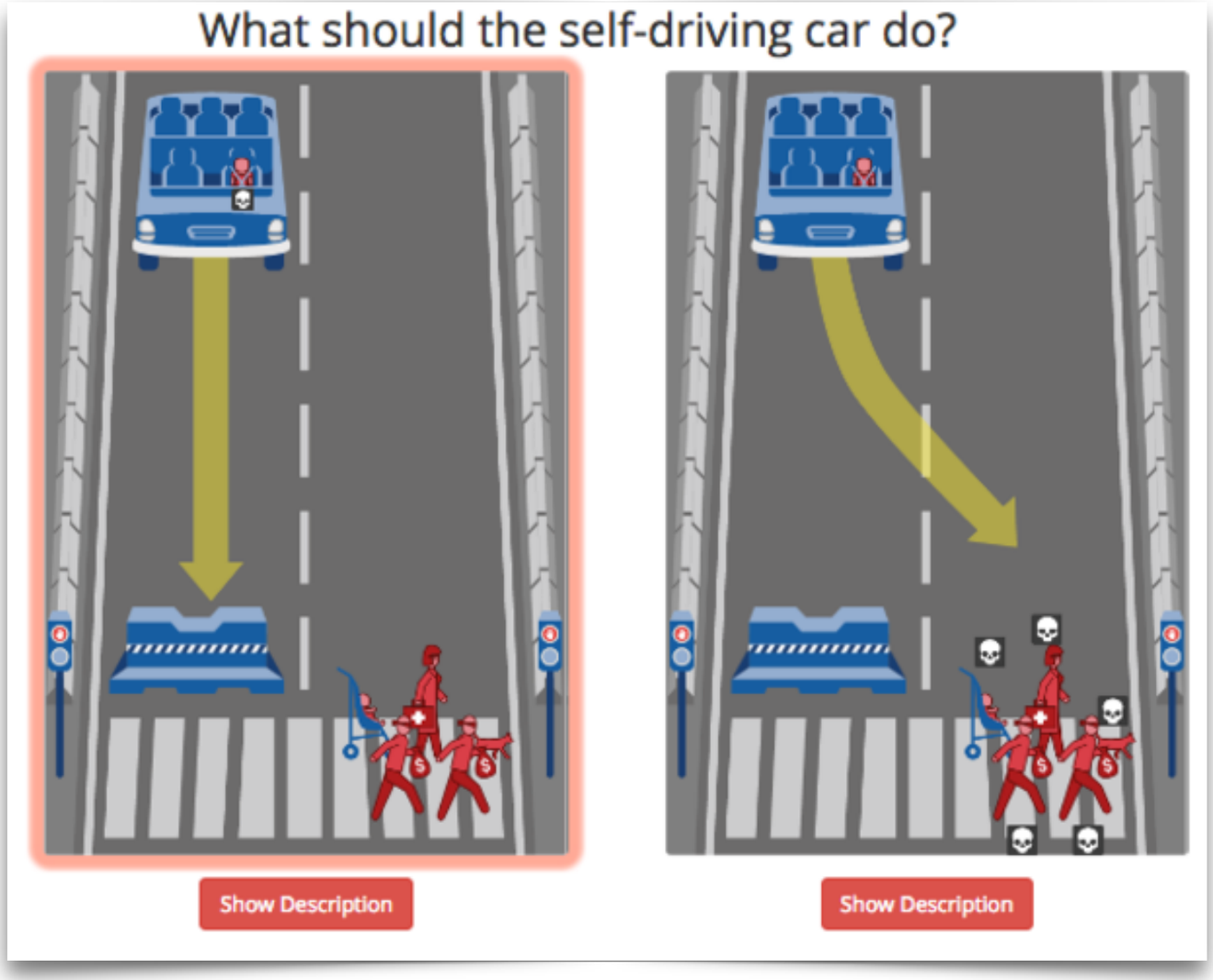
- Neural nets provide no rationales for their decisions
- This is problematic for safety and health-critical and ethical tasks



https://www.weforum.org/agenda/2016/08/the-ethics-of-self-driving-cars-what-would-you-do/

New problems

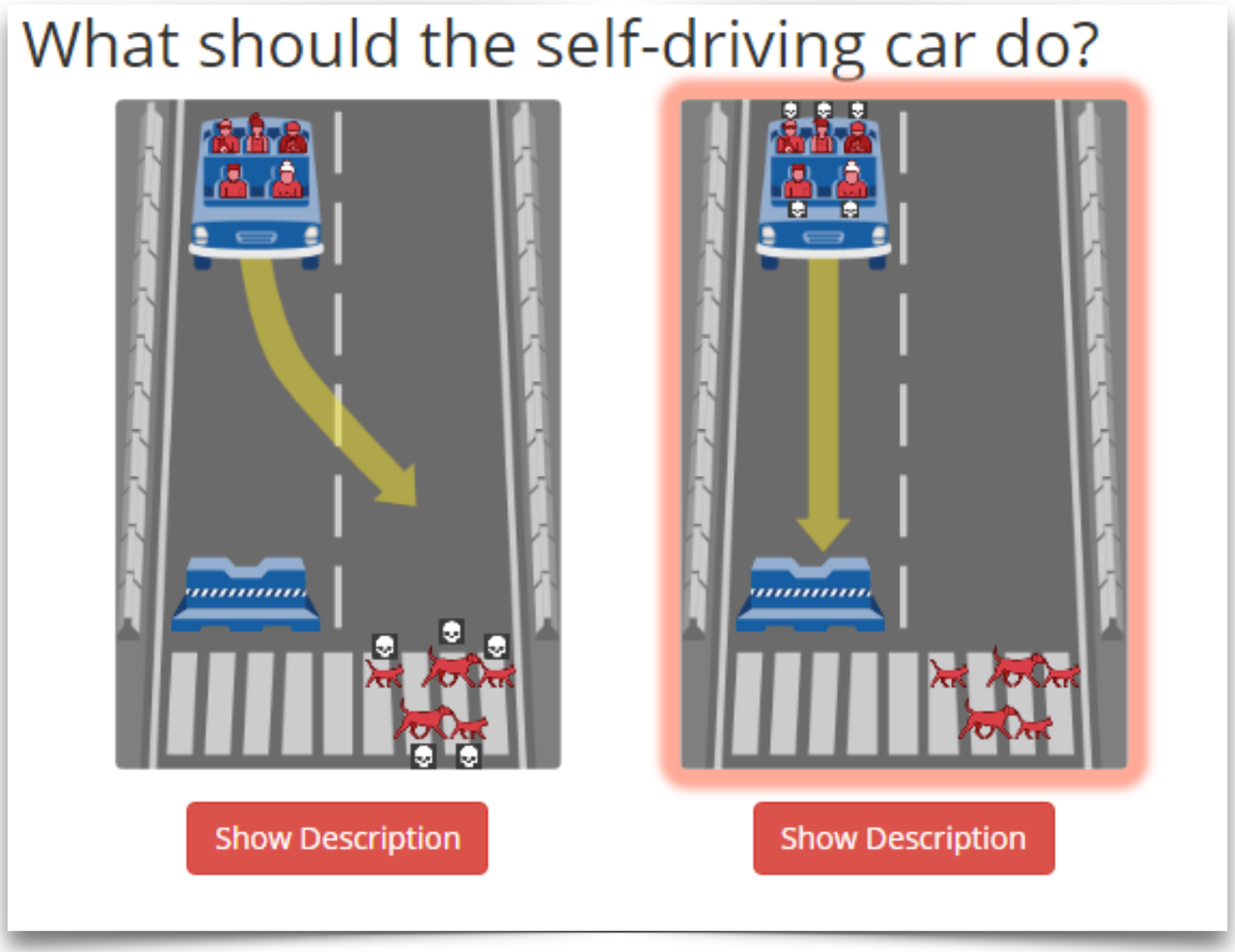
- Neural nets provide no rationales for their decisions
- This is problematic for safety and health-critical and ethical tasks



https://www.weforum.org/agenda/2016/08/the-ethics-of-self-driving-cars-what-would-you-do/

New problems

- Neural nets provide no rationales for their decisions
- This is problematic for safety and health-critical and ethical tasks



https://www.weforum.org/agenda/2016/08/the-ethics-of-self-driving-cars-what-would-you-do/

Tutorials - lab sessions

3 lab sessions planned:

- MNIST with Keras
- MNIST with TensorFlow
- Image Description generation

(Backpropagation)

Invited seminar

Dr Yannis Konstas, Heriot-Watt University, Edinburgh

Building Adaptable and Scalable Natural Language Generation Systems

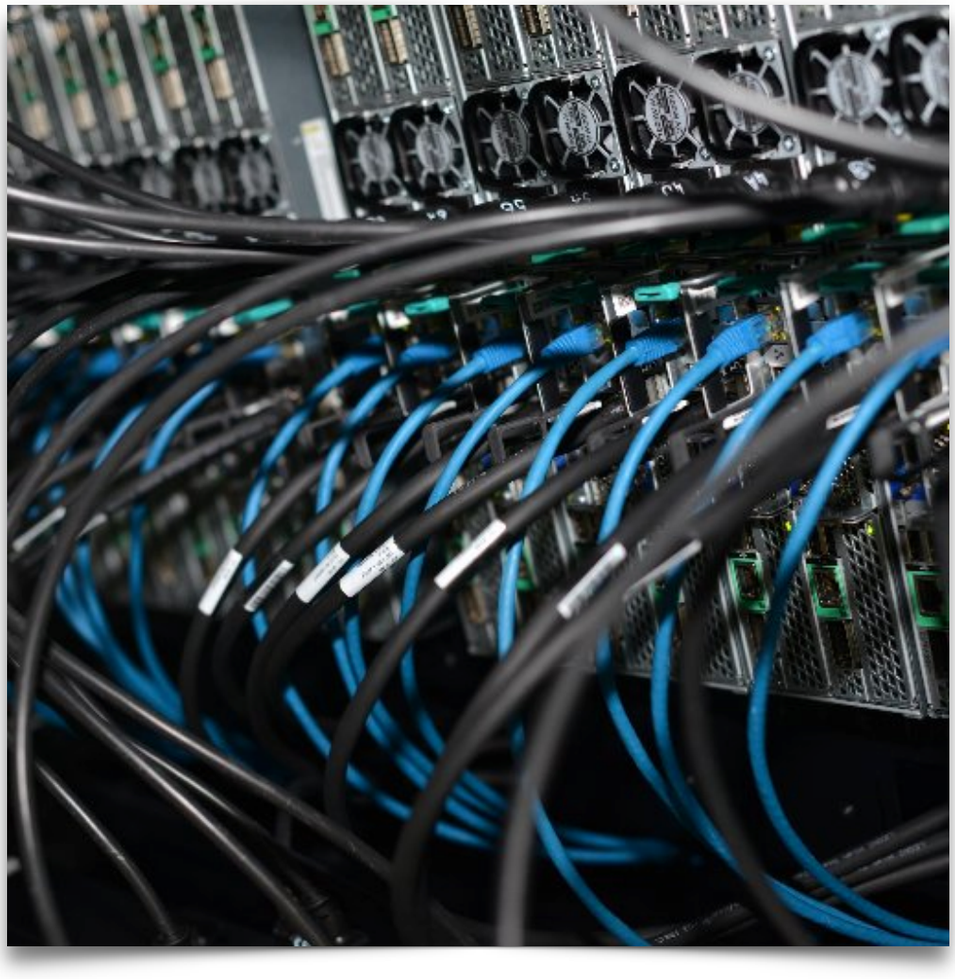
Traditionally, computers communicate with humans by converting computer-readable input to human-interpretable output, for example via graphical user interfaces. My research focuses on building programs that automatically generate textual output from computer-readable input. The majority of existing Natural Language Generation (NLG) systems use hard-wired rules or templates in order to capture the input for every different application, and rely on small manually annotated corpora.

In this talk, I will present a framework for building NLG systems using Neural Network architectures. The approach makes no domain-specific modifications to the input and benefits from training on very large unannotated corpora. It achieves state-of-the-art performance on a number of tasks, including generating text from meaning representations and source code. Such a system can have direct applications to intelligent conversation agents, source code assistant tools and semantic-based Machine Translation.

Viper HPC

Viper tours on Day 2 for visitors during lunch break.

UoH members can arrange for a separate visit.



http://hpc.wordpress.hull.ac.uk/

Projects

- introduction of project work

References

- pointers to tutorials and papers here