~/warwick-wake-ml

session 2: machine learning for modern astronomy

Thomas Killestein

WarwickAstro/WAKE_workshops

Recap/feedback

> Any outstanding technical issues?

> What did you find challenging?

> Anything you wanted more of?

> Hackathon: any ideas?

./machine-learning-in-astronomy

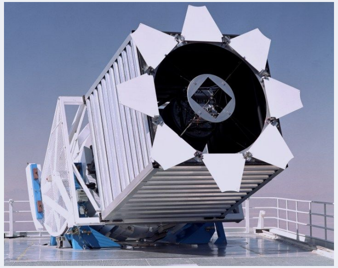
what challenges does astronomy present

machine learning frameworks

prep content for notebooks

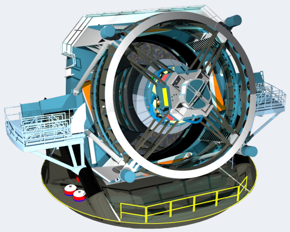
Big data in astronomy

> Astronomy is now among the most data-heavy sciences.

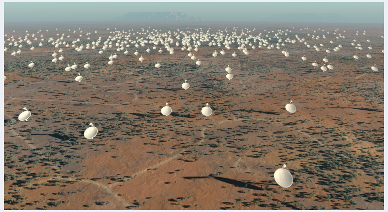


SDSS (1990s)
200GB / night

GOTO (now)
1TB / night



Rubin Observatory (2024-)
20TB / night

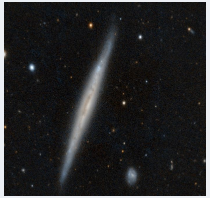


SKA (2030s)
160TB / second

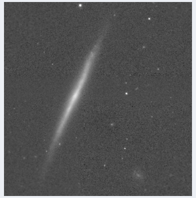
> All of the above make use of machine learning to manage the data volumes involved!

Different data modalities

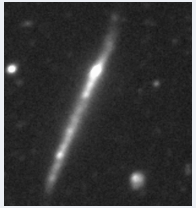
Images:
Spatial



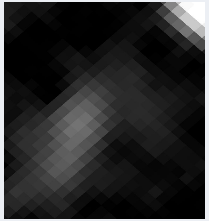
Optical / PS1



NIR / UKIDSS K

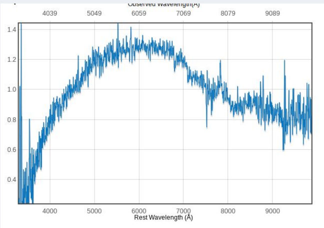


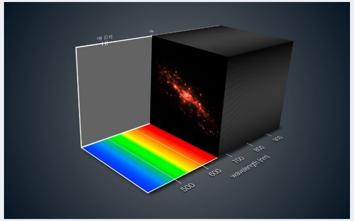
NUV / GALEX



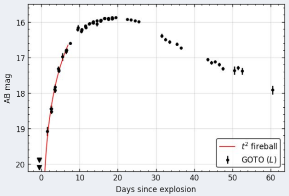
Radio / VLA

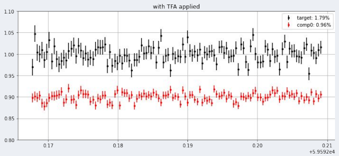
Spectra:
Wavelength



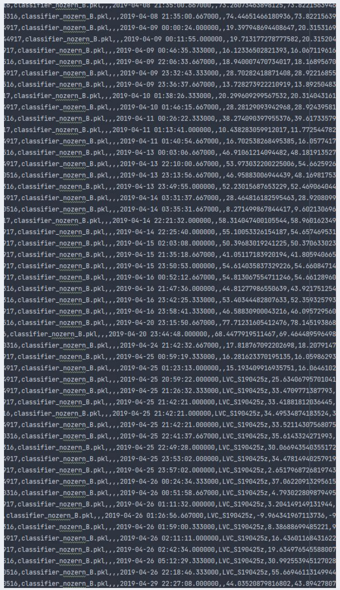


Light curves:
Temporal





Tabular data:
Attributes



Different algorithms for different datasets

> **Images:** rapidly grow in complexity - regular neural networks become prohibitively expensive relatively quickly. *Convolutional neural networks*

> **Time-series:** how can we bake causal behaviour and memory into algorithms? *Recurrent neural networks*

Time series-style techniques also works for wavelength, language, etc.

./deep-learning

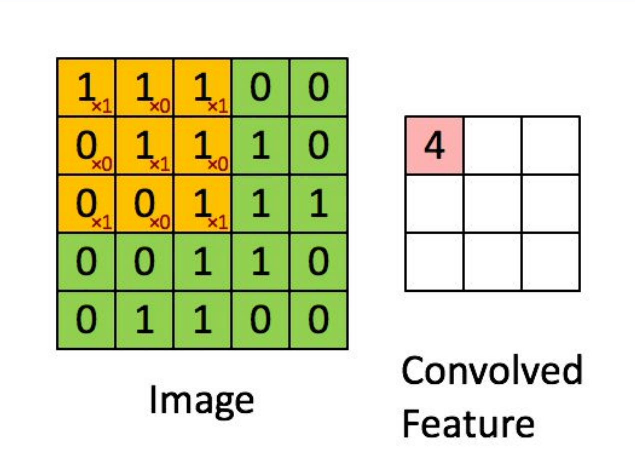
Convolutional neural networks

Instead of using one neuron per input pixel, let's use convolution to create 'feature maps'. Common and well-optimised operation from computer vision!

Filters show strongest 'activation' when neighbourhood of pixels matches the kernel - feature extractors.

Kernel can easily be learned as part of the optimisation process - derivative of convolution is cross-correlation.

Power comes from stacking filters - can very rapidly learn to detect complex features.



Adaptive gradient descent

> **Improve on SGD by:**
>
> > Allowing different model parameters to have different learning rates
>
> > Changing the global learning rate according to the size of our gradients

`adam` (Kingma and Ba, 2014)

Replace gradient with exponentially-weighted average of gradient

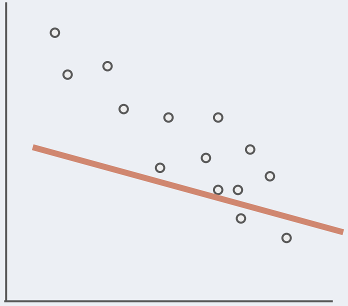
$$x_{n+1} = x_n - \eta \frac{\overline{\nabla \mathcal{L}(x_n)}}{\sqrt{\overline{(\nabla \mathcal{L}(x_n))^2} + \varepsilon}}$$

Compute this update element-wise

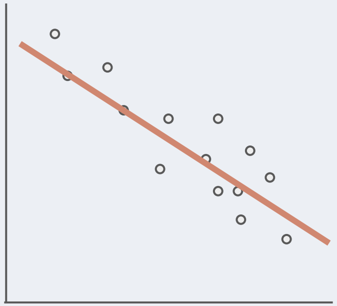
Scale through by exponentially-weighted RMS of gradient.

Overfitting

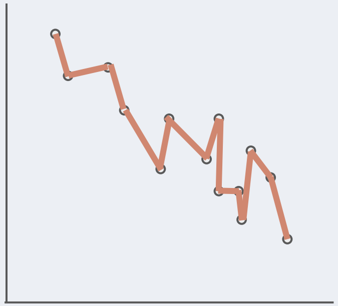
> A central issue in ML is overfitting - where our model simply memorises the input data.



Underfitting:
Model can't represent the data / train harder



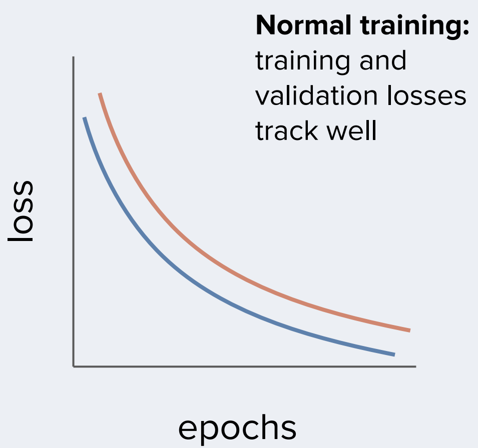
'Good fit':
Model learns a sensible (in context of noise) interpretation of data

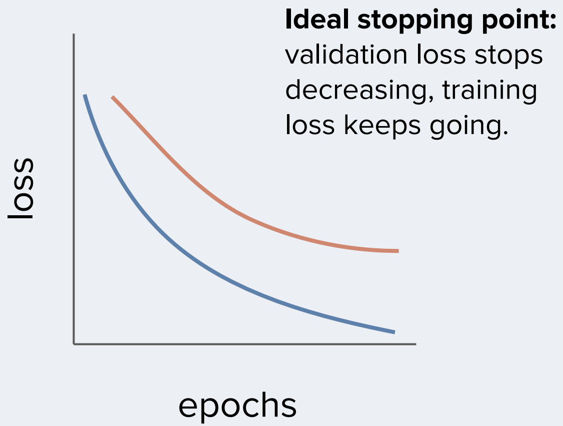


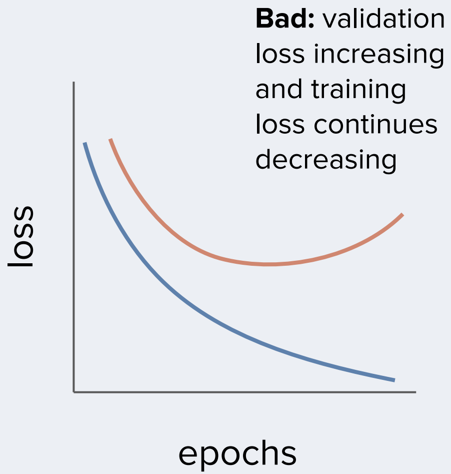
Overfitting:
Model just interpolates between all datapoints, 'learning the noise'

Train, test, **validation** datasets

> So how to guard against overfitting - hold some data out of the training process to check how your model performs on **unseen data**







> You should also leave a portion of data out entirely, as a ‘test’ dataset.

Deep learning frameworks

> Tested and robust implementations of common routines like we wrote yesterday

> Provide a full end-to-end codebase for rapid prototyping, data preparation, training, evaluating, deploying, and testing deep learning algorithms.

> There are many - pick your favourite and stick with it. Very similar to ‘editor wars’

Usually implement:
Model class
Layers
Optimisers/losses/metrics baked in
Data-loaders and pipelining
Some kind of compilation behind the scenes

JAX Flax PyTorch

Theano MXNet TensorFlow

tensorflow and keras

> tensorflow: low-level differentiable linear algebra on CPU/GPU/TPU - provides gradients, execution flow,

> keras: high-level user-friendly API defining configurable layers, optimisers, metrics, and more.

keras
Specify your model, loss, optimiser, and metrics

→

tensorflow [Python]
Builds model graph out of ‘primitives’

→

tensorflow C++ backend
Highly-optimised code for tensor ops

How to build a keras model

```
model = tf.keras.Sequential(
    [
        tf.keras.layers.Something(),
        tf.keras.layers.SomethingElse(),
        tf.keras.layers.Something()
    ]
)

model.compile(optimiser, loss, metrics=[metric1, metric2]
```

Keras models are initialised as stacks of layers

Layers take data as input, apply some transformation, and yield this as output for the next layer.

Compiling a model associates it with an optimiser, loss, and metrics, and makes it trainable.

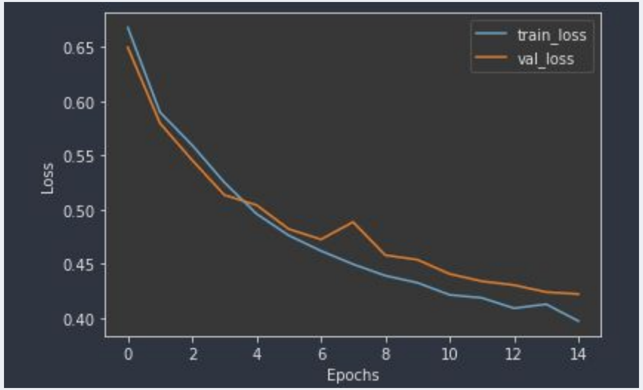
keras models continued

```
history = model.fit(
            train_data, val_data,
            batch_size, epochs
)
```

Our data

Training parameters

```
history.history ->
```



./over to you!

```
~> git clone git@github.com:WarwickAstro/WAKE_workshops.git
~> git checkout stable
~> pip install -r WAKE_workshops/requirements.txt
~> jupyter lab
```

```
https://mybinder.org/v2/gh/WarwickAstro/WAKE_workshops/stable
```