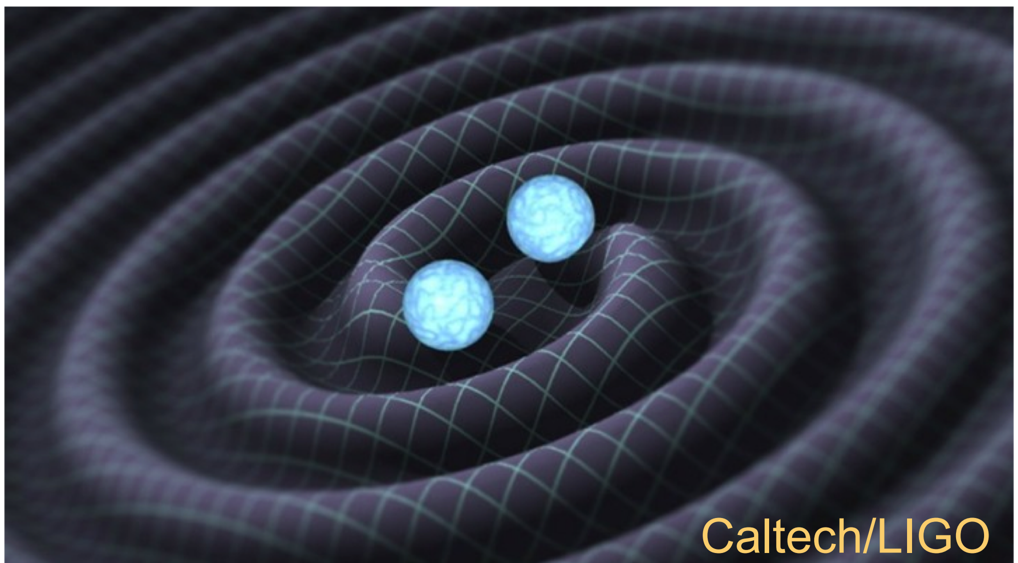
Real-time classification of transients using deep Recurrent Neural Networks

Daniel Muthukrishna
University of Cambridge

Collaborators:
Gautham Narayan (STScI),
Kaisey Mandel (U. Cambridge),
Rahul Biswas (U. Stockholm),
Renee Hložek (U. Toronto)

What have transients been used for?

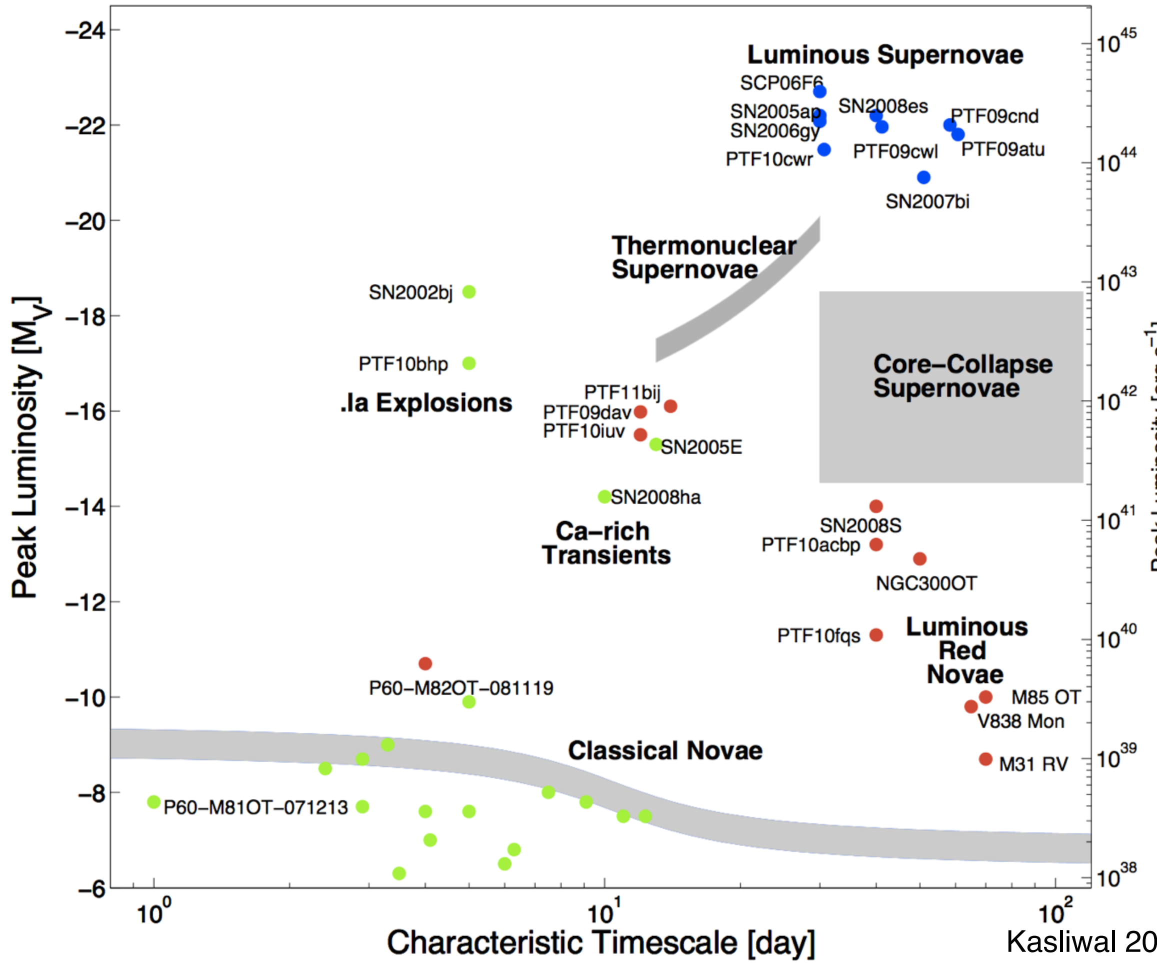
- Discovery of the accelerating expansion of the universe (Type Ia Supernovae)
- Detection of gravitational waves (Kilonovae)
- Production of the universe's heavy elements


Caltech/LIGO


NASA/JPL-Caltech

The known transient universe

- The transient universe remains largely mysterious
- New surveys will observe observe an unprecedented number of transients
- Need to prioritize follow-up based on class and epoch
- Automated, fast, early classifications are required



Kasliwal 2012

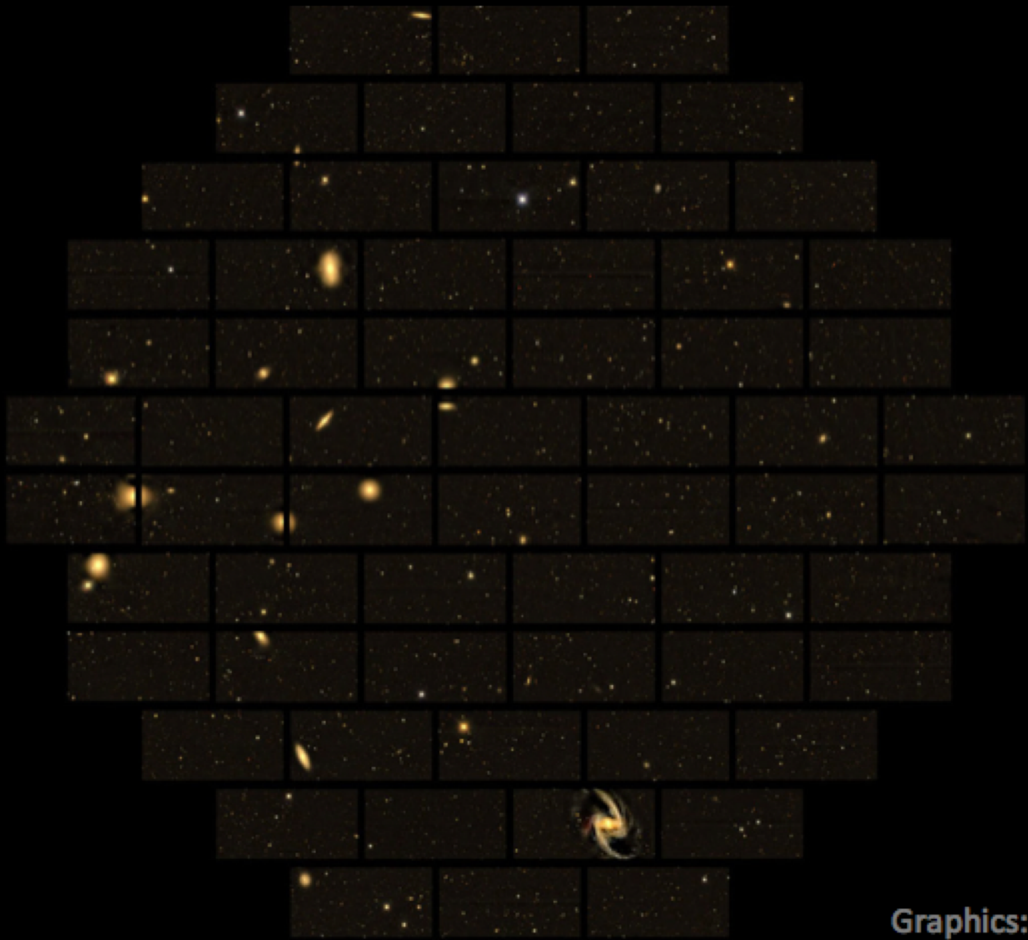

LSST TAKES 20TB OF IMAGES PER NIGHT
~400,000 SNe

Early classification and follow-up

- We have the opportunity to enable detailed studies of progenitor systems and a deeper understanding of a transient's explosion physics.
- Progenitor and explosion mechanism of SNIa is unknown
- ***Since we can't visually examine every alert, we shouldn't just rely on luck to find these events early***

Single Degenerate Channel (Wheelan and Iben '73)

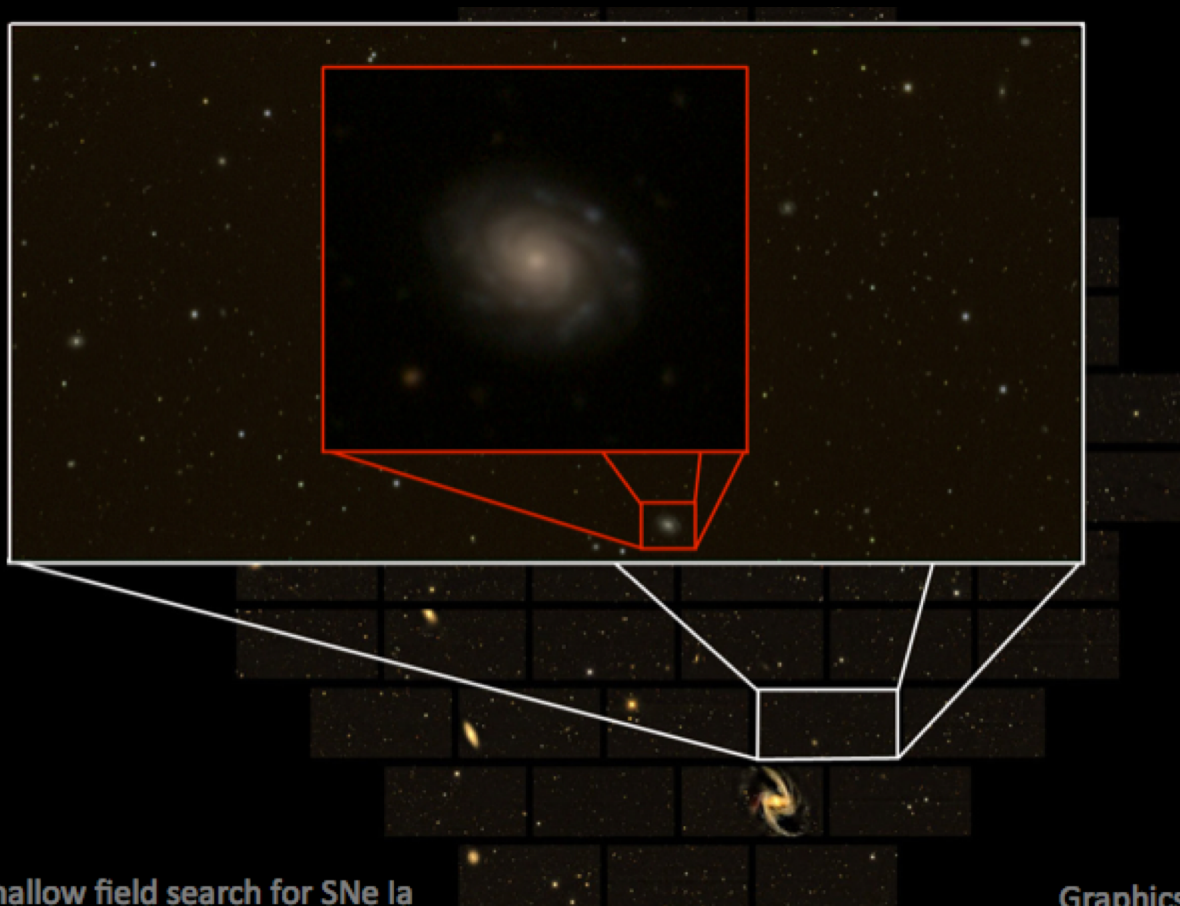
Double Degenerate Channel (Iben & Tutikov '84, Webbink '84)



Graphics: C. D’Andrea

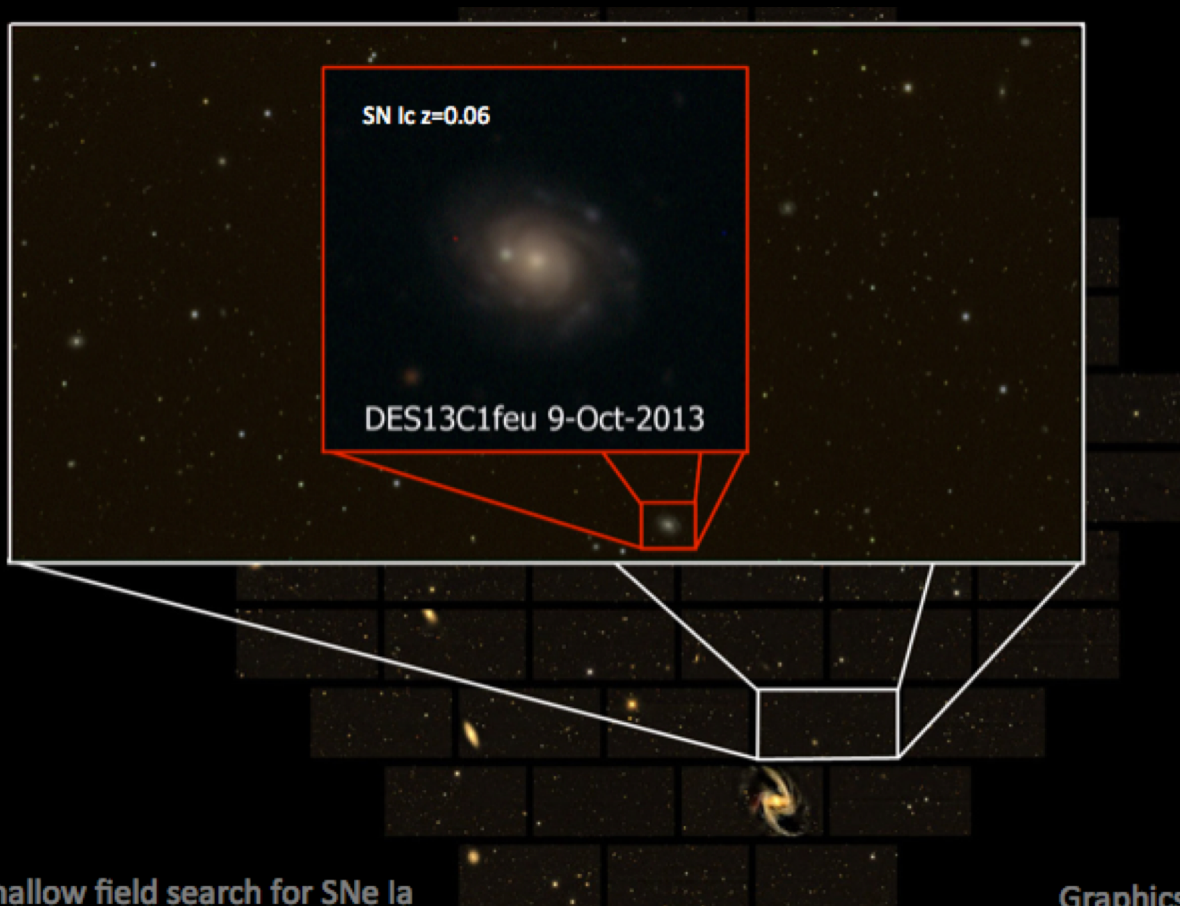
Shallow field search for SNe Ia

Graphics: C. D'Andrea



Shallow field search for SNe Ia

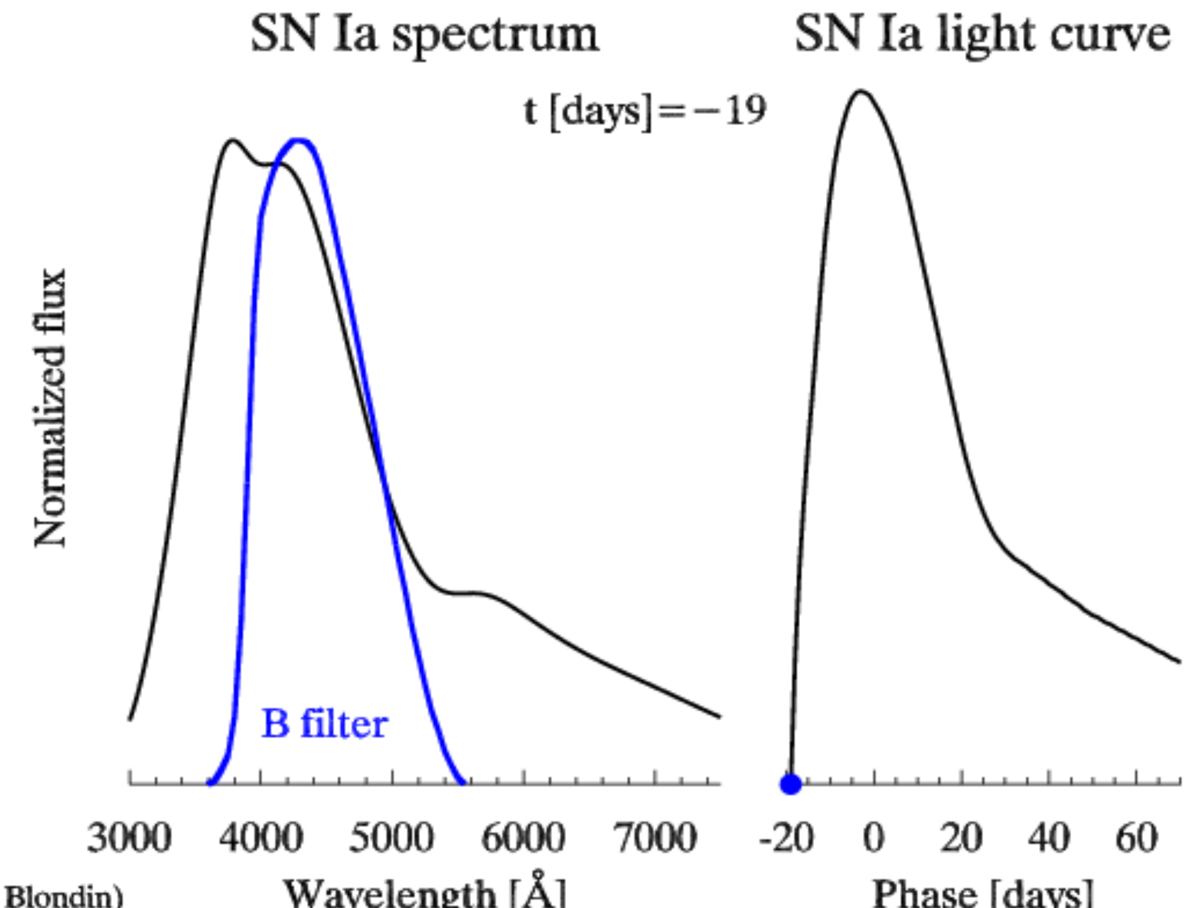
Graphics: C. D'Andrea



Shallow field search for SNe Ia

Graphics: C. D'Andrea

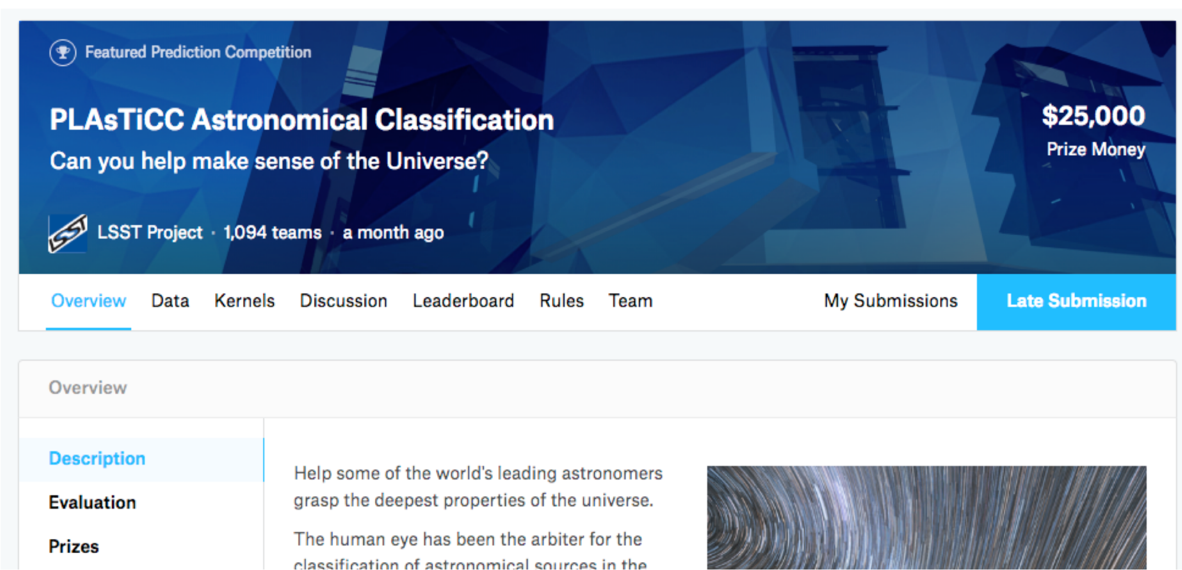
Light curve

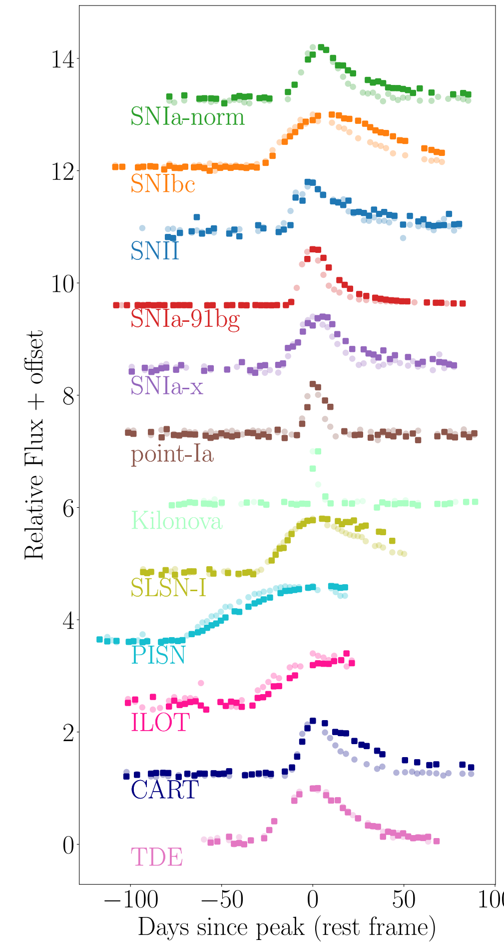


(S. Blondin)

Simulated dataset - PLAsTiCC

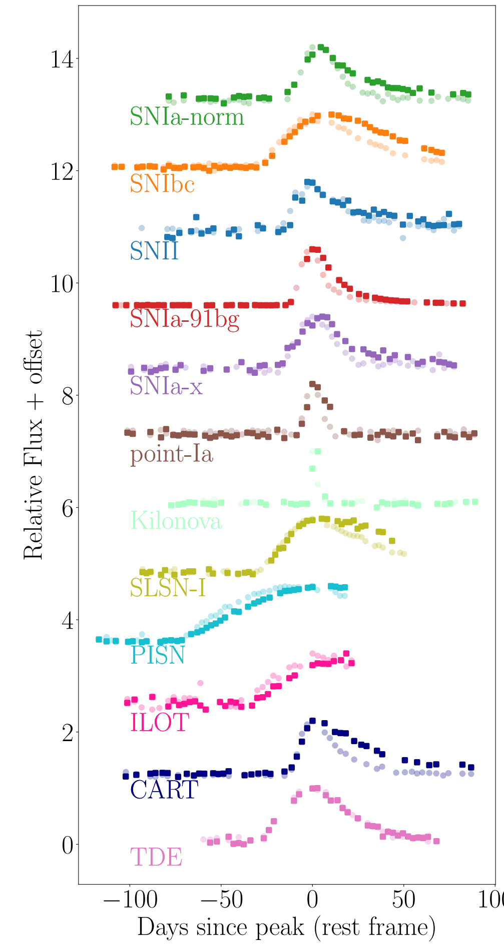
- A comprehensive real training dataset isn't available
  - Cadences/filters/observing conditions vary between surveys
  - Not enough well-covered light curves in a range of classes
- Simulated 48000 light curves split between 12 transient classes with the observing properties of the Zwicky Transient Facility





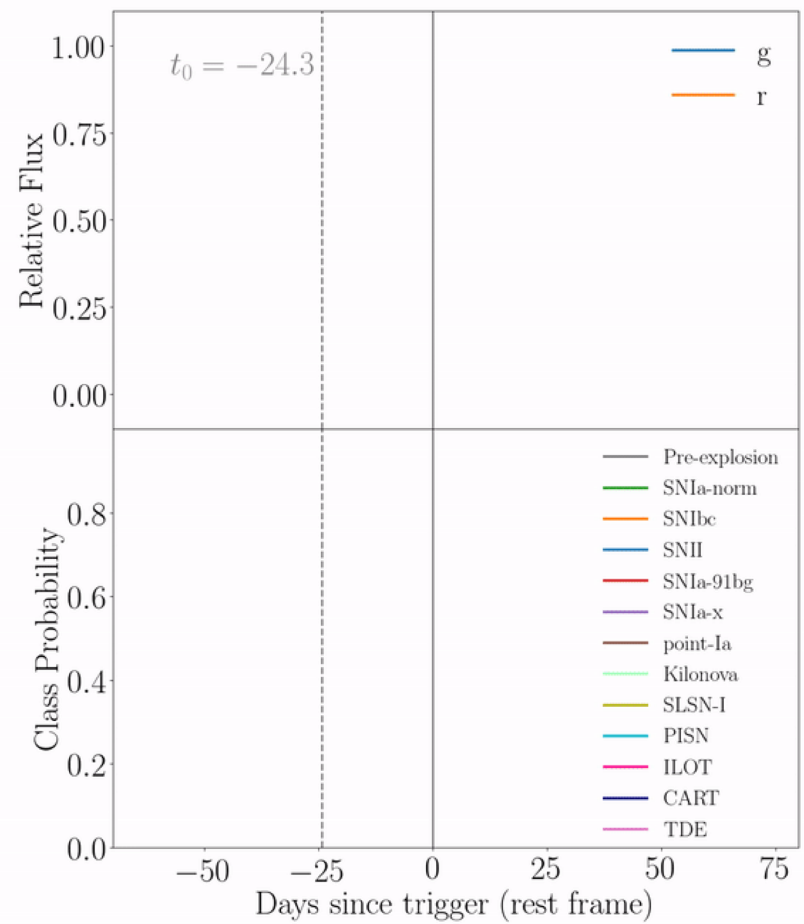
Previous classification attempts

- Require full phase coverage of each light curve
  - Do not make use of the time-series/sequential information
- Very little focus on early classification
- Slow
  - Often require user-defined feature extraction before classification
  - Template matching (slow)
- Often only SNe or SNIa vs non-SNIa classifications



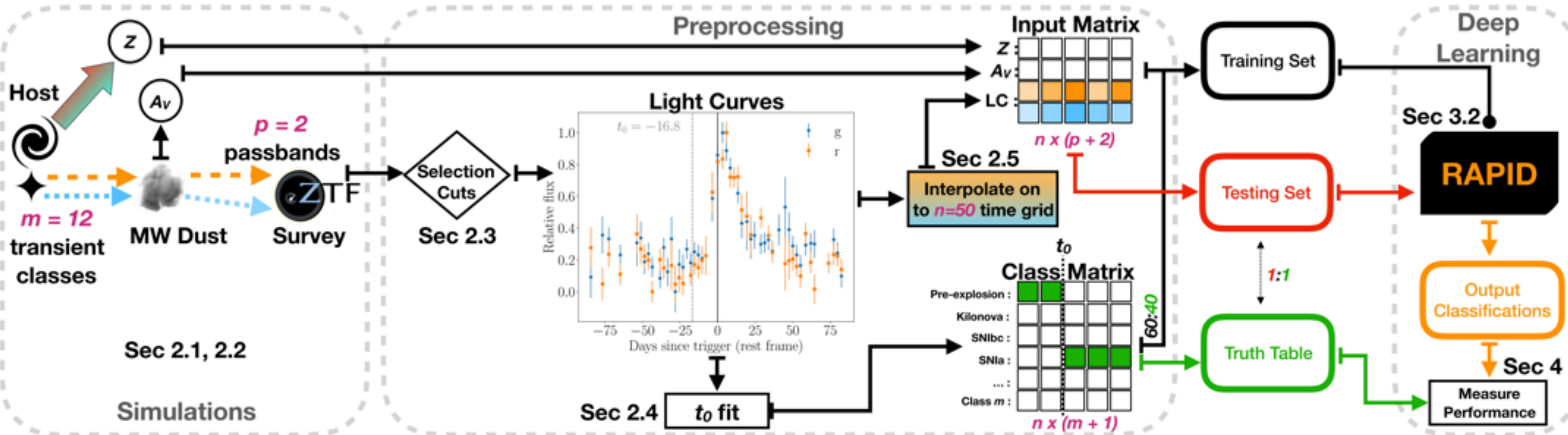
RAPID: Early and real-time classifications

- RAPID: Real-time Automated Photometric IDentification
- Automatically identify transients from within a day of the initial alert to the full life-time of the light curve
- Classifier is trained on 60% of the dataset and is validated on the remaining 40%

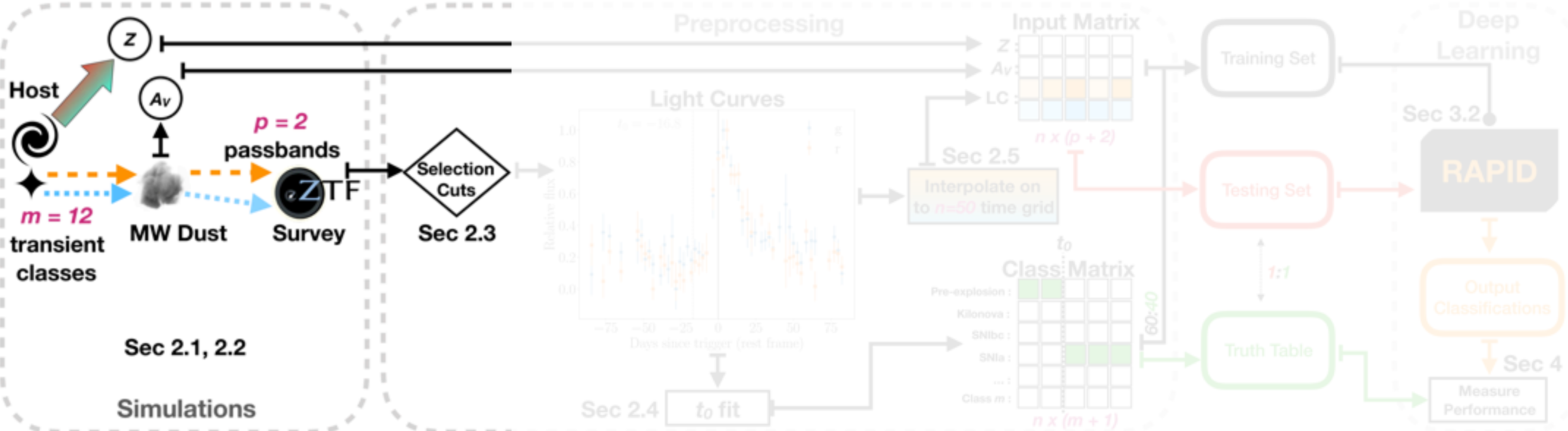


RAPID Design

- Takes multiband photometric information and contextual information as input
- Two classifiers: with and without known redshift

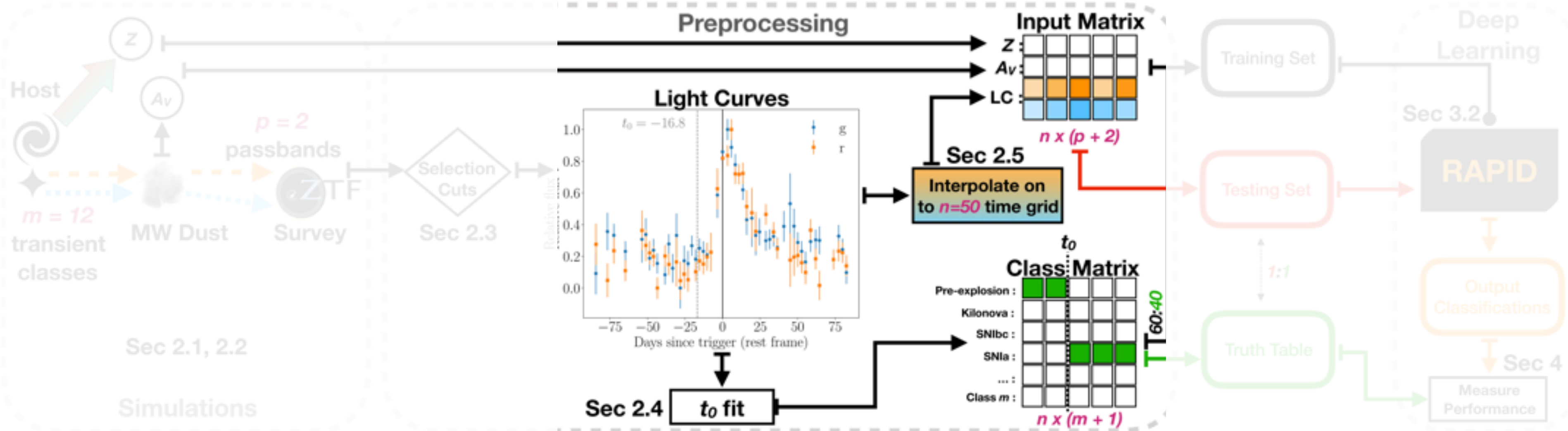


Preprocessing light curves



- Exclude galactic objects
- Correct for Milky Way reddening
- Correct for time dilation and distance if redshift is known

Preprocessing light curves

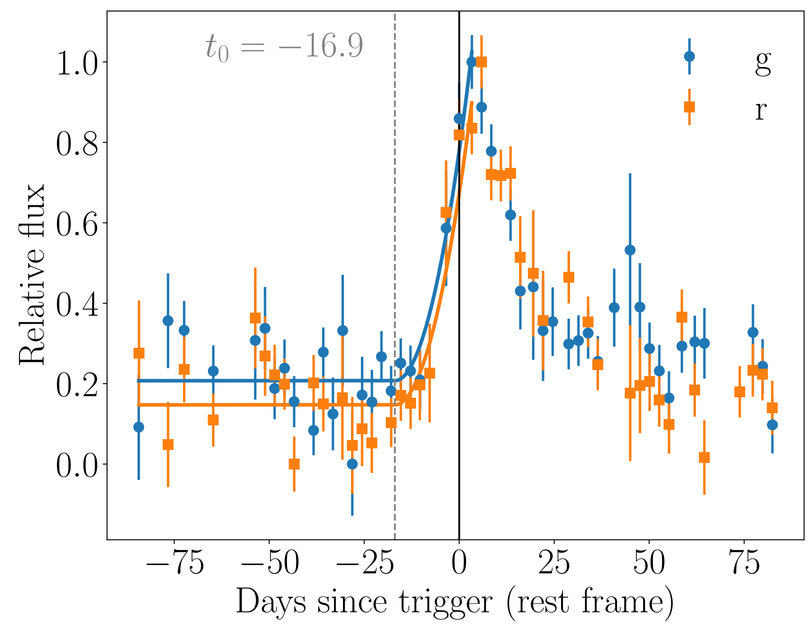


Preprocessing training set

- Estimate explosion time by modelling early part of the light curve with a quadratic step function
- Define Pre-explosion ($t < t_0$) and transient phase ($t \geq t_0$)

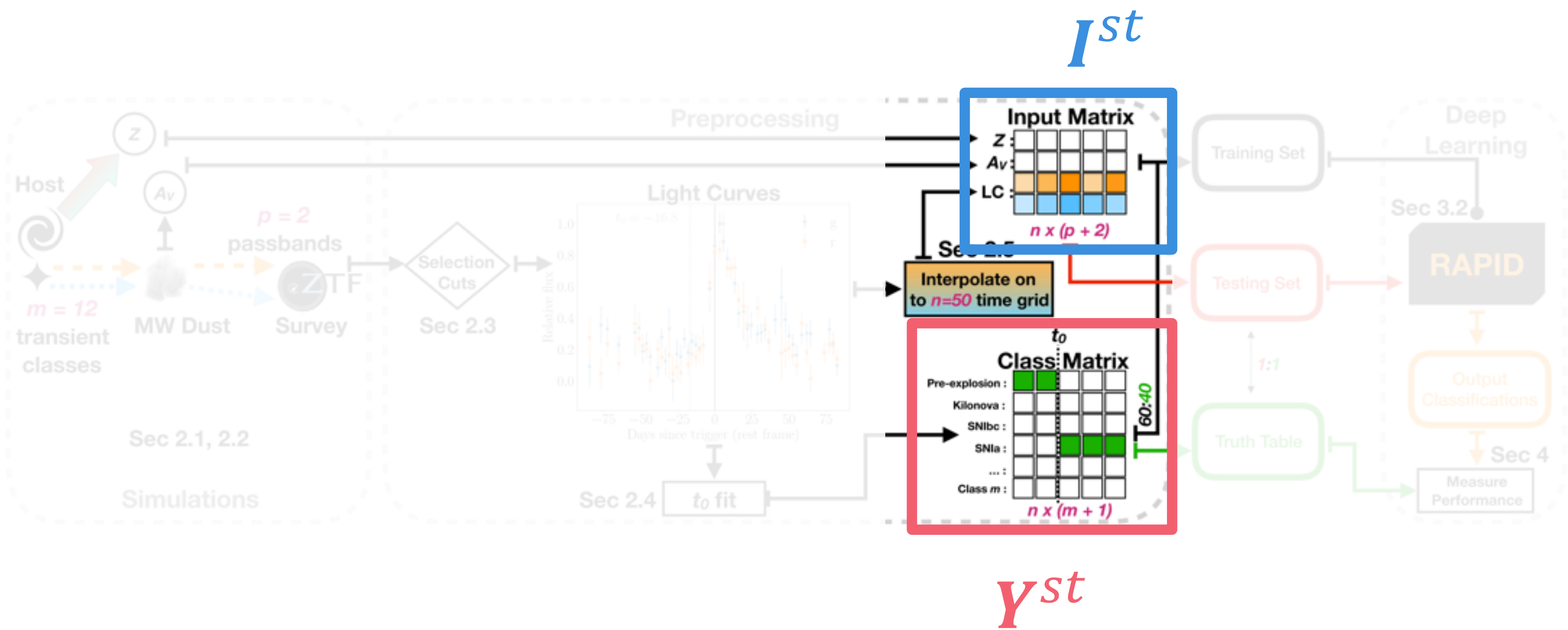
$$L^{\lambda}_{\text{mod}}(t; t_0, a^{\lambda}, c^{\lambda}) = \left[a^{\lambda}(t - t_0)^2\right] \cdot H(t - t_0) + c^{\lambda}$$

$$\chi^2(t_0, \mathbf{a}, \mathbf{c}) = \sum_{\lambda} \sum_{t=-\infty}^{t_{\text{peak}}} \frac{[L^{\lambda}_{\text{data}}(t) - L^{\lambda}_{\text{mod}}(t; t_0, a^{\lambda}, c^{\lambda})]^2}{\sigma^{\lambda}(t)^2}$$



- Sampled the posterior probability $\propto \exp\left(-\frac{1}{2}\chi^2\right)$
- Flat uniform prior on t_0: $f(t_0|t) \sim U(-35, 0)$
- Flat improper prior on other parameters

Preprocessing training set



Model: Framing the Problem

- Aim: Model a function that maps an input multi-passband light curve matrix, $\boldsymbol{I}^{st}$, for transient s up to a discrete time t onto an output probability vector

$$\boldsymbol{y}^{st} = \boldsymbol{f}_t(\boldsymbol{I}^{st}; \boldsymbol{\theta})$$

- To quantify the discrepancy between the model probabilities $\boldsymbol{y}^{st}$ and class labels $\boldsymbol{Y}^{st}$ for class c, we define a weighted categorical cross-entropy (∝ negative log-likelihood of the probabilities of a categorical distribution)

$$H_w(\boldsymbol{Y}^{st}, \boldsymbol{y}^{st}) = -\sum_{c=1}^{m+1} w_c Y_c^{st} \log(y_c^{st})$$

Where the label has a pre-explosion and transient phase:

$$Y_c^{st} = \begin{cases} 1 & \text{if } c \text{ is the true class of transient } s \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

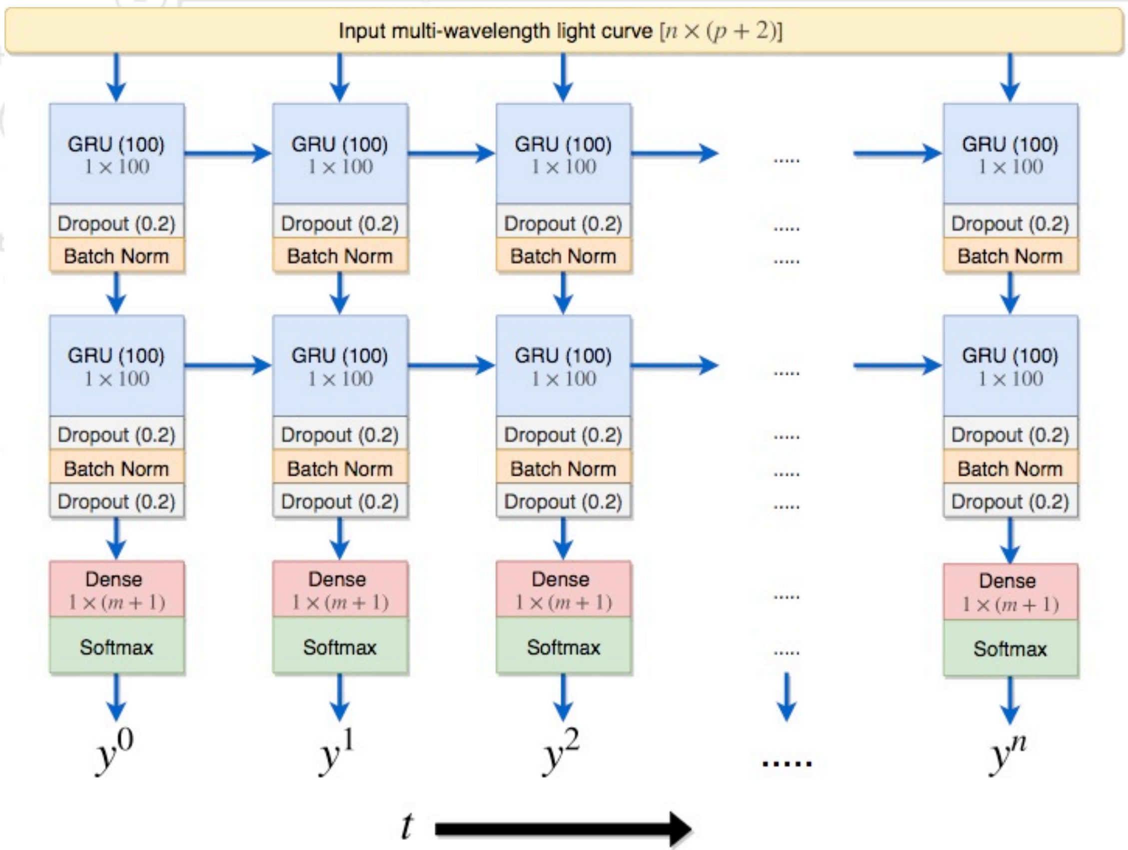
Model: Framing the Problem

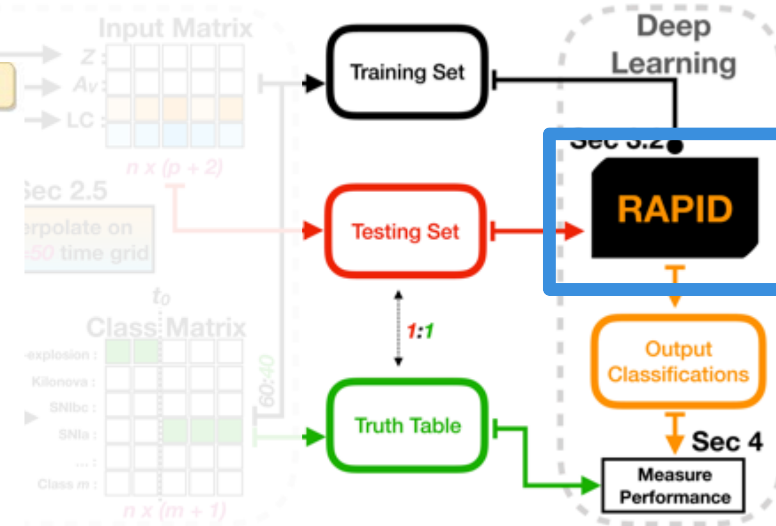
- We define the global objective function as

$$\mathrm{obj}(\boldsymbol{\theta}) = \sum_{s=1}^{N} \sum_{t=0}^{n} H_w(\boldsymbol{Y}^{st}, \boldsymbol{y}^{st})$$

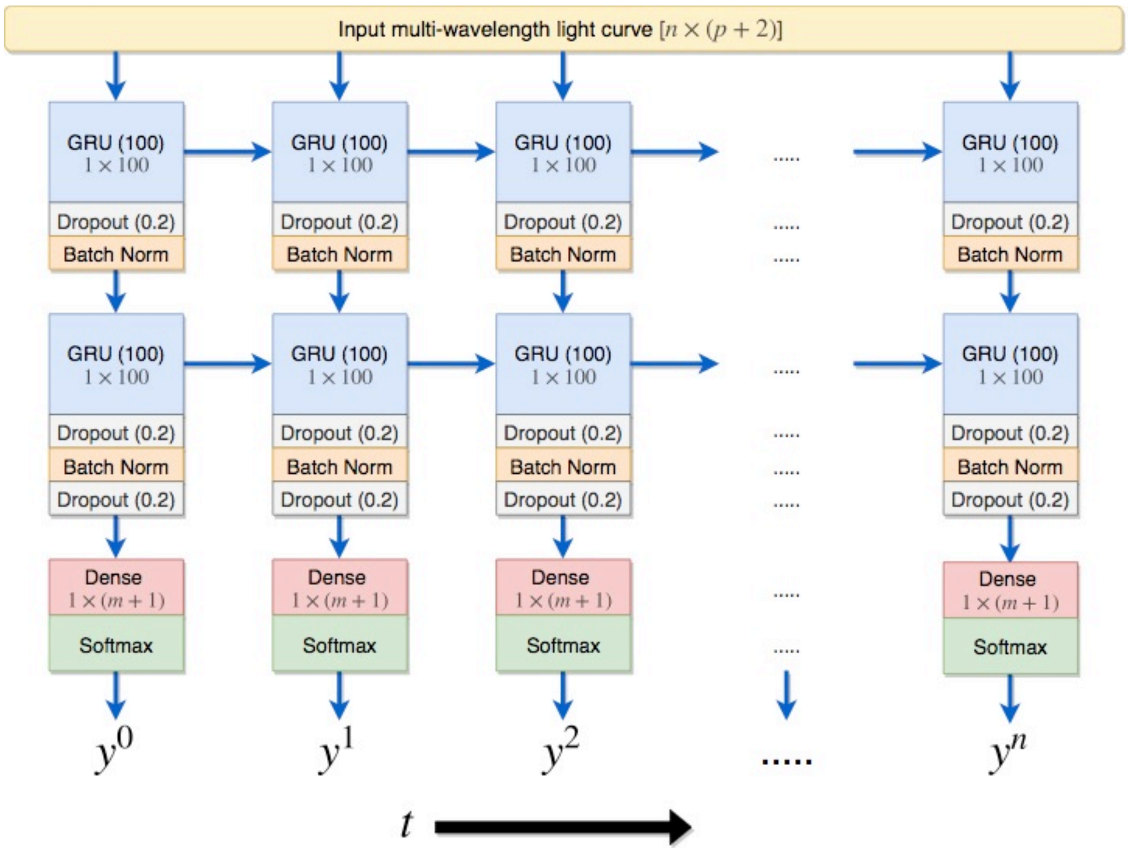
- We use a deep recurrent neural network to determine the optimal values of the parameters, and effectively minimise the objective function with a Stochastic Gradient Descent Optimisation routine: *Adam* (Kingma & Ba 2015)

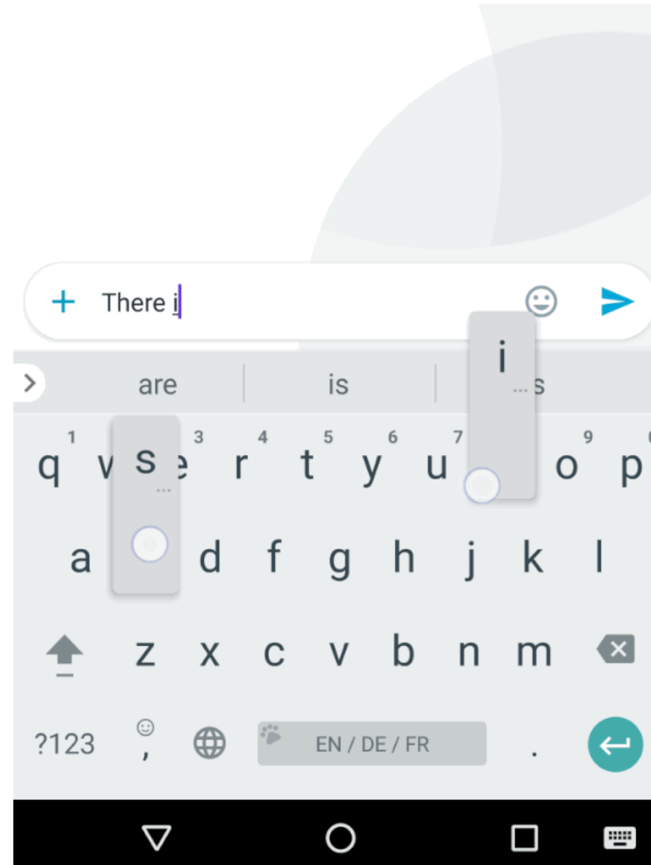
Deep Recurrent Neural Network



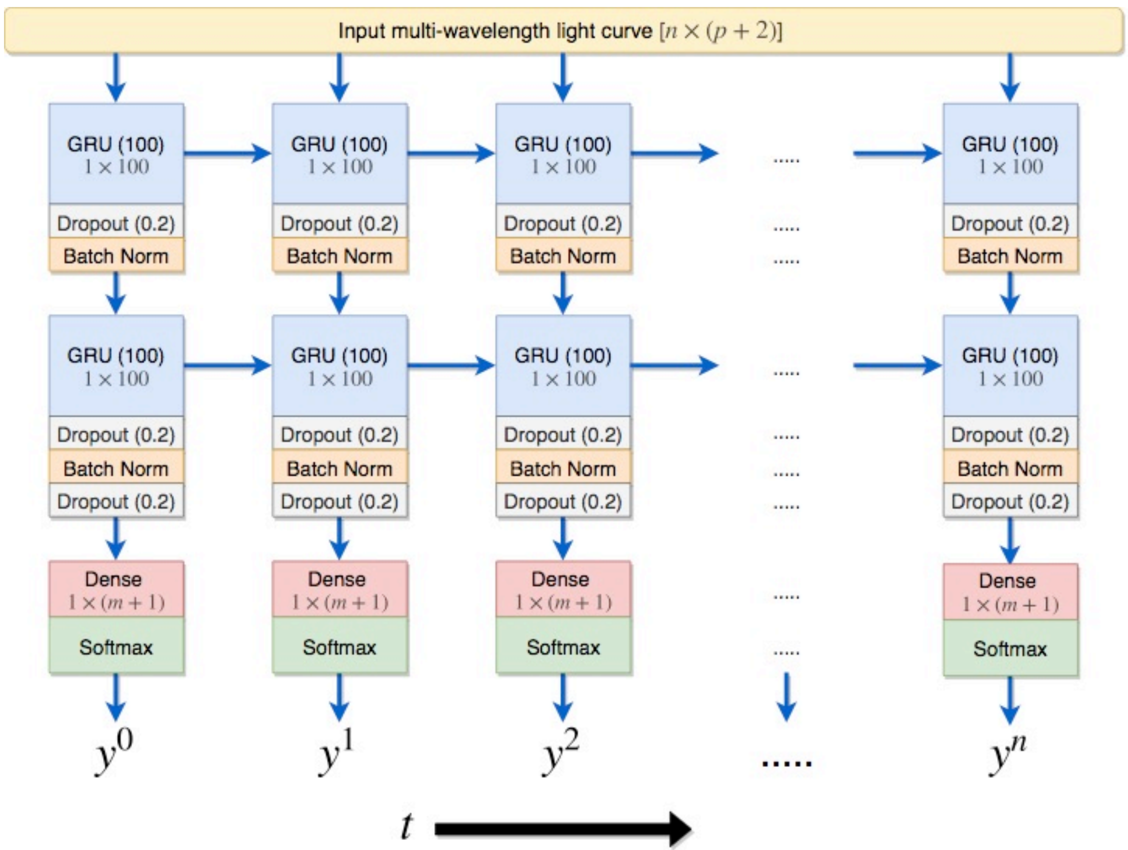


Deep Recurrent Neural Network





Deep Recurrent Neural Network



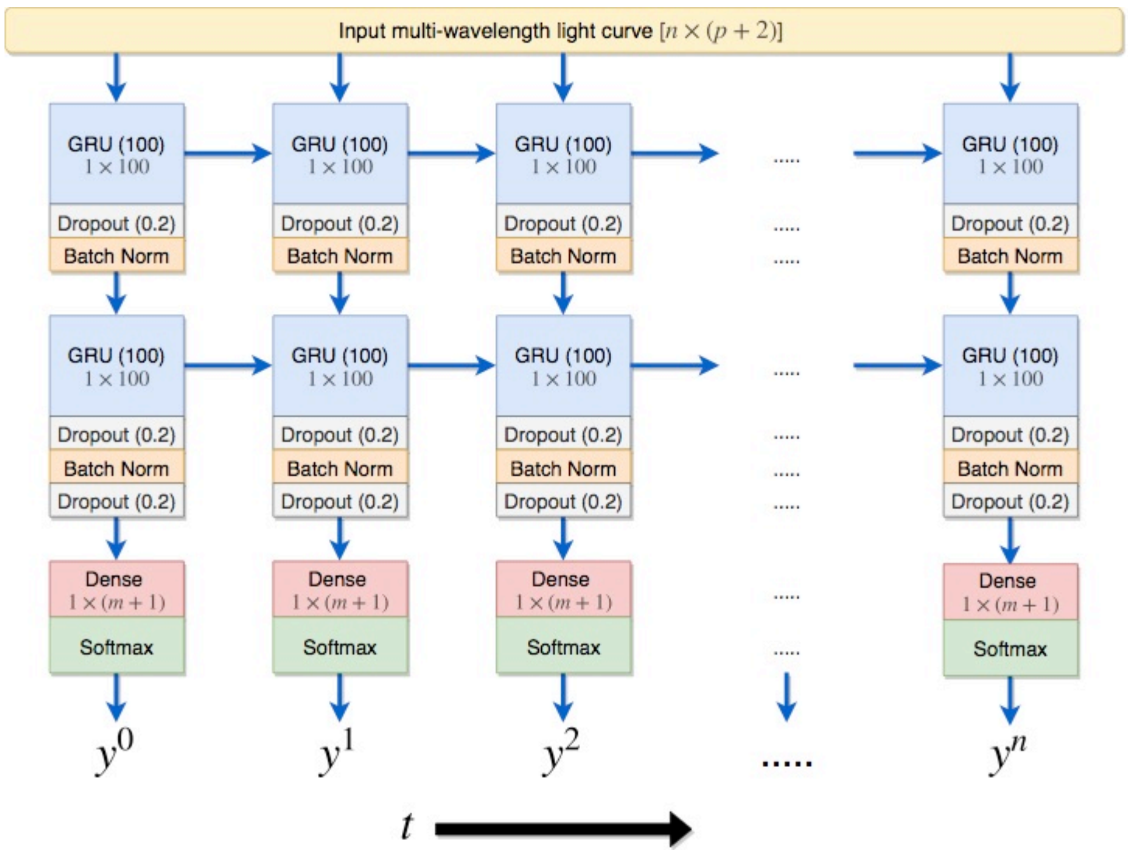
Amazon Echo
(Alexa)

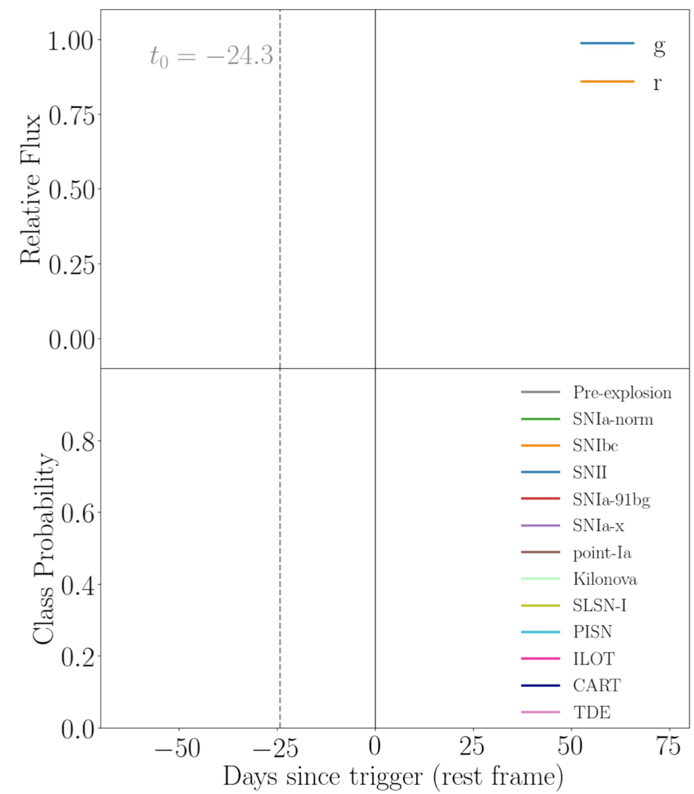
Baidu DuerOS
(xiaodunihao)

Apple Siri
(Hey Siri)

Google Home
(Okay Google)

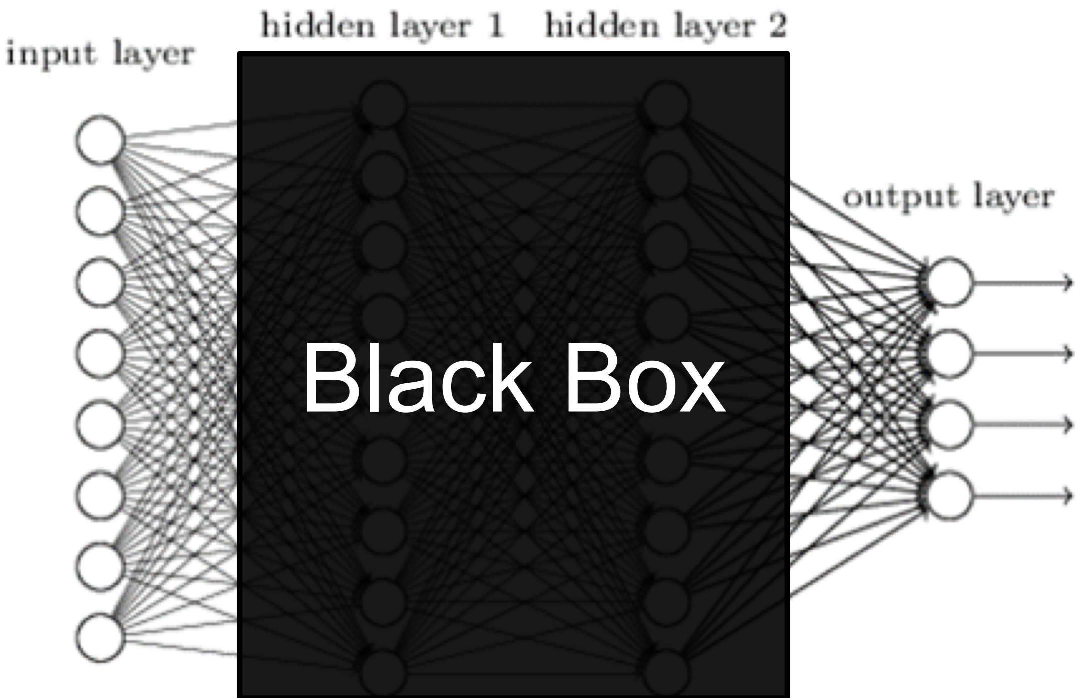
Deep Recurrent Neural Network





Feedforward Neural Network (Multilayer Perceptron)

$$\hat{y}_i = f\left(\sum_{j=1}^{M} W_{ij}\, x_j + b_i\right)$$



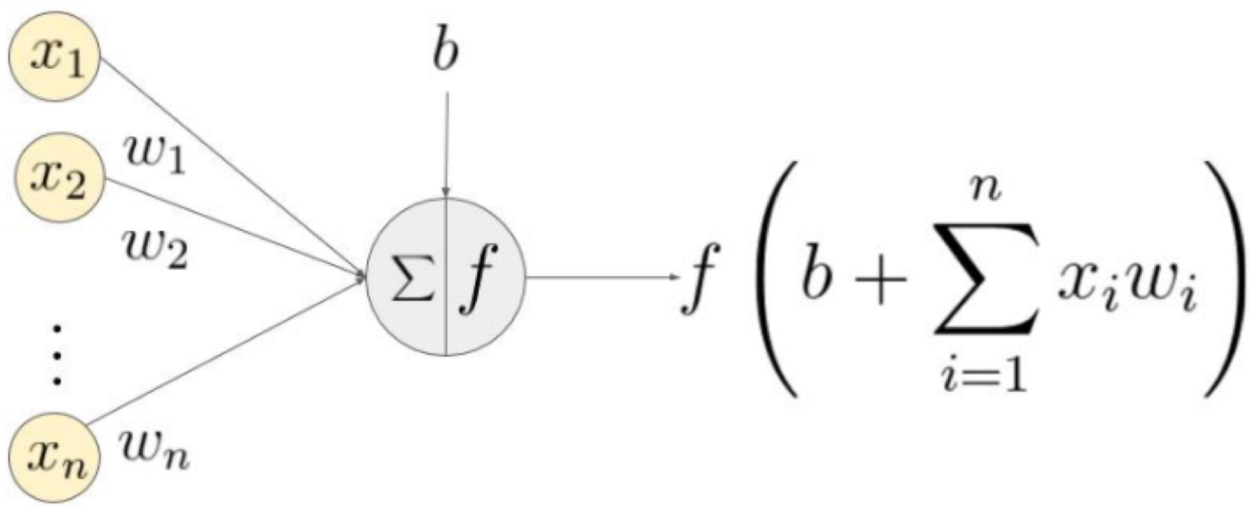
- The value of each node/neuron is computed from all the lines connected to it
- Each line has an associated *weight*
- Each node has an associated *bias*

Activation function

$$\hat{y}_i = f\left(\sum_{j=1}^{M} W_{ij}\, x_j + b_i\right)$$

- Introduces non-linearity into the network
  - Important for stacking layers

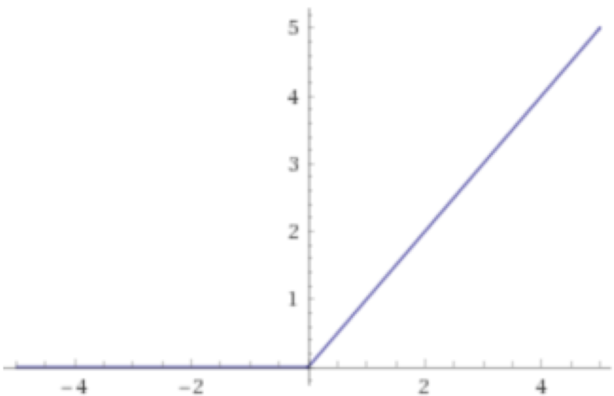
- Can keep output values bounded



Activation function

ReLU Function

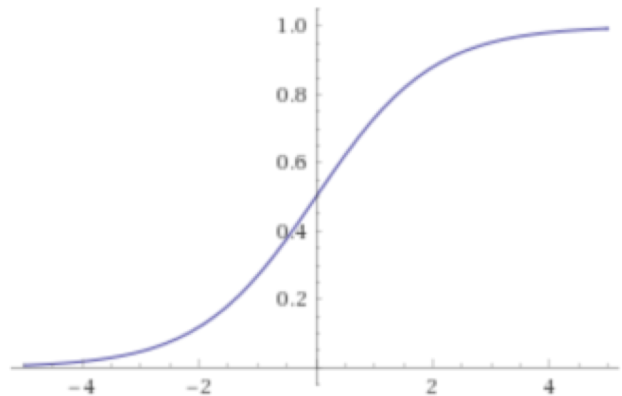
$$f(x) = \max(0, x)$$



Fast, minimal risk of *vanishing gradient problem*

Sigmoid function

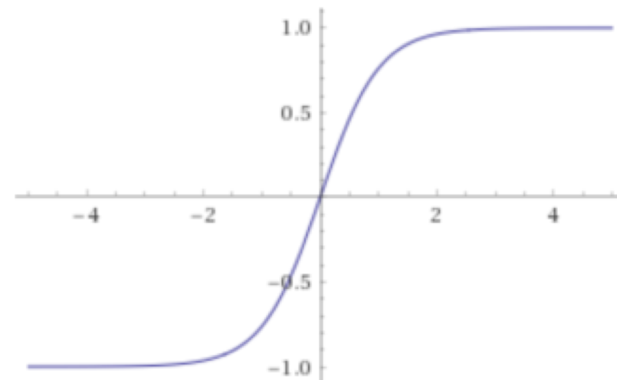
$$f(x) = \frac{1}{1 + e^{-x}}$$



Good for classifiers

Tanh function

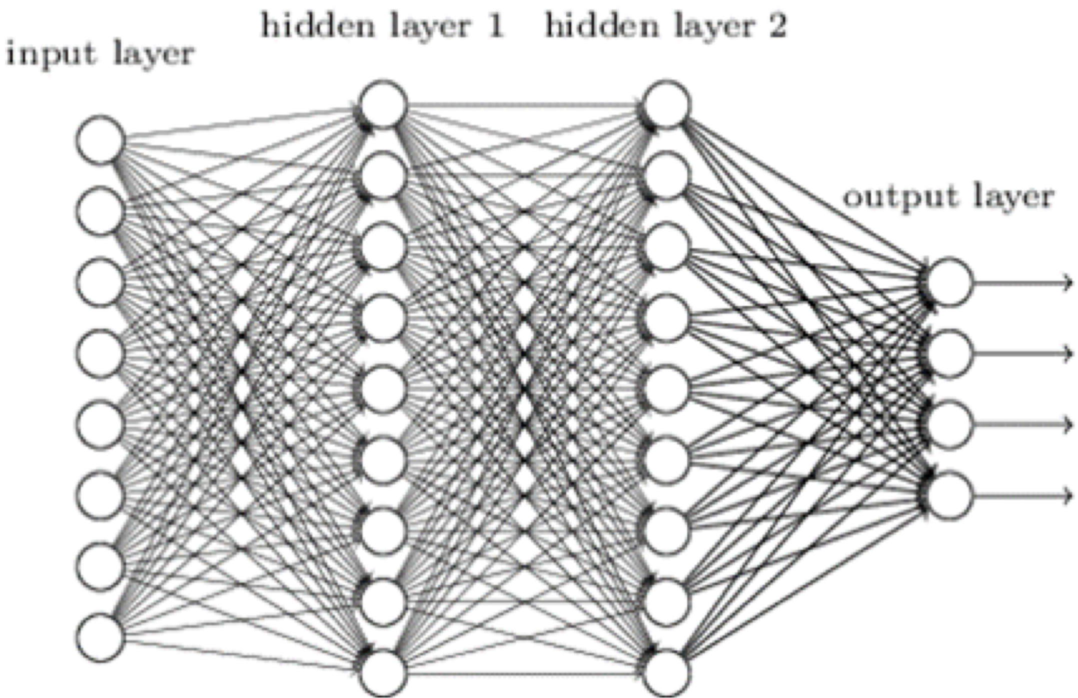
$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$



Steeper gradient strength than sigmoid

Feedforward Neural Network (Multilayer Perceptron)

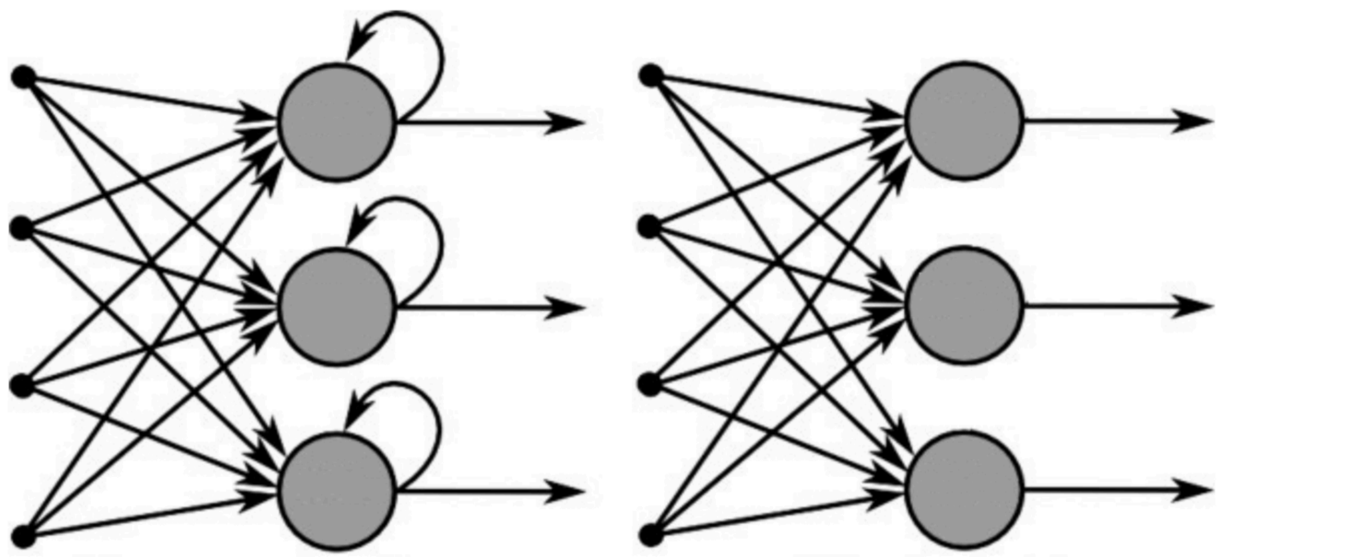
$$\hat{y}_i = f\left(\sum_{j=1}^{M} W_{ij}\, x_j + b_i\right)$$



- Feedforward NN only move in one direction
  - The information never touches a node twice
- Feed-Forward Neural Networks, have no memory of the input they received previously and are therefore bad in predicting what's coming next

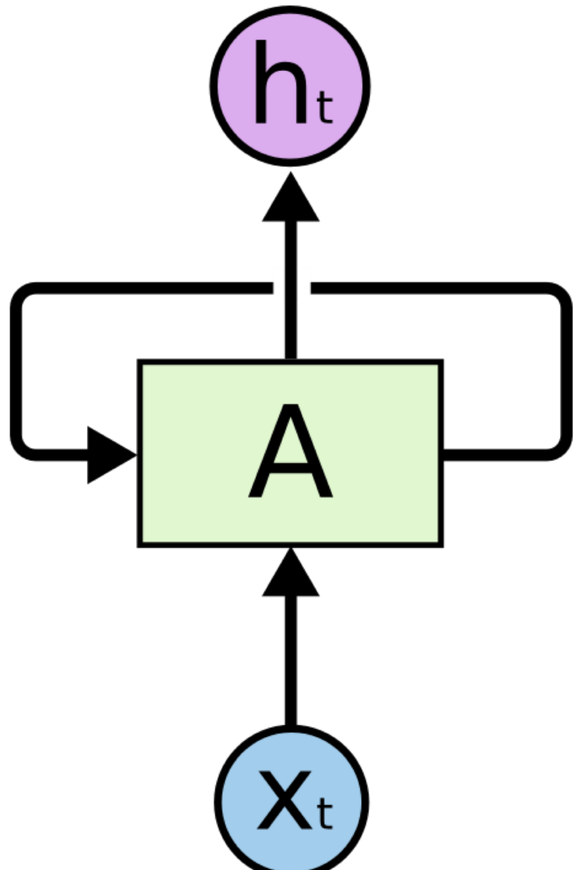
Recurrent Neural Network

- RNNs use *backpropagation through time* to update network weight parameters
- They are able to *remember* information in a sequence



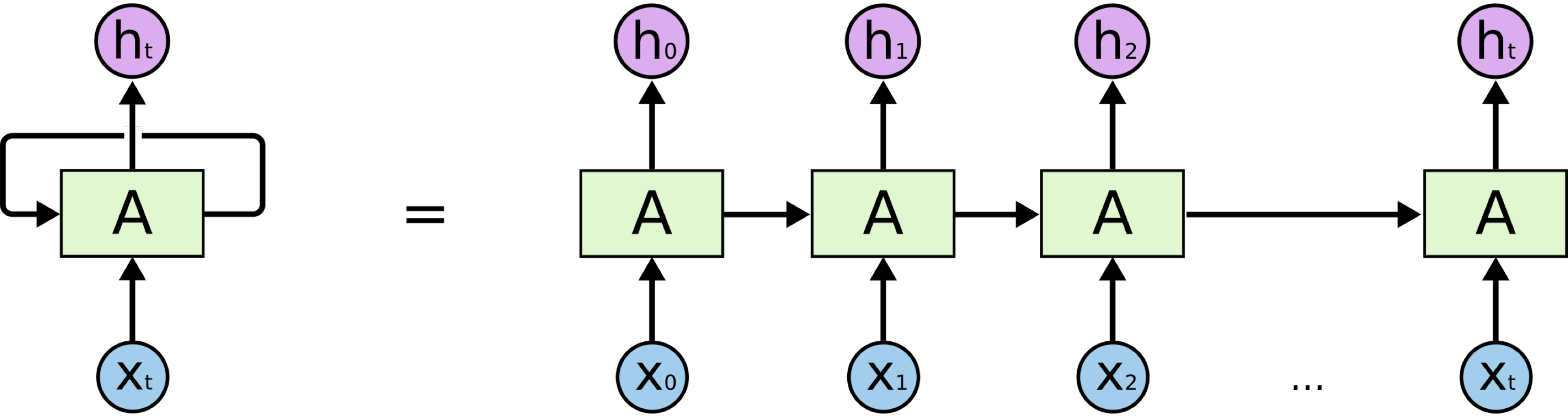
Recurrent Neural Network　　Feed-Forward Neural Network

Recurrent Neural Network



- Each node has two inputs
  1. Current timestep input
  2. Output of previous node
- Can retain a *memory* of previous time steps

Recurrent Neural Network



Recurrent Neural Network

Let a_t represent the ouput from the previous node

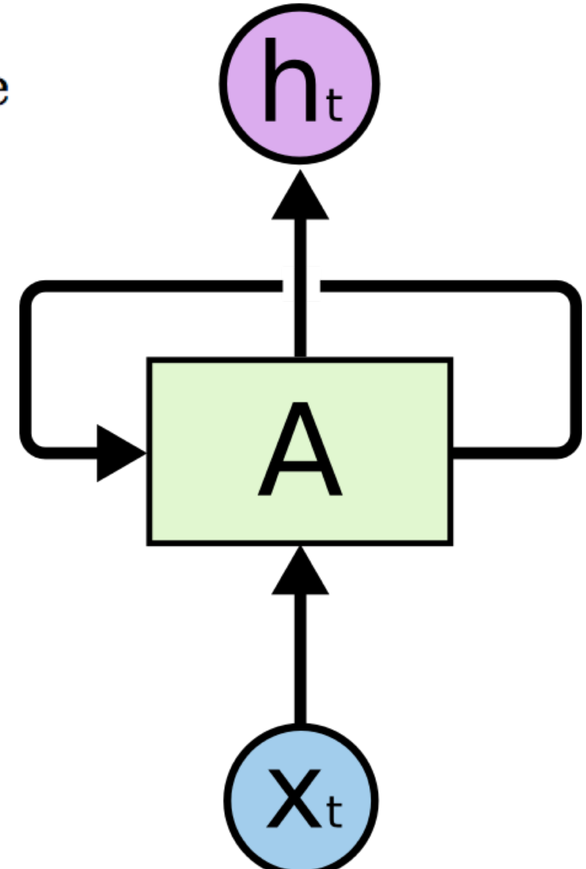
$$a_t = f(h_{t-1}, x_t)$$

$$g(x) = \tanh x$$

$$a_t = g(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

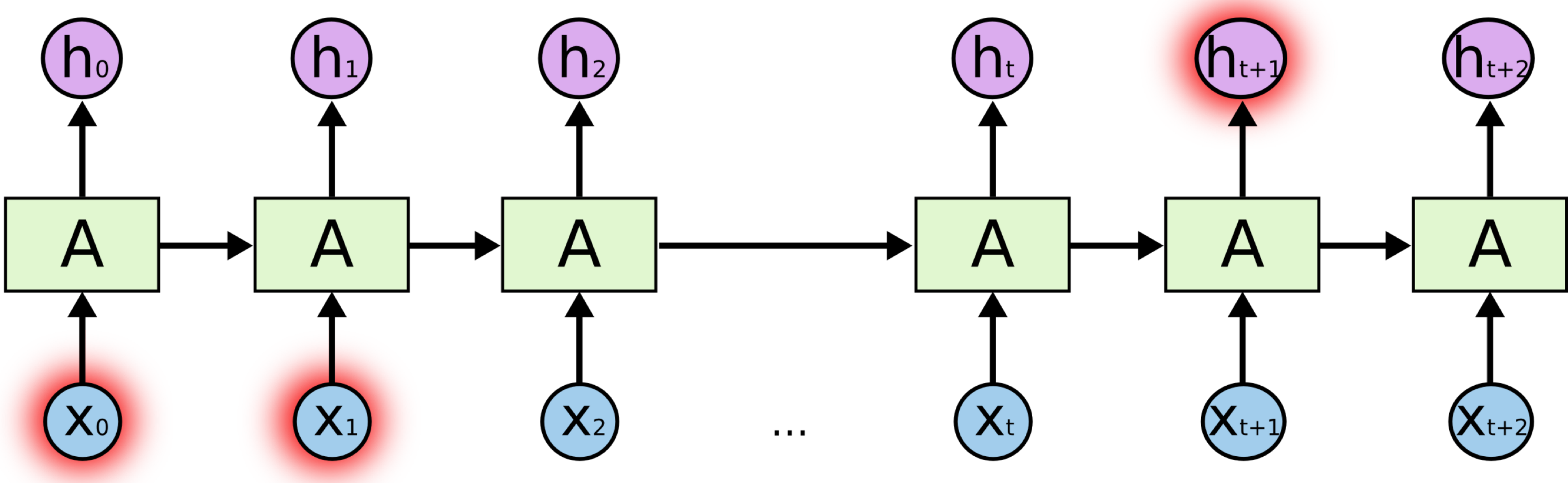
$$a_t = \tanh W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t$$

$$h_t = W_{hy} \cdot a_t$$



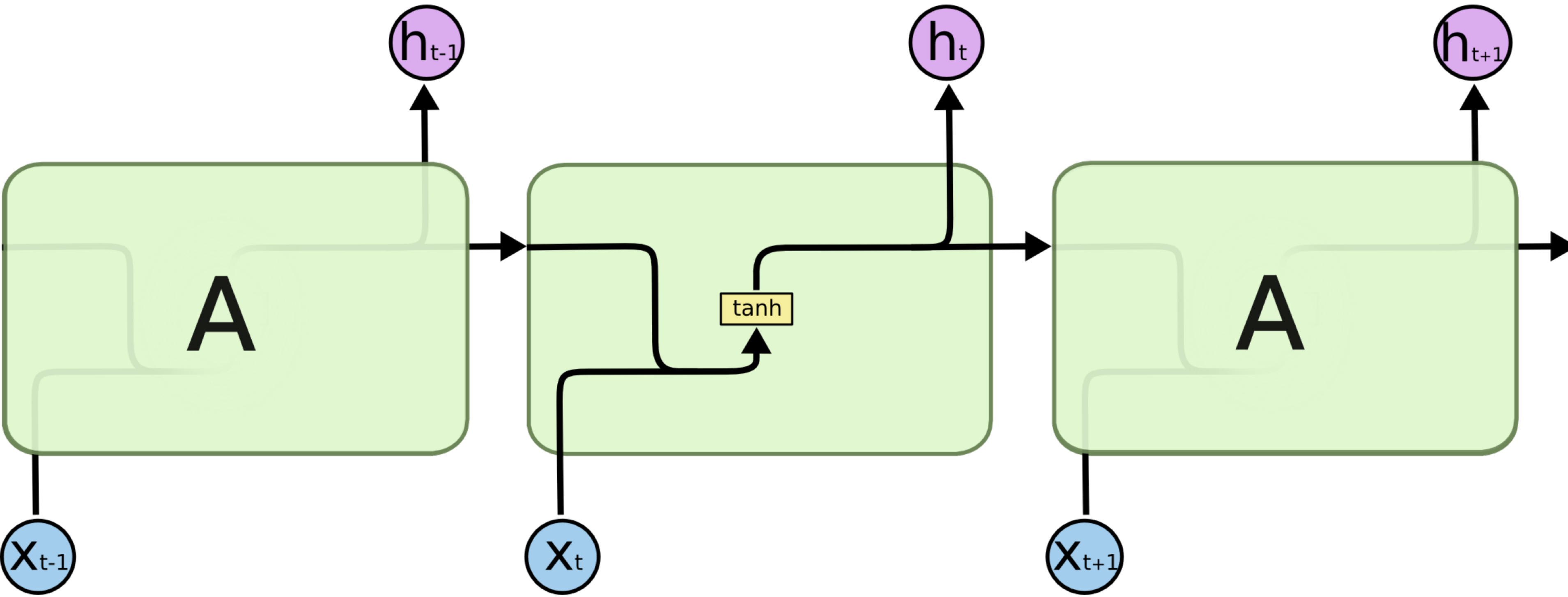
Recurrent Neural Network

› The disadvantage of a standard RNN is that as the time steps increase, it can't derive context from timesteps that are too far behind

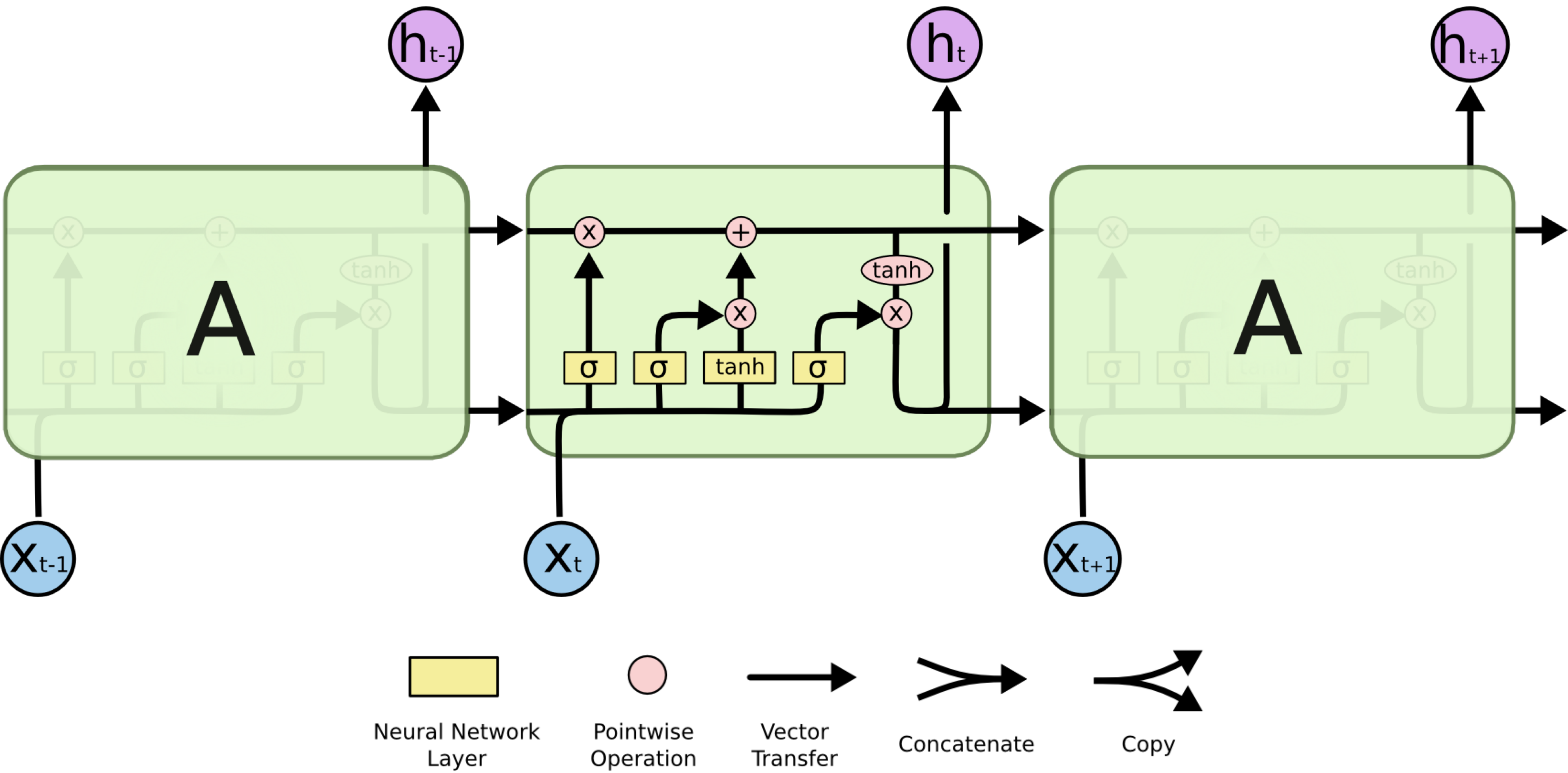


› Long Short Term Memory networks (LSTMs) were introduced to deal with this long-term dependency problem (Hocreiter & Schmidhuber, 1997)

Basic RNN



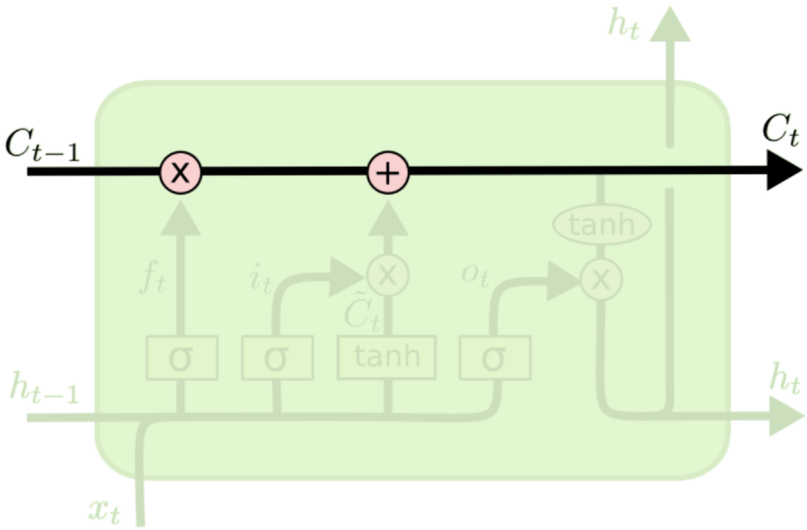
Long Short Term Memory Network (LSTM)



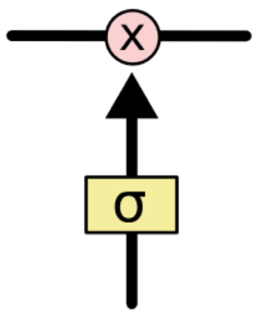
LSTM – The cell state

› The cell state passes between timesteps

- It can flow between nodes unchanged, or can be updated with *gates*



› Gates are composed of a sigmoid neural network layer and pointwise multiplication operation

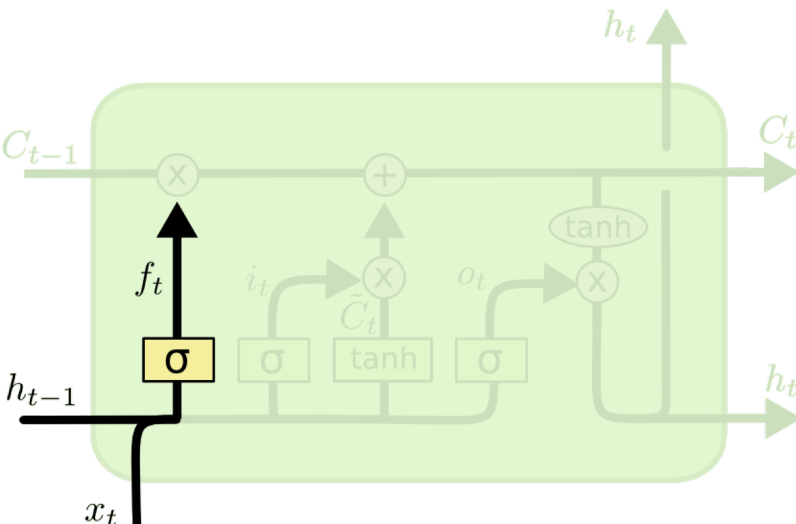


› The LSTM has three gates, to protect and control the cell state:

- 1. Forget gate, 2. Update gate, 3.Output gate

LSTM – Forget gate

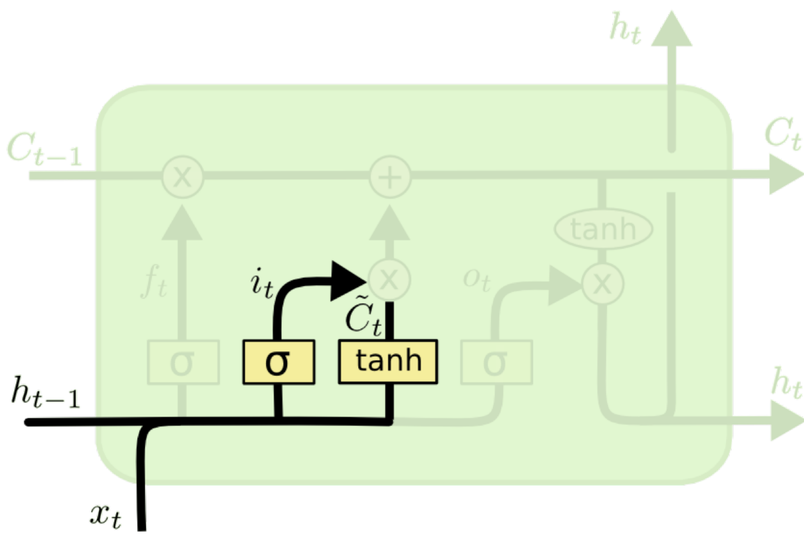
- We take the input from current time step and the learned representation from previous time step and concatenate them
- The sigmoid function outputs a value between 0 and 1, we use this value to determine how much of previous cell state to *remember*



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

LSTM – Update gate

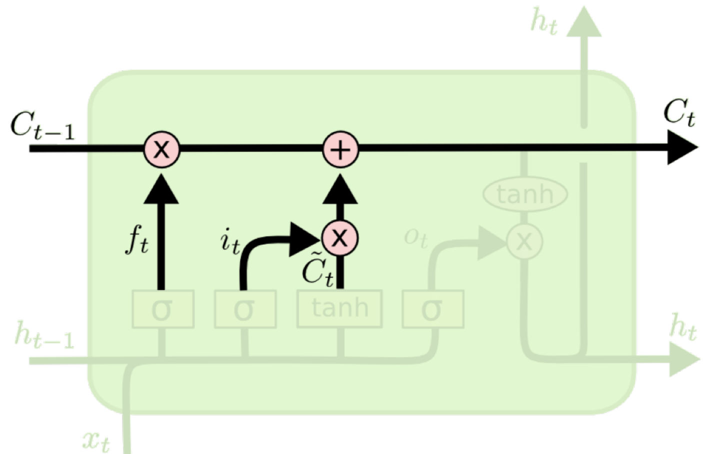
- First, a sigmoid layer called decides which values we'll update.
- Next, a $\tanh$ layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state



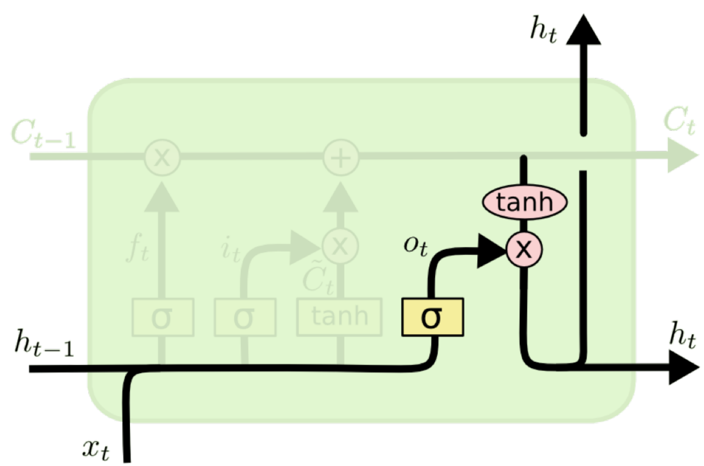
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

LSTM – Output gate



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

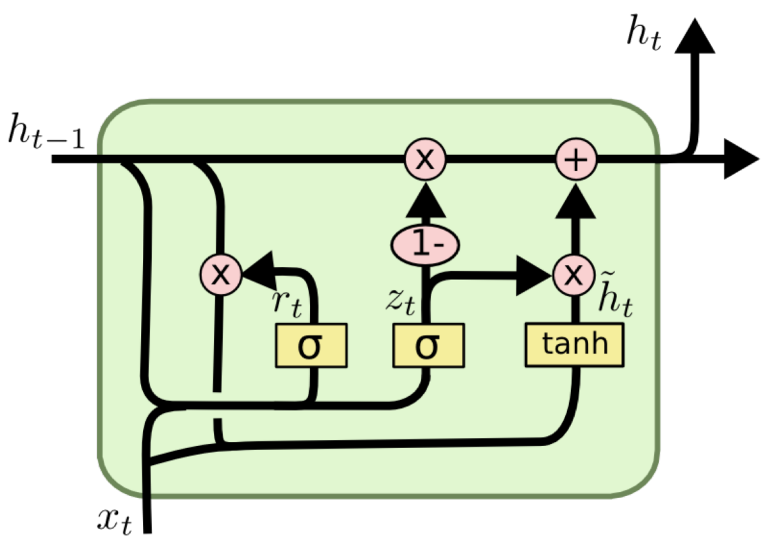


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Gated Recurrent Unit (GRU) Network

- LSTMs can be computationally expensive. GRUs (Cho, et al., 2014) are similar, but reduce the training time
- Performance is similar



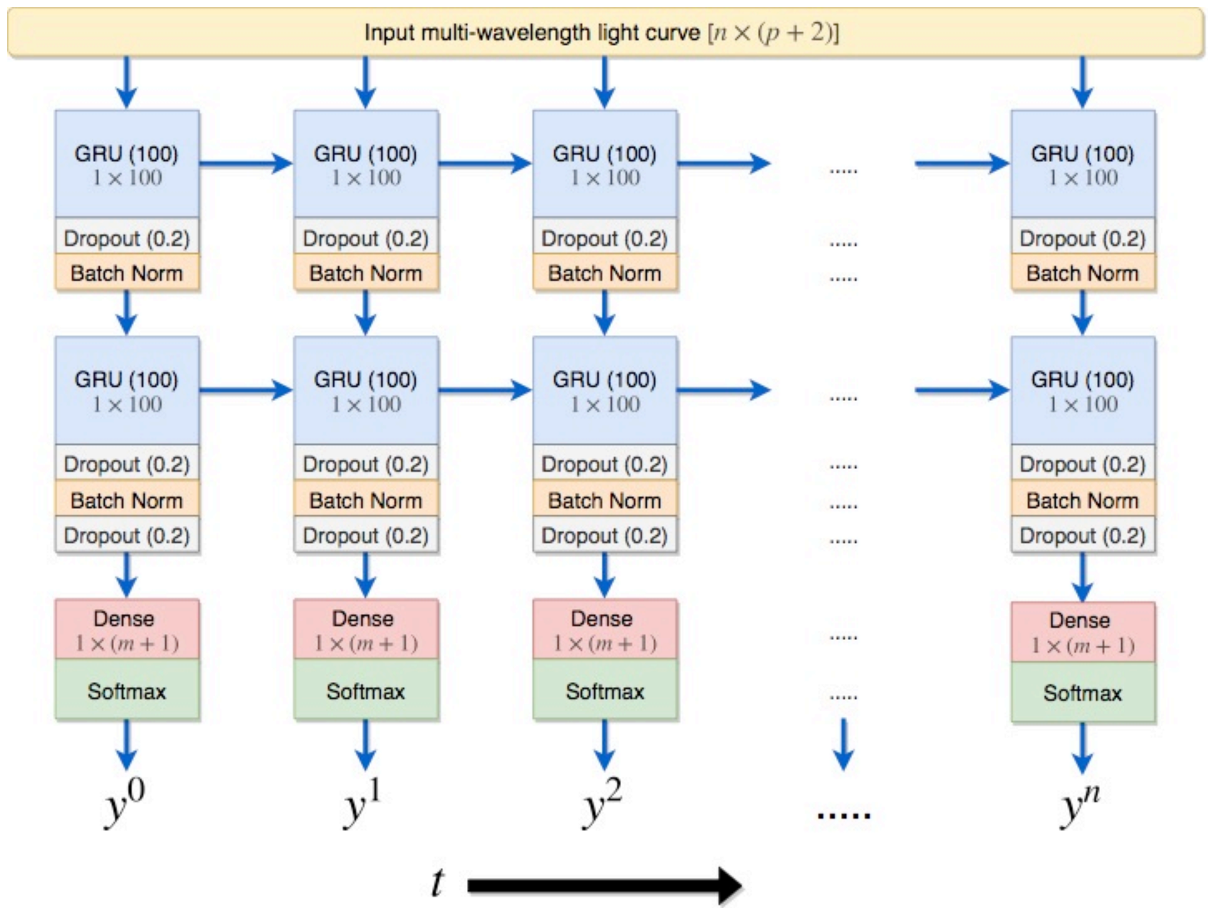
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

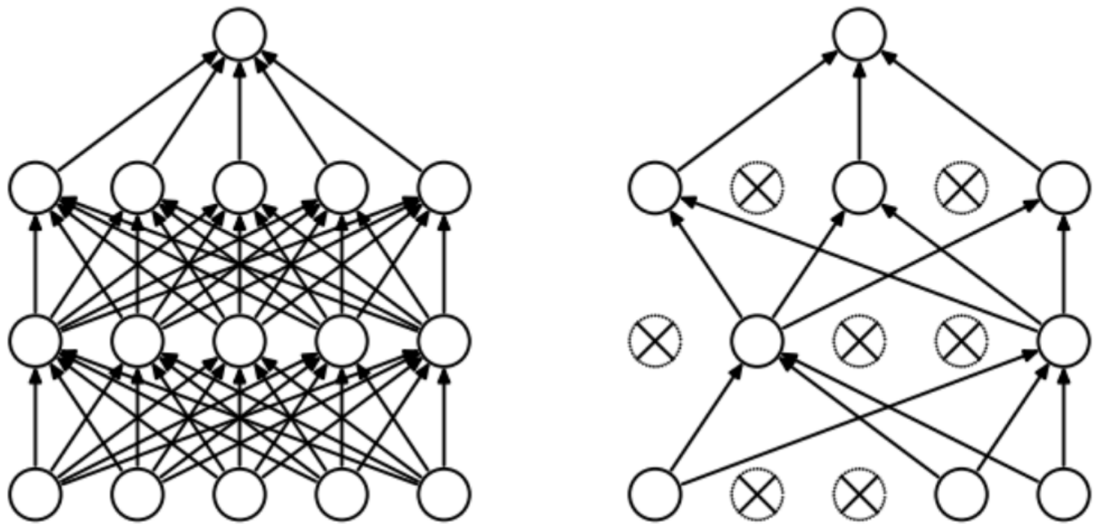
RAPID



Input multi-wavelength light curve $[n \times (p+2)]$
GRU (100)
1×100
Dropout (0.2)
Batch Norm
GRU (100)
1×100
Dropout (0.2)
Batch Norm
Dropout (0.2)
Dense
$1 \times (m+1)$
Softmax
y^0
y^1
y^2
.....
y^n
t

Dropout Regularisation

- Reduces overfitting
- It forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons
- It roughly doubles the number of iterations required to converge

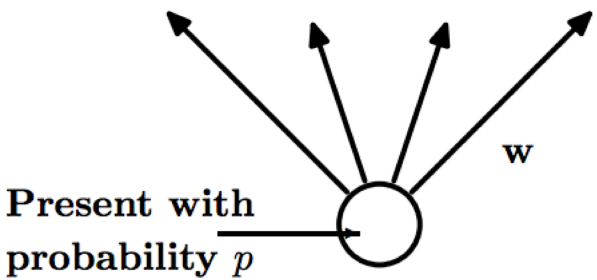


(a) Standard Neural Net (b) After applying dropout.

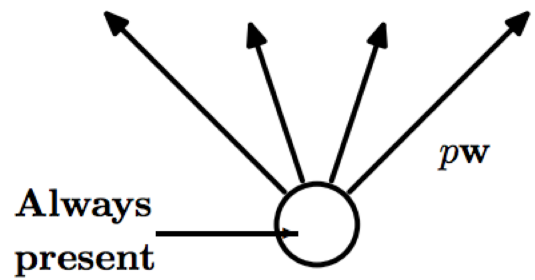
Srivastava, Nitish, et al

Dropout Regularisation

- Reduces overfitting
- It forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons
- It roughly doubles the number of iterations required to converge



(a) At training time



(b) At test time

Srivastava, Nitish, et al

Batch Normalisation

- Normalises the parameters in the network
- Improves learning speed
- Also has slight regularization effect by introducing noise to each hidden layer's activations
- Adds two trainable parameters to each layer

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

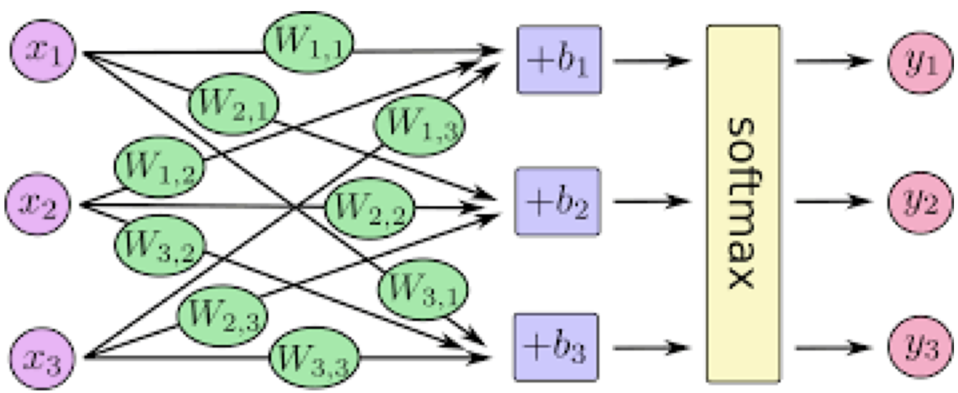
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

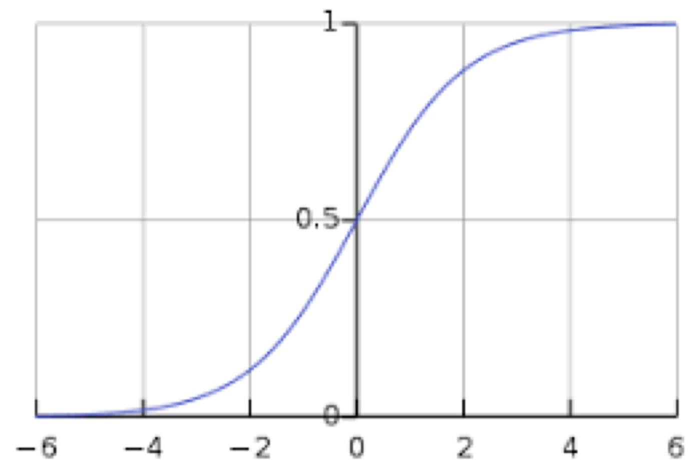
Ioffe & Szegedy 2015

Softmax Regression

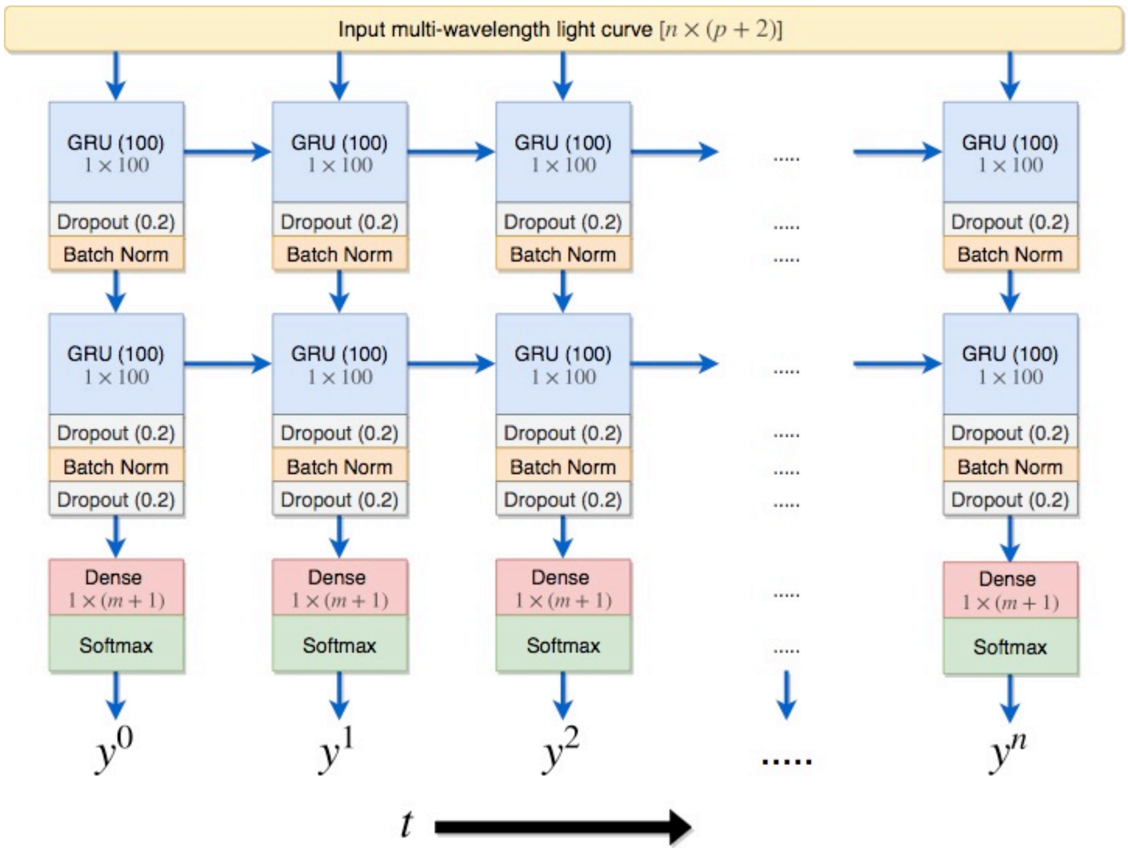
$$\boldsymbol{y} = \mathrm{softmax}(\hat{\boldsymbol{y}})$$

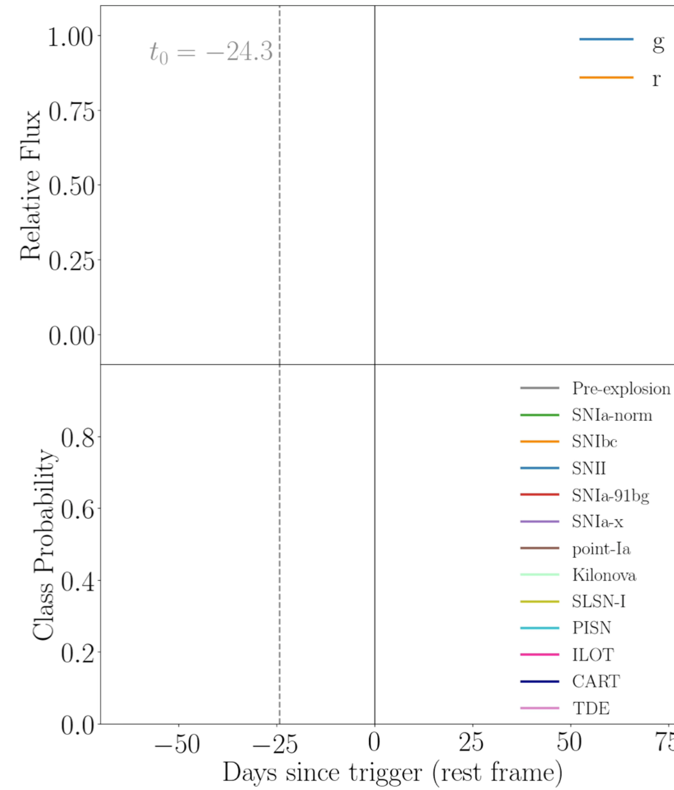
$$\mathrm{softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$



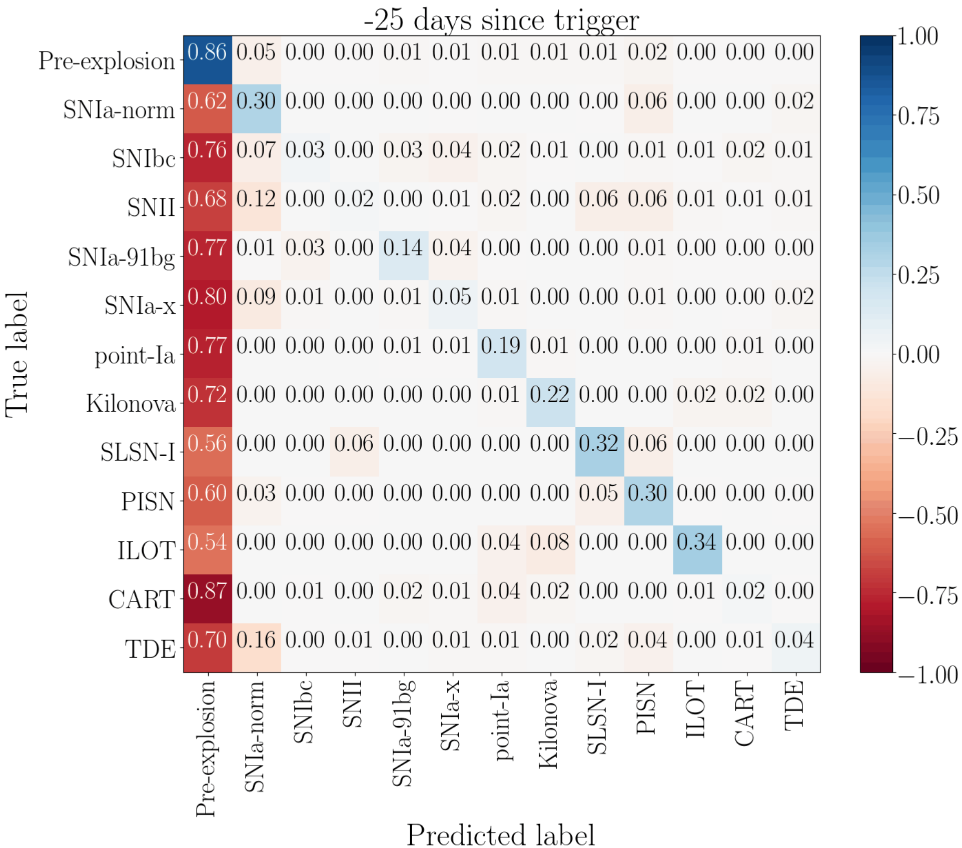


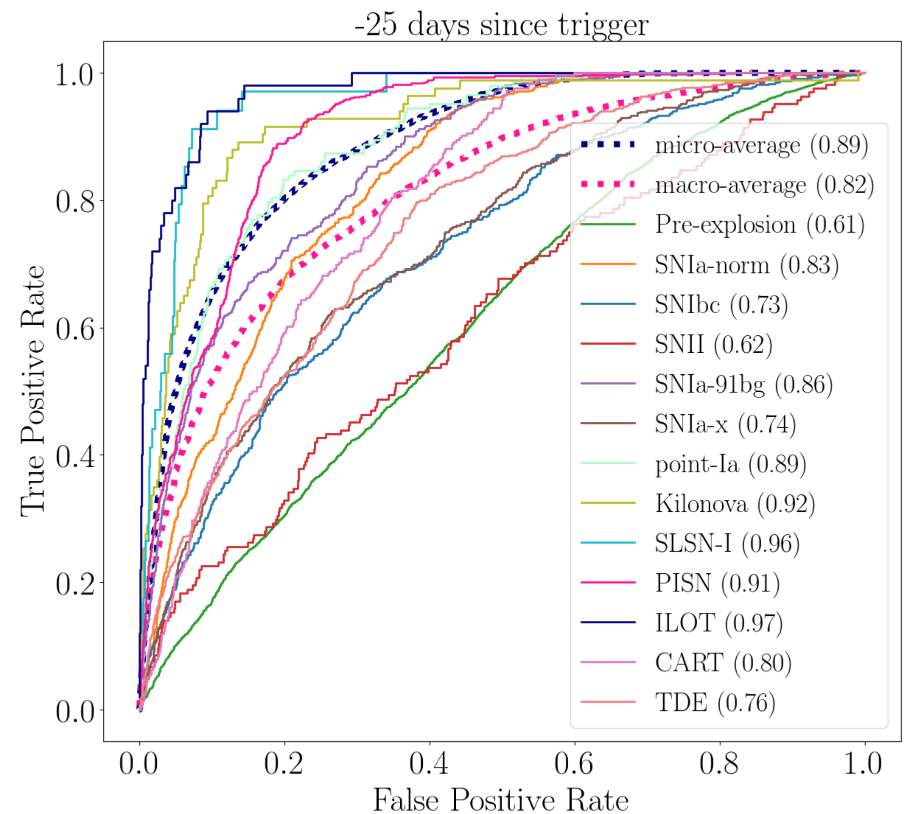
Deep Recurrent Neural Network



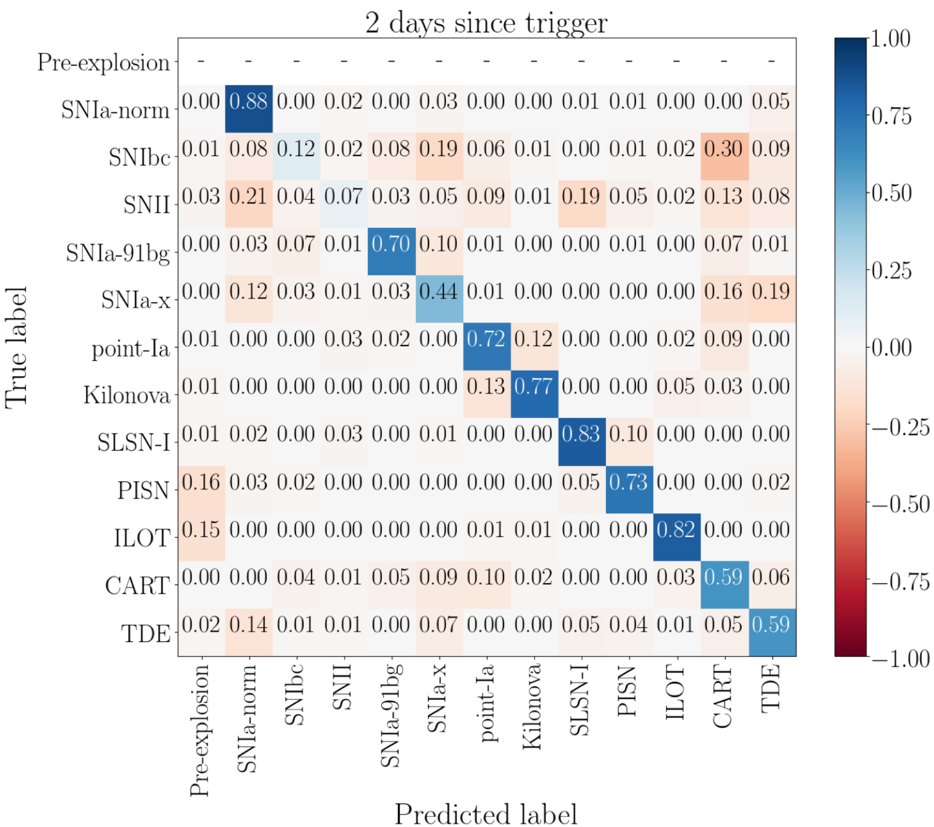


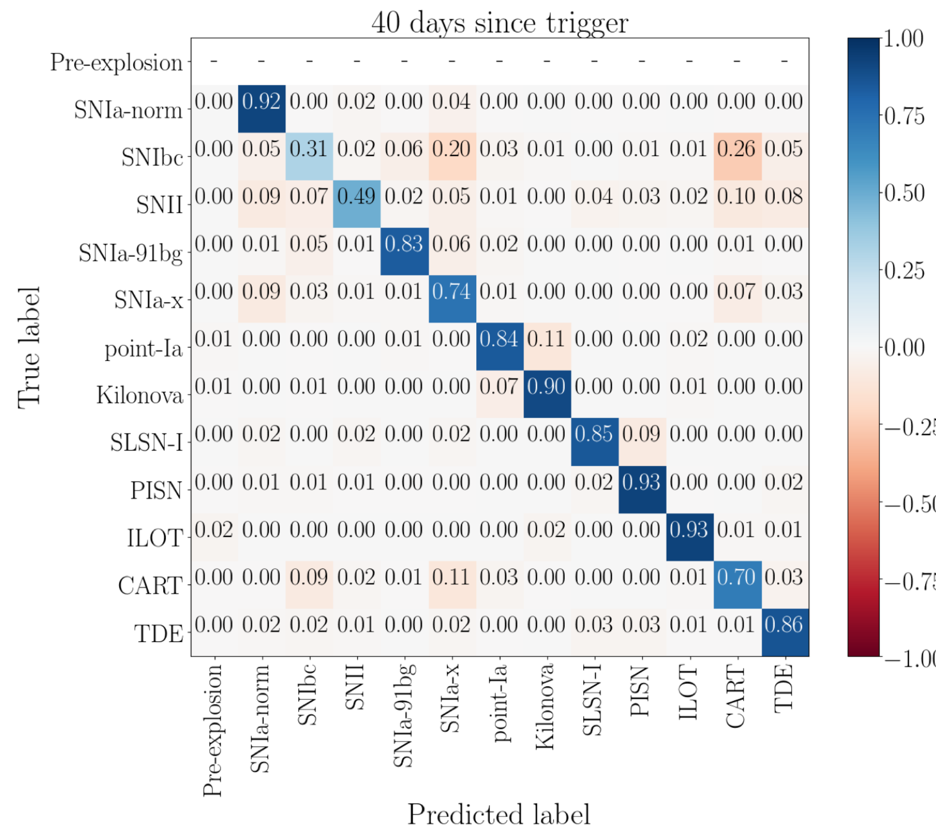
Classification performance



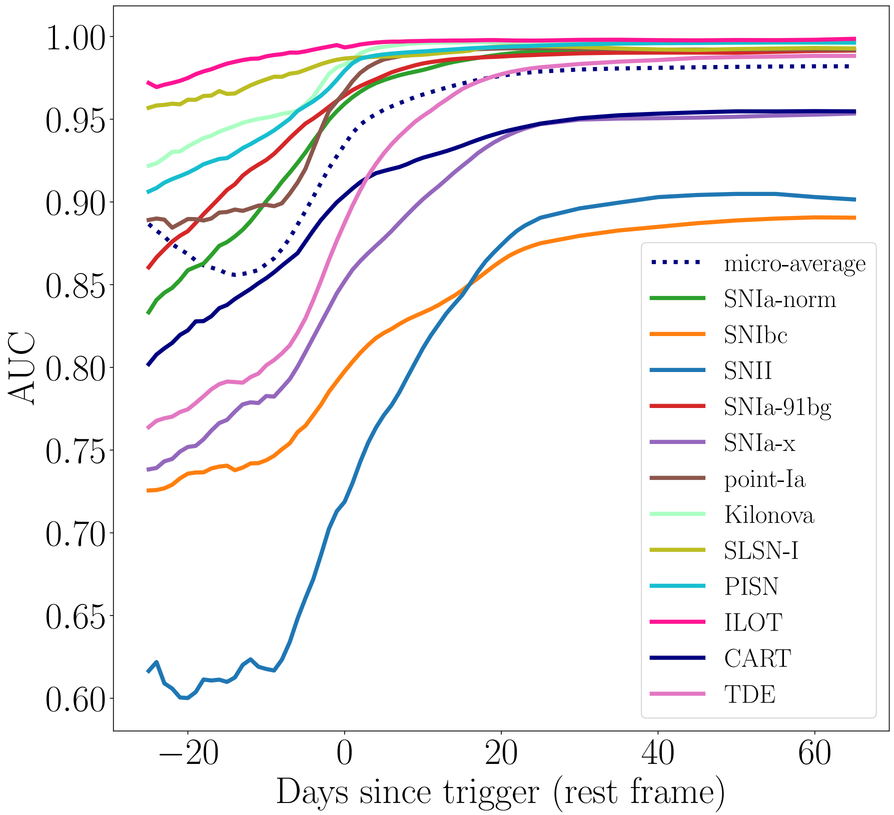


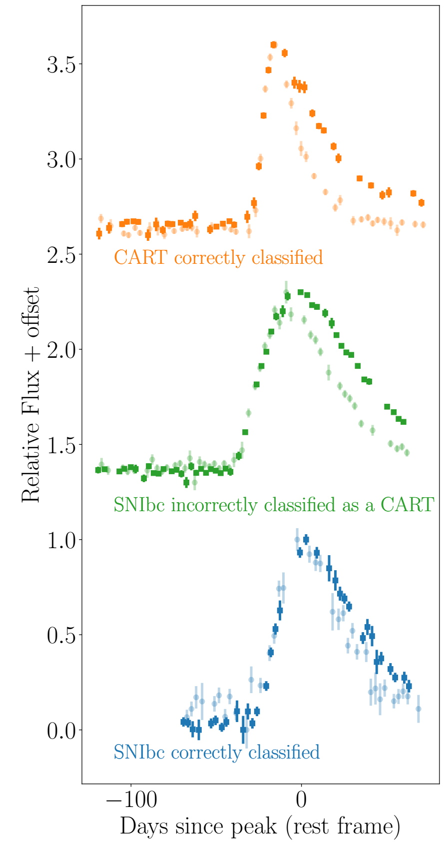
Confusion matrices





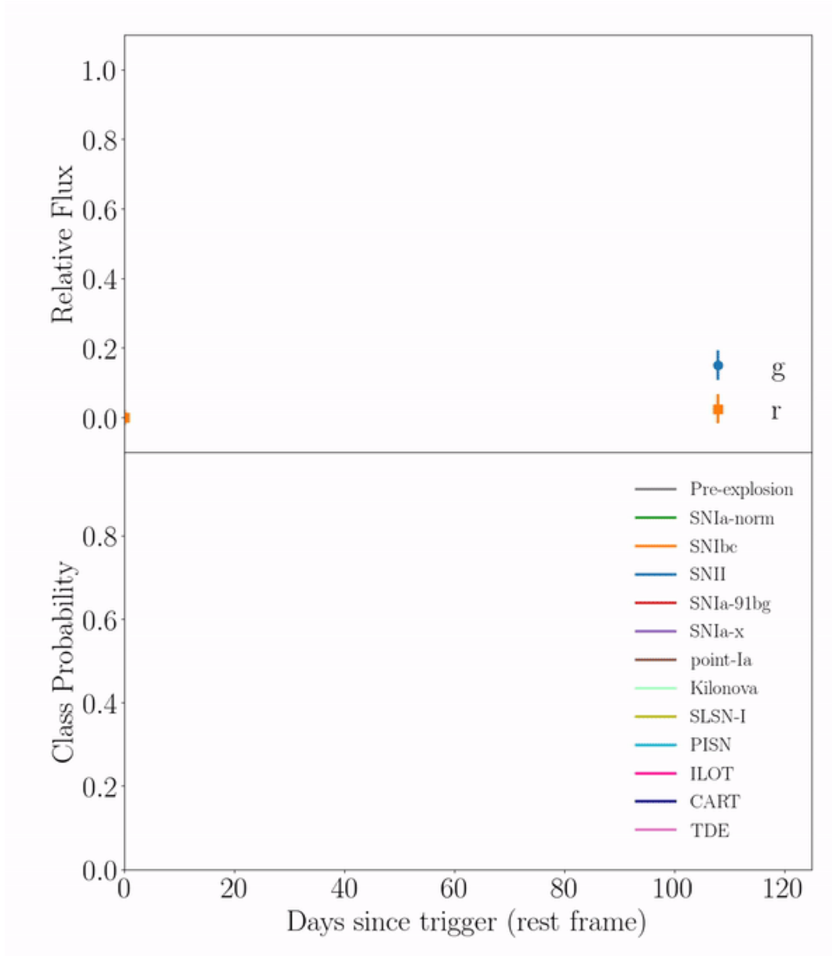
Classification Performance





Need better colour information?

Applied to real data



Python interface

- `pip install astrorapid`
- https://astrorapid.readthedocs.io

astrorapid

latest

Search docs

CONTENTS:

Installation

⊟ Usage

Classify Light curves

Train your own classifier with your own data

Example

Docs » Usage

Edit on GitHub

Usage

Classify Light curves

Use the following example code:

```
from astrorapid.classify import Classify

# Each light curve should be a tuple in this form. Look at the example code for an example of i
light_curve_info1 = (mjd, flux, fluxerr, passband, zeropoint, photflag, ra, dec, objid, redshi
light_curve_list = [light_curve_info1,]

# Classify Light curves
classification = Classify(light_curve_list, known_redshift=True)
predictions = classification.get_predictions()
print(predictions)

# Plot light curve and classification vs time of the light curves at the specified indexes
classification.plot_light_curves_and_classifications(indexes_to_plot=(0,1,4,6))
classification.plot_classification_animation(indexes_to_plot=(0,1,4,6))
```

Train your own classifier with your own data

Conclusions

- RAPID enables **prioritized follow-up** of new large-scale transient surveys based on transient class and epoch

- **Early classification:** The use of a Recurrent Neural Network allows us to classify transients as a function of time

- We can identify **12 different transient classes** within days of its explosion, despite low S/N data and limited colour information

- **It's fast:** Can classify tens of thousands of events that will be discovered in LSST and ZTF within a few seconds