Quick aside: Convolutional Layers and Backpropagation

- You can think of a Convolutional layer as taking every pixel that a single ‘pixel’ of the kernel processes and running it through the same linear activation neuron with the same weight

Machine Learning Intro 5:

An Introduction to transfer learning, and also Recurrent Neural Networks

Machine Learning Intro 5a:

Transfer Learning

Transfer Learning

Transfer Learning is the Process of using all or parts of a pre-trained Neural Network to solve a different problem than the original purpose.

Links:

- Transfer learning blogpost by Sebastian Ruder
- Transfer Learning Tutorial on HackerEarth
- Paper on Transferable Features

Transfer Learning

Examples:

- Retrain a CNN model trained for image classification to recognize other classes
- Retrain a model trained to detect signal anomalies to detect anomalies in another source
- Train a Dense model to accept new inputs or classify new classes

Rules of thumb in Transfer Learning When:

New Data: The data you want to apply your model to
Old Data: The data the pre-trained model was trained on

1. New data is smaller and similar to old Data
2. New data is large and similar to old Data
3. New data is small and different from old Data
4. New data is large and different from old Data

Rules of thumb in Transfer Learning (Example of a Conv-Net)

1. New dataset is smaller and similar to old Data

   Since the data is similar, we can use *all* the Conv. Layers. However, we do *not* fine-tune the model on the new data, since that can cause *Overfitting.*

   So:

- *Remove the fully-connected layers at the end.*
- *Add new FC layers*
- *Initialize them at ϵ*
- *Train the new FC layers*

Rules of thumb in Transfer Learning (Example of a Conv-Net)

2. New dataset is large and similar to old Data

   Since the data is similar, we can use *all* the Conv. Layers. We can also fine-tune the model, since the large amount of data will help prevent overfitting.

   So:

- *Remove the fully-connected layers at the end.*
- *Add new FC layers*
- *Initialize them at ϵ*
- *Train the entire model*

Rules of thumb in Transfer Learning (Example of a Conv-Net)

3. New dataset is smaller and different to old Data

   Since the data is similar, we can use *all* the Conv. Layers. However, we do *not* fine-tune the model on the new data, since that can cause *Overfitting.*

   So:

- *Remove the fully-connected layers at the end, as well as the higher-level Conv. layers.*
- *Add new FC layers (and Conv. layers)*
- *Initialize them at ϵ*
- *Train the FC layers model*

Rules of thumb in Transfer Learning (Example of a Conv-Net)

4. New dataset is large and different to old Data

   You can train from scratch if you want. However you can use the old Conv layers as a starting point.

   So:

- *Remove the fully-connected layers at the end*
- *Add new FC layers*
- *Initialize them at ϵ*
- *Train the entire model*

Differences between Datasets (minus the math)

Original problem: the problem that the NN has been trained to solve
New problem: the problem it or parts of it are being re-purposed to solve

1. The original data and the new data are different, e.g. different sets of images to be classified or different languages to be translated
2. The Distributions of data between new and old are different. e.g. texts which discuss different topics (different images for same classes?)
3. Different labels (outputs) between new and old. Usually occurs in conjunction with (4).
4. Different representation of classes in new and old Data. i.e. some classes are more represented in the old data and some are more represented in the new.

Differences between Datasets (minus the math)

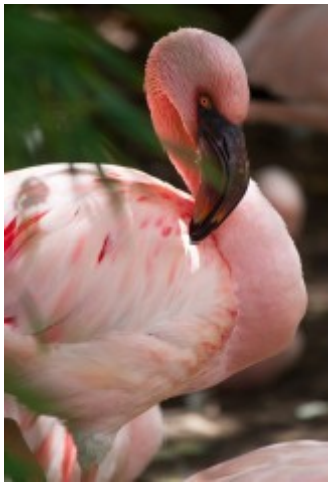
Original problem: the problem that the NN has been trained to solve
New problem: the problem it or parts of it are being re-purposed to solve

1. The original data and the new data are different, e.g. different sets of images to be classified or different languages to be translated

Differences between Datasets (minus the math)

Original problem: the problem that the NN has been trained to solve
New problem: the problem it or parts of it are being re-purposed to solve

2. The Distributions of data between new and old are different. e.g. texts which discuss different topics (different images for same classes?)

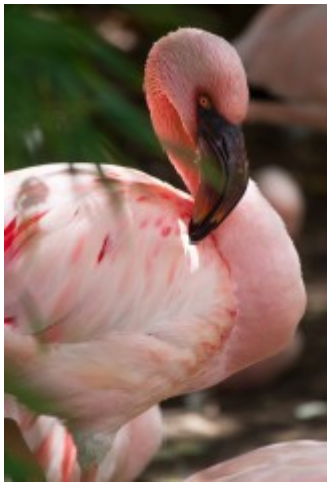



Differences between Datasets (minus the math)

Original problem: the problem that the NN has been trained to solve
New problem: the problem it or parts of it are being re-purposed to solve

3. Different labels (outputs) between new and old. Usually occurs in conjunction with (4).

‘Parrot’



‘Flamingo’

‘Bird’

‘Car’

Differences between Datasets (minus the math)

Original problem: the problem that the NN has been trained to solve
New problem: the problem it or parts of it are being re-purposed to solve

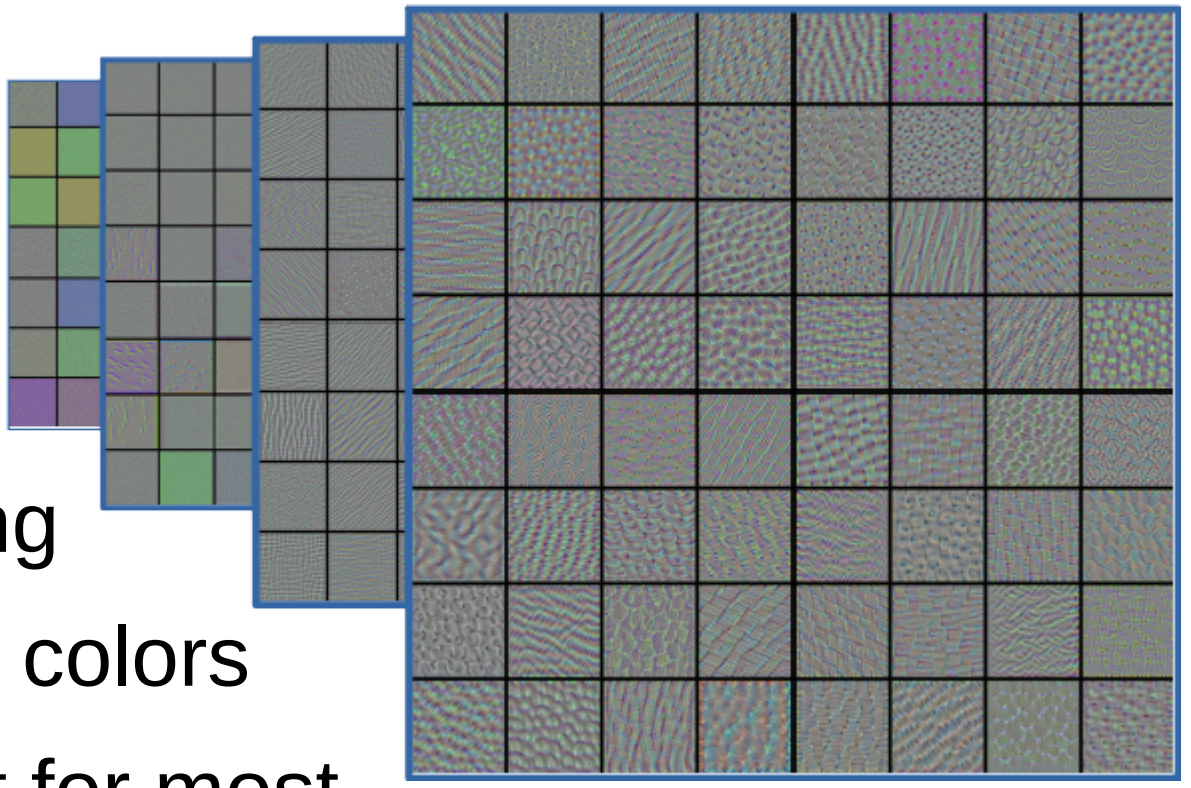
4.Different representation of classes in new and old Data. i.e. some classes are more represented in the old data and some are more represented in the new. Very common.

66% Parrot

66% Flamingo

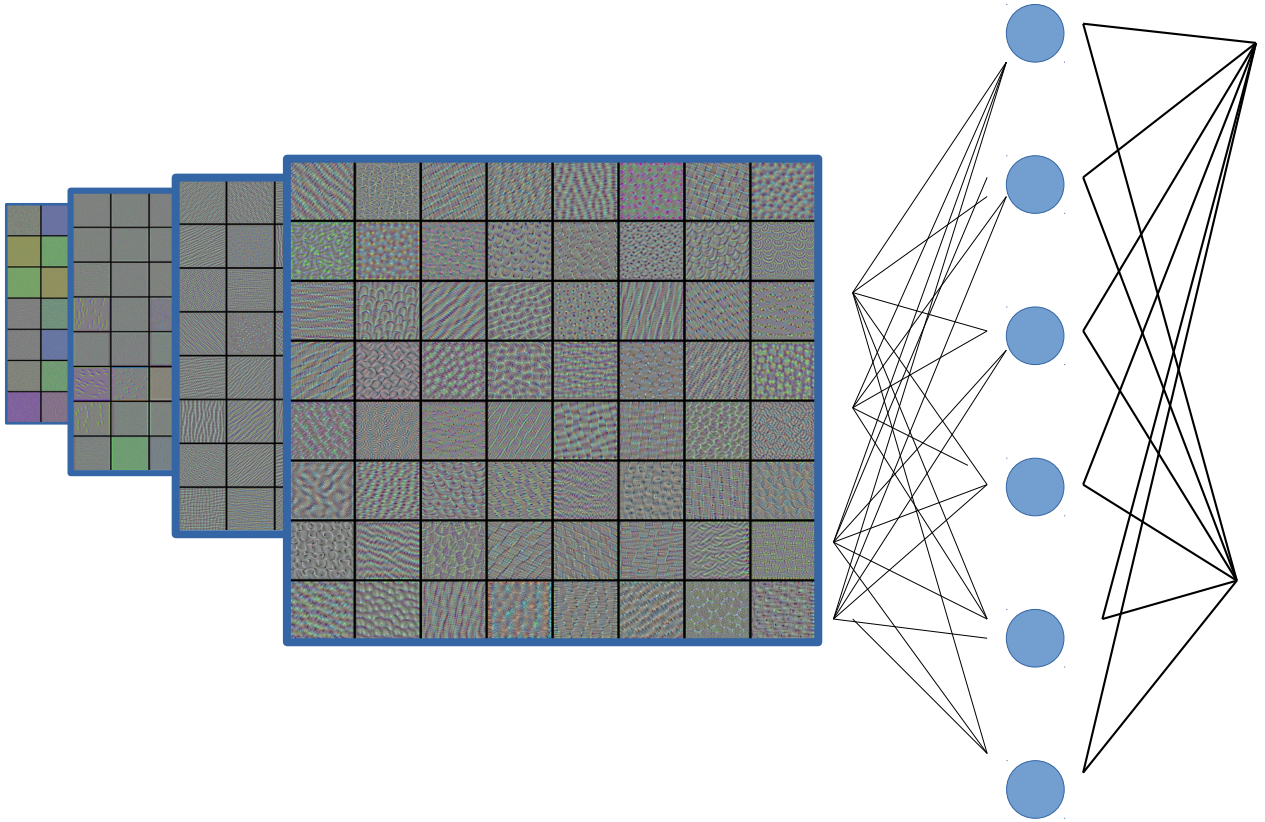
Transfer Learning Methods

- CNN features



All the earlier layers are detecting rough shapes and colors which are relevant for most image classification tasks. These general patterns can be reused.

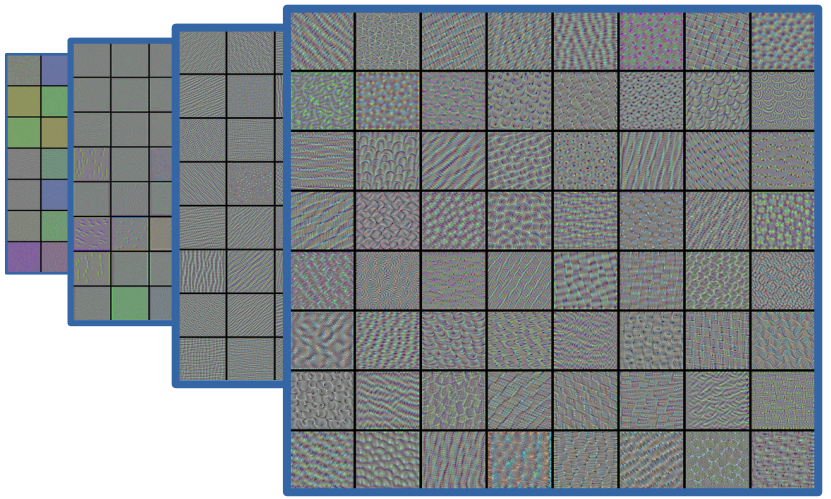
Transfer Learning Methods



Male Bullock's Oriole; by Kevin Cole, WikiMedia.Org, Creative Commons 2.0

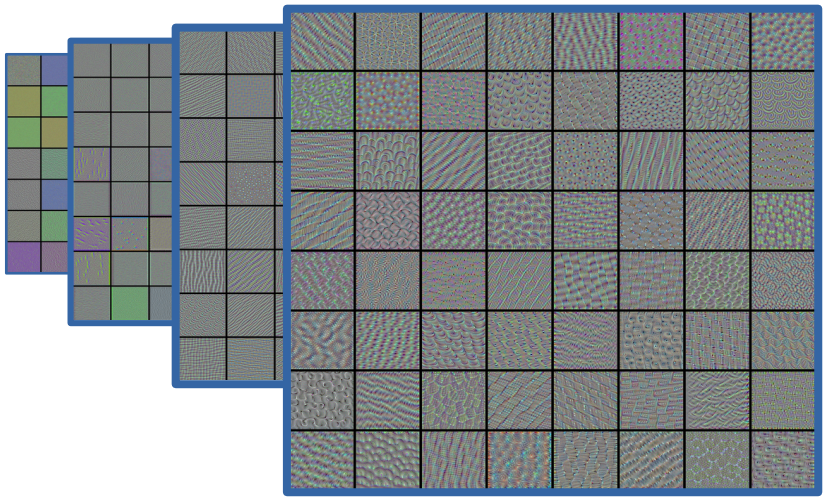
Department of Biology
Norwegian University of Science
and Technology (NTNU)
Trondheim, Norway

Transfer Learning Methods

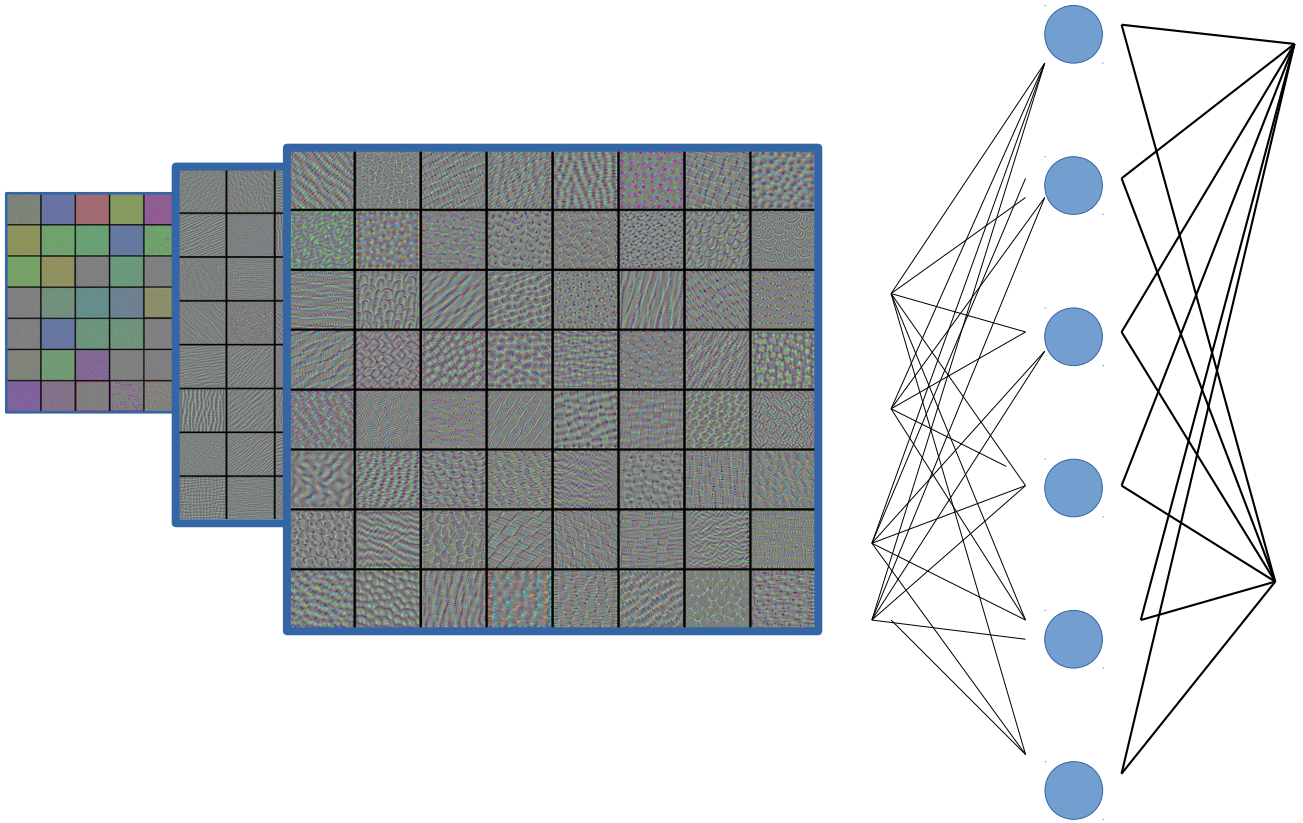


Remove the later layers and output

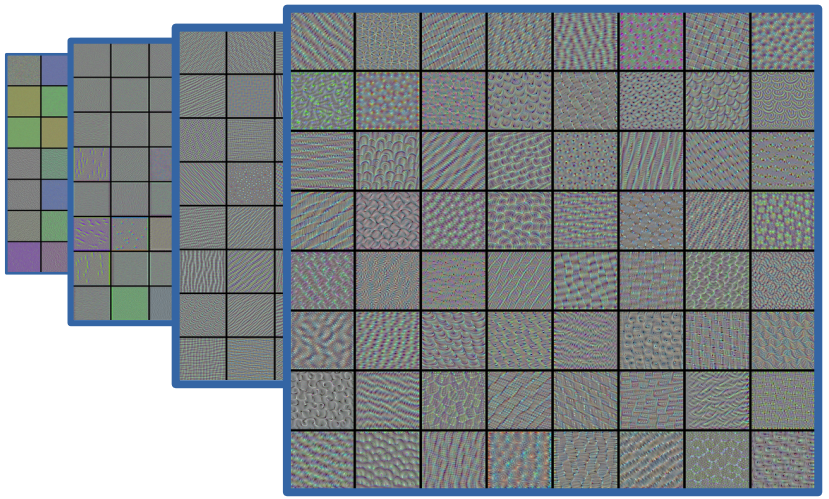
Transfer Learning Methods

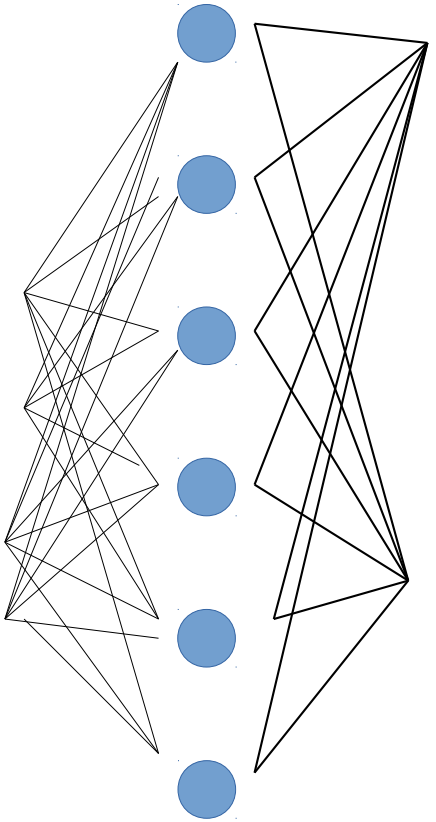


Transfer Learning Methods



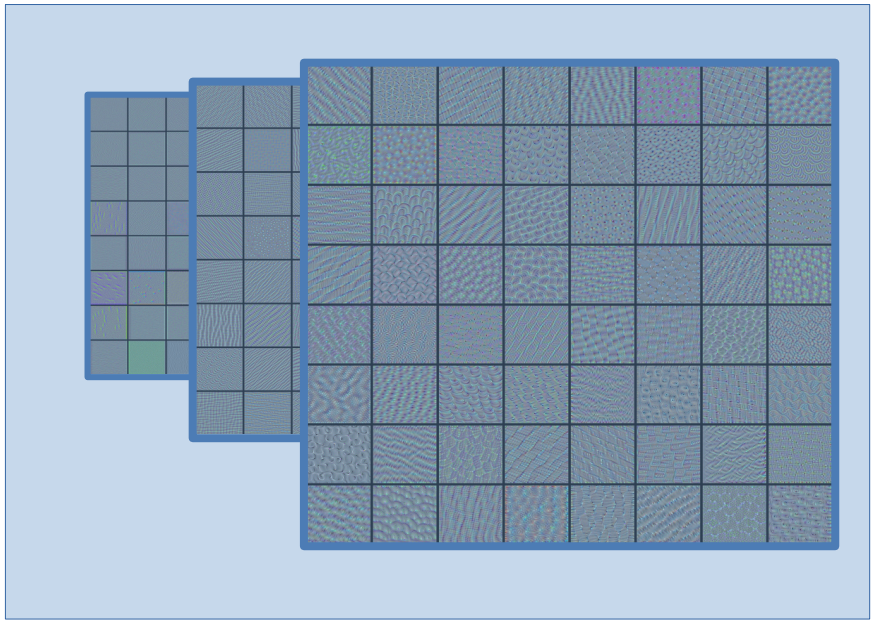
Transfer Learning Methods

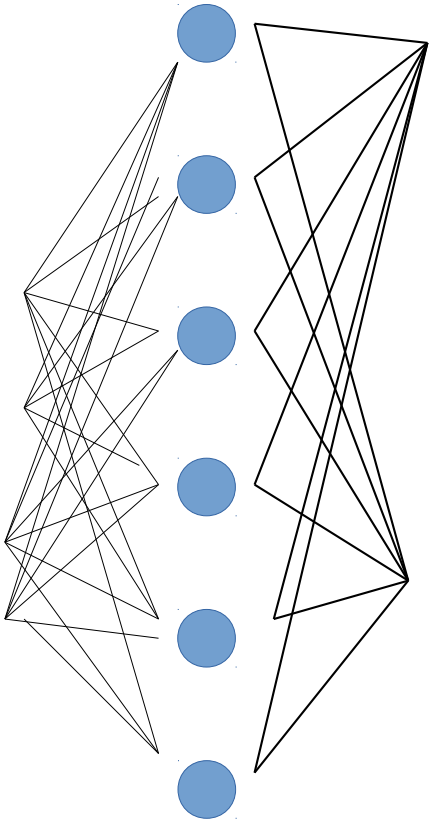




Transfer Learning Methods

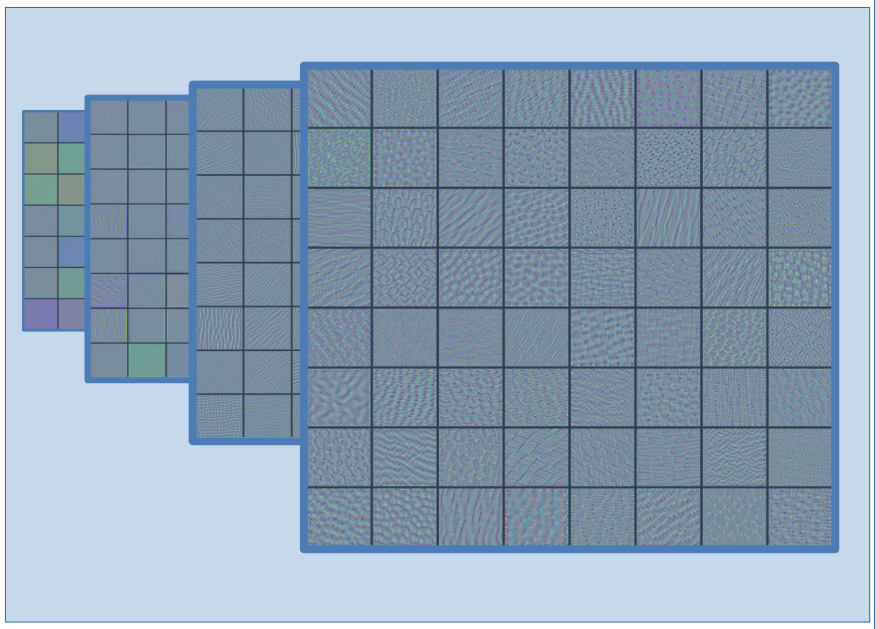
Train the first part lightly



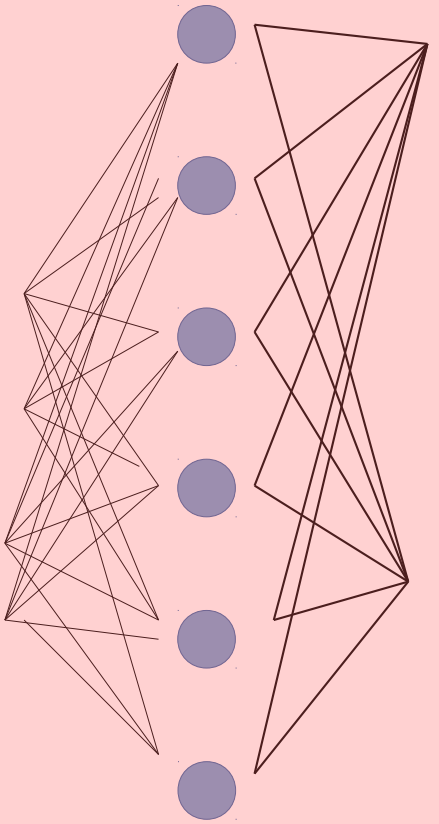


Transfer Learning Methods

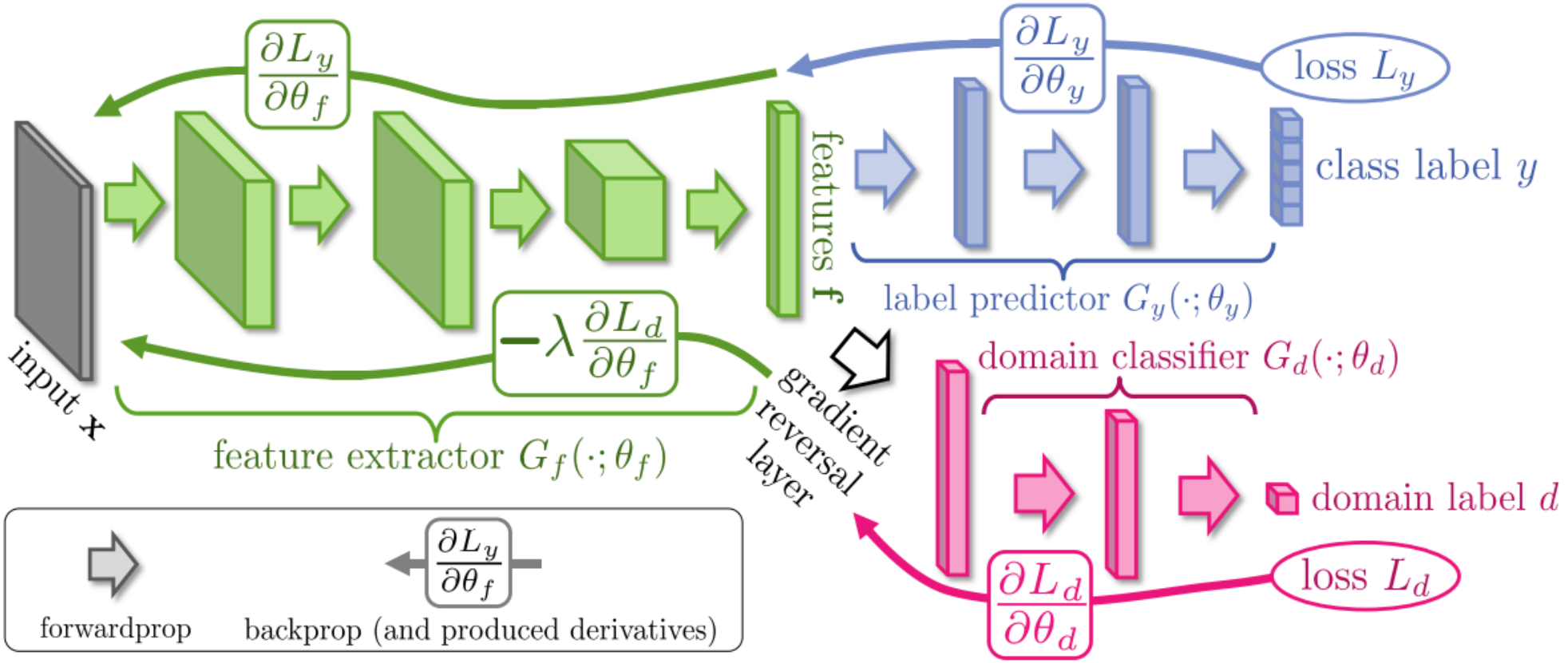
Train the first part lightly



Train the new part intensely

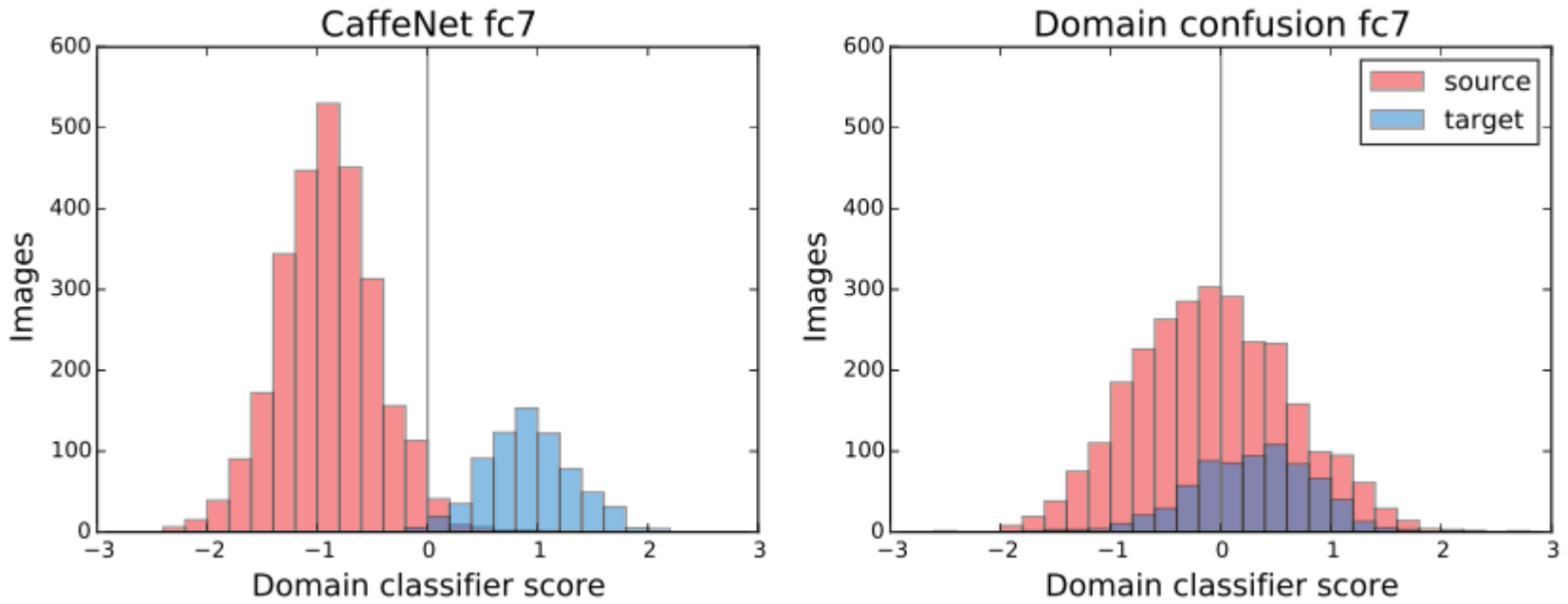


Confusing Domains (Datasets)



https://arxiv.org/abs/1409.7495 , Unsupervised Domain Adaptation by Backpropagation
Yaroslav Ganin, Victor Lempitsky

Add a new loss term, which punishes differentiation between the two domains. Forcing the model to learn the new classification task without learning to differentiate between the two tasks.



Implementing domain confusion results in the model being unable to classify domains (Tzeng et al, 2015)

Machine Learning Intro 5b:

Recurrent Neural Networks

Links

- Three part tutorial on RNNs
- Understanding LSTM networks
- Intro to Recurrent Neural Networks
- Intro to LSTM networks (Video)

Recurrent Neural Network

- Recurrent Neural Networks are created to deal with sequential data, they do this by having a ‘memory’ of the previous data they encountered
- Some use cases are for example Language processing where a word is made up of interdependent letters, or sentences are made up of contextual words.

  Signal processing, stock market predictions, any sort of sequences where patterns can be found and are relevant to the task

Recurrent Neural Network

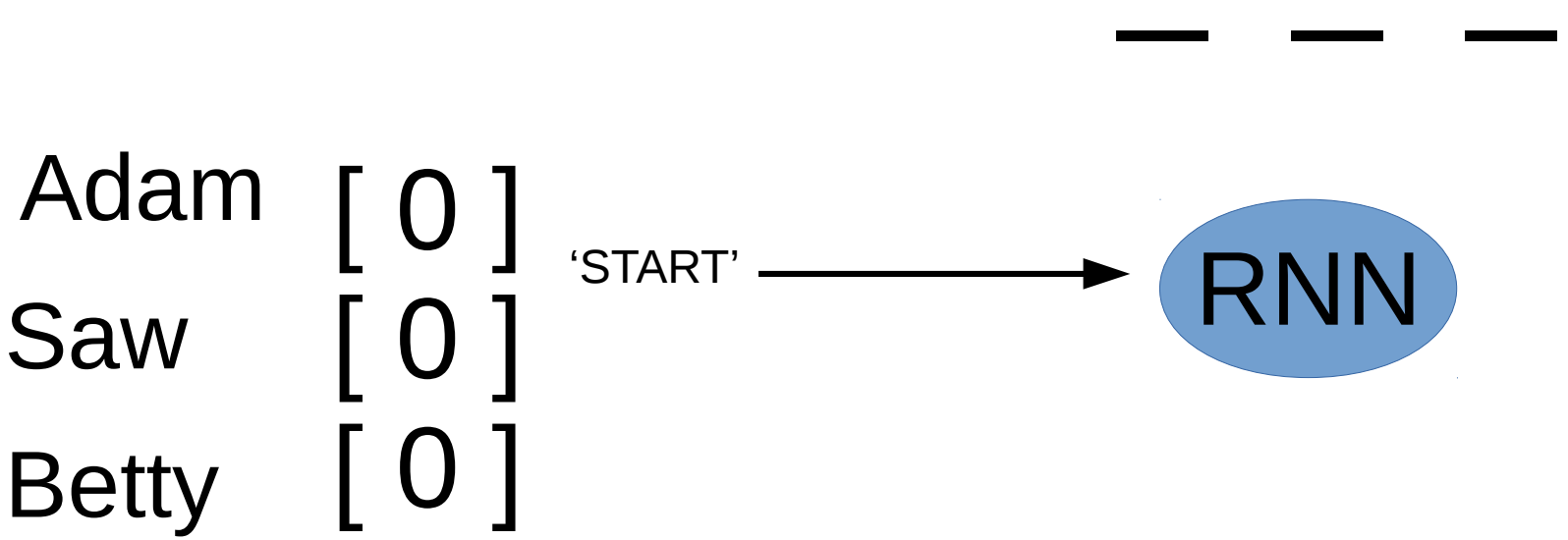
- RNNs learn patterns and dependencies between certain inputs on a temporal basis.

Example problem: Say the phase ‘Adam saw Betty’

- A dense neural network can learn to say ‘saw’ after ‘Adam’ is the input, and ‘Betty’ after ‘saw’.
- But without retaining the information of what came before, it cannot learn to say ‘___ saw ___’ for arbitrary inputs.
- Whereas a RNN can learn that after a blank comes a Name, after a name comes ‘saw’, and after saw comes a name that is not the first.

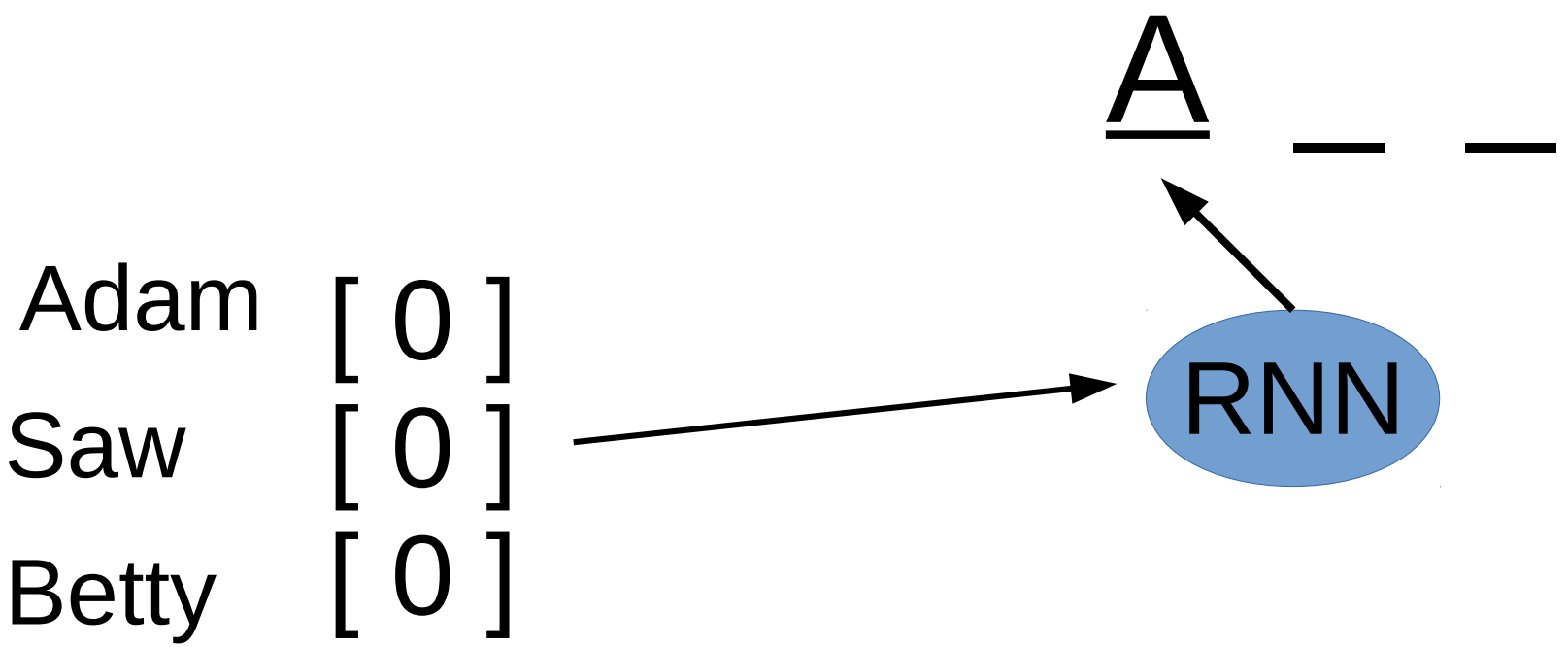
Example problem: Say the phase ‘A saw B’

- You could have the information that came previously as an input for a dense NN, but then what you have is effectively a very dumb RNN.



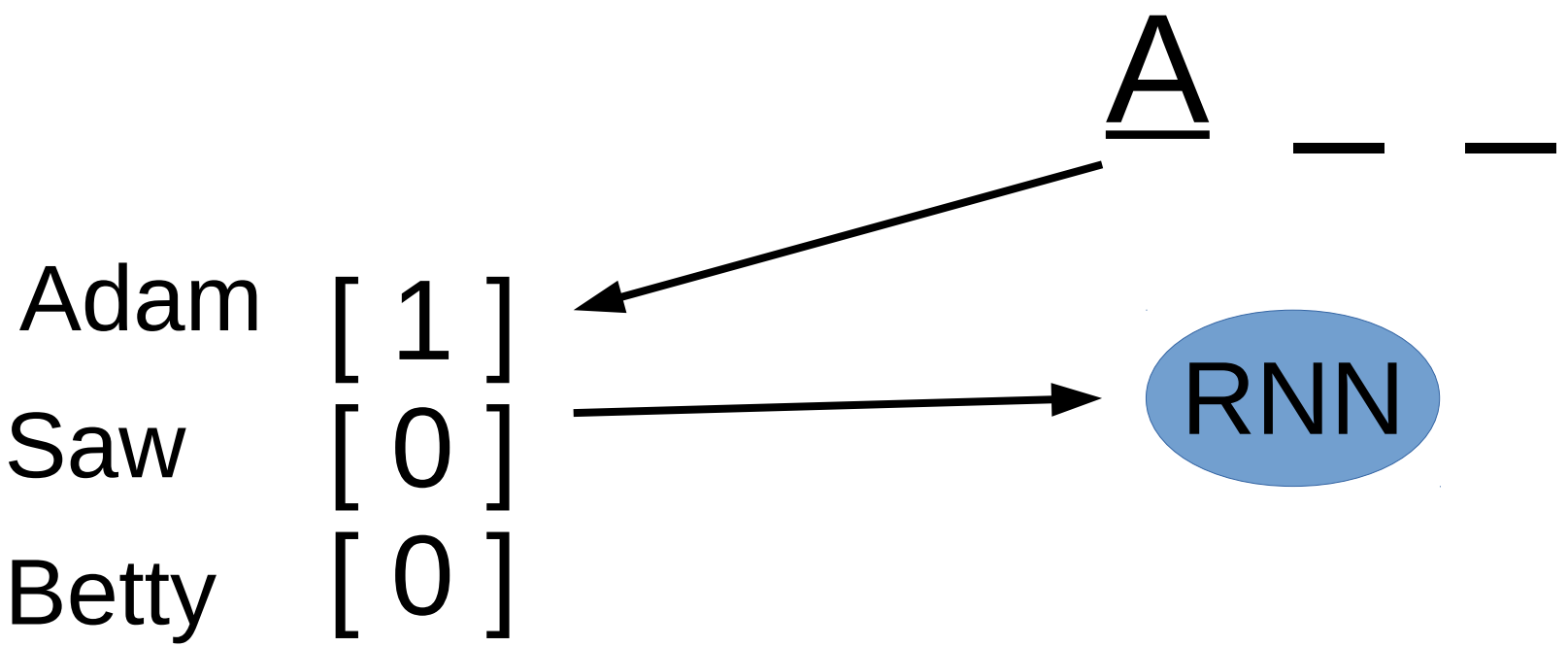
Example problem: Say the phase ‘A saw B’

- You could have the information that came previously as an input for a dense NN, but then what you have is effectively a very dumb RNN.



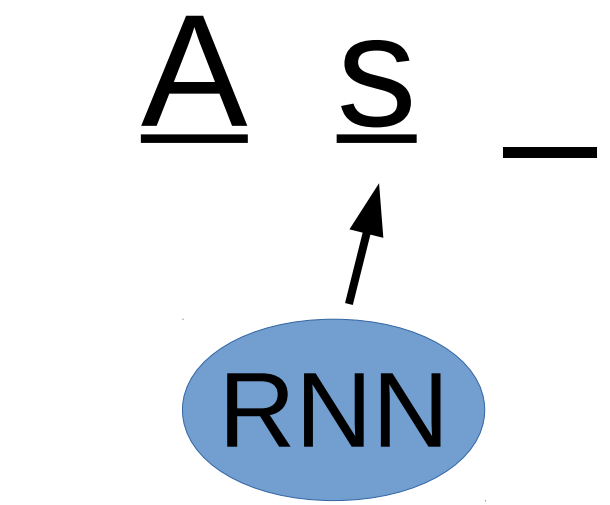
Example problem: Say the phase ‘A saw B’

- You could have the information that came previously as an input for a dense NN, but then what you have is effectively a very dumb RNN.



Example problem: Say the phase ‘A saw B’

- You could have the information that came previously as an input for a dense NN, but then what you have is effectively a very dumb RNN.



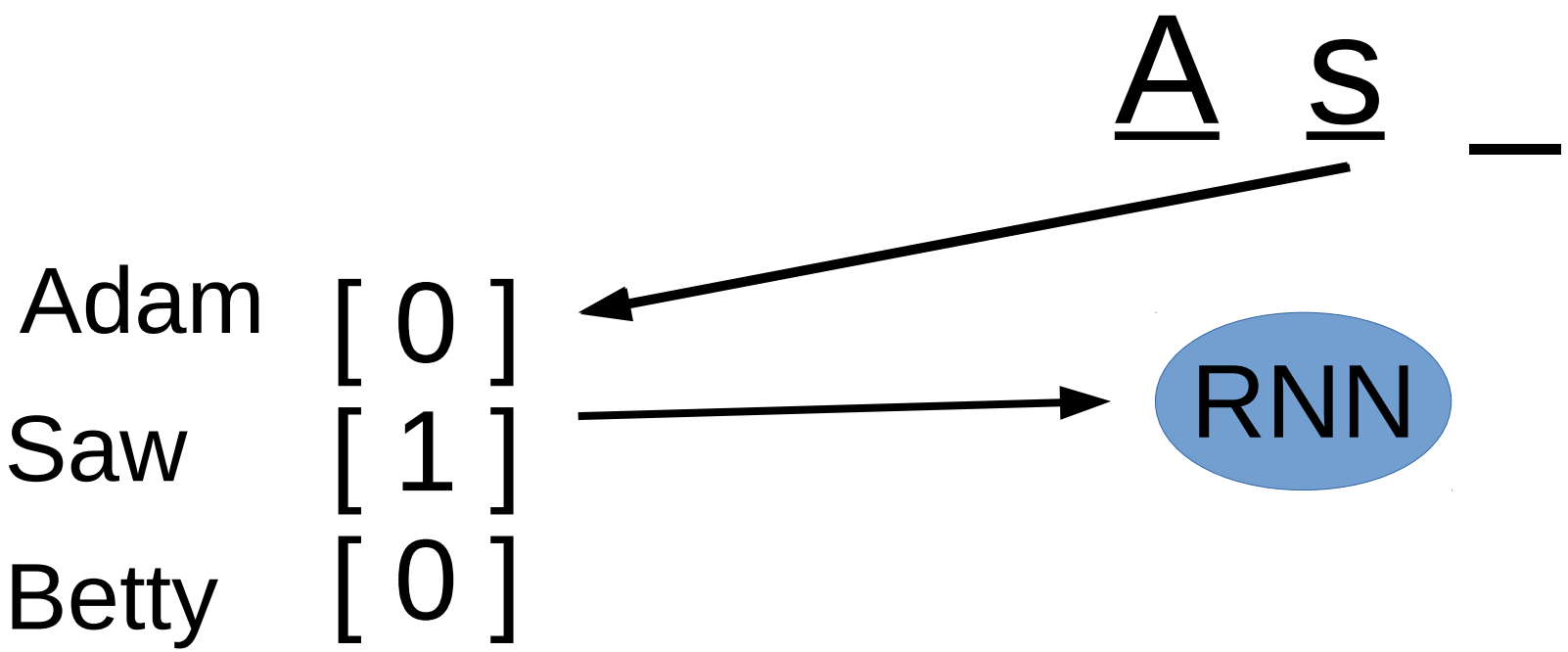
Adam [1]

Saw [0]

Betty [0]

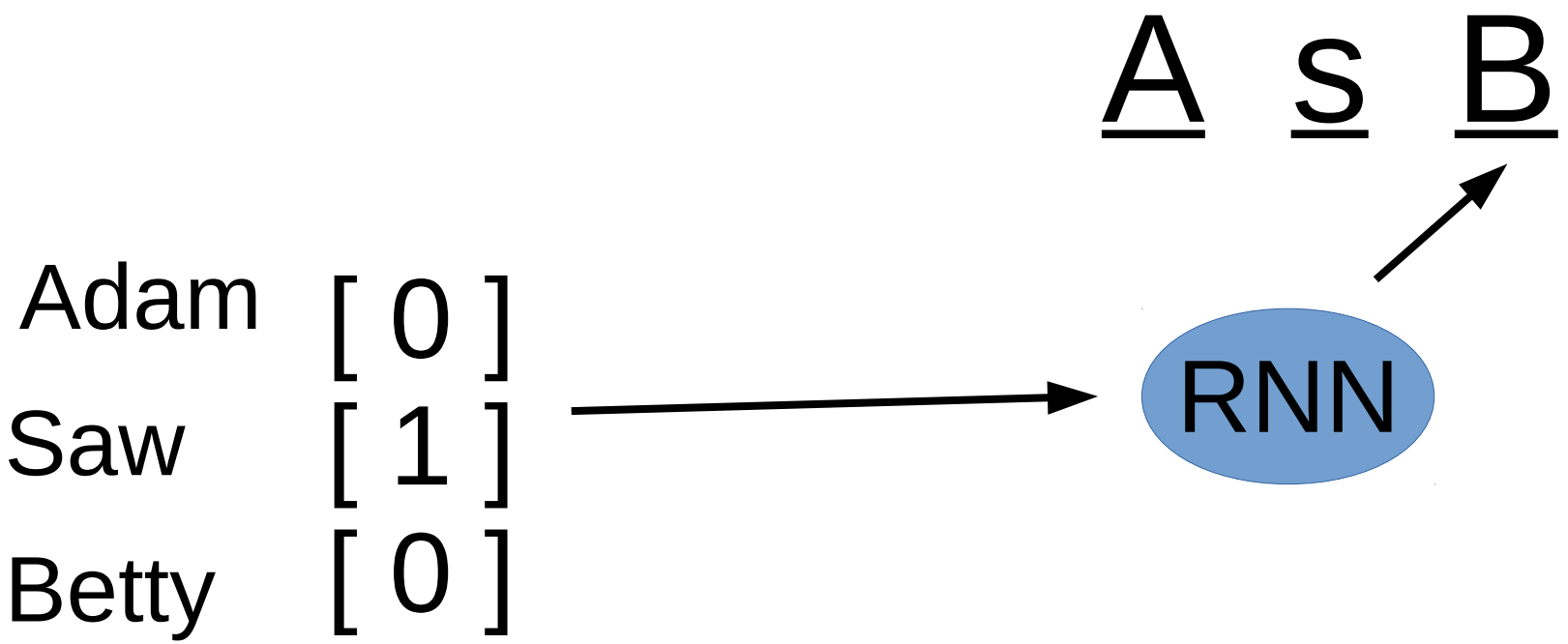
Example problem: Say the phase ‘A saw B’

- You could have the information that came previously as an input for a dense NN, but then what you have is effectively a very dumb RNN.



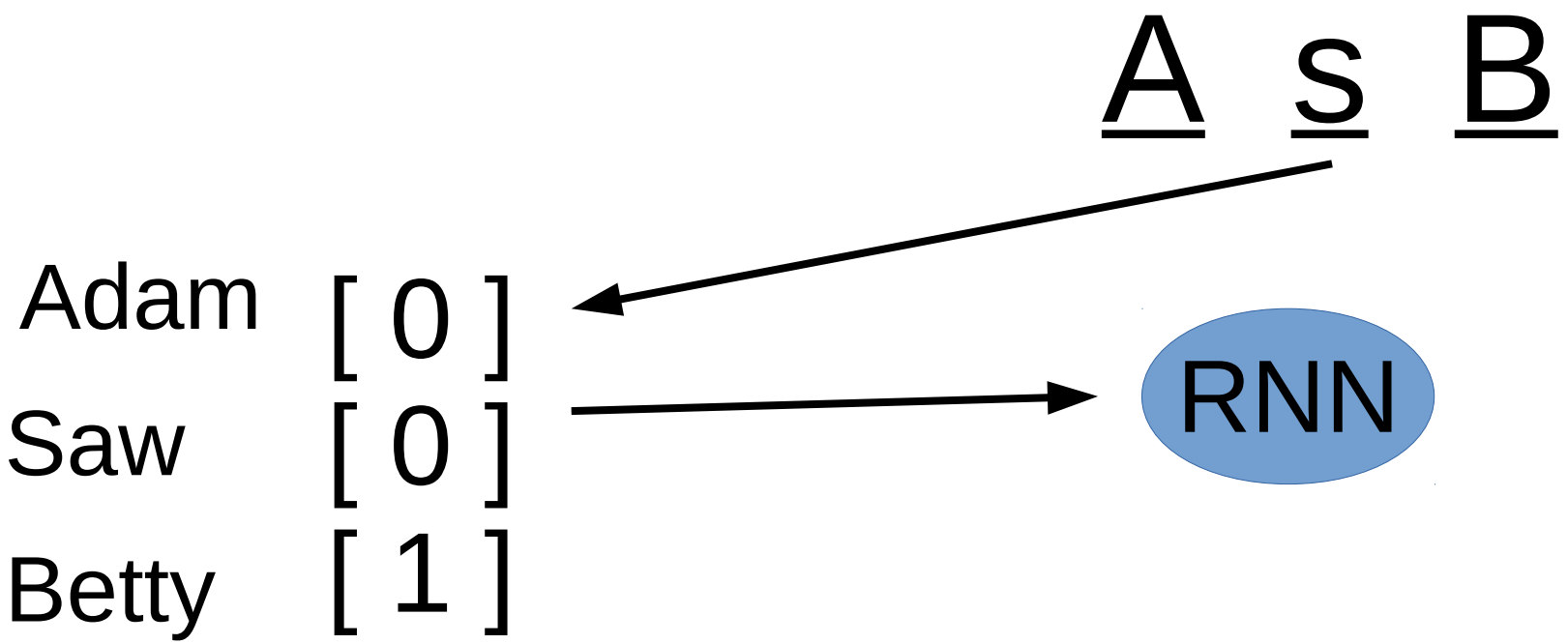
Example problem: Say the phase ‘A saw B’

- You could have the information that came previously as an input for a dense NN, but then what you have is effectively a very dumb RNN.



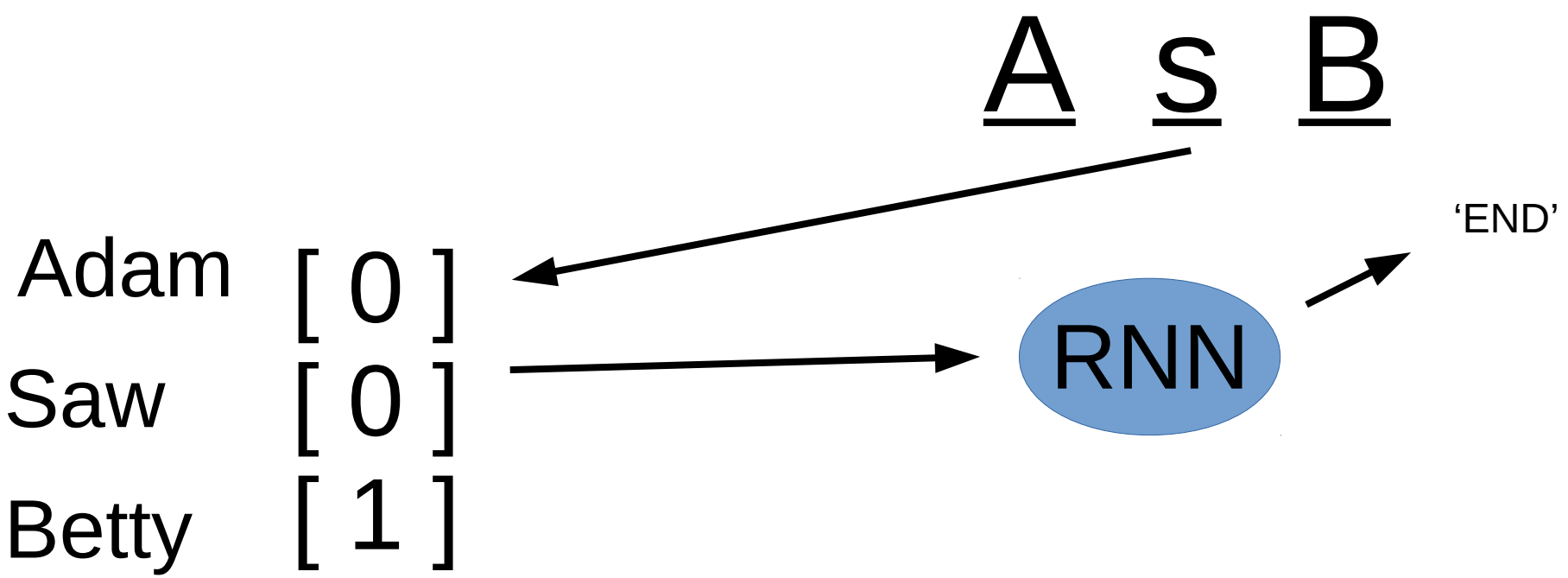
Example problem: Say the phase ‘A saw B’

- You could have the information that came previously as an input for a dense NN, but then what you have is effectively a very dumb RNN.

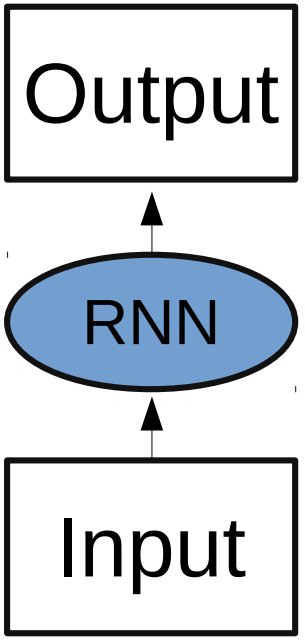


Example problem: Say the phase ‘A saw B’

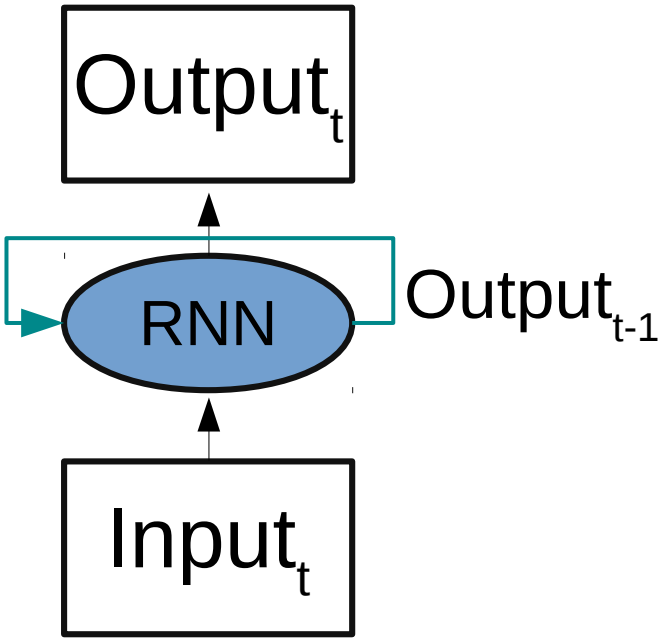
- You could have the information that came previously as an input for a dense NN, but then what you have is effectively a very dumb RNN.



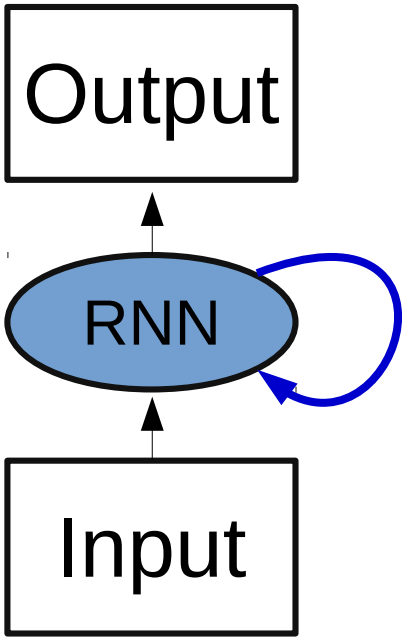
Displaying a Recurrent Neural Network



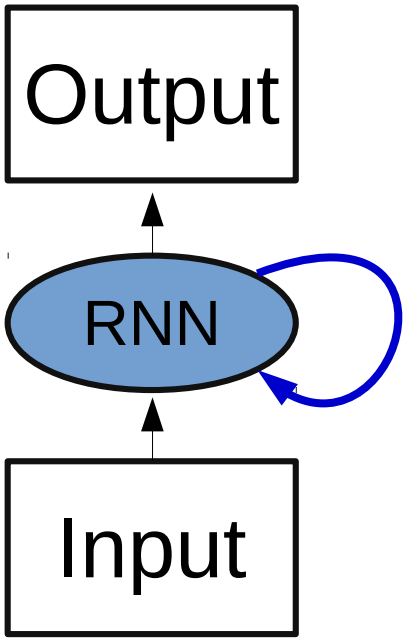
Displaying a Recurrent Neural Network



Displaying a Recurrent Neural Network

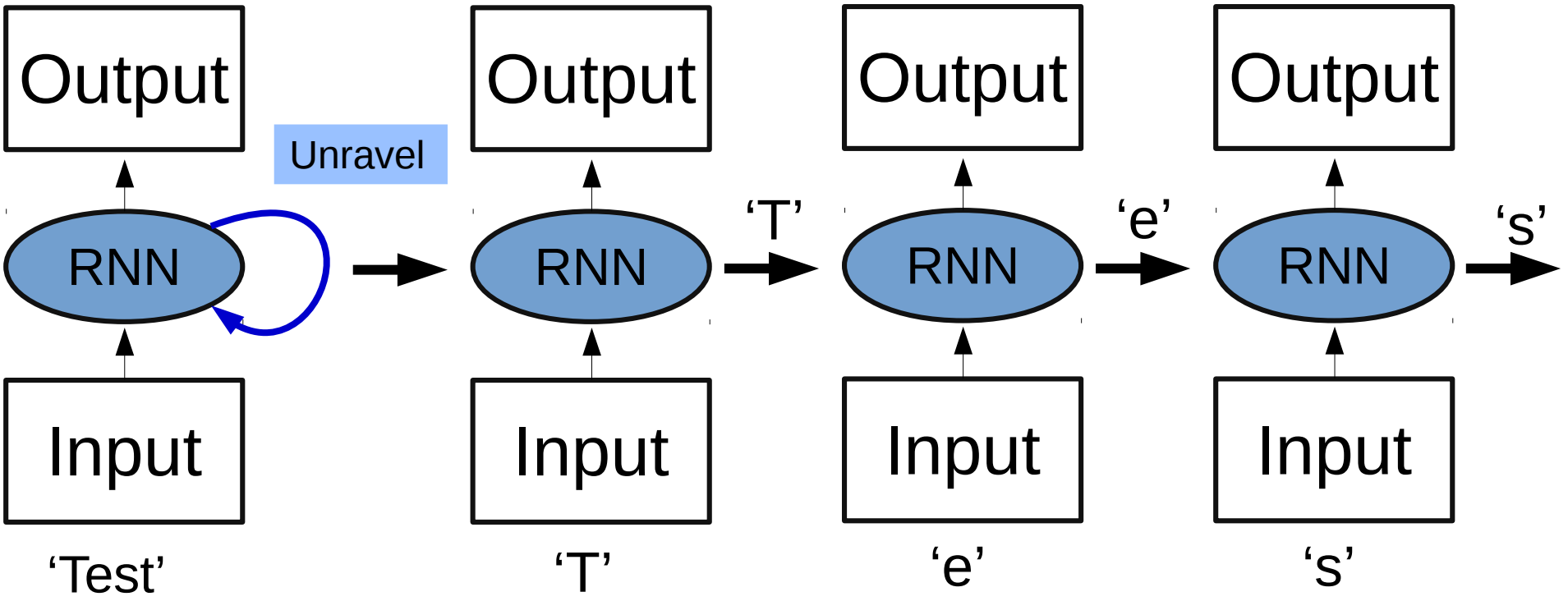


Displaying a Recurrent Neural Network



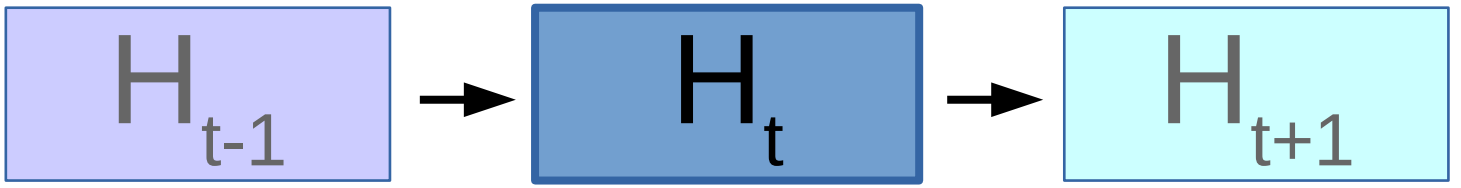
‘Test’

Displaying a Recurrent Neural Network



Recurrent Neural Network

RNNs have a ‘memory’ called the hidden state



Which depends on the previous hidden state.

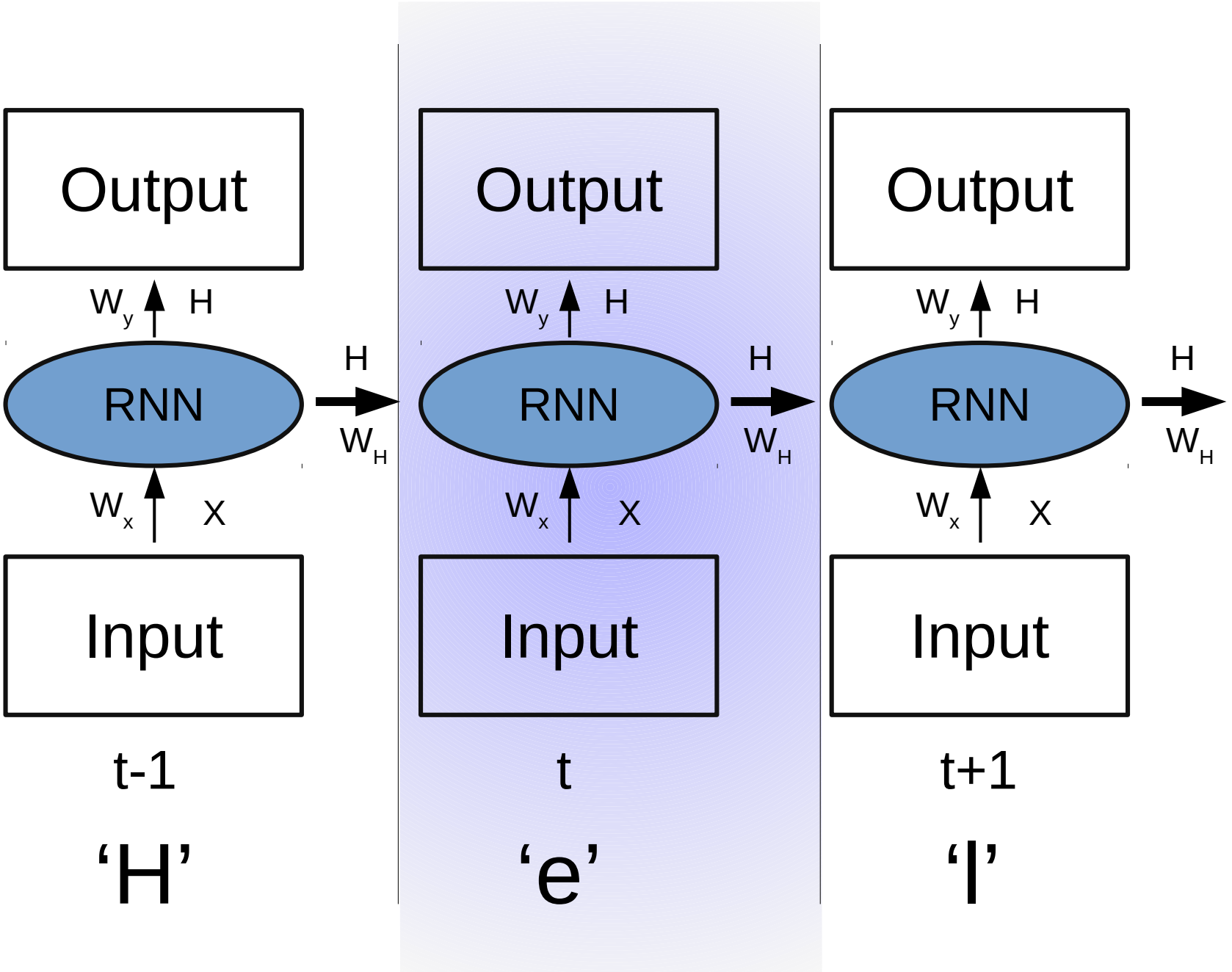
RNNs take X as input and use it to calculate H:

$$H_{new} = f(weights_X * X + weights_H * H_{t-1})$$

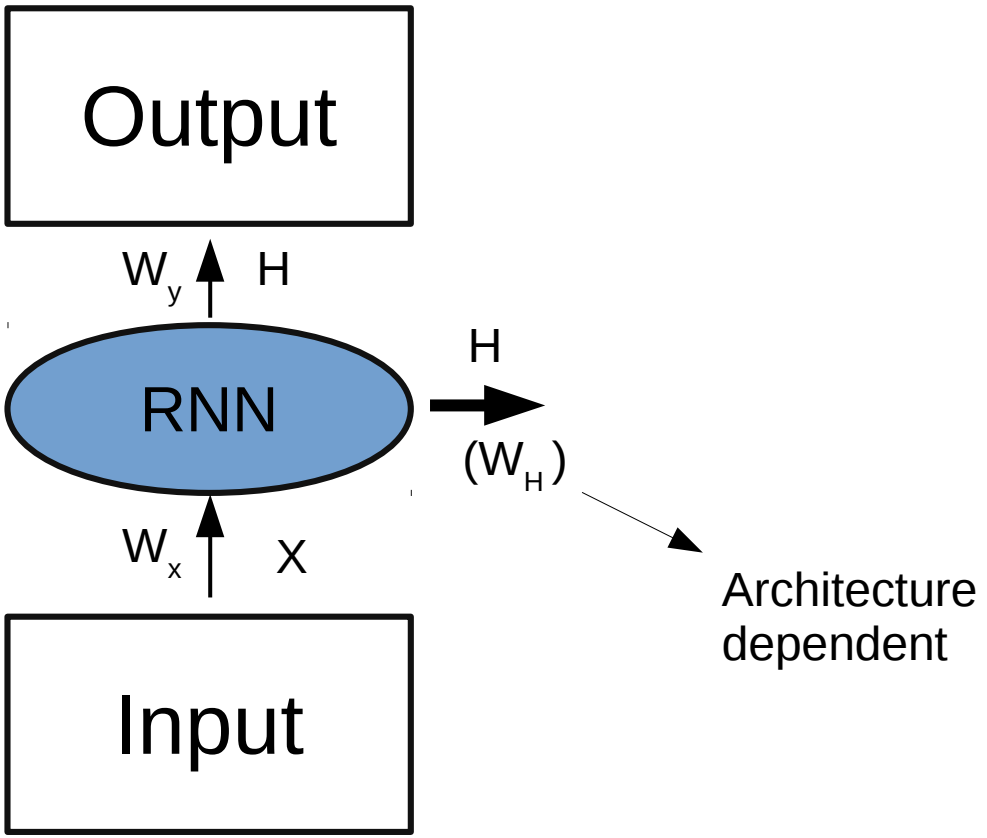
which is then used to calculate the Output:

$$y_t = weights_y * H$$

The Loop

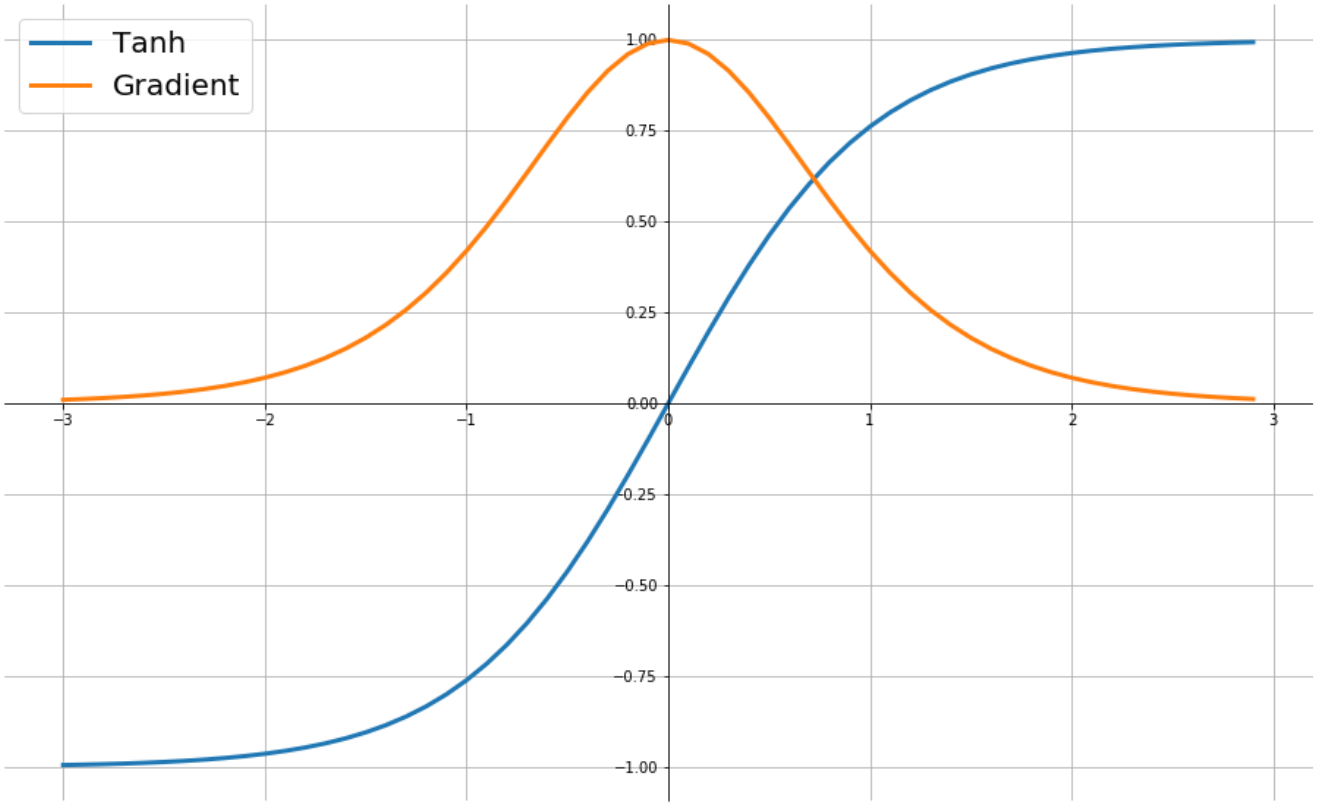

Output
Output
Output
W_y H
W_y H
W_y H
H
H
H
RNN
RNN
RNN
W_H
W_H
W_H
W_x X
W_x X
W_x X
Input
Input
Input
t-1
t
t+1
‘H’
‘e’
‘l’

The Loop



Exploding/ Vanishing Gradients

- When you do backpropagation, calculate the gradient over multiple steps, which can lead the gradient to converge to zero the more steps you calculate



Exploding/ Vanishing Gradients

- The Vanishing Gradient problem can be countered by using ReLu activation functions, Regularization, GRU or LSTM cells.
- The exploding gradient problem can be eliminated by simply clipping the gradient at a max value. It is also easy to detect as the values will quickly exceed the memory limit and become NaN, crashing the programm.

A RNN Cell Examples:

GRU (Gated Recurrent Unit)

GRU Cell (simplified LSTM cell)

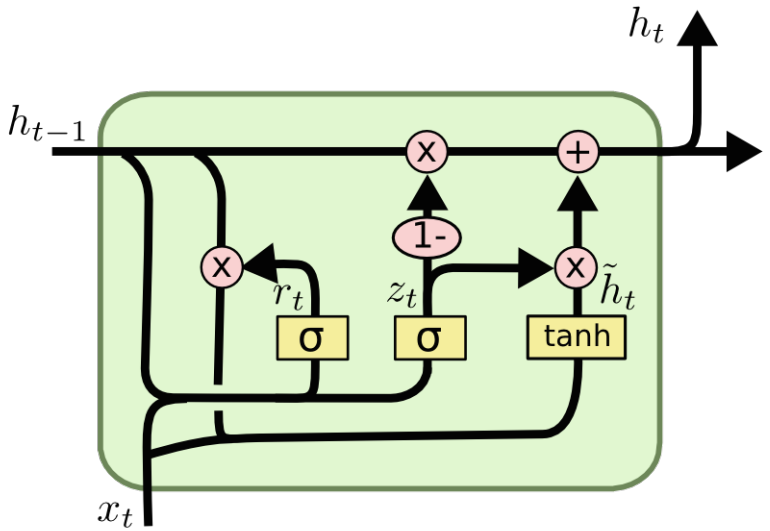
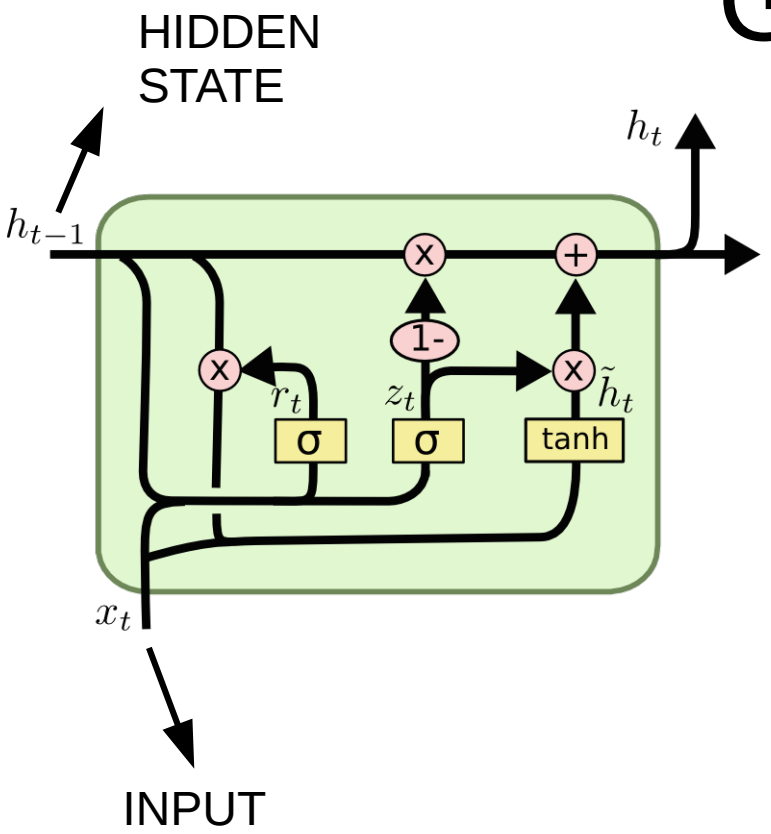
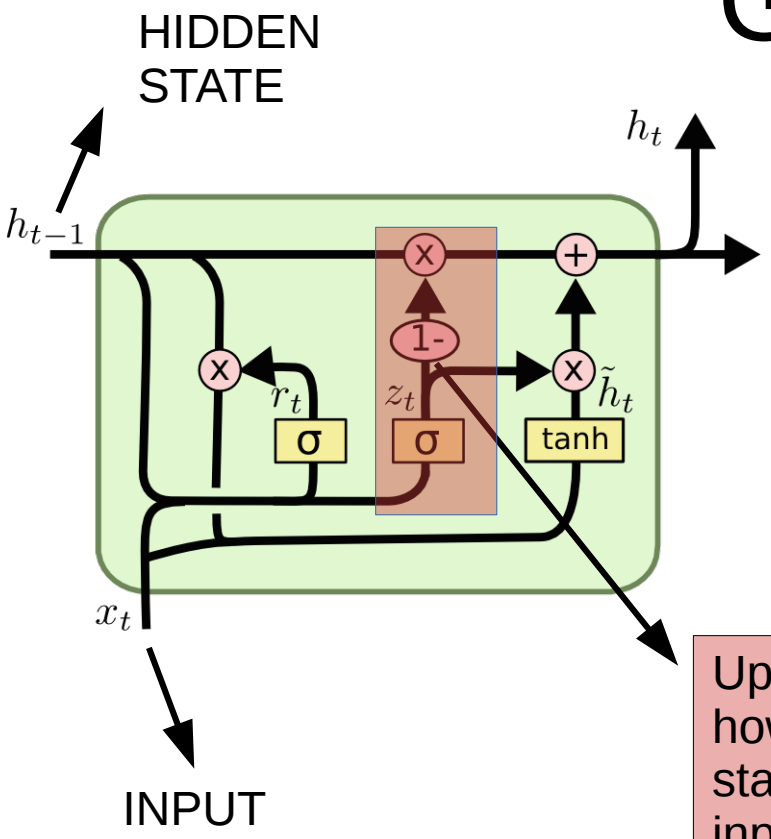


Image from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

GRU Cell

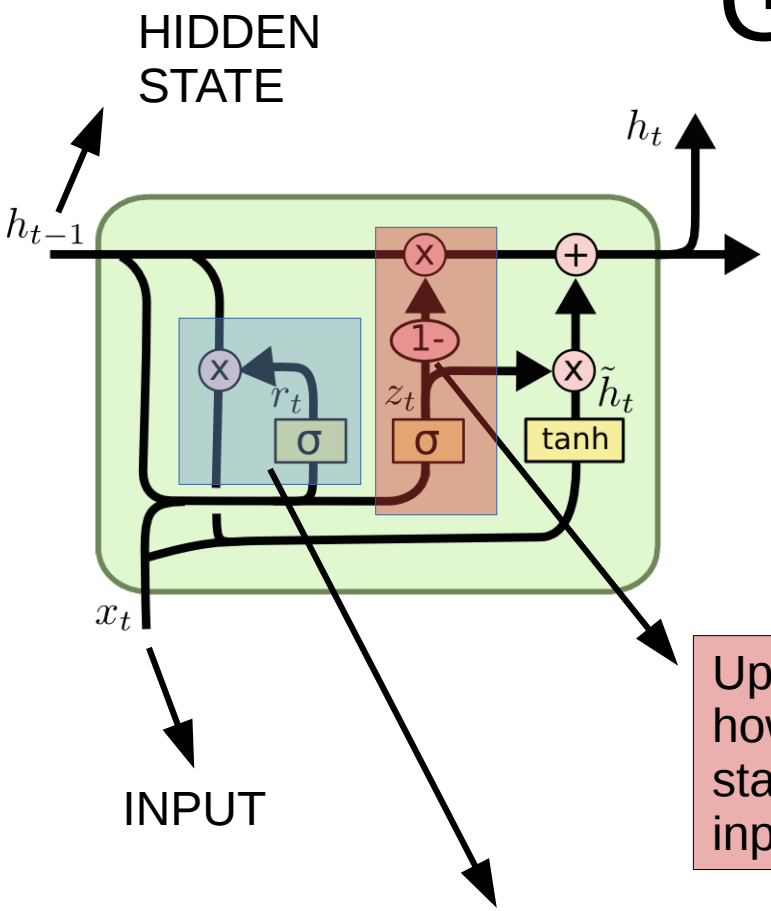


GRU Cell



Update Gate: determines how much of the Hidden state ist changed by the input x

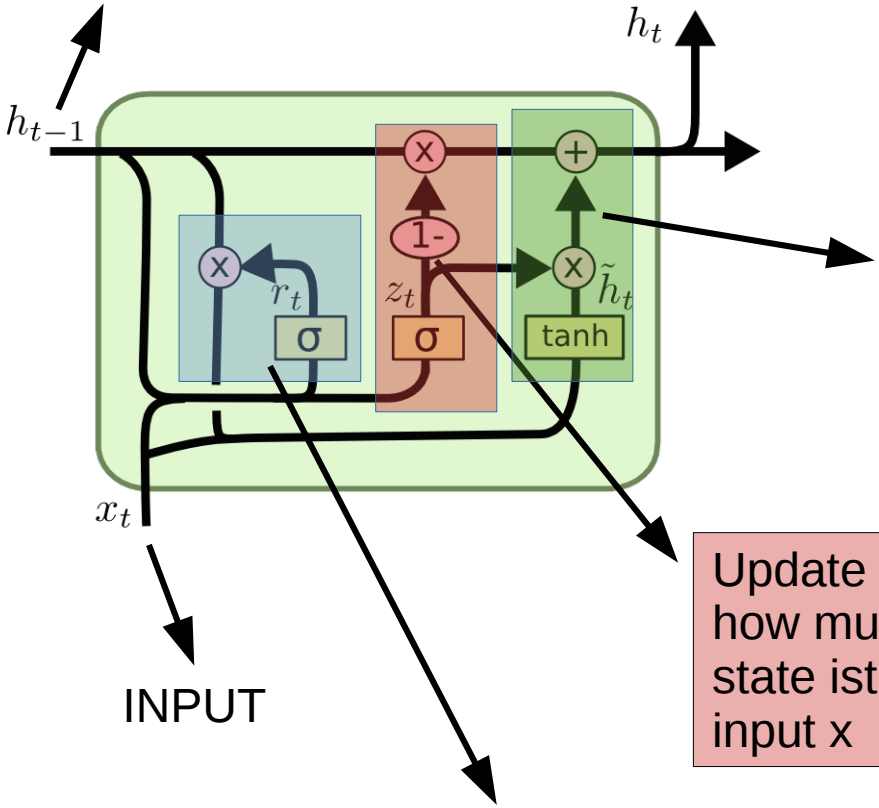
GRU Cell



Update Gate: determines how much of the Hidden state ist changed by the input x

Reset Gate: Combines the input X with the Hidden state

GRU Cell

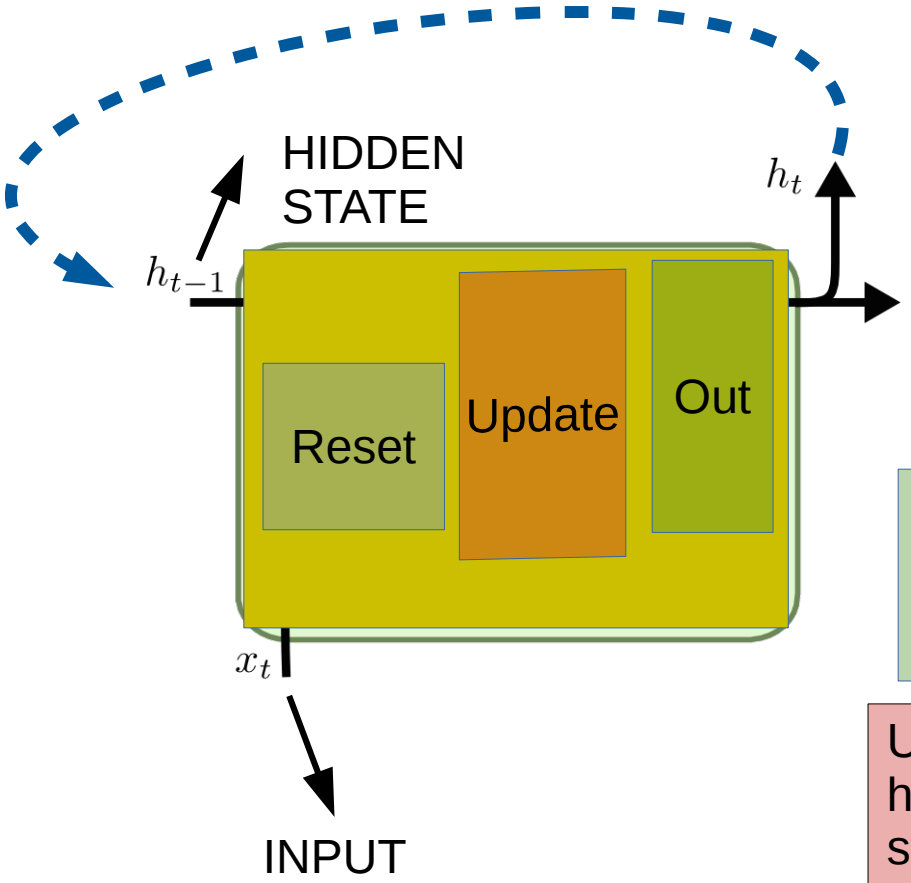


Output is calculated from Using all other gates and inputs

Update Gate: determines how much of the Hidden state ist changed by the input x

Reset Gate: Combines the input X with the Hidden state

GRU Cell



Hidden State is the output and is sent as the hidden state input of the next recursion

Output is calculated from Using all other gates and inputs

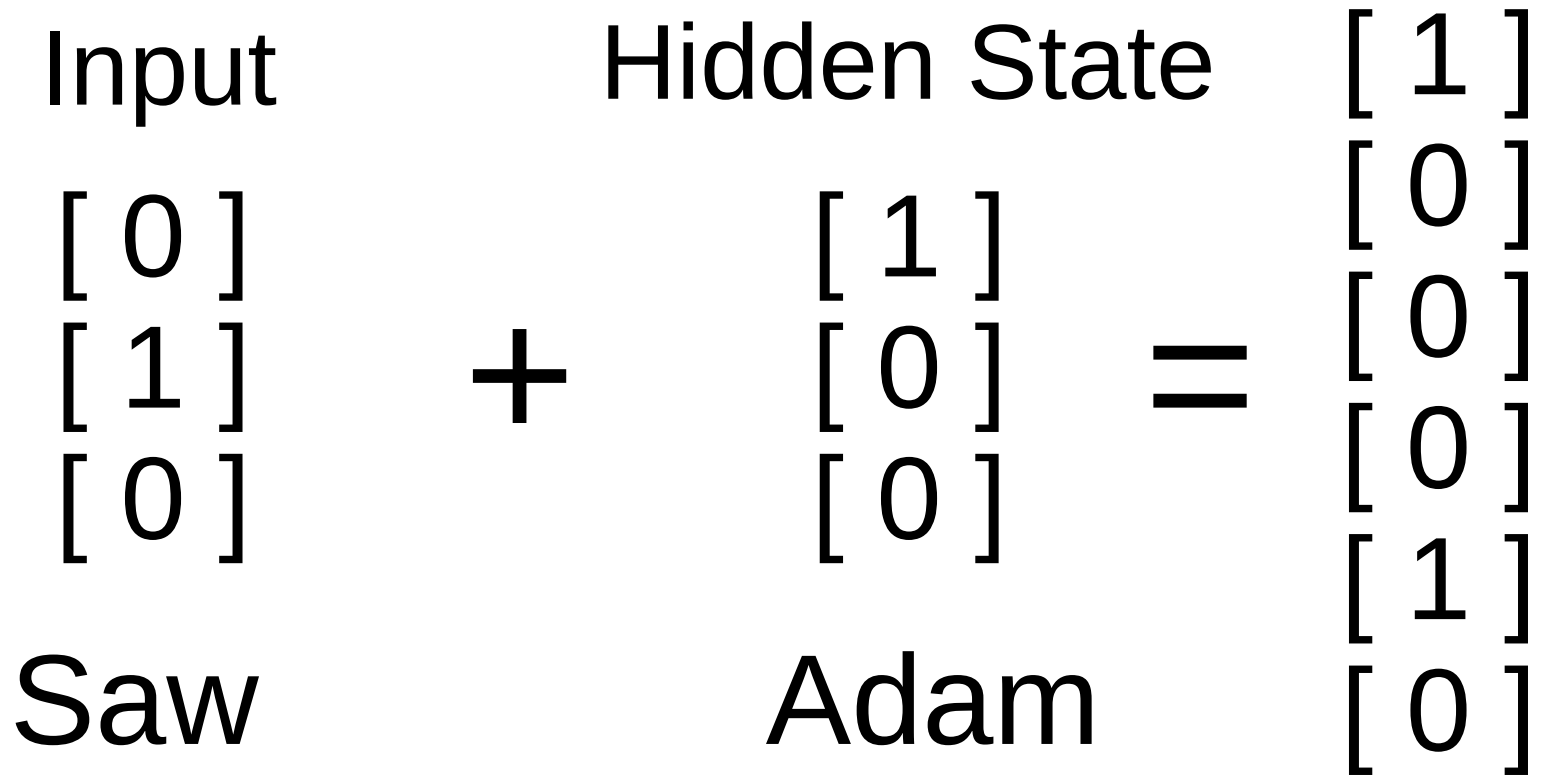
Update Gate: determines how much of the Hidden state ist changed by the input x

Reset Gate: Combines the input X with the Hidden state

GRU Cell

Reset Gate: Combines the input X with the Hidden state

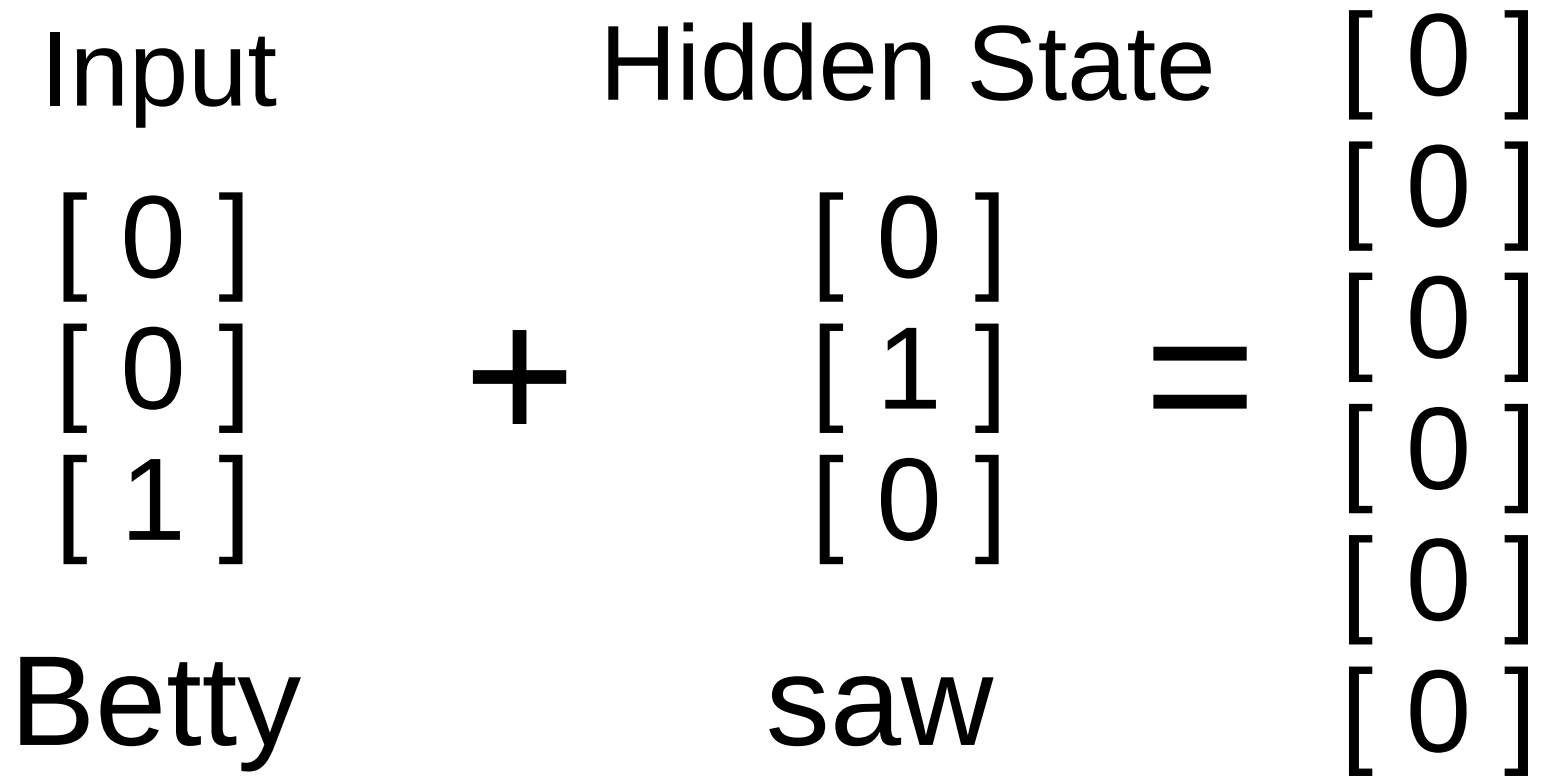
Note that the Hidden State is more complex than just the last input



GRU Cell

Update Gate: determines how much of the Hidden state ist changed by the input x

Note that the Hidden State is more complex than just the last input



Memory

- The Hidden state and it’s weights learn patterns depending on the data
- For example a translation Model might learn the syntax of the languages it’s translating. In English it might be:

‘The’ - Adjective - Noun - ‘is’ - Verb

Questions?