Using Fast Weights to Attend to the Recent Past

Wang CHEN

Poster #191 Tue

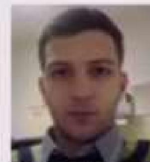
Using Fast Weights to Attend to the Recent Past

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Leibo, Catalin Ionescu
University of Toronto
Google

Geoffrey Hinton

Vlad Minh

Joel Leibo

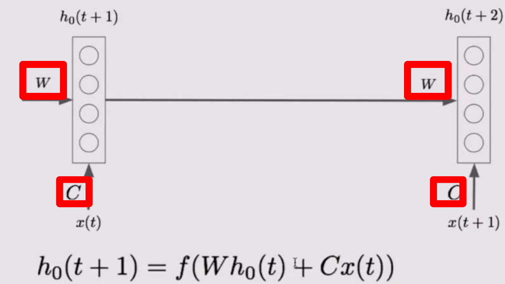

Catalin Ionescu

link1

Poster #191 Tue

The Basic Idea

In recurrent neural networks (RNNs):



- The usual weights: They encode knowledge over the entire training dataset through gradient descent algorithm
  - Slow varying, storing long term information about input and output mapping

link1

Poster #191 Tue

The Basic Idea

Ordinary RNN

$$h_0(t+1) = f(Wh_0(t) + Cx(t))$$

In recurrent neural networks (RNNs):

- Hidden states or activity vectors: They act as a limited working memory storing the current sequence information
    - Change at every timestep

link1

Poster #191 Tue

The Basic Idea

How about short-term memory?

- Where do we store the temporary information?
- The hidden states have to remember the history of the current sequence but they also have to integrate appropriate memory content for the final classifier

Poster #191 Tue

The Basic Idea

- **The slow weight**: slow varying, storing long term information
- **The fast weight**: rapid learning but also decaying rapidly, storing sequence specific temporary information

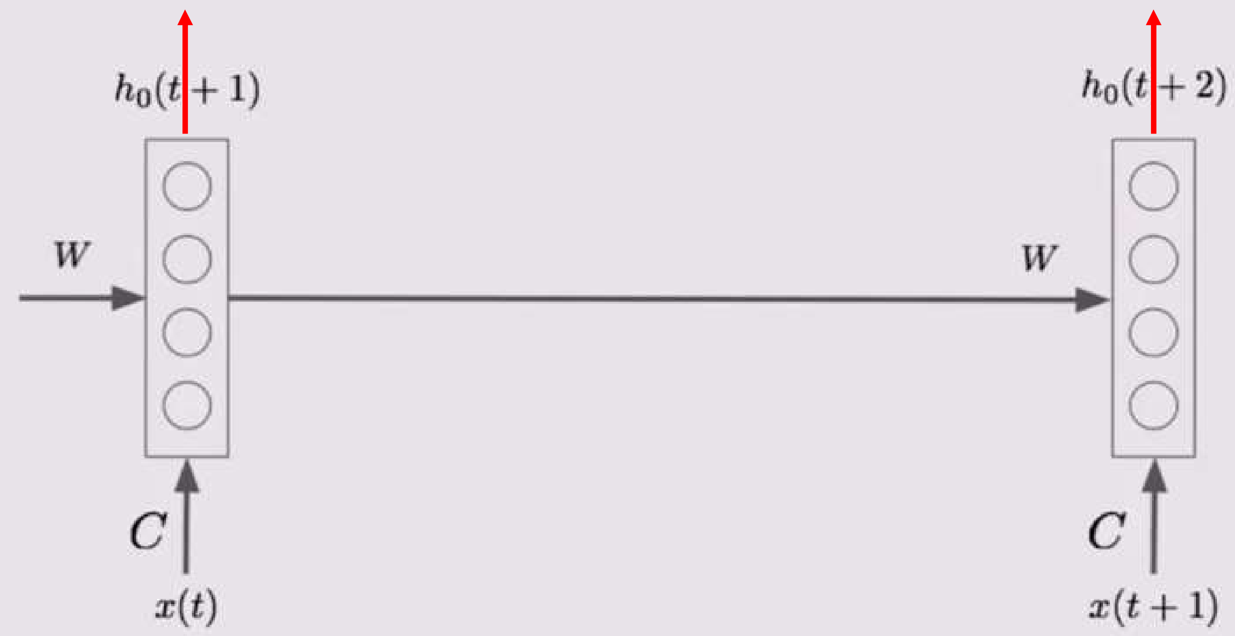
Poster #191 Tue

The Basic Idea

- **The slow weight**: slow varying, storing long term information
- **The fast weight**: rapid learning but also decaying rapidly, storing sequence specific temporary information
    - *Hinton and Plaut, 1987, Using fast weights to deblur old memories*
    - *Schmidhuber, 1993, Reducing the ratio between learning complexity and number of time varying variables in fully recurrent nets*

Poster #191 Tue

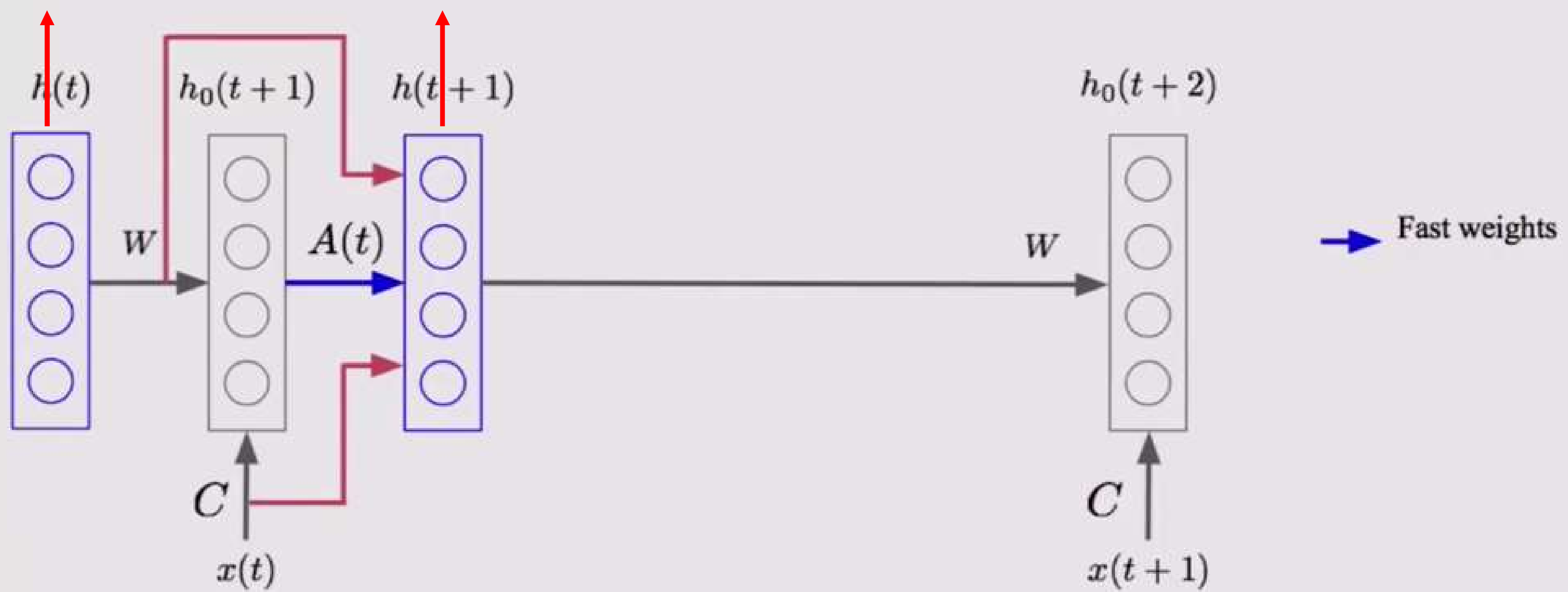
Ordinary RNN



$$h_0(t+1) = f(Wh_0(t) + Cx(t))$$

Poster #191 Tue

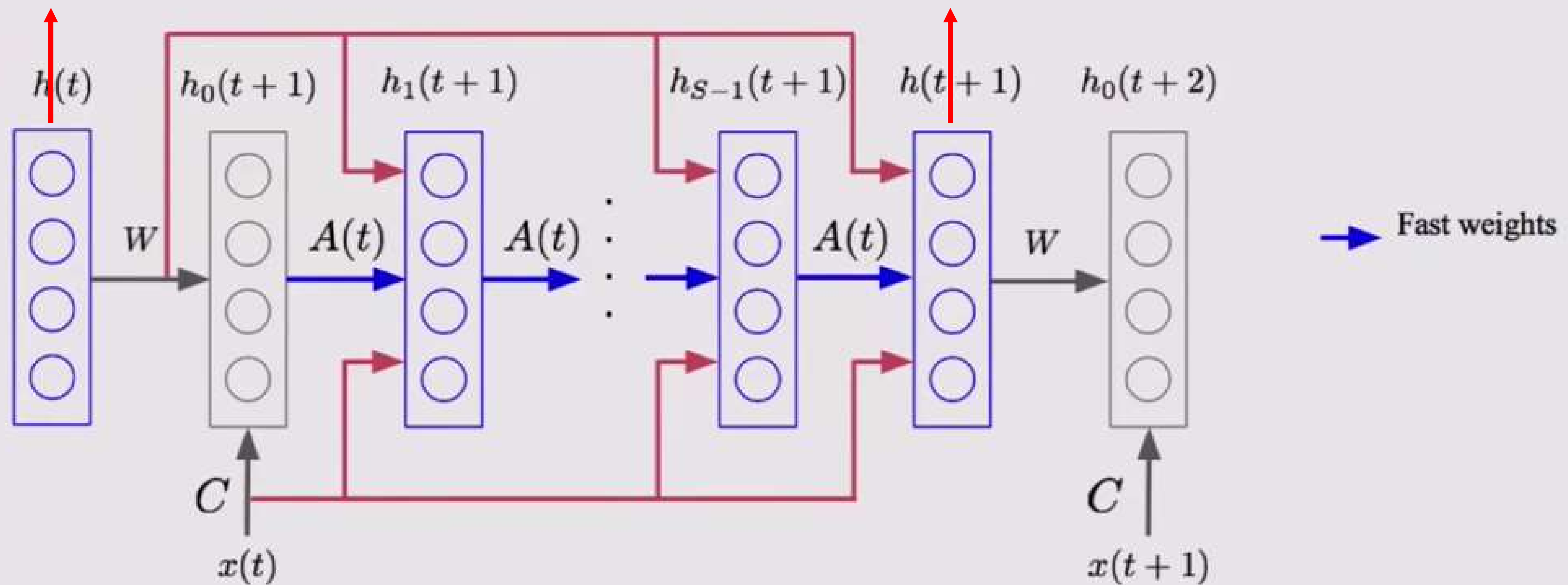
Fast weights RNN



$$h(t+1) = f([Wh(t) + Cx(t)] + A(t)h_0(t+1))$$

Poster #191 Tue

Fast weights RNN



$$h_{s+1}(t+1) = f([Wh(t) + Cx(t)] + A(t)h_s(t+1))$$

Poster #191 Tue

Hopfield Network and Associative Learning

- Fast weights update rule:

$$A(t) = \lambda A(t-1) + \eta h(t)h(t)^T$$

- Embedding a Hopfield network inside an RNN

Poster #191 Tue

Computation Efficiency

- Revisit the fast weights update rule:

$$A(t) = \lambda A(t-1) + \eta h(t)h(t)^T$$

- Summation of rank-one matrices $A(0) = \mathbf{0}$

$$A(t) = \eta \sum_{\tau=1}^{\tau=t} \lambda^{t-\tau} h(\tau)h(\tau)^T$$

Poster #191 Tue

Computation Efficiency

- Fast weights computation is equivalent to attention to the past:

$$A(t)h_s(t+1) = \eta \sum_{\tau=1}^{\tau=t} \lambda^{t-\tau} h(\tau) \underbrace{[h(\tau)^T h_s(t+1)]}_{\text{Attention}}$$

- Store hidden vectors instead of the fast weight matrix

Poster #191 Tue

Attention to the recent past

- Fast weights computation is equivalent to attention to the past:

$$A(t)h_s(t+1) = \eta \sum_{\tau=1}^{\tau=t} \lambda^{t-\tau} h(\tau) \underbrace{[h(\tau)^T h_s(t+1)]}_{\text{Attention}}$$

- Fast weights is a biologically plausible implementation of the attention mechanism

Poster #191 Tue

Application: associative retrieval

c9k8j3f1 ?? c

Poster #191 Tue

Application: associative retrieval

c9k8j3f1 ?? c → 9

Poster #191 Tue

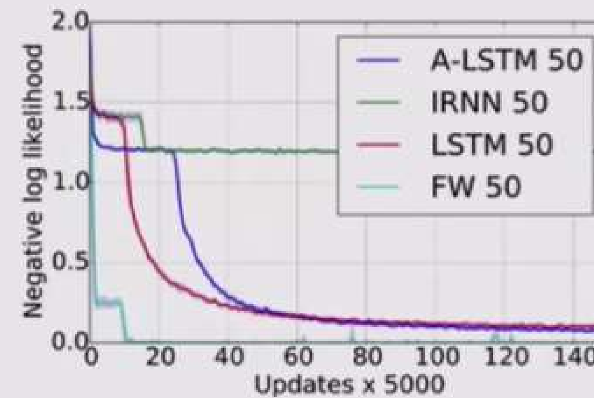
Application: associative retrieval

Input string	Target
c9k8j3f1??c	9
j0a5s5z2??a	5

Poster #191 Tue

Application: associative retrieval

	# of recurrent units		
Model	R=20	R=50	R=100
IRNN	62.11%	60.23%	0.34%
LSTM	60.81%	1.85%	0%
A-LSTM	60.13%	1.62%	0%
Fast weights	1.81%	0%	0%



Poster #191 Tue

Application: recursive vision task

Classify facial expression from 48x48 images into six categories:
{neutral, smile, surprise, squint, disgust and scream}

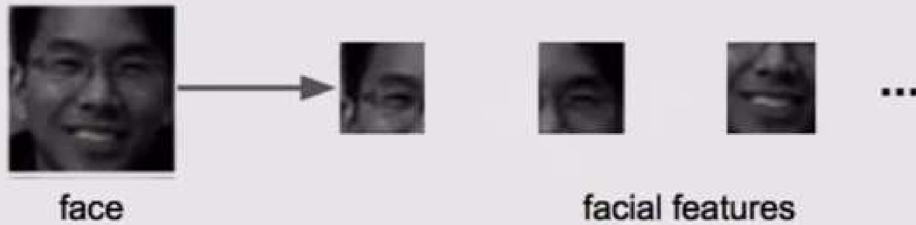
Poster #191 Tue

Application: recursive vision task

```
def get_facial_expression(face):
    expression = None

    for facialFeature in face:

        featureExpression = get_facial_expression(facialFeature)
        expression = integrate_expression(expression, featureExpression)

    return expression
```

Poster #191 Tue

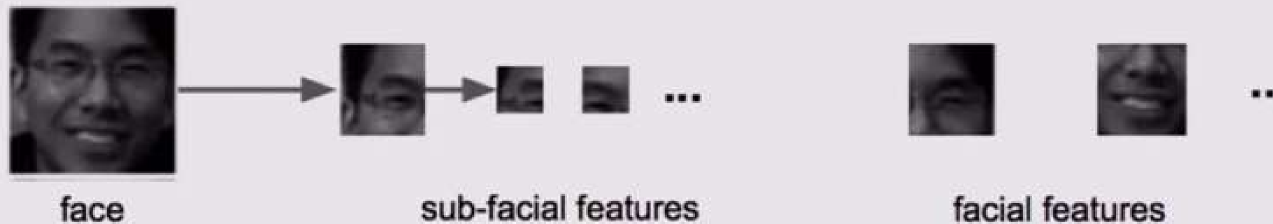
Application: recursive vision task




```
def get_facial_expression(face):
  expression = None

  for facialFeature in face:

     featureExpression = get_facial_expression(facialFeature)
     expression = integrate_expression(expression, featureExpression)

  return expression
```

Poster #191 Tue

Application: recursive vision task



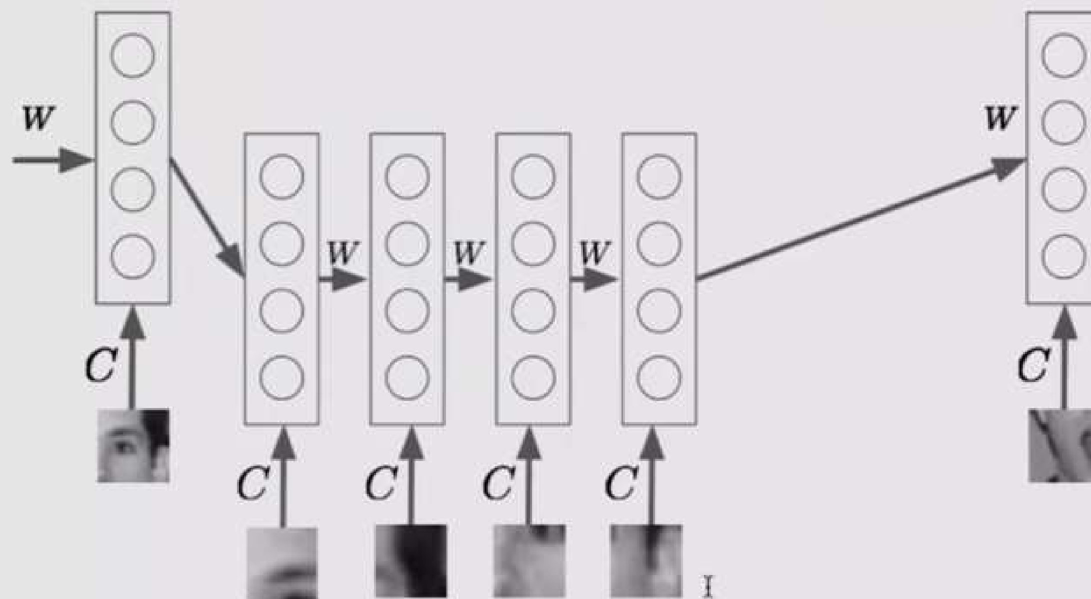

```
def get_facial_expression(face):
    expression = None

    for facialFeature in face:

        featureExpression = get_facial_expression(facialFeature)
        expression = integrate_expression(expression, featureExpression)

    return expression
```

Poster #191 Tue

Application: recursive vision task

Poster #191 Tue

Application: recursive vision task



Poster #191 Tue

Application: recursive vision task

W

A(t)

W

W

W

W

C

C

C

C

C

C

C

Push(update fast weights)

Integration

Pop(fast weights)

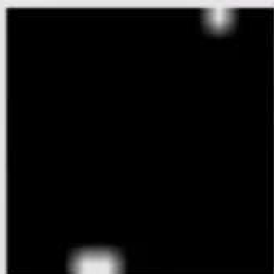
Poster #191 Tue

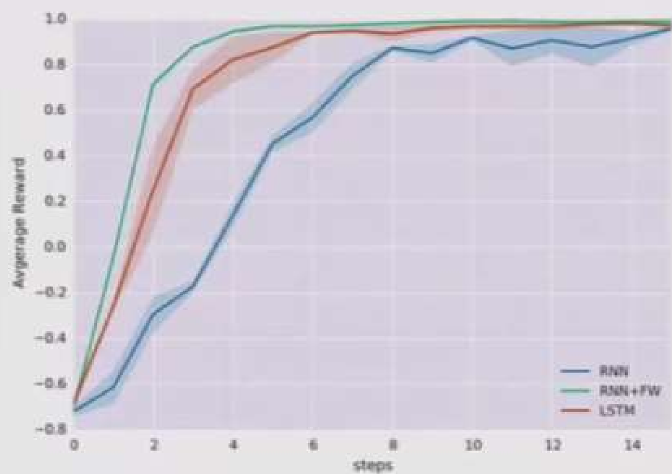
Application: recursive vision task

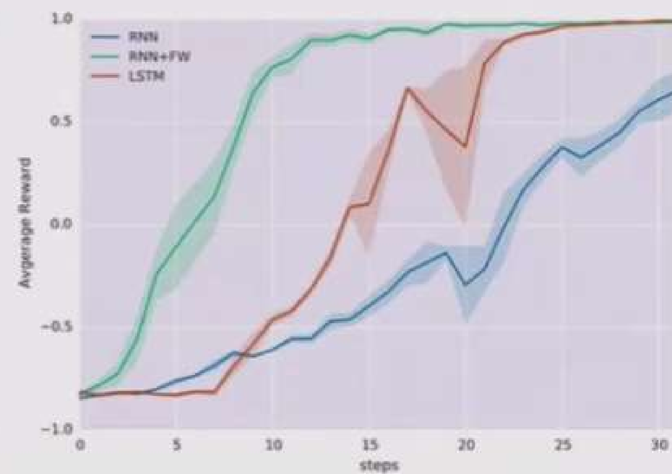
	IRNN	LSTM	ConvNet	Fast Weights
Test accuracy	81.11	81.32	88.23	86.34

Poster #191 Tue

Application: improving RL agents







Poster #191 Tue

Conclusion

- Fast weights provide a powerful *internal* storage mechanism for RNNs
  - Fast associative weights retrieves information by attracting new states of the hidden units towards similar recent hidden states
  - Layer normalization makes this kind of attention works much better
  - Hidden states are freed up to learn appropriate representation for the final classifier
  - Fast weights can be applied to solve recursive tasks without explicitly storing copies of hidden states