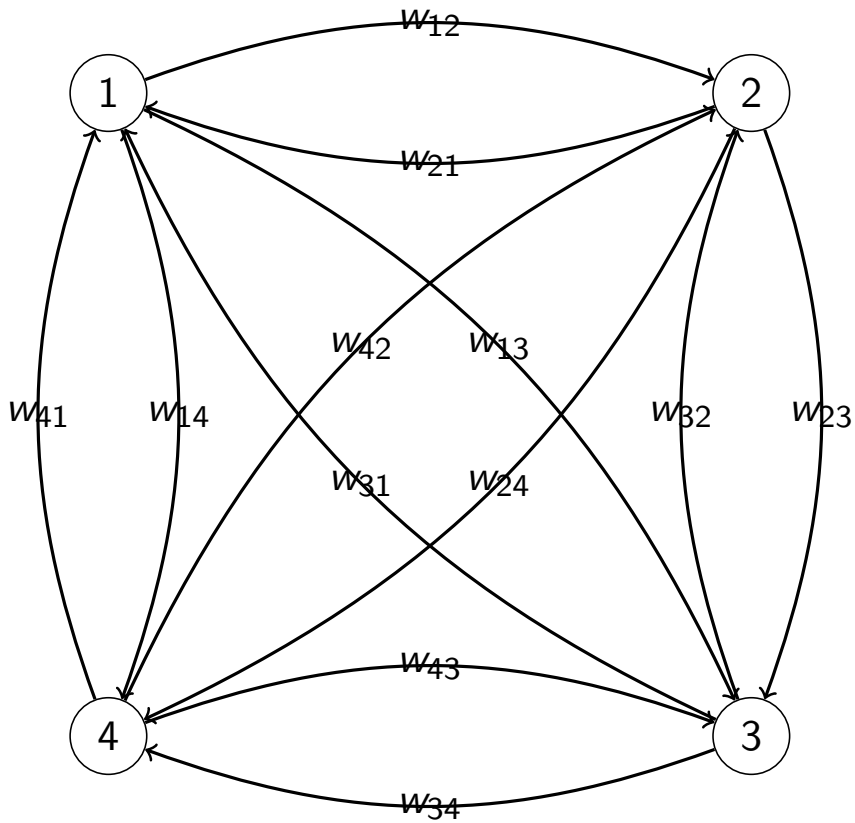
Recurrent Neural Networks

Nan Ye

School of Mathematics and Physics
The University of Queensland

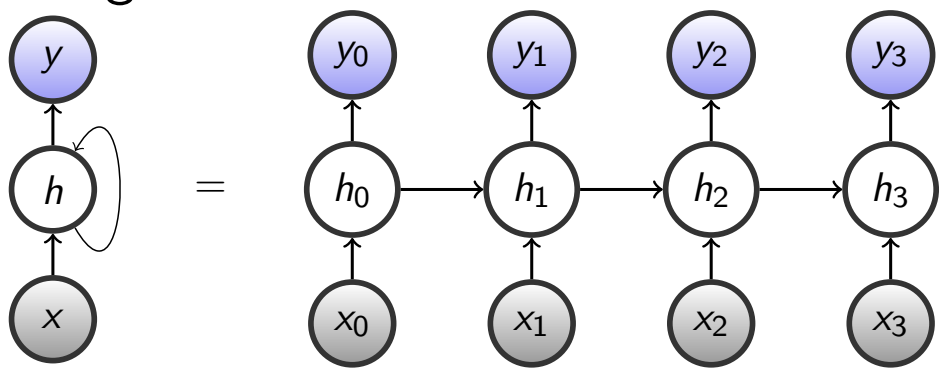
Feedforward and Recurrent Networks

- Neural networks can be broadly classified into two types: feedforward and recurrent.
- Feedforward: network is cycle-free
  - *e.g. Perceptron, Adaline, MLP, CNNs (good for array data)*
- Recurrent: network contains loops
  - *e.g. Hopfield network, and many models for sequence modelling*

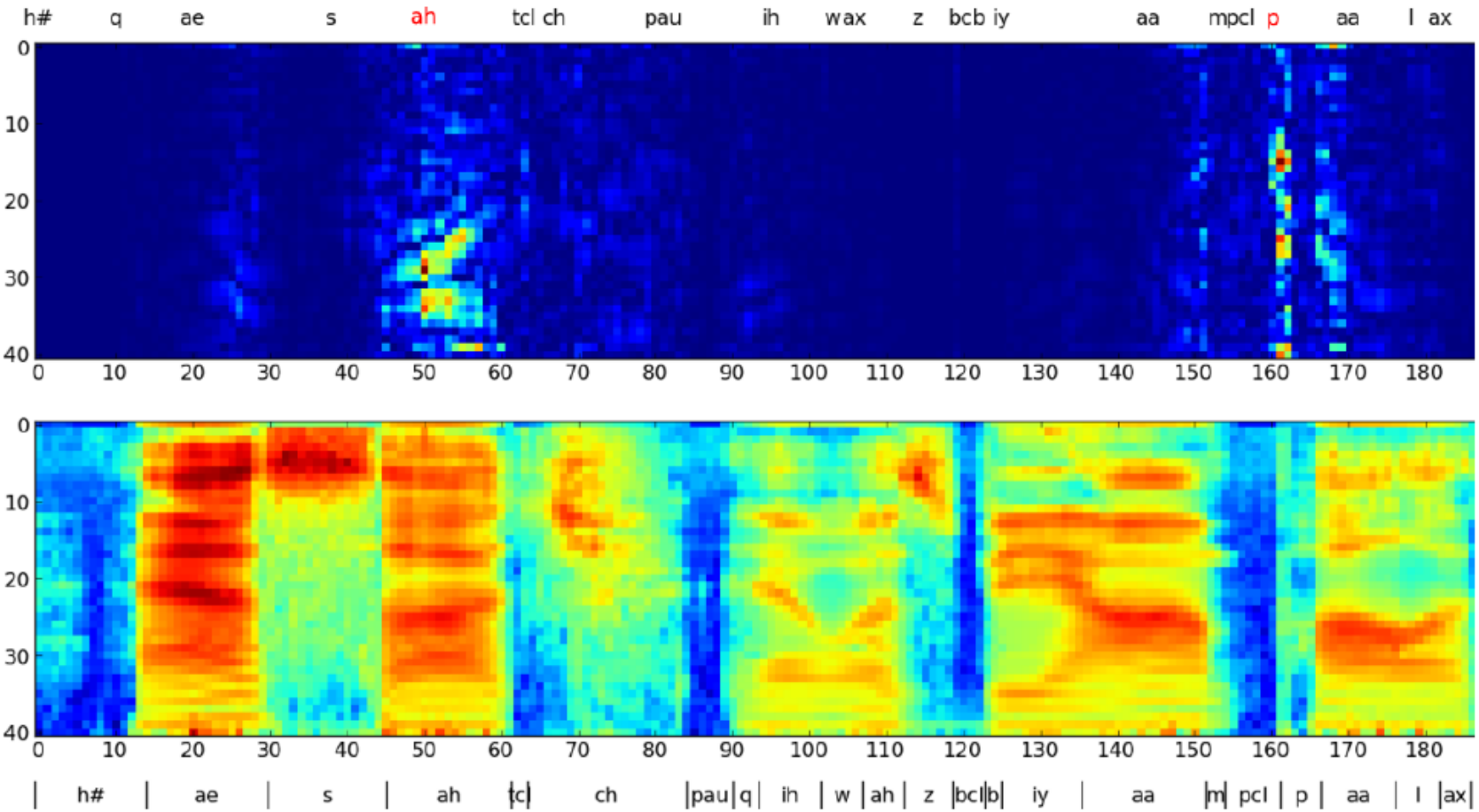


A Hopfield net contains many cycles

- We will consider recurrent neural networks (RNNs) for sequence modelling in this lecture
- E.g. the following basic RNN architecture

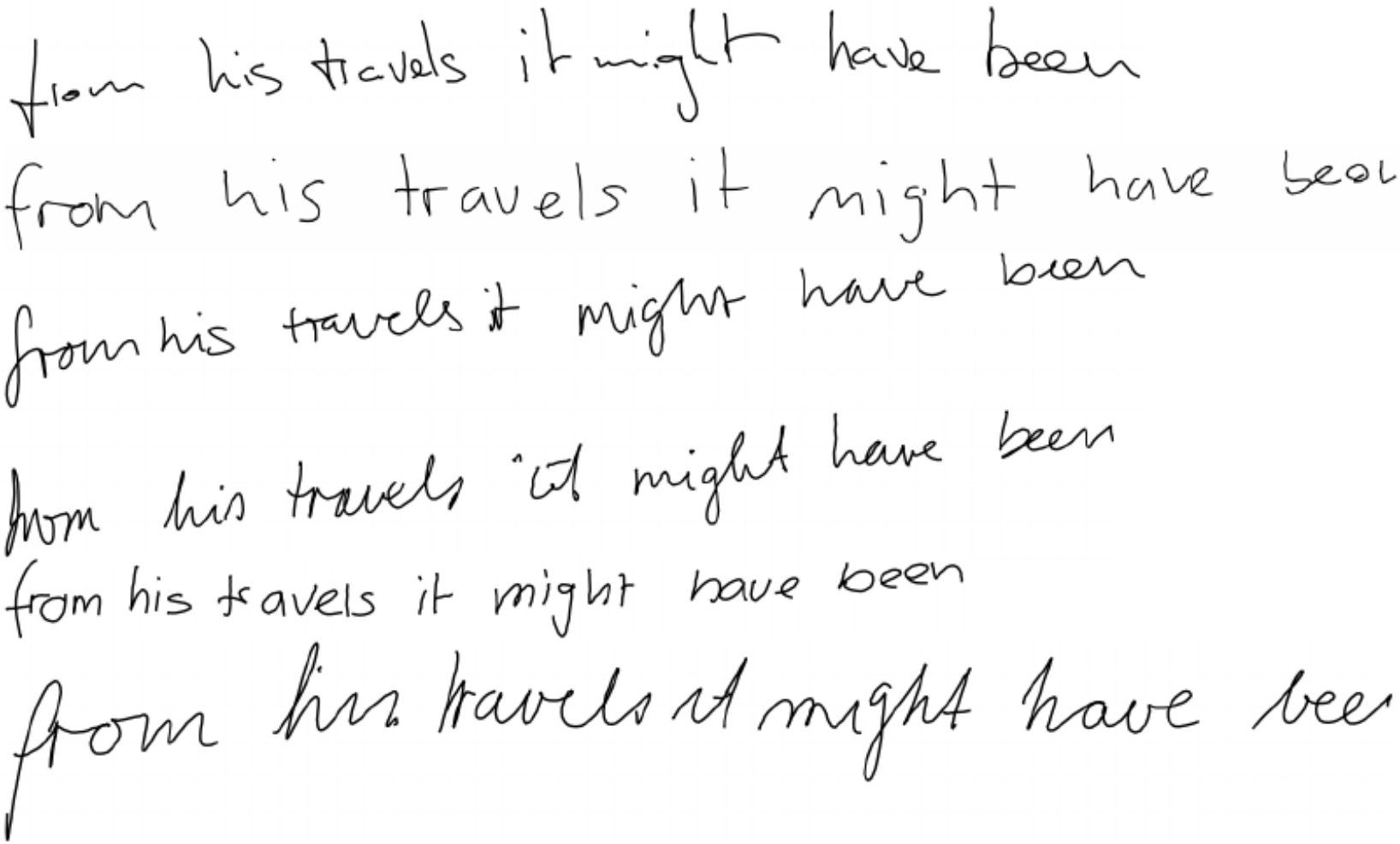


Applications



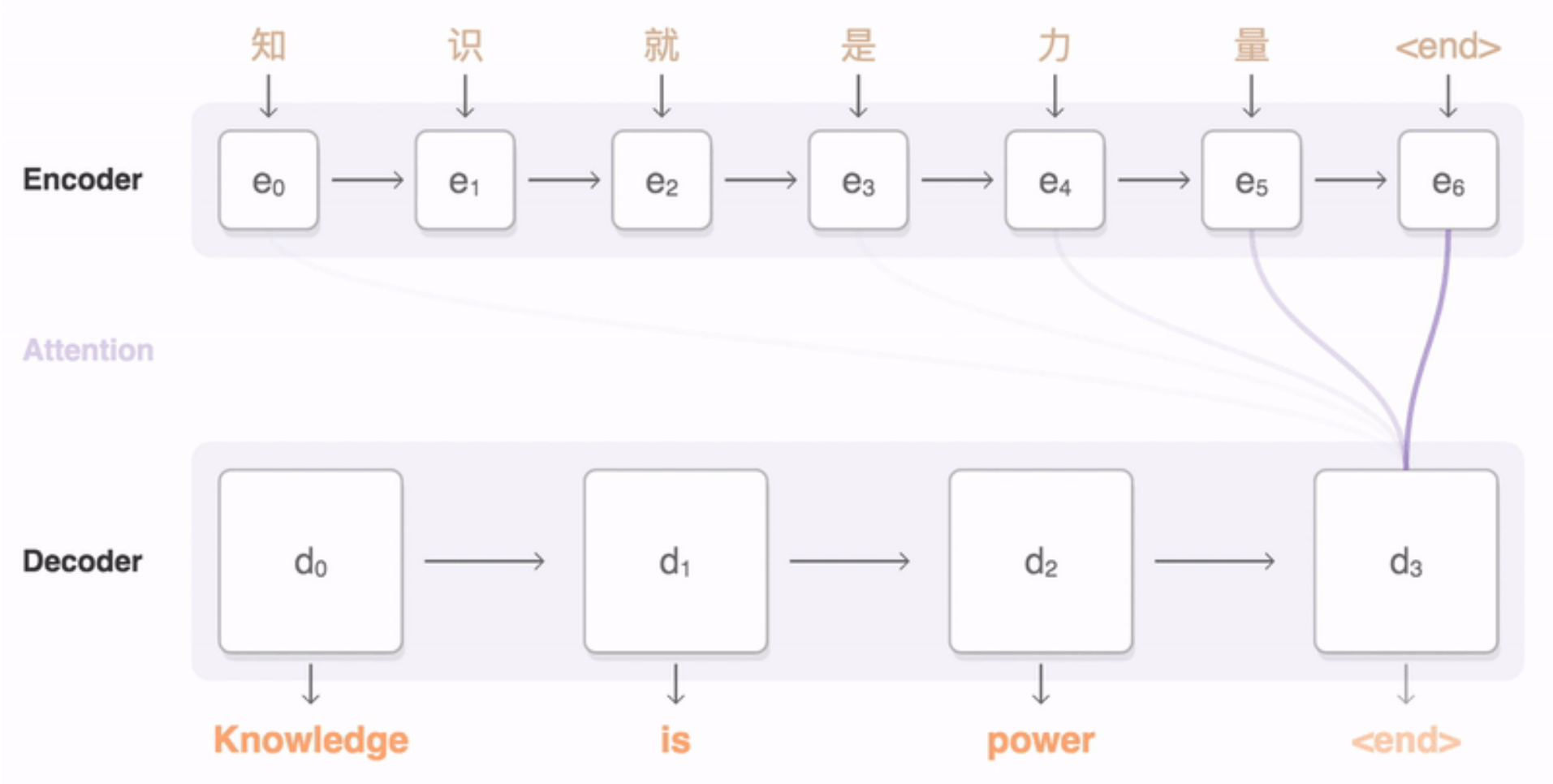
Phoneme recognition

Graves, Mohamed, and Hinton, Speech recognition with deep recurrent neural networks, 2013



Handwriting synthesis

Graves, Generating sequences with recurrent neural networks, 2013



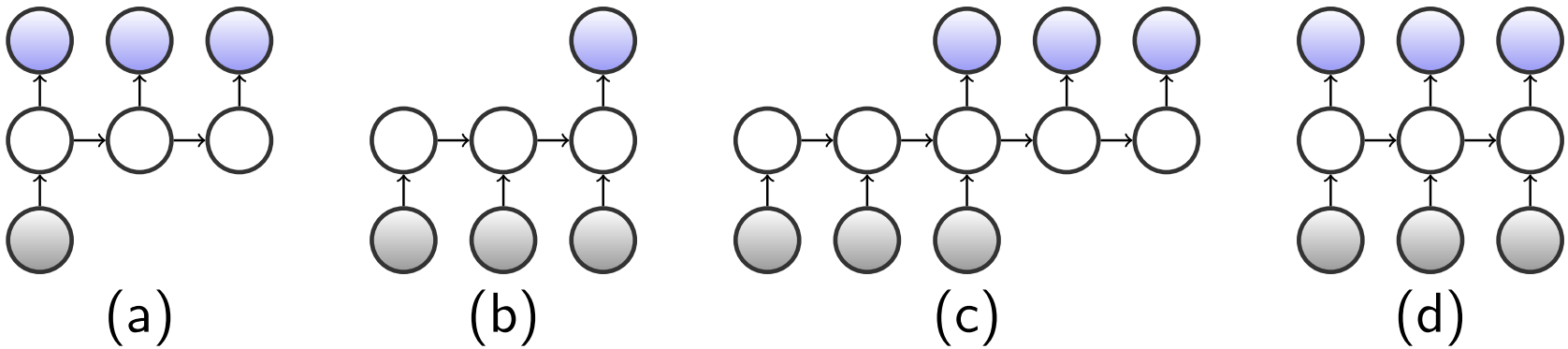
Machine translation

Britz, Goldie, Luong, and Le, Massive exploration of neural machine translation architectures, 2017

Image captioning

Vinyals, Toshev, Bengio, and Erhan, Show and tell: Lessons learned from the 2015 mscoco image captioning challenge, 2016

Sequence Modelling

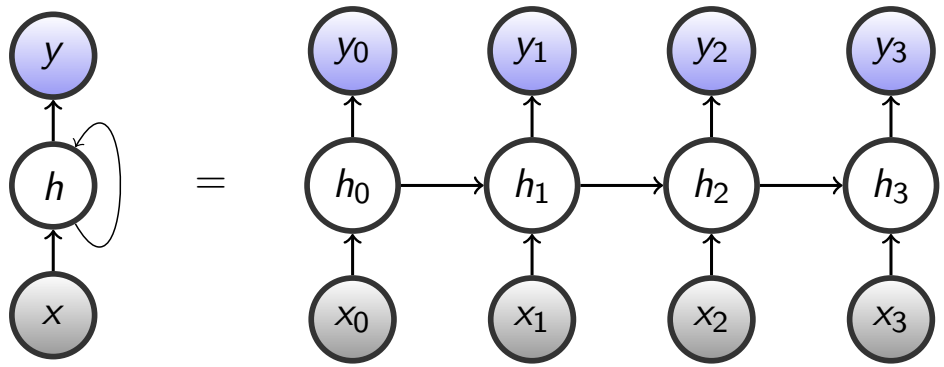


RNNs are good for various sequence modelling problems, including

- **(a)** One to many, e.g. image captioning
- **(b)** Many to one, e.g. video classification
- **(c)** Many to many, e.g. machine translation
- **(d)** Many to many, e.g. video frame classification

RNNs

- The states of hidden neurons in an RNN are updated at each time step.
- For finite sequences, RNNs can be *unfolded* as feedforward networks



- The slices at all time steps share the same parameters W

$$h_t = f_W(h_{t-1}, x_t),$$
$$y_t = g_W(h_t).$$

- Example: a simple RNN for time series prediction

$$h_t = \tanh(w_1 h_{t-1} + w_2 x_t + b),$$
$$y_t = w_3 h_t,$$

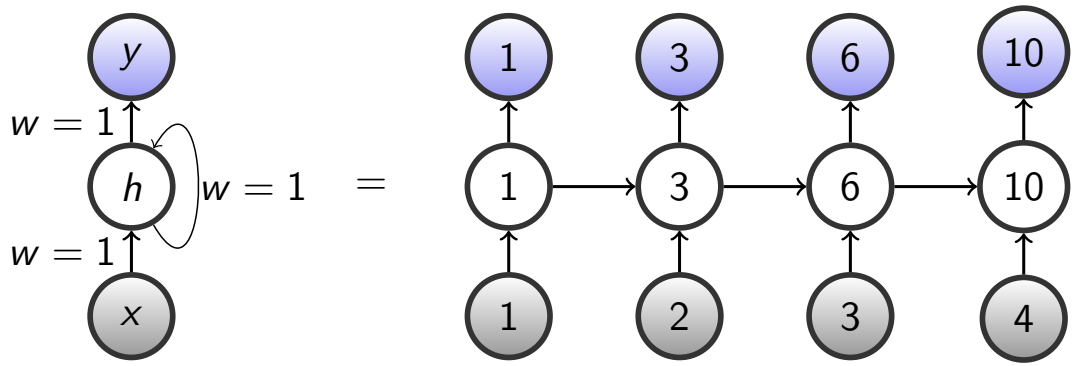
  where the output y_t predicts the next element x_{t+1} in the time series.
- An RNN can have much more complex structure for the hidden neurons (e.g. multiple layers).

Memory

- RNNs can remember information very far away in the past using the hidden states.
  - this is more biologically realistic than memoryless sequence models such as hidden Markov models, linear dynamical systems.
- This allows RNNs to represent very complex functions.
  - also make them hard to train
  - for many years, RNNs did not attract much interest because they are hard to train

An RNN for Summing a Sequence

- The RNN below computes the sum of numbers seen so far



- x is the current input, h is the sum of all seen numbers, and y is the output ($= h$). Activations are identity.
- The network has been unfolded as a feedforward network with

$$h_t = x_t + h_{t-1},$$
$$y_t = h_t.$$

Program Evaluation with RNNs

- Zaremba and Sutskever, 2014 showed that an RNN can execute (very) simple short program.
- Sample results

Input:

```
print(6652).
```

Target:	6652.
"Baseline" prediction:	6652.
"Naive" prediction:	6652.
"Mix" prediction:	6652.
"Combined" prediction:	6652.

Input:

```
print((16*3071)).
```

Target:	49136.
"Baseline" prediction:	49336.
"Naive" prediction:	48676.
"Mix" prediction:	57026.
"Combined" prediction:	49626.

Input:

```
c=2060;
print((c-4387)).
```

Target:	-2327.
"Baseline" prediction:	-2320.
"Naive" prediction:	-2201.
"Mix" prediction:	-2377.
"Combined" prediction:	-2317.

Input:

```
e=1079
for x in range(10):e+=4729
print(e).
```

Target:	48369.
"Baseline" prediction:	48017.
"Naive" prediction:	48011.
"Mix" prediction:	48101.
"Combined" prediction:	48009.

Zaremba and Sutskever, Learning to execute, 2014

Backpropagation Through Time (BPTT)

- For gradient-based learning, we need to compute the gradient.
- To compute the gradient for an RNN, we can unfold it as a layered feed-forward network with weight equality constraints.
- BPTT is a modification of backprop that takes these constraints into account.

BPTT

- Perform forward pass on the unfolded network
- Treat all the weights in the unfolded network as distinct weights, and run usual backprop to compute derivatives wrt to them
- For each weight, set its derivative to be the sum of the derivatives of its copies at different time steps

Why does BPTT work?

- Consider a single weight w, assume that it is repeated for T steps, and $w^{(t)}$ is its copy at time t.
- Assume the objective function is $f(w^{(1)}, \ldots, w^{(T)})$, where other weights unrelated to w have been suppressed in the notation.
- Using the chain rule, we have

$$\begin{aligned}\frac{\partial f}{\partial w} &= \frac{\partial f}{\partial w^{(1)}}\frac{\partial w^{(1)}}{\partial w} + \ldots + \frac{\partial f}{\partial w^{(T)}}\frac{\partial w^{(T)}}{\partial w} \\ &= \frac{\partial f}{\partial w^{(1)}} + \ldots + \frac{\partial f}{\partial w^{(T)}}.\end{aligned}$$

Exploding and Vanishing Gradients

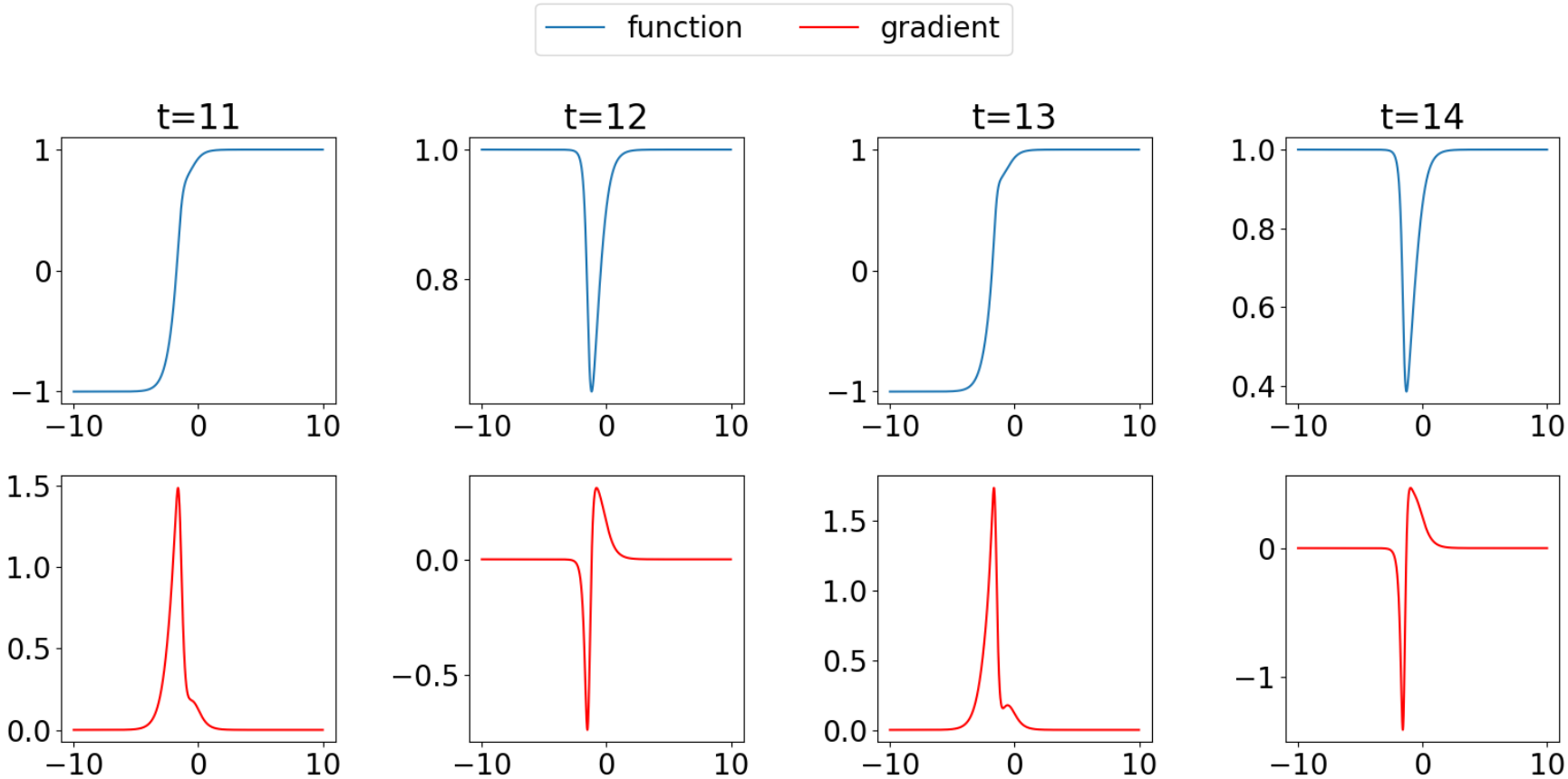
- For vanilla RNN, training can be difficult due to exploding gradients or vanishing gradients for long sequence.
  - Exploding gradient: gradient grows very fast in time steps.
  - Vanishing gradient: gradient vanishes very fast in in time steps.
- These problems happen for deep feedforward networks.
  - recall: GoogLeNet uses auxiliary classifiers to solve this problem.

Example

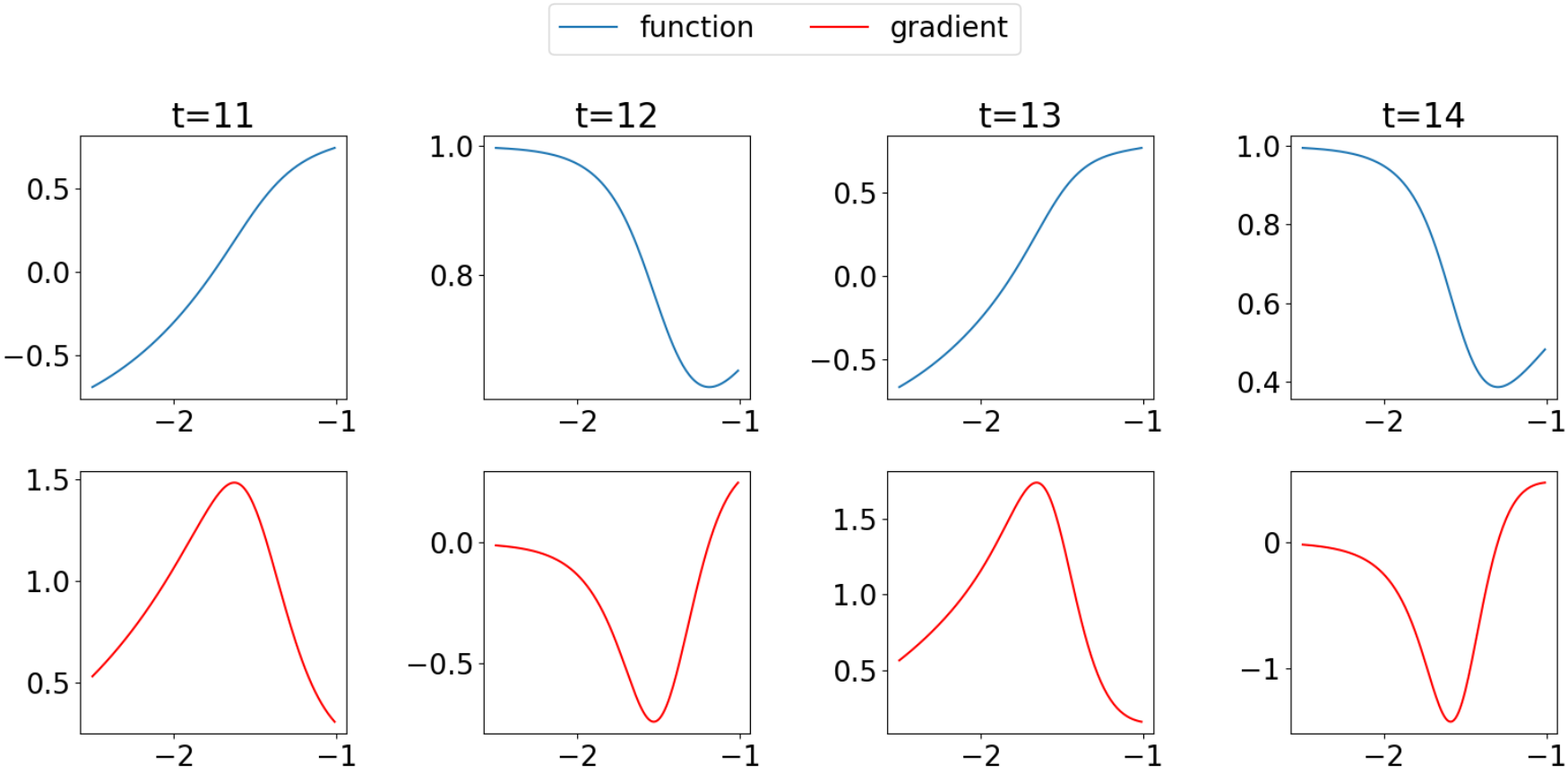
- Consider the following simple RNN

$$h_t = w_1 \tanh(h_{t-1}) + w_2 x_t + b,$$
$$o_t = \tanh(h_t),$$

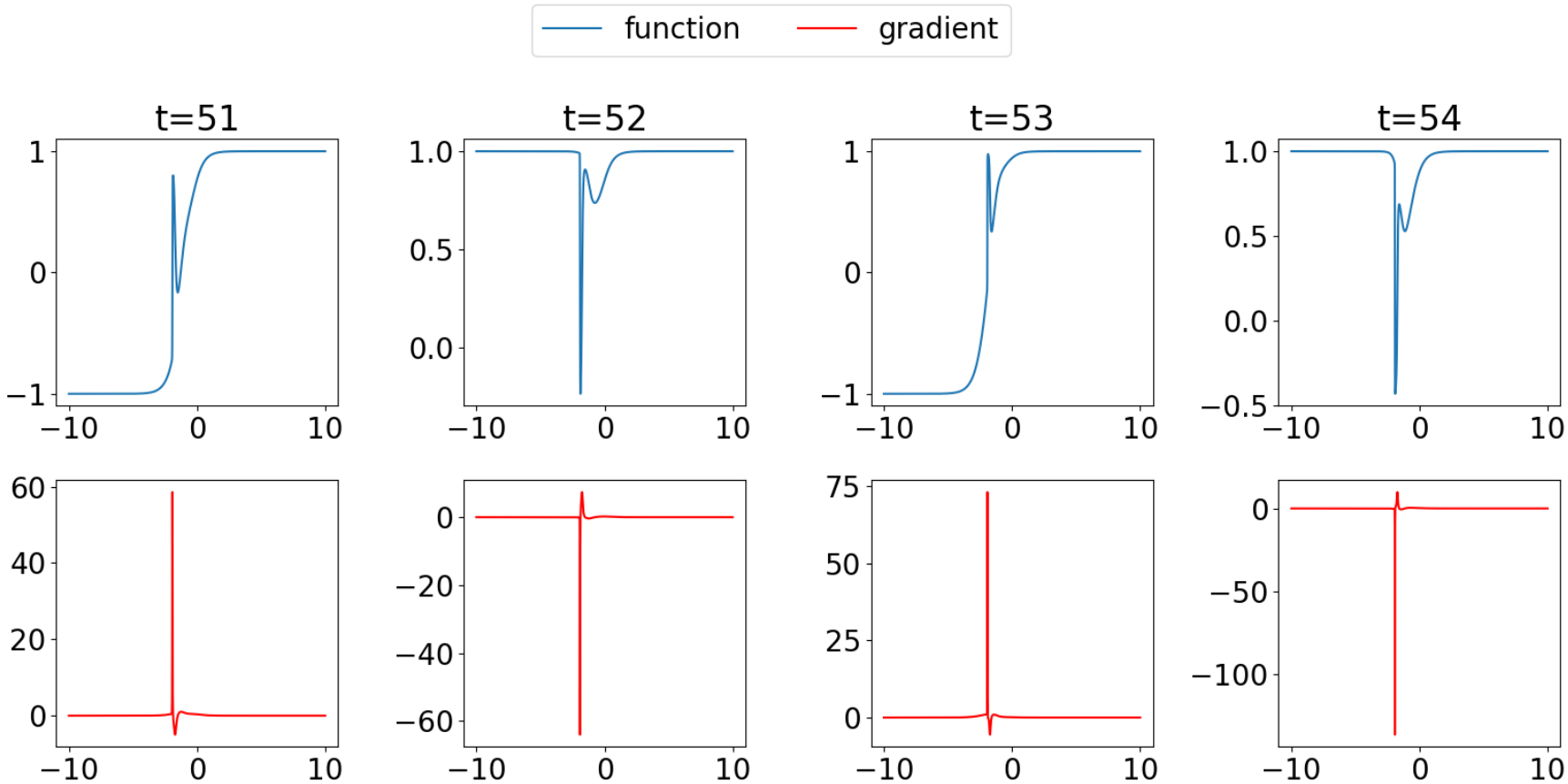
  where all input x's, hidden h's, and output o's are real numbers.
- We only want to predict a target $y_t \in \{-1, 1\}$ correctly for some t by minimizing $L = \frac{1}{2}(y_t - o_t)^2$.
- We consider how the partial derivative $\frac{\partial L}{\partial w_1} = (o_t - y_t)\frac{\partial o_t}{\partial w_1}$ changes as t increases.
- The main term is $\frac{\partial o_t}{\partial w_1}$.



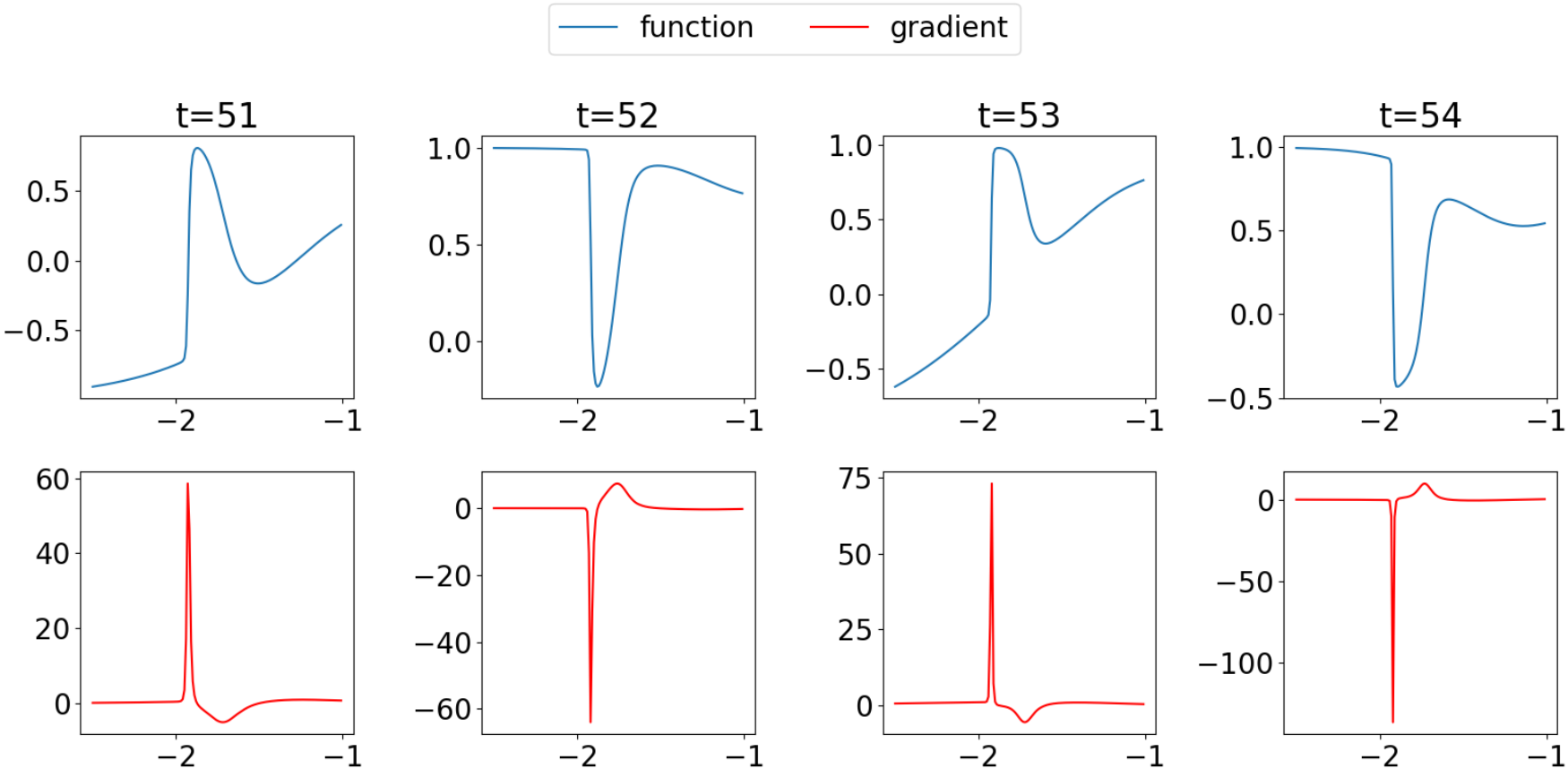
- The range of o_t and $\frac{\partial o_t}{\partial w_1}$ are small for small t's.
- However, there are regions where the gradients are vanishing.



A zoom-in view



Gradients are exploding!



A zoom-in view

Explaining Exploding/Vanishing Gradients

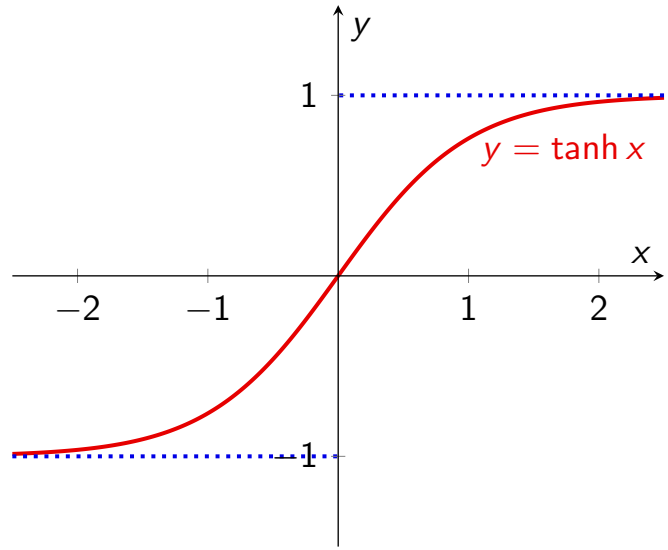
- We don't have a precise explanation for why/when gradients explode/vanish in general.
- We consider the simple RNN example to get some insights on this.

$$h_t = w_1 \tanh(h_{t-1}) + w_2 x_t + b,$$
$$o_t = \tanh(h_t),$$

- We need quite a bit of mathematical calculations even for this simple case.

The hyperbolic tangent

- The hyperbolic tangent $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ is an S shaped curve.



- It is bounded in the range [-1,1]
- Its derivative satisfies $\tanh(x)' = 1 - \tanh(x)^2$.

A formula for the gradient

- Since $o_t = \tanh(h_t)$, we have $\frac{\partial o_t}{\partial w_1} = (1 - o_t^2)\frac{\partial h_t}{\partial w_1}$.
- Let $g_t = \frac{\partial h_t}{\partial w_1}$, then

$$
\begin{aligned}
& & h_t &= w_1 \overbrace{\tanh(h_{t-1})}^{o_{t-1}} + w_2 x_t + b, \\
\Rightarrow & & g_t &= o_{t-1} + w_1(1 - o_{t-1}^2) g_{t-1}, \qquad t \geq 1,
\end{aligned}
$$

  with $g_0 = 0$.

- This gives us a kind of a geometric series representation for g_t

$$
\begin{aligned}
g_t &= o_{t-1} + w_1(1 - o_{t-1}^2) o_{t-2} + w_1(1 - o_{t-1}^2) w_1(1 - o_{t-2}^2) g_{t-2} \\
&= \sum_{i=0}^{t-1} o_i \prod_{j=i+1}^{t-1} \left(w_1(1 - o_j^2)\right)
\end{aligned}
$$

How can the gradient vanish

- The gradient can vanish when w_1 is very large
  - All o_i's will be close to -1 or 1.
  - In addition, $w_1(1 - o_j^2)$ will be very small (roughly $4w_1/e^{2|w_1 o_{i-1} + w_2 x_{j-1} + b|}$).
  - Thus the gradient is very small.
- This has nothing to do with depth — the cause is in the tanh activation.

How can the gradient explode

- The gradient can explode when $|w_1| > 1$ is moderately large, and o_j are small.
- When the depth is large, the exponential dependence of g_t on w_1 causes the gradient to grow very quickly.
- In general, the backprop formula reveals such exponential dependence for feedforward network, and as long as the derivatives of the activations are bounded away from 0 when the weights have absolute values more than 1, the gradient explodes.

Remarks

- The tanh activation is a main cause for the vanishing gradient problem.
- Gradients have an exponential dependence on the depth in general, and this is a main cause for the exploding gradient problem.
- It is hard to tell exactly when/why gradient gradient explodes/vanishes even for a simple RNN.

Your Turn

Which of the following statement is correct? (Multiple choice)

(a) We can design an RNN to calculate sequence sums.

(b) We can design an RNN to calculate the exponential moving averages of a sequence of numbers.

(c) RNNs are easier to train than MLPs.

(d) When training an RNN, the weights are often initialised to small random numbers.

What You Need to Know...

- RNNs are useful for solving various sequence modelling problems.
- Simple RNNs can be unfolded as a feedforward neural network.
  - This leads to BPTT.
- RNNs are prone to the exploding and vanishing gradients problem, with both the choice of activation function and depth playing a role here.