Backpropagation

What is backpropagation?

An algorithm for numerically evaluating exact derivatives of a function

Problem: differentiate $f(x) = x^2 \quad \text{at } x = 2$

Roadmap: how does this get harder?

1. Intermediate processing

chain rule

2. Many inputs/outputs

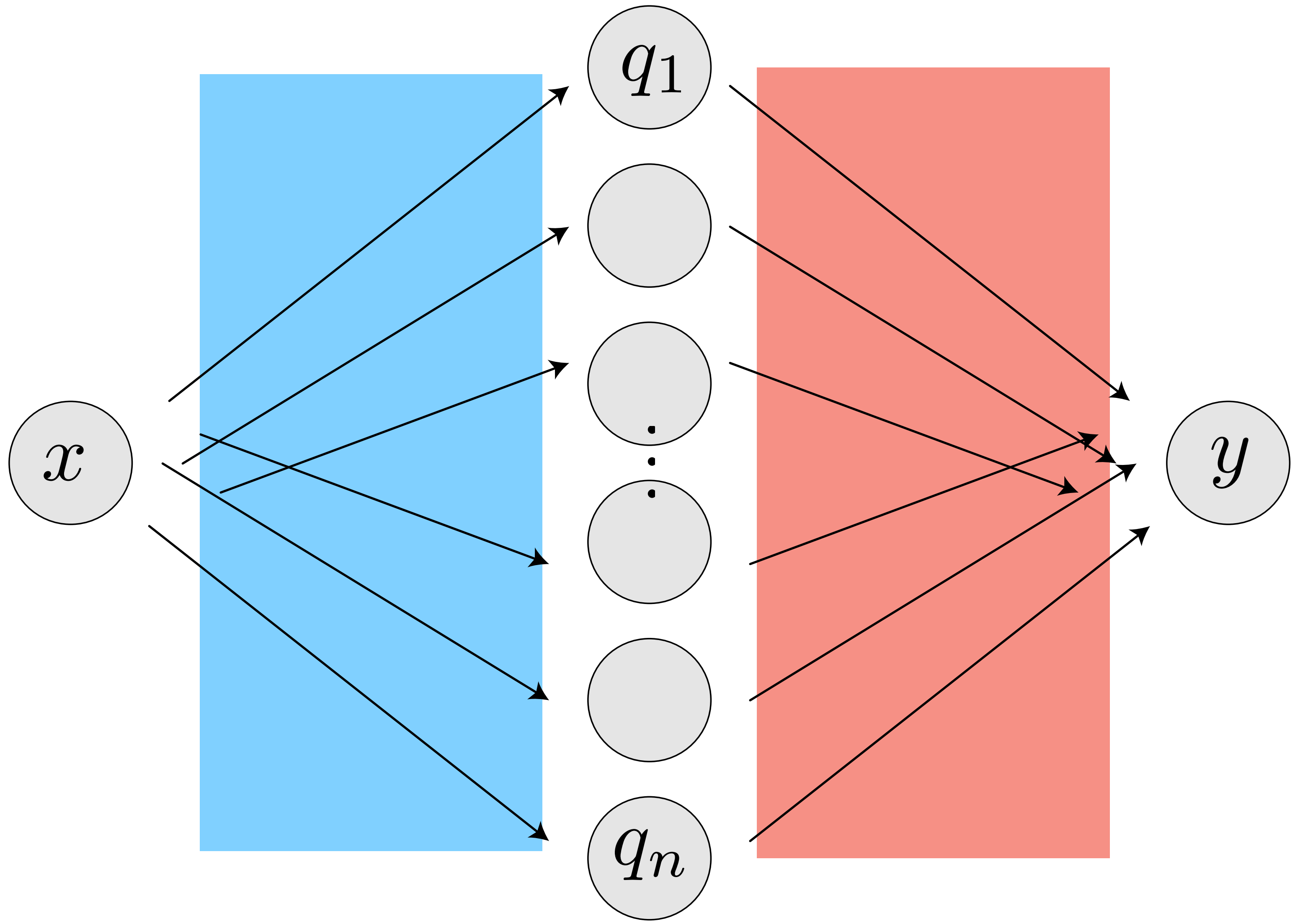
matrix operations

Key facts

$$y = W_1 x_1 + W_2 x_2 + W_3 x_3$$

$$\frac{\partial y}{\partial x_j} = W_j$$

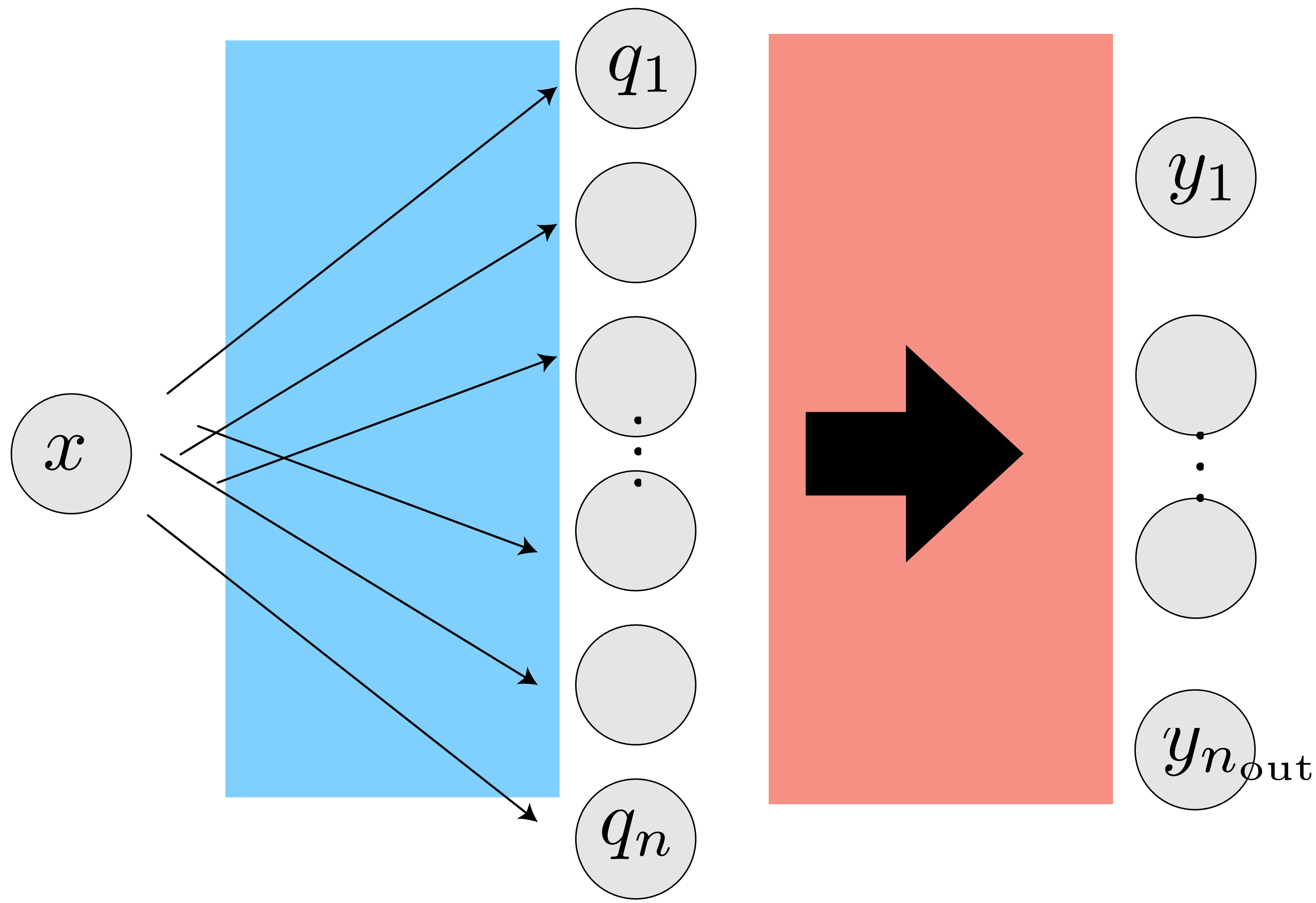
Multivariate chain rule



$$\frac{\partial y}{\partial x} = \sum_{k=1}^{n} \frac{\partial y}{\partial q_k} \frac{\partial q_k}{\partial x}$$

"dot product"

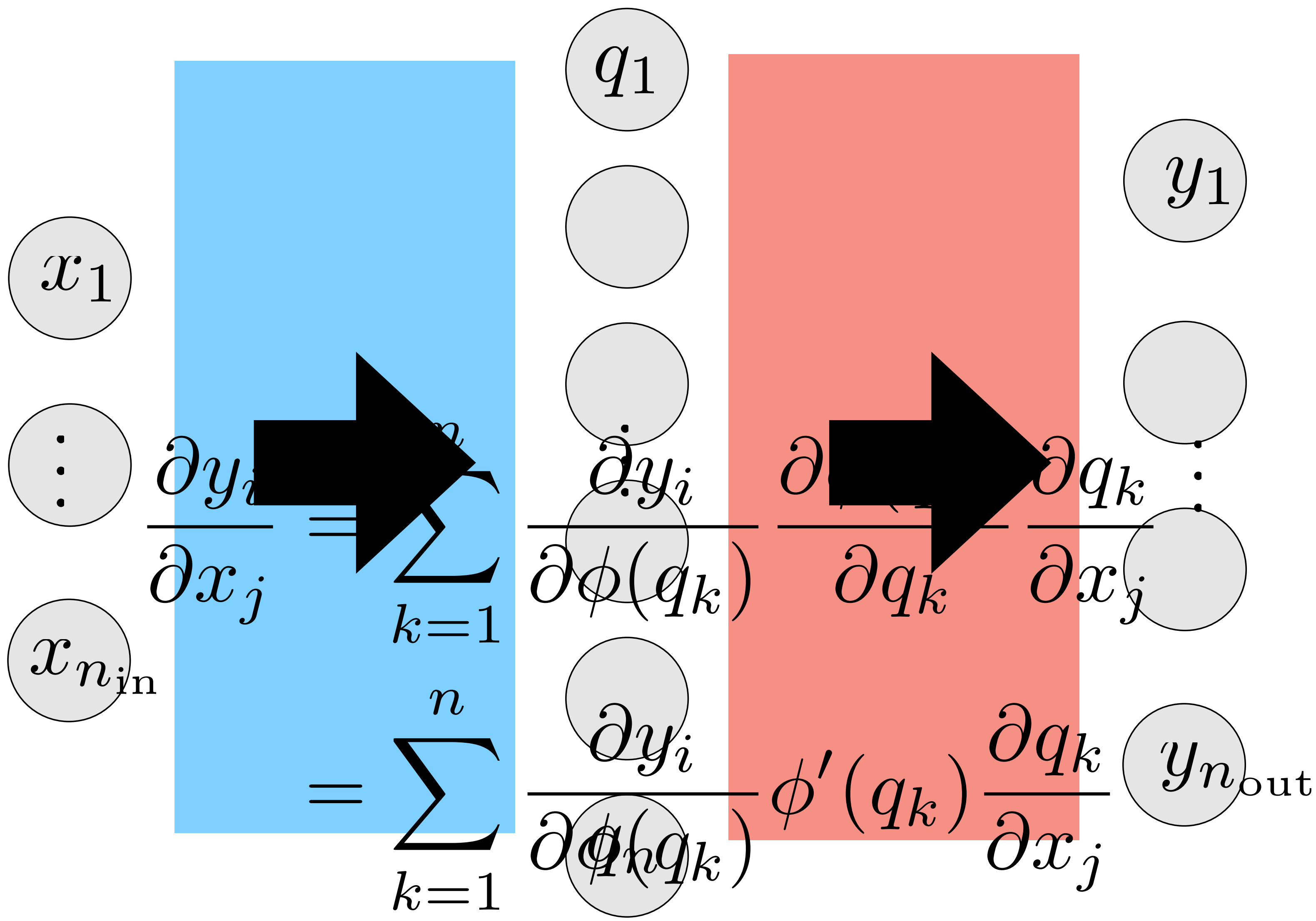
Many output, one input



$$\frac{\partial y_i}{\partial x} = \sum_{k=1}^{n} \frac{\partial y_i}{\partial q_k} \frac{\partial q_k}{\partial x}$$

matrix-vector product

Many input, many output



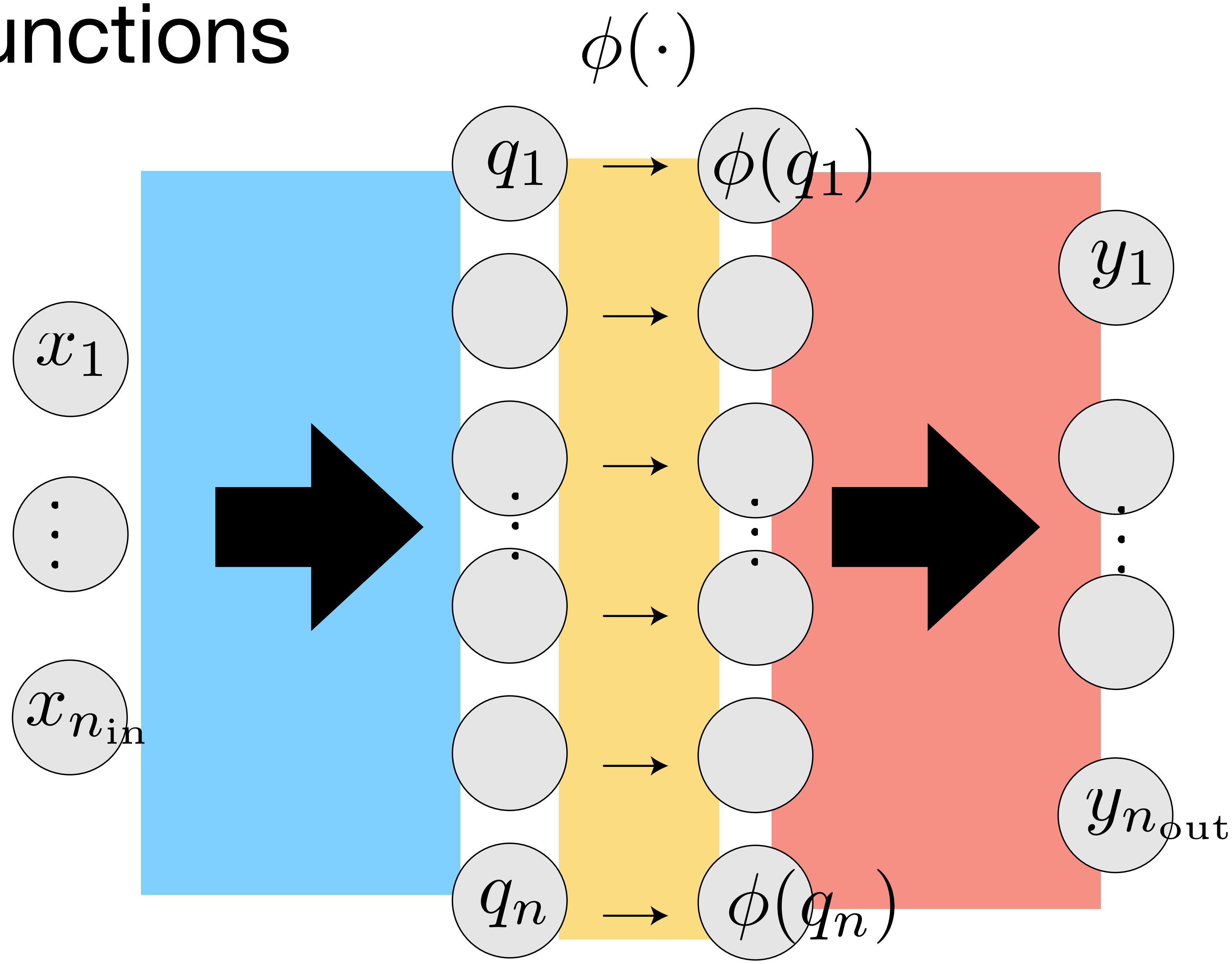
$$\frac{\partial y_i}{\partial x_j} = \sum_{k=1}^{n} \frac{\partial y_i}{\partial \phi(q_k)} \frac{\partial \phi(q_k)}{\partial q_k} \frac{\partial q_k}{\partial x_j}$$

$$= \sum_{k=1}^{n} \frac{\partial y_i}{\partial \phi(q_k)} \phi'(q_k) \frac{\partial q_k}{\partial x_j}$$

$$\frac{\partial y_i}{\partial x_j} = \sum_{k=1}^{n} \frac{\partial y_i}{\partial q_k} \frac{\partial q_k}{\partial x_j}$$

matrix-matrix product

Element-wise functions



$$\frac{\partial y_i}{\partial x_j} = \sum_{k=1}^{n} \frac{\partial y_i}{\partial \phi(q_k)} \frac{\partial \phi(q_k)}{\partial q_k} \frac{\partial q_k}{\partial x_j}$$

$$= \sum_{k=1}^{n} \frac{\partial y_i}{\partial \phi(q_k)} \phi'(q_k) \frac{\partial q_k}{\partial x_j}$$

matrix-matrix product
ft. twist (rescale columns)

Exercise: take this derivative

$\text{Input } x^{(0)}$ (assume all vectors of dimensionality n, matrices n x n)

$$h_i^{(1)} = \sum_j W_{ij}^{(1)} x_j^{(0)}$$

$$a_i^{(1)} = \phi(h_i^{(1)})$$

$$h_i^{(2)} = \sum_j W_{ij}^{(2)} a_j^{(1)}$$

$$a_i^{(2)} = \phi(h_i^{(2)})$$

$$L = \sum_i (a_i^{(2)} - a_i^{*(2)})^2$$

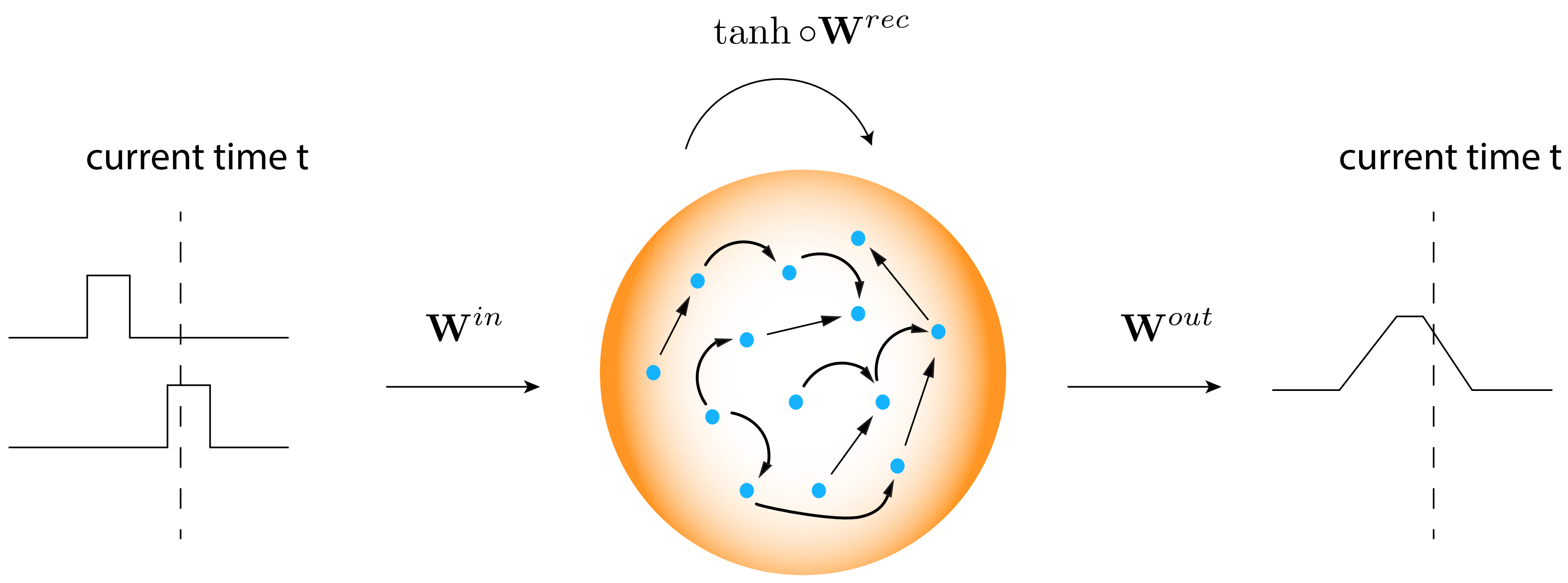
$$\frac{\partial L}{\partial W_{ij}^{(1)}} = ??$$

More generally…

Backprop is an algorithm for calculating any derivative of a directed acyclic computation graph

Works by starting at the outer-most derivative, applying the chain rule backwards and calculate **node** derivatives which can be used to read out the **parameter** derivatives that are ultimately of interest

$$\frac{\partial L}{\partial W_{ij}^{(1)}} \qquad \frac{\partial L}{\partial a_i^{(1)}}$$



$\tanh \circ \mathbf{W}^{rec}$
current time t
$\mathbf{W}^{in}$
$\mathbf{W}^{out}$
current time t

More abstractly…

<u>Recurrence</u>

$$\mathbf{a}^{(t)} = F_{\mathbf{w}}(\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)})$$

All “recurrent” trainable parameters, in particular $\mathbf{W}, \mathbf{W}^{\mathrm{in}}, \mathbf{b}^{\mathrm{rec}}$

<u>Output</u>

$$\mathbf{y}^{(t)} = F^{\mathrm{out}}_{\mathbf{w}_o}(\mathbf{a}^{(t)})$$

So the equations from before are a particular case of F_W, a “vanilla” RNN

$$h_i^{(t)} = \sum_{j=1}^{n} W_{ij}\hat{a}_j^{(t-1)} + \sum_{l=1}^{n_{in}} W^{\text{in}}_{il} x_l^{(t)} + b_i^{\text{rec}}$$

$$\hat{\mathbf{a}}^{(t-1)} = \text{concat}(\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}, 1)$$

$$a_i^{(t)} = \phi(h_i^{(t)})$$

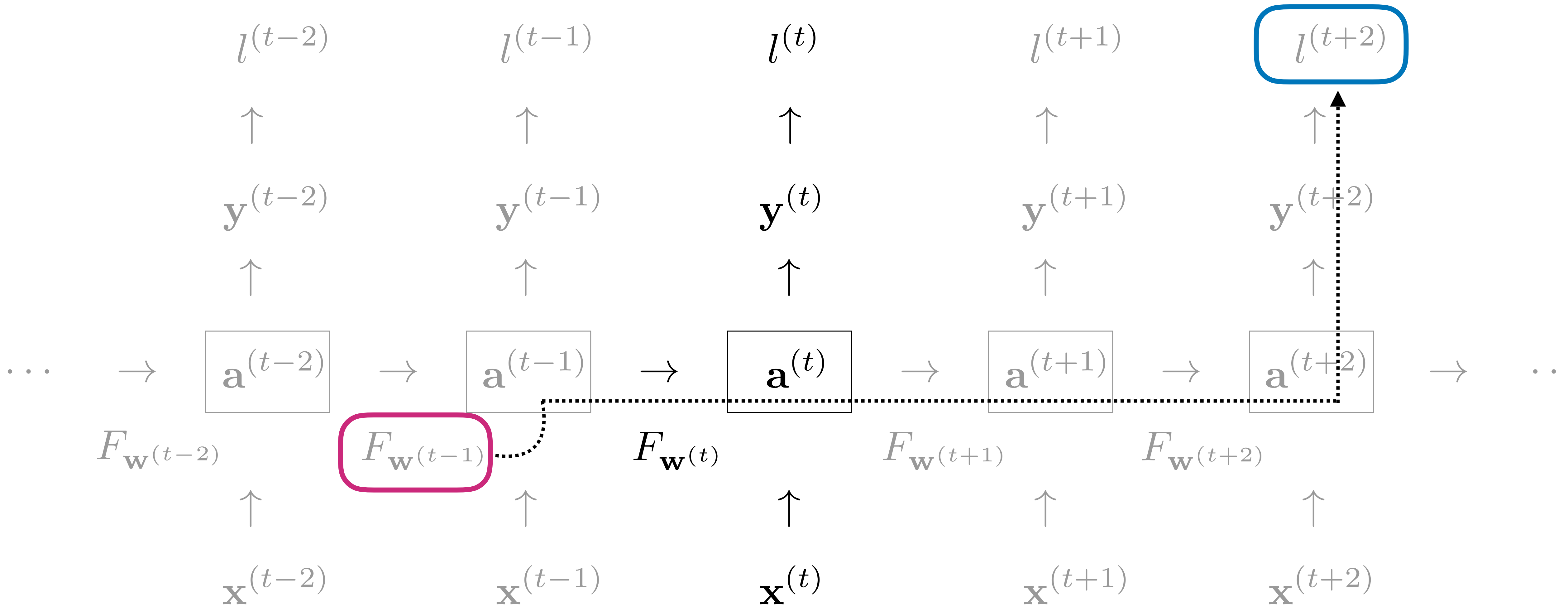
"Unrolling" a network

$$\begin{array}{ccccccccccccc}
\cdots & \rightarrow & \boxed{\mathbf{a}^{(t-2)}} & \rightarrow & \boxed{\mathbf{a}^{(t-1)}} & \rightarrow & \boxed{\mathbf{a}^{(t)}} & \rightarrow & \boxed{\mathbf{a}^{(t+1)}} & \rightarrow & \boxed{\mathbf{a}^{(t+2)}} & \rightarrow & \cdots \\
 & F_{\mathbf{w}^{(t-2)}} & & F_{\mathbf{w}^{(t-1)}} & & F_{\mathbf{w}^{(t)}} & & F_{\mathbf{w}^{(t+1)}} & & F_{\mathbf{w}^{(t+2)}} & & & \\
 & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \\
 & & \mathbf{x}^{(t-2)} & & \mathbf{x}^{(t-1)} & & \mathbf{x}^{(t)} & & \mathbf{x}^{(t+1)} & & \mathbf{x}^{(t+2)} & &
\end{array}$$

Gradient descent

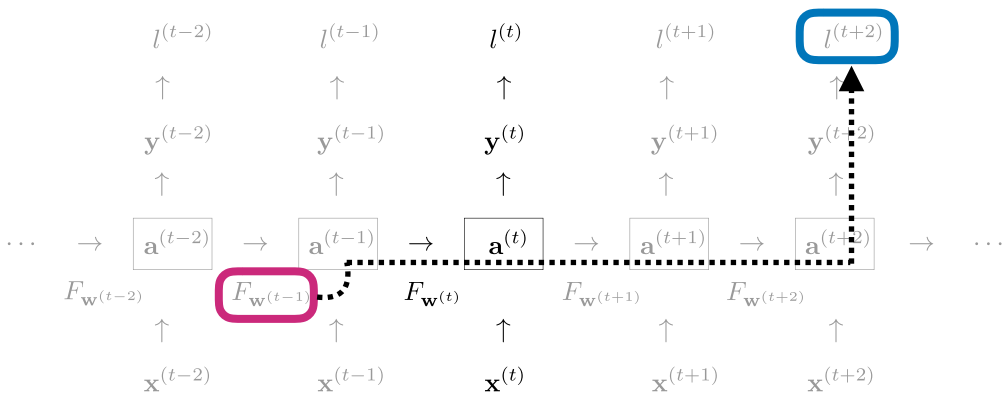
Global loss function $\mathcal{L} = \sum_t l^{(t)}$

Gradient of loss w.r.t. w



$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_t \frac{\partial l^{(t)}}{\partial \mathbf{w}}$$

Key ingredients for calculating $\dfrac{\partial l^{(t)}}{\partial \mathbf{w}^{(s)}}$



$$a_i^{(t)} = \phi(h_i^{(t)}) \qquad h_i^{(t)} = \sum_j W_{ij} \hat{a}_j^{(t-1)}$$

$$l^{(t)} = \frac{1}{2} \sum_k \Big(\sum_i W_{ki}^{\text{out}} a_i^{(t)} - y_k^{*(t)}\Big)^2$$

$$\overline{M}_{kij}^{(t)} = \frac{\partial a_k^{(t)}}{\partial W_{ij}}$$

$$J_{ij}^{(t)} = \frac{\partial a_i^{(t)}}{\partial a_j^{(t-1)}}$$

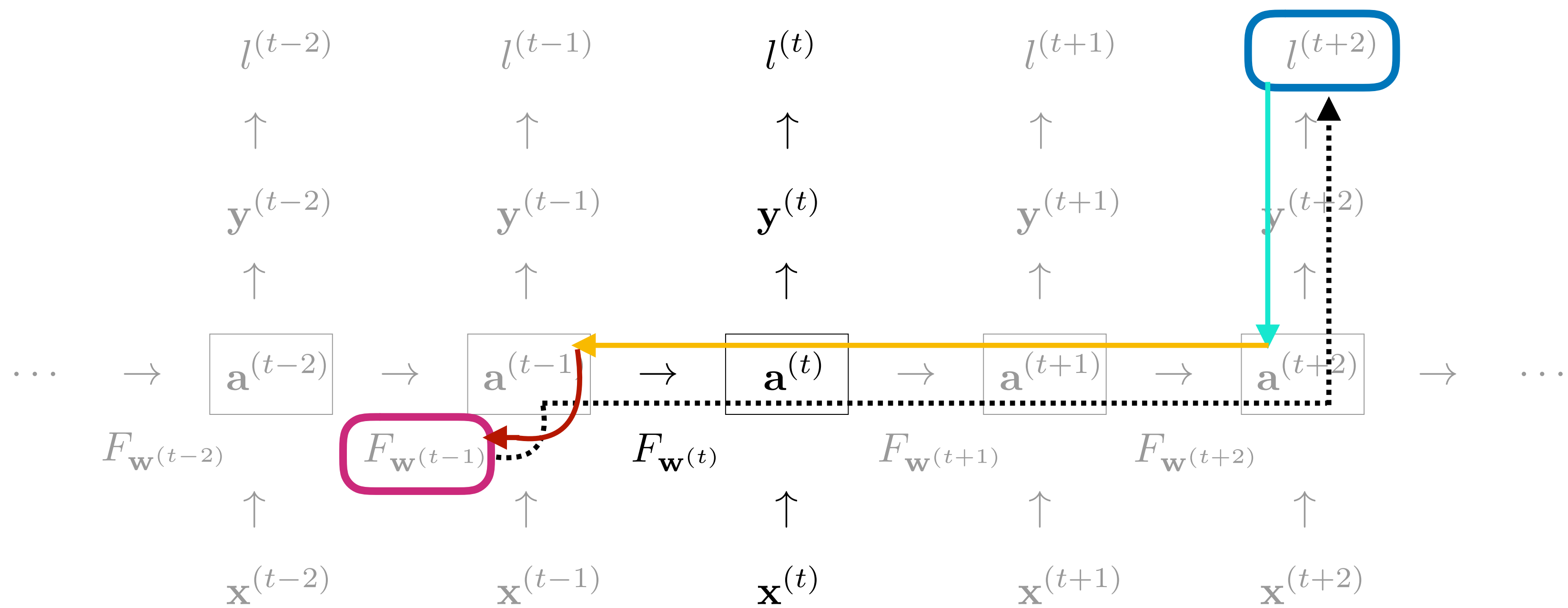
$$q_i^{(t)} = \frac{\partial l^{(t)}}{\partial a_i^{(t)}}$$

Exercise: write expression for each derivative

Backpropagation through time

1. Unroll the graph for some number of t and s

2. Calculate each instance of $\frac{\partial l^{(t)}}{\partial \mathbf{w}^{(s)}}$ going **backwards** from the losses



$$\frac{\partial l^{(t+2)}}{\partial W_{ij}^{(s-1)}} = \mathbf{q}^{(t+2)}\mathbf{J}^{(t+2)}\mathbf{J}^{(t+1)}\mathbf{J}^{(t)}\overline{\mathbf{M}}_{ij}$$

Real-time recurrent learning

1. Keep track of influence matrix **M** that encodes collective effect of all past applications of **w** on **a**

2. Use **M** to calculate gradient