Neural Networks I

Outline:

1. Perceptrons
2. The capacity problem
3. Feedforward networks and their training
4. Recurrent networks

Perceptrons

The simplest network:

N inputs $\{x_i\}$, 1 output O, connection weights J_i

$$O = \mathrm{sgn}\left(\sum_i J_i x_i\right) = \mathrm{sgn}\left(\mathbf{J}\cdot\mathbf{x}\right)$$ ("threshold unit")

Could have multiple outputs, but each one would then be an independent problem)

Could have a "bias": $O = \mathrm{sgn}\left(\sum_i J_i x_i + b\right) = \mathrm{sgn}\left(\mathbf{J}\cdot\mathbf{x} + \mathbf{b}\right)$

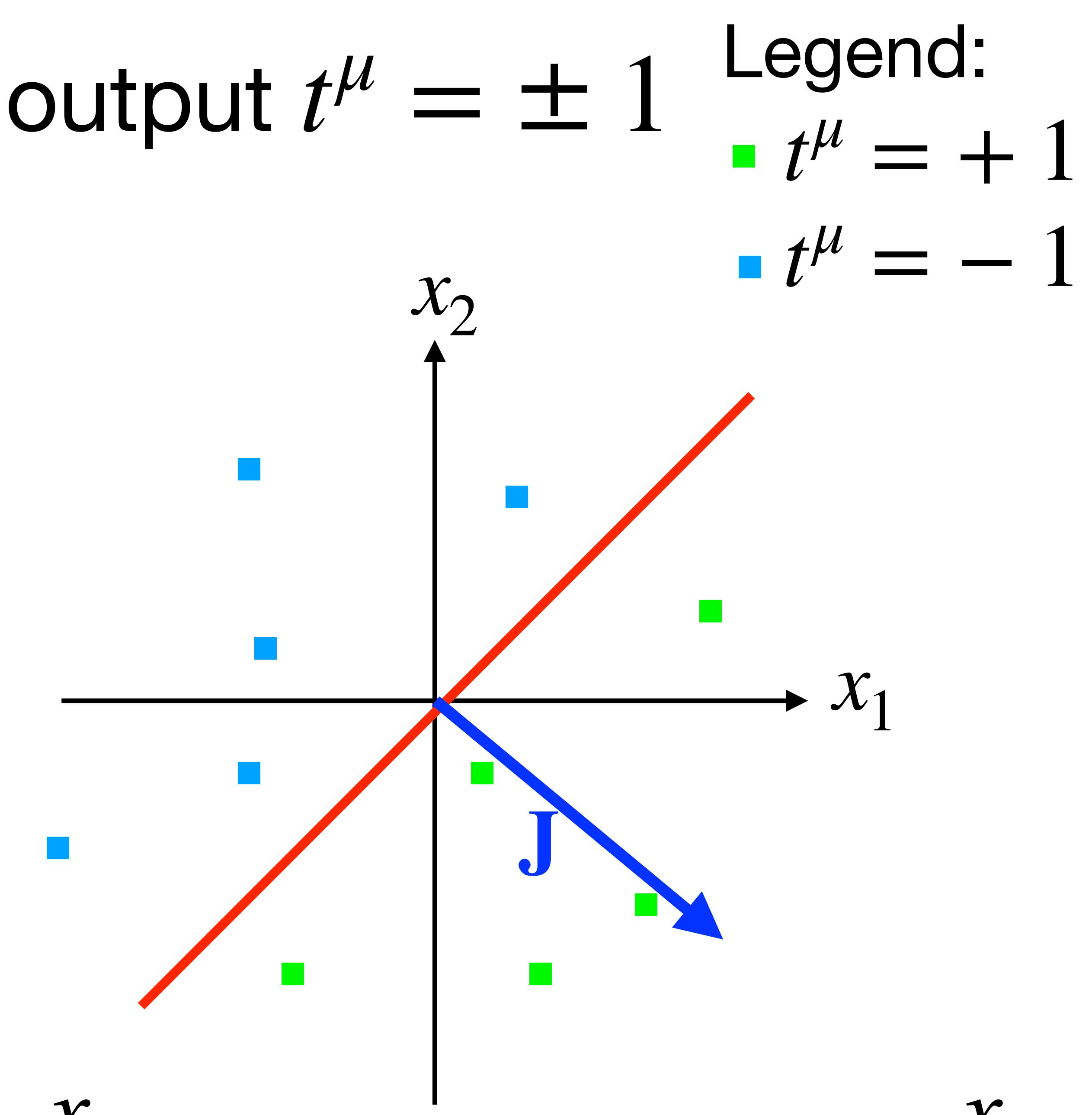
but can represent that by just adding an input $x_0 = b$

Binary classification problem

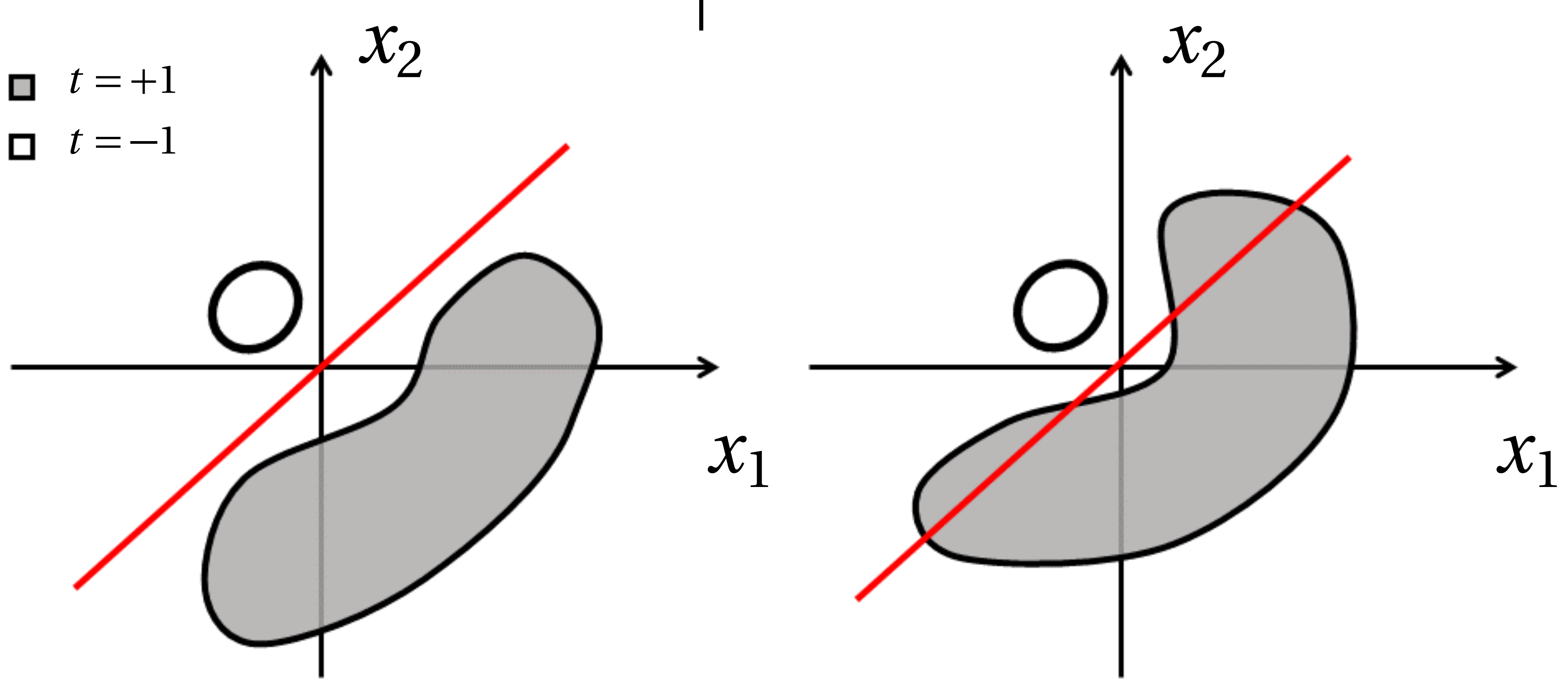
Have a set of p input patterns $\{\mathbf{x}^\mu\}$ and, for each, a desired output $t^\mu = \pm 1$

Want to find a $\mathbf{J}$ for which $\quad \mathrm{sgn}(\mathbf{J} \cdot \mathbf{x}^\mu) = t^\mu$

geometric interpretation:
"Linear separability"



Note: some sets of $\mathbf{x}^\mu$ can't be separated linearly:



(figure: B Mehlig)

Perceptron Learning Algorithm

(F Rosenblatt, 1962)

At each step, choose an $\mathbf{x}^\mu$, compute the output $O^\mu = \mathrm{sgn}(\mathbf{J} \cdot \mathbf{x}^\mu)$

Then change $\mathbf{J}$ by $\Delta\mathbf{J} = \eta(t^\mu - O^\mu)\mathbf{x}^\mu$ (η is learning rate)

This has been proved to converge if a $\mathbf{J}$ that will give $O^\mu = t^\mu$ for every pattern exists

This an "online" algorithm (make changes one input pattern at a time)

Also possible: "batch" learning: $\Delta\mathbf{J} = \eta \sum_\mu (t^\mu - O^\mu)\mathbf{x}^\mu$

or something in between: sum at each step is over some subset of the patterns (actually the most common thing done in everyday applications (though for more complex models than perceptrons)

The Capacity Problem

… “converges if a $\mathbf{J}$ that will give $O^\mu = t^\mu$ for every pattern exists”
But when will this be true?

Specifically, for p independent random input patterns of dimensionality N,
what is the maximum p for which a $\mathbf{J}$ exists that correctly classifies all the patterns?

Cover (1965) proved (combinatorics) that (as long as patterns are all linearly independent)

for $p_{\max} < 2N$ the probability of complete correct classification is $< 1/2$
for $p_{\max} = 2N$ the probability of complete correct classification is $= 1/2$
for $p_{\max} > 2N$ the probability of complete correct classification is $> 1/2$

and the transition gets sharp as p and $N \longrightarrow \infty$

Statistical-mechanical formulation

(Gardner, 1987)

Calculate the volume in $\mathbf{J}$-space in which all p equations

$$O^\mu = \mathrm{sgn}\left(N^{-1/2} \sum_{j=1}^{N} J_j x_j^\mu \right) = t^\mu$$

are satisfied. Constraint: $\sum_j J_j^2 = N$

(needed because multiplying all J_j's by a constant wouldn't change O^μ)

Volume shrinks as number p of constraints increases, $\longrightarrow 0$ at critical p

$$p_c = \alpha N, \quad \alpha = \mathrm{O}(1)$$

Replicas!

Constrained volume:

$$V = \frac{\int d\mathbf{J} \left(\prod_\mu \Theta(t^\mu N^{-1/2} \sum_j J_j x_j^\mu - \kappa) \right) \delta(\sum_j J_j^2 - n)}{\int d\mathbf{J} \delta(\sum_j J_j^2 - n)}$$

introduced κ: margin of stability, (Θ is the unit step (Heaviside) function)

Need to average $\log V$ (just like $\log Z$ in spin glass problems, so introduce replicas:

$$\langle V^n \rangle = \frac{\prod_a \int d\mathbf{J}^a \left(\prod_\mu \Theta(t^\mu N^{-1/2} \sum_j J_j^a x_j^\mu - \kappa) \right) \delta(\sum_j (J_j^a)^2 - N)}{\prod_a \int d\mathbf{J}^a \delta(\sum_j (J_j^a)^2 - N)}$$

Quick description of the replica calculation:

1. There are step functions and δ-funtions in the expression for $\langle V^n \rangle$. Use Fourier integral representations of these. This way, when the random input patterns are averaged over the J's end up occurring at most quadratically in the argument of exponential functions.

2. We then define an order parameter $q_{ab} = (1/N)\sum_j J_j^a J_j^b$ and assume replica symmetry: $q_{ab} = q \quad (a \neq b)$. Enforcing this constraint with yet another delta function and integrating the J's out leads eventually to $\langle V^n \rangle$ = a complicated function $G(q)$. (The calculation is unfortunately too long to give here.)

replica calculation result:

3. Finding its stationary point, $\partial G/\partial q = 0$ lead to this equation for q:

$$\alpha \int \frac{dy}{\sqrt{2\pi}} \mathrm{e}^{-y^2/2} \left[\int_u^\infty dz \mathrm{e}^{-z^2/2} \right]^{-1} \mathrm{e}^{-u^2/2} \frac{t + \kappa\sqrt{q}}{2\sqrt{q}(1-q)^{3/2}} = \frac{q}{2(1-q)^2}$$

with $u = (\kappa + y\sqrt{q})/\sqrt{1-q}$.

4. As the volume in J space where solutions exist shrinks to zero, there will finally be only one solution, so $q \to 1$. Taking $q \to 1$ in the above equation leads to

$$\alpha_c(\kappa) = \frac{1}{\left[\int_{-\kappa}^\infty \frac{dy}{\sqrt{2\pi}} \mathrm{e}^{-y^2/2} (y+\kappa)^2 \right]}$$

and, for no stability margin, $\alpha_c(0) = 2$

(in agreement with Cover's combinatoric results).

Beyond capacity:

(Whyte & Sherrington 1996, Györgyi & Reimann 1997)

What is the solution like for $\alpha > \alpha_c$?

Full replica symmetry breaking, like the SK spin glass

Deeper networks

Real-life problems are not generally linearly separable —
require deeper networks

(continuous-valued unit outputs)

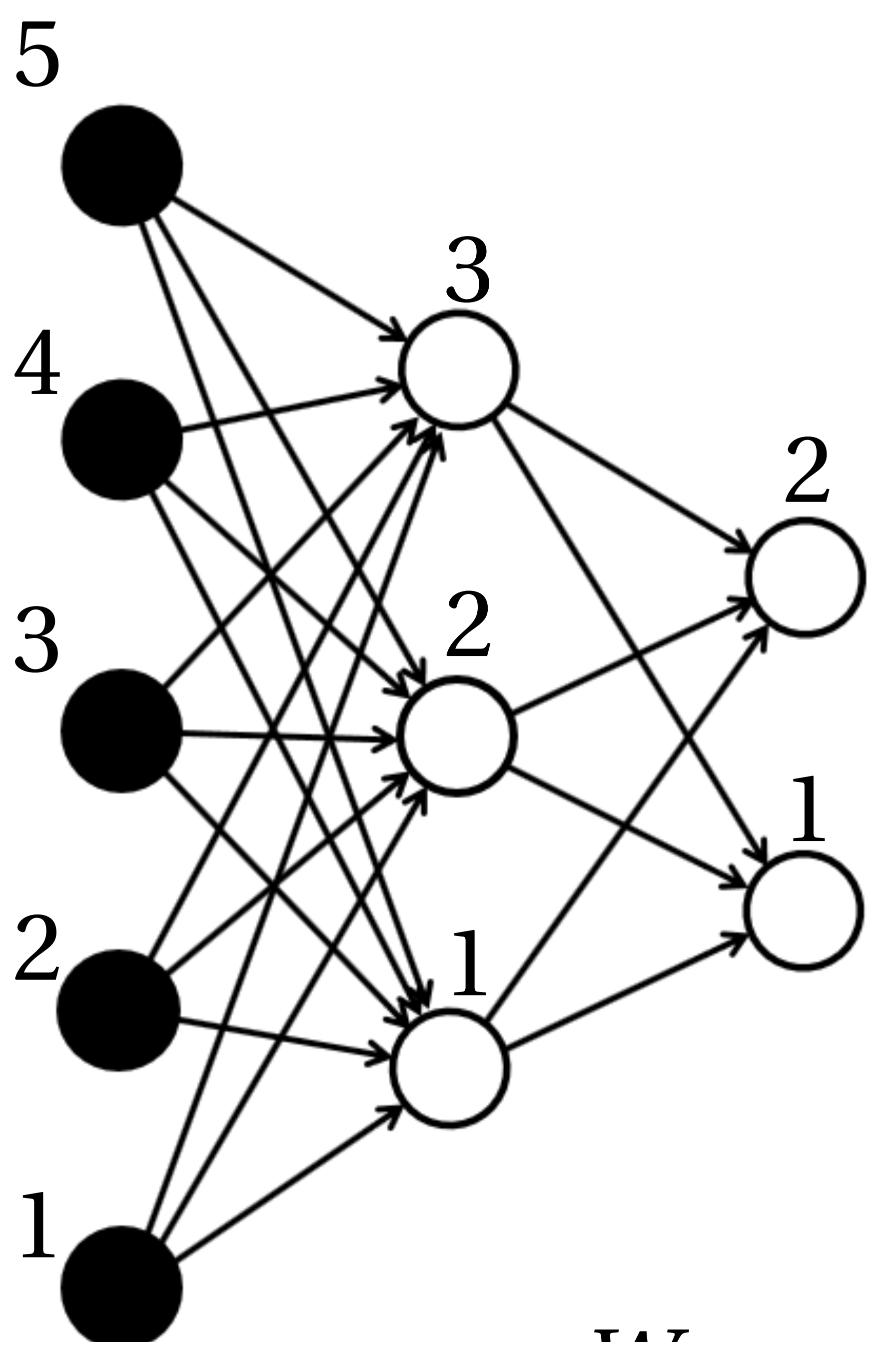
2-layer net

$$O_i = g\left[\sum_j J_{ij}^{(2)} g\left(\sum_k J_{jk}^{(1)} x_k\right)\right]$$

$g(\ )$: activation function

generalizable to any number of layers:

$$O_i = g\left[\sum_j J_{ij}^{(n)} g\left(\sum_k J_{jk}^{(n-1)} g(\sum_l J_{kl}^{n-2} g(\sum_m J_{lm}^{(n-3)} \cdots x_p)\right)\right]$$



Thanks again to Bernhard Mehlig for the figure

Nonlinearity

Activation function g: nonlinear

Commonly take $g(x) = \tanh x$ or $1/(1+\mathrm{e}^{-x})$ (sigmoidal)

or $x\Theta(x)$ ("threshold-linear")

Learning the J's: Gradient descent

For a single-layer network, define an error ("cost" or "loss") function, e.g.

$$E = \frac{1}{2p}\sum_{\mu}(t_i^\mu - O_i^\mu)^2$$

and adjust weights by "Delta rule":

$$\Delta J_{jk} = -\eta\frac{\partial E}{\partial J_{jk}} = \frac{\eta}{p}\sum_{i\mu}(t_i^\mu - O_i^\mu)\frac{\partial O_i^\mu}{\partial J_{ij}} = \frac{\eta}{p}\sum_{\mu} g'(h_j^\mu)(t_j^\mu - O_j^\mu)x_k^\mu$$

with $h_j^\mu = \sum_k J_{jk}x_k^\mu$, the net input to output unit k

(almost same form as perceptron learning)

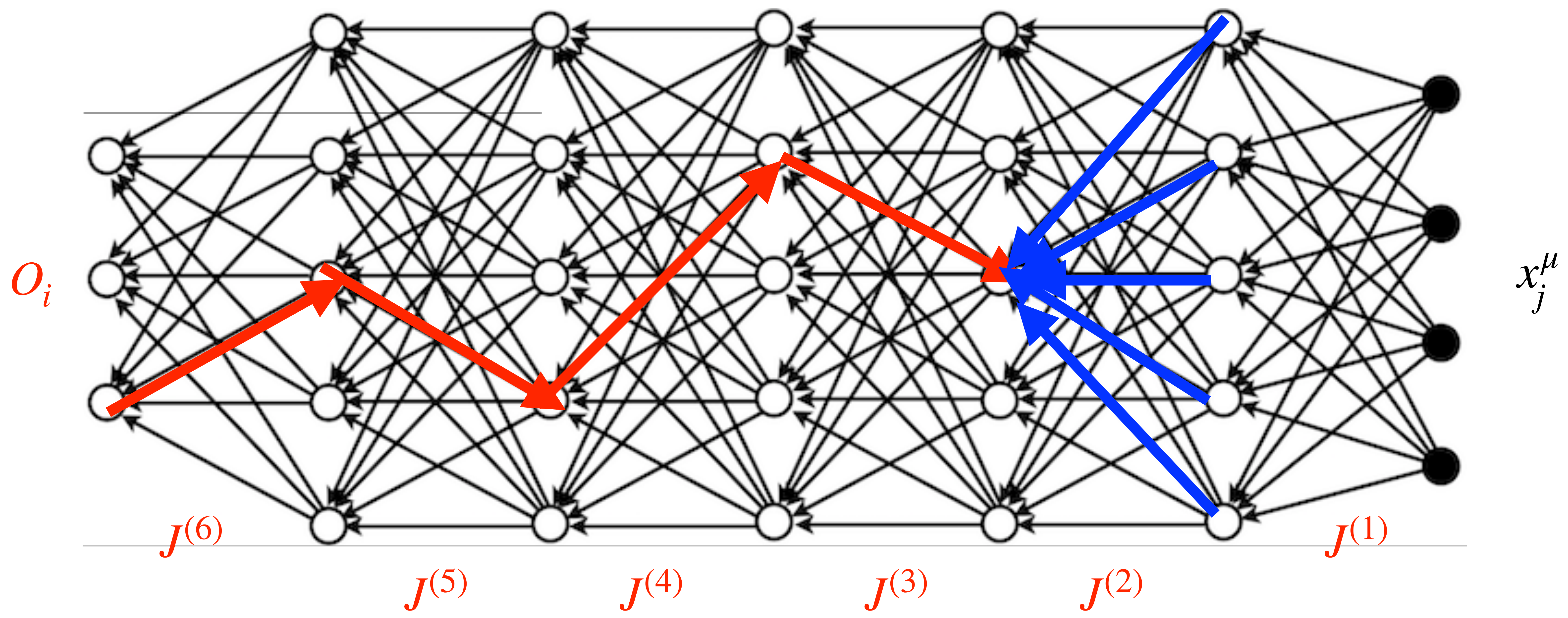
Back-propagation:

2 layers:
(same as 1-layer case for hidden-to-output weights)
input-to-hidden weights: (use chain rule)

$$\Delta J_{pq}^{(1)} = -\eta \frac{\partial E}{\partial J_{pq}^{(1)}} = \frac{1}{p} \sum_{i\mu} (T_i^\mu - O_i^\mu) g'(h_i^{(2),\mu}) J_{ip}^{(1)} g'(h_p^{(1),\mu}) x_q$$

with $h_i^{(2),\mu} = \sum_j J_{ij}^{(2)} g\left(\sum_k J_{jk}^{(1)} x_k^\mu \right), \quad h_j^{(1),\mu} = \sum_k J_{jk}^{(1)} x_k^\mu$

Deep network



General prescription, for any weight $J_{ij}^{(m)}$: consider paths backwards from all output units.

On each link, get a factor $J_{i'j'}^{(m')}$; on each node (n, j) (including (m, i)) get a factor $g'(h_n^{(j),\mu})$

Sum over all paths from all output units to (m, i): gives effective error on (m, i))

Multiply by $\mu_j^{(m-1),\mu} = g(\sum_k J_{jk}^{(m-2)} \mu_k^{(m-2),\mu})$ = output of unit $(m-1, j)$ ($\mu_k^{(0),\mu} = x_k^\mu$)

Cost functions:

Mean-square error ("MSE"): $E = \frac{1}{2p} \sum_{i\mu} (t_i^\mu - O_i^\mu)^2$

Negative log-likelihood ("NLL"): for stochastic Ising output units with

$P(O = \pm 1) = \frac{\mathrm{e}^{\pm h}}{\mathrm{e}^{h} + \mathrm{e}^{-h}}$, where $h = \sum_k J_k^{(n)} \mu_k^{(n-1)}$ is net input to unit

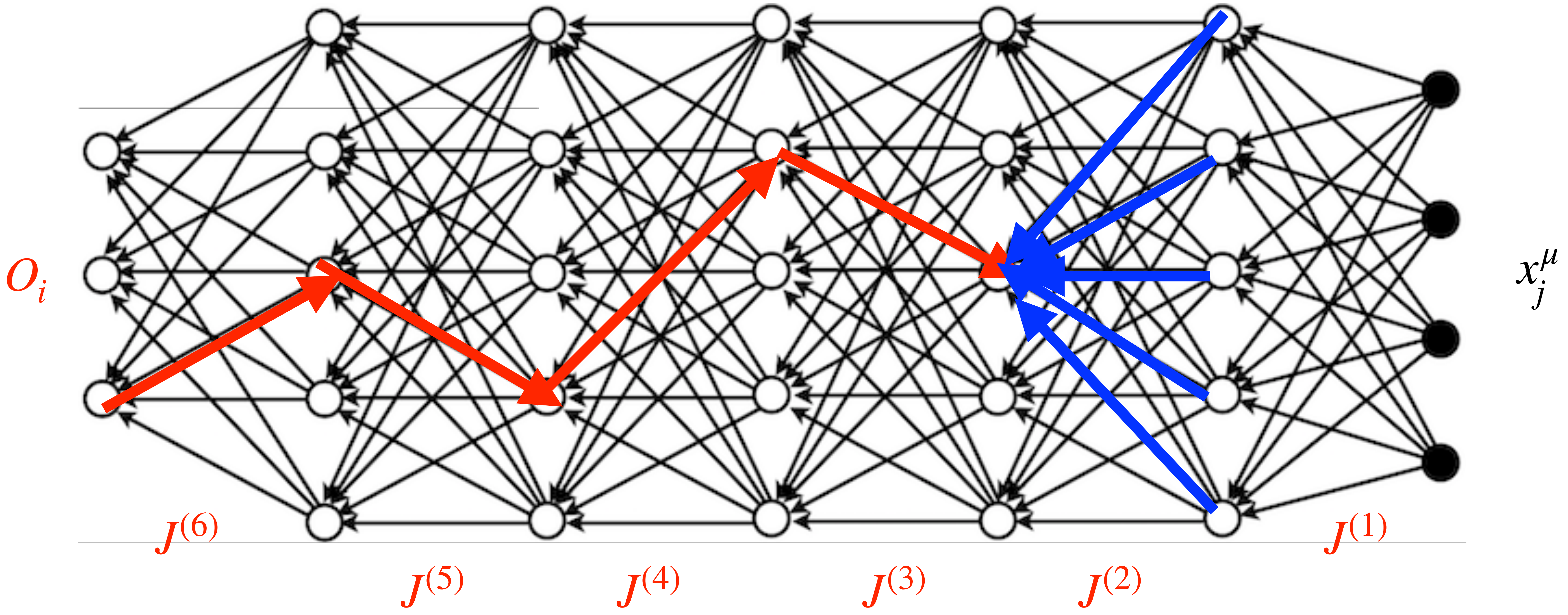
and targets $t = \pm 1$: Probability of correct output = $\frac{\mathrm{e}^{th}}{\mathrm{e}^{h} + \mathrm{e}^{-h}}$, so

$$E = -\frac{1}{p} \sum_\mu [t^\mu h^\mu - \log\cosh h^\mu]$$ (single output case)

and $$\Delta J_k^{(n)} = \frac{\eta}{p} \sum_\mu (t^\mu - \tanh h^\mu) \mu_k^{(n-1),\mu}$$

(like MSE with $g(h) = \tanh h$ except no derivative factor)

Deep network: just one change in backpropagation algorithm



General prescription, for any weight $J_{ij}^{(m)}$: consider paths backwards from all output units.

On each link, get a factor $J_{i'j'}^{(m')}$; on each node (n, j) (EXCEPT (m, i)) get a factor $g'(h_n^{(j),\mu})$

Sum over all paths from all output units to (m, i): gives effective error on (m, i))

Multiply by $\mu_j^{(m-1),\mu} = g(\sum_k J_{jk}^{(m-2)} \mu_k^{(m-2),\mu})$ = output of unit $(m-1, j)$ ($\mu_k^{(0),\mu} = x_k^\mu$)

Online learning and stochastic gradient descent

So far, this was “batch update”.

Online learning: Instead, update one (randomly chosen) example at a time.

The average Δw will be the same as before, but there will be a variance

$$\frac{1}{p}\sum_{\mu}(\Delta J_{ij}^{\mu} - \langle \Delta J_{ij}^{\mu} \rangle_{\mu})^2$$

intrinsic noise in the algorithm

Stochastic gradient descent (“SGD”): (the most commonly used algorithm)

At each step, average over a randomly chosen set of examples (“minibatch”) of size m. Still noisy, but noise variance reduced by a factor m.

Problem with deep networks:

If the ws are too big or too small, the inputs to successive layers can grow or shrink exponentially. (This hindered the use of deep networks for some time.)

One solution: use orthogonal matrices for J^j.

Why? Consider the linear case (Saxe et al, 2014):

$$\mu_i^a = \sum_j J_{ij}^a \mu_j^{a-1} \qquad \text{SVD: } J_{ij}^a = \sum_\beta u_{i\beta} s_\beta v_{\beta j}^T$$

$$\implies \sum_i \mu_{a,i}^2 = \sum_\beta \mu_{a,\beta}^2 = \sum_\gamma s_\gamma^2 \mu_{a-1,\gamma}^2$$

Stable propagation through layers (or time): make all $s_\gamma = 1$ (orthogonal matrix)

(Usually, orthogonal initialisation is sufficient)

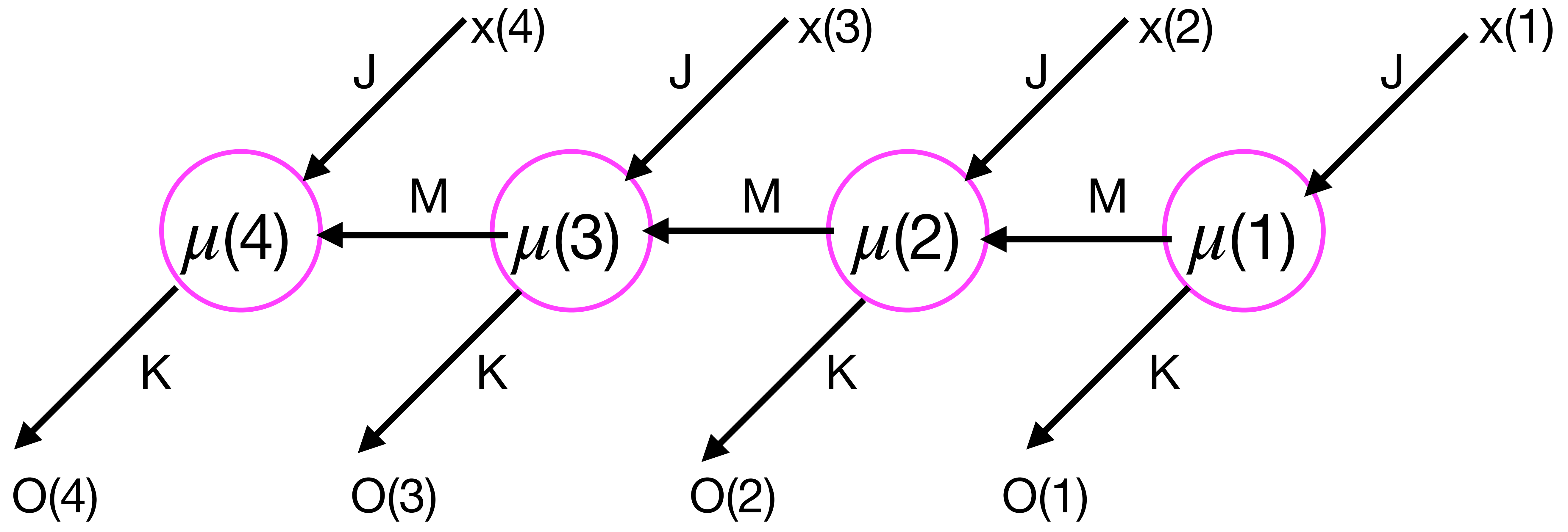
Recurrent networks

Simplest model: input time series $x_i(t), \quad t = 1,2,3\cdots$

$$\mu_i(t) = g\left(\sum_j M_{ij}\mu_j(t-1) + \sum_k J_{ik}x_i(t)\right), \quad O_i(t) = g\left(\sum_j K_{ij}\mu_j(t)\right)$$

recurrent interactions   input   output

like layers in time:



$\longleftarrow$ time

Learning sequences

Training: target $T(t) = x(t+1)$

The memory mechanism:

Consider a recurrent layer with linear units $\mu_t = h_t + M\mu_{t-1}$ $(h = Jx)$

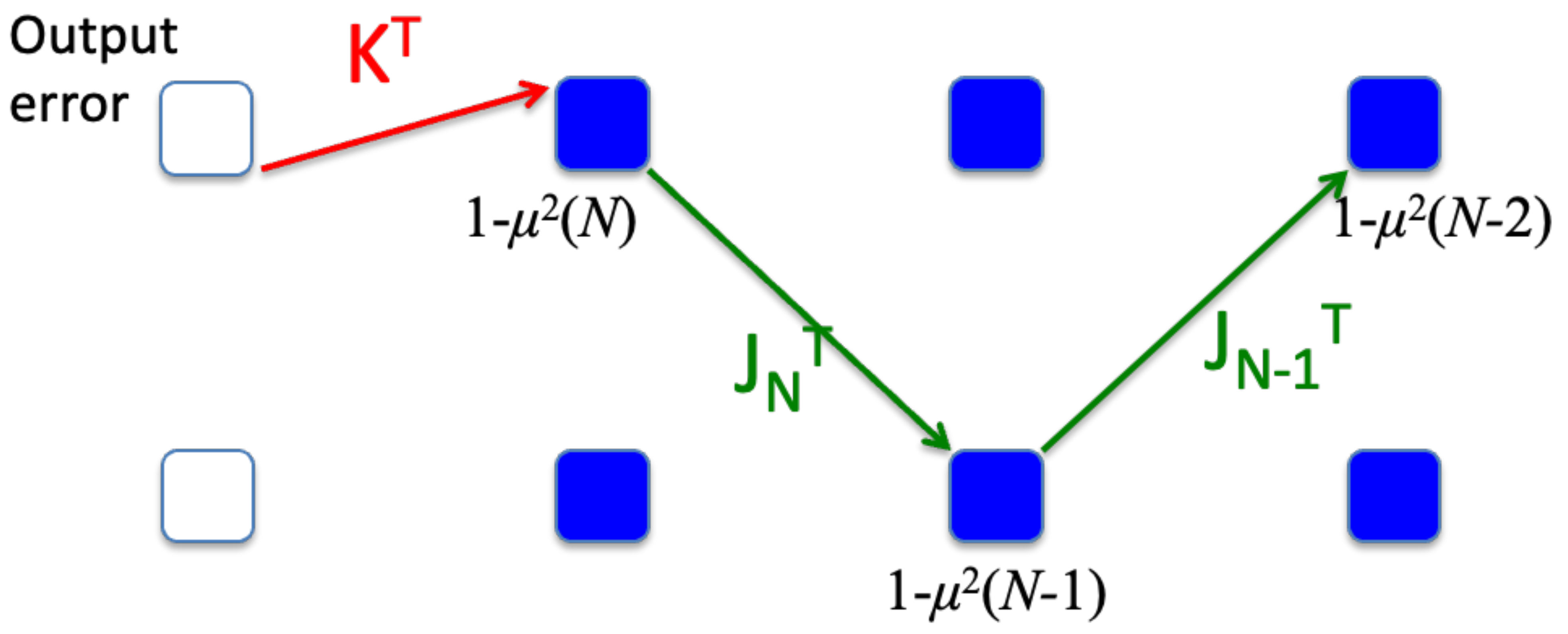
iterate: $\mu_t = h_t + Mh_{t-1} + M^2 h_{t-2} + \cdots + M^t h_0$

In this way, the entire set of past inputs is represented on the hidden units, to be fed onward to the outputs

Training recurrent networks

Backpropagation through time (“BPTT”)

ordinary backprop (through layers):



BPTT: