Lecture Plan:

* Background (Recap from last lecture)

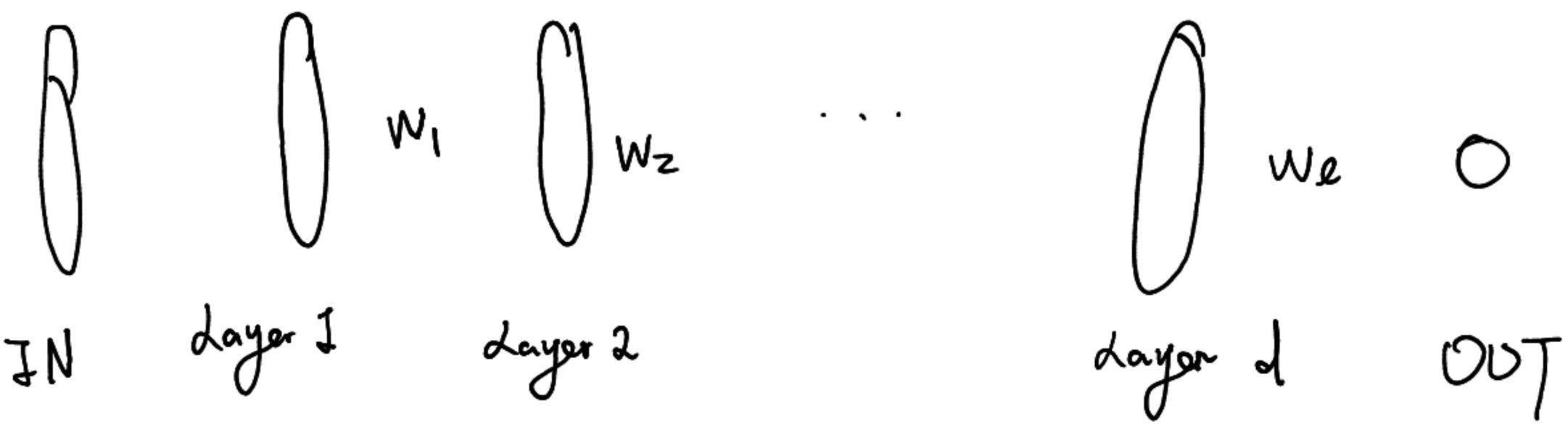
* Warmup: Multi-layer linear feedforward neural network

* The general case:
  - Forward pass
  - Backward pass (with an first attempt)

* Extensions:
  - Acyclic computation graphs
  - Weight sharing (e.g. Conv Nets)
  - Cyclic amputation graphs (RNN)
  - Hessian-vector product

★Background. A natural algorithm for training a neural network is gradient descent. How do we run the gradient descent algorithm efficiently?

Warmup: Linear neural network.



$$\text{Loss}(x, y) = \left(W_d W_{d-1} \cdots W_1 x - y \right)^2$$

$$= \ell(W_1, W_2, \cdots, W_d)$$

Gradients.

$$\frac{\partial \ell(W_1, \cdots, W_d)}{\partial W_i}$$

$$= \ell(W_1, \cdots, W_{i-1}, W_i + \delta, W_{i+1}, \cdots, W_d) - \ell(W_1, \cdots, W_{i-1}, W_i, W_{i+1}, \cdots, W_d)$$

$$= \left(W_d \cdots (W_i + \delta) \cdots x - y\right)^2 - \left(W_d \cdots (W_i) \cdots x - y\right)^2$$

$$= 2\left(W_d \cdots W_{i+1} \cdot \delta \cdot W_{i-1} \cdots W_1 x - y\right) \cdot \left(W_d \cdots W_1 x - y\right) + O(\delta^2)$$

(1)

Matrix calculus. (Taylor expansion of matrix functionals) (Expand a bit more)

$$\alpha(\cdots, W_i+\delta, \cdots) - \alpha(\cdots, W_i, \cdots)$$

$$= \left\langle \frac{\partial \alpha(W_1 \cdots, W_d)}{\partial W_i} , \delta \right\rangle \qquad (2)$$

Remark: inner product $\langle a, b\rangle = \sum_{i=1}^{n} a_i b_i$, for two vectors $a, b \in \mathbb{R}^n$, and

$$\langle X, Y\rangle = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} X_{ij} Y_{ij}, \text{ for two matrices } X, Y \in \mathbb{R}^{n_1 \times n_2},$$

So, our goal is to simplify Eq(1) to a form like Eq(2).

$$W_d \cdots W_{i+1} \delta W_i \cdots W_1 x$$

$$= Tr\left[W_d \cdots W_{i+1} \delta W_i \cdots W_1 x \right]$$

$$= Tr\left[W_i \cdots W_1 x W_d \cdots W_{i+1} \delta \right] \qquad \left(Tr[ABC] = Tr[CAB] \right)$$

$$= \left\langle \left(W_i \cdots W_1 x W_d \cdots W_{i+1}\right)^T, \delta \right\rangle$$

$$= \left\langle \left(W_d \cdots W_{i+1}\right)^T \left(W_i \cdots W_1 x\right)^T, \delta \right\rangle$$

$$\Rightarrow \text{Eq}(2) = \alpha(W_d \cdots W_1 x - y) \cdot \left(\left\langle \left(W_d \cdots W_{i+1}\right)^T \left(W_i \cdots W_1 x\right)^T, \delta \right\rangle - y \right)$$

Hence $\frac{\partial L}{\partial W_i} = 2(W_d \cdots W_1 x - y) \cdot (W_d \cdots W_{i+1})^T \cdot (W_i \cdots W_1 x)^T$. (3)

How do we implement the above gradient computation efficiently?

Forward Pass.

IN: x

output of layer 1: $W_1 \cdot x \leftarrow a_1$

⋮

Output of layer i: $W_i \cdots W_1 x \leftarrow a_i$

Notice that a_i appears in Eq (3)!

Backward Pass.

OUT: y

Gradient wrt to quadratic loss function:

$$g_d = 2(W_d \cdots W_1 x - y)$$

Gradient wrt to the input to the d-th layer:

$$g_{d-1} = \frac{\partial (W_d \cdot z_d - y)^2}{\partial z_d} = 2 W_d^T (W_d \cdot z_d - y)$$

$$= W_d^T \cdot g_d$$

Gradient wrt to the input the (d-1)-th layer:

$$g_{d-2} = \frac{\partial (W_d \cdot W_{d-1} \cdot z_{d-1} - y)^2}{\partial z_{d-1}}$$

$$= 2 W_{d-1}^T W_d^T (W_d \cdot W_{d-1} \cdot z_{d-1} - y)$$

$$= W_{d-1}^T \cdot g_{d-1}$$

$$\vdots$$

$$g_i = W_{i+1}^T \cdot g_{i+1}$$

Put together.

Observe that $\frac{\partial L}{\partial W_i} = g_i \cdot a_i^T$!

Therefore, our full algorithm is:

Backpropagation for linear neural net

IN : $(x_1, y_1) \cdots (x_m, y_m)$

Req : $(W_1^{(t)}, \ldots W_d^{(t)})$, η

OUT : $(W_1^{(t+1)}, \ldots, W_d^{(t+1)})$

Forward pass :

compute $a_1, a_2, \ldots, a_d$

Backward pass :

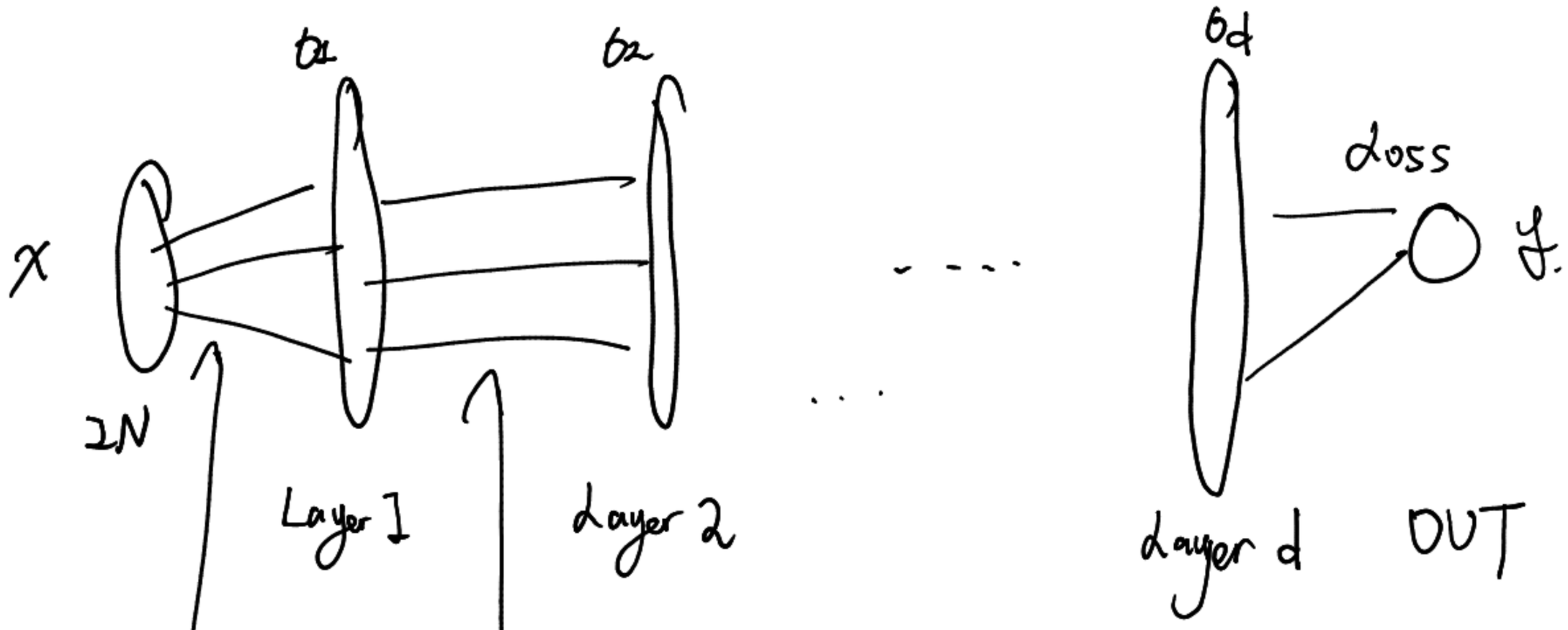
compute $g_d, g_{d-1}, \ldots g_1$

update $W_i^{(t+1)} \leftarrow W_i^{(t)} - \eta \cdot g_i \cdot a_i^T$

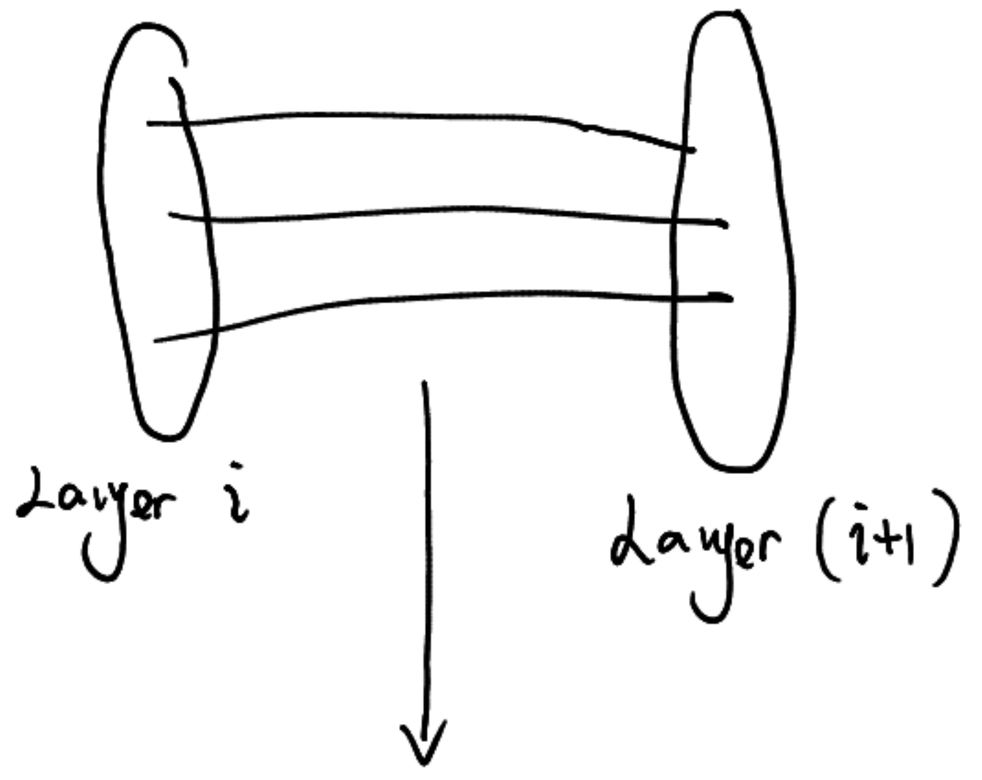
The general case.

Non-linear activation, arbitrary loss function, bias in activation function.

Model.



Chain rule. We would like to calculate $\frac{\partial f(w_1, w_2, \ldots, w_d)}{\partial w_i}$?



$\updownarrow$ abbrev. $\frac{\partial f}{\partial w_i}$

Weight matrix: $W_i \in \mathbb{R}^{r_i \times r_{i+1}}$

Output of layer i: $a_i \in \mathbb{R}^{r_i}$

Input to layer $i+1$: $z_i \in \mathbb{R}^{r_{i+1}}$

Claim 1. $\frac{\partial f}{\partial w_i} = \frac{\partial f}{\partial z_i} \cdot a_i^T$

Proof. Chain rule: $\frac{\partial f}{\partial W_i} = \frac{\partial f}{\partial z_i} \cdot \frac{\partial z_i}{\partial W_i} = \frac{\partial f}{\partial z_i} \cdot a_i^T$.

By Def. $z_i = W_i \cdot a_i = \langle a_i^T, W_i \rangle$, hence $\frac{\partial z_i}{\partial W_i} = a_i^T$.

Claim 2. $\frac{\partial f}{\partial z_i} = \left(W_{i+1}^T \cdot \frac{\partial f}{\partial z_{i+1}} \right) \odot \sigma_{i+1}'(z_i)$.

Notation: - for two vectors $x, y \in \mathbb{R}^n$, their Kronecker product

$$x \odot y = (x_1 y_1, x_2 y_2, \ldots, x_n y_n) \in \mathbb{R}^n.$$

- for a function $\sigma : \mathbb{R} \to \mathbb{R}$, we use the vector notation $\sigma(x) = (\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_n))$, and

$$\sigma'(x) = (\sigma'(x_1), \sigma'(x_2), \ldots, \sigma'(x_n)).$$

Proof. By def. $\nabla f = \langle \frac{\partial f}{\partial z_i}, \nabla z_i \rangle$, for some ∇z_i that goes to zero.

So, suppose we perturb z_i by δ, we would like to know how much is f changed. Note that

$$\nabla f = \langle \frac{\partial f}{\partial z_{i+1}}, \nabla z_{i+1} \rangle.$$

So we are going to track how much z_{i+1} chages.

$$\begin{aligned} z_{i+1} &= W_{i+1} \cdot a_{i+1} \\ &= W_{i+1} \cdot \sigma_{i+1}(z_i + \delta) \\ &= W_{i+1} \cdot (\sigma_{i+1}(z_i) + \sigma_{i+1}'(z_i) \odot \delta) \end{aligned}$$

$$= W_{i+1} \cdot \left(\sigma_{i+1}(z_i) + \sigma'_{i+1}(z_i) \odot \delta \right)$$

$$= W_{i+1} \cdot \sigma_{i+1}(z_i) + W_{i+1} \cdot \left(\sigma'_{i+1}(z_i) \odot \delta \right)$$

$$\Rightarrow \quad \nabla z_{i+1} = W_{i+1} \cdot \left(\sigma'_{i+1}(z_i) \odot \delta \right). \quad \text{Therefore}$$

$$\nabla f = \left\langle \frac{\partial f}{\partial z_{i+1}}, W_{i+1} \cdot \left(\sigma'_{i+1}(z_i) \odot \delta \right) \right\rangle$$

$$\left\langle \frac{\partial f}{\partial z_i}, \delta \right\rangle = \mathrm{Tr}\left[\left(\frac{\partial f}{\partial z_{i+1}} \right)^T \cdot W_{i+1} \cdot \left(\sigma'_{i+1}(z_i) \odot \delta \right) \right]$$

$$= \left\langle W_{i+1}^T \cdot \frac{\partial f}{\partial z_{i+1}}, \sigma'_{i+1}(z_i) \odot \delta \right\rangle.$$

$$\Rightarrow \quad \frac{\partial f}{\partial z_i} = \left(W_{i+1}^T \cdot \frac{\partial f}{\partial z_{i+1}} \right) \odot \sigma'_{i+1}(z_i).$$

Remark. Suppose that $\sigma_{i+1}(z_i) = z_i$ (i.e. linear activation),

then $\sigma'_{i+1}(z_i) = I$, therefore

$$\frac{\partial f}{\partial z_i} = W_{i+1}^T \cdot \frac{\partial f}{\partial z_{i+1}}.$$ Combining Claim 1 and 2,

we have shown that

$$\frac{\partial f}{\partial W_i} = W_{i+1}^T \cdot \frac{\partial f}{\partial z_{i+1}} \cdot a_i^T,$$ which recovers

our earlier result!

Put together, we have that

$$\frac{\partial f}{\partial W_i} = \frac{\partial f}{\partial z_i} \cdot a_i^T ,$$

$$\frac{\partial f}{\partial z_i} = \left(W_{i+1}^T \cdot \frac{\partial f}{\partial z_{i+1}} \right) \cdot \sigma_{i+1}'(z_i) .$$

Goal: Compute $\{a_i\}$, $\left\{\frac{\partial f}{\partial z_i}\right\}$, $\{\sigma_{i+1}'(z_i)\}$.

Recall that z_i is the input to the $(i+1)$-th Layer, and a_i is the output of the i-th layer.

Therefore,

$$z_i = W_i \cdot a_i ,$$

Forward Pass $\{a_i\}$. $a_0 = x$

$$a_{i+1} = \sigma(z_{i+1})$$

$$= \sigma(W_i \cdot a_i) .$$

Backward Pass. $\left\{\frac{\partial f}{\partial z_i}\right\}$. $\frac{\partial f}{\partial z_d} = \frac{\partial \text{Loss}(\sigma_{d+1}(z_d), y^{\text{true}})}{\partial z_d}$.

$$\frac{\partial f}{\partial z_i} = (W_{i+1}^T \cdot \frac{\partial f}{\partial z_{i+1}}) \cdot \sigma_{i+1}'(z_i) .$$

Put together, we have derived the Backpropagation algorithm.

Backpropagation for feedforward neural networks

Forward Pass:

Compute $\{a_i\}$, $\{z_i\}$.

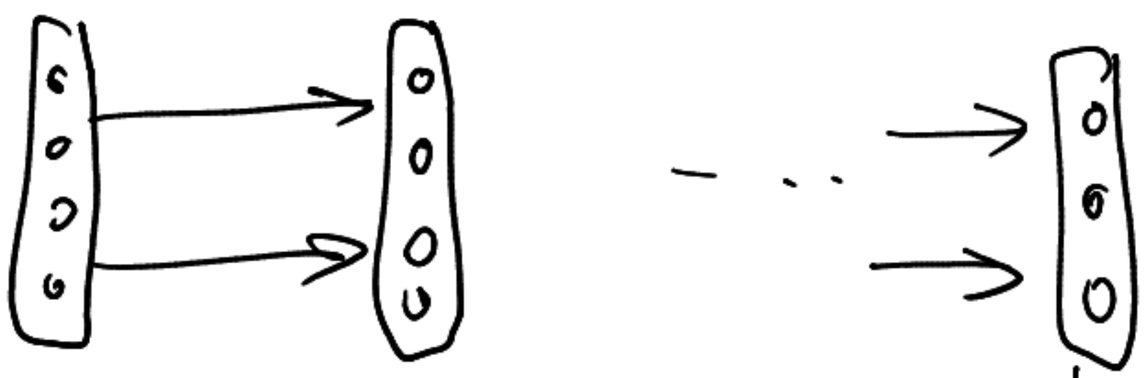
Backward Pass:

Compute $\left\{\frac{\partial f}{\partial z_i}\right\}$, hence $\left\{\frac{\partial f}{\partial w_i}\right\}$.

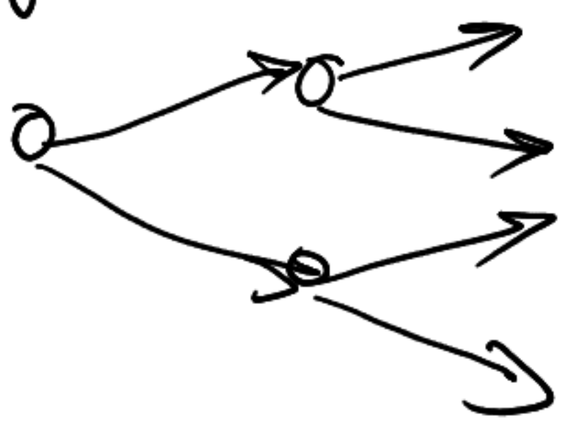
Apply gradient descent over $\{w_i\}$.

Extensions.

- Acyclic neural network



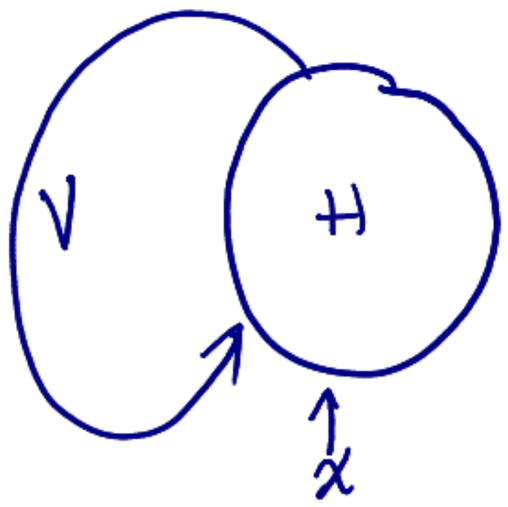
In general, the algorithm works as long as



there are no cycles in the network!

What if there are cycles? e.g. RNN? yes but we may run into issues.

Exploding or Vanishing gradients.



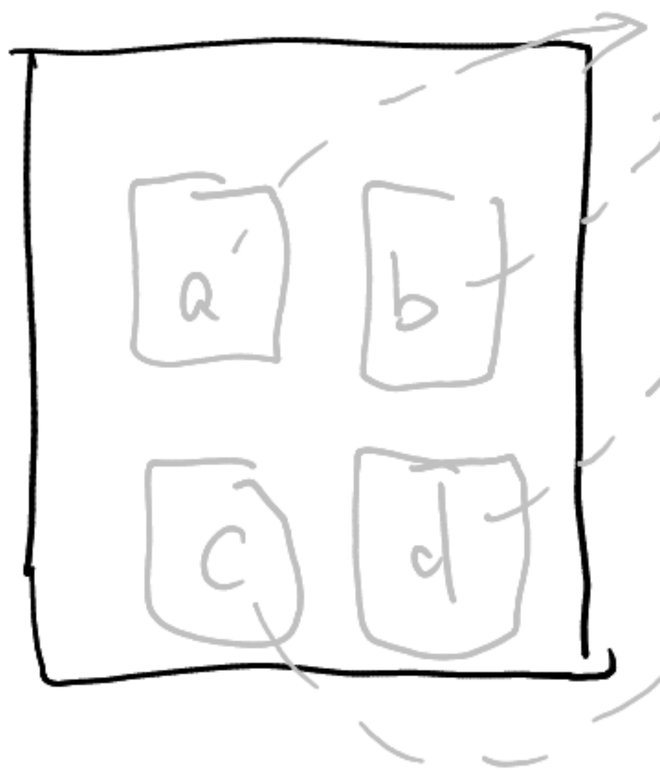
i) if $\|V\|_2 < 1$, then $\|V^i \cdot x\| < \sigma^i_{max}(V) \cdot \|x\|$

ii) if $\|V\|_2 > 1$, then $\|V^i \cdot x\| > \sigma^i_{min}(V) \cdot \|x\|$

Fix. Gradient clipping on Long short-term memory.

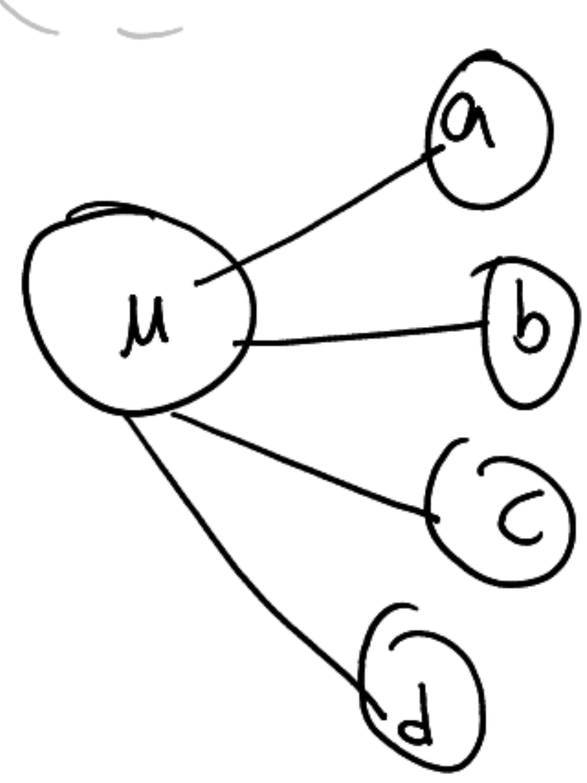
Fix. Backpropagation in time. Expand over the prev. k steps.

— Weight sharing.



multiple patches share the same weight matrix W.

Set
$\mu = a$
$\mu = b$
$\mu = c$
$\mu = d$



$$\frac{\partial f}{\partial \mu} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial \mu} + \cdots$$

$$= \frac{\partial f}{\partial a} + \frac{\partial f}{\partial b}.$$

Second-order methods:

- Hessian : $\frac{\partial f}{\partial w_i \partial w_j}$

- Traditional use of Hessian

- Hessian in neural nets

Conclusions.

- We derived the backprogation algorithm, which involves a forward pass and a backward pass.

— Backpropagation provides a linear-time implementation of gradient descent.

* Key idea : Chain rule in calculus.

— Extensions : Acyclic networks, Weight sharing,

— Discussion : second-order methods.