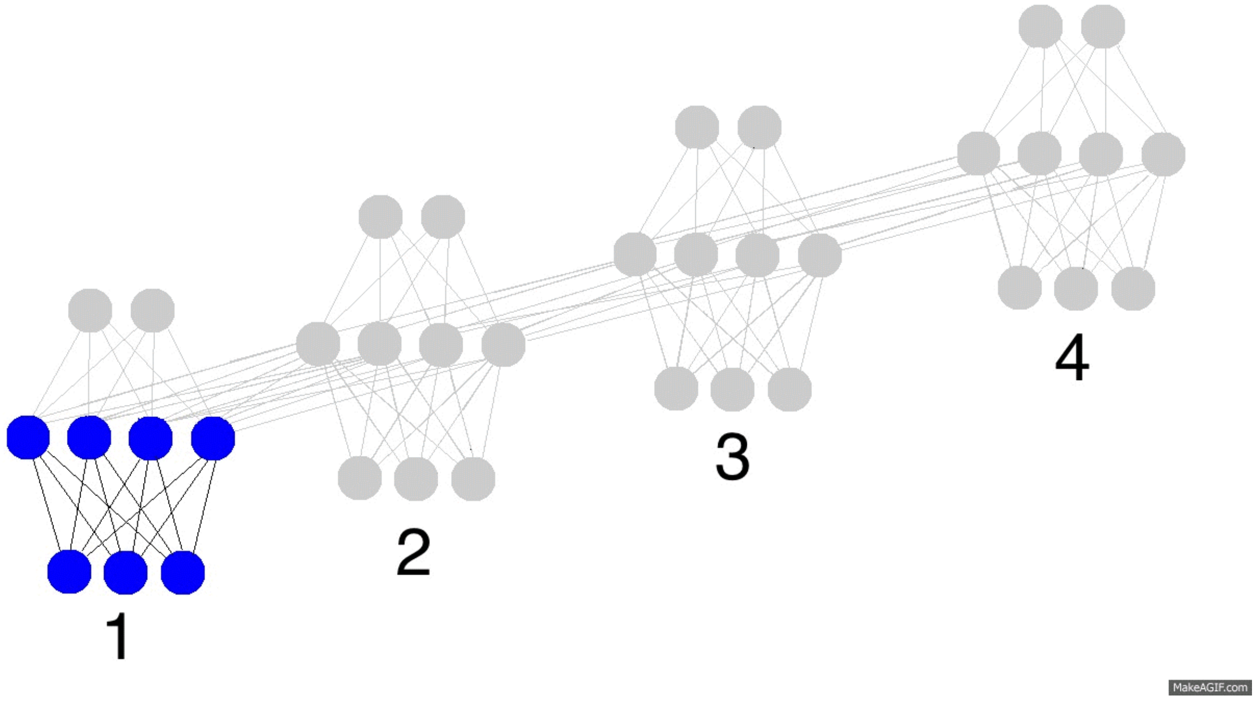
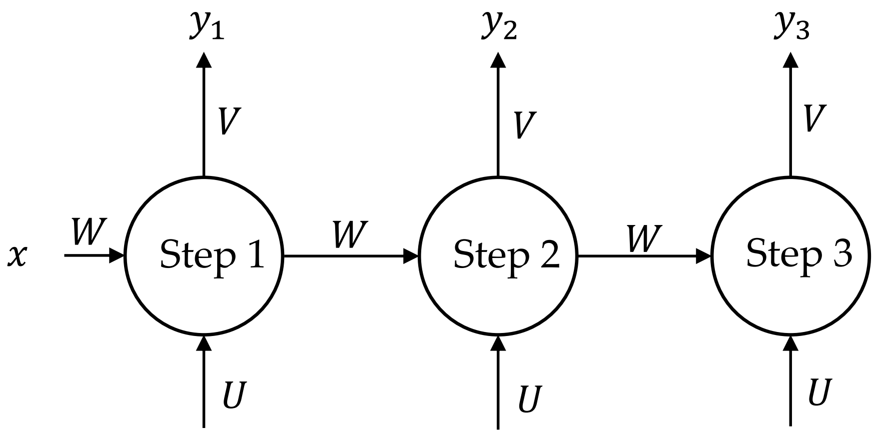
Backpropagation through time

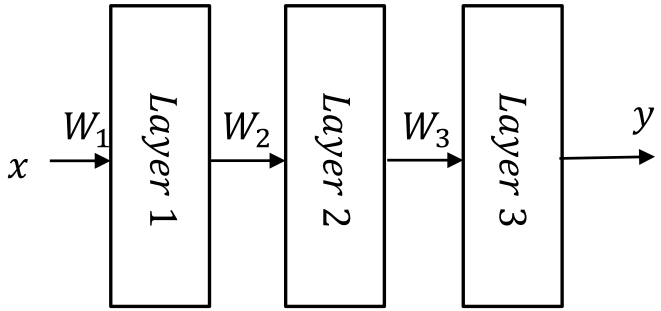


RNNs vs MLPs

- Main difference: Layer-shared parameters *vs* Layer-specific parameters
  - Just mentally switch from 'time steps' to 'layers'
- Question: how to train with shared parameters?
  - Backpropagation … through time



3-step recurrent neural network



3-layer neural network

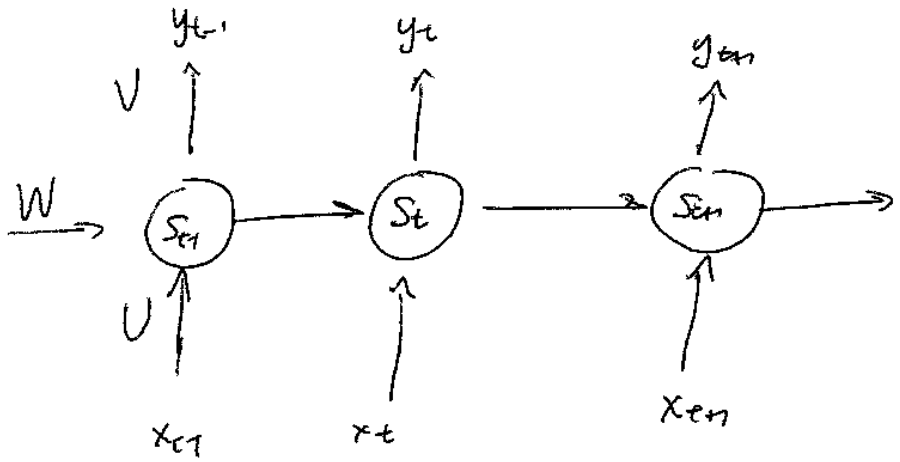
Backpropagation through time (BPTT)

- Basically, chain rule again for $\frac{d\mathcal{L}}{dV}, \frac{d\mathcal{L}}{dU}, \frac{d\mathcal{L}}{dW} \rightarrow$ the same algorithm
  - Caveat: shared computations complicate the chain rule

$$y_t = \text{softmax}(V \cdot \boldsymbol{s}_t)$$
$$\boldsymbol{s}_t = \tanh(U \cdot \boldsymbol{x}_t + W \cdot \boldsymbol{s}_{t-1})$$

- V, U, W are same for $t, t+1, \ldots$
  - Gradients flow <u>not from a 'single path'</u> of previous layer like in MLP
  - The recurrence in the chain rule 'hides' multiple dependencies

Unfolded graph



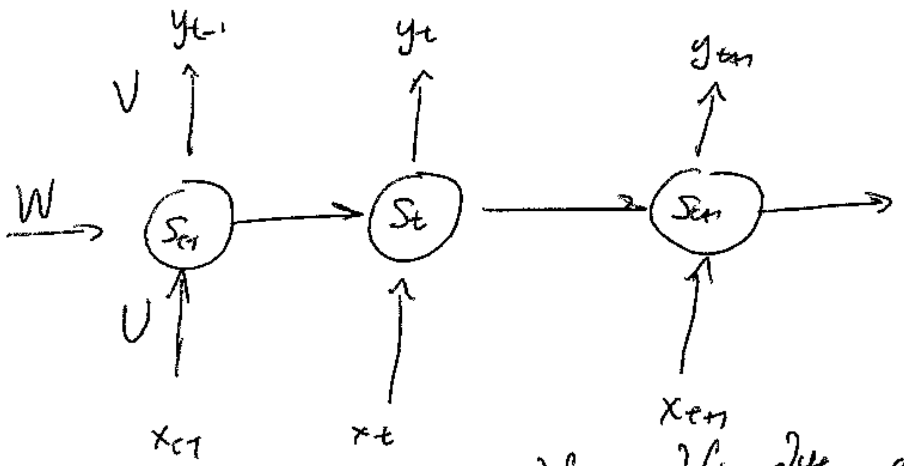
$$y_t = \mathrm{softmax}(V \cdot s_t)$$

$$s_t = h(U \cdot x_t + W \cdot s_{t-1})$$

$$s_{t-1} = h(U \cdot x_{t-1} + W \cdot s_{t-2})$$

$$\vdots$$

Unfolded graph



$$y_t = \text{softmax}(V \cdot s_t)$$
$$s_t = h(U \cdot x_t + W \cdot s_{t-1})$$
$$s_{t-1} = h(U \cdot x_{t-1} + W \cdot s_{t-2})$$
$$\vdots$$

$$\frac{\partial \mathcal{L}_t}{\partial W} = \frac{\partial \mathcal{L}_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial s_t} \cdot \frac{\partial s_t}{\partial W}$$
$$\frac{\partial \mathcal{L}_t}{\partial U} = \frac{\partial \mathcal{L}_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial s_t} \cdot \frac{\partial s_t}{\partial U}$$
$$\frac{\partial \mathcal{L}_t}{\partial V} = \frac{\partial \mathcal{L}}{\partial y_t} \frac{\partial y_t}{\partial V}$$

BPTT: Chain rule for $\partial \mathcal{L}_t / \partial V$ in unfolded graph

- Dependencies from $\boldsymbol{s}_t$ in separate covariate variables

$$\boldsymbol{s} = h(\boldsymbol{s}_t, \boldsymbol{s}_{t-1}, \dots, \boldsymbol{s}_0)$$

- All gradient path flows from $\boldsymbol{s}$ to $\boldsymbol{w}$
  - Via $\boldsymbol{s}_t$
  - Via $\boldsymbol{s}_{t-1}$
  - …

$$\frac{d\boldsymbol{s}}{d\boldsymbol{w}} = \sum_{i=0}^{t} \frac{d\boldsymbol{s}}{d\boldsymbol{s}_i} \cdot \frac{d\boldsymbol{s}_i}{d\boldsymbol{w}}$$

$s = s_2$ (The state s at the current time step 2)

$s_2 = h(z_2)$

$z_2 = Ux_2 + Ws_1$

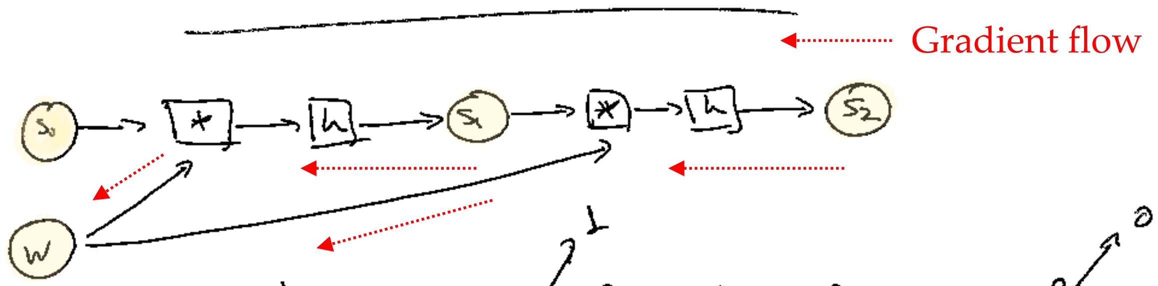
$s_1 = h(z_1)$

$z_1 = Ux_1 + Ws_0$

$s_0 = \text{const.}$

or

$s = s_2 \circ s_1 \circ s_0$



$$\frac{\partial s}{\partial w} = \sum_{i=0}^{t=2} \frac{\partial s}{\partial s_i}\frac{\partial s_i}{\partial w} = \frac{\partial s}{\partial s_2}\frac{\partial s_2}{\partial w} + \frac{\partial s}{\partial s_1}\frac{\partial s_1}{\partial w} + \frac{\partial s}{\partial s_0}\frac{\partial s_0}{\partial w}$$

$$= \frac{\partial s_2}{\partial w} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial w}$$

BPTT: Chain rule by change of variable differentiation

- Same result but more involved
- One must keep in mind that the nonlinearity h and the derivative acts within 'one layer'
  - No recursion for h, only via $\boldsymbol{s}_i$

(1) $s = s_2$ (The state s at the current time step 2)

(2) $s_2 = h(z_2)$

(3) $z_2 = U x_2 + W s_1$

(4) $s_1 = h(z_1)$

(5) $z_1 = U x_1 + W s_0$

(6) $s_0 = \text{const.}$

or

$s = s_2 \circ s_1 \circ s_0$

$y = f(g(x))$
$y' = f'(g(x)) \cdot g'(x)$

$$\frac{\partial s}{\partial W} \overset{(1)}{=} \underbrace{\frac{\partial s}{\partial s_2}}_{1} \frac{\partial s_2}{\partial W}$$

$$\overset{(2)}{=} h'(z_2) \frac{dz_2}{dW} \overset{(3)}{=} h'(z_2)\left(s_1 + W \frac{ds_1}{dW}\right) = h'(z_2) s_1 + h'(z_2) \cdot W \cdot \frac{ds_1}{dW}$$

$$\overset{(4)}{=} h'(z_2) s_1 + h'(z_2) \cdot W h'(z_1) \cdot \frac{dz_1}{dW} \overset{(5)(6)}{=} \underbrace{h'(z_2) s_1 + h'(z_2) \cdot W \cdot h'(z_1) s_0}_{\frac{\partial s}{\partial s_2} \frac{\partial s_2}{\partial W}} \quad (7,8,9) \rightarrow \frac{\partial s}{\partial s_1} \frac{\partial s_1}{\partial W}$$

$$\frac{\partial s}{\partial W} = \frac{\partial s}{\partial s_1} \frac{\partial s_1}{\partial W} \quad (7)$$

$$\frac{\partial s_2}{\partial s_1} = \frac{\partial s_2}{\partial z_2} \frac{\partial z_2}{\partial s_1} = h'(z_2) W \quad (8) \qquad \frac{\partial s_1}{\partial W} = h'(z_1) \cdot s_0 \quad (9)$$

BPTT for memory and input parameters $\partial \mathcal{L}_t / \partial \boldsymbol{W}, \partial \mathcal{L}_t / \partial \boldsymbol{U}$

- Putting everything together

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{W}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial \boldsymbol{s}_t} \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_i} \frac{\partial \boldsymbol{s}_i}{\partial \boldsymbol{W}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial \boldsymbol{s}_t} \left(\prod_{j=i+1}^{t} \frac{\partial \boldsymbol{s}_j}{\partial \boldsymbol{s}_{j-1}} \right) \frac{\partial \boldsymbol{s}_i}{\partial \boldsymbol{W}}$$

- If you have a new loss per time step, sum over time steps

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \sum_{t} \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{W}}$$

- Similar for input parameters $\boldsymbol{U}$

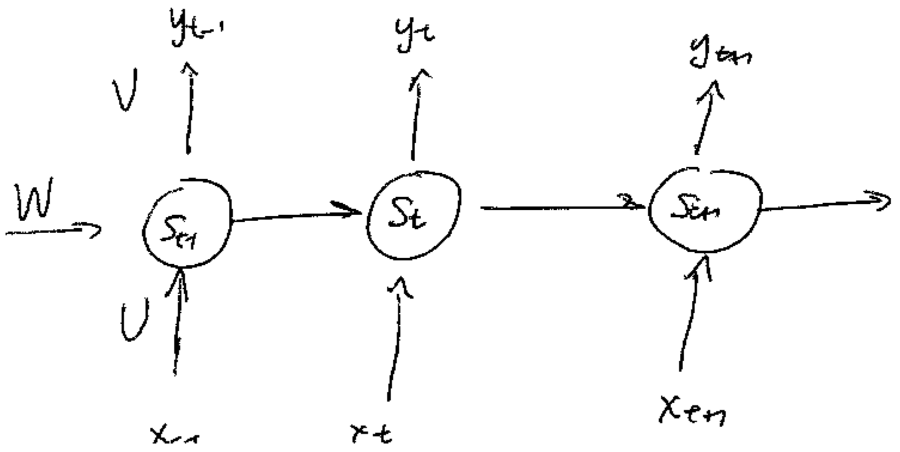
$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{U}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial \boldsymbol{s}_t} \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_i} \frac{\partial \boldsymbol{s}_i}{\partial \boldsymbol{U}}$$

BPTT for output parameters $\partial \mathcal{L}_t / \partial \boldsymbol{V}$

- For the output parameters computations are simpler
  - The parameters $\boldsymbol{V}$ are not influenced by the recurrent state $\boldsymbol{s}_t$

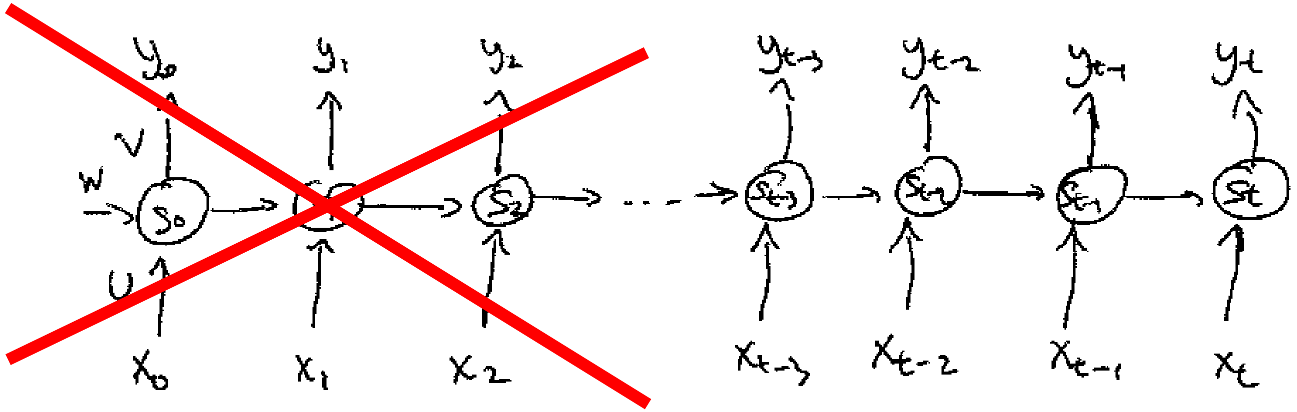
$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{V}} = \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial \boldsymbol{V}}$$

- For one loss per time step, sum over: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{V}} = \sum_t \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{V}}$



Truncating BPTT

- Unrolling forever is not practical or even feasible
- Truncate to t_{trunc} is a usual strategy
  - Then, replace all t in the equations before with t_{trunc}
- More focus on short-term terms
  - Not undesirable, as long-term terms may be irrelevant anyway
  - 'Biases' towards simpler models with shorter-term dependencies



Challenges

- Vanishing gradients
- Exploding gradients
- Misalignment between gradient computations and weight updates
- Bias due to truncation

BPTT: Vanishing and exploding gradients

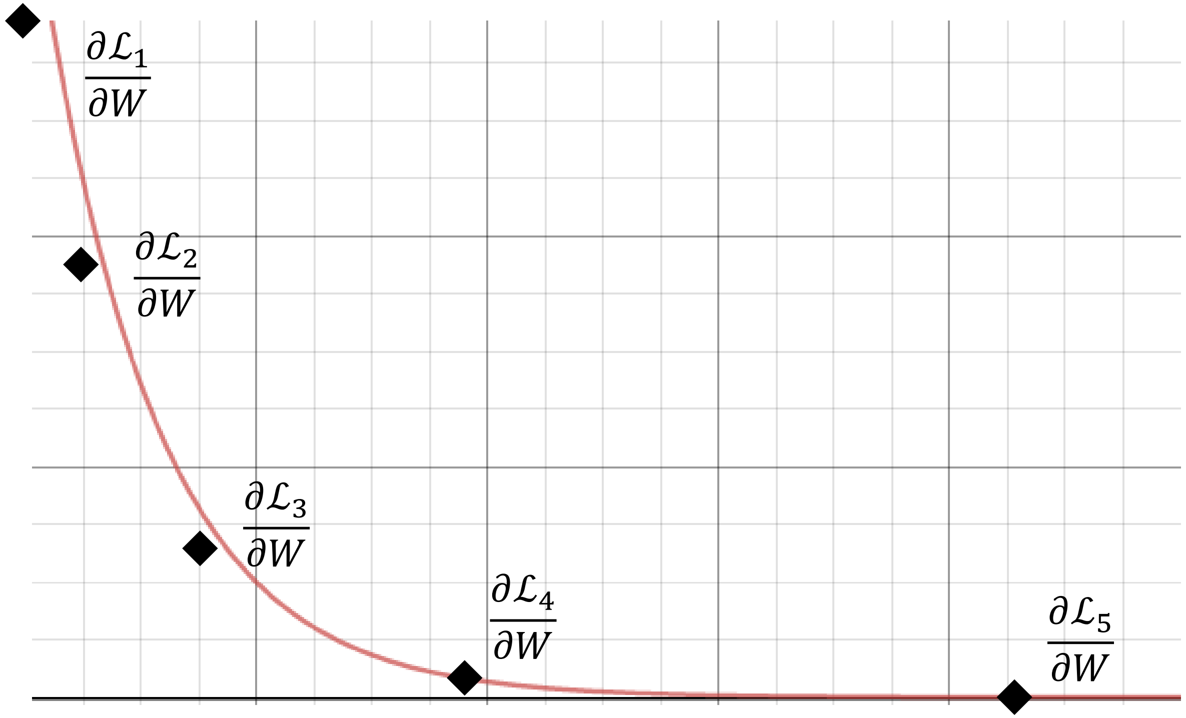
- Gradients vanish or explode even easier because of shared parameters

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{W}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial \boldsymbol{s}_t} \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_i} \frac{\partial \boldsymbol{s}_i}{\partial \boldsymbol{W}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial \boldsymbol{s}_t} \left(\prod_{j=i+1}^{t} \frac{\partial \boldsymbol{s}_j}{\partial \boldsymbol{s}_{j-1}} \right) \frac{\partial \boldsymbol{s}_i}{\partial \boldsymbol{W}}$$

- If $\frac{\partial \boldsymbol{s}_j}{\partial \boldsymbol{s}_{j-1}} < 1 \Rightarrow \frac{\partial \mathcal{L}_t}{\partial W} \ll 1 \rightarrow$ Vanishing gradient
- If $\frac{\partial \boldsymbol{s}_j}{\partial \boldsymbol{s}_{j-1}} > 1 \Rightarrow \frac{\partial \mathcal{L}_t}{\partial W} \gg 1 \rightarrow$ Explodinggradient

BPTT: Vanishing gradients

- Exponentially smaller contribution of longer-term terms
  - Model emphasizes on shorter-term terms as they have larger gradients
- Can be undesirable if the 'distant past' is a key factor



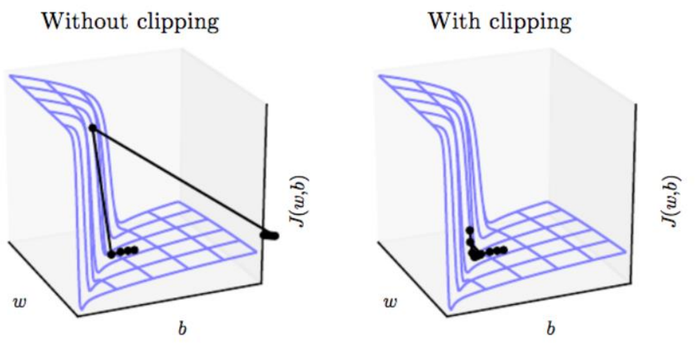
Rescaling gradients to avoid explosions

- Compute the gradient

$$g \leftarrow \frac{\partial \mathcal{L}}{\partial W}$$

- If its norm larger than a threshold γ rescale

$$g \leftarrow \gamma \frac{g}{\|g\|}$$

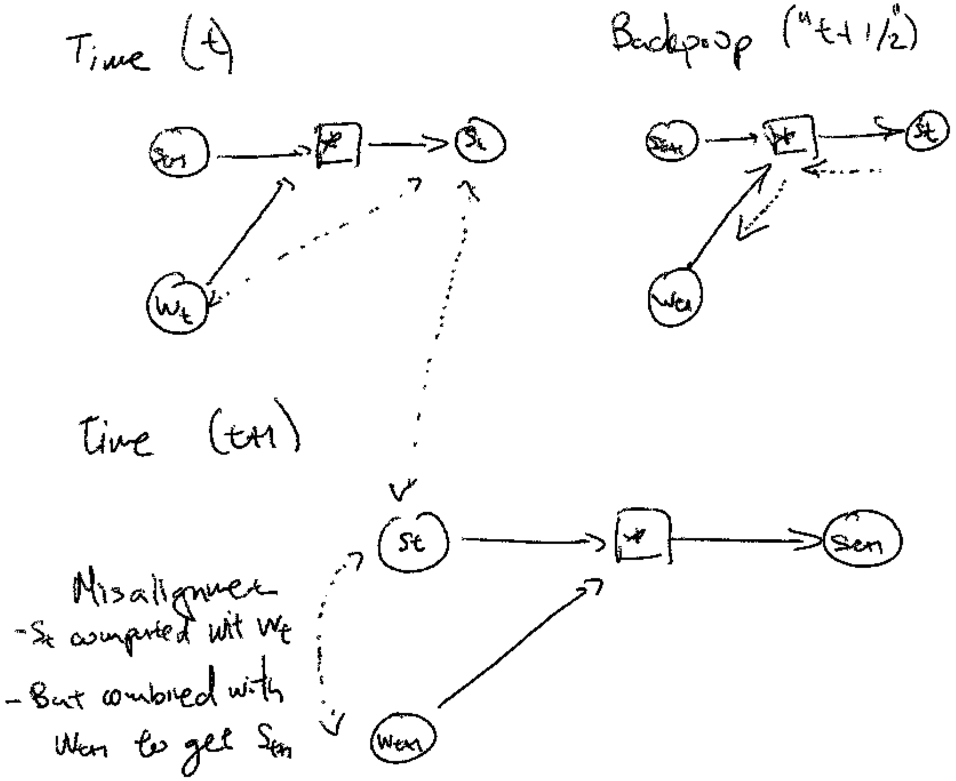


— Goodfellow et al., *Deep Learning*

- It works because exploding gradients are an optimization issue
- We cannot rescale vanishing gradients because it is a gradient accuracy issue
  - A vanishing gradient does not count as a gradient in the first place
- In any case, rescaling still focuses on short-term

BPTT: Misaligning gradients and weights

- For every step we use 'different versions over time' of various variables
- The new gradients are only an estimate
  - Get worse with more backpropagation
- Doing fewer updates helps
  - But might slow down training



BPTT: Truncation bias

- Instead of computing the real gradient for all time steps $0, 1, 2, \dots, t$
  - We compute a gradient approximation up to t_{trunc}
- In practice and for many applications, not much
- It would still be nice to have the model choose what to ignore