CHANGES TODO

- All the back prop stuff has to wait until the lecture on Training
- Can't do LSTM until backprop
- Residual connections depend on back prop
- Visualization OK

Back propagation through time (BPTT)

TL;DR

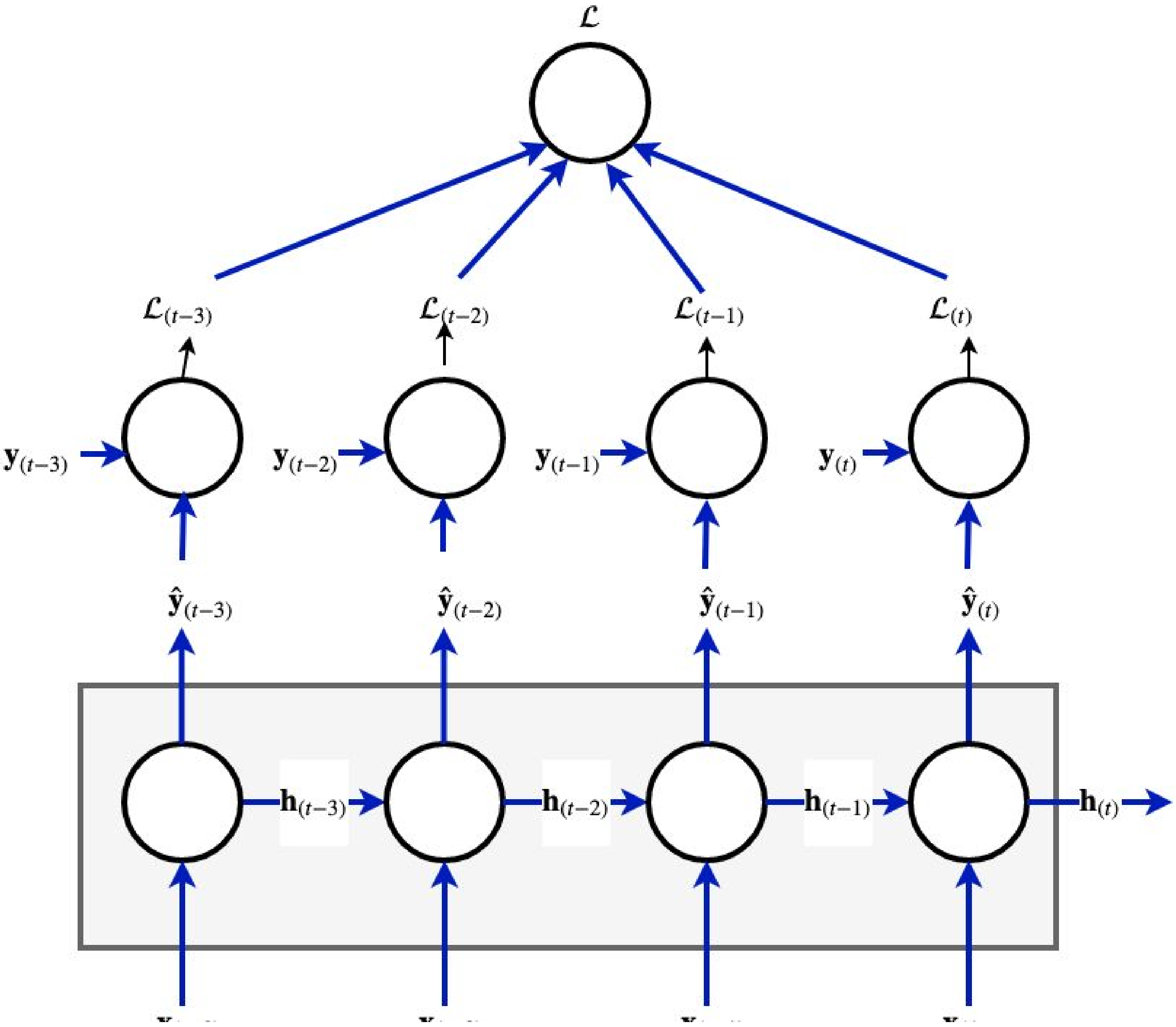
- We can "unroll" the RNN into a sequence of layers, one per time step
- In theory: Back Propagation on the unrolled RNN is the same as for a non-Recurrent Network
- In practice: the unrolled RNN is very deep, which causes issues in Back Propagation.

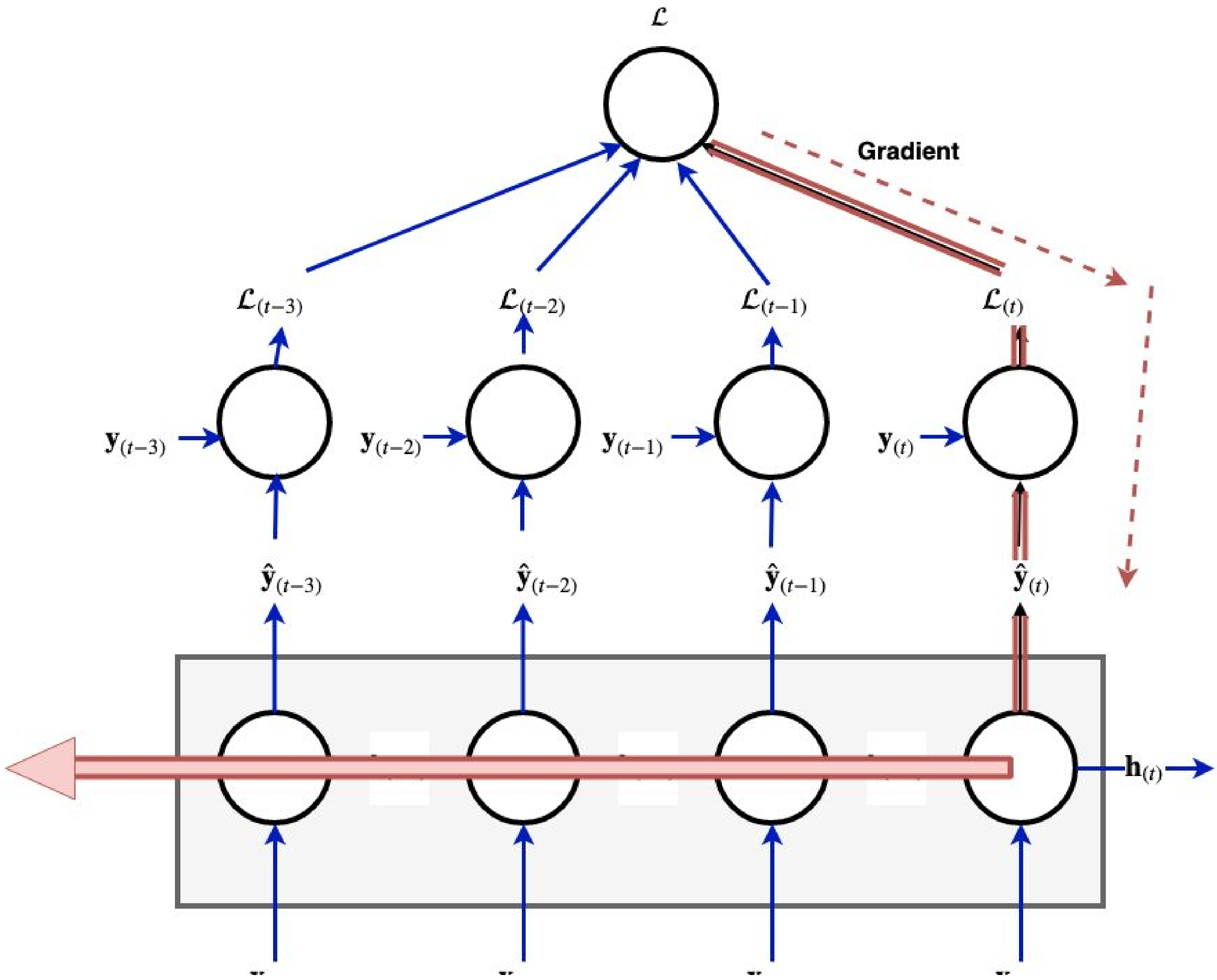
Back Propagation Through Time (BPTT) refers to

- unrolling the RNN computation into a sequence of layers
- performing ordinary Back Propagation in order to update weights

- In a non-Recurrent network:
  - $\mathbf{W}_{(l)}$, the weights of layer l, affect only layers l and greater.
  - This means the backward flow of the gradient with respect to $\mathbf{W}_{(l)}$ stops at layer l.
- In Recurrent Network:
  - All unrolled "layers" share the *same* weights
  - This means the gradients with respect to shared weight $\mathbf{W}$ must flow backward all the way to the input layer at time 0.

RNN Loss: Forward pass


$\mathcal{L}$
$\mathcal{L}_{(t-3)}$
$\mathcal{L}_{(t-2)}$
$\mathcal{L}_{(t-1)}$
$\mathcal{L}_{(t)}$
$\mathbf{y}_{(t-3)}$
$\mathbf{y}_{(t-2)}$
$\mathbf{y}_{(t-1)}$
$\mathbf{y}_{(t)}$
$\hat{\mathbf{y}}_{(t-3)}$
$\hat{\mathbf{y}}_{(t-2)}$
$\hat{\mathbf{y}}_{(t-1)}$
$\hat{\mathbf{y}}_{(t)}$
$\mathbf{h}_{(t-3)}$
$\mathbf{h}_{(t-2)}$
$\mathbf{h}_{(t-1)}$
$\mathbf{h}_{(t)}$


$\mathcal{L}$
Gradient
$\mathcal{L}_{(t-3)}$
$\mathcal{L}_{(t-2)}$
$\mathcal{L}_{(t-1)}$
$\mathcal{L}_{(t)}$
$\mathbf{y}_{(t-3)}$
$\mathbf{y}_{(t-2)}$
$\mathbf{y}_{(t-1)}$
$\mathbf{y}_{(t)}$
$\hat{\mathbf{y}}_{(t-3)}$
$\hat{\mathbf{y}}_{(t-2)}$
$\hat{\mathbf{y}}_{(t-1)}$
$\hat{\mathbf{y}}_{(t)}$
$\mathbf{h}_{(t)}$

The unrolled graph is as deep as the length of $\mathbf{x}^{(\mathbf{i})}$ ($T^{(\mathbf{i})} = |\mathbf{x}^{(\mathbf{i})}|$)

- weights can update only after $T^{(\mathbf{i})}$ input values have been processed, so training can be slow.
- Vanishing Gradients become a concern for large $T^{(\mathbf{i})}$
    - Recall from the Vanishing Gradient lecture: magnitude of gradients diminishes from layer l to layer $(l-1)$ during back propagation

Calculating gradients with BPTT

Back propagation: Refresher

The same math that we used to show how to obtain derivatives (for weight updates in Gradient Descent) will apply to RNN's.

To refresh our memory on notation and results, recall our derivation of back propagation:

Layer l:

- input/output relation of layer l as
  $\mathbf{y}_{(l)} = a_{(l)}(f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)}))$

for

- activation function $a_{(l)}$
- weights $\mathbf{W}_{(l)}$

- $\mathbf{y}_{(l-1)}$ are the outputs of the previous layer

- $f_{(l)}$ is the function computed by layer l

  - function of input $\mathbf{y}_{(l-1)}$ and weights $\mathbf{W}_{(l)}$
  - e.g., `Dense`: $\begin{aligned} f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)}) &= \mathbf{y}_{(l)} \\ &= \mathbf{W}_{(l)}\mathbf{y}_{(l-1)} + \mathbf{b}_{(l)} \end{aligned}$

Note We neglect to add $\mathbf{b}_{(l)}$ as an argument to $f_{(l)}$ to simplify notation

Let

- $\mathcal{L}$ denote loss (computed after final layer L)
- $\mathcal{L}'_{(l)} = \frac{\partial \mathcal{L}}{\partial y_{(l)}}$ denote the derivative of $\mathcal{L}$ with respect to the output of layer l, i.e., $y_{(l)}$,
    - refer to as **loss gradient** (at output of layer l)

We showed how to compute

- $\mathcal{L}'_{(l-1)}$ from $\mathcal{L}'_l$
    - so that we can continue this process as the previous layer (i.e, *propogate loss gradient backwards*)

and we showed how to compute the weight update

- $\frac{\partial \mathcal{L}}{\partial W_{(l)}}$, from $\mathcal{L}'_{(l)}$ for $l \in [1, L]$

Note that $\mathbf{y}_{(l)}$ is a function of

- $\mathbf{y}_{(l-1)}$ (the output of the previous layer)
- and $\mathbf{W}_{(l)}$, the parameters of layer l.

We can compute derivatives of $\mathbf{y}_{(l)}$ with respect to each of its inputs

- $\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$
- $\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}}$

Refer to these as **local gradients**

We used the chain rule to obtain the

- gradient with respect to weights $\mathbf{W}_{(l)}$, given the loss gradient $\mathcal{L}'_{(l)}$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}} = \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}}$$

That is:

- gradient of $\mathcal{L}$ with respect to weight $\mathbf{W}_{(l)}$
- is the loss gradient (at current step), multiplied by
- a local gradient (with respect to input $W_{(l)}$)

So we have the information required to update $\mathbf{W}_{(l)}$ by Gradient Descent.

BPTT: gradient calculation

Let us adapt these results for the case of a *single layer* RNN

- by "unrolling" this RNN, layer l is equated with "time" (of index into input sequence) t

Per example loss $\mathcal{L}^{(\mathbf{i})}$ is now a per example loss *per time step*

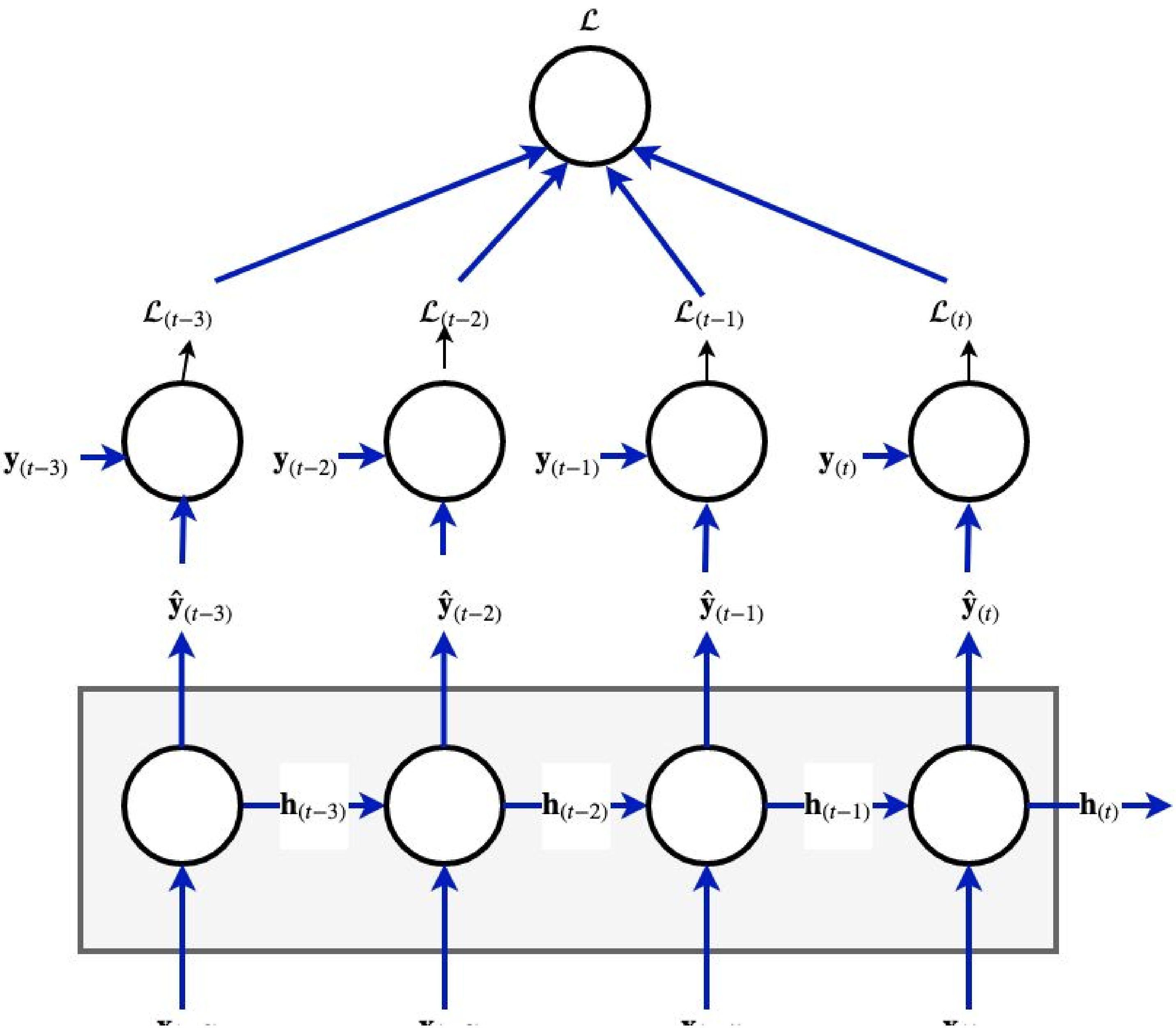
$$\mathcal{L}^{(\mathbf{i})}_{(t)}$$

so

$$\mathcal{L}^{(\mathbf{i})} = \sum_{t=1}^{T} \mathcal{L}^{(\mathbf{i})}_{(t)}$$

We will focus on the per example loss for a single time $\mathcal{L}^{(\mathbf{i})}_{(t)}$

RNN Loss



$\mathcal{L}$
$\mathcal{L}_{(t-3)}$
$\mathcal{L}_{(t-2)}$
$\mathcal{L}_{(t-1)}$
$\mathcal{L}_{(t)}$
$\mathbf{y}_{(t-3)}$
$\mathbf{y}_{(t-2)}$
$\mathbf{y}_{(t-1)}$
$\mathbf{y}_{(t)}$
$\hat{\mathbf{y}}_{(t-3)}$
$\hat{\mathbf{y}}_{(t-2)}$
$\hat{\mathbf{y}}_{(t-1)}$
$\hat{\mathbf{y}}_{(t)}$
$\mathbf{h}_{(t-3)}$
$\mathbf{h}_{(t-2)}$
$\mathbf{h}_{(t-1)}$
$\mathbf{h}_{(t)}$

As per regular backprop, we can obtain the loss update by multiplying the loss gradient by a local gradient

$$\frac{\partial \mathcal{L}_{(t)}^{(\mathbf{i})}}{\partial \mathbf{W}}$$

but note that we use unsubscripted $\mathbf{W}$ (rather than $\mathbf{W}_{(t)}$ because the *same* $\mathbf{W}$ is used at all timesteps.

$$\frac{\partial \mathcal{L}^{(\mathbf{i})}_{(t)}}{\partial \mathbf{W}} = \mathcal{L}'_{(t)} \frac{\partial \mathbf{y}_{(t)}}{\partial W}$$

but now

$$\frac{\partial \mathbf{y}_{(t)}}{\partial W}$$

becomes more complicated, governed by the RNN Update equations

$$
\begin{aligned}
\mathbf{h}_{(t)} &= \phi(\mathbf{W}_{xh}\mathbf{x}_{(t)} + \mathbf{W}_{hh}\mathbf{h}_{(t-1)} + \mathbf{b}_h) \\
\mathbf{y}_{(t)} &= \mathbf{W}_{hy}\mathbf{h}_{(t)} + \mathbf{b}_y
\end{aligned}
$$

Notes

- In this section we will assume ϕ is the identity function to simplify the presentation.
    - There will be no loss of generality.
- Recall that $\mathbf{W}$ is the matrix with embedded sub-matrices $\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{W}_{hy}$
    - For clarity: we will add subscripts to $\mathbf{W}$ in the derivatives to show which part of $\mathbf{W}$ is the cause.

The equation defining $\mathbf{y}_{(t)}$

$$\mathbf{y}_{(t)} = \mathbf{W}_{hy}\mathbf{h}_{(t)} + \mathbf{b}_y$$

shows that $\mathbf{y}_{(t)}$ is

- directly depends on $\mathbf{W}$ (through $\mathbf{W}_{hy}$)
- *and* indirectly depends on $\mathbf{W}$ through its dependence on $h_{(t)}$ (which depends on $\mathbf{W}$)

So

$$\frac{\partial \mathbf{y}_{(t)}^{(\mathbf{i})}}{\partial \mathbf{W}} = \frac{\partial \mathbf{y}_{(t)}^{(\mathbf{i})}}{\partial \mathbf{W}_{hy}} + \frac{\partial \mathbf{y}_{(t)}^{(\mathbf{i})}}{\partial \mathbf{h}_{(t)}} \frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}}$$

Let's expand the term

$$\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}}$$

Recall the recursive definition of $\mathbf{h}_{(t)}$

$$\mathbf{h}_{(t)} = \mathbf{W}_{xh}\mathbf{x}_{(t)} + \mathbf{W}_{hh}\mathbf{h}_{(t-1)} + \mathbf{b}_h$$

$\mathbf{h}_{(t)}$ depends on $\mathbf{h}_{(t-1)}$, which by recursion depends on $\mathbf{h}_{(t-2)}$ which $\ldots$ depends on $\mathbf{h}_{(0)}$.

- and all $\mathbf{h}_{(t)}$ share the *same* $\mathbf{W}_{hh}$.

This means that $\mathbf{h}_{(t)}$ depends on $\mathbf{W}$ through *each* $\mathbf{h}_{(t-k)}$ for $k = 1, \ldots, t$.

$$\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}} = \sum_{k=1}^{t} \frac{\partial \mathbf{h}_{(t-k)}}{\partial \mathbf{W}_{hh}} \frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-k)}}$$

So

$$\frac{\partial \mathcal{L}^{(\mathbf{i})}_{(t)}}{\partial \mathbf{W}} = \mathcal{L}'_{(t)} \frac{\partial \mathbf{y}_{(t)}}{\partial W}$$

and

$$\frac{\partial \mathbf{y}^{(\mathbf{i})}_{(t)}}{\partial W}$$

depends on all time steps from 1 to t.

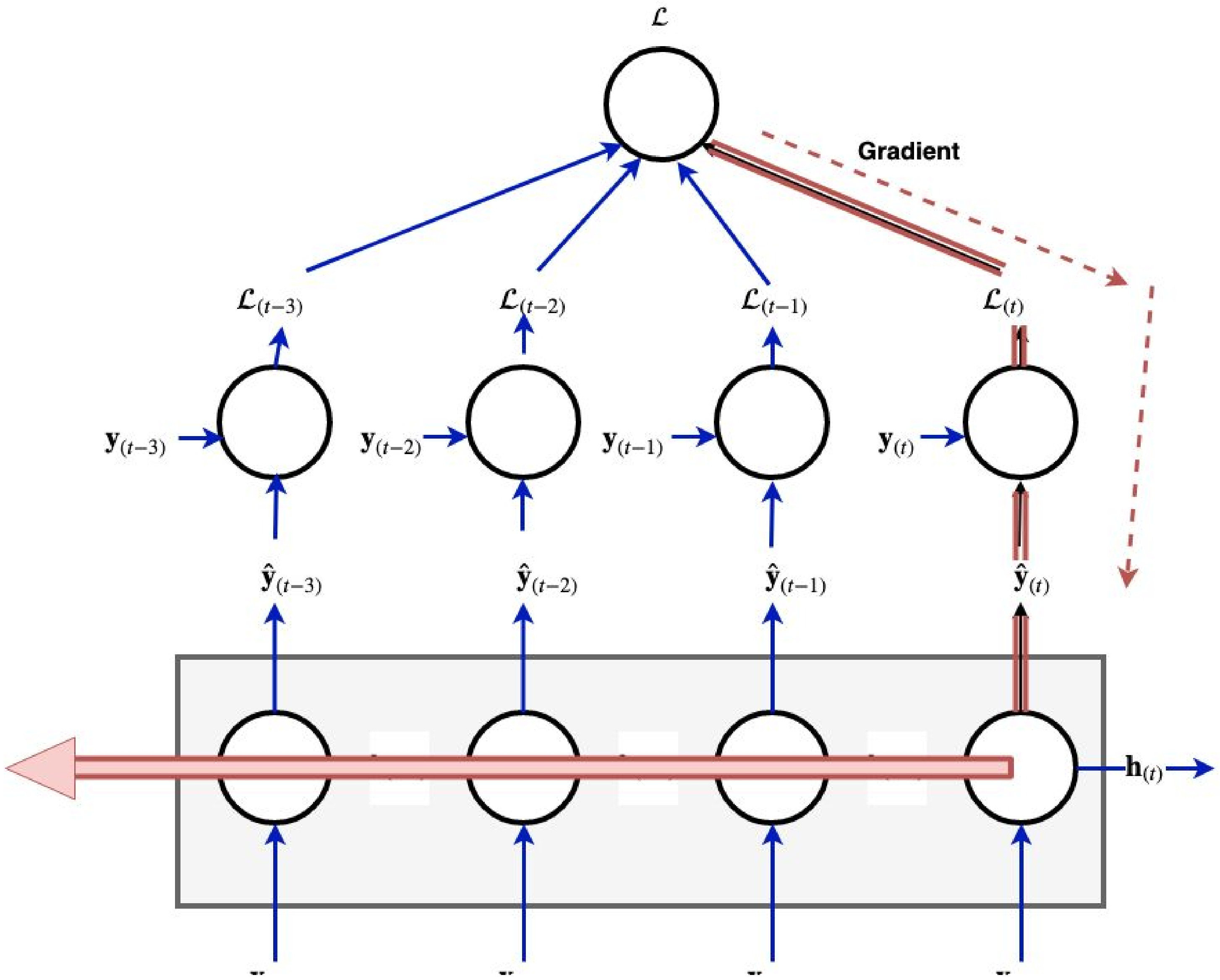
Thus, the derivative update for $\mathbf{W}$ cannot be computed without the gradient (for each time step t) flowing all the way back to time step 0.

Note

Directly expanding the recursion would show

$$\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-k)}} = \prod_{k'=0}^{k-1} \frac{\partial \mathbf{h}_{(t-k')}}{\partial \mathbf{h}_{(t-k'-1)}}$$

It is not necessary now, but will be useful in explaining vanishing/exploding gradients



$\mathcal{L}$
Gradient
$\mathcal{L}_{(t-3)}$
$\mathcal{L}_{(t-2)}$
$\mathcal{L}_{(t-1)}$
$\mathcal{L}_{(t)}$
$\mathbf{y}_{(t-3)}$
$\mathbf{y}_{(t-2)}$
$\mathbf{y}_{(t-1)}$
$\mathbf{y}_{(t)}$
$\hat{\mathbf{y}}_{(t-3)}$
$\hat{\mathbf{y}}_{(t-2)}$
$\hat{\mathbf{y}}_{(t-1)}$
$\hat{\mathbf{y}}_{(t)}$
$\mathbf{h}_{(t)}$

Truncated back progagation through time (TBTT)

TL;DR

- We "unroll" the RNN into a sequence of T layers, one per time step
- We compute the loss at each time step t, for t=1 to T.
- The gradient of the loss of time step t flows backward for a limited number of time steps
    - Rather than flowing backwards al the way to time step 0

This is called *Truncated BPTT (TBPTT)*.

The advantage of TBPTT

- more frequent gradient updates

The disadvantage

- the loss at step t won't affect **all** previous time steps (because of truncation)
- the error signal from time t does not affect any time steps below $t - \tau$.
- this means the RNN has difficulty capturing dependencies longer than τ.

Consider a long piece of text

- The first few words indicate the gender/plurality/age of the subject
- A mis-prediction of, e.g. gender, at word $\tau' > \tau$ causes an error at time step τ'
    - which can't interact with the correct gender in the first few words

Note that there is *no truncation* of the forward pass of the RNN !

Only gradient calculations are truncated.

TBTT: Variations

There are several ways to truncate the Back Propagation.

We will describe them via a function $f(t) = t'$

- describes the earliest time step affecting the gradient of $\mathcal{L}_{(t)}$
- that is, it describes the window τ

- Untruncated BPTT
  - $f(t) = 0$
- k-truncated BPTT
  - $f(t) = \max(0, t - k)$
- subsequence truncated BPTT
  - $f(t) = k * \lfloor t/k \rfloor$

What we refer to as subsequence TBTT seems to be common

- break long sequence $\mathbf{x^{(i)}}$ into subsequences (chunks) of size k
- feed $\mathbf{x^{(i)}}$ forward as usual
    - at the end of a subsequence:
        - immediately compute the loss gradients for all time steps within the chunk

RNN vanishing/exploding gradient problem

TL;DR

- A "single-layer RNN that has been unrolled for T time steps
    - is mathematically equivalent to a simple NN with T layers
    - BUT all layers share the same weights
- This sharing of weights leads to a problem of Vanishing/Exploding gradients
    - Similar to the vanishing gradient problem we derived for simple NN
    - but with a different root cause (weight sharing)

TL;DR

- Why shared weights are different
    - Output y at time t is a function of cell state h at time t
    - Cell state h at time t is recursively defined
        - So it is a function of cell states over all times t' < t as well
        - This means the weight update involves a repeated product: (t - t') times
        - This product tends to 0 (vanishing) or infinity (explode) as (t -t') increases
    - So losses at time step t have difficulty updating gradients for the distant past<
    - RNN has difficulty with long-term dependencies

Returning to the loss gradient we encountered the terms

$$\frac{\partial \mathbf{y}_{(t)}^{(\mathbf{i})}}{\partial \mathbf{W}}$$

We will focus on the part of $\mathbf{W}$ that is $\mathbf{W}_{hh}$

$$\frac{\partial \mathbf{y}_{(t)}}{\partial W_{hh}} = \frac{\partial \mathbf{y}_{(t)}}{\partial \mathbf{h}_{(t)}} \frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}}$$

But recursively defined $\mathbf{h}_{(t)}$ is a function of $\mathbf{h}_{(t-1)}, \mathbf{h}_{(t-1)}, \ldots, \mathbf{h}_{(1)}$ so

$$\frac{\partial \mathbf{y}_{(t)}}{\partial W_{hh}} = \frac{\partial \mathbf{y}_{(t)}}{\partial \mathbf{h}_{(t)}} \sum_{k=0}^{t} \frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-k)}} \frac{\partial \mathbf{h}_{(t-k)}}{\partial \mathbf{W}_{hh}}$$

The summation: $\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}}$, through all intermediate $\mathbf{h}_{(t-k)}$

The problematic term for us is

$$\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-k)}}$$

It can be computed by the Chain Rule as

$$\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-k)}} = \prod_{u=0}^{t-1} \frac{\partial \mathbf{h}_{(t-u)}}{\partial \mathbf{h}_{(t-u-1)}}$$

Each term

$$\frac{\partial \mathbf{h}_{(t-u)}}{\partial \mathbf{h}_{(t-u-1)}}$$

results in a term $\mathbf{W}_{hh}$ so the repeated product compute matrix $\mathbf{W}_{hh}$ raised to the power k.

For simplicity, suppose $\mathbf{W}_{hh}$ were a scalar

- if $\mathbf{W}_{hh} < 1$ then repeatedly multiply $\mathbf{W}_{hh}$ by itself approaches 0
- if $\mathbf{W}_{hh} > 1$ then repeatedly multiply $\mathbf{W}_{hh}$ by itself approaches ∞

In other words:

- as the distance between time steps t and $(t-k)$ increases
- the gradient (for the weight update) either vanishes or explodes.

Since this term is used in the update for our weights

- updates will either be erratic (too big)
- or non-existent, hampering learning of weights.

This was not necessarily a problem in non-recurrent networks

- because each layer had a different weight matrix.

What an RNN does that helps it be parsimonius in number of parameters

- by sharing the weights across all time steps
- hurts us in learning.

For the general case where $\mathbf{W}_{hh}$ is a matrix

- we can show the same resul with the eigenvalues of the matrix

Controlling exploding gradients by clipping

In theory, we can control the explosion by clipping the gradient $\frac{\partial \mathcal{L}}{\partial W_i}$.

We are still left with the vanishing gradient problem.

This means that we can't learn long-term dependencies (i.e., too many steps backward).

This will be "solved" by introducing recurrent architectures that address this issue.

In [3]:

```
print("Done")
```

```
Done
```