Backprop through time: RNN and LSTM gradients

Ten problems on backpropagation through time: the one-step Jacobians of a vanilla RNN, the backward recursion for the hidden-state gradients, the weight gradients as sums over time, a two-step example by hand, vanishing and exploding gradients as a product of Jacobians, many-to-one classification, and the LSTM cell's backward pass and cell-state path, with worked solutions and the mistakes that drop a time step or a factor.

BEFORE YOU START

A recurrent network applies the same layer at every step of a sequence, feeding each step's hidden state into the next. Unrolled over T steps it is a T-layer network whose layers share their weights, and backpropagation through time is ordinary backprop on that unrolled network. Two things make it different from the earlier pages: each hidden state reaches the loss along two routes, through its own output and through the next step, and each weight is used T times, so its gradient is a sum over time. These ten problems derive the backward recursion and the weight gradients, check them on a two-step example by hand, show where vanishing and exploding gradients come from, and then work through the LSTM cell, whose cell state is built to avoid them. The five mistakes are the ones that give a gradient of the right shape: one time step instead of all of them, h_t where h_{t-1} belongs, a missing $\tanh'$, W where $W^\top$ belongs, and an LSTM cell gradient without its path to the next cell.

- The conventions are those of the previous pages: vectors are columns, Jacobians are in numerator layout, a gradient has the shape of the variable it is taken with respect to, and for a scalar L of u with u a function of x, $\nabla_x L = (\partial u/\partial x)^\top \nabla_u L$. $\odot$ is the elementwise product.
- The network: inputs $x_1, \dots, x_T \in \mathbb{R}^d$ and an initial state $h_0 \in \mathbb{R}^m$. For $t = 1, \dots, T$, $a_t = W h_{t-1} + U x_t + b$, $h_t = \tanh(a_t)$ and $z_t = V h_t + c$, with $\tanh$ applied elementwise, $h_t \in \mathbb{R}^m$ the hidden state and $z_t \in \mathbb{R}^k$ the output. The same W, U, b, V, c are used at every step.
- The loss is $L = \sum_{t=1}^{T} \ell_t(z_t)$, where each ℓ_t is a scalar loss of step t's output alone; a step with no target has $\ell_t = 0$.
- $\tanh'(u) = 1 - \tanh^2(u)$, so the derivative of h_t with respect to a_t, entry by entry, is $1 - h_t \odot h_t$ (1 minus each entry of $h_t \odot h_t$).
- Three gradients per step: $e_t := \nabla_{z_t} \ell_t$, the gradient of step t's own loss at its output; $g_t := \nabla_{h_t} L$, the gradient of the whole loss at h_t, including every later step; and $\delta_t := \nabla_{a_t} L$.

- $\|A\|_2$ is the spectral norm, the largest singular value of A. It satisfies $\|AB\|_2 \le \|A\|_2\|B\|_2$, and for a diagonal matrix it is the largest absolute value on the diagonal.

Builds on: Jacobians and the chain rule, One-hidden-layer backprop, the whole backward pass

Problems

1. Give the shapes of W, U, b, V, c and the number of parameters. Which of $h_0, x_1, \dots, x_T$ does h_t depend on? difficulty ·

2. Compute the Jacobians $\partial h_t / \partial h_{t-1}$ and $\partial h_t / \partial x_t$. difficulty ··

3. Derive the backward recursion for g_t and δ_t, starting at $t = T$. difficulty ··

4. Compute $\nabla_W L$, $\nabla_U L$, $\nabla_b L$, $\nabla_V L$ and $\nabla_c L$ in terms of the δ_t, e_t, h_t and x_t. difficulty ···

5. By hand: a scalar linear RNN $h_t = w\, h_{t-1} + u\, x_t$ (no tanh, no bias, $m = d = 1$), with $h_0 = 0$, $x_1 = 1$, $x_2 = 2$, $w = \frac{1}{2}$, $u = 1$, and a loss at the last step only, $L = \frac{1}{2} h_2^2$. Compute $\partial L / \partial w$ and $\partial L / \partial u$ with the recursion, then check them by writing L as a function of w and u. difficulty ··

6. For $t < T$, write $\partial h_T / \partial h_t$ as a product of the Jacobians of Problem 2, and show that $\|\partial h_T / \partial h_t\|_2 \le \|W\|_2^{T-t}$. What does this say about how the loss at step T reaches g_t when $\|W\|_2 < 1$? difficulty ···

7. Many-to-one: a classifier reads the whole sequence and is scored once. $\ell_t = 0$ for $t < T$, and $\ell_T = -\sum_j y_j \log s_j$ with $s = \mathrm{softmax}(z_T)$ and y one-hot. Give the δ_t, $\nabla_W L$ and $\nabla_V L$. difficulty ··

8. Compute $\nabla_{x_t} L$ and $\nabla_{h_0} L$. When is $\nabla_{h_0} L$ needed? difficulty ··

9. An LSTM keeps a cell state $c_t \in \mathbb{R}^m$ beside h_t: difficulty ···

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

and

$$h_t = o_t \odot \tanh(c_t),$$

where the forget, input and output gates $f_t, i_t, o_t \in (0,1)^m$ and the candidate $\tilde{c}_t \in (-1,1)^m$ are computed from h_{t-1} and x_t. Treat these four as inputs to the cell. You are given dh_t, the total gradient of L at h_t, and $\bar{c}_t$, the gradient that reaches c_t from step $t+1$ through c_{t+1}. Compute dc_t, the total gradient at c_t, and the gradients to c_{t-1} (through this cell), f_t, i_t, $\tilde{c}_t$ and o_t.

10. Follow the cell states alone: holding every gate and candidate fixed, compute $\partial c_T / \partial c_t$ for $t < T$, and compare it with Problem 6. difficulty ···

Answers

1. $W : m \times m$, $U : m \times d$, $b : m$, $V : k \times m$, $c : k$; $m^2 + md + m + km + k$ parameters, whatever T is; h_t depends on h_0 and $x_1, \dots, x_t$, and on no later input
2. $\partial h_t / \partial h_{t-1} = \operatorname{diag}(1 - h_t \odot h_t)\, W$ and $\partial h_t / \partial x_t = \operatorname{diag}(1 - h_t \odot h_t)\, U$
3. $g_T = V^\top e_T$; $g_t = V^\top e_t + W^\top \delta_{t+1}$ for $t < T$; $\delta_t = (1 - h_t \odot h_t) \odot g_t$
4. $\nabla_W L = \sum_t \delta_t h_{t-1}^\top$, $\nabla_U L = \sum_t \delta_t x_t^\top$, $\nabla_b L = \sum_t \delta_t$, $\nabla_V L = \sum_t e_t h_t^\top$, $\nabla_c L = \sum_t e_t$
5. $\partial L / \partial w = \frac{5}{2}$ and $\partial L / \partial u = \frac{25}{4}$
6. $\dfrac{\partial h_T}{\partial h_t} = D_T W\, D_{T-1} W \cdots D_{t+1} W$ with $D_s = \operatorname{diag}(1 - h_s \odot h_s)$, and $\left\| \partial h_T / \partial h_t \right\|_2 \le \|W\|_2^{T-t}$
7. $\delta_T = (1 - h_T \odot h_T) \odot V^\top (s - y)$, $\delta_t = (1 - h_t \odot h_t) \odot W^\top \delta_{t+1}$ for $t < T$; $\nabla_W L = \sum_t \delta_t h_{t-1}^\top$ and $\nabla_V L = (s - y)\, h_T^\top$
8. $\nabla_{x_t} L = U^\top \delta_t$ and $\nabla_{h_0} L = W^\top \delta_1$
9. $dc_t = \bar{c}_t + dh_t \odot o_t \odot (1 - \tanh(c_t) \odot \tanh(c_t))$; $f_t \odot dc_t$ to c_{t-1}; $df_t = c_{t-1} \odot dc_t$, $di_t = \tilde{c}_t \odot dc_t$, $d\tilde{c}_t = i_t \odot dc_t$, $do_t = \tanh(c_t) \odot dh_t$
10. $\partial c_T / \partial c_t = \operatorname{diag}(f_{t+1} \odot f_{t+2} \odot \cdots \odot f_T)$

Worked solutions

Problem 1

> Give the shapes of W, U, b, V, c and the number of parameters. Which of $h_0, x_1, \dots, x_T$ does h_t depend on?

$Wh_{t-1} \in \mathbb{R}^m$ for $h_{t-1} \in \mathbb{R}^m$, so W is $m \times m$; $Ux_t \in \mathbb{R}^m$ for $x_t \in \mathbb{R}^d$, so U is $m \times d$; $b \in \mathbb{R}^m$.	All three terms of a_t are added to give the m entries of a_t, so each must be an m-vector.
V is $k \times m$ and $c \in \mathbb{R}^k$.	$z_t = Vh_t + c$ maps the m-vector h_t to the k-vector z_t.
The count is $m^2 + md + m + km + k$.	The sizes of the five arrays. None of them has a time index, because the same arrays are reused at every step.
$h_t = \tanh(Wh_{t-1} + Ux_t + b)$ and, unrolling, h_{t-1} depends on $h_0, x_1, \dots, x_{t-1}$.	Each step adds one new input to what the previous state already carried; by induction from $h_1 = \tanh(Wh_0 + Ux_1 + b)$.
$W : m \times m$, $U : m \times d$, $b : m$, $V : k \times m$, $c : k$; $m^2 + md + m + km + k$ **parameters, whatever T is;** h_t **depends on** h_0 **and** $x_1, \dots, x_t$**, and on no later input**	The parameter count does not grow with the sequence length, which is the point of sharing the weights; the price is that every weight gradient is a sum over steps (Problem 4).

Problem 2

> Compute the Jacobians $\partial h_t / \partial h_{t-1}$ and $\partial h_t / \partial x_t$.

$\partial a_t / \partial h_{t-1} = W$ and $\partial a_t / \partial x_t = U$.	a_t is affine in each of h_{t-1} and x_t, and the Jacobian of $Mv + \text{const}$ is M.

$\partial h_t / \partial a_t = \operatorname{diag}(1 - h_t \odot h_t)$.	$\tanh$ is elementwise, so its Jacobian is diagonal, with $\tanh'(a_{t,i}) = 1 - h_{t,i}^2$ on the diagonal; the forward pass has already computed h_t.
$\partial h_t / \partial h_{t-1} = \operatorname{diag}(1 - h_t \odot h_t)\, W$ **and** $\partial h_t / \partial x_t = \operatorname{diag}(1 - h_t \odot h_t)\, U$	The chain rule in numerator layout puts the outer Jacobian on the left: $(m \times m)(m \times m)$ and $(m \times m)(m \times d)$, the shapes of an m-vector differentiated by an m-vector and by a d-vector.

Problem 3

> Derive the backward recursion for g_t and δ_t, starting at $t = T$.

h_T reaches L only through z_T, so $g_T = (\partial z_T / \partial h_T)^\top e_T = V^\top e_T$.	There is no step $T+1$, and the later losses are the only other way h_T could matter.
For $t < T$, h_t reaches L through z_t and through a_{t+1}, and through nothing else.	h_t is used in exactly two places in the forward pass: the output of step t and the pre-activation of step $t+1$.
$g_t = V^\top e_t + (\partial a_{t+1} / \partial h_t)^\top \delta_{t+1} = V^\top e_t + W^\top \delta_{t+1}$.	The chain rule adds the two routes of step 2; δ_{t+1} is the total gradient at a_{t+1}, so it already carries every step after $t+1$.
$\delta_t = (\partial h_t / \partial a_t)^\top g_t = (1 - h_t \odot h_t) \odot g_t$.	a_t reaches L only through h_t, and a diagonal Jacobian acts as an elementwise product (Problem 2, step 2).
$g_T = V^\top e_T$; $g_t = V^\top e_t + W^\top \delta_{t+1}$ **for** $t < T$; $\delta_t = (1 - h_t \odot h_t) \odot g_t$	The recursion runs backwards from $t = T$ to $t = 1$, one step per forward step, so the backward pass costs about as much as the forward pass; it needs every h_t, which is why training stores the whole sequence of states.

Problem 4

> Compute $\nabla_W L$, $\nabla_U L$, $\nabla_b L$, $\nabla_V L$ and $\nabla_c L$ in terms of the δ_t, e_t, h_t and x_t.

Give step t its own copy $W^{(t)}$ of W, used only in a_t; then $\nabla_W L = \sum_t \nabla_{W^{(t)}} L$ evaluated at $W^{(t)} = W$.	L depends on W only through its T uses, and the chain rule adds the contributions of every use: this is the unrolled network with its shared weights untied.
$W_{ij}^{(t)}$ appears only in $a_{t,i}$, with coefficient $h_{t-1,j}$, so $\partial L / \partial W_{ij}^{(t)} = \delta_{t,i}\, h_{t-1,j}$, that is $\nabla_{W^{(t)}} L = \delta_t h_{t-1}^\top$.	The copy $W^{(t)}$ reaches L only through a_t, and δ_t is the total gradient there; h_{t-1} is an input to step t, so it is not differentiated here, and its own dependence on W is the other copies' contribution.
In the same way $\nabla_{U^{(t)}} L = \delta_t x_t^\top$ and $\nabla_{b^{(t)}} L = \delta_t$.	$U_{ij}^{(t)}$ multiplies $x_{t,j}$ into $a_{t,i}$, and $b^{(t)}$ is added to a_t with coefficient 1.
$\nabla_{V^{(t)}} L = e_t h_t^\top$ and $\nabla_{c^{(t)}} L = e_t$.	$V^{(t)}$ and $c^{(t)}$ appear only in z_t, which reaches L only through ℓ_t, so the gradient at z_t is e_t; the pattern is the outer product of the one-hidden-layer page.
$\nabla_W L = \sum_t \delta_t h_{t-1}^\top$, $\nabla_U L = \sum_t \delta_t x_t^\top$, $\nabla_b L = \sum_t \delta_t$, $\nabla_V L = \sum_t e_t h_t^\top$, $\nabla_c L = \sum_t e_t$	Sums of the copies' gradients, steps 1 to 4, each with the shape of its parameter: $\delta_t h_{t-1}^\top$ is $(m \times 1)(1 \times m)$. The output weights see only the local e_t; the recurrent weights see the δ_t, which carry the future.

Problem 5

> By hand: a scalar linear RNN $h_t = w\, h_{t-1} + u\, x_t$ (no tanh, no bias, $m = d = 1$), with $h_0 = 0$, $x_1 = 1$, $x_2 = 2$, $w = \frac{1}{2}$, $u = 1$, and a loss at the last step only, $L = \frac{1}{2} h_2^2$. Compute $\partial L / \partial w$ and $\partial L / \partial u$ with the recursion, then check them by writing L as a function of w and u.

$h_1 = \frac{1}{2} \cdot 0 + 1 \cdot 1 = 1$ and $h_2 = \frac{1}{2} \cdot 1 + 1 \cdot 2 = \frac{5}{2}$.	The forward pass, which the backward pass needs.

$\delta_2 = g_2 = h_2 = \frac{5}{2}.$	$\frac{d}{dh}\frac{1}{2}h^2 = h$, and without a tanh the factor $1 - h \odot h$ becomes 1, so $\delta_t = g_t$.
$\delta_1 = g_1 = w\,\delta_2 = \frac{5}{4}.$	Problem 3, step 3, with no loss at step 1 ($e_1 = 0$) and $W^\top = w$.
$\partial L/\partial w = \delta_2 h_1 + \delta_1 h_0 = \frac{5}{2}\cdot 1 + \frac{5}{4}\cdot 0 = \frac{5}{2}.$	Problem 4: one term per step, each pairing δ_t with the state that came into step t.
$\partial L/\partial u = \delta_2 x_2 + \delta_1 x_1 = \frac{5}{2}\cdot 2 + \frac{5}{4}\cdot 1 = \frac{25}{4}.$	Problem 4 with x_t in place of h_{t-1}.
Directly: $h_2 = w\,ux_1 + ux_2 = u(w+2)$, so $L = \frac{1}{2}u^2(w+2)^2$, $\partial L/\partial w = u^2(w+2) = \frac{5}{2}$ and $\partial L/\partial u = u(w+2)^2 = \frac{25}{4}$.	Substituting h_1 into h_2 removes the recursion; the two routes agree.
$\partial L/\partial w = \frac{5}{2}$ **and** $\partial L/\partial u = \frac{25}{4}$	The $\frac{5}{4}$ in $\partial L/\partial u$ is the contribution of x_1 through two steps; a gradient that stopped at the last step would be 5.

Problem 6

> For $t < T$, write $\partial h_T/\partial h_t$ as a product of the Jacobians of Problem 2, and show that $\|\partial h_T/\partial h_t\|_2 \le \|W\|_2^{T-t}$. What does this say about how the loss at step T reaches g_t when $\|W\|_2 < 1$?

$$\frac{\partial h_T}{\partial h_t} = \frac{\partial h_T}{\partial h_{T-1}}\frac{\partial h_{T-1}}{\partial h_{T-2}}\cdots\frac{\partial h_{t+1}}{\partial h_t}.$$	h_T depends on h_t only through the chain $h_{t+1}, \dots, h_{T-1}$, with the inputs x_s fixed; numerator layout puts the latest step on the left.
Each factor is $D_s W$ with $D_s = \operatorname{diag}(1 - h_s \odot h_s)$, $s = t+1, \dots, T$.	Problem 2.
$\|D_s\|_2 \le 1.$	D_s is diagonal with entries $1 - h_{s,i}^2$, and $h_{s,i} = \tanh(a_{s,i}) \in (-1, 1)$ puts every entry in $(0, 1]$.

$\left\| \prod_s D_s W \right\|_2 \le \prod_s \|D_s\|_2 \, \|W\|_2 \le \|W\|_2^{T-t}$.	Submultiplicativity applied to the $2(T-t)$ factors, then step 3.
$\dfrac{\partial h_T}{\partial h_t} = D_T W \, D_{T-1} W \cdots D_{t+1} W$ **with** $D_s = \operatorname{diag}(1 - h_s \odot h_s)$, **and** $\left\| \partial h_T / \partial h_t \right\|_2 \le \|W\|_2^{T-t}$	The loss at step T reaches g_t as $(\partial h_T / \partial h_t)^\top V^\top e_T$, so with $\|W\|_2 < 1$ its size shrinks at least geometrically with the distance $T - t$: the vanishing gradient, and saturated units (D_s near 0) shrink it further. With $\|W\|_2 > 1$ the bound allows the product to grow geometrically, the exploding gradient, which is why RNN training clips gradient norms. The bound is one-sided: it guarantees vanishing, it does not guarantee explosion.

Problem 7

> Many-to-one: a classifier reads the whole sequence and is scored once. $\ell_t = 0$ for $t < T$, and $\ell_T = -\sum_j y_j \log s_j$ with $s = \operatorname{softmax}(z_T)$ and y one-hot. Give the δ_t, $\nabla_W L$ and $\nabla_V L$.

$e_t = 0$ for $t < T$ and $e_T = s - y$.	Steps before T have no loss of their own; the softmax page gives the gradient of cross-entropy at the logits.
$g_T = V^\top (s - y)$ and $g_t = W^\top \delta_{t+1}$ for $t < T$.	Problem 3 with step 1: only the route through the next step is left.
$\delta_T = (1 - h_T \odot h_T) \odot V^\top (s - y)$ and $\delta_t = (1 - h_t \odot h_t) \odot W^\top \delta_{t+1}$.	Problem 3, step 4.
$\nabla_V L = \sum_t e_t h_t^\top = (s - y) h_T^\top$.	Problem 4; every term with $t < T$ is 0.

$\delta_T = (1 - h_T \odot h_T) \odot V^\top (s - y)$, $\delta_t = (1 - h_t \odot h_t) \odot W^\top \delta_{t+1}$ **for** $t < T$; $\nabla_W L = \sum_t \delta_t h_{t-1}^\top$ **and** $\nabla_V L = (s - y)\, h_T^\top$	The whole training signal for the early steps arrives through Problem 6's product, so this setup is where vanishing gradients bite hardest: the first inputs of a long sequence barely move the weights.

Problem 8

Compute $\nabla_{x_t} L$ and $\nabla_{h_0} L$. When is $\nabla_{h_0} L$ needed?

x_t is used only in a_t, so $\nabla_{x_t} L = (\partial a_t / \partial x_t)^\top \delta_t = U^\top \delta_t$.	x_t appears in no other step, and δ_t is the total gradient at a_t.
h_0 is used only in a_1, so $\nabla_{h_0} L = (\partial a_1 / \partial h_0)^\top \delta_1 = W^\top \delta_1$.	There is no output z_0, so the route through a step's own output is absent: Problem 3, step 3, with only the second term.
$\nabla_{x_t} L = U^\top \delta_t$ **and** $\nabla_{h_0} L = W^\top \delta_1$	$\nabla_{x_t} L$ continues the backward pass into whatever produced x_t: an embedding table or a lower RNN layer. $\nabla_{h_0} L$ is needed when h_0 is learned, or when it is another network's output, as in an encoder–decoder where the decoder starts from the encoder's last state; with h_0 fixed at zero it is discarded.

Problem 9

An LSTM keeps a cell state $c_t \in \mathbb{R}^m$ beside h_t:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

and

$$h_t = o_t \odot \tanh(c_t),$$

where the forget, input and output gates $f_t, i_t, o_t \in (0,1)^m$ and the candidate $\tilde{c}_t \in (-1,1)^m$ are computed from h_{t-1} and x_t. Treat these four as inputs to the cell. You are given dh_t, the total

gradient of L at h_t, and $\bar{c}_t$, the gradient that reaches c_t from step $t+1$ through c_{t+1}. Compute dc_t, the total gradient at c_t, and the gradients to c_{t-1} (through this cell), f_t, i_t, $\tilde{c}_t$ and o_t.

c_t reaches L through h_t and through c_{t+1}, so $dc_t = \bar{c}_t + dh_t \odot o_t \odot \big(1 - \tanh(c_t) \odot \tanh(c_t)\big)$.	Two routes, added: $\bar{c}_t$ is the second by definition, and the first goes through $h_{t,i} = o_{t,i} \tanh(c_{t,i})$, whose derivative in $c_{t,i}$ is $o_{t,i}(1 - \tanh^2 c_{t,i})$.
$do_t = \tanh(c_t) \odot dh_t$.	o_t appears only in h_t, entry by entry, with coefficient $\tanh(c_t)$.
$c_{t,j} = f_{t,j} c_{t-1,j} + i_{t,j} \tilde{c}_{t,j}$ for each j, so the gradient at each of the four factors is the other factor of its product times dc_t.	Every term is a product of two entries with the same index, and c_t is the only place they appear in this cell.
$dc_t = \bar{c}_t + dh_t \odot o_t \odot (1 - \tanh(c_t) \odot \tanh(c_t))$; $f_t \odot dc_t$ **to** c_{t-1}; $df_t = c_{t-1} \odot dc_t$, $di_t = \tilde{c}_t \odot dc_t$, $d\tilde{c}_t = i_t \odot dc_t$, $do_t = \tanh(c_t) \odot dh_t$	$f_t \odot dc_t$ is the $\bar{c}_{t-1}$ of the previous step, so the cell gradients run backwards by the recursion $\bar{c}_{t-1} = f_t \odot dc_t$. The gate gradients then go through each gate's sigmoid or tanh into the weights, as in Problem 4, and into dh_{t-1}.

Problem 10

Follow the cell states alone: holding every gate and candidate fixed, compute $\partial c_T / \partial c_t$ for $t < T$, and compare it with Problem 6.

With the gates fixed, $\partial c_s / \partial c_{s-1} = \operatorname{diag}(f_s)$.	$c_s = f_s \odot c_{s-1} + i_s \odot \tilde{c}_s$ is elementwise in c_{s-1} with coefficient f_s, and the second term does not involve c_{s-1}.
$\dfrac{\partial c_T}{\partial c_t} = \operatorname{diag}(f_T) \operatorname{diag}(f_{T-1}) \cdots \operatorname{diag}(f_{t+1})$.	The chain rule along $c_{t+1}, \ldots, c_T$, as in Problem 6, step 1.
A product of diagonal matrices is diagonal, with the products of the entries on its diagonal.	Diagonal matrices multiply entry by entry.

$\partial c_T / \partial c_t = \operatorname{diag}(f_{t+1} \odot f_{t+2} \odot \cdots \odot f_T)$	Unlike Problem 6 there is no W and no $\tanh'$ in the product, only gates the network sets for itself: where it keeps $f \approx 1$, the gradient reaches step t almost undiminished however large $T - t$ is. This path is why LSTMs learn long-range dependencies that vanilla RNNs cannot; the full gradient also has paths through the gates and h, which can still vanish.

Where this goes wrong

1. Weight gradient from the last step only

Feed-forward layers each own their weights, and a recurrent layer drawn as a single box looks like one more of them.

$\delta_2 = \frac{5}{2}$ and $\delta_1 = \frac{5}{4}$ in Problem 5	Right so far: Problem 5, steps 2 and 3.
"The layer multiplies its input by u, so $\partial L / \partial u$ is its delta times its input."	The analogy that causes the mistake: the dense-layer rule applied to the rolled-up diagram, which hides that u is used at both steps.
$\partial L / \partial u = \delta_2 x_2 = 5$ ✗	u is used at every step, and the chain rule adds every use: $\delta_2 x_2 + \delta_1 x_1 = \frac{25}{4}$ (Problem 5). The missing $\frac{5}{4}$ is the influence of x_1, so a network trained this way cannot learn to use anything but the latest input.

2. Pairing each delta with the state after the step

In Problem 4 every recurrent term is an outer product of a delta with a hidden state, and it is easy to take the state with the same index.

$\nabla_W L = \sum_t \nabla_{W^{(t)}} L$	Right so far: Problem 4, step 1.

“Step t's delta goes with step t's hidden state.”	The shortcut that causes the mistake: matching indices instead of asking what W multiplies in $a_t = Wh_{t-1} + Ux_t + b$.
$\nabla_W L = \sum_t \delta_t h_t^\top$ ✗	The coefficient of W_{ij} in $a_{t,i}$ is $h_{t-1,j}$, the state coming into step t, so the term is $\delta_t h_{t-1}^\top$. In Problem 5 the wrong pairing gives $\frac{5}{2} \cdot \frac{5}{2} + \frac{5}{4} \cdot 1 = \frac{15}{2}$ instead of $\frac{5}{2}$. The shape is right, so nothing fails.

3. Recurrent gradient without the tanh derivative

The recursion passes a gradient from step to step, and it is tempting to pass g itself along.

$g_t = V^\top e_t + (\partial a_{t+1}/\partial h_t)^\top \nabla_{a_{t+1}} L$	Right so far: Problem 3, step 3, before naming $\nabla_{a_{t+1}} L$.
“The gradient coming back from step $t+1$ is g_{t+1}, sent through $W^\top$.”	The analogy that causes the mistake: the linear RNN of Problem 5, where g_{t+1} and δ_{t+1} are equal.
$g_t = V^\top e_t + W^\top g_{t+1}$ ✗	W acts on h_t to make a_{t+1}, and $h_{t+1} = \tanh(a_{t+1})$ sits between a_{t+1} and g_{t+1}, so the route goes through $\delta_{t+1} = (1 - h_{t+1} \odot h_{t+1}) \odot g_{t+1}$. Dropping the factor removes exactly the saturation that Problem 6 shows makes gradients vanish faster, so the error also hides the problem it would reveal.

4. Sending the gradient back through W instead of W^T

The forward pass multiplies the state by W at every step, and the backward pass seems to need the same matrix.

$g_t = V^\top e_t + (\partial a_{t+1}/\partial h_t)^\top \delta_{t+1}$ with $\partial a_{t+1}/\partial h_t = W$	Right so far: Problem 3, step 3.
“The state goes forward through W, so its gradient comes back through W.”	The analogy that causes the mistake: the forward map run again, rather than its transpose, which is what carries a gradient from outputs back to inputs.

$g_t = V^\top e_t + W\delta_{t+1}$ ✗	W is square, so the shapes give no warning. $(\partial L/\partial h_t)_j = \sum_i \delta_{t+1,i} W_{ij}$ sums over W's first index, which is the product with $W^\top$; the two agree only for a symmetric W.

5. Cell-state gradient without the path from the next cell

The LSTM's output is h_t, and it is natural to send the cell state only the gradient that arrives through h_t.

$h_t = o_t \odot \tanh(c_t)$ and dh_t is given	Right so far: Problem 9.
"c_t affects the loss through h_t, so dc_t is dh_t pulled back through $o_t \odot \tanh$."	The analogy that causes the mistake: the vanilla RNN, whose state reaches the next step only through h_t; the LSTM's cell state also feeds the next cell directly.
$dc_t = dh_t \odot o_t \odot (1 - \tanh(c_t) \odot \tanh(c_t))$ ✗	The term $\bar{c}_t$ from $c_{t+1} = f_{t+1} \odot c_t + \ldots$ is missing (Problem 9, step 1). That term is the path of Problem 10, the one with no W and no $\tanh'$ in it, so dropping it turns the LSTM's long-range gradient back into the vanishing one of Problem 6.

More at eulerpad.app. This is one of 82 free practice sets with worked solutions at eulerpad.app/practice, alongside guides to the topics behind them.

EulerPad, the app, reads your handwritten math on iPad, iPhone and Android and checks it line by line. It's free to start.

This set is also Chapter 16 of *Deep Learning Math by Hand*, a 250-problem workbook with worked solutions: eulerpad.app/workbooks

eulerpad.app/practice/backprop-through-time