Lecture 16 · Part IV · Lecture notes

Recurrent networks

Dataset: Chicago transit ridership

How to use these notes

The primary text for this lecture is Géron, Chapter 13. These notes are not a replacement for it and do not repeat it: they are the lecture's own argument written out at length — the derivation done slowly, the numbers with the runs that produced them, and the reasoning between one slide and the next that a deck can only gesture at. Read the chapter for the method; read these for why the lecture makes the choices it does. Every figure quoted here is printed by this lecture's notebook, and the course checks that mechanically rather than trusting it.

Contents

1 The task, the number to beat, and an honest protocol **2**

2 What a linear model on lags cannot do **3**

2.1 Unrolling, and the problem it creates . . . 3

3 Deeper, wider, and further ahead **5**

3.1 Loss, scale and what to watch . . . 5

3.2 Forecasting k steps ahead . . . 6

3.3 Improvement 1 — deeper, which one seed cannot rank . . . 6

3.4 Improvement 2 — more series, which worked . . . 6

3.5 Improvement 3 — further than one day . . . 7

4 LSTM and GRU, gate by gate **7**

5 The whole application **9**

5.1 Red-team it: what happens when the world changes . . . 9

6 The notebook **11**

7 Design exercise: an energy retailer's next-day demand forecast **11**

8 Exercises **13**

Summary **13**

---

1 The task, the number to beat, and an honest protocol

A transit authority wants tomorrow's rail boardings, one number one day ahead, to schedule trains and crews. The two errors are not equal — too high runs empty trains, too low leaves passengers on platforms — and we use a symmetric metric anyway, saying so. The label is free: the next day's value is the label of today's window.

The series. Plotted over twenty-one years, bus falls, rail rises and then falls, and March 2020 is a cliff. We model one regime, 2016 to the end of May 2019: 1,247 days. Zoomed to ten weeks, a seven-day cycle dominates — working days high, weekends at about half — with an extra trough on public holidays, which the `day_type` column marks as Sundays. Monthly means show a second, slower cycle once a year around a slowly moving level. A repeating pattern with a fixed period is **seasonality**; a slow change of level is **trend**.

The prediction and the metric. Given the last 56 days, a whole number of weeks, predict the next: $\hat{y}_{t+1} = f(y_{t-55}, \ldots, y_t)$. The metric is MAE in riders: the unit the operations team uses, and robust to a handful of holidays. MAPE divides by the truth, so on a holiday, when ridership collapses, the same miss counts several times over — exactly where every model is weakest. It is a second opinion, never the objective.

The number to beat. Four forecasts need no model at all. Over the 1,191 days that have a full window behind them, copying the same weekday last week scores 55,399, 12.0% of a typical day, and 2.4 times better than copying the day before: the weekly cycle is worth more than recency. That is the number to beat — not the mean, not zero.

A linear model on 56 lags. Cut the series into overlapping windows, 56 days in and the next day out, and fit $\hat{y}_{t+1} = b + \sum_{k=0}^{55} w_k \, y_{t-k}$. With $w_6 = 1$ and every other weight zero it is copying last week, so it contains the baseline. Under shuffled five-fold cross-validation, averaged over twenty shuffles, it scores 44,925: 18.9% better than copying last week on the same rows.

Failure condition — a shuffled split of a series

Cross-validation estimates the error on new data only if the held-out rows are independent of the training rows and drawn from the situation the model will meet. A shuffled series breaks both: neighbouring days are nearly the same number, so a held-out day has its neighbours in training; and half the training days lie in its future, which never happens in deployment. The model is asked to fill a gap, not to forecast.

Fit on everything to the end of 2018 and forecast the 151 days that follow, never seeing them: the linear model scores 56,446 against 64,815 for copying last week on the same days, a margin of 12.9%. The baseline must be scored on the same rows, because the forward days are harder for every forecast. A **rolling-origin backtest** moves the cut forward, refits and scores the block that follows, several times, so the result has a spread; on it the margin is 14.7%. With each protocol's baseline scored on its own rows, which cancels how hard the period is, the shuffled split still overstates the model's skill by 7.4%, and by 9.0% with the training and test sizes matched as well. From 18.9% shuffled to 12.9% forward, a third of the case for the linear model was an artefact of the evaluation. Every forward number below is scored on the same 151 days.

2 What a linear model on lags cannot do

That model has a *fixed window*. Anything outside the last 56 days is invisible to it, and every lag gets one weight regardless of context.

A recurrent network keeps a *state* instead, and that state is not bounded by a window:

$$\mathbf{h}_t = \phi\left(\mathbf{W}_x^\top \mathbf{x}_t + \mathbf{W}_h^\top \mathbf{h}_{t-1} + \mathbf{b}\right).$$

This is Lecture 7's $\phi(\mathbf{XW} + \mathbf{b})$ with one extra term, and that term is the whole idea. $\mathbf{W}_x$ and $\mathbf{W}_h$ do not depend on t: one set of weights applied at every step, so reach is decoupled from parameter count — the same argument as weight sharing in Lecture 12, along time instead of across an image.

The state is a summary, and that is a budget

$\mathbf{h}_t$ is everything the network has decided to remember about $x_1, \ldots, x_t$, at *fixed width* whatever the sequence length — so it is lossy by construction. What it keeps is learned, not designed: nothing tells the cell that "the same weekday last week" matters.
Two sequences with the same state are indistinguishable to everything downstream, which is the definition of a sufficient statistic, and the network is trying to learn one. It is also the honest limit: if the task needs more bits than the state has, no amount of training supplies them.

2.1 Unrolling, and the problem it creates

Draw the same cell once per time step and you get a network 56 layers deep with one set of weights, reused. Then differentiate it exactly as Lecture 8 taught — **backpropagation through time** needs nothing new.

Failure condition — one matrix raised to the length of the sequence

Unrolling 56 steps gives a 56-layer network, and by the chain rule a gradient descending 56 layers is multiplied by one factor per layer, 56 times over. Here the factor is *the same matrix* every step, because the weights are shared — so the gradient is governed by a power of one matrix, and powers of a matrix either vanish or explode unless the spectral radius sits almost exactly at one.

Scaling the initial weights so that each factor is about one — the repair for a deep stack of independent layers — cannot fix this, because here every layer is the same matrix. Something has to change in the cell itself.

Shape	Input	Output	Example
sequence to vector	a sequence	one value	today's model; classify a review
vector to sequence	one value	a sequence	a caption from an image
sequence to sequence	a sequence	a sequence	a forecast at every step
encoder–decoder	a sequence	a sequence, after all of it	translation

Four shapes of recurrent model, and the cell is the same in all four. What changes is where you take the output from — the last state, every state, or a second network fed by the last state — and that choice is the architecture. We build the first and the third today; the fourth is Lecture 18.

The model is eleven lines — an `nn.RNN` and a linear head applied to `outputs[:, -1]`, the hidden state after the final step. The head is not optional: the state has 32 components and the target has one, and tanh outputs lie in $(-1, 1)$ while our scaled targets sometimes exceed that. `batch_first=True` because our batches are [batch, time, series]; without it the module silently reads the first dimension as time.

The training loop is Lecture 8's five lines, unchanged. The loss is Huber — quadratic near zero, linear far away — because we are optimising a proxy for MAE, and minimising the absolute error directly gives a gradient that never gets smaller as you approach the optimum. *The metric and the training loss are allowed to differ*: the metric is what the operator reads, the loss is what the optimiser can descend, and here is the case.

Shuffle, but shuffle the right thing

`DataLoader(train_set, shuffle=True)` shuffles the *windows*. It does not shuffle the values inside a window — the sequence within each row is untouched.

Consecutive windows share almost all their content: one starting on day t and one on day $t + 1$ differ in two values out of fifty-six. Gradient descent expects the instances in a batch to be independent draws. Without shuffling, a mini-batch is a run of near-identical, highly correlated windows: its gradient is effectively one example's, no less noisy than a single draw, and training stalls.

And do not let that shuffling cross the split. Windows are shuffled within the training period only. Shuffling and leakage are different operations, and it is easy to conflate them.

On identical shuffled folds the recurrent network scores 38,224, against 55,385 for copying last week and 44,925 for the linear model — 31.0% better than the baseline and 15% better than the linear model. The baseline here is the mean of its five fold scores; 55,399 is the same 1,191 days scored all at once, and the two differ only in how the folds are weighted. Both beat the baseline

comfortably. The recurrent model's lead over the linear one is the argument that it can use the *order* of the window rather than a weighted sum of it — but it is one comparison under one seed, and the ladder below shows how far another seed can move a recurrent score.

Failure condition — and the assumption, stated rather than left implied

Those folds are shuffled, so *none of those numbers is a forecast.* The first section measured what that is worth. They are comparable with each other because the folds are identical, and with nothing else.
Every forward number in the rest of this lecture is on the honest ladder.

3 Deeper, wider, and further ahead

3.1 Loss, scale and what to watch

Squared error punishes a large miss quadratically, so on a series with occasional shocks a handful of days dominate the gradient. MSE is smooth everywhere and dominated by outliers; MAE is robust but its gradient is constant, so it converges slowly near the optimum; Huber,

$$L_\delta(e) = \begin{cases} \frac{1}{2}e^2 & |e| \le \delta, \\ \delta\left(|e| - \frac{1}{2}\delta\right) & |e| > \delta, \end{cases}$$

has both properties for one knob. The choice is the same argument as choosing the metric, with the same answer: it depends on what an error costs the operator.

Normalise by a constant computed on the training period only — the same rule as every other lecture, and easier to break here because the series looks like one object. Divide rather than standardise when the series is positive and multiplicative: ridership is counts, and a ratio is more natural than a difference. And if you modelled the day-to-day change rather than the level, turn the prediction back into a level before reporting it; the metric is in the units the operator uses.

Failure condition — a model scored on day-to-day changes

A model evaluated on day-to-day changes can look excellent and mean nothing: predicting a change of zero every day is nearly right in those units, and it is the naive forecast wearing a disguise. The units the metric was computed in decide what it is able to praise, and the report is usually silent about which they were.

While it trains, log the training and validation loss per epoch (a gap opening between them is overfitting: the model fitting the training window better than the held-out one); the gradient norm, along time here, which on a plain RNN over a long window decays; and the forecast itself, plotted against the truth on a held-out stretch.

Failure condition — the failure a plot catches and a metric does not

A model that simply repeats the previous value scores well on almost any error metric and is worthless. On a plot it is unmistakable: the prediction is the truth, shifted right by one step.

3.2 Forecasting k steps ahead

Three strategies, all defensible, all to be evaluated the same way: forward in time, at the horizon you actually need. **Recursive** predicts one step, feeds it back and repeats, so errors compound and the model never trained on its own output. **Direct** fits a separate model per horizon: k models, none sharing what the others learned. **Sequence-to-sequence** is one model with k outputs at once: more to fit, and the outputs are not independent. A model that is excellent one step ahead and useless seven steps ahead is not a good model if the operator plans a week in advance.

Teacher forcing, and what it costs

Training a sequence-to-sequence model, you can feed the decoder the true previous token or its own previous prediction. The true token (**teacher forcing**) trains fast and stable, but the model has never seen its own mistakes; its own prediction trains slowly, early training is noise, and it matches deployment.

Failure condition — a model trained only on true prefixes

Trained on true history and evaluated on its own output, the model meets a distribution at prediction time it never saw in training — its own errors, compounding. This is **exposure bias**: the forecast drifts, the further ahead the worse, which is precisely the recursive strategy's weakness. And the training loss, computed on true prefixes throughout, never registers it.

3.3 Improvement 1 — deeper, which one seed cannot rank

One argument, `num_layers=3`. Under seed 42 the simple RNN forecasts at 49,073 (train 35,972, sgd recipe) and the three-layer one at 53,491 (train 40,630, adam): depth loses on test and on train.

Do not write that down. Each recipe is scored on July–December 2018, inside the training period, the lower score wins, and the winner is refitted. For the simple RNN seed 42 scored sgd 41,878 and adam 42,879: sgd by 1,001 riders, 2.3%. Change only the seed and the order flips. Under seed 0 the deep RNN wins, 42,878 against 43,360; under seed 1 by far more, 42,299 against 88,517; seed 2 flips it back, 42,027 against 149,707 — and each time the winner on test also wins on train. Two of those refits failed outright, worse than copying last week on the test days and on the training rows too: the recipe the selection slice chose did not carry over to the full training period.

Every fit is seeded, so each of these lines reproduces exactly on one machine, but another processor's rounding can put seed 42 on a different line. *A ranking whose gap is smaller than its spread over seeds is not a finding.* Report it as unresolved.

3.4 Improvement 2 — more series, which worked

We have been forecasting rail from rail alone while a bus series sits unused, and a `day_type` column we established is known in advance. Adding both — rail, bus, and the next day's type one-hot encoded, five columns — changes the architecture by one number, `input_size=5`.

Wait — that is a `shift(-1)`

The shuffled split of the first section let a model train on rows surrounding what it was scored on. This one reads the future *deliberately*.

The test that separates them: *is the value available at prediction time?* The type of the day we are forecasting is on the public calendar and has been for a year. Its ridership is not.

This is the one question about leakage that no static check can answer for you. It is not a property of the code; it is a property of the world.

Model	Forecast MAE	vs copying last week
Copy last week	64,815	—
Simple RNN, rail only	49,073	24% better
Simple RNN, five inputs	40,046	38% better

Eighteen per cent off the rail-only error for one line of feature engineering and one changed constructor argument. The rail-only row is the unstable one: across the four seeds it runs from 42,027 to 88,517, and the five-input row, at 40,046, beats every one of them. Most of it is the day type: the network no longer has to infer that a holiday is coming from the shape of the previous eight weeks — which is impossible — because we told it.

3.5 Improvement 3 — further than one day

The operations team also wants a fortnight. Feed the forecast back — predict day $t + 1$, append it to the window as though it had happened, predict $t + 2$ — and errors accumulate. Or predict all fourteen at once: change the output layer to fourteen units and the target to a vector, one model and no accumulation.

A third way predicts fourteen at *every* step: at step 0 the forecasts for steps 1 to 14, at step 1 for steps 2 to 15, and so on, taking the head over all outputs rather than the last. That is not cheating: a recurrent cell at step t has seen steps 1 to t and nothing more — it is **causal**. The loss simply gains a term at every step, so many more gradients flow and they travel a shorter distance. After training, only the last step's output is used.

Measured, the error grows with distance from 35,997 at one day to 46,586 at fourteen, and stays below the no-model baseline throughout.

4 LSTM and GRU, gate by gate

A plain cell multiplies the state at every step, so $\partial \mathbf{h}_t / \partial \mathbf{h}_{t-1} = \operatorname{diag}(\phi')\mathbf{W}_h^{\mathsf{T}}$ and over T steps the gradient carries T copies of one matrix — a deep stack's product of factors, with the extra force that it is one matrix rather than many independent ones.

What a gated cell does instead

An LSTM keeps a separate cell state updated *additively*,

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t .$$

When the forget gate is near one, $\partial \mathbf{c}_t / \partial \mathbf{c}_{t-1} \approx \mathbf{1}$ and the gradient passes through almost unchanged.
That is the entire trick: *a path along which the gradient is added to rather than multiplied.*

Gate	Form	What it decides
forget $\mathbf{f}_t$	sigmoid	how much of the old cell state to keep
input $\mathbf{i}_t$	sigmoid	how much of the candidate to write
candidate $\tilde{\mathbf{c}}_t$	tanh	what to write, if anything
output $\mathbf{o}_t$	sigmoid	how much of the cell state to expose as $\mathbf{h}_t$

Every gate is a function of the same two inputs $(\mathbf{x}_t, \mathbf{h}_{t-1})$ with its own weights, so a sigmoid in $[0, 1]$ is a *soft switch* and the network learns when to open it. Four gates means roughly four times the parameters of a plain cell.

A **GRU** has two gates. The update gate $\mathbf{z}_t$ merges the LSTM's forget and input gates into one — what is kept is what is not written; the reset gate $\mathbf{r}_t$ sets how much of the previous state to use when forming the candidate; and there is no separate cell state, $\mathbf{h}_t$ doing both jobs:

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t .$$

Still additive, still a near-identity path when $\mathbf{z}_t$ is small — the property that mattered, kept, with about three quarters of the parameters. Which to use? On most problems they are within noise of one another: the GRU is smaller and trains faster, the LSTM has the longer track record. Measure, on your own protocol, rather than inheriting an opinion.

Model, all five inputs	Forecast MAE	Training MAE
Simple RNN	40,046	27,035
LSTM	37,542	24,540
GRU	34,639	20,356

The GRU wins, and the second column says why to be careful: it fits the training rows 25% better than the plain cell but the forecast only 14% better. That gap is overfitting, and it is what to attack next, before reaching for a bigger cell. Neither cell is a large model — a few thousand parameters. Across the whole ladder, every bar a forecast on the same 151 days, the error falls from 64,815 for copying last week to 34,639: 47% better than doing nothing.

5 The whole application

Forecast	What we reported	What it does
Copy last week	55,385	64,815
Linear, 56 lags	44,925	49,312
Simple RNN	38,224	49,073
GRU, five inputs	—	34,639

The top-left number is copy-last-week averaged over the five shuffled folds; over all 1,191 days at once it is 55,399. The two middle rows are the same models measured twice; the simple RNN's right-hand number is the one that moves between seeds. The linear row's left number is the least-squares fit under shuffled folds; its right number is the same model trained like the networks, with the Huber loss (least squares, forward, scored 56,446), on the same 151 days as every other right-hand number. The bottom row is the model we would deploy, and it was never measured the wrong way. *The final number is better than the one first reported — but we reached it by first admitting that the first one was not a forecast.*

Is 34,639 good? It is 47% better than a system a transit authority could run with no data scientist at all, and its typical miss is around 10.4% of the day it forecasts, against 16.5% for copying last week *on those same 151 days*. A naive percentage measured over any other set of days is not the comparison to make here. (As a share of a mean day on those same 151 days — 583,030 riders — the miss is 5.9%, which is a third quantity again.)

What a defensible report says

"Forecasting the five months after the training cut, and never seeing them, the model's mean absolute error is 34,639 boardings a day against 64,815 for copying the same day of the previous week. Both figures are on identical rows. The gain is concentrated on holidays." The protocol, the baseline, the rows, and where the gain comes from — four things the number alone does not carry.

5.1 Red-team it: what happens when the world changes

Fit on everything to the end of 2019 and forecast the four months from March 2020.

	MAE
The linear model, trained to the end of 2019	84,380
Copying last week, on the same 122 days	36,282
The actual mean level over that period	141,404

Failure condition — a regime the training window never contained

Scored on the four months from March 2020, the model is 2.3 times worse than doing nothing and its error is six tenths of the entire level of the series. Nothing in training could have shown this: the failure is that the future stopped resembling the window the model was fitted on.

To be fair to the model: March 2020 is not a leak and not its fault. No forecaster could have produced it from eight weeks of history, and copying last week wins only because it adapts within a week of the collapse. The lesson is not that the model is bad — it is that *an honest protocol told you the model had failed, on the day it failed*. A shuffled split, with those same days spread across its training folds, could not have. Which is why a deployed forecaster is monitored against its own naive baseline, continuously: if the margin disappears, something has changed in the world.

Temporal leakage checklist

1. Is the split by *date*, not by row?
2. Does every feature exist at the moment the forecast is made? Name the moment.
3. Were any statistics — means, scalers, encodings — computed over the whole series before the cut?
4. Do the training windows overlap the scored windows? Purge them.
5. Is the baseline computed on identical rows to the model?
6. Does the reported score come from more than one origin?

The third is the oldest leak in the course — a statistic fitted before the split — and the fourth is the one only a series of overlapping windows has.

Reading a recurrent result honestly

At what horizon?	one step ahead is a different problem from seven
Against which baseline?	copy the same weekday, not the mean
On which protocol?	rolling origin, and how many folds
With what spread?	one origin is one sample
In which units?	levels or changes; scaled or not

A recurrent network that beats the naive forecast by less than the fold-to-fold spread has not beaten it — and a ranking that flips between runs, as the depth comparison did, was never a ranking.

A recipe for a forecasting model

1. Plot the series. Seasonality and trend are usually visible, and cost nothing to find.
2. Compute the naive baselines: yesterday, and the same day last week. These are the numbers to beat.
3. Fix the protocol before fitting anything: horizon, rolling origin, number of folds, metric.
4. Fit a linear model on lags. It is fast, and it is often close.
5. Only then a recurrent model, scored on the same protocol.
6. Report the ladder with its spread, not the best row alone.

Steps 2 and 3 come before step 4 because after you have a number, you will not choose the protocol that makes it look worse.

6 The notebook

The notebook builds the simple recurrent model line by line, then runs the whole ladder of six models under one protocol: each recipe chosen on July to December 2018, inside the training period, and every model scored on the same 151 forward days, then the two univariate RNNs under three more seeds. It also scores the simple RNN on shuffled folds, forecasts fourteen days ahead, and fits the linear model through March 2020.

What to change

Change the seed in the setup cell to one the seed table does not show and re-run the ladder. Watch the simple and the deep RNN: if their order flips, you have reproduced the lecture's point about a ranking that rests on one seed.

7 Design exercise: an energy retailer's next-day demand forecast

The lecture closes with a design exercise in the manner of an ML system-design interview, framed as the ML system behind a company's product.

The task. An energy retailer sells electricity to households and buys what it delivers on the wholesale market the day before. The purchasing desk's order is what the model drives: for each hour of the next day, a forecast of its customers' demand, in the market's units. Buy short, and the gap costs the balancing price the system operator sets; buy long, and the surplus is sold at a loss; finance reads the imbalance bill against margin. Given: years of hourly meter readings summed over the customers, the archived weather forecasts issued each afternoon, and the holiday calendar. Tens of thousands of hourly rows per tariff group and a few groups; one order a day. The output feeds the purchasing desk, the risk team that hedges the rest, and finance, which pays the bill.

1. **Frame it.** The horizon is the market's: every hour of the next day, issued before the auction closes. The metric is the mean absolute error in the units the desk buys in, read beside the no-model forecasts on identical hours; the training loss may be Huber, the metric is not. What an error costs is not symmetric — the balancing price on a shortfall, a loss on the surplus — and that is finance's question, asked before the loss is chosen. *Pitfall:* the first number reported is the one-step-ahead error, because that is what the loop produces, and the next hour is nearly the last hour; the desk buys a day ahead. Fix the protocol before fitting anything — horizon, rolling origin, folds, metric — and score at the horizon the order needs.
2. **The numbers to beat.** Plot the series: a daily cycle, a weekly one, an annual one, holidays that look like weekends. Compute the naive forecasts — the same hour the day before, the same hour the week before, which is what the desk orders on now — then a linear model on lagged hours, which contains the seasonal copy as one weight of one and beats it modestly. *Pitfall:* the network reported against the mean, or against nothing. The seasonal copy is the bar, on the same hours; a network that beats it by less than the fold-to-fold spread has not beaten it; and once you have a number you will not choose the protocol that makes it look worse.
3. **Split by date.** Training years up to a cut, validation after it, test after that, from several

origins. Each example is a window of recent hours with the next day's hours as target; consecutive windows share almost everything, so shuffle them, within the training period only. Divide the series by a constant fitted on the training period — demand is positive and multiplicative — and report in the market's units. *Pitfall:* the scaler fitted on the whole series, because the series looks like one object, and the windows whose targets lie past the cut; the release promises a margin the desk never sees. Fit every statistic before the cut; purge every window that straddles it.

4. **Inputs.** The window of recent demand, one series per tariff group; the type of the day being forecast, the legitimate look ahead because the calendar is public a year early; and the weather forecast for that day as the forecast *issued that afternoon*, from the archive. *Pitfall:* training on the observed temperature of the forecast day. The join is one line, and the model learns to read a number the desk will never have when it orders; the offline error falls, the live error does not, and finance pays the difference. The test is not in the code — is the value available at the moment the order is placed? — and the fix is to train on the archived forecasts, their errors included.

5. **The model.** A GRU over the window — demand, day type and forecast temperature at each step — and a linear head with one output per hour of the next day: sequence to vector, the whole horizon at once. Huber loss, so a few shock days do not own the gradient; the same loop as every network in the course; the recipe chosen on the validation stretch. *Pitfall, exposure bias:* forecasting the day recursively, each hour fed back as though it had happened. The model was trained on true history only, meets its own errors in service, and they compound across the day — the evening peak forecast from a morning the model invented. Predict the whole day at once, or a day at every step with a loss at every step, and evaluate at the horizon either way.

6. **Gradients along time.** A week of hourly steps unrolls into a network a few hundred layers deep sharing one matrix, so the gradient carries a power of it and vanishes or explodes; the gated cell is the structural answer, an additive path. Log the gradient norm each epoch beside both losses. *Pitfall:* a loss that sits flat, or jumps and stays, read as "more epochs" or "a smaller network" while the release slips. Read the norm first; clip it — rescale the gradient whenever its norm passes a threshold chosen from its distribution, since a clip below the median turns the optimiser into something else — and if a plain cell still decays, change the cell, not the learning rate.

7. **Read it honestly.** Every model on the same forward days, from several origins, under several seeds: the ladder with its spread, never the best row. The gap between training and forecast error is overfitting, and it is what to attack before a bigger cell. Plot a held-out week, forecast against truth: a model that outputs the previous day shifted by an hour scores well on almost any metric and is worthless, and on the plot it is unmistakable. *Pitfall:* a deeper network wins on the first run and goes in the release note; change the seed and the order flips. A ranking that rests on one run is not a finding — report it as unresolved.

8. **In production.** Shadow first: the forecast is scored beside the desk's old method on identical hours, and only then writes the order. Truth arrives with the next day's meter data, so the margin over the seasonal copy is a running number; retrain on a rolling window at a fixed cadence, with the data team on call for the days the margin vanishes. A price rise that sends customers to a rival, a tariff that moves use to cheap hours, rooftop solar, a lockdown: none is

in the window. *Pitfall:* the model keeps forecasting the old customer base and is worse than copying last week within a week of the change, because the copy adapts and the model does not, and finance reads it in the imbalance bill. Monitor the margin continuously; when it vanishes, fall back to the copy, page the person on call, and refit on the regime that has begun.

9. **Ship it.** Meter readings → plot, naive and seasonal copies → protocol fixed at the day-ahead horizon → split by date, windows purged at the cut, scaler on the training period → history, day type and archived forecasts → a GRU with one output per hour, Huber loss, gradient norm logged and clipped → the ladder with its spread, and a plot → a shadow run scored daily against the copy → the order; fall back, retrain.

8 Exercises

The five set at the end of this lecture's deck, in the style of the written paper. The solutions are on Lecture 17's deck and in the exercise book.

1. A recurrent network is trained on 56-day windows drawn from one series. Explain why shuffling the *windows* is legitimate while shuffling the *series* is not. [4]

2. A one-layer RNN beats a three-layer RNN on test and on train. Re-run with only the seed changed, the same code reverses the order on both. State what the ranking rested on, and what would settle it. [4]

3. Gates were introduced after a plain recurrent cell failed over long windows. State the mechanism, in terms of what happens to the same matrix over 56 steps. [4]

4. A colleague builds the multivariate forecaster with `day_type.shift(-1)` as a feature and objects that it reads the future. State whether it is a leak, and the one question that settles it. [5]

5. Report the training MAE beside the test MAE. Name the two failures the pair distinguishes, and say why the test column alone cannot. [4]

Summary

A recurrent cell is Lecture 7's layer with one extra term, and that term decouples reach from parameter count — weight sharing along time instead of across an image. The state is a fixed-width lossy summary, which is a budget: if the task needs more bits than it holds, no amount of training supplies them. Unrolling makes training ordinary backpropagation on a 56-layer network with *shared* weights, and that sharing is the problem, because the gradient then carries a power of one matrix rather than a product of independent ones.

Copying last week, 55,399, was the number to beat, and a shuffled split overstated the linear model's margin over it: 18.9% shuffled, 12.9% forward. Then three improvements, measured forward in time. Depth is the one that cannot be ranked: under seed 42 the three-layer network lost to the simple one on test and on train, under seeds 0 and 1 it won on both, and the gap at any one seed is smaller than either model's swing across seeds. A ranking that rests on one seed is not a finding. Adding the bus series and the next day's type — a `shift(-1)` that is legitimate because

the calendar is public — paid more than depth did, and reproduced. Gates buy an additive path along which the gradient is added to rather than multiplied, and the GRU on five inputs reached 34,639.

The ladder ends 47% better than copying last week, and the honest ladder is not the one we first reported: the two middle rows are the same models measured twice. Then the red team — a model trained to the end of 2019 is 2.3 times worse than doing nothing on the months from March 2020. An honest protocol reports that failure on the day it happens; a forecaster in deployment is monitored against its own naive baseline for exactly that reason. The closing design exercise built an energy retailer's next-day demand forecast, the one that decides how much power it buys, in nine steps — the horizon fixed at the market's, the seasonal copy as the bar, the scaler and the windows kept inside a split by date, archived forecasts as inputs, a gated cell predicting the whole day at once, the gradient norm logged and clipped, seeds and a plot before any ranking, a shadow run before the first order, and the margin over the copy monitored for the day the customer base changes.