

Machine Learning in Engineering Science
Sequential Data, LSTM, Attention, and Transformers

Chi-Hua Jonny Yu
NCKU / LAiMM

Fall 2026 – Week 7 – 19 October 2026

Contents

1	Sequential Data, LSTM, Attention, and Transformers	2
1.1	Learning outcomes and evidence	2
1.2	Hour 1: ordered observations and recurrent memory	2
1.2.1	Order, sampling, and the prediction task	2
1.2.2	Chronological splits and available information	3
1.2.3	Baselines and error interpretation	3
1.2.4	RNN state and learning through time	3
1.2.5	The complete LSTM mechanism	4
1.2.6	Implementation, selection, and reproducibility	4
1.3	Hour 2: why attention is useful and how it works	4
1.3.1	The fixed-context bottleneck	4
1.3.2	Queries, keys, values, and shapes	5
1.3.3	An attention calculation with no hidden steps	5
1.3.4	Self-attention, cross-attention, and masks	6
1.3.5	Benefits, costs, and interpretation	6
1.4	Hour 3: assemble and test a Transformer	6
1.4.1	Tokens and position	6
1.4.2	Heads, nonlinear transformations, and residual paths	6
1.4.3	Objective, baseline, and generation	7
1.4.4	Behavioral checks and limits	7
1.5	Completion and explain-back	7
1.6	Primary sources	8

Chapter 1

Sequential Data, LSTM, Attention, and Transformers

Week 7: Monday 19 October 2026, 14:00–17:00.

The three one-hour modules follow one question: how can a learned model use earlier information to make a defensible prediction? Hour 1 develops recurrent memory, Hour 2 develops query-dependent retrieval, and Hour 3 assembles and tests a causal Transformer. About 90 minutes of explanation and worked reasoning are combined with 90 minutes of prediction, guided computation, experimentation, and explain-back across the three hours.

1.1 Learning outcomes and evidence

Students should be able to define a forecast origin and horizon; construct windows without future leakage; trace every LSTM gate; compute attention scores, weights, and a context vector; distinguish self-attention from cross-attention; explain positional information, multi-head attention, residuals, and feed-forward layers; and test a decoder’s causal behavior. The three independent CPU notebooks compare an LSTM against baselines, verify attention arithmetic, and train and inspect a small Transformer. All data are synthetic or self-contained. No paid API or GPU is required. These are class exercises, not additional weekly graded assignments.

1.2 Hour 1: ordered observations and recurrent memory

1.2.1 Order, sampling, and the prediction task

A sequence is an ordered collection $x_1, \dots, x_T$ in which order is part of the problem definition. Sensor samples, process events, and language tokens are examples. $(1, 2, 4, 8)$ and $(8, 1, 4, 2)$ have the same multiset and mean but different transitions. A representation that discards order cannot recover those transitions from the values alone. Equally spaced samples and irregular events also require different sampling assumptions. A sequence model does not automatically infer omitted time intervals.

In the example, the forecast origin is immediately before observing x_t . With window length L ,

$$X_t = (x_{t-L}, \dots, x_{t-1}), \quad y_t = x_t.$$

The Python slice `signal[t-L:t]` excludes index t . Every input index must be strictly less than the target index. This is a one-sample horizon. A fixed-origin multi-step forecast cannot use later actual values that would not yet have been observed.

With B histories, T time steps, and F features at each step, batch-first input has shape (B, T, F) . Many-to-one predicts one value per history; many-to-many predicts outputs at multiple positions. Batch size is not sequence length, and an ID is not necessarily a physical feature.

1.2.2 Chronological splits and available information

The synthetic sensor uses 1600 samples, window length 24, training targets before index 960, validation targets 960–1279, and test targets 1280 onward. This produces 936 training, 320 validation, and 320 test windows. Target periods are disjoint. A test window may use earlier observed validation or test sensor history. This is explicitly a rolling one-step protocol, not a forecast of all future values from one initial origin.

Normalization learns statistics, so fit it on training observations only:

$$z_t = (x_t - \mu_{\text{train}}) / \sigma_{\text{train}}.$$

Global means can leak future distribution information even without labels. Random splits of heavily overlapping windows can produce nearly identical training and test histories. For multiple devices or people, group separation may be needed in addition to temporal separation.

1.2.3 Baselines and error interpretation

Persistence uses $\hat{x}_t = x_{t-1}$ and has no fitted parameters. Ridge regression uses a regularized linear map of the flattened history window. Select regularization on validation, then compare the same test targets:

$$\text{MAE} = \frac{1}{N} \sum_n |\hat{y}_n - y_n|, \quad \text{RMSE} = \sqrt{\frac{1}{N} \sum_n (\hat{y}_n - y_n)^2}.$$

Report original target units and inspect residuals against time or operating range. RMSE emphasizes large errors. Neither score alone reveals every failure. A linear model can outperform an LSTM on a small, structured signal; report the observed comparison rather than choosing the more complex name.

1.2.4 RNN state and learning through time

A basic recurrent network reuses one update at every time step:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b).$$

Unrolling displays repeated applications of shared parameters. The state is a learned summary, not necessarily an interpretable physical variable. Backpropagation through time includes products of

Jacobians:

$$\frac{\partial h_t}{\partial h_{t-k}} = J_t J_{t-1} \cdots J_{t-k+1}.$$

Repeated contractions may shrink gradients, and repeated expansions may amplify them. A scalar example 0.9¹⁰⁰ illustrates one contraction mechanism but not a complete nonlinear Jacobian product. Gradient clipping limits large gradient magnitude; it does not itself solve vanishing gradients.

1.2.5 The complete LSTM mechanism

An LSTM separates cell memory c_t from hidden output h_t . With $u_t = [h_{t-1}; x_t]$, a standard cell uses

$$f_t = \sigma(W_f u_t + b_f), \quad i_t = \sigma(W_i u_t + b_i), \quad (1.1)$$

$$g_t = \tanh(W_g u_t + b_g), \quad o_t = \sigma(W_o u_t + b_o), \quad (1.2)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \quad h_t = o_t \odot \tanh(c_t). \quad (1.3)$$

Here $\sigma(z) = 1/(1 + e^{-z})$ and $\odot$ is elementwise multiplication. Each gate is a vector of coefficients, generally one per hidden dimension. The forget gate retains old cell content; the input gate scales a signed candidate write; the output gate exposes a filtered view of the updated cell. The additive memory path can improve gradient flow but does not guarantee unlimited retention or eliminate every optimization difficulty. The direct cell-state path contains products of forget gates when their other dependencies are held fixed; the full derivative is more complicated.

Choose illustrative values $c_{t-1} = 0.8$, $f_t = 0.9$, $i_t = 0.2$, $g_t = -0.5$, and $o_t = 0.7$. Then

$$c_t = 0.72 - 0.10 = 0.62, \quad h_t = 0.7 \tanh(0.62) \approx 0.386.$$

These are hand-chosen teaching values, not training outputs. If $f_t = 1$, new content can still change the cell. Changing o_t changes hidden output without changing this cell update. Predict those effects before using sliders.

1.2.6 Implementation, selection, and reproducibility

With `batch_first=True`, input/output layouts are (B, T, F) and (B, T, H) . Ordinary non-projected hidden and cell states retain layout (layers $\times$ directions, B, H). PyTorch concatenates gate chunks in input, forget, candidate, output order. The notebook reconstructs a learned cell and compares it numerically with `nn.LSTM`. Training updates use training data, validation selects a checkpoint, and test is evaluated afterward. Save raw data identity, seed, versions, window, split boundaries, normalization, model state, baselines, and measured errors. A seed alone does not guarantee bitwise agreement across environments.

1.3 Hour 2: why attention is useful and how it works

1.3.1 The fixed-context bottleneck

A final-state encoder compresses a source sequence into one fixed-width context. Different output steps may need different earlier details. Attention makes encoder states addressable and forms a context for each query. It does not guarantee correct retrieval or unlimited context. Attention

can be combined with recurrence: Bahdanau and colleagues used recurrent sequence-to-sequence modeling in a 2014 preprint presented in 2015. The Transformer architecture came later. Distinguish a retrieval mechanism from the entire architecture that uses it.

1.3.2 Queries, keys, values, and shapes

Queries encode a retrieval need. Keys encode matching information, while the corresponding value rows carry content to combine. For N_q queries, N_k keys, key width d_k , and value width d_v ,

$$Q \in \mathbb{R}^{N_q \times d_k}, \quad K \in \mathbb{R}^{N_k \times d_k}, \quad V \in \mathbb{R}^{N_k \times d_v}.$$

Values need not equal keys or have their width. A natural-language question is not the same object as the query vector produced by a learned projection.

Additive attention can use

$$e_{ij} = v_a^\top \tanh(W_q q_i + W_k k_j + b).$$

Scaled dot-product attention uses

$$S = QK^\top / \sqrt{d_k}, \quad A_{ij} = \frac{\exp(S_{ij})}{\sum_m \exp(S_{im})}, \quad C = AV.$$

Scores compare, row-wise softmax normalizes, and value multiplication retrieves. Rows are query destinations and columns are key sources. The context has shape (N_q, d_v) .

Under independent zero-mean unit-variance coordinate assumptions, the raw dot-product variance grows with d_k . Scaling by $\sqrt{d_k}$ controls that growth; it is not cosine normalization. Stable softmax subtracts the row maximum before exponentiation without changing normalized probabilities. Before later projections and residuals, the weighted context lies within the convex hull of the available values.

1.3.3 An attention calculation with no hidden steps

Choose teaching vectors

$$Q = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad V = \begin{bmatrix} 10 & 0 \\ 0 & 10 \\ 10 & 10 \end{bmatrix}.$$

Raw dot products are $(1, 0, 1)$ and scaled scores are approximately $(0.7071, 0, 0.7071)$. Softmax gives weights $(0.4011, 0.1978, 0.4011)$. The zero raw score does not give zero weight, because $\exp(0) = 1$. Combining corresponding value coordinates gives

$$\begin{aligned} C_1 &= 0.4011(10) + 0.1978(0) + 0.4011(10) \approx 8.0222, \\ C_2 &= 0.4011(0) + 0.1978(10) + 0.4011(10) \approx 5.9889. \end{aligned}$$

Use full precision in code. Change only the query and predict the new ranking. An optional positive temperature divides the scores: below one sharpens weights, above one softens them. The standard example uses one.

1.3.4 Self-attention, cross-attention, and masks

Self-attention projects one sequence X into $Q = XW_Q$, $K = XW_K$, and $V = XW_V$. Cross-attention derives queries from one stream and keys/values from another. An encoder-decoder model uses decoder queries with encoder memory. A decoder-only model normally has no separate source encoder.

In causal next-token training, input position i predicts token $i + 1$, so it may access itself and earlier inputs: $j \leq i$. Set future scores to $-\infty$ before softmax, yielding zero weights. Padding masks hide nonexistent tokens instead. Combine permissions and exclude padded query outputs from loss. An all-masked row has no valid normalization; prevent it or specify a deliberate special case. Mask conventions differ across APIs. PyTorch scaled-dot-product attention uses Boolean True for allowed entries, whereas some other attention calls use True for blocked entries.

1.3.5 Benefits, costs, and interpretation

One dense attention layer provides a direct query-to-key computational path to a distant position. Known query operations can run in parallel. Dense score interactions involve T^2 pairs per head: doubling context length quadruples pair count. Exact fused implementations can reduce intermediate memory without changing dense pair-count scaling.

Attention weights describe routing within the computation; they do not prove causal importance. Keep Q and K fixed and change V : weights stay identical but the result can change. Removing and renormalizing a source is a different intervention. Explain what a particular experiment supports rather than interpreting an attractive heatmap as a complete causal explanation.

1.4 Hour 3: assemble and test a Transformer

1.4.1 Tokens and position

Token IDs index learned vectors; their numeric magnitude has no inherent semantic distance. The teaching model has eight symbolic tokens, width 32, four heads, and context length 16. Vocabulary size and embedding width differ. Unmasked attention without positional information is permutation-equivariant: permuting input rows permutes outputs correspondingly. Order must therefore be represented deliberately. Our model adds learned absolute position vectors. The original Transformer used sinusoidal encodings; other position mechanisms also exist. A causal mask already introduces order constraints, so do not apply the unmasked equivariance claim unchanged to a masked model.

1.4.2 Heads, nonlinear transformations, and residual paths

Each head uses learned projections and its own routing weights. Concatenate head outputs and apply a learned output projection. At fixed total width D , H heads have width D/H in this implementation. More heads do not automatically multiply the total width, and heads have no assigned semantic roles. Attention mixes positions, while a shared tokenwise MLP transforms features independently at each position. The notebook expands to $4D$, applies GELU, and projects back to D .

Residual addition requires matching dimensions and preserves an identity route. The teaching decoder uses pre-normalization:

$$\begin{aligned} u &= x + \text{Attention}(\text{LN}(x)), \\ y &= u + \text{FFN}(\text{LN}(u)). \end{aligned}$$

The 2017 paper used a full encoder-decoder with post-normalization and ReLU. Our one-block decoder is an explicitly identified educational variant, not a verbatim copy of that architecture. Its causal self-attention has no encoder-decoder cross-attention sublayer.

1.4.3 Objective, baseline, and generation

Shift targets one token ahead. Logits have shape (B, T, V) , and cross-entropy minimizes

$$\mathcal{L} = -\frac{1}{BT} \sum_{b,t} \log p_{\theta}(y_{b,t} \mid x_{b,\leq t}).$$

Pass logits directly to `nn.CrossEntropyLoss`. Training computes many known positions in parallel under the causal mask. During generation, the model predicts one unknown token, appends it, and repeats. Parallel training does not imply arbitrary autoregressive generation is parallel.

Separate synthetic streams use seeds 2026, 2027, and 2028 but share one motif with small random perturbations. This is generalization to new realizations of that process, not transfer to a different engineering domain. A training-only smoothed bigram model is the reference. Uniform prediction over eight tokens has NLL $\log 8 \approx 2.079$ nats/token. Select a checkpoint by validation NLL; evaluate test afterward.

1.4.4 Behavioral checks and limits

Plot an actual trained attention head with query rows and key columns. Upper-triangle zeros are mask constraints, not learned discoveries. Change tokens only at positions 8 onward and compare logits at positions 0–7. They should remain unchanged within the declared tolerance. This tests one concrete causal behavior, not every deployment property. Separately test permutation equivariance of the unmasked operation. Removing positions from an already trained decoder is an inference intervention, not a fair separately trained architecture comparison.

The model generates symbolic states, not evidence-backed natural language. A token generator is only one possible component of an agent. An agent also needs a bounded objective, state handling, validated tools, permissions, stopping rules, and evidence review. Generation alone does not provide factual verification or controlled execution.

1.5 Completion and explain-back

H1 must produce a baseline table, residual plot, a reconstructed learned cell update, and evidence JSON. H2 must produce scores, weights, context, mask assertions, and an interpretation-limiting intervention. H3 must produce a trained-model comparison, head plot, future-token check, and configuration record. Predict the effect of one change before executing, keep the original evidence, and finish with a claim, evidence, limitation, and a test that could challenge the claim.

1.6 Primary sources

- Hochreiter and Schmidhuber (1997), *Long Short-Term Memory*: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- Bahdanau, Cho, and Bengio (2014/2015), recurrent attention: <https://arxiv.org/abs/1409.0473>.
- Vaswani et al. (2017), *Attention Is All You Need*: <https://arxiv.org/abs/1706.03762>.
- Dive into Deep Learning: https://d2l.ai/chapter_recurrent-modern/lstm.html and https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html.
- Executable course notebooks: <https://github.com/CHJonnyYu/MLES-Fall-2026>.