Convolutional Sequence Models

Causal convolutions, dilation, TCNs and WaveNet

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

Why another sequence architecture?

Where we are

We have built two families of **next-token** models over a sequence $x_1, \ldots, x_T$:

- a **fixed-window MLP** — condition on the last k tokens (Lecture 11);
- a **recurrent network** — carry a running state h_t (Lecture 12).

Both estimate the **same** object: the conditional $p(x_t \mid x_{<t})$. Today we add a **third** way to compute it — one built from **convolutions** over time.

The objective is architecture-agnostic

D DERIVATION

Every language model factorizes the sequence the **same** way — the chain rule:

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$$

and is trained by the **same** negative log-likelihood:

$$\mathcal{L} = -\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t})$$

the **objective** is fixed; only the **function** that produces $p(x_t \mid x_{<t})$ changes

Three ways to model one conditional

The architectures differ only in **how** they summarize the past $x_{<t}$:

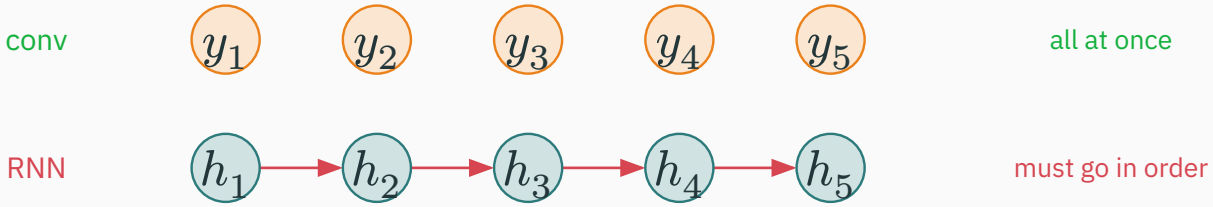
Model	**Conditioning**	**Mechanism**
MLP-LM (L11)	$p(x_t \mid x_{t-k:t-1})$	fixed window of k tokens
RNN-LM (L12)	$h_t = f(h_{t-1}, x_t)$	recurrent state, one step at a time
Conv-LM (today)	$g(\mathrm{CNN}(x_{\leq t}))$	**convolution over time**

same makemore names, same loss — a new way to build the feature that feeds the softmax

Why convolutions?

V VISUAL

An RNN computes h_t **from** h_{t-1} — position t must wait for position $t-1$:



a convolution applies the **same** kernel to **every** position — all outputs computed **in parallel** during training

Four questions for today

To turn a CNN into a sequence model we must answer:

1. how does a **1-D convolution** slide over time?
2. how do we stop an output at t from **seeing the future** $x_{>t}$?
3. how do we reach **long context** without a huge, deep stack?
4. how does the result **compare** to an RNN?

causal convolution · dilation · TCN / WaveNet

1-D convolution over sequences

A sequence is a tensor

Batch a set of sequences into one tensor — three axes:

$X \in \mathbb{R}^{B \times T \times D}$

- B — examples in the batch;
- T — **time** / position along the sequence;
- D — **channels** (features) at each position — e.g. the embedding dimension.

a 1-D convolution slides **along the T axis**, mixing across the D channels — the image conv of L8, one dimension down

1-D convolution, one channel

D DERIVATION

Slide a length-k kernel w along the sequence, taking a weighted sum:

$$y_t = \sum_{i=0}^{k-1} w_i \, x_{t+i}$$

- the **same** weights $w_0, \ldots, w_{k-1}$ at **every** position t — weight sharing;
- each output looks at only k neighbouring inputs — locality.

exactly the 1-D conv of L8 — now the axis it slides along is **time**

Multiple channels in and out

Real layers map C_{in} channels to C_{out} channels, just like a 2-D conv:

Tensor	Shape	Meaning
input X	$T \times C_{\text{in}}$	C_{in} features per step
weights W	$k \times C_{\text{in}} \times C_{\text{out}}$	C_{out} temporal kernels
output Y	$T' \times C_{\text{out}}$	C_{out} feature streams

$$Y_{t,c} = \sum_{i=0}^{k-1} \sum_{c'} W_{i,c',c}\, X_{t+i,c'}$$

each output channel sums over **all** input channels across a k-wide temporal window

Worked example: a scalar 1-D conv

D DERIVATION

Input $x = [1, 2, 3, 4, 5]$, kernel $w = [2, -1]$ (valid cross-correlation):

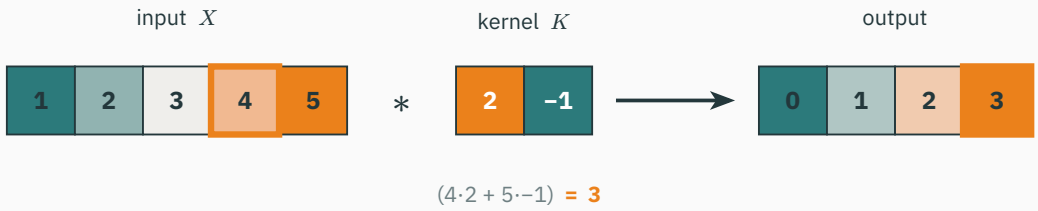
$$y_t = 2\,x_t - x_{t+1}$$

$$y_1 = 2(1) - 2 = 0, \quad y_2 = 2(2) - 3 = 1, \quad y_3 = 2(3) - 4 = 2, \quad y_4 = 2(4) - 5 = 3$$

$$\boxed{y = [0, 1, 2, 3]}$$

The same conv, visually

V VISUAL



a length-2 kernel over 5 inputs → 4 outputs; the **same** $[2, -1]$ is reused at every step

What a temporal kernel detects

A learned temporal kernel responds to one **local pattern in time**:

Kernel	**Detects**
$[-1, 1]$	a **rise** — value going up
$[1, -2, 1]$	a local **peak** / curvature
oriented weights	a short **n-gram** or motif
smoothing weights	a **trend** / running average

As in vision, these kernels are **not** hand-designed — they are **learned** by gradient descent to detect whatever temporal motifs help predict the next token.

Connection to the fixed-window MLP

D DERIVATION

The MLP-LM also read a window of k tokens — but it used a **different** weight block per slot:

$$\underbrace{\text{MLP: separate } W \text{ per relative position}}_{\text{position-specific}} \neq \underbrace{\text{conv: one shared kernel } w}_{\text{temporal weight sharing}}$$

The MLP relearns each position separately; move a motif one step and it hits **different** weights. A convolution applies the **same** kernel everywhere — the pattern is detected wherever it occurs.

Translation equivariance in time

D DERIVATION

Because one kernel is shared across all t, **shifting the input shifts the output the same way**:

$$f(T_\Delta\, x) = T_\Delta\, f(x)$$

where T_Δ is a shift by Δ steps. A motif that occurs later produces the same response, later.

> the temporal analogue of the image equivariance from L8 — one detector, valid at every moment

Interactive: temporal convolution

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/convolution-visualizer

Slide a 1-D kernel over a signal and watch each multiply-add produce one output — change the weights and see how the detected temporal pattern changes.

Q. Which kernel fires on a **falling** edge in the signal?

Causal convolution

The future-information problem

Next-token prediction at t must depend **only** on $x_{\leq t}$ — never on what comes later. A **centered** conv $y_t = \sum_{i=-r}^{r} w_i x_{t+i}$ reads $x_{t+1}, x_{t+2}, \ldots$ — the **future**:

If the representation at t peeks at x_{t+1}, then "predicting" the next token is **cheating** — the answer has already leaked in. Training loss looks great; generation is impossible.

The causal convolution

D DERIVATION

Flip the window so it reaches only **backward** in time:

$$y_t = \sum_{i=0}^{k-1} w_i \, x_{t-i}$$

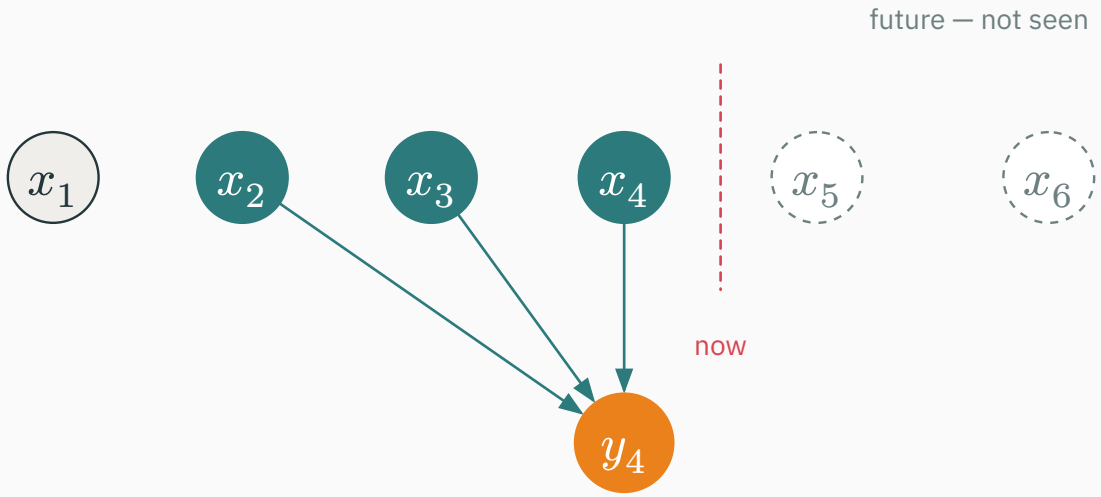
- y_t depends on $x_t, x_{t-1}, \ldots, x_{t-k+1}$ — the present and the past;
- **nothing** to the right of t enters the sum.

same weights, same cost — only the **window's direction** changed

Causal connectivity

V VISUAL

With $k = 3$, the output y_4 is wired to x_2, x_3, x_4 only:



the boundary at **now** is never crossed — every arrow points from the past

Left padding keeps the length

D DERIVATION

To keep y the same length as x, pad $k-1$ zeros on the **left** (not symmetrically):

$$\left[\underbrace{0, \ldots, 0}_{k-1 \text{ zeros}}, x_1, x_2, \ldots, x_T\right]$$

For $k = 3$: prepend $[0, 0]$, so y_1 sees $[0, 0, x_1]$, y_2 sees $[0, x_1, x_2]$, ...

padding is **asymmetric** — all on the left — precisely so no output reaches forward in time

Shifted targets

Feed the sequence in, ask each position to predict the **next** token:

input (positions $1 \ldots T$)	**target** (predict next)
`<BOS> deep learning is`	`deep learning is useful`

the causal representation at position t is trained to predict the token at position $t + 1$

Strict vs inclusive causality

★ OPTIONAL

Two consistent conventions — both correct, if you shift the targets to match:

Convention	y_t uses	Predicts
inclusive	$x_{\leq t}$	x_{t+1}
strict	$x_{<t}$	x_t

The only requirement is **consistency**: whatever y_t is allowed to see, the target must be a token it has **not** seen. Mismatch here is the classic "my LM has zero loss but generates garbage" bug.

Interactive: causal convolution

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/causal-conv

Drag the kernel across the sequence and watch it **forbidden** from crossing the “now” line; toggle left-padding and see the output length change.

Receptive field

How far back can one output see?

The **receptive field** R of an output = how many input positions can affect it.

- a **single** causal conv layer, kernel k: $R = k$;
- stack layers and the field **grows** — deeper outputs see **more** history.

context length is set by the receptive field, not by any fixed window k

Stacking ordinary causal convs

D DERIVATION

Each stride-1 layer adds $k-1$ to the receptive field:

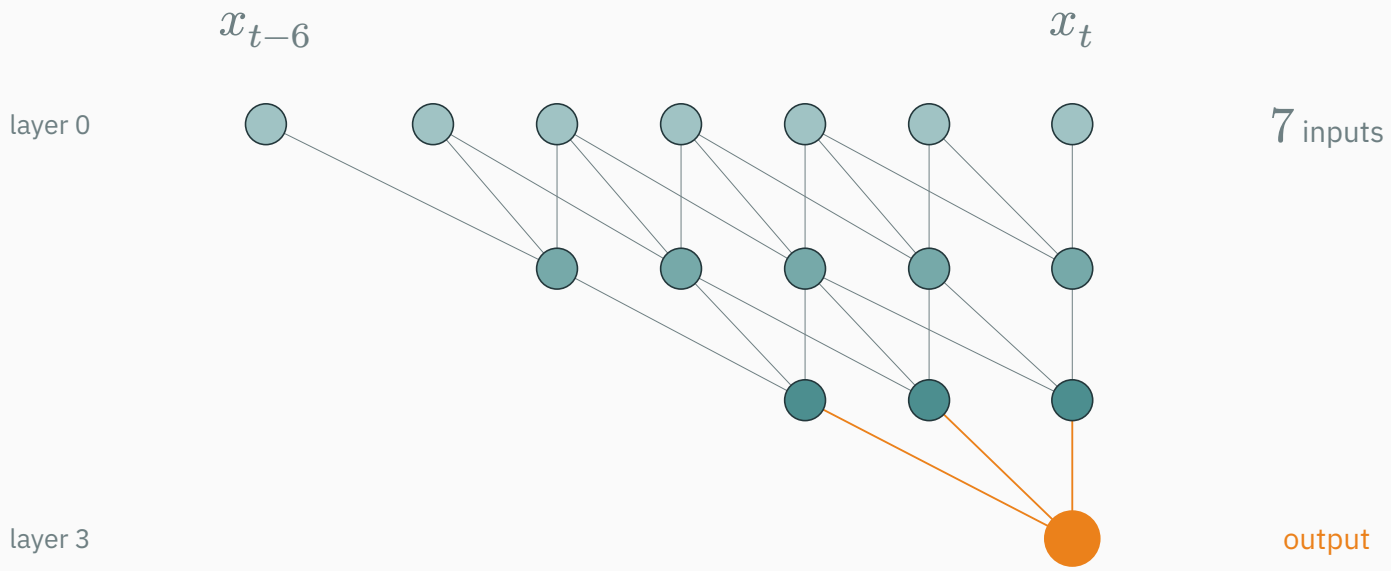
$$R_L = 1 + L(k-1)$$

For $k=3$: $R_L = 1+2L$. Three layers, $R_0 = 1$:

$$R_1 = 3, \quad R_2 = 5, \quad R_3 = 7$$

Stacking ordinary causal convs

D DERIVATION



The problem: linear growth

D DERIVATION

The receptive field grows only **linearly** in depth. To cover $1{,}001$ steps with $k = 3$:

$$1 + 2L = 1{,}001 \quad \Rightarrow \quad L = 500 \quad \text{layers}$$

500 layers just to see 1000 steps back — far too deep to train or run. Stacking ordinary convs cannot reach long context economically.

we need the field to grow faster than linearly — enter dilation

Dilated convolution — the key idea

Skip positions inside the kernel

D DERIVATION

A **dilated** convolution spreads its k taps apart by a factor d:

$$y_t = \sum_{i=0}^{k-1} w_i \, x_{t-di}$$

For $k = 3, d = 2$:

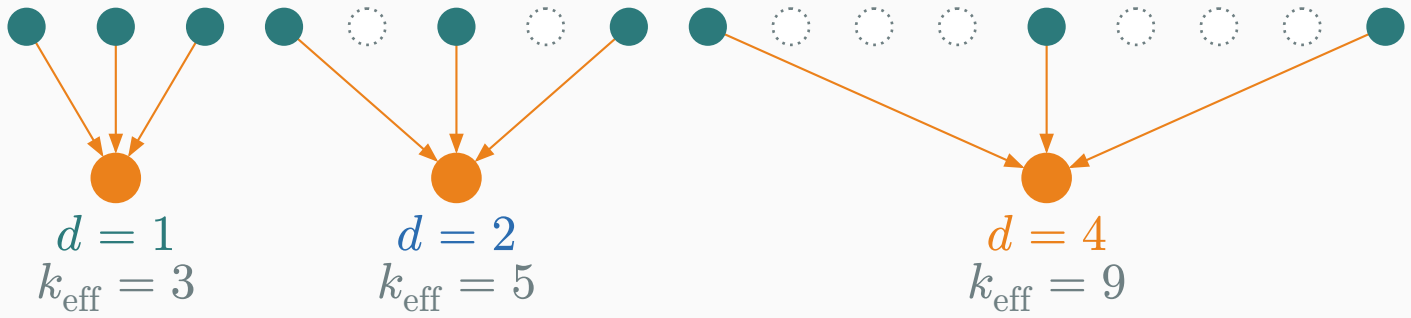
$$y_t = w_0 \, x_t + w_1 \, x_{t-2} + w_2 \, x_{t-4}$$

still only k weights — but they **reach** over d-spaced positions, skipping those between

Dilation, fan by fan

V VISUAL

The **same three taps** reach exponentially further as d doubles:



teal = tapped, dotted = skipped — 3 weights spanning 3, then 5, then 9 positions

Effective kernel width

D DERIVATION

A dilated kernel of size k spans an **effective** width:

$$k_{\text{eff}} = 1 + (k - 1)d$$

For $k = 3, d = 4$:

$$k_{\text{eff}} = 1 + 2 \cdot 4 = 9$$

3 parameters, but a 9-wide temporal footprint — coverage without cost

Receptive field of a dilated stack

D DERIVATION

Summing the per-layer contributions:

$$R_L = 1 + (k-1)\sum_{\ell=1}^{L} d_\ell$$

Choose **exponential** dilation $d_\ell = 2^{\ell-1}$, so $\sum_{\ell=1}^{L} d_\ell = 2^L - 1$:

$$R_L = 1 + (k-1)(2^L - 1)$$

For $k = 2$ this is exactly $R_L = 2^L$ — the field **doubles** with each layer.

linear depth, **exponential** context — the whole point of dilation

Worked dilated receptive field

D DERIVATION

Four layers, $k = 3$, dilations $d = [1, 2, 4, 8]$:

$$R = 1 + (k - 1) \sum d_\ell = 1 + 2(1 + 2 + 4 + 8) = 1 + 2 \cdot 15 = 31$$

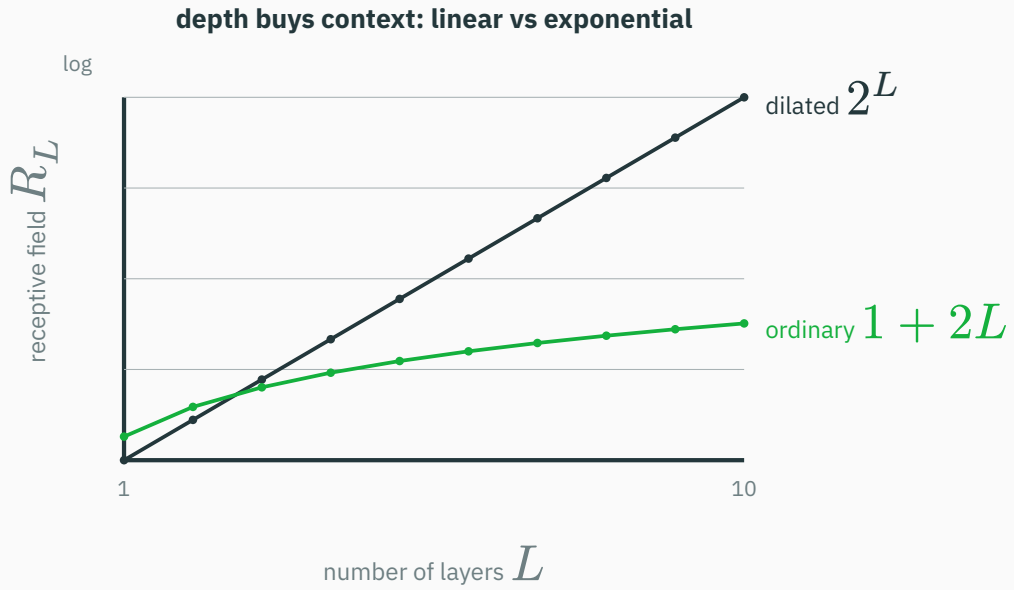
Four **ordinary** causal layers ($k = 3$) reach only:

$$R = 1 + 4 \cdot 2 = 9$$

31 **with dilation vs** 9 **without — same depth, same parameters**

Linear vs exponential, side by side

V VISUAL

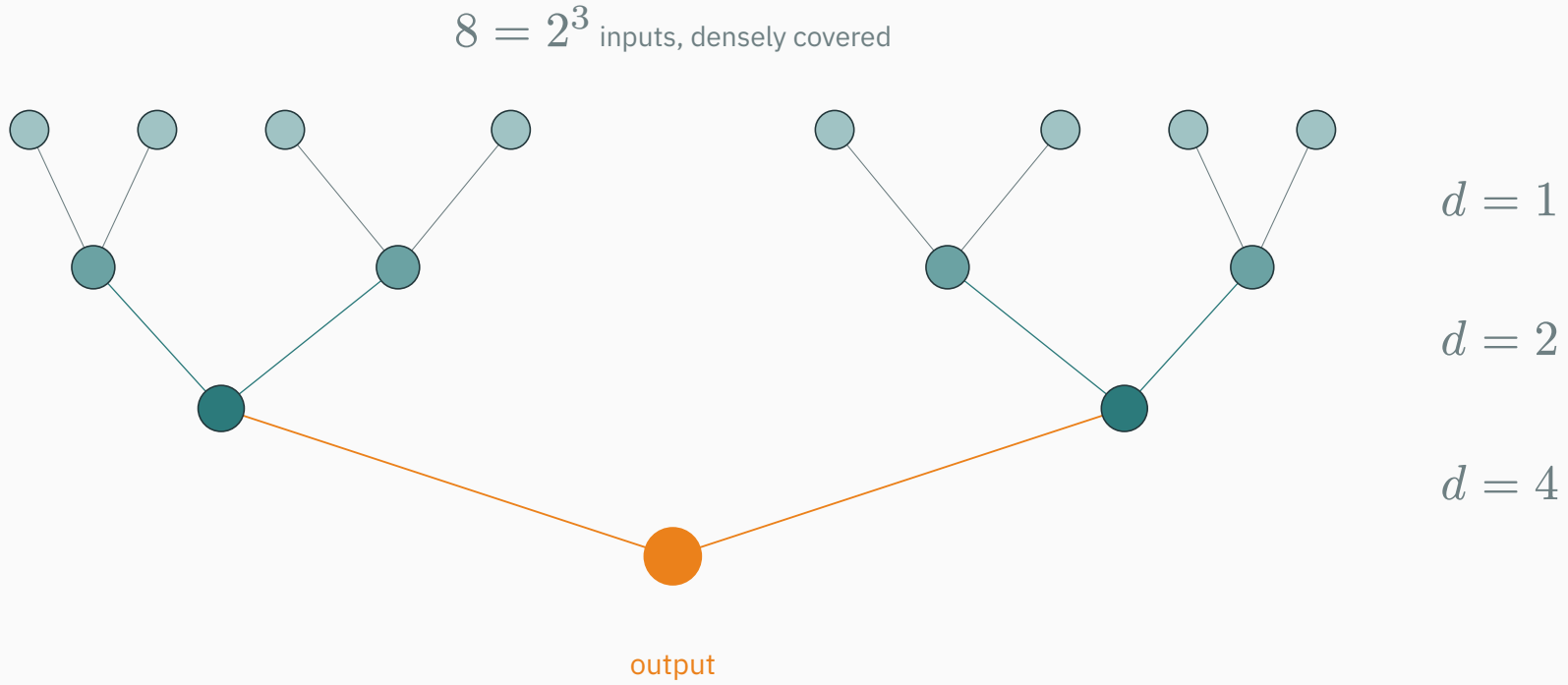


ordinary $1 + 2L$ crawls; dilated 2^L reaches $1{,}024$ in ten layers

Sparse per layer, dense in total

V VISUAL

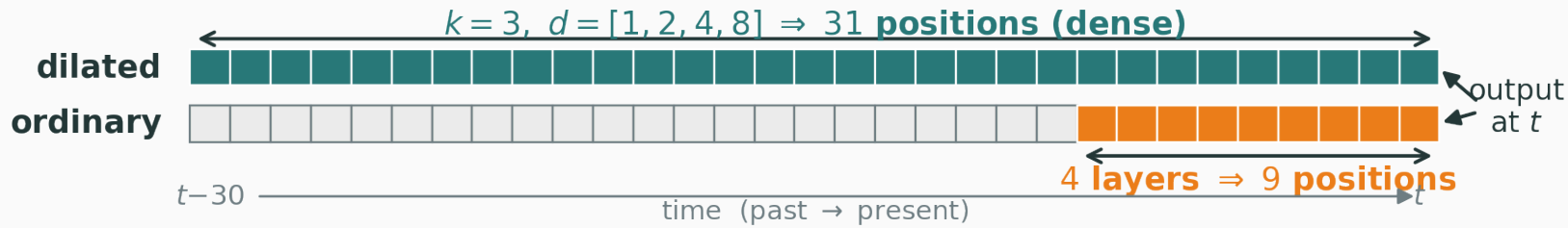
Each dilated layer is **sparse**, but stacked they **tile** the past with no gaps:



$k = 3, d = [1, 2, 4, 8]$ covers 31 contiguous positions; $k = 2, d = [1, 2, 4]$ covers $2^3 = 8$

The coverage, counted

V VISUAL



the top output reaches 31 dense past positions; an ordinary 4-layer stack reaches only 9

The dilation cycle

Stacks usually **cycle** the dilation and repeat:

$1, 2, 4, 8, \ 1, 2, 4, 8, \ 1, 2, 4, 8, \ldots$

- each cycle mixes **local** ($d = 1$) with **long-range** ($d = 8$) structure;
- repeating cycles deepen the network while the field keeps growing;
- this is exactly the WaveNet stacking pattern.

reset-and-grow: fine detail is re-examined at every cycle, never lost to the long jumps

Interactive: dilated receptive field

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/receptive-field-grower

Add dilated layers one at a time — $d = 1, 2, 4, 8, \ldots$ — and watch the receptive field **double** with each layer as it fans back over the input.

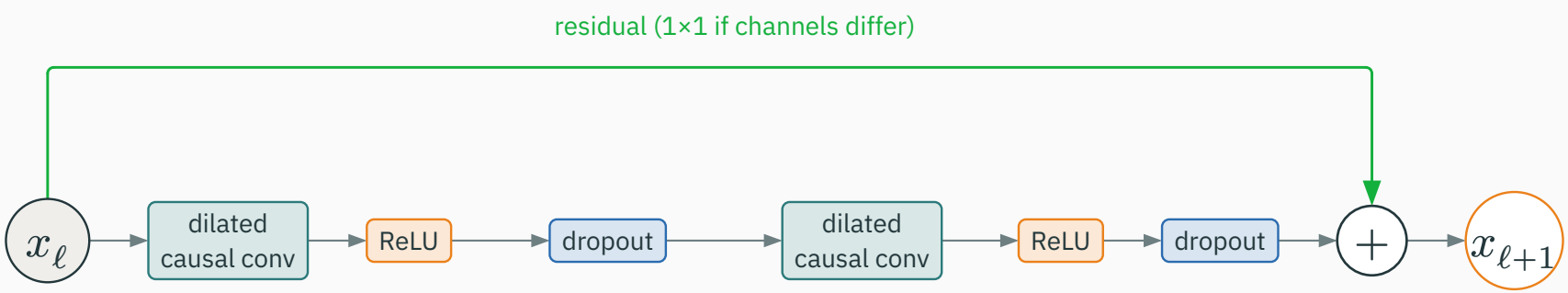
Q. With $k = 2$, how many layers reach a context of $1{,}024$?

The Temporal Convolutional Network

The TCN block

V VISUAL

A TCN block = **two dilated causal convs**, each with activation and dropout, plus a **residual**:



Bai, Kolter & Koltun (2018) — dilated causal conv is the primitive; the block wraps it

Residual keeps deep stacks trainable

D DERIVATION

The block learns a **residual** F added to its input, exactly as in ResNet:

$$x_{\ell+1} = x_\ell + F(x_\ell), \qquad \text{or} \quad x_{\ell+1} = P(x_\ell) + F(x_\ell)$$

- P is a 1×1 conv, used **only** when the channel counts differ;
- the identity path lets gradients skip the dilated convs — deep dilation stacks train.

the same trick that tamed deep MLPs and CNNs (L6, L8b) returns for deep temporal stacks

Sequence length is preserved

Causal (left) padding makes every layer map T inputs to T outputs:

$$T_{\text{in}} = T_{\text{out}} \quad \text{at every layer}$$

- no shrinking with depth — stack as many dilated blocks as you like;
- one output stream per position, ready for a per-position prediction.

unlike a valid conv that shrinks, a TCN holds the timeline fixed end to end

TCN as a next-token model

D DERIVATION

Embed, run the dilated causal stack, project each position to logits:

$$E_t = E[x_t], \qquad H_{1:T} = \mathrm{TCN}(E_{1:T})$$

$$z_t = W_o\, h_t + b_o, \qquad p(x_{t+1} \mid x_{\le t}) = \mathrm{softmax}(z_t)$$

same makemore head as the MLP-LM and RNN-LM — only the feature extractor is a TCN

One forward predicts every position

Because the whole stack is convolutional, a **single** forward pass emits all outputs:

$$z_1, z_2, \ldots, z_T \quad \text{predict} \quad x_2, x_3, \ldots, x_{T+1} \quad \text{at once}$$

Causal padding guarantees z_t never saw $x_{>t}$, so computing **all** positions in parallel is still correct. This is the training-time advantage over an RNN's step-by-step recurrence.

Parameter sharing: across time, not depth

D DERIVATION

Where do the weights get reused?

Model	Weights reused …
RNN	across **time steps** — the **same** transition every step
TCN	across **positions within a layer**; each **layer** has its own kernel

a TCN shares a kernel along time, but **different layers** learn different kernels — depth is not tied

Effective vs theoretical memory

The receptive field gives an **upper bound** on memory — the model need not use all of it:

- theoretical context $= R_L$; **effective** context may be shorter;
- TCNs can achieve **long** effective memory and, on some long-range benchmarks, match or beat LSTMs.

recurrence is not required for long-range sequence modeling

WaveNet — dilated causal convs on raw audio

The objective

WaveNet models a **raw audio waveform** $x_1, ..., x_T$ autoregressively — sample by sample:

$$p(x_1, ..., x_T) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$$

van den Oord et al. (2016) — the **same** chain rule, now over audio **samples** instead of characters

Why audio is hard

Raw audio is a brutal sequence-modeling target:

- **thousands** of samples per second → sequences of tens of thousands of steps;
- prediction must be **exactly causal** (no future leakage);
- needs **large context** (a phoneme spans thousands of samples) at **fine resolution**.

dilated causal convolutions were designed to hit **all** of these at once

Gated activation

D DERIVATION

WaveNet replaces the plain activation with a **gated** one — LSTM-like, inside a conv block:

$$z = \underbrace{\tanh\big(W_f * x\big)}_{\text{candidate}} \odot \underbrace{\sigma\big(W_g * x\big)}_{\text{gate}}$$

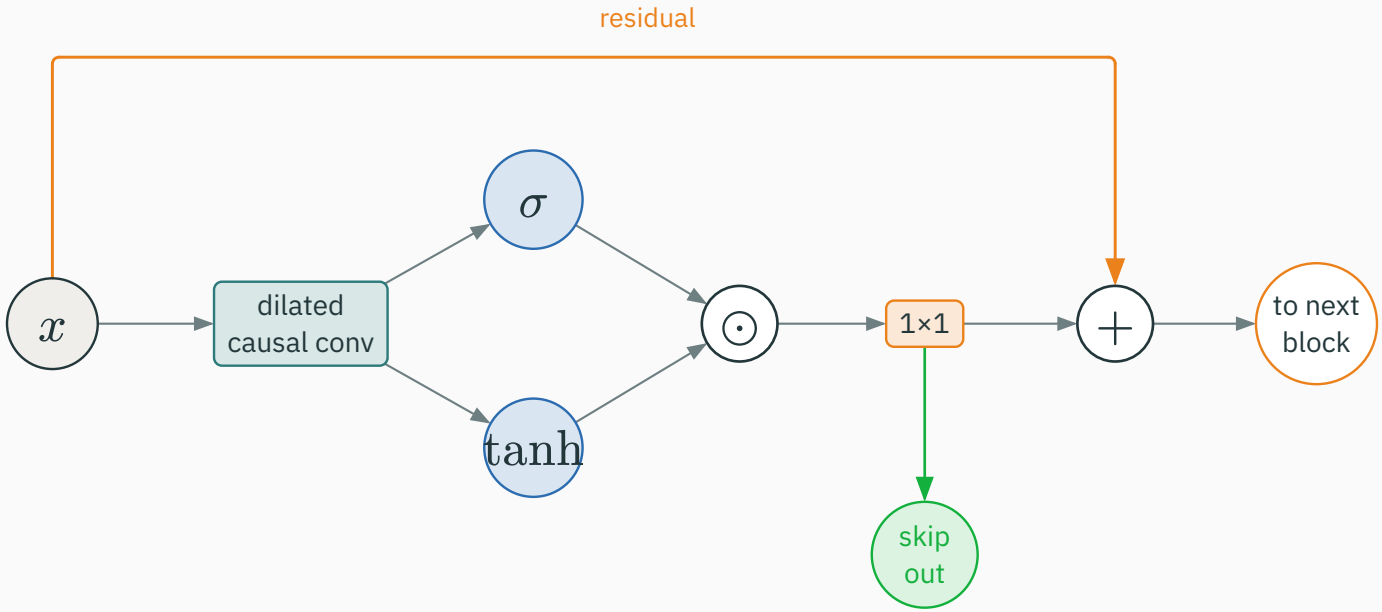
- $\tanh$ proposes a value, $\sigma \in (0, 1)$ decides how much passes;
- $\odot$ is element-wise — the multiplicative gate from L12, now convolutional.

gating gives the block fine control over what each dilated filter contributes

Residual and skip paths

V VISUAL

Each block feeds a **residual** forward and a **skip** to the output sum:



gated conv → 1×1 → split: a residual to the next block, a skip to the final sum

The WaveNet stack

Assemble the blocks into the full model:

1. a causal conv on the input;
2. a stack of **gated, dilated, residual** blocks — dilation $1, 2, 4, \ldots, 512$, then **repeat**;
3. **sum all skip outputs** → ReLU → 1×1 → ReLU → 1×1 → logits.

stacked dilation cycles give a receptive field of **thousands** of samples with modest depth

Training vs generation

The parallelism cuts one way only:

Training
the whole waveform is known → **all** positions in **one** parallel forward pass.

Generation
each sample feeds the next → **sequential**, one step at a time (unless conv states are cached).

fast to **train** on known data; still autoregressive — hence slow — to **sample**

Beyond audio

The same recipe of **causal conv, dilation, residual, and autoregression** reaches far beyond audio:

- time-series forecasting; language modeling (ByteNet);
- sensor streams, financial ticks; biological sequences.

WaveNet's contribution is less "audio" than a **general blueprint** for autoregressive sequence models built entirely from dilated causal convolutions.

Interactive: WaveNet block

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/gated-residual

Step through one gated residual block: watch $\tanh$ and σ combine, the skip branch peel off, and the residual rejoin for the next block.

Convolutional vs recurrent sequence models

Two ways to reach the past

Both see history — but **how** differs fundamentally:

RNN

compresses all of $x_{\leq t}$ into a fixed-size **recurrent state** h_t — a lossy running summary.

TCN

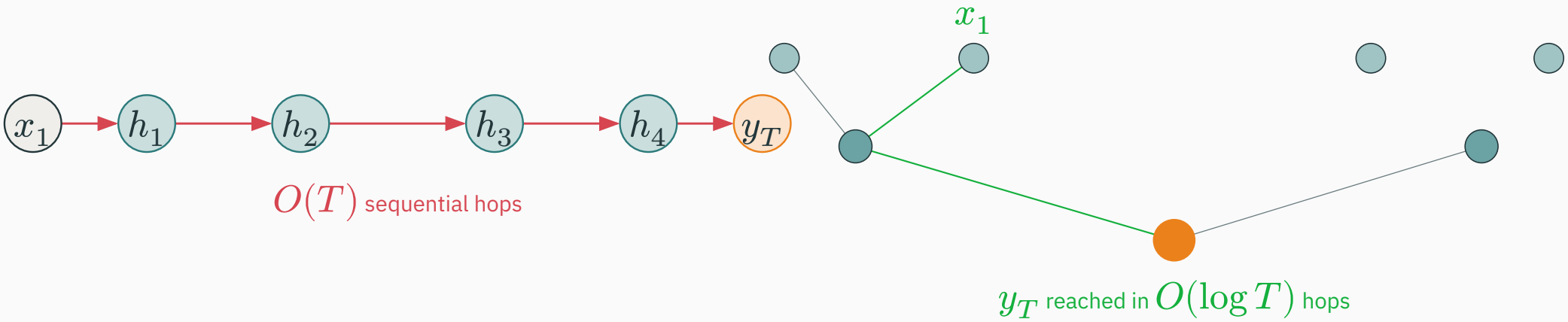
reads a **finite receptive field** of raw positions directly — no compression, but bounded reach.

state (unbounded but lossy) vs receptive field (lossless but finite)

Computational path length

V VISUAL

How many steps must a gradient travel between a distant input and the output?



RNN: $O(T)$ sequential hops from x_1 to y_T · dilated CNN: only $O(\log T)$ layers — far shorter gradient paths

Parallelism, side by side

	RNN	TCN
parallel training	limited (sequential)	high (all positions)
generation	sequential	sequential
context mechanism	recurrent state	receptive field
context length	unbounded (in principle)	finite (R_L)
weight sharing	across steps	across positions
path to distant input	$O(T)$	$O(\log T)$

Inductive bias — neither is universal

Each architecture **assumes** something about the sequence:

- **RNN** — a sequential **state update**: the past matters through an evolving summary;
- **CNN** — **local patterns** composed **hierarchically**: motifs, then motifs of motifs.

Neither is universally better. The right choice depends on sequence **length**, needed **context**, **latency**, **compute**, and the **task**.

Why Transformers still become attractive

The TCN fixes the RNN's three pains — parallel training, short gradient paths, fast field growth. But:

A convolution's aggregation pattern is **predetermined** by its kernel and dilation. **Which** past positions matter is fixed by the architecture, not by the **content**.

attention lets each position dynamically choose which past positions to attend to

content-dependent, all-to-all mixing — the transition we take up next

Preview and summary

Convolutional encoder–decoder

Convolutions can build a **full** sequence-to-sequence model, not just a language model:

$$H = \mathrm{CNN}_{\mathrm{enc}(x)}, \qquad s_u = \mathrm{CNN}_{\mathrm{dec}(y_{<u}, H)}$$

- **ByteNet** and **ConvSeq2Seq** (Gehring et al.) built translation entirely from convolutions;
- a causal decoder conv over $y_{<u}$, conditioned on the encoder features H.

a one-slide preview — the encoder–decoder structure is the topic of the next lecture

Final summary

Idea	**Takeaway**
1-D convolution	detect **local temporal patterns**; weight sharing along time
causal convolution	left padding → no future leakage; predict the next token
receptive field	ordinary stacking grows it only **linearly** $(1 + 2L)$
dilation	$d_\ell = 2^{\ell-1}$ → context grows **exponentially** with depth
TCN	causal dilated convs + residual; parallel over time
WaveNet	gated dilated causal convs + autoregressive generation

Final mental model — one idea

A sequence can be modeled **without recurrence** —

by **convolving over time**, **causally**, with **dilation** for reach.

parallel training · short gradient paths · context that grows with depth

Convolutions reach the past through a **fixed** receptive field.

Next: **sequence-to-sequence and attention** — where each position **chooses** which others to attend to, dynamically.