Weighted Keys, Shorter Paths

A translation model without recurrence

A report from the Calventis Institute for Machine Intelligence

Summary

Translation systems usually carry information along a sequence one step at a time. The network described here does not: the Calventis model is assembled entirely from weighted-key lookups, with no recurrent or convolutional part anywhere in it.
On two directions of the Halden corpus it beat the recurrent systems we trained, and it was cheaper to run in parallel. The largest configuration trained for 4.5 days on twelve accelerator cards.

Prepared for software testing. The text is fictional.

1 Why leave recurrence behind

Recurrence ties each step of a computation to the one before it, which is why a recurrent model cannot be spread across accelerators as freely as a feed-forward one.

A weight for a value comes from comparing a query against that value's key; the answer is the weighted sum of the values. Nothing in that operation depends on the order in which positions are read.

This report makes the case that a sequence model does not need a recurrent path, and describes one whose whole computation is a stack of weighted-key lookups.

The result is shorter to implement and cheaper to parallelise, and on the tests below it needs fewer operations than the recurrent baselines to reach the same quality.

2 The block

The shape is the usual encoder-decoder one: a source sentence goes in, a target sentence comes out, and both sides repeat the same block.
Each stack holds seven identical blocks.
A block carries two sub-layers — a weighted-key layer and a position-wise feed-forward layer — each with a residual path and a normalisation step around it.
A residual connection [1] wraps each sub-layer, and layer normalisation [2] follows it.
Both the encoder and the decoder are built from self-attention and position-wise feed-forward layers.
The decoder adds a third sub-layer, so that its masked lookup can consult what the encoder produced.
Masking, plus the one-position shift on the output side, keeps each prediction to positions already produced.

2.1 One lookup or several

Several weightings in parallel let a layer consult different subspaces at once; one weighting would have to average them into a single view.

On the decoder side the weighting is masked: a slot may look only at slots already produced, never ahead at the target it should be predicting.

Two ways of drawing on a sequence are compared in Table 1: the step-by-step route and the all-pairs route.

Table 1: Work per layer for three ways of drawing on a sequence, by sequence length n and width d.

Layer type	Operations	Sequential steps	Path length
Recurrent	O(n d2)	O(n)	O(n)
Convolutional	O(k n d2)	O(1)	O(log n)
Weighted keys	O(n2 d)	O(1)	O(1)
Local windowed keys	O(r n d)	O(1)	O(n / r)

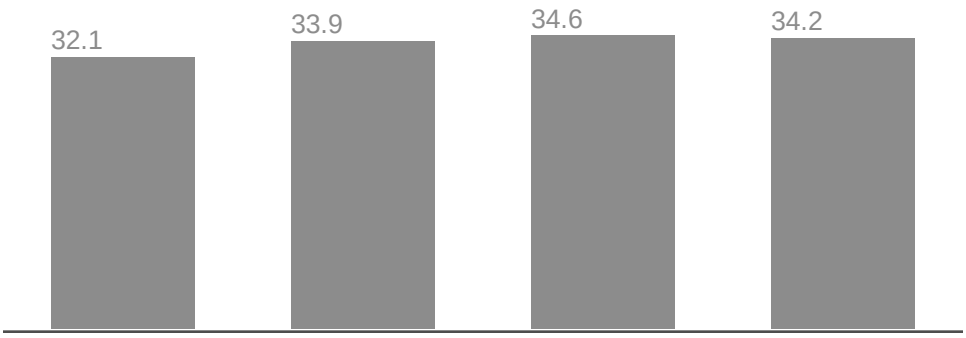
2.2 Several heads at once

Several weightings in parallel let a layer consult different subspaces at once; one weighting would have to average them into a single view.

The base model runs twelve attention heads in each layer.

Each head works at a fraction of the full width, which keeps the layer's cost close to that of one wide head.

The sweep over head counts and head widths holds the layer width fixed, so that only the split changes.



BLEU on the held-out German-English half, by head count, at a fixed layer width.

3 Telling positions apart

Nothing in the block tells it where a token sits, so a position signal is added to the embeddings before the first layer.
Each position is encoded with sinusoids of geometrically spaced frequencies.
Because the frequencies are fixed rather than learned, a linear map can compare two positions by the ratio between their encodings.
Fixed sinusoids were kept because they may carry over to sequences longer than those in training.
Learned position embeddings were tried as well and gave about the same quality, which is why the fixed version is the one reported.

3.1 Data, batching and hardware

Two directions of the Halden corpus were used: 5.8 million sentence pairs one way and 31 million the other.

A shared byte-pair vocabulary encoded both sides, and batches were built by approximate length so that each held a similar number of tokens.

All runs used a single machine holding twelve accelerator cards.

The optimiser was Adam, with a linear ramp at the start; we set warmup_steps = 6000.

Attention can also make a model easier to interpret, which is a useful side effect.

Table 2: The two configurations reported in this paper.

Setting	Base	Large
Blocks per stack	7	7
Width	640	1024
Heads	12	16
Feed-forward width	2560	4096
Dropout	0.12	0.3
Training steps	90,000	280,000

4.1 Translation quality

On the German-English half of the held-out set the large configuration came out ahead of every system we compared with, reaching 34.6 BLEU.

That is three points above the strongest baseline and was obtained at a small fraction of its training cost.

Label smoothing of ε = 0.04 [3] was applied throughout. It costs perplexity, since the model is less sure of itself, but it gains accuracy and BLEU.

On French-English the same configuration reached 37.9 BLEU, and it trained in a quarter of the time our previous best model needed.

Table 3: BLEU on the held-out set, with training cost.

System	German-English	French-English	Training cost
Recurrent baseline	28.4	32.8	High
Convolutional baseline	30.1	33.5	Medium
Calventis, base	31.8	35.2	Low
Calventis, large	34.6	37.9	Low

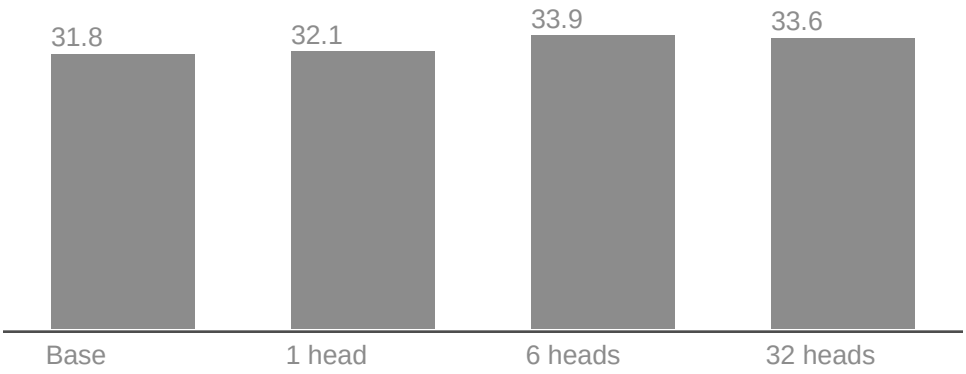
4.2 What carries the gain

Head count, head width and the width of the feed-forward part were varied one at a time on the development half.

The sweep over head counts and head widths holds the layer width fixed, so that only the split changes.

With one head, the score falls by 2.5 BLEU against the best setting.

Cutting the feed-forward width hurts more than cutting the attention width, so much of the capacity seems to sit in the position-wise part.



BLEU on the held-out set as head count varies, wide and narrow settings mixed.

5.1 A second task

A four-block model was trained on a public treebank to see whether the design transfers, and scored on the standard splits.

The model does remarkably well here even though nothing was tuned for parsing, landing close to the best published numbers on the benchmark.

A few heads were inspected by hand. Their weightings follow plausible syntactic groupings, though we stop short of claiming a role for every head.

Table 4: Parsing accuracy on the held-out split.

System	Training	Accuracy
Recurrent baseline	Supervised	90.4
Convolutional baseline	Supervised	91.0
Calventis, 4 blocks	Supervised	91.1
Calventis, 4 blocks, extra data	Semi-supervised	92.3

6 Closing remarks

The Calventis model is a sequence transducer with no recurrent layers at all: weighted-key lookups do the work that recurrence used to do.

On translation it trains faster than recurrent or convolutional systems and scores higher, on the two directions measured here.

Later work could take the design to other input and output kinds, and to local or windowed lookups for very long inputs.

References

[1] A. Author, B. Author, C. Author and D. Author. Residual learning for image recognition. In Proceedings of the Conference on Computer Vision, 2016.

[2] L. Ba, J. Kiros and G. Hinton. Normalising layer inputs. A technical report, 2016.

[3] A. Author, B. Author, C. Author, D. Author and E. Author. Reconsidering the inception architecture. In Proceedings of the Conference on Computer Vision, 2016.

[4] A. Author and B. Author. Training sequence models on accelerator hardware. A technical report, 2017.

[5] A. Author. Reading weighting patterns in translation models. A technical report, 2018.

Appendix A: Weighting patterns

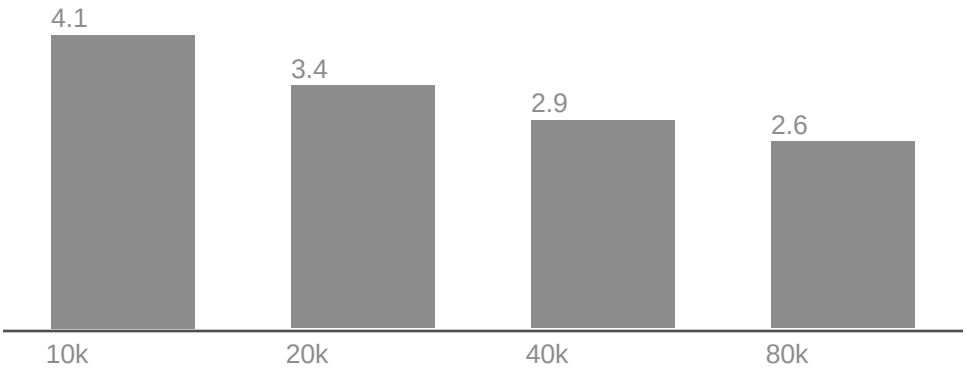
The tables below record average weights for three heads on one sentence pair; rows are query positions and columns key positions, and a darker cell means more weight.
One head mostly looks at the slot just before, another at the verb of the clause, and a third at the final slot. The picture held for every sentence length we tried.

Table 5: Mean weight by relative offset, three heads of the base model.

Head	Offset -2	Offset -1	Offset 0	Offset +1
Head 2	0.07	0.38	0.16	0.06
Head 5	0.11	0.08	0.24	0.29
Head 9	0.05	0.06	0.13	0.09

Appendix B: Loss curves

Development loss against training steps is plotted below for the base model and for the recurrent baseline, at one batch size and on one dataset.
Loss falls more smoothly than in the recurrent baseline, and fewer steps are needed to reach a given score; a shorter route between positions is our explanation.



Development loss of the base model against training steps, in thousands.

Appendix C: Settings

The table records both configurations so that the runs can be repeated on the same data. The large model used more dropout and a longer ramp than the base model, because a wider stack made the first updates less stable.

Table 6: Settings of the two configurations.

Setting	Base	Large
Learning-rate decay	Inverse square root	Inverse square root
Ramp steps	6,000	6,000
Dropout	0.12	0.3
Label smoothing	0.04	0.04
Beam	5	5
Checkpoints averaged	4	4