# Inquisition

## A Reformed™ question-answering system

Andrew Gaut, Shannon Yan, Zeb Mehring

## Background

**Background**: Question-answering is a common NLP task, with applications from web search to virtual assistants and more. Many QA systems are based on the *transformer* architecture, which utilizes a concept called *self-attention* to produce state-of-the-art results on question-answering tasks.

## Problem

**Problem:**

- How can we produce a question-answering system that performs well on SQuAD 2.0?
- Many transformer-based models can be incredibly expensive to train and use. ***How can we make it more efficient?***

**Experiment:**

- Implement QANet from scratch.
- Can we apply the ideas from the **Reformer** architecture to QANet, to improve memory use and compute time without significantly impacting performance?

## Methods

### QANet

- A powerful, transformer-based question-answering system
- Given a "context" paragraph and a question about the paragraph, this model can predict the start and end of the answer to the question contained in the original paragraph

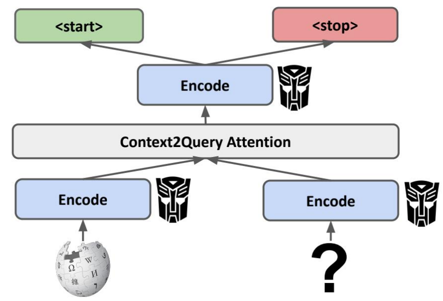



Figure 1. QANet conceptual architecture. The input context and query are encoded separately, then mixed. The resultant representation undergoes further encoding before being used to predict the position of the start and end of the answer (from the context).

***How can we improve efficiency?*** **Key Ideas:**

1. Locality-sensitive hashing self-attention
2. Rev(ersible)Net residual blocks

## Methods (cont.)

### LSH Self-Attention

**Observation:** The $QK^T$ product in self-attention is a *sparse* matrix, made sparser still by the application of the softmax function.
**Idea:** Use locality-sensitive hashing to compute only the nonzero entries.

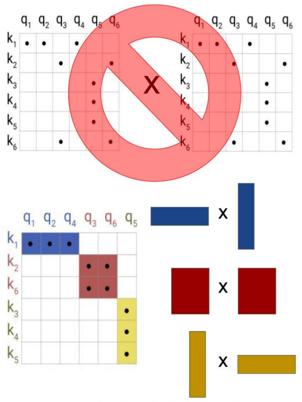


Figure 2. Large matrix multiplication (as is involved in self-attention) is costly in both compute and memory. By leveraging sparsity, we can compute only the products of those regions of the matrix we know to be dense, and save considerable computational burden.

$$\text{Attention} = \text{softmax}\left(QK^{\top}\right)V$$

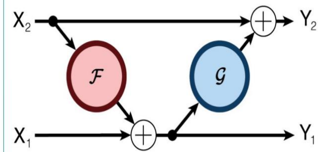
**How?**

1. Recall the equation above.
2. Set $Q = K$ (yes, this really works!).
3. Hash each row of $Q$ using LSH.
4. Group $Q$ by rows with the same hash value.
5. Self-attend (matrix-multiply) *only* within each "bucket".

Instead of an $n \times n$ matrix multiplication, this yields $b$ matrix multiplications each of expected size $n/b \times n/b$, where $b$ is the number of possible hash values.

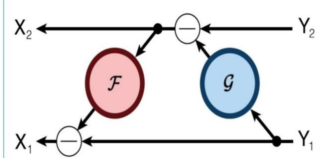
### RevNets

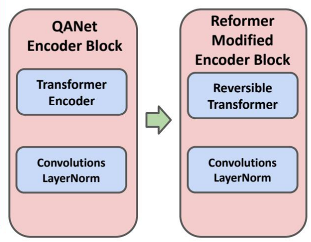
**Observation:** Storing the activations of fully-connected layers for backpropagation requires $O(n^2)$ memory.
**Idea:** Rearrange the equations so that activations can be computed dynamically (rather than stored).




**How?**

1. Let $F$ be the self-attention operation, $G$ be a feed-forward operation.
2. Observe that $X_2 = Y_2 - G(Y_1)$ and $X_1 = Y_1 - F(X_2)$.
3. $X_2$ can be computed as a function of the outputs, and $X_1$ can be computed as a function of $X_2$.
4. The output of the final layer is just the prediction of the block.
5. We can compute activations as we backpropagate!



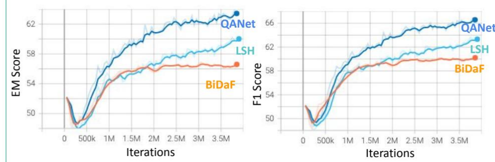



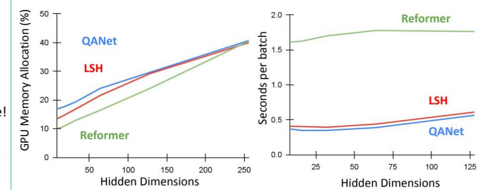

## Experiments and Analysis

**Task:**

- Predict answers to questions from the SQuAD 2.0 dataset.
- Evaluate performance using EM and F1 scores.
- Measure peak memory use and per-epoch.

**Baseline → BiDaF**
**Experiments → QANet**, **LSH** (QANet model with LSH Self Attention), **Reformer-modified** (The QANet model with a RevTransformer in its Encoder Block instead of the Transformer Encoder block)




- QANet vastly outperforms BiDAF as expected (+6.7 EM, +6.32 F1)
- Reformer modifications reduce memory usage as expected (~10% for small dim, decreasing as dimensions increase)
- Dot product self-attention uses *more* memory than LSH, as expected.
- Using LSH instead of dot product self-attention hampers performance significantly (-2.7 EM, -3.53 F1) despite claims to the contrary by the Reformer paper. The Reformer paper's evaluation may have been insufficient. It would be interesting to see a large scale study of these modifications a la ***Do Transformer Modifications Transfer Across Implementations and Applications***.

## Conclusions and Future Work

- Reformer-style modifications to existing transformer-based architectures seem effective at reducing computational burden, though come at a cost to performance
- Future experiments: apply reformer-style modification to popular, expensive language models (GPT-X, BERT, etc.) to make them more accessible