CUDA be good

Conor Williams

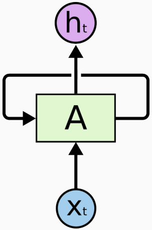
August 23, 2023

Content:

```
for title in ["Hard LSTM", "Multi-blank"]:
```

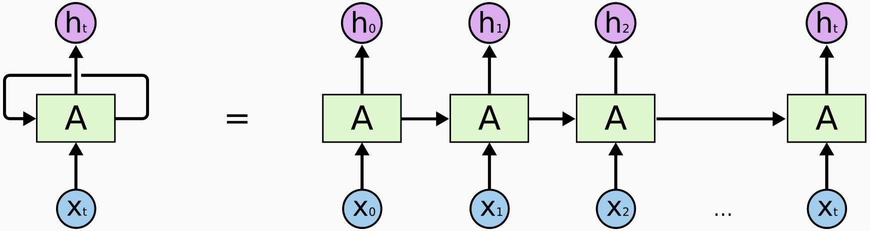
- Background
- Optimization and implementation
- Results
- Further work

LSTM background



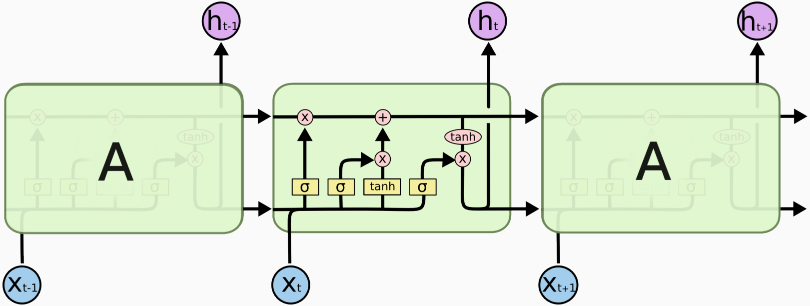
Source: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

LSTM background



Source: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

LSTM background



Source: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

State of affairs:

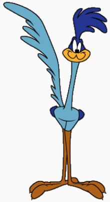
- Pytorch.
- CuDNN.
- Only *soft* activation functions.
- Pure python 3× slower (no autocast).

The challenge:

- Implement a from-scratch LSTM.
- *Correct* forward and backward.
- As fast as CuDNN's hand optimized assembly.

Optimizations:

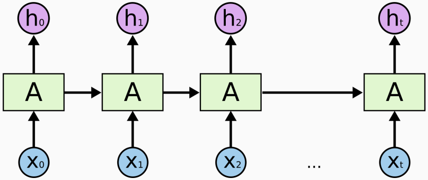
Goal: spend all our time in (big) GEMMs.



Optimizations:

Goal: spend all our time in (big) GEMMs.

- Fuse GEMM across time, let $X_{ab} = x_a^b$:

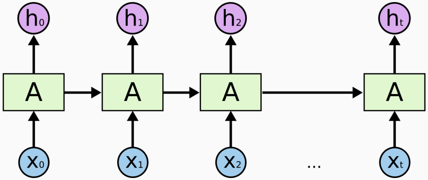


Optimizations:

Goal: spend all our time in (big) GEMMs.

- Fuse GEMM across time, let $X_{ab} = x_a^b$:

$$\{Wx^0, Wx^1, Wx^2, \ldots\} \to W \begin{bmatrix} x^0 & x^1 & \cdots \\ \downarrow & \downarrow & \downarrow \end{bmatrix}$$



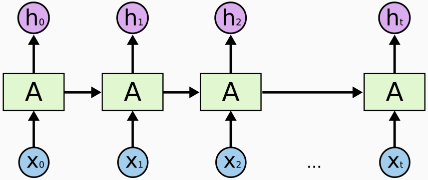
Optimizations:

Goal: spend all our time in (big) GEMMs.

- Fuse GEMM across time, let $X_{ab} = x_a^b$:

$$\{Wx^0, Wx^1, Wx^2, \ldots\} \rightarrow W \begin{bmatrix} x^0 & x^1 & \cdots \\ \downarrow & \downarrow & \downarrow \end{bmatrix}$$

- Recurrent steps $\rightarrow$ CUDA.



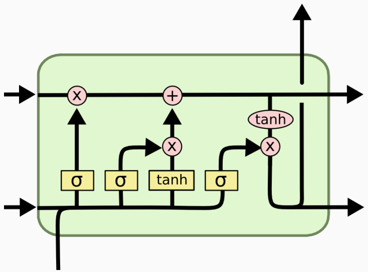
Optimizations:

Goal: spend all our time in (big) GEMMs.

- Fuse GEMM across time, let $X_{ab} = x_a^b$:

$$\{Wx^0, Wx^1, Wx^2, \dots\} \to W \begin{bmatrix} x^0 & x^1 & \cdots \\ \downarrow & \downarrow & \downarrow \end{bmatrix}$$

- Recurrent steps $\longrightarrow$ CUDA.
- Fuse weights and bias.



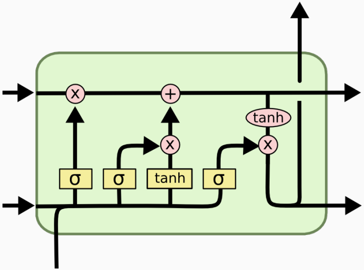
Optimizations:

Goal: spend all our time in (big) GEMMs.

- Fuse GEMM across time, let $X_{ab} = x_a^b$:

$$\{Wx^0, Wx^1, Wx^2, \dots\} \rightarrow W \begin{bmatrix} x^0 & x^1 & \cdots \\ \downarrow & \downarrow & \downarrow \end{bmatrix}$$

- Recurrent steps $\longrightarrow$ CUDA.
- Fuse weights and bias.
- Single activation kernel.



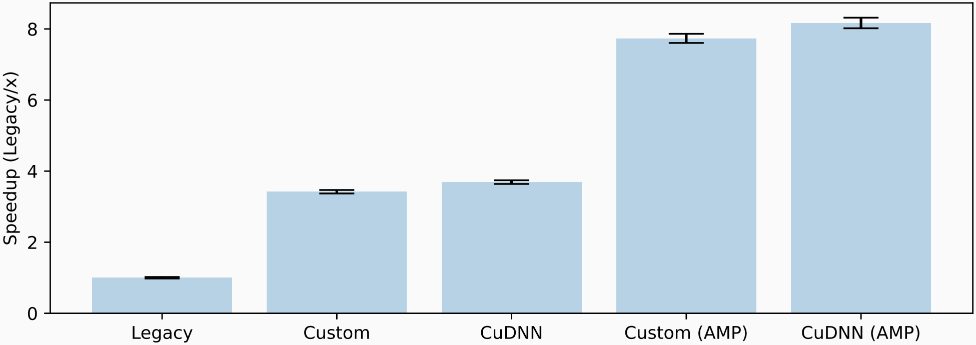
CuBLAS:

```
for t in sequence:
    gates[t] =  y[t] @ R + x[t] @ W + b # Pre-computing

    state = kernel(gates[t]) # CUDA

    y_out[t] = state.out
    y[t + 1] = state.next
```

LSTM results:

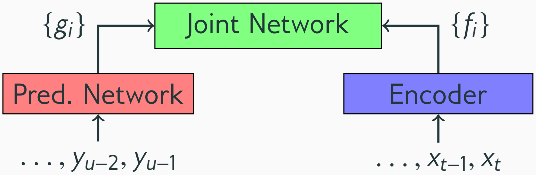


LSTM results:

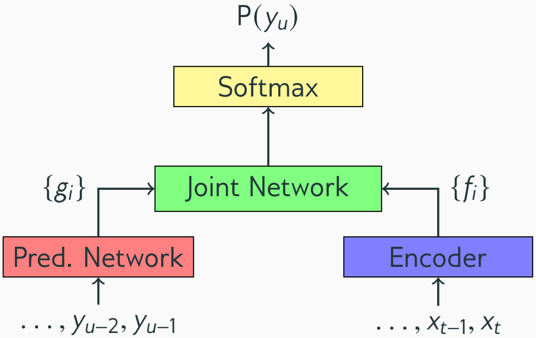
Hard activation functions are now 7.7× faster than the legacy code!

RNN-T inference:

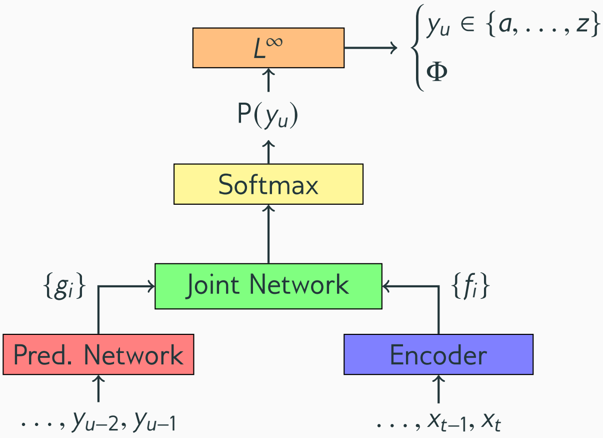
RNN-T inference:



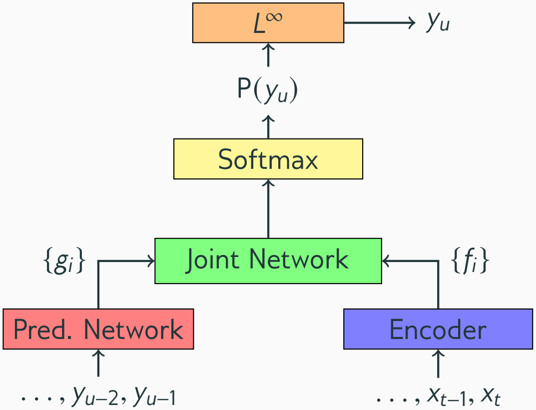
RNN-T inference:



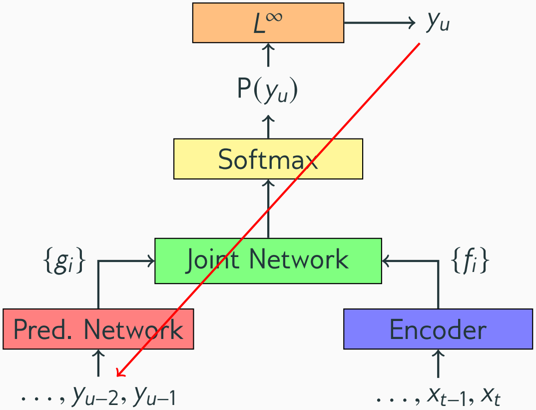
RNN-T inference:



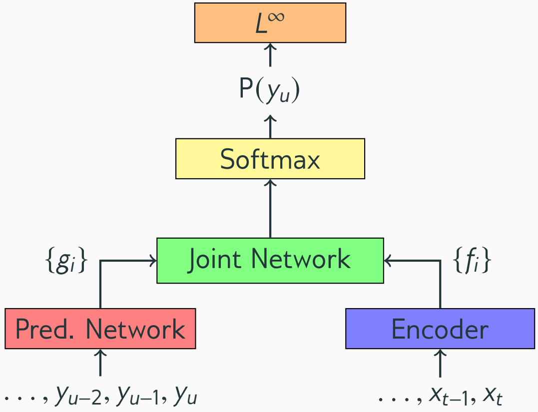
RNN-T inference:



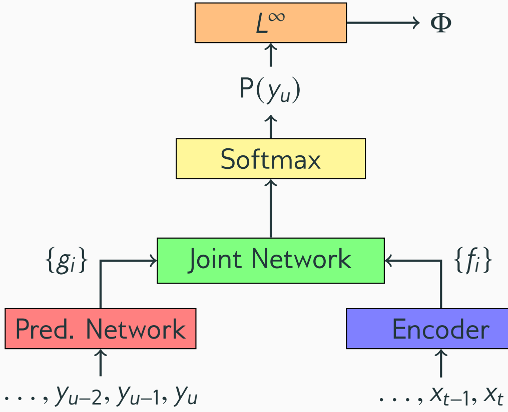
RNN-T inference:



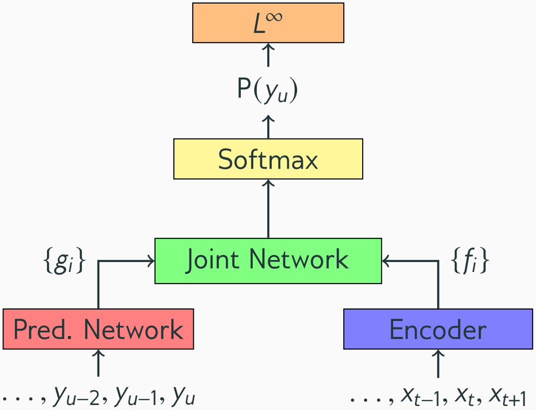
RNN-T inference:



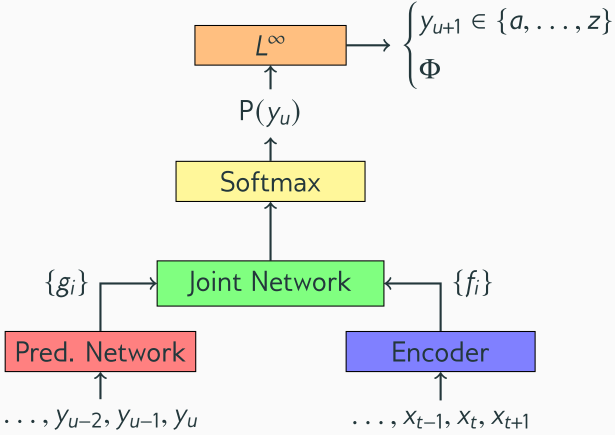
RNN-T inference:



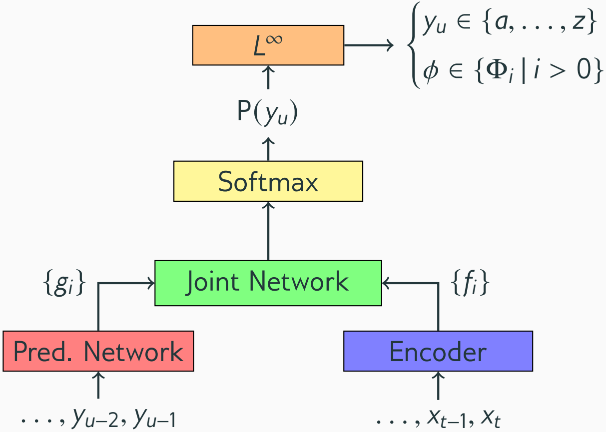
RNN-T inference:



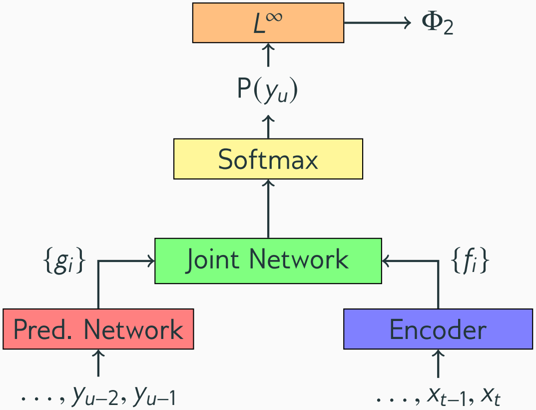
RNN-T inference:



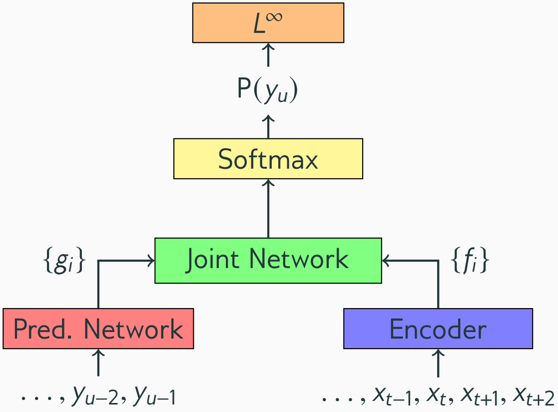
Multi-blank inference:



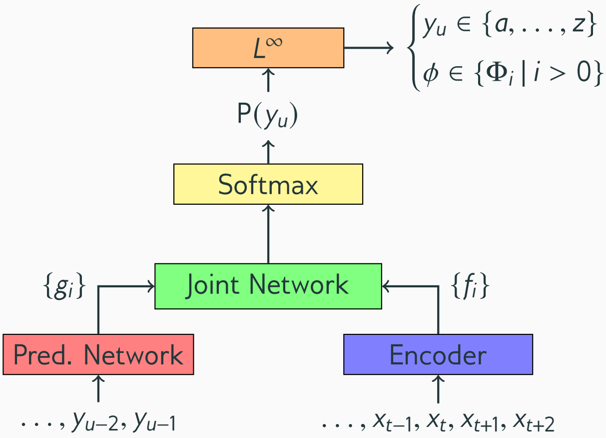
Multi-blank inference:



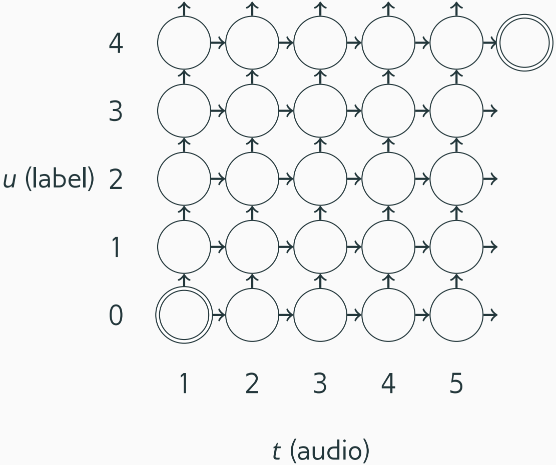
Multi-blank inference:



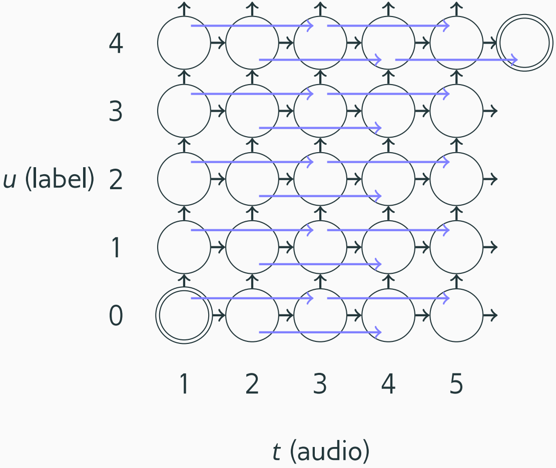
Multi-blank inference:



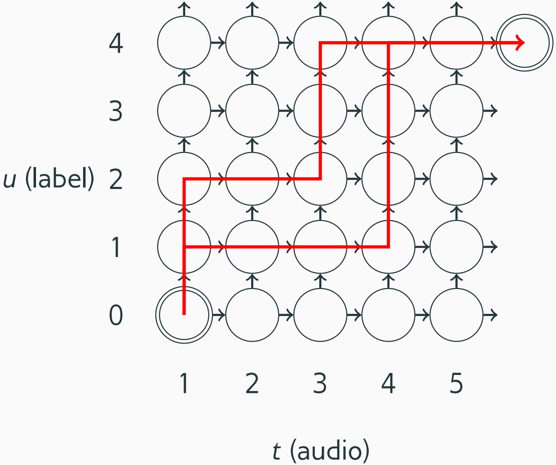
Multi-blank training:



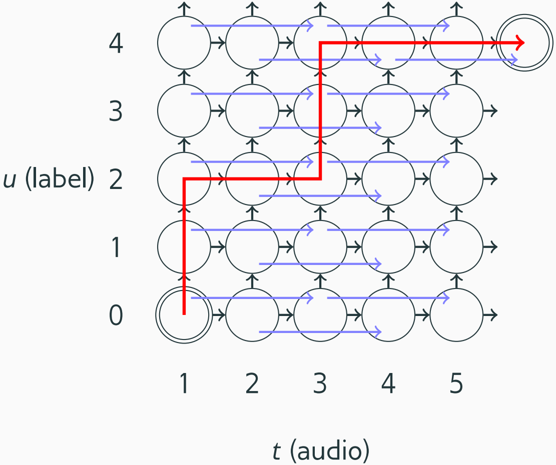
Multi-blank training:



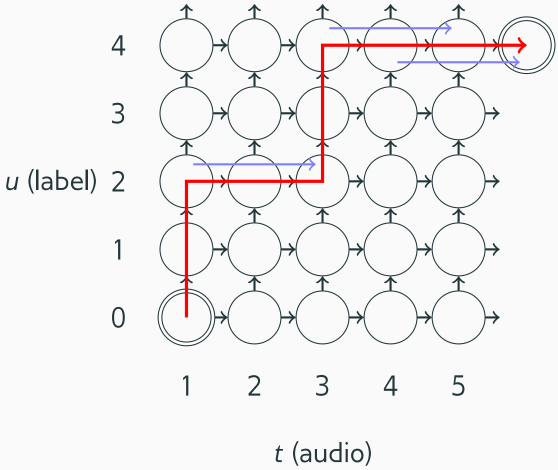
Multi-blank training:



Multi-blank training:



Multi-blank training:



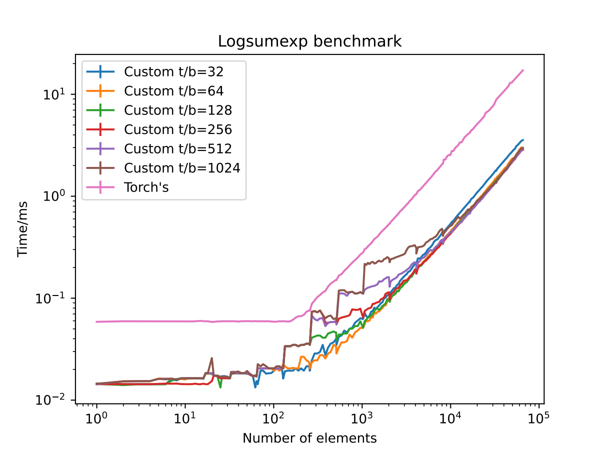
Log softmax:

Let L be the `log_softmax` function $L\colon \mathbb{R}^n \to \mathbb{R}^n$.

$$
\begin{aligned}
L(x)_i &= \log \frac{\exp x_i}{\sum_j \exp x_j} \\
&= x_i - \log \sum_j \exp x_j \\
&= x_i - \alpha - \log \sum_j \exp\left(x_j - \alpha\right) \\
&= x_i - f(x)
\end{aligned}
$$


Always has been
Wait, it's all just map-reduce?

Log sum of exponentials:



Multi-blank results:

- Half memory of the legacy code $\rightarrow$ 85% larger batches.
- 1.2$\times$ faster with no big-blanks.
- Supports multi-blank.
- Unoptimised MB WER same as no MB.
- 1.35$\times$ GPU inference speedup (unoptimized).

Extensions:

- LSTM train with hard activation functions.
- Tune muliblank hyperparameters, e.g. undernomalization, big-blank set.
- Optimize torches map-reduce operations when in-place.
- FastEmit.