6. Modernization Efforts

Cleaning up the code and adding new LSTM technology

Ray Smith, Google Inc.

Code Cleanup

- Conversion to C++ completed
- Thread-compatibility: can run multiple instances in separate threads
- Multi-language (single and mixed) capability:
  copes with jpn, heb, hin, and mixes such as hin+eng
- Removed many hard-coded limits, eg on character set, dawgs
- New beam search
- ResultIterator for accessing the internal data
- More generic classifier API for plug-n-play experiments
- Removed lots of dead code, including IMAGE class.

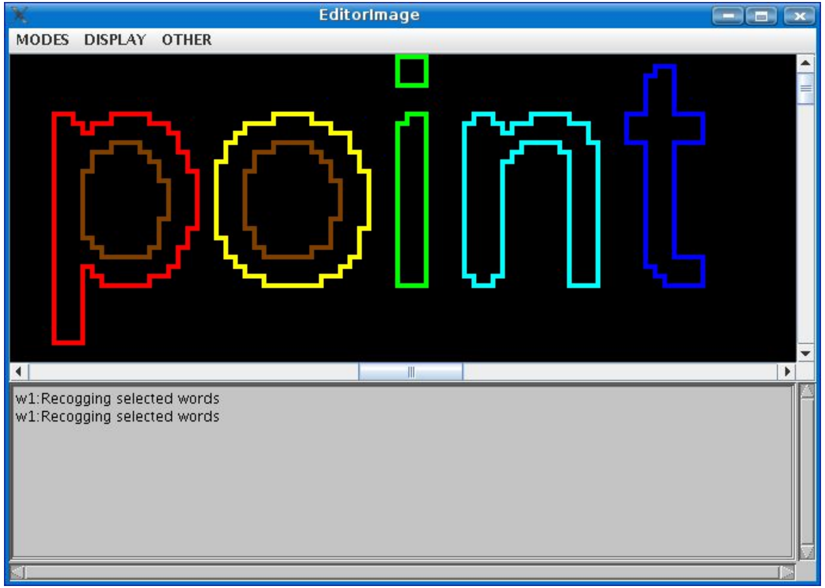
Algorithmic Modernization

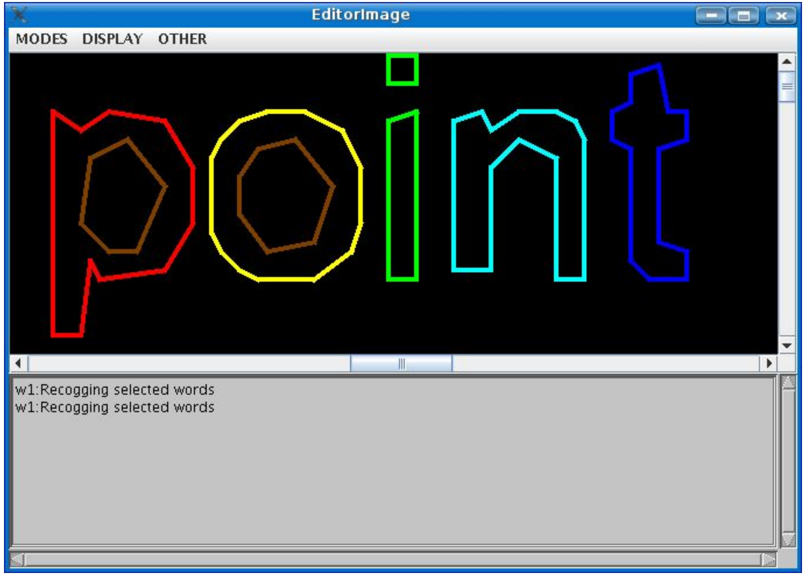
- Cube added Convolutional NN, but improvement was disappointing.
- It would be nice to eliminate the polygonal approximation…
- It would be nice to eliminate the need for accurate baseline normalization

=> New classifier experiments

Eliminate the Polygonal Approximation?

Pixel edge step outlines -> Polygonal approximation





Challenge: Eliminate the Polygonal Approximation!

- Allow feature extraction from greyscale by eliminating the dependence on the polygonal approximation.
- Make it faster and more accurate for CJK, [illegible]ic, and OSD.
- Keep everything else as constant as possible:
  - Same feature definition
  - Same segmentation search/word recognizer
  - Same training data, but add scope for significantly more fonts.

Experiments Failed!

Non-Obvious observation:

Despite being designed over 20 years ago, the current Tesseract classifier is incredibly difficult to beat with so-called modern methods.

(Without changing features or upping the number of training fonts)

Why?

Non-Obvious observation:

Despite being designed over 20 years ago, the current Tesseract classifier is incredibly difficult to beat with so-called modern methods.

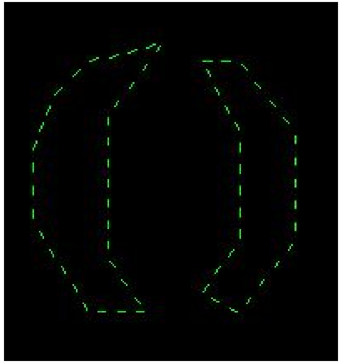
(Without changing features or upping the number of training fonts)

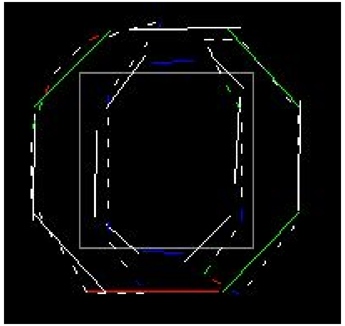
Why?

For what it works with, it is probably as good as you can get.

Statistical Dependence Strikes Again

- Small features to large proto matching accounts well for large-scale features
- Binary quantization of feature space is a loser
- HMMs, DNNs and LSTMs have a better chance than other methods





LSTM Integration

- LSTM code based on OCROpus Python implementation.
- Expanded capabilities including 2-D, variable input sizes.
- Fully integrated with Tesseract at the group-of-similar-words level.
- Visualization with existing Viewer API.
- Training code included (unlike cube).
- Parallelized with openMP.
- Coming in next release of Tesseract.

Tesseract Network Definition Language

- Network defined by a terse string.
- Limited capabilities, but highly flexible within those limits.
- Very easy to use - little to learn.

Legend for following description:

`X`	`Blue`	Literal value.
`n`	`Green`	Variable - substitute a number
(X|Y)	Black	Regular Expression syntax/explanation

Input Layers

`(I|G|N)d,h` Image input in `d` dimensions, of height `h`.

If `d==1`, then the input is `h`-valued, with 1 x-pixel per time-step.

If `d==2`, then the input is 1-valued, with 1 y-pixel per time-step, arranged with width groups of `h` y pixels.

`I` uses a color Image if available, `G` uses Grey, and `N` uses Normalized grey.

Plumbing

`[...]` Execute ... networks in series (layers).

`(...)` Execute ... networks in parallel, with their output dimensions added.

`Rt<net>` Execute `<net>` with time-reversal.

`Ry<net>` Execute `<net>` with y-dimension reversal (only).

`Sx,y` Rescale 2-D input by shrink factor `x,y`, rearranging the data by increasing the depth of the input by factor `xy`.

`Cx,y` Convolves (only - no weights) using a `2x+1, 2y+1` window, with no shrinkage, random infill. Output is `(2x+1)(2y+1)` deeper.

`Px,y` Maxpool the input, reducing each `x,y` rectangle to a single value, independently in each depth dimension.

Functional Units

`Ld,n` d-dimensional (1 or 2) LSTM with n internal states, n outputs.
`L(t|y|ty)d,n` Multi d-dimensional LSTMs with built-in reversal, in time y, or `ty`=both, n states, 2n outputs for `t`,`y` and 4n outputs for `ty`.
`Lt1,n` is short for `(L1,nRtL1,n)`,
`Lty2,n` is very short for `((L2,nRyL2,n)Rt(L2,nRyL2,n))`.
`LQ1,n` 1-dimensional LSTM with n internal states, n outputs that scans each vertical strip independently, sQuashing the y-dimension to 1 output.
`FTn` 1x1 Convolution Tanh with n outputs.
`FSn` 1x1 Convolution clipped Symmetric linear (tanh-like) with n outputs.
`FLn` 1x1 Convolution Logistic with n outputs.
`FPn` 1x1 Convolution clipped Positive linear (logistic-like) with n outputs.
Note: 1x1 Convolution == Fully Connected but shared over time.

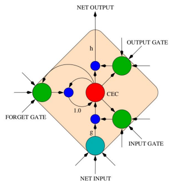
Output Units

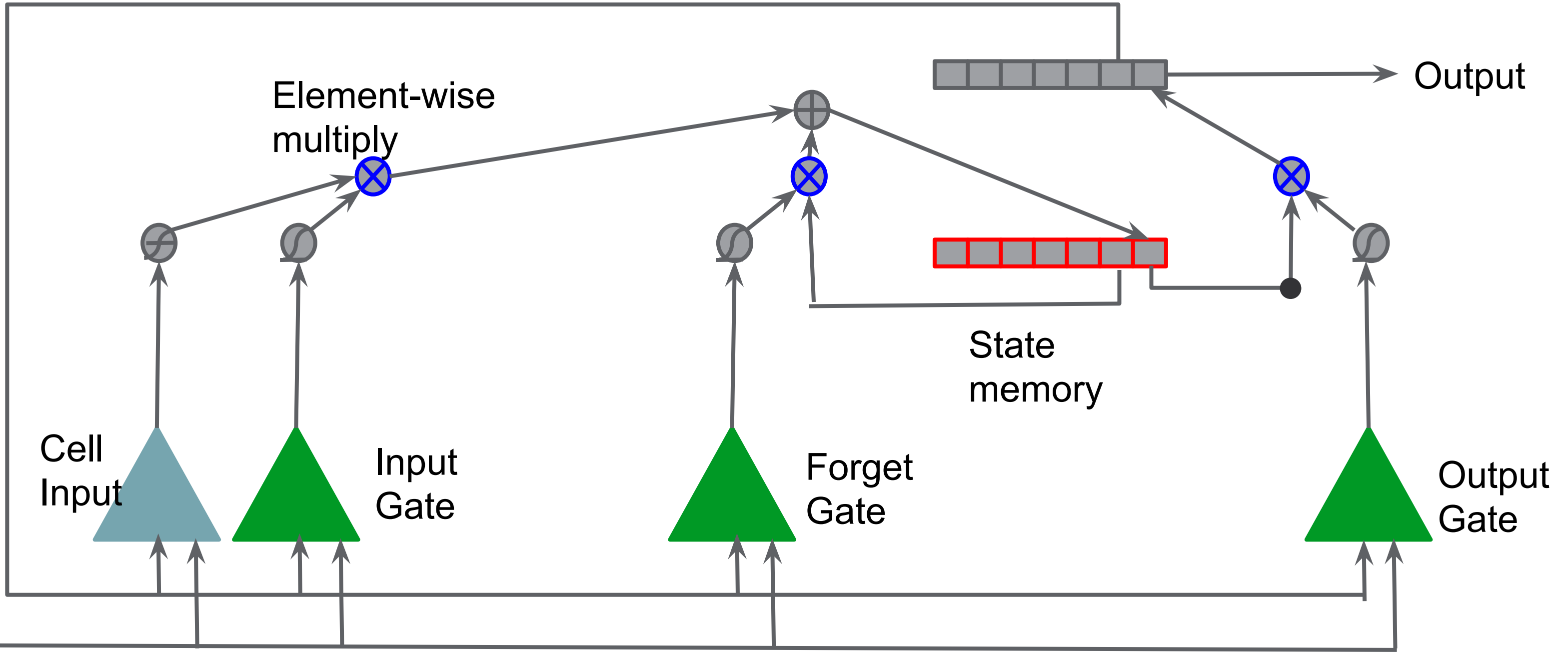
`O` output softmax with number of outputs determined by the unicharset.

`L(S|E)1,n` Single 1-dimensional LSTM with built-in softmax for output, with `n` internal states, number of outputs determined by the unicharset, and extra recurrence from the output of the softmax.
With LS the softmax recurrence is 1-1.
With LE the softmax recurrence is binary Encoded.

Basic LSTM Block (no peep weights)

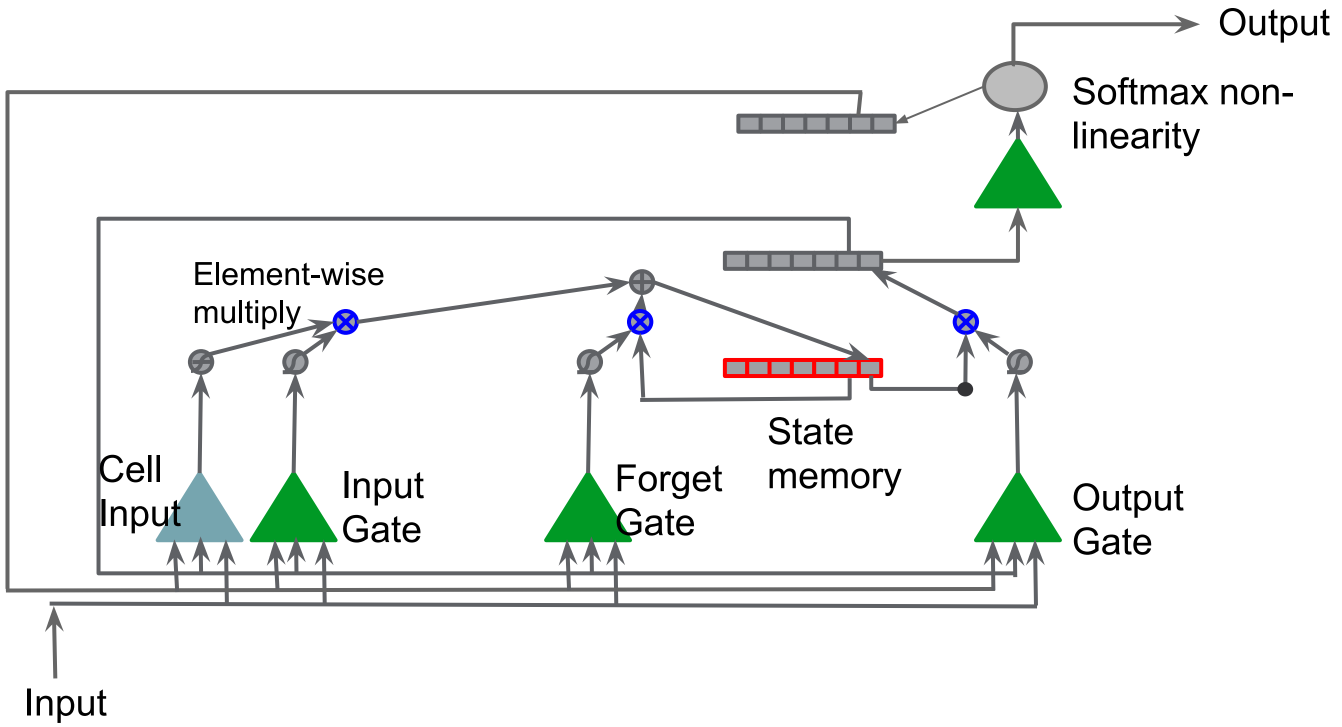
Commonly drawn thus:



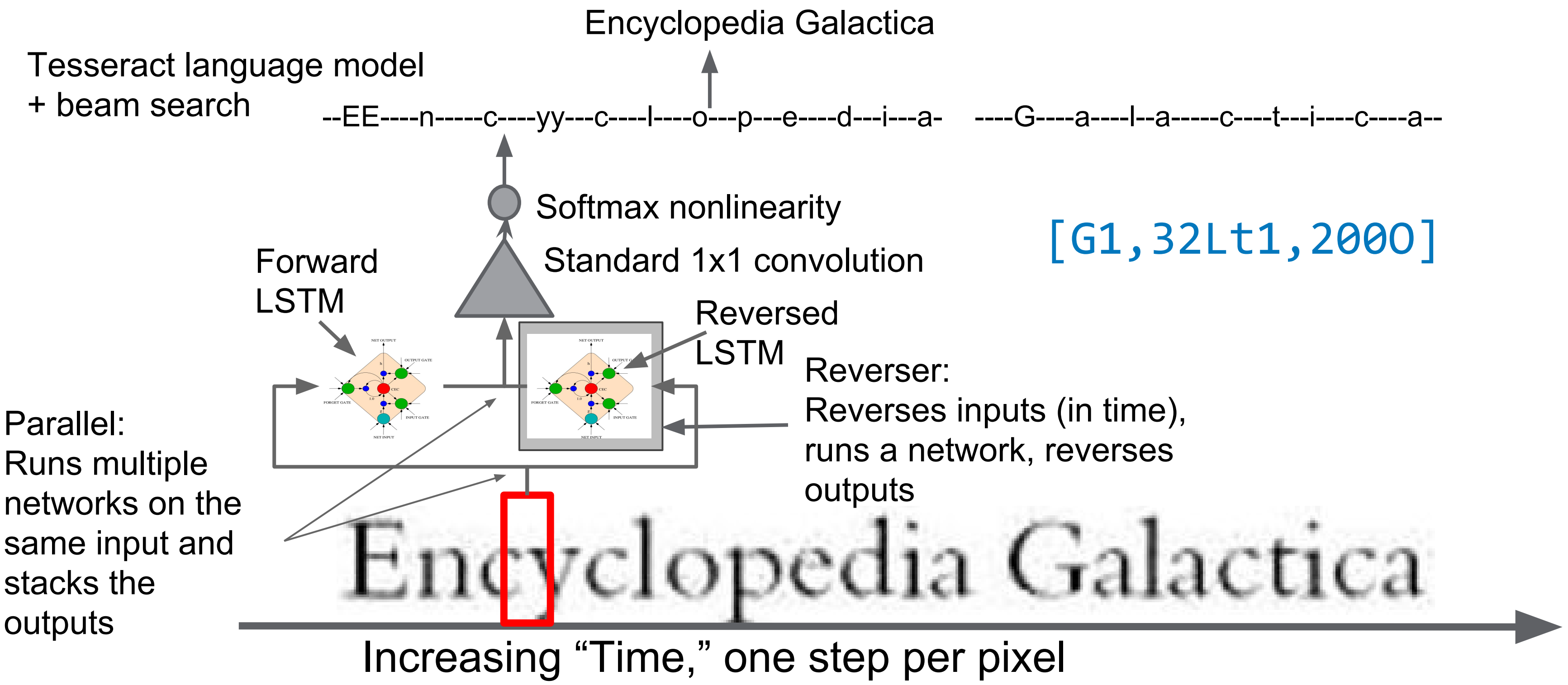


Graphic from: http://googleresearch.blogspot.com/2015/08/the-neural-networks-behind-google-voice.html Credit: Alex Graves

LSTM With Recurrence from Built-in Softmax



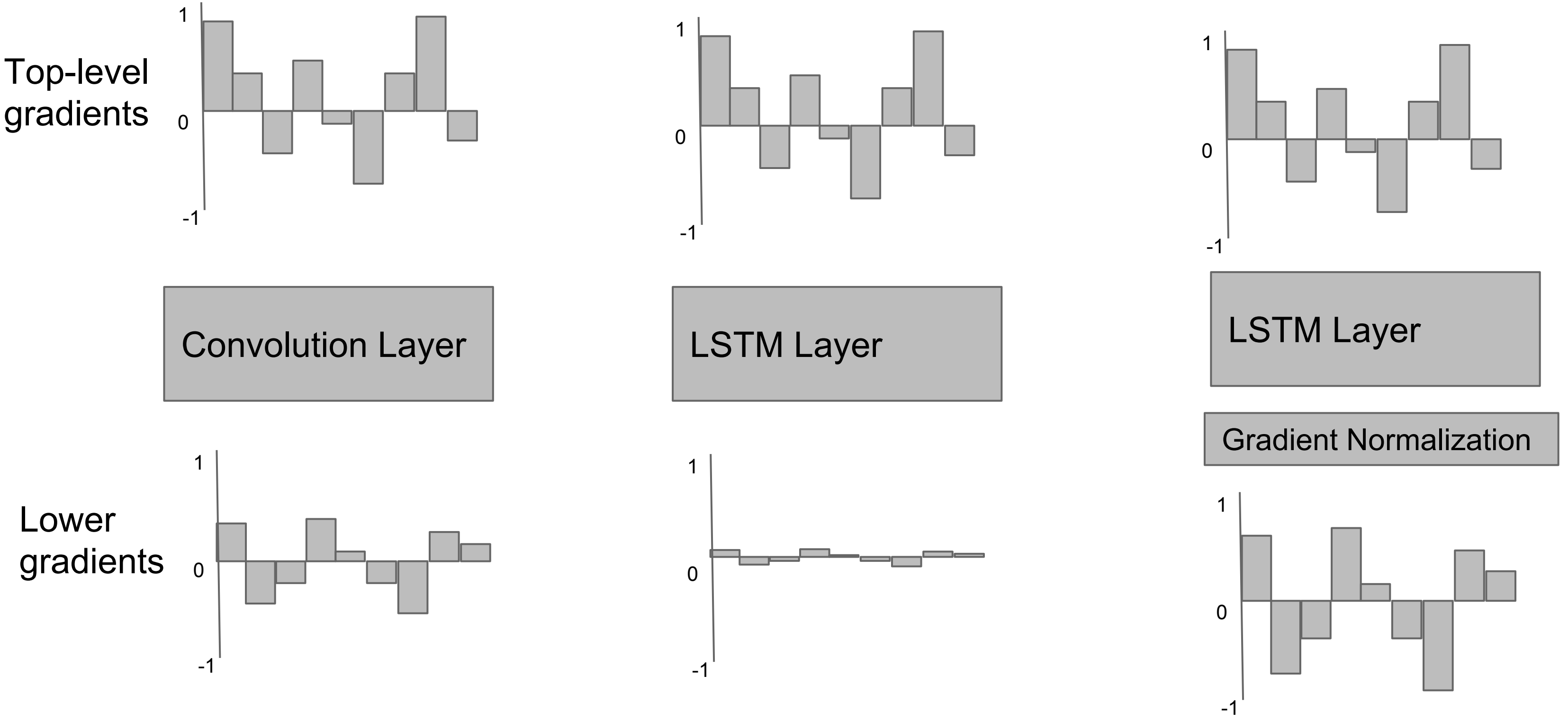
How Tesseract uses LSTMs...



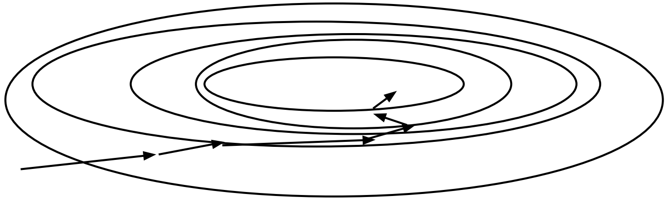
Problems with Baseline Normalization (Still!)

- Simple LSTM is still dependent on good input size/position
- Deep networks can be designed to learn normalization
- Deep LSTM networks train slowly

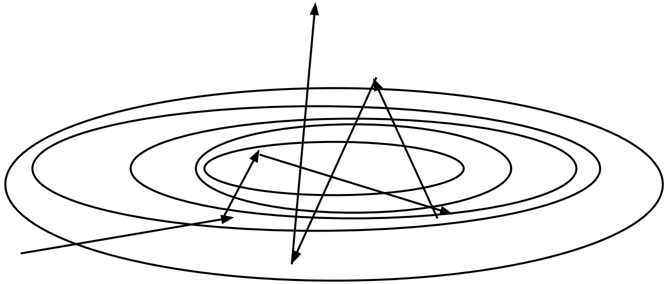
Gradient Normalization



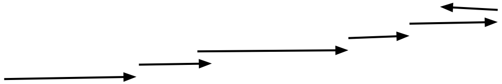
Automatic layer-specific learning rate adjustment



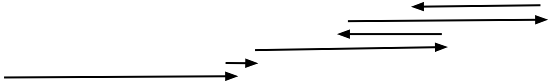
Converging SGD



Diverging SGD



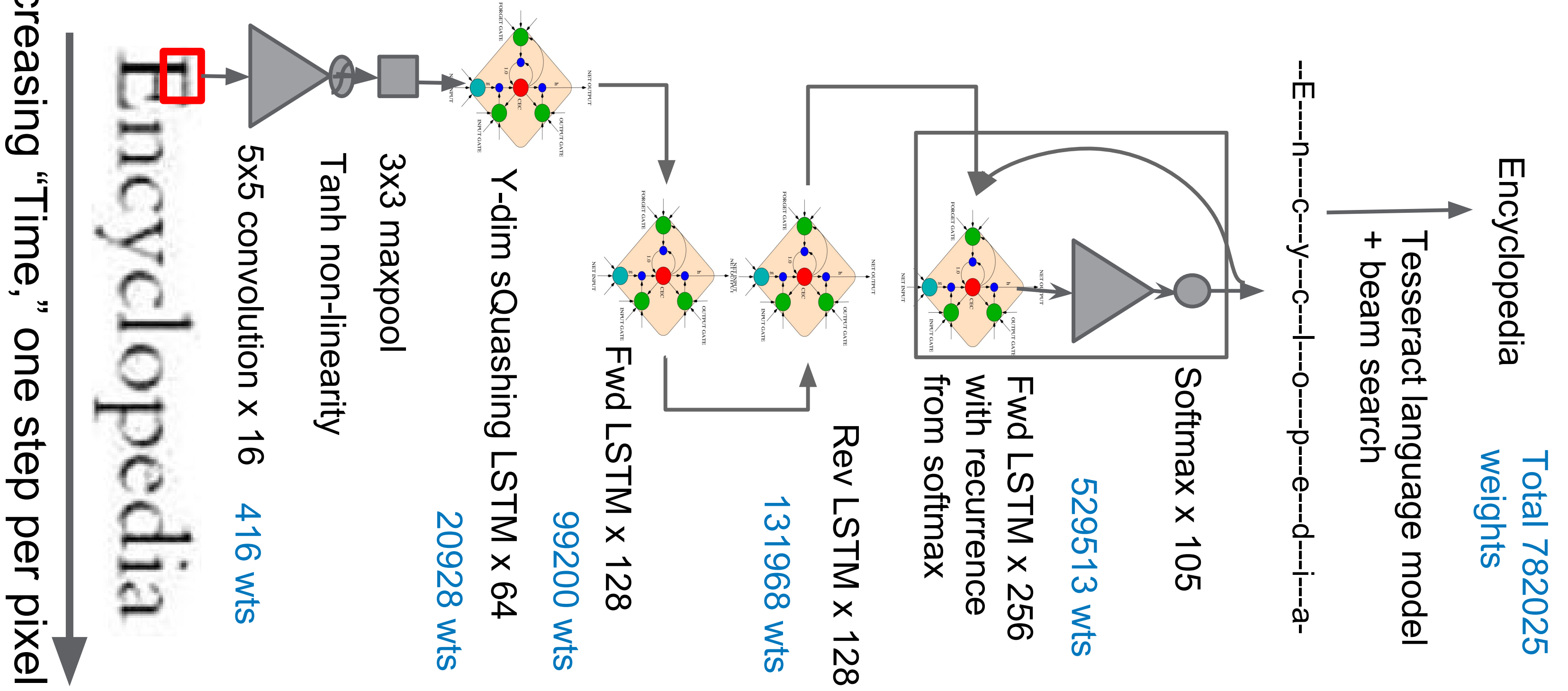
One dimension only,
spread out, mostly
monotonic



One dimension only,
spread out, frequent
sign changes

A More Complex Network Avoids Baseline Normalization

```
[G2,0C2,2FT16P3,3LQ1,64L1,128RtL1,128LS1,256]
```



What about Tensor Flow?

- Tesseract LSTM (T-LSTM) came first, and only supports sequence processing
- Tensor Flow is built for speed, on batches of fixed-size input, but can be run from Tesseract! (Soon!)
- An entire TF graph is treated as a Tesseract Network element.
- TF graph must be designed to support variable width inputs, or work only on fixed-size images.

Thanks for Listening!

Questions?