Lecture Notes for

Machine Learning in Python

Professor Eric Larson

Final Lecture: Case Study in Ethics

Lecture Agenda

- Logistics
  - CNN Grades coming next week (I hope), with penultimate class grades.
  - RNNs due **Last Day of Finals**
- Agenda
  - Ethical Case Study
  - Retrospective and Evaluations

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,
Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
intuition for advanced optimization

Neural Networks and Back Prop.

Numpy
Detailed mathematics for NN operations

Wide and Deep Networks

Convolutional Networks

Recurrent Networks

Keras, Tensorflow
Intuition, Detailed implement.

Ethics in Language Models

ConceptNet
Case studies

Ethics and Bias Case Study in NLP

Janelle Shane @JanelleCShane · 1d

Predictive policing algorithms don't predict who commits crime. They predict who the police will arrest.

Emily M. Bender, professionally... · 11h ···

"AI" can NOT:
* Predict who will commit a crime

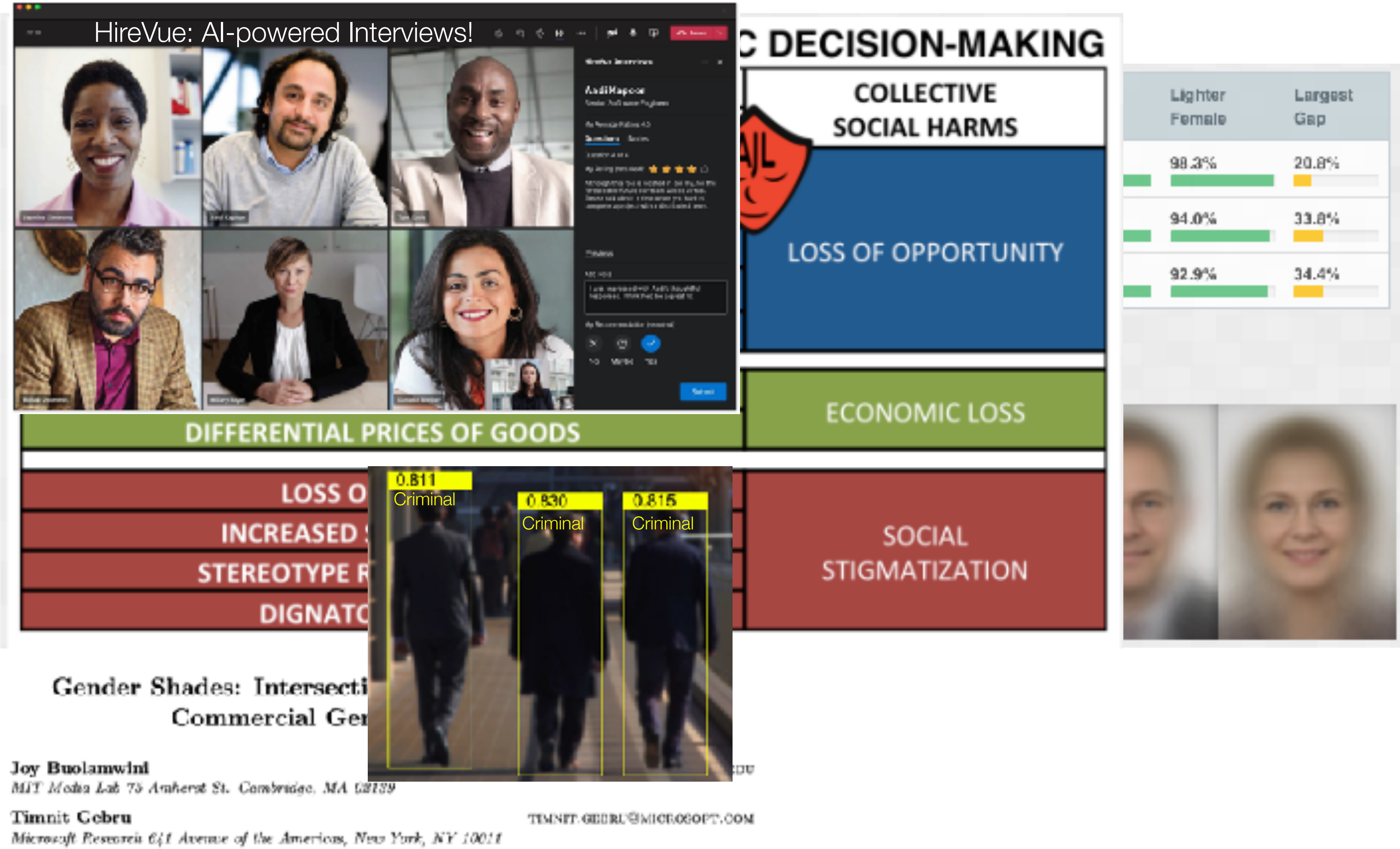
"AI" can:
* Make biased policing look "objective"

Timnit Gebru @timnitGebru

I'm sick of this framing. Tired of it. Many people have tried to explain, many scholars. Listen to us. You can't just reduce harms caused by ML to dataset bias.

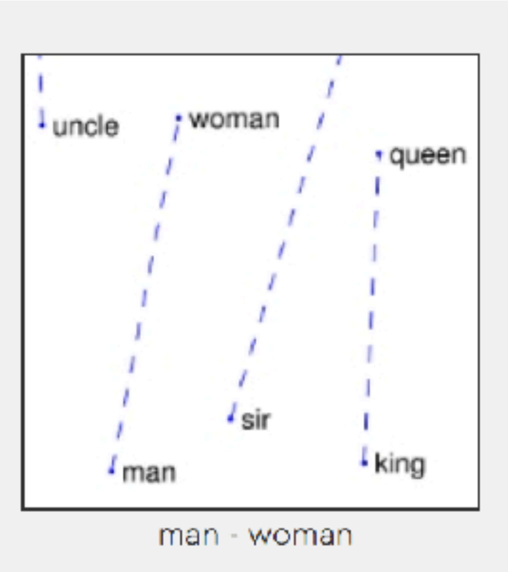
> **Yann LeCun** @ylecun · 19h
> ML systems are biased when data is biased. This face upsampling system makes everyone look white because the network was pretrained on FlickFaceHQ, which mainly contains white people pics....

Timnit Gebru: Gender Shades



http://proceedings.mlr.press/v81/buolamwini18a/buolamwini18a.pdf

Back to RNNs: Word Embedding Analogy



Trained on New York Times

$$W(\text{``woman''}) - W(\text{``man''}) \simeq W(\text{``aunt''}) - W(\text{``uncle''})$$

$$W(\text{``woman''}) - W(\text{``man''}) \simeq W(\text{``queen''}) - W(\text{``king''})$$

$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{computer programmer}} - \overrightarrow{\text{homemaker}}.$$

Extreme *she* occupations

1. homemaker	2. nurse	3. receptionist
4. librarian	5. socialite	6. hairdresser
7. nanny	8. bookkeeper	9. stylist
10. housekeeper	11. interior designer	12. guidance counselor

Extreme *he* occupations

1. maestro	2. skipper	3. protege
4. philosopher	5. captain	6. architect
7. financier	8. warrior	9. broadcaster
10. magician	11. figher pilot	12. boss

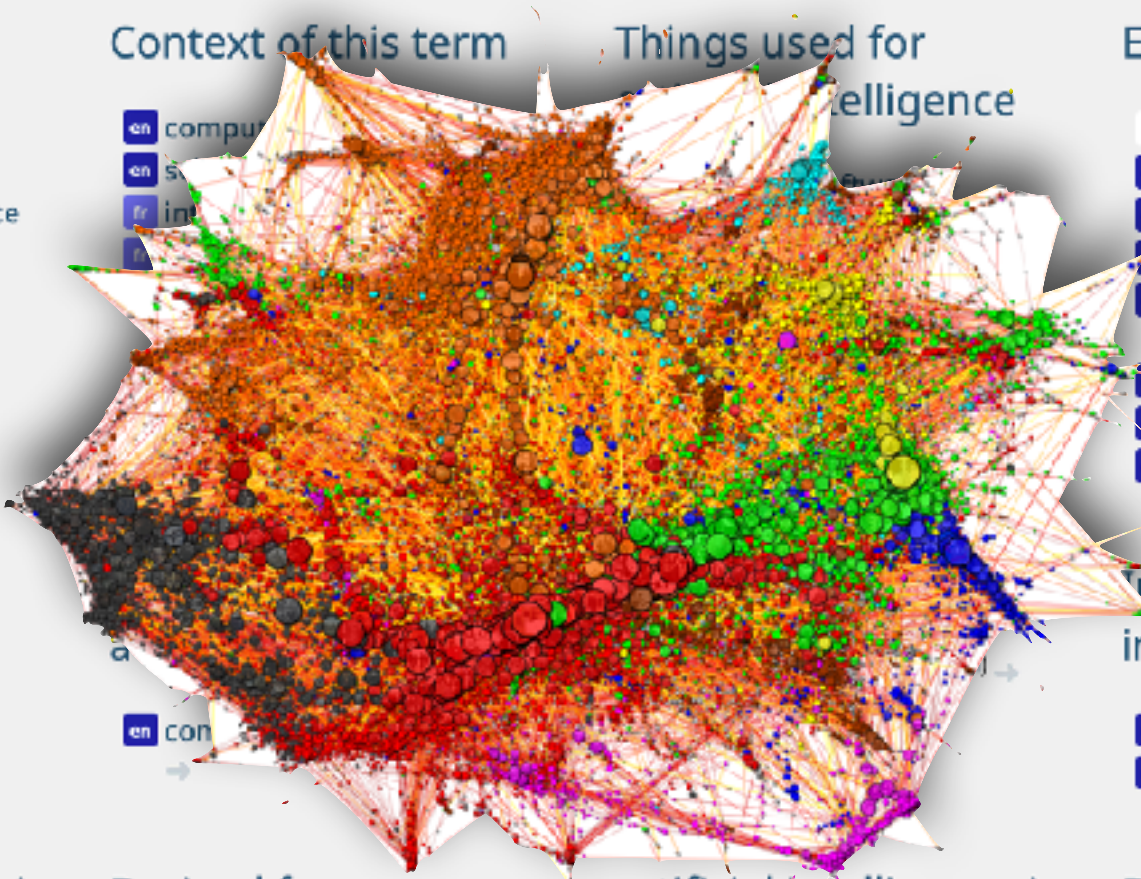
Bolukbasi et al., NeurIPs 2016
https://arxiv.org/pdf/1607.06520.pdf

https://nlp.stanford.edu/projects/glove/

ConceptNet, a Multi-lingual Knowledge Graph

en **artificial intelligence**

An English term in ConceptNet 5.8



Derived terms

- en artificial dumbness →
- en artificial incompetence →
- en artificial lack of intelligence →
- en artificial stupidity →
- en artificial unintelligence →
- en artificially intelligent →

Context of this term

Things used for ...telligence

Etymologically rela...

- bn কৃত্রিম বুদ্ধিমত্তা →
- en artificial dumbness →
- en artificial idiocy →
- en artificial incompetence →
- en artificial lack of intelliger...
- en artificial stupidity →
- en artificial unintelligence →
- en artificially intelligent →

Similar terms

- en expert system →
- en expert systems →

In the field of artifi... intelligence

- en herbert simon →
- en marvin minsky →

artificial intelligence is defined as...

Derived from

- en multi agent system (n) →

artificial intelligence is used for...

People known for artificial intelligenc...

ConceptNet Numberbatch

- **Step One**: Create a Knowledge Graph (from multiple sources with relations like *UsedFor*, *PartOf*, *etc*.)
- **Step Two**: Based on this KG, perturb existing embeddings (like GloVe) to minimize:

$$\Psi(Q) = \sum_{i=1}^{n} \left[\underbrace{\alpha_i \| q_i - \hat{q}_i \|^2}_{\text{(keep similar to original)}} + \underbrace{\sum_{(i,j)\in E} \beta_{ij} \| q_i - q_j \|^2}_{\text{(make similar according to other knowledge)}} \right]$$

new embed

old embed

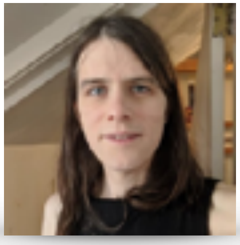
neighbors from KG

- Straight forward to optimize the objective by averaging neighbors in the ConceptNet Knowledge Graph
- Multiple embeddings achieved by merging through “retrofitting” which projects onto a shared matrix space (with SVD)

ConceptNet 5.5: An Open Multilingual Graph of General Knowledge, Speer et al., 2017

Lightning Demo

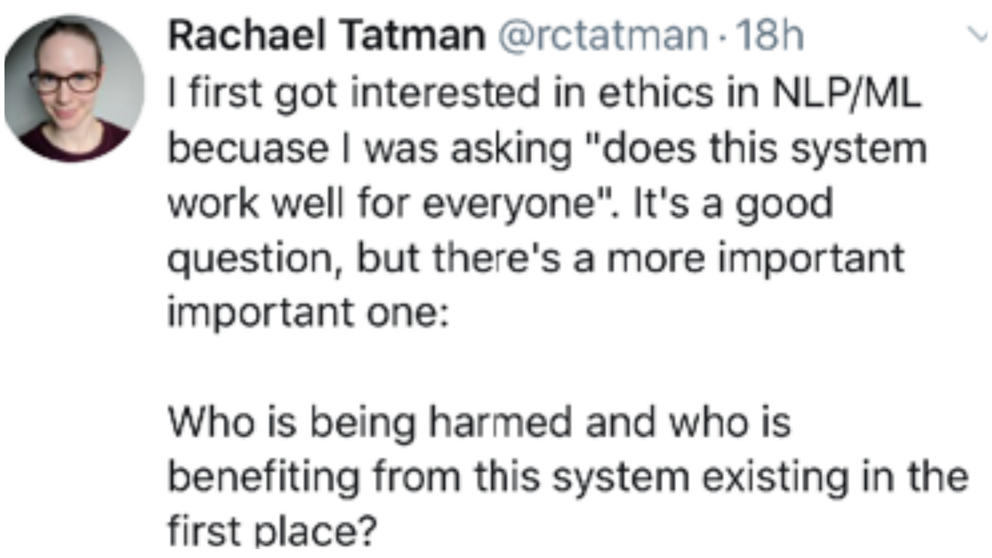
How to Make a Racist AI without Really Trying



Robyn Speer, 2017
http://blog.conceptnet.io/posts/2017/how-to-make-a-racist-ai-without-really-trying/

Debiasing: Man is to Computer Programmer as Woman is to Homemaker? De-biasing Word Embeddings
Bolukbasi et al., NeurIPs 2016
https://arxiv.org/pdf/1607.06520.pdf

ConceptNet 5.5: An Open Multilingual Graph of General Knowledge
Speer et al., AAAI 2017
https://arxiv.org/pdf/1612.03975.pdf



François Chollet @fchollet · 11h

When faced with tech ethics problems, you can either ask hard questions, seek solutions, and take responsibility, or you

Devin Guillory @databoydg · 13h

Watching one of the most influential

Timnit Gebru

A lot of times, people are talking about bias in the sense of equalizing performance across groups. They're not thinking about the underlying foundation, whether a task should exist in the first place, who creates it, who will deploy it on which population, who owns the data, and how is it used?

The root of these problems is not only technological. It's social. Using technology with this underlying social foundation often advances the worst possible things that are happening. In order for technology not to do that, you have to work on the underlying foundation as well. You can't just close your eyes and say: "Oh, whatever, the foundation, I'm a scientist. All I'm going to do is math."

Course Retrospective

- AI winters exi
  machine lear
  repeat)
- Formal meth
- At the end of
- **Open sourc**
  advancemen
  - http://ww

Leading ML researchers issue statement of support for JMLR

From: Michael Jordan [mailto:jordan@CS.Berkeley.EDU]
Sent: Monday, October 08, 2001 5:33 PM
Subject: letter of resignation from Machine Learning journal

Dear colleagues in machine learning,

The forty people whose names appear below have resigned from the Editorial Board of the Machine Learning Journal (MLJ). We would like to make our resignations public, to explain the rationale for our action, and to indicate some of the implications that we see for members of the machine learning community worldwide.

The machine learning community has come of age during a period of enormous change in the way that research publications are circulated. Fifteen years ago research papers did not circulate easily, and as with other research communities we were fortunate that a viable commercial publishing model was in place so that the fledgling MLJ could begin to circulate. The needs of the community, principally those of seeing our published papers circulate as widely and rapidly as possible, and the business model of commercial publishers were in harmony.

Times have changed. Articles now circulate easily via the Internet, but unfortunately MLJ publications are under restricted access. Universities and research centers can pay a yearly fee of $1050 US to obtain unrestricted access to MLJ articles (and individuals can pay $120 US). While these fees provide access for institutions and individuals who can afford them, we feel that they also have the effect of limiting contact between the current machine learning community and the potentially much larger community of researchers worldwide whose participation in our field should be the fruit of the modern Internet.

None of the revenue stream from the journal makes its way back to authors, and in this context authors should expect a particularly favorable return on their intellectual contribution---they should expect a service that maximizes the distribution of their work. We see little benefit accruing to our community from a mechanism that ensures revenue for a third party by restricting the communication channel between authors and readers.

Sincerely yours,

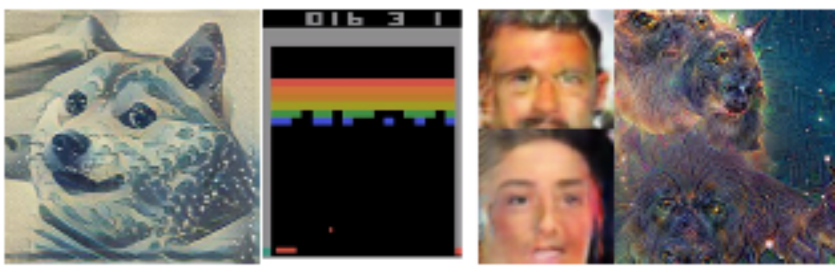
Chris Atkeson
Peter Bartlett
Andrew Barto
Jonathan Baxter
Yoshua Bengio
Kristin Bennett
Chris Bishop
Justin Boyan
Carla Brodley
Claire Cardie
William Cohen
Peter Dayan
Tom Dietterich
Jerome Friedman
Nir Friedman
Zoubin Ghahramani
David Heckerman
Geoffrey Hinton
Haym Hirsh
Tommi Jaakkola
Michael Jordan
Leslie Kaelbling
Daphne Koller
John Lafferty
Sridhar Mahadevan
Marina Meila
Andrew McCallum
Tom Mitchell
Stuart Russell
Lawrence Saul
Bernhard Schoelkopf
John Shawe-Taylor
Yoram Singer
Satinder Singh
Padhraic Smyth
Richard Sutton
Sebastian Thrun
Manfred Warmuth
Chris Williams
Robert Williamson

Topics review

- Data **munging** in pandas and numpy
- Data **visualization** in jupyter with matplotlib, pandas, seaborn, and plotly
- Data preprocessing: **dim reduction**, images, text, categorical features, **embeddings**
- **Linear models**: linear regression, logistic regression, simple neural networks
- **Optimization** strategies: Gradient ascent, Quasi-Newton, Extensions of SGD (RMSProps, AdaM)
- **Back propagation** in MLP (from scratch)
- Tensorflow/Keras for **wide and deep networks**
- **Convolutional** neural networks (up to modern day)
- **Recurrent** neural networks (scratched surface only)

Topics Not Covered

- Transfer/Multi-Task Learning
- Visualizing Deep Convolutional Networks
- Fully Convolutional Networks
- Style Transfer
- Generative Adversarial Networks
- (*partial*) Reinforcement Learning



Syllabus for CSE8321: Machine Learning and Neural Networks

Syllabus for CSE8321: Machine Learning and Neural Networks

Overview

This course extends basic knowledge of the use of Neural Networks in machine learning beyond simple prediction, especially targeted outputs that are generation or alteration of images, text, and audio. This course emphasizes topics of neural networks in the "deep learning" subdomain. This course will survey of important topics and current areas of research, including transfer learning, multi-task and multi modal learning, image style transfer, neural network visualization, deep convolutional generative adversarial networks, and deep reinforcement learning. For grading, students are expected to complete smaller team-based projects throughout the semester, present one research paper in a 15-20 minute group presentation (covering topics in the course), and complete a comprehensive final project that involves a number of different deep learning architectures.

Thank you for a great semester!

- but it could **have been better** somehow, right?
  - how could you learn better, more reliably for an interview?
  - what should **not be cut** or **not changed**?
  - **Already cut**: SVMs, Ensembles, Transformers, many-to-many RNNs,
  - More RNNs? Less RNNs? No RNNs?
  - More convolutional approaches/depth?
  - More APIs? Turi / PyTorch?
  - More flipped Assignments?
  - Self-guided Jupyter notebooks?

Thank You for an Excellent Semester!

Courtesy of Omar Roa

Please fill out the course evaluations!!!!

Backup slides

Lecture Notes for

Machine Learning in Python

Professor Eric Larson

Seq-2-Seq and Transformers

Archived

Lecture Agenda

- Logistics
  - RNNs due **During Finals Time**
- Agenda
  - Sequence to sequence
  - Transformers

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,
Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
intuition for advanced optimization

Neural Networks and Back Prop.

Numpy
Detailed mathematics for NN operations

Wide and Deep Networks

Convolutional Networks

Recurrent Networks

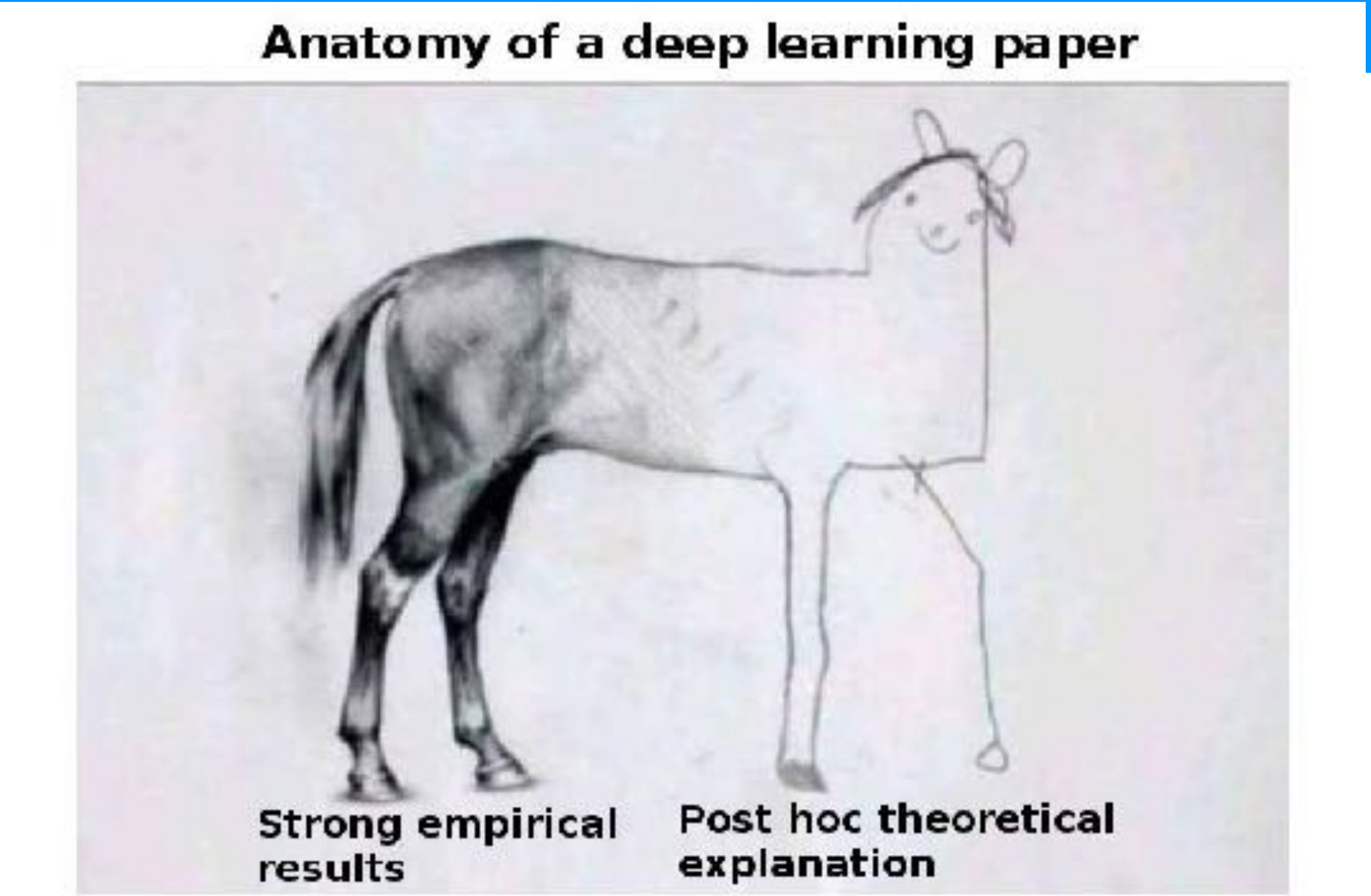
Keras, Tensorflow
Intuition, Detailed implement.

Ethics in Language Models

ConceptNet
Case studies

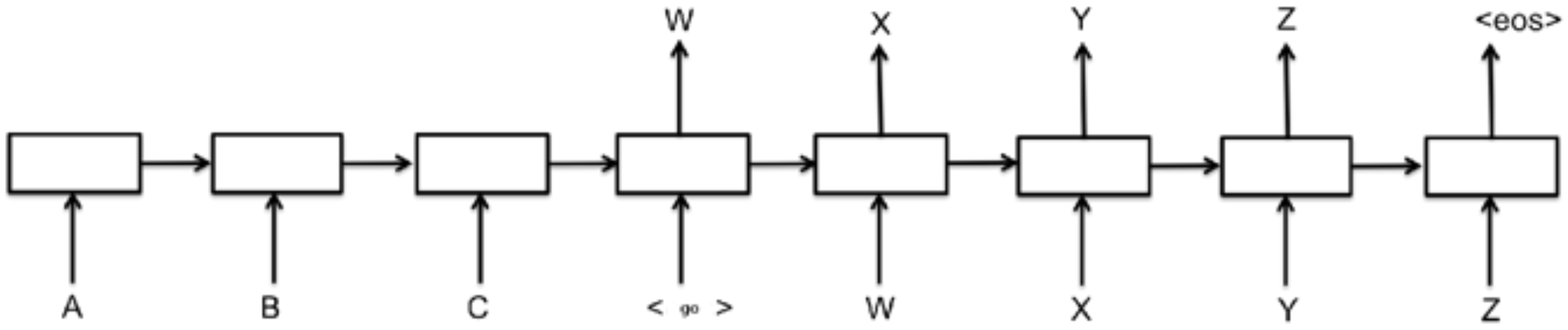
Last Time

Sequence to Sequence



Modeling Sequence to Sequence

Need to translate outputs of unknown size.



- Additional Vocabulary Special Casing:
  - <UNKNOWN>, for unknown input or characters not included in vocabulary
  - <EOS>, end of sentence
  - <GO>, start output sequence
  - <DONTCARE>, outputs before <GO> command

Sutskever et al. Sequence to Sequence Learning with Neural Networks, arXiv. 2014
https://arxiv.org/pdf/1409.3215.pdf

Modeling Sequence to Sequence

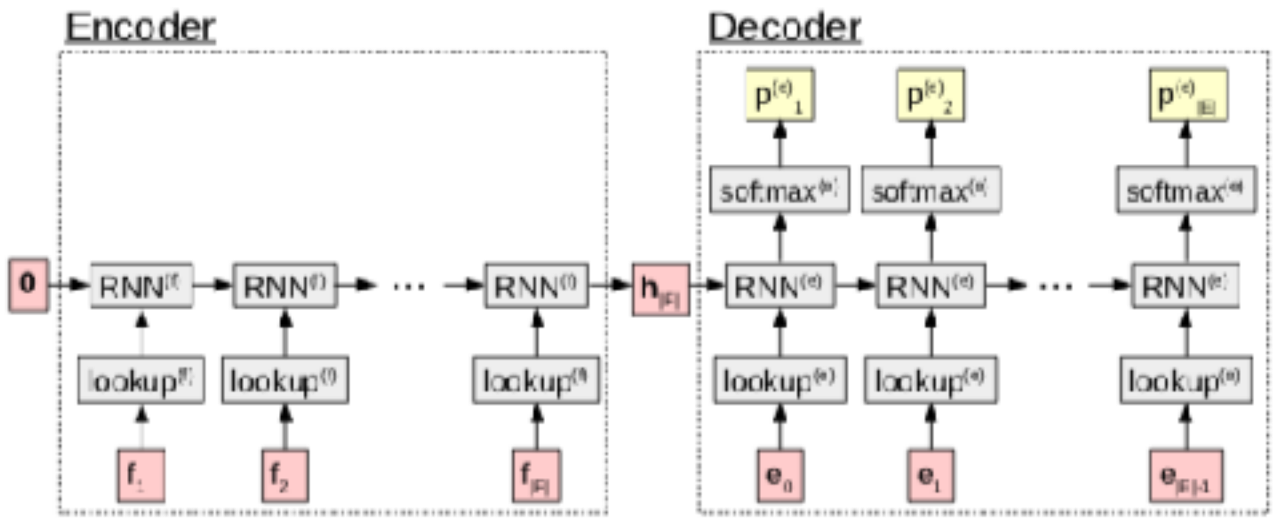


Figure 21: A computation graph of the encoder-decoder model.

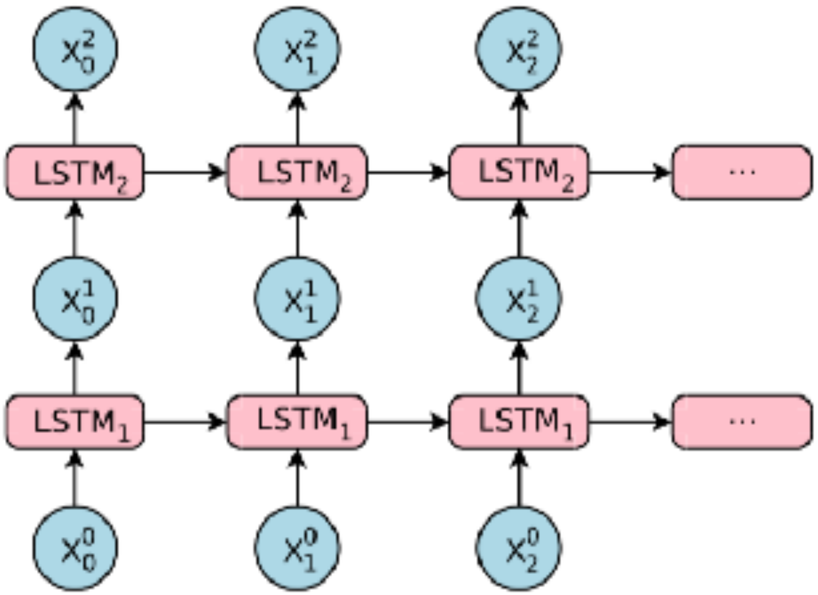
- **Training Process**: Give actual decoded letters for predicting next token
- **Decoding Process** can alter reliability of results:
  - Greedy Search, always choose most likely "next" symbol, seed
  - Keep list of "best" predictions for seeding (i.e., Beam Search)

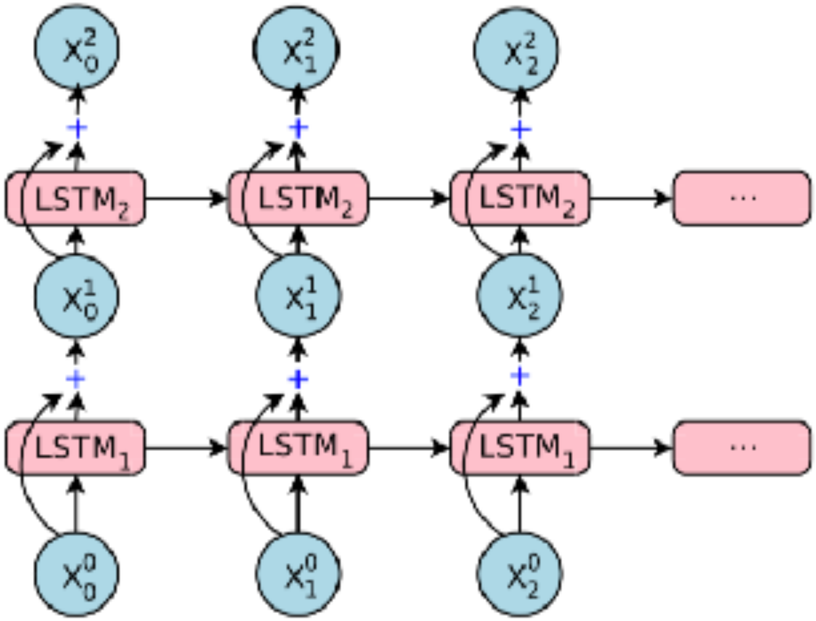
Graham Neubig. 2017
Neural Machine Translation and
Sequence-to-sequence Models: A Tutorial
https://arxiv.org/pdf/1703.01619.pdf

https://github.com/m2dsupsdlclass/lectures-labs/blob/master/labs/07_seq2seq/Translation_of_Numeric_Phrases_with_Seq2Seq_rendered.ipynb

GNMT: Residuals

- Google, 2016

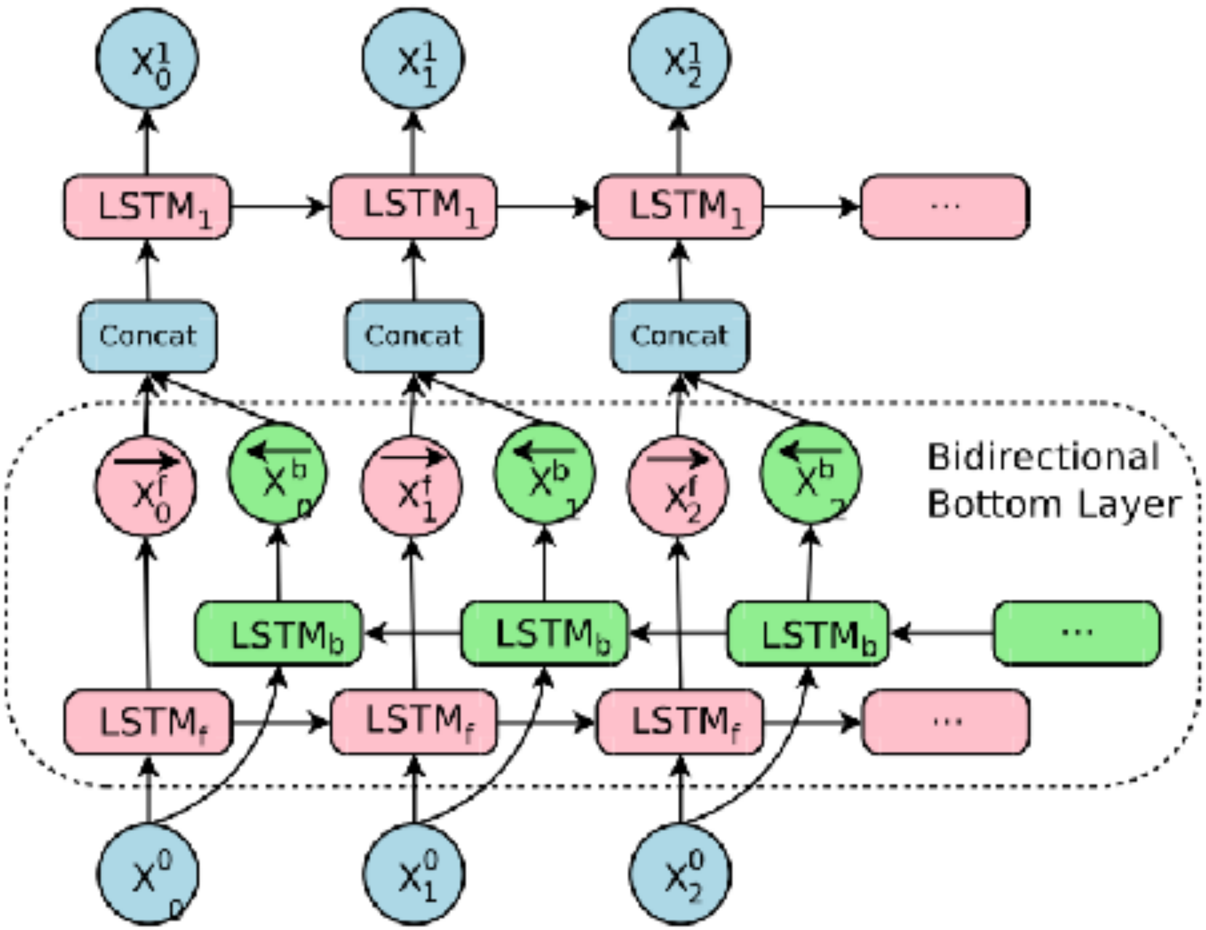




Google's Neural Machine Translation:
https://arxiv.org/pdf/1609.08144.pdf

GNMT: Bidirectionality

- Google, 2016



Google Neural Machine Translation:
https://arxiv.org/pdf/1609.08144.pdf

GNMT: Attention

- Google, 2016

$$s_t = AttentionFunction(\mathbf{y}_{i-1}, \mathbf{x}_t) \quad \forall t, \quad 1 \le t \le M$$

$$p_t = \exp(s_t) / \sum_{t=1}^{M} \exp(s_t) \qquad \forall t, \quad 1 \le t \le M$$

$$\mathbf{a}_i = \sum_{t=1}^{M} p_t . \mathbf{x}_t$$

where $\mathbf{x}_t$ is state of the t^{th} encoder
$\mathbf{y}_{i\text{-}1}$ is the state of the previous decoder
and $\mathbf{a}_i$ is the input for the i^{th} decoder

Google Neural Machine Translation:
https://arxiv.org/pdf/1609.08144.pdf

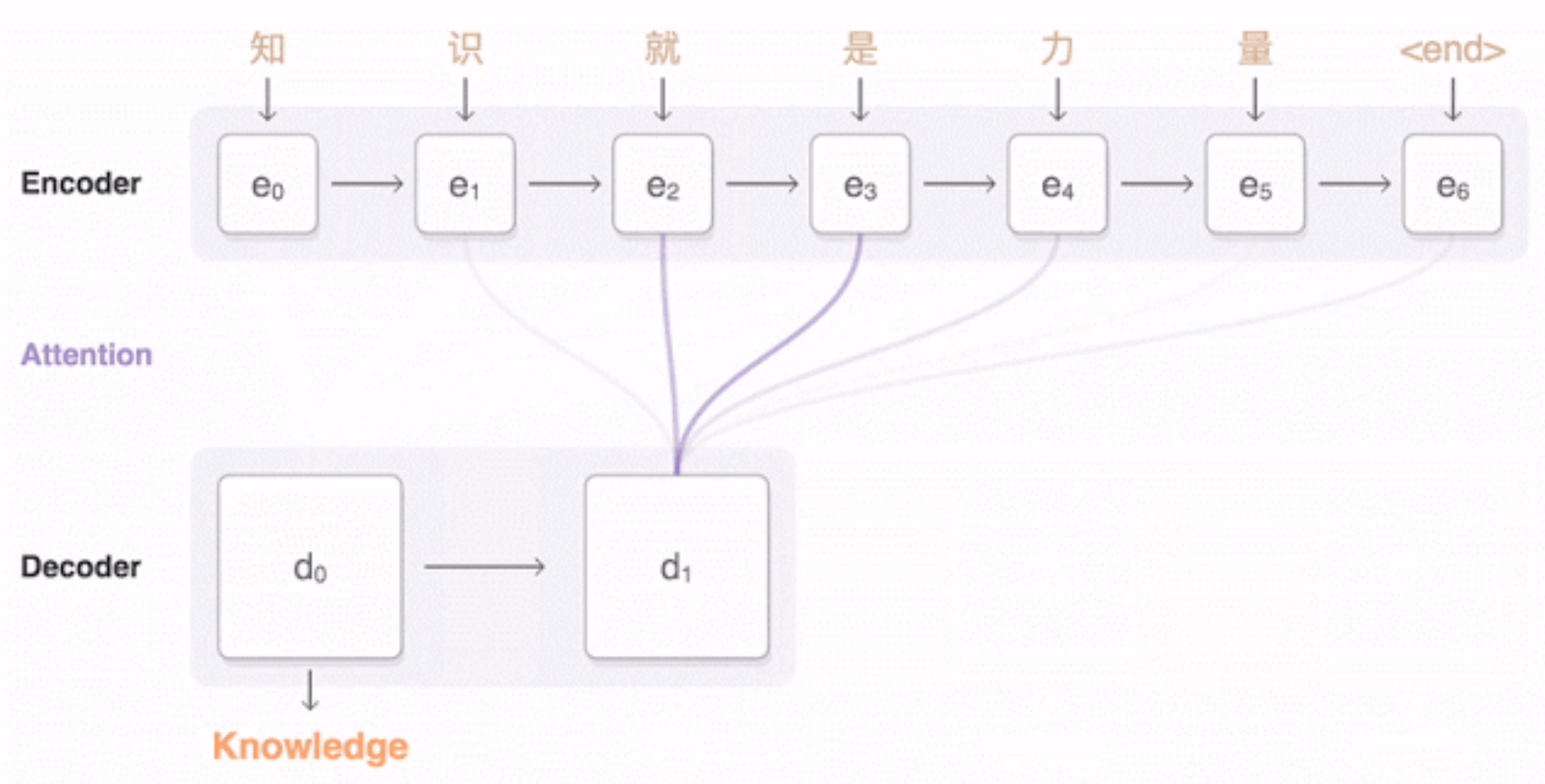
GNMT: Attention

$$s_t = AttentionFunction(\mathbf{y}_{i-1}, \mathbf{x}_t) \quad \forall t, \quad 1 \le t \le M$$

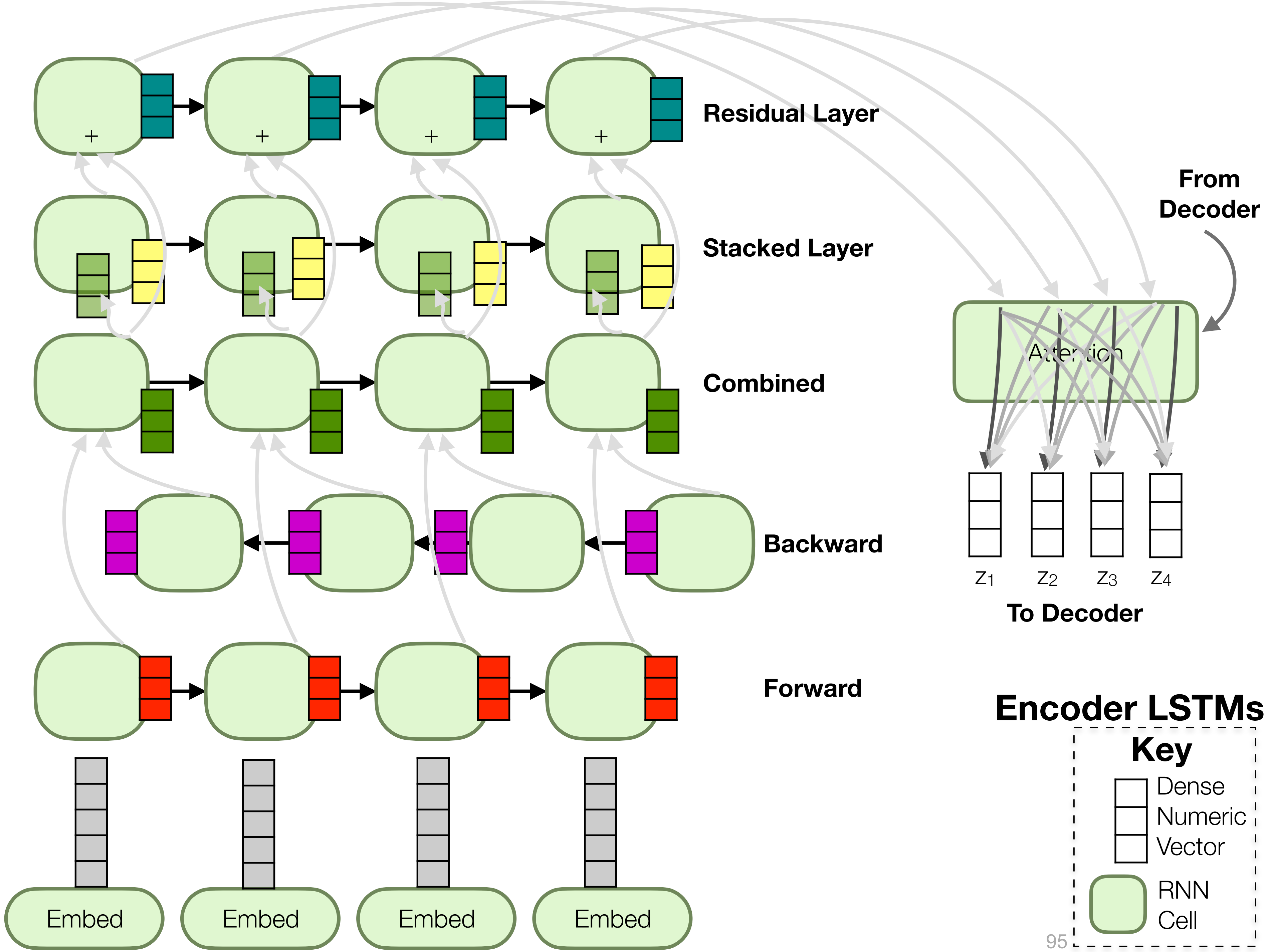
$$p_t = \exp(s_t) / \sum_{t=1}^{M} \exp(s_t) \quad \forall t, \quad 1 \le t \le M$$

$$\mathbf{a}_i = \sum_{t=1}^{M} p_t . \mathbf{x}_t$$

- Google, 2016

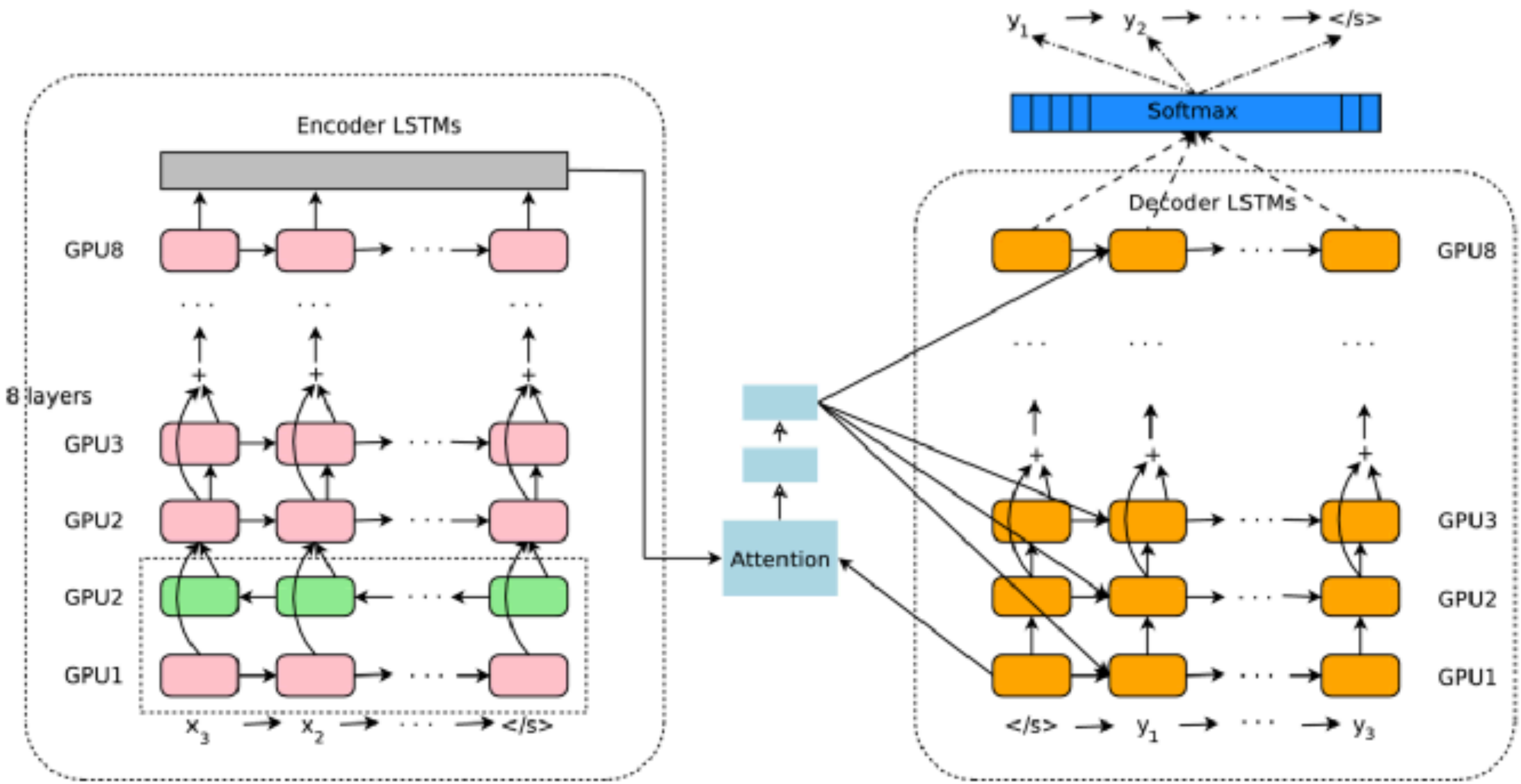


Google Neural Machine Translation:
https://arxiv.org/pdf/1609.08144.pdf
https://medium.com/@Synced/history-and-frontier-of-the-neural-machine-translation-dc981d25422d


Residual Layer
Stacked Layer
Combined
Backward
Forward
Embed
Embed
Embed
Embed
From Decoder
Attention
Z1
Z2
Z3
Z4
To Decoder
Encoder LSTMs
Key
Dense Numeric Vector
RNN Cell

GNMT: Putting it All together

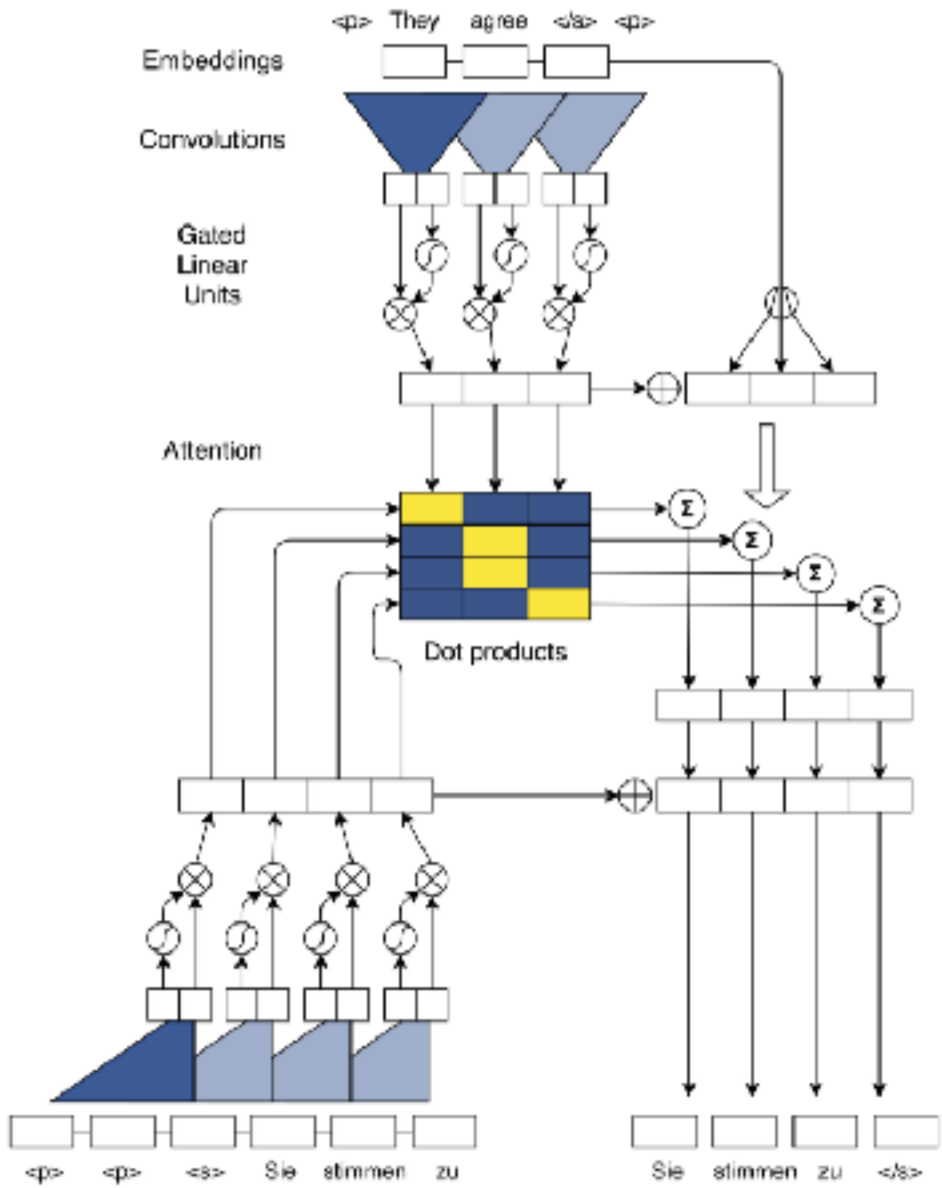
- Google, 2016



Google Neural Machine Translation:
https://arxiv.org/pdf/1609.08144.pdf

CNNs and RNNs

- Can translation also be done using only CNNs?
  - Yes, Facebook AI already did it,
  - 9 times faster than GNMT
  - Similar Performance
  - July, 2017



https://arxiv.org/pdf/1705.03122.pdf

Demo, if time

… from Olivier Grisel

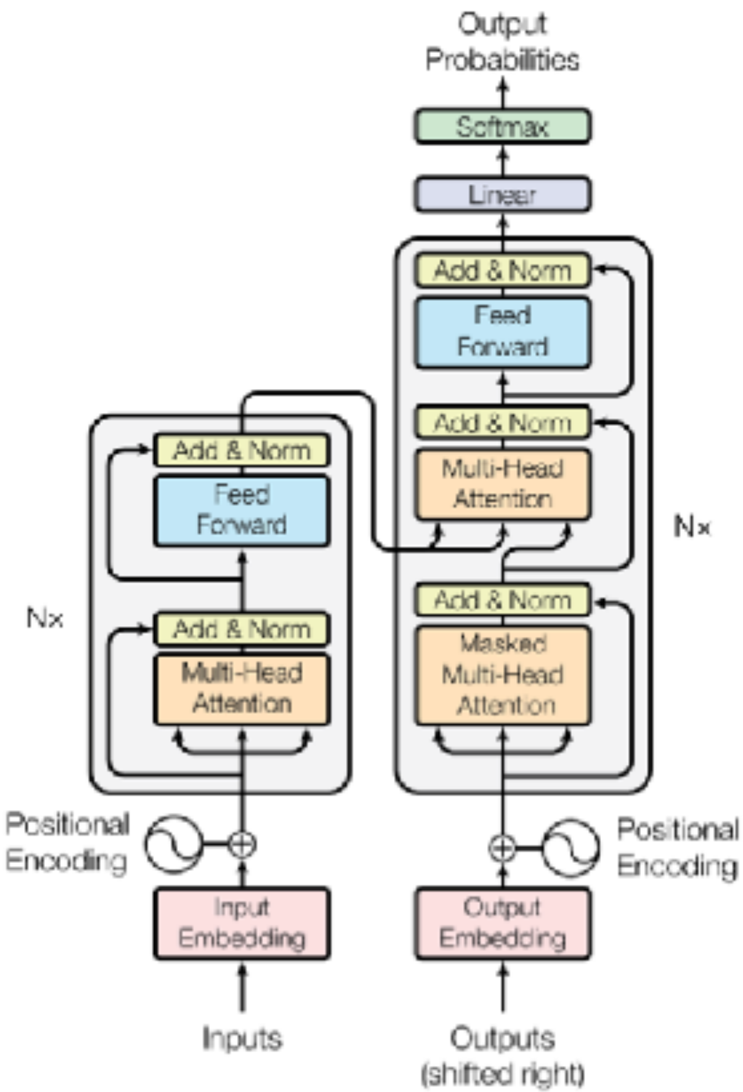
https://github.com/m2dsupsdlclass/lectures-labs/blob/master/labs/07_seq2seq/Translation_of_Numeric_Phrases_with_Seq2Seq_rendered.ipynb

Transformers

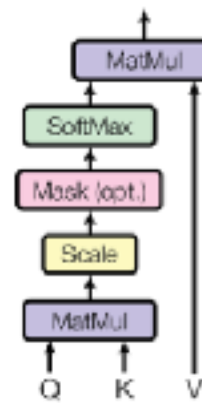
Attention is All You Need

- Well, its a good paper title, but not exactly accurate
- Problem: recurrent networks are not inherently parallelized or efficient at remembering
- Convolution needs many examples from all different word positions (after flattening)
- Filters are not resilient to long-term relationships
- Transformer Solution:
  - Build attention into model from the **beginning**
  - Compare all words to each other through **multi-headed** attention
  - Define a notion of “**position**”in the sentence

Transformer



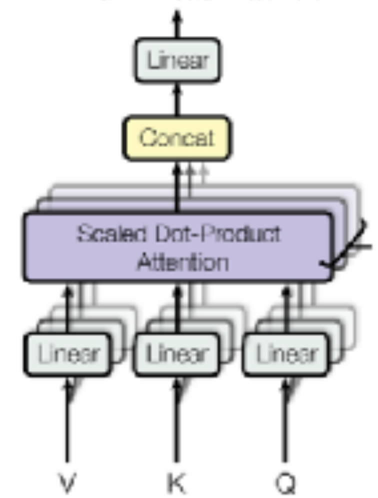
Scaled Dot-Product Attention



for each word

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Head Attention

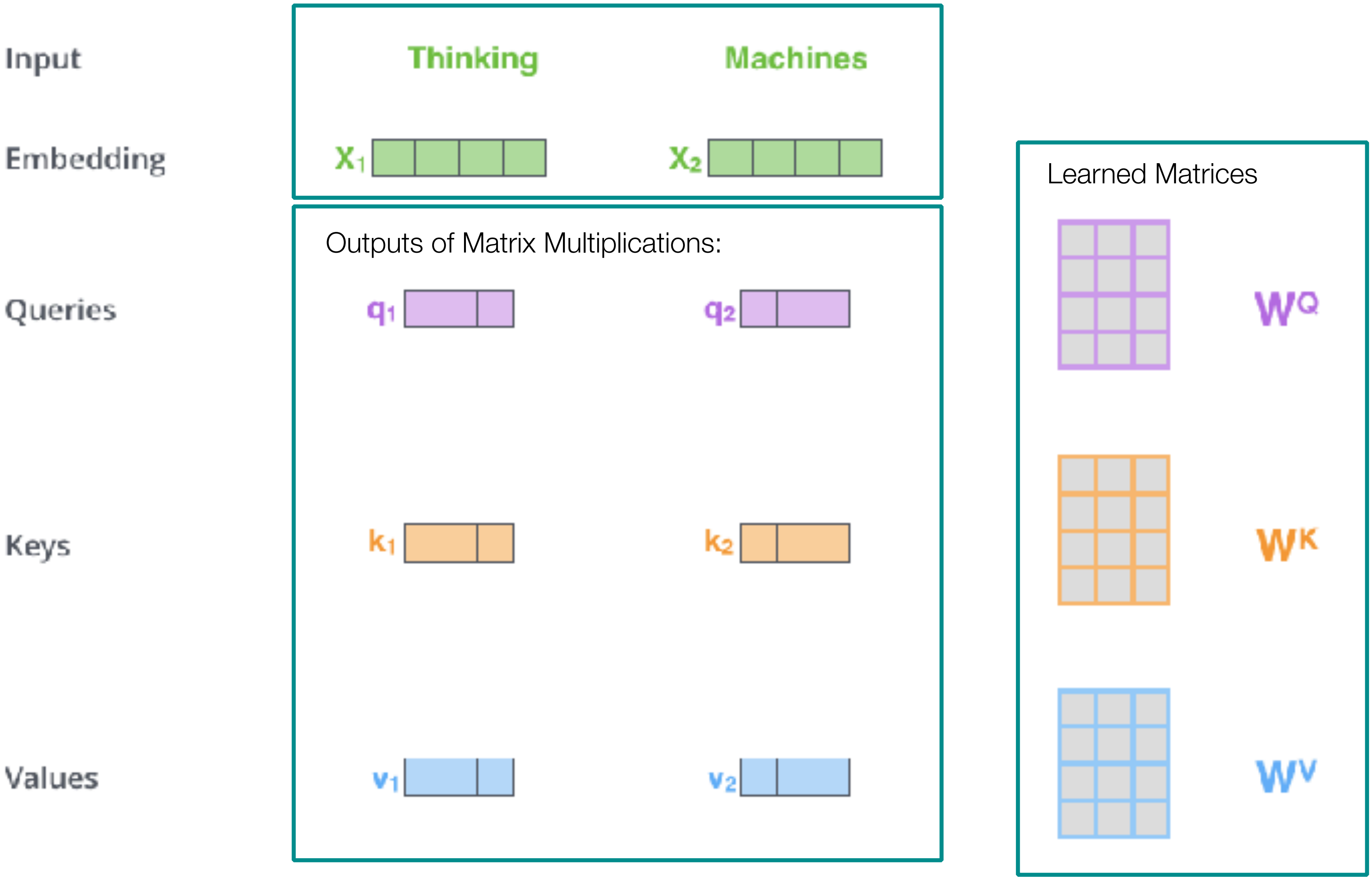


more than one
Q,K,V use in document

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

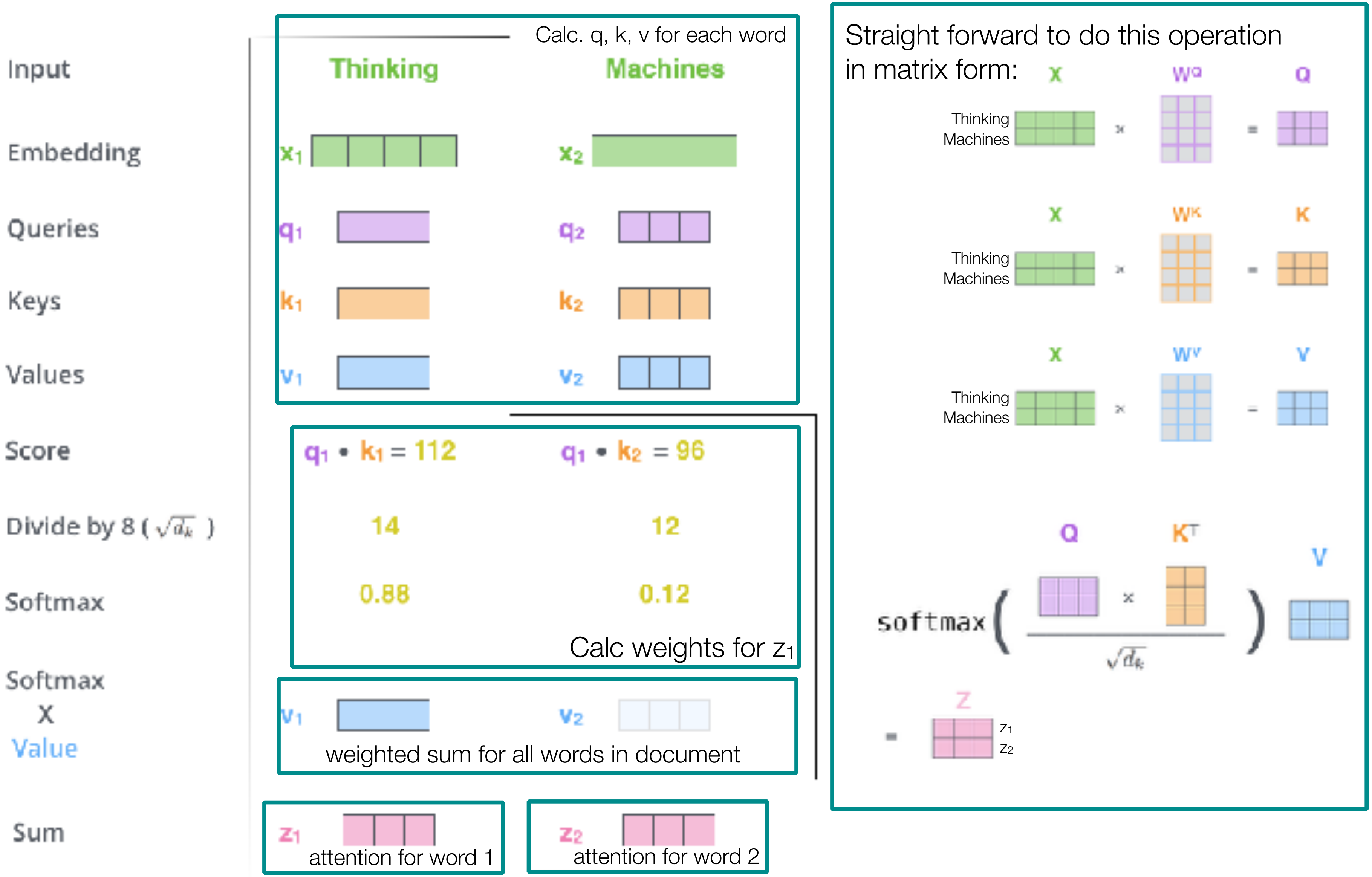
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Transformer: in more detail



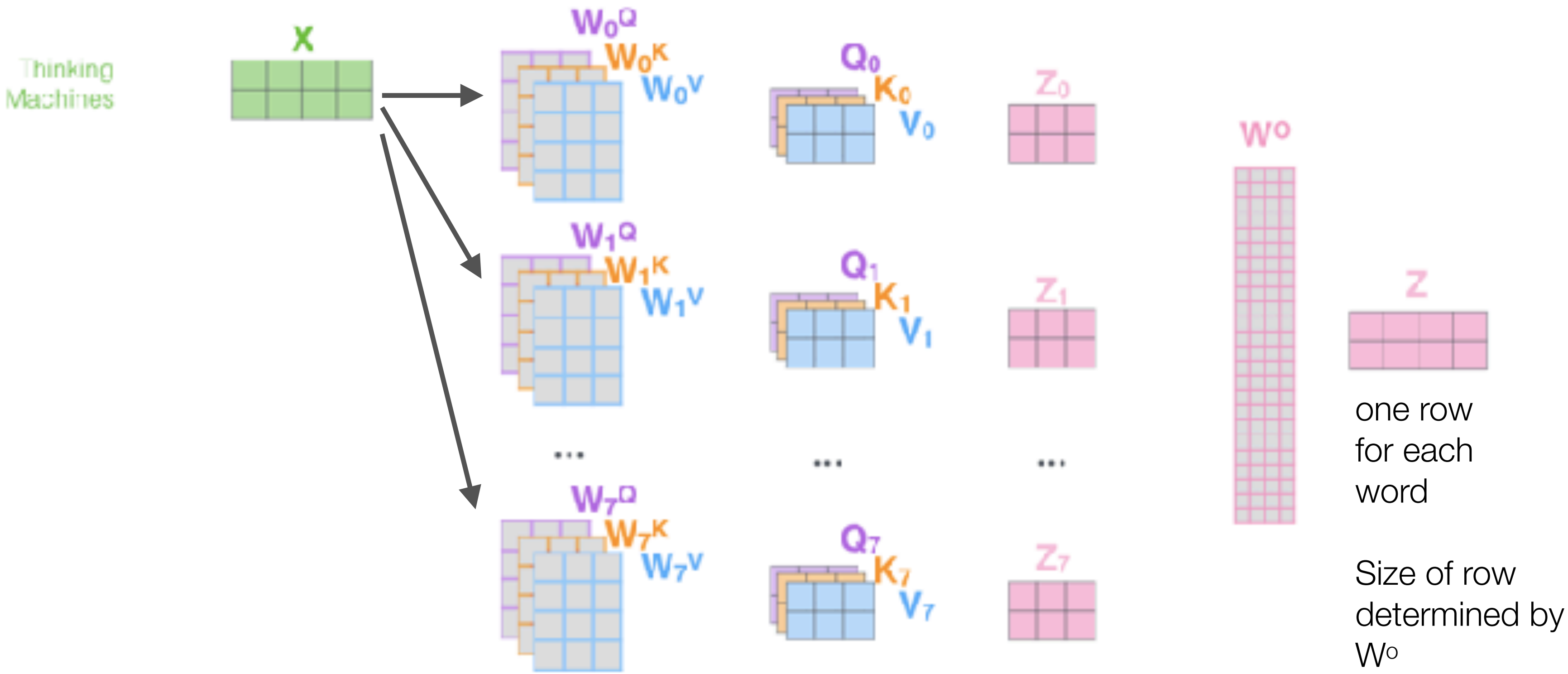
Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

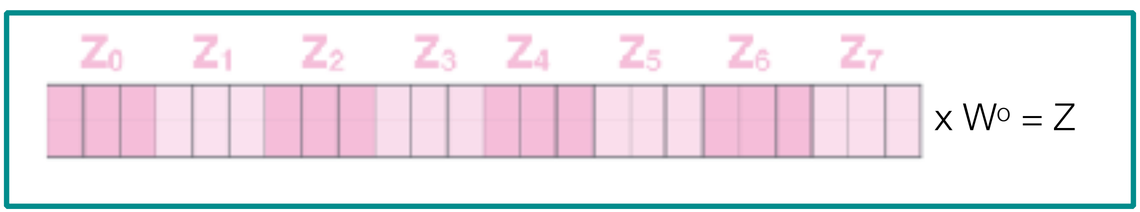
Transformer: in more detail



Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Transformer: Multi-headed Attention



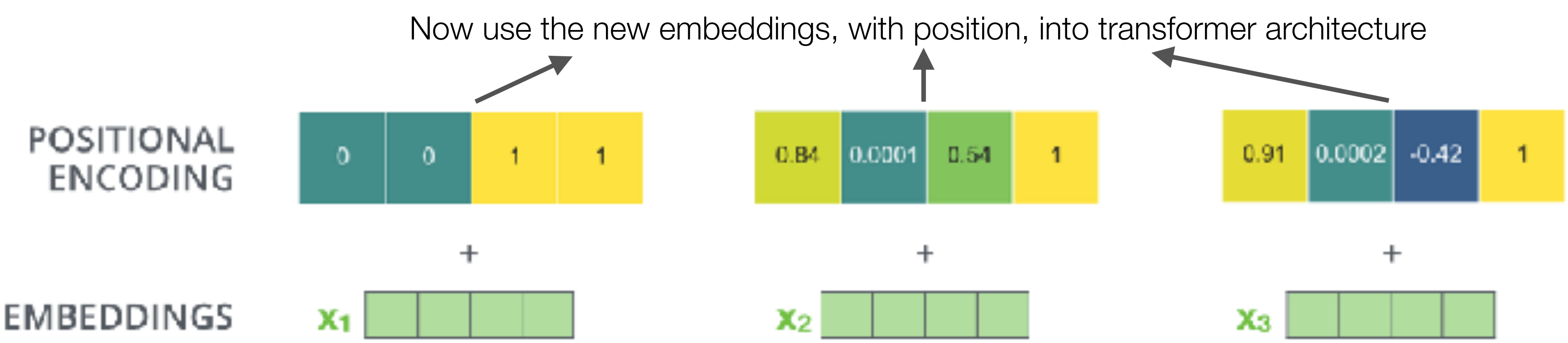


Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Transformer: Positional Encoding

- Objective: add notion of position to embedding
- Attempt in paper: add sin/cos to embedding
- But could be anything that encodes position

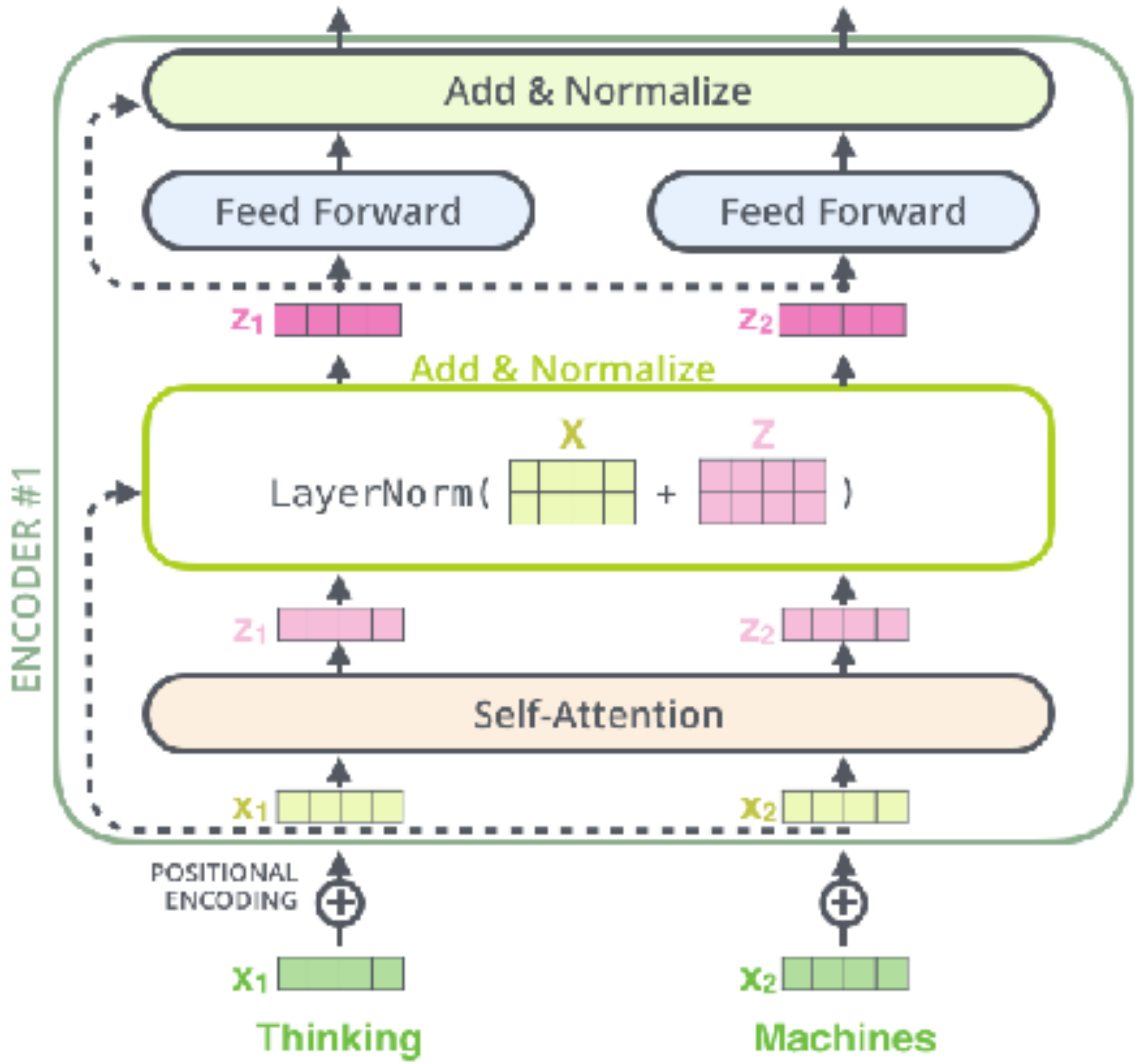
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Hypothesis: Now the word proximity is encoded in the embedding matrix, with other pertinent information. Well, it does help… so it could be true that this is a good way to do it.

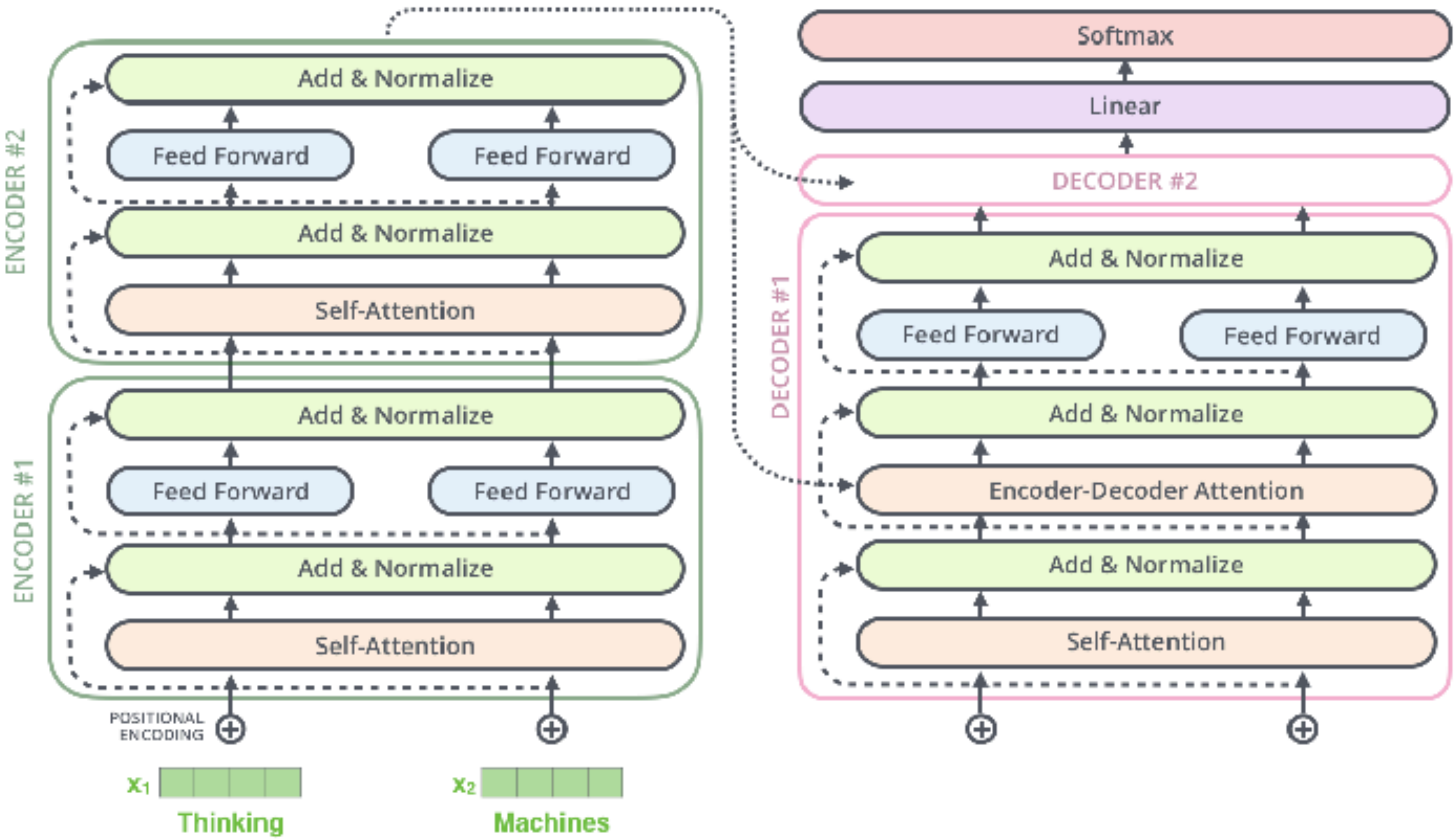
Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Transformer: Residual Connections



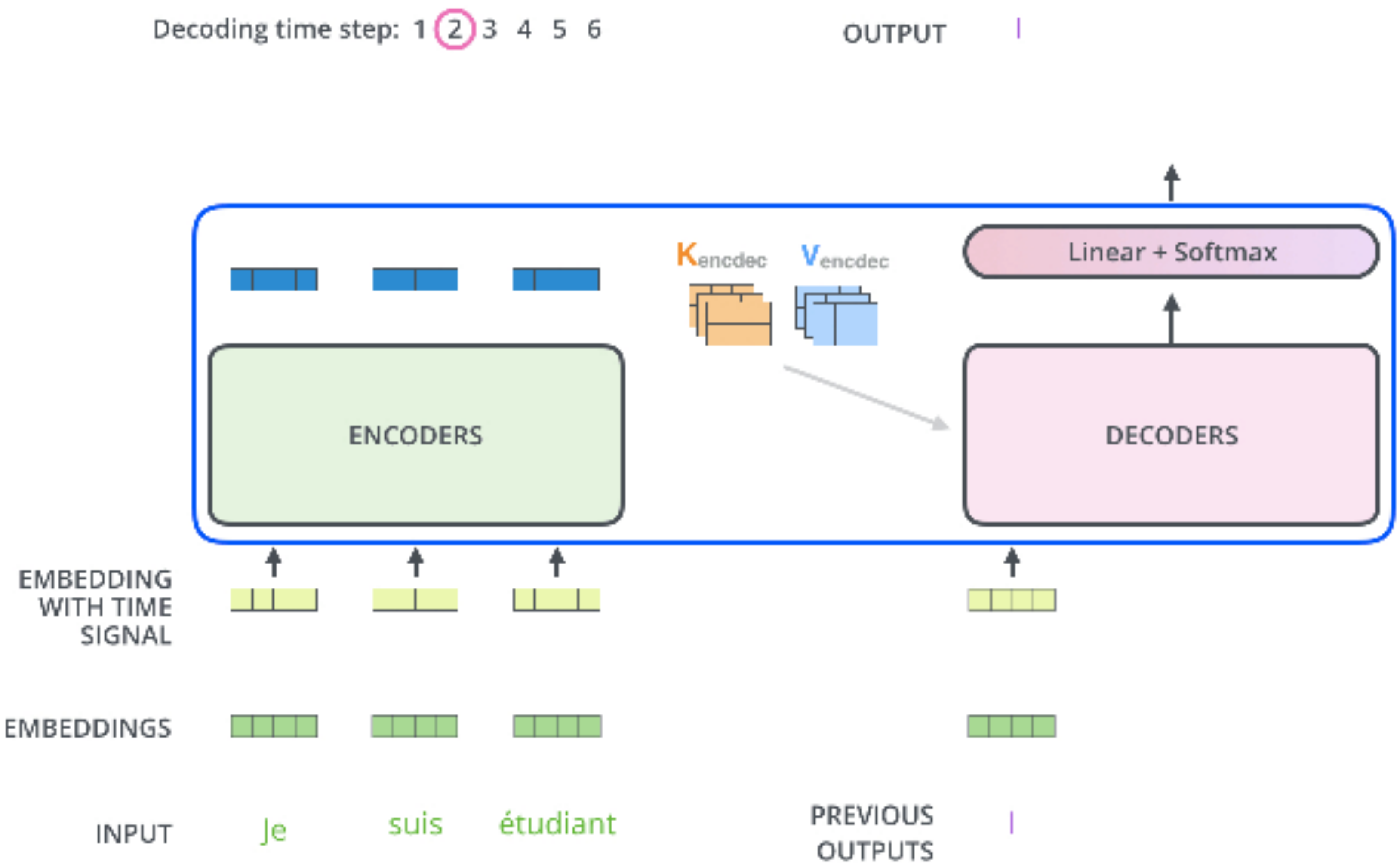
Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Transformer: Putting it all together



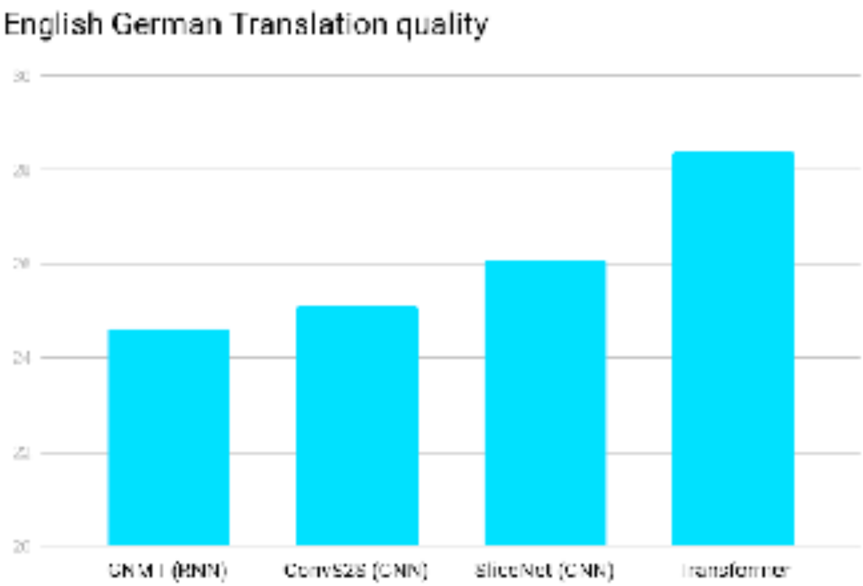
Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

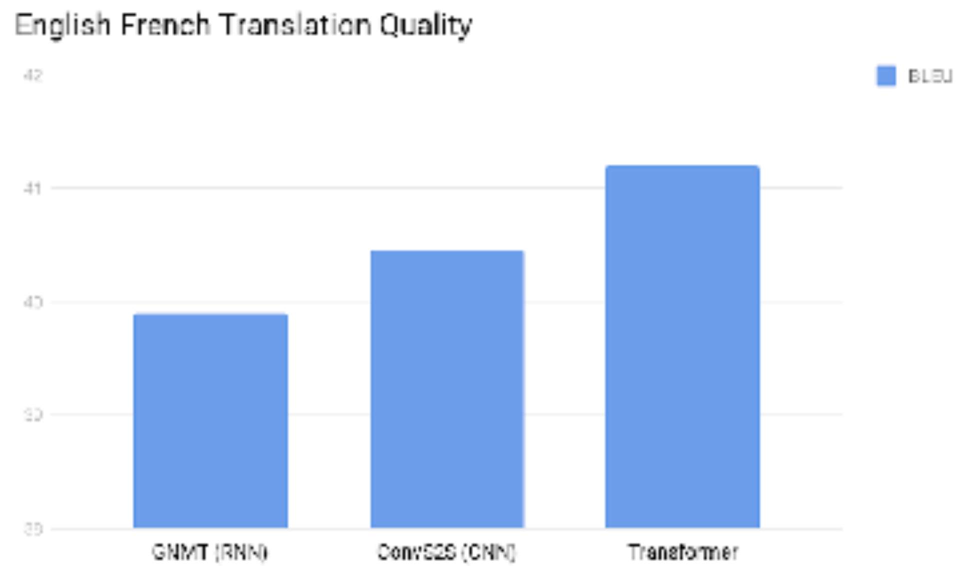
Transformer: Putting it all together



Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Results





https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Implementations:

- Not Native to Keras or Tensorflow, but many Open Source Implementations Exist
- Is Native to PyTorch

Next time

- Class Retrospective

TensorFlow

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

```
with tf.variable_scope('rnn_cell'):
    W = tf.get_variable('W', [num_classes + state_size, state_size])
    b = tf.get_variable('b', [state_size], initializer=tf.constant_initializer(0.0))

def rnn_cell(rnn_input, state):
    with tf.variable_scope('rnn_cell', reuse=True):
        W = tf.get_variable('W', [num_classes + state_size, state_size])
        b = tf.get_variable('b', [state_size], initializer=tf.constant_initializer(0.0))
    return tf.tanh(tf.matmul(tf.concat(1, [rnn_input, state]), W) + b)

state = init_state
rnn_outputs = []
for rnn_input in rnn_inputs:
    state = rnn_cell(rnn_input, state)
    rnn_outputs.append(state)
final_state = rnn_outputs[-1]

#logits and predictions
with tf.variable_scope('softmax'):
    W = tf.get_variable('W', [state_size, num_classes])
    b = tf.get_variable('b', [num_classes], initializer=tf.constant_initializer(0.0))
logits = [tf.matmul(rnn_output, W) + b for rnn_output in rnn_outputs]
predictions = [tf.nn.softmax(logit) for logit in logits]

# Turn our y placeholder into a list labels
y_as_list = [tf.squeeze(i, squeeze_dims=[1]) for i in tf.split(1, num_steps, y)]

#losses and train_step
losses = [tf.nn.sparse_softmax_cross_entropy_with_logits(logit,label) for \
          logit, label in zip(logits, y_as_list)]
total_loss = tf.reduce_mean(losses)
train_step = tf.train.AdagradOptimizer(learning_rate).minimize(total_loss)
```

recurrent networks

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

```
def train_network(num_epochs, num_steps, state_size=4, verbose=True):
    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())
        training_losses = []
        for idx, epoch in enumerate(gen_epochs(num_epochs, num_steps)):
            training_loss = 0
            training_state = np.zeros((batch_size, state_size))
            if verbose:
                print("\nEPOCH", idx)
            for step, (X, Y) in enumerate(epoch):
                tr_losses, training_loss_, training_state, _ = \
                    sess.run([losses,
                              total_loss,
                              final_state,
                              train_step],
                                  feed_dict={x:X, y:Y, init_state:training_state})
                training_loss += training_loss_
                if step % 100 == 0 and step > 0:
                    if verbose:
                        print("Average loss at step", step,
                              "for last 250 steps:", training_loss/100)
                    training_losses.append(training_loss/100)
                    training_loss = 0

    return training_losses
```

TensorFlow

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

```
def train_network(num_epochs, num_steps, state_size=4, verbose=True):
    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())
        for idx, epoch in enumerate(gen_epochs(num_epochs, num_steps)):
            training_state = np.zeros((batch_size, state_size))
            for X, Y in epoch:
                tr_losses, training_loss_, training_state, _ = \
                    sess.run([losses,
                              total_loss,
                              final_state,
                              train_step],
                             feed_dict={x:X, y:Y, init_state:training_state})
```

TensorFlow (simplified)

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

```
cell = tf.nn.rnn_cell.BasicRNNCell(state_size)
rnn_outputs, final_state = tf.nn.rnn(cell, rnn_inputs, initial_state=init_state)
```

```
loss_weights = [tf.ones([batch_size]) for i in range(num_steps)]
losses = tf.nn.seq2seq.sequence_loss_by_example(logits, y_as_list, loss_weights)
```

```
x = tf.placeholder(tf.int32, [batch_size, num_steps], name='input_placeholder')
y = tf.placeholder(tf.int32, [batch_size, num_steps], name='labels_placeholder')
init_state = tf.zeros([batch_size, state_size])

x_one_hot = tf.one_hot(x, num_classes)
rnn_inputs = tf.unpack(x_one_hot, axis=1)

cell = tf.nn.rnn_cell.BasicRNNCell(state_size)
rnn_outputs, final_state = tf.nn.rnn(cell, rnn_inputs, initial_state=init_state)
```

```
with tf.variable_scope('softmax'):
    W = tf.get_variable('W', [state_size, num_classes])
    b = tf.get_variable('b', [num_classes], initializer=tf.constant_initializer(0.0))
logits = [tf.matmul(rnn_output, W) + b for rnn_output in rnn_outputs]
predictions = [tf.nn.softmax(logit) for logit in logits]

y_as_list = [tf.squeeze(i, squeeze_dims=[1]) for i in tf.split(1, num_steps, y)]

loss_weights = [tf.ones([batch_size]) for i in range(num_steps)]
losses = tf.nn.seq2seq.sequence_loss_by_example(logits, y_as_list, loss_weights)
total_loss = tf.reduce_mean(losses)
train_step = tf.train.AdagradOptimizer(learning_rate).minimize(total_loss)
```