Recurrent Neural Networks

Seoul AI Meetup, August 5

Martin Kersner, m.kersner@gmail.com

References

Books

- Hands-On Machine Learning with Scikit-Learn and Tensorflow (Chapter 14. Recurrent Neural Networks)
  - https://www.safaribooksonline.com
  - https://github.com/ageron/handson-ml
- Deep Learning Book (Chapter 10: Sequence Modeling: Reccurent and Recursive Nets)
  - http://www.deeplearningbook.org/
  - https://github.com/HFTrader/DeepLearningBook

Videos

- CS231n Lecture 10 - Recurrent Neural Networks, Image Captioning, LSTM (Andrej Karpathy)
- Lecture 8: Recurrent Neural Networks and Language Models (Richard Socher)
- Deep Learning Summer School 2016, Recurrent Neural Networks (Yoshua Bengio)
- Ch 10: Recurrent/Recursive Nets, DeepLearning Textbook Study Group (Jeremy Howard)
- MIT 6.S094: Recurrent Neural Networks for Steering Through Time (Lex Fridman)

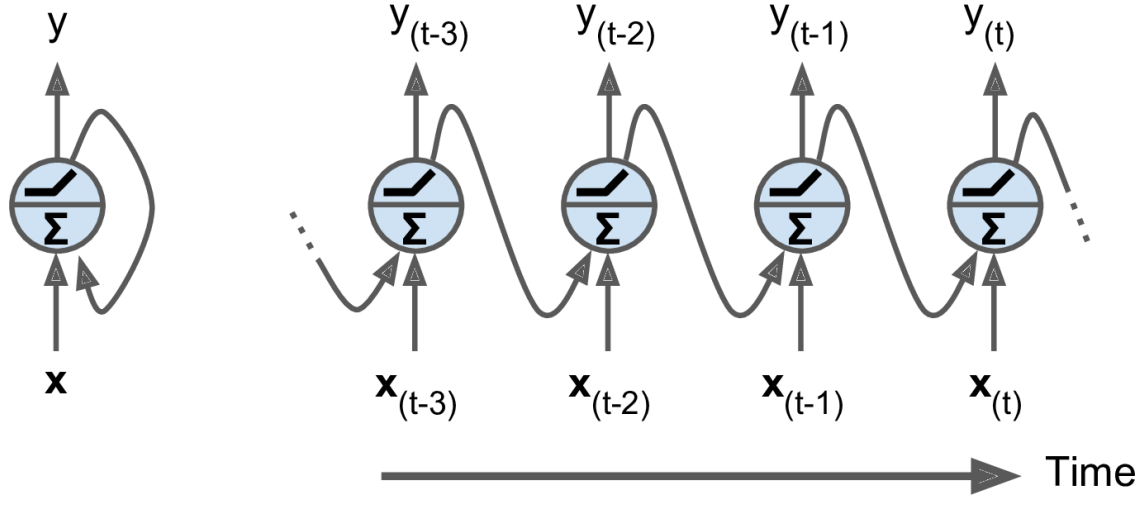
Feed Forward Neural Networks

Feed Forward Neural Networks has following limitations.

- Inputs and outputs have **fixed size**.
- Assume **independence** between input data.

Recurrent Neural Networks (RNN)

- RNN operate over sequences of data.
  - **Sequences** in the **inputs**.
  - **Sequences** in the **outputs**.
  - **Sequences** in both **inputs** and **outputs**.
- Weights and biases are shared over time.



Left: Recurrent neural network with one neuron in cell.

Right: **Unfolded (= unrolled)** recurrent neural network with one neuron in cell.

Implementation of single RNN cell

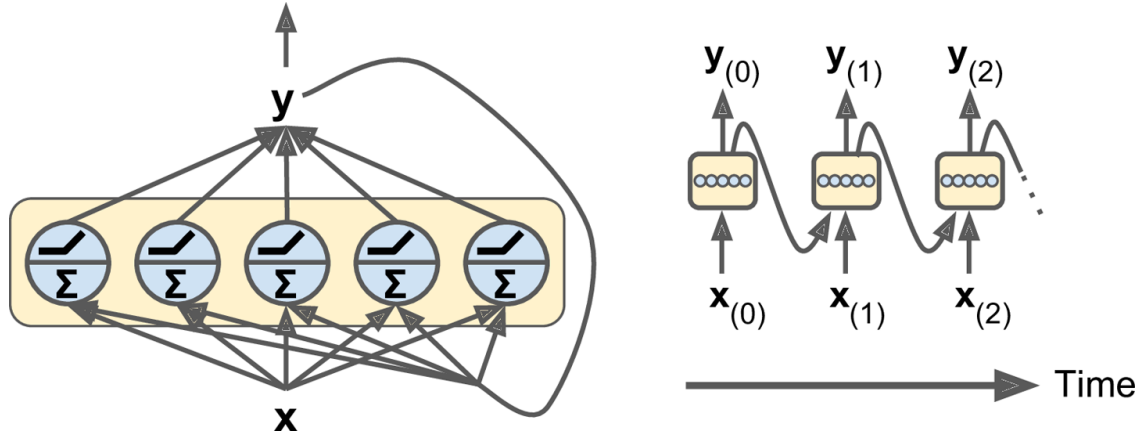
```
# x represents input data             [batch_size, n_input_features]
# h represents hidden state           [batch_size, n_neurons]
# W_xh weights applied to input data  [n_input_features, n_neurons]
# W_hh weights of hidden state        [n_neurons, n_neurons]
# W_hy weights for output             [n_neurons, n_outputs]
# activation function tanh squashes data in between [-1, 1]

h = np.tanh(np.dot(x, W_xh) + np.dot(h, W_hh))

# same as expression above
#h = np.tanh(np.dot(np.hstack((x, h)), np.vstack((W_xh, W_hh))))

# prediction at current time
y = np.dot(h, W_hy)
```

Layer of Recurrent Neurons

Connections between

- input and hidden layer,
- hidden layer in time t_i and hidden layer in time t_{i+1} and
- hidden layer and output layer

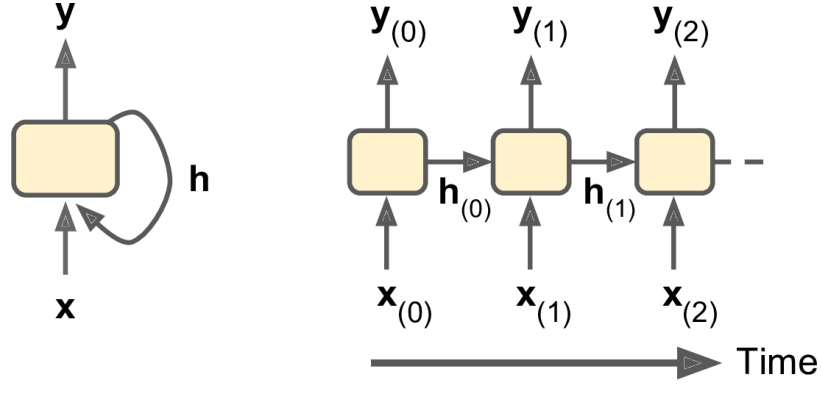
are **fully connected**.



Left: Recurrent neural network with cell with 5 neurons.

Right: **Unfolded (= unrolled)** recurrent neural network with 5 neurons.

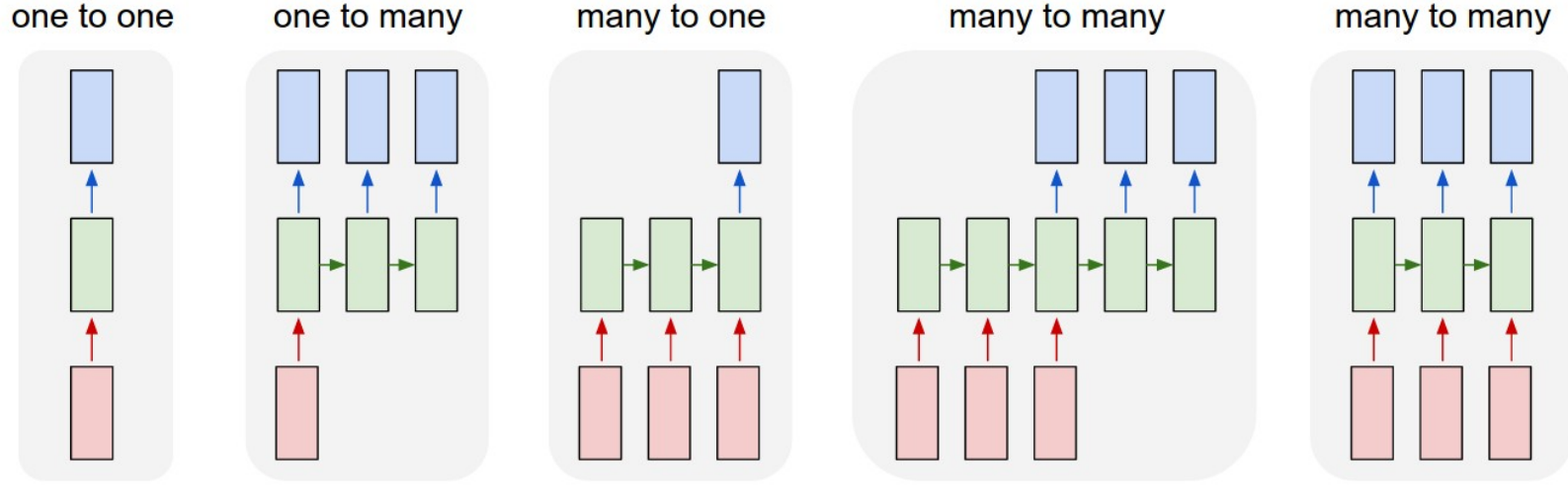
Memory Cell



- Simple recurrent neuron or layer of recurrent neurons.
- Memory cell preserves state across time steps.

Different inputs and output in RNN architectures

- Vector to Vector (Feed Forward Neural Network)
- Vector to Sequence (e.g. Image Captioning)
- Sequence to Vector (e.g. Sentiment Analysis)
- Sequence to Sequence (e.g. Machine Translation)
- Synced Sequence to Sequence (e.g. Video Captioning)



Building RNN in Tensorflow

1. Manully
2. `static_rnn()`
3. `dynamic_rnn()`

Building RNN manually

In [2]:
```
n_features = 3
n_neurons  = 5
n_steps    = 2
```

In [5]:
```
# batch of size 4 for two time steps
X0_batch = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 0, 1]]) # t = 0
X1_batch = np.array([[9, 8, 7], [0, 0, 0], [6, 5, 4], [3, 2, 1]]) # t = 1
```

In [19]:

```
# source https://github.com/ageron/handson-ml

# placeholders for input data at two time steps
X0 = tf.placeholder(tf.float32, [None, n_features])
X1 = tf.placeholder(tf.float32, [None, n_features])

# weight for input data to cell connection
Wx = tf.Variable(tf.random_normal(shape=[n_features, n_neurons], dtype=tf.float32))

# weight for recurrent connection (t-1 => t)
Wy = tf.Variable(tf.random_normal(shape=[n_neurons, n_neurons], dtype=tf.float32))

# bias
b  = tf.Variable(tf.zeros([1, n_neurons], dtype=tf.float32))

# tf.matmul(X0, Wx) : [None, n_features] * [n_features, n_neurons] = [None, n_neuron
s]
Y0 = tf.tanh(tf.matmul(X0, Wx) + b)

# tf.matmul(Y0, Wy) : [None,   n_neurons] * [n_neurons, n_neurons] = [None, n_neuron
s]
# tf.matmul(X1, Wx) : [None,  n_features] * [n_neurons, n_neurons] = [None, n_neuron
s]
# b : [1, n_neurons]
Y1 = tf.tanh(tf.matmul(Y0, Wy) + tf.matmul(X1, Wx) + b)
```

In [21]:
```
def process_batches(X0_batch, X1_batch):
    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        init.run()
        Y0_val, Y1_val = sess.run([Y0, Y1], feed_dict={X0: X0_batch, X1: X1_batch})

    print("Y0\n",   Y0_val)
    print("\nY1\n", Y1_val)
```

In [22]:
```
process_batches(X0_batch, X1_batch)
```

```
Y0
 [[-0.0664006   0.96257669  0.68105787  0.70918542 -0.89821595]
 [ 0.9977755  -0.71978885 -0.99657625  0.9673925  -0.99989718]
 [ 0.99999774 -0.99898815 -0.99999893  0.99677622 -0.99999988]
 [ 1.         -1.         -1.         -0.99818915  0.99950868]]

Y1
 [[ 1.          -1.          -1.           0.40200216 -1.        ]
 [-0.12210433  0.62805319  0.96718419 -0.99371207 -0.25839335]
 [ 0.99999827 -0.9999994  -0.9999975  -0.85943311 -0.9999879 ]
 [ 0.99928284 -0.99999815 -0.99990582  0.98579615 -0.92205751]]
```

Building RNN using `static_rnn()`

- tf.contrib.rnn.BasicRNNCell
- tf.nn.static_rnn creates one cell per time step.
- Each input placeholder (`X0`, `X1`) have to be manually defined.

In [24]:
```
# source https://github.com/ageron/handson-ml
X0 = tf.placeholder(tf.float32, [None, n_features])
X1 = tf.placeholder(tf.float32, [None, n_features])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
output_seqs, states = tf.nn.static_rnn(basic_cell, [X0, X1], dtype=tf.float32)
Y0, Y1 = output_seqs
```

`static_rnn()` output

In [25]:
```
process_batches(X0_batch, X1_batch)
```

```
Y0
 [[ 0.30741334 -0.32884315 -0.65428472 -0.93850589  0.52089024]
 [ 0.99122757 -0.95425421 -0.75180793 -0.99952078  0.98202348]
 [ 0.99992681 -0.99783254 -0.82473528 -0.9999963   0.99947774]
 [ 0.99677098 -0.68750614  0.84199691  0.93039107  0.8120684 ]]

Y1
 [[ 0.99998885 -0.99976051 -0.06679298 -0.99998039  0.99982214]
 [-0.65249437 -0.51520866 -0.37968954 -0.59225935 -0.08968385]
 [ 0.99862403 -0.99715197 -0.03308626 -0.99915648  0.99329019]
 [ 0.99681675 -0.95981938  0.39660636 -0.83076048  0.79671967]]
```

`static_rnn()` with single input placeholder

In [8]:
```
# source https://github.com/ageron/handson-ml
X = tf.placeholder(tf.float32, [None, n_steps, n_features])
X_seqs = tf.unstack(tf.transpose(X, perm=[1, 0, 2]))

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
output_seqs, states = tf.nn.static_rnn(basic_cell, X_seqs, dtype=tf.float32)
outputs = tf.transpose(tf.stack(output_seqs), perm=[1, 0, 2])
```

In [13]:

```
def process_batches2(X0_batch, X1_batch):
    # source https://github.com/ageron/handson-ml
    X0_batch_tmp = X0_batch[:, np.newaxis, :]
    X1_batch_tmp = X1_batch[:, np.newaxis, :]
    X_batch = np.concatenate((X0_batch_tmp, X1_batch_tmp), axis=1)

    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        init.run()
        outputs_val = outputs.eval(feed_dict={X: X_batch})

    # Y0 output at t = 0
    # Y1 output at t = 0
    print("Y0\n",   np.transpose(outputs_val, axes=[1, 0, 2])[0])
    print("\nY1\n", np.transpose(outputs_val, axes=[1, 0, 2])[1])
```

In [14]:
```
process_batches2(X0_batch, X1_batch)
```

```
Y0
 [[-0.45652324 -0.68064123  0.40938237  0.63104504 -0.45732826]
 [-0.80015349 -0.99218267  0.78177971  0.9971031  -0.99646091]
 [-0.93605185 -0.99983788  0.93088669  0.99998152 -0.99998295]
 [ 0.99273688 -0.99819332 -0.55543643  0.9989031  -0.9953323 ]]

Y1
 [[-0.94288003 -0.99988687  0.94055814  0.99999851 -0.9999997 ]
 [-0.63711601  0.11300932  0.5798437   0.43105593 -0.63716984]
 [-0.9165386  -0.99456042  0.89605415  0.99987197 -0.99997509]
 [-0.02746334 -0.73191994  0.7827872   0.95256817 -0.97817713]]
```

Building RNN using `dynamic_rnn()`

- tf.nn.dynamic_rnn
- **No need to unstack, stack and transpose!**
- Input `[None, n_steps, n_features]`.
- Output `[None, n_steps, n_neurons]`

In [46]:
```
# source https://github.com/ageron/handson-ml
X = tf.placeholder(tf.float32, [None, n_steps, n_features])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)

process_batches2(X0_batch, X1_batch)
```

```
Y0
 [[ 0.80872238 -0.52312446 -0.6716494  -0.69762248 -0.54384488]
 [ 0.99547106 -0.02155113 -0.99482894  0.17964774 -0.83173698]
 [ 0.99990267  0.49111056 -0.9999314   0.8413834  -0.9444679 ]
 [-0.80632919  0.93928123 -0.97309881  0.99996096  0.97433066]]

Y1
 [[ 0.9995454   0.99339807 -0.99998379  0.99919224 -0.98379493]
 [-0.06013332  0.4030143   0.02884481 -0.29437575 -0.85681593]
 [ 0.99406189  0.95815992 -0.99768937  0.98646194 -0.91752487]
 [ 0.95047355 -0.51205158 -0.27763969  0.83108062  0.81631833]]
```

Variable-Length Input Sequences in Tensorflow

- Sentences, video, audio, ...
- Parameter `sequence_length` in `dynamic_rnn()` represents the lenghts of input vector.
- Outputs of RNN are **zero vectors** for every time step past the input sequence length.

```
seq_length = tf.placeholder(tf.int32, [None])
...
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32,
                                    sequence_length=seq_length)
```

Initialization of `sequence_length`

In [3]:
```
X_batch = np.array([
        # step 0     step 1
        [[0, 1, 2], [9, 8, 7]], # instance 0
        [[3, 4, 5], [0, 0, 0]], # instance 1 (padded with a zero vector)
        [[6, 7, 8], [6, 5, 4]], # instance 2
        [[9, 0, 1], [3, 2, 1]], # instance 3
    ])

seq_length_batch = np.array([2, 1, 2, 2])
```

Variable-Length Output Sequences in Tensorflow

- Output length is **known**.
  - Solve similarly as with `output_sequences`.
  - Ignore every output past the length of output sequence.

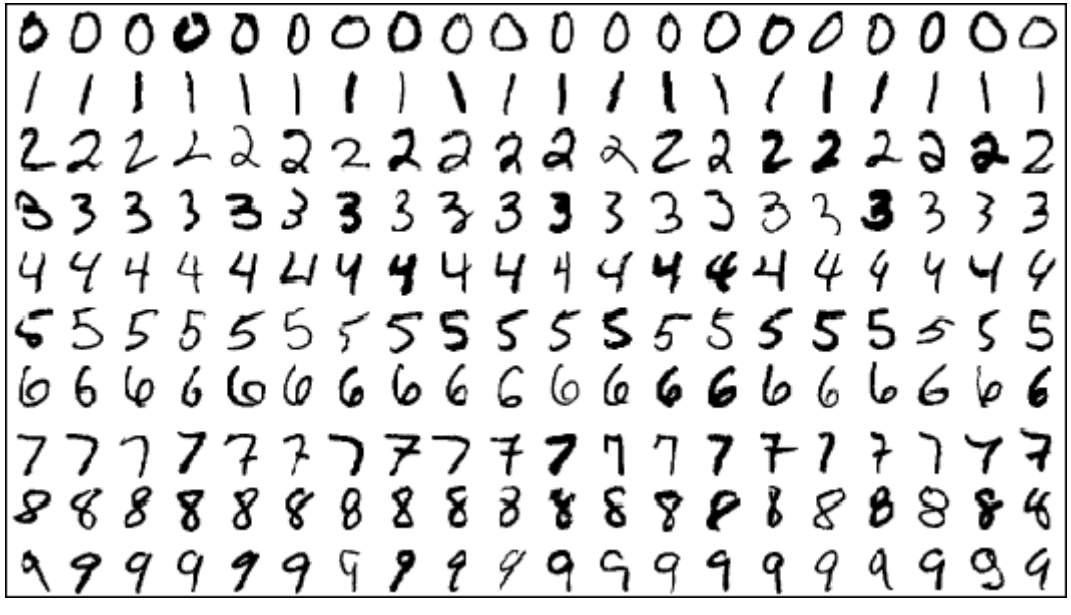
- Output length is **unknown**.
  - Generate EOS (end-of-sequence) token.
  - Ignore every output past the EOS token.

Training RNN in Tensorflow

- Backpropagation Through Time (BPTT)
  - Forward pass
  - Compute cost function $C(Y_0, Y_1, \ldots, Y_{n-1}, Y_n)$.
  - Propagate gradient of cost function through unrolled network.
  - Update model parameters using the gradients computed during BPTT.

MNIST

- Dataset of handwritten digits [0-9]
- 28x28 px
- Grayscale



In [ ]:

```
# source https://github.com/ageron/handson-ml
n_steps   = 28
n_inputs  = 28
n_neurons = 150
n_outputs = 10

learning_rate = 0.001

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.int32,   [None])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)

# states = final outputs, after n_steps = 28
# outputs = outputs at every time step => 28 outputs
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)
```

In [20]:

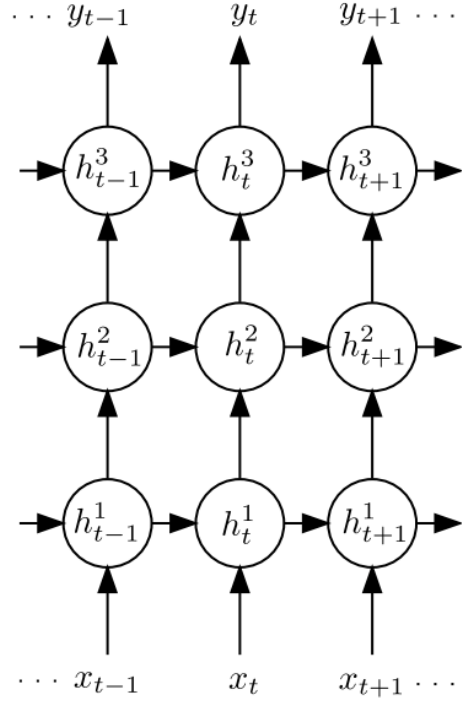
```
# source https://github.com/ageron/handson-ml
# states variable contains state of RNN cell after n_steps = 28
logits = tf.layers.dense(states, n_outputs)
xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
loss = tf.reduce_mean(xentropy)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
correct = tf.nn.in_top_k(logits, y, 1) # only one correct output
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
```

In [2]:
```
# source https://github.com/ageron/handson-ml
n_epochs   = 100
batch_size = 150

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            X_batch = X_batch.reshape((-1, n_steps, n_inputs)) # 150, 28, 28
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})

        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: X_test, y: y_test})
        #print(epoch, "Train accuracy:", acc_train, "Test accuracy:", acc_test)
```

```
...
97 Train accuracy: 1.0 Test accuracy: 0.9809
98 Train accuracy: 0.986667 Test accuracy: 0.9761
99 Train accuracy: 0.986667 Test accuracy: 0.9769
```

Deep RNN

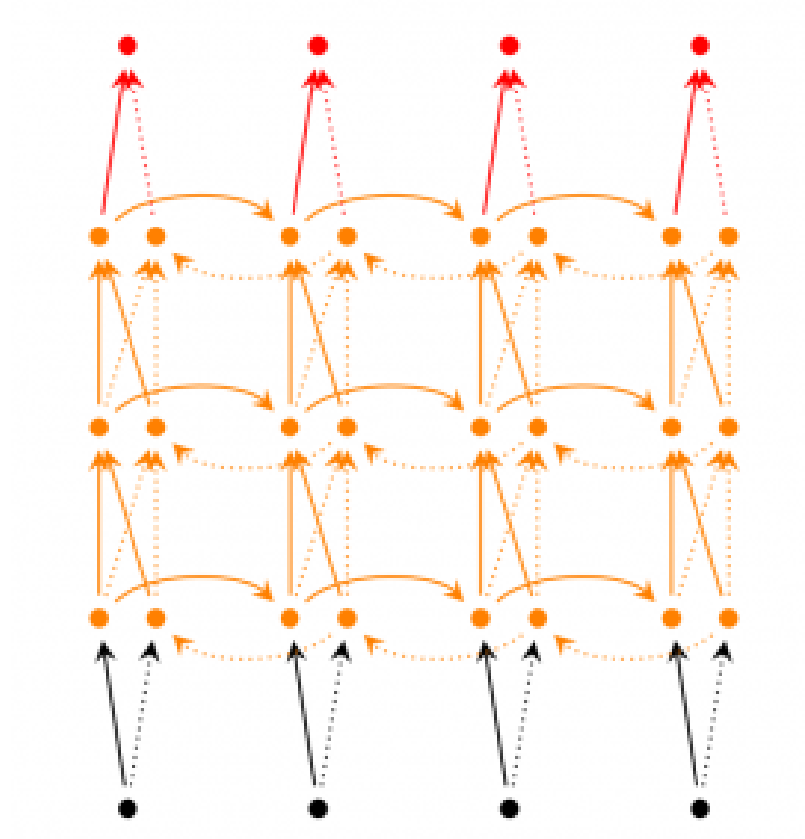
- Stack of multiple layers of cells.
- tf.contrib.rnn.MultiRNNCell



Implementation of deep RNN in Tensorflow

```
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
multi_layer_cell = tf.contrib.rnn.MultiRNNCell([basic_cell] * n_layers)
outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)
```

Bidirectional Recurrent Neural Networks

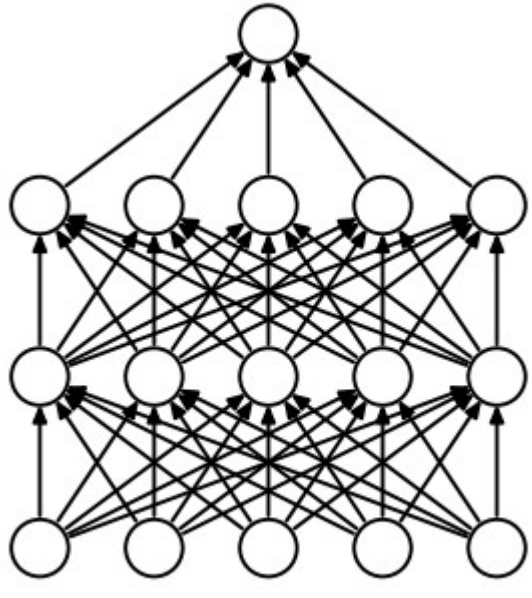


Dropout

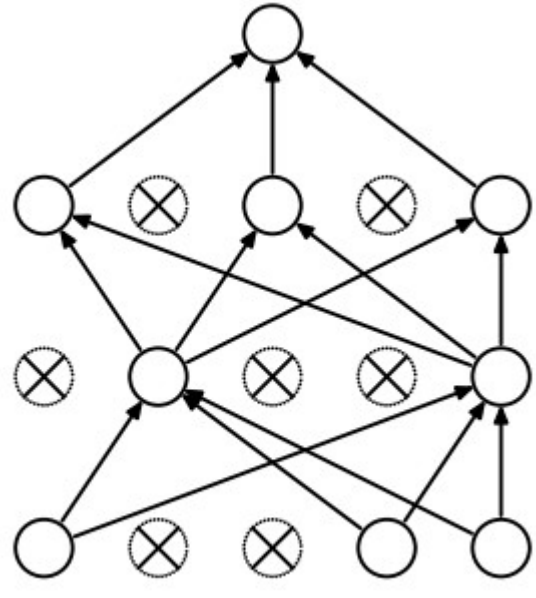
tf.contrib.rnn.DropoutWrapper applies dropout during both training and testing phase!

Solution

- Create own wrapper.
- Create two graphs; one for training, one for testing.



(a) Standard Neural Net



(b) After applying dropout.

Dropout with two graphs

```
keep_prob = 0.5
cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
if is_training:
    cell = tf.contrib.rnn.DropoutWrapper(cell, input_keep_prob=keep_prob)

...

with tf.Session() as sess:
    if is_training:
        init.run()
        for iteration in range(n_iterations):
            # train the model
        save_path = saver.save(sess, "model.ckpt")
    else:
        saver.restore(sess, "model.ckpt")
        # use the model
```

RNN problems

With long input sequences RNN suffers from several problems.

- Vanishing/Exploding gradients
- Non-convergance
- Memory of the first inputs fade away
- Training of long sequences is slow

Partial solutions

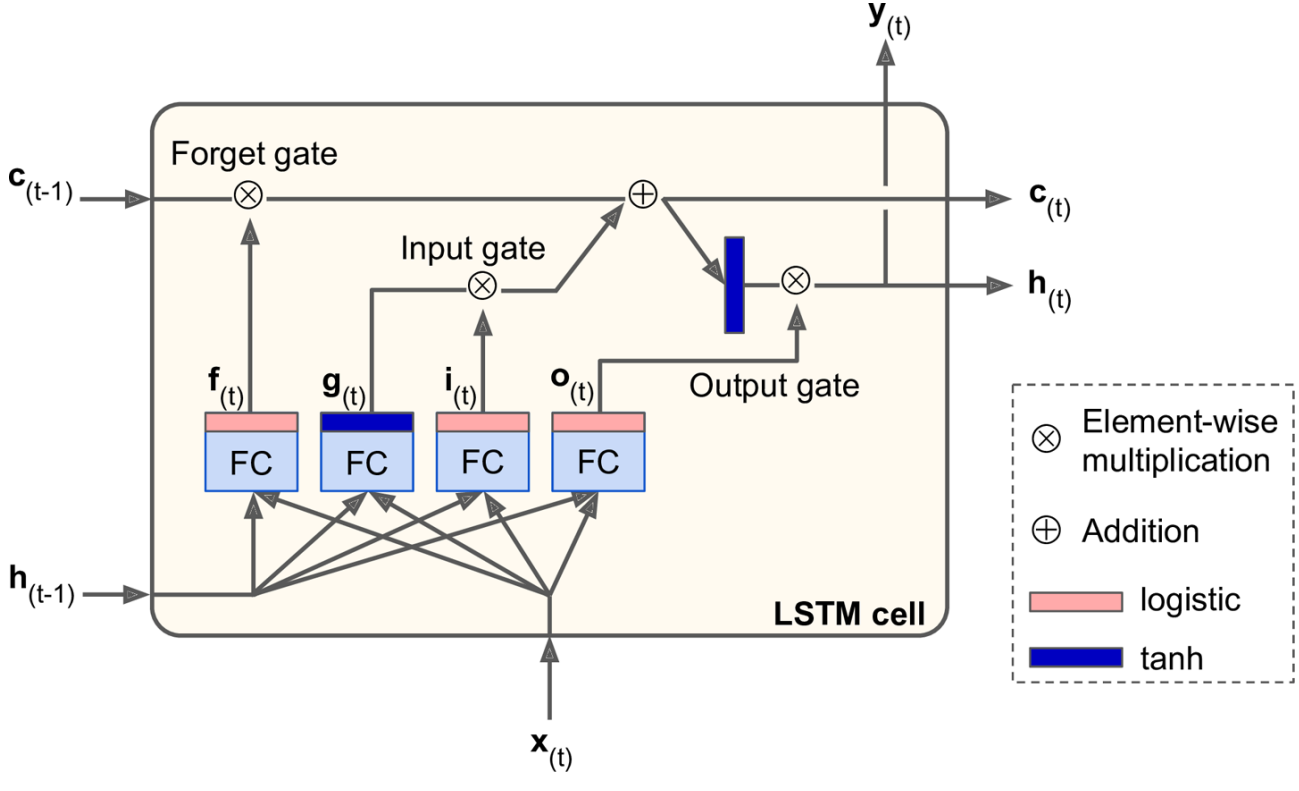
- Good parameter initialization (weights initialized as identity matrix)
- Nonsaturating activation functions (e.g., ReLU)
- Batch Normalization
- Gradient Clipping
- Faster optimizers
- **Truncated** Backpropagation Through Time => model cannot learn long-term dependencies.

LSTM Cell

- Long Short-Term Memory, S. Hochreiter and J. Schmidhuber (1997)
- Same inputs and outputs as basic RNN cell, but state is split.
- Faster convergence.
- Detect long-term dependencies in data.
- 4 different fully connected layers
- 3 gates (learn what to store in the long-term state, what to throw away, and what to read from it)
  - Input
  - Forget
  - Output
- 2 states
  - short-term
  - long-term

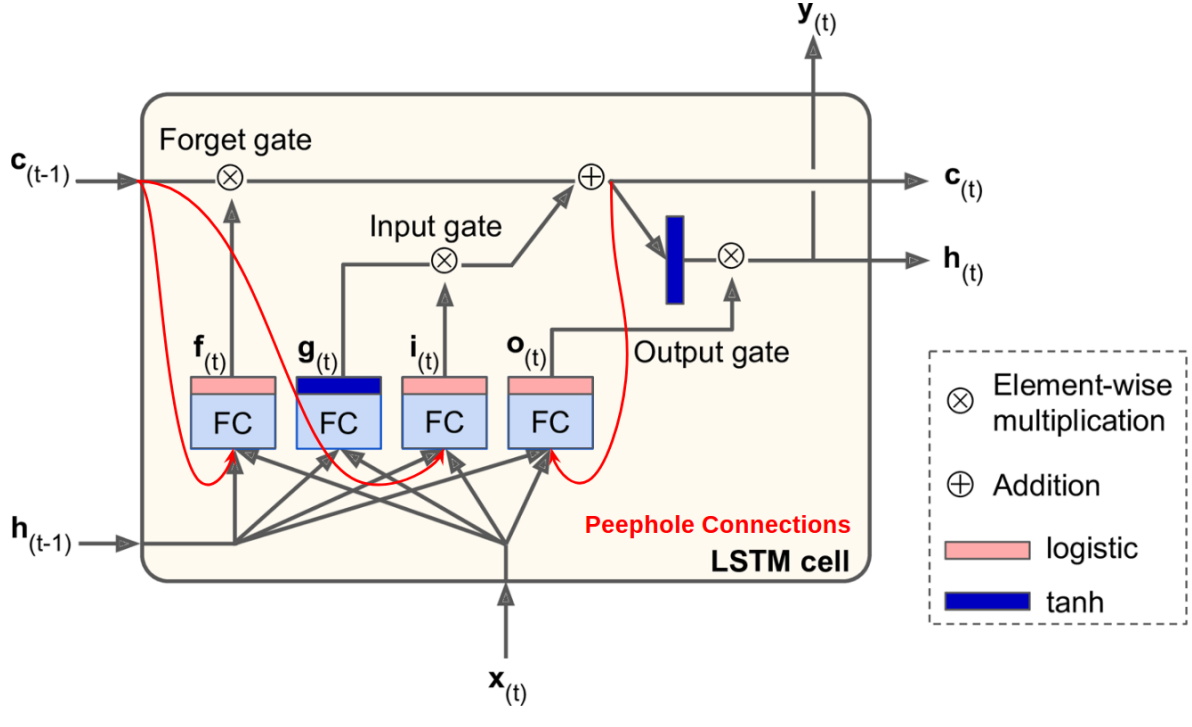
- Tensorflow tf.contrib.rnn.BasicLSTMCell
- Keras keras.layers.recurrent.LSTM

Visualization of LSTM cell



Peephole Connections

- Recurrent Nets that Time and Count, F. Gers and J. Schmidhuber (2000)
- In LSTM gate controllers utilize only previous state and current input.
- Peephole connections allow them to use ("peep") long-term state as well.

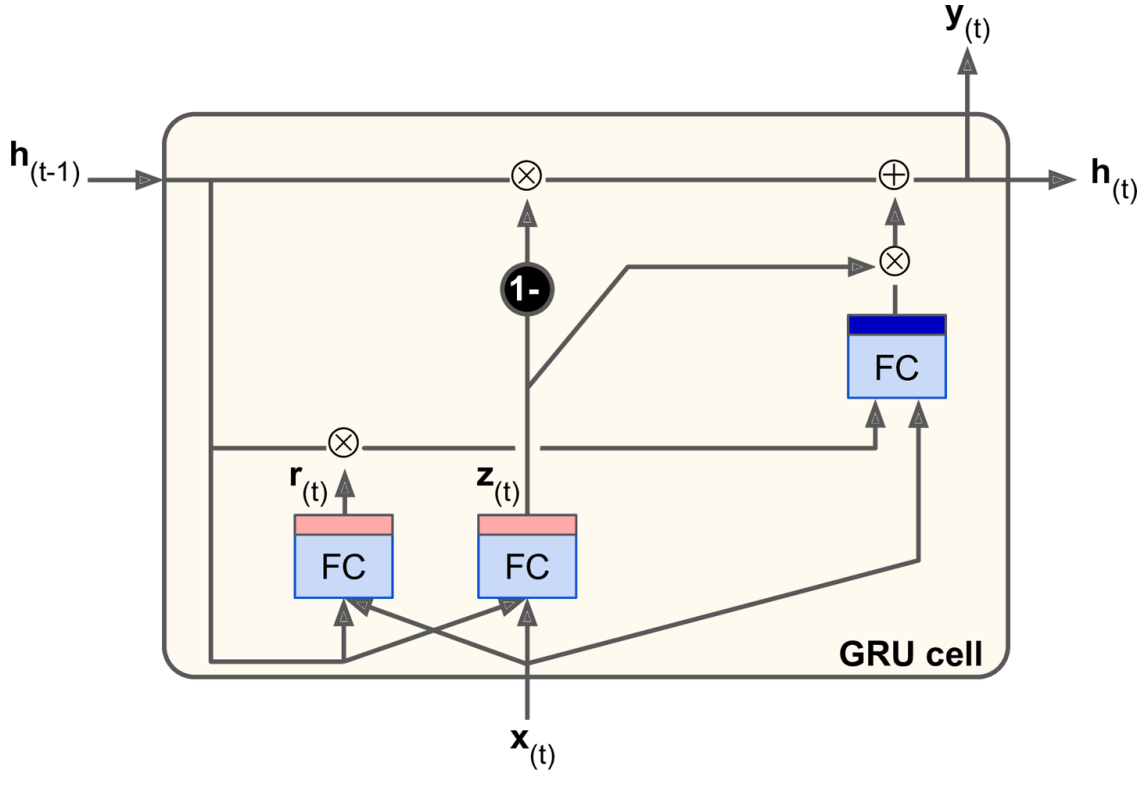


Gated Recurrent Unit Cell (GRU)

- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, K. Cho et al. (2014)
- Simplified version of LSTM cell
- Single state vector
- Gates (reset and update gate)
- Single gate controller (instead of input and forget gate)
  - 1 => the **input gate** is **open**, the **forget gate** is **closed**
  - 0 => the **input gate** is **closed**, the **forget gate** is **open**

- Tensorflow tf.contrib.rnn.GRUCell
- Keras keras.layers.recurrent.GRU

Visualization of GRU Cell



RNN usage and Examples

- Machine Translation
- Automatic Summarization
- Image/Video Captioning
- Sentiment Analysis
- ...

Sum Binary Numbers

- Inspired by Neural Networks for Machine Learning lecture, (Geoffrey Hinton)
- Jupyter notebook with Tensorflow (martinkersner)

```
  1       1
    1 0 0 0 1
+   1 1 1 0 1
-----------------
1 0 1 1 1 0
-----------------
```

Tips

- Make sure you start to feed from the least significant bit :)
- Don't randomly generate training data.

Character-Level Text Generation

- Blog post (Andrej Karpathy)
- Source code (Justin Johnson)

Multilayer recurrent neural network language model with **dropout** regularization. Softmax on the top.

Arguments of LanguageModel:

- `idx_to_token`: A table giving the vocabulary for the language model, mapping integer ids to string tokens.
- `model_type`: "lstm" or "rnn"
- `wordvec_size`: Dimension for word vector embeddings
- `rnn_size`: Hidden state size for RNNs
- `num_layers`: Number of RNN layers to use
- `dropout`: Number between 0 and 1 giving dropout strength after each RNN layer

Latex generation

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariconly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma **??**. Namely, by Lemma **??** we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition* **??** *(without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation* **??**. *Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (**??**). On the other hand, by Lemma **??** we see that

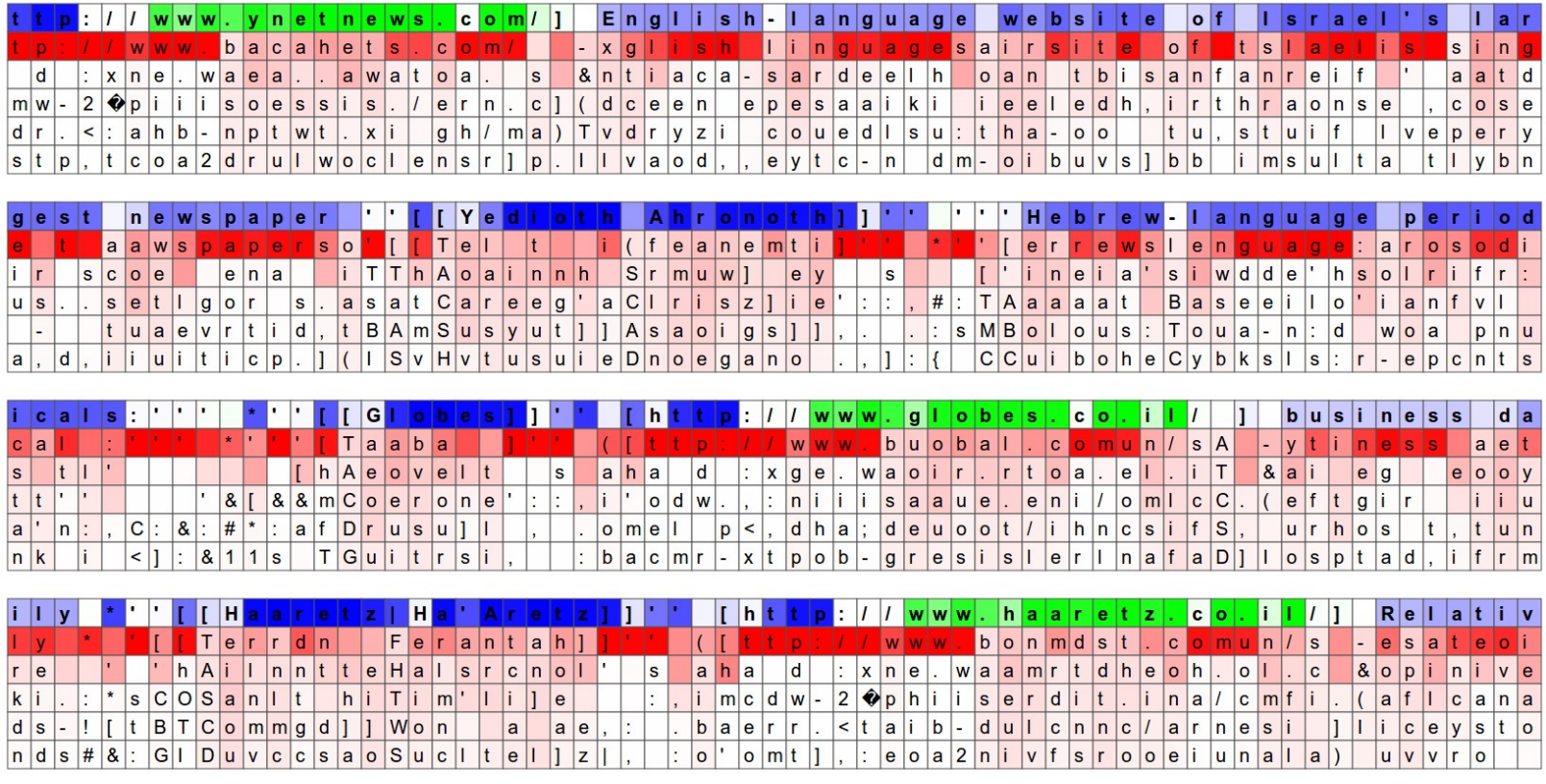
$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

C code generation

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 1
2;
  return segtable;
}
```

Visualization of predictions



Bible generation

- Bible source
- 858,195 words
- torch-rnn
- 50 epochs

Examples

- *Genesis 39:2* And the LORD was with Joseph, and he was a prosperous man; and he was in the house of his master the Egyptian.
- *Numbers 15:41* I am the LORD your God, which brought you out of the land of Egypt, to be your God: I am the LORD your God.
- *Revelation 22:13* I am Alpha and Omega, the beginning and the end, the first and the last.

Generated text

- *Daniel 7:3* Hear now thine hand shall make him all that I will live from an atonement of his three forth.
- *Chronicles 10:22* Then he searched in the land of Abram the Jedaliah, and the Egyptian, and foods of Jerusalem doth oil.
- *Kings 14:17* And their herds of the holy bones, which I shall deliver juspiah of God upon the LORD, that he had sold the destroying of Jerusalem.

thine = yours

atonement = reconciliation

doth = archaic third person singular present of do

***juspiah** does not exist*

QA bAbI tasks

- https://research.fb.com/downloads/babi/
- Synthetic dataset of 20 different tasks for testing text understanding and reasoning.

Example of task with two supporting facts (QA2):

```
1 Mary got the milk there.
2 John moved to the bedroom.
3 Sandra went back to the kitchen.
4 Mary travelled to the hallway.
5 Where is the milk?  hallway 1 4
```

Question Answering Solution Using Keras

http://smerity.com/articles/2015/keras_qa.html

Following information are always related to **Two Supporting Facts (QA2)** which can be found in *tasks_1-20_v1-2/en/qa2two-supporting-facts[train|test].txt*.

- QA2 subdataset contains 1,000 traing and 1,000 testing samples.
- The length of stories and questions **differ**.
- Test accuracy **31 %**, knowing possible answers (6) accuracy of **random** prediction is **16 %**
- Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks, J. Weston et al. (2015)
    - Weakly supervised LSTM, 20 %

Data Preprocessing

```
(# story
 ['Mary', 'got', 'the', 'milk', 'there', '.',
  'John', 'moved', 'to', 'the', 'bedroom', '.',
  'Sandra', 'went', 'back', 'to', 'the', 'kitchen', '.',
  'Mary', 'travelled', 'to', 'the', 'hallway', '.'],
 # question
 ['Where', 'is', 'the', 'milk', '?'],
 # answer
  'hallway')
```

Word vocabulary

Only 35 (36) words!

```
['.', '?', 'Daniel', 'John', 'Mary', 'Sandra', 'Where', 'apple', 'back', 'bathroom',
 'bedroom', 'discarded', 'down', 'dropped', 'football', 'garden', 'got', 'grabbed',
'hallway', 'is', 'journeyed', 'kitchen', 'left', 'milk', 'moved', 'office',
'picked', 'put', 'the', 'there', 'to', 'took', 'travelled', 'up', 'went']
```

Conversion stories to vectors

```
# pre-padded with zeros
[0 ... 5 17 29 24 30  1  4 25 31 29 11  1  6 35  9 31 29 22  1  5 33 31 29 19  1]
```

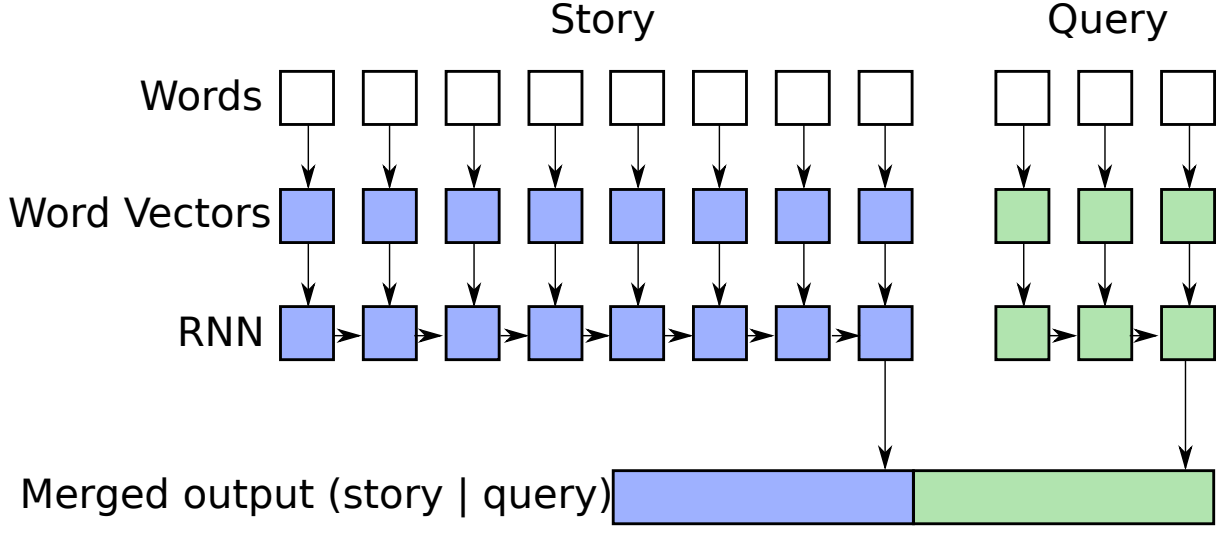
Applied RNN models

Following models can be applied to all bAbI tasks, but have to be trained separately for each task.

Model #1 (August 5, 2015)

```
sentrnn = Sequential()
sentrnn.add(Embedding(vocab_size, EMBED_HIDDEN_SIZE, mask_zero=True))
sentrnn.add(RNN(EMBED_HIDDEN_SIZE, SENT_HIDDEN_SIZE, return_sequences=False))

qrnn = Sequential()
qrnn.add(Embedding(vocab_size, EMBED_HIDDEN_SIZE))
qrnn.add(RNN(EMBED_HIDDEN_SIZE, QUERY_HIDDEN_SIZE, return_sequences=False))

model = Sequential()
model.add(Merge([sentrnn, qrnn], mode='concat'))
model.add(Dense(SENT_HIDDEN_SIZE + QUERY_HIDDEN_SIZE, vocab_size, activation='softma
x'))
```

Architecture



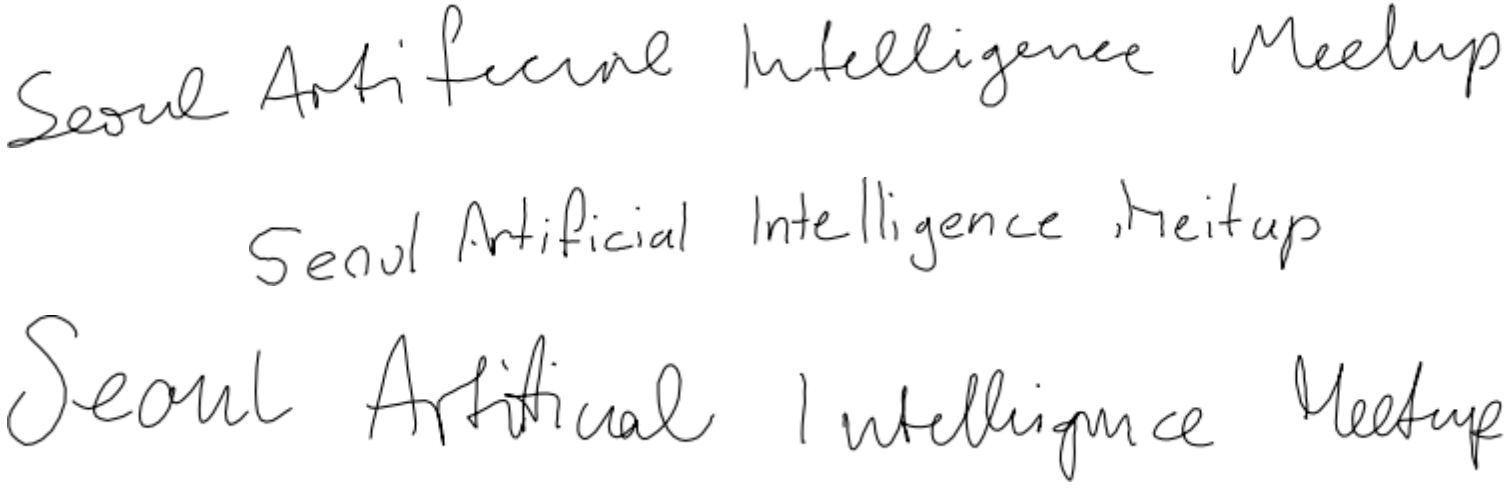
Model #2

- keras.layers.add sum tensors with same dimensions.
- keras.layers.core.Dropout rate: float between 0 and 1. Fraction of the input units to drop.

```
sentence = layers.Input(shape=(story_maxlen,), dtype='int32')
encoded_sentence = layers.Embedding(vocab_size, EMBED_HIDDEN_SIZE)(sentence)
encoded_sentence = layers.Dropout(0.3)(encoded_sentence)

question = layers.Input(shape=(query_maxlen,), dtype='int32')
encoded_question = layers.Embedding(vocab_size, EMBED_HIDDEN_SIZE)(question)
encoded_question = layers.Dropout(0.3)(encoded_question)
encoded_question = RNN(EMBED_HIDDEN_SIZE)(encoded_question)
encoded_question = layers.RepeatVector(story_maxlen)(encoded_question)

merged = layers.add([encoded_sentence, encoded_question])
merged = RNN(EMBED_HIDDEN_SIZE)(merged)
merged = layers.Dropout(0.3)(merged)
preds = layers.Dense(vocab_size, activation='softmax')(merged)
```

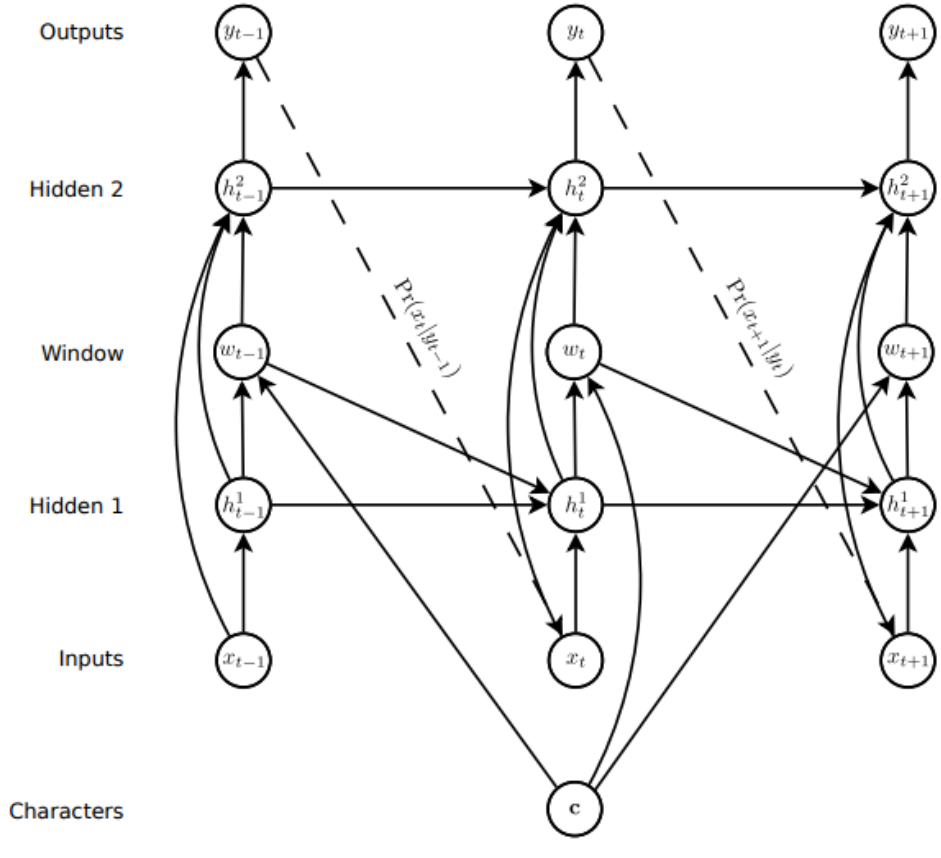
Handwriting Generation

- Generating Sequences With Recurrent Neural Networks, A. Graves, 2015
- Source code
- Online demo



Network Visualization

- Window layer as discrete convolution with a mixture of K Gaussian functions.
- $Pr(x_t|y_{t-1})$ is a multinomial distribution.



Udacity challenge: Prediction of steering angles

- Causal predictions = Only past frames are used to predict the future steering decisions.
- Blog post about winning solutions
- Source code for all winning solutions

1. The first place, Team Komanda solution

    - Mapping from sequences of images to sequences of steering angle measurements.
    - Applied 3D convolution on input image sequences.
    - Then two other layers, **LSTM** and a simple **RNN**, respectively.
    - The predicted angle, torque and speed serve as the input to the next timestep.
2. The third place, Team Chauffeur solution

    - Utilized CNN for feature extraction.
    - Cropped the top of network in order to get 3,000 features.
    - Those features used as input to **LSTM**.


Actual = -0.02793, Predicted = -0.00610

Pixel RNN

- Pixel Recurrent Neural Networks, A. Oord et al. (2016)
- Image inpainting, deblurring, generation
- The network scans the image one row at a time and one pixel at a time within each row. For each pixel it predicts the conditional distribution over the possible pixel values given the scanned context.
- Pixels represented as discrete values using a multinomial distribution implemented with a simple softmax layer.
- 12 **LSTM** layers with residual connections.