Autoregressive Models

Sun Woo Kim[1]

[1]School of Informatics, Computing, and Engineering
Indiana University Bloomington

March 20th, 2018

Outline

Autoregressive Models

- Sequential data with natural order for predictions: images, audio, temporal sequences, etc.
- Try to predict the next term in the input sequence with an advance of n steps [1]
- Autoregressive model: value from a sequence is regressed on previous values from that same sequence

$$X_t = w_{t-1}X_{t-1} + w_{t-2}X_{t-2} + \cdots + w_{t-p}X_{t-p} + c$$

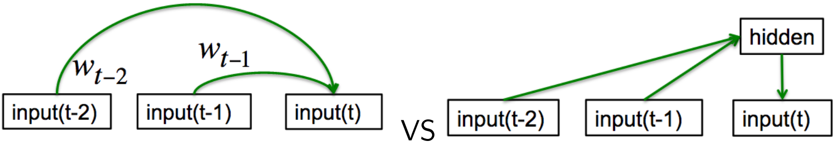


Figure: Autoregressive vs. Feed-forward NN

[1] https://www.cs.toronto.edu/ hinton/csc2535/notes/lec10new.pdf

Outline

Generative Image Modeling (1/2)

- Central problem in unsupervised learning[2]
- Objective: Given some training data, use probabilistic density models to generate new images from the same distribution

Figure 1. Image completions sampled from a PixelRNN.

- Endless amounts of image data available to learn from

[2] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. arXiv preprint arXiv:1601.06759, 2016.

Generative Image Modeling (2/2)

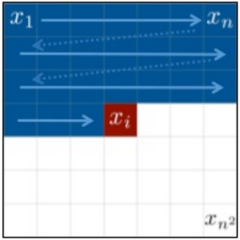
- But images are high dimensional and highly structured
- Need to build complex and expressive models that are also **tractable** and **scalable**
- PixelRNN and PixelCNN are a part of the class of Autoregressive models that fulfill both of these conditions[3]

[3]https://www.slideshare.net/suga93/conditional-image-generation-with-pixelcnn-decoders

Outline

Autoregressive Image Modeling (1/4)

- Scan the image one row at a time and one pixel at a time within each row
- For each pixel sequentially predict the conditional distribution over the possible pixel values given the scanned context[4]



- Joint distribution over the image pixels is factorized into a product of conditional distributions

[4] https://www.slideshare.net/thinkingfactory/pr12-pixelrnn-jaejun-yoo

Autoregressive Image Modeling (2/4)

- Given image $\mathbf{x}$ formed of $n \times n$ pixels, model the conditional distribution of every individual pixel given previous pixels

$$p(\mathbf{x}) = p(x_1, \cdots, x_{n^2}) = \prod_{i=1}^{n^2} p(x_i | x_1, \cdots, x_{i-1})$$

  Write $\mathbf{x}$ as a one-dimensional sequence, where pixels are taken from the image row by row
- Assign probability $p(\mathbf{x})$ to every pixel of the $n \times n$ image
- Sequentially predict pixels instead of predicting the whole image at once (GAN, VAE)

Autoregressive Image Modeling (3/4)

- For color images, each pixel x_i is jointly determined by three values, one for each of the color channels

- The conditional probabiliy of the i-th pixel becomes

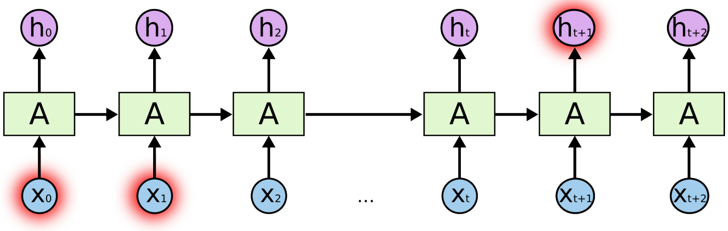
$$p(x_i|\mathbf{x}_{<i}) = p(x_{i,R}|\mathbf{x}_{<i}) \times p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R}) \times p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

- Each color is conditioned on other colors as well as on all the previously generated pixels
- Capture full generality of pixel inter-dependencies and between the RGB color values within each pixel [5]

[5] https://gist.github.com/shagunsodhani/e741ebd5ba0e0fc0f49d7836e30891a7

Autoregressive Image Modeling (4/4)

- Use probabilistic density models to quantify the pixels of an image as a product of conditional distributions
- Turn the modeling problem into sequence problem wherein the next pixel value is determined by all the previously generated pixel values
- Need to process these non-linear and long term dependencies between pixel values and distributions[6]

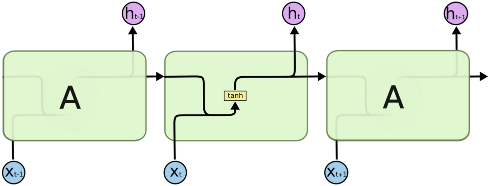


[6] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Outline

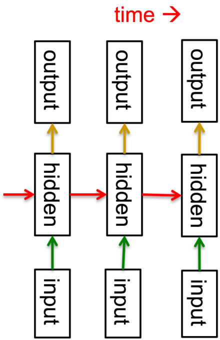
Recurrent Neural Networks (Recap) (1/2)

- Powerful models that offer a compact, shared parameterization of a series of conditional distributions[7]
- Extremely efficient in handling sequence models
- Machine Translation, Speech Recognition, NLP



[7] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

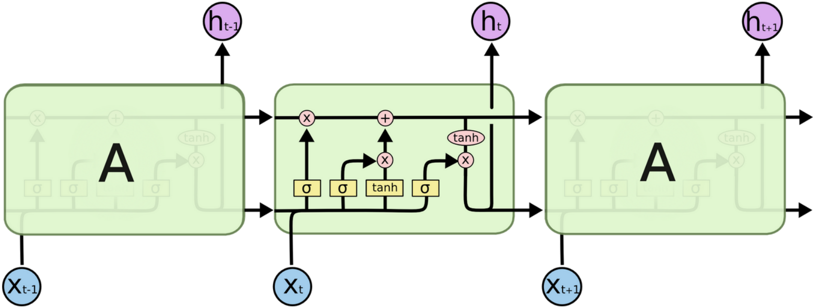
Recurrent Neural Networks (Recap) (2/2)



- Distributed hidden state allows them to store a lot of information about the past efficiently
- Non-linear dynamics that allows them to update their hidden state in complicated ways[8]

[8] https://www.cs.toronto.edu/ hinton/csc2535/notes/lec10new.pdf

Long Short Term Memory Networks (Recap)



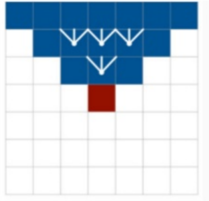
- Performs better at avoiding the vanishing gradient problem[9]
- Can model longer dependencies

[9] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Outline

Row LSTM

- Unidirectional layer that processes the image row by row from top to bottom computing features for a whole row at once
- Computation is performed with a one-dimensional convolution; kernel size $k \geq 3$

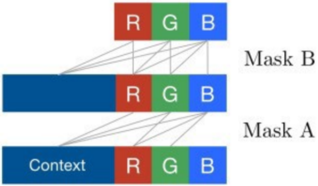


- For a pixel x_i the layer captures a roughly triangular context above the pixel[10]
- Weight sharing in the convolution ensures translation invariance of the computed features along each row

[10] https://blog.acolyer.org/pixelrnn-fig-2-centre-jpeg/

Row LSTM Architecure

- First input layer
- LSTM layer(s)
  - Input-to-state component
  - State-to-state component
- 1x1 Convolution layer(s)
- Final 256-way softmax layer
- Masks:
  - Masked to include only the valid content
  - Two types: A and B[11]



[11] https://github.com/tensorflow/magenta/blob/master/magenta/reviews/pixelrnn.m

Masked Convolutions (1/2)

- Masks enforce certain restrictions on the connections in the network
- Mask A
  - Applied only to the first convolution layer
  - Restricts connections to only those neighbouring pixels and color channels that have already been seen

1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

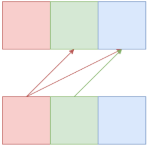
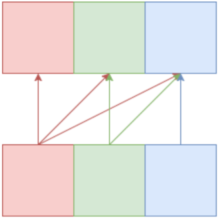


Figure: Mask A for pixels and channels[12]

[12]https://towardsdatascience.com/summary-of-pixelrnn-by-google-deepmind-7-min-read-938d9871d6d9

Masked Convolutions (2/2)

- Mask B
  - Applied to all subsequent input-to-state convolutional transitions
  - Relaxes restrictions of mask A by also allowing the connection from a color to itself



  - Ensures the image has R,G,B values throughout the network[13]

[13] https://towardsdatascience.com/summary-of-pixelrnn-by-google-deepmind-7-min-read-938d9871d6d9

Row LSTM Implementation: First layer (1/2)

- First layer is a 7x7 masked convolution

```
inputs = tf.placeholder(tf.float32, [None, height, width, channel])

weights_shape = (kernel_d, kernel_d, channel, num_outputs)
weights = tf.get_variable("weights", weights_shape, tf.float32,
                tf.contrib.layers.xavier_initializer())

maskA = get_mask_A(kernel_d, weights_shape)
weights_masked = weights * tf.constant(maskA, dtype=tf.float32)

conv1 = tf.nn.conv2d(
            inputs, weights_masked, [1, stride, stride, 1], "SAME")
```

Row LSTM Implementation: First layer (2/2)

- Mask A

```python
def get_mask_A(kernel_dim, shape):
    mask = np.ones(shape, np.float32)
    center = kernel_dim // 2
    mask[center,   center:, :,:] = 0.
    mask[center+1:,:       , :,:] = 0.
    return mask
```

Row LSTM Implementation: LSTM layer

- Input-to-state
  - 3x1 convolution that uses mask of type B

```
batch, width, height, channel = l_hid.get_shape().as_list()

weights_shape = [kernel_w, kernel_h, channel, num_outputs]
weights = tf.get_variable("weights", weights_shape,
            tf.float32, tf.contrib.layers.xavier_initializer())

mask = get_type_B(kernel_h, kernel_w, weights_shape)

input_to_state = tf.nn.conv2d(l_hid, weights * mask, [1,1,1,1],
                    padding="SAME", name='outputs')
```

Row LSTM Implementation: LSTM layer

- Mask B

```
def get_mask_B(kernel_w, kernel_h, shape):
    mask = np.ones(shape, np.float32)
    center_w = kernel_w // 2
    center_h = kernel_h // 2
    mask[center_w, center_h+1:, :, :] = 0.
    mask[:, center_h+1:, :, :] = 0.
    return mask
```

Row LSTM Implementation: LSTM layer

- State-to-state
  - 3x1 state-to-state convolution unmasked

```
batch, width, height, channel = get_shape(input_to_state)

rnn_inputs = tf.reshape(input_to_state,
                        [-1, height, width * channel])
cell = RowLSTMCell(hidden_dims, width, channel)
outputs, states = tf.nn.dynamic_rnn(cell, inputs=rnn_inputs,
                                    dtype=tf.float32)

outputs = tf.reshape(outputs, [-1, height, width, hidden_dims])
```

Row LSTM Implementation: LSTM layer

- RowLSTMCell

```
class RowLSTMCell(rnn_cell.RNNCell):
  def __init__(self, hidden_dims, width):
    self._width = width
    self._hidden_dims = hidden_dims
    self._num_units = self._hidden_dims * self._width

  def __call__(self, i_to_s, state):
    # ..............................
```

Row LSTM Implementation: LSTM layer

```
class RowLSTMCell(rnn_cell.RNNCell):
  def __init__(self, hidden_dims, width):
    # ...................................

  def __call__(self, i_to_s, c_prev, h_prev):
    batch, width, height, channel = h_prev.get_shape().as_list()
    num_outputs = 4 * self._hidden_dims
    kernel_w, kernel_h = 3, 1

    weights_shape = [kernel_w, kernel_h, channel, num_outputs]
    weights = tf.get_variable("weights", weights_shape,
                tf.float32, tf.contrib.layers.xavier_initializer())

    conv_s_to_s = tf.nn.conv2d(h_prev,
                    weights, [1,1,1,1], padding="SAME", name='s_to_s')

    s_to_s = tf.reshape(conv_s_to_s,
                [-1, self._width * self._hidden_dims * 4])
    lstm_matrix = tf.sigmoid(s_to_s + i_to_s)

    i, g, f, o = tf.split(lstm_matrix, 4, 1)
    c = f * c_prev + i * g
    h = tf.multiply(o, tf.tanh(c), name='hid')
    return c, h
```

Row LSTM Implementation: LSTM layer

- State-to-state (Recap)
  - 3x1 state-to-state convolution unmasked

```
batch, width, height, channel = get_shape(input_to_state)

rnn_inputs = tf.reshape(input_to_state,
                [-1, height, width * channel])
cell = RowLSTMCell(hidden_dims, width, channel)
outputs, states = tf.nn.dynamic_rnn(cell, inputs=rnn_inputs,
                    dtype=tf.float32)

outputs = tf.reshape(outputs, [-1, height, width, hidden_dims])
```

Row LSTM Implementation: LSTM layer

- LSTM layer: main recurrent layers

```
l_hid = First_Layer(inputs)
for idx in range(recurrent_length):
    l_hid = Input_to_State(l_hid)
    l_hid = State_to_State(l_hid)
```

Row LSTM Implementation: Output Recurrent layer

- The feature map is then passed through a couple of 1x1 convolution layers consisting of ReLU and mask type B

```
batch, width, height, channel = l_hid.get_shape().as_list()
kernel_d = 1

weights_shape = [kernel_d, kernel_d, channel, num_outputs]
weights = tf.get_variable("weights", weights_shape,
            tf.float32, tf.contrib.layers.xavier_initializer())
mask = get_mask_B(kernel_d, kernel_d, weights_shape)

l_hid = tf.n.conv2d(l_hid, weights * mask, [1,1,1,1], padding="SAME")
l_hid = tf.nn.relu(l_hid)
```

Row LSTM Implementation: Output Recurrent layer

- With addition of the output recurrent layer

```
l_hid = First_Layer(inputs)
for idx in range(recurrent_length):
    l_hid = Input_to_State(l_hid)
    l_hid = State_to_State(l_hid)
for idx in range(out_recurrent_length):
    l_hid = Out_Recurrent(l_hid)
```

Row LSTM Implementation: Final layer

- 256-way softmax layer

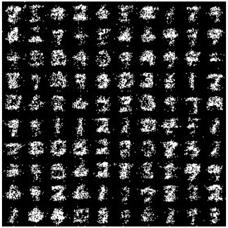
```
batch, width, height, channel = l_hid.get_shape().as_list()
kernel_d = 1
num_outputs = 1

weights_shape = [kernel_d, kernel_d, channel, num_outputs]
weights = tf.get_variable("weights", weights_shape,
            tf.float32, tf.contrib.layers.xavier_initializer())
mask = get_mask_B(kernel_d, kernel_d, weights_shape)

l_hid = tf.n.conv2d(l_hid, weights * mask, [1,1,1,1], padding="SAME")
output = tf.nn.sigmoid(l_hid)
```

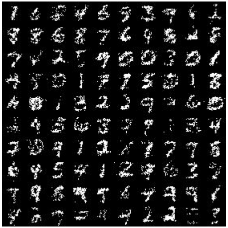
Row LSTM Implementation: Final layer

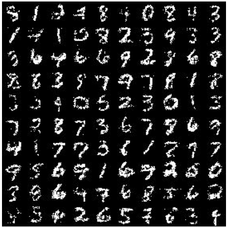
- With addition of the last layer

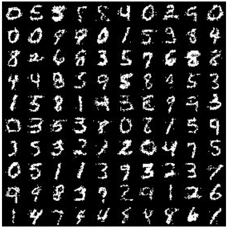
```
l_hid = First_Layer(inputs)
for idx in range(recurrent_length):
    l_hid = Input_to_State(l_hid)
    l_hid = State_to_State(l_hid)
for idx in range(out_recurrent_length):
    l_hid = Out_Recurrent(l_hid)
output = Final_Softmax(l_hid)

loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(
            logits=output, labels=inputs))

optimizer = tf.train.AdamOptimizer(learning_rate)
ops = optimizer.minimize(loss)
```

Outputs





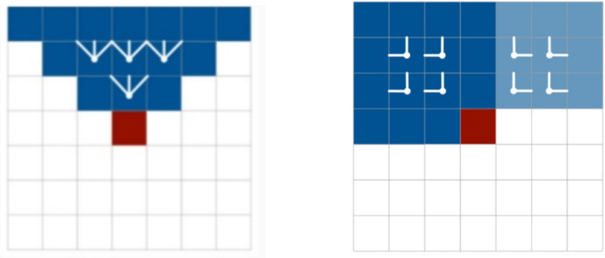




Outline

Diagonal BiLSTM

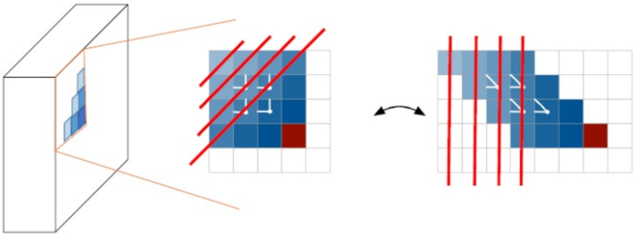
- Diagonal BiLSTM's dependency field covers the entire available context in the image



- Bidirectional layer that processes the image in a diagonal fashion

Diagonal BiLSTM

- Differences in architecture
  - 1x1 convolution input-to-state layer
  - 1x2 convolution state-to-state layer
- Each step in the computation computes at once the LSTM state along a diagonal in the image
- To optimize, the feature map is skewed so it can be parallelized



Diagonal BiLSTM Implementation

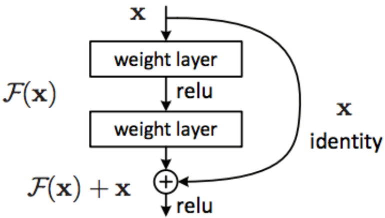
- Changes
  - Addition of skew operations
  - 1x1 IS and 1x2 SS convolutions in LSTM layers

```
l_hid = First_Layer(inputs)
l_hid = skew(l_hid)
for idx in range(recurrent_length):
    l_hid = Input_to_State(l_hid)
    l_hid = State_to_State(l_hid)
l_hid = unskew(l_hid)
for idx in range(out_recurrent_length):
    l_hid = Out_Recurrent(l_hid)
output = Final_Softmax(l_hid)

loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(
            logits=output, labels=inputs))

optimizer = tf.train.AdamOptimizer(learning_rate)
ops = optimizer.minimize(loss)
```

Outline

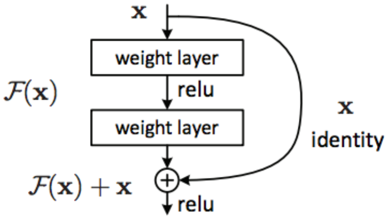
Residual and Skip Connections (1/3)

- Degradation: As network depth increases, accuracy gets saturated and degrades rapidly
- Solution: Copy the layer from the learned shallower model, and add them with identity mapping



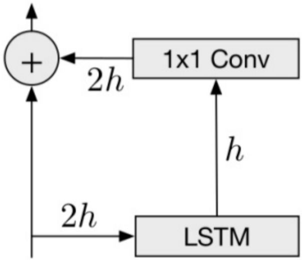
- Deeper model should produce no higher training error than the shallower model

Residual and Skip Connections (2/3)



- Assumption: Optimal function being modelled is closer to an identity mapping than a zero mapping
- Simplifies Optimization: Easier to find the perturbations with reference to an identity mapping than to a zero mapping
- Subsequent blocks fine-tune the output of a previous block, instead of generating the desired output from scratch

Residual and Skip Connections (3/3)

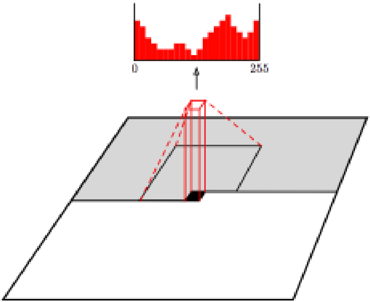


- Deep network: PixelRNN 12 layers
- Residual connection increase convergence speed and propagate

Outline

Motivation

- Row and Diagonal LSTM layers have long range dependencies in the images
- Also, each state needs to be computed sequentially making the training slow
- Standard convolutional layers can capture a bounded receptive field and compute features for all pixel positions at once



Outline