Neural Networks and Deep Learning Lecture 11

Wei WANG

cs5242@comp.nus.edu.sg

NUS National University of Singapore

School of Computing

Administrative

- Saturday session for quiz and assignment 2 on April 14, 2-4PM.
  - COM1-0206
- Top-3 teams will be invited to share their solution on Week 13.

Recap

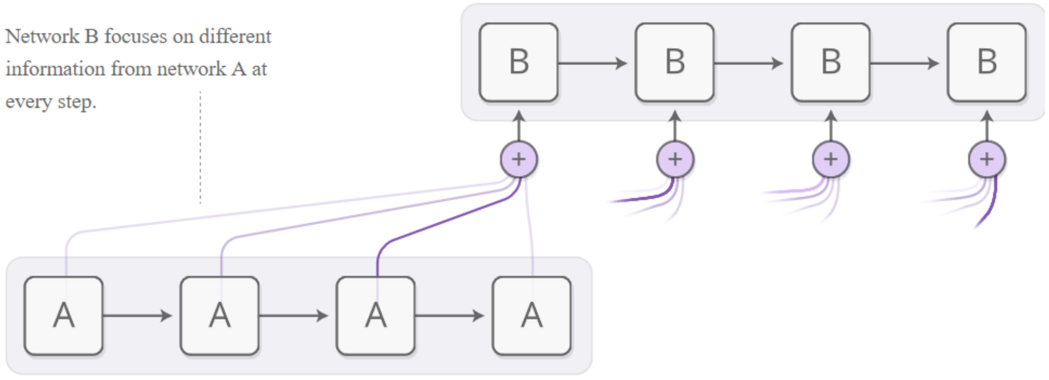
RNN applications

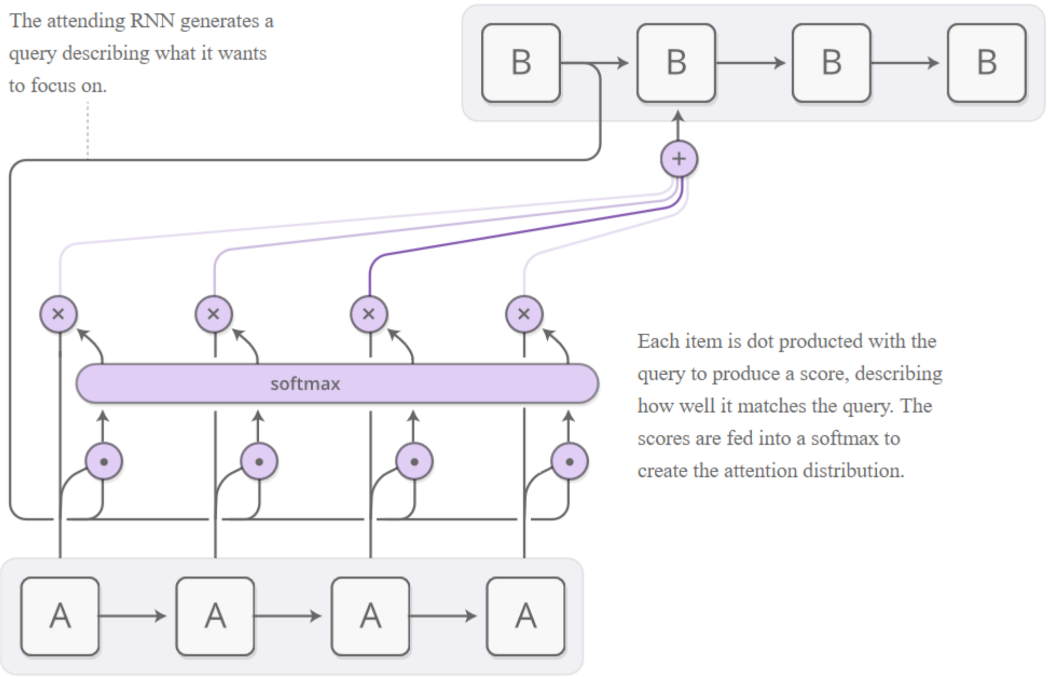
- Bi-directional RNN
- Image caption generation
- Machine translation
- Question answering

Hands-on Tutorials

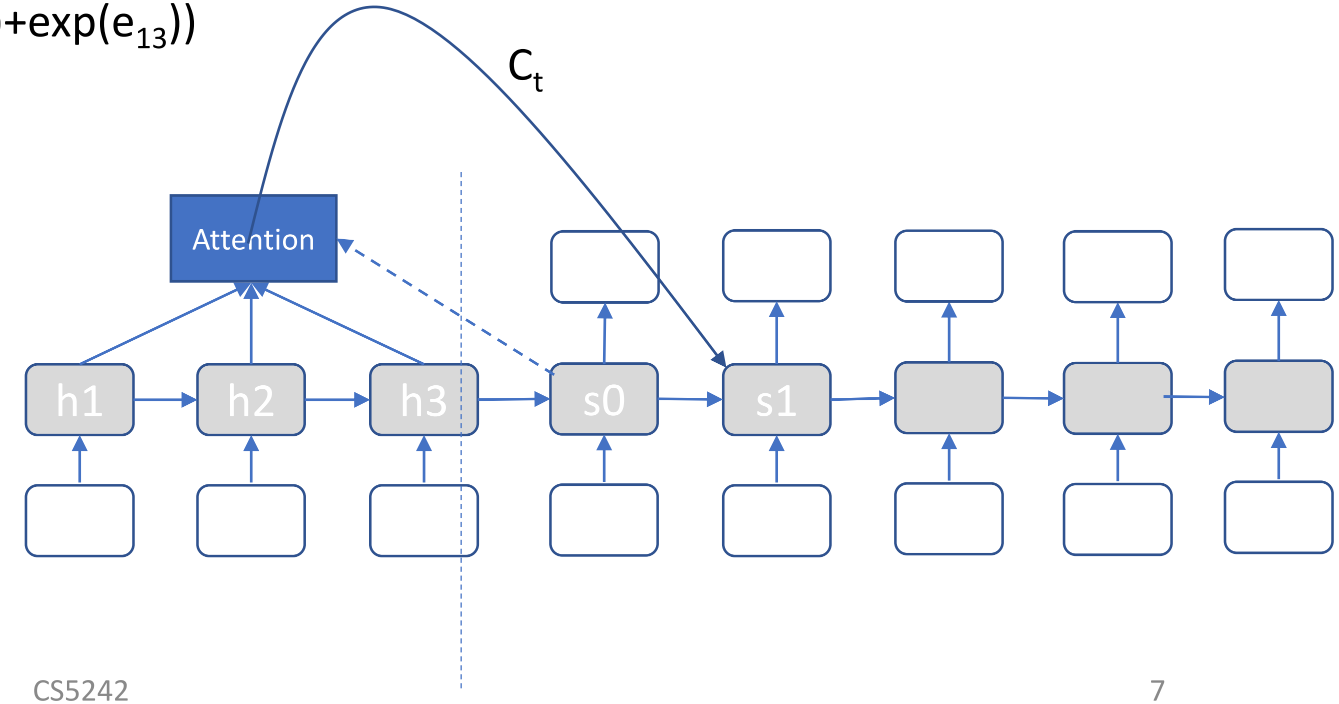
Seq2Seq with attention modelling for machine translation

- Recurrent neural networks for sequence to sequence modelling
  - https://distill.pub/2016/augmented-rnns/#attentional-interfaces





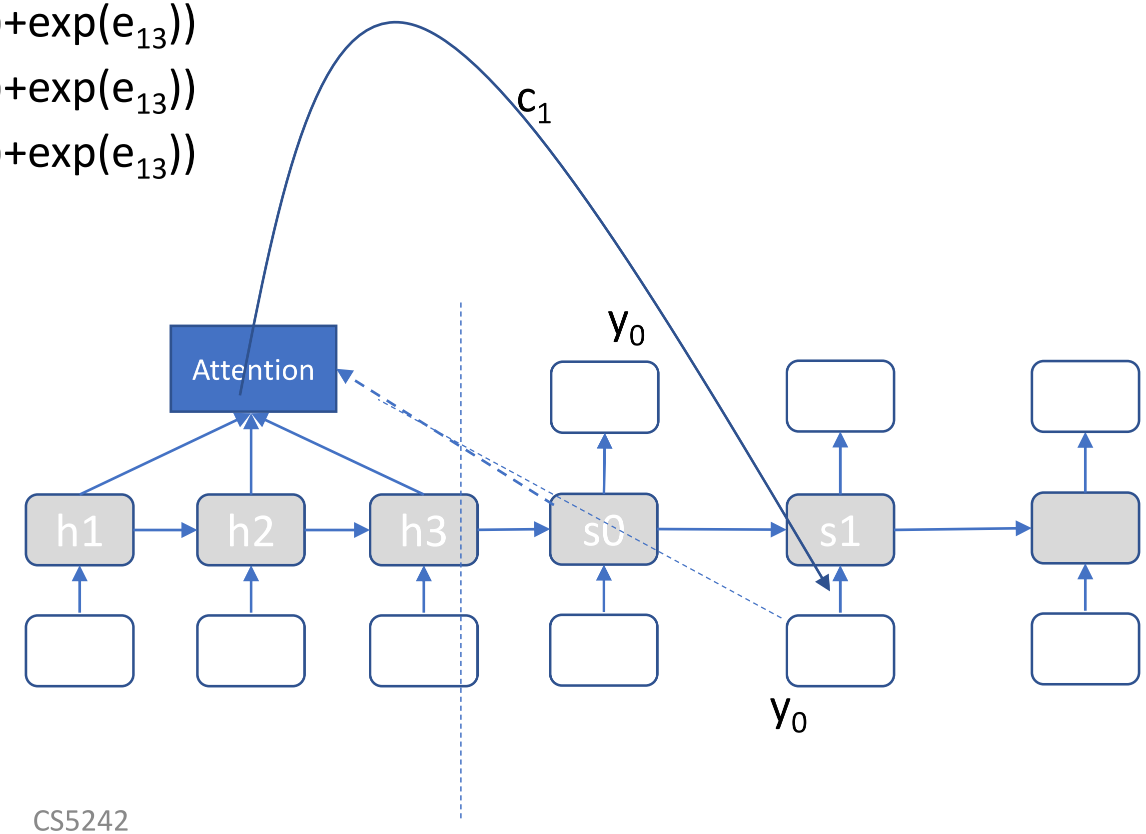
- Attention modelling I
  - Given hidden state vector s_0
    - To compute the weights of h1, h2, h3 for computing s1
      - $e_{11} = a(s_0, h_1) = v^T \tanh(W_a s_0 + U_a h_1)$
      - $e_{12} = a(s_0, h_2) = v^T \tanh(W_a s_0 + U_a h_2)$
      - $e_{13} = a(s_0, h_3) = v^T \tanh(W_a s_0 + U_a h_3)$
      - $\alpha_{11} = \exp(e_{11}) / (\exp(e_{11})+\exp(e_{12})+\exp(e_{13}))$
      - $\alpha_{12} = \exp(e_{12}) / (\exp(e_{11})+\exp(e_{12})+\exp(e_{13}))$
      - $\alpha_{13} = \exp(e_{13}) / (\exp(e_{11})+\exp(e_{12})+\exp(e_{13}))$
      - $c_1=\alpha_{11}h_1+\alpha_{12}h_2+\alpha_{13}h_3$
  - $y_1 = GRU(s_0, [y_0; c_1])$
    - $[y_0; c_1]$, concatenate features
  - Parameters: {v, W_a, U_a}



- Attention modelling II
  - Given hidden state vector s_0
    - To compute the weights of h1, h2, h3 for computing s1
      - $e_{11}=a(s_0, h_1) = v_1^T[s_0; y_0]$
      - $e_{12} = a(s_0, h_2) = v_2^T [s_0; y_0]$
      - $e_{13} =a(s_0, h_3) = v_3^T [s_0; y_0]$

      MLP with concatenated input $[s_0; y_0]$

      - $\alpha_{11} = \exp(e_{11}) / (\exp(e_{11})+\exp(e_{12})+\exp(e_{13}))$
      - $\alpha_{12} = \exp(e_{12}) / (\exp(e_{11})+\exp(e_{12})+\exp(e_{13}))$
      - $\alpha_{13} = \exp(e_{13}) / (\exp(e_{11})+\exp(e_{12})+\exp(e_{13}))$
      - $c_1=\alpha_{11}h_1+\alpha_{12}h_2+\alpha_{13}h_3$
  - $y_1=GRU(s_0, W[c_1;y_0])$
  - Parameters: $\{v_1, v_2, v_3, W\}$



Download the jupyter notebook from IVLE.
The code on fast.ai github has bugs.

```python
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, n_layers=1, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        # for generting the attention weights using previous hidden state and new data as the input
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        # to transform the combined feature of new data + attention info from the encoder
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)
```

Using attention modelling II

```python
    def forward(self, input, hidden, encoder_output, encoder_outputs):
        embedded = self.embedding(input)
```

Input is a matrix of word index, shape: batch_size * max_length

```python
        embedded = self.dropout(embedded)
        attn_weights = F.softmax(self.attn(torch.cat((embedded, hidden[0]), 1)))
```

MLP with concatenated input $[s_0; y_0]$ + Softmax to get α

```python
        attn_applied = torch.bmm(attn_weights.unsqueeze(1), encoder_outputs)
```

$c_1=\alpha_{11}h_1+\alpha_{12}h_2+\alpha_{13}h_3$

```python
        output = torch.cat((embedded, attn_applied.squeeze(1)), 1)
        output = self.attn_combine(output).unsqueeze(0)
```

$W[c_1;y_0]$

```python
        for i in range(self.n_layers):
            output = F.relu(output)
            output, hidden = self.gru(output, hidden)
```

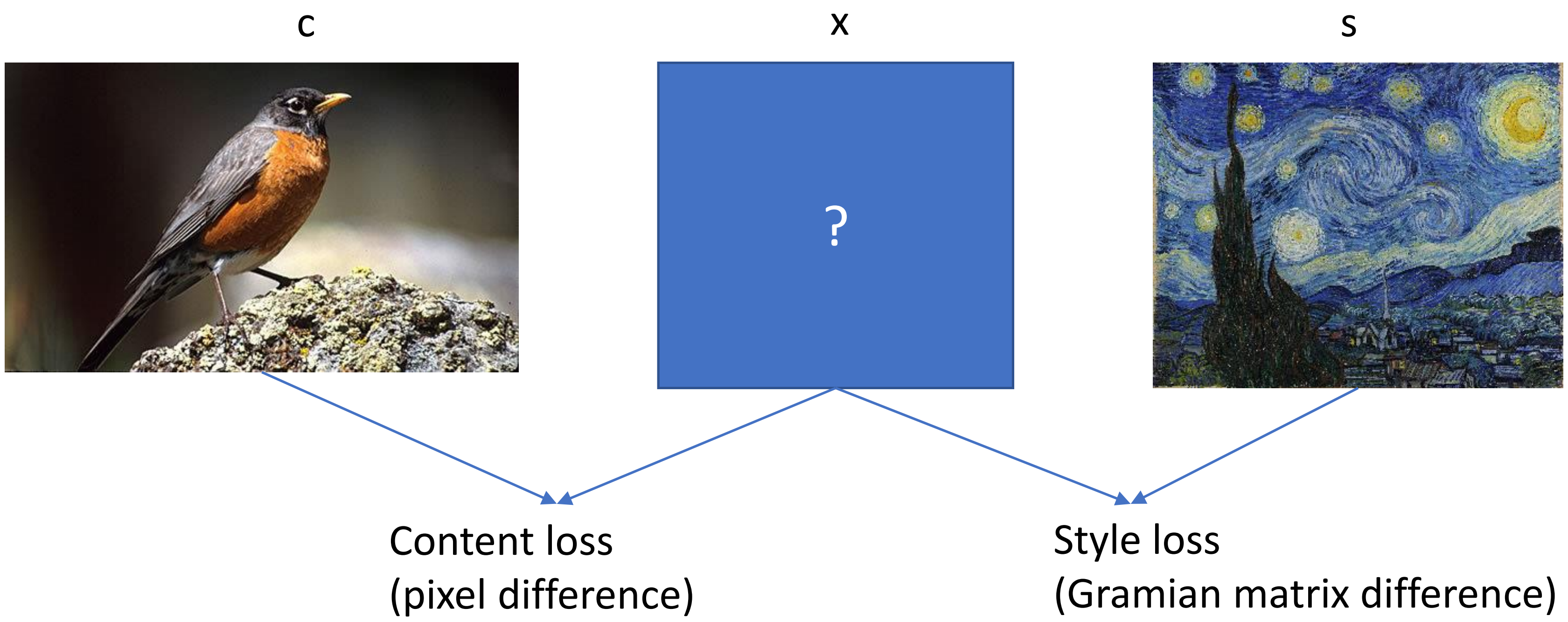
Output shape: (1, batch size, hidden size)

```python
        output = F.log_softmax(self.out(output[0]))
        return output, hidden, attn_weights
```

Output shape: (batch size, output size)

Neural style transfer

- https://github.com/fastai/courses/blob/master/deeplearning2/neural-style.ipynb



Content Loss

- Denote the feature maps from one conv layer of VGG (or other ConvNets) as f(x) for input x; f is fixed (parameters are fixed)
- Lcontent = $||f(x) – f(c)||^2$
  - c is the reference image; x is the image to be generated
  - Update x to make the content loss smaller
- Optimize x to make Lcontent small based on gradient dLcontent/dx
  - Optimizer, SGD or LBFGS

```
model = VGG16_Avg(include_top=False)
```

Here we're grabbing the activations from near the end of the convolutional model).

```
layer = model.get_layer('block5_conv1').output
```

And let's calculate the target activations for this layer:

```
layer_model = Model(model.input, layer)
targ = K.variable(layer_model.predict(img_arr))
```

Keras functional API to create a model;
Extract feature of the reference image

```
loss = metrics.mse(layer, targ)
grads = K.gradients(loss, model.input)
```

targ is fixed (the feature of the reference image)
Gradient dLcontent/dx

Style loss

- Denote the feature maps from one conv layer of VGG (or other ConvNets) as f(x) for input x; f is fixed (parameters are fixed)
- Reshape f(x) to merge the height and width dimension
  - f(x): (channel, (height x width))
  - Gramian matrix: $G(x)=dot(f(x), f(x)^T)$

```
def gram_matrix(x):
    # We want each row to be a channel, and the columns to be flattened x,y locations
    features = K.batch_flatten(K.permute_dimensions(x, (2, 0, 1)))
    # The dot product of this with its transpose shows the correlation
    # between each pair of channels
    return K.dot(features, K.transpose(features)) / x.get_shape().num_elements()
```

- Lstyel = $||G(x) – G(s)||^2$
- Optimize x to make Lstyle small based on gradient dLstyle/dx
  - Optimizer, SGD or LBFGS

```python
def style_loss(x, targ): return metrics.mse(gram_matrix(x), gram_matrix(targ))
```

```python
loss = sum(style_loss(l1[0], l2[0]) for l1,l2 in zip(layers, targs))
grads = K.gradients(loss, model.input)
style_fn = K.function([model.input], [loss]+grads)
evaluator = Evaluator(style_fn, shp)
```

Compare the features from multiple conv layers to calculate the style difference

Content loss + style loss

```
style_wgts = [0.05,0.2,0.2,0.25,0.3]
```

Weights of different layers for style loss aggregation

```
loss = sum(style_loss(l1[0], l2[0])*w
           for l1,l2,w in zip(style_layers, style_targs, style_wgts))
loss += metrics.mse(content_layer, content_targ)/10
grads = K.gradients(loss, model.input)
transfer_fn = K.function([model.input], [loss]+grads)
```

Style loss aggregated from a couple of layers with different weight

Content loss