AI研习社

CS231n Assignment3

王煦中
知乎专栏：喵神大人的深度工坊

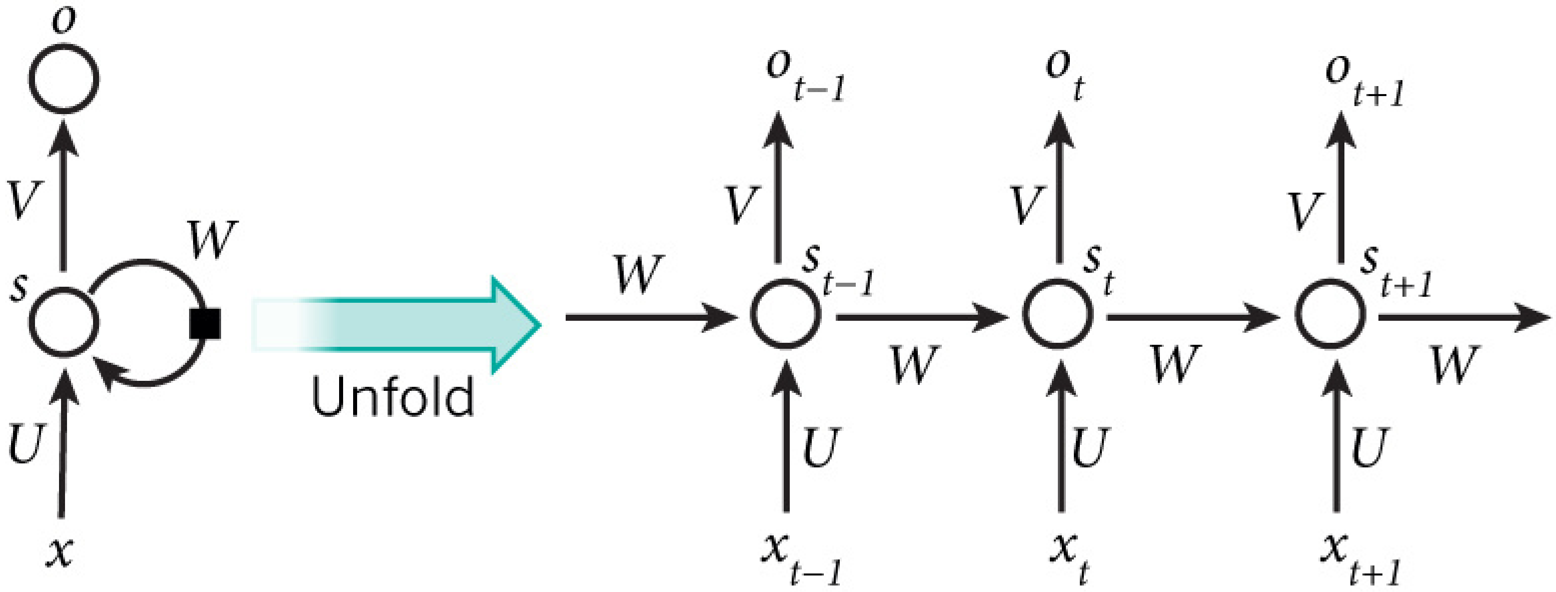
Catalogue

AI研习社

- Part 1 Image Captioning with Vanilla RNNs
- Part 2 Image Captioning with LSTMs
- Part 3 Network Visualization
- Part 4 Style Transfer
- Part 5 Generative Adversarial Networks(GAN)

Part 1 Image Captioning with Vanilla RNNs

AI研习社



➢Vanilla RNN 单元(前向):

- (右箭头) h_{t}从左到右的传播过程 $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$
- (上箭头) h_{t}temporal softmax输出层 $y_t = softmax(W_y * h_t + b_y)$

```
next_h = np.tanh(x.dot(Wx) + prev_h.dot(Wh) + b)
cache = (x, prev_h, Wh, Wx, b, next_h)
```

Part 1 Image Captioning with Vanilla RNNs

AI研习社

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$

➢Vanilla RNN 单元(反向)：上游导数记为dout

- db = sum(**dout** * (1 – tanh^2(x)))
- dWx = **dout** * (1 – tanh^2(x)) **dot** x
- dWh = **dout** * (1 – tanh^2(x)) **dot** h
- dx = **dout** * (1 – tanh^2(x)) **dot** Wx
- dh = **dout** * (1 – tanh^2(x)) **dot** Wh

```
(x, prev_h, Wh, Wx, b, next_h) = cache
dtheta = dnext_h * (1 - next_h ** 2) #(N,H)
db = np.sum(dtheta, axis=0)
dWx = x.T.dot(dtheta) #(D,N) * (N,H)
dWh = prev_h.T.dot(dtheta) #(H,N) * (N,H)
dprev_h = dtheta.dot(Wh.T) # (N,H) * (H,H)= (N,H)
dx = dtheta.dot(Wx.T) # (N,H)*(H,D)=(N,D)
```

Part 1 Image Captioning with Vanilla RNNs

AI研习社

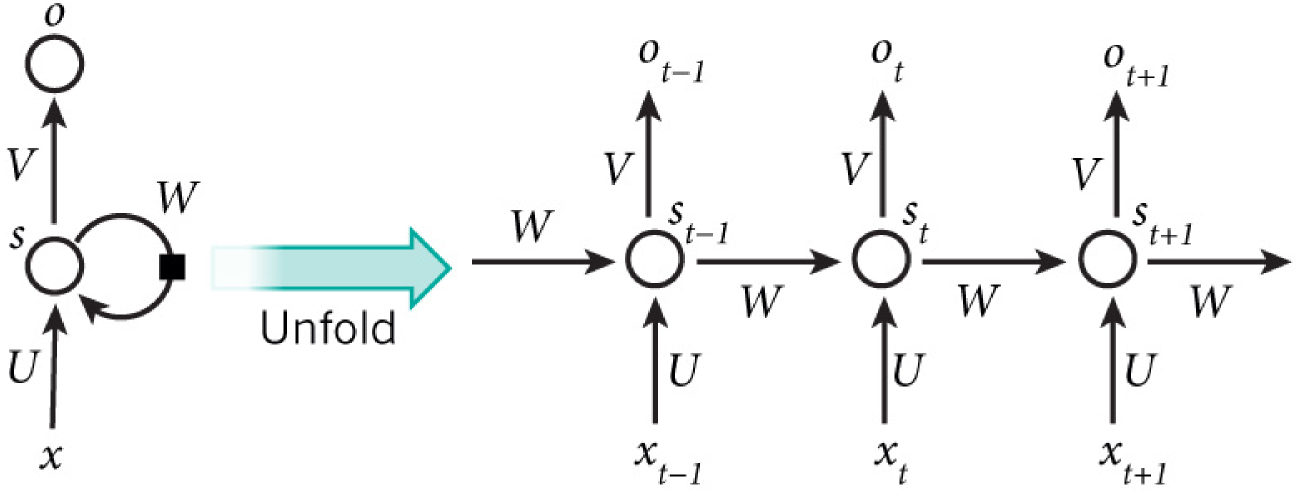
➢完整 Vanilla RNN (前向):

- 对RNN单元循环T次，T是序列的长度 $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$

```python
prev_h = h0
h = np.zeros((N, T, H))
for t in range(T):
    xt = x[:, t, :]
    next_h, cache = rnn_step_forward(xt, prev_h, Wx, Wh, b)
    h[:, t, :] = next_h
    prev_h = next_h
```

Part 1 Image Captioning with Vanilla RNNs

AI研习社



➢完整 Vanilla RNN (反向):

- 一部分来自右边记为dprev_h
- 一部分来自上面记为dh
- 首先逆序：

```
for t in range(T-1, -1, -1):
```

- 对于每一个时间片t，上面来的导数是dh[:, t, :]（形状是(N, T, H)），对于最后一个单元，它的右边没有传来导数，所以初始化dprev_h是0。
- 于是对于每一个RNN单元，需要传进去的导数就是dh[:, t, :]+dprev_h。

```
cache = (xt, prev_h, Wh, Wx, b, next_h)
dnext_h = dh[:, t, :] + dprev_h#上面+右边,初始右边=0
dx[:, t, :], dprev_h, dWxt, dWht, dbt = rnn_step_backward(dnext_h, cache)
```

- dWx，dWh，db和dh这四个参数在不同时刻是共享的，因此需要在每一个时刻把它们都加起来做更新。

```
dWx += dWxt #不同时刻共享的
dWh += dWht
db += dbt
dh0 = dprev_h
```

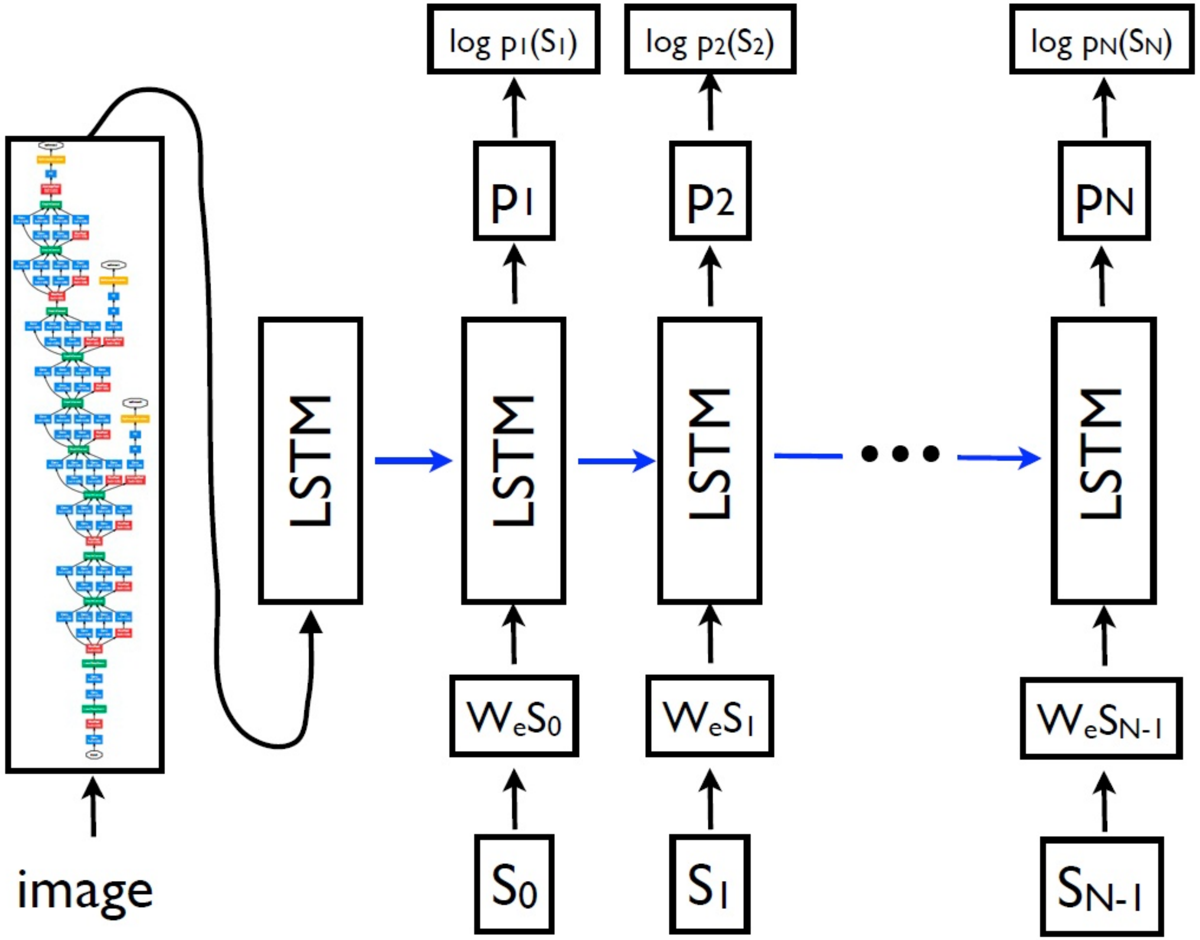
Part 1 Image Captioning with Vanilla RNNs

AI研习社

➢Image Captioning

- 隐藏层：从训练好的vgg16的fc7层中取出特征，当做h0输入到RNN中
- 单词的表示：
- Word embedding，把onehot编码的单词映射为一个向量
- Word embedding(反向)：类似relu、dropout的反向

```
dW = np.zeros(W.shape)
np.add.at(dW, x, dout)#按照x的下标给出dout
```



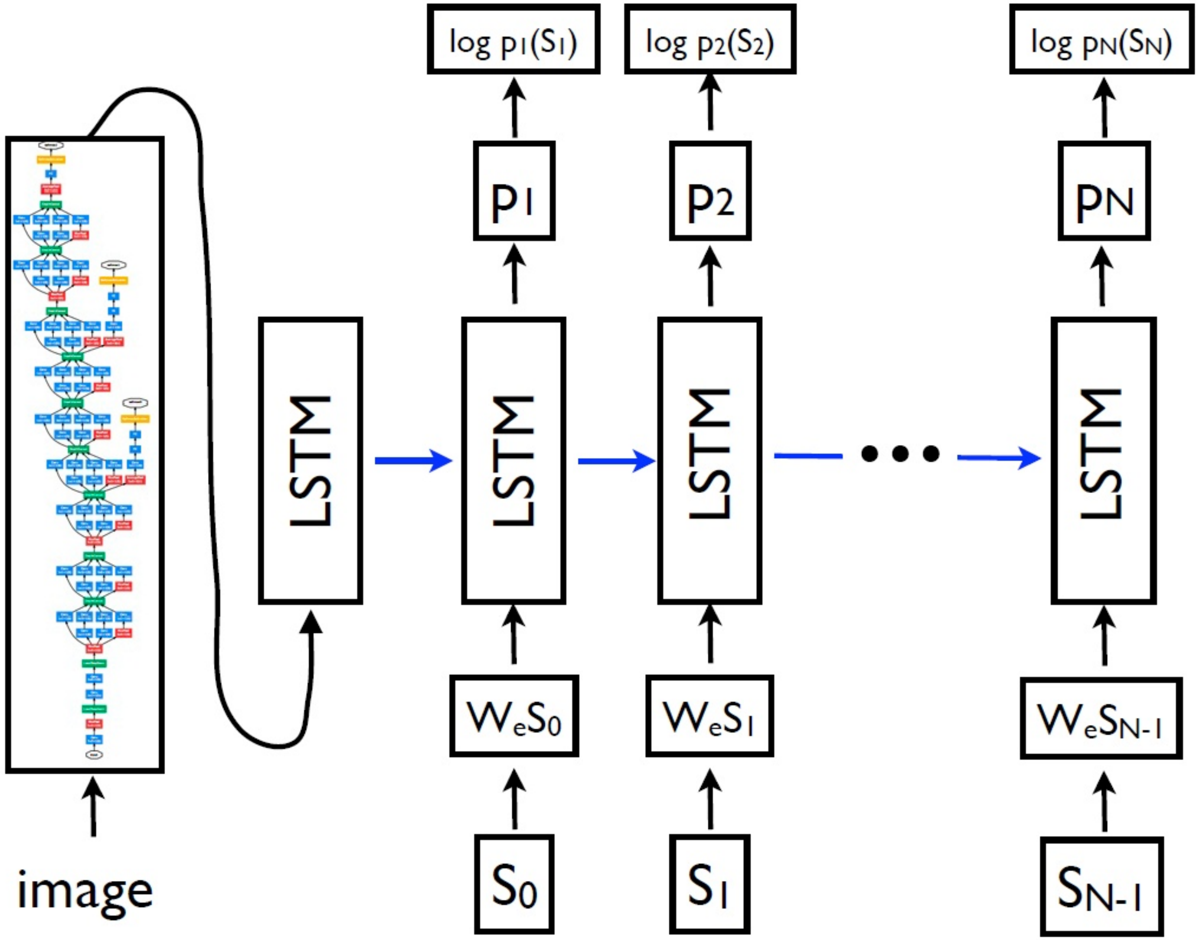
Part 1 Image Captioning with Vanilla RNNs

AI研习社

➢Image Captioning 训练

- 下一个词当做当前词的label。captions的第一个词是start标记，最后一个词是end标记。

```
# forward
# affine hidden state(N,D) -> (N,H)
affine, cache_affine = affine_forward(features, W_proj, b_proj)
# word embedding
embed, chahe_embed = word_embedding_forward(captions_in, W_embed)
# rnn
if self.cell_type == 'rnn':
    h, cache_rnn = rnn_forward(embed, affine, Wx, Wh, b)
elif self.cell_type == 'lstm':
    h, cache_rnn = lstm_forward(embed, affine, Wx, Wh, b)
# temporal_affine
h_affine, cache_h_affine = temporal_affine_forward(h, W_vocab, b_voca
# softmax
loss, dx = temporal_softmax_loss(h_affine, captions_out, mask)
```



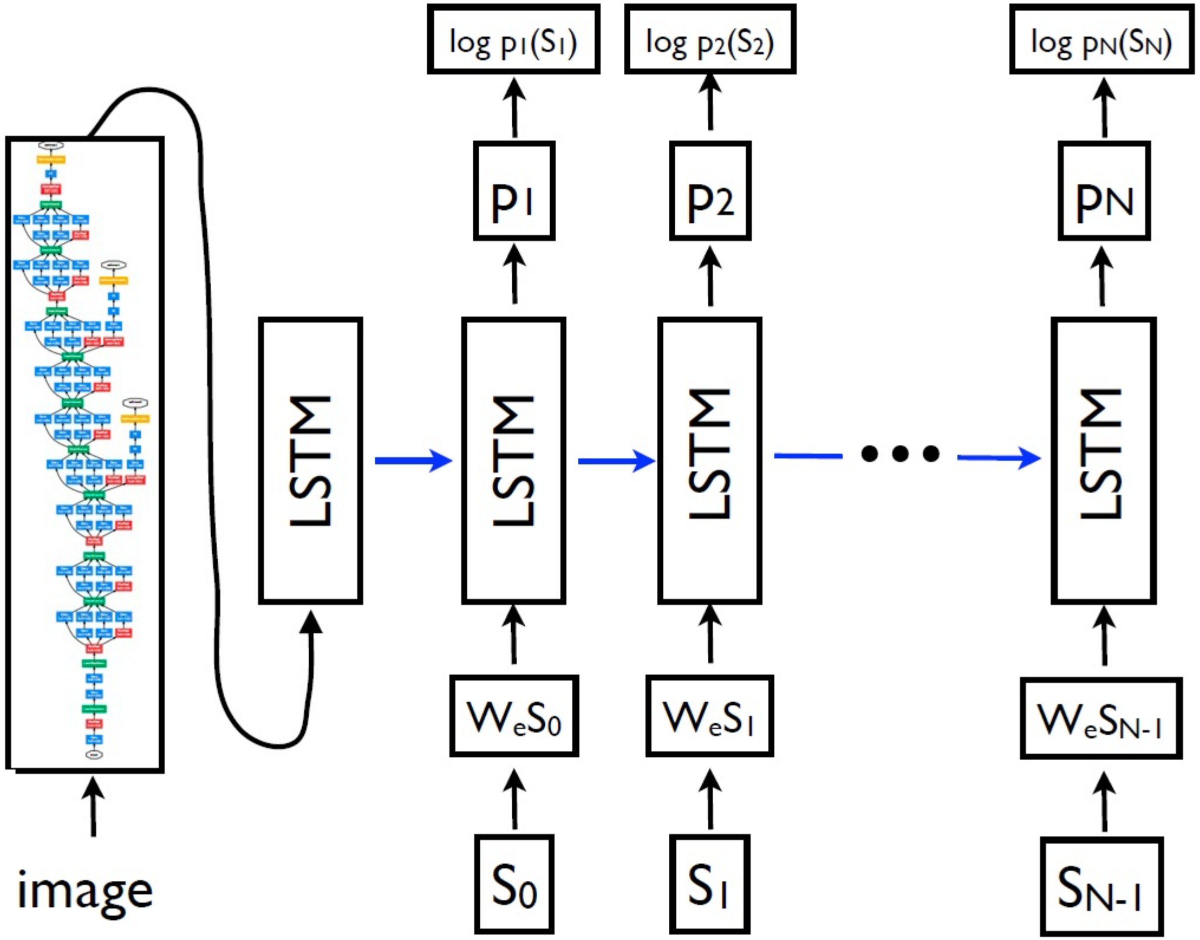
Part 1 Image Captioning with Vanilla RNNs

AI研习社

➢Image Captioning 测试

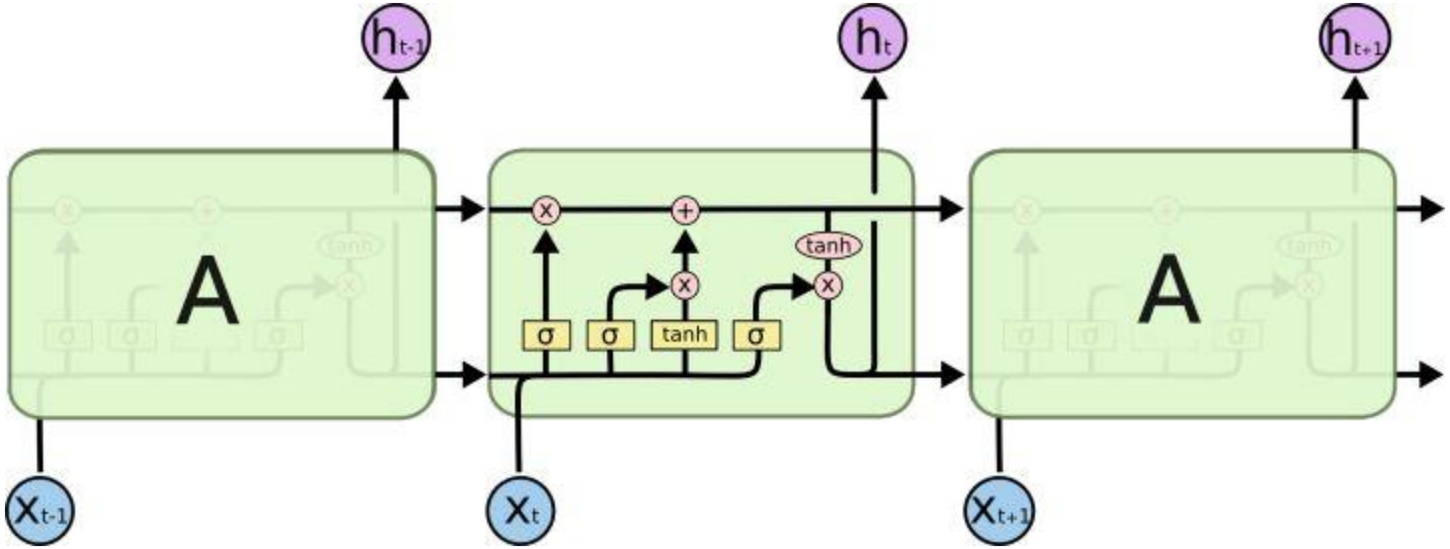
- 构造一个输入，它的第一个单词是start标记。和训练时不同的是，测试时我们不知道T的大小，因此限定了最大输出长度

```
# for each step
for t in range(1,max_length):
    # word embed
    embed, chahe_embed = word_embedding_forward(x, W_embed)
    # rnn step
    if self.cell_type == 'rnn':
        next_h, cache = rnn_step_forward(embed, prev_h, Wx, Wh, b)
   prev_h = next_h
   # affine
   h_affine, cache_h_affine = affine_forward(next_h, W_vocab, b_vocab)
   # max
   x = np.argmax(h_affine, axis=1)
   captions[:,t] = x
```



Part 2 Image Captioning with LSTMs

AI研习社



➢**LSTM 单元(前向):**

- (上右箭头) c_{t}
- (下右箭头) h_{t}
- (上箭头) h_{t} softmax

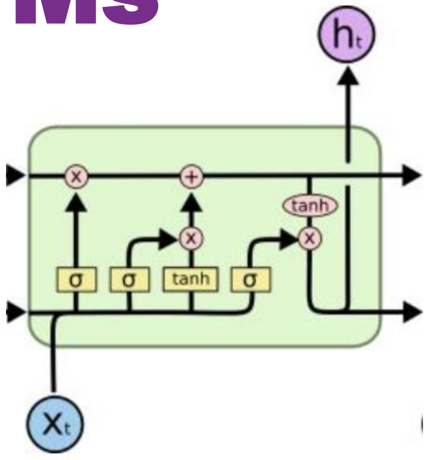
$$
\begin{aligned}
f_t &= \mathrm{sigmoid}(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \\
i_t &= \mathrm{sigmoid}(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \\
g_t &= \tanh(W_{gx}x_t + W_{gh}h_{t-1} + b_g) \\
c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\
o_t &= \mathrm{sigmoid}(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

- f、i、g、o的线性部分可以通过以一次计算完成

```
z = x.dot(Wx) + prev_h.dot(Wh) + b
```

Part 2 Image Captioning with LSTMs

AI研习社



➢**LSTM 单元(反向):**

- 从右侧传回来的值有两个，分别是dnext_h和dnext_c
- 四个线性部分记作i，f，o，g

$$
\begin{aligned}
f_t &= \text{sigmoid}(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \\
i_t &= \text{sigmoid}(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \\
g_t &= \tanh(W_{gx}x_t + W_{gh}h_{t-1} + b_g) \\
c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\
o_t &= \text{sigmoid}(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

df = (dnext_h + dh2c) * prev_c * f * (1 - f)

di = (dnext_h + dh2c) * g * i * (1 - i)

dg = (dnext_h + dh2c) * i * (1 - g**2)

dprev_c = (dnext_h + dh2c) * f

do = dnext_h * tanh(c) * o * (1 - o)

dh2c = dnext_h * o * (1 - tanh(c)**2)

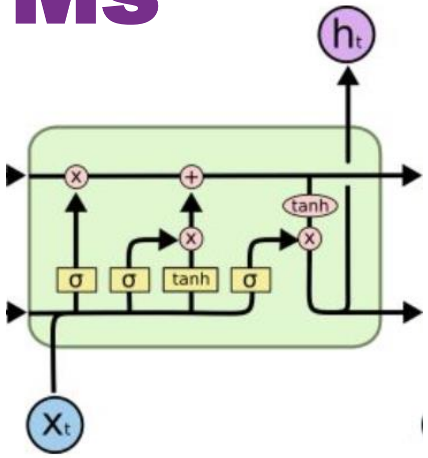
```
d = np.hstack((di, df, do, dg)) #(N, 4H)
```

Part 2 Image Captioning with LSTMs

AI研习社

➢完整 LSTM (前向):

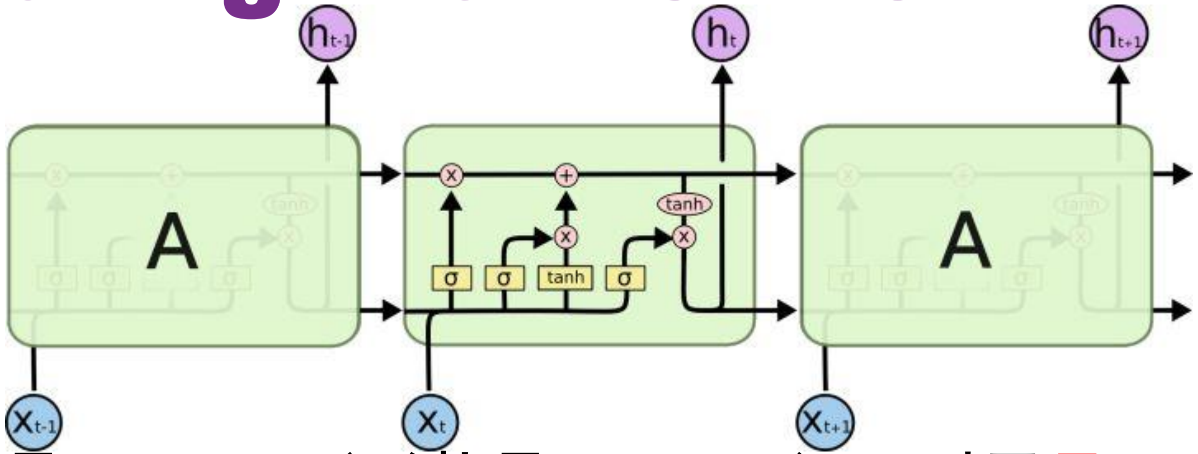
- 比RNN多了一个c，初始为0




```
prev_h = h0
h = np.zeros((N, T, H))
prev_c = np.zeros((N, H))
cache = {}
for t in range(T):
    xt = x[:, t, :]
    next_h, next_c, cache[t] = lstm_step_forward(xt, prev_h, prev_c, Wx, Wh, b)
    h[:, t, :] = next_h
    prev_h = next_h
    prev_c = next_c
```

AI研习社

Part 2 Image Captioning with LSTMs



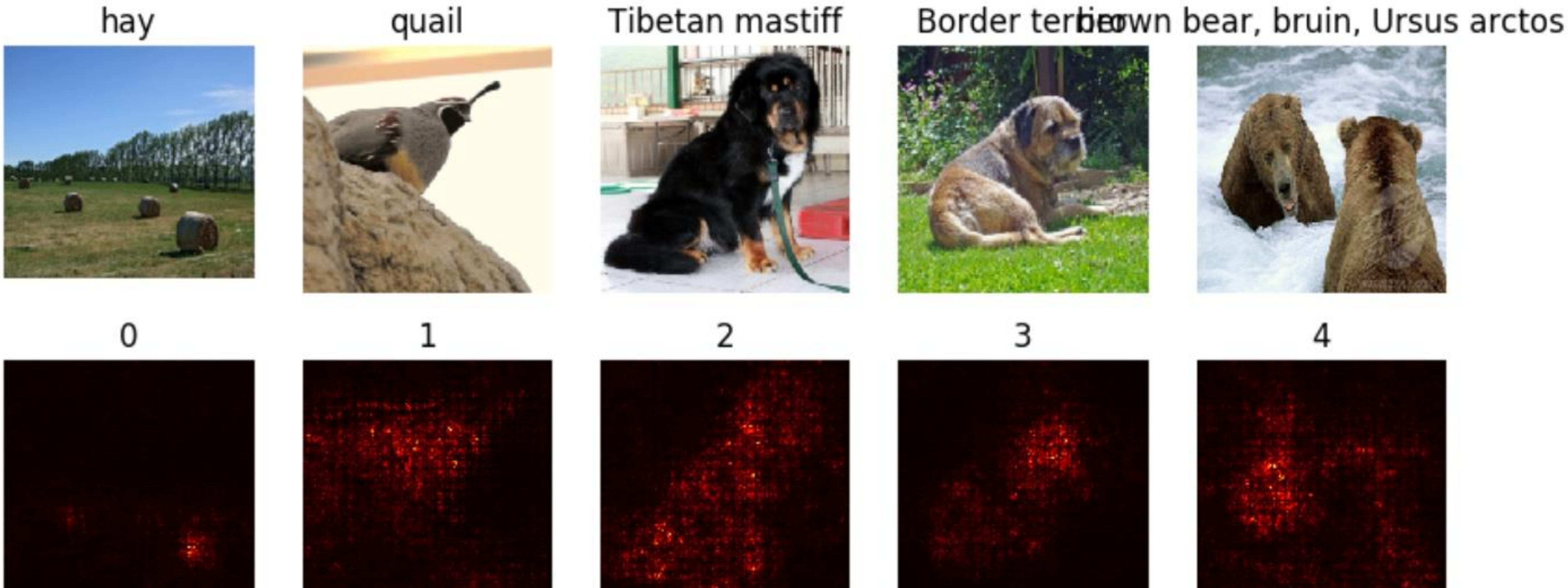
➤完整 LSTM (反向):

- 比RNN多了一个c
- 首先逆序：
```
for t in range(T-1, -1, -1):
```
- 对于每一个时间片t，上面来的导数是dh[:, t, :]（形状是(N, T, H)），对于最后一个单元，它的右边没有传来导数，所以初始化dprev_h与dnext_c都是0。
- 于是对于每一个LSTM单元，需要传进去的导数就是dh[:, t, :]+dprev_h。
```
dnext_h = dh[:, t, :] + dprev_h#上面+右边,初始右边=0
dx[:, t, :], dprev_h, dprev_c, dWxt, dWht, dbt = lstm_step_backward(dnext_h, dnext_c, cache[t])
dnext_c = dprev_c
```
- dWx，dWh，db和dh这四个参数在不同时刻是共享的，因此需要在每一个时刻把它们都加起来做更新。
```
dWx += dWxt #不同时刻共享的
dWh += dWht
db += dbt
dh0 = dprev_h
```

Part 3 Network Visualization

AI研习社

➢Saliency maps



Part 3 Network Visualization

AI研习社

- ➢Fooling Images
- 调整图片，使其骗过我们训练好的网络
- 也就是用目标分类对输入图片的梯度来迭代更新输入图片

Part 3 Network Visualization

AI研习社

➢Class Visualization

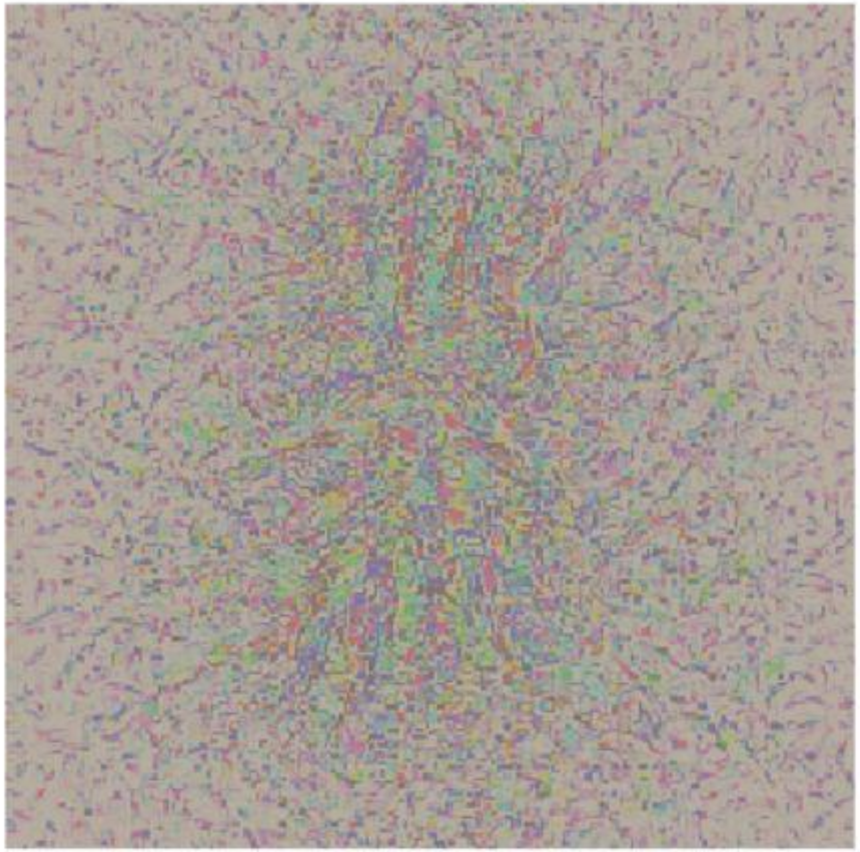
- 网络关注的每个类别 上更新目标
  分类的梯度

- 首先计算梯度

```
    loss = tf.nn.softmax_cross_entro s=model.classifier) # scala
r loss
    loss -= l2_reg * tf.nn.l2_loss(m
    grad = tf.gradients(loss, model. e, same size as model.image
```

- 迭代，更新噪声图片

```
clip = tf.clip_
X = sess.run(cl
```

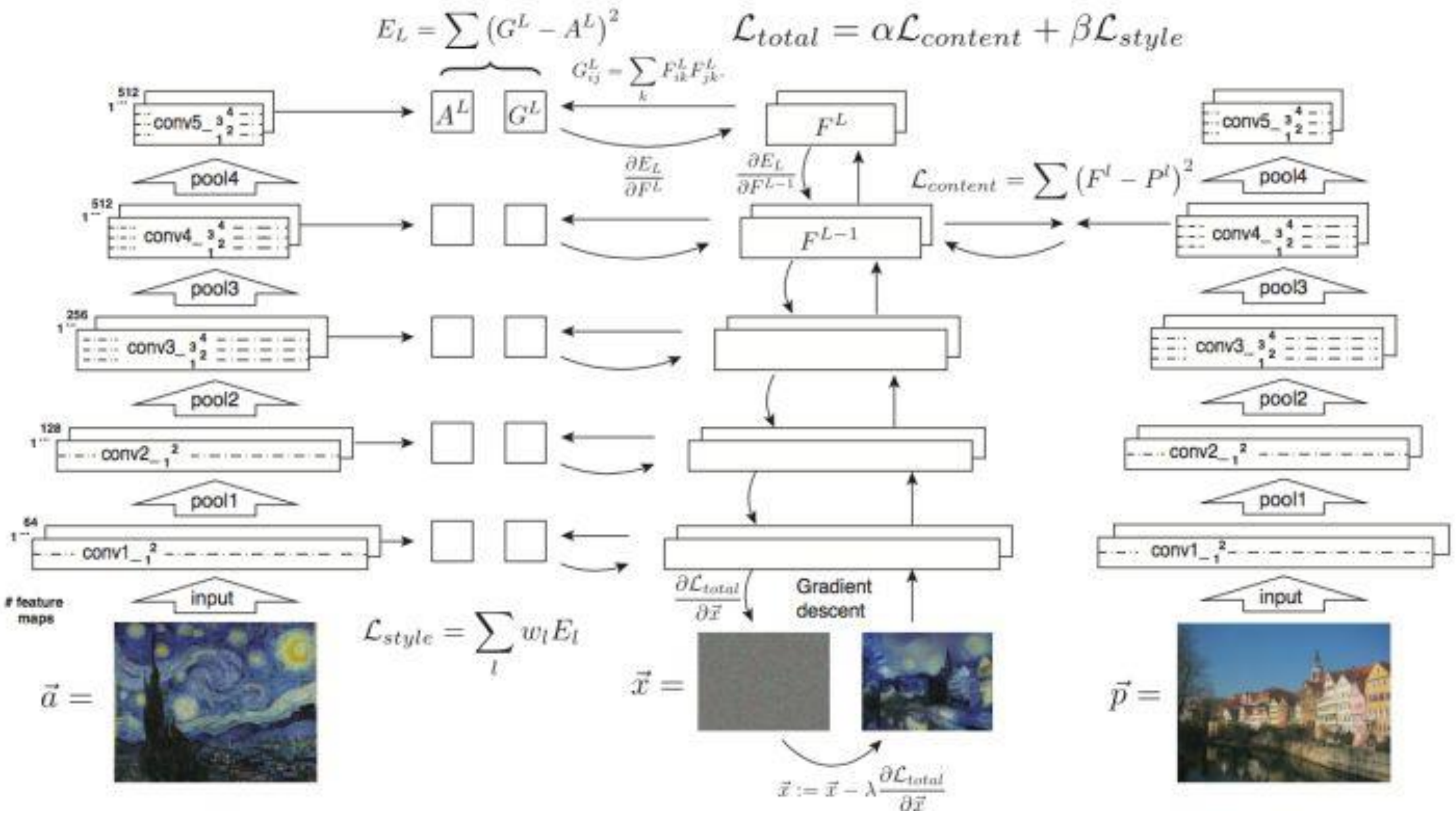


Part 4 Style Transfer

AI研习社

➢网络结构

- 定义一个新的loss，然后对随机噪声图片进行梯度更新。
- 注意三个图片是经过同样的训练好的网络的。



Part 4 Style Transfer

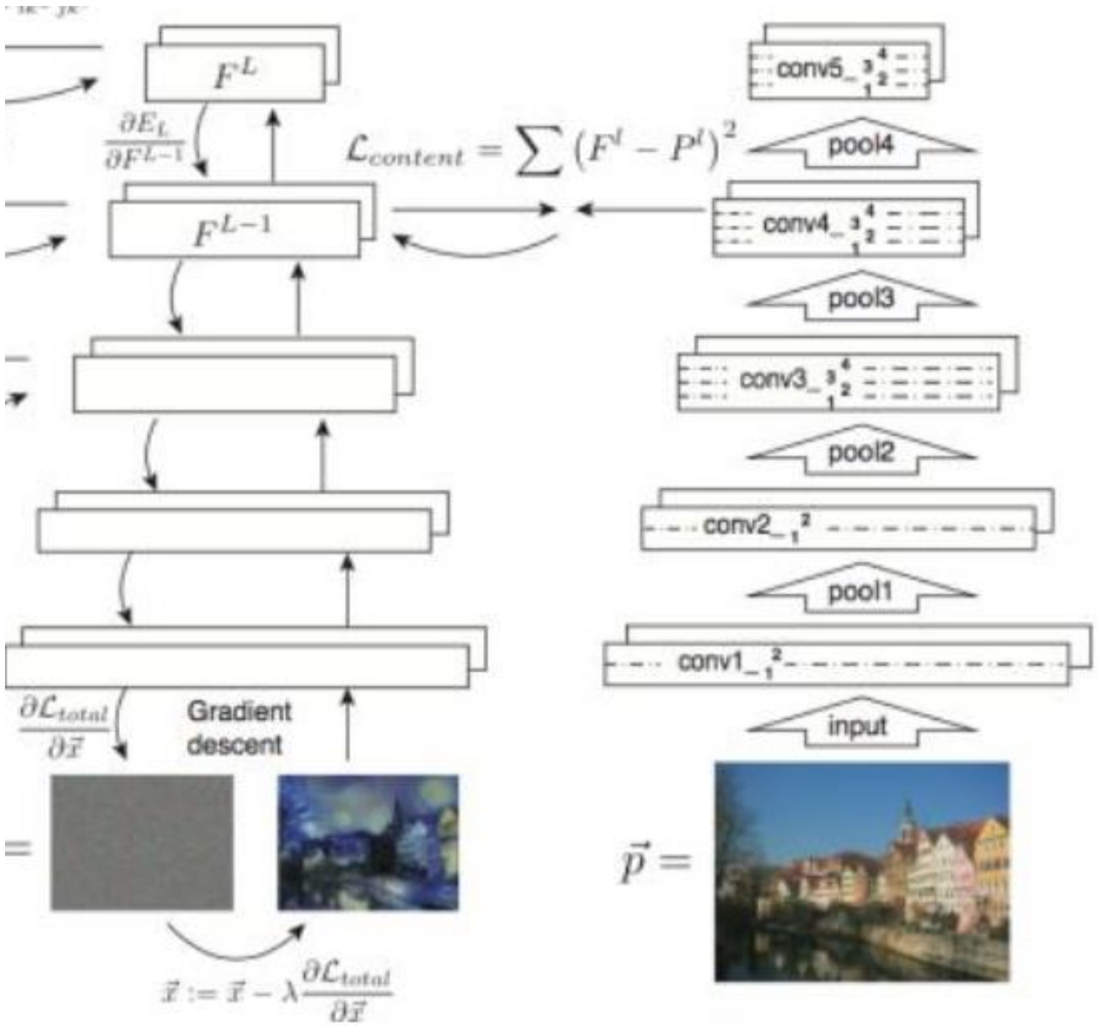
AI研习社

➢content loss

- 内容loss反映了生成的图片内容和源内容图片的差异

$$L_c = w_c * \sum (F_{i,j}^l - P_{i,j}^l)^2$$

- 其中 F_{i,j}^{l} 是生成图片在网络中第 l 层的feature map，P_{i,j}^{l} 是源内容图片在网络中第 l 层的feature map



Part 4 Style Transfer

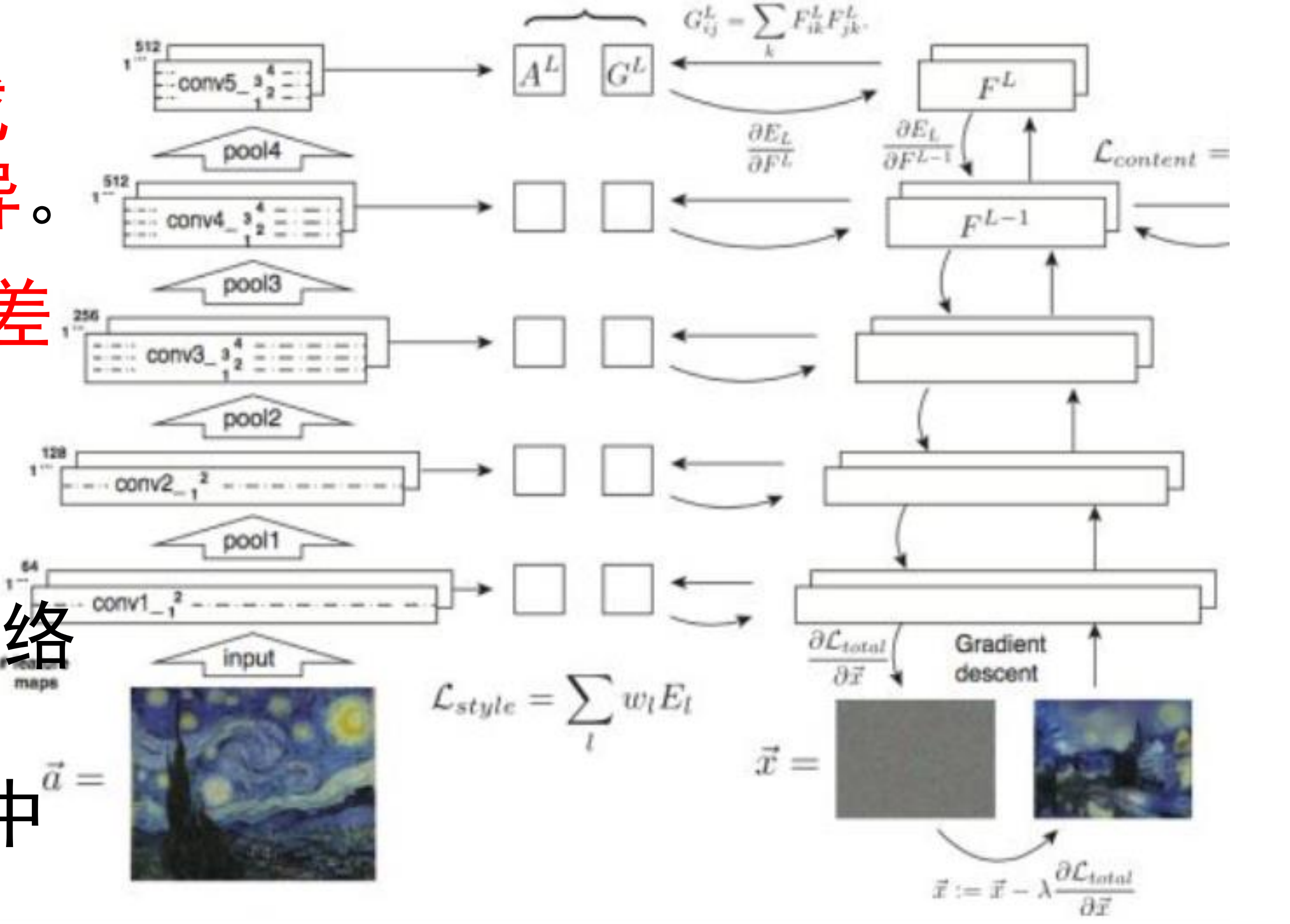
AI研习社

➢style loss

- 风格loss目的就是为了衡量生成的图片风格和源风格图片的差异。
- 而风格是用feature map的协方差矩阵度量。 $G = F^T * F$

$$L_s = w_s * \sum (G_{i,j}^l - A_{i,j}^l)^2$$

- 其中 G_{i,j}^{l} 是生成图片在网络中第 l 层的feature map，A_{i,j}^{l} 是源风格图片在网络中第 l 层的feature map



Part 4 Style Transfer

AI研习社

➢total variation loss

- 使生成图片更平滑

$$L_{tv} = w_t * \sum_{c=1}^{3} \sum_{i=1}^{H-1} \sum_{j=1}^{W-1} (x_{i,j+1,c} - x_{i,j,c})^2 + (x_{i+1,j,c} - x_{i,j,c})^2$$

- 这个式子是在H和W维度上计算相邻像素的差值的平方和

```
pixel_dif1 = img[:, 1:, :W-1, :] - img[:, :H-1, :W-1, :]
pixel_dif2 = img[:, :H-1, 1:, :] - img[:, :H-1, :W-1, :]
```

- tf里提供了绝对值版本的total variation loss：tf.image.total_variation()。也就是不算平方，只用绝对值。

Part 4 Style Transfer

AI研习社

- 最后3个loss加在一起就行了
- 可以调的参数：loss的三个权重，内容loss和风格loss的层选择
- 当风格loss权重为0时，是对源内容图像的重建

Content Source Img.

Style Source Img.

Part 5 Generative Adversarial Networks

AI研习社

➢GAN

- GAN其实就是生成器和判别器之间的博弈，一方面判别器从真实数据和生成数据中不断学习提高自己的判别能力，另一方面生成器不断迭代以提高自己的欺骗能力。

$$minimize_G\ maximize_D\ \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(x))]$$

- 1. 更新生成器使得判别器做出正确判断的概率下降
- 2. 更新判别器使得判别器做出正确判断的概率上升

$$maximize_D\ \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(x))]$$

- 1. 等价于 1.* 更新生成器使得判别器做出错误判断的概率上升

$$maximize_G\ \mathbb{E}_{x\sim p_z(x)}[logD(G(x))]$$

Part 5 Generative Adversarial Networks

AI研习社

➢Vanilla Gan

- Discriminator
  - Fully connected layer from size 784 to 256
  - LeakyReLU with alpha 0.01
  - Fully connected layer from 256 to 256
  - LeakyReLU with alpha 0.01
  - Fully connected layer from 256 to 1
- Generator
  - Fully connected layer from tf.shape(z)[1] (the number of noise dimensions) to 1024
  - ReLU
  - Fully connected layer from 1024 to 1024
  - ReLU
  - Fully connected layer from 1024 to 784
  - TanH (To restrict the output to be [-1,1])

Part 5 Generative Adversarial Networks

AI研习社

- Vanilla Gan Loss： sigmoid_cross_entropy loss

$$maximize_D \ \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(x))]$$

$$maximize_G \ \mathbb{E}_{x\sim p_z(x)}[logD(G(x))]$$

```
with tf.variable_scope("") as scope:
    #scale images to be -1 to 1
    logits_real = discriminator(preprocess_img(x))
    # Re-use discriminator weights on new inputs
    scope.reuse_variables()
    logits_fake = discriminator(G_sample)
D_label = tf.ones_like(logits_real)
G_fake_label = tf.zeros_like(logits_fake)
G_real_label = tf.ones_like(logits_fake)
D_loss1 = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = logits_real, labels = D_label))
D_loss2 = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = logits_fake, labels = G_fake_label))
G_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = logits_fake, labels = G_real_label))
return D_loss1 + D_loss2, G_loss
```

Part 5 Generative Adversarial Networks

AI研习社

➢Least Squares GAN

- Loss由交叉熵改变为Least Squares：

$$\ell_G = \frac{1}{2}\mathbb{E}_{z \sim p(z)}[(D(G(z)) - 1)^2]$$

$$\ell_D = \frac{1}{2}\mathbb{E}_{x \sim p_{\text{data}}}[(D(x) - 1)^2] + \frac{1}{2}\mathbb{E}_{z \sim p(z)}[(D(G(z)))^2]$$

```
D_loss = tf.reduce_mean(tf.square(score_real - 1)) + tf.reduce_mean(tf.square(score_fake))
G_loss = tf.reduce_mean(tf.square(score_fake - 1))
return 0.5 * D_loss, 0.5 * G_loss
```

Part 5 Generative Adversarial Networks

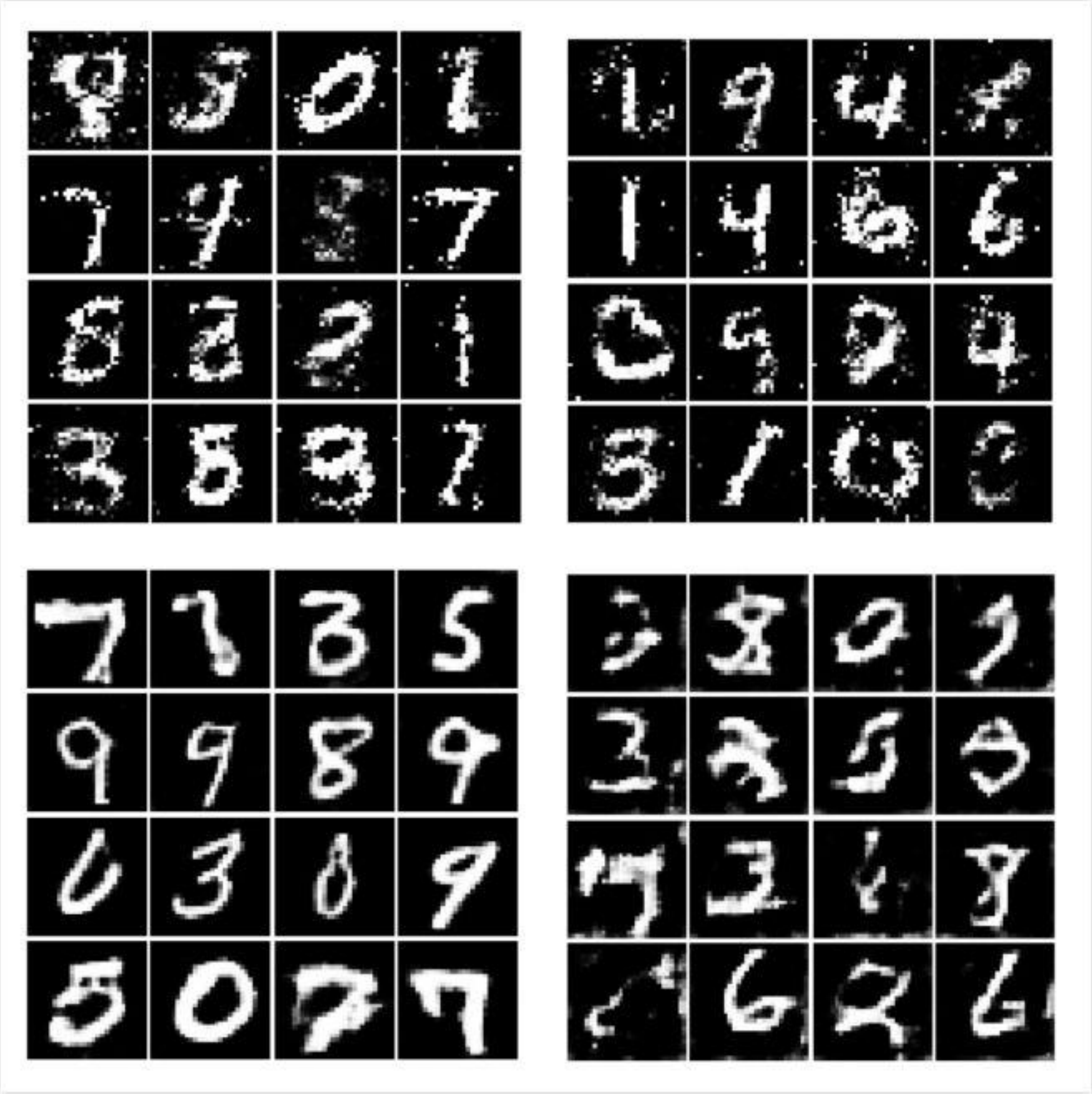
AI研习社

➢Deep Convolutional GANs

- Discriminator
  - 32 Filters, 5x5, Stride 1, Leaky ReLU(alpha=0.01)
  - Max Pool 2x2, Stride 2
  - 64 Filters, 5x5, Stride 1, Leaky ReLU(alpha=0.01)
  - Max Pool 2x2, Stride 2
  - Flatten
  - Fully Connected size 4 x 4 x 64, Leaky ReLU(alpha=0.01)
  - Fully Connected size 1
- Generator
  - Fully connected of size 1024, ReLU
  - BatchNorm
  - Fully connected of size 7 x 7 x 128, ReLU
  - BatchNorm
  - Resize into Image Tensor
  - 64 conv2d^T (transpose) filters of 4x4, stride 2, ReLU
  - BatchNorm
  - 1 conv2d^T (transpose) filter of 4x4, stride 2, TanH

AI研习社

Part 5 works



➢WGAN-G

- Discriminat
  - 64 Filters of 4
  - 128 Filters of
  - BatchNorm
  - Flatten
  - Fully connecte
  - Fully connecte
- Generator：
- Loss： grad
  https://www

课后讨论

那么多GAN，究竟哪个更好？

=====>https://zhuanlan.zhihu.com/p/31563676

AI研习社

CS231n Assgnment3 Fin