Modern CNNs, RNNs and Sequential Processing

Bill Freeman, Antonio Torralba, Phillip Isola
6.819 / 6.869
Today: parts of chapter 10

Review lecture 7

Other good resources for RNNs:
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Alexnet — [Krizhevsky et al. NIPS 2012]

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)
What filters are learned?
What filters are learned?
Get to know your units

11x11 convolution kernel
(3 color channels)
Get to know your units
Get to know your units
Get to know your units
Get to know your units
Get to know your units
Get to know your units
Get to know your units

96 Units in conv1
Gabor wavelets

\[ \psi_c(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \cos(2\pi u_0 x) \]

\[ \psi_s(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \sin(2\pi u_0 x) \]
Comparing Human and Machine Perception

**FIGURE 1** Schematic overview of the processing done by the early visual system. On the left, are some of the major structures to be discussed; in the middle, are some of the major operations done at the associated structure; in the right, are the 2-D Fourier representations of the world, retinal image, and sensitivities typical of a ganglion and cortical cell.
Fig. 5. Top row: illustrations of empirical 2-D receptive field profiles measured by J. P. Jones and L. A. Palmer (personal communication) in simple cells of the cat visual cortex. Middle row: best-fitting 2-D Gabor elementary function for each neuron, described by (10). Bottom row: residual error of the fit, indistinguishable from random error in the Chi-squared sense for 97 percent of the cells studied.
Deep Neural Networks for Visual Recognition

2012: AlexNet
5 conv. layers
Error: 15.3%

2014: VGG
16 conv. layers
Error: 8.5%

2015: GoogLeNet
22 conv. layers
Error: 7.8%

2016: ResNet
>100 conv. layers
Error: 4.4%
2012: AlexNet
5 conv. layers

11x11 conv, 96, /4, pool/2

5x5 conv, 256, pool/2

3x3 conv, 384

3x3 conv, 384

3x3 conv, 256, pool/2

fc, 4096

fc, 4096

fc, 1000

Error: 15.3%
2014: VGG
16 conv. layers
3x3 conv, 64, pool/2
3x3 conv, 128
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256, pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512, pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512, pool/2
fc, 4096
fc, 4096
fc, 1000
Softmax

Very Deep Convolutional Networks for Large-Scale Image Recognition


Small convolutional kernels: 3x3
ReLu non-linearities
>100 million parameters.

Error: 8.5%
Chaining convolutions

\[ 3 \times 3 \odot 3 \times 3 = 5 \times 5 \]

25 coefficients, but only 18 degrees of freedom.

\[ 3 \times 3 \odot 1 \times 3 = 3 \times 3 \]

9 coefficients, but only 6 degrees of freedom.

Only separable filters… would this be enough?
Dilated convolutions

3x3

5x5

\[
\begin{array}{ccc}
a & 0 & b & 0 & c \\
0 & 0 & 0 & 0 & 0 \\
d & 0 & e & 0 & f \\
0 & 0 & 0 & 0 & 0 \\
g & 0 & h & 0 & i \\
\end{array}
\]

25 coefficients
9 degrees of freedom

= 

7x7

49 coefficients
18 degrees of freedom

What is lost?

Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) $F_1$ is produced from $F_0$ by a 1-dilated convolution; each element in $F_1$ has a receptive field of $3 \times 3$. (b) $F_2$ is produced from $F_1$ by a 2-dilated convolution; each element in $F_2$ has a receptive field of $7 \times 7$. (c) $F_3$ is produced from $F_2$ by a 4-dilated convolution; each element in $F_3$ has a receptive field of $15 \times 15$. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.
Deep Residual Learning for Image Recognition


Figure 2. Residual learning: a building block.
If output has same size as input:

\[ \mathcal{F}(x) + x \]

If output has a different size:

\[ \mathcal{F}(x) + Wx \]
Other good things to know

- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance

**Good training**: hidden units are sparse across samples and across features.

[Derived from slide by Marc'Aurelio Ranzato]
Other good things to know

- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance

Bad training: many hidden units ignore the input and/or exhibit strong correlations.

[Derived from slide by Marc'Aurelio Ranzato]
Other good things to know

- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance
- Visualize filters

Good training: learned filters exhibit structure and are uncorrelated.

[Derived from slide by Marc'Aurelio Ranzato]
Normalization layers

\[ \hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}} \]
Batch processing
"Tensor flow"

\[ \mathbf{x}^{(l)} \in \mathbb{R}^{N_{\text{batch}} \times H^{(l)} \times W^{(l)} \times C^{(l)}} \]

\[ \mathbf{x}^{(l+1)} \in \mathbb{R}^{N_{\text{batch}} \times H^{(l+1)} \times W^{(l+1)} \times C^{(l+1)}} \]
Normalization layers

Normalize w.r.t. a single hidden unit's pattern of activation over training examples (a batch of examples).

[Figure from Wu & He, arXiv 2018]
Normalization layers

Normalize w.r.t. the mean and variance of the activations of all the hidden units (neurons) on a particular layer (c).

[Figure from Wu & He, arXiv 2018]
Normalization layers

Normalize w.r.t. the mean and variance of the activations of all the hidden units (neurons) on a particular layer (c) that process a particular location (h,w) in the image.

[Figure from Wu & He, arXiv 2018]
Normalization layers

Might as well…

[Figure from Wu & He, arXiv 2018]
Convolutions in time
It bothered him that the dog at three fourteen (seen from the side) should have the same name as the dog at three fifteen (seen from the front).
— “Funes the Memorius”, Borges 1962

“The Persistence of Memory”, Dali 1931
Memory unit

Rufus

Rufus!

W

W

time
Recurrent Neural Networks (RNNs)

Outputs

Hidden

W

Inputs

Hidden

Outputs

W

Inputs

Hidden

Outputs
Recurrent Neural Networks (RNNs)

Outputs $\hat{y}$

Hidden $h$

Inputs $x$

Time
Recurrent Neural Networks (RNNs)

\[ h^{(t)} = f(h^{(t-1)}, x^{(t)}) \]
\[ y^{(t)} = g(h^{(t)}) \]
**Recurrent** Neural Networks (RNNs)

\[ h^{(t)} = f(h^{(t-1)}, x^{(t)}) \]

\[ y^{(t)} = g(h^{(t)}) \]
Recurrent Neural Networks (RNNs)

\[
\begin{align*}
\mathbf{a}^{(t)} &= \mathbf{W} \mathbf{h}^{(t-1)} + \mathbf{U} \mathbf{x}^{(t)} + \mathbf{b} \\
\mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}) \\
\mathbf{o}^{(t)} &= \mathbf{V} \mathbf{h}^{(t)} + \mathbf{c} \\
\hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)})
\end{align*}
\]
Deep Recurrent Neural Networks (RNNs)

Outputs  $\hat{y}$

$\mathbf{h}_L$

Hidden

$\mathbf{h}_1$

Inputs  $\mathbf{x}$

$W_L$

$V$

$U_L$

$W_1$

$U_2$

$U_1$

time
Backprop through time

Outputs $\hat{y}$

Hidden $h$

Inputs $x$

$$\frac{\partial \hat{y}^{(t)}}{\partial x^{(0)}} = \frac{\partial \hat{y}^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \cdots \frac{\partial h^{(1)}}{\partial h^{(0)}} \frac{\partial h^{(0)}}{\partial x^{(0)}}$$
$$\frac{\partial J}{\partial W} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}(\hat{y}(t), y(t))}{\partial W}$$
Recurrent linear layer

\[
\begin{align*}
\mathbf{a}^{(t)} &= \mathbf{W} \mathbf{h}^{(t-1)} + \mathbf{U} \mathbf{x}^{(t)} + \mathbf{b} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t-1)}} \mathbf{W} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(t)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t)}} \mathbf{U} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t-1)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t)}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(t)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t)}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t-1)}} \frac{\partial \mathbf{h}^{(t-1)}}{\partial \mathbf{W}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(t)}} \frac{\partial \mathbf{x}^{(t)}}{\partial \mathbf{W}} \\
\frac{\partial J}{\partial \mathbf{W}} &= \sum_{t=0}^{T} \frac{\partial \mathcal{L}(\mathbf{y}^{(t)}, \mathbf{y}^{(t)})}{\partial \mathbf{W}}
\end{align*}
\]
The problem of long-range dependences

Capturing long-range dependences requires propagating information through a long chain of dependences.

- Old observations are forgotten
- Stochastic gradients become high variance (noisy), and gradients may \textit{vanish} or \textit{explode}
The problem of long-range dependences

Why not remember everything?

- Memory size grows with $t$
- This kind of memory is **nonparametric**: there is no finite set of parameters we can use to model it
- RNNs make a Markov assumption — the future hidden state only depends on the immediately preceding hidden state
- By putting the right info in to the hidden state, RNNs can model dependences that are arbitrarily far apart
The problem of long-range dependences

Other methods exist that do directly link old “memories” (observations or hidden states) to future predictions:

- Temporal convolutions
- Attention (see https://arxiv.org/abs/1706.03762)
- Memory networks (see https://arxiv.org/abs/1410.3916)
LSTMs
Long Short Term Memory

A special kind of RNN designed to avoid forgetting.

Related to resnets: inductive bias is that state transition is an identity function.

This way the default behavior is not to forget an old state. Instead of forgetting by default, the network has to learn to forget.
[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]
$C_t = \text{Cell state}$

[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]
Decide what information to throw away from the cell state.

Each element of cell state is multiplied by \( \sim 1 \) (remember) or \( \sim 0 \) (forget).

\[
f_t = \sigma \left( W_f \cdot [h_{t-1}, x_t] + b_f \right)
\]
Decide what new information to add to the cell state.

\[
i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)
\]

\[
\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)
\]

which indices to write to

what to write to those indices
Forget old selected old information, write selected new information.

\[ C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \]

[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]
After having updated the cell state’s information, decide what to output.

\[
o_t = \sigma \left( W_o \left[ h_{t-1}, x_t \right] + b_o \right) \\
h_t = o_t \cdot \tanh(C_t)
\]

[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]
Texture synthesis by non-parametric sampling

Synthesizing a pixel

non-parametric sampling

Models $P(p|N(p))$

Input image

[Efros & Leung 1999]
Texture synthesis with a deep net

Input partial image

Predicted color of next pixel

“white”

[PixelRNN, PixelCNN, van der Oord et al. 2016]
Input partial image

Predicted color of next pixel

“white”

[PixelRNN, PixelCNN, van der Oord et al. 2016]
Recall from lecture 12: we can represent colors as discrete classes

\[ \mathbf{y} \in \mathbb{R}^{H \times W \times K} \]

\[ \mathcal{L}(\mathbf{y}, f_{\theta}(\mathbf{x})) = H(\mathbf{y}, \text{softmax}(f_{\theta}(\mathbf{x}))) \]
And we can interpret the learner as modeling $P(\text{next pixel} \mid \text{previous pixels})$:

**Softmax regression** (a.k.a. multinomial logistic regression)

\[ \hat{y} \equiv [P_\theta(Y = 1 \mid X = x), \ldots, P_\theta(Y = K \mid X = x)] \quad \text{predicted probability of each class given input } x \]

\[ H(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k \quad \text{picks out the -log likelihood of the ground truth class } y \text{ under the model prediction } \hat{y} \]

\[ f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} H(y_i, \hat{y}_i) \quad \text{max likelihood learner!} \]
Network output

... →

| Turquoise | Blue | Green | Red | Orange | White | Gray | Black | ...

P(next pixel | previous pixels)

\[ P(p_i|p_1, \cdots, p_{i-1}) \]
Network output

\[ p_i \sim P(p_i \mid p_1, \ldots, p_{i-1}) \]
Network output

\[ p_i \sim P(p_i | p_1, \ldots, p_{i-1}) \]
Network output

\[ p_i \sim P(p_i | p_1, \cdots, p_{i-1}) \]
Network output

\[ p_i \sim P(p_i | p_1, \ldots, p_{i-1}) \]
\[ p_1 \sim P(p_1) \]
\[ p_2 \sim P(p_2|p_1) \]
\[ p_3 \sim P(p_3|p_1, p_2) \]
\[ p_4 \sim P(p_4|p_1, p_2, p_3) \]

\[ \{p_1, p_2, p_3, p_4\} \sim P(p_4|p_1, p_2, p_3) \cdot P(p_3|p_1, p_2) \cdot P(p_2|p_1) \cdot P(p_1) \]

\[ p_i \sim P(p_i|p_1, \ldots, p_{i-1}) \]

\[ \mathbf{p} \sim \prod_{i=1}^{N} P(p_i|p_1, \ldots, p_{i-1}) \]
Autoregressive probability model

\[ p \sim \prod_{i=1}^{N} P(p_i | p_1, \ldots, p_{i-1}) \]

\[ P(p) = \prod_{i=1}^{N} P(p_i | p_1, \ldots, p_{i-1}) \quad \text{← General product rule} \]

The sampling procedure we defined above takes exact samples from the learned probability distribution (pmf).

Multiplying all conditionals evaluates the probability of a full joint configuration of pixels.
Autoregressive probability model

\[ p \sim P(p) \]

Models that allow us to sample, i.e. generate, images from scratch are called **generative models**.

We will see more examples in a future lecture.
Samples from PixelRNN

[PixelRNN, van der Oord et al. 2016]
Image completions (conditional samples) from PixelRNN

[PixelRNN, van der Oord et al. 2016]