Neural networks in computer vision
Key idea

• Build complex functions by composing simple ones.
• Usually have a library of possible functions
  • Some functions have learnable parameters
• Fix hypothesis class by fixing network architecture
  • Fixed set of functions composed in fixed ways
  • Different settings of the learnable parameters yield different hypotheses
Recall: Empirical Risk Minimization

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(h(x_i; \theta), y_i)$$

Neural network

$$\theta^{(t+1)} = \theta^{(t)} - \lambda \frac{1}{N} \sum_{i=1}^{N} \nabla L(h(x_i; \theta), y_i)$$

Gradient descent update
Computing the gradient of the loss

\[ \nabla L(h(x; \theta), y) \]

\[ z = h(x; \theta) \]

\[ \nabla_\theta L(z, y) = \frac{\partial L(z, y)}{\partial z} \frac{\partial z}{\partial \theta} \]
Learning with function compositions

• $F = f_5 \circ f_4 \circ f_3 \circ f_2 \circ f_1$

• Suppose $f_i$ has learnable parameters $w_i$, takes input $z_{i-1}$ and produces output $z_i$

• Need to compute $\frac{\partial F}{\partial w_i}$. How?

• Key idea: recurrence
  • If we know $\frac{\partial F}{\partial z_i}$, then chain rule gives: $\frac{\partial F}{\partial z_i} \frac{\partial z_i}{\partial w_i}$, second term only requires each function be differentiable

• Also $\frac{\partial F}{\partial z_i} = \frac{\partial F}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial z_i}$
Learning with function compositions

Backpropagation
Backpropagation for a sequence of functions

\[ z_i = f_i(z_{i-1}, w_i) \]
\[ z_0 = x \]
\[ z = z_n \]

\[ \frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial z_i} \]

\[ \frac{\partial z}{\partial w_i} = \frac{\partial z}{\partial z_i} \frac{\partial z_i}{\partial w_i} \]
Backpropagation for a sequence of functions

\( z_i = f_i(z_{i-1}, w_i) \)
\( z_0 = x \)
\( z = z_n \)

- Assume we can compute partial derivatives of each function

\[
\frac{\partial z_i}{\partial z_{i-1}} = \frac{\partial f_i(z_{i-1}, w_i)}{\partial z_{i-1}} \quad \frac{\partial z_i}{\partial w_i} = \frac{\partial f_i(z_{i-1}, w_i)}{\partial w_i}
\]

- Use \( g(z_i) \) to store gradient of \( z \) w.r.t \( z_i \), \( g(w_i) \) for \( w_i \)

- Calculate \( g_i \) by iterating backwards

\[
g(z_n) = \frac{\partial z}{\partial z_n} = 1 \quad g(z_{i-1}) = \frac{\partial z}{\partial z_i} \frac{\partial z_i}{\partial z_{i-1}} = g(z_i) \frac{\partial z_i}{\partial z_{i-1}}
\]

- Use \( g_i \) to compute gradient of parameters

\[
g(w_i) = \frac{\partial z}{\partial z_i} \frac{\partial z_i}{\partial w_i} = g(z_i) \frac{\partial z_i}{\partial w_i}
\]
Backpropagation for a sequence of functions

• Each “function” has a “forward” and “backward” module

• Forward module for \( f_i \)
  • takes \( z_{i-1} \) and weight \( w_i \) as input
  • produces \( z_i \) as output

• Backward module for \( f_i \)
  • takes \( g(z_i) \) as input
  • produces \( g(z_{i-1}) \) and \( g(w_i) \) as output

\[
g(z_{i-1}) = g(z_i) \frac{\partial z_i}{\partial z_{i-1}} \quad g(w_i) = g(z_i) \frac{\partial z_i}{\partial w_i}
\]
Backpropagation for a sequence of functions

\[ f_i(z_{i-1}, w_i) \rightarrow z_i \]
Backpropagation for a sequence of functions

\[ g(z_{i-1}) \rightarrow f_i \rightarrow g(z_i) \]

\[ g(w_i) \]
Chain rule for vectors

\[
\frac{\partial a}{\partial b} = \frac{\partial a}{\partial c} \frac{\partial c}{\partial b}
\]

\[
\frac{\partial a_i}{\partial b_j} = \sum_k \frac{\partial a_i}{\partial c_k} \frac{\partial c_k}{\partial b_j}
\]

\[
\frac{\partial a}{\partial b} (i, j) = \frac{\partial a_i}{\partial b_j}
\]

Jacobian

\[
\frac{\partial a}{\partial b} = \frac{\partial a}{\partial c} \frac{\partial c}{\partial b}
\]
Loss as a function
Beyond sequences: computation graphs

- Arbitrary *graphs* of functions
- No distinction between intermediate outputs and parameters
Computation graph - Functions

• Each node implements two functions
  • A “forward”
    • Computes output given input
  • A “backward”
    • Computes derivative of $z$ w.r.t input, given derivative of $z$ w.r.t output
Computation graphs

\[ f_i \]
Computation graphs
Computation graphs
Computation graphs
Module Library 1

- **Linear**
  - \( y = W x + b \)
  - Learnable parameters: \( W, b \)
  - Hyperparameter: Output dimensionality

- **Rectified Linear Unit (ReLU)**
  - \( y = \max(x, 0) \)
  - Learnable parameters: None
  - Inspired from signal processing ("half-wave rectification"); easy to implement in hardware.

- **Softmax**
  - \( y_k = \frac{e^{x_k}}{\sum_i e^{x_i}} \)
  - Learnable parameters: None
  - Converts a set of scores into probabilities (bounded in \([0,1]\), sum to 1)
Multilayer perceptrons

Example function
Reducing capacity
Reducing capacity
Idea 1: local connectivity

• Inputs and outputs are *feature maps*
• Pixels only related to nearby pixels
Idea 2: Translation invariance

• Pixels only related to nearby pixels
Local connectivity + translation invariance = convolution

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>0.1</td>
<td>3.6</td>
</tr>
<tr>
<td>1.8</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>1.1</td>
<td>3.4</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Local connectivity + translation invariance = convolution

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>0.1</td>
<td>3.6</td>
</tr>
<tr>
<td>1.8</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>1.1</td>
<td>3.4</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Local connectivity + translation invariance = convolution
Convolution as a primitive

Note: convolution independent of input height/width
Convolution with subsampling

• *Subsampling* = reducing resolution by dropping rows and columns
• Can be done with *strided* convolution
  • Stride of *k* means output pixel every *k* input pixels
• Typically done *without anti-aliasing*, though *anti-aliasing helps*\(^1\)

\(^1\)https://richzhang.github.io/antialiased-cnns/
Convolution with subsampling
Invariance to deformations
Effect of subsampling

• Same sized filters captures larger neighborhoods on lower resolution features
• Magnitude of translations / deformations reduce with lower resolution
• Convolution in earlier steps detects more local patterns less resilient to deformations / translations
• Convolution in later steps detects more global patterns more resilient to deformations / translations
• Subsampling allows capture of larger, more invariant patterns
Pooling

• Similar to convolution, but take *max* or *average* across window for every channel
• No learnable parameters
Global Average Pooling

• Special case: take average across entire input space for every channel
• Useful for converting feature maps to vector of image features
Exploring convnet architectures
Deeper is better

Challenge winner's accuracy

- 2010: 29 layers
- 2011: 28 layers
- 2012: 16 layers
- 2013: 14 layers
- 2014: 16 layers
Deeper is better

![Bar chart showing challenge winner's accuracy from 2010 to 2014, with Alexnet in 2011 and VGG16 in 2014.](image_url)
The VGG pattern

• Every convolution is 3x3, padded by 1
• Every convolution followed by ReLU
• ConvNet is divided into “stages”
  • Layers within a stage: no subsampling
  • Subsampling by 2 at the end of each stage
• Layers within stage have same number of channels
• Every subsampling \(\rightarrow\) double the number of channels
Challenges in training: exploding / vanishing gradients

- Vanishing / exploding gradients

\[
\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \ldots \frac{\partial z_{i+1}}{\partial z_i}
\]

- If each term is (much) greater than 1 \(\rightarrow\) explosion of gradients
- If each term is (much) less than 1 \(\rightarrow\) vanishing gradients
Challenges in training: dependence on init
Solutions

• Careful init

• Batch normalization

• Residual connections
Careful initialization

• Key idea: want variance to remain approx. constant
  • Variance increases in backward pass => exploding gradient
  • Variance decreases in backward pass => vanishing gradient
• “MSRA initialization”
  • weights = Gaussian with 0 mean and variance = 2/(k*k*d)
Batch normalization

• Key idea: normalize so that each layer output has zero mean and unit variance
  • Compute mean and variance for each channel
  • Aggregate over batch
  • Subtract mean, divide by std

• Need to reconcile train and test
  • No ”batches” during test
  • After training, compute means and variances on train set and store

Residual connections

• In general, gradients tend to vanish
• Key idea: allow gradients to flow unimpeded

\[ z_{i+1} = f_{i+1}(z_i, w_{i+1}) \]
\[ \frac{\partial z_{i+1}}{\partial z_i} = \frac{\partial f_{i+1}(z_i, w_{i+1})}{\partial z_i} \]

\[ \frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i} \]
Residual connections

• In general, gradients tend to vanish
• Key idea: allow gradients to flow unimpeded

\[ z_{i+1} = g_{i+1}(z_i, w_{i+1}) + z_i \]
\[ \frac{\partial z_{i+1}}{\partial z_i} = \frac{\partial g_{i+1}(z_i, w_{i+1})}{\partial z_i} + I \]

\[ \frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i} \]
Residual connections

• Assumes all $z_i$ have the same size
• True within a stage
• Across stages?
  • Doubling of feature channels
  • Subsampling
• Increase channels by 1x1 convolution
• Decrease spatial resolution by subsampling

$$z_{i+1} = g_{i+1}(z_i, w_{i+1}) + \text{subsample}(Wz_i)$$
A residual block

• Instead of single layers, have residual connections over block
Bottleneck blocks

• Problem: When channels increases, 3x3 convolutions introduce many parameters
  • $3 \times 3 \times c^2$

• Key idea: use 1x1 to project to lower dimensionality, do convolution, then come back
  • $c \times d + 3 \times 3 \times d^2 + d \times c$
The ResNet pattern

• Decrease resolution substantially in first layer
  • Reduces memory consumption due to intermediate outputs

• Divide into stages
  • maintain resolution, channels in each stage
  • halve resolution, double channels between stages

• Divide each stage into residual blocks

• At the end, compute average value of each channel to feed linear classifier
Putting it all together - Residual networks

Challenge winner's accuracy
DenseNets