Foundations of Data Science, Fall 2020

14. Neural Networks II

Prof. Dan Olteanu

DaST Data•(Systems+Theory)

University of Zurich[UZH]

Nov 24, 2020

https://lms.uzh.ch/url/RepositoryEntry/16830890400

https://uzh.zoom.us/j/96690150974?pwd=cnZmMTduWUtCeWoxYW85Z3RMYnpTZz09

Outline

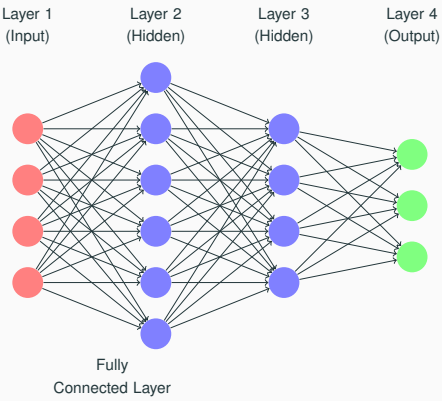
Recap

- Multi-layer perceptrons
- Backpropagation to compute gradients

Today, we'll look at

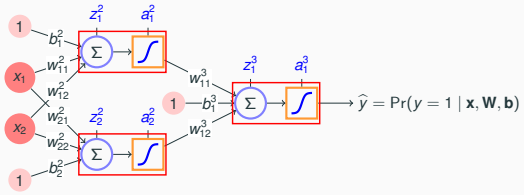
- Training neural networks: Difficulties and known hacks
- Convolutional neural networks (also Practical 3)

Recall: Feedforward Neural Networks



Units are organised in a layered directed-acyclic graph (no loops)

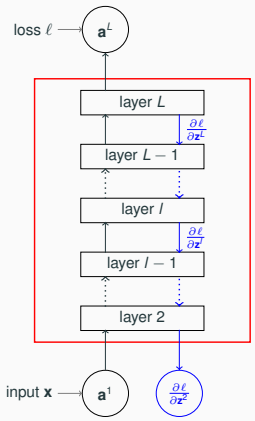
Recall MultiLayer Perceptron Example



Layers: (1) Input; (2) Hidden, with two units; (3) Output, here classification

- w_{ij}^{ℓ}: weight for the connection to unit i in layer ℓ from unit j in layer $\ell - 1$
- b_i^{ℓ}: bias term for unit i in layer ℓ
- z_i^{ℓ}: pre-activation for unit i in layer ℓ, sum of weighted outputs from $\ell - 1$
- a_i^{ℓ}: activation for unit i in layer ℓ, non-linear function of pre-activation

Recall: Forward and Backward Equations in the Backpropagation Algorithm



Forward Equations

(1) $\mathbf{a}^1 = \mathbf{x}$ (input)

(2) $\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$

(3) $\mathbf{a}^l = f_l(\mathbf{z}^l)$

(4) $\ell(\mathbf{a}^L, y)$

Back-propagation Equations

(1) Compute $\frac{\partial \ell}{\partial \mathbf{z}^L} = \frac{\partial \ell}{\partial \mathbf{a}^L} \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}$

(2) $\frac{\partial \ell}{\partial \mathbf{z}^l} = \frac{\partial \ell}{\partial \mathbf{z}^{l+1}} \mathbf{W}^{l+1} \frac{\partial \mathbf{a}^l}{\partial \mathbf{z}^l}$

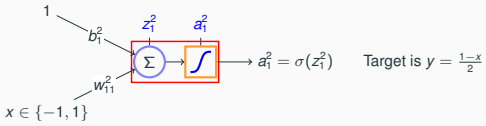
(3) $\frac{\partial \ell}{\partial \mathbf{W}^l} = \left(\mathbf{a}^{l-1} \frac{\partial \ell}{\partial \mathbf{z}^l} \right)^{\mathsf{T}}$

(4) $\frac{\partial \ell}{\partial \mathbf{b}^l} = \frac{\partial \ell}{\partial \mathbf{z}^l}$

Training Deep Neural Networks

- Back-propagation gives gradient
- Stochastic gradient descent is the method of choice
- Regularisation
  - How do we add ℓ_1 or ℓ_2 regularisation?
  - Don't regularise bias terms
- How about convergence?
- What did we learn in the last 10 years, that we didn't know in the 80s?
  - Current best practice: Known hacks not exact science
  - Training difficulties: saturation, vanishing gradient, overfitting
  - Avoiding saturation and overfitting: early stopping, adding data, dropout

Saturation Example: NOT Gate



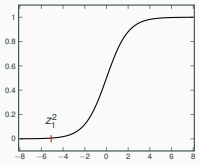
Squared Loss Function

$\ell(a_1^2, y) = (a_1^2 - y)^2$

$\frac{\partial \ell}{\partial z_1^2} = 2(a_1^2 - y)\frac{\partial a_1^2}{\partial z_1^2} = 2(a_1^2 - y)\sigma'(z_1^2)$

Recall: $\sigma'(z_1^2) = \sigma(z_1^2)\left(1 - \sigma(z_1^2)\right)$

If $x = -1$, $w_{11}^2 = 5$, $b_1^2 = 0$, then $\sigma'(z_1^2) \approx 0$

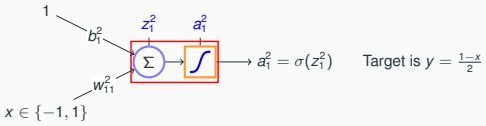


Gradient steps may not make much progress, as the gradient is very small

Yet the loss function $(a_1^2 - y)^2$ has a relatively large value!

Saturation: Pre-activations are in range where the activation function is very flat

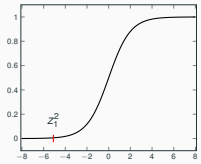
Saturation: Changing the Square Loss with Cross-Entropy Loss



Cross-Entropy Loss Function

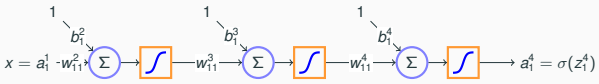
$\ell(a_1^2, y) = -\left(y \log a_1^2 + (1-y)\log(1 - a_1^2)\right)$

$\frac{\partial \ell}{\partial z_1^2} = \frac{a_1^2 - y}{a_1^2(1-a_1^2)}\frac{\partial a_1^2}{\partial z_1^2} = a_1^2 - y$



Large gap between a_1^2 and y also means large magnitude of the gradient

Vanishing Gradients during Back-Propagation



Cross-entropy loss: $\ell(a_1^4, y) = -\left(y \log a_1^4 + (1-y)\log(1 - a_1^4)\right)$

The gradients are computed as follows:

- $\frac{\partial \ell}{\partial z_1^4} = a_1^4 - y$
- $\frac{\partial \ell}{\partial z_1^3} = \frac{\partial \ell}{\partial z_1^4}\frac{\partial z_1^4}{\partial a_1^3}\frac{\partial a_1^3}{\partial z_1^3} = (a_1^4 - y)w_{11}^4\sigma'(z_1^3)$
- $\frac{\partial \ell}{\partial z_1^2} = \frac{\partial \ell}{\partial z_1^3}\frac{\partial z_1^3}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2} = (a_1^4 - y)w_{11}^4\sigma'(z_1^3)w_{11}^3\sigma'(z_1^2)$

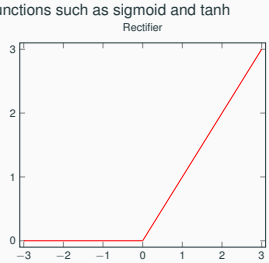
Vanishing Gradient Problem: The gradient product ≈ 0 in case of many layers

We need to multiply many gradients $\sigma'(z_i^l)$ and $\sigma'(z) \in [0, 1/4]$

The product with weights w_{ij}^l may avoid vanishing yet may explode the gradient

Avoiding Saturation using ReLU

Saturation is inherent to many activation functions such as sigmoid and tanh



ReLU: Rectified Linear Unit

Rectifier non-linearity
$f(z) = \max(0, z)$

Pre-activation $z = \mathbf{a} \cdot \mathbf{w} + b$, hence
$\max(0, \mathbf{a} \cdot \mathbf{w} + b)$

Advantage of ReLU over sigmoid activation functions:

- The rectifier only saturates on one side
- The rectifier has derivative 1 for large z, so no saturation
- *To some extent*, this solves the saturation and vanishing gradient problems
- Sparsely activated: It might not activate at all, often desirable

ReLU: Model Sparsity

Inactivation (saturation) on one side often desirable

- Not all neurons should fire for each input
- Concise models that often have better predictive power and less overfitting
- Example: Neuron that identifies animal eyes not active for building images

Dying ReLU problem

- Once a neuron gets negative, it is unlikely to recover
- Always outputs 0 as its pre-activation remains negative
- Neuron becomes useless, large part of network doing nothing
- Possible reasons: Large negative bias, too high learning rate
- How to solve: Lower learning rate. If not, then use leaky ReLU instead

Beyond ReLU: Leaky ReLU and Parametric ReLU

Leaky ReLU: The slope for negative input is a constant, e.g., 0.001

$$f(z) = \begin{cases} 0.001z & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

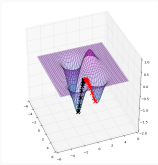
- Solves the dying ReLU problem, it does not have zero-slope parts
- Not always superior to ReLU

Parametric ReLU: The slope for negative input is a network parameter a

$$f(z) = \begin{cases} az & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

Initialising Weights and Biases

Initialising is important when minimising non-convex functions. We may get very different results depending on where we start the optimisation.



Weight initialisation for sigmoid/ReLU units

- Suppose there are D weights $w_1, \ldots, w_D$
- Draw w_i randomly from $\mathcal{N}(0, \frac{1}{D})$

Bias initialisation

- For sigmoid: Use a random value around 0
- For ReLU: Use a small positive constant

Overfitting in Neural Networks

Deep Neural Networks have a lot of parameters

- Fully connected layers with $n_1, n_2, .., n_L$ units have at least $n_1 n_2 + n_2 n_3 + \cdots + n_{L-1} n_L$ parameters
- MNIST dataset: MLP for digit recognition has 2 million parameters and 60,000 training images
- For image detection, the famous neural net by Krizhevsky, Sutskever, Hinton (2012) has 60 million parameters and 1.2 million training images
- How do we prevent overfitting in deep neural networks?
  - Beyond regularisation: early stopping, adding data, dropout

Early Stopping

Assumption: Training using iterative optimisation methods

Idea:

- Keep aside a validation set
- Measure performance of the classifier after each gradient step
- Stop training when validation error stops decreasing (reached local min)

What are the computational costs?

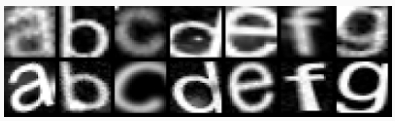
- Need to compute validation error
- Can do this every few iterations to reduce overhead

What are the advantages?

- If validation error flattens, or starts increasing, we can stop the optimisation
- Prevents overfitting (paper by Hardt, Recht and Singer, ICM 2016)

Add Data: Modified Data

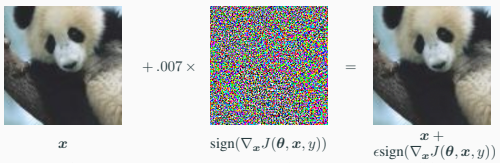
- Typically, getting additional data is either impossible or expensive
- Fake the data!
- Images can be translated/rotated slightly, their brightness changed, etc.
- Google Offline Translate trained on entirely fake generated data!



Google Research Blog (July 2015)

Add Data: Adversarial Training

- Take trained (or partially trained model)
- Create examples by modifications "imperceptible to the human eye", but where the model fails
- Linear perturbation of non-linear models:



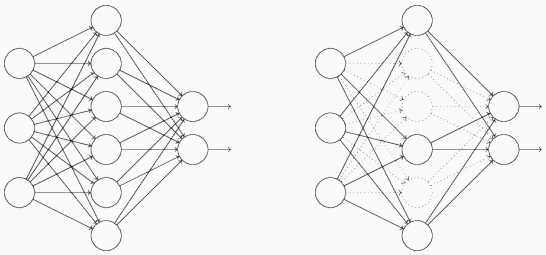
Szegedy *et al.* and Goodfellow *et al.*

Bagging (Bootstrap Aggregation)

Bagging (Leo Breiman - 1994)

- Given dataset $\mathcal{D}$, sample $\mathcal{D}_1, \cdots, \mathcal{D}_k$ of size N from $\mathcal{D}$ with replacement
- Train classifiers $f_1, \ldots, f_k$ on $\mathcal{D}_1, \ldots, \mathcal{D}_k$
- When predicting use majority (or average if using regression)
- This approach is not practical for deep networks
- Dropout: A variant of bagging that works for neural networks

Dropout: An Approach to Model Averaging



- During each training step, a fraction of the hidden units are ignored
- Weights and biases of the other units are updated in the gradient step
- Choice of units to ignore: random and different at each gradient update step
- This prevents so-called co-adaptation among different neurons
- At test time, the entire network is used
- All weights need to be scaled: halved if dropout rate was 1/2

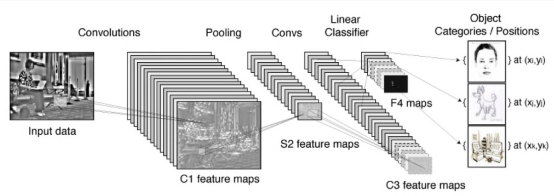
Further Ideas on Avoiding Overfitting

- Use parameter sharing (weight tying) in the model
  - Reduced computational cost
- Exploit invariances to translation, rotation
  - Important to achieve high accuracy
- Exploit locality in images, audio, text
  - Important to achieve high accuracy

Convolutional Neural Networks (convnets) do all the above!

Convolutional Neural Networks

- One of the key reasons behind the recent popularity of machine learning
- Made significant advances in image, audio, signal processing
- Led to highly accurate image recognition
- Based on works by Fukushima, LeCun, Hinton (1980s)



LeNet: One of the earliest CNNs

Convolution

Convolution operation on two real-valued functions

- Expresses how the shape of one function is modified by the other
- In neural networks, it is common to use cross-correlation instead

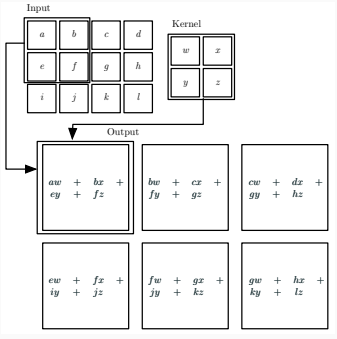
Typical case: Input 2D image I, 2D filter (or kernel) K

- Tensor I maps coordinates to, eg, black/white or RGB colour code
- Tensor K is made up of weight parameters that are learned
- K slides or convolves across I

$$\underbrace{S(i,j)}_{\text{feature map}} = (I * K)(i,j) = \underbrace{\sum_m \sum_n}_{\text{sum of}} \underbrace{I(i+m, j+n)K(m,n)}_{\text{(Hadamard) element-wise products}}$$

- It is common to have a non-linear activation function applied to S

Convolution: Example



$$\begin{aligned} S(i,j) = &I(i,j)K(0,0) + \\ &I(i+1,j)K(1,0) + \\ &I(i,j+1)K(0,1) + \\ &I(i+1,j+1)K(1,1) \end{aligned}$$

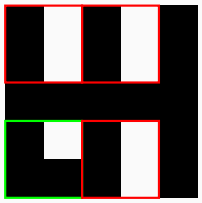
We slide filter K over input I

- one column at a time
- when finished with a row, then move to the next row

We compute S for:
$(0,0), (1,0), (2,0), (0,1), (1,1), (2,1)$

- Stride a: Sliding in steps of a columns/rows
- Padding: Add columns/rows of 0 so that output size = input size

Filters Can Detect Patterns in Images: Black-White Vertical Edges



Black = 0, White = 1

Filter 1: Detect black-white vertical edges

Use bias = -1, activation = rectifier, and filter $\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$

The output for one application is:

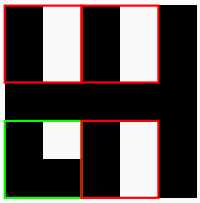
$$\max\left(0, \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} - 1\right) =$$

$$\max(0, -a + b - c + d - 1)$$

$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$: 2×2 sub-matrices

This is > 0 when $b = d = 1$ and $a = c = 0$: $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$

Filters Can Detect Patterns in Images: Black L Shapes



Black = 0, White = 1

Filter 2: Detect black L shapes

Use bias = 0, activation = rectifier, and filter $\begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix}$

The output for one application is:

$$\max\left(0, \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix}\right) =$$

$$\max(0, -a + b - c - d)$$

$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$: 2×2 sub-matrices

This is > 0 when $b = 1$ and $a = c = d = 0$: $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$

Which Patterns Can Be Detected?

Basic patterns detected by filters placed in the first layers of convnet:

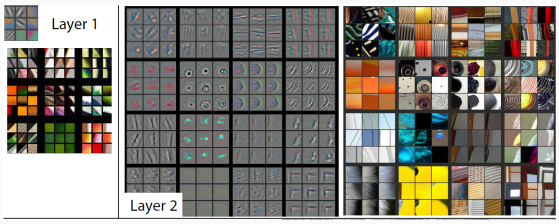
- edges
- shapes: corners, circles, squares
- textures
- curves
- objects
- colors

More sophisticated objects detectable in later layers:

- eyes
- ears
- hair or fur
- feathers
- scales
- beaks

In even deeper layers: full dogs, cats, lizards, and birds

Which Patterns Can Be Detected?



- Shapes on the left detected in the images on the right using filters
- The filters are derived automatically when training the network

Source: Visualizing and Understanding Convolutional Networks, Zeiler and Fergus (ECCV 2014)

Which Patterns Can Be Detected?

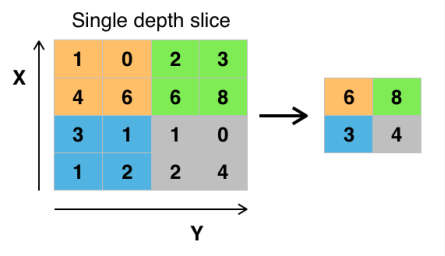
Toy Convolutional Network

Spreadsheet encoding a simple convnet developed by Jeremy Howard, fast.ai

- Input layer: 2D tensor representing image of hand-written number 7
- Convolutional layer 1 consists of two 3×3 filters
- Convolutional layer 2 consists of two 3×3 filters
- Max-pool layer reduces each 2×2 sub-matrix to one value
- Layer of dense weights, one per value in the previous layer
- Layer of dense activations aggregating previous layers
- Softmax output for multiclass classification

Spreadsheet available on OLAT. Let us play with it :)

Why Max Pooling?



- Helps generalise, combines several values into a single one
- Decreases the chance of overfitting, reduces the number of parameters
- Provides basic translation invariance and therefore more robust matching of features in the presence of small distorsions

  Once we know a specific feature is in the input (its activation value is high), its exact location is less important relative to the other features

Some Popular Convolutional Neural Networks

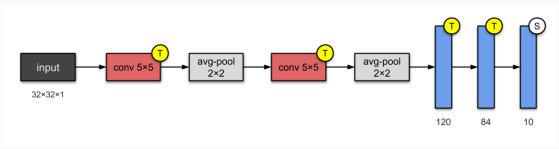
Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
VGG16	528 MB	0.713	0.901	138,357,544	23
InceptionV3	92 MB	0.779	0.937	23,851,784	159
ResNet50	98 MB	0.749	0.921	25,636,712	-
Xception	88 MB	0.790	0.945	22,910,480	126
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
ResNeXt50	96 MB	0.777	0.938	25,097,128	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

Source: `https://towardsdatascience.com/illustrated-10-cnn-architectures`

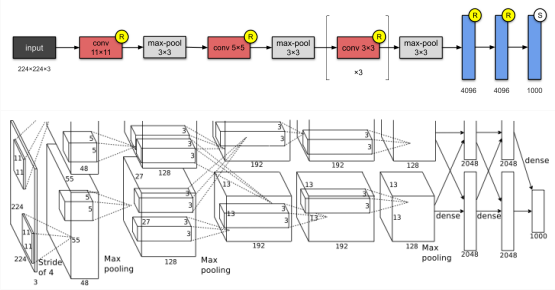
LeNet-5 (1998)



- Two convolutional layers with 5×5 filers and tanh non-linear activation
- Two 2×2 avg-pool layers
  - Softer version of max-pooling
  - For activations crossing several pooling neighbourhoods, avg-pool gives a strong signal in the middle and soft at edges
  - Gives more information on where the feature edges are localised
  - Might not extract good features if all not all inputs are needed
- Three fully connected layers, 2 with tanh, one with softmax at end
- 60K parameters in total

Source: `https://towardsdatascience.com/illustrated-10-cnn-architectures`

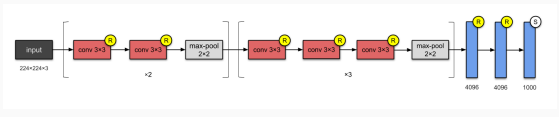
AlexNet (2012)



- Five convolutional layers with ReLU non-linear activation
- Two 3×3 max-pool layers
- Three fully connected layers, 2 with ReLU, one with softmax at end
- 60M parameters in total

Source: `https://towardsdatascience.com/illustrated-10-cnn-architectures`

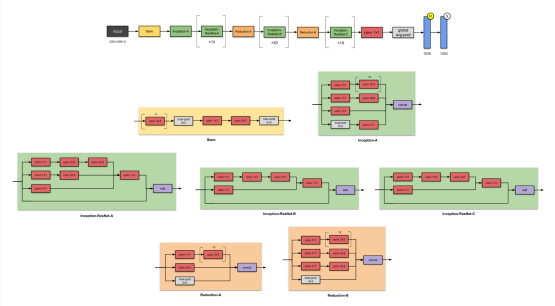
VGG-16 (2014)



- Thirteen convolutional layers with 3×3 filers and ReLU non-linear activation
- Five 2×2 max-pool layers
- Three fully connected layers, 2 with ReLU, one with softmax at end
- 138M parameters in total

Source: `https://towardsdatascience.com/illustrated-10-cnn-architectures`

Inception-ResNet-V2 (2016)



- Google-scale construction
- Much of everything, words fail me
- 56M parameters in total

Source: `https://towardsdatascience.com/illustrated-10-cnn-architectures`

How Can We Train Convolutional Neural Networks?

- As for neural networks: Gradient descent using backpropagation
- What is new: Convolutional and pool layers
  - The weights used in convolutional layers are shared as the filters convolve across the input layer
  - To do: Give the derivatives of the newly introduced weights

Consider convolutional layer between layer l and $l+1$

- Output of layer l has shape $\underbrace{m_l}_{\text{width}} \times \underbrace{n_l}_{\text{height}} \times \underbrace{F_l}_{\#\text{filters}}$
- We index the elements of the output of layer l as $a^l_{i,j,f}$
- We apply F_{l+1} filters, each a tensor of shape $W_f \times H_f \times F_l$
- We assume no zero-padding and stride of 1 in each direction
- We index the entries in this convolutional layer as $z^{l+1}_{i',j',f'}$

Formula for Entries in a Convolution Layer

$$z^{l+1}_{i',j',f'} = b^{l+1,f'} + \sum_{i=1}^{W_{f'}} \sum_{j=1}^{H_{f'}} \sum_{f=1}^{F_l} a^l_{i'+i-1,j'+j-1,f}\, w^{l+1,f'}_{i,j,f}$$

In words:

The entry z at layer $l+1$, position (i',j'), filter f' of width $W_{f'}$ and height $H_{f'}$

=

Bias at layer $l+1$ for filter f'

+

Activation at layer l, position $\left(i' + [0, W_{f'} - 1], j' + [0, H_{f'} - 1]\right)$, filter $[1, F_l]$

*

Weight at layer $l+1$, position $\left([1, W_{f'}], [1, H_{f'}]\right)$, filter $[1, F_l]$

Partial Derivative Equations for Convolutional Layers

$$z^{l+1}_{i',j',f'} = b^{l+1,f'} + \sum_{i=1}^{W_{f'}} \sum_{j=1}^{H_{f'}} \sum_{f=1}^{F_l} a^l_{i'+i-1,j'+j-1,f}\, w^{l+1,f'}_{i,j,f}$$

$$\frac{\partial z^{l+1}_{i',j',f'}}{\partial w^{l+1,f'}_{i,j,f}} = a^l_{i'+i-1,j'+j-1,f}$$

$$\frac{\partial \ell}{\partial w^{l+1,f'}_{i,j,f}} = \sum_{i',j'} \frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}}\, a^l_{i'+i-1,j'+j-1,f}$$

Backpropagation also needs to compute $\frac{\partial \ell}{\partial a^l_{i,j,f}}$ using $\frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}}$:

$$\frac{\partial z^{l+1}_{i',j',f'}}{\partial a^l_{i,j,f}} = w^{l+1,f'}_{i-i'+1,j-j'+1,f}$$

$$\frac{\partial \ell}{\partial a^l_{i,j,f}} = \sum_{i',j',f'} \frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}}\, w^{l+1,f'}_{i-i'+1,j-j'+1,f}$$

$$\frac{\partial \ell}{\partial z^l_{i,j,f}} = f'(z^l_{i,j,f}) \sum_{i',j',f'} \frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}}\, w^{l+1,f'}_{i-i'+1,j-j'+1,f}$$

Partial Derivative Equations for Max-Pooling Layer

Let $\Omega(i',j')$ be the set of (i,j) coordinates in layer l that are involved in maxpool

$$s^{l+1}_{i',j'} = \max_{i,j \in \Omega(i',j')} a^l_{i,j}$$

$$\frac{\partial s^{l+1}_{i',j'}}{\partial a^l_{i,j}} = \mathbb{I}\left((i,j) = \operatorname*{argmax}_{(\tilde{i},\tilde{j}) \in \Omega(i',j')} a^l_{\tilde{i},\tilde{j}}\right)$$

In words: The partial derivative is 1 precisely for the coordinates of the max value in the set and 0 otherwise.