Deep Learning for Image Analysis

Computer Assisted Image Analysis I

Joakim Lindblad
joakim@cb.uu.se

Uppsala University

2018-11-29

Centre for Image Analysis
Uppsala University

Outline

UPPSALA UNIVERSITET

Further reads/links

- Get going in MATLAB
  https://se.mathworks.com/help/nnet/examples/create-simple-deep-learning-network-for-classification.html
- Machine learning by Andrew Ng (Coursera)
  https://www.youtube.com/playlist?list=PLZ9qNFMHZ-A4rycgrgOYma6zxF4BZGGPW
- Stanford CS231n deep learning course by Fei Fei's group, 2016 version (skip to 2nd lecture, w. Andrej Karpathy)
  https://www.youtube.com/watch?v=g-PvXUjD6qg&list=PLlJy-eBtNFt6EuMxFYRiNRS07MCWN5UIA&index=1
  2017 version https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv
- Recent deep learning summer school in Toronto http://videolectures.net/DLRLsummerschool2018_toronto/
- Ian Goodfellows book on deep learning http://www.deeplearningbook.org/
- Stat212b: Topics Course on Deep Learning http://joanbruna.github.io/stat212b/
- fast.ai Making neural nets uncool again http://www.fast.ai/
- Yann LeCun's "Gradient-based learning applied to document recognition"
  http://ieeexplore.ieee.org/document/726791/?arnumber=726791
- An overview of gradient descent optimization algorithms http://ruder.io/optimizing-gradient-descent/
- WILDML http://www.wildml.com/
- Deep Learning Glossary http://www.wildml.com/deep-learning-glossary/
- colah's blog http://colah.github.io/
- https://icml.cc/Conferences/2017/Tutorials , https://icml.cc/2016/index.html
- https://arxiv.org
- http://www.aiindex.org/2017-report.pdf
- And many many more ...

UPPSALA UNIVERSITET

Introduction

UPPSALA UNIVERSITET

Introduction

- Deep neural networks, the current state-of-the-art in classification.
- Deep learning algorithms are consistently winning the major competitions.
- Can learn hierarchical features from the input, together with the classification.

UPPSALA UNIVERSITET

Object detection

Hui Li, et al., Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs. Jan 2016

UPPSALA UNIVERSITET

Cell segmentation

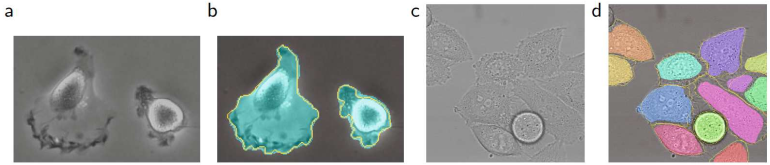
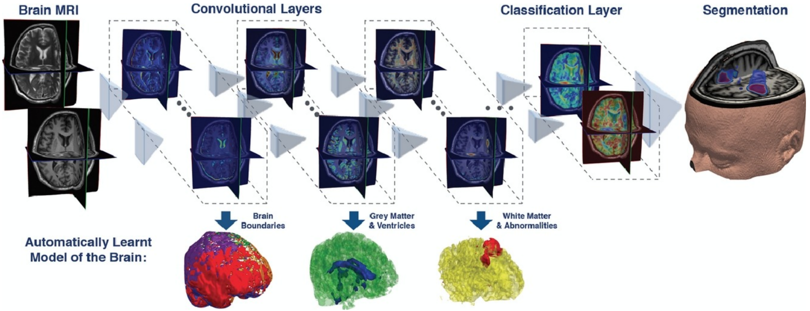


Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

Olaf Ronneberger, et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015

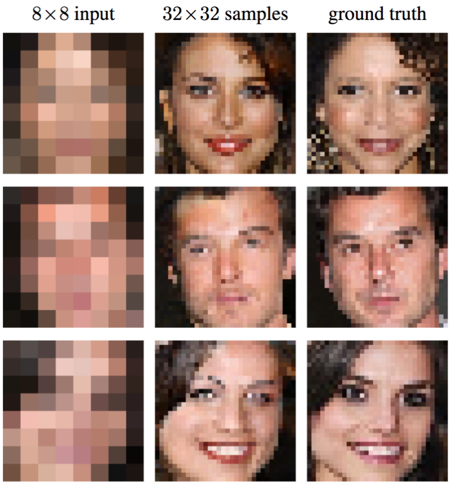
Medical image segmentation



Konstantinos Kamnitsas et al., Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. February 2017

UPPSALA UNIVERSITET

Super resolution



Ryan Dahl, et al, Pixel Recursive Super Resolution, February 2017

Face transfer/lip-syncing

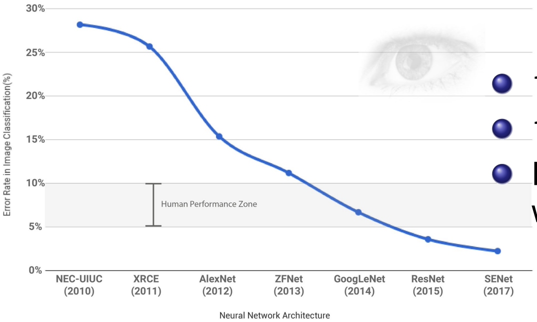
A. Bansal, S. Ma, D. Ramanan, Y. Sheikh Recycle-GAN: Unsupervised Video Retargeting. In ECCV, Sept. 2018.

Playing games

The front cover of Nature, in late January, 2016.

UPPSALA UNIVERSITET

ImageNet Large Scale Visual Recognition Challenge



Top 5 error

- 1000 classes
- 1.2 million images
- From 2012 onwards all won by deep CNNs

UPPSALA UNIVERSITET

Andrej Karpathy blog

The state of Computer Vision and AI: we are really, really far away.

Oct 22, 2012

The picture above is funny.

How does a neural network work?

UPPSALA UNIVERSITET

A linear classifier and how to train it

Problem formulation

Image classification

Switching to Stanford slides…

CS231n: Convolutional Neural Networks for Visual Recognition

Image Classification: a core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

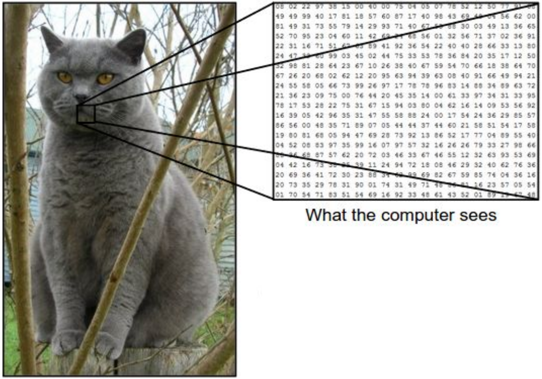
→ cat

The problem:
semantic gap

Images are represented as 3D arrays of numbers, with integers between [0, 255].

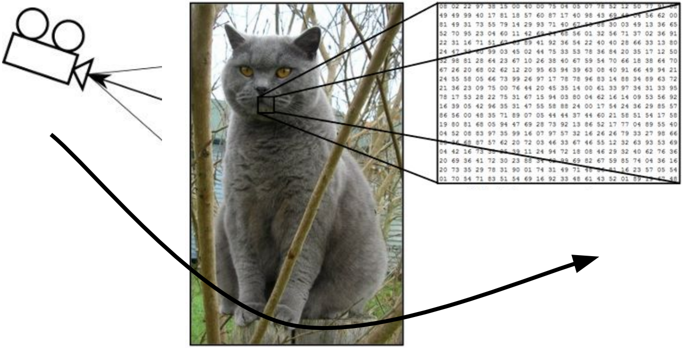
E.g.
300 x 100 x 3

(3 for 3 color channels RGB)



What the computer sees

Challenges: Viewpoint Variation



Challenges: Illumination

Challenges: Deformation

Challenges: Occlusion

Challenges: Background clutter

Challenges: Intraclass variation

An image classifier

```python
def predict(image):
    # ????
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

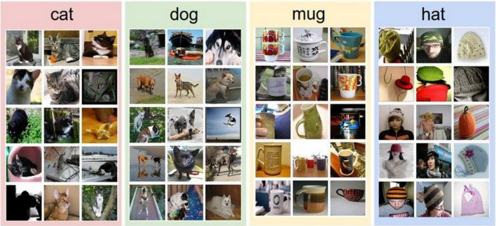
Data-driven approach:

1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```python
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set

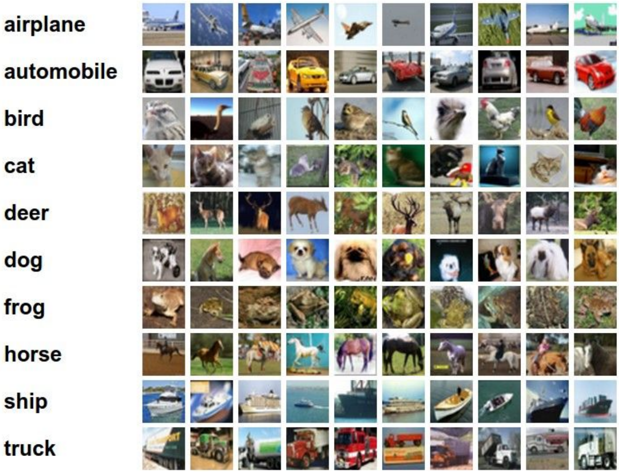


Data driven approach to image classification

Task: Design a classifier $f(x, W)$ that tells us which class $y_i \in \{1, 2, \ldots, N\}$ an image x_i belongs to.

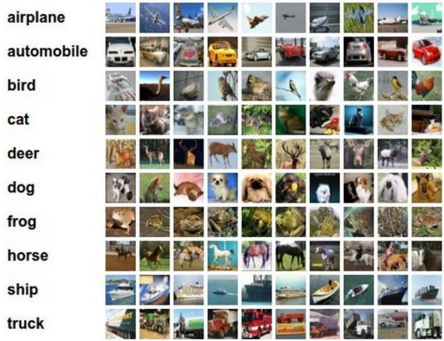
Approach:

1. Select a classifier type
   - we start with a linear (affine) classifier $y = Wx + b$
2. Select a performance measure
   - I'll mention two loss functions
3. For your data set, find the parameters W which maximize performance, that is, minimize the overall loss
   - This is the "learning" part



Example dataset: **CIFAR-10**
10 labels
50,000 training images
each image is **32x32x3**
10,000 test images.

Example dataset: **CIFAR-10**
10 labels
50,000 training images
10,000 test images.



For every test image (first column), examples of nearest neighbors in rows

Linear Classification

Parametric approach

[32x32x3]
array of numbers 0...1
(3072 numbers total)

image parameters

$f(\mathbf{x},\mathbf{W})$ →

10 numbers, indicating class scores

Parametric approach: **Linear classifier**

$$f(x, W) = Wx$$

[32x32x3]
array of numbers 0...1

10 numbers,
indicating class
scores

Parametric approach: **Linear classifier**

$$\boxed{f(x, W)} = \boxed{W}\boxed{x}$$

3072x1

10x1

10x3072

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”

Parametric approach: **Linear classifier**

$$\boxed{f(x, W)} = \boxed{W}\boxed{x} \quad \boxed{(+b)}$$

3072x1

10x1

10x1

10x3072

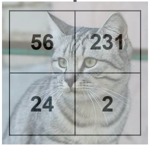
[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Stretch pixels into column



Input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \begin{array}{l} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{array}$$

Data driven approach to image classification

Task: Design a classifier $f(x, W)$ that tells us which class $y_i \in \{1, 2, \ldots, N\}$ an image x_i belongs to.

Approach:

1. Select a classifier type
   - we start with a linear (affine) classifier $y = Wx + b$
2. Select a performance measure
   - SVM loss (a.k.a. hinge loss) or SoftMax.
3. For your data set, find the parameters W which maximize performance, that is, minimize the overall loss
   - This is the "learning" part

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x,W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

			
cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 10.9)/3
= **4.6**

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: what if the sum was instead over all classes? (including j = y_i)

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$\boxed{s = f(x_i; W)}$$

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s=f(x_i;W)$$

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \boxed{\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad s = f(x_i; W)$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

scores = unnormalized log probabilities of the classes.

$$\boxed{P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$ where $$\boxed{s = f(x_i; W)}$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$\boxed{L_i = -\log P(Y = y_i|X = x_i)}$$

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

scores = unnormalized log probabilities of the classes.

$$\boxed{P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}}$$ where $$\boxed{s=f(x_i;W)}$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$\boxed{L_i=-\log P(Y=y_i|X=x_i)}$$

---

in summary: $$L_i=-\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat	**3.2**
car	5.1
frog	-1.7

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**		**24.5**
car	5.1	$\xrightarrow{\text{exp}}$	164.0
frog	-1.7		0.18

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

$\xrightarrow{\text{exp}}$ $\xrightarrow{\text{normalize}}$

unnormalized log probabilities

probabilities

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp →	unnormalized probabilities	normalize →	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

→ L_i = -log(0.13)
= **0.89**

unnormalized log probabilities

probabilities

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q: What is the min/max possible loss L_i?

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

→ L_i = -log(0.13)
= **0.89**

Data driven approach to image classification

Task: Design a classifier $f(x, W)$ that tells us which class $y_i \in \{1, 2, \ldots, N\}$ an image x_i belongs to.

Approach:

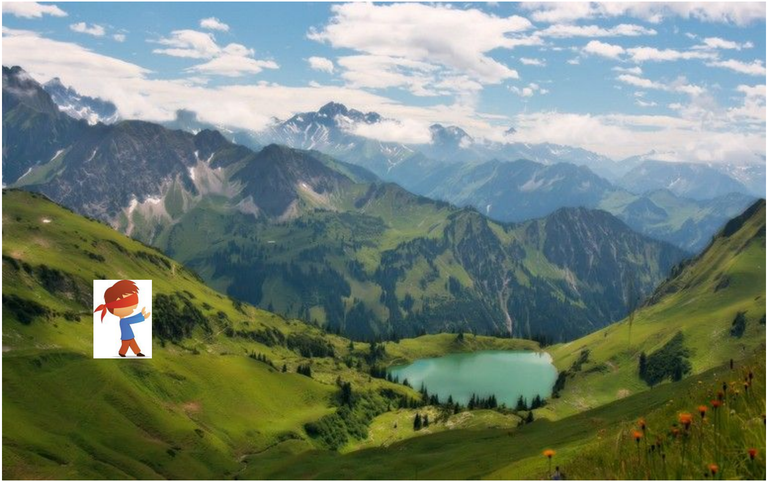
1. Select a classifier type
   - we start with a linear (affine) classifier $y = Wx + b$
2. Select a performance measure
   - SVM loss (a.k.a. hinge loss) or SoftMax.
3. For your data set, find the parameters W which maximize performance, that is, minimize the overall loss
   - This is the "learning" part

UPPSALA UNIVERSITET

Data driven approach to image classification

Minimize the loss over the training data

$$\arg\min_{W} \text{loss}(\text{training data})$$

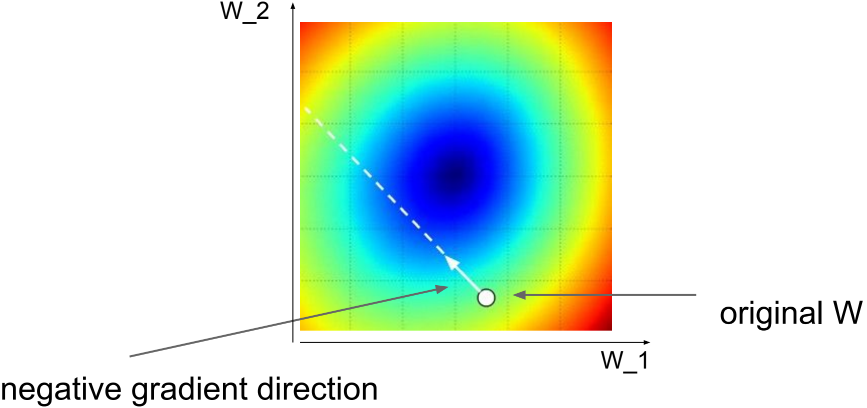


Strategy #2: **Follow the slope**

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives).


W_2
W_1
original W
negative gradient direction

Minimize the loss over the training data

$$\arg\min_{W} \text{loss(training data)}$$

using Gradient Descent to minimize the loss L:

1. Initialize weights W_0
2. Compute the gradient w.r.t. W, $\nabla L(W_k; \vec{x}) = (\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots)$
3. Take a small step in the direction of the negative gradient $W_{k+1} = W_k - \text{stepsize} \cdot \nabla L$
4. Iterate from (2) until convergence

Demo 1

Linear classifier

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

```
layer_defs = [];
layer_defs.push({type:'input', out_sx:1, out_sy:1, out_depth:2});
layer_defs.push({type:'fc', num_neurons:1, activation:'tanh'});
layer_defs.push({type:'svm', num_classes:2});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {learning_rate:0.01, momentum:0.1, batch_size:10, l2_decay:0.001});
```

UPPSALA
UNIVERSITET

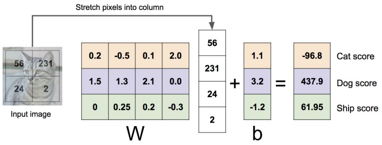
Linear classifiers and their limits

UPPSALA
UNIVERSITET

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Algebraic Viewpoint

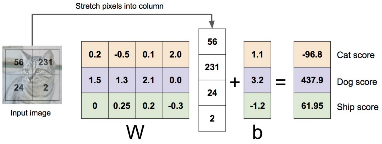
$$f(x,W) = Wx$$

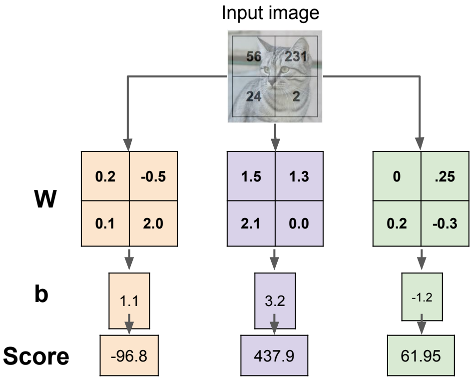


Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

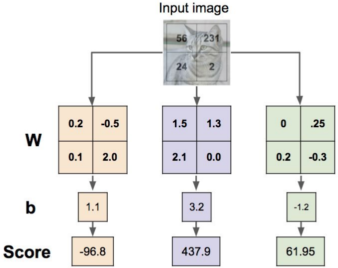
Algebraic Viewpoint

$$f(x,W) = Wx$$

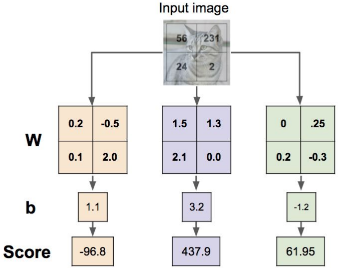




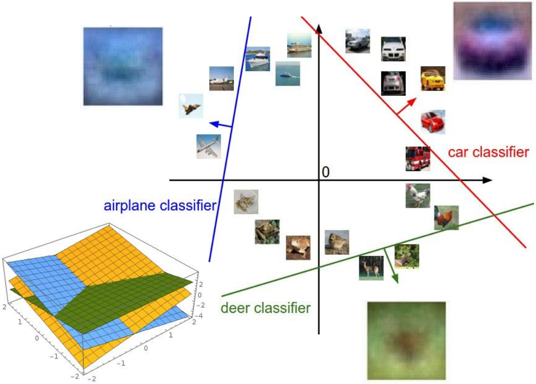
Interpreting a Linear Classifier



Interpreting a Linear Classifier: <u>Visual Viewpoint</u>



Interpreting a Linear Classifier: Geometric Viewpoint



$$f(x,W) = Wx + b$$

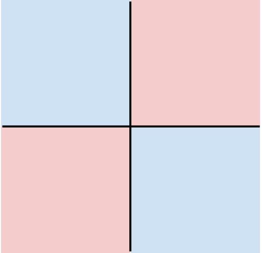
Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Hard cases for a linear classifier

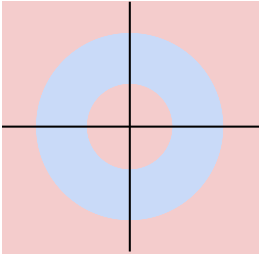
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



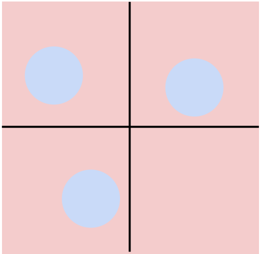
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



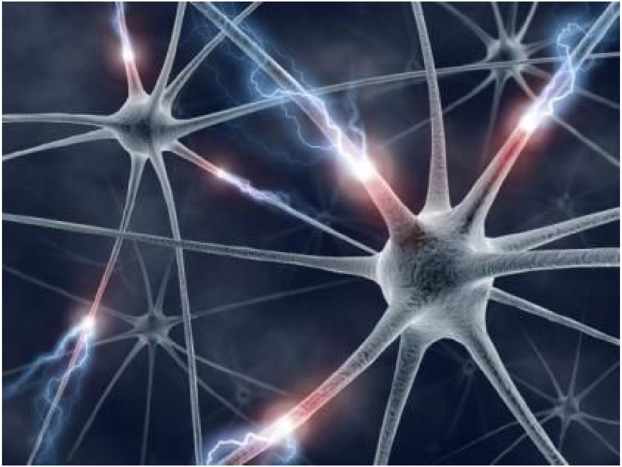
Class 1:
Three modes

Class 2:
Everything else



Neural networks – stacked non-linear classifiers

UPPSALA
UNIVERSITET



Neural Network: without the brain stuff

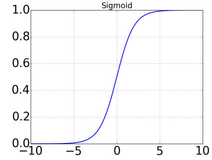
(**Before**) Linear score function: $f = Wx$

Neural Network: without the brain stuff

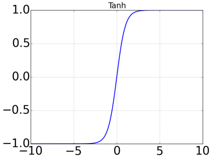
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

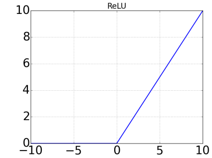
Activation functions



$$sigmoid(x) = \frac{1}{1+e^{-x}}$$



$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = 2sigmoid(2x) - 1$$



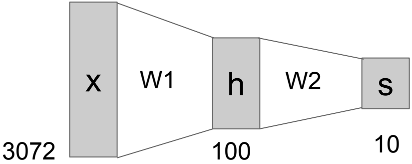
$$ReLU(x) = max(0, x)$$

UPPSALA UNIVERSITET

Neural Network: without the brain stuff

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



Neural Network: without the brain stuff

(**Before**) Linear score function: $$f = Wx$$

(**Now**) 2-layer Neural Network $$f = W_2 \max(0, W_1 x)$$

or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Simple Neural network classifier

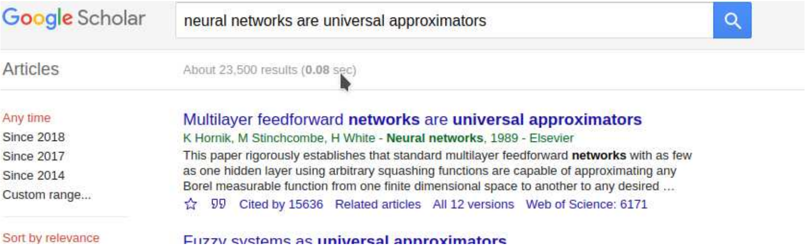
`https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html`

UPPSALA UNIVERSITET

Deep Convolutional Neural Network

UPPSALA
UNIVERSITET

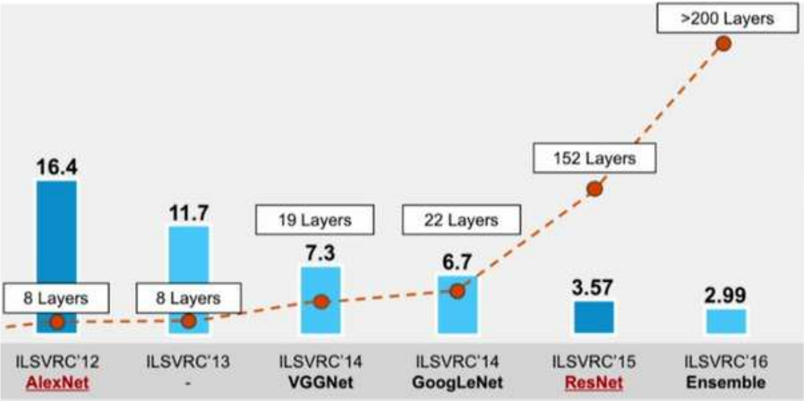
Universal approximators. . .



A feed-forward network with a **single** hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n

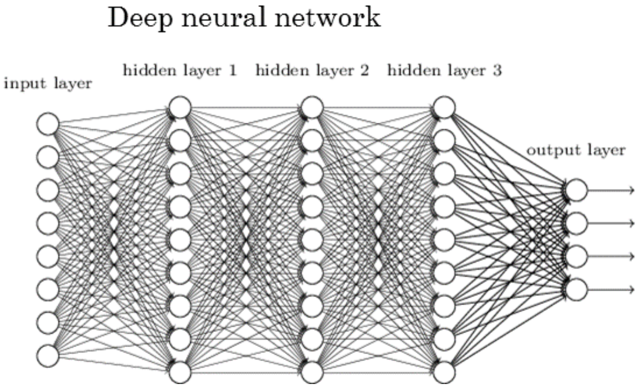
UPPSALA UNIVERSITET

Going deeper. . .



Deeper networks seem to generalize better. . .

What used to be seen as a deep neural network. . .



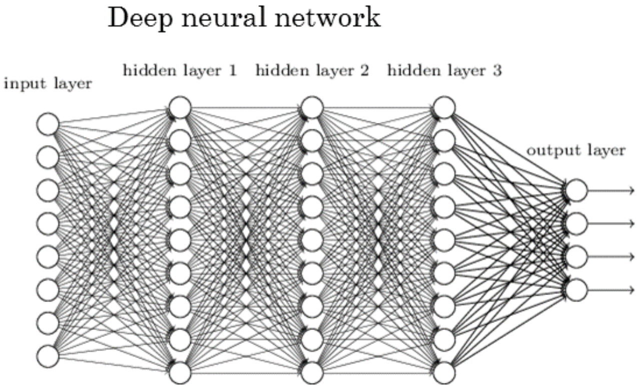
Fully connected Neural network

Src. http://www.rsipvision.com/exploring-deep-learning/

Exponential growth of the number of weights!

Can we be smarter?

What used to be seen as a deep neural network...



Fully connected Neural network

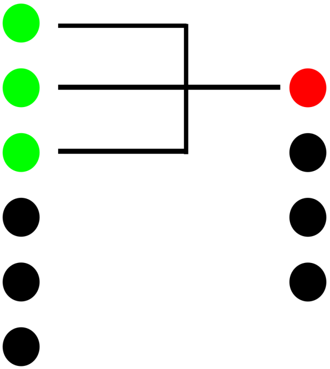
Src. http://www.rsipvision.com/exploring-deep-learning/

Exponential growth of the number of weights!

Can we be smarter? Recycle the weights!

Convolutional neural network

Sharing weights over the image



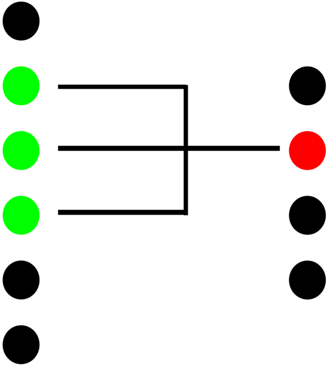
Contains convolutional layers

- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- Widely used in image analysis

UPPSALA UNIVERSITET

Convolutional neural network

Sharing weights over the image

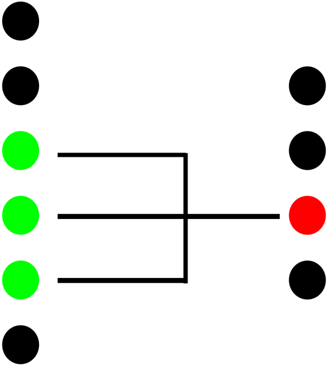


Contains convolutional layers

- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- Widely used in image analysis

Convolutional neural network

Sharing weights over the image



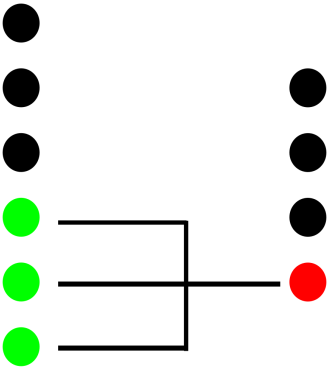
Contains convolutional layers

- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- Widely used in image analysis

UPPSALA UNIVERSITET

Convolutional neural network

Sharing weights over the image

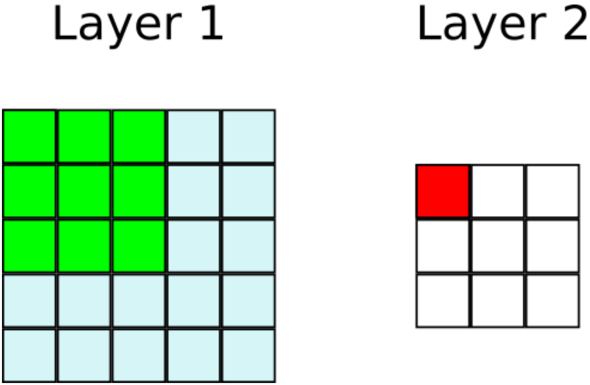


Contains convolutional layers

- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- Widely used in image analysis

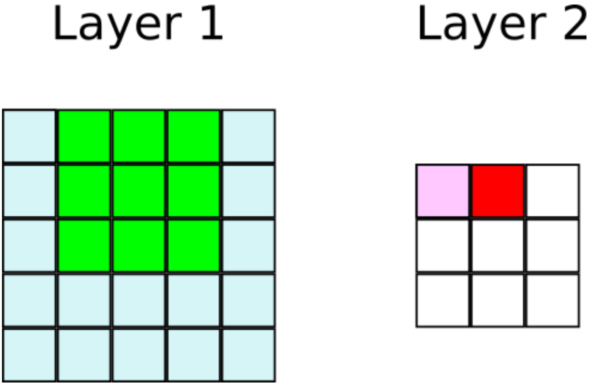
UPPSALA UNIVERSITET

2d convolutions



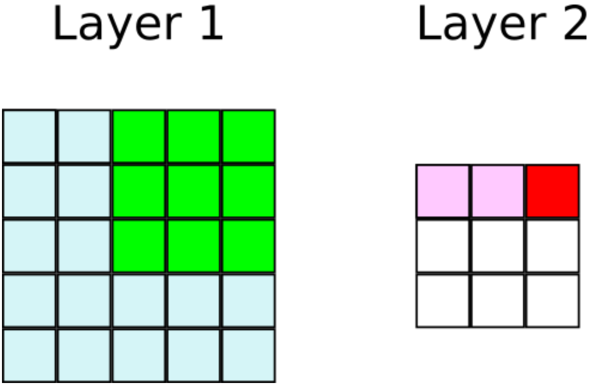
UPPSALA UNIVERSITET

2d convolutions

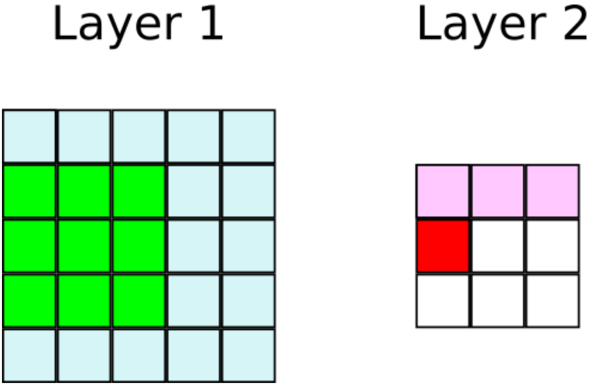


UPPSALA UNIVERSITET

2d convolutions

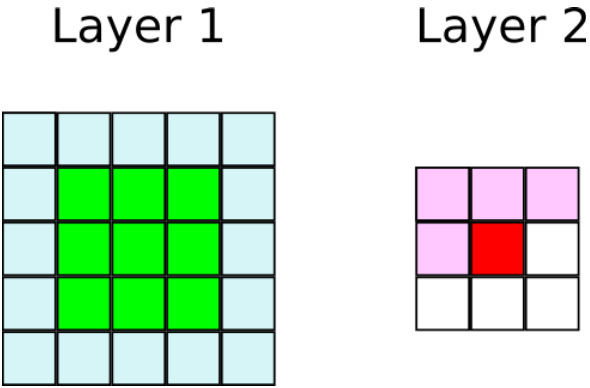


2d convolutions



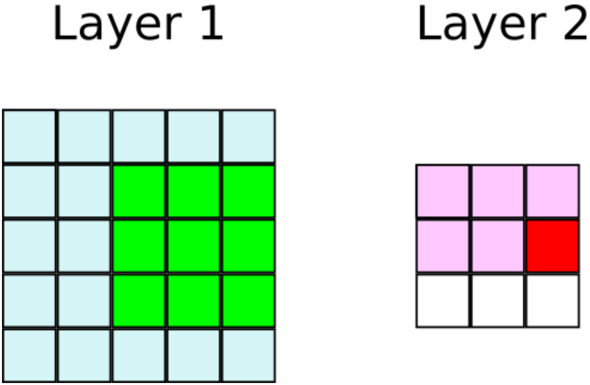
UPPSALA UNIVERSITET

2d convolutions

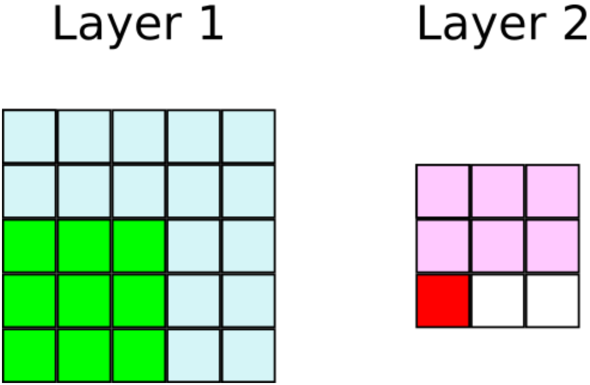


UPPSALA UNIVERSITET

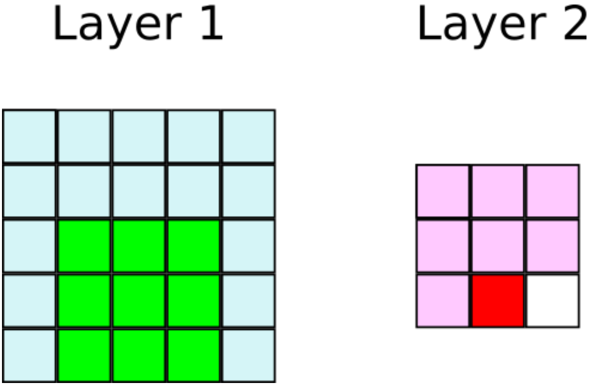
2d convolutions



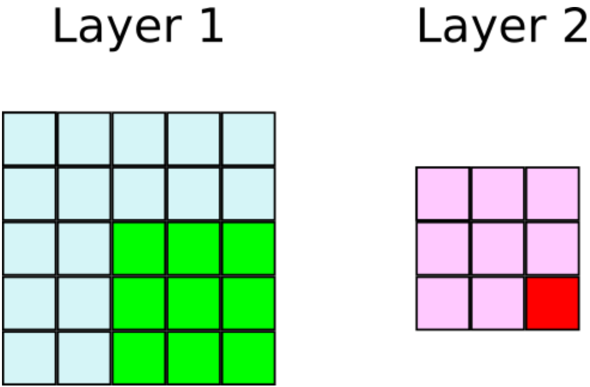
2d convolutions



2d convolutions

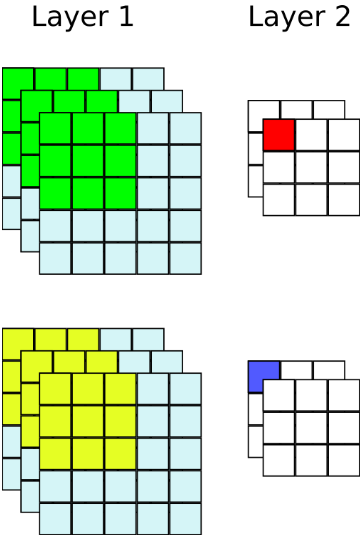


2d convolutions



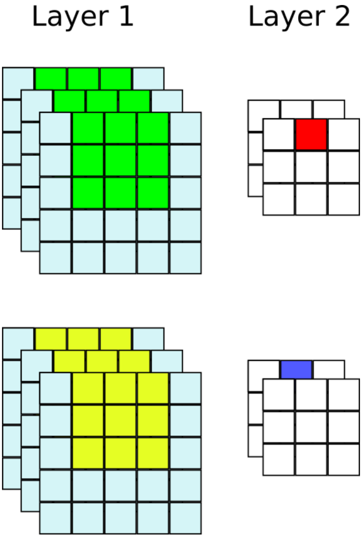
UPPSALA UNIVERSITET

3d convolutions



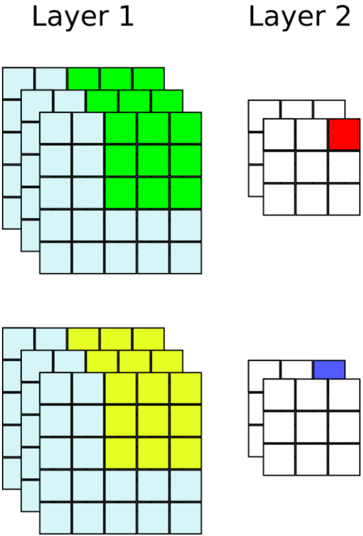
- Filter coefficients are learned from data
- Can be implemented as matrix multiplication (faster)
- Efficient GPU implementations are possible
- Implemented as tensor multiplications/additions
- Hierarchical feature extraction

3d convolutions



- Filter coefficients are learned from data
- Can be implemented as matrix multiplication (faster)
- Efficient GPU implementations are possible
- Implemented as tensor multiplications/additions
- Hierarchical feature extraction

3d convolutions

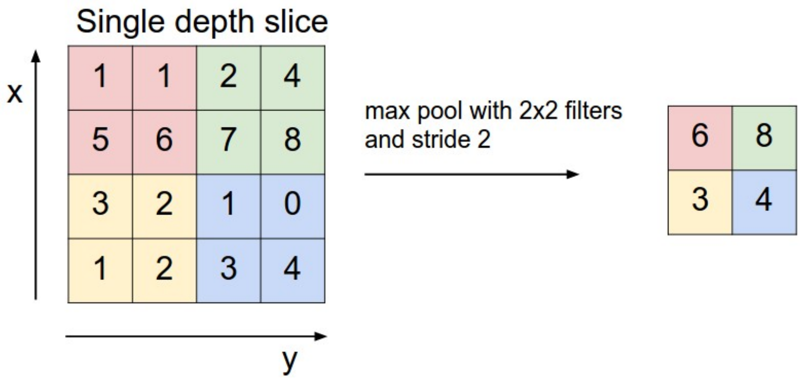


- Filter coefficients are learned from data
- Can be implemented as matrix multiplication (faster)
- Efficient GPU implementations are possible
- Implemented as tensor multiplications/additions
- Hierarchical feature extraction

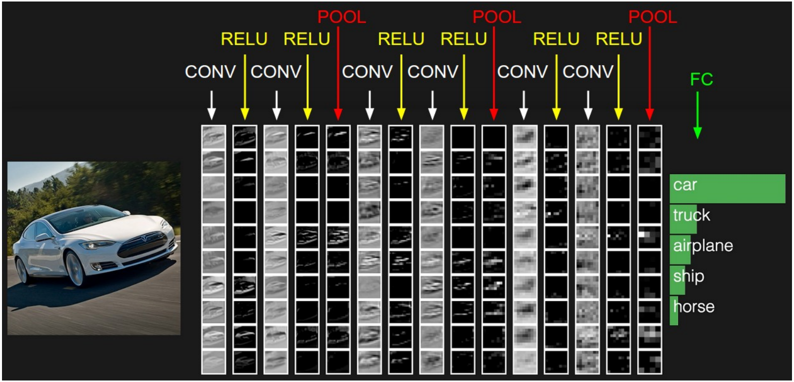
Pooling

Reduce the spatial size of the data – Subsampling

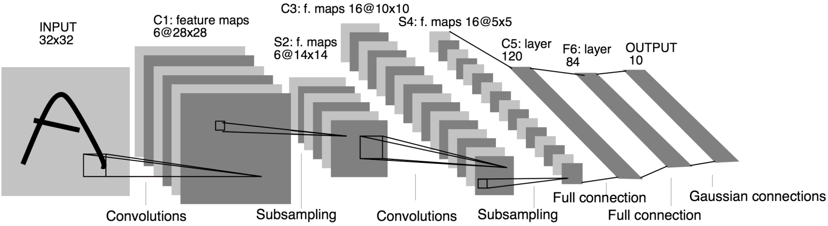
Instead of average (small important parts get lost in the crowd), pick the maximal (most important) response.



A complete Convolutional Neural Network (CNN, ConvNet)



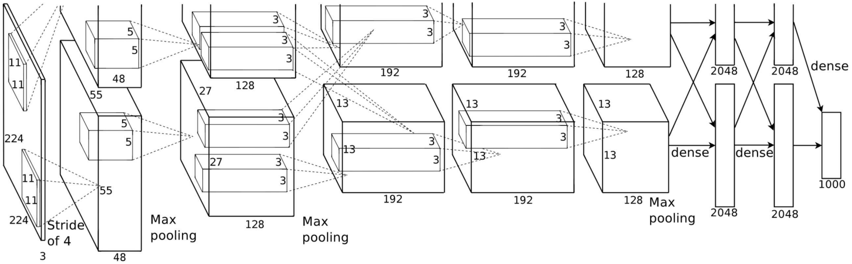
Lenet



Src. Yann LeCun, et al, Gradient-based learning applied to document recognition, 1998

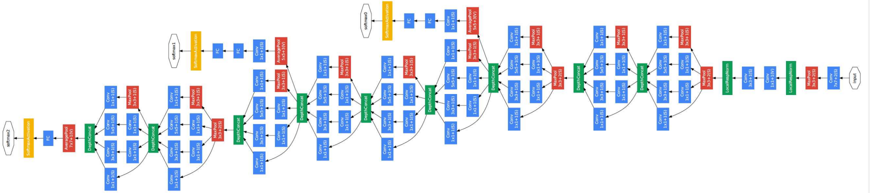
UPPSALA UNIVERSITET

Alexnet



Src. Alex Krishevsky et al, ImageNet Classification with Deep Convolutional Neural Networks, 2012

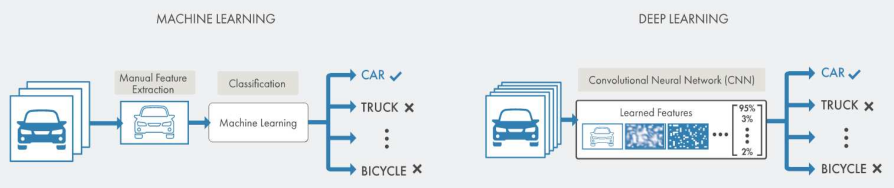
Googlenet



Src. Going deeper with convolutions

UPPSALA UNIVERSITET

Shallow vs. Deep Learning



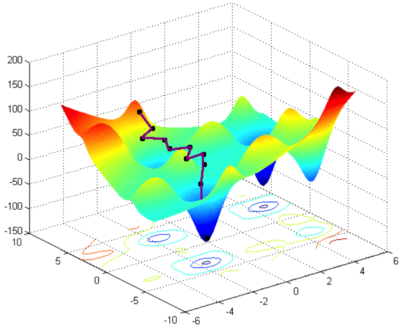
Classic “Shallow” Machine Learning vs. Deep Learning

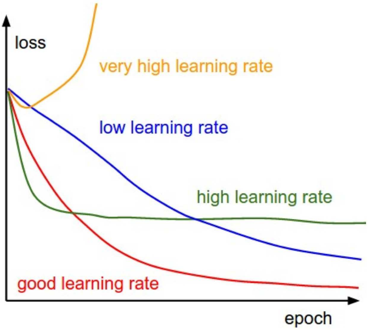
Optimization

- Choice of Loss function to minimize
- Stochastic Gradient Descent and its variants
- Initialization
- Hyper parameters
- Problems of over fitting, local minima, saddle points, vanishing gradients
- Regularization

UPPSALA UNIVERSITET

Stochastic Gradient descent





Learning rate

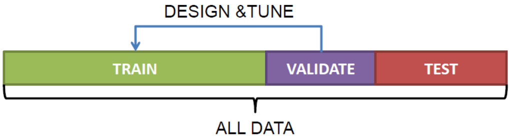
Src. http://www.phoenix-int.com/software/benchmark_report/bird.php

UPPSALA UNIVERSITET

Training, validation, testing: Classifier and its parameters

Divide the set of all available labeled samples (patterns) into:

training, validation, and test sets.

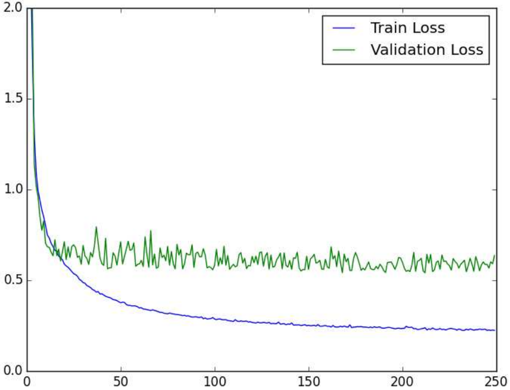


Training set: Represents data faithfully and reflects all the variation.
Contains large number of training samples.
Used to define the classifier.

Validation set: Used to tune the parameters of the classifier.
(Bias –Variance trade-off to prevent over-fitting)

Test set: Used for final evaluation (estimation) of the classifier's performance on the samples not used during the training.

Training, validation, testing



Remember to keep your test set locked away!

UPPSALA UNIVERSITET

Summary

How does a neural network learn?

- Learns from its mistakes.
- Contains hundreds of parameters/variables.
- Find the effect of each parameter when making mistakes.
- Increase/decrease the parameter values as to make less mistakes.
- Do all the above several times.

UPPSALA UNIVERSITET

How does a neural network learn?

- Learns from its mistakes. Loss function
- Contains hundreds of parameters/variables.
- Find the effect of each parameter when making mistakes.
- Increase/decrease the parameter values as to make less mistakes.
- Do all the above several times.

UPPSALA UNIVERSITET

How does a neural network learn?

- Learns from its mistakes. Loss function
- Contains hundreds of parameters/variables.
- Find the effect of each parameter when making mistakes. Back propagation
- Increase/decrease the parameter values as to make less mistakes.
- Do all the above several times.

UPPSALA UNIVERSITET

How does a neural network learn?

- Learns from its mistakes. Loss function
- Contains hundreds of parameters/variables.
- Find the effect of each parameter when making mistakes. Back propagation
- Increase/decrease the parameter values so as to make less mistakes. Stochastic Gradient Descent
- Do all the above several times.

UPPSALA UNIVERSITET

How does a neural network learn?

- Learns from its mistakes. Loss function
- Contains hundreds of parameters/variables.
- Find the effect of each parameter when making mistakes. Back propagation
- Increase/decrease the parameter values so as to make less mistakes. Stochastic Gradient Descent
- Do all the above several times. Iterations

UPPSALA UNIVERSITET

Demo

http://cs.stanford.edu/people/karpathy/convnetjs/

UPPSALA UNIVERSITET

Recap

What we have learnt so far

- A linear classifier $y = Wx$ encoding a "one hot" vector
- Two loss functions (performance measures) $L(x; W)$, hinge loss (SVM loss) and multiclass cross-entropy
  - $\mathit{softmax} = \frac{e^{s_{y_i}}}{\sum_j e^{s_{y_j}}}$, loss: $L = -\log(\mathit{softmax})$
- Touched upon Gradient descent for minimizing the loss
- Send the output through a nonlinearity (activation function) $y = f(Wx)$, e.g. ReLU.
- Send the output to another classifier, and another...
  $y = f(W_3 f(W_2 f(W_1 x)))$ = Neural network

UPPSALA UNIVERSITET

Recap

What we have learnt so far

- Training the network = find the weights W which minimize the loss $L(W;\vec{x})$
$$\arg\min_{W} L(W;\vec{x})$$
- Gradient descent to minimize the loss L:
    1. Initialize weights W_0
    2. Compute the gradient w.r.t. W, $\nabla L(W_k;\vec{x}) = (\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots)$
    3. Take a small step in the direction of the negative gradient $W_{k+1} = W_k - \text{stepsize} \cdot \nabla L$
    4. Iterate from (2) until convergence

Recap

What we have learnt so far

- How to compute the derivatives $\nabla L(W_k; \vec{x}) = (\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots)$
- Use a computational graph (impractical to write out the looong equation)
- Back propagation - "Backprop"
- Using the chain rule, derivatives are propagating backwards up through the net $\frac{\partial L}{\partial \text{input}} = \frac{\partial L}{\partial \text{output}} \frac{\partial \text{output}}{\partial \text{input}}$
  - forward: compute result of an operation and save any intermediates needed for gradient computation in memory
  - backward: apply the chain rule to compute the gradient of the loss function with respect to the inputs

Bonus material

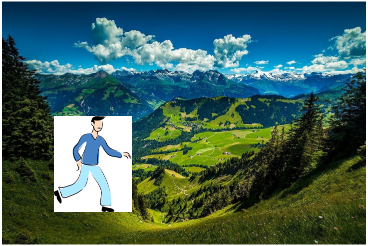
How to compute derivatives – Backpropagation

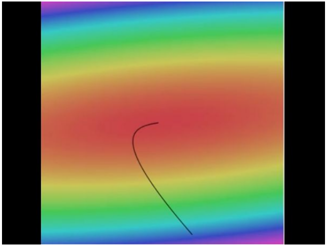
- Gradient descent to minimize the loss L:
  1. Initialize weights W_0
  2. Compute the gradient w.r.t. W, $\nabla L(W_k; \vec{x}) = (\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots)$
  3. Take a small step in the direction of the negative gradient $W_{k+1} = W_k - \text{stepsize} \cdot \nabla L$
  4. Iterate from (2) until convergence

- Backprop: Using the chain rule, derivatives are propagating backwards up through the net $\frac{\partial L}{\partial \text{input}} = \frac{\partial L}{\partial \text{output}} \frac{\partial \text{output}}{\partial \text{input}}$
  - forward: compute result of an operation and save any intermediates needed for gradient computation in memory
  - backward: apply the chain rule to compute the gradient of the loss function with respect to the inputs

UPPSALA UNIVERSITET

Optimization





```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Gradient descent

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient

Neural Network: without the brain stuff

(**Before**) Linear score function: $$f = Wx$$

(**Now**) 2-layer Neural Network $$f = W_2 \max(0, W_1 x)$$

or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Convolutional network (AlexNet)

input image

weights

loss

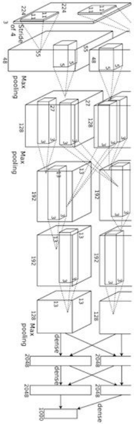


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Neural Turing Machine

input image

loss

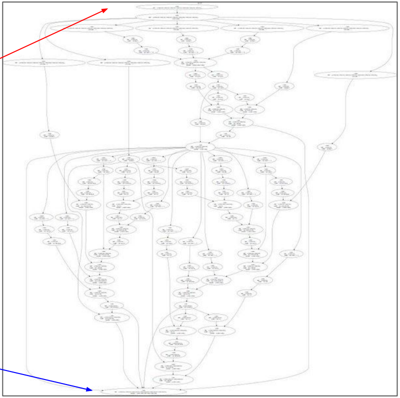


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

Neural Turing Machine

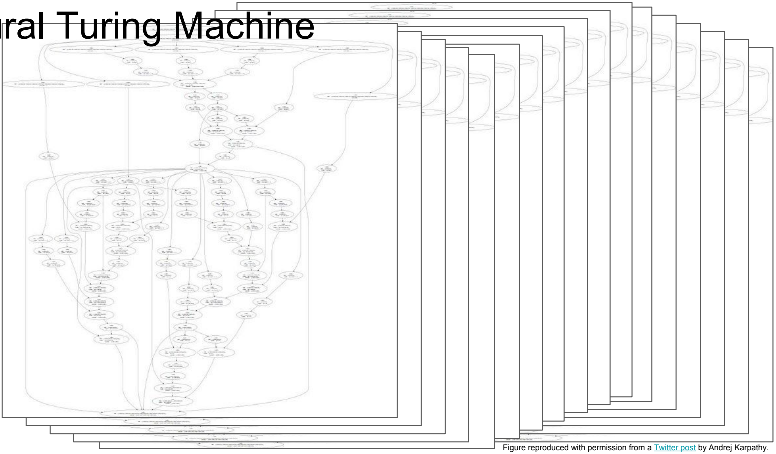
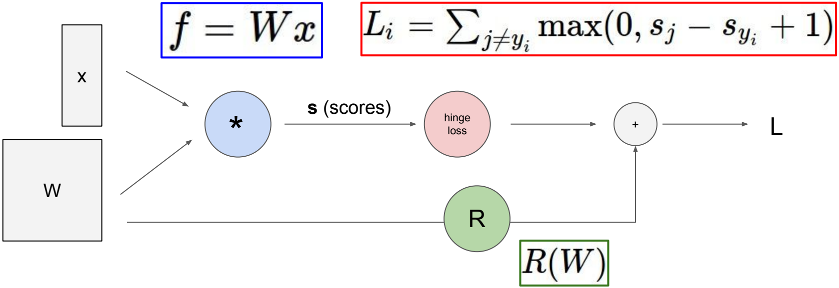


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

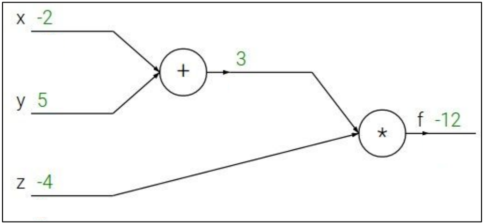
Computational graphs



Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Backpropagation: a simple example

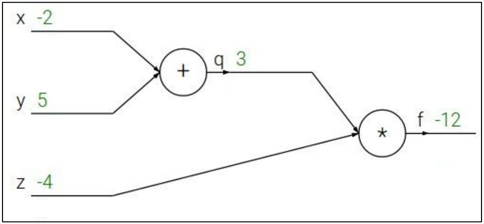
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

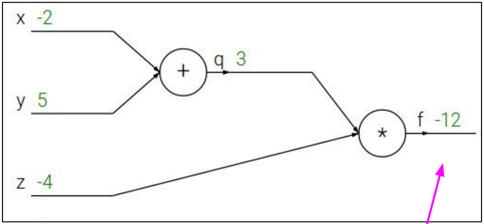
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$\dfrac{\partial f}{\partial f}$

Backpropagation: a simple example

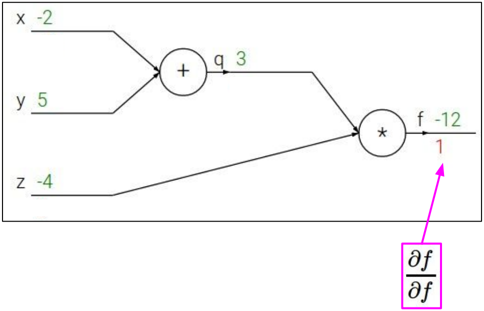
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

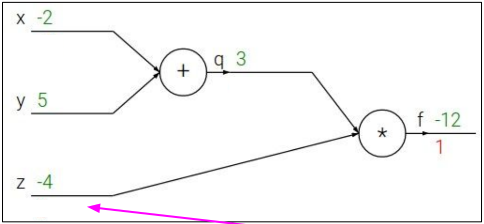
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

Backpropagation: a simple example

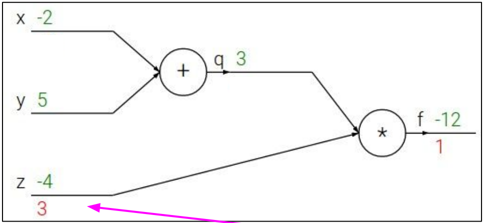
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

Backpropagation: a simple example

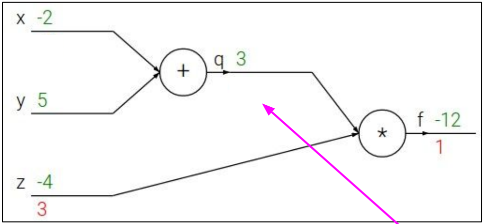
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: a simple example

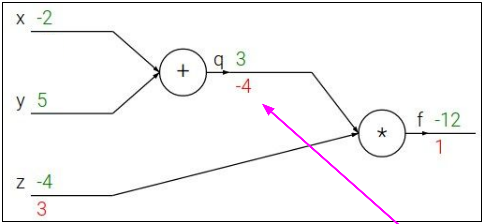
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: a simple example

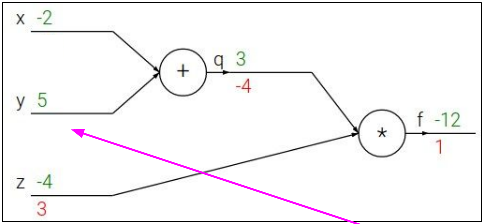
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Backpropagation: a simple example

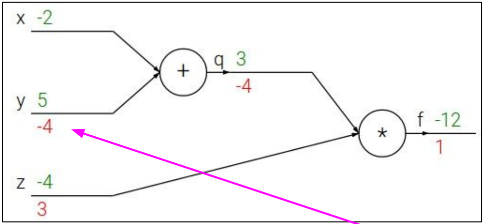
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial y}}$$

Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Backpropagation: a simple example

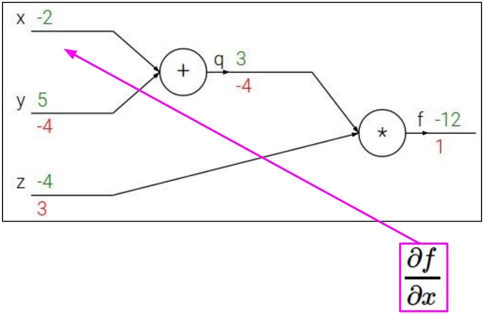
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

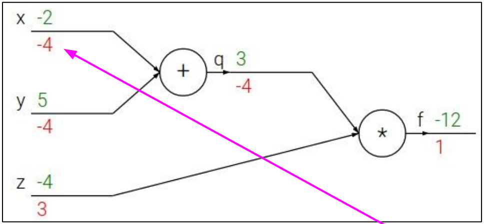
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

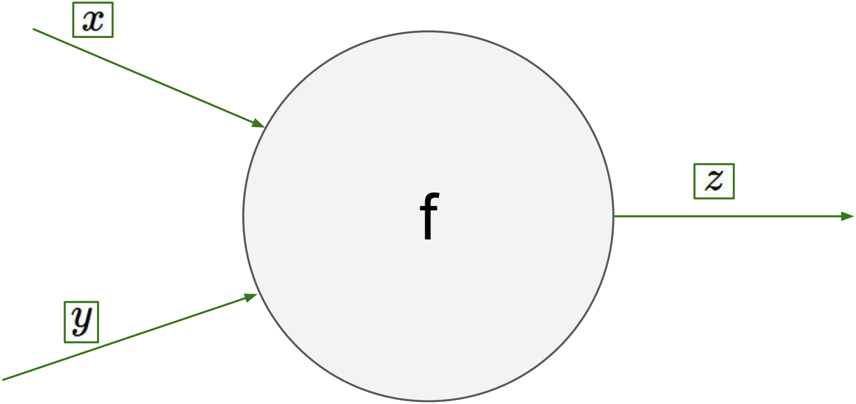
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



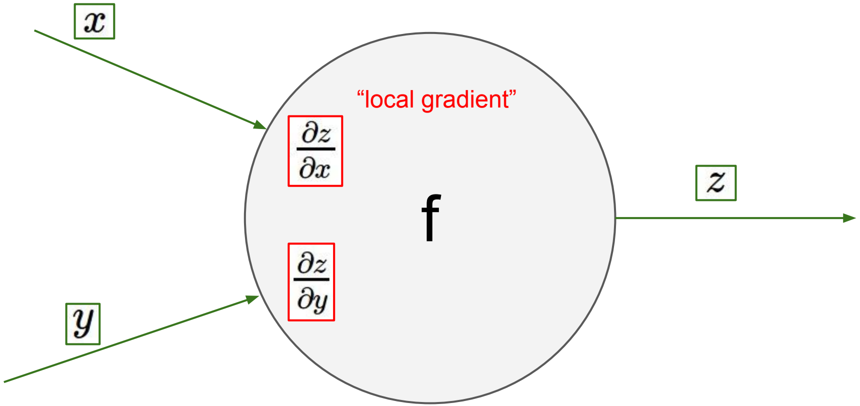
$$\frac{\partial f}{\partial x}$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

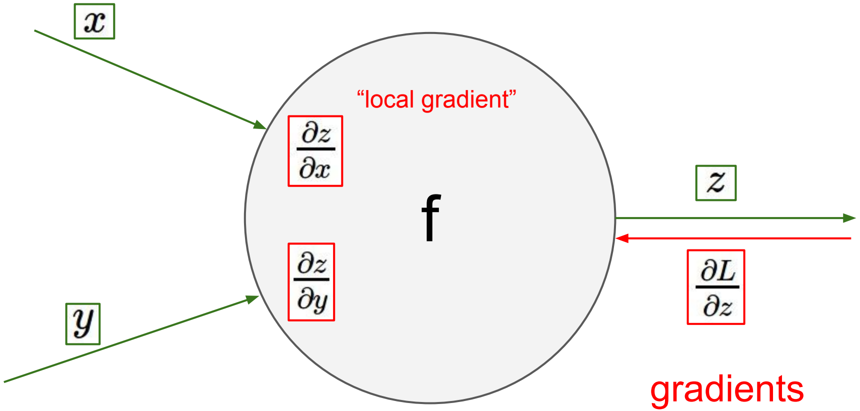


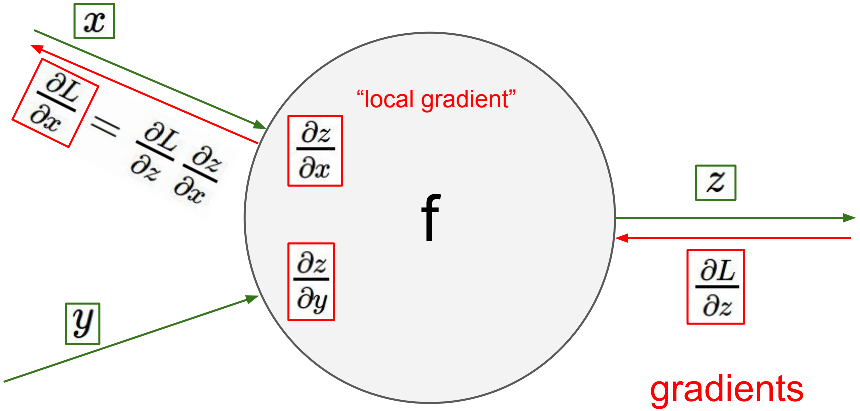
x
y
f
z

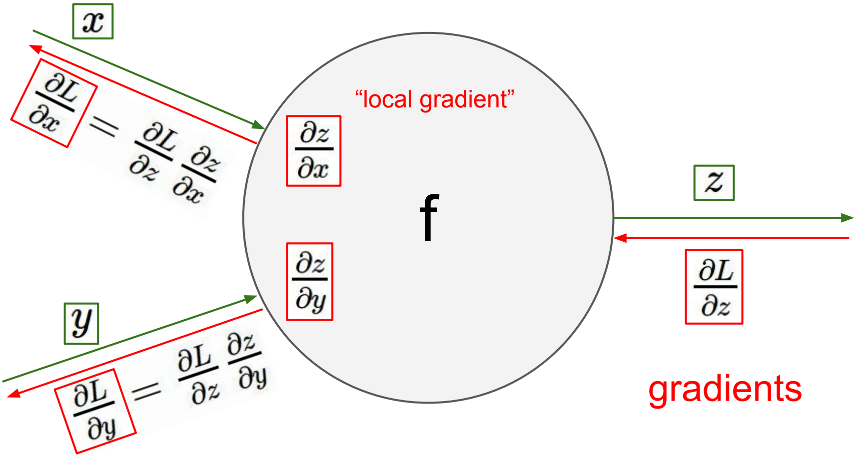


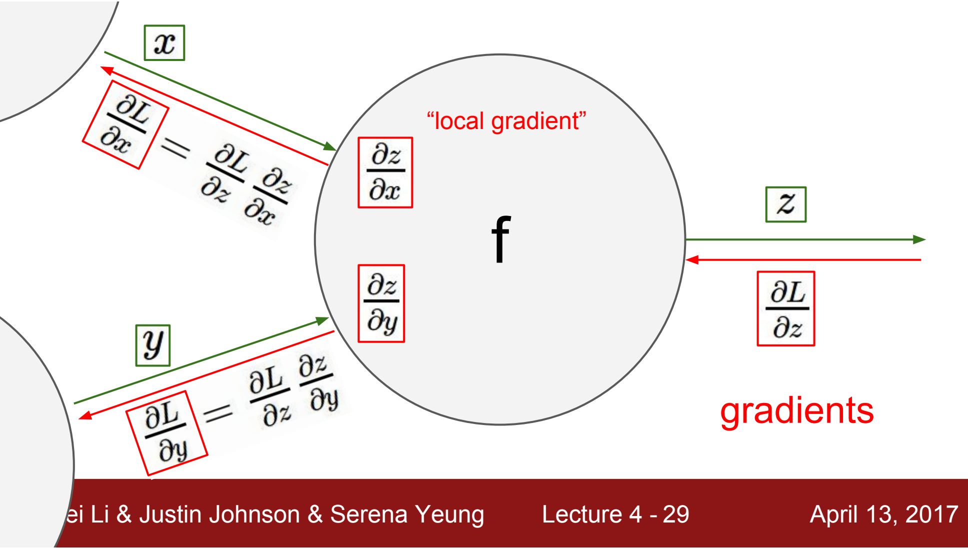
x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y

OPENING THE BLACK BOX




x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
gradients


"local gradient"
f
gradients
x
y
z
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

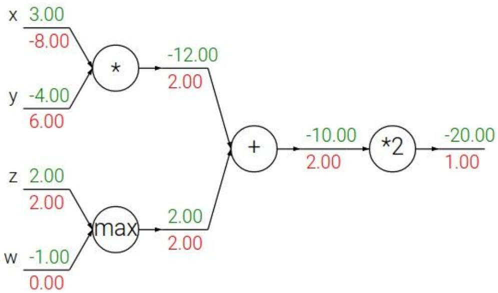

x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

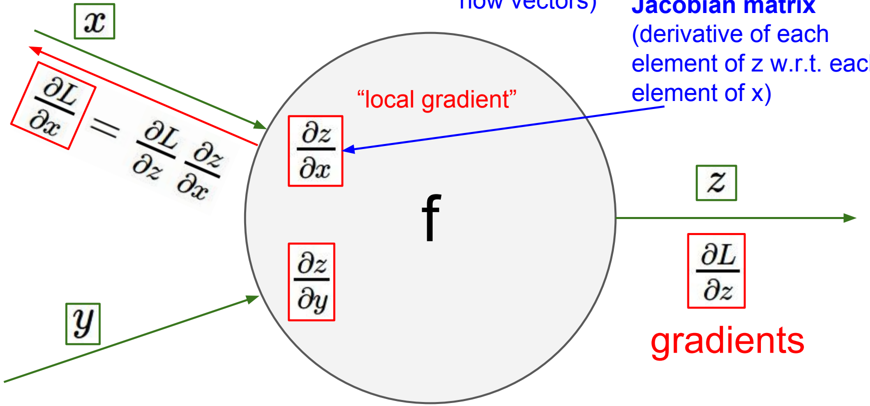
mul gate: gradient switcher



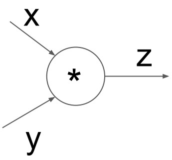
Gradients for vectorized code

(x,y,z are now vectors)

This is now the **Jacobian matrix** (derivative of each element of z w.r.t. each element of x)



Modularized implementation: forward / backward API



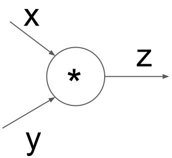
(x,y,z are scalars)

```python
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$$\boxed{\frac{\partial L}{\partial z}}$$

$$\boxed{\frac{\partial L}{\partial x}}$$

Modularized implementation: forward / backward API



(x,y,z are scalars)

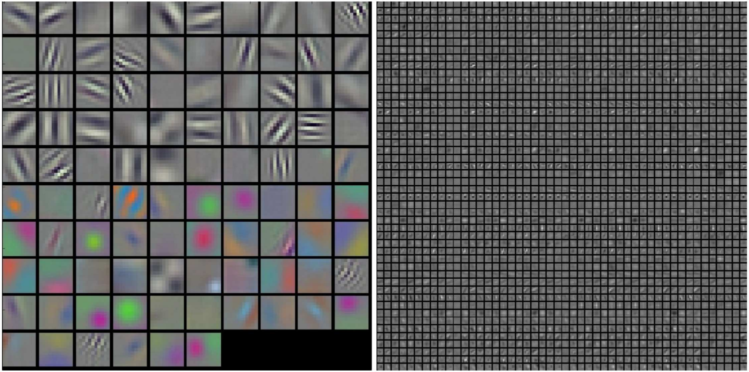
```python
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Yes you should understand backprop!

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

UPPSALA UNIVERSITET

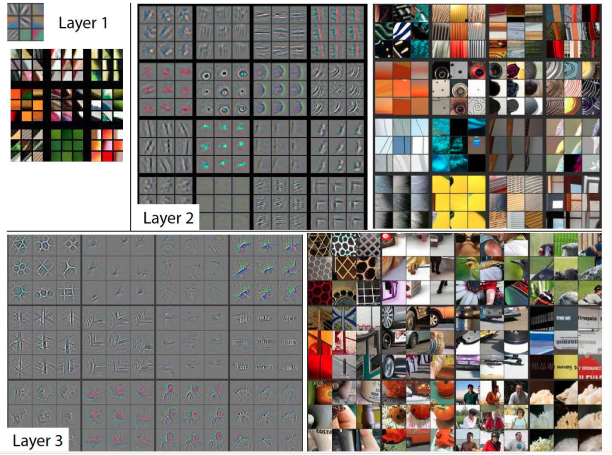
Filter visualization



First and second layer features of Alexnet

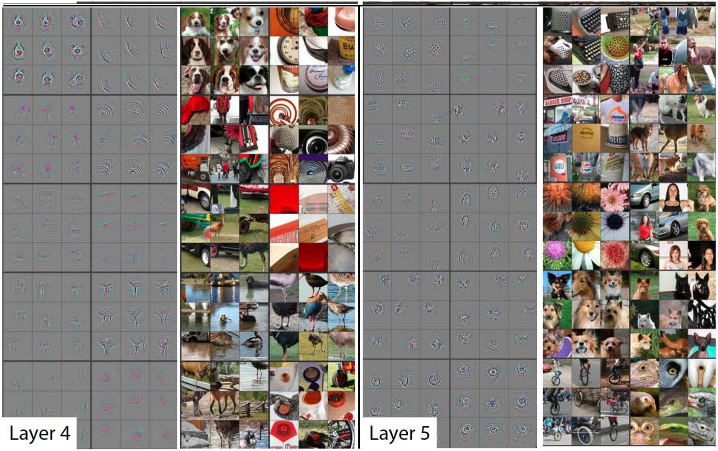
Src. http://cs231n.github.io/understanding-cnn/

Filter visualization



Src. Matthew D. Zeiler, et al, Visualizing and Understanding Convolutional Networks, ECCV 2014

Filter visualization



Src. Matthew D. Zeiler, et al, Visualizing and Understanding Convolutional Networks, ECCV 2014

DeepDream

DeepDream is a program created by Google engineer Alexander Mordvintsev

Finds and enhances patterns in images via algorithmic pareidolia, thus creating a dream-like hallucinogenic appearance in the deliberately over-processed images.

The optimization resembles Backpropagation, however instead of adjusting the network weights, the weights are held fixed and the input is adjusted.

Pouff - Grocery Trip

`https://www.youtube.com/watch?v=DgPaCWJL7XI`

UPPSALA UNIVERSITET

Further reads/links

- Get going in MATLAB
  https://se.mathworks.com/help/nnet/examples/create-simple-deep-learning-network-for-classification.html
- Machine learning by Andrew Ng (Coursera)
  https://www.youtube.com/playlist?list=PLZ9qNFMHZ-A4rycgrgOYma6zxF4BZGGPW
- Stanford CS231n deep learning course by Fei Fei's group, 2016 version (skip to 2nd lecture, w. Andrej Karpathy)
  https://www.youtube.com/watch?v=g-PvXUjD6qg&list=PLlJy-eBtNFt6EuMxFYRiNRS07MCWN5UIA&index=1
  2017 version https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv
- Recent deep learning summer school in Toronto http://videolectures.net/DLRLsummerschool2018_toronto/
- Ian Goodfellows book on deep learning http://www.deeplearningbook.org/
- Stat212b: Topics Course on Deep Learning http://joanbruna.github.io/stat212b/
- fast.ai Making neural nets uncool again http://www.fast.ai/
- Yann LeCun's "Gradient-based learning applied to document recognition"
  http://ieeexplore.ieee.org/document/726791/?arnumber=726791
- An overview of gradient descent optimization algorithms http://ruder.io/optimizing-gradient-descent/
- WILDML http://www.wildml.com/
- Deep Learning Glossary http://www.wildml.com/deep-learning-glossary/
- colah's blog http://colah.github.io/
- https://icml.cc/Conferences/2017/Tutorials , https://icml.cc/2016/index.html
- https://arxiv.org
- http://www.aiindex.org/2017-report.pdf
- And many many more ...

UPPSALA UNIVERSITET