Deep Learning for Computer Vision Part I: Introduction

Lecture Notes

Today

- Multilayered neural networks is an essential tool in computer vision and image analysis
- Enhancement, detection, segmentation, recognition,…

Machine learning for visual perception

- Discovering and leveraging **patterns**
- Pattern recognition by seeing lots of **examples**
- Predictions based on visible patterns
- Algorithms that can
  - determine **useful** patterns, e.g. shape, texture, size and constellation of objects, …
  - **useful** : helps you accomplish the task
  - ignore unrelated variation, e.g. pose, illumination, contrast, …

Two main types of learning: Supervised

- There is a **task**, e.g. segmentation, detection, recognition, …
- Examples have both **features** (images) and **labels** related to the task
- At prediction you only have the images
- Example task: segmentation
  - Appearance variation across classes
  - Ignore variance within instances of the same class
- Examples are extremely important!

Two main types of learning: Unsupervised

- Model / represent / describe the variability within the data
- There is no specific task
- Examples only have images (or features more generally)
- Prior information / regularization for a variety of inverse problems in image analysis, e.g. noise removal, super-resolution, deconvolution,

Section I: Mathematical basics

Basic notation

- Features $\mathbf{x} = \{x_1, ..., x_d\}$
  - Observed information
  - Numerical or categorical
  - Hand-crafted explicit features such as HoG or SIFT
  - Haar features, integral images
  - Raw intensities, RGB values, gray levels
  - Multispectral images can have even more raw intensity channels
- Labels

$$\mathbf{y} = \{y_1, ..., y_m\}$$

  - Unobserved information
  - Numerical (regression) or categorical (classification)
  - Semantic segmentation labels – pixel-wise
  - Object categories – image-wise
  - Unblurred intensities
  - Denoised intensities
  - Diagnosis, clinical outcomes

Basic concepts - mapping

- Model a mapping between features and labels
- We will focus on parametric mappings with parameters: θ
  $\mathbf{y} = f(\mathbf{x}; \theta)$
- Non-parametric mappings are also possible, think about nearest neighbor models

Basic concepts - learning

- Determine the "best" parameters using the examples
- Labeled examples used for learning are called "Training set"
- The examples form a paired dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$

- "Best" parameters are the ones that minimize a cost defined between predictions and "ground-truth" labels
  $\theta^* = argmin_\theta \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}_n, f(\mathbf{x}_n; \theta))$

Basic concepts – cost function

- We model the definition of “best” using the cost function
- Task dependent and ideally defined for your final goal
- **Regression**: e.g. general p-norm for regression
  $\mathbf{L}(\mathbf{y}_n, f(\mathbf{x}_n; \theta) = \|\mathbf{y}_n - f(\mathbf{x}_n; \theta)\|_p$
  - Popular choices for p=1, 2 corresponding to L1 and L2 norms.
- **Classification**: e.g. cross-entropy for classification
  $y_n \in \mathcal{C}, \mathcal{C}$: set of possible classes
  $f(\mathbf{x}; \theta) = [f_{c_1}(\mathbf{x}; \theta), f_{c_2}(\mathbf{x}; \theta), ...], \quad \sum_{k \in \mathcal{C}} f_k(\mathbf{x}; \theta) = 1$
  - where predictions are considered as class probabilities
    $\mathcal{L}(y_n, f(\mathbf{x}_n; \theta)) = \sum_{k \in \mathcal{C}} log(f_k(\mathbf{x}_n; \theta)) \mathbf{1}(y_n = k)$

Basic concepts - prediction

- The model and the best parameters are determined
- Prediction for a new sample also depends on your task
- For **regression**:
  $\hat{\mathbf{y}} = f(\mathbf{x}; \theta^*)$
  - Prediction will have errors
  - Most commonly used: Mean Squared Error (MSE), Mean Absolute Error (MAE)
    $MSE = \frac{1}{T} \sum_t^T \|\hat{\mathbf{y}}_t - \mathbf{y}_t\|_2^2, MAE = \frac{1}{T} \sum_t^T |\hat{\mathbf{y}}_t - \mathbf{y}_t|$
- For **classification**:
  $\hat{\mathbf{y}} = argmax_k f_k(\mathbf{x}; \theta^*)$
  - Prediction will have errors
  - Most commonly used: Classification error (Cerr)
    $Cerr = \frac{1}{T} \sum_t^T \mathbf{1}(\hat{\mathbf{y}}_t \neq \mathbf{y}_t)$

Notes on basic concepts

- Features:
  - Remember SIFT, Gradient Histograms, Integral images
  - Neural networks will focus on using directly the images
- Mapping:
  - Lots of research in this area
  - We will further focus on neural networks
- Cost functions:

    - There is on-going research
    - L1 loss, Adversarial loss, Dice loss for segmentation
- Error computation:
    - Very important area
    - Estimation of generalization error is a difficult task
    - Statistical methods: cross-validation, bootstrapping, …

Linear and Logistic Regression Models

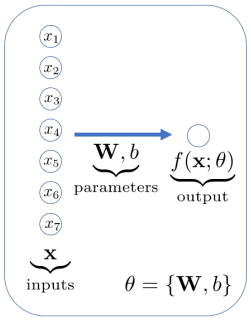
- Linear model is the main building block
- Assumes a linear relationship between features and labels
- For simplicity, let us assume one dimensional label: $\mathbf{y} = y$
  $\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = \theta^T \mathbf{x}$
- Parameters of the linear model: θ
- Linear model is for continuous labels
- Logistic regression is the extension to binary labels

Linear regression model
$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$
$y = f(\mathbf{x}; a, b) = ax + b$
$y = f(\mathbf{x}; \theta) = \theta^T \mathbf{x}$
$\theta^* = argmin_\theta \sum_{n=1}^N \|y_n - \theta^T \mathbf{x}\|_2^2$

Logistic regression model
$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$
$p(y = 1) = \sigma(\theta^T \mathbf{x})$
$p(y = 1) = f(\mathbf{x}; \theta) = \frac{1}{1+e^{-\theta^T \mathbf{x}}}$
$\theta^* = argmin_\theta \sum_n^N -log(f_k(\mathbf{x}_n; \theta))y_n - log(1 - f_k(\mathbf{x}_n; \theta))(1 - y_n)$
Logistic regression is the building block of the perceptron model

Section 2: Perceptron model and Multilayer extension
Basic perceptron model

- Model of binary classification
  $p(y = 1) = f(\mathbf{x}; \theta) = \sigma(\mathbf{W}\mathbf{x} + b)$
- Formed of two different parts
    1. Activation
       $\underbrace{a}_{\text{activation}} = \underbrace{\mathbf{W}}_{\text{weights}} \mathbf{x} + \underbrace{b}_{\text{bias}}$
    2. Nonlinearity
       $f(\mathbf{x}; \theta) = \sigma(a)$



Activation

- Linear transformation of the features
- d + 1 number of parameters $\mathbf{W} \in \mathbb{R}^{1\times d}, b \in \mathbb{R}$

Non-linearity

- Maps real line to probabilities, i.e. $\sigma : \mathbb{R} \mapsto (0, 1)$
- Element-wise application
  Sigmoid Func. $\sigma(a) = \frac{1}{1+exp-a}$
  Tangent Hyperbolic $\sigma(a) = tanh(a)/2.0 + 0.5$

Decision boundary

Notes on perceptron model

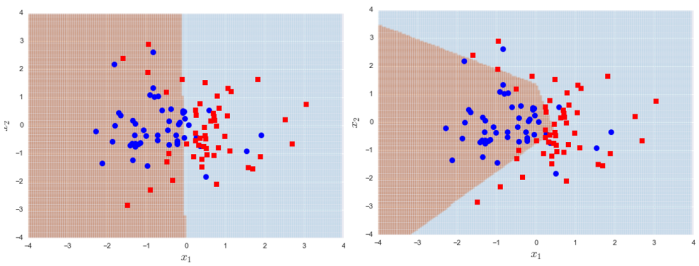
- Logistic regression is the building block
- Essentially a linear model
- Linear boundary separation and cannot model more complicated separation
- Building block for more complicated models
- We have not seen training yet, will come back that
- First let's see more complicated models

Multilayer perceptron (MLP)

- $f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$

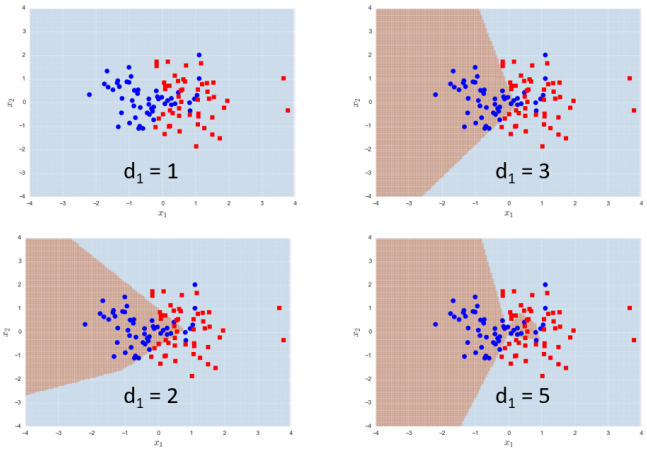
$f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$ $\quad$ $f(\mathbf{x};\theta) = f_2(f_1(\mathbf{x};\theta_1);\theta_2)$

Linear separation $\quad$ Non-linear separation



- $f(\mathbf{x};\theta) = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + b_1) + b_2)$

- $\mathbf{W}_1 \in \mathbb{R}^{d_1 \times d} \Rightarrow \sigma(\mathbf{W}_1\mathbf{x} + b_1) \in \mathbb{R}^{d_1}$
  $\mathbf{W}_2 \in \mathbb{R}^{1 \times d_1} \Rightarrow f(\mathbf{x}; \theta) \in \mathbb{R}$
  d_1 is the width of the hidden layer



- $f(\mathbf{x}; \theta) = f_2(f_1(\mathbf{x}; \theta_1); \theta_2)$
- $\theta = \{\theta_1, \theta_2\} = \{\mathbf{W}_1, \mathbf{W}_2, b_1, b_2\}$

MLP – notes on width

- $d_1 = d$ -- only nonlinear transformation
- $d_1 > d$ -- mapping to a higher dimensional space
  - often it becomes easier to separate classes in higher dimensions
  - able to model complicated decision boundaries
  - Larger number of model parameters
- $d_1 < d$ – compression / bottleneck – possible information loss
  - becomes interesting for determining low-dimensional representations, remember PCA? – will talk about this later

MLP - depth

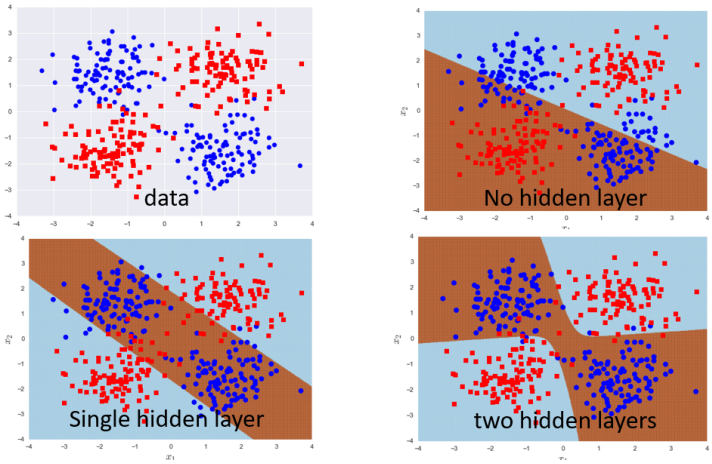
- Network with one hidden layer
  $f(\mathbf{x}; \theta) = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + b_1) + b_2)$
- Hidden layers
  $h_l = \sigma(\mathbf{W}_l h_{l-1} + b_l)$
  $h_0 = \mathbf{x}, h_2 = f(\mathbf{x}; \theta)$
- Function view
  $h_l = f_l(h_{l-1}; \theta_l) = f_l(f_{l-1}(h_{l-2}, \theta_{l-1}); \theta_l)$
  $h_2 = f(\mathbf{x}; \theta) = f_2 \circ f_1(\mathbf{x})$

MLP – increasing depth

- Network with L-1 hidden layer

- Hidden layers
  $h_l = \sigma(\mathbf{W}_l h_{l-1} + b_l)$
  $h_0 = \mathbf{x}, h_l = f(\mathbf{x}; \theta)$
- Function view
  $f_l(h_{l-1}; \theta_l) = \sigma(\mathbf{W}_l h_{l-1} + b_l)$
  $f(\mathbf{x}; \theta) = f_L \circ \cdots \circ f_2 \circ f_1(\mathbf{x})$
  $\theta = \{\theta_L, ..., \theta_1\}$

Decision boundary at different depths [width = 2 at all depths]



MLP – notes on depth

- No hidden layer – logistic regression / linear
- Increasing depth
  - Allows more complicated models
  - Leads to larger number of model parameters
  - Needs more samples to fit reliably
- May become difficult to train – will talk about it later

MLP – notes on depth and width

- Complicated interaction between depth and width
- Both allow modeling more complicated class separation
- Choices of depth and width are **architectural** design choices.
- No accepted, widely used method for automatically determining architecture for a given problem
- Problem specific – mostly trial and error-based strategy
- Engineering / intuition / art
- Prediction accuracy on a validation set

Non-linearity – more recent models

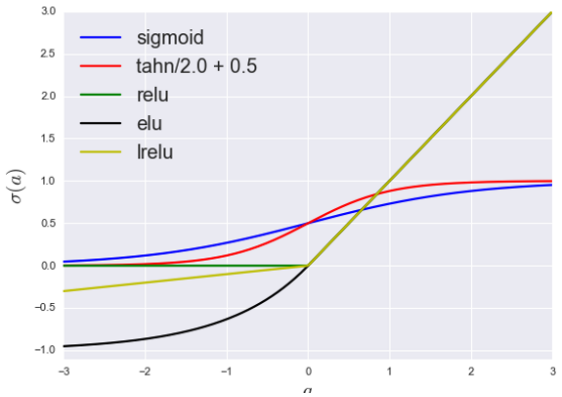
- Often used in connections between internal layers

- Element-wise application

  Rectified Linear Unit (Relu) $\sigma(a) = \begin{cases} a, & a \geq 0 \\ 0, & a < 0 \end{cases}$

  Leaky Relu $\sigma(a) = \begin{cases} a, & a \geq 0 \\ \alpha a, & a < 0 \end{cases}$

  Exponential Linear Unit (Elu) $\sigma(a) = \begin{cases} a, & a \geq 0 \\ exp(a) - 1, & a < 0 \end{cases}$



Notes on non-linear functions

- Main difference between non-linear functions is their derivatives
- Sigmoid and tanh saturate for high values, their derivatives diminish
- Relu has constant derivative for positive and 0 for negative values, it does not saturate for positive values [Hahnloser et al. 2000, Hahnloser and Seung 2001]
- Elu [Clevert, Unterthiner and Hochreiter 2015], Leaky Relu and Parametric Relu [He et al. 2015] are variations on the Relu idea.
- Relu is the activation of infinite number of binary sigmoid nodes combined together [Nair and Hinton 2010]
- Relu, Elu and Leaky Relu are yield good empirical performance

MLP – feature space view

- There is a linear model at the very final layer
  $f(\mathbf{x}; \theta) = f_L(\phi(\mathbf{x}); \theta_L) = \sigma(\mathbf{W}_L \phi(\mathbf{x}) + b_L)$
- There is a feature map that transforms the input
  $\phi(\mathbf{x}) : \mathbb{R}^d \mapsto \mathbb{R}^{d_{L-1}}$
- Automatically determined non-linear feature map

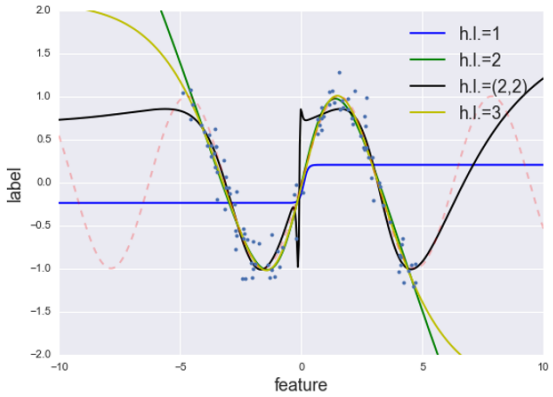
MLP – why non-linear feature map

- Non-linearity is due to the activation functions $\sigma(\cdot)$

- What happens if all activations were linear functions?
  $\sigma(a) = \mathbf{V}a + c$
  $f_l(h_l - 1) = \mathbf{V}(\mathbf{W}_l h_{l-1} + b) + c = \tilde{\mathbf{W}} h_{l-1} + \tilde{b}$
  $y = f_L \circ f_{L-1} \circ \cdots \circ f_1(\mathbf{x}) = \hat{\mathbf{W}}\mathbf{x} + \hat{b}$
- The final map is effectively linear – same as a single linear model, such as logistic regression
- Non-linearity is extremely important for complicated models

Fully Connected Classification Network

- Multilayer perceptron model with multiple outputs
- In the generic case, applies to multi-label classification
- Last layer of the network is constructed to make the network perform binary or multi-class classification
- It is common to not use any non-linear activation at the final layer
- Binary classification: $a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}$
  $p(y = 1) = \frac{1}{1+e^{-a_L}}, p(y = 0) = 1 - p(y = 1)$
- Multi-label classification – K classes: $a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}^K$
  $p(y = k) = f_k(\mathbf{x}; \theta) = \frac{e^{a_{L,k}}}{\sum_{k' \in \mathbb{C}} e^{a_{L,k'}}}, \sum_{k \in \mathcal{C}} p(y = k) = 1$
- Soft-max function
- Regression network is very similar to the classification network
- For an M dimensional regression problem
  $y \in \mathbb{R}^M \qquad f(\mathbf{x}; \theta) \in \mathbb{R}^M$
- It is common to not use any non-linear activation at the final layer



Notes on prediction and extrapolation

- Fully connected networks are very powerful generic models that can represent very complicated functions
- It is important to choose the network architecture correctly for the best prediction accuracy
- This is done on a validation set
- Models can interpolate between samples they have seen
- They do not extrapolate well – that requires knowledge about the underlying system

Information flow is forward while predicting

- Flow of information if **forward** when predicting
- The input is fed through the layers successively until the outputs
- **Feed-forward network** architecture

Training of multilayered networks

- Training / validation / test set
- Cost functions
- Backpropagation
- Stochastic optimization
- Initialization
- Avoiding over-fitting

Three sets: Training, validation and test

- Determining the best model parameters $\rightarrow$ Training set
- Determining the hyper-parameters $\rightarrow$ Validation set
- Estimating generalization accuracy $\rightarrow$ Test set

Training set

- **Main role**: Determining the best model parameters
  $\mathcal{D}_{training} = \{\mathbf{x}_n, \mathbf{y}_n\}$
  $\theta^* = argmin_\theta \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}_n, f(\mathbf{x}_n; \theta))$
- Model parameters that minimize a cost for the training set
- Size of the training set is **important**. Larger number of training examples are needed for models with larger number of parameters.

Validation set

- **Main role**: Determining the hyper-parameters
  $\mathcal{D}_{validation} = \{\mathbf{x}_n, \mathbf{y}_n\}$
- Hyper-parameters of a model
  - Architecture – number of layers, width of layers, non-linearities, …
  - Number of training iterations
  - Using / Not using regularization, regularization coefficients
  - Batch size used during training
- Ideally no overlap
  $\mathcal{D}_{training} \bigcap \mathcal{D}_{validation} = \emptyset$

Test set

- **Main role**: Estimating model prediction accuracy
  $\mathcal{D}_{test} = \{\mathbf{x}_n, \mathbf{y}_n\}$
- Absolutely no overlap
  $\mathcal{D}_{test} \bigcap \mathcal{D}_{training} = \emptyset, \mathcal{D}_{test} \bigcap \mathcal{D}_{validation} = \emptyset$
- Model parameters / hyper-parameters should **not** be changed based on test accuracy
- Ideally should come from the same distribution as training and validation sets
- Variations in distributions can reduce prediction accuracy

Notes

- Generalization accuracy, computed on the test set, will often be lower than the one on training set.
- Too much difference between test and training accuracy points out to “over-fitting”, where the model fits to noise in training set that is by chance correlated with labels
- For small datasets one often uses cross-validation, bootstrap resampling, jack knife resampling, …

Cost functions - classification

- Depends on the task
- Important to choose an appropriate cost function
- Sums are over training samples
- Binary classification
  $\mathcal{L} = \sum_{n=1}^{N} -y_n ln f(\mathbf{x}_n; \theta) - (1 - y_n) ln(1 - f(\mathbf{x}_n; \theta))$
- Multi-label classification
  $\mathcal{L} = \sum_{n=1}^{N} \sum_{k \in \mathcal{C}} -\mathbf{1}(y_n = k) ln f_k(\mathbf{x}_n; \theta)$

Cost functions - regression

- Sums are over training samples
- Mean Squared Error, L2 loss
  $\mathcal{L} \frac{1}{N} \sum_{n=1}^{N} (y_n - f(\mathbf{x}_n; \theta))^T (y_n - f(\mathbf{x}_n; \theta))$
- Mean Absolute Error, L1 loss
  $\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{M} |y_{n,j} - f_j(\mathbf{x}_n; \theta)|$

Optimization

- Whether classification or regression
  $\theta^* = argmin_\theta \mathcal{L} \qquad \theta = [\theta_1, \theta_2, ...]$
- The weights and biases of the network

- Even the simplest network may lead to complicated optimization surface, most likely non-convex
  Gradient based optimization – gradient descent (ascent if maximization)
- In its simplest form: $\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}}{\partial \theta_i}$

Gradients via chain rule

- $\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}(f(\mathbf{x};\theta))}{\partial \theta_i}$
  $f(\mathbf{x};\theta) = f_L \circ f_{L-1} \circ \cdots \circ f_1(\mathbf{x})$
  $f_l = \sigma(a_l) = \sigma(\mathbf{W}_l f_{l-1} + b_l)$
- Let's focus on $w_{ij,l}, b_{i,l}$, one of the weight and one bias in layer l, the partial derivatives are given as
  $\frac{\partial \mathcal{L}}{\partial w_{ij,l}} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}} \frac{\partial a_{l,i}}{\partial w_{ij,l}}$
  $\frac{\partial \mathcal{L}}{\partial b_{i,l}} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}} \frac{\partial a_{l,i}}{\partial b_{i,l}}$
- $\frac{\partial f_{l,i}}{\partial a_{l,i}} = \sigma'(a_{l,i})$
  $\frac{\partial a_{l,i}}{\partial b_{i,l}} = 1$
  $\frac{\partial a_{l,i}}{\partial w_{ij,l}} = f_{l-1,j}$
- define the error signal $\delta_{l,i} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}}$
- Gradients with respect to the error signal are
  $\frac{\partial \mathcal{L}}{\partial w_{ij,l}} = \delta_{l,i} f_{l-1,j}$
  $\frac{\partial \mathcal{L}}{\partial b_{i,l}} = \delta_{l,i}$

Error signal propagates backwards

$$\delta_{l,i} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}} = \left(\sum_k \frac{\partial \mathcal{L}}{\partial f_{l+1,k}} \frac{\partial f_{l+1,k}}{\partial a_{l+1,k}} \frac{\partial a_{l+1,k}}{\partial f_{l,i}} \right) \frac{\partial f_{l,i}}{\partial a_{l,i}}$$

$$= \left(\sum_k \frac{\partial \mathcal{L}}{\partial f_{l+1,k}} \frac{\partial f_{l+1,k}}{\partial a_{l+1,k}} w_{ki,l+1} \right) \frac{\partial f_{l,i}}{\partial a_{l,i}}$$

$$= \left(\sum_k \delta_{k,l+1} w_{ki,l+1} \right) \frac{\partial f_{l,i}}{\partial a_{l,i}}$$

Error signal at layer l is a function of error signal and the weights at layer l+1

$$= \left(\sum_k \delta_{k,l+1} w_{ki,l+1} \right) \sigma'(a_{l,i})$$

Backpropagation [Rumelhart, Hinton and Williams 1986, Nature]

- We start at the very final layer L
  $\delta_{L,i} = \frac{\partial \mathcal{L}}{\partial f_{L,i}} \sigma'(a_{L,i})$
  $\frac{\partial \mathcal{L}}{\partial w_{ij,l}} = \delta_{l,i} f_{l-1,j}$
  $\frac{\partial \mathcal{L}}{\partial b_{i,l}} = \delta_{l,i}$

- Propagate errors backwards and compute gradients
  $\delta_{l,i} = (\sum_k \delta_{l+1,k} w_{ki,l+1}) \sigma'(a_{l,i})$
  $\frac{\partial \mathcal{L}}{\partial w_{ij,l}} = \delta_{l,i} f_{l-1,j}$
  $\frac{\partial \mathcal{L}}{\partial b_{i,l}} = \delta_{l,i}$
- A matrix form for the error signals

$$\delta_l = \underbrace{(\delta_{l+1,1}, ..., \delta_{l+1,d+1})}_{\frac{\partial \mathcal{L}}{\partial a_{l+1}}} \mathbf{W}_{l+1} \begin{pmatrix} \sigma'(a_{l,1}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma'(a_{l,d_l}) \end{pmatrix}$$

Direct optimization can be slow

- Training with very large number of samples
  $\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}}{\partial \theta_i}|_{\theta_i^t} = \theta_i^t - \eta \sum_n \frac{\partial \mathcal{L}_n}{\partial \theta_i}|_{\theta_i^t}$
- Forward and backward pass required for each sample for each update step
- This would be very slow as the number of samples reach even thousands of images.

Stochastic gradient descent (SGD)

- Instead at every step a random batch of training samples are chosen, and parameters are updated accordingly
- A random batch of training samples are chosen at each update step
  $\mathcal{B} \subset \mathcal{D}_{training}$
- Gradient with respect to parameters are approximated
  $\sum_n \frac{\partial \mathcal{L}_n}{\partial \theta_i}|_{\theta_i^t} \approx \sum_{n_m \in \mathcal{B}} \frac{\partial \mathcal{L}_{n_m}}{\partial \theta_i}|_{\theta_i^t}$
  $\theta_i^{t+1} = \theta_i^t - \eta \sum_{n_m \in \mathcal{B}} \frac{\partial \mathcal{L}_{n_m}}{\partial \theta_i}$

Notes on optimization techniques

- Learning rate is crucial - not too low, not too high – often set by empirical tests on validation set
- Many variations present, including momentum and second order gradient approximations
  - RMSProp [Hinton]
  - Adagrad [Duchi et al. 2011]
    Basic update model with momentum
    $\Delta\theta_i = \alpha \Delta\theta_i - (1-\alpha)\eta \frac{\partial \mathcal{L}}{\partial \theta_i}$
    $\theta_i^{t+1} = \theta_i^t + \Delta\theta_i$
  - Adam [Ba and Kingma 2014]
  - …
- Different algorithms have different convergence properties, e.g.
  - Adam works well out of the box with minimal parameter tuning
  - SGD may lead to better results but needs more tuning [Wilson et al. 2017, NIPS]

Initialization

$\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}}{\partial \theta_i}|_{\theta_i^t} = \theta_i^t - \eta \sum_n \frac{\partial \mathcal{L}_n}{\partial \theta_i}|_{\theta_i^t}$

- The optimization need to start from somewhere $w_{ij,l}^0 = ?, b_{i,l}^0 = ?$
- It turns out that the initialization is quite important

Very Naïve Initialization

- Zero (constant) initialization $w_{ij,l}^0 = 0, b_{i,l}^0 = 0$
- Initialization with constant weights is problematic
- Complete symmetry in the network
- No matter what the input is the values of the hidden units will be the same
- All the weights in the one layer will get the same updates
- Effectively, one neuron thick networks
- Initializing bias with 0 is used commonly – does not cause symmetry problems

Random initialization

- Initialize weights with random values biases with zeros
- Not symmetric any more
- Different weights will get different updates
- Used quite often with random samples from uniform or normal

Other heuristics

- Random initialization is the common approach
- The main question is what distribution should one use
- Normal and uniform are the obvious choices but with what standard deviation?
- [Glorot and Bengio 2010]
  $w_{ij,l} \propto \mathcal{U}[-\frac{1}{\sqrt{d_{l-1}}}, \frac{1}{\sqrt{d_{l-1}}}]$
- [He et al. 2015]
  $w_{ij,l} \propto \mathcal{N}[0, \frac{2}{\sqrt{d_{l-1}}}], \quad w_{ij,l} \propto \mathcal{N}[0, \frac{2}{\sqrt{d_{l+1}}}]$
- Both considering that variance of a signal at the input and output of a layer should be similar

Tools

- Python is arguably the dominant programming language
- Tensorflow
- PyTorch
- A lot of fantastic tools for Matlab also exist
- For C++ lovers – Caffe

- In the exercises you will continue using python

Deep Learning for Computer Vision Part II: Convolutional Neural Networks

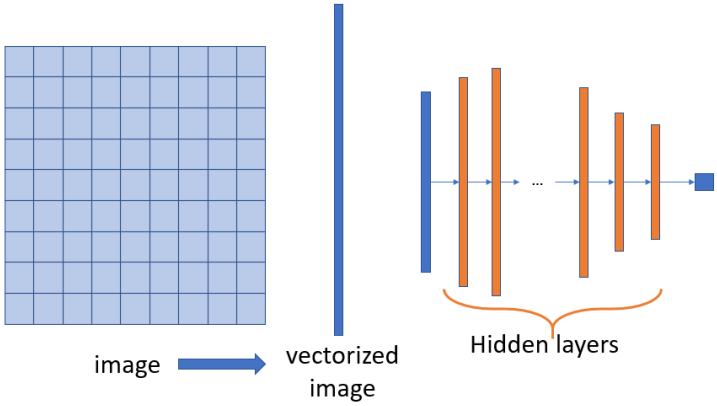
Lecture Notes

Today

- Multilayered neural networks is an essential tool in computer vision and image analysis
- Enhancement, detection, segmentation, recognition,…

We always considered x as a vector. What to do when x is an image?

Naïve approach



The activation is a bit problematic

- $a_l = \mathbf{W}_l h_{l-1} + b_l$
- The linear transformation has three issues
  1. Fully connected links leads to too many parameters.
     Assume the input is an image of size $N_0 \times M_0$, i.e.
     And the hidden layer has d_1 size, i.e. $h_1 \in \mathbb{R}^{d_1}$
     $\mathbf{W}_1 \in \mathbb{R}^{(d_1)\times(N_0 \times M_0)}$
     For example, $N_0 \times M_0 = (64, 64)$ and $(d_1) = (128)$ leads to 524288 parameters in only the first layer!
     With a huge compression from 64x64 dimensions to 128!
  2. Images are composed of a hierarchy of local statistics
  3. Lack of translation invariance

The non-linearity will remain the same

- Fully connected architecture
  $a_{l,k} = \sum_j w_{l,kj} h_{l-1,j} + b_{l,k}$
  $h_{l,k} = \sigma(\sum_j w_{l,kj} h_{l-1,j} + b_{l,k})$
- Convolutional architecture
  $a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$
  $h_{l,k} = \sigma(\sum_j w_{l,kj} * h_{l-1,j} + b_{l,k})$

Convolutional layers

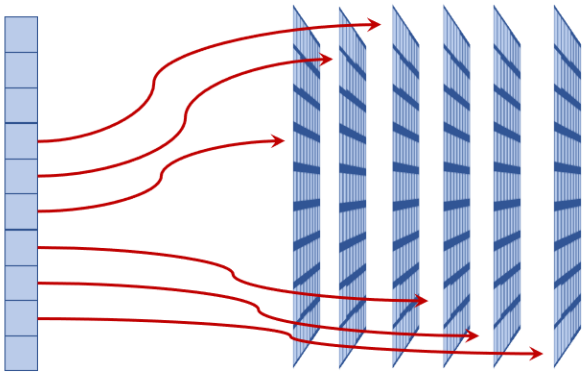
Remember convolution $w * x$

- $a_{ij} = \sum_p \sum_q x_{(i-p)(j-q)} w_{(p)(q)}$

Convolutions instead of projections

- $a_{l,k} = \sum_j w_{l,kj} h_{l-1,j} + b_{l,k}$
  - Each $h_{l-1,j}$ is a number and so is $a_{l,k}$
  - Each h_l is a vector of neurons and al is a vector of activation
  - There is a separate wl,kj that is linking each neuron $h_{l-1,j}$ to each $a_{l,k}$.
  - High dimensions of h_{l-1} and al lead to high number of weights
- $a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$
  - Each $h_{l-1,j}$ is **an image of neurons** and so is $a_{l,k}$.
  - There is a separate **convolutional kernel** linking images hl-1,j and $a_{l,k}$. Same kernel applies to the entire image of neurons
  - In the literature $h_{l,j}$ are called **channels**
  - $w_{l,kj}$ are convolutional filters.

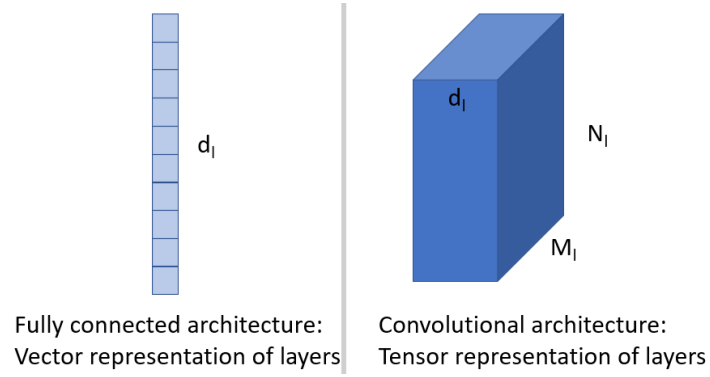
Analogy between two: each neuron becomes a channel



Fully connected
Vector of neurons form a layer

Convolutional architecture
Vector of channels form a layer

Tensor representation



Fully connected architecture:
Vector representation of layers

Convolutional architecture:
Tensor representation of layers

We can also use the same graphical representation

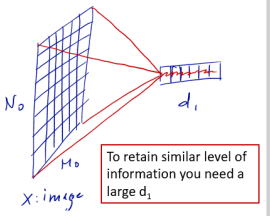
- Fully connected architecture:d_l neurons $\Rightarrow$ Convolutional architecture: d_l channels
- Connections represent convolutions
  Fully connected link: Multiplication followed by addition $\Rightarrow$ Convolutional link: Convolution followed by addition
- Same filter is used for all neurons in the same channel: **Weight sharing**

Layer size and number of parameters at the beginning

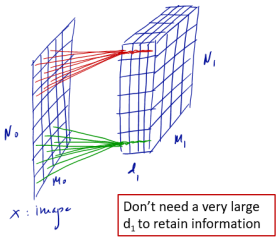
- **Larger** layers with **sparser** connections with lower number of parameters

Fully connected	Convolutional
$x \in \mathbb{R}^{N_0 \times M_0}$	$x \in \mathbb{R}^{N_0 \times M_0}$
$h_1 \in \mathbb{R}^{d_1}$	$h_1 \in \mathbb{R}^{d_1 \times (N_1 \times M_1)}$
$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (N_0 \times M_0)}$	$\mathbf{W}_1 = \{w_{1,1}, w_{1,2}, \ldots, w_{1,d_1}\}$
	$w_{1,j} \in \mathbb{R}^{k_1 \times k_2}$
	$(k_1 \times k_2)$: kernel size
	$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (k_1 \times k_2)}$

Local vs. global information gathering



Fully connected architecture:
The entire image needs to be represented in a single vector



Convolutional architecture:
Local neighborhoods are represented in a single vector

Channel size

- Channel size is linked with kernel size and the type of convolution
- $a_{ij} = \sum_p \sum_q x_{(i-p)(j-q)} w_{(p)(q)}$
  - When the kernel is placed in the image – no problem
  - When it is placed on the boundary – it is not well defined
  - Out-of-boundary values are not defined

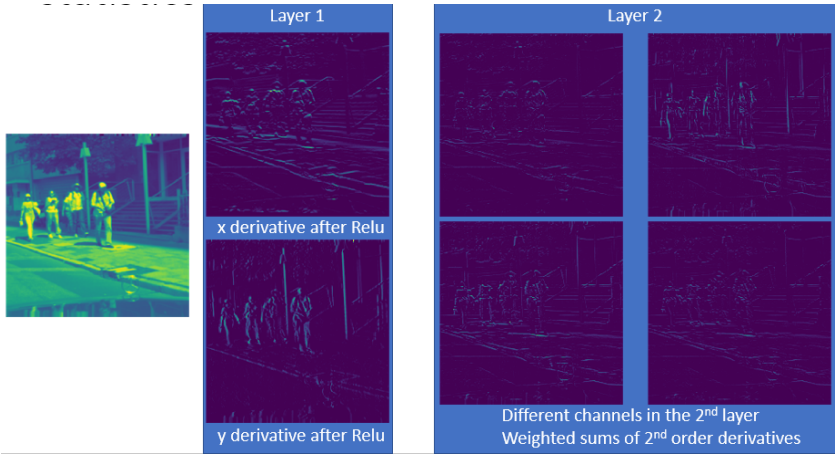
    - Two options:
        - Valid convolution: only evaluate convolution when all the elements are defined
        - Padding (Same): pad the boundaries so that result of the convolution will have the same size
- If the kernel is centered, i.e. $w_{(0)(0)}$ is the center of the kernel, then convolution can only be evaluated within the red area
    - You loose a pixel at each end of the picture
    - If the kernel center is the top left corner, then the green area is the valid area
    - You loose two pixels at the bottom and right of the image

      $M_l = M_{l-1} - k_1 + 1 \qquad N_l = N_{l-1} - k_2 + 1$
- Alternatively you can pad the image on the boundaries so that channels will have the same size across layers
    - Where you pad depends on where the center of the kernel is.
    - Commonly you would use centered kernels – padding around the image as shown on the left
    - The value you pad is a parameter, 0 is used often but you can use symmetric padding for certain applications

      $M_l = M_{l-1} \qquad N_l = N_{l-1}$

Hierarchically aggregating local spatial features

- Extracting task specific features from the images.
- As the layers progress, more global information is encoded.



Translation invariance is NOT native to fully connected networks

- These images will most likely lead to a very different activations in the hidden layer
- Rest of the network will see different activations

- In most vision applications, these images should lead to identical outputs
- What are some applications where these two images should lead to
  - Identical outputs
    - Recognition
    - Detection
  - Different outputs
    - Localization
    - Segmentation
- It is possible to teach a fully connected network to be invariant to translations by having random translations of each image in your training set
- Not the most elegant way
- Convolution operation can help – it is translation equivariant

Translation equivariance

- Equivariance: applying a transformation to the input yields the same result as applying the transformation to the output
  $f(T \circ x) = T \circ f(x)$
- Convolution is linear shift invariant, it has translation equivariance
- It does not have translation invariance
- In what application do we need
  translation equivariance $\rightarrow$ segmentation, localization
  translation invariance $\rightarrow$ recognition

Convolutional layers is a great idea but

- Translation invariance would be very useful
- Dimensionality reduction requires many parameters
  - Assume we use convolution kernels of size 5x5 and valid padding in a recognition system >> output channel size should be 1x1 (with number channels equal to the number of classes)
  - You would need many layers.
  - Or very large kernels.
  - There might another way to do this…

Strides

- In a normal convolution you only move one pixel in each direction, not skipping any pixels
- However, you can decide to skip several pixels while shifting your kernel >> instead you can skip 2 pixels
- You take a stride of 3 instead of 1
- Reduction in channel size increases

- Channel size change with valid convolutions
  $M_l = \lfloor \frac{M_{l-1}-k_1}{s_1} \rfloor + 1, \qquad N_l = \lfloor \frac{N_{l-1}-k_2}{s_2} \rfloor + 1$
- The dimension drops very quickly. The rate of drop will be faster if stride is increased.
- You lose information. Higher stride means higher loss of information.
  You do not gain translation invariance.
- If used, it is most common to use stride 2 in all directions.

Pooling layers
Pooling

- Pool information in a neighborhood
- Represents the region with one number >> summarizes information
- Applied to each channel separately
- **Max-pooling** – maximum of the activation values
- **Min-pooling** – minimum of the activation values
- Both are non-linear operations, like median filtering
- **Averaging pooling** – linear operator
- Max-pooling is the most commonly used version

Max pooling

- Represents the entire region with the neuron that achieves the highest activation
- Leads to partial local translation invariance
- Does not lead to complete translation invariance
- Often applied with strides equal to the size of the kernel

Dimensionality reduction

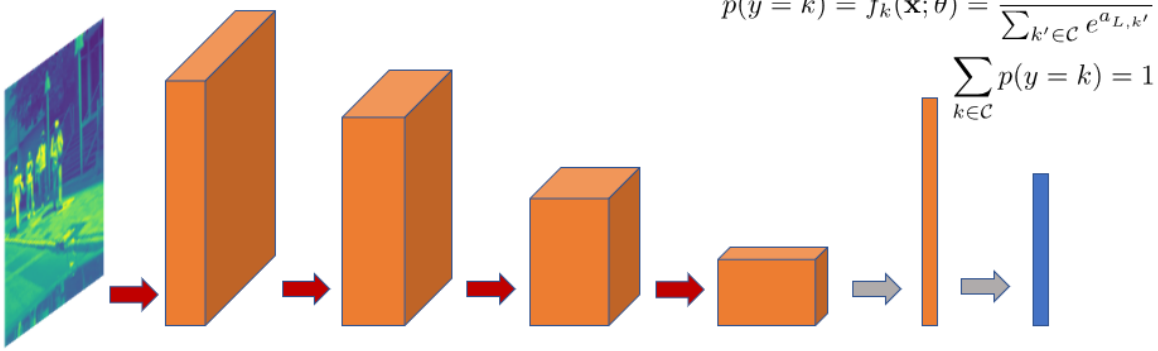
- Leads to a substantial dimensionality reduction
- Even when the pooling kernel is of size 2x2, it can halve the image!
- As the size of the pooling kernel increase, the reduction increases as well
- Non-linear dimensionality reduction
- Only the most prominent activation is transmitted to the next layer
- More advanced pooling mechanisms exist CapsuleNets [Sabour, Frosst and Hinton 2017]

Network architecture for a simple object recognition system

$$a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}^K$$

$$p(y=k) = f_k(\mathbf{x};\theta) = \frac{e^{a_{L,k}}}{\sum_{k' \in \mathcal{C}} e^{a_{L,k'}}}$$

$$\sum_{k \in \mathcal{C}} p(y=k) = 1$$



Convolution followed by non-linearity followed by max-pooling

Fully connected layer: transformation followed by non-linearity

Output
Number of neurons
equal number of classes

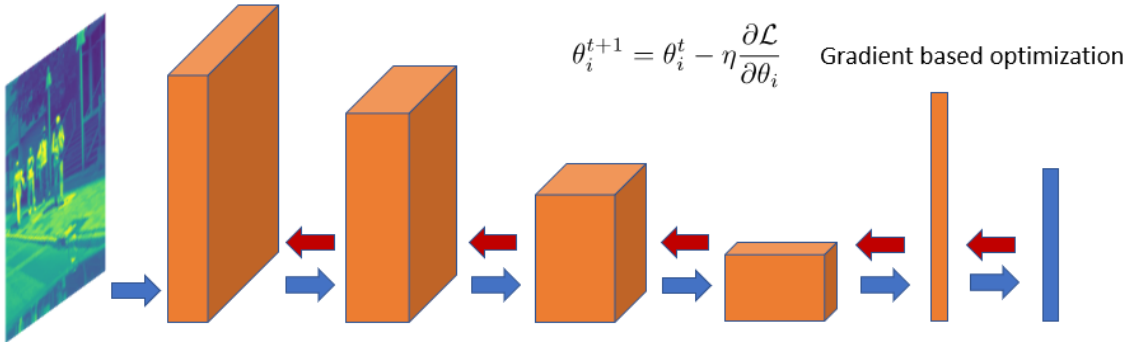
Putting it all together: Classification Convolutional neural network (CNN)

$$a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}^K \qquad p(y=k) = f_k(\mathbf{x};\theta) = \frac{e^{a_{L,k}}}{\sum_{k' \in \mathcal{C}} e^{a_{L,k'}}} \qquad \sum_{k \in \mathcal{C}} p(y=k) = 1$$

Cost function $$\mathcal{L}(y_n, f(\mathbf{x}_n;\theta)) = -\sum_{k \in \mathcal{C}} \log(f_k(\mathbf{x}_n;\theta)) \mathbf{1}(y_n = k)$$

Optimization $$\theta^* = \arg_\theta \min \sum_{n=1}^{N} \mathcal{L}\left(\mathbf{y}_n, f(\mathbf{x}_n;\theta)\right)$$

$$\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}}{\partial \theta_i}$$ Gradient based optimization



Progressively aggregating spatial information to reach a global decision

- Local features are extracted and aggregated throughout the network
- The last layers “sees” the entire image and the features encode global information

Essential blocks lead to powerful algorithms

- Convolutional layers and pooling are the essential blocks
- They have been used to create complicated networks
- First one

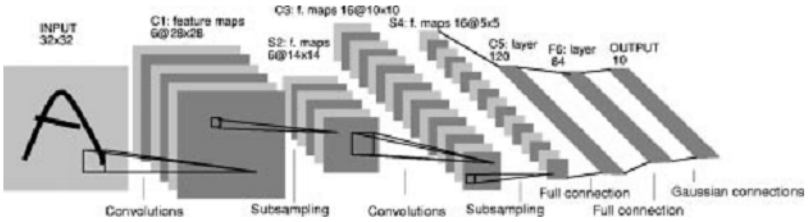


Fig. 2. Architecture of LeNet-5, a convolutional NN, here used for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical.

[Lecun, Bottou, Bengio and Haffner; Gradient-based learning applied to document recognition; **1998**]

- Then silence for a long time

Why silence

- Models had too many parameters
- They overfit for small datasets
- We did not have very large datasets
- Even for large sets, we did not have enough computation power to train the models until…

Then first this happened in 2006

- A fast algorithm to train deep belief networks by stacking restricted Boltzmann machines
- Layer-wise pre-training with contrastive divergence
- Set a state-of-the-art performance on MNIST
- However, this did not use GPUs yet

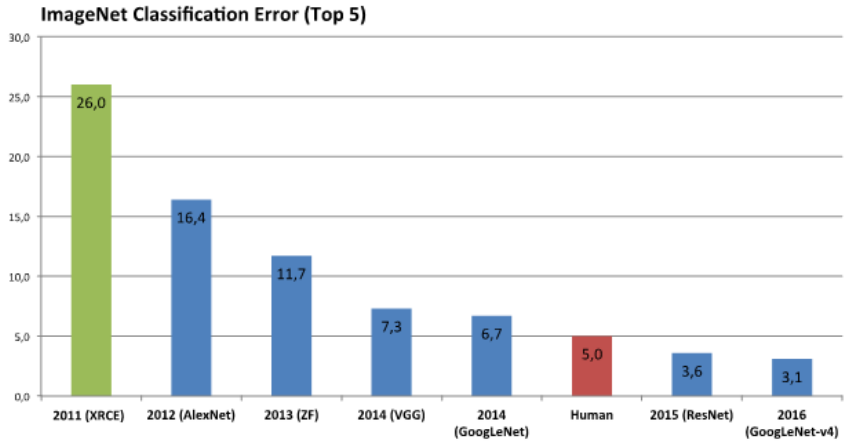
Then in 2012

- Krizhevsky et al. almost halved the error rate in the ImageNet challenge
- A simple CNN

A word about the ImageNet challenge

- Large scale object recognition challenge started in 2010
- 1.2 million training images
- 1000 object categories
- 200k validation and test images
- 2017 – 3 challenges:
  - Object localization
  - Object detection
  - Object detection from video

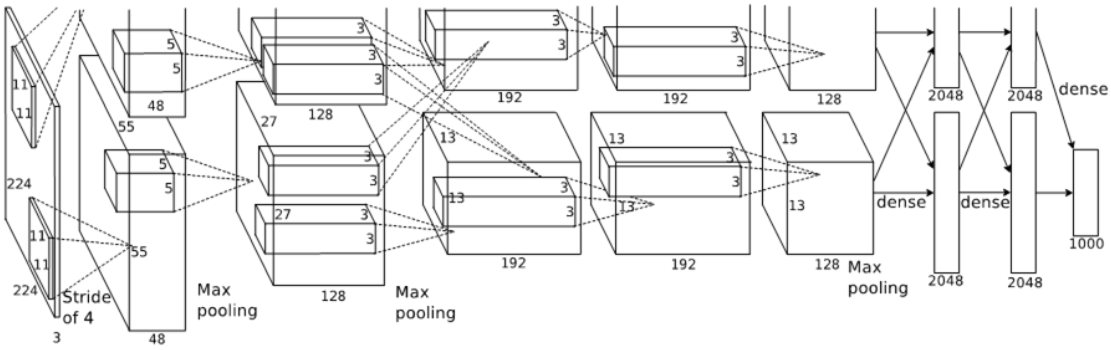
Historical evolution of the ImageNet Challenge



How the networks evolved - 2012

GPU implementation of the basic CNN using two GPUs and communication between the units

Varying size of convolutional filters in the different layers

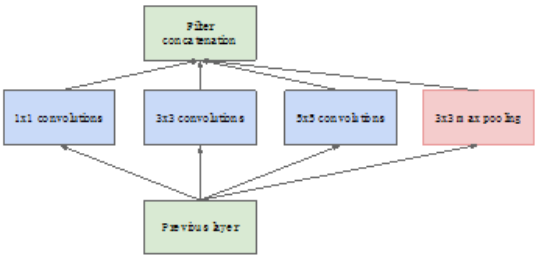


VGGNet- 2014

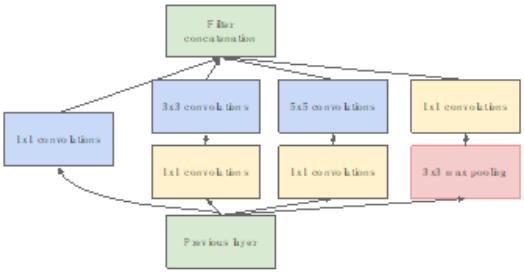
- VGG network
- First **deep** network
- 3x3 convolutional filters across the network
- Still the same principles as 1998

Inception - 2014

- Combining different kernels – a new idea
- Allowed VERY deep networks
- GoogLeNet



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2: Inception module

More on the inception module

- Conventional way was to combine multiple convolutional filters of same size.
- Inception network combines multiple convolutional filters of different sizes.
- 1x1 convolutions do not aggregate information over space but only over different channels
- Inception module aggregates all this information in one layer.
- It learns to use the necessary components from each filter size

ResNet - 2015

- Residual modeling – a new idea
- Allowed VERY deep networks

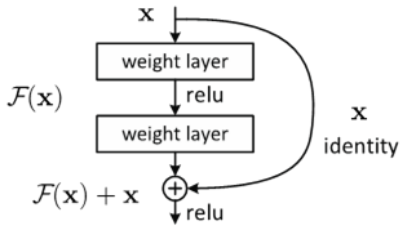


Figure 2. Residual learning: a building block.

More on Residual units and ResNet

- Conventional layer
  $h_l = f_l(h_{l-1})$
  difficult to model an identity function
- Residual layer
  $h_l = f_l(h_{l-1}) + h_{l-1}$

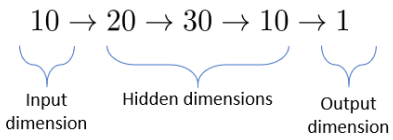
only modeling the residual allows identity maps

To help with training

- CNNs have large number of parameters
- It is important to have lots of training images to be able to train deep networks
- In smaller training size scenarios there are various tools that may help
  - Drop-out
  - Regularization
  - Data augmentation
  - Transfer learning

Number of parameters in a network

- Let us assume the following network



- Total number of parameters: 10x20 + 20 + 20x30 + 30 + 30x10 + 10 + 10 = 1170
- To determine the large number of parameters we need large number of samples
- Modern networks have many number of parameter, reaching millions

Over-fitting

- When the model has too many parameters and not enough training samples
- Model can learn noise in the training samples
- Perfect prediction on training data
- Bad prediction in validation data
- Will not be able to generalize

Over-fitting – how to overcome it?

- Two most obvious strategies:
  - Best strategy is to have **more data**
    - Most reliable strategy
    - Data collection can be expensive or not even feasible
    - Labeling can be expensive
  - Reduce the size of your network, i.e. less parameters
    - Also reliable strategy
    - It may lead to performance loss
  - Regularization
  - Drop-out

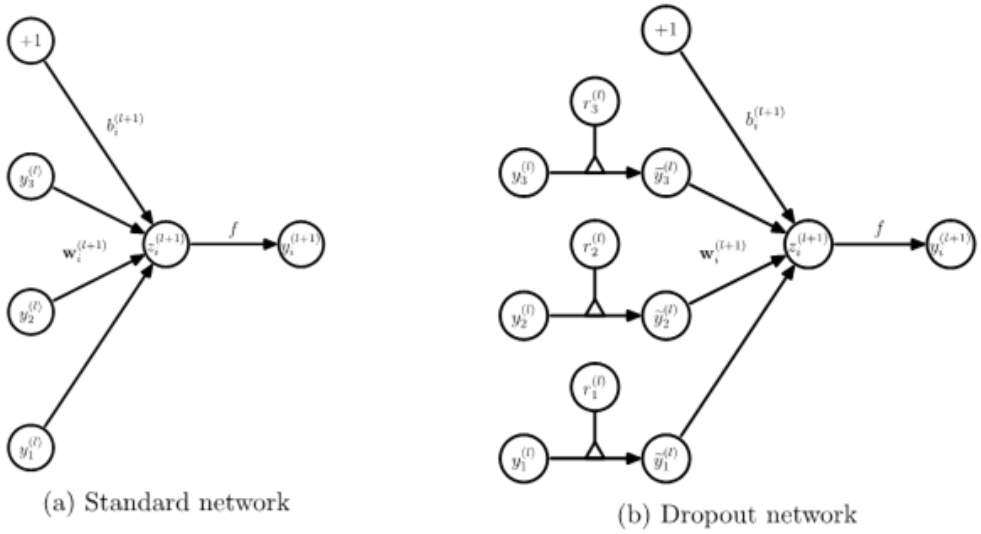
- Multi-task learning
- Data augmentation
- Transfer learning

Regularization

- Regularization [Krogh and Hertz, NIPS 1992]
- Similar strategy is taken for many other learning algorithms, ridge regression, SVM, sparse regression,…
- $min\mathcal{L} + \lambda\mathcal{R}(\theta)$
  $\mathcal{R}(\theta) = \frac{1}{2}\sum\theta_i^2$: Weight decay L2 regularization
  $\mathcal{R}(\theta) = \sum|\theta_i|$: Sparse weights L1 regularization

Drop-out

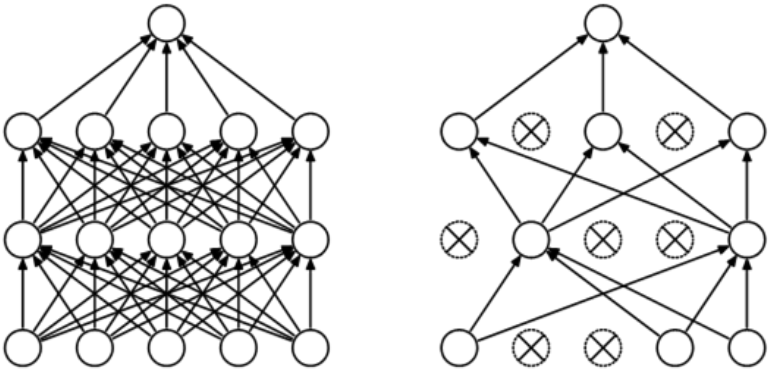
- Drop-out [Hinton et al. 2012, Srivastava, Hinton, Krizhevsky, JMLR 2014]
- Randomly set some of the activations to zero during training
- At test time run it will all the activations on.



(a) Standard network (b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

- Network builds redundancies, need to create multiple paths
- Effectively smaller networks than constructed



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Multi-task learning

- There may be more than one task for the same dataset
  - Recognition of objects + distinguishing indoor from outdoor scenes
  - Segmentation + denoising of images
- Minimization of two loss functions simultaneously
  $\mathcal{L}(\theta) = \lambda_1 \mathcal{L}_1(\theta) + \lambda_2 \mathcal{L}_2(\theta)$
- Note the similarity to regularization
  $\mathcal{L}(\theta) + \lambda \mathcal{R}(\theta)$

Data augmentation

- Augment the training set with random transformations of the observed samples:
- Rotations, scaling, up-down & left-right flip, …
- This simple approach is used very widely

Transfer learning – finetuning

- There are many networks available online that has been trained with ImageNet – with > 1 million images
- Transfer learning takes a pre-trained network and retrains it for the new task and new dataset starting from the learned weights
- You can also consider taking a part of the network instead of the whole
- Initial task can also be unsupervised – for instance denoising with added noise.

Deep Learning for Everything in Computer Vision (1)

Deep Learning for Everything in Computer Vision (2)

Note: these two parts of lecture is highly descriptive and full of pictures, so please refer to original materials.

Deep Learning for Computer Vision Part V: Advanced Topics

Lecture Notes

Advanced topics and applications

- Visualization and diagnostics
- Unsupervised localization and classification
- Unsupervised learning
    - Distribution learning
    - Unsupervised learning of features / unsupervised representation learning

Visualization and diagnostics

How does a network work?

- How does this network predict?
- What features does it extract?
- Which areas does it look in the image?
- Can we interpret it?
- Can we explain how it reaches a decision?

Why do we want to understand?

- Applications in critical domains:
    - Medicine
    - Policymaking
    - Policing
    - Law
    - Autonomous vehicles
- Critically evaluating models from a human perspective
    - Do the predictions make sense?
    - Is the network focusing on reasonable aspects in the data?
    - Is the method fair?

- Can we identify avenues for improvement?

Understanding how a network works

- Definition of ‘understanding’ or ‘interpretation’ is crucial
  - What do you exactly want to get out of the system?
    Causes of the predictions, e.g. diagnosis in medicine
    Relationships between variables, e.g. scientist
    Visualization of the output, e.g. risk analyst
    Fairness towards differences in race and gender, e.g. surveillance, loan analyst
  - There are different approaches with different definitions
- Challenging task
  - Multivariate interactions, information in different areas of the image are used in interaction with each other
  - Nonlinear mapping between features and labels
  - Hierarchical mapping, information gathered in multiple layers

Visualizing features

- Discussion based on [Zeiler and Fergus 2013]
- Visualizing the input that activates a neuron in any layer
- "Deconvolutional" network

Image dependent visualization

- The activation level in the neuron depends on the input image
- For different inputs it will be activated at different levels
  If it was linear, neuron’s activation would be based on the respective linear projection
- Same pattern for all images
- Magnitude of the activation may change depending on the dot product $\mathbf{w}_j \cdot \mathbf{x}$
- For general neural networks things are different than the linear model
- The pattern that activates a neuron changes with the input image
- The difference is due to the non-linearity of the network (activation function + pooling)
- Analysis should be based on the input image.
- The new question“In the input image which pattern caused the activation in a given neuron in an intermediate layer?”

Size of the pattern in the input image

- Size of the input pattern changes with respect to the receptive field of the neuron we are investigating
- Depending on the layer the neuron sits, its receptive field changes

- The size of the input pattern changes as well.
- The image patch that activates the blue neuron is larger than the one that activates the red neuron

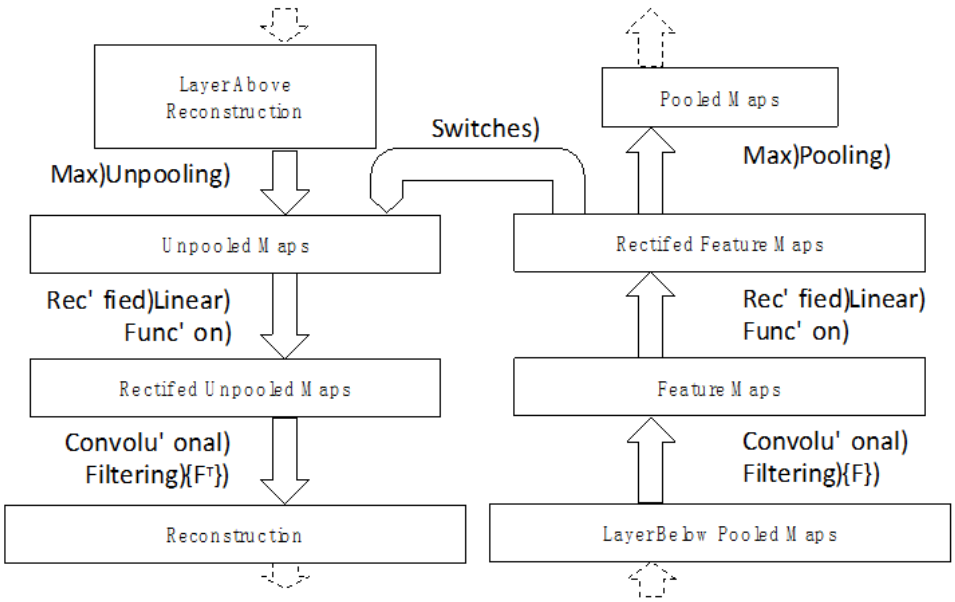
Operations in the forward pass

- Consider the operations we need to do in order to compute the activation in the blue neuron from the layer below
- Three operations
  - Convolutions with a set of filters
  - Non-linearity with ReLu function
  - Max-pooling

The idea: revert the operations

- Run an input image forward and compute all the features
- Keep the activation of the neuron you want and set the rest to 0.
- Starting from the same layer run the operations in reverse order.
- Unpool
- Rectify – run through ReLU again
- Transposed convolution
- Effectively: A linked reverse ”deconvolutional” network
- Modified layers are the inputs

The idea



Inverting the operations – 1. max pooling

- It is not possible to regenerate this information

- Pooling leads to information loss
- Keep the max locations while forward passing the image
- While unpooling place the values to the respective positions
- Instead, zero values are placed for the locations where activations are discarded during forward pass

Inverting the operations – 2. ReLu

- Only keeps the positive layers
- ReLu yields only positive activation maps
- The same function is used for the reverse operation
- ReLu is used again to keep the activations positive during the reconstruction
- You can in theory, also use the inverse of a function

Inverting the operations – 3. convolution

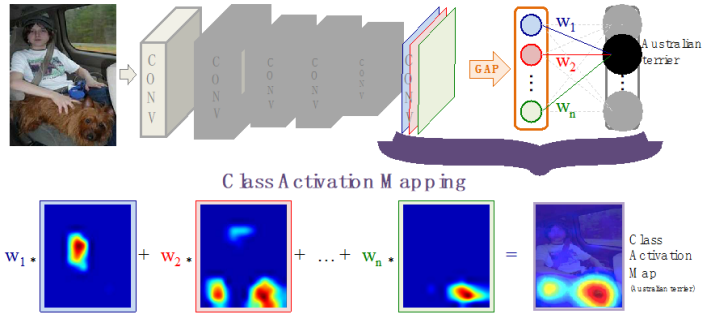
- Transposed convolution
  - Sometimes referred to as deconvolution but that is not correct terminology.
  - The kernel is the kernel used in the forward pass, flipped horizontally and vertically
- The kernel is the kernel used in the forward pass, flipped horizontally and vertically

Feature maps
Further deeper features
Even further deeper

Unsupervised localization and classification
Class activation maps

- Classification = predicting class assignment
- Localization = determining the location of an object
- Visualization of the extracted features for interpretation
- Where does a network look at in an image?
- For the classification task
- Based on global average pooling idea
- Discussion based on [Zhou et al. 2016]
- Using Global Average Pooling
- Like normal pooling, applies to each channel in a layer separately
  $H_{l,k} = \frac{1}{M_{l-1}N_{l-1}} \sum_{x,y} h_{l-1,k}^{(x,y)}$
  Averaging all the information to a single number!
  Then continue as usual
  $a_{L,j} = \sum_k w_{L,jk} h_{l,k}$

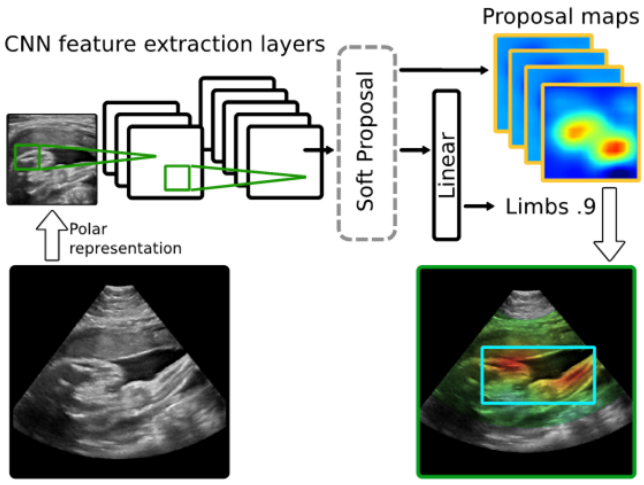
Activation Maps



- Per-class weighted sum of all the channels before global average pooling yields the class-specific activation map
- Network architecture preceding the GAP layer can change
- Form of weak-supervision for localization

Various applications

- Especially in medical imaging
- Labels are expensive and difficult to get.
- Approximate localization with CAM allow identifying areas of interest
- Weak supervision to train stronger localization algorithms



Unsupervised learning

Very coarse view on supervised learning

- Patterns between two types of data
- Goal: predicting one from the other
- Examples have both types of data
- At prediction only one exist

General idea in the supervised approach

- Algorithms assume a mathematical model between features and labels
- Estimate the parameters of the model to best predict labels from features in the training examples $y = f(\mathbf{x}|\theta)$

Unsupervised learning: learning of distributions

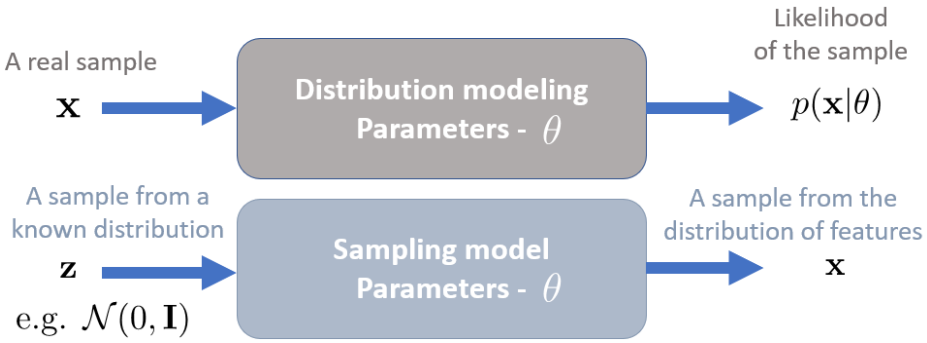
- Patterns within the data
- Goal: describe variability in the data
- Estimate the distribution
- Possibly very high dimensional distribution
- There is still a training dataset
- Examples have only features
- Unsupervised in the sense that there are no labels

Unsupervised learning: learning of features

- Filters are important for performing image analysis tasks
- So far, we determine features in a supervised way, task-specific manner
- Determine features in an unsupervised manner
- Examples have only features
- Are there universal features for computer vision?
- Unsupervised in the sense that there are no labels

General idea in unsupervised distribution learning

- Probability distribution
  - Likelihood
  - Sampling



Why is this useful?

- Sample from the distribution of image to generate synthetic images
- Style transfer
- Bayesian reconstruction of medical images
- Improving resolution of an image

- Many more applications:
  - In-painting
  - Realistic video and image editing
  - Video frame prediction
  - Outlier detection
  - …
- Scientifically
  - Building a model of the visual world
  - Possibly an important component in human learning.
    - We do not see 100s of cups to understand what a cup is
    - We constantly observe around and get visual input to our brains.

Images are big

- Images are very high dimensional
- Consider a small image of 64x64
- Even that is 4096 dimensional!
- We need to keep that in mind when we think about unsupervised learning.

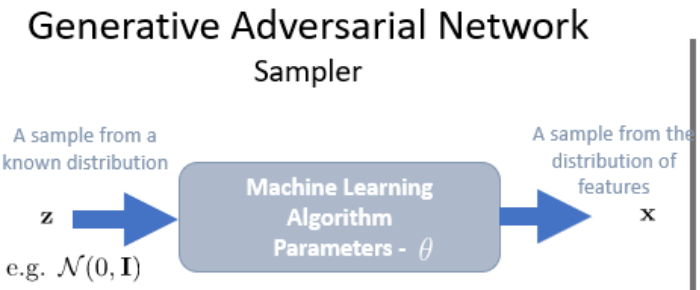
The most straightforward way

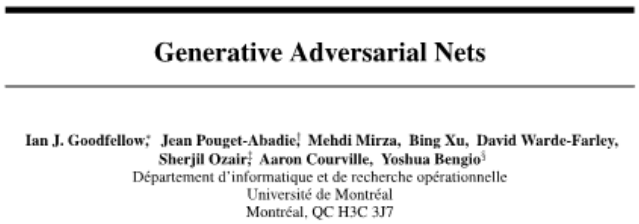
- Kernel density estimation (KDE)
- Given a sample set of images the naïve way is
  $p(x|\theta) = \frac{1}{N} \sum_n K_\theta(x, x_n)$
  $\int K_\theta(x, x_n) dx = 1$
- Place a ”kernel” around each training sample
- Determine the likelihood of a new sample based on these kernels
- If kernels depends on Euclidean distance, e.g. Gaussian kernel, then likelihood is related to the distance in Euclidean space.

Bad idea due to the dimensions

- For the KDE to work, roughly speaking, you need to somehow “fill” the space, e.g.
- To fill a space of 4096 dimensions, you need a lot of samples, we need to find a better solution.

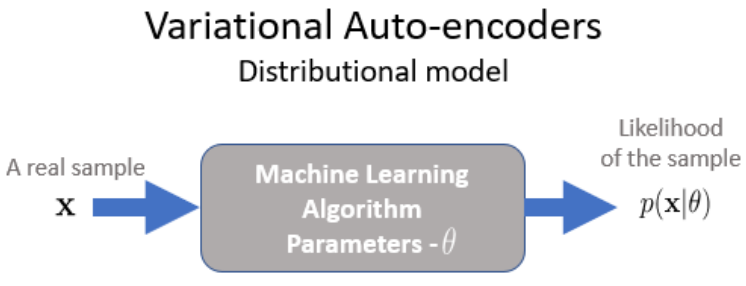
Two avenues – both end of 2013

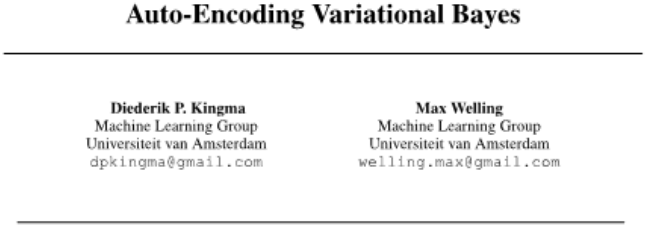




Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

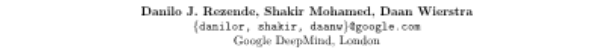




Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

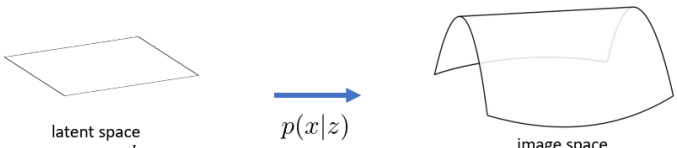


Stochastic Backpropagation and Approximate Inference in Deep Generative Models

Danilo J. Rezende, Shakir Mohamed, Daan Wierstra
{danilor, shakir, daanw}@google.com
Google DeepMind, London

Latent variable models

- Assume that images live in a lower dimensional sub-space
- We build a mapping between them

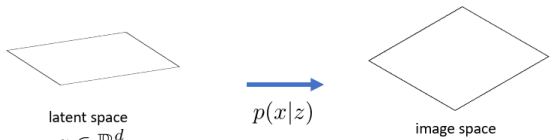


$$p(x) = \int p(x|z)p(z)dz$$

$$x \in \mathbb{R}^D,\ z \in \mathbb{R}^d,\ d << D$$

Probabilistic principal component analysis

- Assumes the mapping is a linear one
- Probabilistic principal component analysis [Tipping & Bishop 1999]



$$p(x) = \int p(x|z)p(z)dz \qquad p(x|z) = \mathcal{N}\left(Wz + \mu_x, \sigma^2\right)$$

$$x \in \mathbb{R}^D,\ z \in \mathbb{R}^d,\ d << D \qquad p(z) = \mathcal{N}(0, I) \qquad \mu_x:\ \text{mean image}$$

$$W \in \mathbb{R}^{D \times d} \qquad \sigma^2:\ \text{noise}$$

Link to PCA

- Maximum likelihood estimate of the parameters yield the PCA
- Eigenvalues and eigenvectors of the sample covariance matrix

- Derivation in [Tipping and Bishop 1999]

$$W_{ML} = U_d \left(\Lambda_d - \sigma^2 I\right)^{1/2} R$$

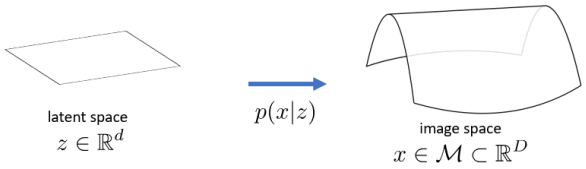
$$W_{ML} = \underbrace{U_d}_{\text{d eigenvectors}} \left(\underbrace{\Lambda_d}_{\text{d eigenvalues}} - \sigma^2 I \right)^{1/2} \underbrace{R}_{\text{arbitrary rotation}}$$

μ_x : sample mean image

$$\sigma^2_{ML} = \frac{1}{D-d} \sum_{q=d+1}^{D} \lambda_q$$

Non-linear maps

- In supervised learning linear maps were not enough
- The same idea applies here



$$p(x) = \int p(x|z)p(z)dz$$

$$x \in \mathbb{R}^D,\ z \in \mathbb{R}^d,\ d << D$$

$$\boxed{p(x|z) = \mathcal{N}\left(f(z|\theta) + \mu_x, \sigma^2\right)}$$

$$p(z) = \mathcal{N}(0, I)$$

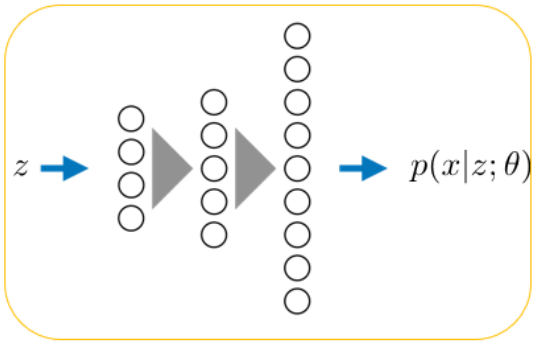
μ_x : mean image

σ^2 : noise

Density networks[MacKay, Nucl. Inst. Met. In Physics Research 1995]

$p(x|z;\theta)$: Parameterize with a network with parameters $\boldsymbol{\theta}$

$$p(x;\theta) = \int p(x|z;\theta)p(z)dz$$



For the given samples, maximize with respect to $\boldsymbol{\theta}$

$$\prod_n p(x_n;\theta)$$

$$= \prod_n \int p(x_n|z;\theta)p(z)dz$$

using Monte Carlo integration

$$\sum_n \ln \frac{1}{R} \sum_r p(x_n|z_r;\theta),\ z_r \sim p(z)$$

Sampling was not efficient for very large dimensional problems, need too many samples
MacKay hinted importance sampling

Variational auto-encoders

Builds on density networks concept but instead of Monte-Carlo uses variational inference with a network parameterized sampling (approximate) distribution

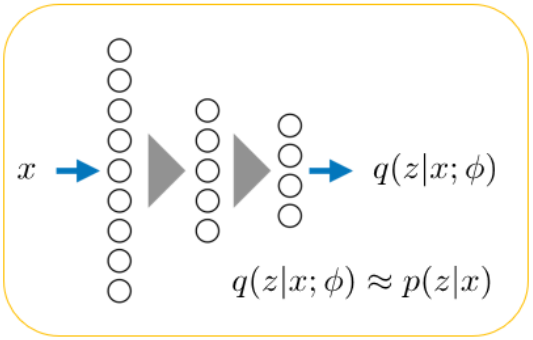
$$
\begin{aligned}
\ln p(x;\theta) &= \ln \int p(x|z;\theta)p(z)dz \\
&= \ln \int p(x|z;\theta)p(z)\frac{q(z|x;\phi)}{q(z|x;\phi)}dz \\
&\geq \int q(z|x;\phi)\ln p(x|z;\theta)\frac{p(z)}{q(z|x;\phi)}dz \\
&= \mathbb{E}\left[\ln p(x|z;\theta)\right] - D_{KL}\left[q(z|x;\phi)\|p(z)\right]
\end{aligned}
$$

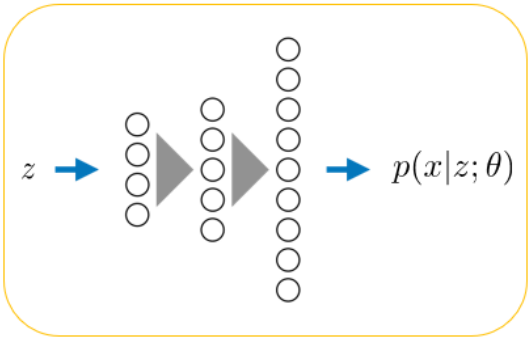
Let's find a distribution more focused so I will sample for less for the same approximation
Importance Sampling
Best option is the posterior p(z|x)

Jensen's Inequality

$$\ln p(x;\theta) \geq \mathbb{E}_{q(z|x;\phi)}\left[\ln p(x|z;\theta)\right] - D_{KL}\left[q(z|x;\phi)\|p(z)\right]$$

Evidence lower-bound : Maximize this instead of real likelihood





Encoding Model

Takes an image and maps it to the posterior distribution in the latent space.
Encodes to the lower dimensional space

Decoding Model

Takes the lower dimensional representation and maps to an image.
Can be used as a sampler.
Can be used as a reconstruction tool

$$q(z|x;\phi) = \mathcal{N}(z;\mu_z(x;\phi),\Sigma_z(x;\phi)) \qquad p(x|z;\theta) = \mathcal{N}(x;\mu_x(z;\theta),\Sigma_x(z;\theta))$$

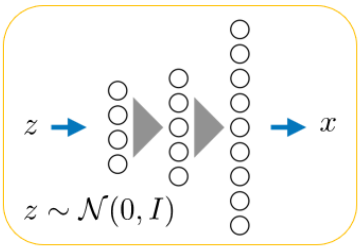
Both Gaussian Models

Homework: Can you determine the link with the probabilistic PCA model?

Difference with PCA

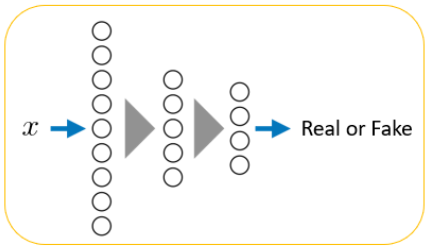
Generative adversarial networks

- Instead of an explicit probabilistic model, a GAN is a sampling tool that generates samples from the data distribution



Generator

Generates realistic looking images from random samples in the latent space.



Discriminator

Tries to classify images into two categories: Real or generated (Fake)

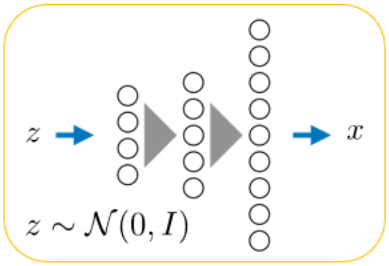
During training they compete

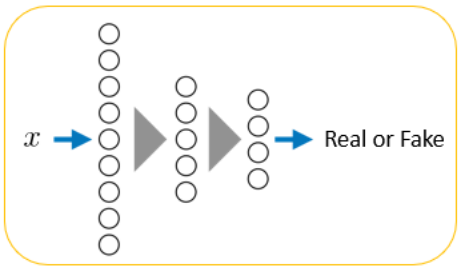
Generator - G

Tries to create samples that can fool the discriminator

Discriminator - D

Tries to identifies the images the generator creates





Solve this problem: Optimize the network weights with a two-player game

$$\min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p_{\text{real}}}(x) \left[\ln D(x; \phi)\right] + \mathbb{E}_{z \sim p(z)} \left[\ln(1 - D(G(z; \theta))\right]$$

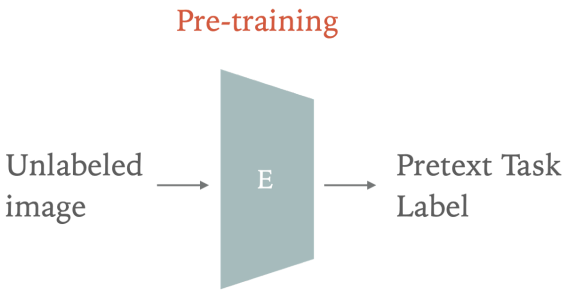
Very active area of research

- Other very interesting direction is the flow-based models

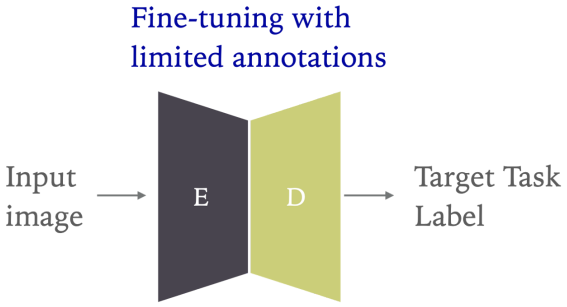
Unsupervised learning of features

- Features are important, they are the essential building blocks
- For any task it is important to get the right features
- It requires large number of labelled images to do this
  - It would be wonderful if we could do it with only few images
  - Humans do not seem to require lots of labelled images for good features, assuming humans do have good features
  - Are there universal features that can be useful for ANY visual task?

Main idea: self-supervised learning with a pretext task

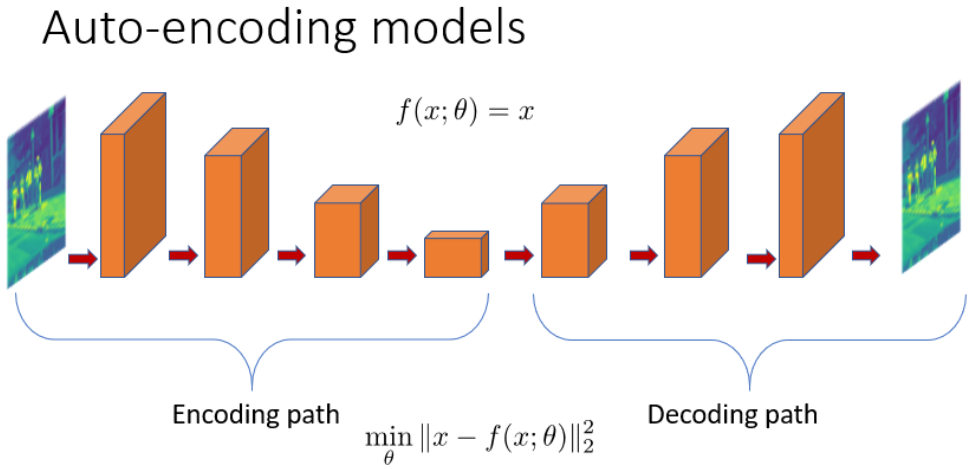


Then finetune the model or use the representation / encoder



Auto-encoding models

- The bottleneck layer does not allow the network to learn an identity map
  It learns to summarize the most important information for reconstruction

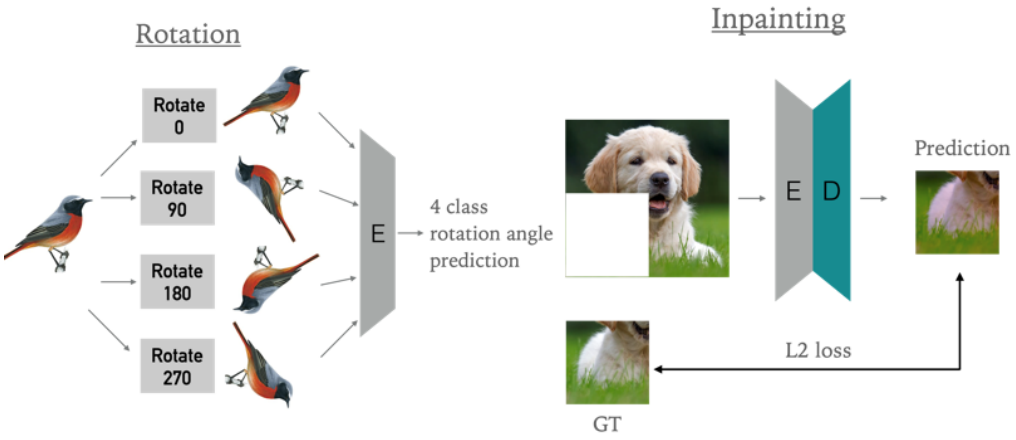


- Minimization only requires the images. The goal is to be able to reconstruct the image with high fidelity.
- Features learnt here can then be translated to another task either directly or by fine-tuning, i.e. starting the optimization from the pre-learnt weights

In practice

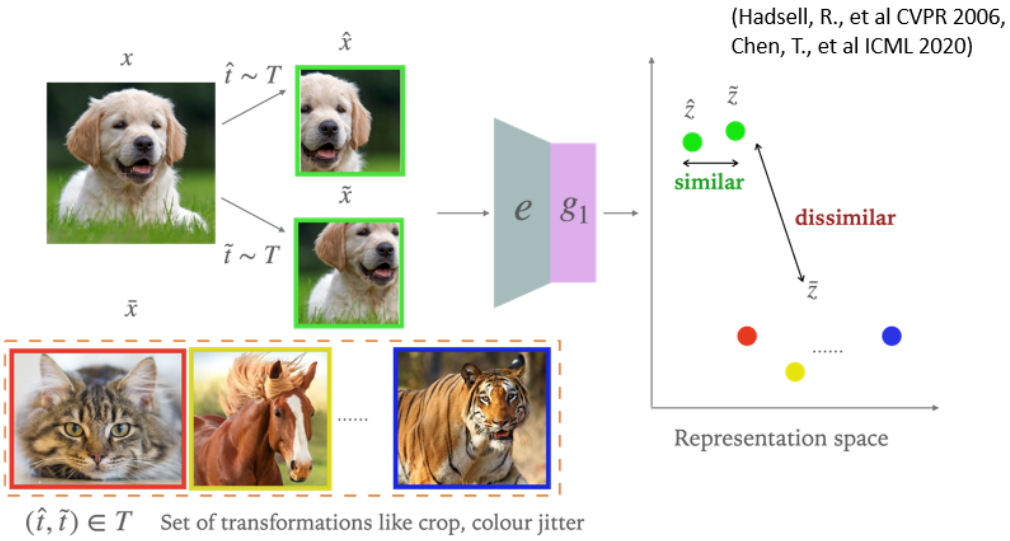
- The features learnt from a simple auto-encoder are not extremely useful
- In the end, you may still need large number of labelled examples
- Not as large as training from scratch though

Other options



(Gidaris, S., et al ICLR 2018, Pathak, D., et al CVPR 2016)

Contrastive learning



Contrastive loss

➤ Global Contrastive loss for a given positive pair:

$$l(\tilde{x}, \hat{x}) = -\log \frac{e^{\text{sim}(\tilde{z},\hat{z})/\tau}}{e^{\text{sim}(\tilde{z},\hat{z})/\tau} + \sum_{\bar{x}\in\Lambda^-} e^{\text{sim}(\tilde{z},g_1(e(\bar{x})))/\tau}} \quad ;\tilde{z} = g_1(e(\tilde{x})),\ \hat{z} = g_1(e(\hat{x}))$$

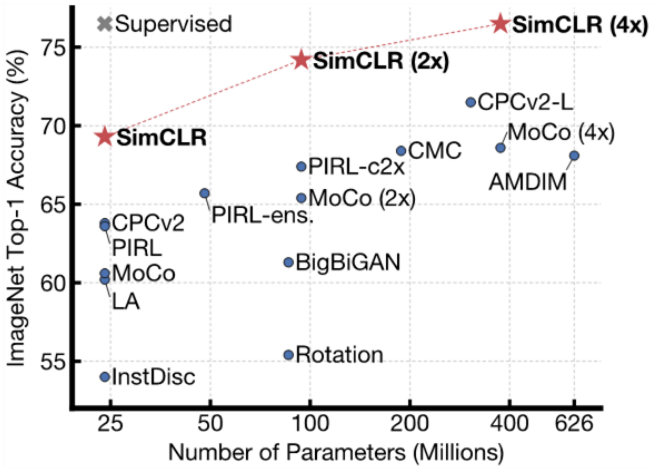
Positive pairs term — Negative pairs terms

Positive pair set Λ^+	Negative samples set Λ^-
Cosine similarity	$\text{sim}(a,b) = a^T b/\|a\|\|b\|$
Temperature scaling parameter	τ

➤ Total loss over all positive pairs in the mini-batch:

$$L_g = \frac{1}{|\Lambda^+|} \sum_{\forall(\tilde{x},\hat{x})\in\Lambda^+} [l(\tilde{x},\hat{x}) + l(\hat{x},\tilde{x})]$$

Impressive results with LARGE networks


Supervised
SimCLR (4x)
SimCLR (2x)
CPCv2-L
MoCo (4x)
SimCLR
CMC
PIRL-c2x
AMDIM
MoCo (2x)
PIRL-ens.
CPCv2
PIRL
MoCo
LA
BigBiGAN
Rotation
InstDisc
ImageNet Top-1 Accuracy (%)
75
70
65
60
55
25
50
100
200
400
626
Number of Parameters (Millions)