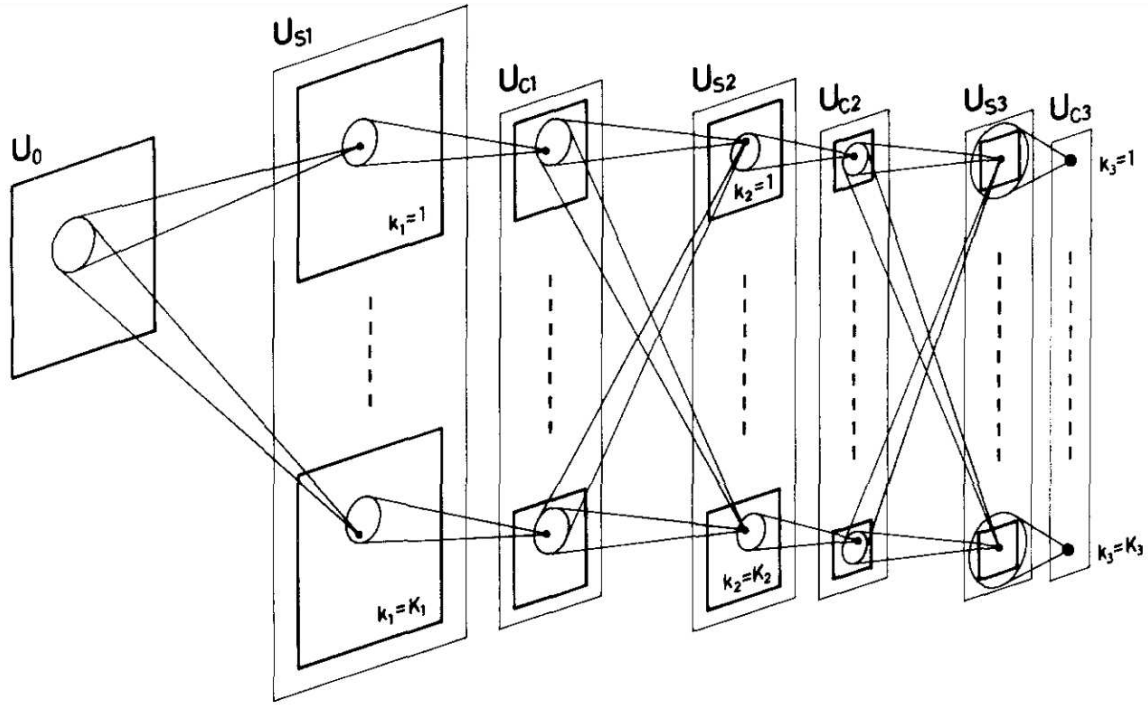
University of Cambridge
Engineering Part IIB

Module 4F12: Computer Vision

Handout 5: Deep Learning for Computer Vision



Ignas Budvytis
November 2021

Syllabus

1. Introduction

- Computer vision: what is it, why study it and how?
- Vision as an information processing task
- Geometrical and statistical frameworks for vision
- 3D interpretation of 2D images

2. Image structure

- Image intensities and structure
- 2D convolution with gaussians for low-pass filtering
- Edge detection, the aperture problem and corner detection
- Image pyramids, blob detection with band-pass filtering
- The SIFT feature descriptor for matching
- Characterising textures.

3. Projection

- Orthographic projection
- Planar perspective projection, vanishing points and lines
- Homogeneous coordinates and the projection matrix
- Camera calibration, recovery of world position
- Weak perspective and the affine camera

4. Stereo vision and Structure from Motion

- Recovery of depth by triangulation
- Epipolar geometry and the essential matrix
- Uncalibrated cameras and the fundamental matrix
- The correspondence problem
- Structure from motion
- 3D shape examples from multiple view stereo and photometric stereo

5. Deep Learning for Computer Vision

- Basic architectures for deep learning in computer vision
- Detection, classification and semantic segmentation
- Recognition, feature embedding and metric learning
- Transformers, scaling laws for computer vision, neural architecture search
- Self-supervised learning and pseudo-labelling

Course book: V. S. Nalwa. *A Guided Tour of Computer Vision*, Addison Wesley, 1993 (CUED shelf mark: NO 219).

What is Deep Learning?

Deep learning is a machine learning technique designed to automatically discover features in data.

A **deep neural network** (DNN) is trained to automatically interpret data (e.g. images I) by learning a complex *deep* **representation** R. Training is performed by a **minimisation** of a desired **objective function** L applied to the network output Y, in order to solve a **task** T (e.g. binary classification).

$$\begin{aligned}
\textbf{representation} &\rightarrow \quad R(I) = F(G(...(I, W_{...})...), W_G), W_F) \\
\textbf{network output} &\rightarrow \quad Y = T(R, W_T) \ \text{(e.g. logistic reg.)} \\
\textbf{objective function} &\rightarrow \ L(Y, \hat{Y}) \ \text{(e.g. cross entropy)} \\
\textbf{minimisation} &\rightarrow \ \{W^*_{...}, ...\}, W^*_T = \text{argmin}_{\{W_{...},...\},W_T} L(Y, \hat{Y})
\end{aligned}$$

Key properties:

Deep representation: learnt, formulated as a sequence of nested parametrized functions, referred to as layers or features.

Non-linear feature transformations: using non-linear activation functions allows for complex decision boundaries (classification) and function approximations (regression).

Simple optimisation strategies: stochastic gradient descent optimisation and its variants.

Brief History of Deep Learning

- **McCulloh-Pitts neuron** [1] (McCulloch and Pitts, 1943)
  - Early computational model of brain function.
  - Model: $y = f(\mathbf{w}^T\mathbf{x})$. Boolean inputs and outputs.
  - Activation function $f(a)$ is a Heavyside step function with a threshold t.
  - Model weights $\mathbf{w}$ and threshold t were set by hand.
- **Perceptron** [2] (Rosenblatt, 1958)
  - Based on McCulloh-Pitts [1] neuron.
  - Could learn weights from examples of two categories.
  - Weight update performed only for wrongly classified data points.
  - Proven to find an optimal solution for a linearly separable pattern.
  - Implemented in hardware - *Mark 1 Perceptron*.
  - Designed for image recognition (20×20 photocells).
- **ADAptive LInear NEuron (ADALINE)** [3] (Widrow and Hoff, 1960)
  - Linear (identity) activation function: $y = \Sigma_i\, x_i w_i + b$, where b - bias.
  - Objective function - squared error (i.e. $\Sigma_n (y_n - \hat{y_n})^2$).
  - Iterative weight ($\{\mathbf{w}, b\}$) update procedure making use of gradients of the objective function w.r.t. weights.

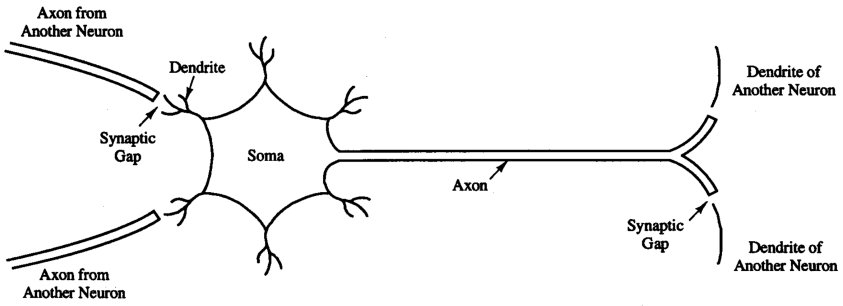
- **Backpropagation** [5]
  (Rumelhart et. al., 1986a; LeCun, 1987)
  - Efficient gradient calculation by utilizing chain rule and dynamic programming.
  - Widely used in the training of neural networks.
- **Neocognitron** [4] (Fukushima, 1980)
  - A hierarchical model architecture for processing images (e.g. handwritten character recognition).
  - Inspired by the structure of the mammalian visual system.
  - Alternating layers of S-cells and C-cells.
  - The basis for the modern convolutional neural networks (CNN).
- **LeNet-5** [6] (LeCun, 1998)
  - Impressive performance on digit recognition task beating hand-crafted features.
  - Used by several banks to recognize hand-written numbers on checks.
  - Believed to be very hard to train.
- **AlexNet** [8] (Krizhevsky et. al., 2012)
  - CNN for a large scale image classification task.
  - Highly superior performance to methods based on hand-crafted features.
  - The starting point of the most recent wave of deep learning.

Recent Successes in Computer Vision

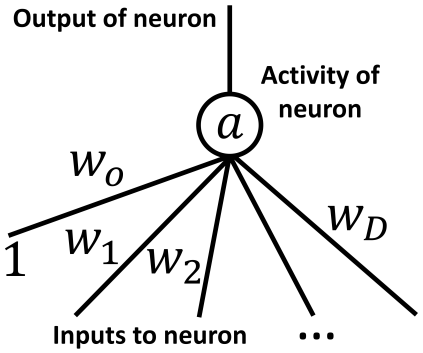
- Healthcare
  - Cancer radiotherapy (e.g. breast cancer detection)
  - Detecting eye disease from OCT scans
  - Robot assisted surgeries
- Agriculture
  - Fruit and vegetable picking robots
  - Crop monitoring and harvest prediction
  - Food inspection
- Manufacturing and Industry
  - Monitoring of manufacturing process
  - Enhanced mobility - autonomous driving
  - Multi-purpose, retrainable robots
- Digital Services
  - Augmented reality
  - Image and video search
  - Frictionless payment via face recognition

Single Neuron

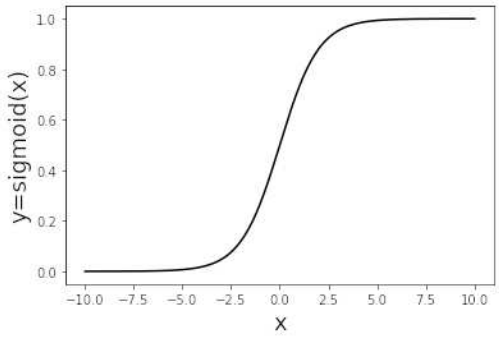
Biological neuron



Computational neuron



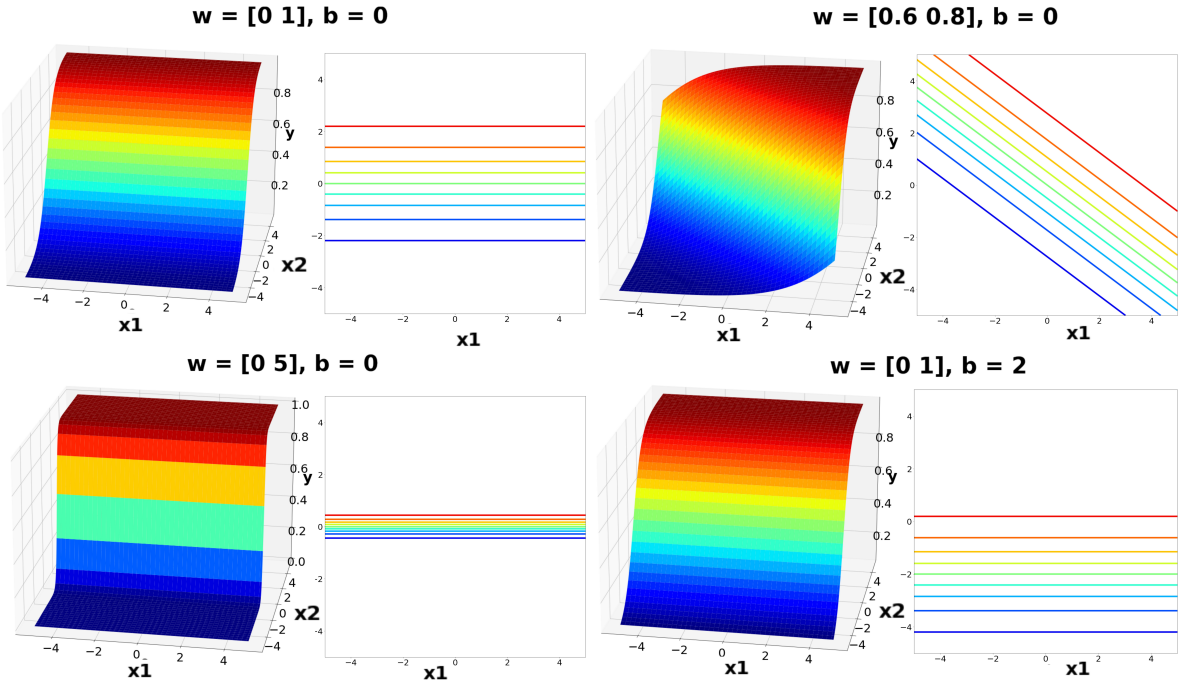
Sigmoid activation function



- (Inputs, Outputs): (dendrites, axons) $\rightarrow (\mathbf{x}, y)$
- Each dendrite contributes to input signal being weighted in different proportions $\rightarrow x_i w_i$
- Soma (or cell body) sums the signal from dendrites $\rightarrow a = \Sigma\, x_i w_i = \mathbf{w}^T\mathbf{x}$
- Neuron produces an exciting or inhibiting signal along its axons in non-linear (e.g. sigmoid function) way $\rightarrow y = f(a) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$

Single Neuron in 2D

Neuron output visualisation for $y = \frac{1}{1+e^{-w_1 x_1 - w_2 x_2 - b}}$:



Properties:

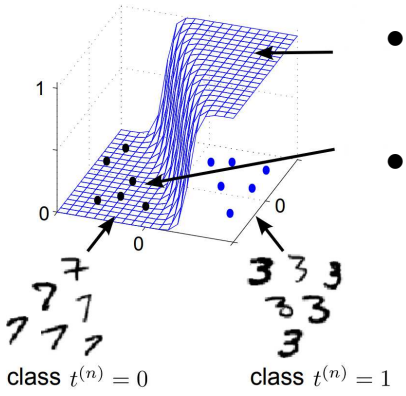
- $\frac{\mathbf{W}}{|\mathbf{W}|}$ sets the orientation of the contour
- $|\mathbf{w}|$ sets the steepness of the contour
- Bias b translates the decision plane along its normal

Single Neuron - Training

Training data

- Inputs: $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$, where $\mathbf{x}^{(n)} \in R^D$.
- Targets: $\{t^{(n)}\}_{n=1}^{N}$, where $t^{(n)} \in \{0, 1\}$ - class labels.

Desired result



- Neuron outputs $y^{(n)} = f(\mathbf{w}^T\mathbf{x}^{(n)}) \approx 1$ for $t^{(n)} = 1$.
- Neuron outputs $y^{(n)} = f(\mathbf{w}^T\mathbf{x}^{(n)}) \approx 0$ for $t^{(n)} = 0$.

Objective function

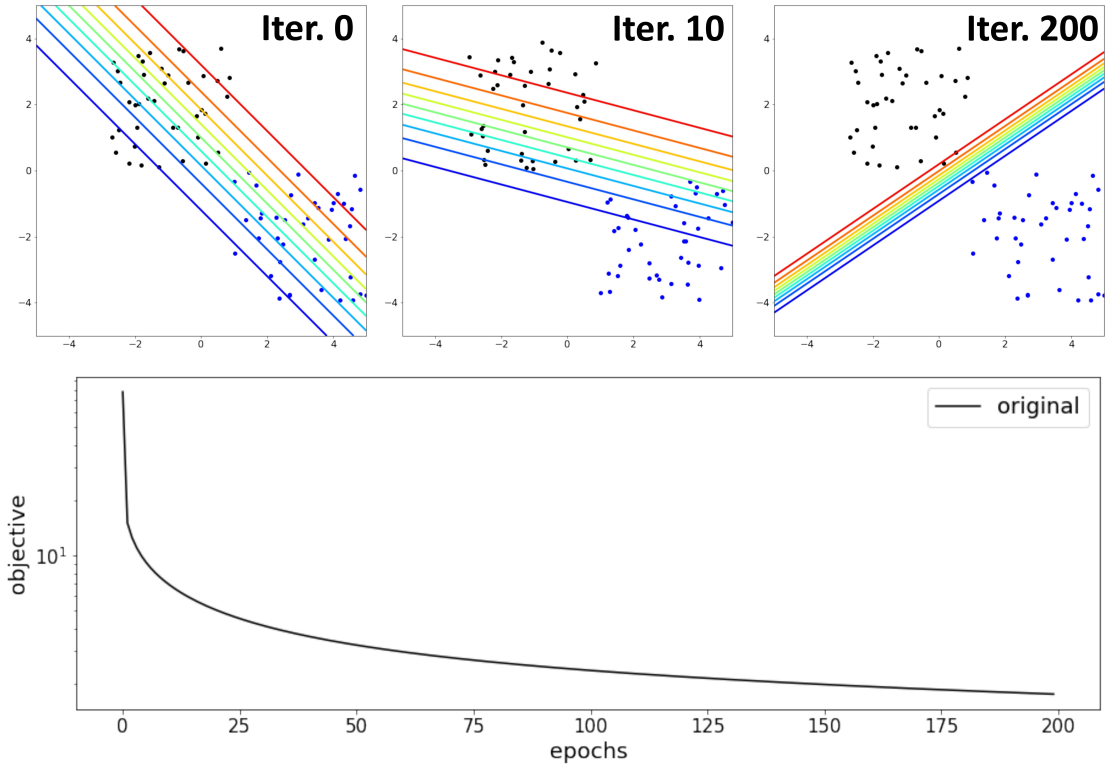
- Neuron output as probability distribution:
  $P(t = 1|\mathbf{w}, \mathbf{x}) = y$, $P(t = 0|\mathbf{w}, \mathbf{x}) = 1 - P(t = 1|\mathbf{w}, \mathbf{x})$.
- Minimise negative log-likelihood of:
  $P(t|\mathbf{w}, \mathbf{x}) = P(t = 0|\mathbf{w}, \mathbf{x})^{\mathbf{1}[t=0]} P(t = 1|\mathbf{w}, \mathbf{x})^{\mathbf{1}[t=1]}$.
- Objective function for the whole dataset:
  $L(\mathbf{w}) = -\Sigma_n[t^{(n)} \log f(\mathbf{w}^T\mathbf{x}^{(n)}) + (1-t^{(n)}) \log(1-f(\mathbf{w}^T\mathbf{x}^{(n)}))]$.
- Equivalent to minimising cross entropy: $H(p, q) = -\Sigma_i\, p(i) \log q(i)$ or relative entropy: $KL(p||q) = \Sigma_i\, p(i) \log(\frac{p(i)}{q(i)})$.
- When $t^{(n)} = 1$, loss term $-\log(y^{(n)})$ encourages neuron output $y^{(n)}$ to match training data. Similarly for $t^{(n)} = 0$.

Gradient Descent Iterative Update

Gradient descent iterative update procedure

- Goal of the learning procedure: $\mathbf{w}^* = \text{argmin}_{\mathbf{W}} L(\mathbf{w})$.
- Analytic closed form solution is not available.
- $\frac{\mathrm{d}\, L(\mathbf{W})}{\mathrm{d}\, \mathbf{W}}$ gives the direction of the steepest ascent over objective function L w.r.t. its parameters $\mathbf{w}$.
- General update rule for gradient descent:
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\mathrm{d}\, L(\mathbf{W})}{\mathrm{d}\, \mathbf{W}}$, where η - learning rate.
- Specific update rule for a single neuron:
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \left(- \Sigma_n (t^{(n)} - y^{(n)}) \mathbf{x}^{(n)} \right)$.

Training visualisation

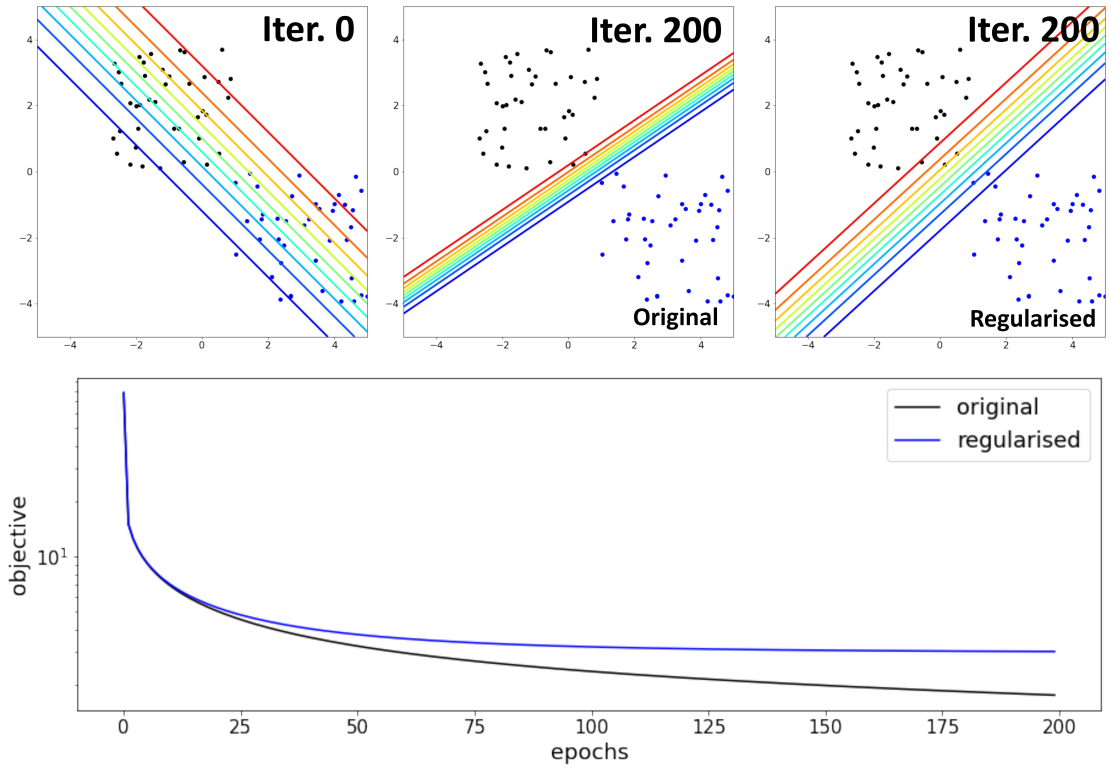


Training with Regularisation

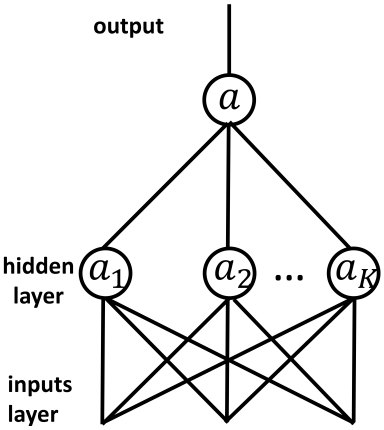
Gradient descent for regularised objective function

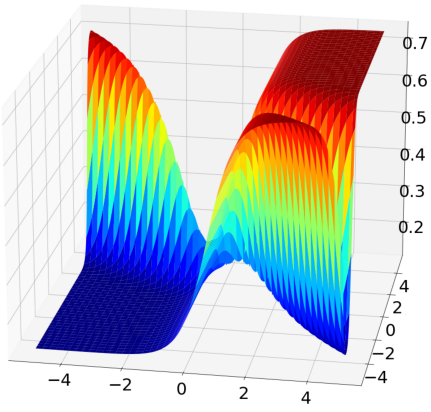
- Regularizer discourages extreme weights.
- $L_{reg}(\mathbf{w}) = L(\mathbf{w}) + \alpha L_R(\mathbf{w})$, where $L_R(\mathbf{w}) = \frac{1}{2}\Sigma_d\, w_d^2$ and α is regularisation weighting parameter.
- General update rule: $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\mathrm{d}\, L_{reg}(\mathbf{w})}{\mathrm{d}\, \mathbf{w}}$.
- Update rule for a single neuron: $\mathbf{w} \leftarrow \mathbf{w} - \eta \left(-\Sigma_n (t^{(n)} - y^{(n)}) \mathbf{x}^{(n)} + \alpha \mathbf{w} \right)$.

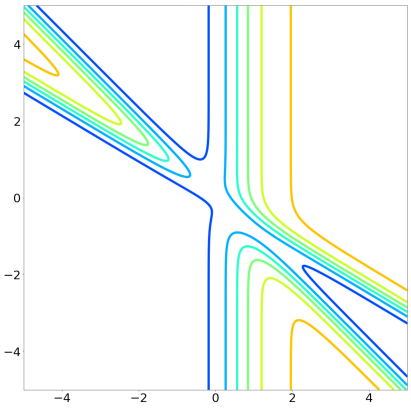
Training visualisation



Single Hidden Layer Network







Description:

- Input (layer): $\mathbf{x} = [x_1, x_2, ..., x_D]^T$
- Hidden layer activity: $a_k = \Sigma_{d=1}^{D} W_{k,d} x_d$
- Hidden layer output: $y_k = f(a_k) = \frac{1}{1+\exp(-a_k)}$
- Output neuron activity: $a = \Sigma_{k=1}^{K} w_k y_k$
- Output of a network: $y = f(a) = \frac{1}{1+\exp(-a)}$
- Biases ignored for simplicity
- This is a feed-forward network - no cycles

Objective function

$L(W, \mathbf{w}) = -\Sigma_n [t^{(n)} \log y^{(n)} + (1 - t^{(n)}) \log(1 - y^{(n)})]$

Regularizer

$L_R(W, \mathbf{w}) = \frac{1}{2} \Sigma_d w_d^2 + \frac{1}{2} \Sigma_{k,d} W_{k,d}^2$

Optimisation goal

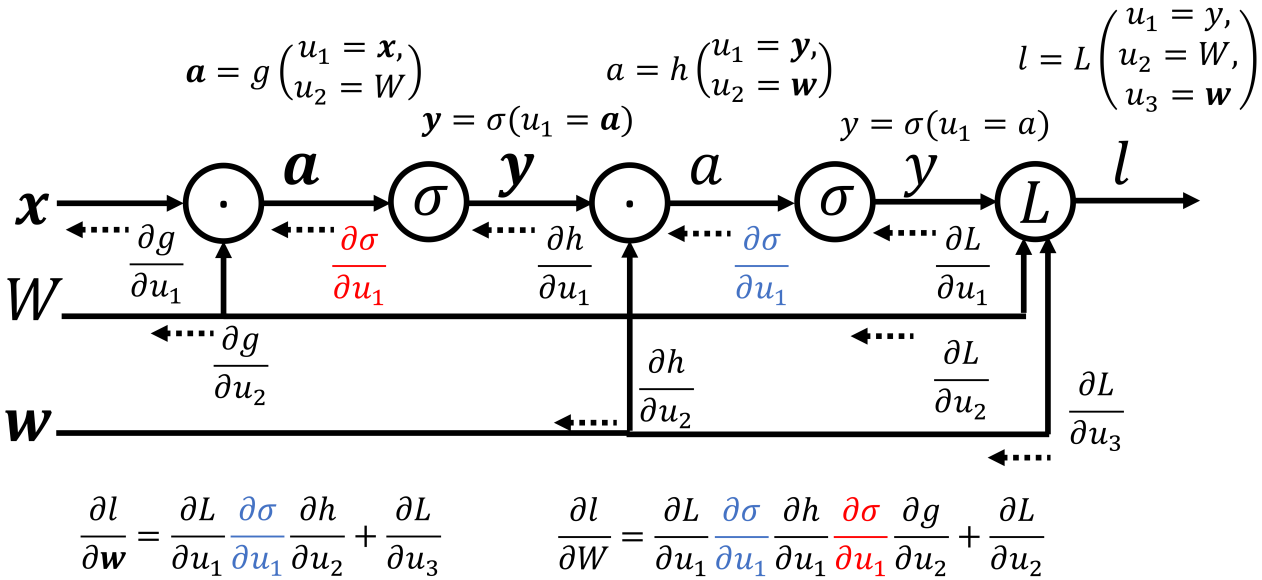
$\{W^*, \mathbf{w}^*\} = \text{argmin}_{W,\mathbf{w}} L_{reg}(W, \mathbf{w}) = \text{argmin}_{W,\mathbf{w}} [L(W, \mathbf{w}) + \alpha L_R(W, \mathbf{w})]$

Backpropagation via Chain Rule

Straight forward application of chain rule:

$$\frac{\mathrm{d}\,L(W,\mathbf{W})}{\mathrm{d}\,W_{k,d}} = \sum_n \frac{\mathrm{d}\,L(W,\mathbf{W})}{\mathrm{d}\,y^{(n)}}\frac{\mathrm{d}\,y^{(n)}}{\mathrm{d}\,W_{k,d}} = \sum_n \frac{\mathrm{d}\,L(W,\mathbf{W})}{\mathrm{d}\,y^{(n)}}\frac{\mathrm{d}\,y^{(n)}}{\mathrm{d}\,a^{(n)}}\frac{\mathrm{d}\,a^{(n)}}{\mathrm{d}\,W_{k,d}} =$$
$$= \sum_n \frac{\mathrm{d}\,L(W,\mathbf{W})}{\mathrm{d}\,y^{(n)}}\frac{\mathrm{d}\,y^{(n)}}{\mathrm{d}\,a^{(n)}}\frac{\mathrm{d}\,a^{(n)}}{\mathrm{d}\,y_k^{(n)}}\frac{\mathrm{d}\,y_k^{(n)}}{\mathrm{d}\,W_{k,d}} = \sum_n \frac{\mathrm{d}\,L(W,\mathbf{W})}{\mathrm{d}\,y^{(n)}}\frac{\mathrm{d}\,y^{(n)}}{\mathrm{d}\,a^{(n)}}\frac{\mathrm{d}\,a^{(n)}}{\mathrm{d}\,y_k^{(n)}}\frac{\mathrm{d}\,y_k^{(n)}}{\mathrm{d}\,a_k^{(n)}}\frac{\mathrm{d}\,a_k^{(n)}}{\mathrm{d}\,W_{k,d}}$$

Backpropagation:



Vectorised Backpropagation

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \dots \\ x_D \end{bmatrix}_{(D+1)\times 1} \quad W = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \dots \\ \mathbf{w}_K \end{bmatrix}_{K\times(D+1)} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_K \end{bmatrix}_{K\times 1} \quad \tilde{\mathbf{w}} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}_{(K+1)\times 1},$$

$\mathbf{a} = W\mathbf{x} \in R^{K\times 1}$, $\mathbf{y} = \sigma(\mathbf{a}) \in R^{K\times 1}$, $\tilde{\mathbf{y}} = \begin{bmatrix} 1 \\ \mathbf{y} \end{bmatrix}_{(K+1)\times 1}$, $a = \tilde{\mathbf{w}}^T\tilde{\mathbf{y}}$,

$y = \sigma(a)$, $l = -(t\log(y) + (1-t)\log(1-y))$

Jacobian for $\mathbf{f}(\mathbf{x})$: $J = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots \frac{\partial f_2}{\partial x_n} \\ \dots & & \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots \frac{\partial f_m}{\partial x_n} \end{bmatrix}_{m\times n}$.

Chain rule: $\frac{\partial \mathbf{f}(\mathbf{g}(\mathbf{x}))}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}}{d\mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{x}}$.

$\frac{\partial l}{\partial y} = -\left[\frac{t}{y} - \frac{1-t}{1-y}\right]_{1\times 1} = -\left[\frac{t-y}{y(1-y)}\right]_{1\times 1}$,

$\frac{\partial l}{\partial a} = \frac{\partial l}{\partial y}\frac{\partial y}{\partial a} = -[t-y]_{1\times 1}$,

$\frac{\partial l}{\partial \mathbf{y}} = \frac{\partial l}{\partial a}\frac{\partial a}{\partial \mathbf{y}} = -[t-y]_{1\times 1}\mathbf{w}^T \in R^{1\times K}$,

$\frac{\partial l}{\partial \mathbf{a}} = \frac{\partial l}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{a}} = -[t-y]_{1\times 1}\mathbf{w}^T diag(\mathbf{y}\odot(\mathbf{1}-\mathbf{y})) = -[(t-y)]_{1\times 1}\mathbf{w}^T \odot (\mathbf{y}\odot(\mathbf{1}-\mathbf{y}))^T = \mathbf{d}^T \in R^{1\times K}$,

$$\frac{\partial l}{\partial W} = ??? = \begin{bmatrix} \frac{\partial l}{\partial W_{1,0}} & \frac{\partial l}{\partial W_{1,1}} & \dots & \frac{\partial l}{\partial W_{1,D}} \\ \frac{\partial l}{\partial W_{2,0}} & \frac{\partial l}{\partial W_{2,1}} & \dots & \frac{\partial l}{\partial W_{2,D}} \\ \dots & & & \\ \frac{\partial l}{\partial W_{K,0}} & \frac{\partial l}{\partial W_{K,1}} & \dots & \frac{\partial l}{\partial W_{K,D}} \end{bmatrix} = \begin{bmatrix} d_1x_0 & d_1x_1 & \dots & d_1x_D \\ d_2x_0 & d_2x_1 & \dots & d_2x_D \\ \dots & & & \\ d_Kx_0 & d_Kx_1 & \dots & d_Kx_D \end{bmatrix}_{K\times(D+1)} =$$

$= \mathbf{d}\mathbf{x}^T \in R^{K\times(D+1)}$.

Hence $\frac{\mathrm{d}\, l}{\mathrm{d}\, W_{k,d}} = -(t-y)y_k(1-y_k)w_kx_d$.

Partial derivatives for functions of any shape can be obtained by input and output vectorisation: $\frac{\partial F}{\partial X} \approx \frac{\partial \mathsf{VECT}_{(F)}}{\partial \mathsf{VECT}_{(X)}}$.

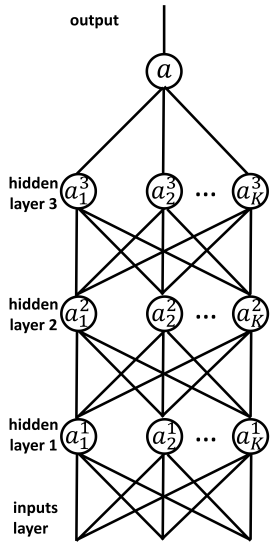
Multi-Layer Networks

Universal approximation theorem

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, under mild assumptions on the activation function.

Why use more than one hidden layer?

- Visual scenes are hierarchically organized.
- Biological vision is hierarchically organized.
- Shallow architectures are inefficient at representing deep functions despite *universal approximation* capabilities.



Why use anything else but dense layers?

- Optimisation is difficult (e.g. large number of parameters, vanishing gradients, local minima, etc).
- Many image properties are translation invariant (e.g. object and view-point translation).
- Learned low-level features are likely to be local (e.g. edge detector).
- Learned high level features are likely to be coarser (e.g. biology).

Building Blocks of a CNN

Convolution operation

- Convolve $K \times K$ weight patch $w_{k,l}$ with input image X of size $H \times W$. Receptive field - $K \times K$.
- Activation: $a_{i,j} = \Sigma_{k,l}\, w_{k,l} x_{i-k,j-l}$.
- With padding of P pixels and stride of S pixels, output resolution is $M_H \times M_W$, where $M_H = \left\lfloor \frac{H+2P-K}{S} + 1 \right\rfloor$.
- Usually has a bias term b.

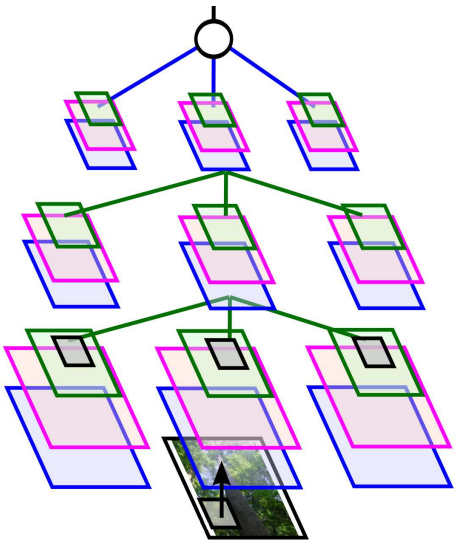
Non-linear activations

- Sigmoid: $\sigma(a) = \frac{1}{1+\exp(-a)} \in (0, 1)$. Saturated gradients, not zero centered.
- Hyperbolic tangent: $\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} = 2\sigma(2a) - 1 \in (-1, 1)$, saturated gradients, zero centered.
- ReLU: $[a]_+ = \max(0, a) \in [0, \inf)$. No saturation in (+) region, not zero centered, fast to compute.
- Softmax: $f_c(\mathbf{a}) = \frac{e^{a_c}}{\sum_{j=1}^{N} e^{a_j}} \in (0, 1)$. Formulates a probability distribution, last layer for classification networks.
- Other (for reference only): Leaky ReLU, ELU, etc.

Pooling operations

- Max pooling: $x_{i,j} = \max_{k,l} y_{i-k,j-l}$.
- Average pooling: $x_{i,j} = \frac{1}{K^2} \Sigma_{k,l}\, y_{i-k,j-l}$.
- Strided convolution can be used as a subsampling operation.

Simple CNN



Key properties:

- Convolutional Neural Network (CNN) with three convolutional layers (blue).
- Each convolutional layer is followed by non-linear activation (pink) and pooling (green) stages.
- Each convolutional layer (assume $S = 1$, $P = \lfloor K/2 \rfloor$) consists of three different convolution filters of size $K \times K$.
- Multi-filter convolution is performed as: $a_{i,j,c_{out}} = \Sigma_{k,l,c_{in}} w^{c_{out}}_{k,l,c_{in}} x_{i-k,j-l,c_{in}} + b^{c_{out}}$, where c_{in} - number of input filters and c_{out} - number of output filters.
- Each pooling stage subsamples its input by a factor of M.
- Input is a single channel (gray) image of size $H \times W$.
- Total number of parameters (assuming biases): $3\frac{WH}{M^6} + 21K^2 + 10$.
- A multi-layer perceptron with 3 hidden layers of D neurons each would have $WHD + 2D^2 + 4D + 1$ parameters.

Mini-Batch Gradient Descent

Batch (vanilla) gradient descent

- $L(W) = \Sigma_{n=1}^{N} L^{(n)}(W)$, where $L^{(n)}$ - a loss for a single data point n.
- Update: $W \leftarrow W - \eta \frac{\mathrm{d}\, L(W)}{\mathrm{d}\, W}$.
- Takes long time to accumulate derivatives over all training data points: 1 epoch $\rightarrow$ 1 update step.

Mini-batch (stochastic) gradient descent

- For each step randomly select $M \leq N$ data points (a mini-batch) and estimate gradients from them.
- If M=1 then it is called stochastic gradient descent.
- Objective function for a mini-batch:
  $L_b(W) = \frac{N}{M} \Sigma_{m=1}^{M} L^{(m)}(W) \approx \Sigma_{n=1}^{N} L^{(n)}(W)$.
- Update: $W \leftarrow W - \eta \frac{\mathrm{d}\, L_b(W)}{\mathrm{d}\, W}$.
- Efficient and frequent updates: 1 epoch $\rightarrow \frac{N}{M}$ updates.
- Classes may need to be balanced in mini-batches.

Mini-batch gradient descent with momentum

- Helps to escape "flat" regions of the objective function.
- Calculate gradient with momentum: $v \leftarrow \rho v - \eta \frac{\mathrm{d}\, L_b(W)}{\mathrm{d}\, W}$.
- Update: $W \leftarrow W + v$.
- Hyper-parameter ρ is called *momentum* and is typically set to $\rho = 0.9$ or 0.99.

Batch Normalization [7]

During training, updates of weights of a previous layer may significantly change the distribution of input values to the current layer. This slows down learning as current layer (and following layers) needs to adapt to these changes. Normalization of the outputs of the previous layer could help to alleviate this issue.

If the statistics of each layer's output over the entire dataset is known, those outputs can be normalised so that all output vectors have zero mean and unit variance. Since obtaining mean and variance over the whole dataset is computationally expensive, these values can be estimated by computing them for each batch during training.

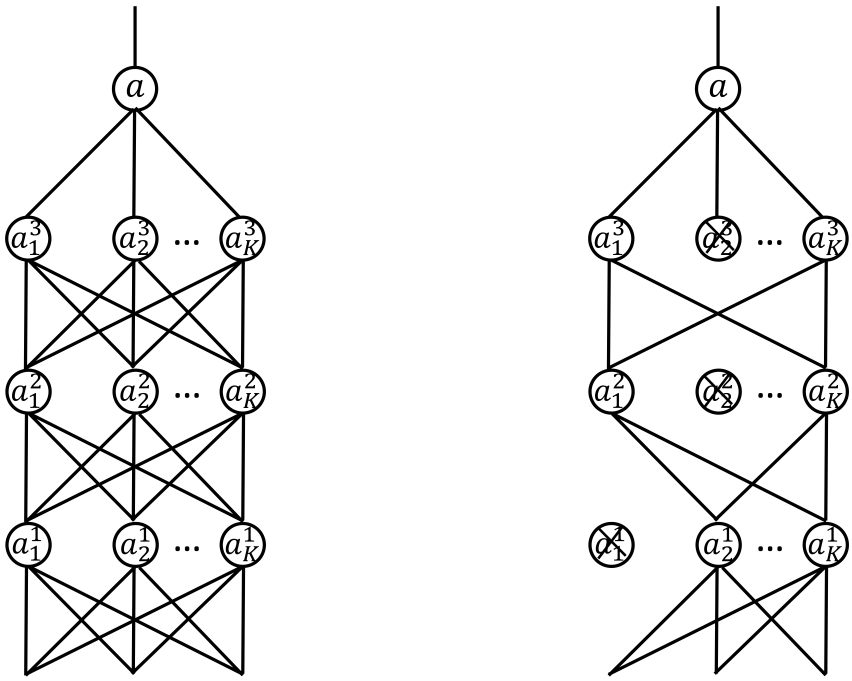
For a single neuron output y_k, the normalization is performed by computing $\hat{y_k}^{(m)} = \frac{y_k^{(m)} - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}}$, where $y_k^{(m)}$ is the value of y_k for a data point m within a mini-batch. Mini-batch mean of y_k is computed as $\mu_k = \frac{1}{M} \Sigma_m \, y_k^{(m)}$ and mini-batch variance is computed as $\sigma_k^2 = \frac{1}{M} \Sigma_m (y_k^{(m)} - \mu_k)^2$.

Normalized outputs of a mini-batch are also scaled and shifted $y_k^{'} = \gamma \hat{y}_k + \beta$ with learnt scaling parameter γ and shift parameter β in order to provide an ability for the network to *unlearn* the normalisation. Note $y_k^{'}$ is then used as the input to the subsequent layers in the network.

Once the network is trained, the true values of μ_k and σ_k^2 can be computed from the training dataset and kept fixed during testing.

Dropout

Dropout prevents network from overfitting by preventing co-adaptation (when some neurons are highly dependent on output of others). It also introduces redundancy in representation (e.g. multiple neurons perform similar task) and increases robustness to erroneous or missing feature responses.



For each forward pass, randomly set some neurons to zero with probability α.

At test time all the neurons are active so their outputs have to be multiplied by probability α.

Softmax Classifier

Softmax classifier is a generalization of a binary Logistic Regression classifier to multiple classes. We define a categorical distribution of C classes for data point $\mathbf{x}$:

$$P(t|\mathbf{x}, W) = \prod_c P(t = c|\mathbf{x}, W)^{\mathbf{1}[t=c]}, \text{ where}$$

$$P(t = c|\mathbf{x}, W) = y_c^{(n)} = f_c(W^T\mathbf{x}) = \frac{e^{\mathbf{w}_c^T\mathbf{x}}}{\sum_j e^{\mathbf{w}_j^T\mathbf{x}}}.$$

Here f_c is a softmax function, $W = \begin{bmatrix} \mathbf{w}_1 & \dots & \mathbf{w}_C \end{bmatrix}$ is classifier weight matrix made of class specific weight vectors $\mathbf{w}_c$.

Class prediction is made by $t^* = \text{argmax}_t\, P(t|\mathbf{x}, W)$.

Objective function

The following objective function corresponds to the negative log likelihood function over the whole dataset:

$$L(W) = -\sum_n \log P(t = t^{(n)}|\mathbf{x}^{(n)}, W).$$

Minimisation of this objective function is equivalent to minimising cross entropy or relative entropy between training data and predicted labels. Hence it is sometimes referred to as categorical cross entropy ($C > 2$) or binary crossentropy ($C = 2$).

Extension to DNN

Classifier weights W can be incorporated into a neural network via a layer consisting of C neurons with weights $\{\mathbf{w}_c\}$ (and biases $\{b_c\}$) and softmax activation functions.

Cross Entropy

The cross entropy objective function is widely used for training neural networks which perfom classification.

Cross entropy $H(P, Q)$ is an information theoretic concept which measures the expected number of bits (or nats) needed to identify an event drawn from a set of events if a coding scheme used for this set is optimized for an estimated probability distribution Q, rather than the true distribution P.

The following holds: $H(P, Q) = H(P) + D_{KL}(P||Q)$, where:

$H(P)$ is an entropy of distribution P which is defined as an the average level of *information* (or *surprise*) inherent in the variable's possible outcomes: $H(P) = -\sum_i P(i) \log P(i)$.

$D_{KL} = \sum_i P(i) \log \frac{P(i)}{Q(i)}$ is the Kullback-Leibler divergence. It is a common measure of how much one distribution diverges from another. It measures the expected number of extra bits that must be transmitted to identify a value drawn, if a code used corresponds to the probability distribution Q, rather than the true distribution P.

Note, $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ and $D_{KL}(P||Q) \geq 0$.

It is easy to show that: $H(P, Q) = -\sum_i P(i) \log Q(i)$.

Cross Entropy (continued)

Cross entropy objective function for a neural network which outputs a C dimensional vector $\mathbf{y}^{(n)}$, where $y_c^{(n)}$ encodes the probability of class c given input $\mathbf{x}^{(n)}$ and network weights W, can be written as:

$$L(W) = -\sum_n \sum_c t_c^{(n)} \log y_c^{(n)}.$$

Note here it is assumed that ground truth labels are encoded using one-hot encoding scheme (e.g $[0, 1, 0, ..., 0]$), where vector $\mathbf{t}^{(n)}$ encodes the correct class label (e.g. class 2) as 1 (e.g. $y_2^{(n)} = 1$) and other labels (e.g. $c \neq 2$) as 0 (e.g $y_c^{(n)} = 0$).

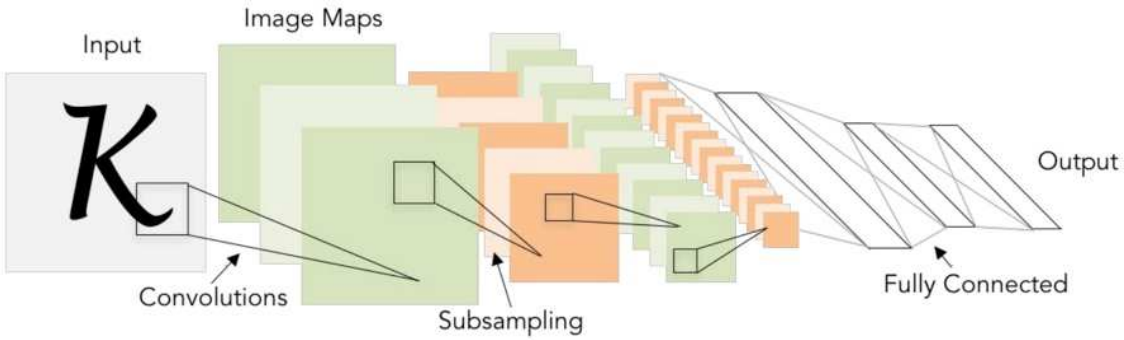
For example, an objective function for a network which takes a dataset of N D-dimensional inputs $\{\mathbf{x}^{(n)}\}$ with corresponding one hot encoded labels $\{\mathbf{t}^{(n)}\}$ and applies a single layer consisting of C neurons with softmax activation function to produce a C-dimensional output vector $\mathbf{y}^{(n)}$ would have this particular form:

$$L(W) = -\sum_n \sum_c t_c^{(n)} \log \frac{\exp\left(\mathbf{w}_c^T \mathbf{x}^{(n)}\right)}{\Sigma_j \exp\left(\mathbf{w}_j^T \mathbf{x}^{(n)}\right)}.$$

If bias parameter was made explicit, then the objective function would be:

$$L(W) = -\sum_n \sum_c t_c^{(n)} \log \frac{\exp\left(\mathbf{w}_c^T \mathbf{x}^{(n)} + b_c\right)}{\Sigma_j \exp\left(\mathbf{w}_j^T \mathbf{x}^{(n)} + b_j\right)}.$$

LeNet-5 [6]



- Modern CNN architecture for Computer Vision.
- $< 1\%$ error rate (close to the state-of-the-art) on MNIST dataset.

Task: Written digit classification.

Inputs: Images of shape $32 \times 32 \times 1$.

Outputs: A 10-dimensional vector corresponding to probabilities of 10 digit classes.

Architecture*:

(CONV1, K = 5×5, S = 1, C = 6, A = tanh), OS - $28 \times 28 \times 6$
(AVG-POOL1, K = 2×2, S = 2), OS - $14 \times 14 \times 6$,
(CONV2, K = 5×5, S = 1, C = 16, A = tanh), OS - $10 \times 10 \times 16$
(AVG-POOL2, K = 2×2, S = 2), OS - $5 \times 5 \times 16$
(CONV3, K = 5×5, S = 1, C = 120, A = tanh), OS - $1 \times 1 \times 120$
(FC1, C = 84, A = tanh), OS - 84
(FC2, C = 10, A = softmax), OS - 10

K - kernel size, S - stride, A - activation function, C - number of channels for convolutional layer and number of output units for a fully connected layer, OS - output shape. No padding was applied.

* - architecture is simplified

Other details: categorical cross-entropy loss, total number of parameters ≈ 60000, optimizer - SGD.

Deep Learning in Practice

Data preparation

- Creating train, validation, test data split.
  - One should never train or determine hyper-parameters on test data split.
- Input data pre-processing: mean subtraction or normalisation.
- Data augmentation: rotation, horizontal flips, random crops and scales, colour jitter, randomized contrast and brightness, utilizing synthetic data, etc.

Weight initialization

- Randomly initializing weights from a carefully chosen distribution (e.g. weighted normal distribution).
- Sequential training by adding one layer at a time to the network. Fine-tuning on the whole architecture.
- Using pre-trained weights on other tasks or datasets.

Aiding optimization

- Regularization: weight regularization (e.g. L2 norm), dropout.
- Using batch normalization.
- Using advanced optimizers (for reference only): AdaGrad, RMSPRop, Adam.
- Using learning rate schedulers (for reference only): polynomial, exponential, step-wise.

Deep Learning in Practice

Performance tuning

- Using small data (e.g. 1-10 images) for initial experiments.
- Analysing train and validation loss graphs in order to determine learning rate anomalies (e.g. large fluctuation or even increase in training loss) and overfitting (e.g. large difference between training and validation loss).
- Avoiding over-fitting. Selecting regularisation parameters which increase performance on validation data.
- Performing hyper parameter optimization directly on train-validation data split or by using K-fold cross-validation: splitting dataset into K groups and randomly using one group for testing (K times).
- Training ensembles of models.

Tools

- Modularised layers that define forward and backward passes for auto-differentiation.
- Backbones: Tensorflow (Google), PyTorch (Facebook), Caffe2 (Facebook), MXNet (Amazon).
- High level wrappers: Keras (Tensorflow), Sonnet (Tensorflow), torch.nn (Pytorch).

Note

A large portion of the material presented until this point was adapted from Dr. Rich Turner's slides used in 2018-2019 for 4F12 Computer Vision course. Note there are some changes in notation used (see Examples Papers and historical 4F12 IIB Tripos).

Image Classification

dog, cat, ..., elephant?

Training data

- Inputs: images $\{I^{(n)}\}_{n=1}^{N}$, where $I^{(n)} \in R^{H \times W \times 3}$, H - height, W - width and N - number of training images.
- Targets: one-hot encoded class labels $\{\mathbf{t}^{(n)}\}$ of C classes, where $\mathbf{t}^{(n)} \in R^{C}$.

Hand designed vs learnt features

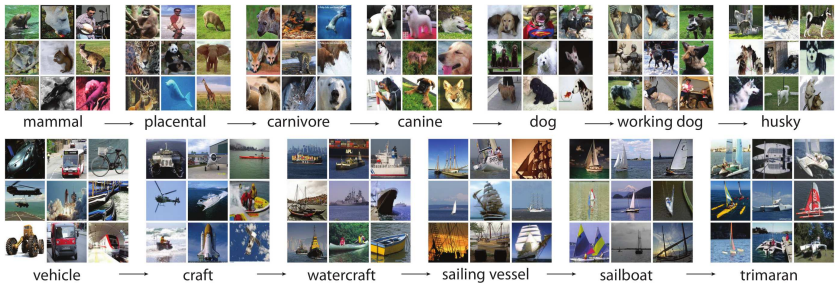
- Hand designed features: histogram of gradients, SIFT, filter banks, etc.
  - Very fast and relatively robust to simple variations in images: brightness or contrast change, orientation, etc.
  - Weak discriminative power due to their lack of ability to model hierarchies of features.
- Learned features: response of convolutional layers.
  - High accuracy and robustness to complex variations (e.g. deforming objects) in images.
  - Ability to construct feature hierarchies.
  - Requires a lot of data to learn.

Examples of classifiers

Logistic regression (two class problems only), softmax classifier (multi-class), nearest neighbour classifier (for reference only), multi-class Support Vector Machine (for reference only), etc.

Large Scale Visual Recognition

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [9]



- Key benchmark for large scale image classification.
- Often used for evaluating convolutional neural network (CNN) architecture designs.
- Built on ImageNet [10] dataset:
  - Total number of images: 14,197,122.
  - Images are labelled with presence of hierarchically organised categories - 21841 different categories.
  - Examples of high level object categories: person, device, animal, fruit, etc.
- Main classification benchmark: $\approx$ 1.2M training images covering 1000 object categories (one category per image), 50K validation and 100K test images.

Performace over years

2010-2011 Classifiers using hand designed features $\rightarrow$ **top 5** error $> 25.8\%$.

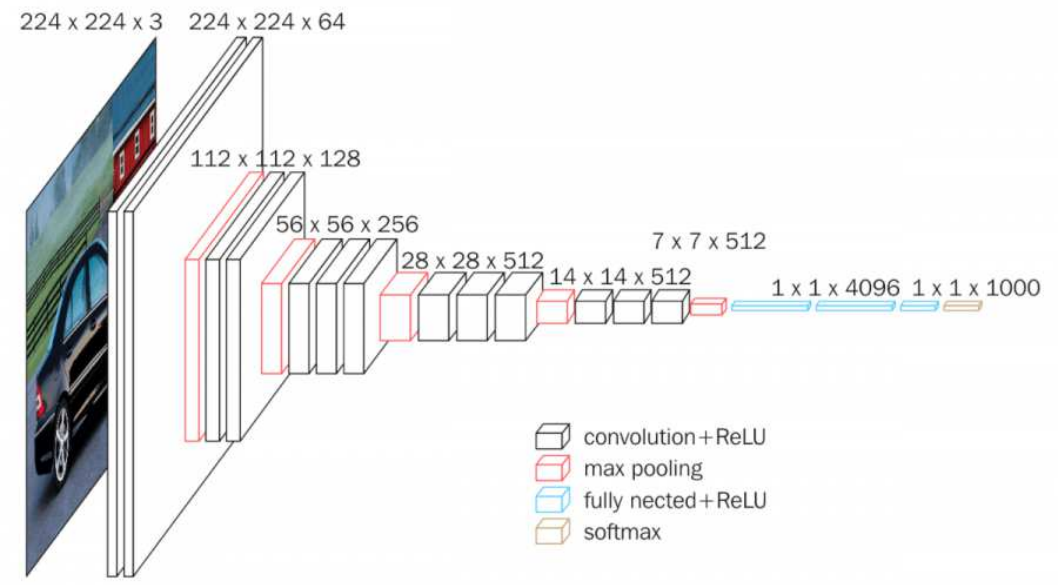
2012 First CNN wins: AlexNet [8] (8 layers)$\rightarrow$ err. $\approx 16.4\%$.

2014 Winner: GoogleNet (22 layers) $\rightarrow$ err. $\approx 6.7\%$.

2014 Runner up: VGG-19 [11] (19 layers) $\rightarrow$ err. $\approx 7.3\%$.

2015 Winner: ResNet-152 [12] (152 layers) $\rightarrow$ err. $\approx 3.6\%$.

VGG-16 [11]



Contribution

Showing that increasing depth can result in superior accuracy.

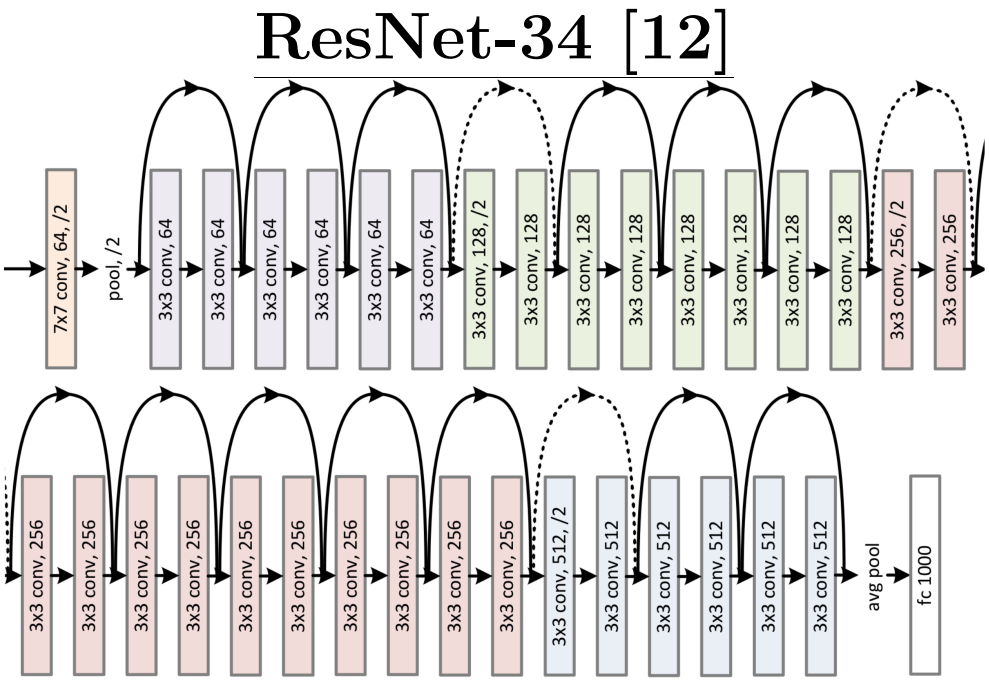
Observations

- Can build deeper architectures (e.g. 16 convolution and fully connected layers) by using small filters (3×3).
- Stack of three 3×3 convolutional filters of stride 1 have the same effective receptive field as one 7×7 convolution and less parameters ($3 \times (3^2C^2)$ vs 7^2C^2). Here C - number of channels. Biases are not counted.
- Also it allows for learning more complex features by allowing for more depth.

VGG-16 [11] (continued)

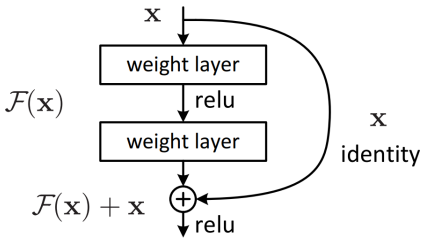
Properties

- Very simple and homogeneous architecture.
- Convolution layers perform of 3×3 convolutions with stride 1 and padding 1. ReLU is used as an activation function.
- Pooling layers perform 2×2 max-pooling with stride 2 and no padding.
- The final convolutional layer is flattened and passed through two fully connected layers with 4096 units each.
- Finally a fully connected layer with 1000 units (number of classes) and softmax activation function is applied in order to provide class prediction probabilities.
- Total parameters: $\approx$138M (not counting biases).
- Features of pre-final FC layer generalise well to other tasks.



Contribution

Demonstrating that extremely deep networks (e.g. 152 trainable layers) can be trained successfully when residual connections are used.



Observation

Simply stacking convolutional, non-linearity and max-pooling layers performs worse on both train and test data after certain depth. Hence it must be an optimisation problem.

Solution

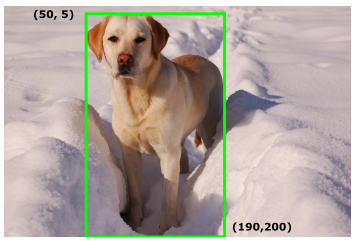
Residual connections enable network to fit a (potentially) simpler residual value $F(\mathbf{x})$ instead of directly fitting the desired value $H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}$. Also this allows gradients to be backpropagated easier to early layers. Note, a deeper model with residual connections should perform at least as good as a shallow one.

ResNet-34 [12] (continued)

Properties

- Input image ($224 \times 224 \times 3$) is convolved with 7×7 convolution of 64 channels at stride 2.
- Followed by 3×3 max-pooling operation with stride 2.
- The main architecture of ResNet-34 is made of stacked residual blocks consisting of two 3×3 convolutional layers (padding 1, stride 1) as illustrated in the figure in the previous slide.
- Periodically the number of channels is doubled and filters are downsampled using convolution of stride 2 (indicated by dashed lines). Before addition (in a residual block) input $\mathbf{x}$ is passed through a 1×1 convolution layer with stride 2 and appropriate number of channels.
- Redundant fully connected (FC) layers (compared to VGG) are removed at the end. Instead, global average pooling (averages over spatial dimensions for each channel) after the final convolution layer is applied.
- Only one fully connected layer with softmax activation function is used at the end to output 1000 class probability vector.
- Heavy use of batch normalization - applied after each convolution layer (before activation function).
- Commonly used architecture, adopted for many tasks (e.g. segmentation, detection, etc). Approx. 21M parameters.

Object Detection

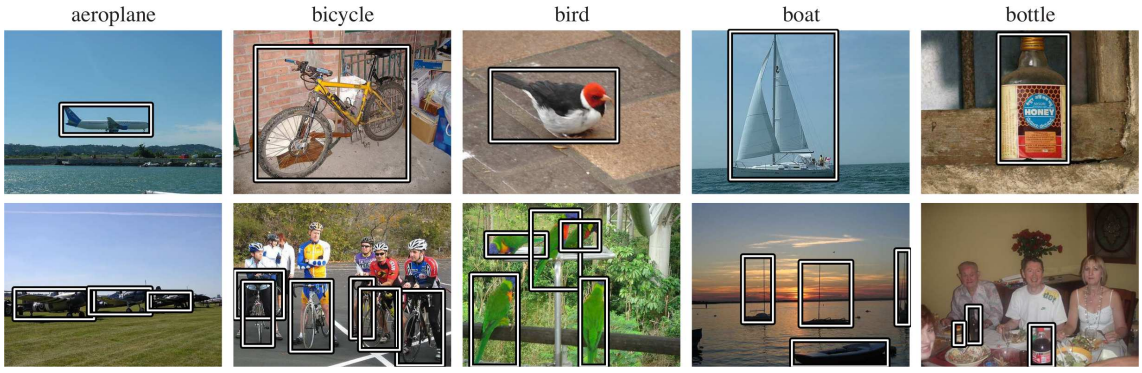


{(x=50, y=5, w=140, h=195, dog), ... }

Training data

- Inputs: images $\{I^{(n)}\}_{n=1}^{N}$ as in image classification.
- Targets: $\{\{(x^{(k,n)}, y^{(k,n)}, w^{(k,n)}, h^{(k,n)}, \mathbf{t}^{(k,n)})\}_{k=1}^{K^{(n)}}\}_{n=1}^{N}$, where $K^{(n)}$ is a number of objects in image n. Tuple $(x^{(k,n)}, y^{(k,n)})$ corresponds to the top left corner of the bounding box of the object k in image n. Tuple $(w^{(k,n)}, h^{(k,n)})$ corresponds to the width and height of the bounding box and $\mathbf{t}^{(k,n)} \in R^C$ is a corresponding class label. Often has a confidence score associated with class label prediction.

PASCAL Visual Object Classes (VOC) Challenge [13]



- Important benchmark for object detection.
- Combined train and validation datasets consists of 11530 images containing 27450 annotated objects.
- 20 object categories (e.g. person, bird, cat, bicycle, etc).

Object Detection Evaluation

Note that there is no one-to-one mapping between predicted bounding boxes and ground truth bounding boxes. Hence the following procedure is used.
If a confidence score of a predicted box is higher than a chosen threshold then:

- An intersecion over union (IoU) score (overlapping area divided by total area) is computed for a predicted bounding box and all nearby ground truth (GT) bounding boxes of the same class.
- If a GT box exist for which the IoU score is ≥ 0.5, then this prediction is assumed to be a true positive (TP), otherwise, a false positive (FP). If multiple predictions are made for the same GT bounding box, only one of the predictions is considered to be TP.

Ground truth boxes which do not overlap with any predicted bounding box of the same class are marked as false negatives - FN.

Precision - a fraction of correctly identified bounding boxes among all predictions made: $P = \frac{|TP|}{|TP|+|FP|}$.

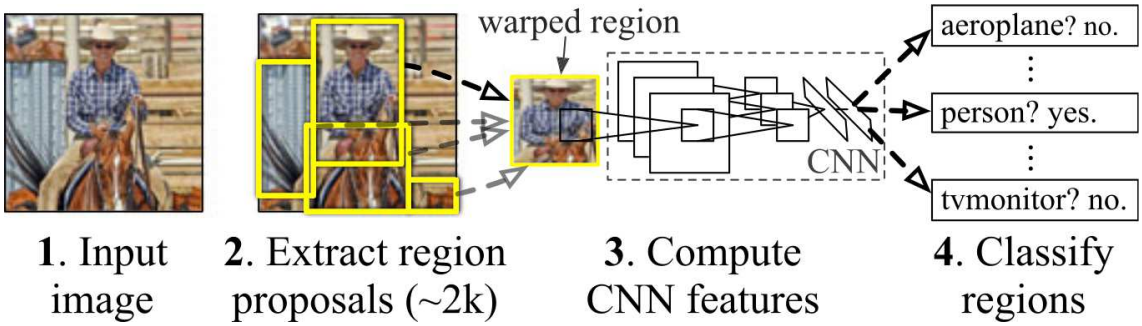
Recall - a fraction of correctly retrieved bounding boxes among all ground truth bounding boxes: $R = \frac{|TP|}{|TP|+|FN|}$. Different levels of recall can be achieved by using a threshold on the confidence of the bounding box classifier.

Average precision measures* expected precision for uniformly sampled recall levels.

Mean average precision calculates the mean AP accross all classes: $mAP = \frac{\sum_c AP(c)}{C}$. It is used as a performance metric for detection algorithms.

** - PASCAL VOC uses a slightly more complex version of AP but it is out of the scope of this course.*

RCNN [14]



Approach

- Image region candidates are obtained by a fixed algorithm - selective search [15] (for reference only).
- Each image region is warped to a fixed size square image.
- Resized image is fed into a CNN (e.g. VGG-16) which is pretrained on ILSVRC [9] dataset.
- 1000-way specific classification layer is replaced with a randomly initialized C+1 way classification layer.
- The additional class is background class (no object of interest).
- All object proposals with > 0.5 IoU with respect to some ground truth bounding box are considered as positive examples of the particular class and the rest - as negatives.
- Finetuning of CNN features and classification layers is performed on PASCAL VOC [13] dataset.
- Mini batches are rebalanced to contain many examples of objects as most proposed regions are background.
- Improved bounding box values are regressed along with predicted class labels (for reference only).

Properties

- Slow training and testing as this approach needs to classify $\approx$ 2K region proposals per image.
- Region proposal algorithm does not adapt to the dataset used.

Extensions to RCNN (e.g. Faster-RCNN [16]) significantly improved the effficiency of the network by using CNN extracted features for both region proposal and subsequent classification (for reference only).

Semantic Segmentation

Is the class of pixel (x=125, y=75) dog, cat, ... ?

Training data

- Inputs: $\{I^{(n)}\}_{n=1}^{N}$ as in image classification.
- Targets: $\{T^{(n)}\}_{n=1}^{N}$, a multi-dimensional array of one-hot encoded pixel level labels, where $T^{(n)} \in R^{H \times W \times C}$.

Objective function

- Any classifier based objective function which is applied for each pixel and summed over all pixels.
- Pixel level cross entropy: $-\sum_n \sum_{x,y} \sum_c T_{x,y,c}^{(n)} \log Y_{x,y,c}^{(n)}$. Here $Y_{x,y,c}^{(n)}$ is a multi-dimentional array output of a CNN encoding class label c probability for pixel (x, y).

Extension to DNN

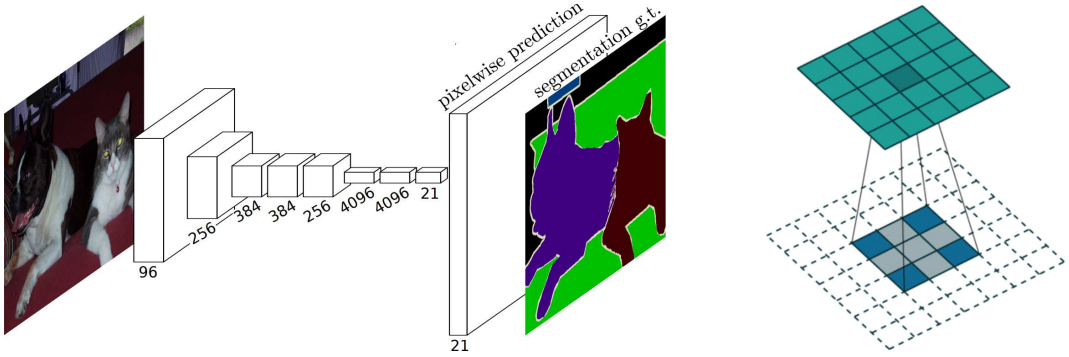
- Apply a softmax classifier for each pixel (x, y) individually. E.g. as a 1×1 convolution (stride - 1, padding - 0) with C channels and a softmax activation function.
- To obtain class predictions, calculate most likely class label for each pixel: $c^* = \text{argmax}_c \, Y_{x,y,c}^{(n)}$.
- Evaluate performance by computing mean IoU over all classes:
  - IoU for each class is computed as follows: $IoU = \frac{|TP|}{|TP|+|FP|+|FN|}$. Here TP - a set of correctly identified pixels, FP - a set of pixels with incorrectly predicted class, FN - a set of ground truth pixels for which a particular class was not predicted.

Microsoft COCO Dataset [17]

Microsoft Common Objects in Context Dataset [17]

- Important benchmark for semantic segmentation of images.
- Images of complex scenes which contain common objects in their usual context.
- 330K images in total.
- 80 object categories with a well-defined shape (e.g. car, person, etc.), referred as *things*.
- 2.5 million object instances labelled.
- 91 *stuff* categories which do not have well defined shape (e.g. sky, grass, water, etc).

Fully Convolutional Network [18]



Approach

- Fully Convolutional Network (FCN) poses semantic segmentation problem as efficient per pixel classification problem.
- Uses an existing image classification network (e.g. VGG-16 or AlexNet [8]) to obtain convolutional image features at low resolution.
- Fully connected layers at the end of a classification architecture (e.g. like in VGG-16) are replaced with convolutions (e.g. 7×7 and 1×1).
- Additional convolutional layer (kernel 1×1) is applied with 21 (number of classes) channels.
- Backwards strided convolution (also called deconvolution) is used for up-sampling to obtain per pixel class label predictions at the original resolution.
  - Aims at *inverting* the convolution operation so that output of which would be of increased resolution.
  - It can be performed by applying convolution on an image obtained by inserting zero value rows and columns between original pixels.

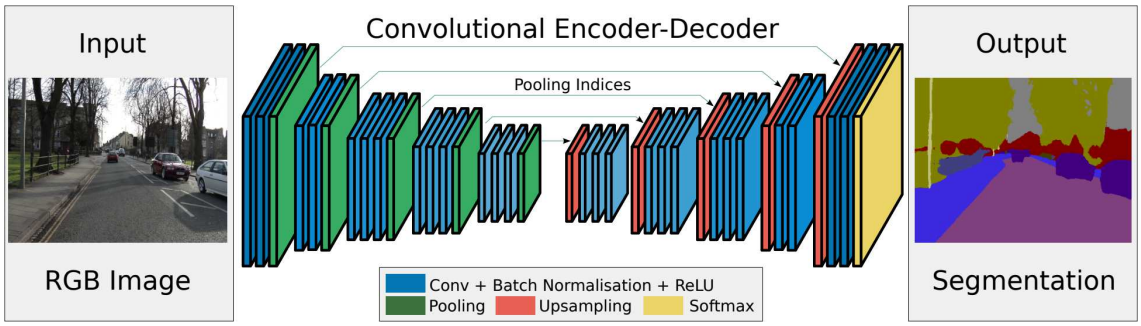
FCN [18] (continued)

Approach (continued)

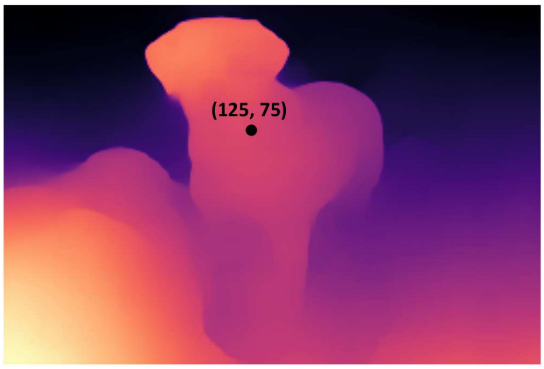
- Softmax activation can be used for the final layer in order to produce per pixel class probabilities.
- FCN is trained with pixel level cross entropy loss.

Properties

- Fully convolutional architecture allows for efficient per pixel prediction avoiding the need of running a network multiple times.
- Reusing architecture of a classification network allows for pretraining of this network on image classification task which can utilise large datasets (e.g. ImageNet [10]).
- Aggressive learnt upsampling (32 times) struggles to preserve sharp edges.
- Subsequent semantic segmentation frameworks such as SegNet [19] proposed solutions which can preserve detailed edges. For example, gradual upsampling of feature maps can be performed by using memorized max-pooling indices from the corresponding VGG-16 layers (for reference only).



Monocular Depth Estimation



Depth of pixel (x=125, y=75) $= 2m$**?**

Training data

- Inputs: $\{I^{(n)}\}_{n=1}^{N}$ as in image classification.
- Targets: $\{D^{(n)}\}_{n=1}^{N}$, a multi-dimensional array of pixel level depth values, where $D^{(n)} \in R^{H \times W \times 1}$.

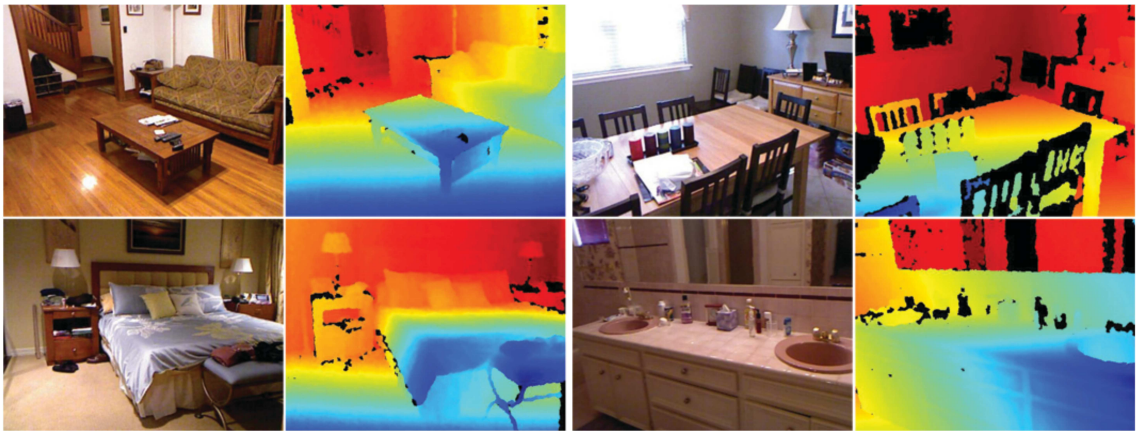
Objective function

- A pixel level function measuring distance between network output $Y_{x,y,1}^{(n)}$ and ground truth depth $D_{x,y,1}^{(n)}$.
- Example - sum of squared differences: $\sum_{n} \sum_{x,y} (Y_{x,y,1}^{(n)} - D_{x,y,1}^{(n)})^2$.

Extension to DNN

- Apply a convolutional layer with a single output channel and Leaky ReLU or other suitable activation function as a final layer.
- To obtain depth predictions, simply use network output.
- Evaluate performance by computing root mean squared error for each image: $RMSE = \sqrt{\frac{1}{WH} \sum_{x,y} \left(Y_{x,y,1}^{(n)} - D_{x,y,1}^{(n)} \right)^2}$.

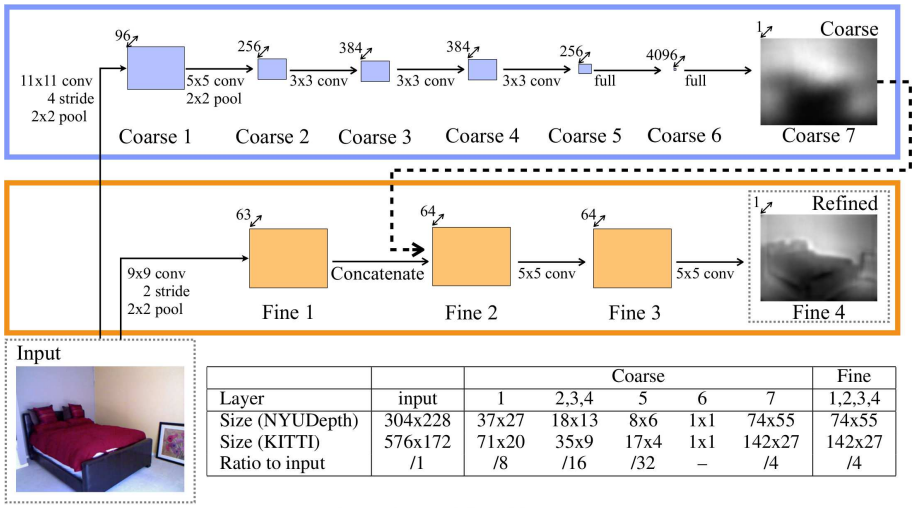
NYU Depth V2 Dataset [20]



The NYU Depth V2 dataset is comprised of multiple video sequences from a variety of indoor scenes. It is recorded using both the RGB and Depth cameras from the Microsoft Kinect.

- Spans 464 indoor scenes.
- Consists of 407,024 unlabeled frames.
- Also has 1449 densely labeled pairs of aligned RGB and depth images.
- Each object is labeled with a class and an instance number.

Eigen et. al. [21]



		Coarse					Fine
Layer	input	1	2,3,4	5	6	7	1,2,3,4
Size (NYUDepth)	304x228	37x27	18x13	8x6	1x1	74x55	74x55
Size (KITTI)	576x172	71x20	35x9	17x4	1x1	142x27	142x27
Ratio to input	/1	/8	/16	/32	–	/4	/4

Approach

- Formulated fine depth prediction as two tasks of coarse global depth prediction and fine detail refinement.
- Coarse-scale network is used to predict overall depth map structure using a global view of the scene.
- Final two layers are fully connected hence the receptive field is the whole image.
- Final FC layer output is reshaped into an image shape.
- Coarse-scale network output is concatenated as an additional feature map into the second layer of the fine scale network.
- Fine-scale network edits coarse prediction in order to align it with local details such as object or wall edges.
- Coarse-scale network is trained first and is kept fixed when training the fine-scale network.
- Used a more sophisticated scale invariant objective function since total error is heavily affected by how well mean depth is predicted.

Properties

- Hard to maintain accurate high level details in predictions. Many alternative up-sampling techniques have been proposed since.
- Hard to obtain training data, hence various unsupervised techniques have been explored.

Image Matching

Image 1

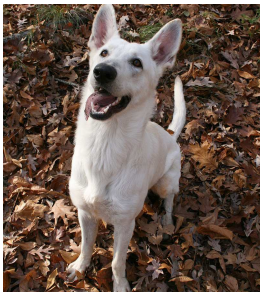


Image 2

Similar to 1 or 2?

Training data

- Inputs: a pair of query images $I_1^{(n)}$ and $I_2^{(n)}$.
- Targets: similarity score $s^{(n)} \in R$ between the images. E.g. $s^{(n)} = 1$ if images are similar and $s^{(n)} = 0$ if images are dissimilar.

Extension to DNN

- Apply a convolutional layer of K layers to form a K-dimensional embedding vector $\mathbf{e}$ per image.
- See slide on Deep Metric Learning for training details.
- Evaluate using an appropriate procedure for your task (e.g. classification accuracy for face recognition).

Applications

Descriptor learning, image retrieval, localisation, face recognition, etc.

Labeled Faces in the Wild: a database for studying face recognition in unconstrained environments [22]

- A database for unconstrained face recognition.
- 13,000 images of faces from the web.
- 1680 of the people have two or more photos.

Distance and Similarity Learning

Distance function (metric)

- Distance function over set X is a pairwise function $D : X \times X \rightarrow R$.
- $D(\mathbf{x}, \mathbf{x}^{'}) \geq 0$ (non-negativity)
- $D(\mathbf{x}, \mathbf{x}^{'}) = 0$ if and only if $\mathbf{x} = \mathbf{x}^{'}$ (identity of indiscernibles)
- $D(\mathbf{x}, \mathbf{x}^{'}) = D(\mathbf{x}^{'}, \mathbf{x})$ (symmetry)
- $D(\mathbf{x}, \mathbf{x}^{''}) \leq D(\mathbf{x}, \mathbf{x}^{'}) + D(\mathbf{x}^{'}, \mathbf{x}^{''})$ (triangle inequality)
- Example: Mahalanobis distance - $D_M = \sqrt{(\mathbf{x} - \mathbf{x}^{'})^T W (\mathbf{x} - \mathbf{x}^{'})}$.
- **Pseudo distance** - same as above, with exception to identity of indiscernibles.

Similarity function

- A similarity function over set X is a pairwise function $S : X \times X \rightarrow R$. Quantifies the similarity between two objects.
- $S(x, x^{'}) = S(x^{'}, x)$ (symmetry)
- Example: Cosine similarity $S_{cos}(\mathbf{x}, \mathbf{x}^{'}) = \frac{\mathbf{x}^T \mathbf{x}^{'}}{||\mathbf{x}||_2 ||\mathbf{x}^{'}||_2}$.

Training procedure

Given a metric find its parameters W^* such as $W^* = \text{argmin}_W L(W, S, D, R)$, where:

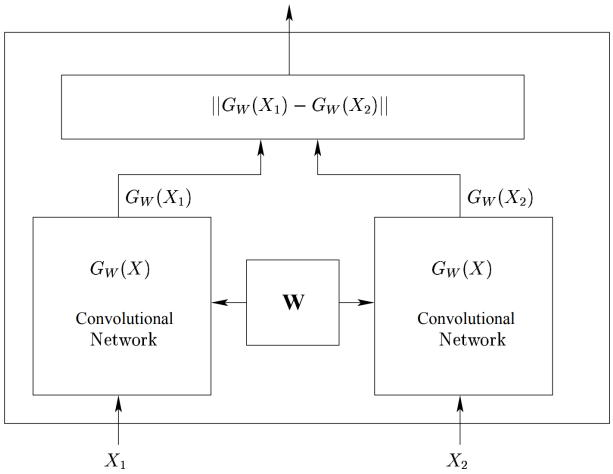
- L is a loss function that penalizes violated constraints for different pairs of data. A regularizer on weights can be added.
- S=$\{(\mathbf{x}_i, \mathbf{x}_j)$: $\mathbf{x}_i$ and $\mathbf{x}_j\}$ is a set of tuples of similar data points and
- D=$\{(\mathbf{x}_i, \mathbf{x}_j)$: $\mathbf{x}_i$ and $\mathbf{x}_j\}$ is a set of tuples of dissimilar datapoints.
- R = $\{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$: $\mathbf{x}_i$ shoud be more similar to $\mathbf{x}_j$ than to $\mathbf{x}_k\}$ is a set of tuples of relatively ranked datapoints.

Deep Metric Learning

Definition

- Construct a learnable metric* $D_W = ||G_W(\mathbf{x}) - G_W(\mathbf{x}^{'})||$, where G_W is a deep neural network parametrised with W.
- * - D_W is a pseudo-metric.
- Example: Siamese Networks [23].

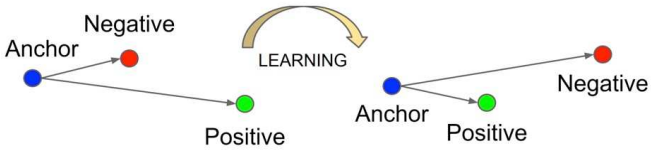
Siamese Networks [23]



Approach

- Two images I_1 and I_2 are passed into two networks with shared weights.
- Each network outputs a K-dimensional vector.
- An objective function referred to as *contrastive loss* is applied: $L = s\frac{1}{2}D_W^2 + (1-s)\frac{1}{2}(\max(0, \Delta - D_W))^2$. Here Δ is a margin parameter.
- This objective function encourages data points which are similar to be encoded to vectors which are close to each other and points which are dissimilar to be encoded to vectors which are far from each other (outside margin).

Face Recognition - FaceNet [24]



Approach

- CNN is used to learn a 128 dimension unit vector $\mathbf{e}$ for image I.
- Two images are considered to share identity if euclidean distance between corresponding vectors is smaller than a set threshold.
- A very large dataset consisting of $100M$-$200M$ face thumbnails and $8M$ identities is created for training.
- Triplets of anchor image I_A, positive image I_+ (same identity as anchor) and negative image I_- (different identity than anchor) are extracted.
- Objective function refferred as *triplet loss* is used: $L = \sum_{m=1}^{M} \max(0, ||\mathbf{e}_A^{(m)} - \mathbf{e}_+^{(m)}||_2^2 - ||\mathbf{e}_A^{(m)} - \mathbf{e}_-^{(m)}||_2^2 + \Delta)$. Here Δ is margin and M a number of triplets in a batch.
- To speed up convergence hard-positive and hard-negative mining is used for selecting informative triplets.

Properties

- Triplet loss enforces only relative ranking (with margin) between acnhor-positive (AP) and anchor-negative (AN) distances. Whereas contrastive loss explicitly minimizes AP and maximizes AN distances (with margin), which may unnecessary force potentially highly different embeddings to be close to each other.
- Despite requirement of large amounts of data - a very elegant method providing the state-of-the-art performace and a blueprint for other image matching tasks.

References

[1] McCulloch, W. and Pitts, W. *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics, 5:115-133, 1943.

[2] Rosenblatt, F. *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological Review, 65(6), 386-408, 1958.

[3] Widrow, B. and Hoff, M. E. *Adaptive switching circuits.* In IRE WESCON Convention Record, 4, 96-104, 1960.

[4] Fukushima, K. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.* Biological Cybernetics, 36, 193202, 1980.

[5] Rumelhart, D., Hinton, G., and Williams, R. *Learning representations by back-propagating errors.* Nature, 323, 533536, 1986.

[6] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. *Gradient based learning applied to document recognition.* In Proceedings of the IEEE, 86(11), 22782324, 1998.

[7] Ioffe, S., Szegedy, C., *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* In Proceedings of the International Conference on Machine Learning (ICML), 2015.

[8] Krizhevsky, A., Sutskever, I., Hinton, G. *Imagenet classification with deep convolutional neural networks.* Advances in Neural Information Processing Systems (NIPS), 1097-1105, 2012.

[9] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., Fei-Fei, L. *ImageNet large scale visual recognition challenge.* International Journal of Computer Vision (IJCV), 115 (3), 211-252, 2015.

[10] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, K. *ImageNet: A Large-Scale Hierarchical Image Database.* In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009.

[11] Simonyan, K., Zisserman, A., *Very deep convolutional networks for large-scale image recognition.*, In Proc. International Conference on Learning Representations (ICLR), 2015.

[12] He, K., Zhang, Ren, S., Sun, J., *Deep residual learning for image recognition.* In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[13] Everingham, M., Van-Gool, L., Williams, C. K. I., Winn, J., Zisserman, A. *The Pascal Visual Object Classes (VOC) Challenge.* International Journal of Computer Vision (IJCV), 88(2), 303-338, 2010.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. *Region based convolutional networks for accurate object detection and segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(1),142-158, 2016.

[15] Uijlings, J., van de Sande, K., Gevers, T., Smeulders, A., *Selective search for object recognition.* International Journal of Computer Vision (IJCV), 104(2), 154171, 2013.

[16] Ren, S., He, K., Girshick, R., Sun, J. *Faster R-CNN: Towards real-time object detection with region proposal networks.*, Advances in Neural Information Processing Systems (NIPS), 2015

[17] Lin, T-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., Dollár, P. *Microsoft COCO: Common Objects in Context* In Proc. European Conference on Computer Vision (ECCV), 2014

[18] Long, J., Shelhamer, E., Darrell, T. *Fully convolutional networks for semantic segmentation.* In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[19] Badrinarayanan V, Kendall A, Cipolla R. *SegNet: A deep convolutional encoder-decoder architecture for image segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12):2481-2495, 2017.

[20] Silberman, N., Kohli, P., Hoiem, D., Fergus, R., *Indoor segmentation and support Inference from RGBD Images.* In Proc. European Conference on Computer Vision (ECCV), 2012.

[21] Eigen, D., Puhrsch, C., Fergus, R., *Depth Map Prediction from a Single Image using a Multi-Scale Deep Network.* Advances in Neural Information Processing Systems (NIPS), 2014.

[22] Huang, G. B., Ramesh, M., Berg, T., Learned-Miller, E. *Labeled Faces in the Wild: a database for studying face recognition in unconstrained environments* University of Massachusetts, Amherst, Technical Report 07-49, 2007.

[23] Chopra, S., Hadsell, R., LeCun, Y., *Learning a similarity metric discriminatively, with application to face verification* In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005.

[24] Schroff, F., Kalenichenko, D., Philbin, J. *FaceNet: A unified embedding for face recognition and clustering* In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.