CS6421: Deep Neural Networks

Gregory Provan

Spring 2020

Lecture xx: Practical Issues

Based on notes from John Canny, Ismini Lourentzou

Overview

- Practical Issues in Deep Networks: Overview
- Initialisation
  - Architecture
  - Activation functions
  - Loss functions
  - Initial parameters
- Data Preprocessing
- Training

Motivation

- Building high-performance deep networks is challenging
  - Many parameters to set
  - Architectures can be difficult to design
  - May need data pre-processing

Practical Issues (to be addressed)

- Network initialisation
- Data pre-processing
- Network training
- Inference issues

Overview

- Practical Issues in Network
  Initialisation: Overview
  - Architecture
  - Activation functions
  - Loss functions
  - Initial parameters
- Data Preprocessing
- Training

General Guidelines

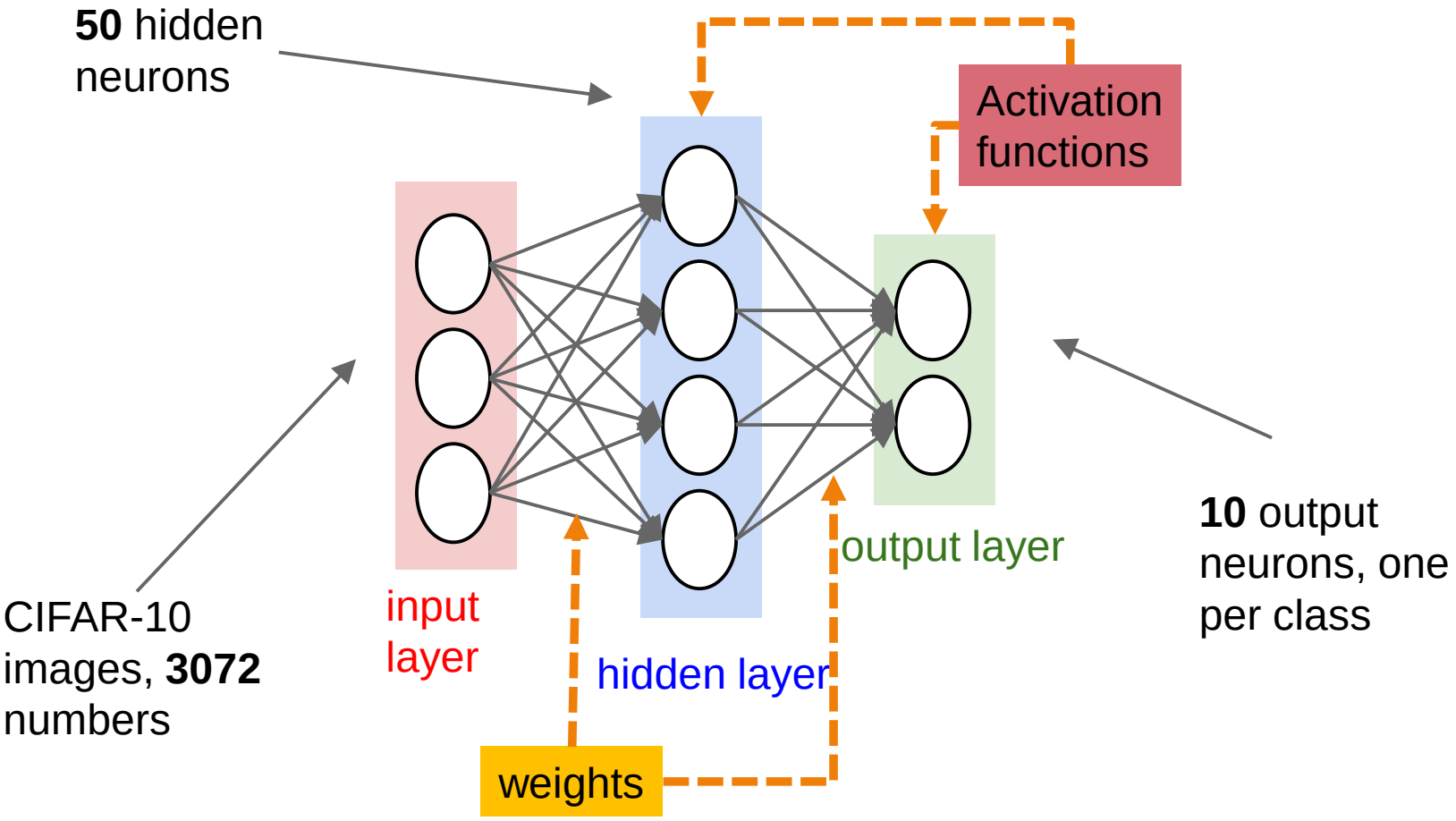
- Match application to architecture
  - Structured data: CNN
    - Images, text, voice
  - Temporal data: RNN/LSTM
    - Time-series analysis (stock forecasting)
- Use appropriate activation functions at each layer
- Weight initialisation
- Use hyper-parameter optimisation
  - Number of layers, size of layers, batch size, stride-length, etc.

Overview

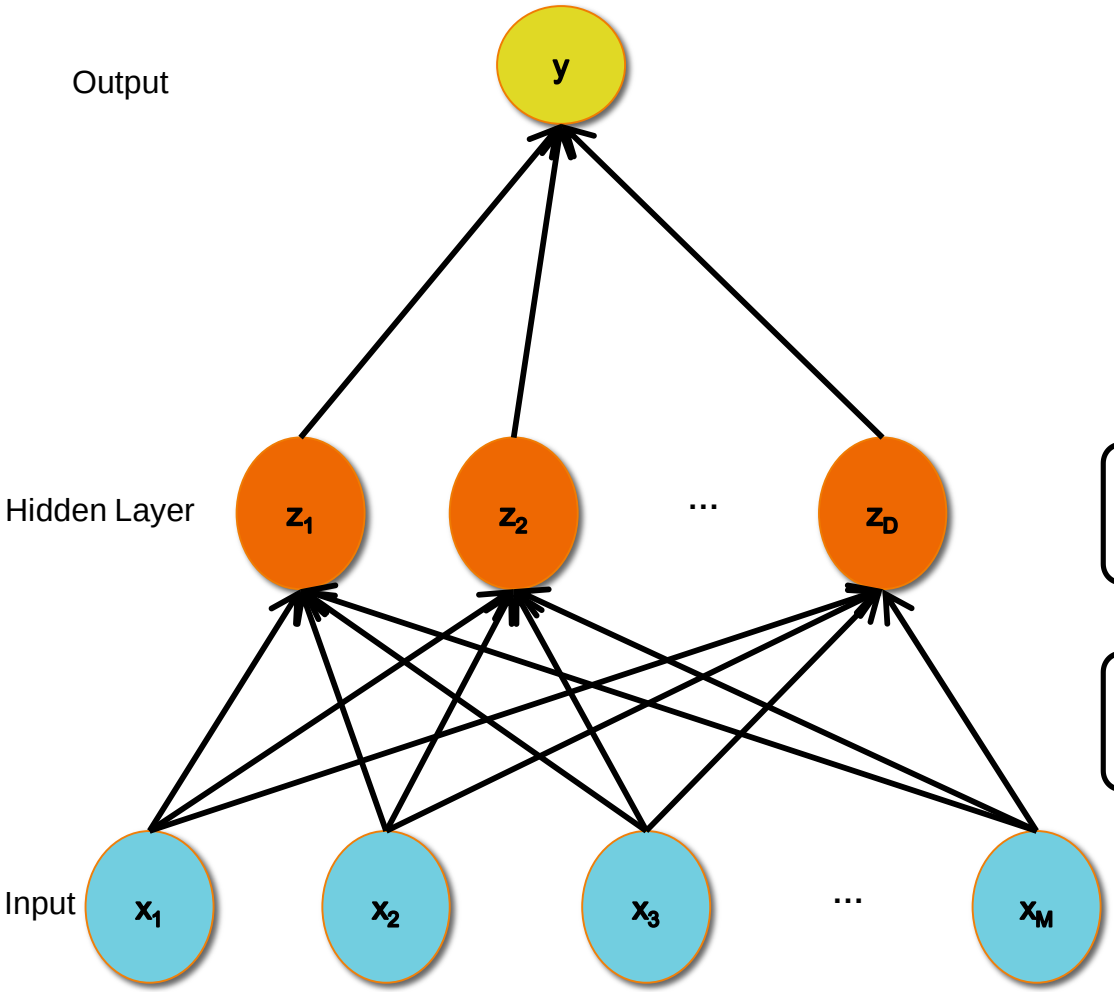
- Practical Issues in Network Initialisation: Overview
  - **Architecture**
  - Activation functions
  - Loss functions
  - Initial parameters
- Data Preprocessing
- Training

Choose the architecture

Example: start with one hidden layer of 50 neurons:



Neural Network



(F) Loss

$$J = \frac{1}{2}(y - y^*)^2$$

(E) Output (sigmoid)

$$y = \frac{1}{1+\exp(-b)}$$

(D) Output (linear)

$$b = \sum_{j=0}^{D} \beta_j z_j$$

(C) Hidden (sigmoid)

$$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$$

(B) Hidden (linear)

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$$

(A) Input

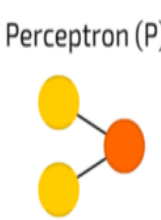
Given $x_i, \ \forall i$

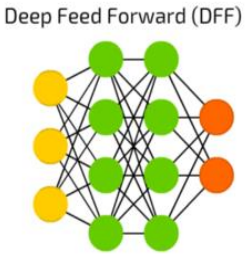
Network Architecture

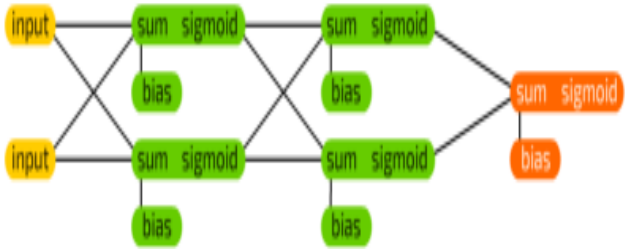
- Map application to architecture
- Always try state-of-the-art architectures first
  - Often available
  - These architectures work for good reasons
- Example: YOLO for vision applications

Zoo of Architectures

Asimovinstitute.org





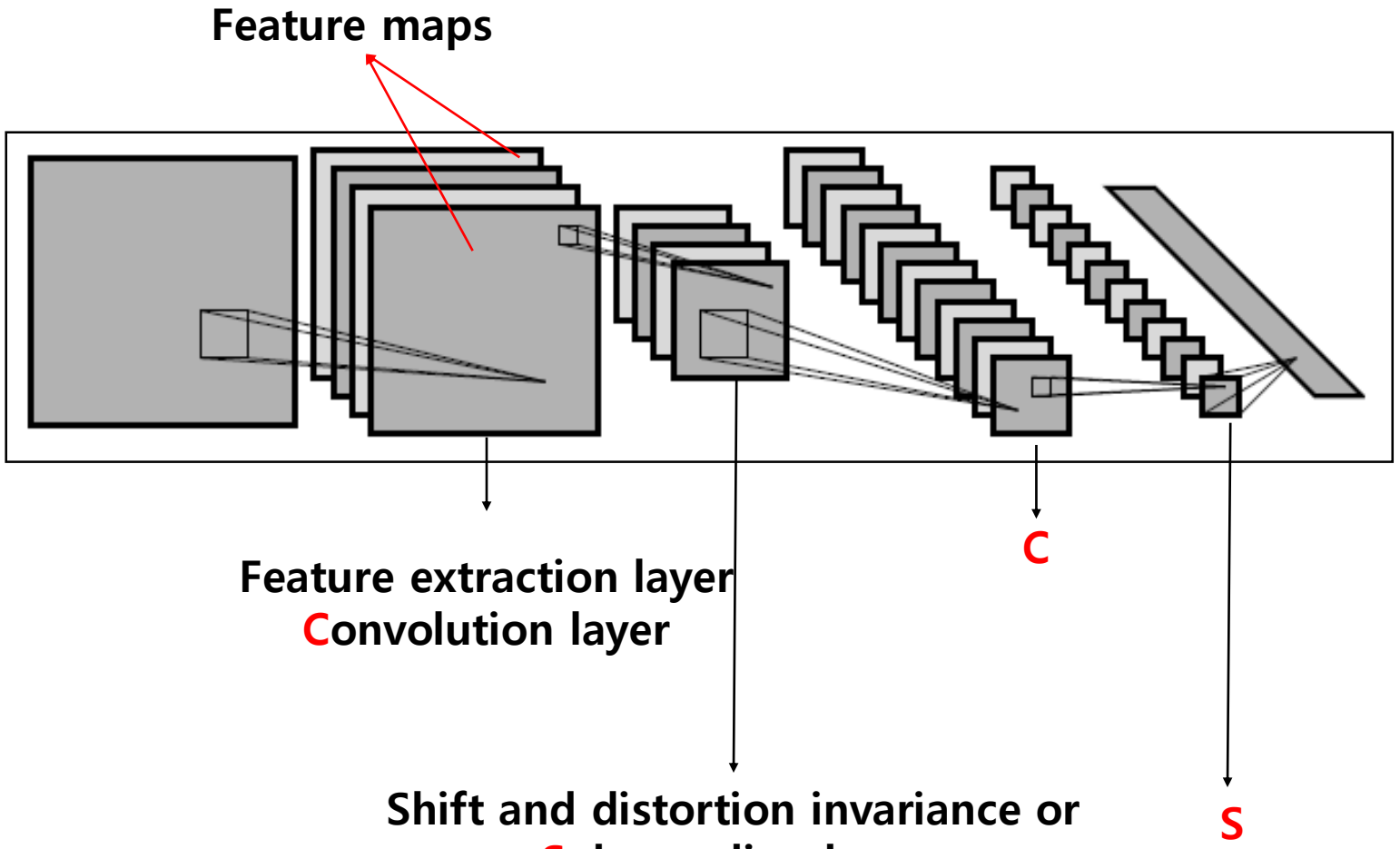


Deep Feed Forward Example

Example: CNN

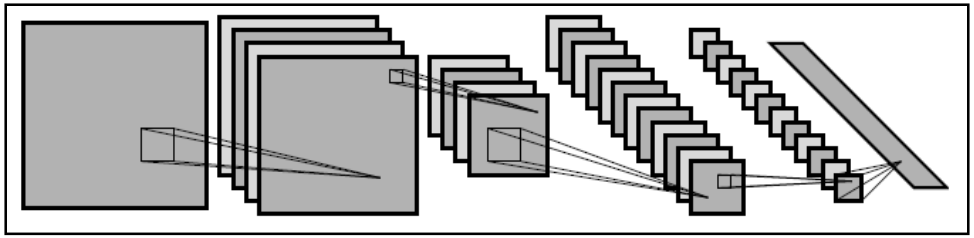
- Novel CNN architecture made huge impact
- Improved image recognition accuracy
- Nowadays almost all vision processing is done using deep learning

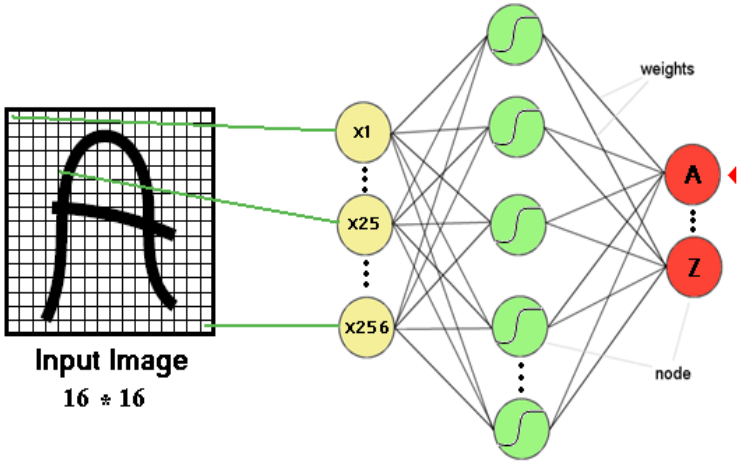
CNN's Topology

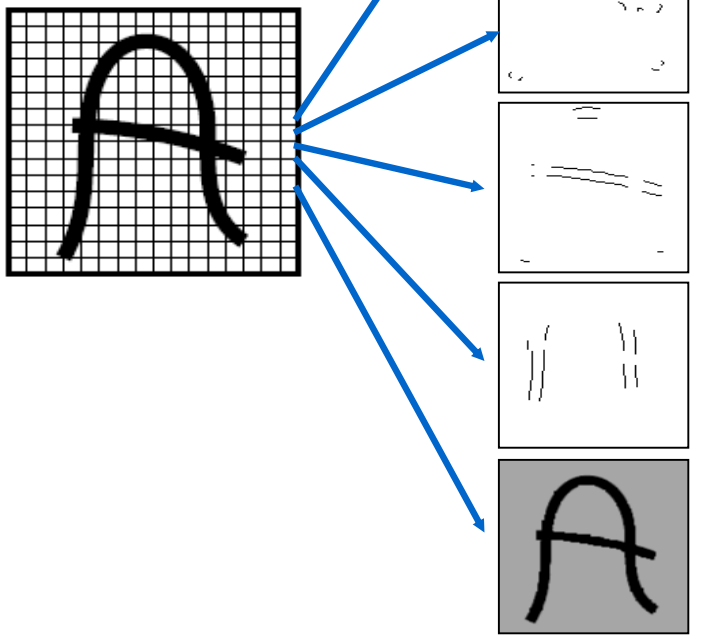


Feature extraction layer or Convolution layer

Detect the same feature at different positions in the input image.

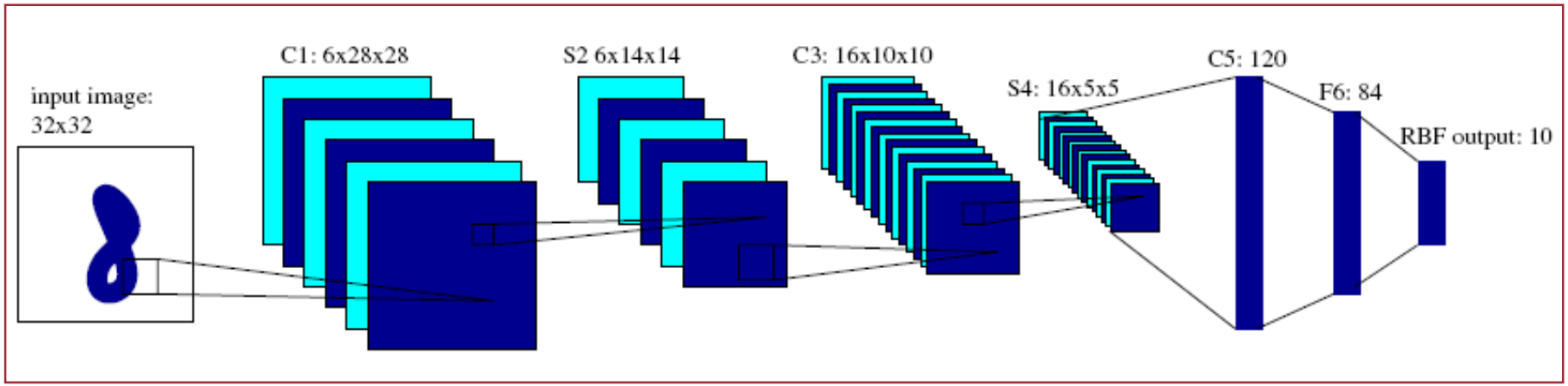






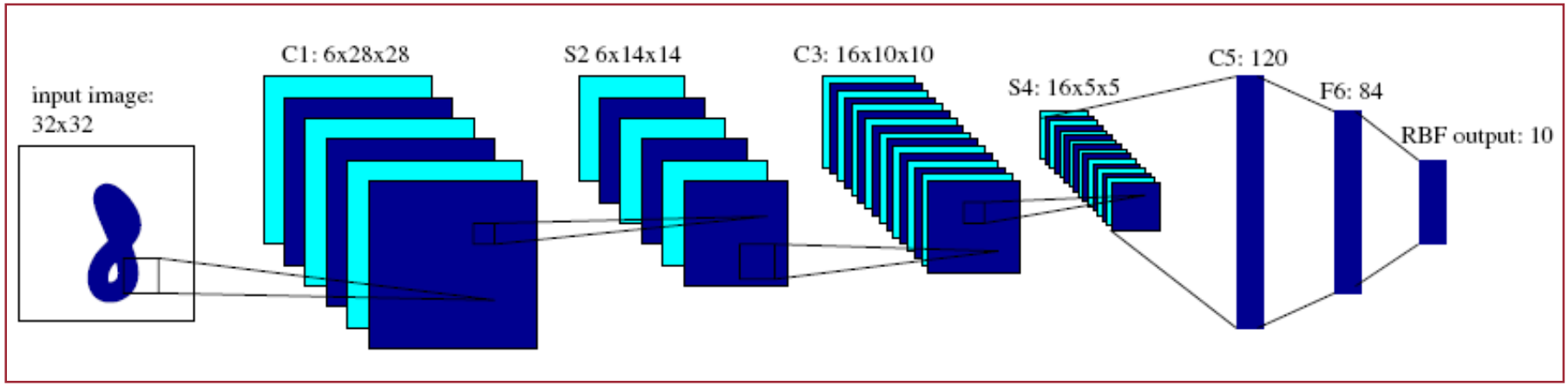
LeNet5

- Introduced by LeCun.
- raw image of 32 × 32 pixels as input.



LeNet5

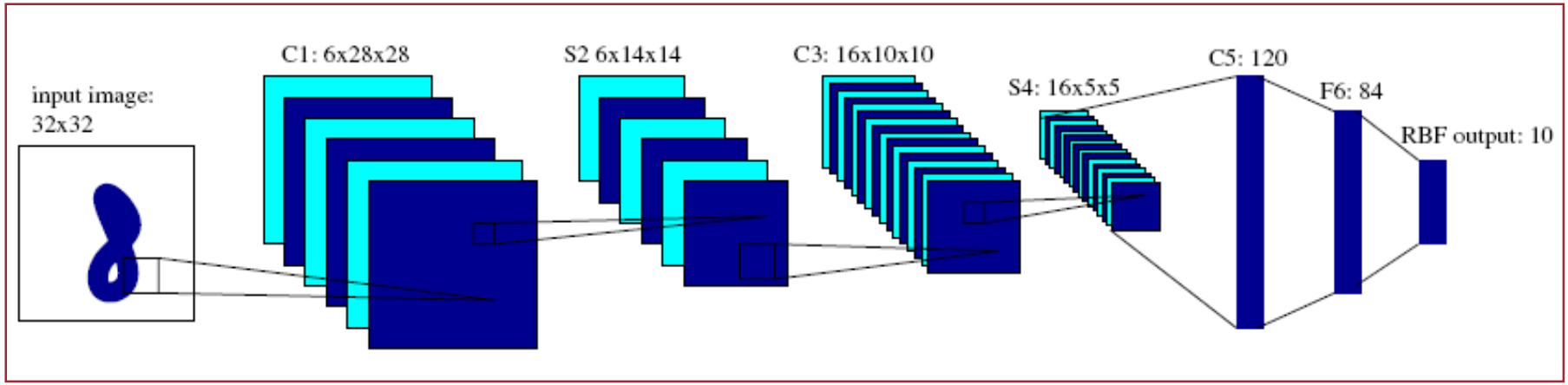
- C1,C3,C5 : Convolutional layer.
- 5 × 5 Convolution matrix.
- S2 , S4 : Subsampling layer.
- Subsampling by factor 2.
- F6 : Fully connected layer.



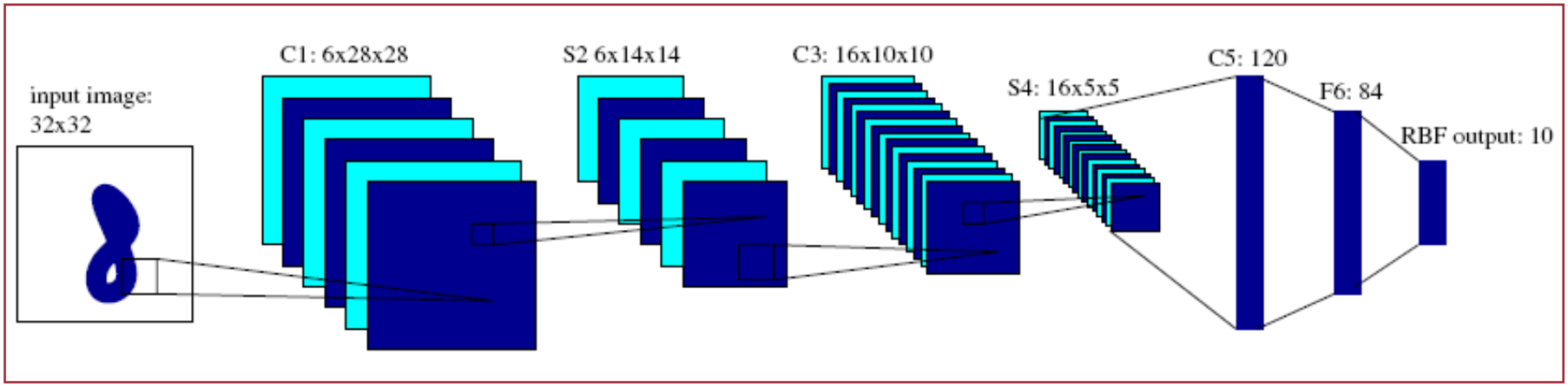
LeNet5

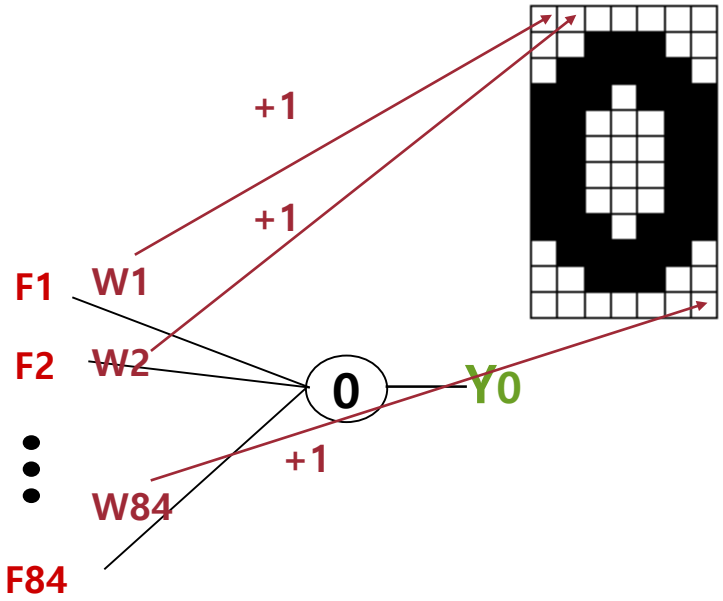
- All the units of the layers up to F6 have a sigmoidal activation function of the type:

$$y_j = \varphi(v_j) = A\tanh(Sv_j)$$



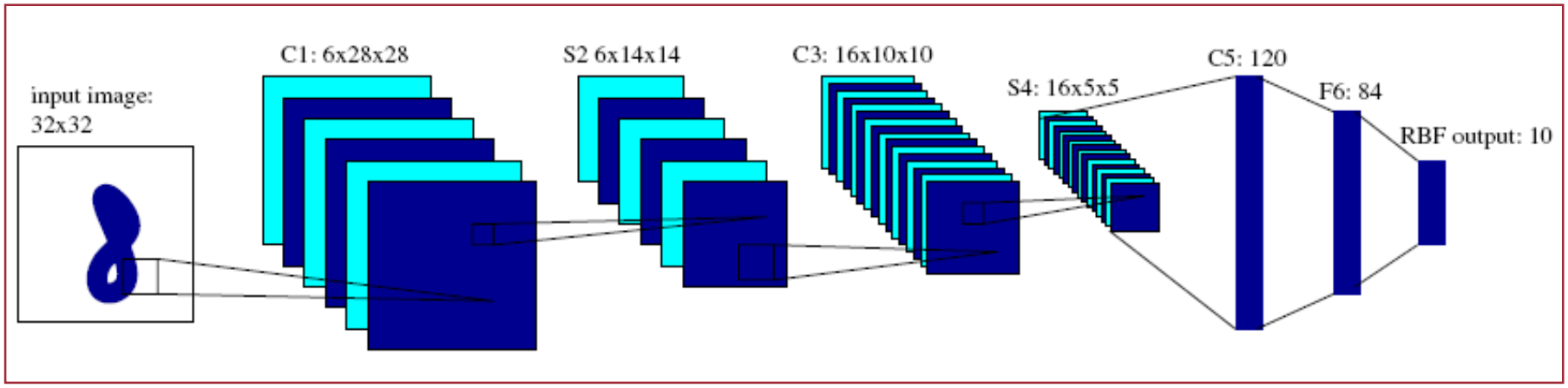
LeNet5



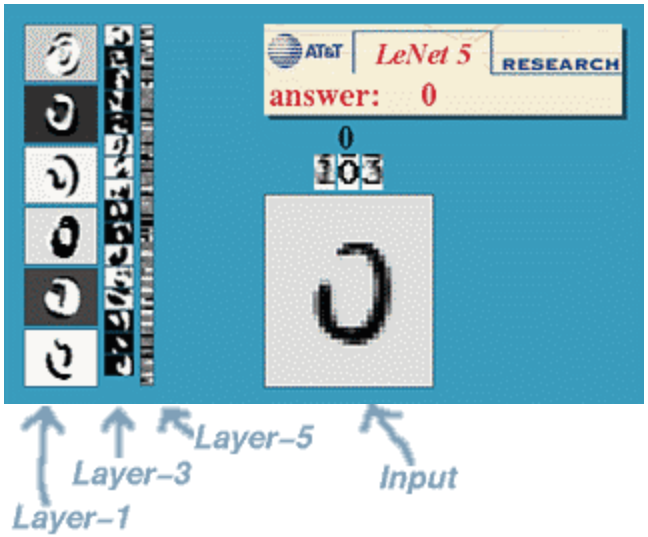


$$Y_j = \sum_{i=1}^{84} (F_i - W_{ij})^2, \; j = 0, \ldots, 9$$

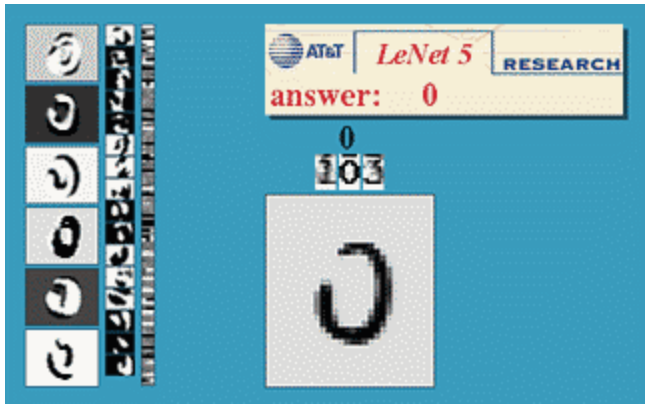
LeNet5



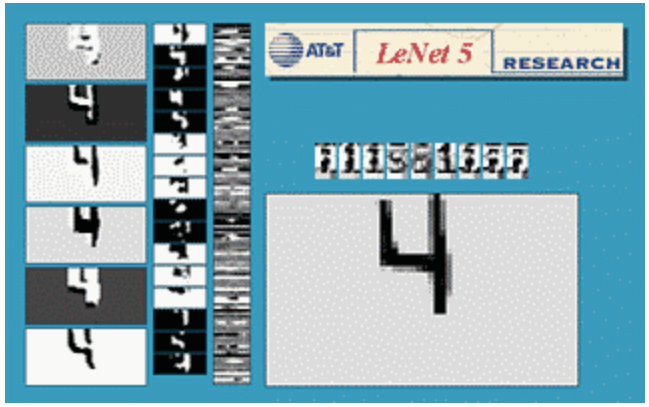
- About 187,000 connections
- About 14,000 trainable weights

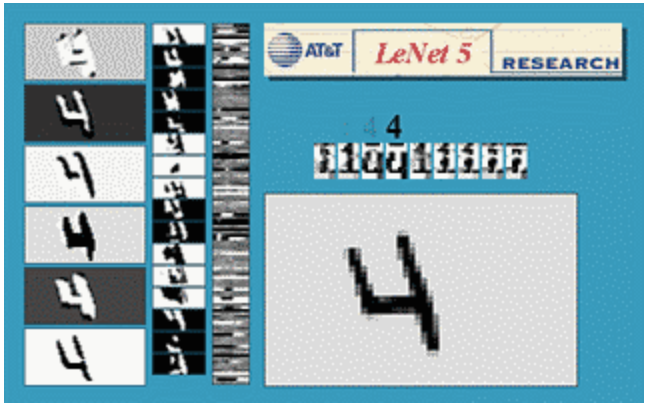


LeNet5

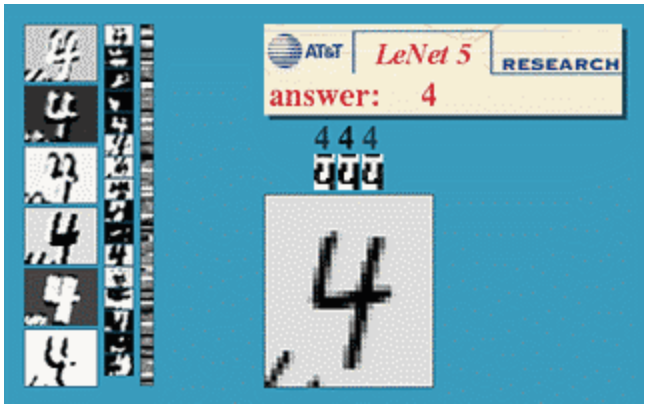

AT&T
LeNet 5
RESEARCH
answer: 0

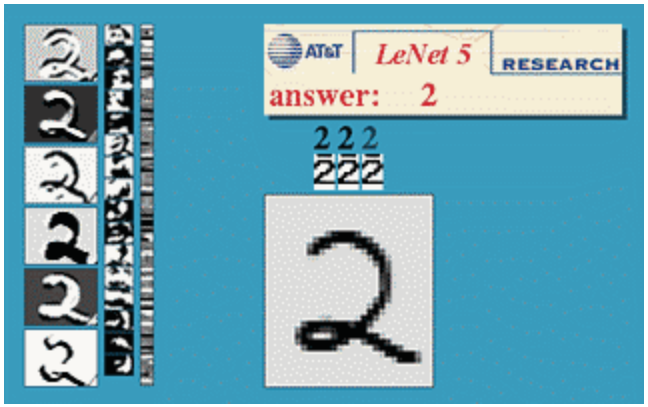

AT&T
LeNet 5
RESEARCH

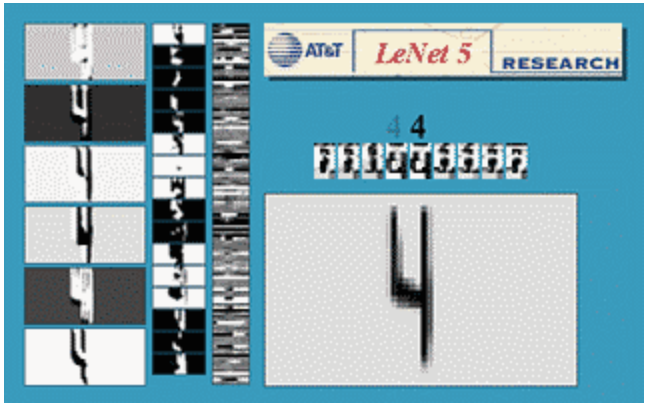

AT&T
LeNet 5
RESEARCH


AT&T
LeNet 5
RESEARCH

LeNet5


AT&T
LeNet 5
RESEARCH
answer: 4


AT&T
LeNet 5
RESEARCH
answer: 2


AT&T
LeNet 5
RESEARCH


AT&T
LeNet 5
RESEARCH
answer: 6

Overview

- Practical Issues in Network Initialisation: Overview
  - Architecture
  - **Activation functions**
  - Loss functions
  - Initial parameters
- Data Preprocessing
- Training

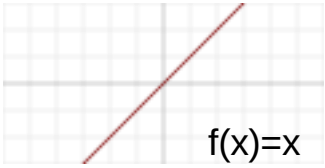
Activation functions

- Function types
  - Sigmoid
  - Softmax
  - Tanh
  - ReLU
  - LeakyReLU
- Properties of functions
- Implications for classification, training, etc.

Overview

- Practical Issues in Network Initialisation: Overview
  - Architecture
  - Activation functions
  - **Loss functions**
  - Initial parameters
- Data Preprocessing
- Training

Loss functions and output

	Classification	Regression
Training examples	R^n x {class_1, ..., class_n} (one-hot encoding)	R^n x R^m
Output Layer	Soft-max [map R^n to a probability distribution] $$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}}$$	Linear (Identity) or Sigmoid 
Cost (loss) function	Cross-entropy $$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)} \log \hat{y}_k^{(i)} + \left(1 - y_k^{(i)}\right) \log\left(1 - \hat{y}_k^{(i)}\right)\right]$$	Mean Squared Error $$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$ Mean Absolute Error $$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\lvert y^{(i)} - \hat{y}^{(i)}\right\rvert$$

Overview

- Practical Issues in Network Initialisation: Overview
  - Architecture
  - Activation functions
  - Loss functions
  - **Initial parameters**
- Data Preprocessing
- Training

Weight initialization

- Typically: initialize to random values sampled from zero-mean Gaussian: $w \sim \mathcal{N}(0, \sigma^2)$
  - Standard deviation matters!
  - Key idea: avoid reducing or amplifying the variance of layer responses, which would lead to vanishing or exploding gradients
- Common heuristics:
  - $\sigma = 1/\sqrt{n_{\text{in}}}$, where n_{in} is the number of inputs to a layer
  - $\sigma = 2/\sqrt{n_{\text{in}} + n_{\text{out}}}$ (Glorot and Bengio, 2010)
  - $\sigma = \sqrt{2/n_{\text{in}}}$ for ReLU (He et al., 2015)
- Initializing biases: just set them to 0

More details: http://cs231n.github.io/neural-networks-2/#init

Overview

- Practical Issues in Network Initialisation: Overview
  - Architecture
  - Activation functions
  - Loss functions
  - Initial parameters
- **Data Preprocessing**
- Training

Data PreProcessing

- May need to preprocess data
  - Insufficient data of particular types
    - Augmentation
  - Zero centering
  - Missing data
  - Etc.

Data augmentation

- Introduce transformations not adequately sampled in the training data
- Limited only by your imagination and time/memory constraints!
- Avoid introducing obvious artifacts

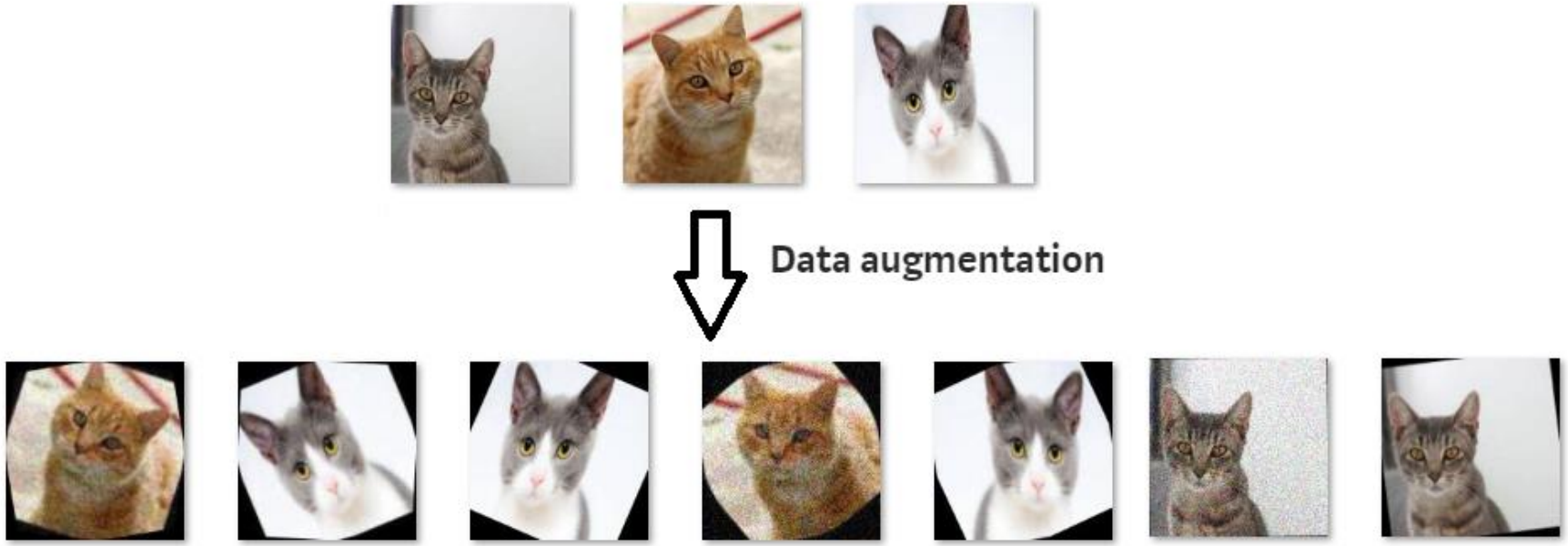


Image source

Overview

- Practical Issues in Network Initialisation: Overview
  - Architecture
  - Activation functions
  - Loss functions
  - Initial parameters
- Data Preprocessing
- **Training**

Training: Many Issues to Consider

- Training is non-linear optimisation

Training: optimize network parameters to minimise loss over training set

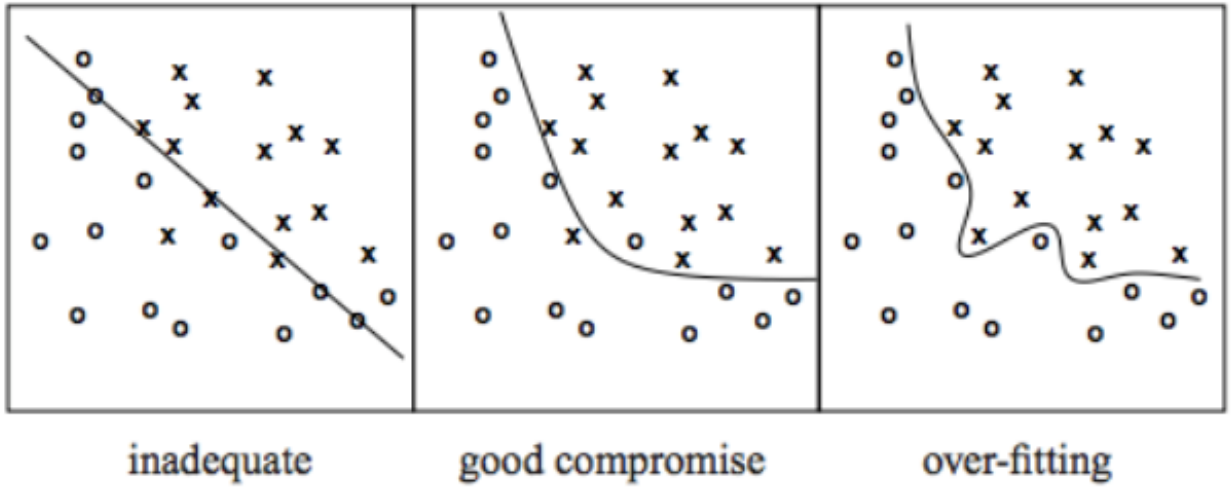
$$\theta^* = \arg\min_{\theta} \sum_{(x,y)\in(X,Y)} J[y, fL(x, \theta_{1,\ldots L})]$$

- Training is extremely complex process
- Tools exist to assist in training
- Hyper-parameter optimisation
  - Define hyper-parameters
  - Monitor and optimise over training process
- Overfitting

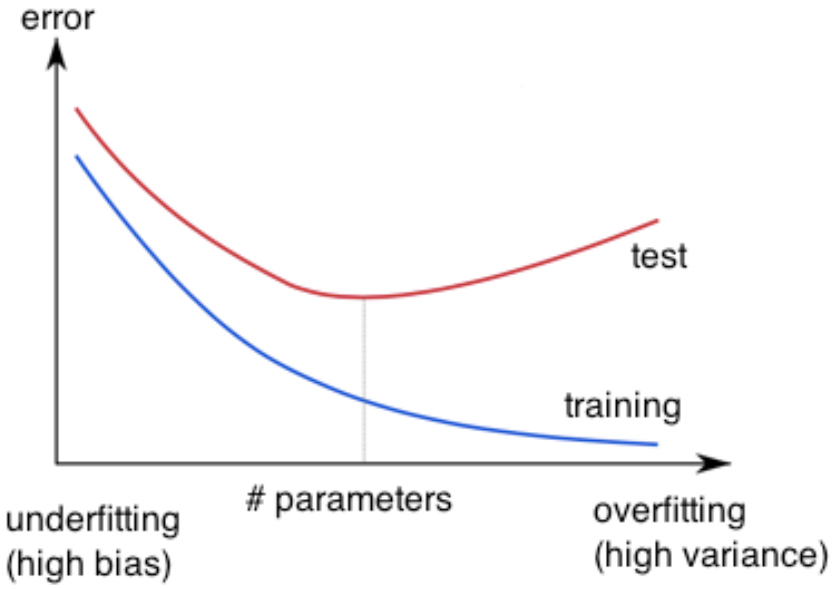
How to choose hyper-parameters?

- The learning rate η
- Mini-batch size m
- Early stopping
- Learning rate schedules
- Regularization parameter λ
- Grid search and the automated technique

Overfitting



http://wiki.bethanycrane.com/overfitting-of-data



https://www.neuraldesigner.com/images/learning/selection_error.svg

Learned hypothesis may **fit** the training data very well, even outliers (**noise**) but fail to **generalize** to new examples (test data)

Regularisation

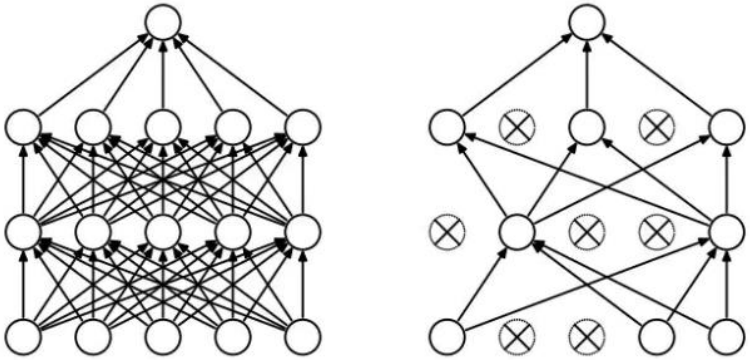
- Trading off performance vs. size/parameter-space of network
  - Bias/variance trade-off
- Regularisation parameter λ

$$J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$$

Regularization methods

- Address problem of overfitting
- Methods
  - Weight decay
    - L^2 normalization
    - L^1 normalization
  - Dropout
  - Artificial expansion of the training data

Regularization



Dropout

- Randomly drop units (along with their connections) during training
- Each unit retained with fixed probability p, independent of other units
- Hyper-parameter p to be chosen (tuned)

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research (2014)

L2 = weight decay

- Regularization term that penalizes big weights, added to the objective

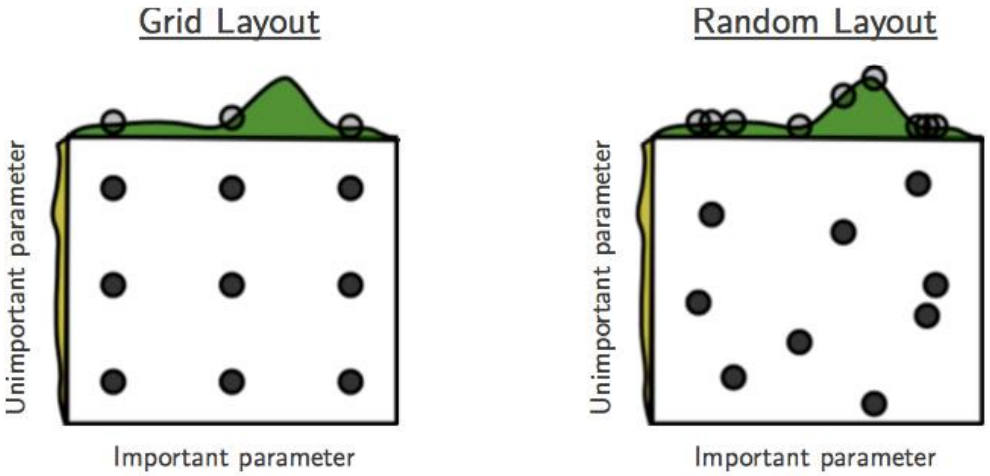
$$J_{reg}(\theta) = J(\theta) + \lambda \sum_{k} \theta_k^2$$

- Weight decay value determines how dominant regularization is during gradient computation
- Big weight decay coefficient → big penalty for big weights

Early-stopping

- Use validation error to decide when to stop training
- Stop when monitored quantity has not improved after n subsequent epochs
- n is called patience

Tuning hyper-parameters



$g(x) \approx g(x) + h(y)$

g(x) shown in green
h(y) is shown in yellow

Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research, Feb (2012)

"Grid and random search of 9 trials for optimizing function $g(x) \approx g(x) + h(y)$
With grid search, nine trials only test g(x) in three distinct places.
With random search, all nine trials explore distinct values of g. "

Both try configurations randomly and **blindly**
Next trial is independent to all the trials done before

Bayesian optimization for hyper-parameter tuning:

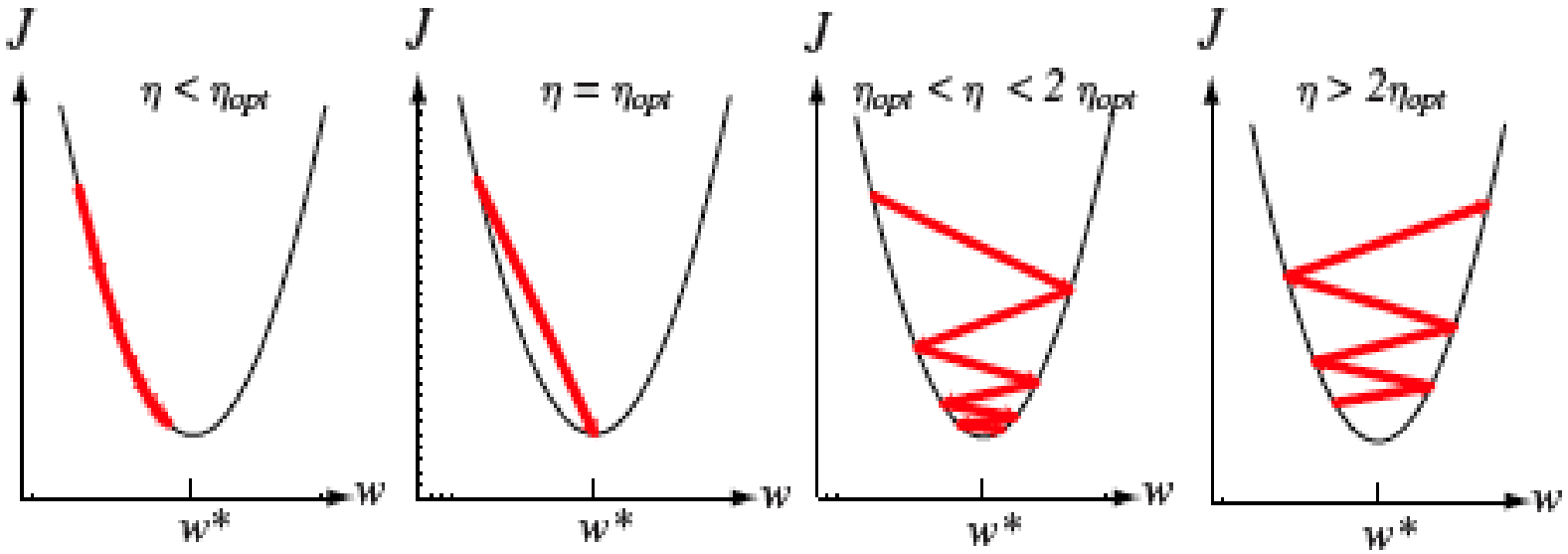
Library available!

Make smarter choice for the next trial, minimize the number of trials

1. Collect the performance at several configurations
2. Make inference and decide what configuration to try next

Learning rates

- Always use smaller rates



From [Duda&Hart:2001]

Summary

- Overview of Practical Issues in Deep Networks
- Key issues
  - Initialisation
  - Data Preprocessing
  - Training
- Will examine each major issue