IT/PC/B/T/411

Machine Learning

Deep Learning Basics

Lecture 10: Practical Methodology

Dr. Pawan Kumar Singh
Department of Information Technology
Jadavpur University
pawankrsingh.cse@gmail.com
+91-6291555693

Designing process

Practical methodology

- Important to know a variety of techniques and understand their pros and cons

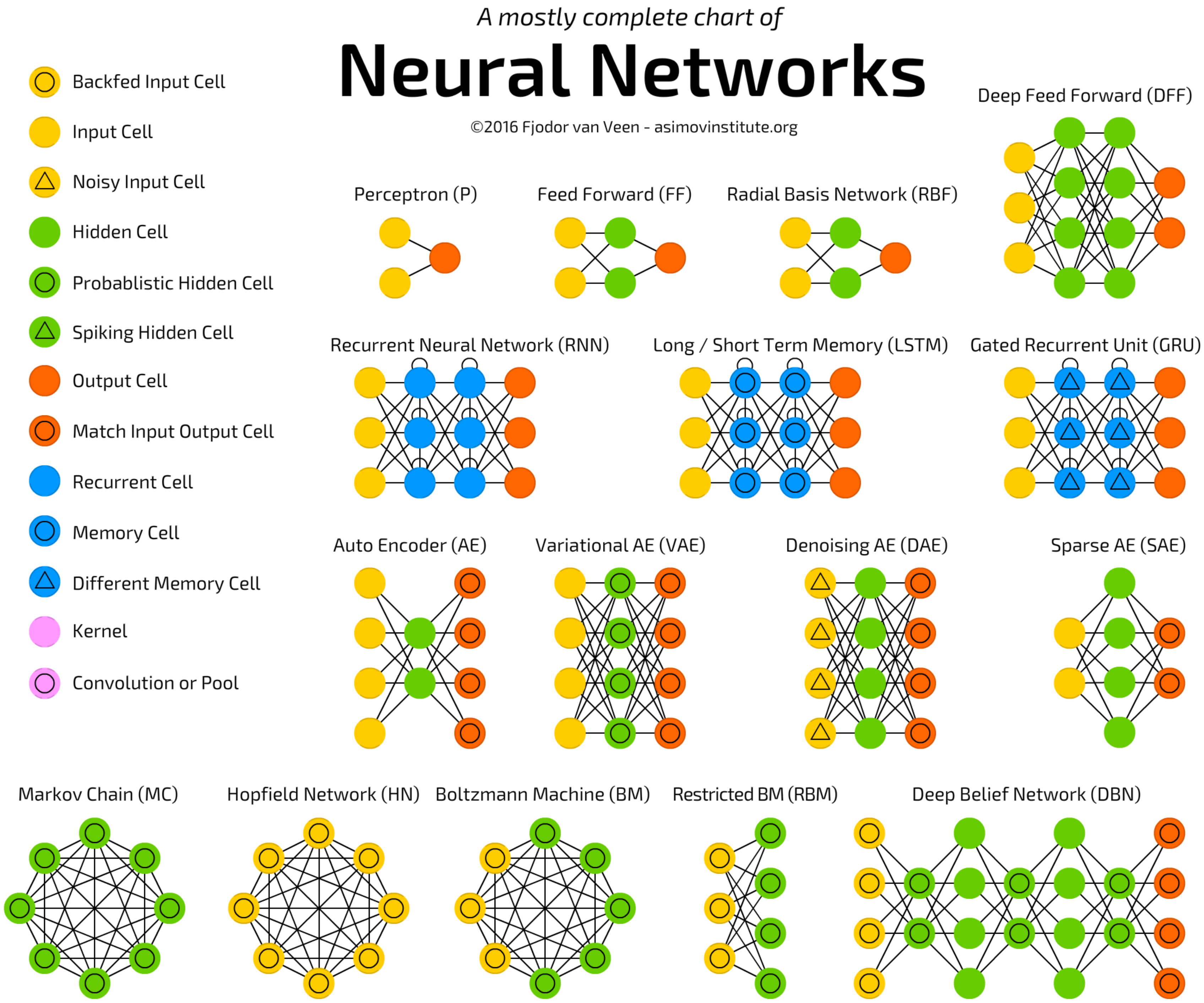
- In practice, “can do much better with a correct application of a commonplace algorithm than by sloppily applying an obscure algorithm”

Practical designing process

1. Determine your goals: input and output; evaluation metrics
2. Establish an end-to-end pipeline
3. Determine bottlenecks in performance
4. Repeatedly make incremental changes based on findings

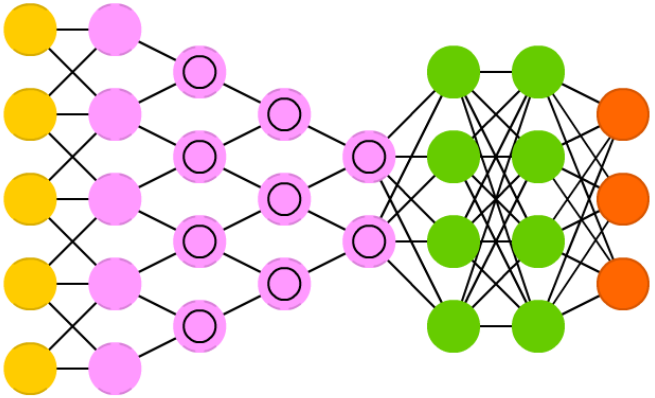
Practical designing process

1. Determine your goals: input and output; evaluation metrics
   - What is the input of the system?
   - What is the output of the system?
   - What can be regarded as a good system? Accuracy? Speed? Memory? …
2. Establish an end-to-end pipeline
3. Determine bottlenecks in performance
4. Repeatedly make incremental changes based on findings

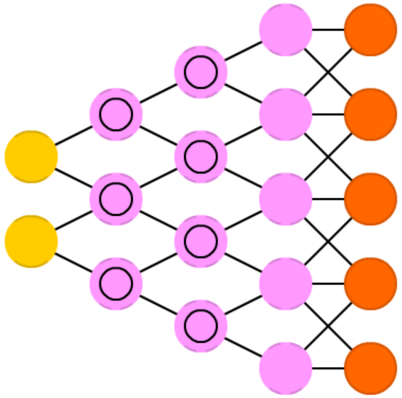


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)

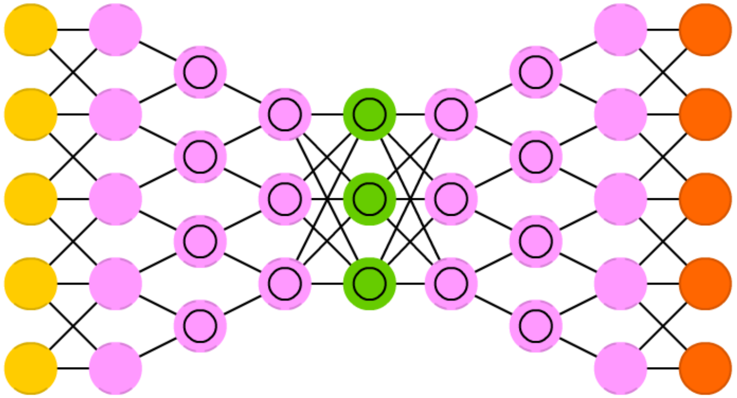
Deep Convolutional Network (DCN)



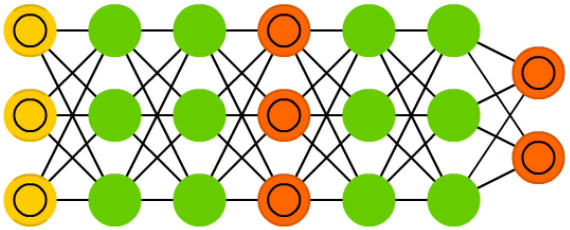
Deconvolutional Network (DN)



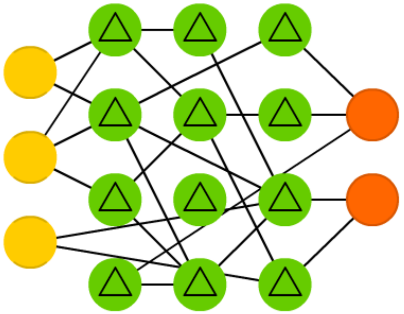
Deep Convolutional Inverse Graphics Network (DCIGN)



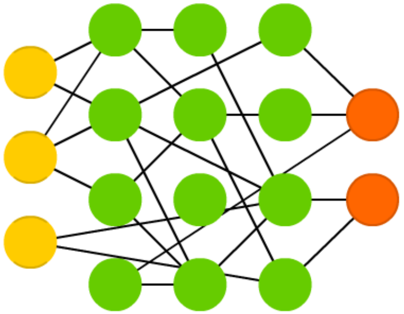
Generative Adversarial Network (GAN)



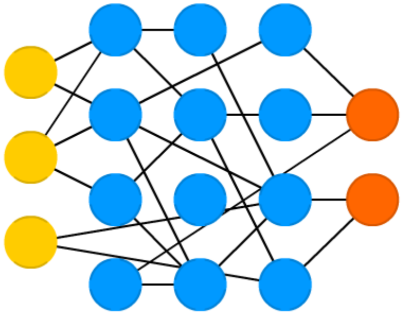
Liquid State Machine (LSM)



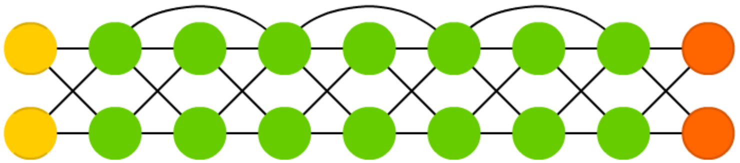
Extreme Learning Machine (ELM)



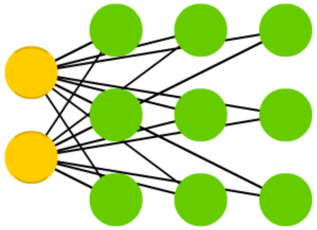
Echo State Network (ESN)



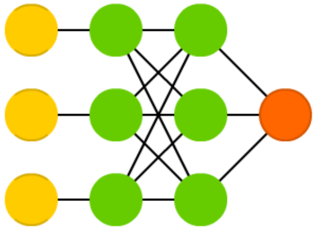
Deep Residual Network (DRN)



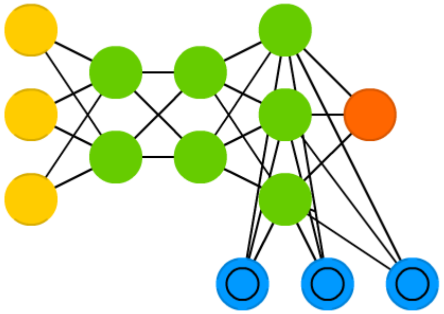
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)



Practical designing process

1. Determine your goals: input and output; evaluation metrics
2. Establish an end-to-end pipeline
    - Design the system as soon as possible, no need to be perfect
    - Can be based on existing systems for similar goals
3. Determine bottlenecks in performance
4. Repeatedly make incremental changes based on findings

Practical designing process

1. Determine your goals: input and output; evaluation metrics
2. Establish an end-to-end pipeline
3. Determine bottlenecks in performance …
    - Divide the system into components
    - Diagnose which component performing worse than expected
    - Overfitting? Underfitting? Bugs in the software? Bad/too small dataset?
4. Repeatedly make incremental changes based on findings

Practical designing process

1. Determine your goals: input and output; evaluation metrics
2. Establish an end-to-end pipeline
3. Determine bottlenecks in performance
4. Repeatedly make incremental changes based on findings
    - Do not make big changes (unless the system just too bad)
    - Replace system component? Change optimization algorithm? Adjust hyperparameters? Get more/new data?

To begin with

Deep learning?

- First question: do you really need deep learning systems?
- Maybe simple models like logistic regression/SVM suffice for your goals (i.e., shallow models)

- Choose deep learning if
  - The task fall into the areas that deep learning is known to perform well
  - The task is complicated enough that deep models have a better chance to win

Which networks to choose?

- Based on the input and the goal

- Vector input, supervised learning: feedforward networks
  - If know input topological structure, use convolution
  - Activation function: typically ReLU

Which networks to choose?

- Based on the input and the goal

- Vector input, unsupervised: generative model; autoencoder; energy based model
  - Highly depend on your goal

Which networks to choose?

- Based on the input and the goal

- Sequential input: Recurrent network
  - LSTM (long-short term memory network)
  - GRU (Gated Recurrent Unit)
  - Memory network
  - Attention-based variants

Which optimization algorithm?

- SGD with momentum and a decaying learning rate
- Momentum: 0.5 at the beginning and 0.9 at the end
- Learning rate decaying schemes
  - linearly until reaching a fixed minimum learning rate
  - decaying exponentially
  - decreasing the learning rate by a factor of 2-10 each time validation error plateaus

What regularizations?

- l_2 regularization
- Early stopping
- Dropout
- Batch Normalization: can replace dropout

- Data augmentation if the transformations known/easy to implement

Reusing models

- If your task is similar to another task studied: copy the model/optimization algorithm/hyperparameters, improve them

- Even can copy the trained models and then fine-tune it

Whether to use unsupervised pretraining?

- NLP: yes, use word embeddings almost all the time
- Computer vision: not quite; unsupervised now only good for semi-supervised learning (a few labeled data, a lot of unlabeled data)

Tuning hyperparameters

Why?

- Performance: training/test errors; reconstruction; generative ability…

- Resources: training time; test time; memory…

Two types of approaches

- Manually tune: need to understand the hyperparameters and their effects on the goals

- Automatically tune: need resources

Manually tune

- Need to know: the relationship between hyperparameters and training/test errors and computational resources (memory and runtime)

- Example: increase number of hidden units in each layer will
    - Increase the model capacity
    - Increase the generalization error (= test error – training error)
    - Increase memory and runtime

Automatically tune

- Grid search
- Random search
- Model-based optimization (another level of optimization)
  - Variables: hyperparameters
  - Objective: validation errors

Debugging strategies

Difficulties

- Do not know a prior what performance/behavior to expect
- Components of the model can adapt for each other
  - One components fails but the other components adapt to cover the failure

Debugging

- Try a small dataset
  - Faster, save time
- Inspect components
  - Monitor histograms of activations and gradients
  - Compare symbolic derivatives to numerical derivatives
- Compare training/validation/test errors
  - Overfitting or underfitting?
- Focus on worst mistake
  - On which data points it perform worst? Why?