DeepLearn2017

July 22, 2017

Contents

1 Algorithms for reasoning with probabilistic graphical models (Dechter, Ihler)

1.1 Introduction and inference

1.1.1 Basics of graphical models

Mechanism for describing large complex system $F(X)$ made of smaller, "local" interactions $f_\alpha(x_\alpha)$.

Examples:

- maximization (MAP)

- summation and marginalization
- mixed interence: planning, optimal decision-making, multi-agent problems

Consists of:

- set of variables X
- set of domains D
- set of functions or factors F
- combination operator defines an overall function from individual factors

X_i values are called *states*. A *configuration* is a set of states taken by variables. The *scope* of a factor f is its set of arguments.

For discrete variables, we can think of functions as tables.

1. Canonical forms

   Multiplication (of nonnegative factors) and summation are typically interchangeable by taking log or exp.

2. Graphical visualization

   (a) Primal graph

   - Variables -> nodes
   - Factors -> cliques

   (b) Example: map coloring

   Combination operator is "and", factors are individual constraints.

   Tasks: "max" = is there a solution? "sum" = how many solutions?

   (c) Example: Bayesian networks

   Overall function is product of conditional probabilities, factors are these conditions (?).

   Tasks: "max" what's the most probable state? "sum" = what's the probability of Y, given X? (belief)

   (d) Dual graph

   - Factor scopes -> nodes

   - Intersections among scopes -> edges

(e) Factor graph

   More explicit than the others

   - Variables -> circle nodes
   - Factors -> square nodes
   - Belonging to a scope -> edge between a square and a circle

(f) Exanple: Boltzmann machine

$$p(x) = \frac{1}{Z} e^{\sum_i a_i x_i + \sum_{i,j} w_{ij} x_i x_j}$$

3. Algorithms overview

   (a) Types of queries

   - max-inference
   - sum-inference
   - mixed-inference

   NP-hard!

   (b) Special case: trees

   If the final case is a tree: can be processed in linear time and memory.

   (c) Transforming into a tree

   i. By inference (thinking)
   Transform the model into a single, equivalent tree.
   Treewidth = *max cluster size* - 1.

   ii. By conditionaing (guessing)
   Transform it into many tree-like sub-problems.
   By assigning (guessing) certain variables, the graph structure becomes simpler as cycles are removed. So, search through the tree of possible assignments.
   This is as "slow" as inference, but more compact in space.

   iii. Mixed approach
   Search + inference: guess some variables, apply inference on the remaining.

   (d) Conditioning and elimination operators

i. Conditioning on observations
   Observing a variable's value reduces the scope of the factor (in the sense that the variable has a fixed value, so the factor is restricted).
   Creates several sparser graphs.
ii. Conditional independence
   Is X conditionally independent of Y given Z? If, when Z is removed from the graph, there is no path from X to Y, then they're independent.
iii. Combination of factors
iv. Elimination in a factor
   Creates a denser graph.

1.1.2 Inference algorithms, exact

1. Bucket elimination for trees Perform computation step by step via distributive rule over the function.

2. Bucket elimination

   Group factors in buckets in a certain order, each corresponding to a variable. Eliminate variables one at a time.

   Complexity: exponential over induced width -> we want to find the smallest induced width.

   Non commutative operations restrict the orders that can be chosen.

3. Jointree clustering

   Bucket-tree elimination: allows messages both ways through the bucket elimination process in an efficient manner.

   Build *jointree*: merge non-maximal buckets (in the dual graph) into maximal clusters. Further merging allows to trade memory for time.

   (a) Tree decomposition
   (b) Message passing on the tree decomposition

4. Elimination orders

   Finding an order that induces the smallest width is NP-complete. There are several greedy algorithms.

   (a) Greedy orderings heuristics

    - min-induced-width
    - min-fill (most popular)
    - max-cardinality search

  (b) Chordal graphs A graph is chordal if every cycle of length at least 4 has at least one chord (a diagonal).

  (c) Anytime algorithms

1.1.3 Approximate elimination

1. Decomposition bounds

   Upper and lower bounds via approximate problem decomposition (not requiring each occurrence of a variable to have the same value).

   Reparametrization (cost shifting) can allow to tighten the bounds.

2. Mini-bucket and weighted mini-bucket

   When a bucket is too large, split it and perform separate calculations, approximating/bounding the joint result.

   Decompositions:

   - $\max(f_1 + f_2) \leq \max f_1 + \max f_2$
   - $\sum f_1 f_2 \leq (\sum f_1^{\frac{1}{w}1})^{w_1}(\sum f_2^{\frac{1}{w}2})^{w_2}$ (Holder's inequality)

3. Belief propagation

   Apply two pass algorithm (for trees) locally on the graph.

1.2 Search

1.2.1 And/Or search spaces

1. Conditioning: the probability tree Exponential time but **linear space**.

2. AND/OR search space Decompose the problem in an and/or tree built from the pseudotree (e.g. depth first search tree) formed by the graph.

   This is much cheaper than the OR tree. However, a path to a leaf is not a configuration: instead, for each OR node it includes one of the children, and for each AND node it includes both of them.

   (a) And/Or counting value
   Allows to count the number of valid solutions (e.g. for constraint satisfaction problems) under each node.

    (b) Pseudotree

    Pseudotree: a tree spanning its nodes, where all arcs in the graph not in the tree are back arcs (they connect nodes to their ancestors). A chain, by definition, is always a pseudotree.

3. And/Or search graphs

   If two subtrees are identical, we would like to merge them and solve them only once.

   No longer linear memory (caches are needed to recover previously solved subtrees), but search space becomes smaller.

   (a) Merging based on context

   context = ancestors of X in pseudotree that are connected either to X or to descendants of X. Theorem: max #context = induced width.

   Any query is best computed over the context-minimal And/Or graph.

   (b) Variable elimination over the And/Or graph

   Variable elimination can be done bottom-to-top through the AO graph, processing "chunks" of nodes (from the same variable at a time).

4. Building good psudotrees

   The bucket tree previously discussed is actually a pseudotree.

   Finding small height/width pseudotrees is NP-hard.

   Optimality of induced width and pseudotree height cannot be achieved at once.

   (a) Min-Fill

   (b) Hypergraph partitioning

5. Brute-force And/Or

1.2.2 Heuristic search for And/Or spaces

1. Basic Heuristic search

2. Depth-first AO

   AOBB: And/Or branch & bound.

   Breadth-rotating AOBB: takes turns processing sub-problem.

3. Best-first AO

   AOBF

1.3 Approximate inference

1.3.1 Introduction

1. Example: DBMs

   784 px ↔ 500 mid ↔ 500 high ↔ 2000 top ↔ 10 labels

   Induce width: ~2000 :(⇒ can't use exact inference

2. Algorithms

   We want anytime algorithms: very fast and approximate, or slower and more accurate.

1.3.2 Variational methods

1. Vector space representation

   - concatenate the tables of the factors into a vector. Also include the possible values of the variables into a similar vector, so that evaluating factors becomes a dot product.

2. Inference tasks

   - distribution is an exponential family
   - tasks os interest are convex functions of the model

3. Tree reweighted MAP (TRW MAP)

   Let T_1, T_2 be two tree-structured models. By convexity,

   - $\max_x \theta x \leq w_1 \max_x \theta^{(1)} x + w_2 \max_x \theta^{(2)} x$
     Later we can try to minimize this bound.

   TRW MAP is equivalent to MAP decomposition.

4. Tree reweighted sum

   Same principle as TRW.

5. Negative TRW

   Extrapolation gives lower bounds.

6. Variational perspectives

   (a) Mean field

1.3.3 Monte-carlo sampling

1. Monte Carlo estimator

   - Basic form: empirical estimate of probability.
   - For this, we need to be able to sample from the target distribution or, at least, evaluate p(x) explicitly or up to a constant.
   - Good anytime properties: use time to geneerate more samples and improve the approximation.
   - Central limit theorem kicks in very fast
   - Almost all the mass is around the average, so this can give us finite sample confidence intervals.

2. Sampling in Bayes nets

   (a) No evidence The structure shows exactly how to sample from the distributions: start from root(s), sample downward.

   (b) With evidence

   Relative error bounds are better than finite error bounds in case of small probabilities.

   When estimating posteriors, rejection sampling and 'estimate the ratio' don't work well with small probabilities.

   (c) Exact sampling via imference

   Draw samples from $P[A \mid E = e]$ directly? Build oriented tree decomposition and sample. This process is slow (exponential), but sampling is fast.

3. Importance sampling

   Choose $q(x)$ easy to sample from.

$$\int p(x)u(x) = \int q(x)\frac{p(x)}{q(x)}u(x) \approx \frac{1}{m}\sum_i \frac{p(\tilde{x}^{(i)})}{q(\tilde{x}^{(i)})}u(\tilde{x}^{(i)}), \tilde{x}^{(i)} \sim q(x)$$

   IS is unbiased or at least asymotically unbiased.

   Can give poor performance:

   - if $q(x) << u(x)p(x)$: rare (unlikely) but very high weights
   - To have guarantees, analitically bound variance

   WMB-IS gives and improves bounds as samples are drawn. There are other choices of $[q(x)]$ proposals: based on belief propagation, adaptive importance sampling...

4. Markov Chain Monte Carlo

   (a) Markov Chain

   Simple temporal model where the state at time t only depends on state at t-1. It's homogeneous in the sense that $p(X_t \mid X_{t-1})$ does not depend on t. Examples:

   - random walks
   - finite state machine

   When (if) a Markov chain gets to a stable situation, that's called a *stationary distribution.* This stationary distribution may not exist (e.g. a deterministic cycle of states doesn't stabilise). Sufficient conditions:

   - $p(. \mid .)$ is acyclic
   - $p(. \mid .)$ is irreducible

   (b) Markov Chain Monte Carlo

   Create a Markov chain where each state is a complete configuration of our distribution. As the chain goes forward, if the initial states /samples were carefully chosen, the stationary distribution will be our $p(x)$.

   i. Metropolis-Hastings sampling
   Pick function q to get next step.

        - At each step, propose new value $x' \sim q(x' \mid x)$
        - Decide wether we should move there according to p.

    ii. Gibbs sampling Proceed in rounds: sample each variable in turn given all others' most recent values. Conditional distributions depend only on the Markov blanket.
    It's easy to see that $p(x)$ is stationary.
    Some advantages over Metropolis: no rejections (although there's the possibility of staying in place), no free parameters; **but** moves are local.

    iii. Example: DBMs
    MCMC is probably the most popular algorithm to train RBMs/DBMs. Used in both model training (contrastive divergence, persistence CD. . .) and model validation (annealed & reverse annealed importance sampling. . .).

    iv. MCMC and common queries
    Estimating expectations is easy!

    v. . . .
    Samples from $p(x)$ asymptotically (in time), but they're not independent.
    Rate of convergence depends on proposal distribution for MH and variable dependence for Gibbs. Mixing rate is difficult to measure though.

(c) Inference within samples

2 Deep learning for speech recognition (Deng)

2.1 Rescuing from gradient vanishing

- Pre-training DBN
- Discriminative x? -> pre-training
- random weights controlling variance
- ReLU

2.1.1 LSTM (long short-term memory)

For RNNs?

Simplification: Gated recurrent unit (GRU)

2.2 Speech

2.2.1 Deep Generative models

Difficulties:

- inference (is NP-hard; approximations via variational techniques)
- explaining away (?)

A stack of RBMs is not a DBM but a DBN.
Decoding from DBN is simple, training is slooow.

3 Deep Generative Models and Unsupervised Learning (Wu)

3.1 Overview

3.1.1 Modes of learning

Generative models, density estimation: approximate a probability distribution from observed data

3.1.2 Deep Learning

Using CNNs we can build generative models: top-down deconvolutional image synthesis.

3.1.3 Latent variable model

A normal distribution for the latent (hidden) variables h is assumed.

Interpolation in the latent space can be done: infer the latent variables for several images and interpolate in h to then rebuild images through the top-down synthesis.

3.1.4 Energy based model

$$p(X; W) = \frac{1}{Z(W)} \exp(f(X; W)) q(X)$$

The parameters W are learned during the bottom-up convolutional feature extraction, looking to minimize the energy function. This model cannot be sampled directly though.

The sampling can be approximated (?) using MCMC. This is called a *descriptive net* (since it's sampled as in descriptive statistics).

3.1.5 Dual networks

Bottom-up net:

- variational bayes autoencoder: inference net
- generative adversarial training: discriminator net
- cooperative learning: descriptor net

Top-down net: generator net

3.2 Latent Variable Models

The aim is to obtain a vector of hidden variables that explain the inputs.

3.2.1 Linear latent variable models

Top-down from hidden variables: $X_i = Wh_i + \varepsilon_i, i = 1, \ldots, n$.

1. Factor analysis

   Zero-mean normal distributions are assumed for h_i and ε_i.

   Example: decathlon $p = 10$, $h_i =$ (strength, speed, endurance), $d = 3$.

   We can generalize the model by assuming other distributions, and non-linear mappings.

2. Independent component analysis

   $d = p \Rightarrow W$ is square.

   $h_{ik} \sim p_k$ independently. $X_i = Wh_i, h_i = AX_i, p(X) = p(h)|\det(A)|$.

3. Sparse coding

   $d > p$ the number of latent factors is larger than the visible factors. However, h_i is sparse vector.

4. Non-negative matrix factorization

   h_i is positive vector. $X_i = Wh_i + \varepsilon_i$.

5. Recommender system

   User i's rating of item j, $x_{ij} = \langle w_j, h_i \rangle + \varepsilon_{ij}$. The latent features refer to the user's desires, and the visible ones to their desirabilities.

6. Restricted Boltzmann machine

$$(h_i, X_i) \sim p(h, X \mid W) = \frac{1}{Z(W)} \exp(X^T W h)$$

   Explicit inference distributions:

   - $p(h \mid X, W) : h_k \sim \text{logistic}(\sum_j w_{j,k} x_j)$
   - $p(X \mid h, W) : X = Wh + \varepsilon, \varepsilon \sim \mathcal{N}(0, \sigma^2 I_p)$

7. Autoencoder

   - Encoding: \$h$_\mathrm{k}$=sigmoid($\sum_\mathrm{j}$ w$_\mathrm{j,k}$x$_\mathrm{j}$)
   - Decoding: $X = Wh + \varepsilon$

8. Stacked RBM or AE

   Trat h as new input, learn layer-wise.

3.2.2 Generator network (CNN in the top-down process)

Powerful non-linear approximation.

Learning generator network in an unsupervised way: assume $h \sim \mathcal{N}(0, I_d)$, let $X = g(h; W) + \varepsilon$ and for each layer l, $h^{(l-1)} = g_l(W_l h^(l) + b_l)$ where $h^{(L)} = X$, $h^{(0)} = h(?)$.

1. Alternating back-propagation

   Loss function
$$L = \sum_i \|X_i - g(h_i; W)\|^2$$

   - Inference: $h_i \leftarrow h_i + \gamma \frac{\partial L_i}{\partial h_i}$
   - Learning: $W \leftarrow W + \gamma \frac{\partial L}{\partial W}$

   Most of the computation is shared among inference and learning, so the unsupervised part (inference) comes almost for free.

   Langevin dynamics = Gradient descen + adding some noise.

   While learning, inference should be not as good, so we add noise so that we force the network to learn.

3.2.3 Learning and inference

1. Unsupervised learning

   $$\text{Likelihood}(\theta) = \Pi_i p(X - i; \theta)$$

   Maximum likelihood: $\hat{\theta} = \text{argmax}_\theta L(\theta)$

   - most plausible explanation of data
   - most accurate unbiased estimator

   Kullback-Leibler divergence view: $\hat{\theta} = \text{argmin}_\theta KL(P_{\text{data}} \mid p_\theta)$. We can think of this as a projection of P_{data} onto $\{p_\theta, \forall \theta\}$.

2. Max likelihood with latent variables

   Something about multiple guesses

3. Learned inference

   Instead of learning a function, learn a posterior distribution. Then, the decoding direction will be easy to compute:

   - decoding: $X_i = g(h_i; W_{down}) + \varepsilon_i$
   - encoding: $h_i \sim N(f_\mu(X_i; W_{up}), f_{\sigma^2}(X_i; W_{up}))$

   Wake-sleep algorithm.

   - Sleep: $W_{down} \rightarrow$ dream data. $(h_i, X_i) \rightarrow W_{up}$
   - Wake: real $X_i \rightarrow h_i$ by W_{up}. $(h_i, X_i) \rightarrow W_{down}$

4. Variational Bayes

   - Variational autoencoder

3.3 Dual Nets

3.3.1 Introduction

- Bottom-up convnet: energy ← signal (descriptor net)
- Top-down convnet: latent variables → signal (generator net)

In the bottom up direction, we compute an energy function optimize so that its values are low. Based on statistical mechanics, where low energy states are more likely. So if we sample from this distribution, most of the samples have a low energy state.

3.3.2 Energy-based model

1. Descriptor net

   $$p(X;\theta) = \frac{1}{Z(\theta)} \exp(f(X;\theta))p_0(X),$$

   where $p_0(X)$ is the reference distribution (e.g. gaussian white noise) and Z is the normalizing constant.

   The associated energy function is E, so that $p(X;\theta) = 1/Z(\theta)\exp(-E(X))$. This way, if the energy of the *state* (X) is low, its probability is very high.

   How do we estimate θ from observed images? We can use max likelihood learning. The log-likelihood is $L_p(\theta) = \frac{1}{n}\sum_i \log p(X_i;\theta)$. We can also minimize the Kullback-Leibler divergence.

   If we take the derivative of the log likelihood, we can use gradient descent to update our parameters. But it includes an expectation, which can be hard to evaluate. To solve that, we choose to sample from the distribution (Langevin revision, on the style of gradient descent). The samples drawn should be of low energy, so we then *shift* our density function (density shifting) so that it sits closer to the low energy regions, thus updating our parameters.

2. ConvNet

   For the energy minimization, we need the same kind of derivatives than when we sample the distribution, they can be computed via backpropagation and share some common terms.

   We can say that the sampling of synthesized examples is a *dreaming* phase, and the updates of parameters allow to make the dreams more realistic.

   Note: dreams are not reconstructions of observed images, although they maintain similar appearance to them.

3. Multigrid scheme

   Sampling images is costly (even with contrastive divergence) but we can use several stages (4x4 -> 16x16 -> 64x64).

4. Recruit a sampler

   The bottom-up net for the energy-based model can recruit a generator network as a sampler

3.3.3 Latent variable model

Can generate samples using MCMC.

1. Alternating backprop/graddesc

   This time we use loss $L = \sum_i \|X_i - g(h_i;\alpha)\|^2$.

   For inference, we update h_i, whereas for learning we update the parameters α.

   The inference step is rather costly. Graddesc is used but we can also use Langevin dynamics.

   (a) thinking, explaining away reasoning (from the observed images).

   (b) make the thinking more accurate.

3.3.4 Cooperative learning

Both models involve gradient descent in a difficult phase. However, the simple/direct phases of each one are complementary.

We can see the descriptor net as a teacher, and the generator net as a student:

1. student generates "draft"
2. teacher runs gradient descent on the energy perspective and revises the draft
3. student learns from draft and reconstruct revised draft (because ir knows the latent factors)
4. teacher learns from "outside review", student's samples (shifts from initial towards revised)

This way the inference step in the generator is not needed, and we get a sampler for the descriptor net.

Inference is inconvinient because there are no guarantees that the model infered is accurate or realistic.

1. MCMC teaching

   Using gradient descent, from an information theoretic perspective, the generator net is trained to "move" closer to the descriptor.

3.3.5 Related models

1. Heimholtz machine

   In the previous generator network: only first layer is stochastic, rest of layers are deterministic.

   In a Heimholtz machine, each layer is binary and stochastic: latent layer is bernoulli(p), rest of layers are logistic regression-like, bernoulli(sigmoid(linear comb)). It is trained by wake-sleep.

2. Deep Boltzmann machine

   The energy function involves many layers. Also, latent variables are binary. Its training is expensive.

   This model is related to the DBN (RBM + sigmoid).

3. Generative Adversarial Net

   Instead of energy-based Descriptor, we have a Discriminator and Generator. They play a minimax game where the generator tries to deceive the Discriminator and the Discriminator learns not to be fooled with its examples.

4. Introspective generative modeling

   Progressively learning by repeated discriminations. Do classification, improve distribution via SGD and repeat.

   Along the way, it trains a classifier that works better than just learning from the observed data.

5. Auto-regressive models, PixelRNN

6. ICA generalization

   $d = p$, use $X_i = g(h_i; W)$ and $h_i = g^{-1}(X_i; W)$ with some restrictions so that weights can be updated. Apply auto-regressive structure on h.

   Example: Real NVP.

7. Activation maximization

   Example: Plug and Play Network, uses denoising autoencoder to sample h from an implicit $p(h)$ [Nguyen et al 2017]

8. Diffusion model

4 Deep Learning at NVIDIA (Breuel)

Site: `http://9x9.com` (slides, reading lists, etc.)

- Berger's Statistical Decision Theory and Bayesian Analysis

4.1 Different views of Deep Learning

4.1.1 NVidia stuff/promo

1. Tesla V100

   Includes own tensor core: 120 Tensor TFLOPS

   DGX-1 = 400 servers in a box

2. Embedded AI processor for autonomous machines

3. Tensorrt

   Compiler for Tensorflow

4. Nvidia drive

   Working with Toyota

5. Project Holodeck

6. Isaac robot simulator

7. NVidia mainly sells hardware

   Easy collaboration with academic groups and other companies, standard research environment, etc.

4.1.2 Deep learning view

1. Qs

   - What is L1 loss and when would you use it?
   - batch normalization?
   - architecture of a GAN?
   - autoencoding for pre-training?

2. Primary DL frameworks

   - PyTorch (!)

- automatic, dynamic differentiation when desired

- Tensorflow

(a) Pipe notation tfspecs/dlpipes

Shorthand for some models, e.g. `lecun89 = Cs(12, 5)**2 | Flat() | Fs(30) | Fs(10)`

Similar to `%>%` in R?

3. Distributed training

   Hundreds of servers to store data ⇒ hundreds of nodes to lightweight preprocessing and shuffle the data ⇒ 8x 8-GPU nodes to compute stuff.

   Stochastic Gradient Descent is robust to failures and numeric errors `:D` This is why DL infrastructures are built like web infrastructures, where nodes are assumed to fail.

4.1.3 Is DL all you need?

A lot of recent advances in DL look like very general purpose methods/techniques and are not domain[problem]-specific.

1. NFL, Bayesian theorems

   "For most decision rules, you can find some prios that makes that decision rule Bayes-optimal". Similar to No Free Lunch theorems.

   Conclusion: there's no universal [artificial] neural network, the domain matters!

4.1.4 Computer vision view

1. Edge detection with Canny

   Assumes noisy step edges, construct an edge detector using an optimal linear filter [Canny, 1986]. Precursor of DL for edge detection.

   Deep Learning approach: train one filter for edge localization and multiple additional filters for false positive suppresion. DL **can** also supress spurious boundary responses: instead of a single localizer, train two localization filters (each offset by one pixel from the actual localization).

   Lessons: even simple suboptimal nonlinear models can perform better than optimal linear models.

4.1.5 Computer science view

1. Sigmoidal units converge to linear threshold units: $\lim_{\alpha \to \infty} \sigma(\alpha M x + \alpha b) = floor(Mx + b > 0)$

2. NNs and boolean circuits

   Any neural networks has a boolean circuit equivalent (and vice versa!)

   Bool circuits as NNs:

   - start with an algorithm or boolean circuit
   - binary -> real, and -> multiply, or -> add + sigmoid

   NNs as bool circuits:

   - boolean circuit complexity transfers to NNs

3. Complexity results

   AC0?

   PAC learnable?

   Perceptron analysis?

4. How does complexity show up in DL?

   Computational complexity problems turn into exponential numbers of local minima.

   Example: pick bool satisfabiability problem, build an equivalent NN, try to solve via SGD, this doesn't get around NP-hardness!

4.1.6 Pattern recognition view

How do fully connected layers relate to traditional ML algorithms?
What is the difference between PCA and a linear autoencoder?
A NN can act like PCA, ICA, VQ...

4.1.7 Signal processing view

1. Convolutions

   Convolutions are linear operations on the input. Convolutional layers are just a special case of a fully connected layer, but with some parameter tying and specific Toeplitz (diagonal-like) matrix structure.

These two properties are not really present (at least, not forced) in our biological vision system.

Translation invariants can be built into the algorithms (e.g. CNN) or learned from data. Object recognition is **not** necessarily translation-invariant (blue patch above horizon is sky, below horizon is puddle of water)

2. Footprint

   In a convolutional layer, pixels in the output depend only on a subset of the pixels of the input. Max-pooling and stacking conv layers extend this footprint.

3. Checkerboards as aliasing

   From an image processing point of view, image generation with CNNs lacks anti-aliasing.

4. Separability

   Cl(n, r) ˜ Cl(n’, (1, e)) | Cl(n’, (r, 1))

5. Filters and convolutional layers

   Classic useful nonlinear filters: median, percentile, morphology. Conv layers (with nonlinearities) cannot implement such filters in general but, for a given input distribution, they often can give a good approximation. Batch normalization may make nonlinearities much more effective.

4.1.8 Decision theoretic view

1. Loss functions

   (a) Zero-one

   Classification occurs according to the maximum of the posterior probability (proof needed).

   (b) Decision matrix

   (c) Stuff

   Both sigmoidal outputs with mean squared error and softmax outputs asymptotically approximate posterior probabilities, but convergence is difference.

(d) Example: impalanced training set

When resampling or using training weights, prior probabilities are altered! We need to take this into account to correct posteriors.

4.2 Sequence modeling

4.2.1 Hidden Markov models

1. Markov chain

   Markov chains are discrete time sequences described by state graphs: with some probability we pass from one state to other.

2. Markov models and probabilities

   Markov property: Markov models are memory-less, the probabilities of the current state only depend on the probabilities from the previous one.

3. Language models

   A language model assigns probabilities to strings.

   (a) Useful language models

   The set of possible strings over an alphabet is infinite. So we want language models where we can get information like "what is the most likely sequence?".

   (b) Example: most likely sequence

   log probaility of a path is equal to the sum of the log probabilities of each traversed node.

   Finding the most likely path consists in finding a shortest path.

4. Hidden markove models

   HMMs are like Markov models but you cannot observe (some of) the states directly.

5. HMM algorithms

   Inference:

   - given seq of observations, infer the state sequence
   - Viterbi algorithm: what is the most likely seq
   - Forward backward: what is the probability distribution at time t

Training:

- Baum-Welch training: update the parameters given sample sequences. It's a type of EM algorithm.

6. Transducers

   Slight generalization of HMMs encoding I/O transformations, written as states with transitions on the edges.

7. HMM speech recognition with transducers

   Can build a modular, classic speech recognition algorithm with transducers. Modern algorithms (nn-based) are less modular but more efficient.

4.2.2 Simple RNNs

At each time point, the current input is feeded to the network as well as the previous output. This process can be unrolled for training, but this suffers from the vanishing gradient problem.

1. HMMs vs RNNs

   An HMM/transducer is a generative model, but an RNN is a discriminative model.

   Can we use HMMs for prediction as well? Yes, **but** performance would be poor (inference is non-Markovian: what we decide at a later step can actually depend on inputs way before that), and you'd lose all of the other nice properties.

2. CTC

   Sequences of different lengths -> add an "empty" symbol #

   We now need to align/match the ground truth with the obtained output so that we train our sequence model "in the right place": e.g. "LIEBE KATZE" generates "MO#CE COT##", ground truth is "NICE CAT", aligned ground truth is "NI#CE CAT##".

   Algorithm: "reverse" Viterbi of Forward backward.

4.3 LSTMs

Motivation for LSTMs is 2-fold:

1. An architectural design for a memory cell that can be stored and then read. This design can be applied to NNs replacing hard switches for multipliers and input signals for learned models.

2. Getting a structure unrolled in time that solves the problem of vanishing gradients. LSTMs achieve this.

4.3.1 Capabilities

1. Delay

   LSTM can implement the delay of a signal. In order to implement the delay, you need to store all the bits according to the period, so this can test the memory capacity of your LSTM

2. Mod 2

   LSTMs can detect pairs (or more) pulses, and change its state (up or down) for the first and reset it for the second. Same for mod-n.

3. Backwards delay

   LSTMs and RNNs are *causal filters*.

4. Linear filters

4.3.2 Computer Science View

LSTMs can learn (some) regular languages, (some) context free languages, and (some) context dependent languages.

Note vXgJ3M9C-$_{\text{E}}$ includes some fancy visualizations of the behavior of LSTMs.

4.3.3 LSTMs are drop-in replacements for convolutions

Both have the same kind of input and output. But they have many more capabilities: large linear filters, lg/variable delay, mod n, some regular, context free and context dependent languages.

However, LSTMs may be more costly than conv layers when both work, and don't parallelize as well. LSTMs probably are not as efficient for applications invoolving memorizing a large number of patterns in weights.

4.3.4 Multidimensional LSTM

- 2DLSTMs are constructed like separable filtes
- 2DLSTM are drop-in replacements for conv2d

4.4 Text recognition

4.4.1 The Dropbox system

Pipeline: take photo of document -> line / word detection -> word detection -> word recognition.

word detection: didn't use DL :(

word recognizer: stack of convlayers -> stac of bi-directional lstm layers -> CTC layer

They also used synthetic data.

Their total word recognition rate was 44%. Sources of error: places were they didn't use DL. Some corrections using dictionaries.

4.4.2 Ideas for improvement

Don't use word detection! Agglutinative languages exist (Finnish!), and the concept of words isn't useful for japanese/chinese.

Don't ignore state of the art!

4.4.3 Fully Deep Learning based pipeline

1. Layout analysis

   Distinguish columns/sections of text, and different zones such as math formulas, images and tables. Also, discard border noise, including parts of pages we don't want.

   ConvNets fail miserably here, partly because they can't propagate info through long distances and they are designed not to discern the absolute position of elements.

2. Rational DL Design

   To design a good DL system:

   - start with a non-DL image processing pipeline
   - replace each module with a equivalently trainable neural network architecture

    - ?
    - profit

3. Layout analysis with MDLSTM

   Yo can think of the layout analysis problem as a simple context-free or context-dependent language that can be learned by the MDLSTM.

4.5 Image segmentation with conv nets

Problems:

- semantic segmentation (distinguish different objects from background and from each other)
- object localization (knowing what object is in the image, localize it)
- human pose estimation
  - hierarchical/sequential approach: first localize center of body, then other locations relatively
- video classification
  - uses two kinds of resolutions, akin to eye's fovea
- tracking (track a car/person/object along time in a video)
  - pretrains a denoising autoencoder to get a rich feature representation
  - at tracking time, add a final logistic layer to the autoencoder representation to obtain a classifier for the target rectangle
  - combine this with particle filtering for final tracking
- superresolution
- edge detection
- semantic edge detection
- intrinsic image decomposition
- optical flow
-

4.5.1 Analysis

All these methods maintain a common AlexNet-like architecture, upscale at the end when needed and introduce skip connections (connections from early layers to the last ones) to transfer some of the original information.

Level sets are interesting because they model regions and edges simultaneously.

5 Emotion, Top-Down Attention, and Brain Internal States for Next-Generation Chatbots (Lee)

5.1 Introduction

5.2 Emotion

5.3 Top-Down Attention

5.3.1 Perception

Audiovisual senses require much more processing than olfatory, smell and touch.

Photosensors in our retina: several millions, ˜100 refreshes per sec (?), 3 colors = some MB/s? Audio: roughly 40 KB/s

Our audio and video information are coupled in the brain: McGurk and Stroop effects.

5.3.2 Classical approaches

A/V integration: early or late

- Early integration: features are concatenated and passed to the same learners
- Late integration: separate learners, outputs are concatenated

Early integration sounds more plausible but is not accurate to the brain structure. Late integration cannot explain/simulate the McGurk effect among other phenomena.

5.3.3 Bottom-up attention

Doesn't generally use any bias.

5.3.4 Top-down attention

Gets influenced by bias: memory, "personality"...

Essentially, previous information allows to fill any missing stuff or eliminate stuff we don't care about, thus *improving* our attention according to our bias.

1. Feature similarity gain model

   An *attention gain* is attached to the inputs

2. Multiplied (gated) model

   The attention parameters may be calculated by the input data (bottom-up) or by generated outputs (top-down) or both.

3. Top-down selective attention (TDSA)

   Top-down attention starts acting when the classifier is undecided between several classes. This way, the classifier attempts to recognize the data as one of the possible classes, modifying the attention (in a top-down fashion) so that the classifier obtains a new class output. If the difference is very small, then the class is (likely to be) incorrect. If its probability goes way up, then it's (likely) correct. Attention then shifts to other possibilities and the cycle is repeated.

5.4 Brain Internal States

6 (Principe)

6.1 Requisites for a cognitive architecture

6.1.1 Human inspired sensory processing

1. Functions of human cognition Cognition is a metal process by which knowledge is acquired, incl. perception, intuition and reasoning. Four fundamental functions: atttention (capacity to filter unnecessary data from a huge stream), memory, perception action cycle, intelligence.

2. Perception-action-reward cycle

   3 sources of knowledge: genetics, interaction with the environment, society.

   Brain is a real time system that does predictions: only has information in the past, acts on the future.

3. Fuster's hierarchy

   Each perception area in the brain has a corresponding area for execution/motor functions.

4. Perception as an active process

   Idea: perception is an **active** process based on context and predictions. In the brain, there are top-down connections to the sensors: this way sensors may be biased to extract what they consider important.

   Human visual system: ventral pathway (object identification, works in the foveal region, uses lt memory to stor representations of objects) and dorsal pathway (object location, spatial perception and motion, requires working memory).

   Idea: model perception as a continuous loop to extract events in the real world, mimicking saccades.

6.1.2 Cognitive perception architecture

Generative model for learning and memory

- Feature extraction for movies: sparse coding of low-level features
- (Locally) Invariant representations
- Feature hierarchies

6.1.3 How technologists approach memory and perception

Differently from biology. Memory in a Turing machine is a second-class citizen.

6.2 Putting the cognitive architecture together

6.2.1 Cognitivev model for object recognition in video

It learns autonomously to represent the external world. It's bidirectional so that the top levels can be mixed with sensory data.

Generative model: parameters are trained to represent the input in an effective way.

6.2.2 Hierarchical dynamic model with unknown inputs

It's a hierarchical model, it can be stacked, each module comes up with causes for the inputs (?).

On a patch of the video image:

- feature extraction: create an overcomplete state representation of the patch -> infer states. This uses a dynamic sparse coding model, with an energy function (neg log likelihood). Nestrov's smoothness,
- pooling: extract invariants on the image -> infer causes. Minimize an energy functional

6.2.3 Learned features and invariances

The measurement matrix C (uses sparse coding) is acting like the visual cortex of the brain.

1. Receptive fields

   The model predicts that images are smooth in time (they don't change abruptly).

6.2.4 Multi-layered architecture

Just a tree structure with tiling of scene at the bottom.

Scalable version: use convolutions for bigger images.

6.2.5 Recognition in noisy conditions

A connection from the previous top cause to the input can compensate for noise/occlusion in images (and audio).

6.3 Attention based video recognition

This takes inpiration on the saccades so that the model fixates on small parts of the image, scan them and do recognition, and repeat the process with another part.

They used a Lytro camera (pictures that can be refocused) to approximate the foveation.

6.3.1 Salicency

The quality by which an object stands out relative to its neighbors. This correlates well with eye-tracking systems.

6.3.2 Human inspired scene understanding

Use the refocusing and saliency to approximate the saccade. When a image patch is recognised, negative saliency is applied to it in order to look for other objects.

Saliency could also work top-down: the system can focus on what it knows is important.

Internal attention is better than bottom-up saliency.

7 Foundations of Deep Learning and Recent Advances (Salakhutdinov)

On Youtube: `https://simons.berkeley.edu/talks/tutorial-deep-learning`

8 Ideas

8.1 Conv-multilabel

en lugar de softmax, función de binarización en la capa de salida

8.2 Autoencoder composition

or online adding a smaller encoding layer and progressively fine-tuning