Deep Learning

Sunita Sarawagi

September 15, 2016

IIT Bombay

Table of contents

Introduction

Conventional learning –> Deep learning

- Conventional learning:
  - Requires domain expertise to engineer features
  - Mostly linear models, non-linear models based on kernels very expensive
- Deep learning:
  - Automatically learns ”multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level into a representation at a higher, slightly more abstract level”.
    - Deep learning also called Representation Learning
  - Non-linear: non-generic discriminatory features automatically learned.
  - Beating existing records in speech and image recognition
  - Very promising results in Natural Language Processing and conventional tasks like recommendation

Reasons for recent success of deep learning

- Unsupervised and transfer learning methods to learn internal feature representations under limited labeled data:
- (such pre-training later found not useful in applications with lots of labeled data.)
- GPUs to speed up computation 10 to 20 times: early success on speech recognition tasks
- Wider networks trained with specific hidden units (RELU, discussed later), less prone to local minimas

Feedforward Network Architecture

Feed forward networks

Multiple layers: input later, one or more hidden layers, output layers

Each layer a function: neural network == nested function.

Common template of each layer: affine transform of the input followed by unit-wise non-linearity. e.g. ReLU

Inspired by working of the brain: a layer has many parallel units, each unit acts like a neuron

Now, the brain analogy is less relevant. More about choosing functions which can be optimized in practice and which generalize well.

Example XOR

Neural networks can model decisions that conventional linear classifiers cannot.

$y = f^*(x) = x_1 \oplus x_2$

Training data = all four combinations.

Linear classifier $\hat{y} = w_1 x_1 + w_2 x_2 + b$ trained with least square loss yields $w_1 = w_2 = 0, b = 1/2$

Cannot discriminate

Non-linear classifier such as one with $x_1 x_2$ as feature ($\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + b$) can discriminate but the burden is on us to create the useful non-linear features.

Example XOR

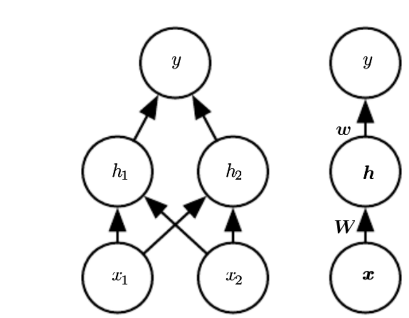
A generic two layer neural network with ReLU:

$y = f(x) = w^T \max(0, W^T x + c) + b$

Role of non-linear transform.

$W = [1\ 1\ 1\ 1]$

$c = [0\ \ -1]^T, w = [1\ \ -2], \quad b = 0$



Designing Feed forward networks

Need to choose functions at each layer so that they are easily trainable in spite of being non-convex.

- Output layers: depends on the output type (Generalized linear model from the exponential family)
    - Binary class labels: sigmoid function transforms arbitrary reals to probability of Bernoulli
    - Multi-class class labels: softmax provides multinomial probabilities
    - Real: output is mean of Gaussian distribution.
- Advantage of all of above: Probability distribution over output. Maximum likelihood training loss is convex (only in terms of parameters of outer-most layer.)
- Similar to conventional training.

Generalized Linear Models

A general framework for converting a vector of reals x into a discrete distribution over a dependent y: $P(y|x)$.

1. Choose an exponential family distribution for y. Determined by the data Bernoulli, Multi-nomial, Gaussian.
   1.1 Sufficient statistic is identity $T(y) = y$. E.g. for Bernoulli and multinormial and scaled version of Gaussian.
   1.2 Scaled version of exponential family: $P(y) = h(y, s) \exp(\frac{y\eta - A(\eta)}{s})$ Works with $s = \sigma^2$
2. Equate the mean of y, μ_y to $r(wx)$, where r is a response function.
3. $\eta = \psi^{-1}(\mu_y)$. Canonical form is: $\psi(z) = r(z)$. Thus, $P(y|x) = h(y, s) \exp(\frac{ywx - A(wx)}{s})$

Advantages: Maximum likelihood training loss is convex in w

Generalized Linear Models: examples

y is binary. Exponential family function is Bernoulli. Response function for the canonical form can be worked out to be sigmoid.

y is multi-valued discrete. Exponential family function is Multi-normial. Response function is softmax.

y is real-valued. Exponential family function is Gaussian. Response function is identitiy.

Designing hidden units

More trial and error, many options

Considerations: want some non-linearity, informative gradient (e.g. when convex), fast computation, close to linear

Role of the gradient of the hidden unit during training

Training objective of DNN with one hidden unit $h = g(w_1 x)$

$J(w_1, w_2, x, y) = L(h w_2 y) = L(g(w_1 x) w_2 y)$

Gradient of above w.r.t w_1 is $L' w_2 y g' x$

If $g' = 0$, the gradient becomes zero and we do not know in what direction to move w_1.

Hidden units types

- RELU: not differential but okay since gradient is informative. second-derivative zero in most places (useful for optimization)
  - Caution: watch out for inactive RelU: initialize affine input bias parameter to small positives. Gradient zero ==> information flow to lower layers is blocked.
- Generalized RELU: $\max(0, z) + \alpha \min(0, z)$: subsumes RELU, $|z|$, leaky relu ($\alpha = 0.001$), parameteric Relu.
- Maxout units: Form groups of units and retain Max within each group. Has the effect of learning the activation function. Requires more parameters –> more need to regularize
  - Can implement any convex function for large enough k
- Sigmoid/Tanh: tanh(z) = 2 sigmoid($2z$). Non-convex. Well-behaved (linear) only for small values of z, gradients very small for small or large z, problem for multi-layer network. Ok in output units because of the log-likelihood cost function.

Network architecture

Choosing the number of layers and width of the network and connection between layers

Universal approximation theorem: A network with one hidden layer (sigmoid type activation) can approximate any continuous function from a closed and bounded set given enough hidden units.

Proof also extended to work for RELU activations.

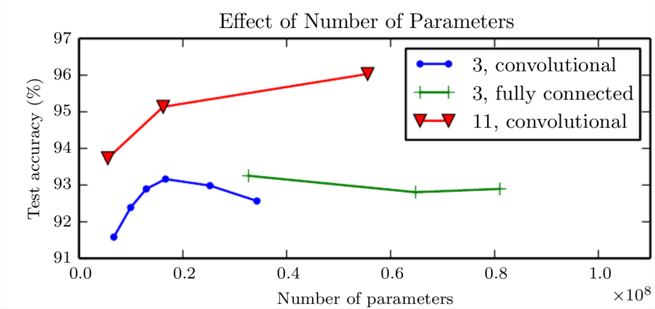
Not useful in practice:

number of hidden units required may be exponentially large,

the parameters of the network may not be easily learnable: might overfit on a wrong function.

Effect of depth

- Many functions can be efficiently represented with multiple hidden layers but require exponential width with single hidden layer
- The number of linear regions carved out via d inputs, l+1 depth, n units per hidden layer is $O\left(C(n,d)^{dl} n^d\right)$
- Empirically too, larger depth leads to better generalization and lower error.



Computation issues: forward and backward propagation

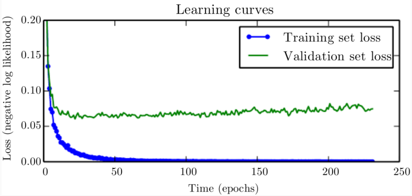
Regularization

Regularization: Parameter norm penalty

Regularization: reduce generalization error of a model even at the cost of training error

- Training objective = $\min_\theta cost(data,\theta) + \alpha regularizer(\theta)$
  - $L2$ and L1 norms and their effect on parameters same as in normal classification
  - Special about deep learning: effect of non-convex objective
  - Regularizer in constraints better for neural network training as it avoids local optimum.
  - Python demo.

Regularization: Early stopping



- Training error reduces with training iteration but validation error dips and then increases
- Stop training when error on a set-aside validation set increases more than a certain number of times.
- Why does early stopping help? restricts search space among parameters reachable within a limited capacity of the starting point. Prevents over-fitting.

Regularization: Drop out

•

Other methods for regularization

- Dataset augmentation: perturb x to generate more examples while keeping y-same. useful only in applications where valid perturbations well-understood, e.g. image processing
- Label smoothing: for multi-class classification with cross-entropy loss on softmax outputs, hard labels encourage large weights. Replace hard labels with soft labels ϵ for all incorrect and remaining for the one correct label.
- Semi-supervised learning: enforcing smooth function by using near-by unlabeled data clusters to have the same predicted label
- Multi-task learning: Related tasks share some hidden layer. A feature useful for many tasks is less likely to overfit.

Other methods for regularization

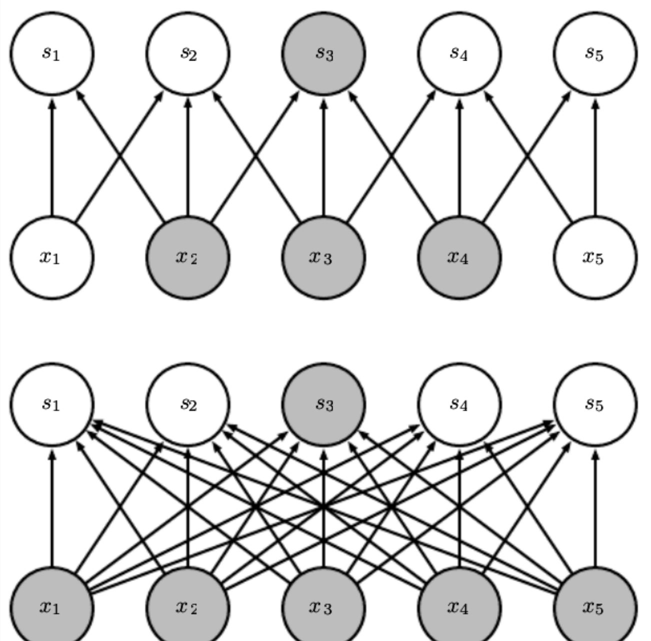
- Representation sparsity: constrain hidden vector h to be sparse via a regularizer.

Convolution Neural Networks (CNNs)

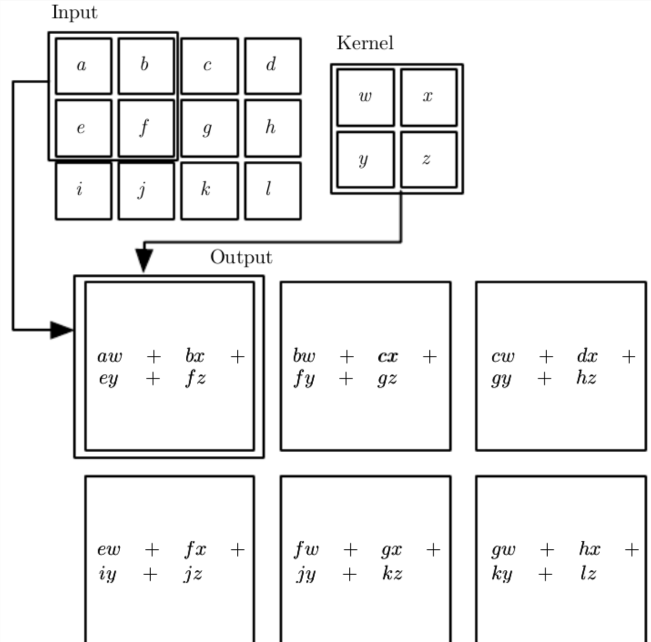
CNNs

- A special layer suitable when input features are regular and inter-changeable
- Popularly used in images where object location is not pinned, variable length sequences, time series data.
- Examples: 1-D, 2-D
- Reasons for CNN: sparse connections, parameter sharing, variable length input

1D CNN



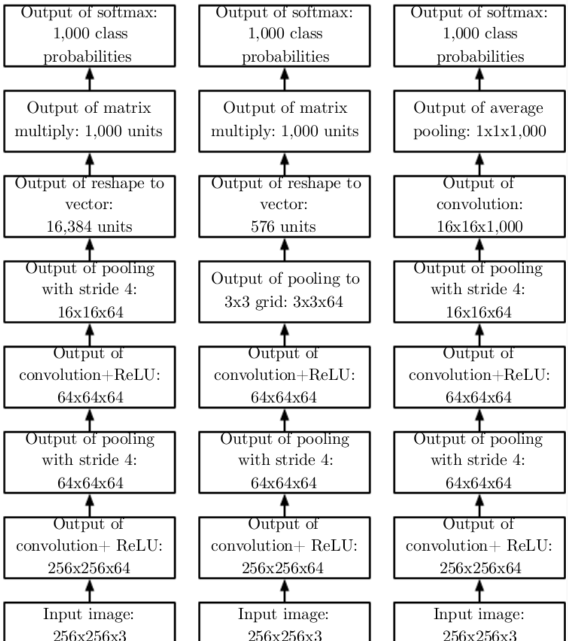
2D CNN


Input
a
b
c
d
e
f
g
h
i
j
k
l
Kernel
w
x
y
z
Output
aw + bx + ey + fz
bw + cx + fy + gz
cw + dx + gy + hz
ew + fx + iy + jz
fw + gx + jy + kz
gw + hx + ky + lz

Pooling

- Replace output by a summary over near-by nodes.
- Example summary functions: max, average, L2 norm
- Reason: position invariance
- Stride: reduce size of the input

Example CNN,Pooling for image recognition

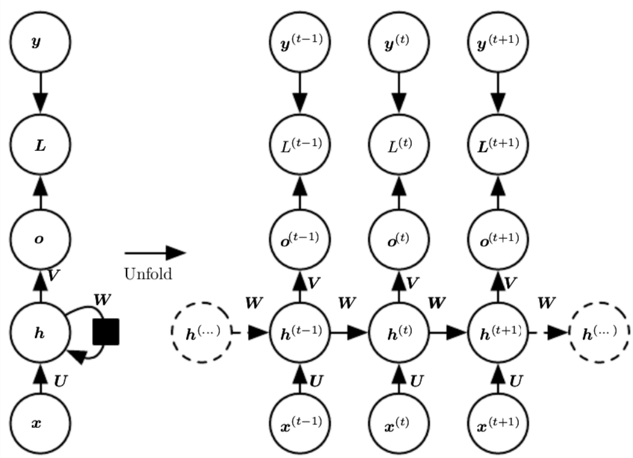


Recurrent Neural Networks (RNNs)

RNNs

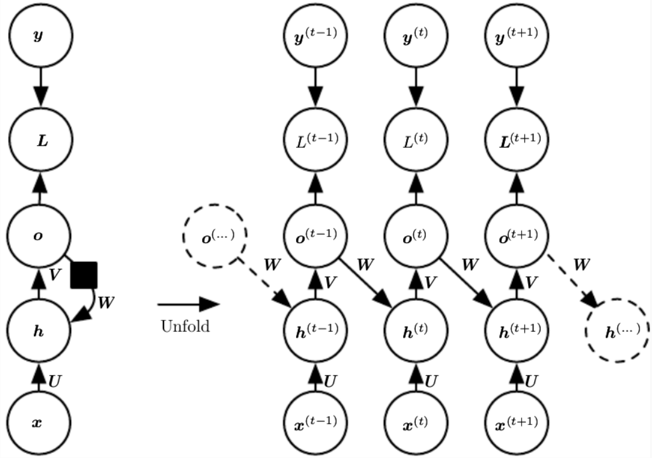
- A model to process variable length 1-D input
- In CNN, each output is a function of corresponding input and some immediate neighbors.
- In RNN, each output is a function of a 'state' summarizing all previous inputs and current input. State summary computed recursively.
- RNN allows deeper, longer range interaction among parameters than CNNs for the same cost.

RNN type-I



$$o^t = c + Vh^t, \quad h^t = \sigma(b + Wh^{t-1} + Ux^t)$$

RNN type-2

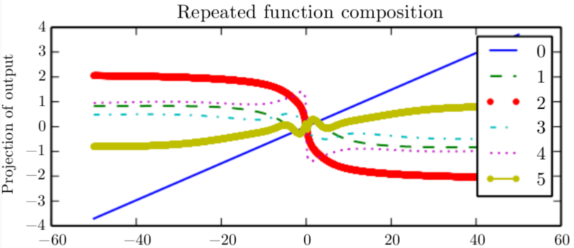


$$o^t = c + Vh^t, \quad h^t = \sigma(b + Wo^{t-1} + Ux^t)$$

Backpropagation through time

Challenges of capturing long-term dependencies

Exploding and vanishing gradient problem

- Linear-only units: simple-case where hidden state h is updated as $h^{(t)} = (W^t)^T h^{(0)}$. Use Eigen decomposition $W = Q\Gamma Q^T$, then $W^t = Q\Gamma^t Q^T$
- Product of non-linear interactions: gradient either small or large 

Training Neural Networks

Mini-batch size

Why justified. Factors influencing choice of batch size simultaneous updates

-

Ill conditioning

Large curvature (ill-conditioning) easily fixed by second-order methods in convex problems. Problem in NN.

-

Many local minima

Not as much a problem because..

- A prime source of local minima is non-identifiability of hidden variables.
  1. Many equivalent solutions obtained by permuting inputs and corresponding outputs of hidden layers. (Weight space symmetry)
  2. Scaling inputs and corresponding parameters (no regularizer)

  Many local-minimas with similar objective value
- Often objective decreases even with gradient norm being large implying that at termination local minima is not the problem.

Saddle points

- Many critical points (small gradient) are saddle points.
- Saddle points more likely in high-dimensions — theoretically for random functions and observed empirically too.
- Saddle points are typically observed at high values and local minimas at low values.
- Failure of second-order methods in NN training: cannot handle saddle points and local maxima because it seeks regions of zero gradient. Gradient descent uses gradient just as a descent direction –> successfully navigates around saddle point