Deep Learning II

Riashat Islam

Reasoning and Learning Lab
McGill University

1st November 2017

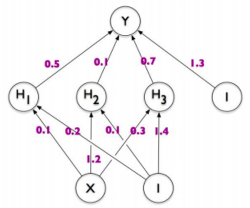
Bayesian Neural Networks

Motivation - Why BNNs

- With Bayesian methods, we obtain a distribution of answers to a question rather than a point estimate
- This can help address regularization and model comparison without a held-out validation set
  - Compare and choose architectures, regularizers, and other hyperparameters
- Can also compute a distribution over outputs: place error bars on $P(y \mid x, \mathcal{D})$

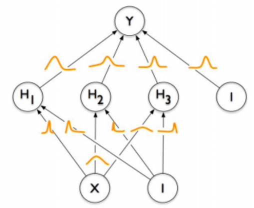
Comparison

Standard Neural Net



- Parameters represented by *single, fixed values (point estimates)*
- Conventional approaches to training NNs can be interpreted as *approximations* to the full Bayesian method (equivalent to MLE or MAP estimation)

Bayesian Neural Net



- Parameters represented by *distributions*
- Introduce a prior distribution on the weights $P(\mathbf{w})$ and obtain the posterior $P(\mathbf{w} \mid \mathcal{D})$ through Bayesian learning
- Regularization arises naturally through the prior $P(\mathbf{w})$
- Enables principled model comparison

Images from: Blundell, C. et al. Weight Uncertainty in Neural Networks. *ICML 2015*.

Bayesian Approximation

Minimizing:

Is Equivalent To:

Squared error (no regularization)

$$\sum_{i=1}^{N} (y(x^{(i)}; \mathbf{w}) - t^{(i)})^2$$

$\Rightarrow$

Maximum likelihood estimation

$$\mathbf{w}_{MLE} = \arg\max_{\mathbf{w}} P(\mathcal{D} \mid \mathbf{w})$$

Squared error (+ L^2 regularization)

$$\beta \sum_{i=1}^{N} (y(x^{(i)}; \mathbf{w}) - t^{(i)})^2 + \alpha ||\mathbf{w}||^2$$

$\Rightarrow$

MAP estimation with a Gaussian prior

$$\mathbf{w}_{MAP} = \arg\max_{\mathbf{w}} P(\mathbf{w} \mid \mathcal{D})$$

where $\mathbf{w} \sim \mathcal{N}(0, \sigma_w^2)$

Role of Integration in Bayesian Methods

- Many problems addressed by Bayesian methods involve *integration:*
  - Evaluate *distribution of network outputs* by integrating over weight space

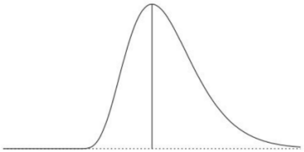
$$P(y \mid x, \mathcal{D}) = \int P(y \mid \mathbf{w}, x) P(\mathbf{w} \mid \mathcal{D}) d\mathbf{w}$$

  - Compute the *evidence* for *Bayesian model comparison*

$$P(\mathcal{D} \mid \mathcal{H}_i) = \int P(\mathcal{D} \mid \mathbf{w}, \mathcal{H}_i) P(\mathbf{w} \mid \mathcal{H}_i) d\mathbf{w}$$

- These integrals are often intractable, and must be approximated

Approximating Integrals

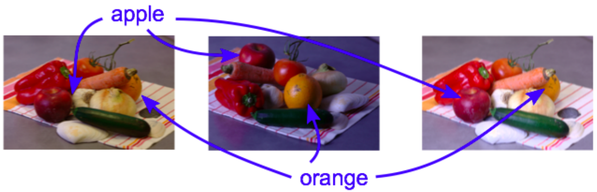


- Gaussian approximation (1/2): Allows the integrals to be evaluated *analytically*
    - Optimization involved in finding the mean of the Gaussian
- Monte Carlo methods (2/2): Draw samples to evaluate the integral directly
- Variational inference (next week): Convert integration into optimization
    - Minimize KL divergence between the posterior and a proposed parametric function

Convolutional Neural Networks

Big Picture

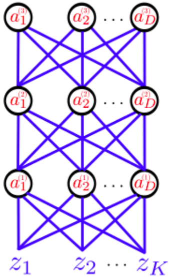
- **Goal**: how to produce good internal representations of the visual world to support recognition...
  - detect and classify objects into categories, independently of pose, scale, illumination, conformation, occlusion and clutter
- how could an artificial vision system learn appropriate internal representations automatically, the way humans seem to by simply looking at the world?
- **previously in CV and the course**: hand-crafted feature extractor
- **now in CV and the course**: learn suitable representations of images



Why use Hierarchical Multi-Layered Models

Argument 1: visual scenes are hierachically organised

$$\begin{tikzcd}
\text{object} & \text{trees} \\
\text{object parts} \arrow[u] & \text{bark, leaves, etc.} \arrow[u] \\
\text{primitive features} \arrow[u] & \text{oriented edges} \arrow[u] \\
\text{input image} \arrow[u] & \text{forest image} \arrow[u]
\end{tikzcd}$$


$a_1^{(3)}$ $a_2^{(3)}$ $\cdots$ $a_D^{(3)}$
$a_1^{(2)}$ $a_2^{(2)}$ $\cdots$ $a_D^{(2)}$
$a_1^{(1)}$ $a_2^{(1)}$ $\cdots$ $a_D^{(1)}$
z_1 z_2 $\cdots$ z_K

Why use Hierarchical Multi-Layered Models

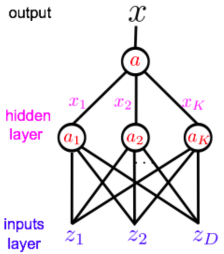
Argument 2: biological vision is hierachically organised

$$\begin{tikzcd}
\text{object} & \text{trees} & \text{Inferotemporal cortex} \\
\text{object parts} \arrow[u] & \text{bark, leaves, etc.} \arrow[u] & \text{V4: different textures} \\
\text{primitive features} \arrow[u] & \text{oriented edges} \arrow[u] & \text{V1: simple and complex cells} \\
\text{input image} \arrow[u] & \text{forest image} \arrow[u] & \text{photo-receptors retina}
\end{tikzcd}$$

Why use Hierarchical Multi-Layered Models

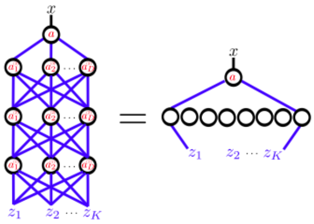
Argument 3: shallow architectures are inefficient at representing deep functions

single layer neural network implements: $x = f_\theta(\mathbf{z})$



networks we met last lecture with large enough single hidden layer can implement **any** function **'universal approximator'**

shallow networks can be computationally inefficient



however, if the function is 'deep' a very large hidden layer may be required

Why not Standard Neural Networks

How many parameters does this neural network have?

$$|\theta| = 3D^2 + D$$

For a small 32 by 32 image:

$$|\theta| = 3 \times 32^4 + 32^2 \approx 3 \times 10^6$$

Hard to train

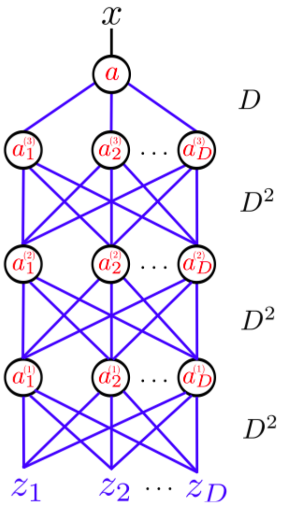
over-fitting and local optima

Need to initialise carefully

layer wise training

unsupervised schemes

Convolutional nets reduce the number of parameters

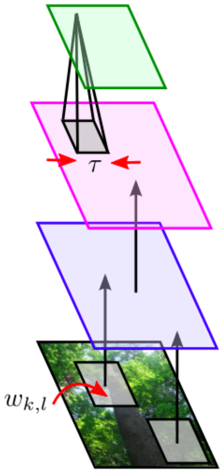


x
a
D
$a_1^{(3)}$ $a_2^{(3)}$ $\cdots$ $a_D^{(3)}$
D^2
$a_1^{(2)}$ $a_2^{(2)}$ $\cdots$ $a_D^{(2)}$
D^2
$a_1^{(1)}$ $a_2^{(1)}$ $\cdots$ $a_D^{(1)}$
D^2
z_1 z_2 $\cdots$ z_D

Key Ideas behind CNNs

- **image statistics are translation invariant** (objects and viewpoint translates)
  - build this translation invariance into the model (rather than learning it)
  - tie lots of the weights together in the network
  - reduces number of parameters
- **expect learned low-level features to be local** (e.g. edge detector)
  - build this into the model by allowing only local connectivity
  - reduces the numbers of parameters further
- **expect high-level features learned to be coarser** (c.f. biology)
  - build this into the model by subsampling more and more up the hierarchy
  - reduces the number of parameters again

CNN Building Blocks



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \mathrm{sigmoid}(a)$

non-linear stage

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

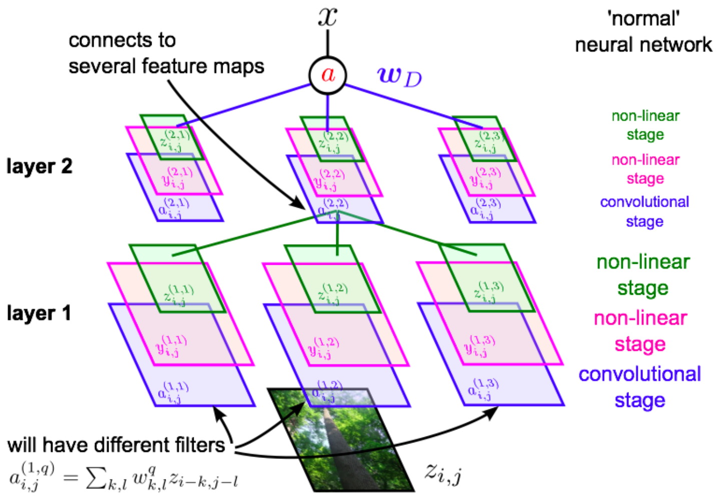
only parameters

convolutional stage

$z_{i,j}$

input image

Full Convolutional Neural Networks



How many parameters does CNN have

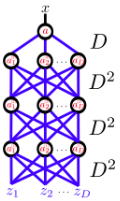
How many parameters does this neural network have?

$$|\theta| = 3K^2 + 9K^2 + 9K^2 + 3(D/S)^2$$
$$= 21K^2 + D$$

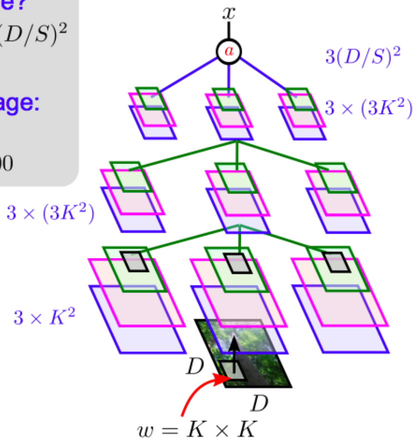
For a small 32 by 32 image:

$K = 5 \qquad S = 2$

$$|\theta| = 21 \times 5^2 + 4^2 \approx 600$$



$$|\theta| = 3D^2 + D \quad \approx 3 \times 10^6$$



CNN Training

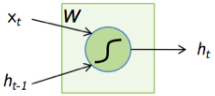
- **back-propagation for training**: stochastic gradient ascent
  - like last lecture output interpreted as a class label probability, $x = p(t = 1|\boldsymbol{z})$
  - now x is a more complex function of the inputs $\boldsymbol{z}$
  - can optimise same objective function computed over a mini-batch of datapoints
- **data-augmentation**: always improves performance substantially (include shifted, rotations, mirroring, locally distorted versions of the training data)
- **typical numbers**:
  - 5 convolutional layers, 3 layers in top neural network
  - 500,000 neurons
  - 50,000,000 parameters
  - 1 week to train (GPUs)

Cautionary Words

- hierarchical modelling is a **very old idea** and not new
- the ‘deep learning’ revolution has come about mainly due to new methods for initialising learning of neural networks
- current methods aim at invariance, but this is far from all there is to computer and biological vision: **e.g. instantiation parameters should also be represented**
- **classification can only go so far**: "tell us a story about what happened in this picture"

Recurrent Neural Networks

RNNs



Input at time t is $\mathbf{x_t}$

State at the end of time (t-1) is $\mathbf{h_{t-1}}$

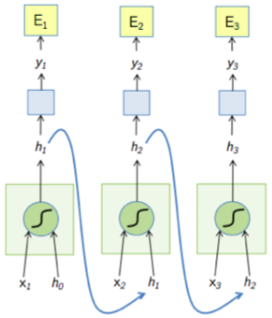
State at the end of time t is $\mathbf{h_t}$

$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

You can obviously replace **tanh** by your favorite non-linear differentiable function

How is it different from a standard neuron ??

RNNs



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

You can output **y_t** at each time step (based on your problem), based on **h_t**

$$y_t = F(h_t)$$

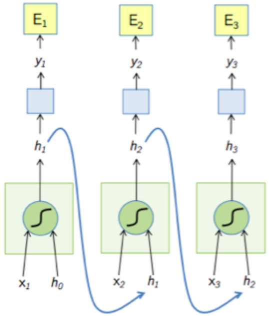
often linear function of **h_t**

Finally define error **E_t** based on **y_t** and the target output at time **t**

$$E_t = Loss(y_t, GT_t)$$

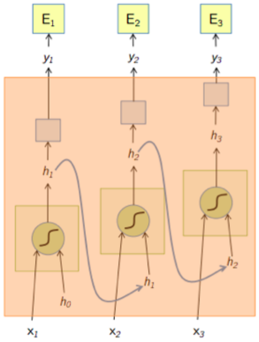
GT_t is target at time **t**

RNNs



- Note that the weights are shared over time
- Essentially, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps

Backpropagation Through Time (BPTT)



- Treat the unfolded network as one big feed-forward network
- This unfolded network accepts the whole time series as input
- The weight update is computed for each copy in the unfolded network, then summed (or averaged) and then applied to RNN weights

Issue with RNNs

Recall, back propagation requires computation of gradients with respect to each variable. For RNNs, consider the gradient of **E**$_t$ with respect to **h**$_1$

$$\begin{aligned}\frac{\delta E_t}{h_1} &= \left(\frac{\delta E_t}{\delta y_t}\right)\left(\frac{\delta y_t}{\delta h_1}\right)\\ &= \left(\frac{\delta E_t}{\delta y_t}\right)\left(\frac{\delta y_t}{\delta h_t}\right)\left(\frac{\delta h_t}{\delta h_{t-1}}\right)\cdots\left(\frac{\delta h_2}{\delta h_1}\right)\end{aligned}$$

Product of a lot of terms can shrink to zero (**vanishing gradient**) or explode to infinity (**exploding gradient**). Exploding gradients are often controlled by clipping the values to a max value. One way to handle vanishing gradient problem is to try to have some relationship between **h**$_t$ and **h**$_{t-1}$ such that each $\left(\frac{\delta h_t}{\delta h_{t-1}}\right) = 1$

Long Short Term Memory (LSTM) try to do that, but vanishing gradients still a problem. As a result, capturing long range dependencies is still challenging.

Summary

- RNNs allow for processing of variable length inputs and outputs by maintaining state information across time steps
- Various input / output scenarios possible (Single / Multiple)
- Exploding gradients are handled by gradient clipping, LSTMs are a way towards handling vanishing gradients.

You can read more about LSTMs at http://arunmallya.github.io/ . A major part of RNN slides were taken from there.

Questions ??