kptan86@gmail.com

Technical difficulties around neural network

Very notable characteristic of neural network: **Large number of latent variables (weights) to be fitted**

For neural network to work, we need

No	Requirements	Solution
1	Large amount of data for the fitting	Network architecture, Reinforcement learning, other tricks (no clear solution yet)
2	Efficient optimization methods for the fitting	Back-propagation (+architecture/activation function design), hardware solution
3	Specfial attention on ill-conditioning and overfitting prevention	Drop-out regularization, initialization method

Exploiting data properties and structure

- Fully connected deep neural network is an universal function approximator, however it often has ***way too many weights*** to fit
- For specific purpose, or data of different types, one often employs architecture of less density
  - Design minimal connections (weights) to capture the relationship in the data
    - (sacrificing universality for speed in a smart way)
    - (keep in mind that all variants of network is a subset of a fully connected network)
- **Designing such architecture and intelligent use of data properties (minimal connections for maximum information) is one of the most important topics in deep learning**

Exploiting data properties and structure

The more well-studied data architecture are (i) image (ii) time sequence

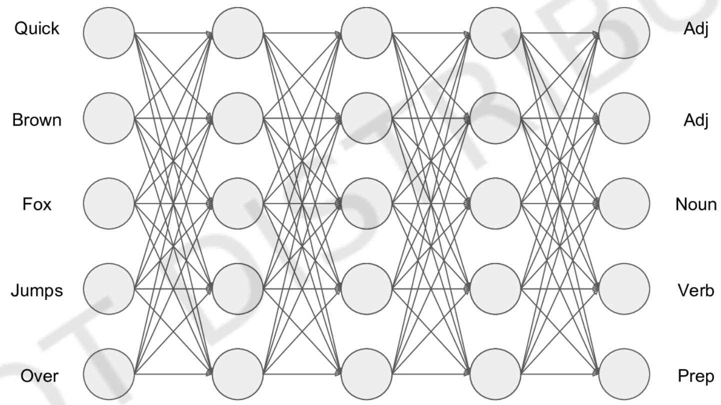
Data type	Architecture
Image	CNN
Time series	RNN
Text	?
Video	?
Audio	?
Others	?

(Others: tables, compressed content, source code, ...)

kptan86@gmail.com

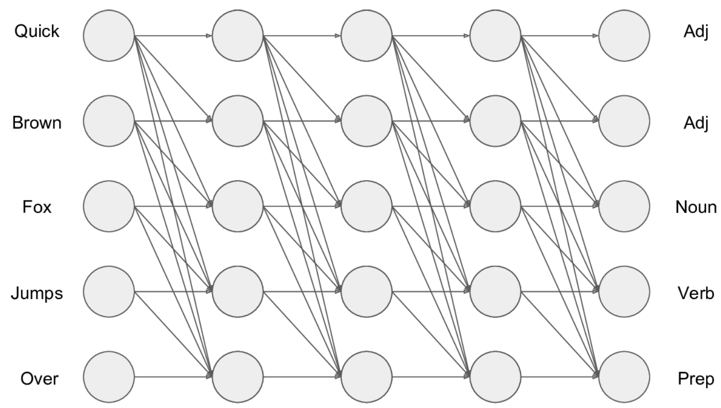
Recurrent Neural Network (RNN)

- **Fully connected neural-network**



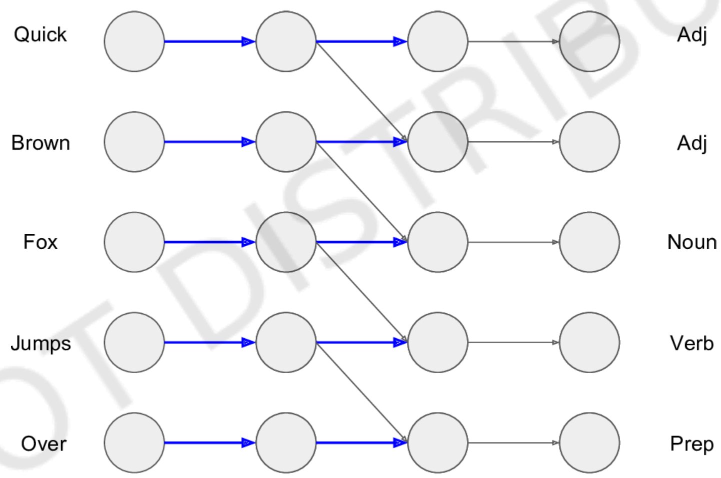
Recurrent Neural Network (RNN)

- **Information only flows unidirectionally from past to future** - remove back-in-time connections



Recurrent Neural Network (RNN)

- Further simplification (i) not deep (ii) only connect to immediate neighbour



blue line: copy over

DO NOT DISTRIBUTE

kptan86@gmail.com

Recurrent Neural Network (RNN)

The activation function takes into consideration of

1. Current input (x_t)
2. Previous cell output (o_{t-1})

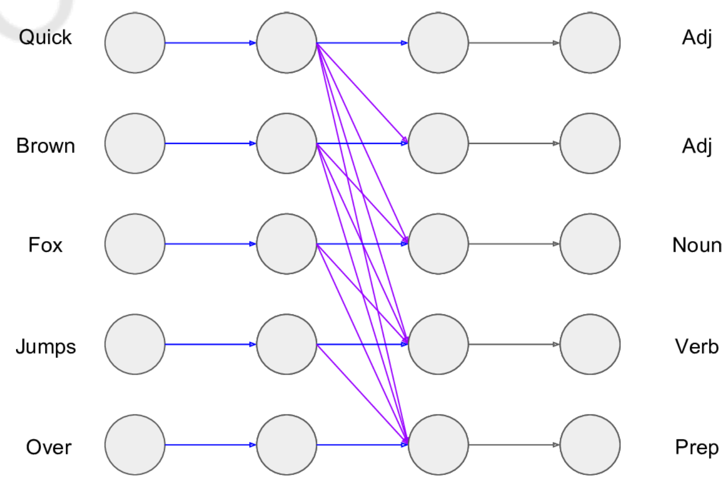
so is of the form

$$o_t = Wx_t + Uo_{t-1} + b$$

where W, U and b are weights and biases to be learned

Recurrent Neural Network (RNN)

- (variable gap dependence not modelled)



Long Short-Term Memory (LSTM)

Problem: Variable gap dependence not modelled

Solution: Make cell remember its past states $(s_t, s_{t-1}, s_{t-2}, \ldots)$

As there could have too many past states to remember, the states are summarized with one function

$$c_t = F(s_t, s_{t-1}, s_{t-2}, \ldots)$$

We notice that in the next step, the cell memory is simply the (next) current state and the memory of past state:

$$c_{t+1} = F(s_{t+1}, \boxed{s_t, s_{t-1}, s_{t-2}, \ldots})$$

It hinted to us that c_t could be written in a recursive form*:

$$c_t = \alpha(s_t) + \beta(c_{t-1})$$

(note: this is not exactly an elegance solution - as recursive form still uses window size of 1, and the gap dependence is not explicitly modelled)*

kptan86@gmail.com

Long Short-Term Memory (LSTM)

In effect, LSTM works by incorporate one more parameter in every neuron, dubbed "cell state (c_t)", to model the variable gap dependence.

LTSM-RNN neuron output (h_t) is computed as the "normal" output (o_t) "modified" by the "cell state" (c_t)

$$h_t = o_t \circ \sigma_h(c_t)$$

Note: the modification function (σ_h)

- could be of arbitrary form
  - usually chosen as $\sigma(x) = tanh(x)$ or $\sigma(x) = x$
- is usually parameter-free
  - as the parameters need fitting are taken care of in c_t

Cell state

As mentioned, it is hinted that cell state could be written in recursive form $c_t = \alpha(s_t) + \beta(c_{t-1})$

The roles of α and β are to control the balance between the current input and past memory during the update.

In RNN, we use the symbols

$$c_t = f_t \circ c_{t-1} + i_t$$

where i_t and f_t are the "input modifier" and "forget modifier" respectively.

Forget gate and input gate

The modifiers are dubbed "gates", and are just functions to be learned from data (in the same manner of the main activation function)

Logistic regression is the most commonly form for the functions

Forget gate

- $f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$

Input gate

- $i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$

where σ_g is a sigmoid function, and W, U, b are the weights and biases to be fitted.

kptan86@gmail.com

LTSM basic form

Putting all formula together

$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$

$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$

$c_t = f_t \circ c_{t-1} + i_t$

$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$

$h_t = o_t \circ \sigma_h(c_t)$

LSTM variants

There are multiple variants of LTSM implementations

"Vanilla LTSM": One more set of parameters to model the cell state input value modifier

$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$

$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$

$c_t = f_t \circ c_{t-1} + i_t \circ \boxed{\sigma_c(W_c x_t + U_c h_{t-1} + b_c)}$

$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$

$h_t = o_t \circ \sigma_h(c_t)$

LSTM variants

There are multiple variants of LTSM implementations

"Peephole LTSM": Use the cell state (instead of output) as input from the previous neuron

$f_t = \sigma_g(W_f x_t + U_f \boxed{c_{t-1}} + b_f)$

$i_t = \sigma_g(W_i x_t + U_i \boxed{c_{t-1}} + b_i)$

$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c \boxed{c_{t-1}} + b_c)$

$o_t = \sigma_g(W_o x_t + U_o \boxed{c_{t-1}} + b_o)$

$h_t = o_t \circ \sigma_h(c_t)$

kptan86@gmail.com

LSTM variants

There are multiple variants of LTSM implementations

"(fully) Gated Recurrent Unit (GRU)": Merging output gate (o_t) into cell state (c_t) (+*slighly (unimportant?) different functional form for cell state*). <u>One less set of parameters to fit.</u>

$$f_t = \sigma_g(W_z x_t + U_z c_{t-1} + b_z)$$

$$i_t = \sigma_g(W_r x_t + U_r c_{t-1} + b_r)$$

$$c_t = f_t \circ c_{t-1} + \boxed{(1 - f_t)} \circ \sigma_h(W_h x_t + U_h(\boxed{i_t} \circ c_{t-1}) + b_h)$$

$$\boxed{o_t = c_t}$$

$$\boxed{h_t = o_t}$$

LSTM variants

There are multiple variants of LTSM implementations

"(minimal) Gated Recurrent Unit (GRU)": Further merging input gate (i_t) and forget gate (f_t). <u>One lesser set of parameters to fit.</u>

$$f_t = \sigma_g(W_z x_t + U_z c_{t-1} + b_z)$$

$$\boxed{i_t = f_t}$$

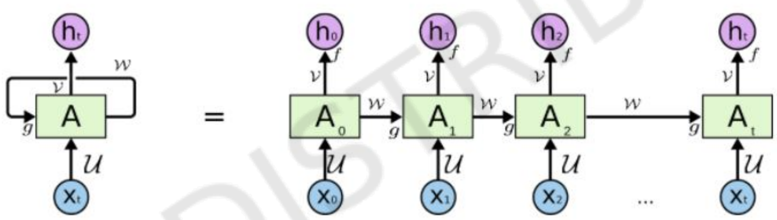
$$c_t = f_t \circ c_{t-1} + (1 - f_t) \circ \sigma_h(W_h x_t + U_h(i_t \circ c_{t-1}) + b_h)$$

$$o_t = c_t$$

$$h_t = o_t$$

"Folded"

- RNN is usually represented in "folded" form (hence the name "recurrent")
- **"Folded" RNN can take variable number of inputs**



kptan86@gmail.com

Discussion

Think of an example in language processing where RNN is not suitable (e.g. meaning of former word depends on what comes after)

Example:

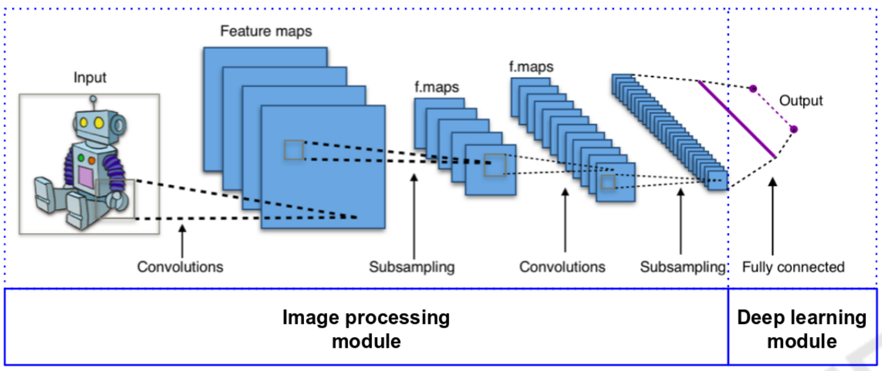
Input	道[1]	可[2]	道[3]	非[4]	常[5]	道[6]
Attribute	noun or verb ?	verb	verb	adjective	adjective	noun

The attribute of the first 道[1] is undetermined at the point in time, and only resolved at the point of the second 道[2].

Convolutional neural network (CNN)

- Mainly used for image related problem (recognition, classification, segmentation, denoising etc)
- Typically composed of two main components

1. Image processing / properties summarizing module
    - **Convolutional** filters: for image processing operations
    - Pooling layer: to reduce image size
2. Deep learning network module

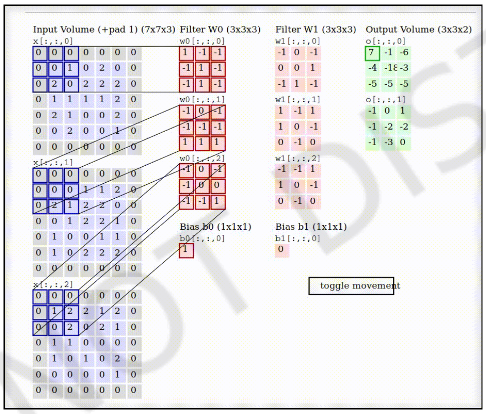


kptan86@gmail.com

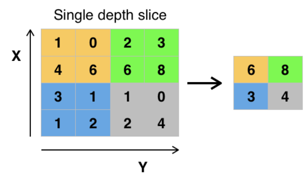
CNN common operator

- Pooling layer reduces number of neuronal connections between layers
- Convolutional layer *usually* designed to reduce connections as well

Convolution

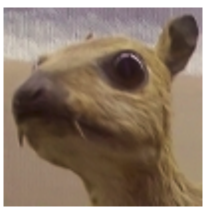
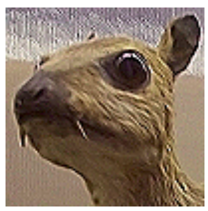
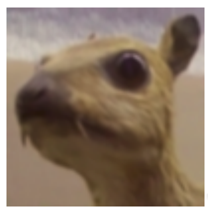
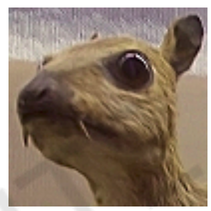


Pooling / Subsampling



Convolutional filters

- Convolutional filters are routinely used in image processing (and is not anything novel)
- Backprop in the learning enable the network to learn the best filters for the task

Original	Edge detection	Sharpen	Gaussian blur	Unsharp	...
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	...
					...

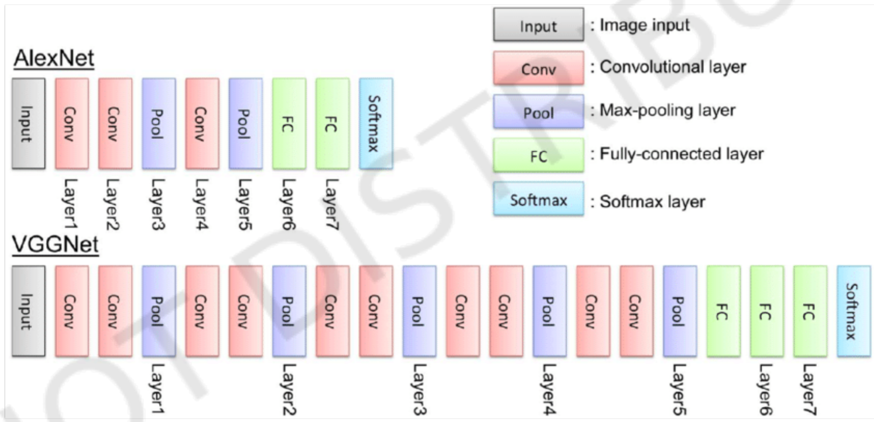
Notable CNN Network Architectures

1. AlexNet 2012
   - First high-performance "deep" CNN
   - Introduces **ReLU**
   - Introduces **Dropout** regularization
2. VGGNet 2014
   - Introduces improvements on **convolution filter** architecture design
3. GoogLeNet (Inception) 2014
   - Further improvement on **convolution filter** architecture design
   - Introduces **modularization** ("Inception module")
4. ResNet 2015
   - Introduces a variant on **modularization** (skip connection)
5. Xception 2016
   - Combination of GoogLeNet and ResNet

kptan86@gmail.com

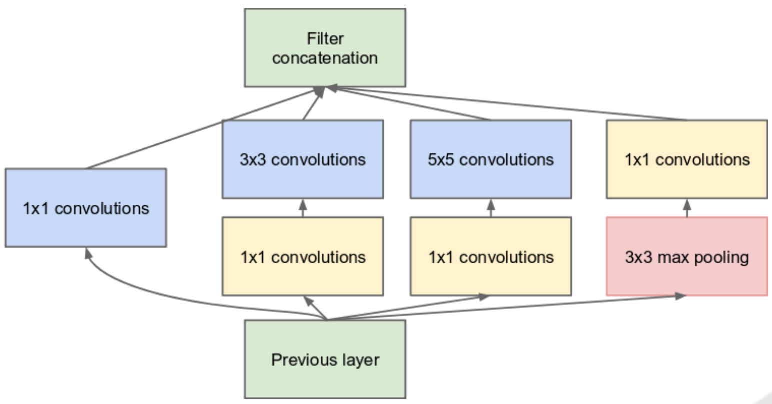
AlexNet and VGGNet

- VGGNet introduces more convolutional layers as compared to AlexNet



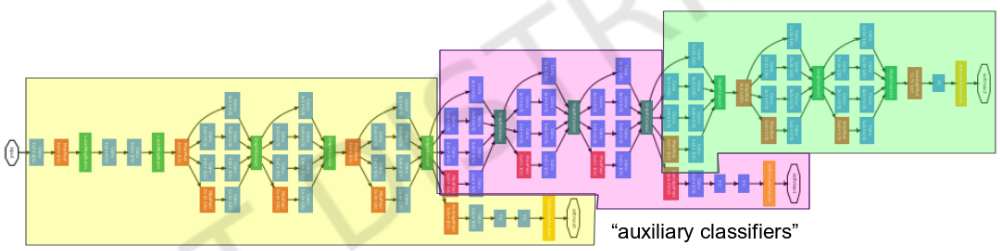
GoogLeNet

- Stacking output from filters of different sizes (1x1, 3x3, 5x5) together
- (Hopefully) Different filters capture information of different hierarchical levels



GoogLeNet

- Modularize network into 3 blocks
- Each block with a (auxilliary) classifier to reinforce error back-propagation

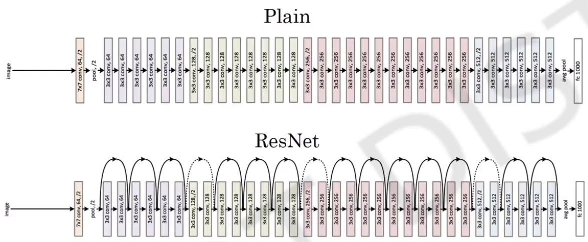


kptan86@gmail.com

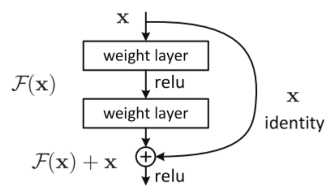
ResNet

- The last block is a standard neural network
- The "residual" error from the last block is modelled by the second last block (etc etc)
- (can be viewed as a variant of boosting algorithm)

Architecture



"Residual Block"



Xception