Lecture 4: Introduction to Deep Learning

Assoc Prof Tham Chen Khong
Dept of Electrical & Computer Engineering (ECE)
NUS
E-mail: eletck@nus.edu.sg

Acknowledgement: some materials from Géron, Hands On ML

Overview

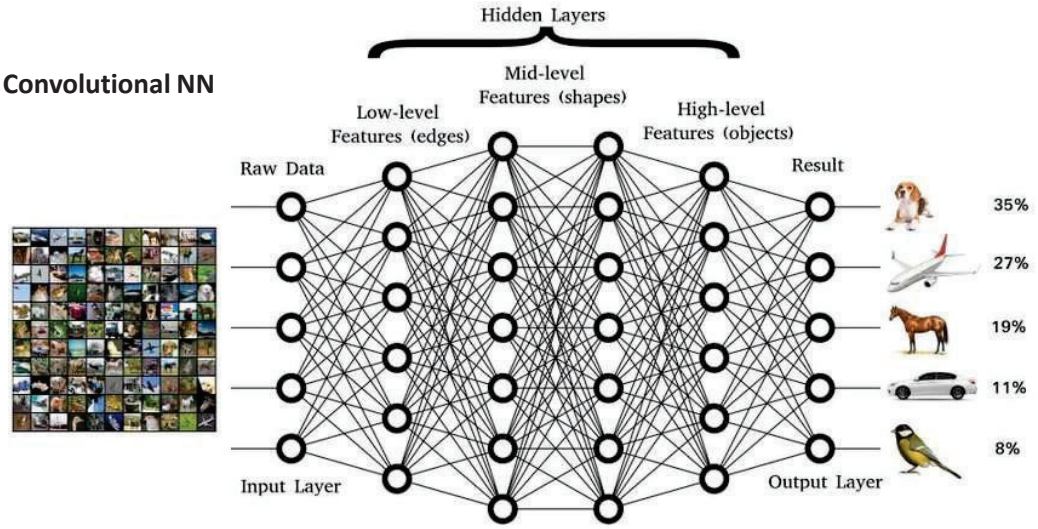
- Deep Learning
- Challenges
  - Unstable gradients
    - Batch Normalization
    - Layer Normalization
  - Overfitting
    - Regularization
    - Dropout

Deep Learning

- Usually 10 layers or more and usually refers to Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN)
- Performs very well in perceptual tasks, e.g. object recognition
  - also: handwriting (English, Asian), traffic signs, people, objects, animals, scenes, speech, medical imaging, language translation etc.

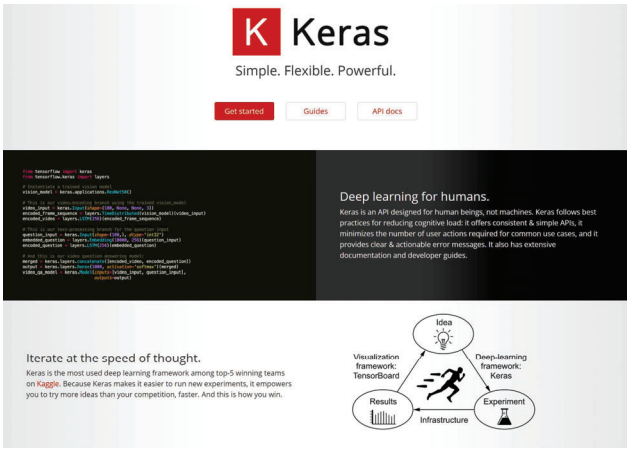
Deep Learning

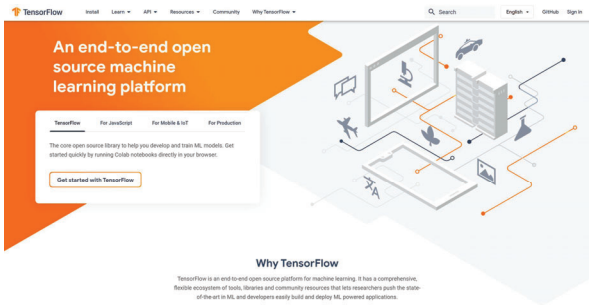
Convolutional NN

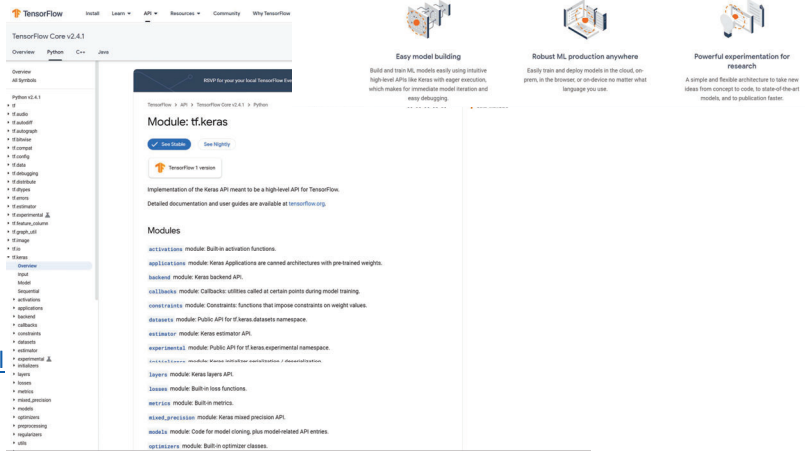


Modern Deep Learning systems for computer vision may have 20 or more layers.
Source: https://www.researchgate.net

TensorFlow2 and Keras







Useful links:
https://keras.io
https://www.tensorflow.org
https://www.tensorflow.org/api_docs/python/tf/keras
https://www.tensorflow.org/guide/keras/sequential_model
https://www.tensorflow.org/tensorboard/get_started

Challenges of Training Deep Neural Networks

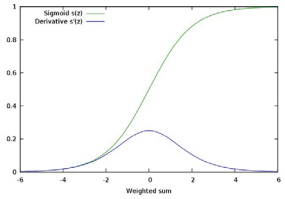
- Challenges
  1. You may be faced with the tricky vanishing gradients problem or the related exploding gradients problem. This is when the gradients grow smaller and smaller, or larger and larger, when flowing backward through the DNN during training. Both of these problems make lower layers very hard to train.
  2. You might not have enough training data for such a large network, or it might be too costly to label.
  3. Training may be extremely slow.
  4. A model with millions of parameters would severely risk overfitting the training set, especially if there are not enough training instances or if they are too noisy.

far away from o/p layer

(1) Unstable Gradients - 1

- In Gradient Descent, gradients of cost wrt parameters/weights often get smaller and smaller as the algorithm progresses down to the lower layers.
- As a result, the Gradient Descent update leaves the lower layers' connection weights virtually unchanged, and training never converges to a good solution. This is the *vanishing gradients* problem.
- In some cases, the opposite can happen: the gradients can grow bigger and bigger until layers get insanely large weight updates and the algorithm diverges. This is the *exploding gradients* problem, which tends to happen in recurrent neural networks (see later).
- More generally, deep neural networks suffer from *unstable gradients*, and different layers may learn at widely different speeds.

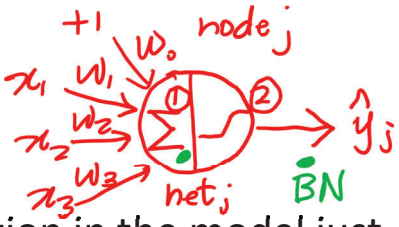
Unstable Gradients - 2



- The problem is caused by increasing variance over the layers and saturation arising from the weights initialization method (e.g. Gaussian with zero mean and std dev 1) and the logistic sigmoid activation function.
- For the signal to flow properly, Glorot & Bengio (2010) argue that we need the variance of the outputs of each layer to be equal to the variance of its inputs, and we need the gradients to have equal variance before and after flowing through a layer in the reverse direction.
- Achieve this through (1) proper weights initialization, and (2) non-saturating activation functions, e.g. (leaky) ReLU
- and
- (3) Batch Normalization

also, Layer Normalization

Batch Normalization



- Ioffe S., Szegedy C. (2015)
- The technique consists of adding an operation in the model just before or after the activation function of each hidden layer.
- This operation simply zero-centers and normalizes each input, then scales and shifts the result using two new parameter vectors per layer: one for **scaling**, the other for **shifting (offset)**.
- The model ***learns*** the **scaling** and **offset** for each of the layer's inputs through back-propagation (BP).

Batch Normalization

Equation 11-3. Batch Normalization algorithm

vectors

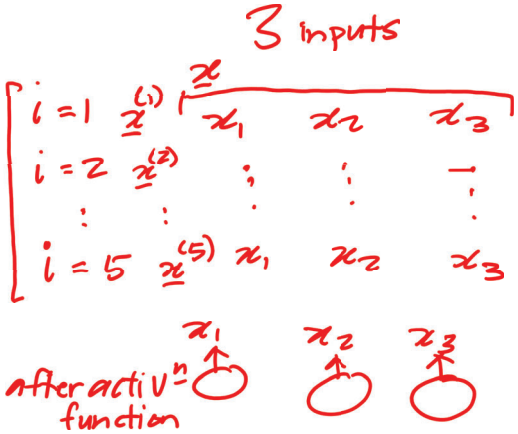
$$1.\quad \mu_B = \frac{1}{m_B}\sum_{i=1}^{m_B}\mathbf{x}^{(i)}$$

$$2.\quad \sigma_B^2 = \frac{1}{m_B}\sum_{i=1}^{m_B}\left(\mathbf{x}^{(i)} - \mu_B\right)^2$$

$$3.\quad \hat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$4.\quad \mathbf{z}^{(i)} = \boldsymbol{\gamma} \otimes \hat{\mathbf{x}}^{(i)} + \boldsymbol{\beta}$$

element wise multiplication



In this algorithm:

- μ_B is the vector of input means, evaluated over the whole mini-batch B (it contains one mean per input).
- σ_B is the vector of input standard deviations, also evaluated over the whole mini-batch (it contains one standard deviation per input).
- m_B is the number of instances in the mini-batch.
- $\hat{\mathbf{x}}^{(i)}$ is the vector of zero-centered and normalized inputs for instance i.

source: Géron, Hands-on ML

Batch Normalization

- γ is the output scale parameter vector for the layer (it contains one scale parameter per input).
- $\otimes$ represents element-wise multiplication (each input is multiplied by its corresponding output scale parameter).
- β is the output shift (offset) parameter vector for the layer (it contains one offset parameter per input). Each input is offset by its corresponding shift parameter.
- ε is a tiny number that avoids division by zero (typically 10^{-5}). This is called a *smoothing term*.
- $\mathbf{z}^{(i)}$ is the output of the BN operation. It is a rescaled and shifted version of the inputs.

During training, BN standardizes its inputs, then rescales and offsets them.
Also, provides regularization effect.

However, determining mean and std deviation over a mini-batch is not straightforward during testing/inference.
→ estimate them using exponential moving average (done in Keras)

$$\bar{\mu}^{(t)} \leftarrow \alpha \mu^{(t)} + (1-\alpha)\bar{\mu}^{(t-1)}$$

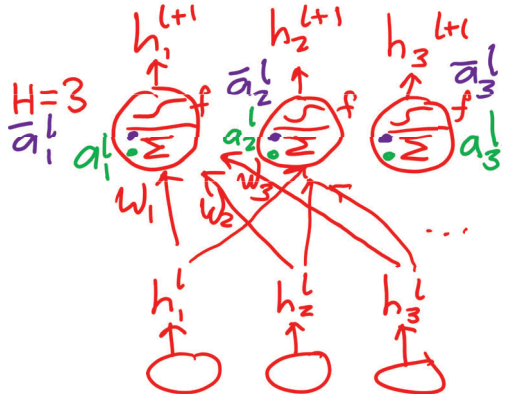
Layer Normalization

- Another form of normalization is Layer Normalization
- LN works better with recurrent neural networks (RNNs), as will be explained later
- This idea was introduced by Jimmy Lei Ba *et al* (2016): it is very similar to Batch Normalization, but instead of normalizing across the batch dimension, it normalizes across the **features** or **inputs (to a layer)** dimension.
- → **per layer** mean and standard deviation

scalar $a_i^l = w_i^{l\top} h^l$

scalar $\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$

scalar $\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$



Layer Normalization

- Layer Normalization also ***learns*** a **scale** and an **offset** parameter

output of LN

scalar $\bar{a}_i^l = \frac{g_i^l}{\sigma_i^l}(a_i^l - \mu_i^l) + b_i^l$

activation function

scalar $h_i^{l+1} = f(\bar{a}_i^l + b_i^l)$

- One advantage is that it can compute the required statistics on the fly, at **each time step, independently for each instance**. This also means that it behaves the same way during training and testing (as opposed to BN), and it does not need to use exponential moving averages to estimate the feature statistics across all instances in the training set.

Regularization

- Deep neural networks typically have tens of thousands of parameters, sometimes even millions.
- This gives them an incredible amount of degrees of freedom and means that they can fit a huge variety of complex datasets, i.e. *low bias*, *high variance*.
- However, this great flexibility also makes the network prone to overfitting the training set.
- Hence, regularization is needed. $C = Loss + \lambda R(w)$
- Recall the cost function in Lecture 3 which has a regularization penalty term, e.g. tries to constrain the values of the connection weights, or sets some of them to zero.
- Batch normalization and Dropout (next slide) are also regularizers

also, LN

Dropout

Srivastava *et al* (2014)

- Fairly simple algorithm: at every training step, every neuron (including the input neurons, but always excluding the output neurons) has a probability *p* of being temporarily "dropped out," meaning it will be entirely ignored during this training step, but it may be active during the next step.
- The hyperparameter *p* is called the *dropout rate*, and it is typically set between 10% and 50%: closer to 20-30% in recurrent neural nets, and closer to 40-50% in convolutional neural networks. After training, neurons do not get dropped anymore.
- The resulting neural network can be seen as an averaging ensemble of smaller neural networks with missing neurons, thus avoiding overfitting.

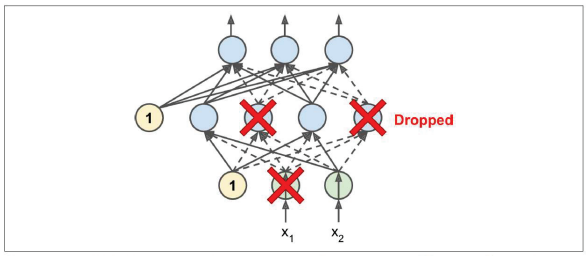


Figure 11-9. With dropout regularization, at each training iteration a random subset of all neurons in one or more layers—except the output layer—are "dropped out"; these neurons output 0 at this iteration (represented by the dashed arrows)

Thank You

Questions?

Assoc Prof Tham Chen Khong

E-mail: eletck@nus.edu.sg