TARAN
ADVISORY IN DATA & ANALYTICS

Data Science 2

April 21, 2024

Data Science 2

Faculty of Mathematics and Physics

Agenda

Neural Networks Introduction

Gradient Based Optimization

Activation functions

Regularization

Convolutional neural networks

Convolutional neural network case studies

Classical approaches to NLP

Sequence models

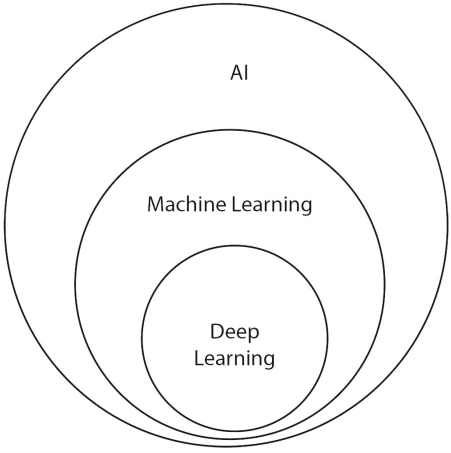
Word embeddings

Large Language Models

Neural Networks Introduction

Introduction

- What is artificial intelligence, machine learning and deep learning?
- And how do they relate?



Introduction
Artificial Intelligence

Definition (Artificial intelligence)

Artificial intelligence is an effort to automate intellectual tasks normally performed by humans.

- AI is a superset of machine learning.
- AI includes also algorithms, that are not based on any learning - you can learn computer to play chess by defining set of rules to be obeyed.
- AI was born around 1950s.
- **Symbolic AI** - at the beginning it was believed that human level AI can be achieved by sufficiently large set of rules.

Introduction
Artificial Intelligence

Definition (Artificial intelligence)

Artificial intelligence is an effort to automate intellectual tasks normally performed by humans.

- AI is a superset of machine learning.
- AI includes also algorithms, that are not based on any learning - you can learn computer to play chess by defining set of rules to be obeyed.
- AI was born around 1950s.
- **Symbolic AI** - at the beginning it was believed that human level AI can be achieved by sufficiently large set of rules.

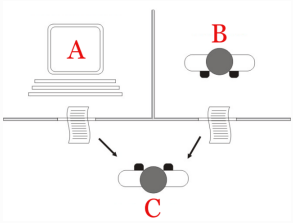
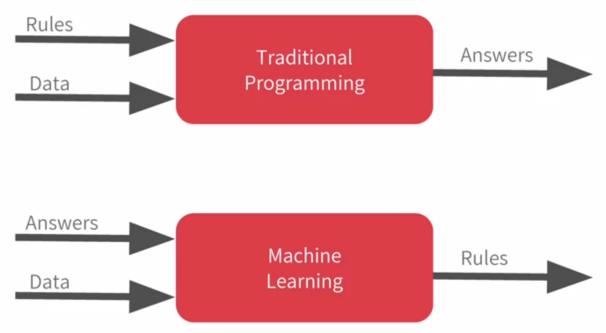


Figure: Turing's test

Definition (Machine Learning)

The use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data.



Introduction
Deep Learning

Definition (Deep learning)

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

Deep learning achievements:

- Image enhancement
- Image generation
- Object detection
- Text translation
- Chatbots
- Speech recognition
- Optical Character Recognition (OCR)
- Super-human level game playing
- ...

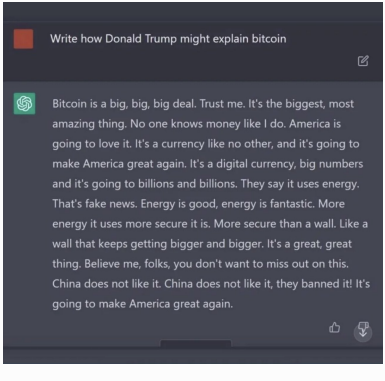
Introduction
Deep Learning

Definition (Deep learning)

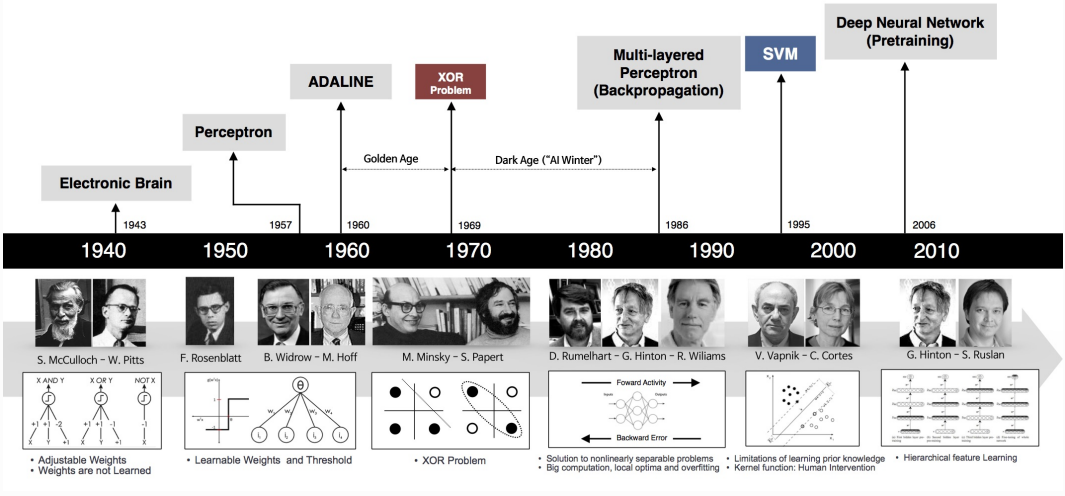
Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

Deep learning achievements:

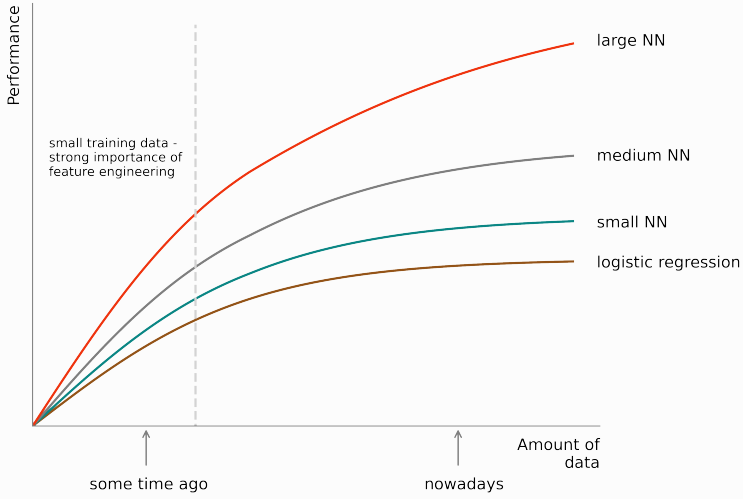
- Image enhancement
- Image generation
- Object detection
- Text translation
- Chatbots
- Speech recognition
- Optical Character Recognition (OCR)
- Super-human level game playing
- ...



Introduction
AI History

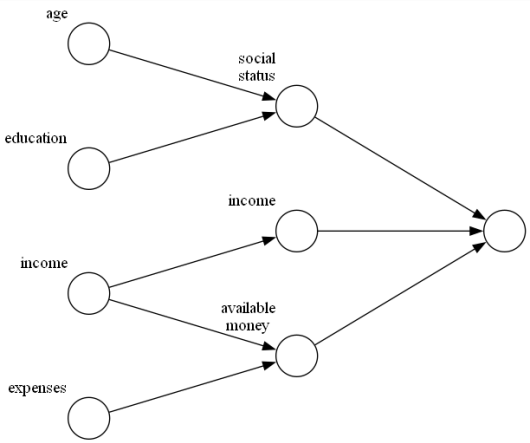


Introduction
Why NN?



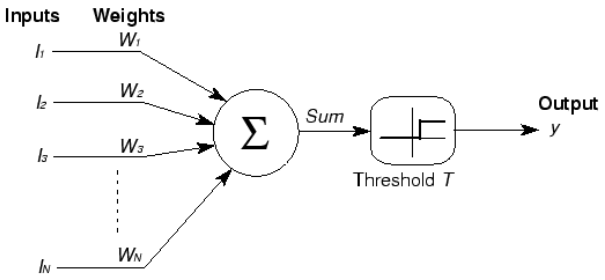
Introduction
Why NN?

Neural Networks do feature engineering automatically.



Neural Networks
Perceptron

In 1958, Frank Rosenblatt introduced an idea of perceptron and called it Mark I Perceptron.

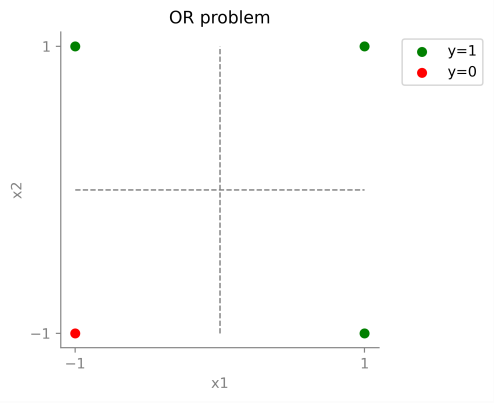


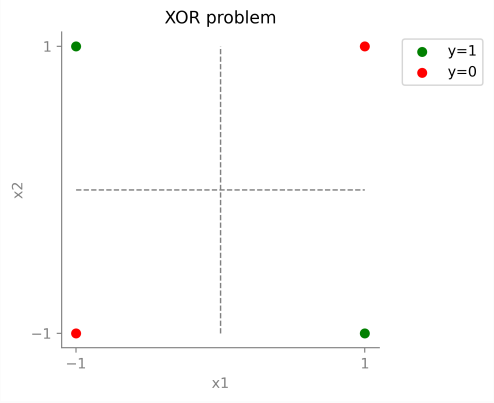
Weights of Mark I Perceptron are 'learnt' through successively passed inputs, while minimizing the difference between desired and actual output.

https://playground.tensorflow.org/

Neural Networks
XOR Problem

Mark I Perceptron is not capable of solving XOR problem.

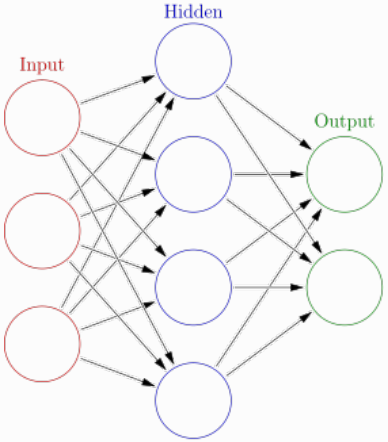




Neural Networks

Multi-layer perceptron

To solve XOR problem, we need to use multi-layer perceptron (MLP) neural network.



Neural Networks
Input data

Input data for neural networks are usually passed in form of multi-dimensional Numpy arrays, also called **tensors**:

- Scalars (0D tensors)
- Vectors (1D tensors)
- Matrices (2D tensors)
- 3D and higher dimensional tensors

Tensor is defined by tree key attributes:

- Number of dimensions
- Shape
- Data type

For tensors, dimension is often called an *axis*. First axis is usually allocated for sample axis. In neural network models data are not processed all at once, but in batches.

Gradient Based Optimization

Gradient based optimization

Each neuron transforms input data using following equation:

$$out = f\left(\sum_{j=1}^{N} w_i x_i + b\right) \tag{1}$$

where

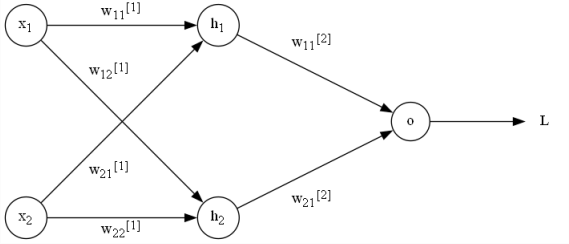
- f ... activation function
- w_i ... weight assigned to a connection coming from i-th input.
- x_i ... value of i-th input
- b ... bias

In multilayer layout weights are represented by weight matrix **W** and biases by bias vector **b**.

Gradient Based Optimization

Backpropagation

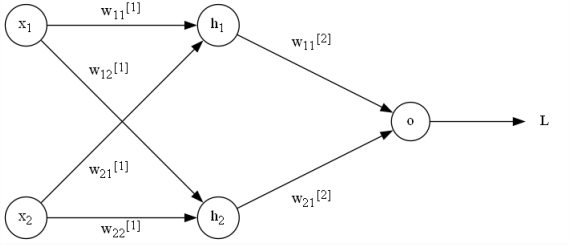
Lets have following network:



Gradient Based Optimization

Backpropagation

Lets have following network:



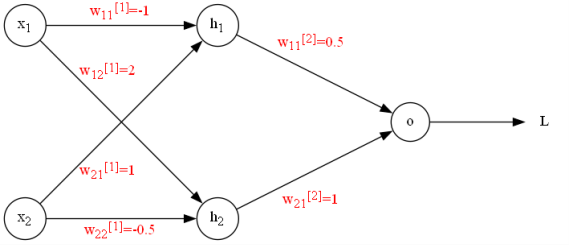
where:

$$
\begin{aligned}
h_1 &= a(w_{11}^{[1]} \cdot x_1 + w_{21}^{[1]} \cdot x_2) \\
h_2 &= a(w_{21}^{[1]} \cdot x_1 + w_{22}^{[1]} \cdot x_2) \\
a(x) &= ReLU(x) = max(0, x) \\
o &= w_{11}^{[2]} \cdot h_1 + w_{21}^{[2]} \cdot h_2 \\
L &= (o - y)^2
\end{aligned}
\tag{2}
$$

Gradient Based Optimization

Backpropagation

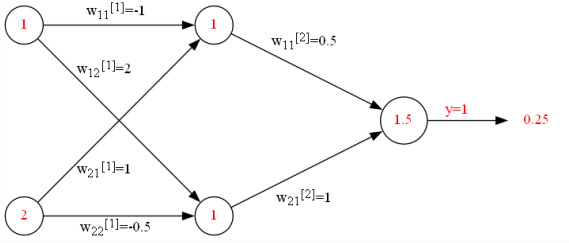
Lets first randomly initialize weights:



Gradient Based Optimization

Backpropagation

Lets first randomly initialize weights:



and calculate forward pass

$$
\begin{aligned}
y &= 1 \\
x_1 &= 1 \\
x_2 &= 2 \\
h_1 &= max(0, -1 \cdot 1 + 1 \cdot 2) = 1 \\
h_2 &= max(0, 1 \cdot 2 + (-0.5) \cdot 2) = 1 \\
o &= 0.5 \cdot 1 + 1 \cdot 1 = 1.5 \\
L &= (1.5 - 1)^2 = 0.25
\end{aligned}
\tag{3}
$$

Gradient Based Optimization

Backpropagation

How do we update weight $w_{11}^{[1]}$? By calculating derivative of loss L with respect to $w_{11}^{[1]}$.

$$\frac{\partial L}{\partial w_{11}^{[1]}} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h_1} \frac{\partial h_1}{\partial w_{11}^{[1]}} \tag{4}$$

Gradient Based Optimization

Backpropagation

How do we update weight $w_{11}^{[1]}$? By calculating derivative of loss L with respect to $w_{11}^{[1]}$.

$$\frac{\partial L}{\partial w_{11}^{[1]}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial w_{11}^{[1]}} \tag{4}$$

$$\begin{aligned}
\frac{\partial L}{\partial o} &= \frac{\partial (o-y)^2}{\partial o} = 2(o-y) = 2(1.5-1) = 1 \\
\frac{\partial o}{\partial h_1} &= \frac{\partial (w_{11}^{[2]} \cdot h_1 + w_{21}^{[2]} \cdot h_2)}{\partial h_1} = w_{11}^{[2]} = 0.5 \\
\frac{\partial h_1}{\partial w_{11}^{[1]}} &= \frac{\partial max(0, w_{11}^{[1]} \cdot x_1 + w_{21}^{[1]} \cdot x_2)}{\partial w_{11}^{[1]}} = (0 \; if \; h_1 <= 0 \; else \; x_1) = 1
\end{aligned} \tag{5}$$

Gradient Based Optimization

Backpropagation

How do we update weight $w_{11}^{[1]}$? By calculating derivative of loss L with respect to $w_{11}^{[1]}$.

$$\frac{\partial L}{\partial w_{11}^{[1]}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial w_{11}^{[1]}} \tag{4}$$

$$\begin{aligned}
\frac{\partial L}{\partial o} &= \frac{\partial (o-y)^2}{\partial o} = 2(o-y) = 2(1.5-1) = 1 \\
\frac{\partial o}{\partial h_1} &= \frac{\partial (w_{11}^{[2]} \cdot h_1 + w_{21}^{[2]} \cdot h_2)}{\partial h_1} = w_{11}^{[2]} = 0.5 \\
\frac{\partial h_1}{\partial w_{11}^{[1]}} &= \frac{\partial max(0, w_{11}^{[1]} \cdot x_1 + w_{21}^{[1]} \cdot x_2)}{\partial w_{11}^{[1]}} = (0 \; if \; h_1 <= 0 \; else \; x_1) = 1
\end{aligned} \tag{5}$$

$$\frac{\partial L}{\partial w_{11}^{[1]}} = 1 \cdot 0.5 \cdot 1 = 0.5$$

Gradient Based Optimization

Backpropagation

The update of $w_{11}^{[1]}$ is given by:

$$w_{11}^{[1]} \longleftarrow w_{11}^{[1]} - \alpha \frac{\partial L}{\partial w_{11}^{[1]}} \tag{6}$$

Gradient Based Optimization

Backpropagation

The update of $w_{11}^{[1]}$ is given by:

$$w_{11}^{[1]} \longleftarrow w_{11}^{[1]} - \alpha \frac{\partial L}{\partial w_{11}^{[1]}} \tag{6}$$

We can do the same for all weights:

$$\frac{\partial L}{\partial w_{11}^{[1]}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial w_{11}^{[1]}} = 0.5$$

$$\frac{\partial L}{\partial w_{12}^{[1]}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_2}\frac{\partial h_2}{\partial w_{12}^{[1]}} = 1$$

$$\frac{\partial L}{\partial w_{21}^{[1]}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial w_{21}^{[1]}} = 1$$

$$\frac{\partial L}{\partial w_{22}^{[1]}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial w_{22}^{[1]}} = 2$$

$$\frac{\partial L}{\partial w_{11}^{[2]}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial w_{11}^{[2]}} = 1$$

$$\frac{\partial L}{\partial w_{21}^{[2]}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial w_{21}^{[2]}} = 1$$

Gradient Based Optimization

Minibatch Optimization

A/ **Gradient Descent**

- Use all observations to calculate $\vec{g}$
- Most accurate
- High computational costs

B/ **Stochastic Gradient Descent**

- Use only one sample at the time for weights update
- Large variance in $\vec{g}$

C/ **Minibatch**

- Use sub-sample of size M to calculate $\vec{g}$
- Usually, M is power of 2

Minibatches are sampled from the data without replacement. Therefore, after $\lceil \#observation/M \rceil$ every observation contributed to one weights update.

Training neural network

Algorithm (NN training)

Repeat until stopping criterion is met:

1. *Sample a minibatch of n observations from training sample.*
2. *Feed forward step - propagate each observation through neural net and calculate outputs*
3. *Calculate derivatives of loss function with respect to trainable parameters*
4. *Backpropagation step - update trainable parameters based on gradient averaged over batch*

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta^{(t)}} L(output_i, label_i) \tag{7}$$

- **Minibatch** - Subsample to be used for one update of weights (forward and backward pass)
- **Epoch** - All the training data was used exactly once to update weights

Gradient descent
SGD with momentum

Algorithm (SGD with momentum)

Repeat until stopping criteria is met:

- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t-1}}), y^{(i)})$
- $\mathbf{v_t} = \beta \mathbf{v_{t-1}} - \alpha \boldsymbol{g_t}$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} + \mathbf{v_t}$

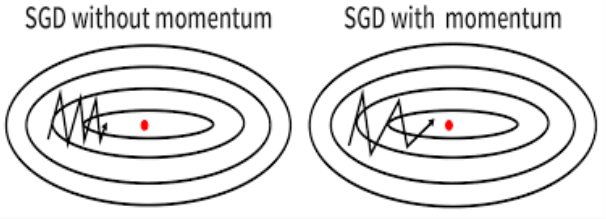
Gradient descent
SGD with momentum

Algorithm (SGD with momentum)

Repeat until stopping criteria is met:

- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t-1}}), y^{(i)})$
- $\mathbf{v_t} = \beta \mathbf{v_{t-1}} - \alpha \boldsymbol{g_t}$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} + \mathbf{v_t}$

- Good value for β is 0.9



Gradient descent
Nesterov accelerated gradient descent

Algorithm (Nesterov accelerated GD)

Repeat until stopping criteria is met:

- $\boldsymbol{\theta_{t_0}} = \boldsymbol{\theta_{t-1}} + \beta \mathbf{v_{t-1}}$
- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t_0}}), y^{(i)})$
- $\mathbf{v_t} = \beta \mathbf{v_{t-1}} - \alpha \boldsymbol{g_t}$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t_0}} - \alpha \boldsymbol{g_t}$

Gradient descent

Nesterov accelerated gradient descent

Algorithm (Nesterov accelerated GD)

Repeat until stopping criteria is met:

- $\boldsymbol{\theta_{t_0}} = \boldsymbol{\theta_{t-1}} + \beta \mathbf{v_{t-1}}$
- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t_0}}), y^{(i)})$
- $\mathbf{v_t} = \beta \mathbf{v_{t-1}} - \alpha \boldsymbol{g_t}$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t_0}} - \alpha \boldsymbol{g_t}$

- We know we will use momentum, so we can calculate gradient after applying momentum, rather than before.

Gradient descent
Adagrad

Adagrad = Adaptive gradient algorithm

Algorithm (Adagrad)

Repeat until stopping criteria is met:

- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t-1}}), y^{(i)})$
- $\mathbf{r_t} = \mathbf{r_{t-1}} + \boldsymbol{g_t}^2$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\alpha}{\sqrt{\mathbf{r_t} + \varepsilon}} \mathbf{g_t}$

Gradient descent
Adagrad

Adagrad = Adaptive gradient algorithm

Algorithm (Adagrad)

Repeat until stopping criteria is met:

- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t-1}}), y^{(i)})$
- $\mathbf{r_t} = \mathbf{r_{t-1}} + \boldsymbol{g_t}^2$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\alpha}{\sqrt{\mathbf{r_t} + \varepsilon}} \mathbf{g_t}$

- Increase learning rate for parameter related to infrequent features
- Decrease learning rate for parameter related to frequent features
- Usually $\varepsilon = 10^{-8}$

Gradient descent
RMSProp

RMSProp = Root mean square propagation

Algorithm (RMSProp)

Repeat until stopping criteria is met:

- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t-1}}), y^{(i)})$
- $\mathbf{r_t} = \beta \mathbf{r_{t-1}} + (1 - \beta) \boldsymbol{g_t}^2$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\alpha}{\sqrt{\mathbf{r_t} + \varepsilon}} \mathbf{g_t}$

Gradient descent
RMSProp

RMSProp = Root mean square propagation

Algorithm (RMSProp)

Repeat until stopping criteria is met:

- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t-1}}), y^{(i)})$
- $\mathbf{r_t} = \beta \mathbf{r_{t-1}} + (1-\beta)\boldsymbol{g_t}^2$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\alpha}{\sqrt{\mathbf{r_t}+\varepsilon}}\mathbf{g_t}$

- suppress old gradients and put more weight to new one (in terms of adaptivity)
- Usually $\varepsilon = 10^{-8}$
- Usually $\beta = 0.9$

Gradient descent
Adam

Adam = Adaptive Moment Estimation

Algorithm (Adam)

Set $\mathbf{s_0} = \mathbf{0}$, $\mathbf{r_0} = \mathbf{0}$
Repeat until stopping criteria is met:

- *Randomly select m samples from training data.*
- $\mathbf{g_t} = \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta_{t-1}}), y^{(i)})$
- $\mathbf{s_t} = \beta_1 \mathbf{s_{t-1}} + (1 - \beta_1) \boldsymbol{g_t}$
- $\mathbf{r_t} = \beta_2 \mathbf{r_{t-1}} + (1 - \beta_2) \boldsymbol{g_t}^2$
- $\hat{\mathbf{s}}_\mathbf{t} = \frac{\mathbf{s_t}}{1-\beta_1^t}$
- $\hat{\mathbf{r}}_\mathbf{t} = \frac{\mathbf{r_t}}{1-\beta_2^t}$
- $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\alpha}{\sqrt{\hat{\mathbf{r}}_\mathbf{t}} + \varepsilon} \hat{\mathbf{s}}_\mathbf{t}$

- Default values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\alpha = 0.001$
- Usually $\varepsilon = 10^{-8}$

Gradient Descent
Algorithms Overview

- SGD with Nesterov Momentum
  - momentum
- AdaGrad (Adaptive Gradient Algorithm)
  - adaptive learning rate
- RMSProp (Root Mean Square Propagation)
  - adaptive learning rate
- Adam (Adaptive Moment Estimation)
  - momentum
  - adaptive learning rate

https://keras.io/api/optimizers/

Learning rate schedules

Even for algorithms with adaptive learning rates, further fine-tuning of learning rate can improve performance. Decreasing learning rate can be performed each batch/epoch/several epochs.

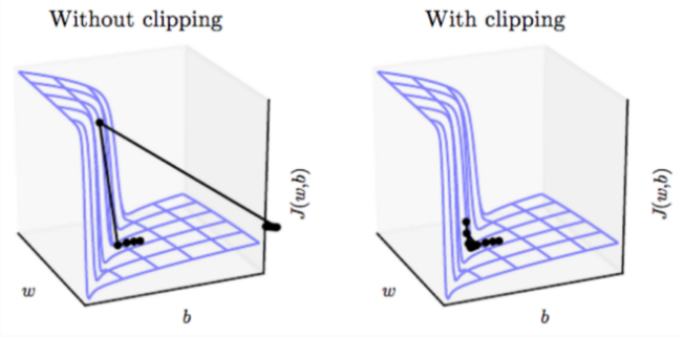
- Exponential decay
  $eta_t = \eta_0 \cdot c^t$
- Polynomial decay
  - Inverse time decay
    $\eta_t = \eta_0 \cdot \frac{1}{t}$
  - Inverse-square decay
    $\eta_t = \eta_0 \cdot \frac{1}{\sqrt{t}}$
- Cosine decay, restarts, ...

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/

Gradient clipping

Algorithm (Gradient clipping)

- $\mathbf{g} = \nabla_\theta L(f(\mathbf{x}; \theta), y)$
- *if* $\|\mathbf{g}\| \geq$ *treshold then* $\mathbf{g} =$ **treshold** *end if*



Activation functions

Activation functions

Each neuron transforms input data using following equation:

$$out = f\left(\sum_{j=1}^{N} w_i x_i + b\right) \tag{8}$$

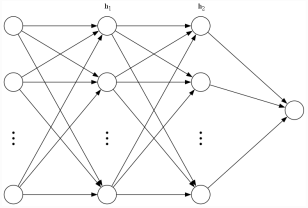
- Activation function adds non-linearity at each neuron
- It helps to control neuron output values range
- Desired features:
  - Differentiability (required)
  - Know analytical solution of first derivative
  - Low computational expense

https://www.tensorflow.org/api_docs/python/tf/keras/activations

Activation Functions

Why Use Non-linear Activation

Let us consider NN with two hidden layers (h_1, h_2), both with no activation function applied.



$$\mathbf{h_1} = \mathbf{W^{[1]}} \cdot \mathbf{x} + \mathbf{b^{[1]}}$$

$$\mathbf{h_2} = \mathbf{W^{[2]}} \cdot \mathbf{h_1} + \mathbf{b^{[2]}} = \mathbf{W^{[2]}} \cdot \left(\mathbf{W^{[1]}} \cdot \mathbf{x} + \mathbf{b^{[1]}}\right) + \mathbf{b^{[2]}} =$$

$$= \underbrace{\mathbf{W^{[2]}} \cdot \mathbf{W^{[1]}}}_{\mathbf{W'}} \cdot \mathbf{x} + \underbrace{\mathbf{W^{[2]}} \cdot \mathbf{b^{[1]}} + \mathbf{b^{[2]}}}_{\mathbf{b'}} \tag{9}$$

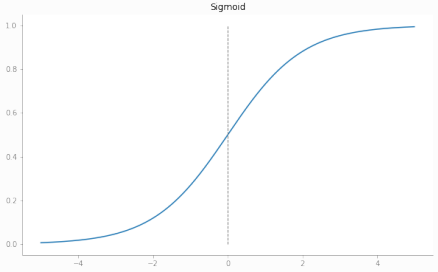
we do not need to have second hidden layer

Activation functions

Sigmoid

Definition:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{10}$$



Derivative:

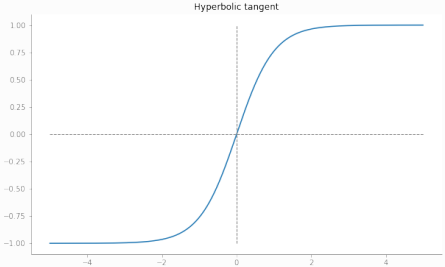
$$\frac{\mathrm{d}\sigma(x)}{\mathrm{d}x} = \sigma(x)(1 - \sigma(x)) \tag{11}$$

Activation functions

Hyperbolic tangent

Definition:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{12}$$



Derivative:

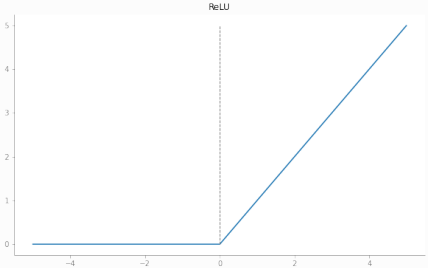
$$\frac{\mathrm{d}\tanh(x)}{\mathrm{d}x} = 1 - \tanh(x)^2 \tag{13}$$

Activation functions

Rectified Linear Unit

Definition:

$$\mathrm{ReLu}(x) = \max(0, x) \tag{14}$$



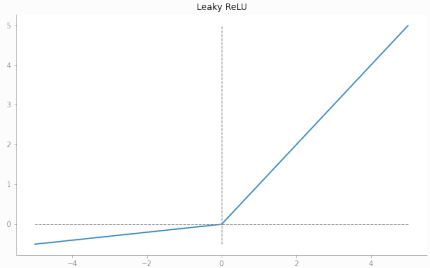
Derivative:

$$\frac{\mathrm{dReLu}(x)}{\mathrm{d}x} = \begin{cases} 1 & \text{if } x > 0 \\ NaN & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{15}$$

Activation functions
Leaky ReLU

Definition:

$$\mathrm{LeakyReLU}(x) = \max(\alpha x, x) \tag{16}$$



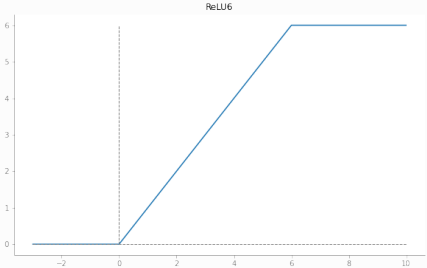
Derivative:

$$\frac{\mathrm{dLeakyReLU}(x)}{\mathrm{d}x} = \begin{cases} 1 & \text{if } x > 0 \\ NaN & \text{if } x = 0 \\ \alpha & \text{if } x < 0 \end{cases} \tag{17}$$

Activation functions
ReLU6

Definition:

$$\mathrm{ReLu6}(x) = \min(\max(0, x), 6) \tag{18}$$



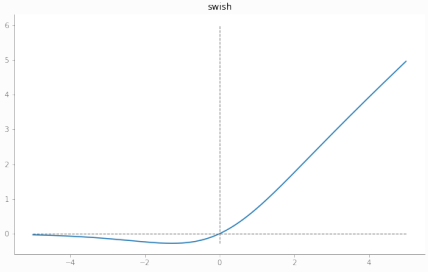
Derivative:

$$\frac{\mathrm{dReLu6}(x)}{\mathrm{d}x} = \begin{cases} 1 & \text{if } x \in (0, 6) \\ NaN & \text{if } x = 0 \text{ or } x = 6 \\ 0 & \text{if } x < 0 \text{ or } x > 6 \end{cases} \tag{19}$$

Activation functions
Swish

Definition:

$$\text{swish}(x) = x\sigma(\beta x) \tag{20}$$



Derivative:

$$\frac{\text{dswish}(x)}{\text{d}x} = \text{swish}(x) + \sigma(x)(\beta - \text{swish}(x)) \tag{21}$$

Activation functions

Activation on output layer

Common activations on output layer:

- None - linear regression if there are no hidden layers
- $\sigma(x)$ - binary classification

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{22}$$

- $\mathrm{softmax}$ - multi-class classification

$$\mathrm{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{23}$$

Vanishing and exploding gradient

- Vanishing gradient is a consequence of chain rule used in backpropagation algorithm.
- Activation functions such as sigmoid or hyperbolic tangent has derivative in interval $(0, 1)$
- With inappropriate set up derivatives of weights in former layers might **vanish** because of multiplying values lower then 1 in magnitude many times
- If, for instance, the initial loss is big, the gradient might **explode**

Vanishing and exploding gradient

- Vanishing gradient is a consequence of chain rule used in backpropagation algorithm.
- Activation functions such as sigmoid or hyperbolic tangent has derivative in interval $(0, 1)$
- With inappropriate set up derivatives of weights in former layers might **vanish** because of multiplying values lower then 1 in magnitude many times
- If, for instance, the initial loss is big, the gradient might **explode**

Symptoms of vanishing/exploding gradient:

Vanishing gradient

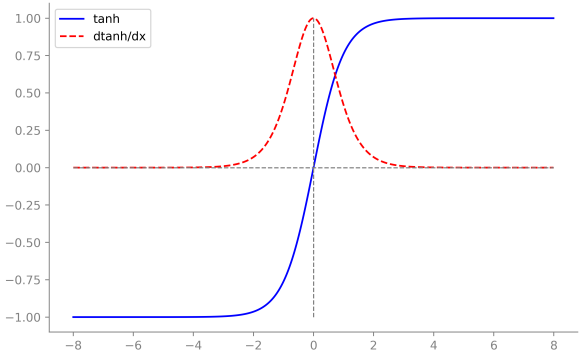
- Model improves very slowly
- Weights closer to output layer changes more dynamically

Exploding gradient

- Large changes in loss
- Model loss is NaN
- Model weights grow exponentially
- Model weights are NaN

Saturating non-linearities

With large input to neuron, derivative of saturating activation functions is close to 0



Weights Initialization

- In general practice biases are initialized with 0.
- Initialization of weights by 0 (or any constant) is not a good idea, since neurons' activation in one layer will never break the symmetry.

Weights Initialization

- In general practice biases are initialized with 0.
- Initialization of weights by 0 (or any constant) is not a good idea, since neurons' activation in one layer will never break the symmetry.

  Let us consider two neurons in hidden layer and weights in network initialized by a constant.

$$h_1 = f\left(\sum_i w_{i1} x_i + b\right)$$
$$h_2 = f\left(\sum_i w_{i2} x_i + b\right) = h_1 \tag{24}$$

Weights Initialization

- In general practice biases are initialized with 0.
- Initialization of weights by 0 (or any constant) is not a good idea, since neurons' activation in one layer will never break the symmetry.

  Let us consider two neurons in hidden layer and weights in network initialized by a constant.

$$\begin{aligned} h_1 &= f\left(\sum_i w_{i1}x_i + b\right) \\ h_2 &= f\left(\sum_i w_{i2}x_i + b\right) = h_1 \end{aligned} \tag{24}$$

$$\frac{\partial h_1}{\partial w_{i1}} = x_i f'(y)|_{y=\sum_i w_{i1}x_i+b} = x_i f'(y)|_{y=\sum_i w_{i2}x_i+b} = \frac{\partial h_2}{\partial w_{i2}} \tag{25}$$

  Weights of connections from one input neuron to hidden neurons of the following layer will be updated by the same amount => activation on following layer neurons remains the same. Neurons in one layer cannot learn different "features".
  https://www.deeplearning.ai/ai-notes/initialization/index.html

Weights initialization

- In general practice biases are initialized with 0.
- Initialization of weights by 0 is not a good idea, since such weights will never break the symmetry inside one layer.
- Glorot initialization is trying to fix the variance of activations as well as gradients across layers:

$$W_{Glorot} \sim U\left[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right] \tag{26}$$

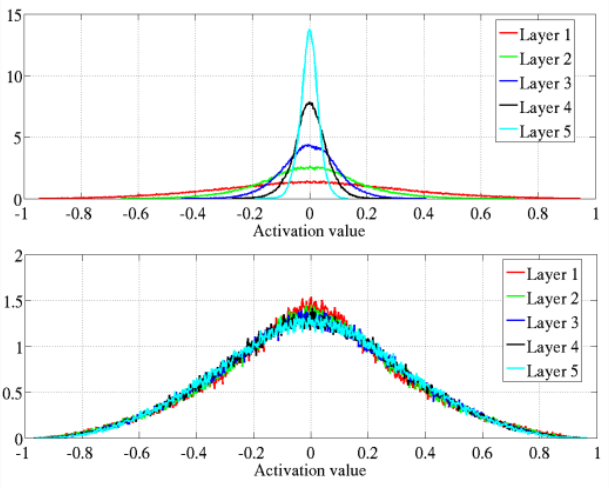
  where n is number of neurons in previous layer and m is a number of neurons in next layer.
- For ReLU activation, HeUniform might be more suitable:

$$W_{He} \sim U\left[-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}\right] \tag{27}$$

  where n is a number of input units.

https://www.tensorflow.org/api_docs/python/tf/keras/initializers/

Weights initialization



Source: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

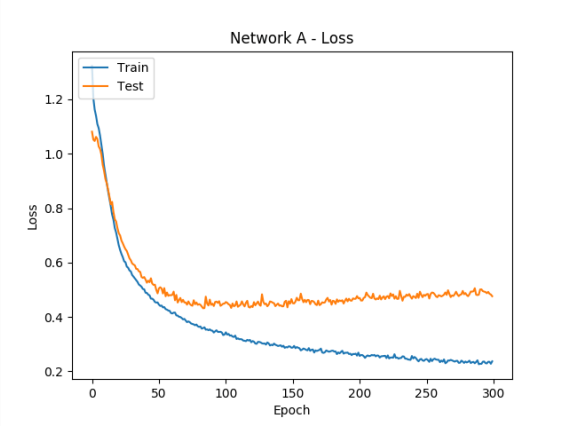
Regularization

Regularization

Regularization allows to control generalization error of model.

- Early stopping
- L2, L1 regularization
- Dataset augmentation
- Ensembling
- Dropout
- Label smoothing

Regularization
Early stopping



- Loss on train set will always decrease
- Stop training once loss on test set stop decreasing

Regularization

L2, L1 regularization

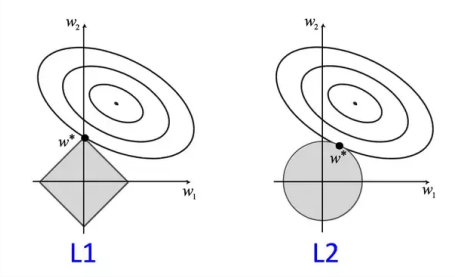
Penalization of large weights. Distribute the model strength amongst more ways in neural net, rather than emphasize one specific way.

- L2 regularization

$$L_{L2}(\boldsymbol{\theta}; \mathbf{X}) = L(\boldsymbol{\theta}; \mathbf{X}) + \lambda \|\boldsymbol{\theta}\|_2^2 \tag{28}$$

- L1 regularization

$$L_{L1}(\boldsymbol{\theta}; \mathbf{X}) = L(\boldsymbol{\theta}; \mathbf{X}) + \lambda \|\boldsymbol{\theta}\|_1 \tag{29}$$



Regularization
Data augmentation

Modifying input data can force model to focus on general patterns rather than focusing on details, which can lead to overfitting.

- Image processing
  - Translation
  - Horizontal flips
  - Scaling
  - Rotation
  - Colour adjustment
  - Mixup - two images are combined using their weighted combination
- Speech recognition
  - Additional noise
  - Frequency change

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing

Regularization
Ensembling

Model ensembling is a technique to reduce generalization error by combining several models. Usually, model outputs are averaged.

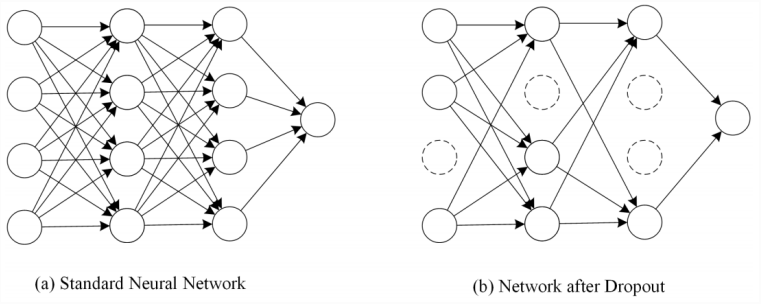
Possible set ups:

- Generate random training samples using sampling with replacement.
- Use different random weight initialization.
- Average model obtained at different time steps (one hour of training, one day of training, ...)

Ensembling often has high computational requirements.

Regularization
Dropout

- Dropout is applied to a layer.
- Each neuron of the layer is randomly excluded with certain probability in each batch.
- Technically, dropout is performed by setting value of the neuron to 0.



(a) Standard Neural Network

(b) Network after Dropout

Regularization
Dropout

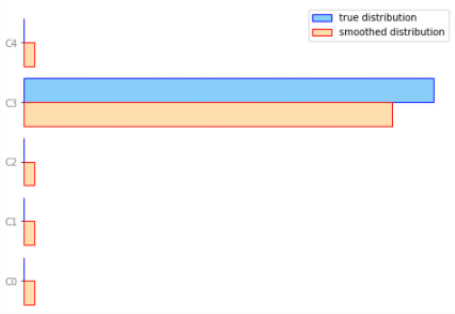
During inference, no neurons are dropped out. This means that neuron receives signal from more neurons than during training. Scaling should be performed:

- Reduce the activation signal during inference by the factor of $1 - p$, where p is a probability of dropout from previous layer.
- Enhance the activation signal during training by the factor of $\frac{1}{1-p}$

Regularization
Label smoothing

In multi-class classification problem, when using softmax, the model is never satisfied with achieved results. If model prediction for the true category is 0.999, it is still trying to improve.
To overcome this, true label distribution is modified from $\mathbf{1}_{true}$ to

$$(1-\alpha)\mathbf{1}_{true} + \alpha\frac{1}{\text{number of classes}}$$



Convolutional neural networks

TARAN

ADVISORY IN DATA & ANALYTICS

Convolutional Neural Networks

Motivation

Convolutional neural networks bring three main advantages:

Convolutional Neural Networks
Motivation

Convolutional neural networks bring three main advantages:

- Local interactions
  - Neighbouring pixels in picture might share an information rather than those far from each other.
  - Neighbouring words in a sentence might have a special meaning.

Convolutional Neural Networks
Motivation

Convolutional neural networks bring three main advantages:

- Local interactions
  - Neighbouring pixels in picture might share an information rather than those far from each other.
  - Neighbouring words in a sentence might have a special meaning.
- Parameter sharing
  - This concept allow to reduce number of trainable parameters.

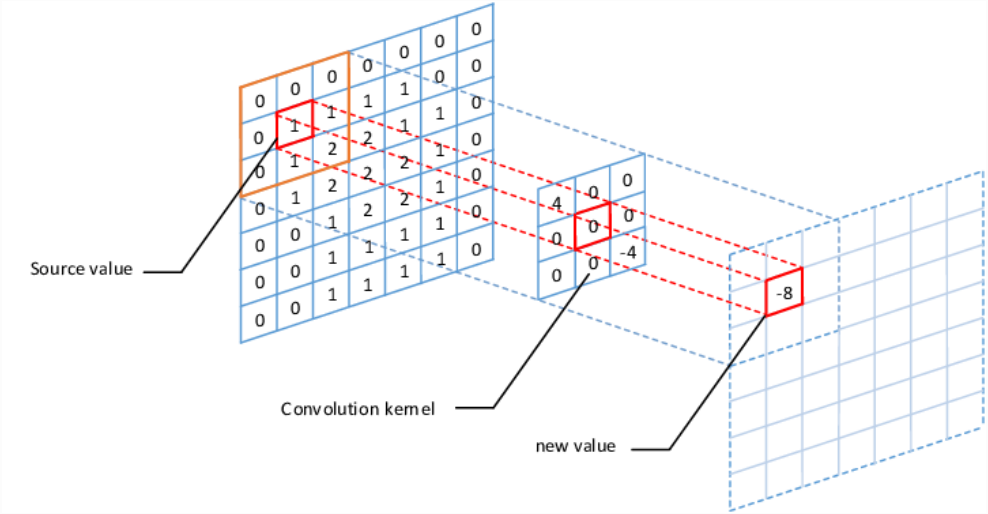
Convolutional Neural Networks
Motivation

Convolutional neural networks bring three main advantages:

- Local interactions
  - Neighbouring pixels in picture might share an information rather than those far from each other.
  - Neighbouring words in a sentence might have a special meaning.
- Parameter sharing
  - This concept allow to reduce number of trainable parameters.
- Shift invariance
  - A pattern should be recognized independently on its position.

Convolutional Neural Networks

Convolution process



Convolutional Neural Networks

Convolution operation

Convolution of two functions is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) \mathrm{d}\tau \tag{30}$$

Convolutional Neural Networks

Convolution operation

Convolution of two functions is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) \mathrm{d}\tau \tag{30}$$

In 2D it takes form:

$$(f * g)(t, u) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\tau, \xi) g(t - \tau, u - \xi) \mathrm{d}\tau \mathrm{d}\xi \tag{31}$$

Convolutional Neural Networks

Convolution operation

Convolution of two functions is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) \mathrm{d}\tau \tag{30}$$

In 2D it takes form:

$$(f * g)(t, u) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\tau, \xi) g(t - \tau, u - \xi) \mathrm{d}\tau \mathrm{d}\xi \tag{31}$$

In discrete domain we have:

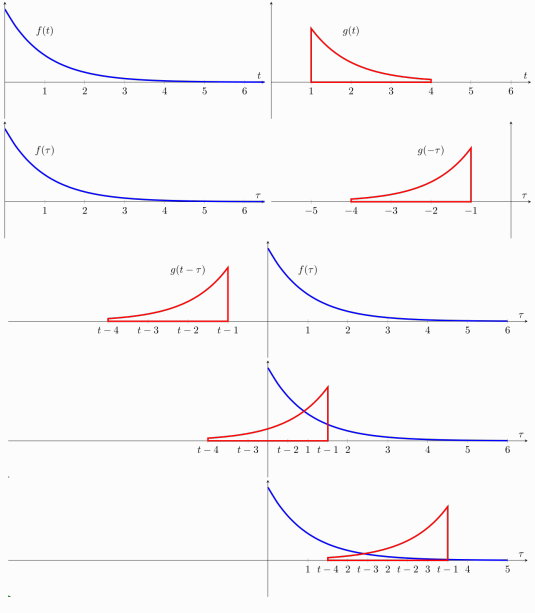
$$(\mathbf{w} * \mathbf{x})_t = \sum_i w_i x_{t-i} \tag{32}$$

Or for 2D case:

$$(\mathbf{K} * \mathbf{I})_{i,j} = \sum_{m,n} \mathbf{K}_{m,n} \mathbf{I}_{i-m,j-n} \tag{33}$$

Convolutional Neural Networks

Convolution of two functions



Convolutional Neural Networks
Cross-correlation

CNN actually uses cross-correlation instead of convolution:

$$(\mathbf{K} \star \mathbf{I})_{i,j} = \sum_{m,n} \mathbf{K}_{m,n} \mathbf{I}_{i+m,j+n} \tag{34}$$

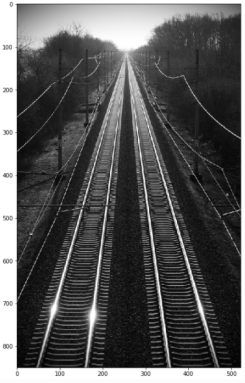
Convolutional Neural Networks

Cross-correlation

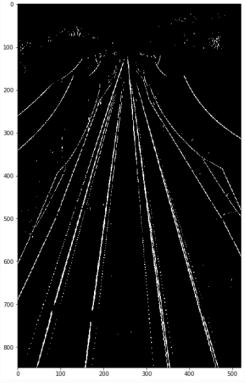
CNN actually uses cross-correlation instead of convolution:

$$(\mathbf{K} \star \mathbf{I})_{i,j} = \sum_{m,n} \mathbf{K}_{m,n} \mathbf{I}_{i+m,j+n} \tag{34}$$

Cross-correlation can be used to detect edges in image. Edge is a space with big differences in pixel values.



-1	0	+1
-2	0	+2
-1	0	+1



Convolutional Neural Networks
Channels

Matrix **K** is called **kernel** or **filter**.

Coloured image consists of three **channels** - RGB. The weights inside the kernel are different for each channel. Thus, we have

$$(\mathbf{K} \star \mathbf{I})_{i,j} = \sum_{m,n,c} \mathbf{K}_{m,n,c} \mathbf{I}_{i+m,j+n,c} \tag{35}$$

Also we want to use more than one kernels - one kernel can detect vertical lines, another horizontal lines and yet another can detect rounded objects. Neural net will decide on its own what shapes are relevant, but we need to provide kernels, whose weights can be trained.

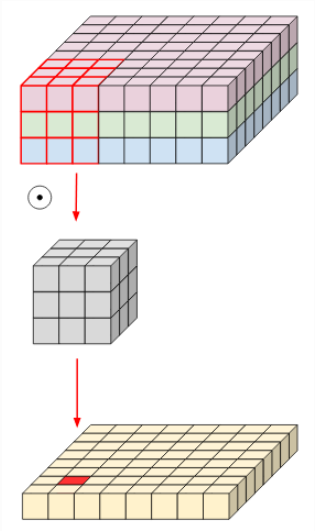
$$(\mathbf{K} \star \mathbf{I})_{i,j,o} = \sum_{m,n,c} \mathbf{K}_{m,n,c,o} \mathbf{I}_{i+m,j+n,c} \tag{36}$$

Kernel is four dimensional tensor with dimensions:

- W ... kernel width
- H ... kernel height
- C ... input channels
- F ... output channels (number of filters we want to train)

Convolutional Neural Networks

Processing multiple input channels



Convolutional Neural Networks

Padding

Padding scheme decides how to behave at the image edges.

- valid - Only use valid pixels. The kernel cannot overhang the image. Results in output being smaller than the input.
- same - Pad edges of the image with zeros so that resulting output is the same size as input.

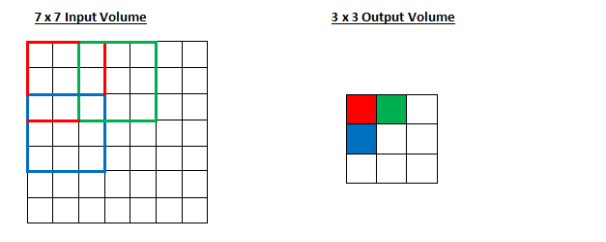
0	0	0	0	0	0	0	0
0	**3**	**3**	**4**	**4**	**7**	**0**	0
0	**9**	**7**	**6**	**5**	**8**	**2**	0
0	**6**	**5**	**5**	**6**	**9**	**2**	0
0	**7**	**1**	**3**	**2**	**7**	**8**	0
0	**0**	**3**	**7**	**1**	**8**	**3**	0
0	**4**	**0**	**4**	**3**	**2**	**2**	0
0	0	0	0	0	0	0	0

Convolutional Neural Networks

Stride

Stride defines kernel step when computing cross-correlation.

$$(\mathbf{K} \star \mathbf{I})_{i,j,o} = \sum_{m,n,c} \mathbf{K}_{m,n,c,o} \mathbf{I}_{i \cdot S+m, j \cdot S+n, c} \tag{37}$$

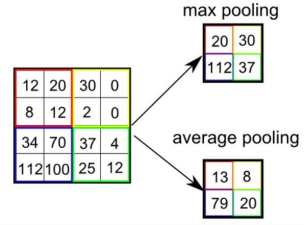


Applying stride reduces size of the output. If $S = 2$, then output is half the size of input.

Convolutional Neural Networks
Pooling

Pooling is an operation that reduces width and height of the image. It is fixed operation - it does not include trainable parameters.

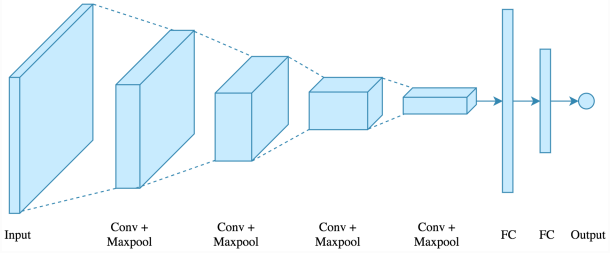
- Max pooling
- Average pooling



Reducing size of the image allows to detect high level patterns in the picture

Convolutional Neural Networks
CNN architecture

- Double number of channels when performing pooling.
- Some architectures do not use fully connected layers at the end of the network.



Convolutional Neural Networks

Batch normalization

Let us first assume multi layer perceptron neural network. As the weights of the network updates a neuron inside the layer receives different distribution of inputs. This effect is called **internal covariate shift**.

Batch normalization is trying to normalize inputs at each neuron (by normalizing outputs of previous layer). Since we are using minibatch SGD, the inputs are normalized within batches as well.

Empirical mean and variance of the batch B of size m is given by

$$\begin{aligned} \mu_B &= \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_B^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \end{aligned} \tag{38}$$

Convolutional Neural Networks

Batch normalization

Batch normalization layer takes the output of each neuron of previous layer $h_i^{(k)}$ (i denotes a layer and k denotes a neuron) and transforms it in two steps:

1. Normalization

$$\hat{h}_i^{(k)} = \frac{h_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \varepsilon}} \tag{39}$$

2. Shift and scale

$$x_i^k = \gamma^k \hat{h}_i^{(k)} + \beta^{(k)} \tag{40}$$

γ^k and $\beta^{(k)}$ are trainable parameters.

During inference μ and σ are calculated as a population statistics.

For convolutional networks we don't want to break its properties, mainly the shift invariance. Therefore we perform batch normalization across minibatch and spacial/temporal dimensions.

Convolutional neural network case studies

TARAN

ADVISORY IN DATA & ANALYTICS

Case studies
Networks overview

Motivation:

- Existing networks can help you to select proper architecture and hyperparameters

Case studies
Networks overview

Motivation:

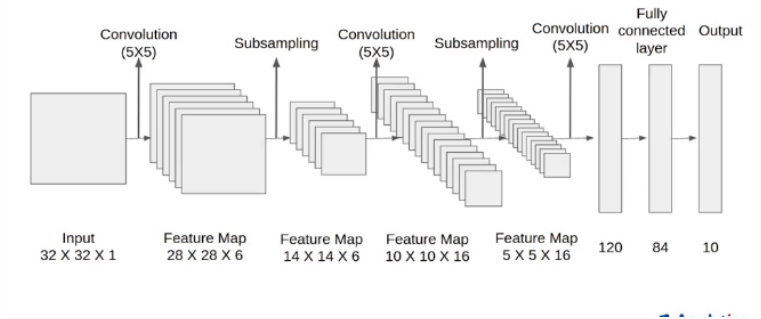
- Existing networks can help you to select proper architecture and hyperparameters

Classic networks:

- LeNet-5
- AlexNet
- VGG-16
- ResNet
- Inception

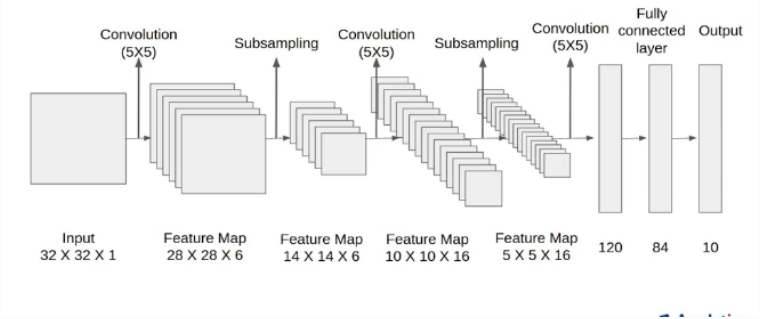
Case studies
LeNet-5

Developed in 1989 by Yann LeCun to classify hand-written digits.



Case studies
LeNet-5

Developed in 1989 by Yann LeCun to classify hand-written digits.

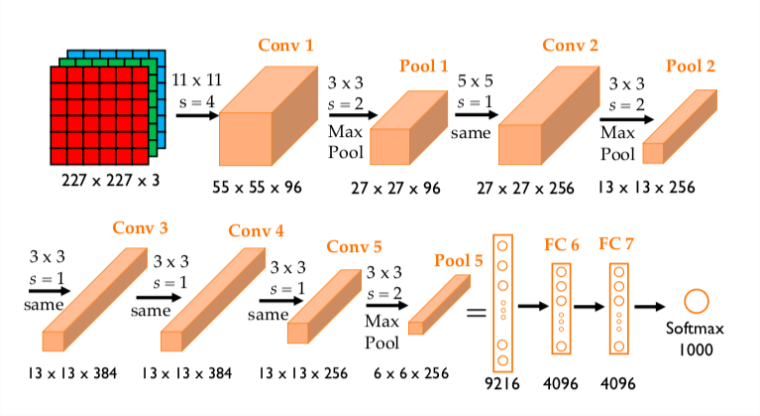


- About 60K trainable parameters

paper: Gradient-Based Learning Applied to Document Recognition (Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner)

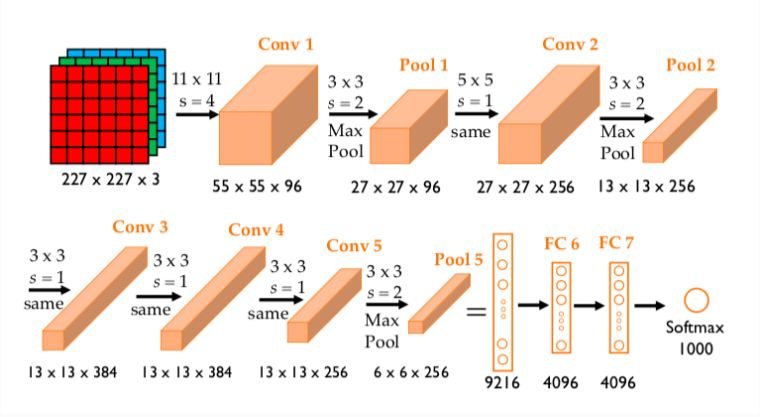
Case studies
AlexNet

Developed in 2012 to compete in the ImageNet Large Scale Visual Recognition Challenge (image classification of 1000 categories).



Case studies
AlexNet

Developed in 2012 to compete in the ImageNet Large Scale Visual Recognition Challenge (image classification of 1000 categories).

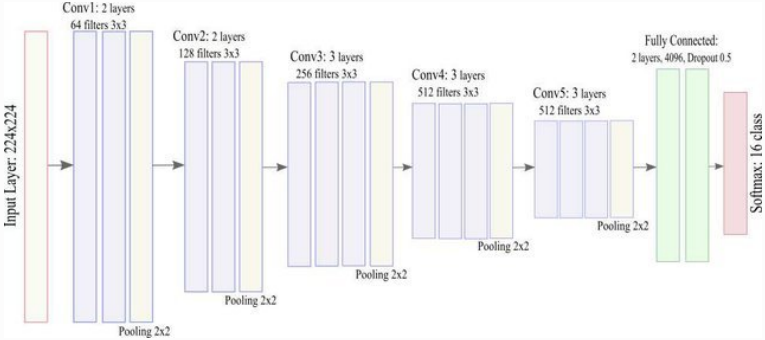


- About 60M trainable parameters

paper: ImageNet Classification with Deep Convolutional Neural Networks (Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton)

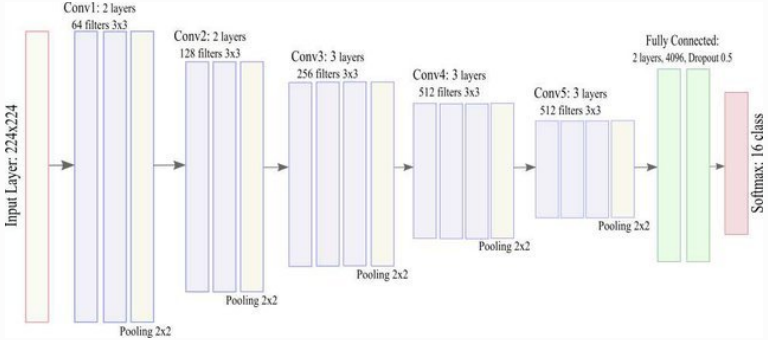
Case studies
VGG-16

Developed in 2014.



Case studies
VGG-16

Developed in 2014.



- About 138M trainable parameters

paper: Very deep convolutional networks for large-scale image recognition (Karen Simonyan, Andrew Zisserman)

Case studies
ResNet

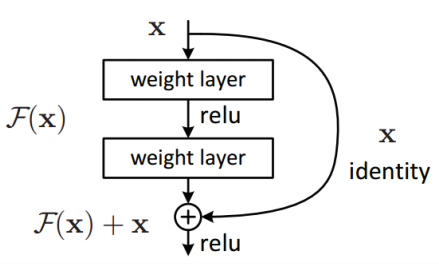
Residual neural networks utilizes skip connections.

- Residual networks allows to train deep networks.
- Skip connections help with vanishing gradient problem.

Case studies
ResNet

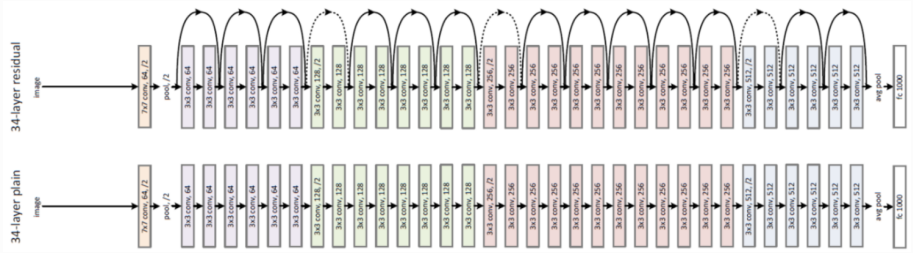
Residual neural networks utilizes skip connections.

- Residual networks allows to train deep networks.
- Skip connections help with vanishing gradient problem.



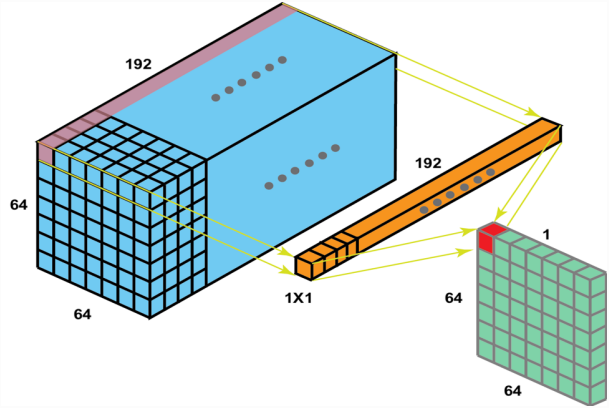
paper: Deep Residual Learning for Image Recognition (Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun)

Case studies
Residual connections

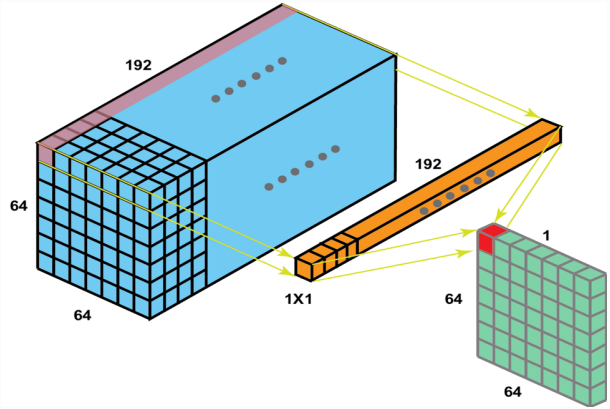


Case studies

1x1 convolution



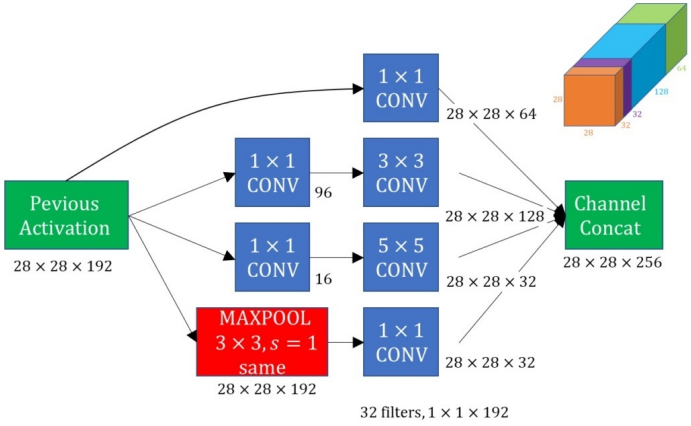
Case studies
1x1 convolution



- Can be used to shrink channels
- Can add additional non-linearity (keeping channel dimensionality)
- Can reduce computational costs

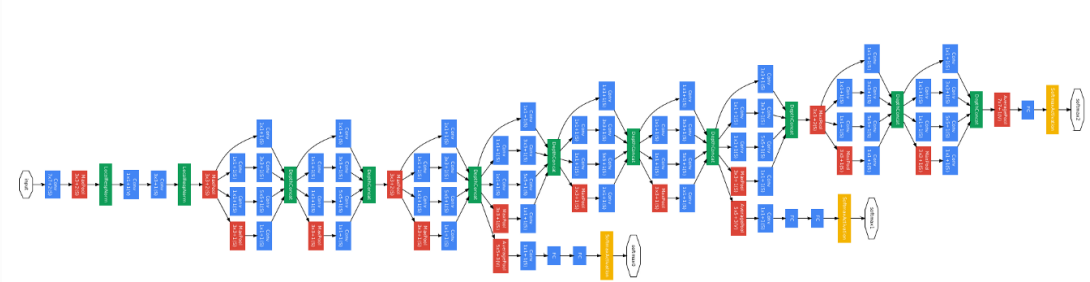
Case studies
Inception module

An example of an Inception module



Case studies
Inception network

Developed in 2014.



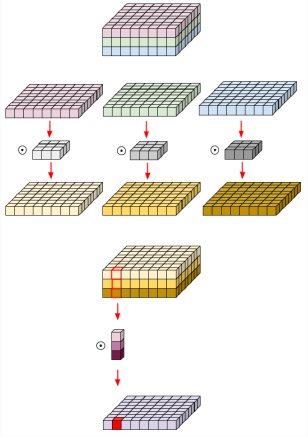
paper: Going Deeper with Convolutions (Szegedy et al.)

Case studies

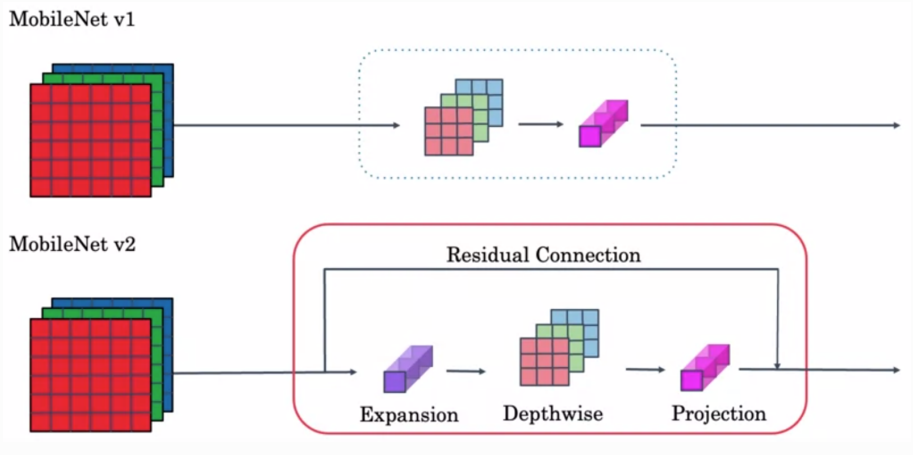
Depth-wise separable convolution

This convolution originated from the idea that depth and spatial dimension of a filter can be separated. For instance Sobel filter for edge detection can be written as vector multiplication.

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{41}$$



Case studies
MobileNet



Convolutional Neural Networks

Transfer learning

It might be useful to utilize network trained on different dataset.

- First layers of pre-trained network capture low level features such as horizontal lines and vertical lines.
- Pre-trained network harvested large dataset.
- Using pre-trained network can save lot of computational costs.

Convolutional Neural Networks

Transfer learning

It might be useful to utilize network trained on different dataset.

- First layers of pre-trained network capture low level features such as horizontal lines and vertical lines.
- Pre-trained network harvested large dataset.
- Using pre-trained network can save lot of computational costs.

We can:

- Retrain only classification layer (low amount of data).

Convolutional Neural Networks

Transfer learning

It might be useful to utilize network trained on different dataset.

- First layers of pre-trained network capture low level features such as horizontal lines and vertical lines.
- Pre-trained network harvested large dataset.
- Using pre-trained network can save lot of computational costs.

We can:

- Retrain only classification layer (low amount of data).
- Unfreeze last n layers of pre-trained network (larger dataset).

Convolutional Neural Networks

Transfer learning

It might be useful to utilize network trained on different dataset.

- First layers of pre-trained network capture low level features such as horizontal lines and vertical lines.
- Pre-trained network harvested large dataset.
- Using pre-trained network can save lot of computational costs.

We can:

- Retrain only classification layer (low amount of data).
- Unfreeze last n layers of pre-trained network (larger dataset).
- Unfreeze all layers and keep only architecture (very large dataset).

Classical approaches to NLP

Overview

Terminology:

- NLP = Natural Language Processing
- Corpus - Set of texts (documents) to be used for training
  - Set of emails
  - ...
- Document - A text unit to be processed
  - Single email
  - ...

Overview

Terminology:

- NLP = Natural Language Processing
- Corpus - Set of texts (documents) to be used for training
  - Set of emails
  - ...
- Document - A text unit to be processed
  - Single email
  - ...

Since lot of methods are based on word frequency analysis, data preprocessing is of essence:

- Stemming/lemmatisation
- Collocation detection
- Synonyms replacement

Other techniques used in NLP:

- TF-IDF (term frequency - inverse document frequency) - for words encoding
- SVD (singular value decomposition) - for dimensionality reduction

Text pre-processing
Stemming

Stemming is a process of reducing inflected words to their word stem.

Example:

- fishing -> fish
- fished -> fish
- fisher -> fish

Text pre-processing
Stemming

Stemming is a process of reducing inflected words to their word stem.

Example:

- fishing -> fish
- fished -> fish
- fisher -> fish
- argued -> argu
- arguing -> argu

Text pre-processing
Stemming

Stemming is a process of reducing inflected words to their word stem.

Example:

- fishing -> fish
- fished -> fish
- fisher -> fish
- argued -> argu
- arguing -> argu

How to do stemming:

a/ Use predefined mapping
- simple, fast
- new and unfamiliar words are not handled

b/ Automated mapping
- suffix stripping - remove "ed", "ing", "ly"
- works poorly for exceptional relations such as "run" and "ran"

Text pre-processing
Lemmatisation

lemma = canonical form

Lemmatisation replaces inflected words with their canonical form.

Lemmatisation uses vocabulary and morphological analysis:

- Lemmatisation uses part-of-speech (POS) tagging (recognition of nouns, verbs and other types)
- Lemmatisation uses vocabulary to determine word lemma
- Lemma for "saw" will be either "saw" or "see" - depending on the word was a noun or a verb

Text pre-processing
Collocations

Collocation refers to a group of two or more words that usually go together.

Examples:

- post office
- fast food

Text pre-processing
Collocations

Collocation refers to a group of two or more words that usually go together.

Examples:

- post office
- fast food

Collocation can be detected automatically using normalized point-wise mutual information (NPMI):

$$\begin{aligned} PMI &= \log\left[\frac{p(x,y)}{p(x)p(y)}\right] \\ NPMI &= \frac{PMI}{-\log p(x,y)} = \frac{\log[p(x)p(y)]}{\log p(x,y)} - 1 \end{aligned} \tag{42}$$

Words encoding
TF-IDF

TF-IDF = term frequency - inverse document frequency

- term frequency - frequency of a word in a document

$$TF(t,d) = \frac{\text{frequency of t in d}}{\text{number of words in d}} \tag{43}$$

- document frequency - measures document importance in a corpus

$$DF(t) = \text{number of documents with presence of t} \tag{44}$$

- inverse document frequency - measures informativeness of t

$$IDF(t) = \log \frac{N}{DF(t)+1} \tag{45}$$

- tf-idf

$$TF\text{-}IDF(t,d) = \frac{TF(t,d)}{IDF(t)} \tag{46}$$

Text pre-processing
Other tips

- stop words - most common words with low informative value. Remove them.
  - the
  - is
  - to
  - ...
- n grams - sequence of contiguous words
- To improve performance synonyms can be replaced

Dimensionality reduction
SVD

Let's assume we have TF-IDF matrix A with m rows that stand for terms and n columns that stands for documents. Rank of the matrix is $r \leq min(m, n)$.

Singular value decomposition of A is given by:

$$A = U\Sigma V^T \tag{47}$$

where U is an $m \times r$ orthogonal matrix, Σ is $r \times r$ diagonal matrix with singular values on diagonal and V is $r \times n$ orthogonal matrix.

- Singular values in Σ are greater than zero
- Largest singular value is in upper left corner of Σ

Dimensionality reduction
SVD

SVD can be viewed as sum of rank one matrices:

$$A = \sum_{i=1}^{r} \Sigma_{ii} U_{,i} V_{i,}^T \tag{48}$$

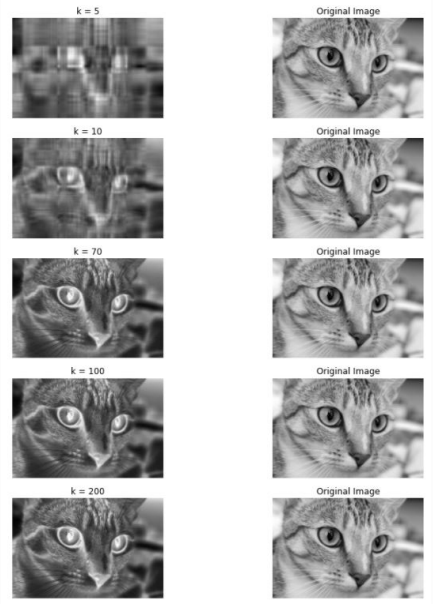
where $U_{,i}$ is i-th column of U and $V_{i,}$ is i-th row of V.
We can approximate A using only $k < r$ strongest singular values:

$$A \approx A_k = \sum_{i=1}^{k} \Sigma_{ii} U_{,i} V_{i,}^T = U_k \Sigma_k V_k^T \tag{49}$$

To project a document d represented by m-dimensional vector to k-dimensional space, we can use:

$$\hat{d} = U_k^T d \tag{50}$$

Dimensionality reduction
SVD



Sequence models

Sequence models
Examples

Examples of sequence data:

- Speech recognition
- Music generation
- Sentiment classification
  The food was awful. $\mapsto$ ● ○ ○ ○ ○
- Machine translation
  La nourriture était horrible. $\mapsto$ *The food was awful.*
- Video recognition
- Name entity recognition
  Yesterday, Harry met Hermione. $\mapsto$ *Yesterday, Harry met Hermione.*

Sequence models
Notation

We will use following notation:

- x ... input sequence
- $x^{<i>}$ ... i-th element of sequence x
- T_x ... total number of elements in x
- y ... output sequence
- $y^{<i>}$ ... i-th element of sequence y
- T_y ... total number of elements in y
- $X^{(i)<t>}$ ... t-th element of i-th training sequence
- $Y^{(i)<t>}$ ... t-th element of output sequence for i-th training sequence
- $T_x^{(i)}$ ... total number of elements in i-th training sequence
- $T_y^{(i)}$ ... total number of elements in output sequence for i-th training sequence

Sequence models
Vocabulary

How do we represent words in a sentence?

Vocabulary is enumerated list of known words.

- It can include punctuation.
- $<$ *UNKNOWN* $>$ can be added to represent word not included in vocabulary.

Each word can be transformed using one-hot encoding.

Sequence models

Forward propagation

Usually:

$$a^{<0>} = \mathbf{0} \tag{51}$$

After first layer:

$$\begin{aligned} a^{<1>} &= g_1\left(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a\right) \\ \hat{y}^{<1>} &= g_2\left(W_{ya}a^{<1>} + b_y\right) \end{aligned} \tag{52}$$

Note: In W_{ax}, first subscript (a) means that W_{at} is used to calculate a-like quantity, while second subscript (x) means that W_{ax} will be multiplied by x

Simplified notation:

$$\begin{aligned} W_a &= [W_{aa} \mid W_{ax}] \\ W_y &= W_{ya} \end{aligned} \tag{53}$$

Thus:

$$\begin{aligned} a^{<t>} &= g_1\left(W_a[a^{<t-1>}, x^{<t>}] + b_a\right) \\ \hat{y}^{<t>} &= g_2\left(W_y a^{<t>} + b_y\right) \end{aligned} \tag{54}$$

Sequence models
RNN types

Different types of RNN:

(a) **one-to-one**
This is standard neural network

(b) **many-to-many ($T_x = T_y$)**
example: name entity recognition

(c) **many-to-one**
example: semantic classification

(d) **one-to-many ($T_x \neq T_y$)**
example: music generation

(e) **many-to-many ($T_x \neq T_y$)**
example: machine translation

Sequence models
Backpropagation

Let us assume many-to-many architecture with $T_x = T_y$. After each generated $y^{<t>}$ we can calculate loss $l^{<t>}$. Final loss function is given by:

$$L = \frac{1}{T_y} \sum_{t=1}^{T_y} l^{<t>}(\hat{y}^{<t>}, y^{<t>}) \tag{55}$$

We have:

$$\begin{aligned} \hat{y}^{<t>} &= \hat{y}^{<t>}(a^{<t>}, W_y) \\ a^{<t>} &= a^{<t>}(x^{<t>}, a^{<t-1>}, W_a) \end{aligned} \tag{56}$$

Let us look at the derivative of L with respect to w_k, where w_k is from W_a:

$$\frac{\partial L}{\partial w_k} = \frac{1}{T_y} \sum_{t=1}^{T_y} \frac{\partial l^{<t>}}{\partial \hat{y}^{<t>}} \cdot \frac{\partial \hat{y}^{<t>}}{\partial a^{<t>}} \cdot \frac{\partial a^{<t>}(x^{<t>}, a^{<t-1>}, W_a)}{\partial w_k} \tag{57}$$

Sequence models
Backpropagation

If T_y is large, computational costs to evaluate gradient are growing. We have 3 options:

i) Full computation - can be slow.

ii) Truncating time step - derivate $a^{<t>}$ only from steps $[\tau; t]$. Focuses on short-term patterns

iii) Randomized truncation
   a) Define sequence π_t ($0 \leq \pi_t \leq 1$)
   b) ξ_t:

$$\begin{aligned} \mathrm{P}[\xi_t = 0] &= 1 - \pi_t \\ \mathrm{P}[\xi_t = \pi_t^{-1}] &= \pi_t \end{aligned} \tag{58}$$

ξ_t is used in loss function to reduce number of required derivatives. Note that $\mathrm{E}[\xi_t] = 1$

Sequence models
Language model

Let's assume speech recognition system:

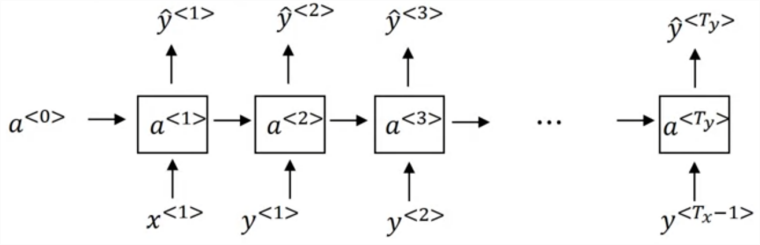
Most crimes are committed at night.
Most crimes are committed at knight.

Both sentences have the same pronunciation.

Language model assigns probability to each word given preceding words in a sentence.

$$\mathbb{P}\left[night \mid Most, crimes, are, committed, at\right] > \mathbb{P}\left[knight \mid Most, crimes, are, committed, at\right] \tag{59}$$

Sequence models
Language model

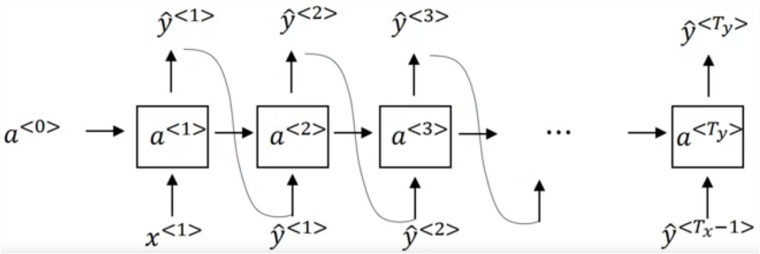


- $\hat{y}^{<t>}$ predicts probability of each word in vocabulary given previous words.
- $x^{<1>} = a^{<0>} = \mathbf{0}$
- loss function:

$$l^{<t>}(y_i^{<t>}, \hat{y}_i^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$
$$L = \sum_t l^{<t>}(y_i^{<t>}, \hat{y}_i^{<t>}) \tag{60}$$

Sequence models

Sampling novel sequence



- Stop generating sequence if $< EOS >$ or after fixed number of iteration
- $x^{<1>} = a^{<0>} = \mathbf{0}$
- $\hat{y}^{<t>}$ is drawn from distribution given be words probabilities
- Character level language model can be trained

Sequence models
Vanishing gradient problem

The cat, which ate ..., was full.

The cats, which ate ..., were full.

Sometimes probability of a word depends on very distant positions.

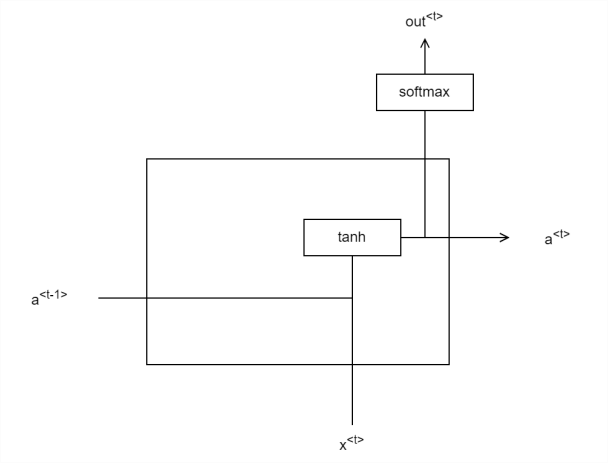
- Output $\hat{y}^{<t>}$ is strongly influenced by close inputs ($x^{<t>}$, $x^{<t-1>}$).
- It is not easy to propagate gradient from the end of sequence to the beginning.
- Exploding gradient can be dealt with by gradient clipping.
- What about vanishing gradient?

Sequence models
RNN unit

$$a^{<t>} = g(W_a \left[a^{<t-1>}, x^t\right] + b_a) \tag{61}$$

Sequence models
RNN unit

$$a^{<t>} = g(W_a \left[a^{<t-1>}, x^t\right] + b_a) \tag{61}$$

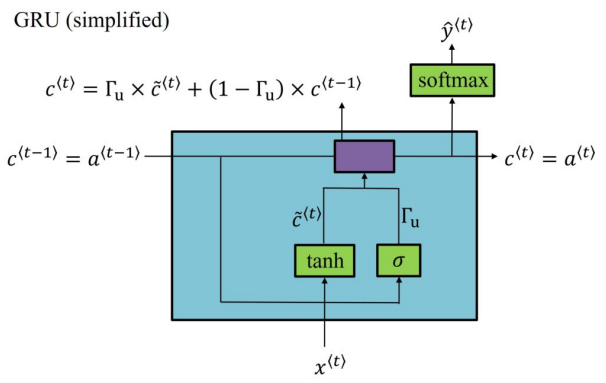


Sequence models
Gated recurrent unit

$$
\begin{aligned}
c^{<t-1>} &= a^{<t-1>} \\
\tilde{c}^{<t>} &= \tanh\left(W_c[c^{<t-1>}, x^{<t>}] + b_c\right) \\
\Gamma_u &= \sigma\left(W_u[c^{<t-1>}, x^{<t>}] + b_u\right) \\
c^{<t>} &= \Gamma_u \times \tilde{c}^{<t>} + (1 - \Gamma_u) \times c^{<t-1>}
\end{aligned}
\tag{62}
$$

Sequence models
Gated recurrent unit

$$
\begin{aligned}
c^{<t-1>} &= a^{<t-1>} \\
\tilde{c}^{<t>} &= \tanh\left(W_c[c^{<t-1>}, x^{<t>}] + b_c\right) \\
\Gamma_u &= \sigma\left(W_u[c^{<t-1>}, x^{<t>}] + b_u\right) \\
c^{<t>} &= \Gamma_u \times \tilde{c}^{<t>} + (1 - \Gamma_u) \times c^{<t-1>}
\end{aligned}
\tag{62}
$$

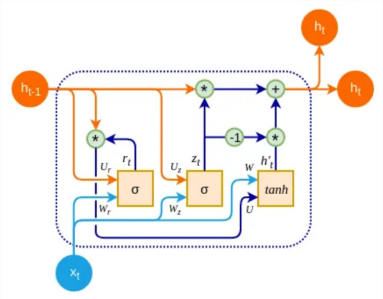


Sequence models
Gated recurrent unit

$$
\begin{aligned}
c^{<t-1>} &= a^{<t-1>} \\
\Gamma_r &= \sigma\left(W_r[c^{<t-1>}, x^{<t>}] + b_r\right) \\
\tilde{c}^{<t>} &= \tanh\left(W_c[\Gamma_r \times c^{<t-1>}, x^{<t>}] + b_c\right) \\
\Gamma_u &= \sigma\left(W_u[c^{<t-1>}, x^{<t>}] + b_u\right) \\
c^{<t>} &= \Gamma_u \times \tilde{c}^{<t>} + (1 - \Gamma_u) \times c^{<t-1>}
\end{aligned}
\tag{63}
$$

Sequence models
Gated recurrent unit

$$
\begin{aligned}
c^{<t-1>} &= a^{<t-1>} \\
\Gamma_r &= \sigma\left(W_r[c^{<t-1>}, x^{<t>}] + b_r\right) \\
\tilde{c}^{<t>} &= \tanh\left(W_c[\Gamma_r \times c^{<t-1>}, x^{<t>}] + b_c\right) \\
\Gamma_u &= \sigma\left(W_u[c^{<t-1>}, x^{<t>}] + b_u\right) \\
c^{<t>} &= \Gamma_u \times \tilde{c}^{<t>} + (1 - \Gamma_u) \times c^{<t-1>}
\end{aligned}
\tag{63}
$$

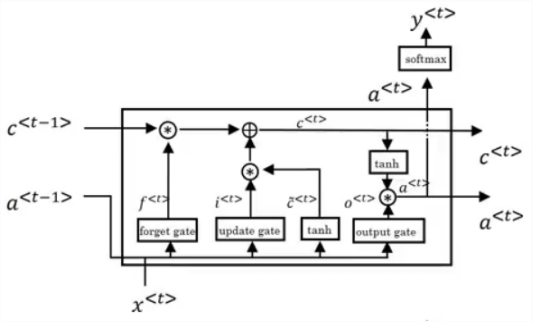


Sequence models
Long Short Term Memory

$$
\begin{aligned}
\Gamma_f &= \sigma\left(W_f[a^{<t-1>}, x^{<t>}] + b_f\right) \\
\Gamma_u &= \sigma\left(W_u[a^{<t-1>}, x^{<t>}] + b_u\right) \\
\tilde{c}^{<t>} &= \tanh\left(W_c[a^{<t-1>}, x^{<t>}] + b_c\right) \\
c^{<t>} &= \Gamma_u \times \tilde{c}^{<t>} + \Gamma_f \times c^{<t-1>} \\
\Gamma_o &= \sigma\left(W_o[a^{<t-1>}, x^{<t>}] + b_o\right) \\
a^{<t>} &= \Gamma_o \times \tanh(c^{<t>})
\end{aligned}
\tag{64}
$$

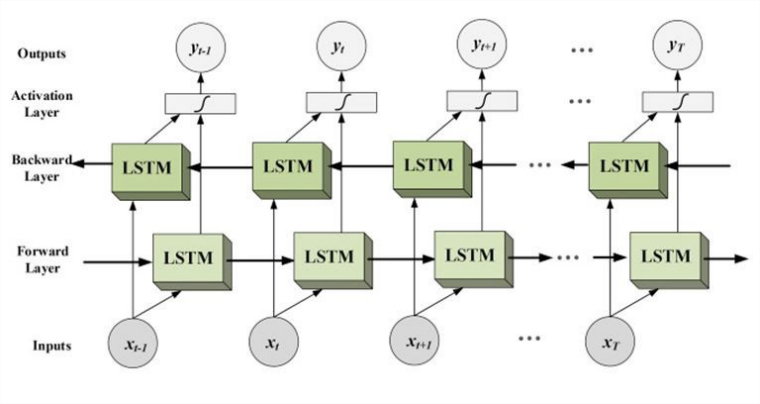
Sequence models
Long Short Term Memory

$$
\begin{aligned}
\Gamma_f &= \sigma\left(W_f[a^{<t-1>}, x^{<t>}] + b_f\right) \\
\Gamma_u &= \sigma\left(W_u[a^{<t-1>}, x^{<t>}] + b_u\right) \\
\tilde{c}^{<t>} &= \tanh\left(W_c[a^{<t-1>}, x^{<t>}] + b_c\right) \\
c^{<t>} &= \Gamma_u \times \tilde{c}^{<t>} + \Gamma_f \times c^{<t-1>} \\
\Gamma_o &= \sigma\left(W_o[a^{<t-1>}, x^{<t>}] + b_o\right) \\
a^{<t>} &= \Gamma_o \times \tanh(c^{<t>})
\end{aligned}
\tag{64}
$$

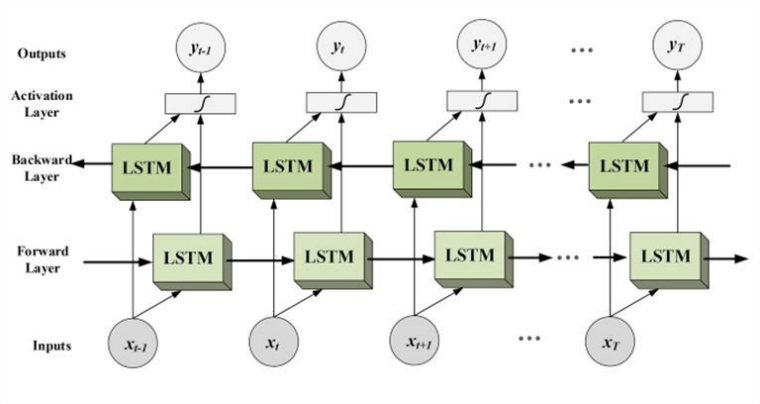


Nice LSTM description: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequence models
Bidirectional RNN

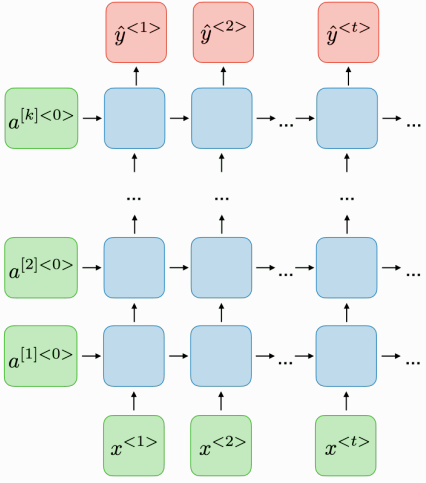


Sequence models
Bidirectional RNN



+ To predict $\hat{y}^{<t>}$ the whole sequence is used, not just sequence up to $x^{<t>}$
- To calculate prediction, you need to wait until the whole sequence is finished

Sequence models
Deep RNN



- Each layer has one weight matrix assigned
- 3 layers network might be already quite large
- Deep RNN can be build using RNN, GRU, LSTM or BRNN layers

Word embeddings

TARAN

ADVISORY IN DATA & ANALYTICS

Word embeddings

Word representation

So far we were using one-hot encoding for words representation.

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Water (9532)	Beer (982)
0	0				
⋮					
1	⋮				
		⋮	⋮	⋮	⋮
⋮	1				
	⋮				
0	0				

Word embeddings

Word representation

Instead one-hot, we might want to use feature representation.

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Water (9532)	Beer (982)
Gender	-0.97	0.98	-0.95	0.93	0.02	0.01
Royal	0.04	-0.02	0.96	0.95	0.01	-0.04
Beverage	-0.03	0.01	0.02	- 0.04	0.97	0.98
⋮						

- Features in embeddings don't have nice meaning like in the table above
- Vectors for similar things (water and beer) are similar in feature space

Word embeddings
t-SNE

t-SNE = t-distributed stochastic neighbour embedding

Let us have a set of N high-dimensional objects $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

$$p_{j|i} = \frac{\frac{\exp(-\|\mathbf{x_i}-\mathbf{x_j}\|^2)}{2\sigma_i^2}}{\sum_{k\neq i}\frac{\exp(-\|\mathbf{x_i}-\mathbf{x_k}\|^2)}{2\sigma_i^2}} \qquad \forall i \neq j$$
$$p_{i|i} = 0 \tag{65}$$

"The similarity of datapoint x_j to datapoint x_i is the conditional probability $p_{j|i}$ that x_i would pick x_j as its neighbour if neighbours were picked in proportion to their probability density under a Gaussian centred at x_i."

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$
$$p_{ii} = 0 \tag{66}$$

Word embeddings
t-SNE

We want to learn d-dimensional representation $\mathbf{y}_1, ..., \mathbf{y}_N$ of $\mathbf{x}_1, ..., \mathbf{x}_N$.

$$q_{ij} = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_{l \neq k} \left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1}} \qquad (67)$$
$$q_{ii} = 0$$

Locations of $\mathbf{y}_1, ..., \mathbf{y}_N$ are given by minimization of distance between distributions P and Q.

$$KL(P\|Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \qquad (68)$$

Word embeddings
Similarity functions

What if, for a given word w, we want to find a most similar word in embedding space:

$$\underset{w_i}{\operatorname{argmax}}\, sim(e_w, e_{w_i}) \tag{69}$$

Similarity functions can be:

- Cosine similarity

$$sim(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2} \tag{70}$$

- Negative Euclidean distance

$$sim(u, v) = -\|u - v\|^2 \tag{71}$$

Word embeddings
Learning embeddings

Let's o_i be the one-hot encoding of i-th word in vocabulary.

$$o_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i - th\ position \tag{72}$$

Embedded vector for i-th word is given by embedding matrix E.

$$e_i = E \cdot o_i = \begin{bmatrix} e_{11} & \dots & e_{1m} \\ \vdots & \ddots & \vdots \\ e_{n1} & \dots & e_{nm} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \tag{73}$$

For vocabulary size 10000 and 300-dimensional embedding space, $E \in \mathbb{R}^{300x10000}$

Word embeddings
Learning embeddings

Let's have a language model.

I want a glass of orange juice.

Based on "I want a glass of orange", we want to predict "juice".

Word embeddings
Learning embeddings

Let's have a language model.

I want a glass of orange juice.

Based on "I want a glass of orange", we want to predict "juice".

Model:

- Start with one-hot encoding
- Apply embedding
- Add output layer with softmax activation (output layer size is given by vocabulary size)

Embedding matrix is learned through model training (*E* consists of trainable parameters).

Word embeddings
Learning embeddings

Let's have a language model.

I want a glass of orange juice.

Based on "I want a glass of orange", we want to predict "juice".

Model:

- Start with one-hot encoding
- Apply embedding
- Add output layer with softmax activation (output layer size is given by vocabulary size)

Embedding matrix is learned through model training (*E* consists of trainable parameters).

Words used for predicting target word are called context.

- n words left from target word
- n words left + n word right from target word
- 1 word left
- nearby one word (skip-gram)

Word embeddings
Learning embeddings

I want a glass of orange juice.

context	target
glass	orange
glass	of
glass	want
⋮	⋮

$$\mathbb{P}[t|c] = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}} \tag{74}$$

Loss function:

$$L(\hat{y}, y) = -\sum_{i=1}^{10000} y_i \log \hat{y}_i \tag{75}$$

Word embeddings
Word2vec

Problem with previous approach is the computational cost.

We will use negative sampling to create a training sample:

context	target	y
orange	juice	1
orange	football	0
orange	paper	0
⋮	⋮	0
orange	of	0

For one positive context-target pair we added K negative pairs.

The output layer size will still be equal to vocabulary size, but each neuron will calculate sigmoid instead of softmax.

$$\mathbb{P}[y = 1|c, t] = \sigma(\theta_t^T e_c) \tag{76}$$

Word embeddings
Word2vec

How to choose negative samples:

- x We can draw negative samples based on distribution over word frequencies
  Leads to high presence of words such as "the", "of", "a"
- x We can sample words uniformally
  This is very non-representative
- √ Use following distribution (discovered empirically)

$$\mathbb{P}[w_i] = \frac{f(w_i)^{3/4}}{\sum_j f(w_j)^{3/4}} \tag{77}$$

Large Language Models

TARAN

ADVISORY IN DATA & ANALYTICS

Large Language Models
Content

What we will talk about:

- What is LLM (introduction)
- Model architecture
  - Self-attention
  - Transformer
  - Different architectures for different tasks
- Prompt engineering
- Generative configuration settings
- Computational challenges
- Model fine-tuning
- Performance metrics

Large Language Models
Introduction

Definition (Generative AI)

Generative AI refers to a class of artificial intelligence systems that are capable of creating new data or content that is similar to, but distinct from, the data it was trained on. These systems learn the underlying patterns and structures of a dataset and then use that knowledge to generate new examples that share similarities with the original data.

Generative AI models are used for generating:

- images
- video
- audio
- **text**
- speech

Large Language Models
Introduction

What is LLM:

- LLM are nowadays state of the are solution for NLP tasks
- They are powered by Transformers architecture
- They are trained on massive datasets over even weeks or months

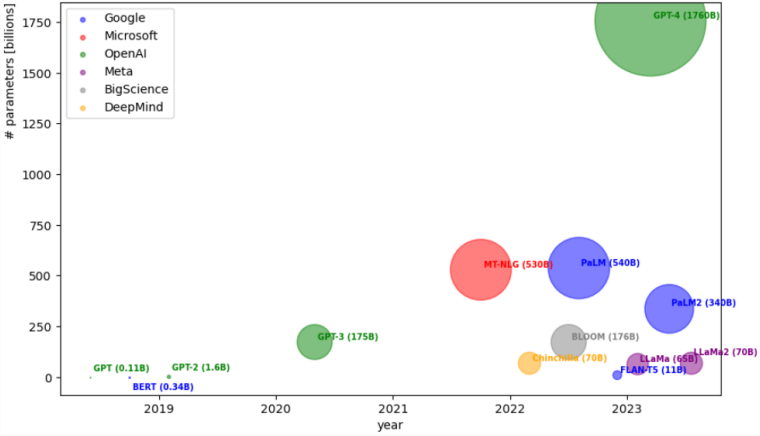
LLM use cases are:

- ChatBot
- Write an essay
- Text summarization
- Translation tasks
- Generate program code
- Information retrieval
- LLMs can be enriched by allowing them to connect to external data sources to fetch up to date data.

Large Language Models

Existing models

There are many trained LLM models that differ in size and performance.



Large Language Models

Model size

What is the importance of number of parameters?

- With increasing number of parameters, model's subjective understanding of language also increases
- Large number of parameters might be needed for multi-task models
- Smaller models can be fine-tuned to perform well on specific tasks

Large Language Models

Attention algorithm

Let's assume a translation task. Standard RNN first processes input text **x** into context vector **c** (hidden state after processing all the input words). Decoder then generates words in recurrent fashion by maximizing conditional probability and generated output text having joint probability

$$p(\mathbf{y}) = \prod_{t=1}^{T} p\left(y_t | \{y_1, ..., y_{t-1}\}, c\right) \tag{78}$$

Problem here is that the whole input is transformed into one context vector **c**.

Attention algorithm uses conditional probabilities in a form

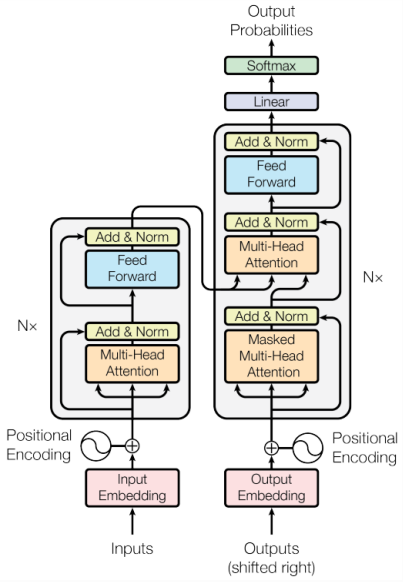
$$p(\mathbf{y}) = \prod_{t=1}^{T} p\left(y_t | \{y_1, ..., y_{t-1}\}, c_t\right) \tag{79}$$

Each predicted word in output sequence has its own context.

Large Language Models

Transformer architecture

- Transformer was first introduced in paper **Attention is all you need** in 2017.
- Components:
  - Input embedding
  - Positional encoding
  - Multi-Head Attention
  - Feedforward NN

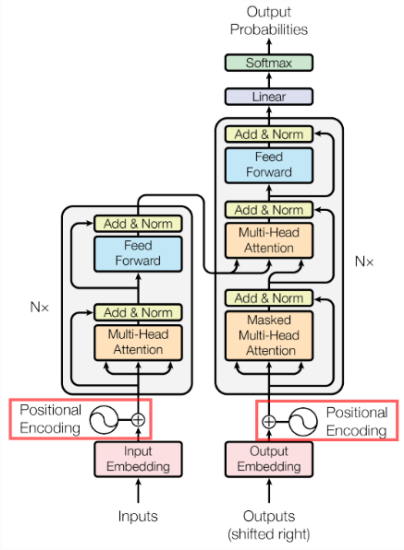


Link to the original paper: https://arxiv.org/pdf/1706.03762.pdf

Large Language Models

Positional encoding

- The transformer does not contain recurrence or convolution - we need to add information about the position.
- Positional encoding should satisfy:
  - Unique encoding for each position in sequence.
  - Distance between two positions should not depend on time length.
  - Values should be limited to reasonable range ($[-1; 1]$).
  - Encoding should generalize for longer sentences.



Large Language Models

Positional encoding

Proposed positional encoding:

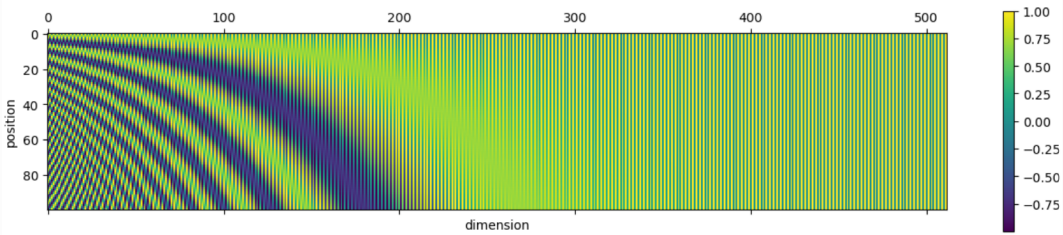
$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \end{aligned} \tag{80}$$

d_{model} is dimensionality of encoding (512 in original paper).

Model can easily attend to relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos} (for proof see Transformer Architecture: The Positional Encoding)

Large Language Models

Positional encoding

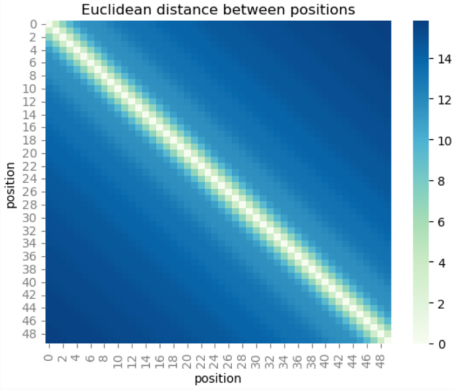


Large Language Models

Positional encoding

Encoding has following properties:

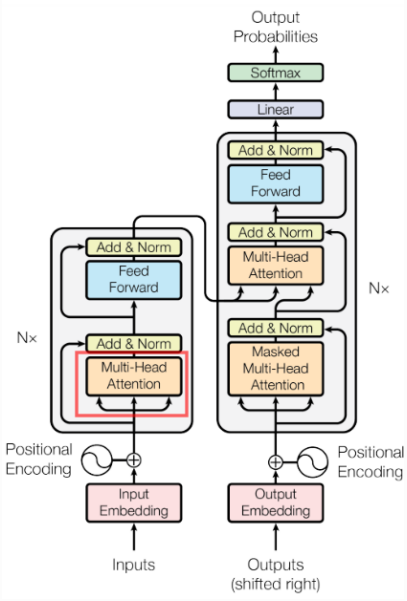
- Distance between two positions is symmetrical.
- Distance between positions with the same offset is independent on position.
- Larger difference in positions means larger distance in encoding space.



Large Language Models

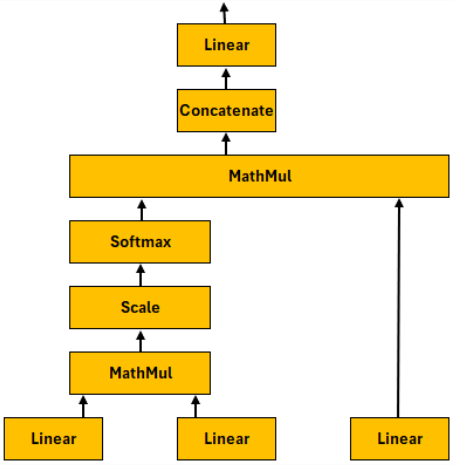
Multi-head attention

- Self-attention extracts assess words relations inside one sentence.
- To get self-attention, similar approach as in retrieval systems will be used.



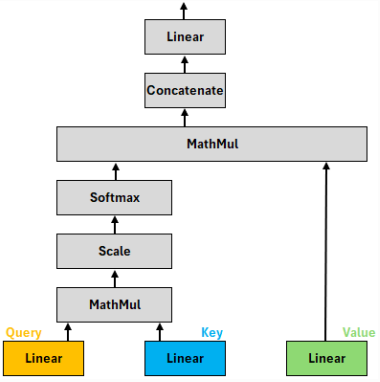
Large Language Models

Multi-head attention

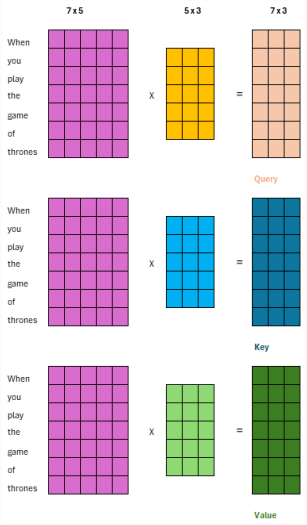


Large Language Models

Multi-head attention



- Mapping inputs to outputs
- Changing matrix dimensions



Large Language Models

Multi-head attention

Let us recall cosine similarity:

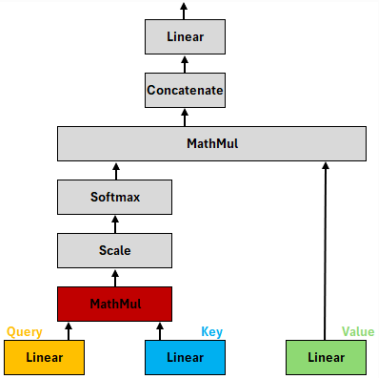
$$cos(A, B) = \frac{A \cdot B^T}{|A| \cdot |B|} \tag{81}$$

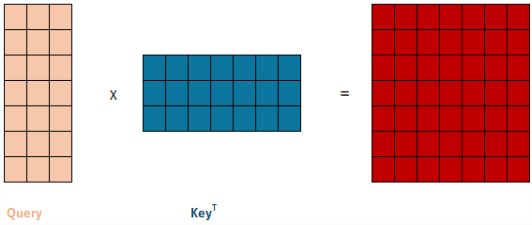
In general:

$$sim(A, B) = \frac{A \cdot B^T}{scaling} \tag{82}$$

Large Language Models

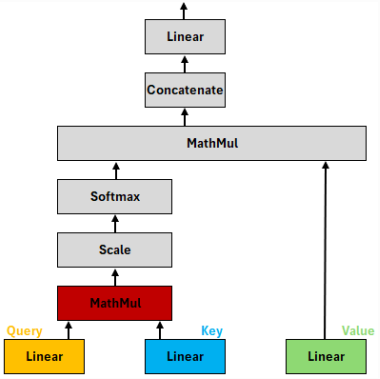
Multi-head attention





Large Language Models

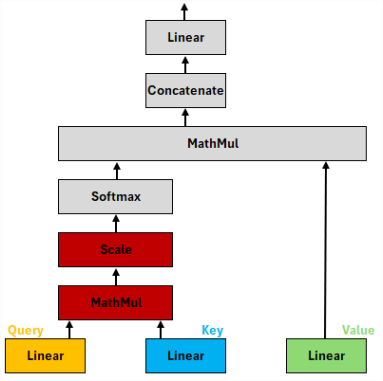
Multi-head attention



	When	you	play	the	game	of	thrones
When	89	20	41	10	55	10	59
you	20	90	81	22	70	15	72
play	41	81	95	10	90	30	92
the	10	22	10	92	88	40	89
game	55	70	90	88	98	44	87
of	10	15	30	40	44	85	59
thrones	59	72	92	90	95	59	99

Large Language Models

Multi-head attention

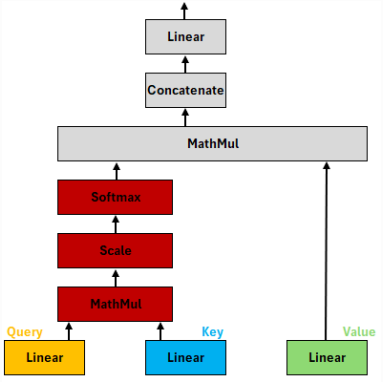


	When	you	play	the	game	of	thrones
When	89	20	41	10	55	10	59
you	20	90	81	22	70	15	72
play	41	81	95	10	90	30	92
the	10	22	10	92	88	40	89
game	55	70	90	88	98	44	87
of	10	15	30	40	44	85	59
thrones	59	72	92	90	95	59	99

$$\frac{}{\text{sqrt}(d_k)}$$

Large Language Models

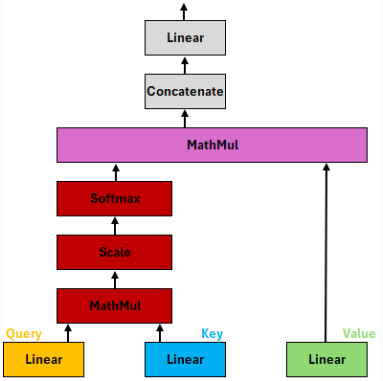
Multi-head attention



	When	you	play	the	game	of	thrones
When	1.00	0.00	0.00	0.00	0.00	0.00	0.00
you	0.00	0.97	0.03	0.00	0.00	0.00	0.00
play	0.00	0.00	0.68	0.00	0.10	0.00	0.22
the	0.00	0.00	0.00	0.65	0.14	0.00	0.21
game	0.00	0.00	0.03	0.02	0.72	0.00	0.23
of	0.00	0.00	0.00	0.00	0.00	1.00	0.00
thrones	0.00	0.00	0.05	0.17	0.00	0.00	0.75

Large Language Models

Multi-head attention



$$Attention(Q, K, V) = \text{softmax}\left(\frac{Q \cdot (K)^T}{\sqrt{d_k}}\right) V \qquad (83)$$

Large Language Models

Multi-head attention

On previous slide we described single-head attention. We apply single-head attention multiple-time to arrive at multi-head attention. Final output is given by concatenation.

$$\begin{aligned} \mathit{MultiHead}(Q, K, V) &= \mathrm{concat}(\mathit{head}_1, ..., \mathit{head}_h)W^O \\ \mathit{head}_i &= \mathit{Attention}(Q, K, V) \end{aligned} \tag{84}$$

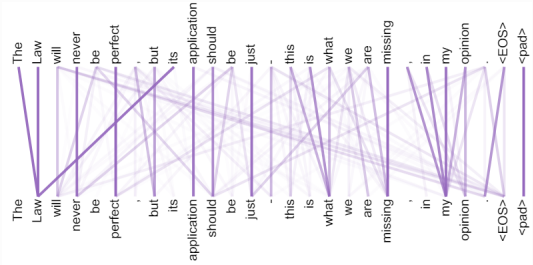
The dimensions are $W_i^Q \in \mathrm{R}^{d_{model} \times d_k}$, $W_i^K \in \mathrm{R}^{d_{model} \times d_k}$, $W_i^V \in \mathrm{R}^{d_{model} \times d_v}$ and $W_i^O \in \mathrm{R}^{hd_v \times d_{model}}$.

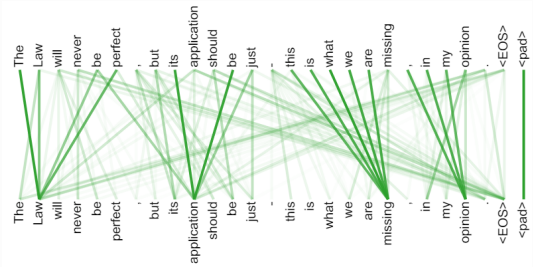
Each head can learn different semantic meaning. For instance, one head can focus on sentiment, other can focus on entities included in the text and so on.

Large Language Models

Masked multi-head attention

Attention maps for different heads



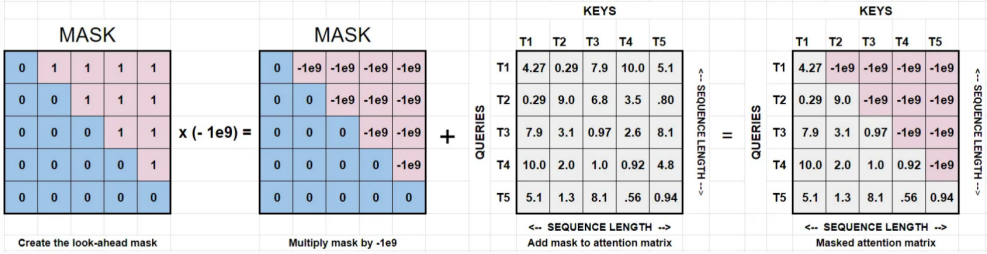


Large Language Models

Masked multi-head attention

Attention matrix gives us an information about how the words in the sentence relates to each other. In decoding phase, at stage where first n words were predicted, we cannot use attention with following words. They are unknown at this stage.

We apply mask just before softmax to ensure that attention to future words are zero.



Large Language Models

Rest of the transformer

1. Feed forward
   Feed forward neural network consists of two layers (in original paper) with relu activation.
2. Residual connections and normalization
   To improve information propagation residual connections are added and normalization helps to overcome vanishing gradient.
3. Combining inputs from encoder and decoder
   Inputs from encoder and decoder are combined in multi-head attention layer. Keys and Queries are provided from encoder, while Values are taken from decoder part.
4. Final linear layer
   Linear layer before decoder output is just fully connected layer, which transfer hidden state to required dimensionality. Typically, output dimensionality is size of the vocabulary - each one neuron represents one word from vocabulary.
5. Softmax
   Finally, we apply softmax to extract probabilities of each word being the next word of sentence.

Large Language Models

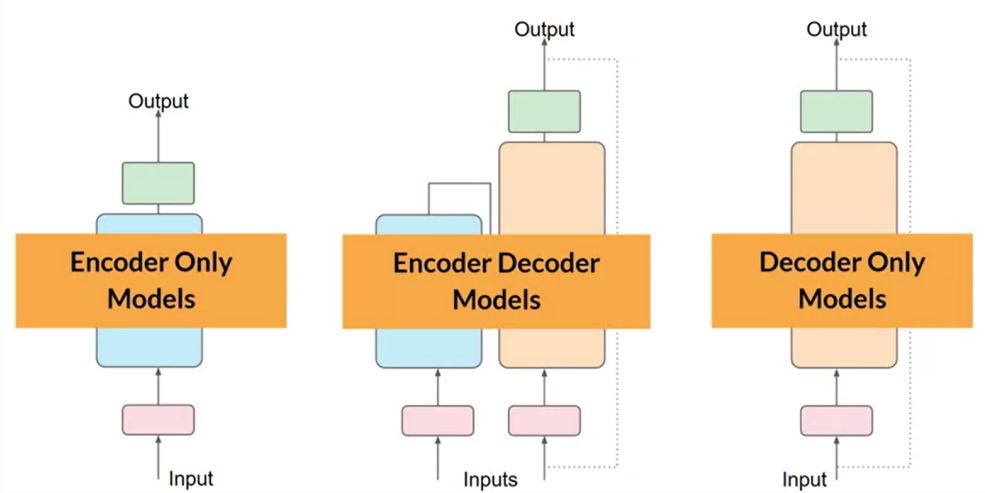
Rest of the transformer

1. Feed forward
   Feed forward neural network consists of two layers (in original paper) with relu activation.
2. Residual connections and normalization
   To improve information propagation residual connections are added and normalization helps to overcome vanishing gradient.
3. Combining inputs from encoder and decoder
   Inputs from encoder and decoder are combined in multi-head attention layer. Keys and Queries are provided from encoder, while Values are taken from decoder part.
4. Final linear layer
   Linear layer before decoder output is just fully connected layer, which transfer hidden state to required dimensionality. Typically, output dimensionality is size of the vocabulary - each one neuron represents one word from vocabulary.
5. Softmax
   Finally, we apply softmax to extract probabilities of each word being the next word of sentence.

Note: Search for 'Visual Guide to Transformer Neural Networks' on youtube - 4 videos about transformers with very nice visualizations and explanations.

Large Language Models

Architectures



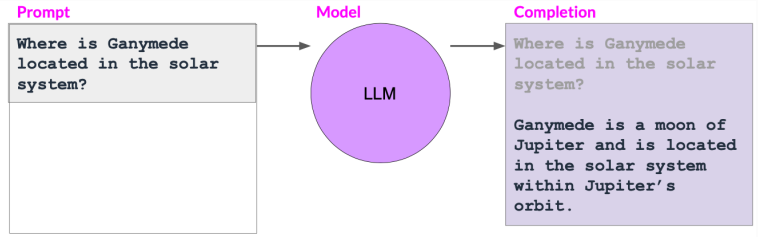
Large Language Models
Architectures

1. Encoder only models
   - Tasks focused on understanding semantic context such as
     - Named entity recognition
     - Sentiment analysis
   - Example: BERT (Bidirectional Encoder Representations from Transformers).
2. Decoder only models
   - Models focused on generating sequences.
   - In chat-bot, we ask question which is treated as a start of the sequence. The goal is to finish the sequence (with answer).
   - Example: GPT (Generative Pre-trained Transformer).
3. Encoder-decoder models
   - Sequence to sequence tasks such as
     - Text translation
   - BART (Bidirectional and Auto-Regressive Transformer)

Large Language Models
Prompting

Terms overview:

- **Prompt** - text we feed into the model
- **Inference** - model generated text
- **Completion** - output text
- **Context window** - full amount of text on input



Large Language Models
Prompting

The model does not have to perform well always on the first try. The prompt may be revised several time before the answer is correct.

The process of developing and improving the prompt is called **prompt engineering**.

Let us assume we want to classify sentiment of given statement:

```
INPUT PROMPT:
Statement:
This lecture is great.

Sentiment:
```

Large Language Models
Prompting

The model does not have to perform well always on the first try. The prompt may be revised several time before the answer is correct.

The process of developing and improving the prompt is called **prompt engineering**.

Let us assume we want to classify sentiment of given statement:

```
INPUT PROMPT:
Statement:
This lecture is great.

Sentiment:
```

```
OUTPUT:
Statement:
This lecture is great.

Sentiment: The lecture is interesting.
```

Situation described above is called **zero shot inference**. Largest LLM models are surprisingly good at this, smaller models might have a problem with such a task.

Large Language Models
Prompting

Model prediction can benefit from seeing what we want to achieve more explicitly in the prompt. Let's provide one full example of the task at hand followed by task to be processed - **one shot inference**.

```
INPUT PROMPT:
Statement:
The teacher does not know what he is talking about.

Sentiment: Negative

Statement:
This lecture is great.

Sentiment:
```

Large Language Models
Prompting

Model prediction can benefit from seeing what we want to achieve more explicitly in the prompt. Let's provide one full example of the task at hand followed by task to be processed - **one shot inference**.

```
INPUT PROMPT:
Statement:
The teacher does not know what he is talking about.

Sentiment: Negative

Statement:
This lecture is great.

Sentiment:
```

```
OUTPUT:
Statement:
The teacher does not know what he is talking about.

Sentiment: Negative

Statement:
This lecture is great.

Sentiment: Positive
```

We can provide multiple examples in an input prompt, so called **few shot inference**. However, remember that our context window is limited. In general, if the model does not behave correctly with 5 or 6 examples, we should consider **fine tuning model** with specific data, rather than adding more examples.

Large Language Models

Generative configuration

There are several **inference parameters** that can be used to control:

- Length of the output.
- How creative the output is.

Large Language Models

Generative configuration

There are several **inference parameters** that can be used to control:

- Length of the output.
- How creative the output is.

Some of those parameters are:

- *max_new_tokens* - limit maximum token to be used in the output
- *top_k* - randomly select next word from k words with highest predicted probability
- *top_p* - randomly select next word from n most probable words selected such that their cumulative probability does not exceed p.
- *temperature* - scale predicted probabilities

$$\mathrm{softmax}(z_i, T) = \frac{e^{\frac{z_i}{T}}}{\sum_{j=1}^{K} e^{\frac{z_i}{T}}} \tag{85}$$

  - temperature < 1 - reduce randomness
  - temperature > 1 - increase randomness

Large Language Models

max new tokens

max_new_tokens can be used to limit maximum tokens in an output. It is used as a cut-off, after max new tokens limit is reached, output stops generating additional text.

```
INPUT PROMPT:
Can you give me a summary of what the transformer is?

OUTPUT (max_new_token=200):
The transformer is a device that converts electrical energy into electrical energy.
OUTPUT (max_new_token=10):
The transformer is a device that converts electrical
```

Large Language Models

Random sampling - top k

Greedy vs random sampling

At each step LLM outputs probabilities of next word for every word in vocabulary.

- Greedy approach - take the word with the highest probability.
- Random sampling - select word randomly using probability distribution generated by model output.

Random sampling helps to generate more natural text avoiding excessive repetition of a word.

```
INPUT PROMPT:
Can you give me a summary of what the transformer is?

OUTPUT top_k=50 run 0:
A metering device combining digital and analog inputs.
OUTPUT top_k=50 run 1:
The 'daddy' is a unit of the United States Air Force that can deliver power to civilian units.
OUTPUT top_k=50 run 2:
a circuit forming circuit is a circuit, circuit, or circuit that is transformed into a piece of electrically active material w
ithin the field of physics, an electric theory, or a scientific theory.
```

Large Language Models

Random sampling - temperature

In contrast to *top_k* and *top_p*, temperature actually modifies the output of the model. It scales probabilities. For values below 1, probability focuses more on highest probable words, for values above 1 it is vice versa.

```
INPUT PROMPT:
Can you give me a summary of what the transformer is?

OUTPUT temperature=0.2 run 0:
The transformer is a device that converts a single transistor into a single-circuit transistor.
OUTPUT temperature=0.2 run 1:
The transformer is a device that converts electrical energy into electrical energy.
OUTPUT temperature=0.2 run 2:
The transformer is a device that converts a single digit integer into a multiple digit integer.
```

```
INPUT PROMPT:
Can you give me a summary of what the transformer is?

OUTPUT temperature=5.0 run 0:
The transistor converts the primary conductor into the second path for wiring where the ground bus becomes the intermediate.
OUTPUT temperature=5.0 run 1:
Electrical generator sold individually among independent transformer manufacturer
OUTPUT temperature=5.0 run 2:
a switch switch which operates the voltage regulator between two electric devices whose function to convert output energy can cause power loss:
```

Large Language Models

Computational challenges

What amount of memory we need when training model with 1B parameters?

Large Language Models
Computational challenges

What amount of memory we need when training model with 1B parameters?

1 parameter = 4 bytes (32-bit float)
1B parameters = $4 * 10^9$ bytes = 4GB

Large Language Models
Computational challenges

What amount of memory we need when training model with 1B parameters?

1 parameter = 4 bytes (32-bit float)
1B parameters = $4 * 10^9$ bytes = 4GB

We also need to store:

- Optimizer (2 states) - 8 bytes per parameter
- Gradients (2 states) - 4 bytes per parameter
- Activations and temporary variables - 8 bytes per parameter

Large Language Models
Computational challenges

What amount of memory we need when training model with 1B parameters?

1 parameter = 4 bytes (32-bit float)
1B parameters = $4 * 10^9$ bytes = 4GB

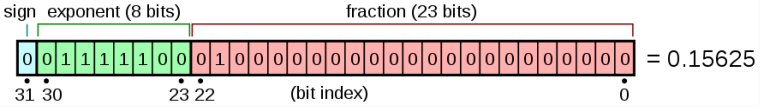
We also need to store:

- Optimizer (2 states) - 8 bytes per parameter
- Gradients (2 states) - 4 bytes per parameter
- Activations and temporary variables - 8 bytes per parameter

We have **24GB** of memory for 1B parameters with 32-bit full precision.

Large Language Models
Quantization

Let's see how the 32-bit float is represented in memory:



The final number is given by

$$value = (-1)^{sign} * 2^{\sum_{i=0}^{7} b_{23+i} 2^i - 127} * \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right) \tag{86}$$

With 32-bit float a range of $[-3.402e^{38}; +3.402e^{3}8]$ can be represented.

One option to reduce amount of consumed memory is to reduce parameter precision.

Large Language Models

Quantization

Parameter type	exponent [bits]	fraction [bits]	Range	Required memory per parameter
FP32	8	23	$[-3.402e^{38}; +3.402e^{38}]$	4 bytes
FP16	5	10	$[-65504; 65504]$	2 bytes
BFLOAT16	8	7	$[-3.402e^{38}; +3.402e^{38}]$	2 bytes
INT8	0	7	$[-128; 127]$	1 byte

Large Language Models
Quantization

Parameter type	exponent [bits]	fraction [bits]	Range	Required memory per parameter
FP32	8	23	$[-3.402e^{38}; +3.402e^{38}]$	4 bytes
FP16	5	10	$[-65504; 65504]$	2 bytes
BFLOAT16	8	7	$[-3.402e^{38}; +3.402e^{38}]$	2 bytes
INT8	0	7	$[-128; 127]$	1 byte

Let us look into an example of how π is represented with different data types:

	π
real value	3.1415926535897932384
FP32	3.1415925025939941406
FP16	3.140625
BFLOAT16	3.140625
INT8	3

Large Language Models
Quantization

Parameter type	exponent [bits]	fraction [bits]	Range	Required memory per parameter
FP32	8	23	$[-3.402e^{38}; +3.402e^{38}]$	4 bytes
FP16	5	10	$[-65504; 65504]$	2 bytes
BFLOAT16	8	7	$[-3.402e^{38}; +3.402e^{38}]$	2 bytes
INT8	0	7	$[-128; 127]$	1 byte

Let us look into an example of how π is represented with different data types:

	π
real value	3.1415926535897932384
FP32	3.1415925025939941406
FP16	3.140625
BFLOAT16	3.140625
INT8	3

Note that with 500B parameters and FP32 precision, we need 12000GB just to store data for training. We need to split training across hundreds of GPUs, which can be pretty expensive.

Large Language Models

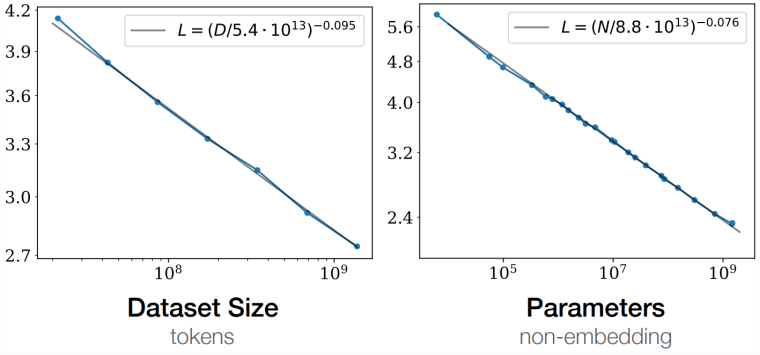
Scaling laws

How does model performance scales with the dataset size and number of trainable parameters?

Large Language Models

Scaling laws

How does model performance scales with the dataset size and number of trainable parameters?

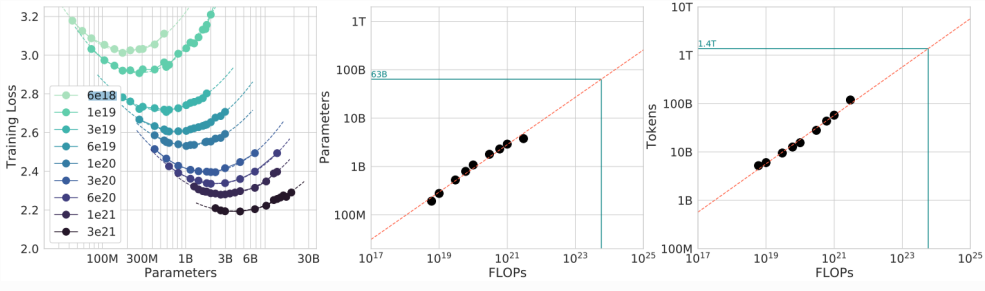


Kaplan et al., 2020: Scaling laws for neural language models
https://arxiv.org/pdf/2001.08361.pdf

Large Language Models

Scaling laws

Hoffmann, Borgeaud, Mench, 2022: Training Compute-Optimal Large Language Models
https://arxiv.org/pdf/2203.15556.pdf



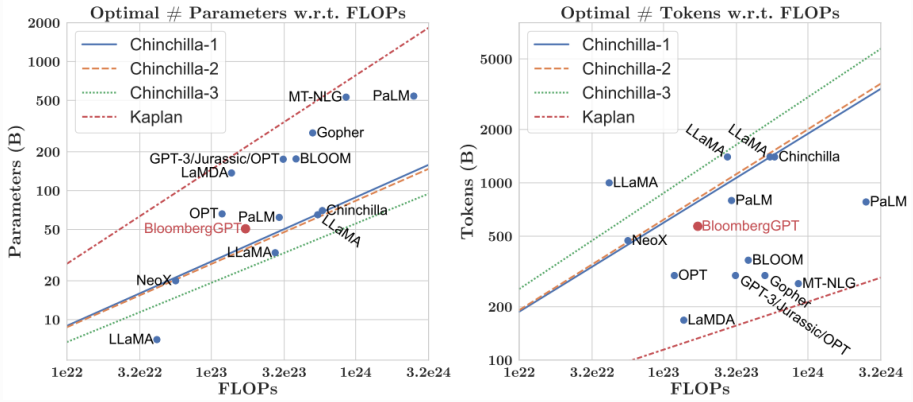
$$N_{opt} \propto C^a$$
$$D_{opt} \propto C^b \tag{87}$$

Both c and b were calculated to be very close to 0.5. Thus it make sense to scale data and model size with the same pace.

Large Language Models

Scaling laws

Based on Chinchilla paper findings, many models are too big in size and trained on smaller datasets than optimal.



Wu, Irsoy, Lu, 2023: BloombergGPT: A Large Language Model for Finance
https://arxiv.org/pdf/2303.17564.pdf

Large Language Models

Scaling laws

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

Large Language Models

Fine-tuning

Model **pre-training** trains the model on the vast amount of text data.

To improve performance on specific task, we can further **fine-tune** the model using task specific labelled data.

Large Language Models
Fine-tuning

Model **pre-training** trains the model on the vast amount of text data.

To improve performance on specific task, we can further **fine-tune** the model using task specific labelled data.

Options to fine-tune the model contains:

- Instruction fine-tuning
- Parameter Efficient Fine-Tuning (PEFT)

Large Language Models

Instruction fine-tuning

In instruction fine-tuning we create prompt/completion pair that focuses on the task we want to improve in.

Large Language Models
Instruction fine-tuning

In instruction fine-tuning we create prompt/completion pair that focuses on the task we want to improve in.

Classify review:
The movie was absolutly great!
Sentiment: Positive

Classify review:
I don't like math.
Sentiment: Negative

We feed this data to model training and update the weights.

- Full fine-tuning - all the model weights are updated.
- Instruction fine-tuning is the most common fine-tuning approach.
- Good results can be achieved with the small datasets when fine-tuning for a single task.
- However, catastrophic forgetting can occur - while model is improved for one task, on other tasks the performance can significantly drop

Large Language Models
Parameter Efficient Fine-Tuning

Full fine-tuning has a full memory requirements, including storing parameters, gradient, etc.

In PEFT, we update only small subset of parameters. Several approaches are possible:

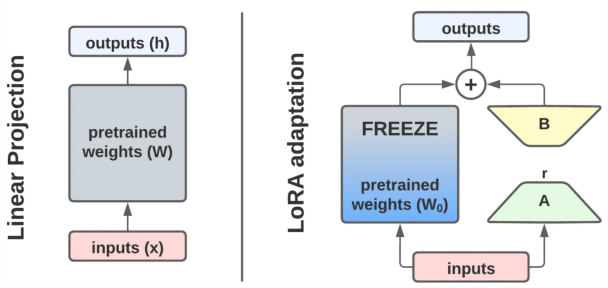
- Selective
  - Update only specific components of the model
  - Update only specific layers
  - Update only specific parameter types

  Results of this approach is mixed.
- Reparametrization
  - Model weights are replaced with their low rank rank representation.
  - Low Rank Adaptation (LoRA)
- Additive

  Introduce additional trainable layers or parameters to the model. Two approaches:
  - Adapters - add new layers, typically inside the encoder or decoder of the model, after the attention or feed-forward components.
  - Soft Prompts - add additional parameters to the input encoding or retraining the embedding weights.

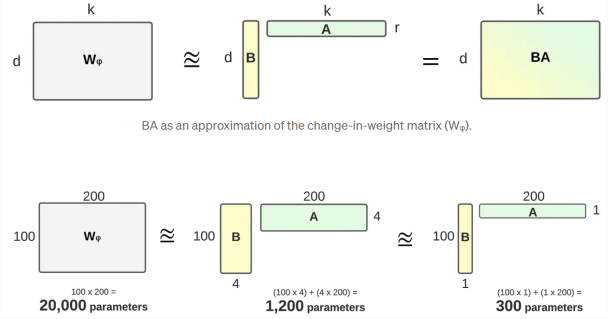
Large Language Models
Low Rank Adaptation

In LoRA, we freeze matrices inside self-attention layer and add newly trained matrices, whose multiplication has the same size as the original matrix.



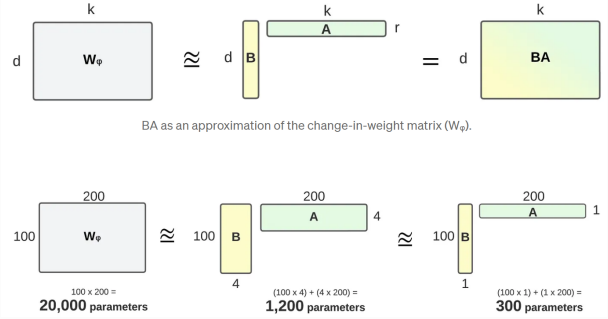
Large Language Models
Low Rank Adaptation

Dimension *r* of matrices *A* and *B* is a hyper-parameter that can be used to control number of new parameters to train.



Large Language Models
Low Rank Adaptation

Dimension *r* of matrices *A* and *B* is a hyper-parameter that can be used to control number of new parameters to train.



- LoRA can be used with other than self-attention layer component, however self-attention layer is common choice.
- We can have LoRA parameters for different tasks. During inference we can use appropriate version based on the task at hand.

Large Language Models

Model evaluation metrics

Traditional ML models might measure performance with:

$$accuracy = \frac{correct\_predictions}{total\_predictions} \tag{88}$$

With LLM the correct answer might not be fully deterministic. Answers like

The match was spectacular. The game was excellent.

might be equally good.

Large Language Models

Model evaluation metrics

Traditional ML models might measure performance with:

$$accuracy = \frac{correct\_predictions}{total\_predictions} \tag{88}$$

With LLM the correct answer might not be fully deterministic. Answers like

The match was spectacular. The game was excellent.

might be equally good.

On the other hand, very similar answers

The game was exciting. The game was not exciting.

have very different meaning.

Large Language Models

Model evaluation metrics

We will discuss two evaluation metrics:

- ROUGE
  - Recall Oriented Understudy for Gisting Evaluation
  - Used for text summarization
  - Compares machine generated summarizations to human generated ones
- BLEU Score
  - BiLingual Evaluation Understudy
  - Used for text translation
  - Compares machine generated translations to human generated ones

Large Language Models

Model evaluation metrics

We will discuss two evaluation metrics:

- ROUGE
  - Recall Oriented Understudy for Gisting Evaluation
  - Used for text summarization
  - Compares machine generated summarizations to human generated ones
- BLEU Score
  - BiLingual Evaluation Understudy
  - Used for text translation
  - Compares machine generated translations to human generated ones

To remind terminology:

- unigram = any single word in a sentence.
- bigram = two neighbouring words in a sentence.
- ngram = n neighbouring words in a sentence.

Large Language Models
Rouge

Human reference:
It is cold outside.

Generated text:
It is very cold outside.

$$Rouge1\ Recall = \frac{unigram\ matches}{unigrams\ in\ reference}$$

$$Rouge1\ Precision = \frac{unigram\ matches}{unigrams\ in\ output}$$

$$Rouge1\ F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Large Language Models

Rouge

Human reference:
It is cold outside.

Generated text:
It is very cold outside.

$$Rouge1\ Recall = \frac{unigram\ matches}{unigrams\ in\ reference}$$

$$Rouge1\ Precision = \frac{unigram\ matches}{unigrams\ in\ output}$$

$$Rouge1\ F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

In our example:

$$Rouge1\ Recall = \frac{4}{4} = 1$$

$$Rouge1\ Precision = \frac{4}{5} = 0.8$$

$$Rouge1\ F1 = 2 \cdot \frac{1 \cdot 0.8}{1 + 0.8} = 0.89$$

Large Language Models

Rouge

Human reference:
It is cold outside.

Generated text:
It is very cold outside.

$$\textit{Rouge2 Recall} = \frac{\textit{bigram matches}}{\textit{bigrams in reference}}$$

$$\textit{Rouge2 Precision} = \frac{\textit{bigram matches}}{\textit{bigrams in output}}$$

$$\textit{Rouge2 F1} = 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

In our example:

$$\textit{Rouge2 Recall} = \frac{2}{3} = 0.67$$

$$\textit{Rouge2 Precision} = \frac{2}{4} = 0.5$$

$$\textit{Rouge2 F1} = 2 \cdot \frac{0.67 \cdot 0.5}{0.67 + 0.5} = 0.57$$

Large Language Models
Rouge

Rather than extending Rouge to ngrams, let us take different approach.

$$LCS = \textit{longest common subsequence}$$

Human reference:
It is cold outside.

Generated text:
It is very cold outside.

$$\textit{RougeL Recall} = \frac{LCS(\textit{reference}, \textit{generated})}{\textit{unigrams in reference}}$$

$$\textit{RougeL Precision} = \frac{LCS(\textit{reference}, \textit{generated})}{\textit{unigrams in output}}$$

$$\textit{RougeL F1} = 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

In our example:

$$\textit{RougeL Recall} = \frac{2}{4} = 0.5$$

$$\textit{RougeL Precision} = \frac{2}{5} = 0.4$$

$$\textit{RougeL F1} = 2 \cdot \frac{0.5 \cdot 0.4}{0.5 + 0.4} = 0.44$$

Large Language Models

Rouge hacking

Consider following situations:

Human reference:
It is cold outside.

Generated text:
cold cold cold cold

Large Language Models

Rouge hacking

Consider following situations:

Human reference:
It is cold outside.

Generated text:
cold cold cold cold

Precision will be very high. To improve on that, we can apply clipping:

$$\textit{RougeL Precision} = \frac{\textit{clip}(\textit{unigram matches})}{\textit{unigrams in output}}$$

Large Language Models

Rouge hacking

Consider following situations:

Human reference:
It is cold outside.

Generated text:
cold cold cold cold

Precision will be very high. To improve on that, we can apply clipping:

$$\textit{RougeL Precision} = \frac{\textit{clip}(\textit{unigram matches})}{\textit{unigrams in output}}$$

Still, following output will have a perfect precision.

Generated text:
outside cold it is

Large Language Models
BLEU

$$BLEU = AVG(presicion\ across\ range\ of\ ngrams\ sizes) \tag{89}$$

To calculate BLEU metric, use implementations from various libraries.

Large Language Models
BLEU

$$BLEU = AVG(presicion\ across\ range\ of\ ngrams\ sizes) \tag{89}$$

To calculate BLEU metric, use implementations from various libraries.

Example:

- Reference:
  I am very happy to say that I am drinking a warm cup of tea.
- Output
  I am very happy that I am drinking a cup of tea. - BLEU 0.495
  I am very happy that I am drinking a warm cup of tea. - BLEU 0.730
  I am very happy to say that I am drinking a warm tea. - BLEU 0.798

Large Language Models
Benchmarks

Both BLUE and ROUGE are relatively simple metrics to be used for diagnostics during model development.

To compare model performance with other LLM models use evaluation benchmarks developed by the researchers.

- GLUE - General Language Understanding Evaluation (2018)
  `https://gluebenchmark.com/leaderboard`
- SuperGLUE (2019)
  `https://super.gluebenchmark.com/leaderboard`
- MMLU - Massive Multitask Language Understanding (2021)
  `https://paperswithcode.com/dataset/mmlu`
- BIG-bench - Beyond the Imitation Game benchmark (2022)
  3 versions based on sizes - BIG-bench Hard, BIG-bench, Lite
- HELM - Holistic Evaluation of Language Models (2022)
  `https://crfm.stanford.edu/helm/lite/latest/#/leaderboard`