高等机器学习

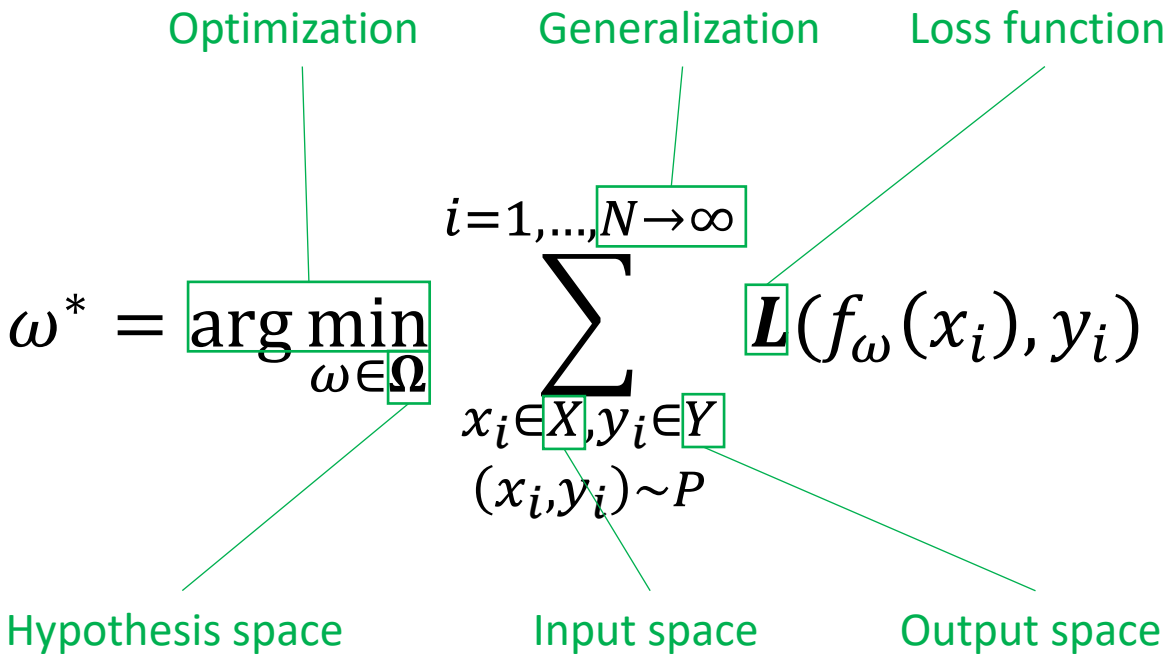
深度学习基础

张卫强

Microsoft

清華大學
Tsinghua University

Last Class: Formulation for Supervised Learning



Last Class: ML Models

- Machine learning models
  - Linear regression
  - Neural networks
  - Decision trees
  - Support vector machines
  - Boosting
- Model ensemble

Last Class: ML Theory

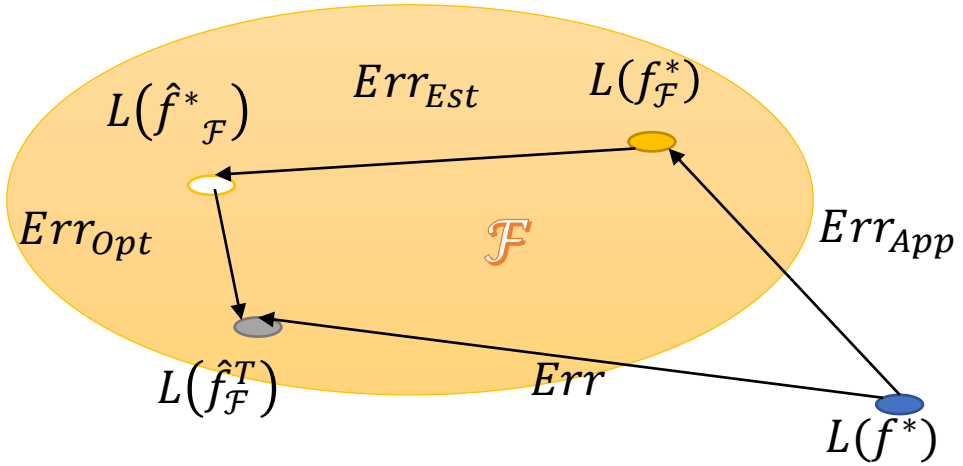
Excess risk:

Optimization Error

Estimation Error
(Generalization Error)

Approximation Error

$$L(\hat{f}_{\mathcal{F}}^{T}) - L(f^*) = \left(L(\hat{f}_{\mathcal{F}}^{T}) - L(\hat{f}^*_{\mathcal{F}})\right) + \left(L(\hat{f}^*_{\mathcal{F}}) - L(f_{\mathcal{F}}^*)\right) + \left(L(f_{\mathcal{F}}^*) - L(f^*)\right)$$

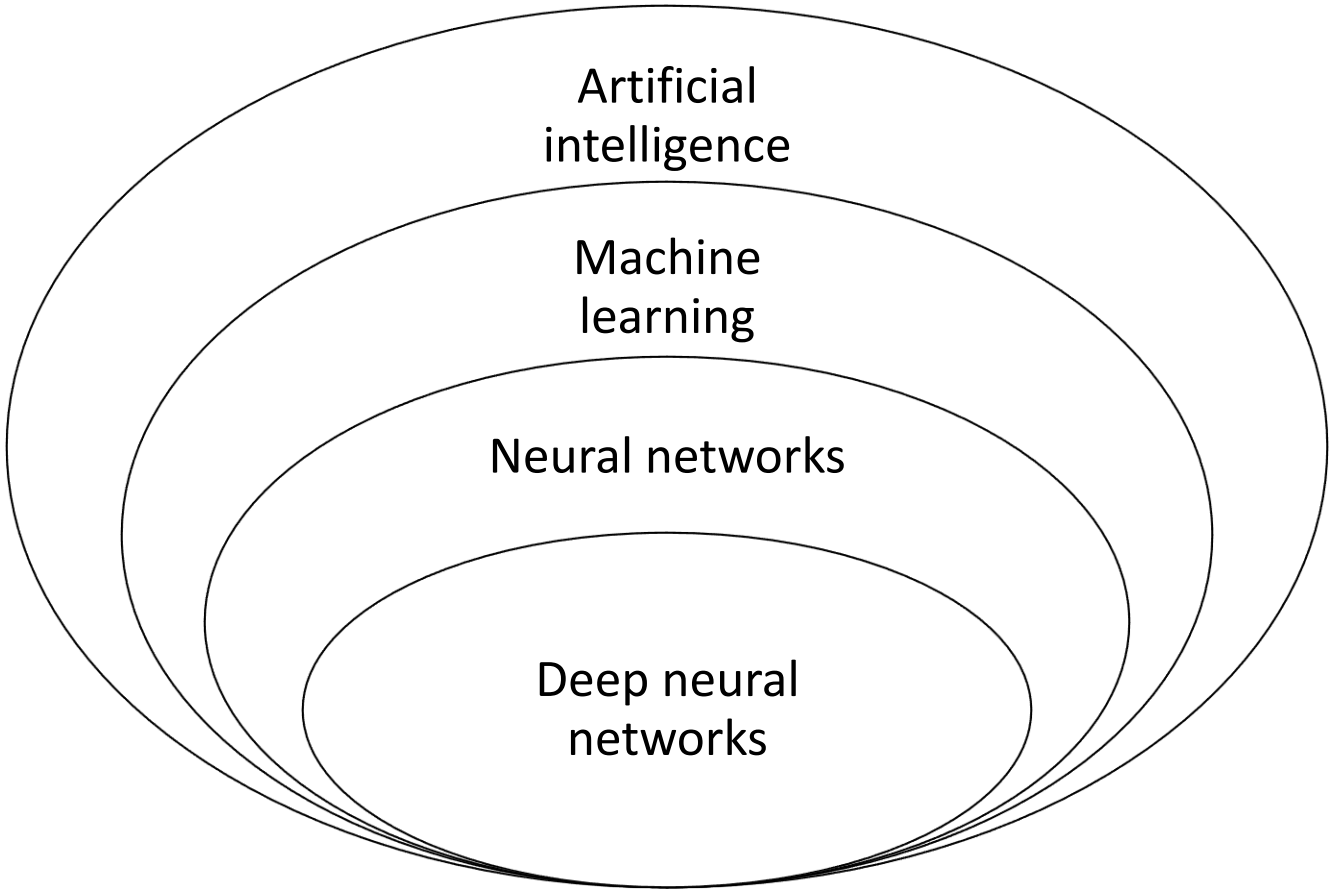


Last Class: Diagnostics for learning

- A better approach: Diagnose the possible problem based on the observation.

Train loss	Validation loss	Test loss	Diagnosis
high	high	high	?
low	Low	low	done
low	high	high	?
low	low	high	?

This Class: Deep Neural Networks



Ups and Downs of Neural Networks

1958

Perceptron

Frank Rosenblatt

Linear Model

Critisim

Marvin Minsky

Perceptron cannot learn the XOR function

1969

1980s

Multilayer Perceptron

At least three layers

Backpropagatior

David Rumelhart, Geoffrey Hinton, Ronald Williams

1986

2006

RBM

Restricted Boltzman Machine

for MLP initialization

ASR

automatic speech recognition

2011

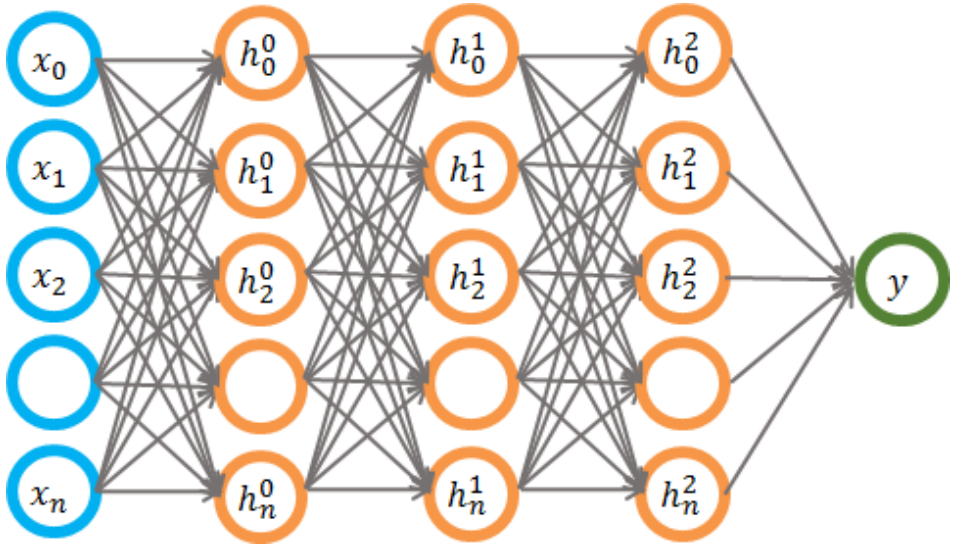
2012

AlexNet

Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E

Deep Neural Networks

- Fully-connected NN
- Convolutional NN
- Recurrent NN
- Transformer networks



- Optimization
- Model initialization
- Regularization
- Theory

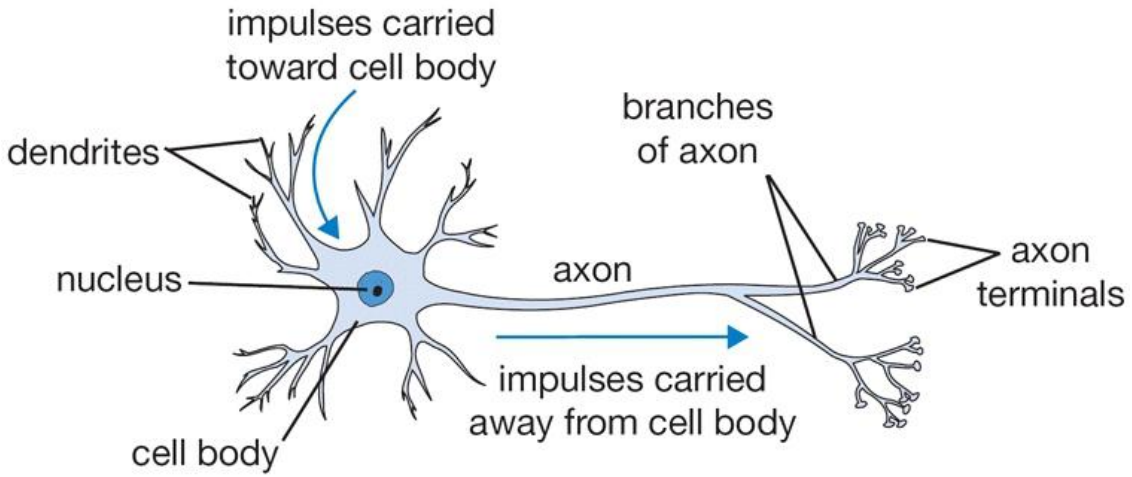
Outline

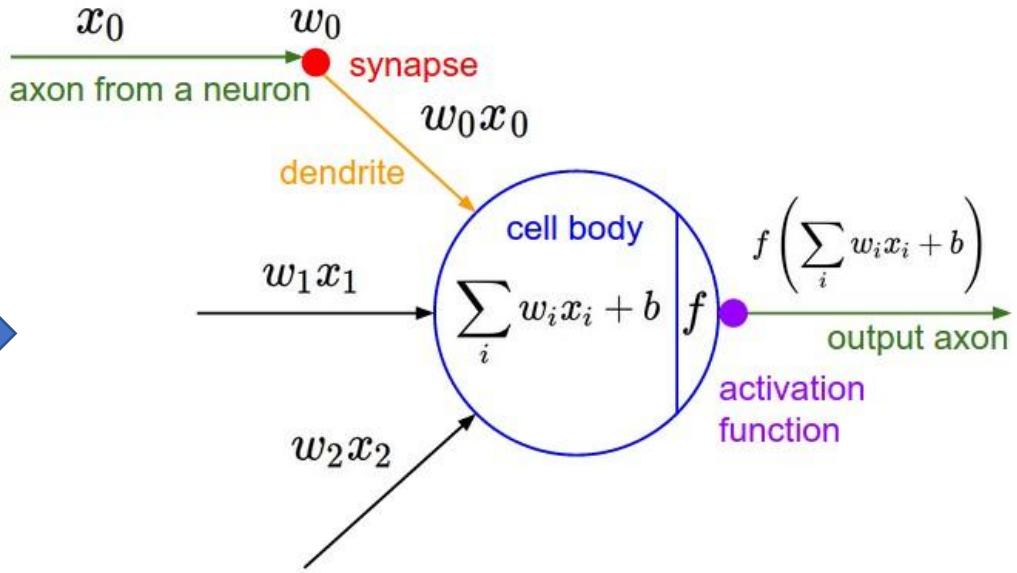
- DNN Basics
  - Feedforward networks (MLP)
  - Optimization
- DNN Models
  - Convolutional networks
  - Recurrent networks
  - Transformer networks
- Initialization and Regularization
- Theory

DNN Basics

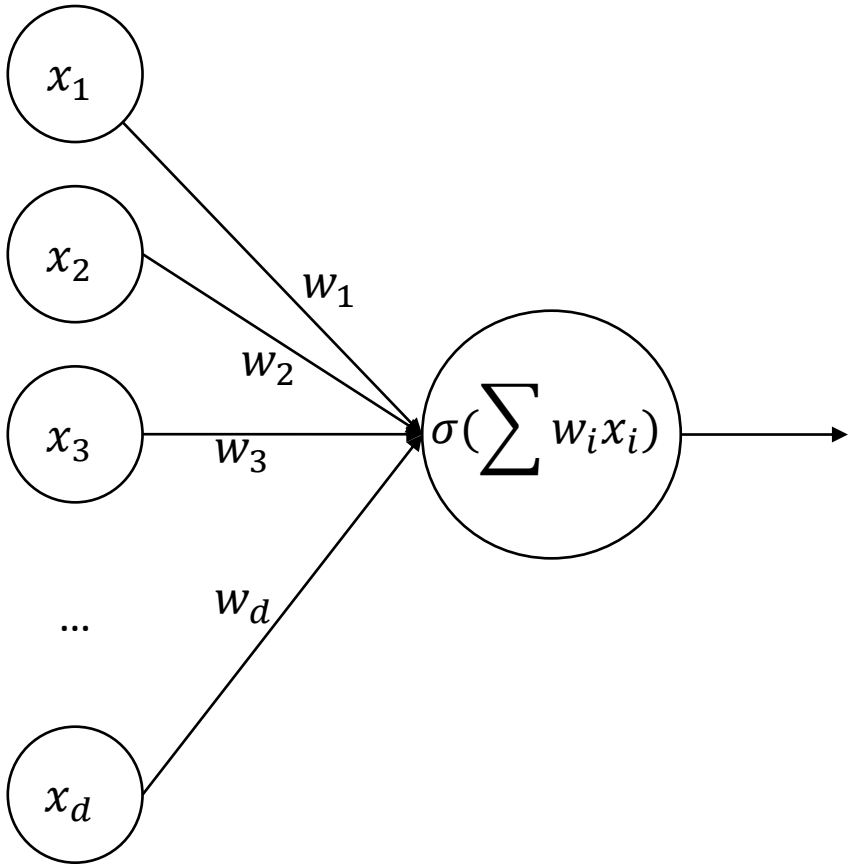
Biological Motivation and Connections

Dendrite， 树突，接收信号
Synapse，突触，放大信号
Axon，轴突，输出信号





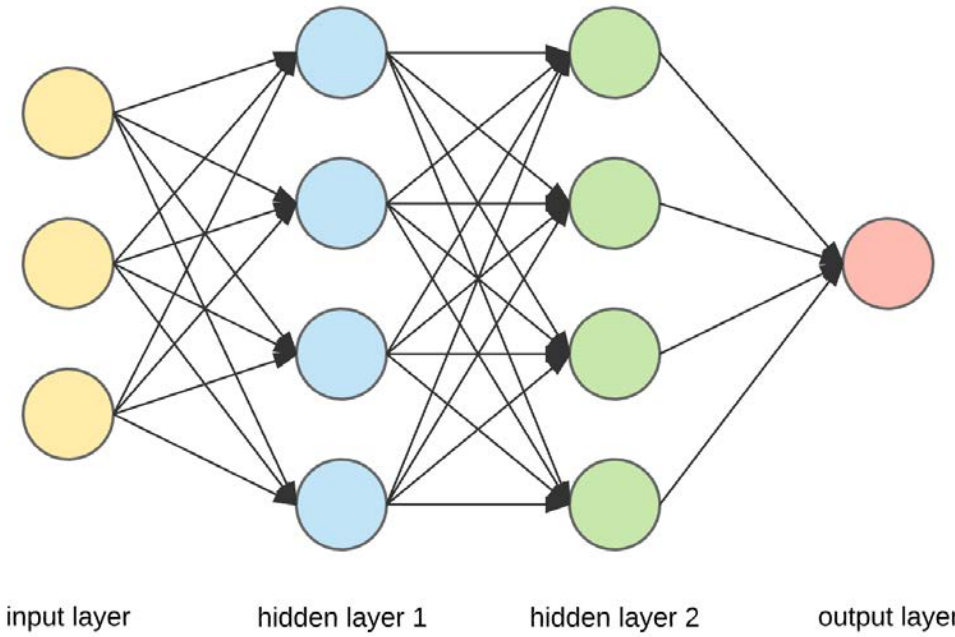
Perceptron



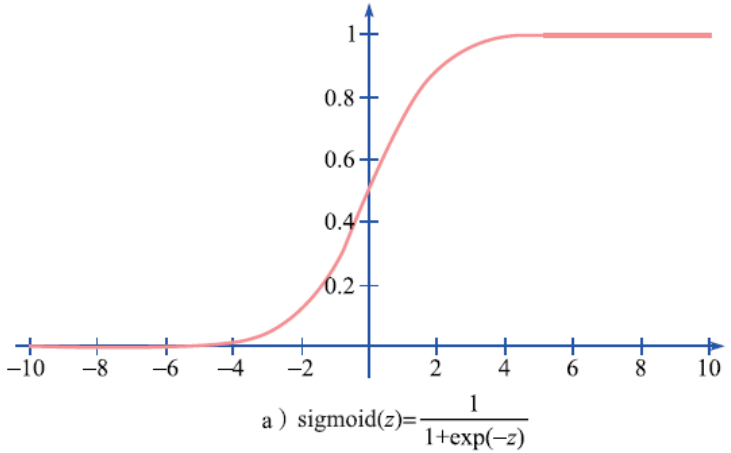
No hidden layer
A single output unit

Fully-Connected Neural Networks

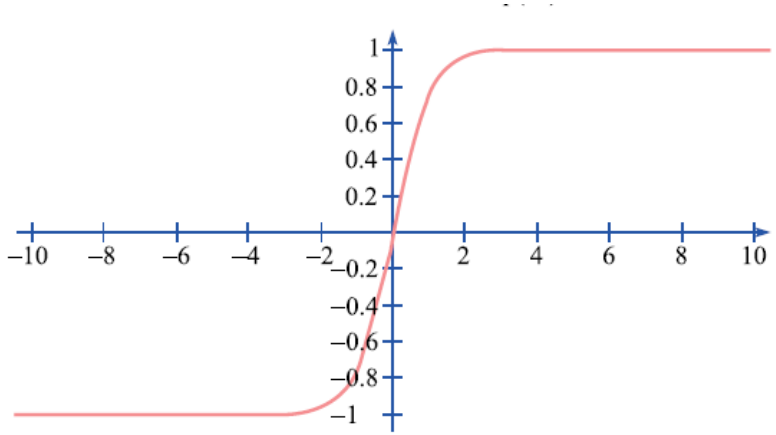
- Network structure: multiple hidden layers
- For each node: $y_i = f\left(\sum_j W_{ij} x_j + b_i\right) = f\left(W_i^T x + b_i\right)$, where W is the weight matrix and b is the bias vector



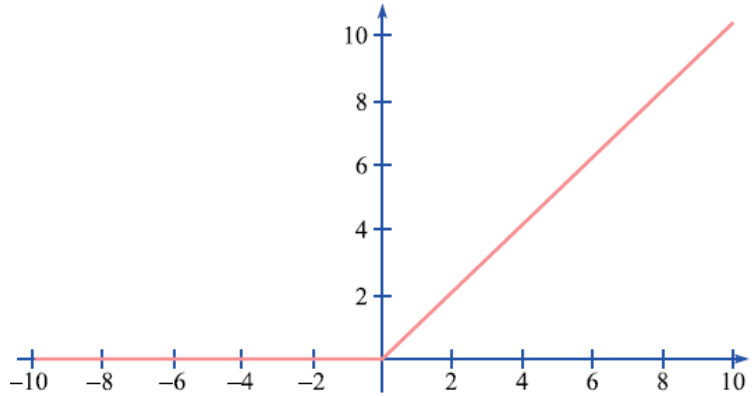
Activation Functions



a) $\text{sigmoid}(z)=\frac{1}{1+\exp(-z)}$



b) $\tanh(z)=2\cdot\text{sigmoid}(2z)-1$



c) $\text{ReLU}(z)=\max(0,z)$

Rectified linear units	$f(z)=\max(0,z)=\max(0,W^Tx+b)$	
Absolute value rectification	$f(z)=\max(0,z)+\alpha\min(0,z)$	$\alpha=-1$ ➔ $f(z)=\lvert z\rvert$
Leaky ReLU		fixes α to a small value like 0.01
Parametric ReLU		Learns α

Identity	Sigmoid	TanH	ArcTan
ReLU	Leaky ReLU	Randomized ReLU	Parameteric ReLU
Binary	Exponentional Linear Unit	Soft Sign	Inverse Square Root Unit (ISRU)
Inverse Square Root Linear	Square Non-Linearity	Bipolar ReLU	Soft Plus

https://mlfromscratch.com/optimizers-explained/#/

ReLU -> ELU, GELU

ELU: Exponential Linear Unit:

- a smooth approximation to the rectifier function
- Advantages: generates negative outputs

$$ELU(\alpha, x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$

GELU: Gaussian Error Linear Unit

- used in LM and transformer models like GPT-x and BERT
- no vanishing gradients
- a continuous derivative at 0, which can sometimes make training faster

$$\text{GELU}(x) = 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715x^3)\right)\right)$$

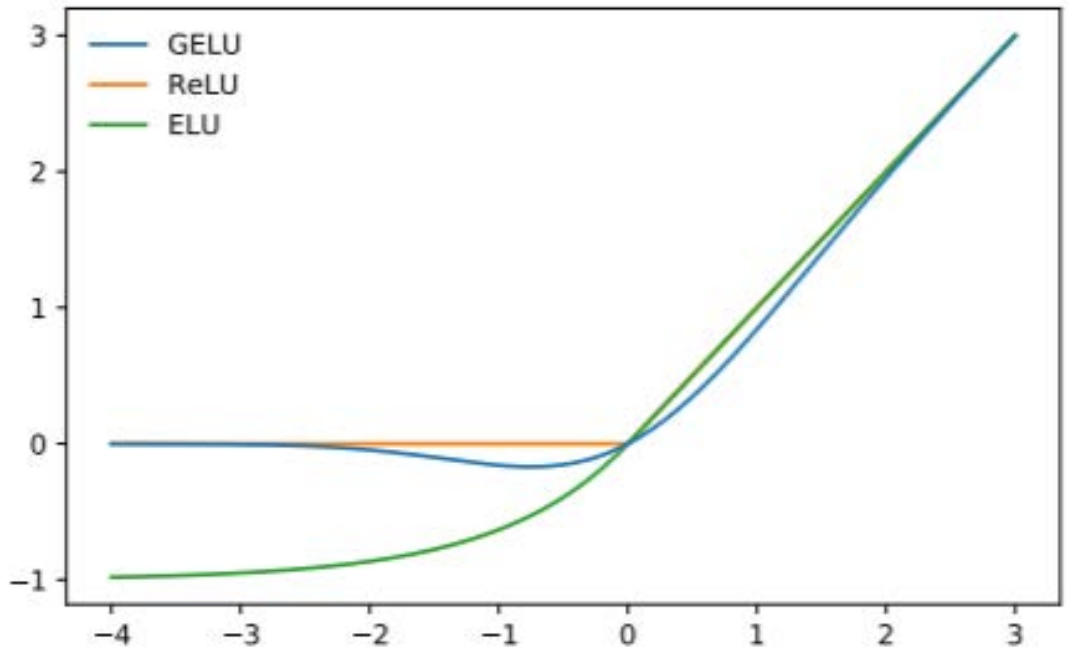


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

Output Units

- Linear output unites: $y = W^T h + b$
- Softmax operator for multi-class classification

$$\text{softmax}(y)_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

Representation Power

Universal Approximation Theorem

(Hornik 1989) Feedforward networks with only a single hidden layer (containing finite number of hidden nodes) can approximate any continuous function uniformly on any compact set and any measurable function arbitrarily well.

For $\forall\, f \in \mathcal{F}, \exists NN \in \mathcal{N}, s.t.$

$$\forall x \in [0,1]^d, |NN(x) - f(x)| \leq \epsilon.$$

Loss Function

- Loss function measures the goodness of a prediction, e.g., similarity or compatibility between a prediction (e.g. the class scores in classification) and the ground truth label

$$\text{Overall loss } L = \frac{1}{n}\sum_i l(f(x_i), y_i)$$

Classification loss

$$l(f(x_i), y_i) = \sum_{j \neq y_i} \max\left(0, f_j(x_i) - f_{y_i}(x_i) + 1\right)$$

$$l(f(x_i), y_i) = -\log \frac{e^{f_{y_i}(x_i)}}{\sum_j e^{f_j(x_i)}}$$

Regression loss

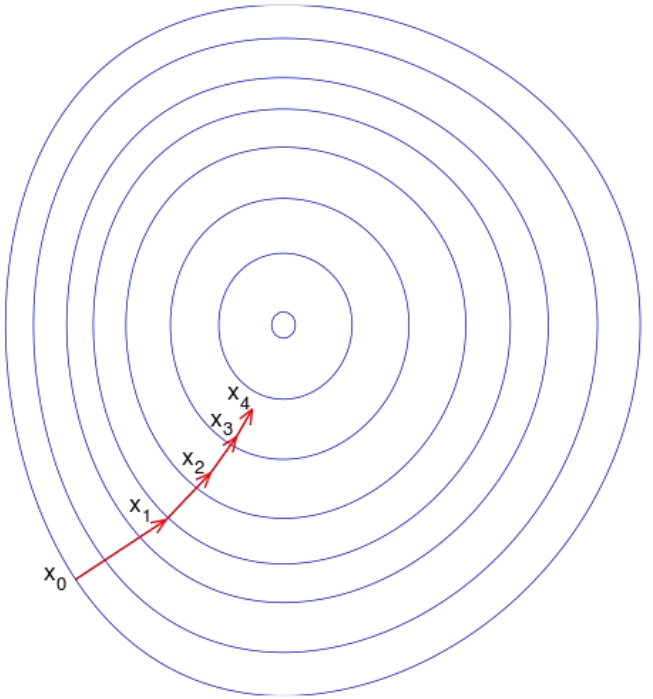
$$l(f(x_i), y_i) = \left|\left|f(x_i) - y_i\right|\right|_2^2$$

$$l(f(x_i), y_i) = \left|\left|f(x_i) - y_i\right|\right|_1$$

Gradient Descent

- A first-order iterative optimization algorithm.
- To find a local minimum of a function
- Takes steps proportional to the negative of the gradient of the function at the current point

$$\theta \leftarrow \theta - \epsilon \nabla L(\theta)$$
$$= \theta - \epsilon \sum_i \Delta L_i(\theta)$$



Disadvantage: huge cost while computing a single gradient in the case of large-scale machine learning problems.

Stochastic Gradient Descent (SGD)

SGD: Sweeps through the training set, computes gradient, and performs update for each training example

$$\theta \leftarrow \theta - \epsilon_k \nabla L_k(\theta)$$

- A sufficient condition to ensure convergence

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \qquad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

- Step decay: decay learning rate by 0.5 every 5 epochs, by 0.1 every 20 epochs, or by validation error
- 1/t decay: $\epsilon_k = \epsilon_0/(1+kt)$

Disadvantage: large variance, not stable

Minibatch Stochastic Gradient Descent

Input: learning rate η_k and initial model parameter θ

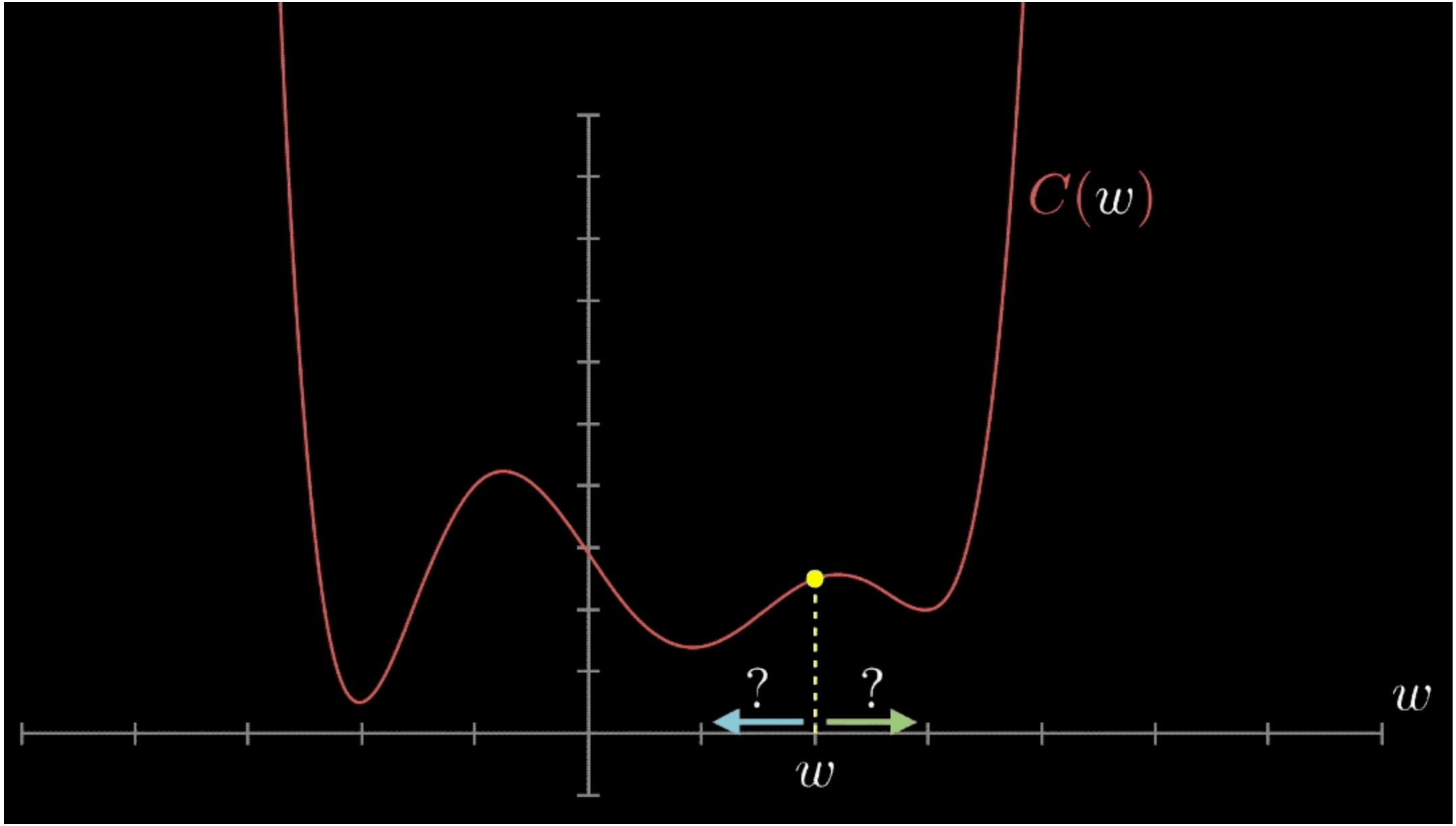
While stopping criterion not met **do**

Sample a minibatch of m samples $\left\{x^{(i)}, y^{(i)}\right\}_{i\in[m]}$ from the training data

Compute the gradient $g = \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} L(f\left(x^{(i)};\theta\right), y^{(i)})$

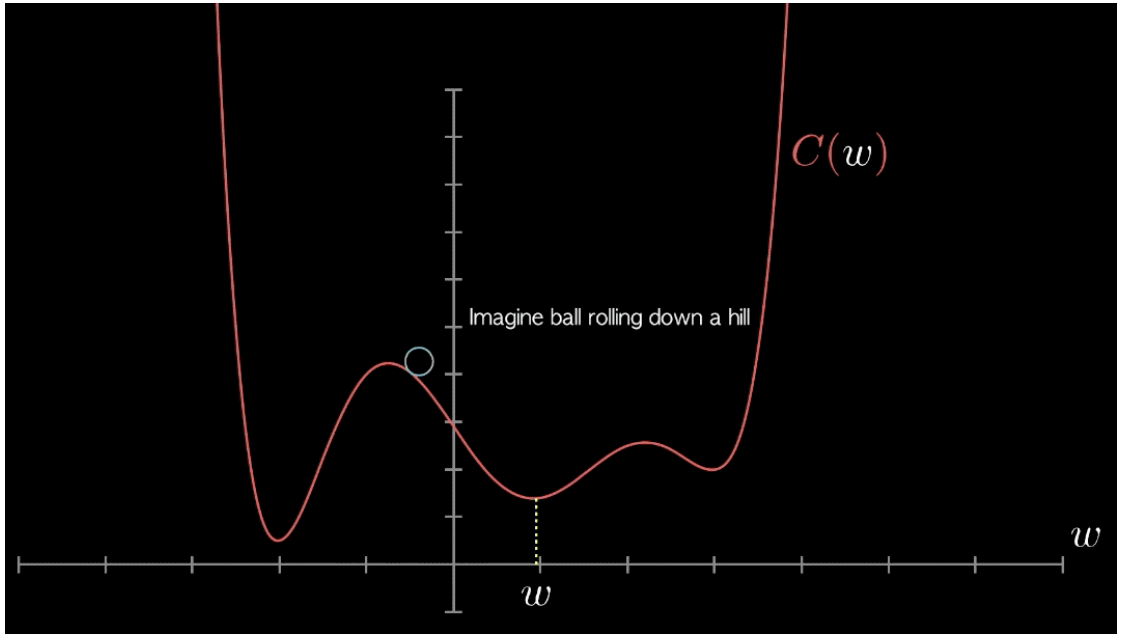
Update the model $\theta \leftarrow \theta - \epsilon_k g$

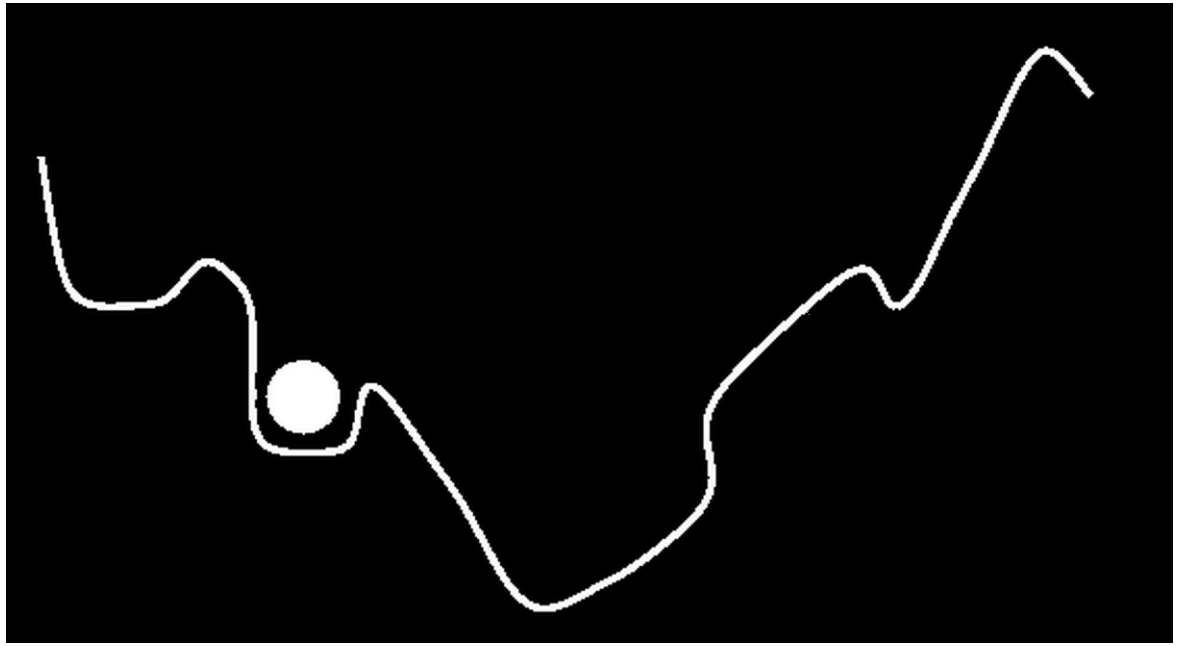
End while



https://mlfromscratch.com/optimizers-explained/#/

Limitations of SGD





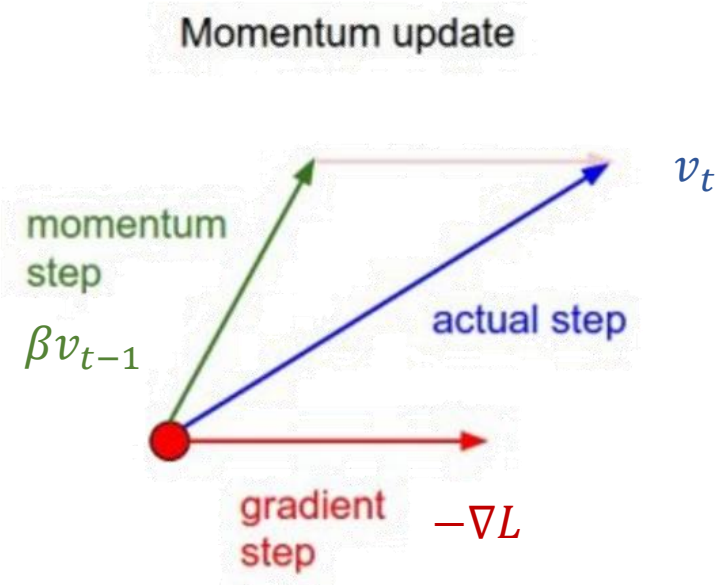
https://mlfromscratch.com/optimizers-explained/#/

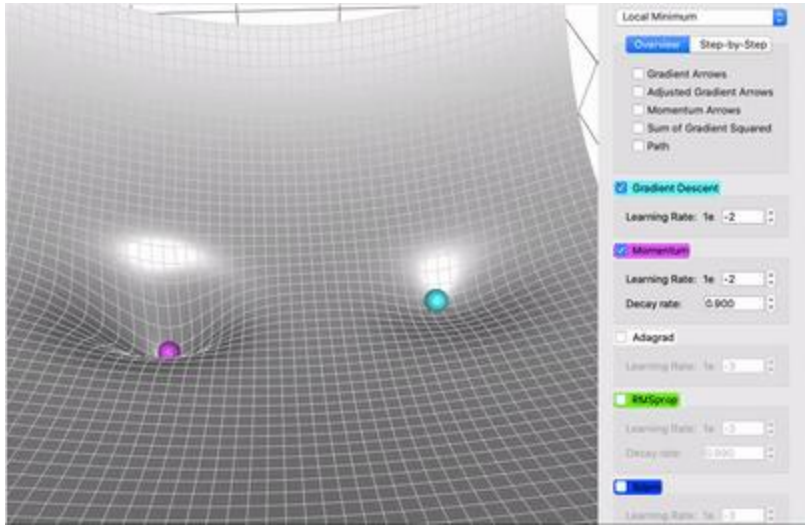
https://www.quora.com/profile/Shehroz-Khan-2

Momentum SGD

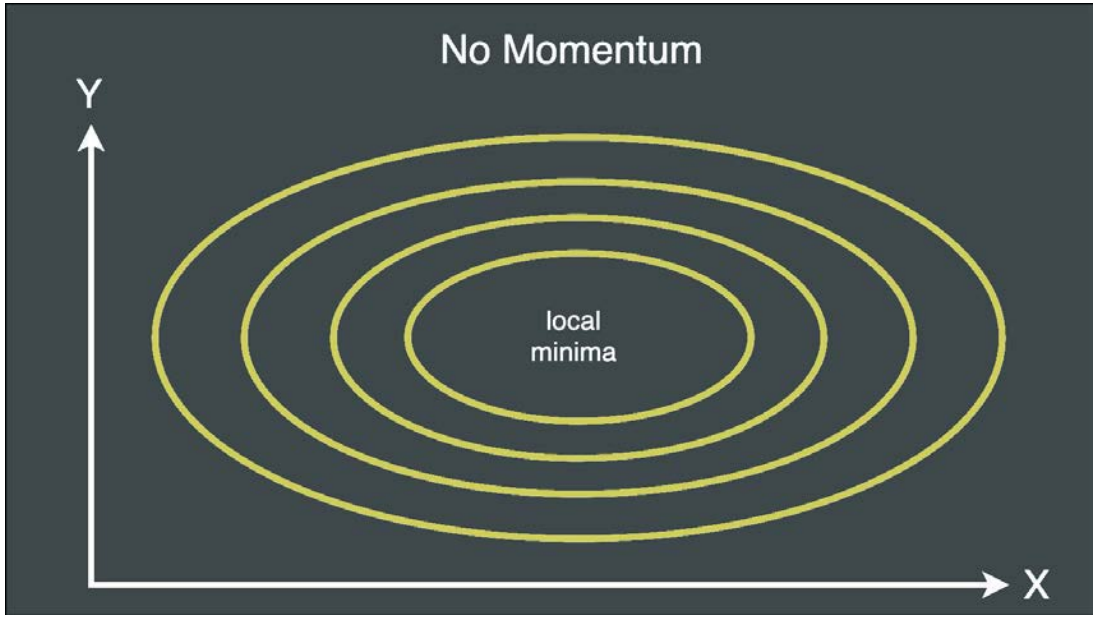
$$v_t = \beta v_{t-1} - \nabla L$$
$$\theta = \theta + \alpha v_t$$

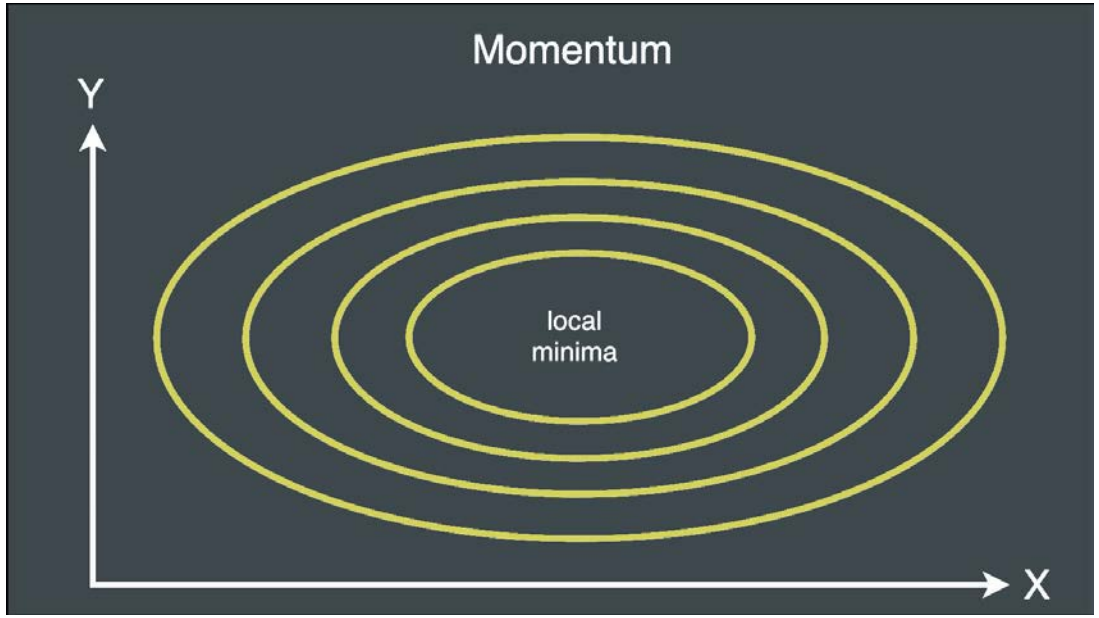
- Momentum is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients.
- The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.





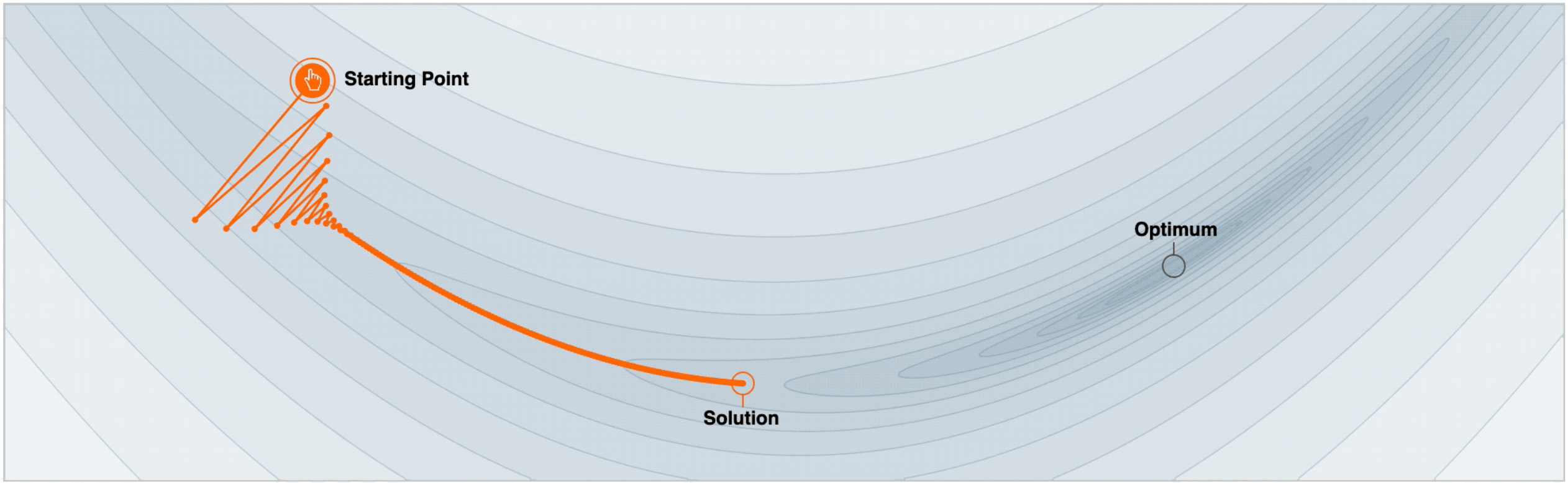
Momentum SGD: Faster Convergence





https://mlfromscratch.com/optimizers-explained/#/

$$v_t = \beta v_{t-1} - \nabla L$$
$$\theta = \theta + \alpha v_t$$



Step-size α = 0.0030

0 0.003 0.006

Momentum β = 0.0

0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

image credit: https://mlfromscratch.com/optimizers-explained/#/

https://distill.pub/2017/momentum/

Nesterov Momentum

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right),$$

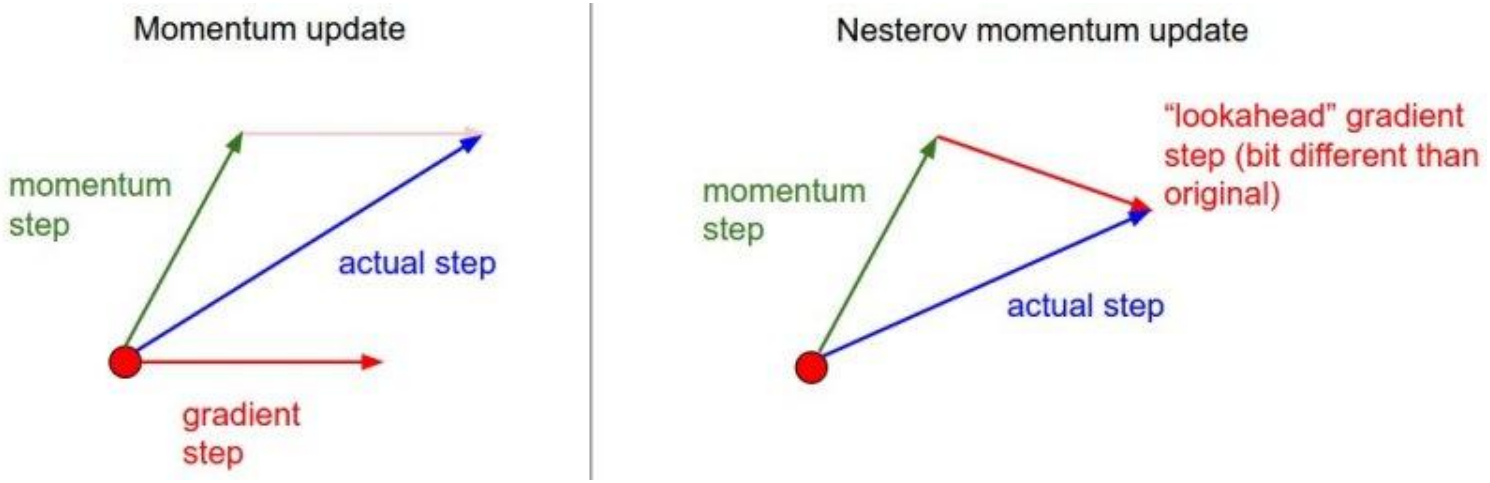
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

Momentum

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right],$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

Nesterov momentum



AdaGrad

[PDF] **Adaptive subgradient methods** for online learning and stochastic optimization.

J Duchi, E Hazan, Y Singer - Journal of machine learning research, 2011 - jmlr.org

… We present a new family of **subgradient methods** that dynamically incorporate knowledge of the … **adaptive subgradient methods** outperform state-of-the-art, yet non-**adaptive**, **subgradient** …

☆ Save 99 Cite Cited by 11573 Related articles All 22 versions »

- Different parameters, different learning rates
- Individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the second order momentum.
- The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

It's famous for not being published, yet being very well-known

RMSProp

- AdaGrad shrinks the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving at such a convex structure
- RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho\boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$

Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta+\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta+\boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

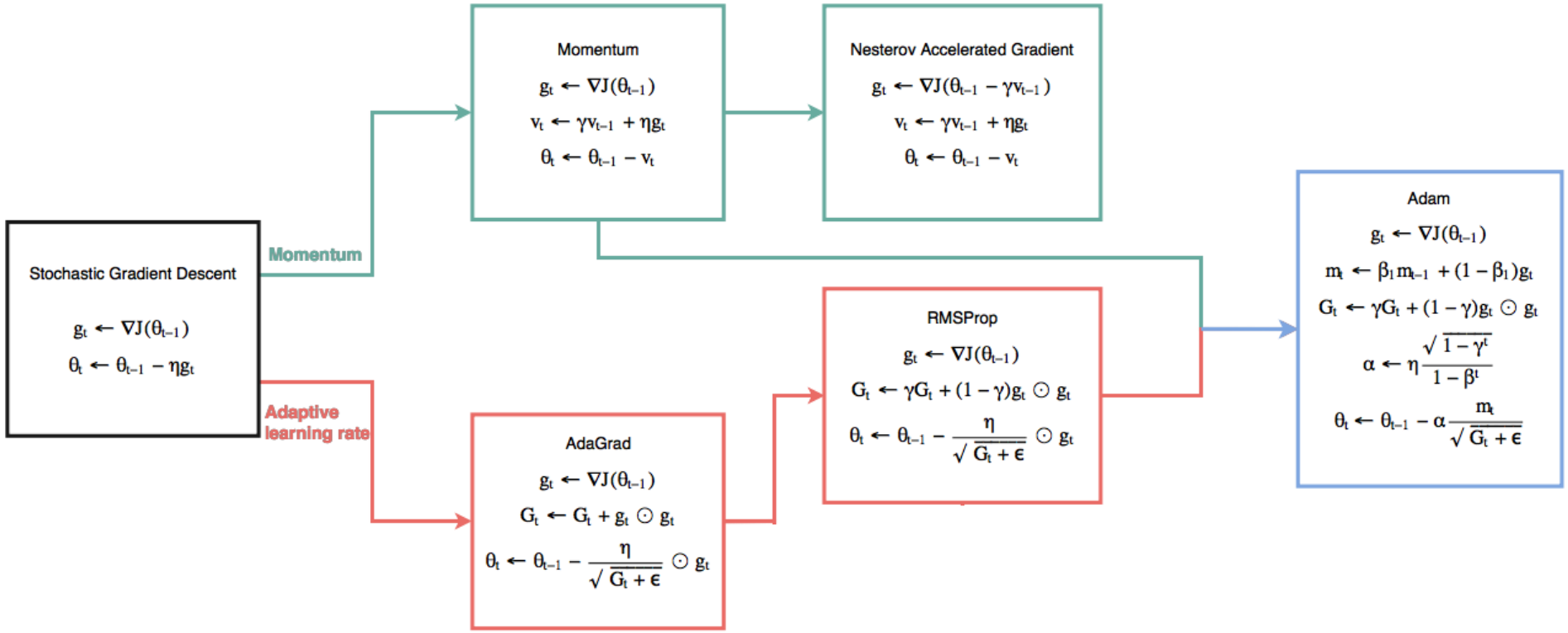
Adam: A method for stochastic optimization
DP Kingma, J Ba - arXiv preprint arXiv:1412.6980, 2014 - arxiv.org
… to first-order **methods**. We propose **Adam**, a **method** for efficient **stochastic optimization** that only … The **method** computes individual adaptive learning rates for different parameters from …
☆ Save 99 Cite Cited by 137272 Related articles All 21 versions »

Adam

- An unbiased variant on the combination of RMSProp and momentum
- Combination of first and second order momentum

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t+1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

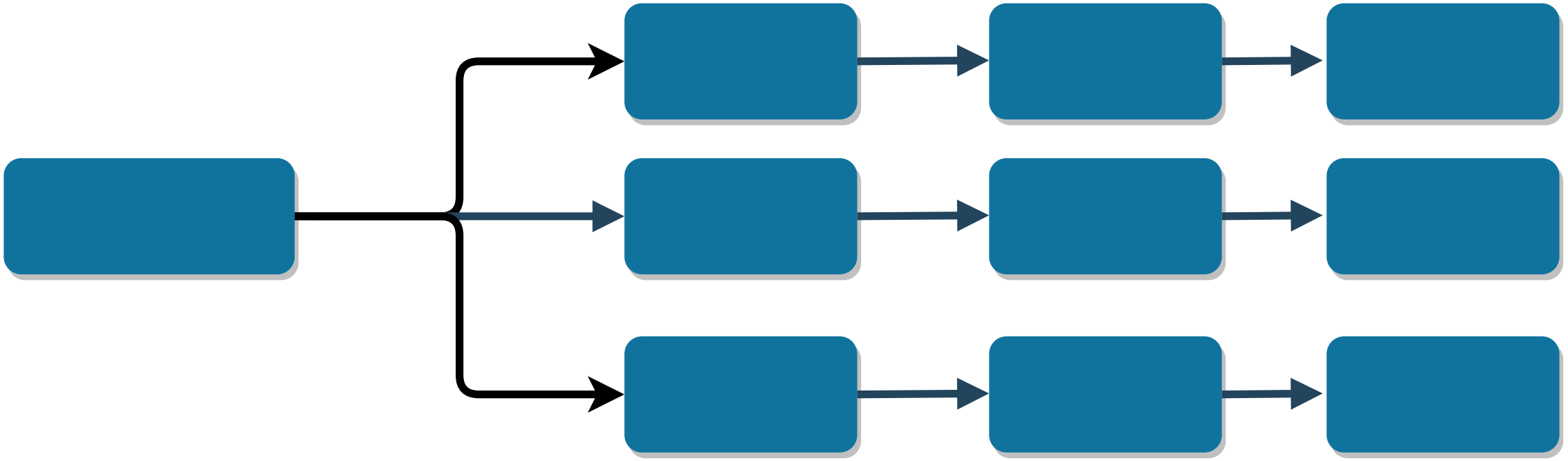
Summary about SGD Algorithms



A Simple Example

- https://pytorch.org/tutorials/index.html
- https://docs.microsoft.com/learn/paths/pytorch-fundamentals/?wt.mc_id=aiml-7486-cxa
- https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/361322e82f6f3158670d917a717329e1/intro.ipynb

DNN Models

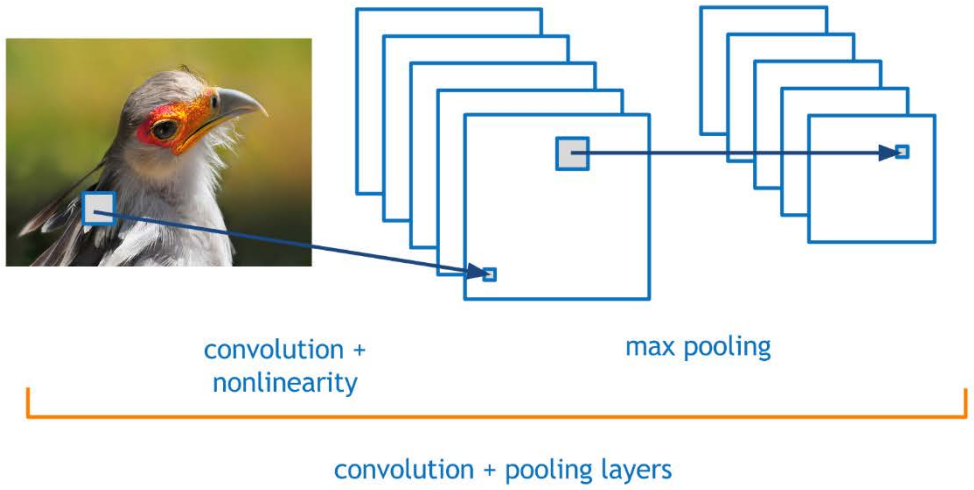


Convolutional Neural Networks (CNN)

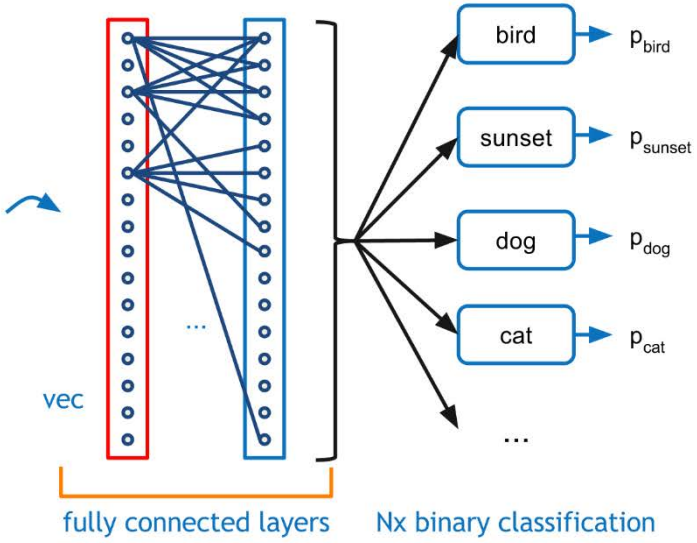
- Inspired by biology:
  - The visual cortex contains cells that are sensitive to small sub-regions, tiled to cover the entire visual field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.

Convolutional Neural Networks (CNN)

- Key ideas
  - Convolution
    - Weight sharing
  - Pooling

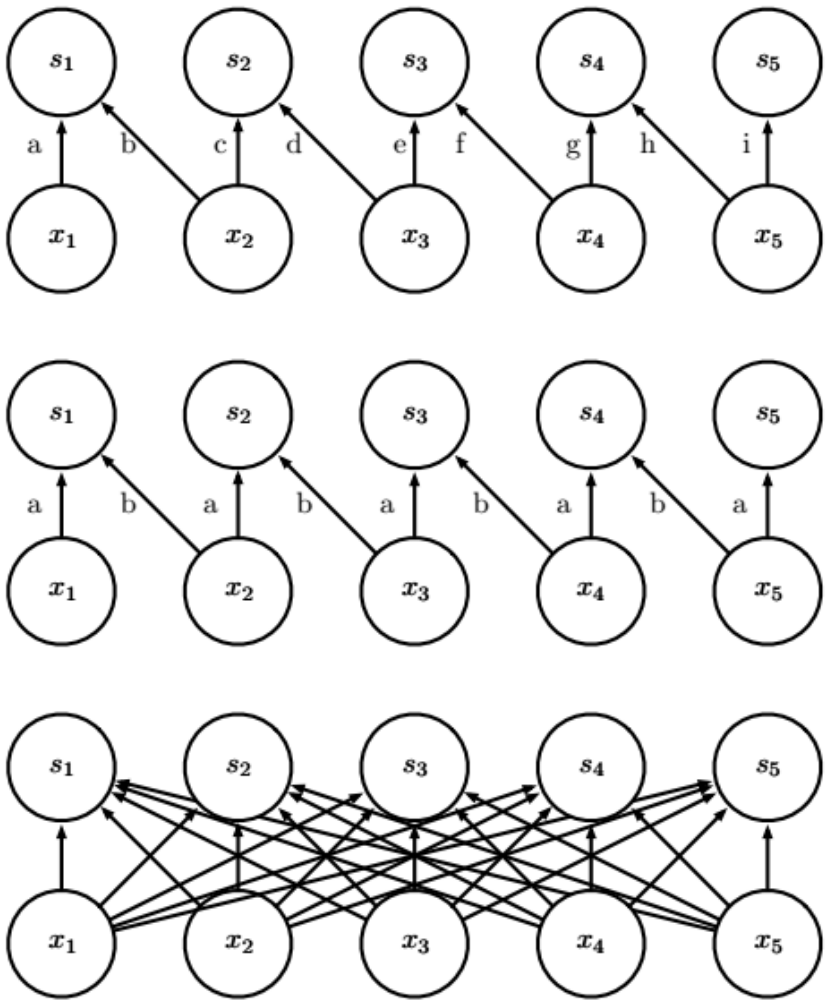


- Layers
  - Convolutional layers
  - Pooling layers
  - Fully connected layers



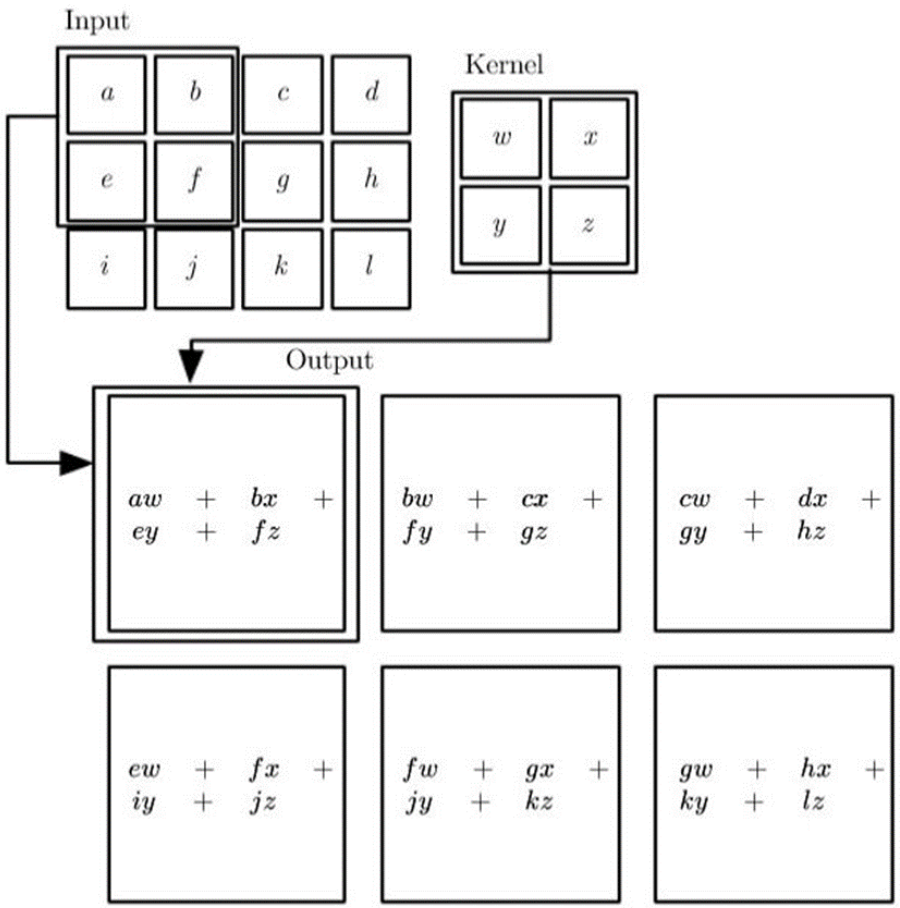
1D Convolution

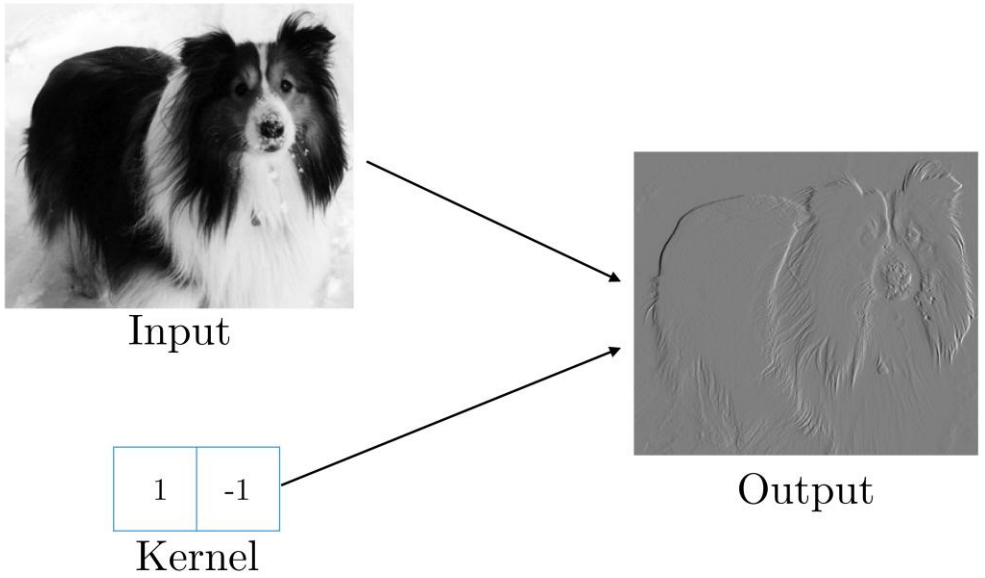
- Local connection
- Sparse connection
- Parameter sharing



(Goodfellow 2016)

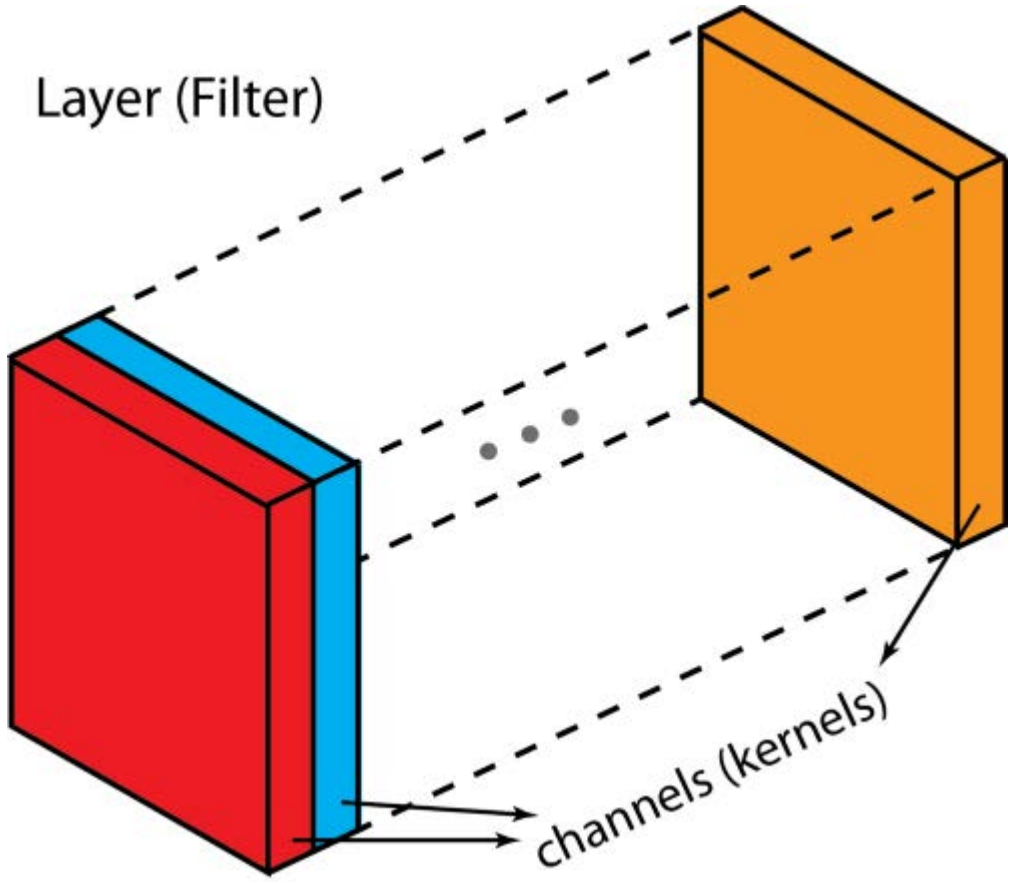
2D Convolution



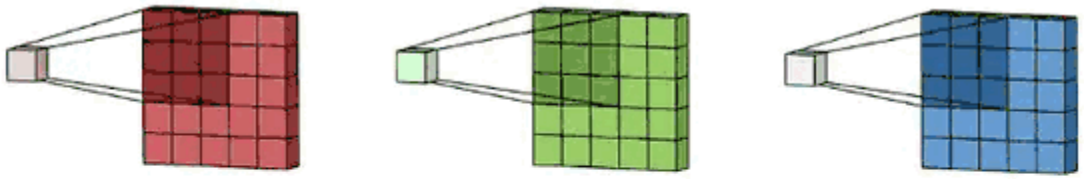


(Goodfellow 2016)

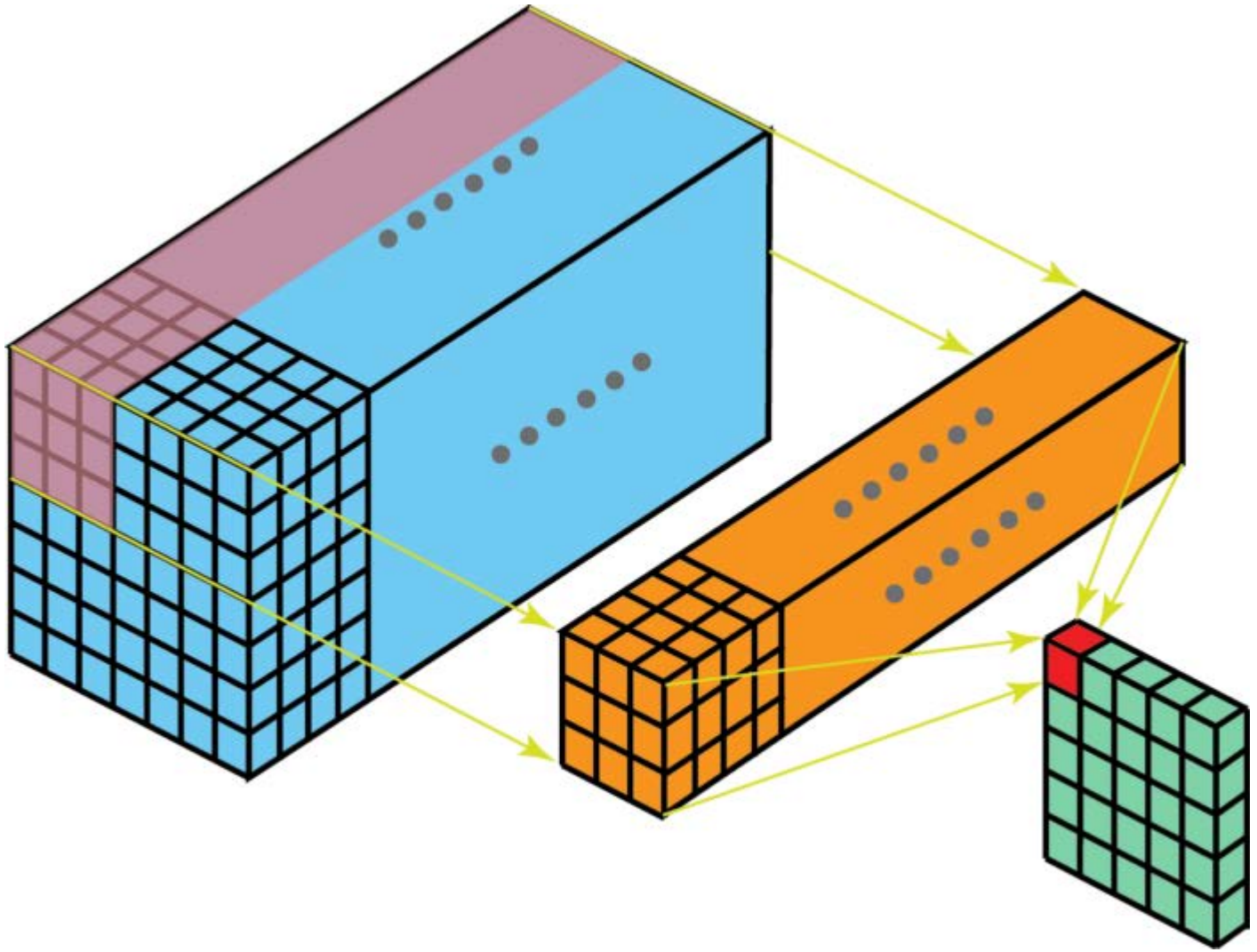
Multi-Channel Convolution



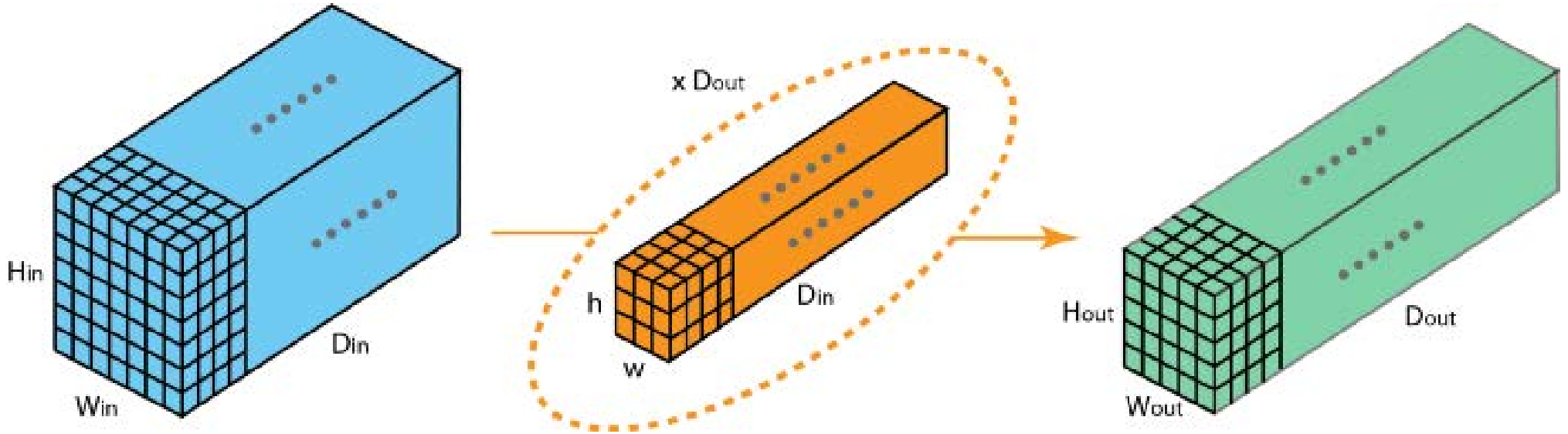
Multi-Channel Convolution



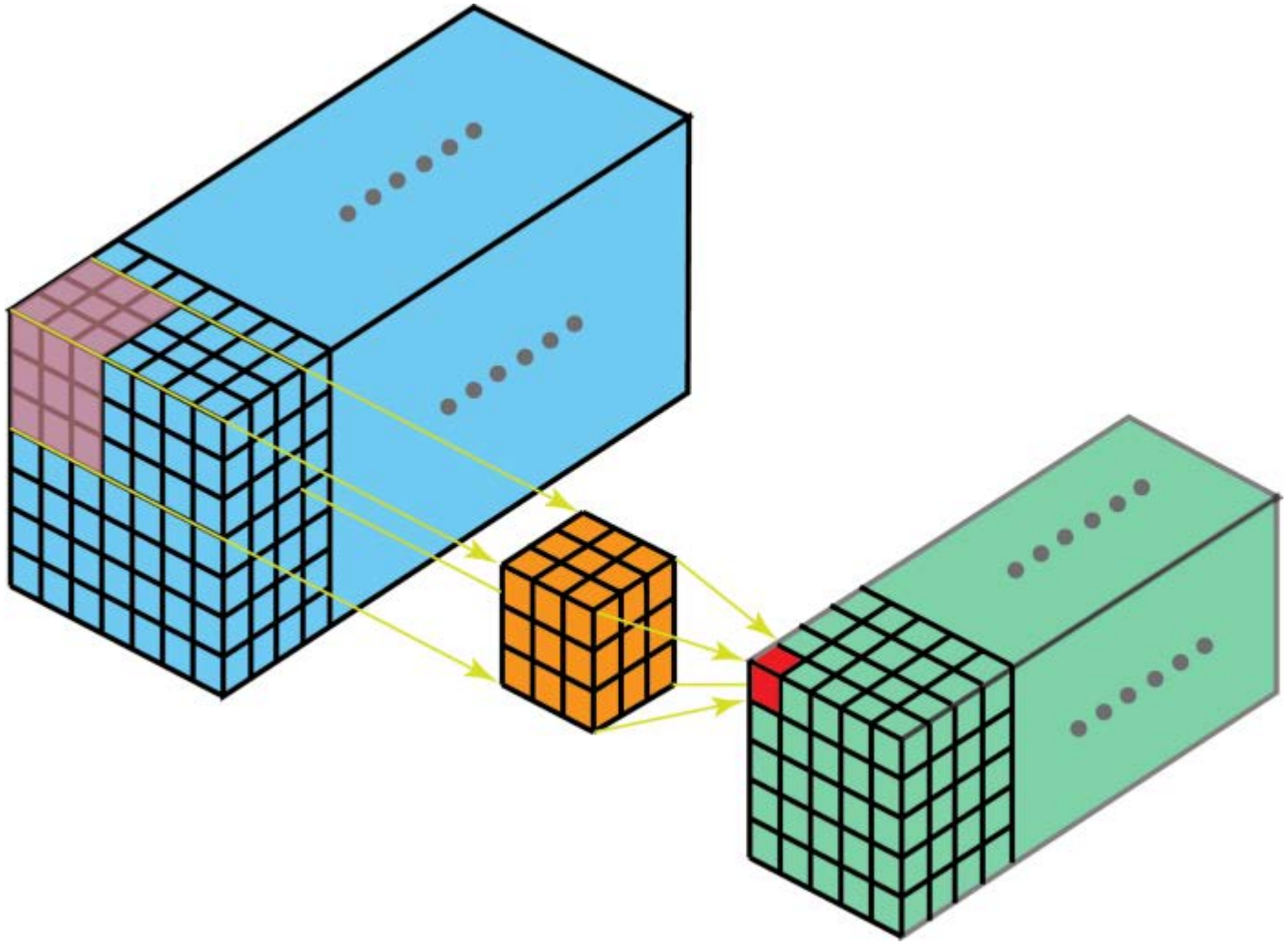
Multi-Channel Convolution



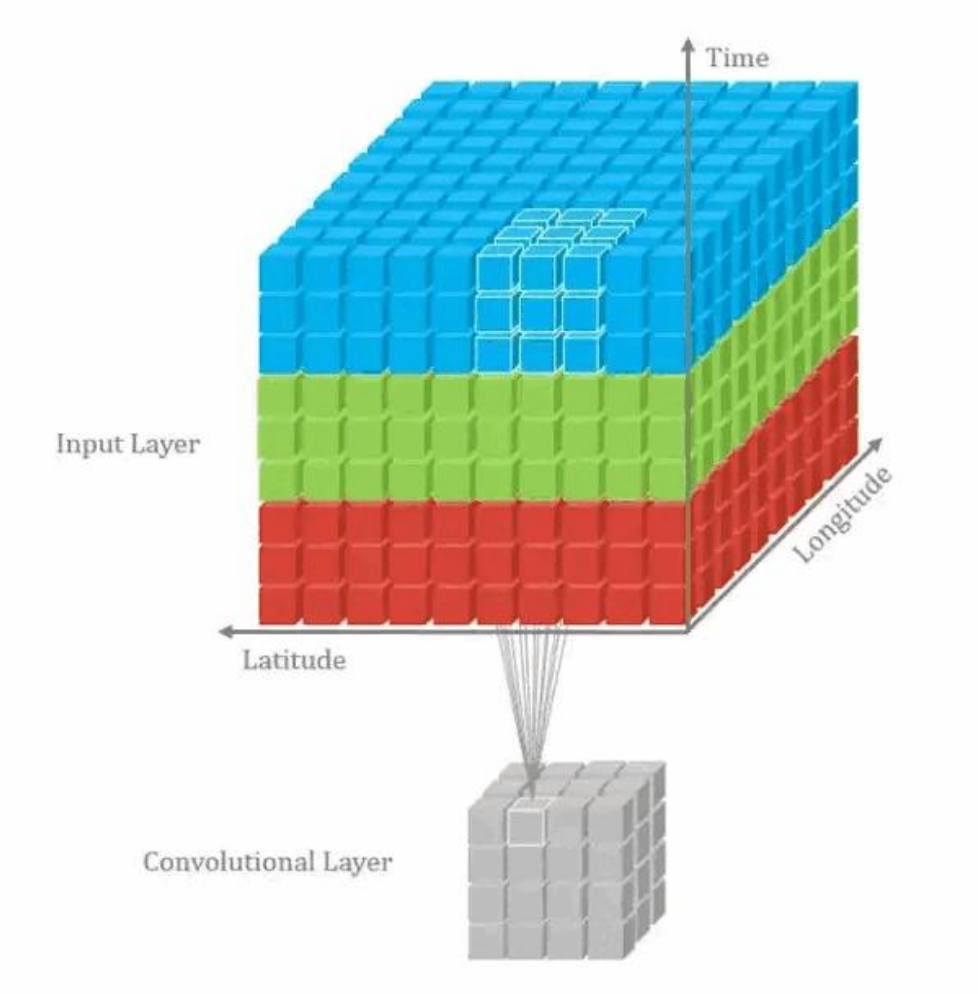
Multi-Channel Convolution



3D Convolution

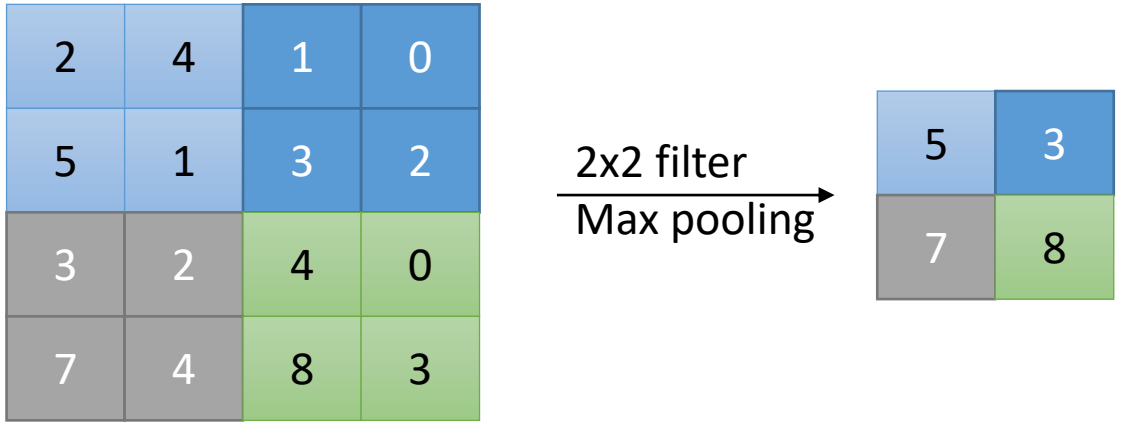


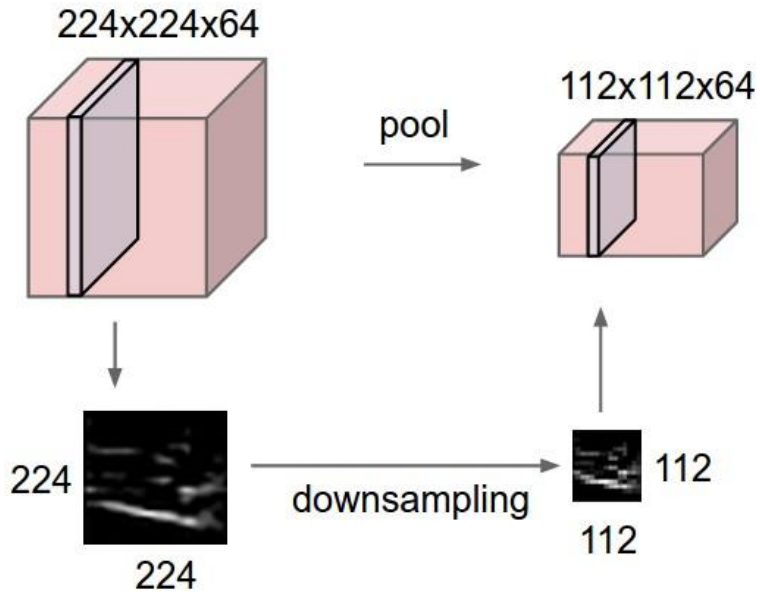
3D Convolution



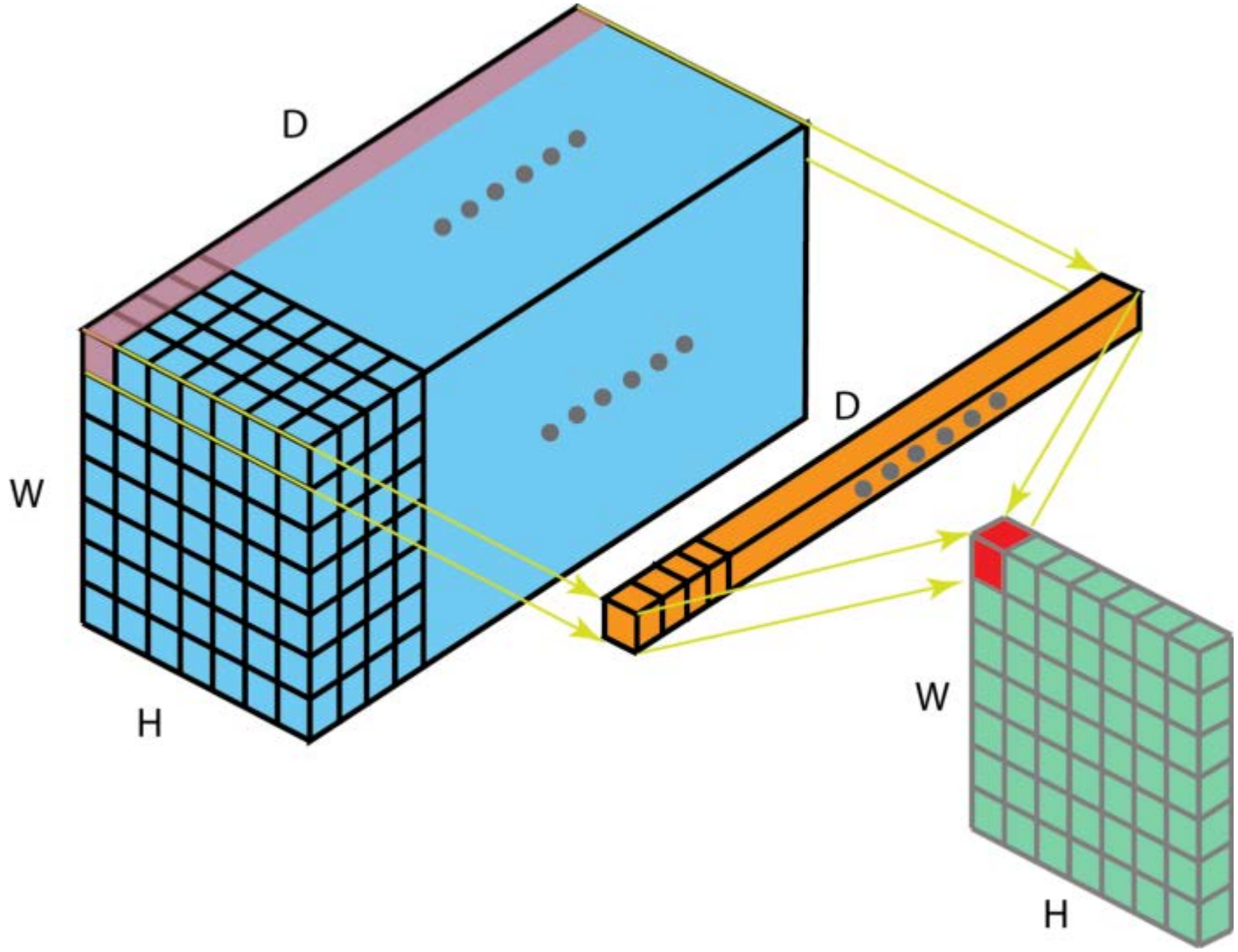
Pooling

- Reduce dimension
  - Control overfitting to some extent
- Achieve translation invariance
- Popular choice: MAX pooling

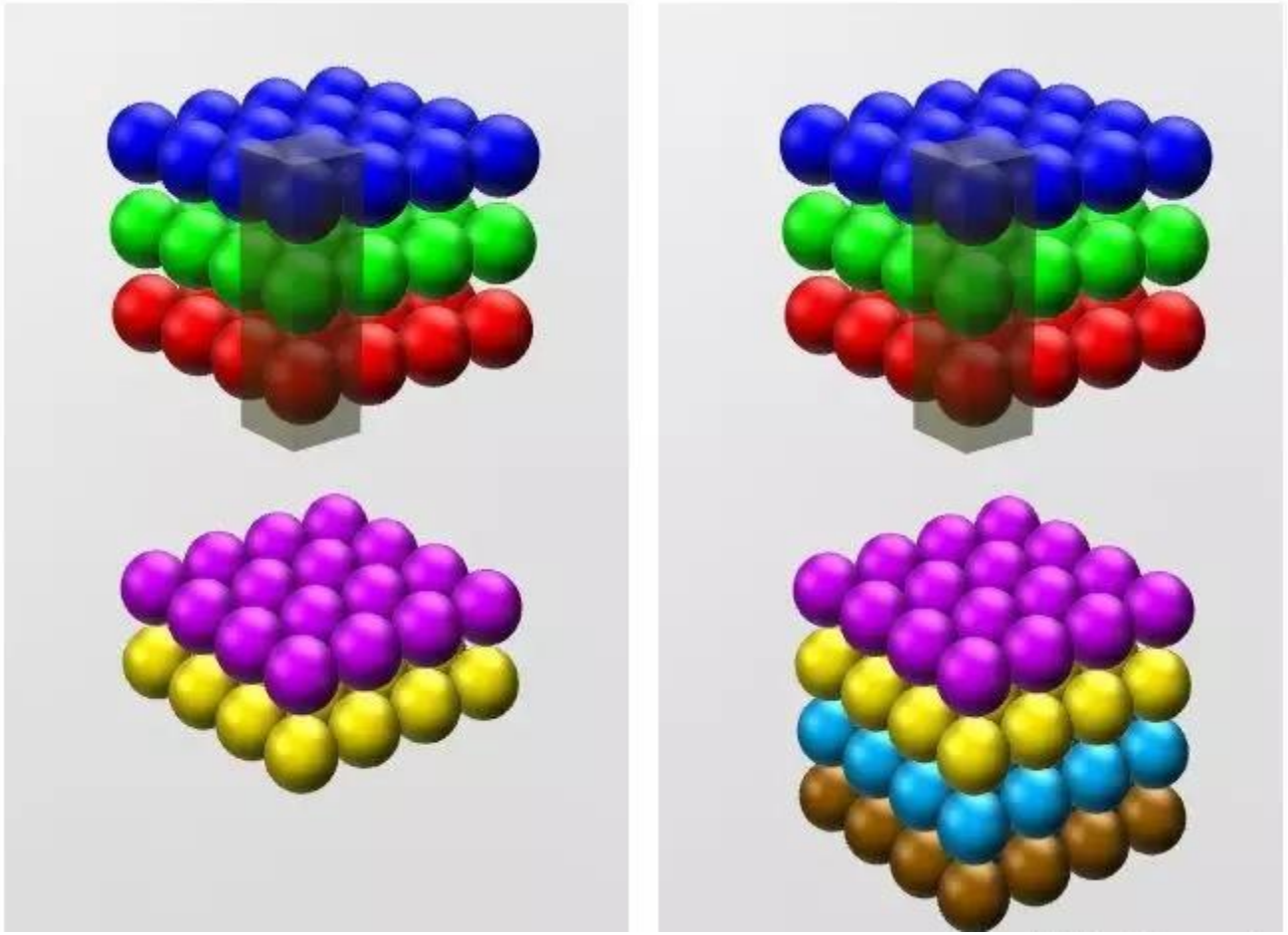




1x1 Convolution

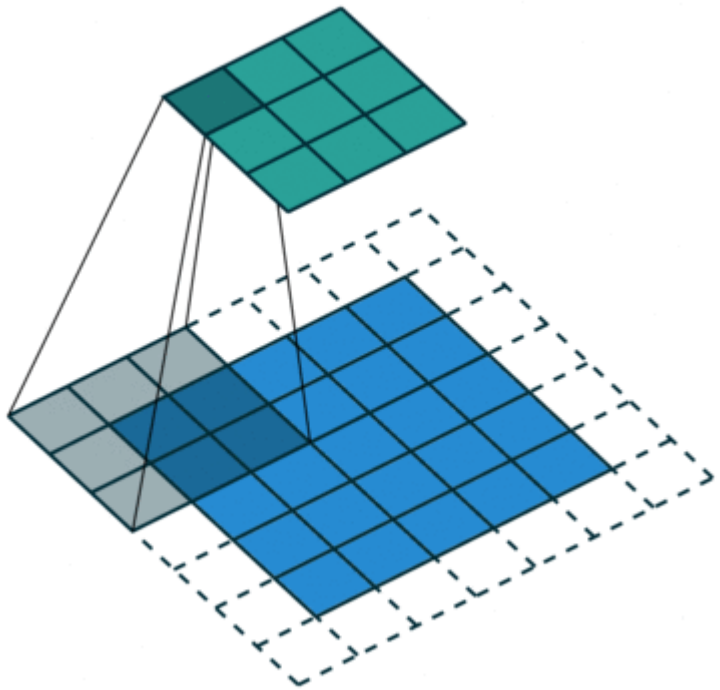


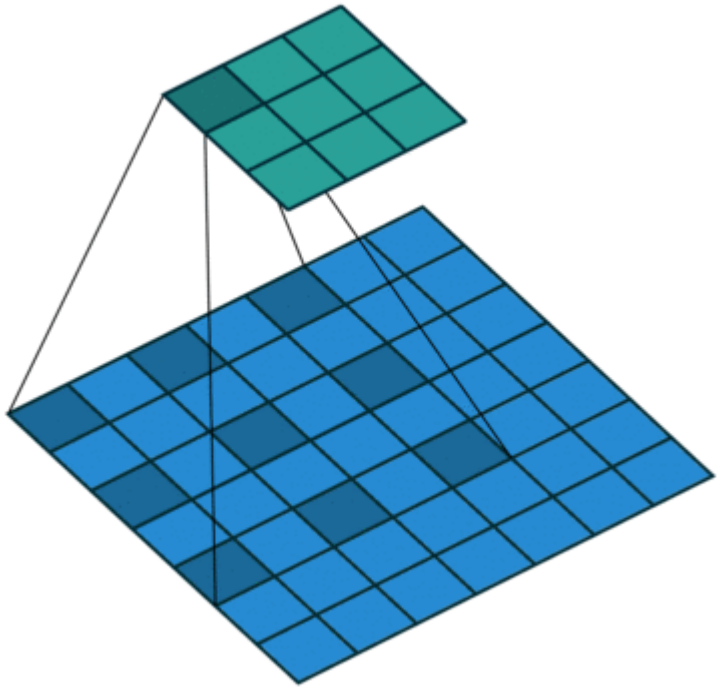
1x1 Convolution



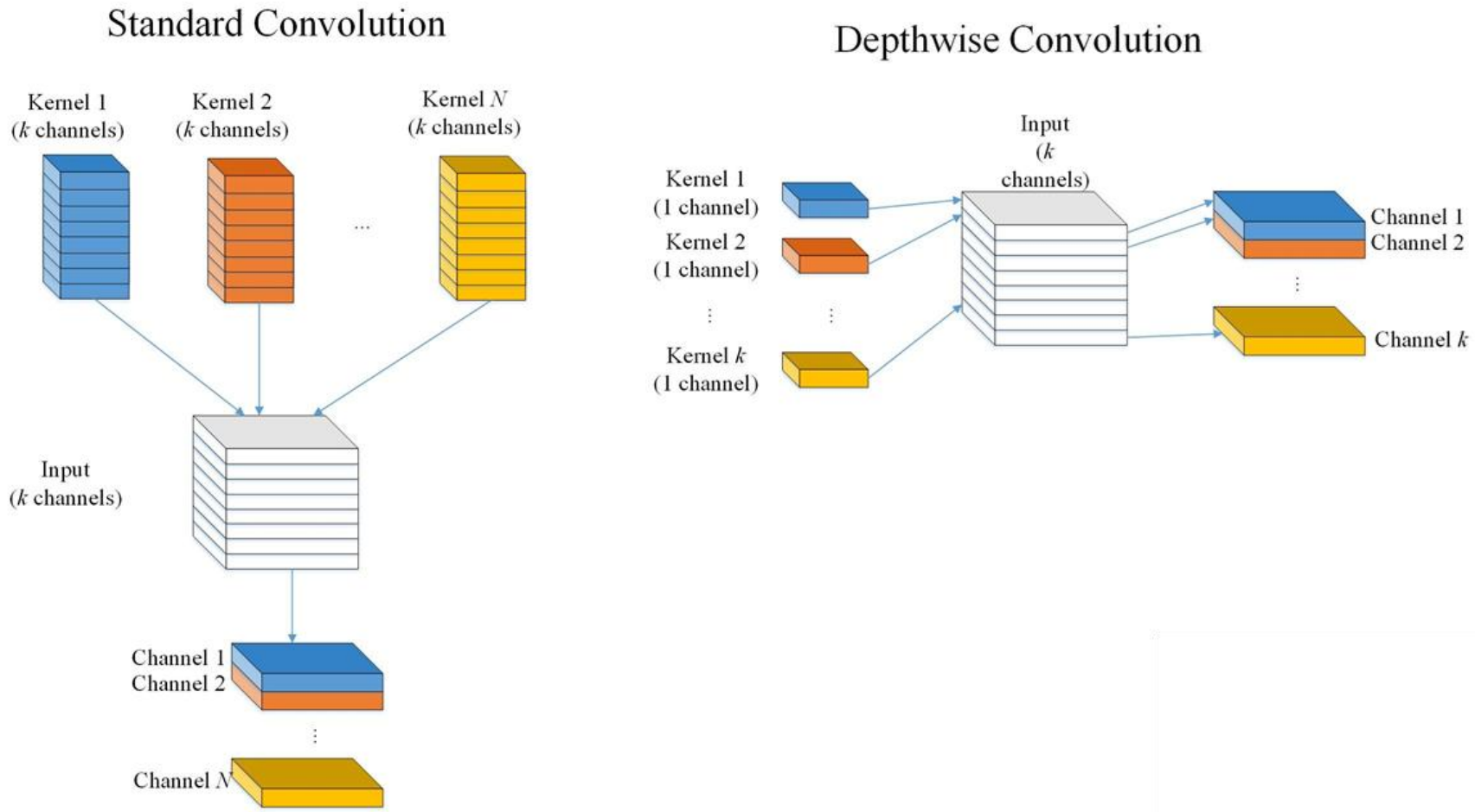
Dilated Convolution

Larger receptive field with less layers

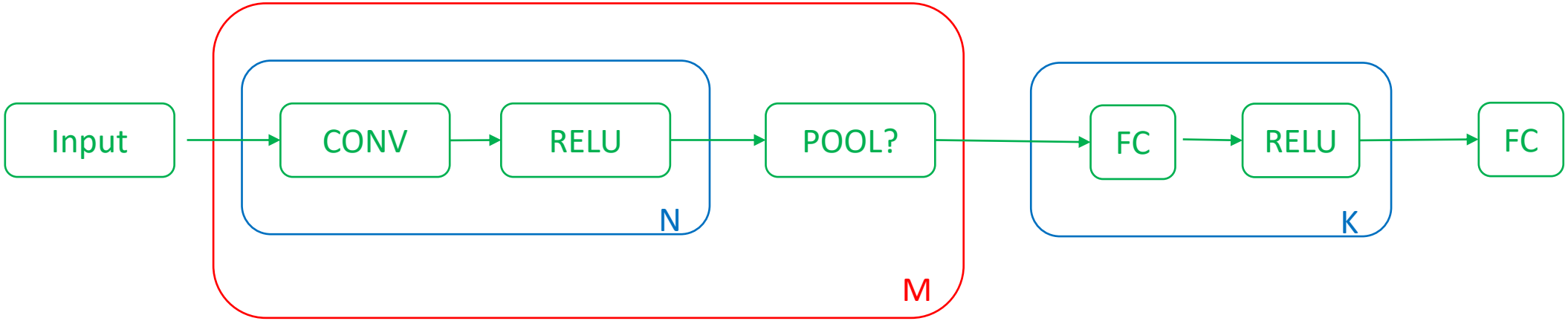




Depthwise Separable Convolution



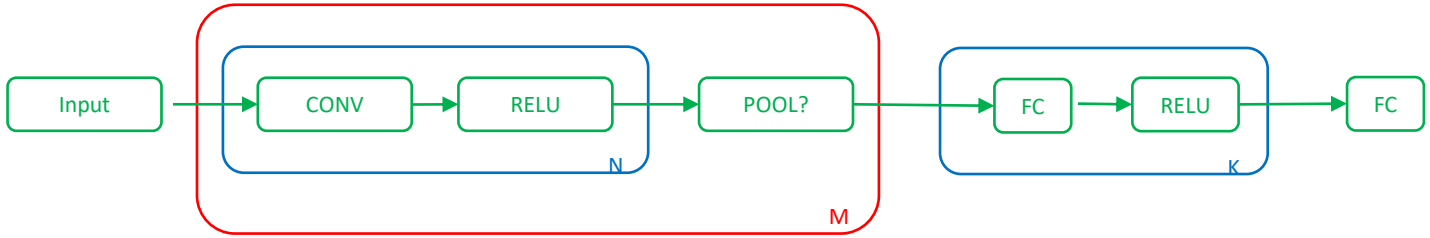
ConvNet Architectures

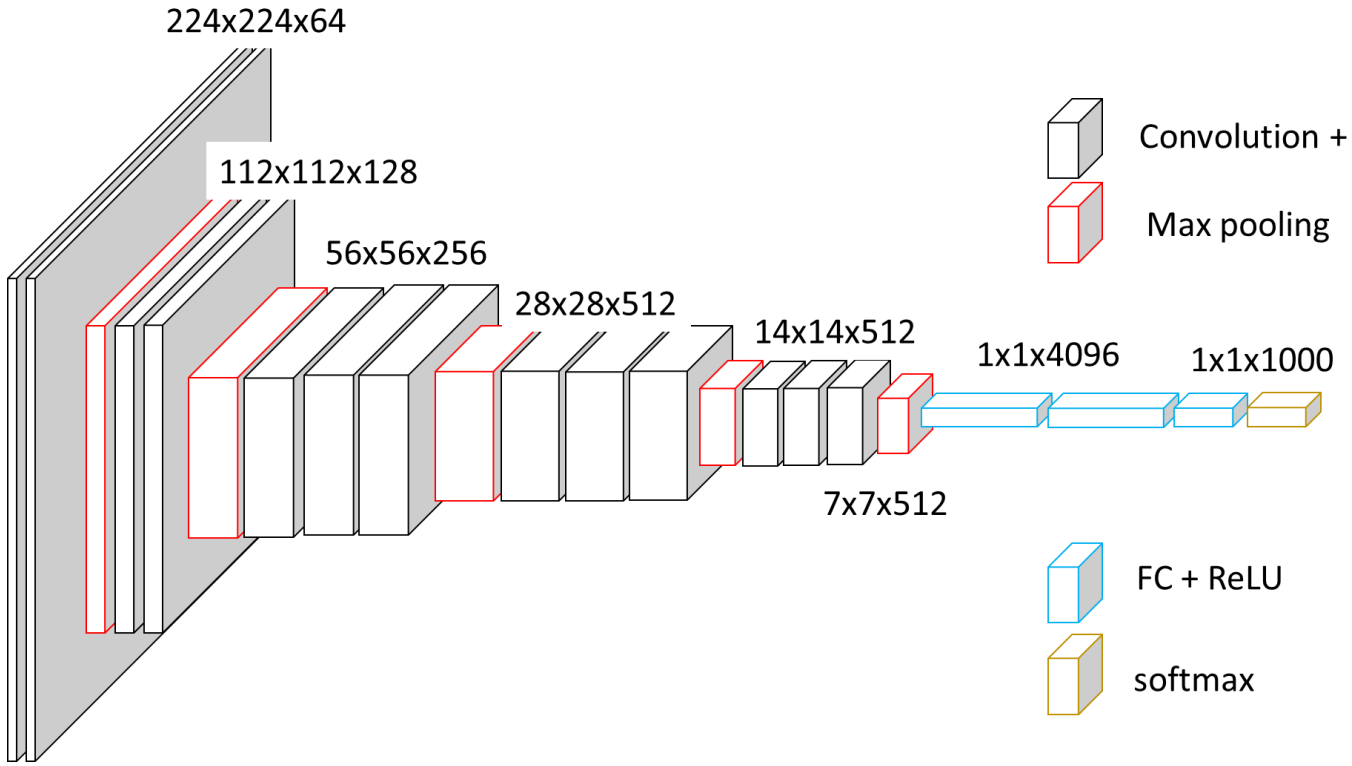


- Examples
  - Google's Inception architectures
  - Residual Networks from Microsoft Research Asia

An Example: VGGNet

Very deep convolutional networks for large-scale image recognition
K Simonyan, A Zisserman - arXiv preprint arXiv:1409.1556, 2014 - arxiv.org
… In this work we evaluated **very deep convolutional networks** (up to 19 weight layers) for largescale image classification. It was demonstrated that the representation depth is beneficial …
☆ Save 99 Cite Cited by 96939 Related articles All 41 versions »





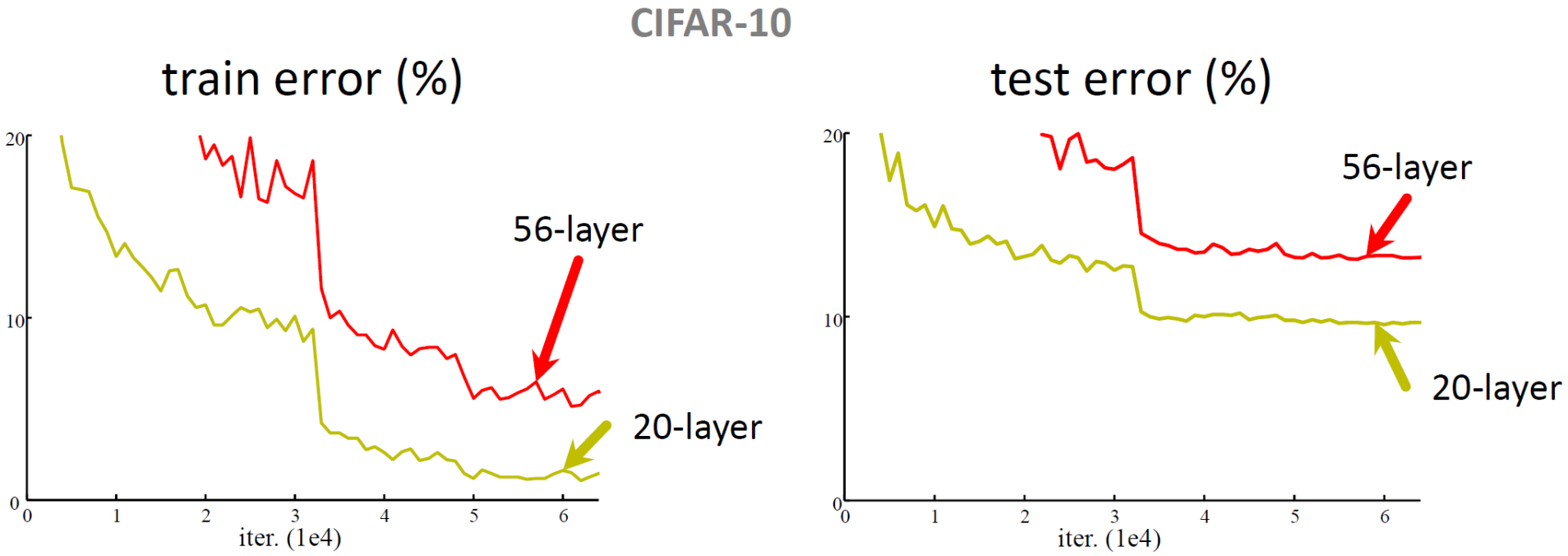
Convolution + ReLU

Max pooling

FC + ReLU

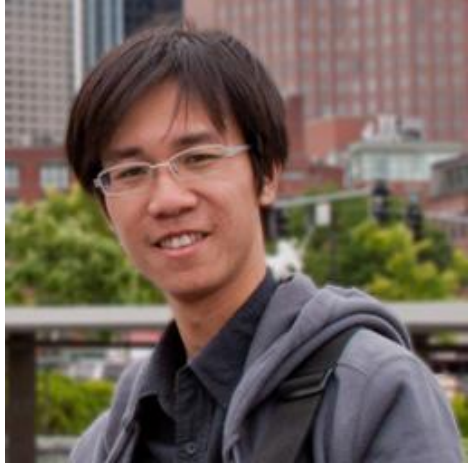
softmax

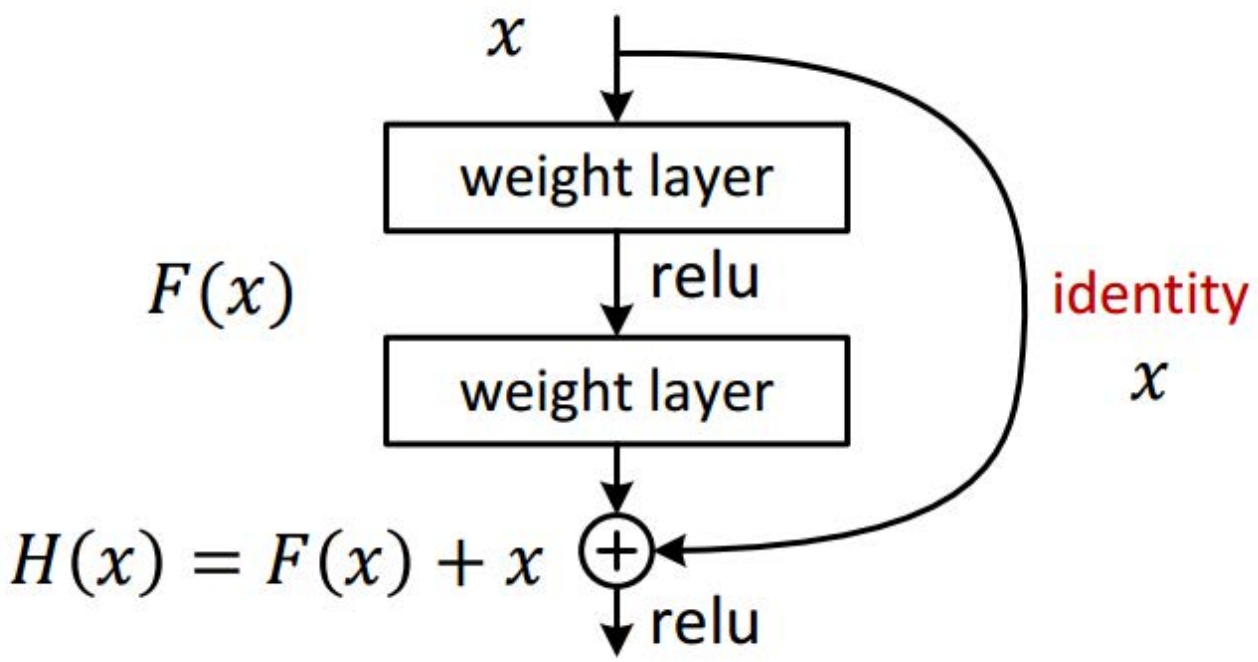
Going Deeper?

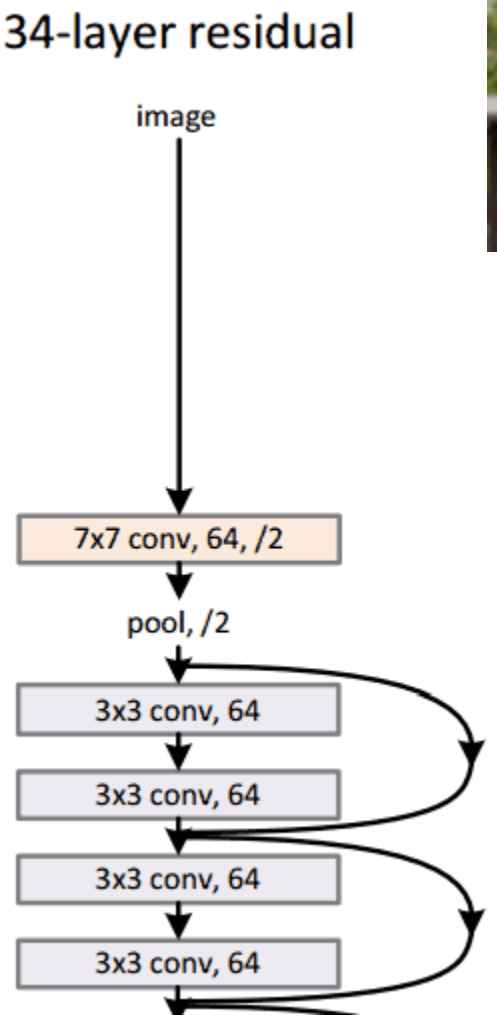


- Simply stacking more layers does not work
  - 56-layer net has higher training error than a 20-layer net

ResNet







Deep residual learning for **image recognition**

K He, X Zhang, S Ren, J Sun - … and **pattern recognition**, 2016 - openaccess.thecvf.com

… **Deeper** neural **networks** are more difficult to train. We present a **residual learning** framework to ease the training of **networks** that are substantially **deeper** than those used previously. …

☆ Save 99 Cite Cited by 155299 Related articles All 73 versions »

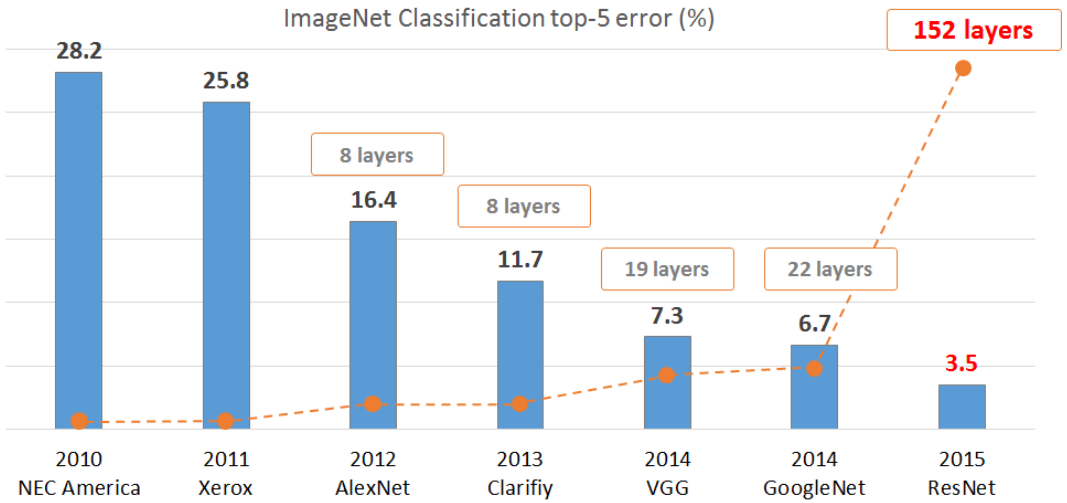
Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

Microsoft



What's Missing?

- Feedforward network and CNN
  - Can only handle inputs and outputs with fixed dimension
  - Memoryless
- However, many applications involve sequences with variable lengths
  - Speech recognition
  - Sequence labeling
  - Language modeling
  - Question answering/generation
  - Chatbot
  - Machine translation
  - Image/video captioning

Recurrent Neural Networks (RNN)

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

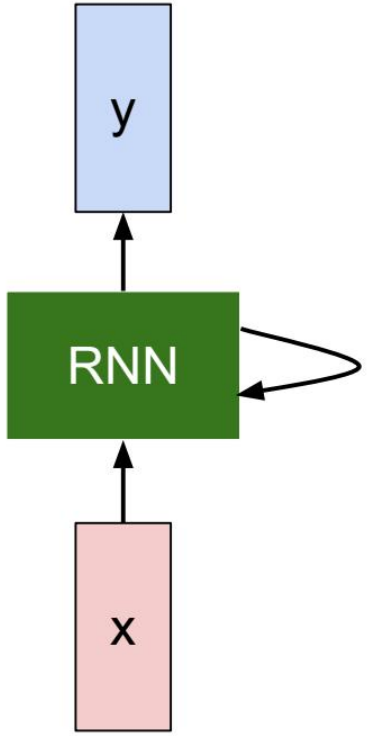
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

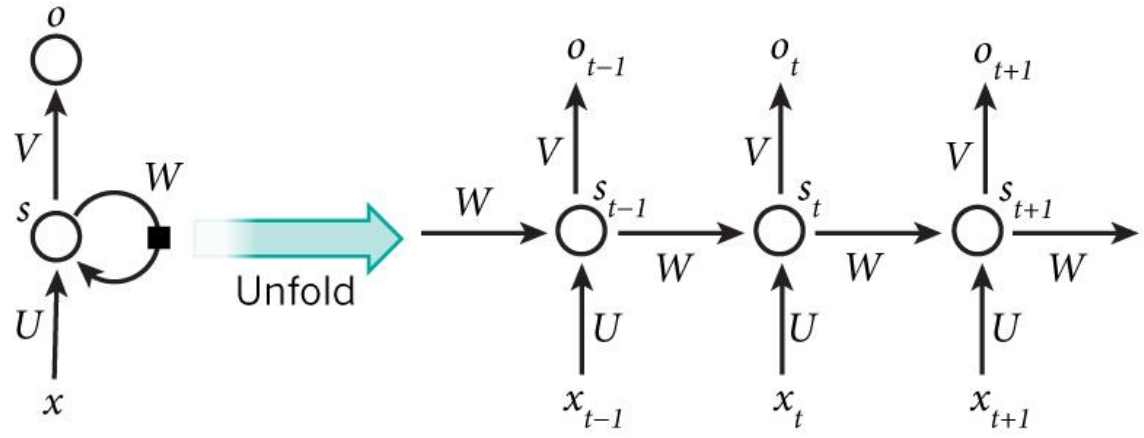
input vector at some time step

some function with parameters W

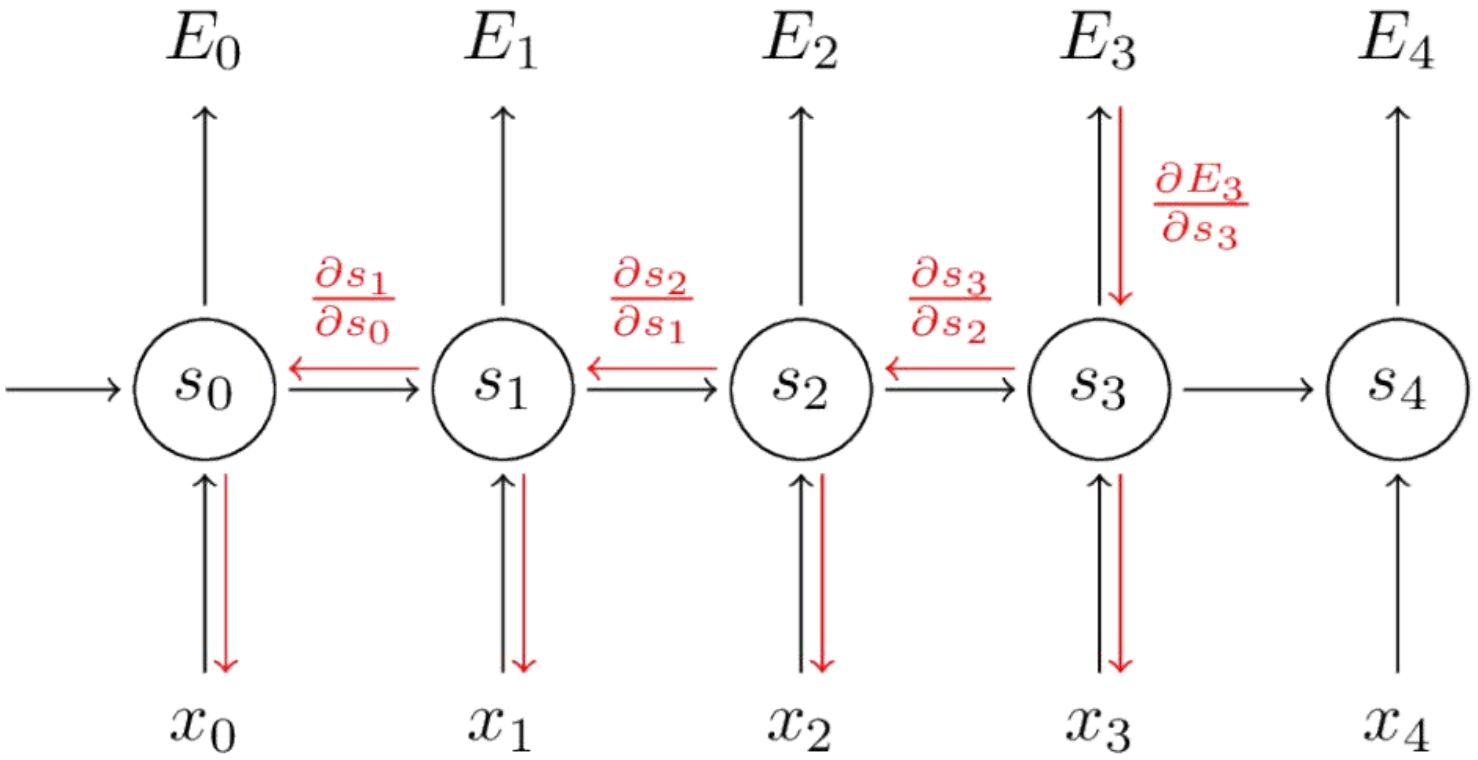


Recurrent Neural Networks (RNN)

- Contains feed-back connection, so the activations can flow round in a loop
- Handle inputs with variable lengths by weight sharing
- Enable the networks to do temporal processing and learn sequences.



Backpropagation Through Time (BPTT)



The Challenge of Long-Term Dependencies

Recurrent function

$$h^{(t)} = f\left(Wh^{(t-1)}\right)$$

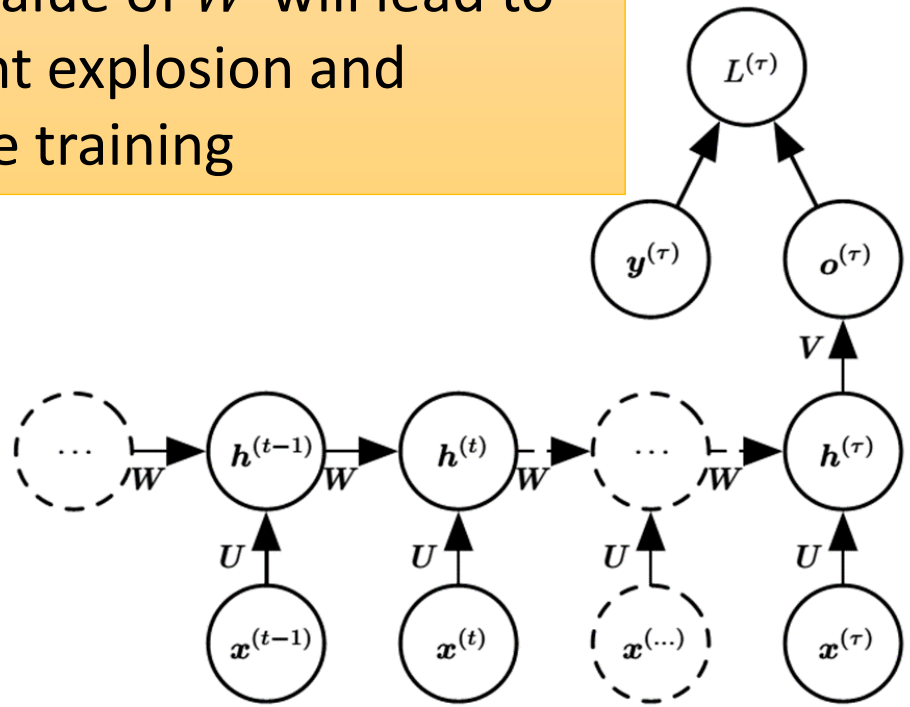
Consider a linear recurrent function

$$h^{(t)} = Wh^{(t-1)} = W^t h^{(0)}$$

Eigen value decomposition

$$W = Q\Lambda Q^T$$
$$h^{(t)} = Q\Lambda^t Q^T h^{(0)}$$

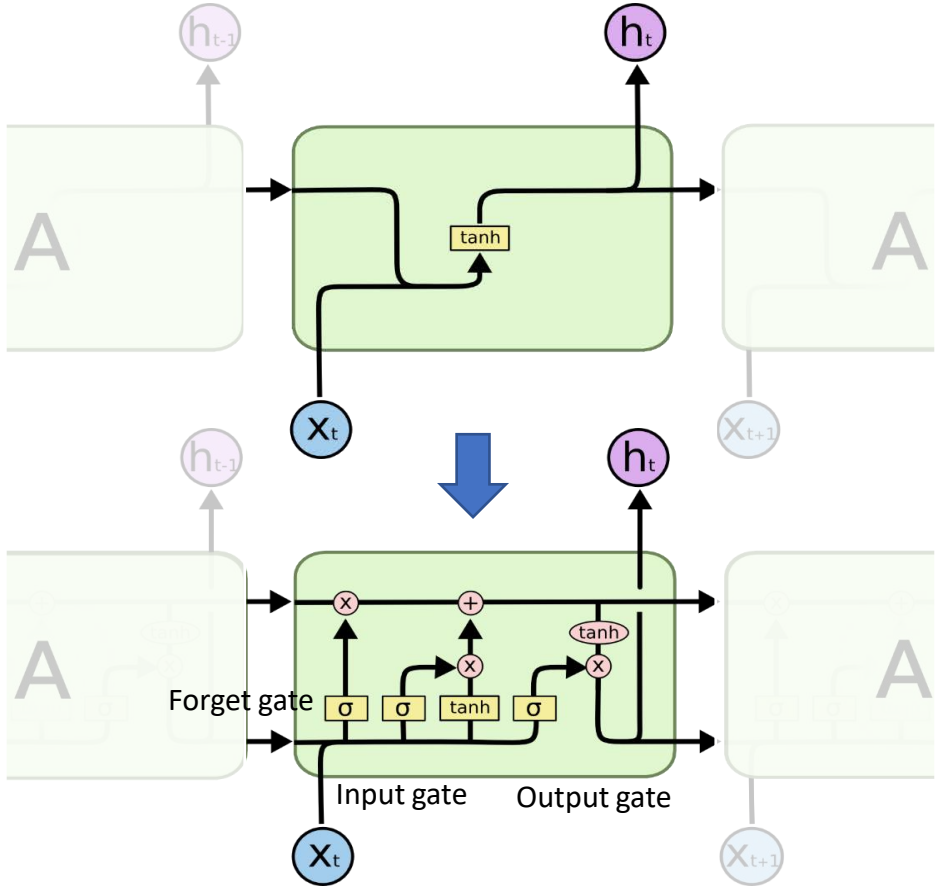
Gradient explosion: large eigen value of W will lead to gradient explosion and instable training



Information/gradient vanishing: any component of $h^{(0)}$ that is not aligned with the largest eigen vector of W will eventually be discarded

Long Short Term Memory (LSTM)

- Add direct information flows
- Separate hidden state and memory (cell)
- Three parameterized gates:
  - Forget gate
  - Input gate
  - Output gate



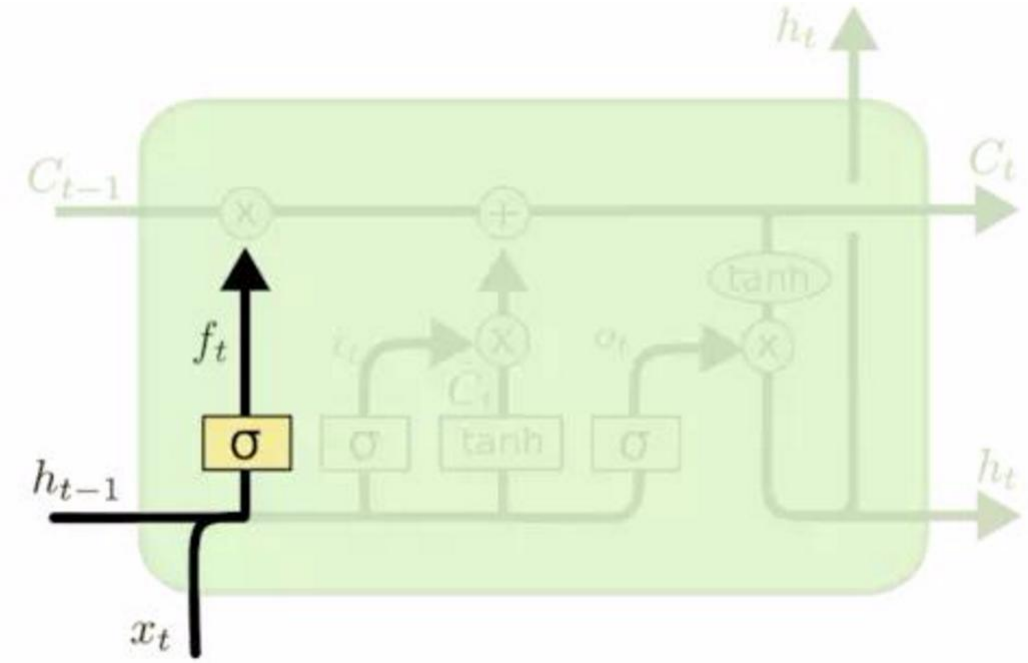
Long short-term memory

S Hochreiter, J Schmidhuber - Neural computation, 1997 - ieeexplore.ieee.org

… (**short-term memory**, as opposed to **long-term memory** … learning what to put in **shortterm memory**, however, take too … and corresponding teacher signals are **long**. Although theoretically …

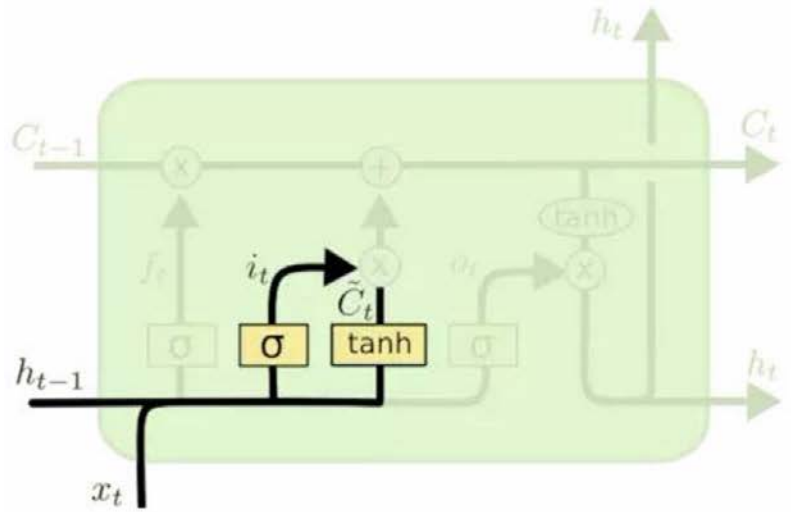
☆ Save 99 Cite Cited by 79853 Related articles All 44 versions

LSTM – Forget Gate



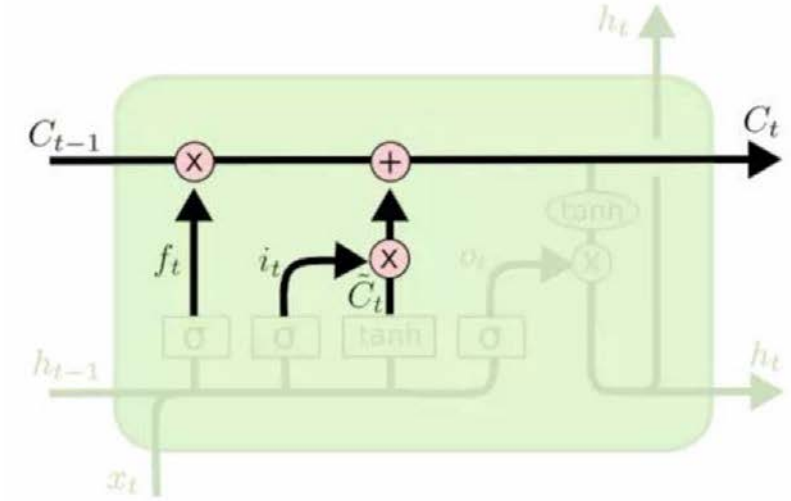
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM – Input Gate and Memory Update



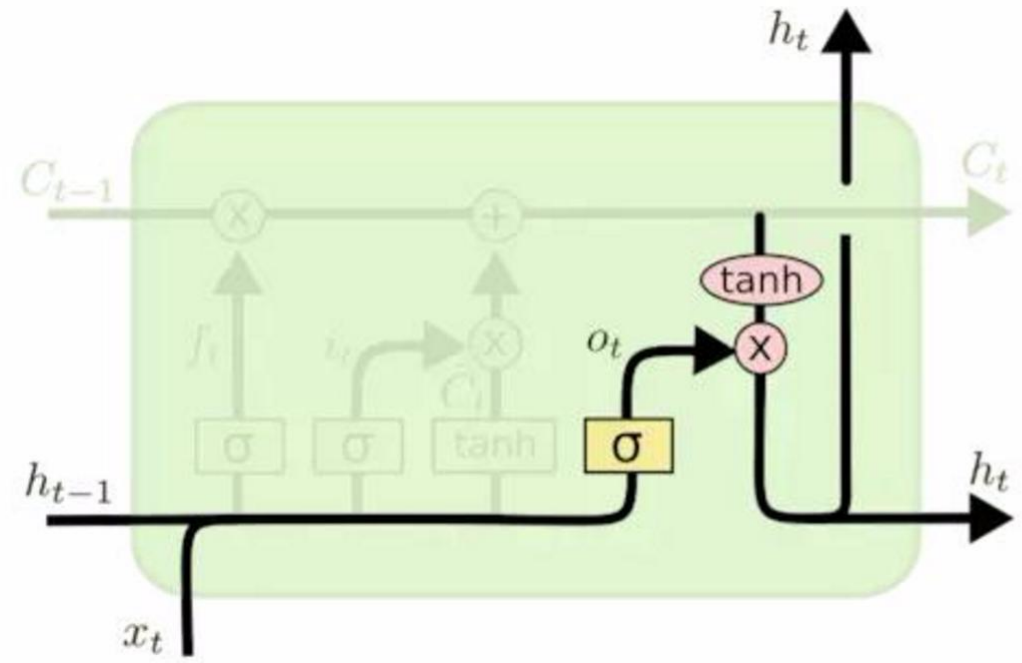
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM – Output Gate

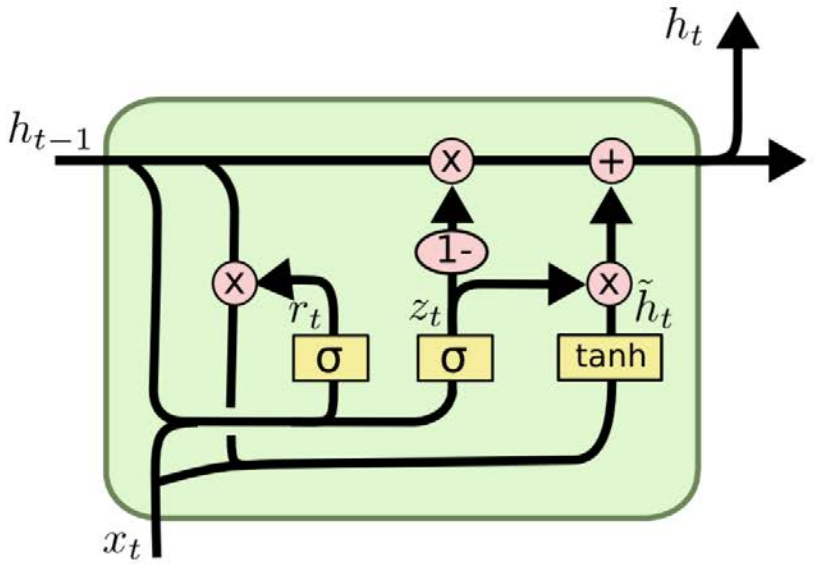


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Gated Recurrent Units

- Merge the input and forget gates: z_t, update gate; r_t, reset gate
- Merge memory (cell) and hidden state
- Reduce the number of parameters



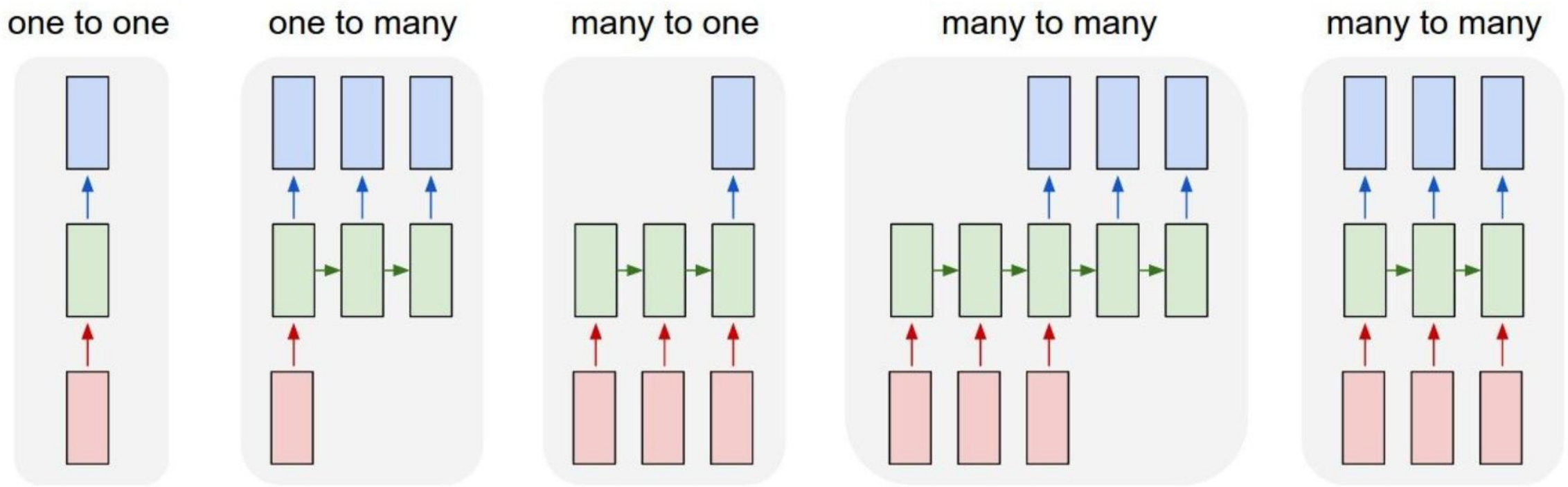
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

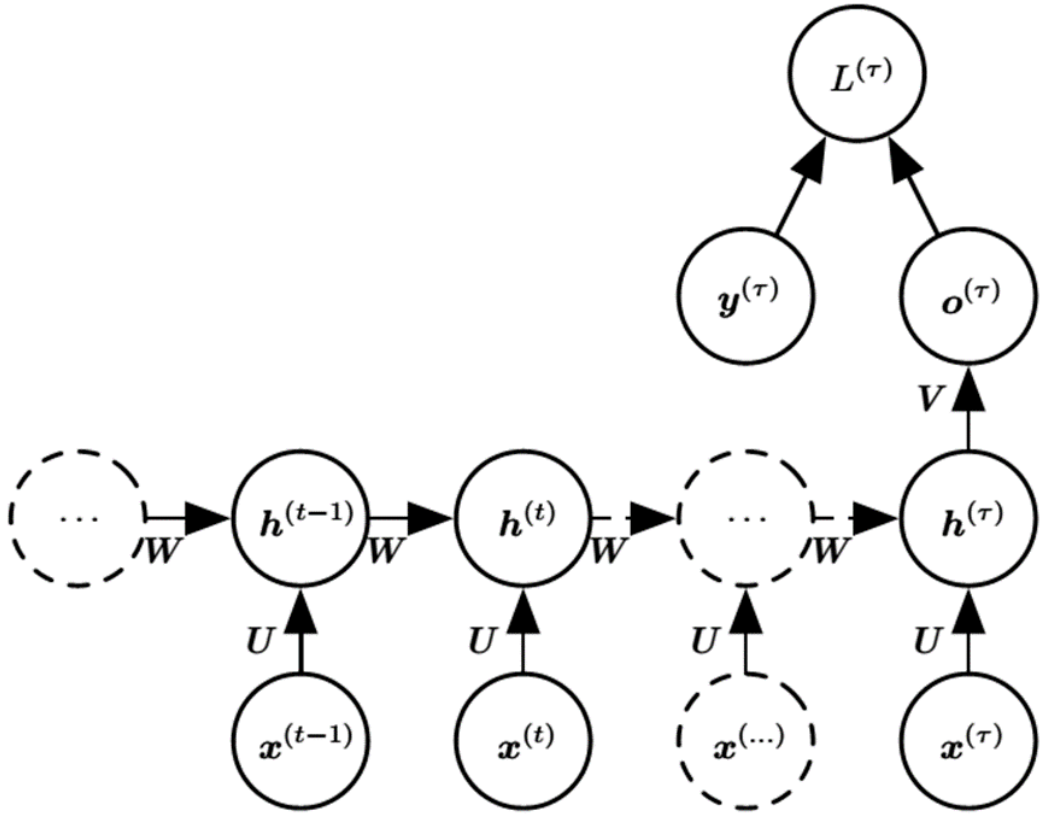
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

RNN: Flexibilities and Possibilities



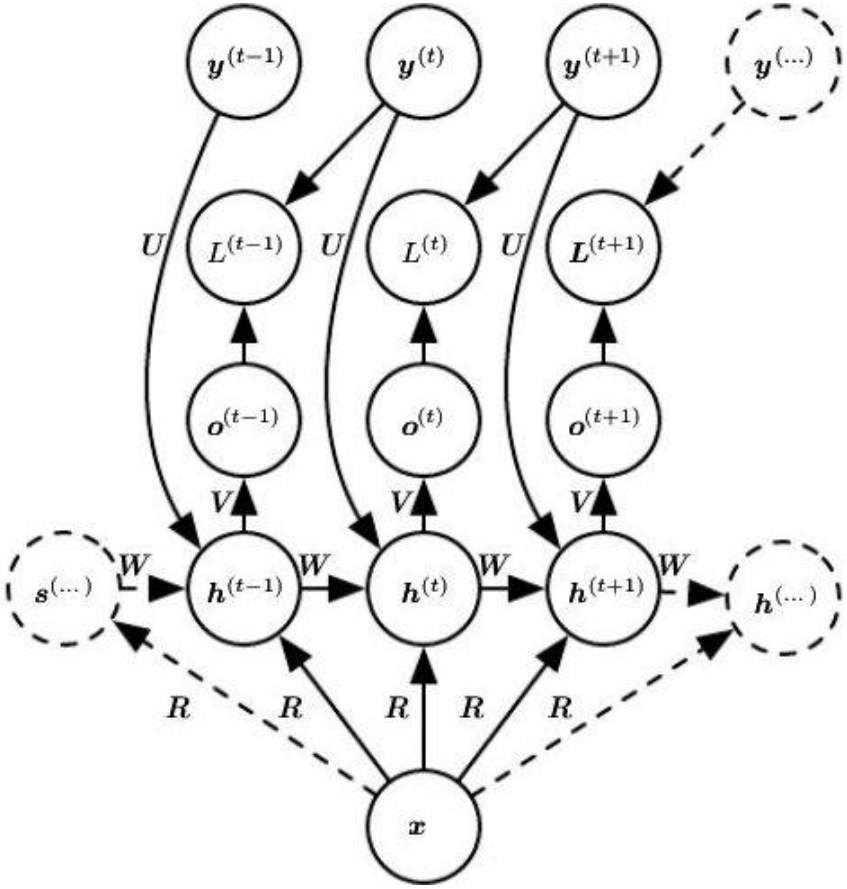
Many to One

- Encoder RNN
- Example: sentiment analysis
  - Input: text sequence
  - Output: 1/-1, negative/positive



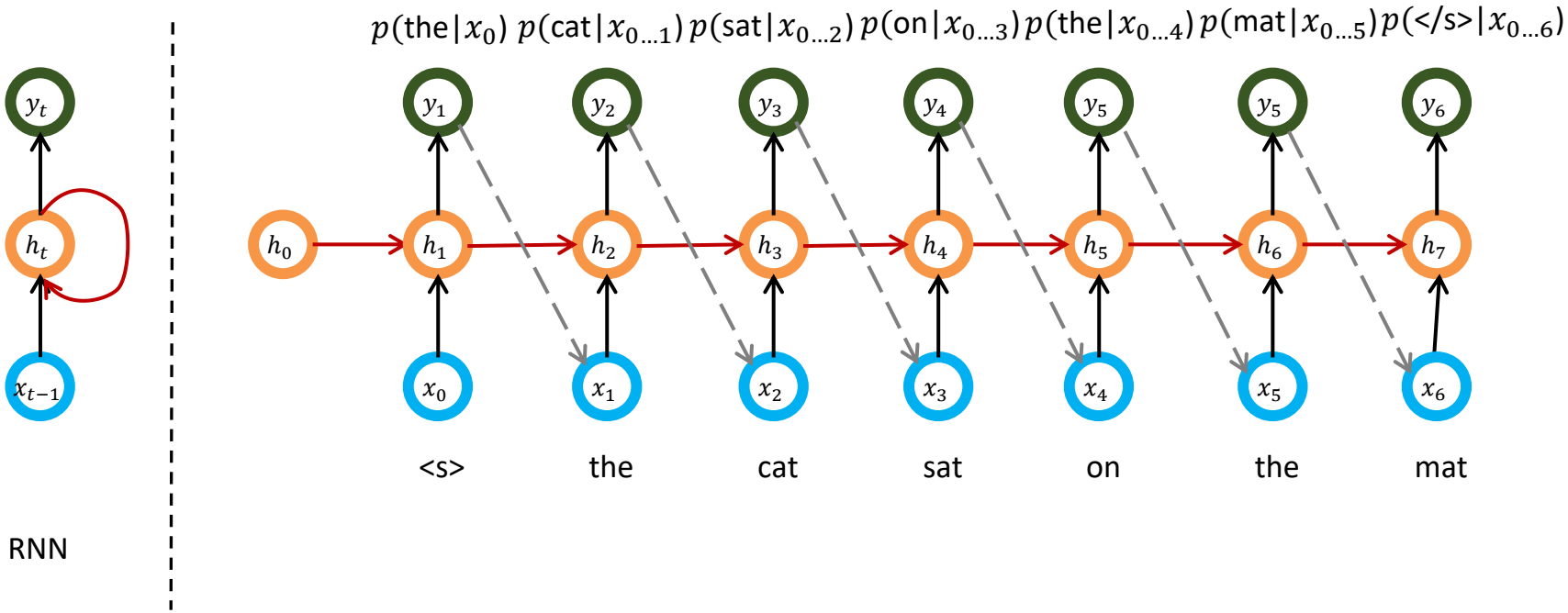
One to Many

- Decoder RNN
- Example: image captioning
  - Input: image (vector) x
  - Output: text sequence
- How to use x
  - As an extra input at each time step
  - As the initial state of $h^{(0)}$



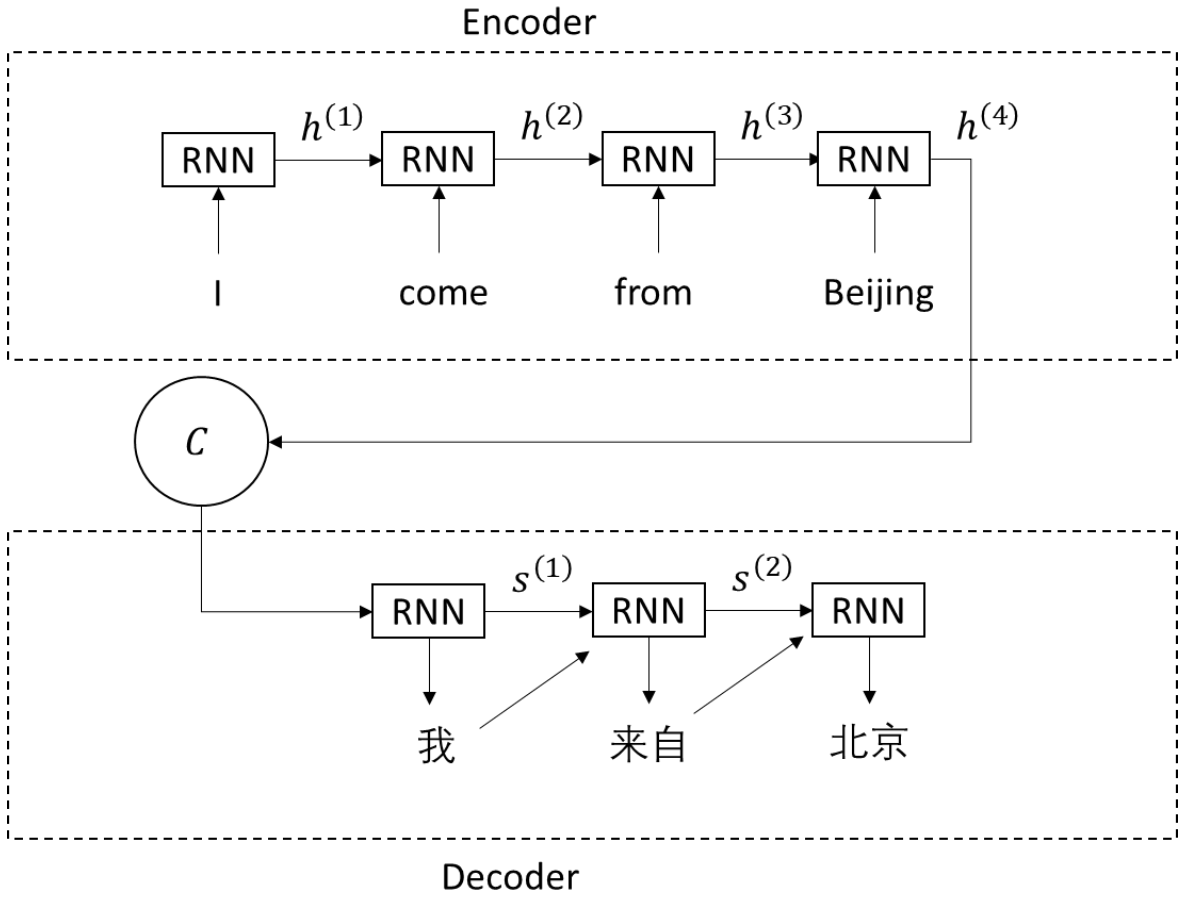
Many to Many

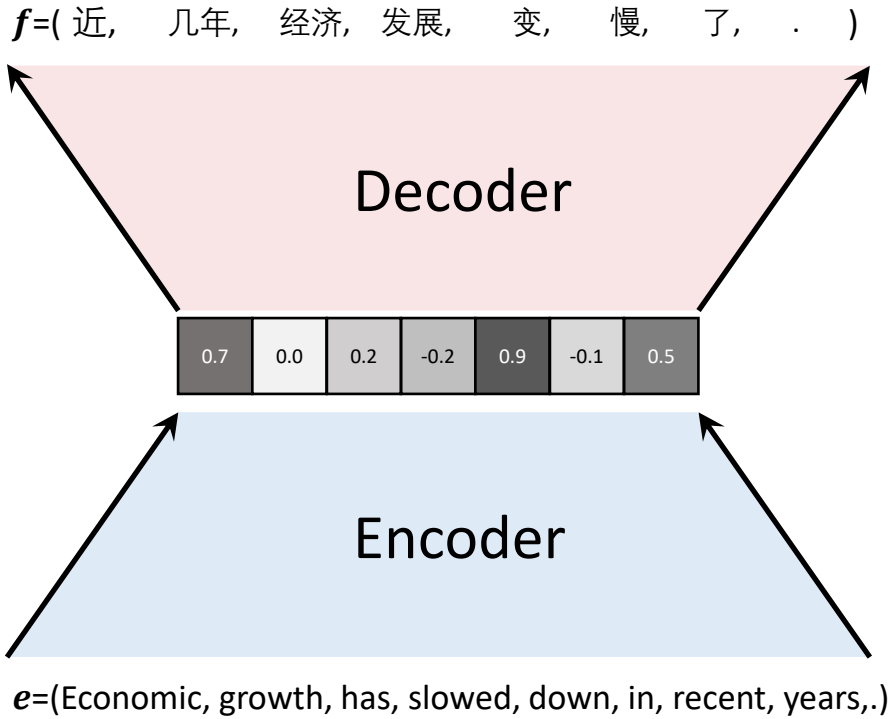
- Language Modeling



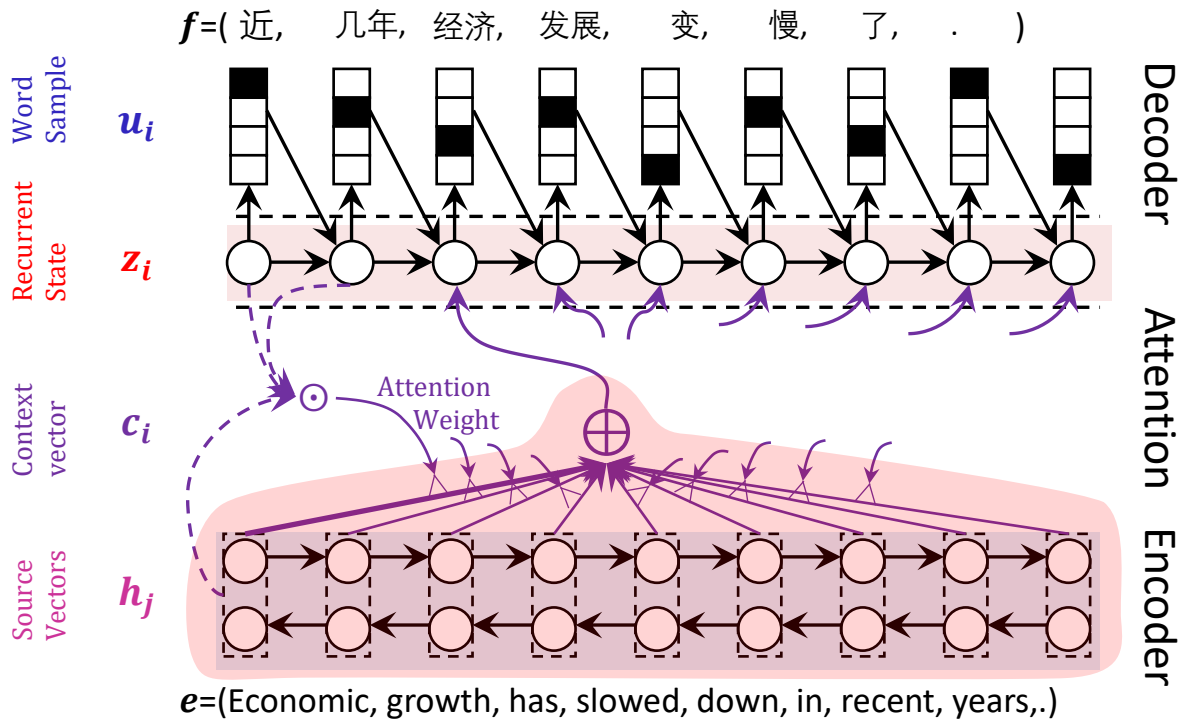
Unfolded RNN for Language Modeling

Many to Many: Encoder-Decoder for Sequence Generation





Attention based Encoder-Decoder



Neural **machine translation** by **jointly** learning to align and **translate**

D Bahdanau, K Cho, Y Bengio - arXiv preprint arXiv:1409.0473, 2014 - arxiv.org

… an extension to the encoder–decoder model which learns to align and **translate jointly**.

Each time the proposed model generates a word in a **translation**, it (soft-)searches for a set of …

☆ Save 99 Cite Cited by 27500 Related articles All 28 versions »

Transformer Networks

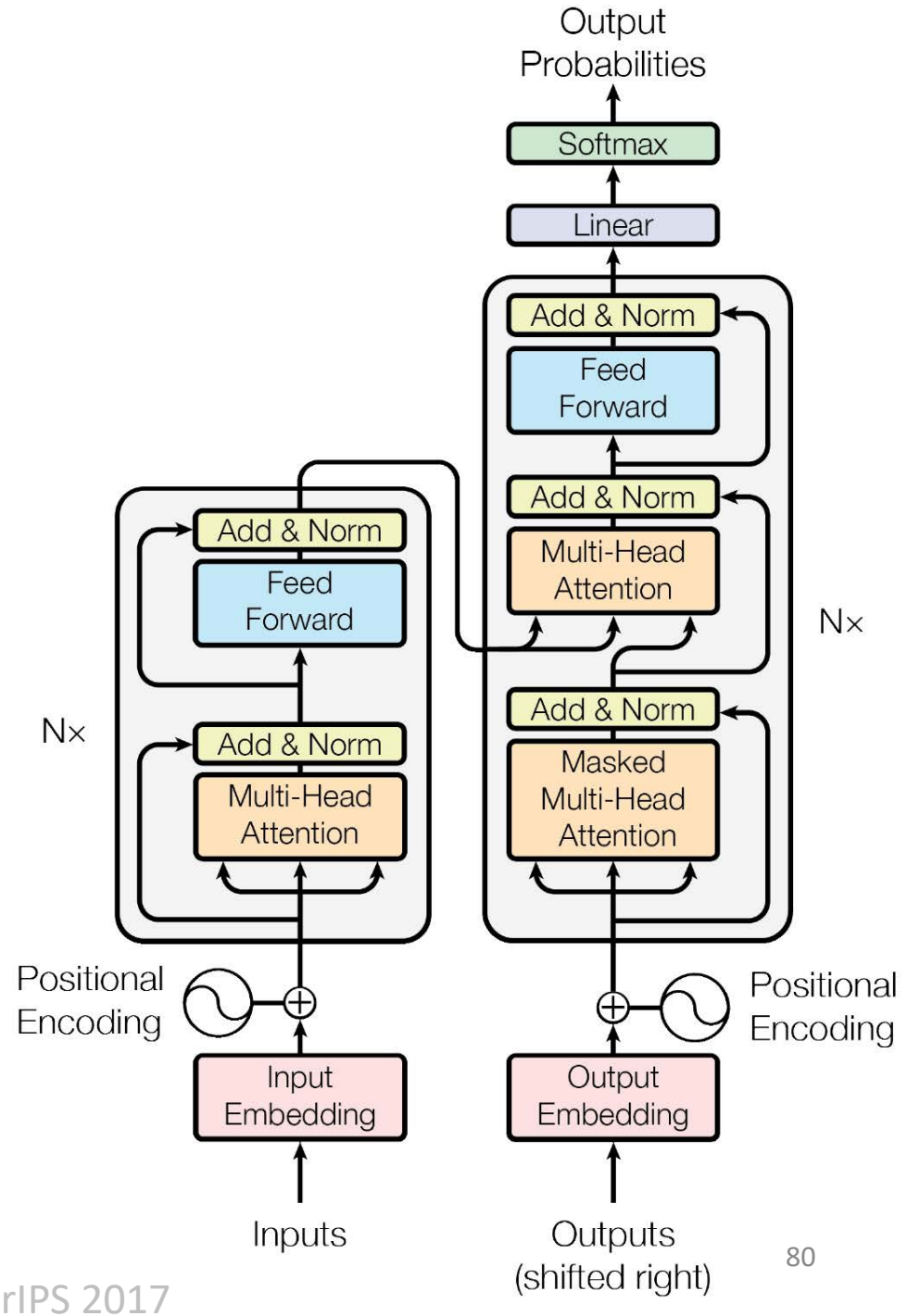
- Attentions in the model
  - Encoder-decoder attention layers
  - Self-attention layers in the encoder
  - Masked self-attention layers in the decoder

Attention is all you need

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

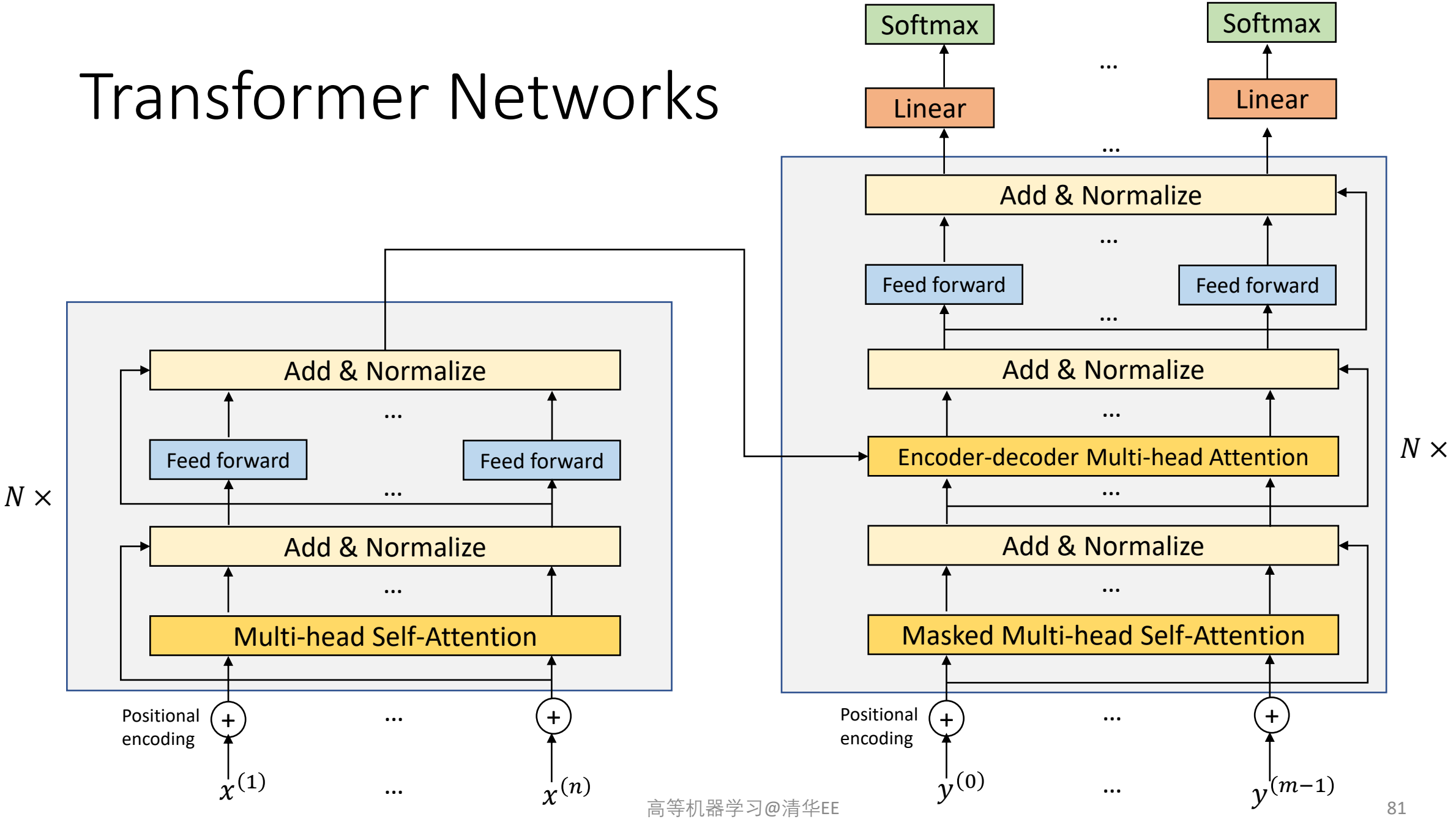
… to attend to **all** positions in the decoder up to and including that position. **We need** to prevent … **We** implement this inside of scaled dot-product **attention** by masking out (setting to −∞) …

☆ Save 99 Cite Cited by 67167 Related articles All 46 versions »

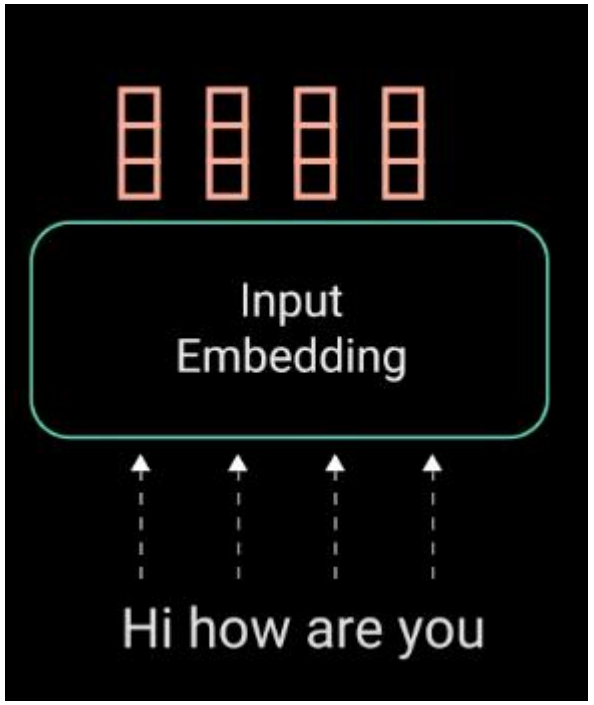


Attention is all you need, NeurIPS 2017

Transformer Networks



Input Embedding and Positional Encoding

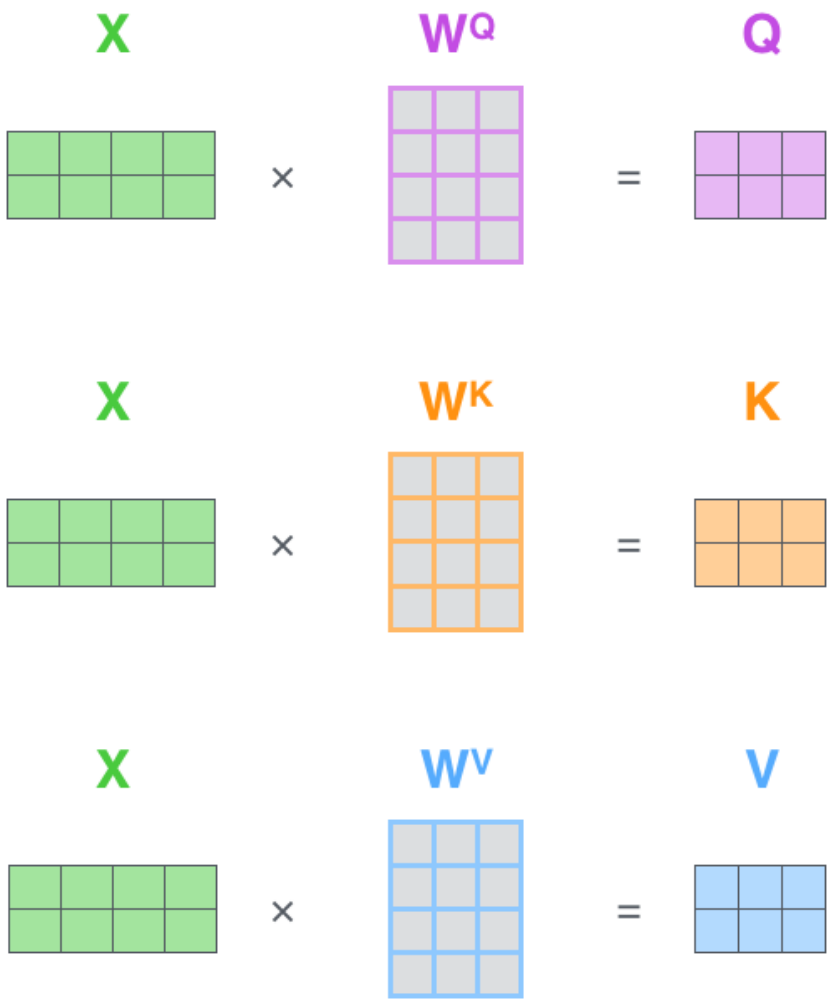


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

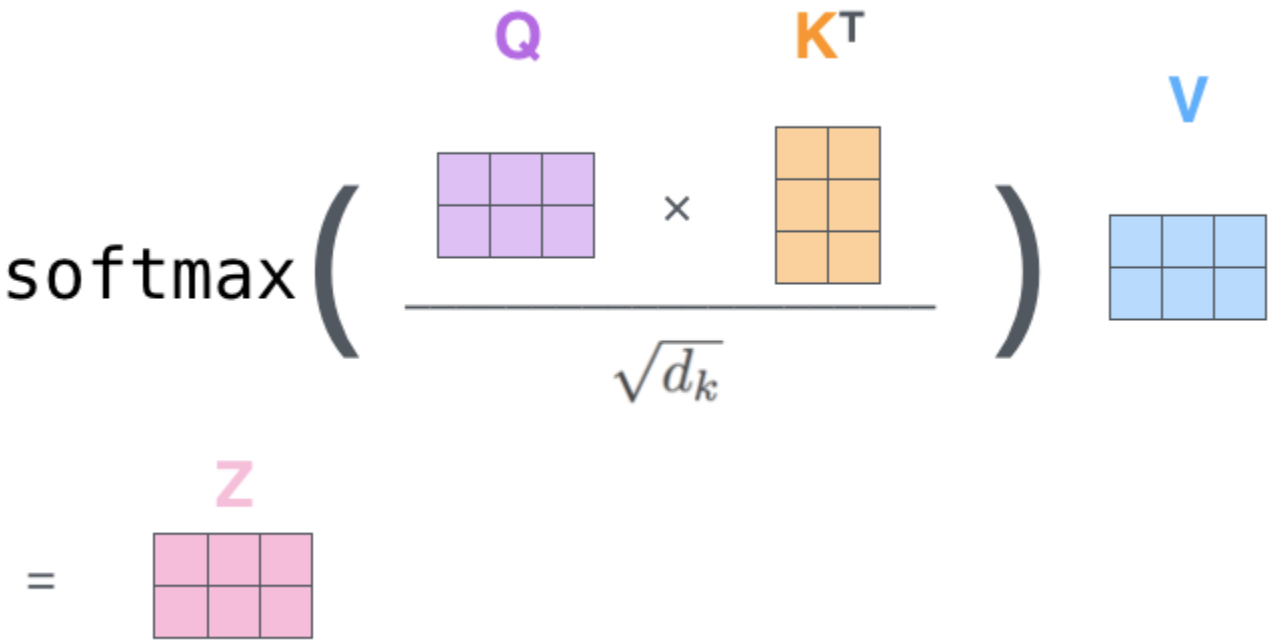
Query, Key, Value

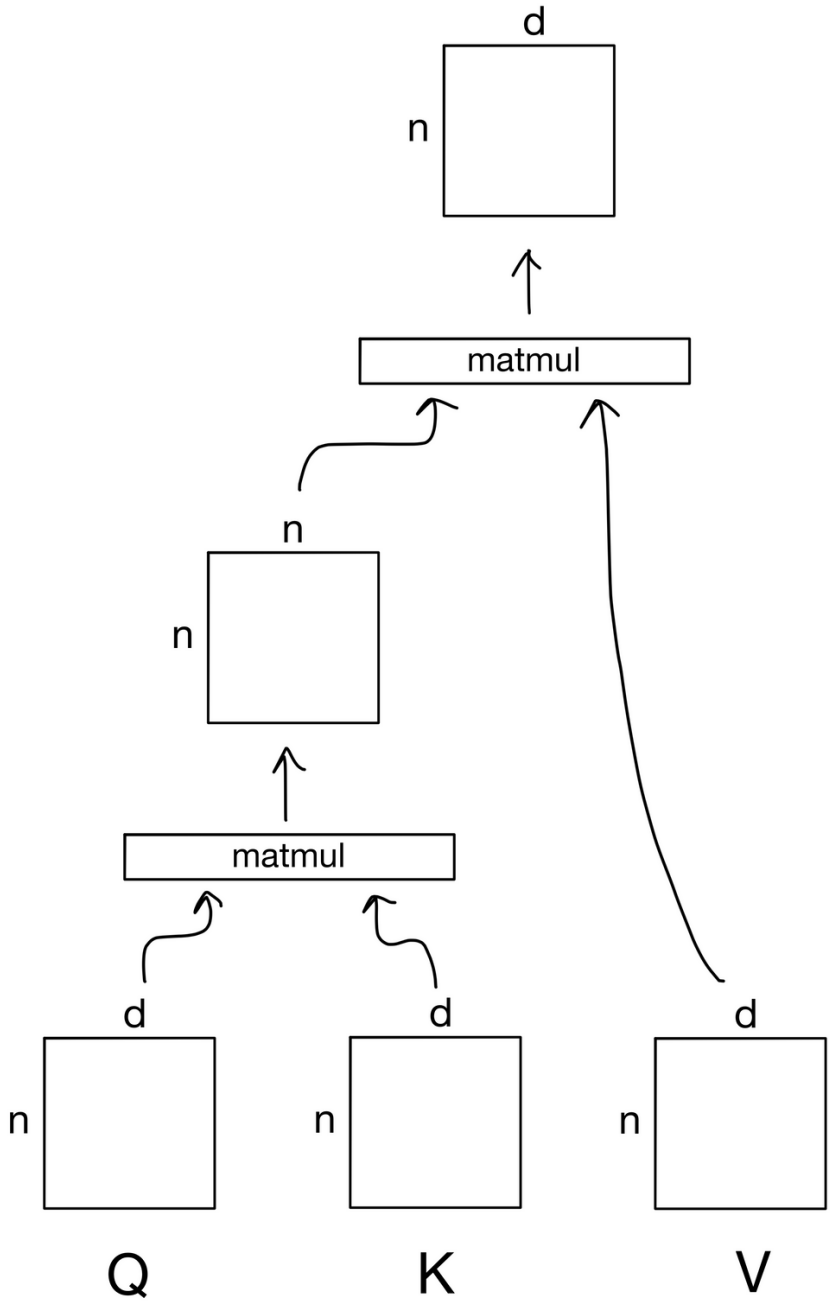
- The key/value/query concept is analogous to retrieval systems.
- For example, when you search for videos on Youtube, the search engine will map your **query** (text in the search bar) against a set of **keys** (video title, description, etc.) associated with candidate videos in their database, then present you the best matched videos (**values**).



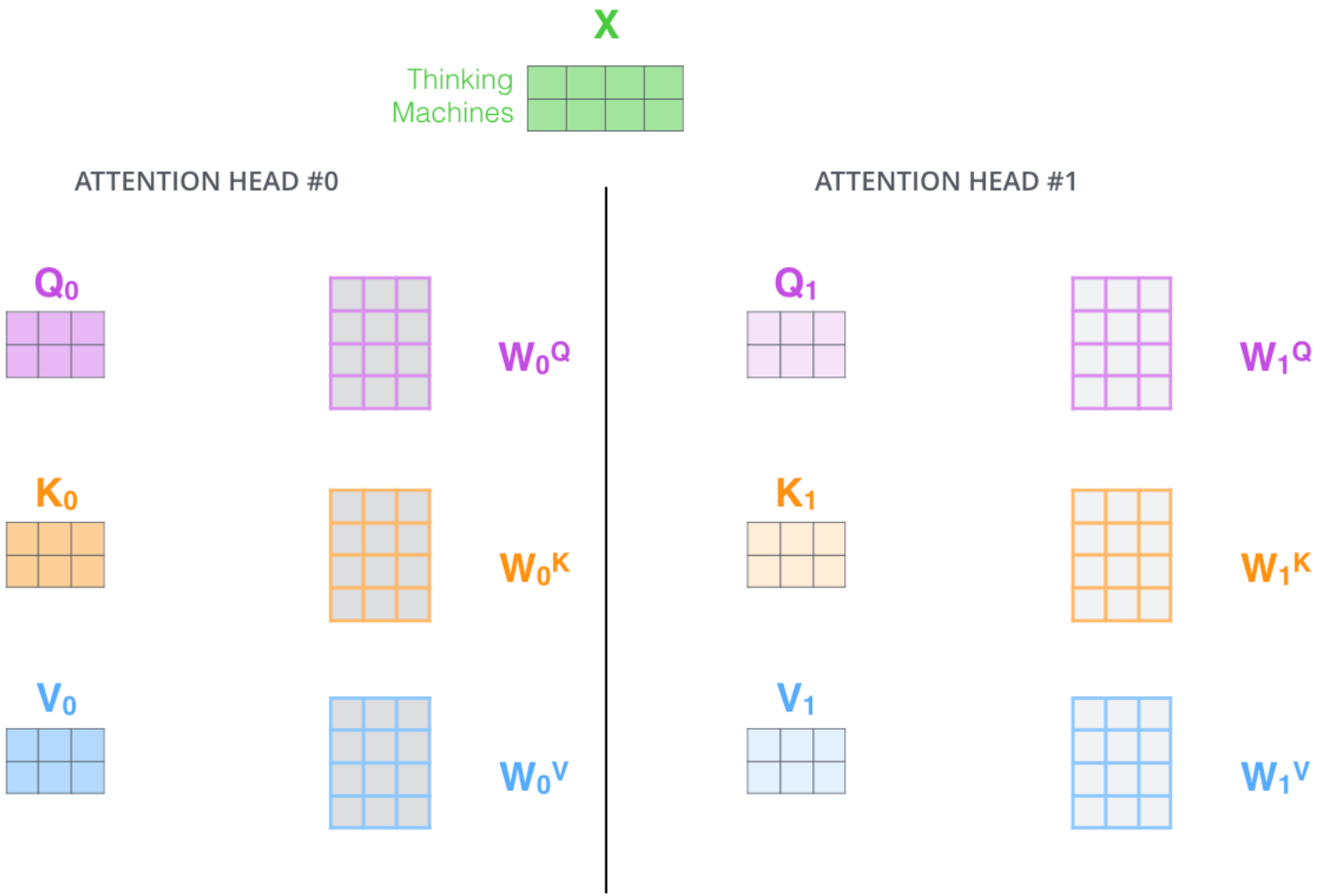
http://jalammar.github.io/illustrated-transformer/

Self-attention



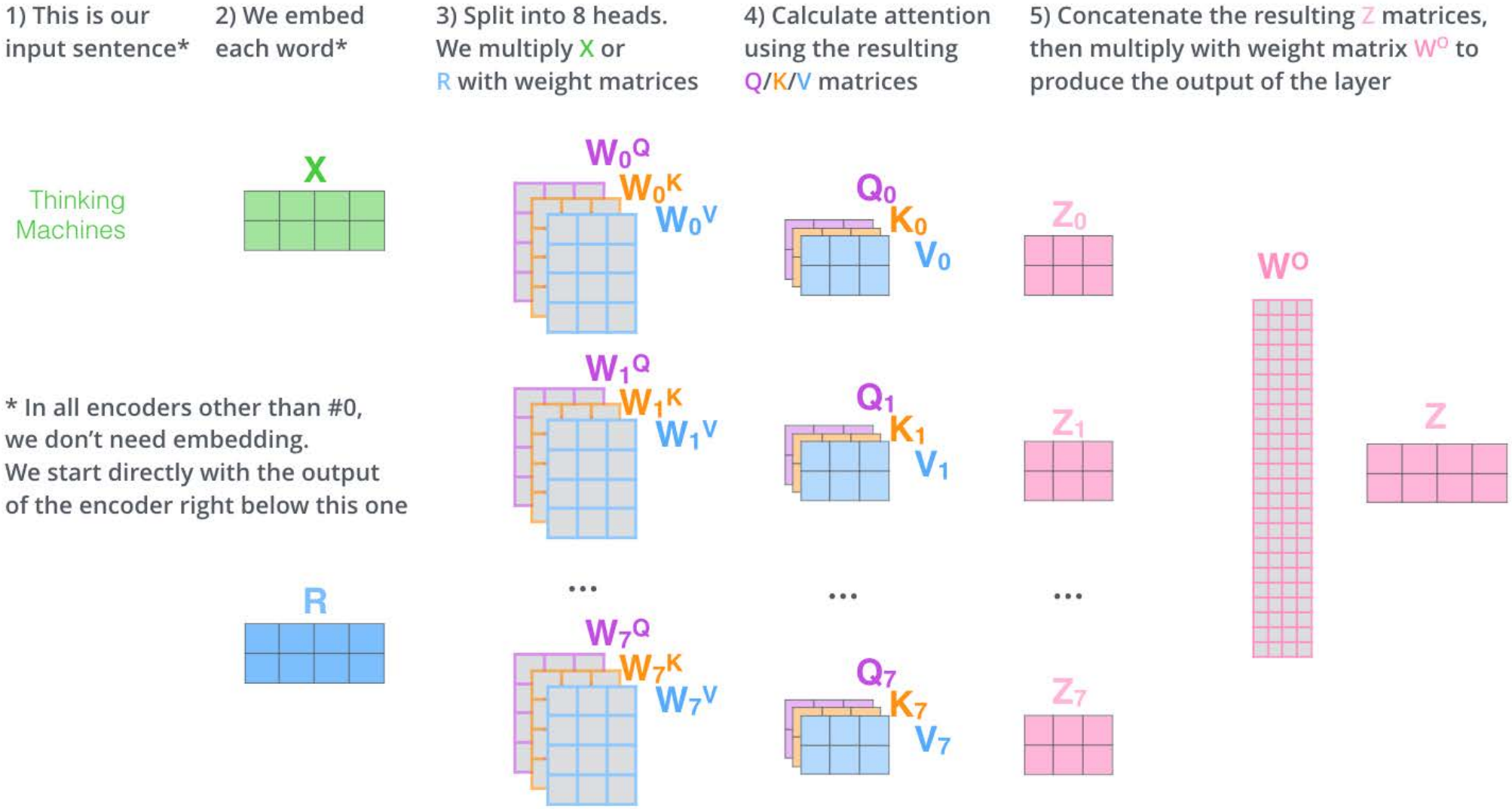


Multi-head Attention

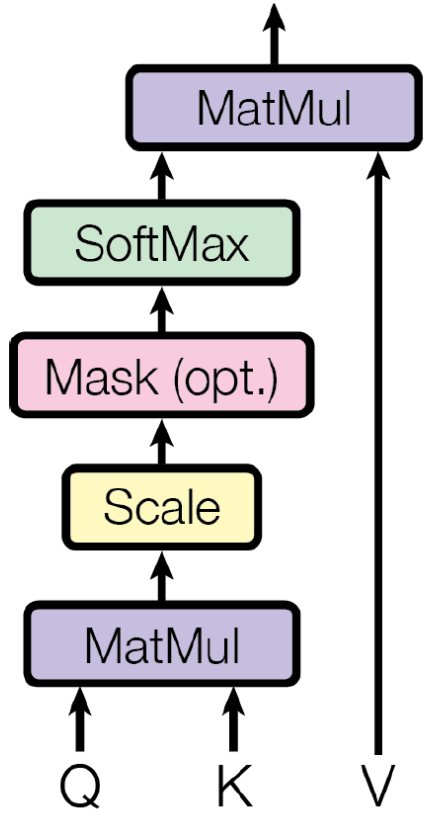


http://jalammar.github.io/illustrated-transformer/

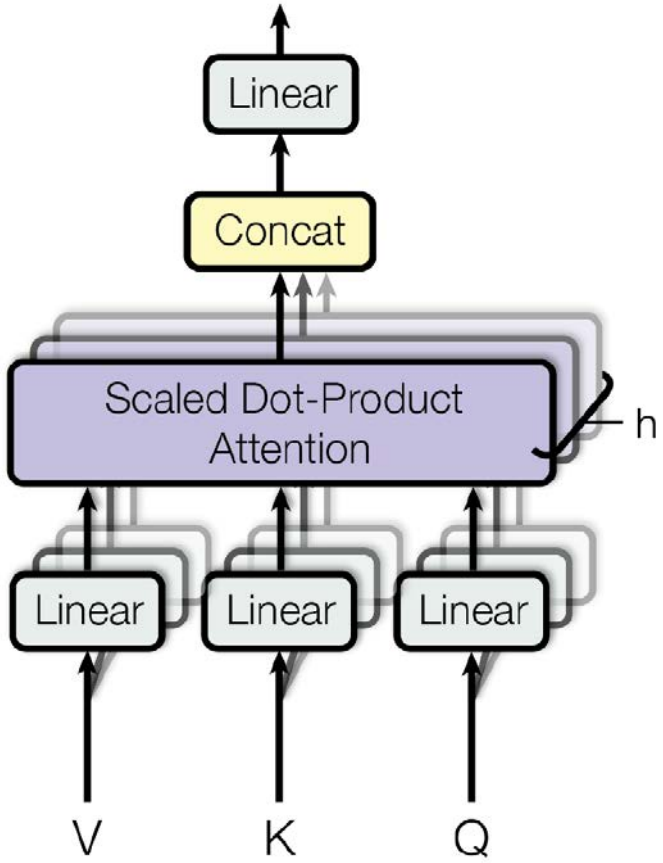
Multi-head Attention



Scaled Dot-Product Attention



Multi-Head Attention

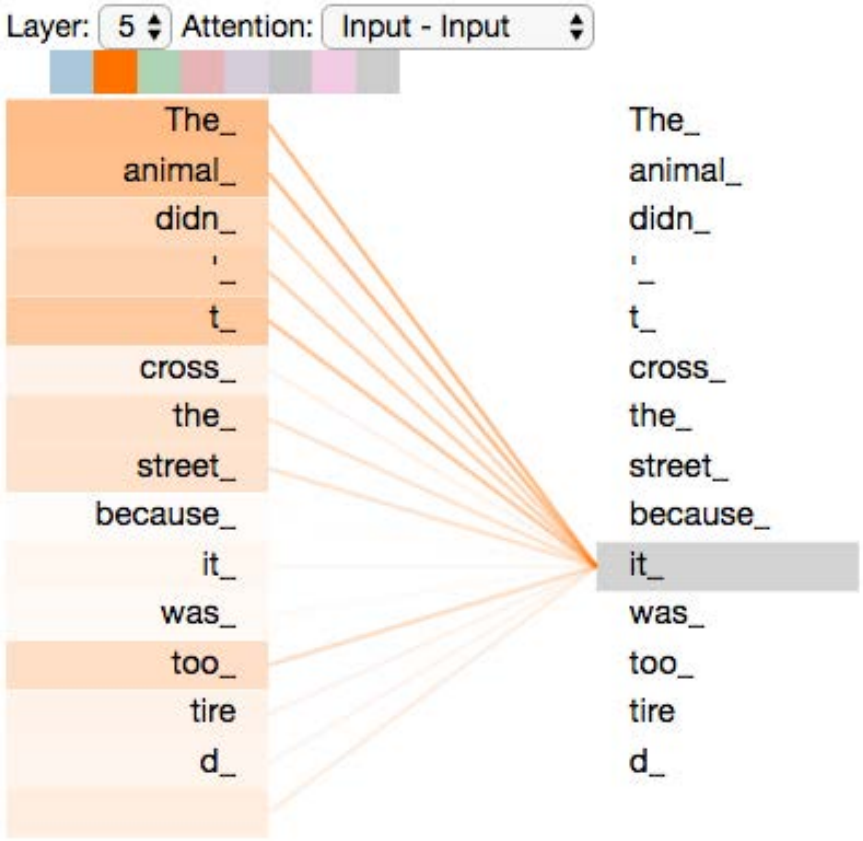


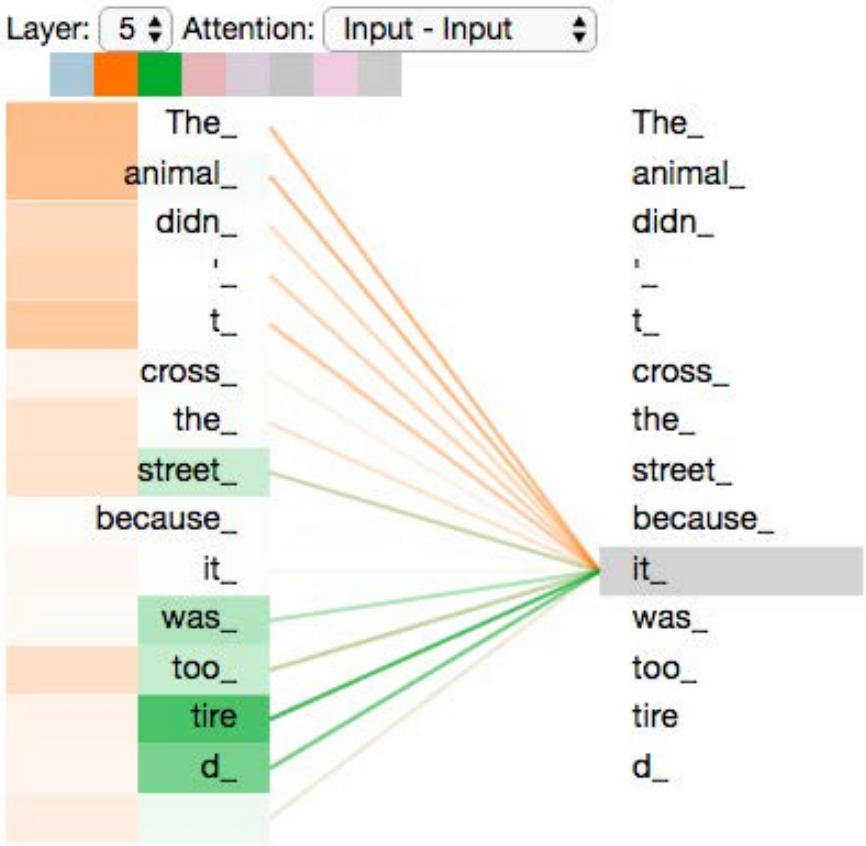
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

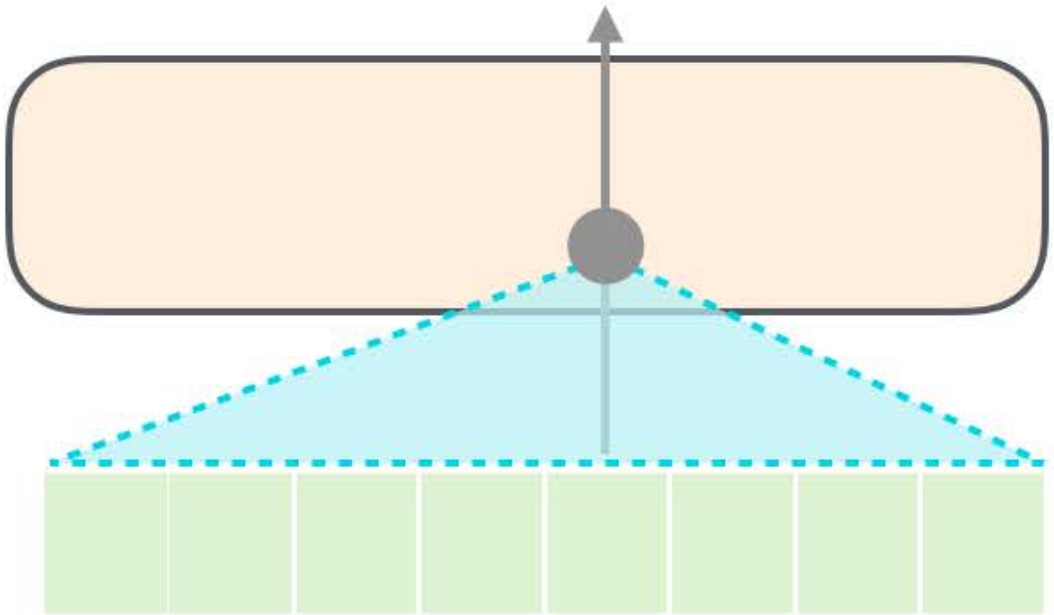
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Visualization of Multi-Head Attention

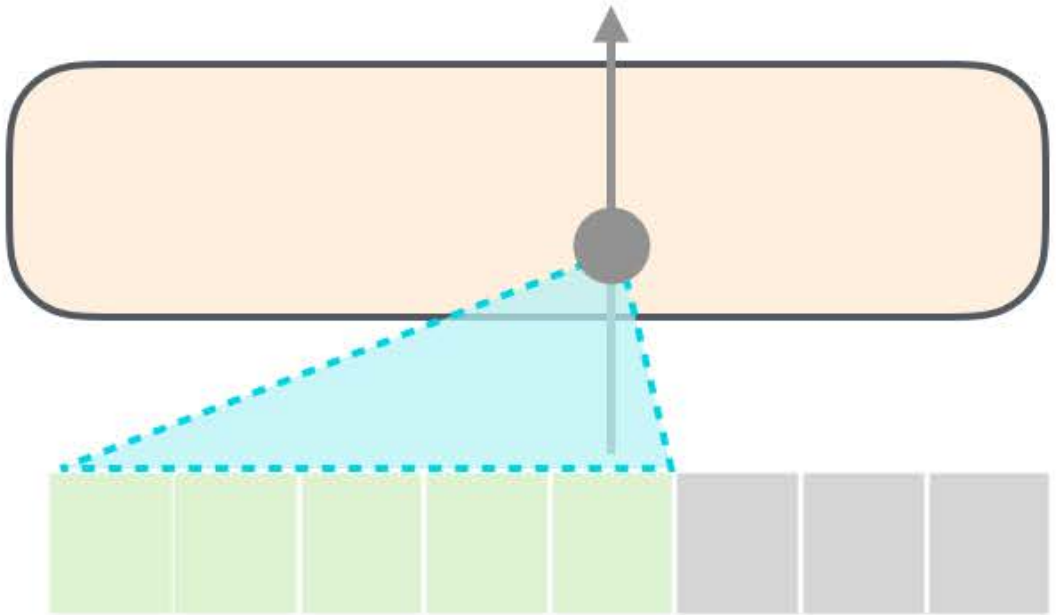




Self-Attention



Masked Self-Attention



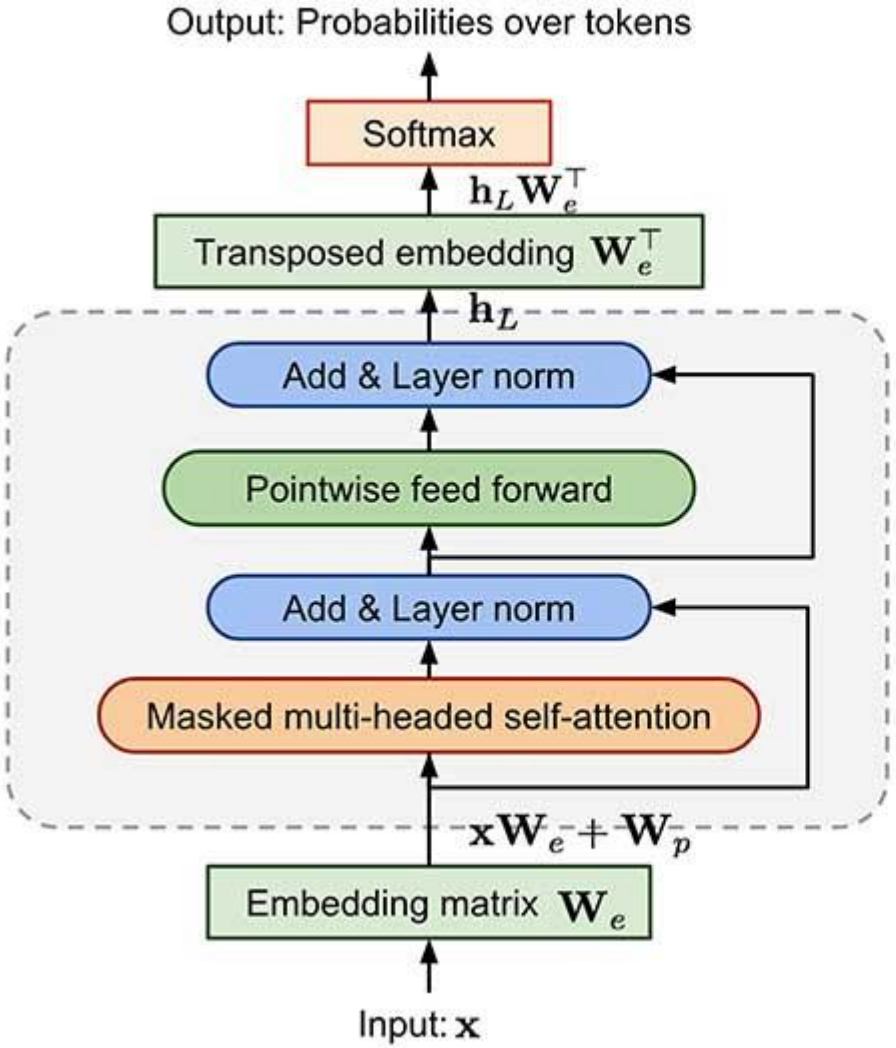
Computational complexity

- n is the sequence length,
- d is the representation dimension,
- k is the kernel size of convolutions
- r the size of the neighborhood in restricted self-attention

Transformer is of much less complexity

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

GPT: Generative Pre-training Transformer



- Pretrain a deep Transformer decoder

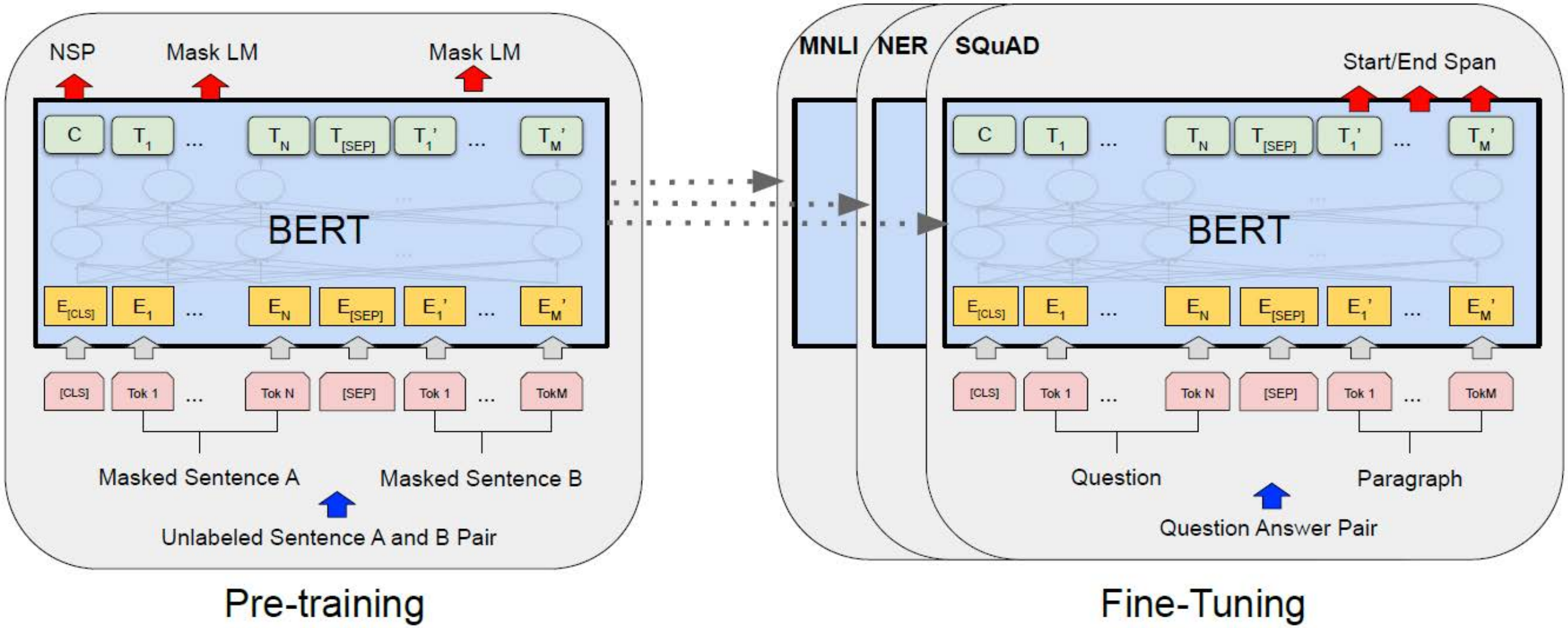
$$\mathcal{L}_{\text{LM}} = -\sum_i \log p(x_i \mid x_{i-k}, \ldots, x_{i-1})$$

Transformer Block
Repeat x L=12

$$\mathbf{h}_\ell = \text{transformer\_block}(\mathbf{h}_{\ell-1})$$
$$\ell = 1, \ldots, L$$

[PDF] Improving **language** understanding by **generative** pre-training
A Radford, K Narasimhan, T Salimans, I Sutskever - 2018 - cs.ubc.ca
… Zero-shot Behaviors We'd like to better understand why **language model** … **generative model** learns to perform many of the tasks we evaluate on in order to improve its **language modeling** …
☆ Save 99 Cite Cited by 4971 Related articles All 9 versions »

BERT pretraining and fine-tuning



Initialization and Regularization

Parameter Initialization

- Convergence or not are sensitive to initial points
  - with some initial points being so unstable that the algorithm encounters numerical difficulties and fails altogether.
- Convergence speed and final training error depend on the initial point
- Initial points can affect the generalization as well.
  - Points of comparable cost can have wildly varying generalization error
- (Too) small initial weights lead to information loss in forward/ backpropagation through the linear component of each layer
- Initial weights that are too large may, however, result in exploding values during forward propagation or back-propagation

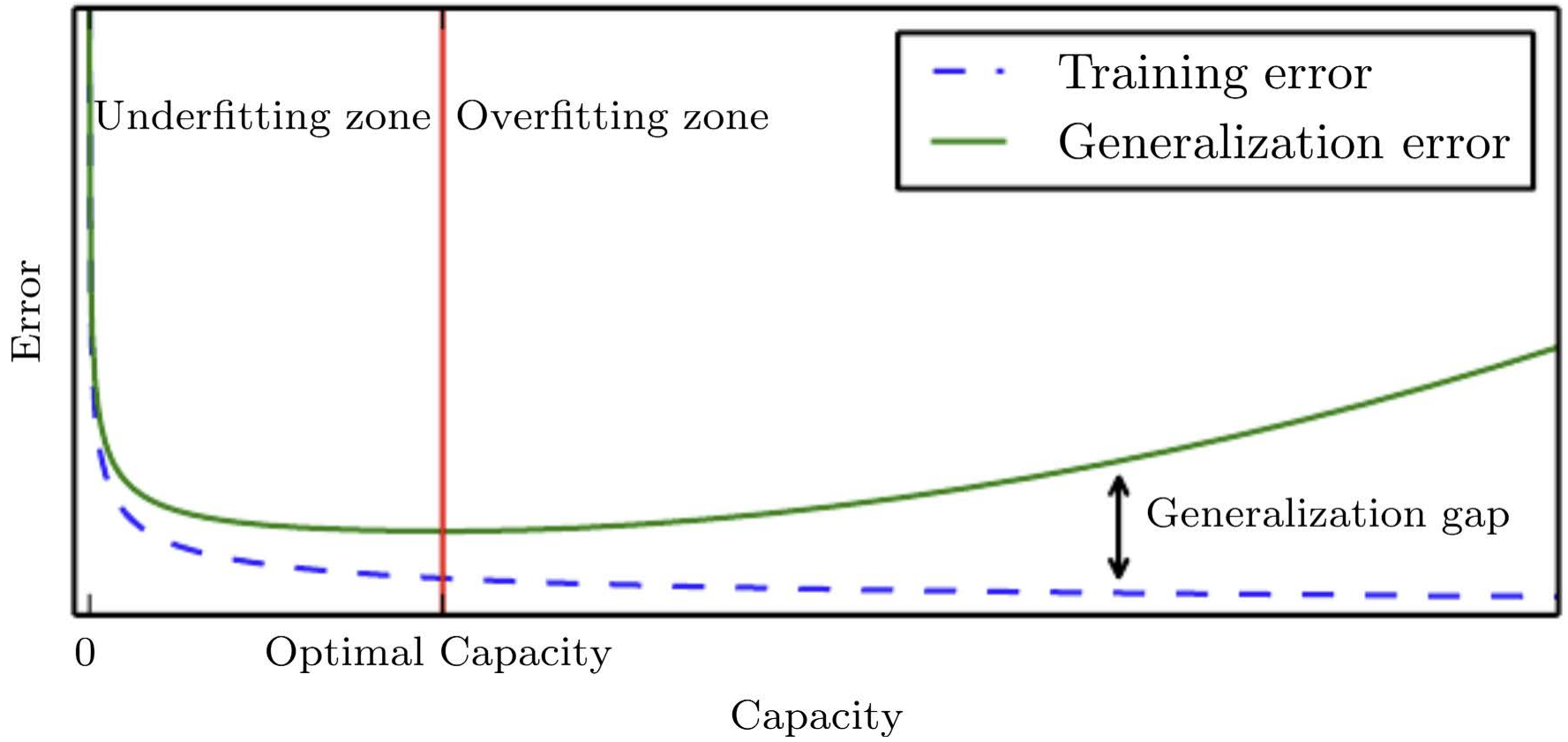
Basic Idea for Weight Initialization

- Variance changes across layers
  - $y = \sum_{i=1}^{n} w_i x_i$
  - $\mathrm{Var}(y) = n\mathrm{Var}(w)\ \mathrm{Var}(x)$
- In order to remove this change, we need to ensure
  - $\mathrm{Var}(w) \sim \frac{1}{n}$
- Random initialization
  - Gaussian or uniform distributions: $N(0, \frac{1}{n})$ or $U(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}})$

Commonly-used Initializations

- Xavier initialization
  - Consider different number of nodes in each layer
  - Consider the properties of sigmoid & tanh activation functions
  - $w \sim U(-\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}})$
- He initialization
  - Consider the properties of ReLU activation function
  - $w \sim N(0, \frac{2}{n})$

Overfitting w.r.t Model Complexity



http://www.deeplearningbook.org/contents/ml.html

Regularization

- Definition: “Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

  ---《Deep Learning》 Ian Goodfellow et al.

- Data augmentation
- DropOut
- Normalizations
- Weight decay
- Early stopping

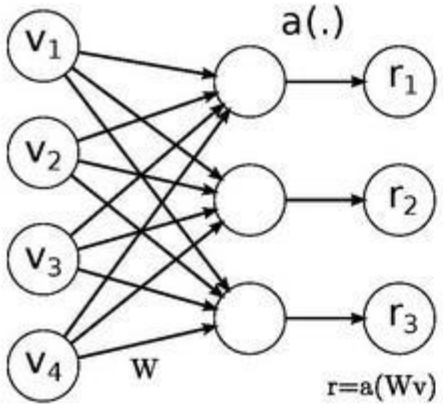
Data Augmentation

- Translating, rotation, scaling, randomly cropping
  - Very effective for object recognition
  - Be careful! ‘b’ v.s ‘d’, ‘6’ v.s ‘9’
- Noise injection
  - Inject into input data, e.g., adding Gaussian noise
  - Inject into hidden layers
  - Inject into output targets, label smoothing
- Leverage unlabeled data
  - Pretraining
  - Semi-supervised learning

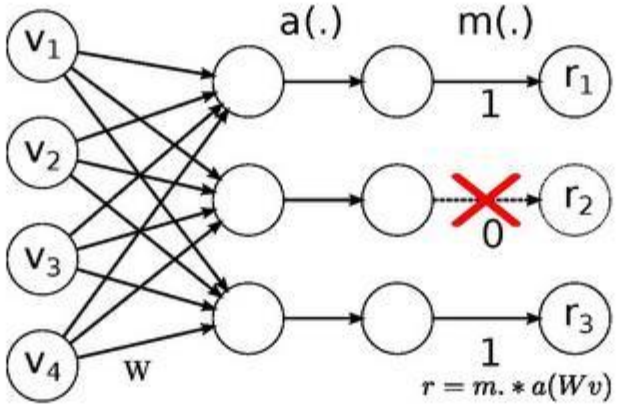
The best way to make a ML model generalize better is to train it on more data.

DropOut/DropConnect

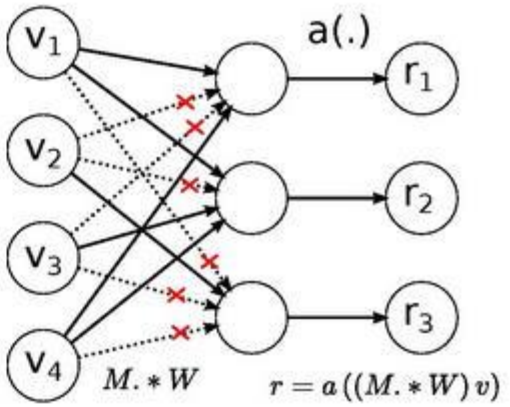
- Introduce noises to the training process
  - During training, multiply neuron output (or weight) by a random bit with probability of (1 - p)
  - How does it work during testing?



Original network



DropOut network



DropConnect network

DropOut/DropConnect

- For every sample, the number of parameters (e.g., edge weights in network structure) becomes effectively smaller, and thus the model becomes easier to train (in terms of both computational complexity and overfitting).
- DropOut/DropConnect can improve generalization ability, e.g., by reducing the Rademacher Average of the preceding layers.

$$\hat{R}_\ell(\mathcal{F}) \leq p\left(2\sqrt{k}dB_s n\sqrt{d}B_h\right)\hat{R}_\ell(\mathcal{G})$$

- Randomness in the training process can enable better exploration and avoid being trapped into local optimum.

Normalizations

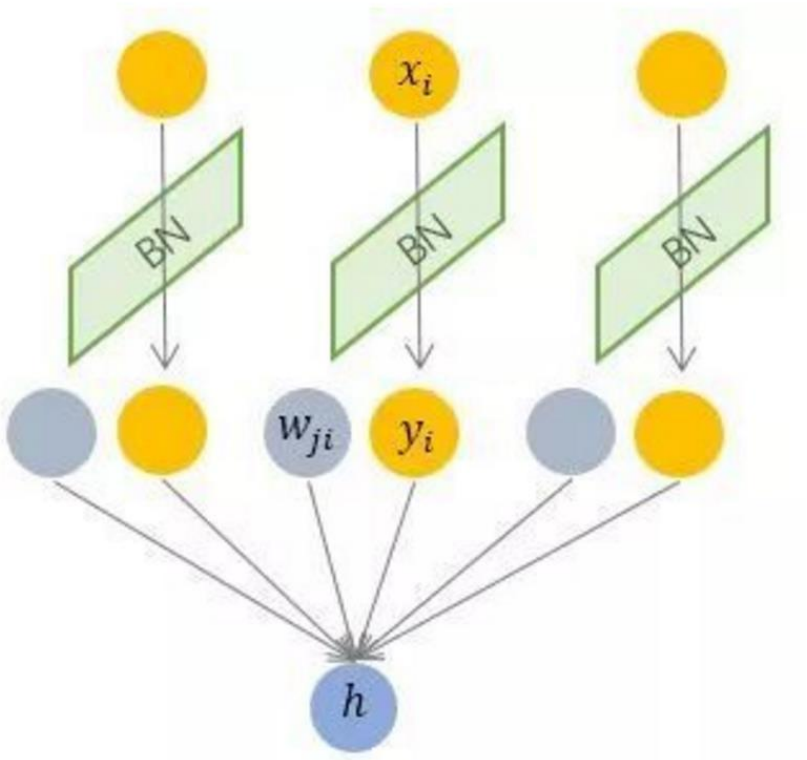
- Why normalization?
  - Normalization is a way to regulate the distribution of hidden outputs
    - The distribution of each layer’s inputs changes during training as the parameters of the previous layers change.
    - This will slow down the training by requiring lower learning rates and careful parameter initialization and makes it notoriously hard to train models with saturating nonlinearities.
- Normalization is also a way to restrict the possible value that hidden node or weight could take
  - Restricted functional class will lead to better generalization

Normalizations

$$h = f(g\frac{x-\mu}{\sigma} + b)$$

- Batch norm:
  - Vertical normalization of data (one dimension, multiple instances)
- Layer norm:
  - Horizontal normalization of data (one instance, multiple dimensions)
- Weight norm:
  - Normalization of weights

Batch Normalization



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

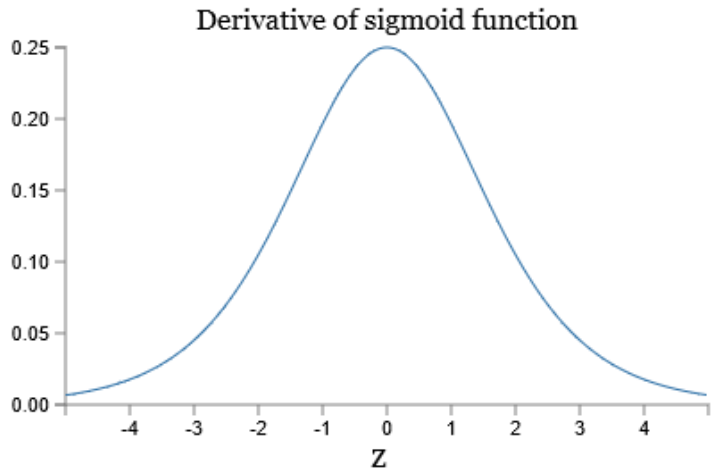
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Standardize the activations before nonlinear transformation across mini-batch

Batch Normalization

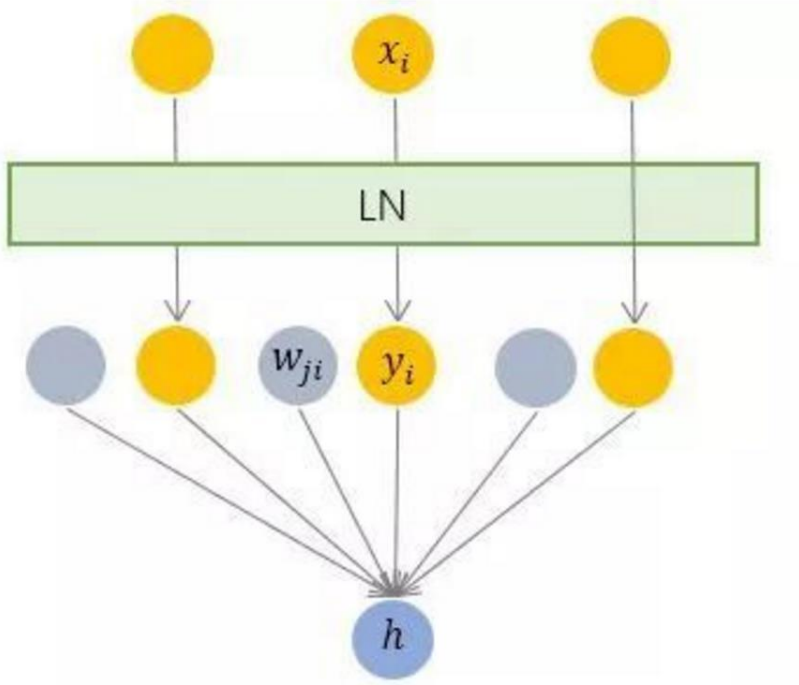
- Training time: use mini-batch for normalization;
- Test time: use global mean and standard deviation for normalization
- Batch normalization standardizes the activation z at each layer, and therefore can avoid gradient vanishing and allow relatively large learning rate.
- Batch normalization reduces the variance across mini-batches (and thus make the training data less complex), which can resolve the overfitting problem to some extent.

$\sigma' z$ is maximized after whitening



Layer Normalization

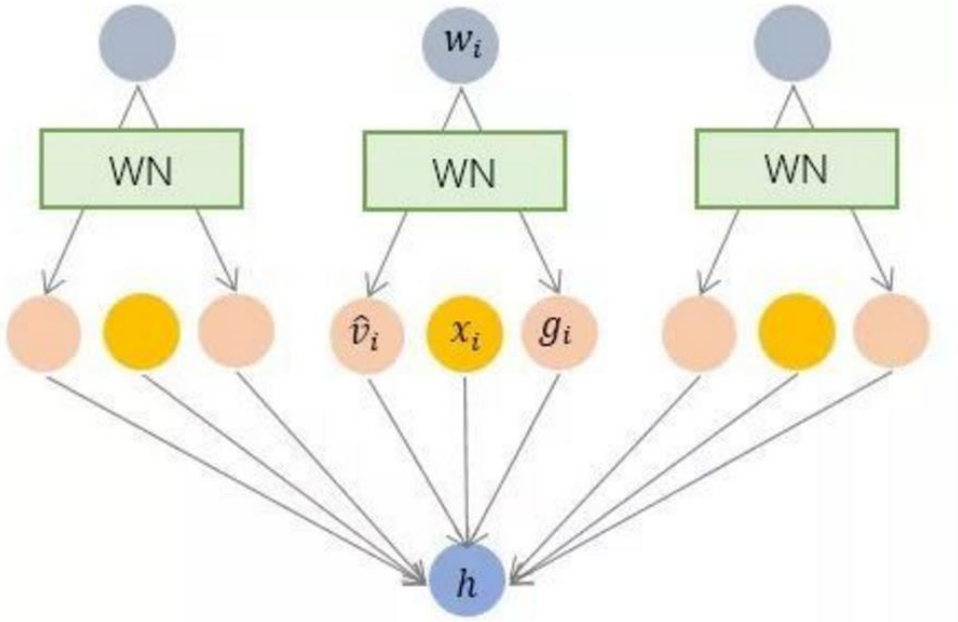
- Normalization with respect to one single instance
- Normalization across the output of different nodes (dimensions) in the same layer
- May potentially restrict the expressiveness power of the neural networks model



Weight Normalization

- Normalize the weight vector by its norm, to restrain the function class
- After normalization, the direction of the weight vector remains the same, but its norm changes to a predefined constant g.

$$w = g\frac{v}{\|v\|}$$



Other Regularization Strategies

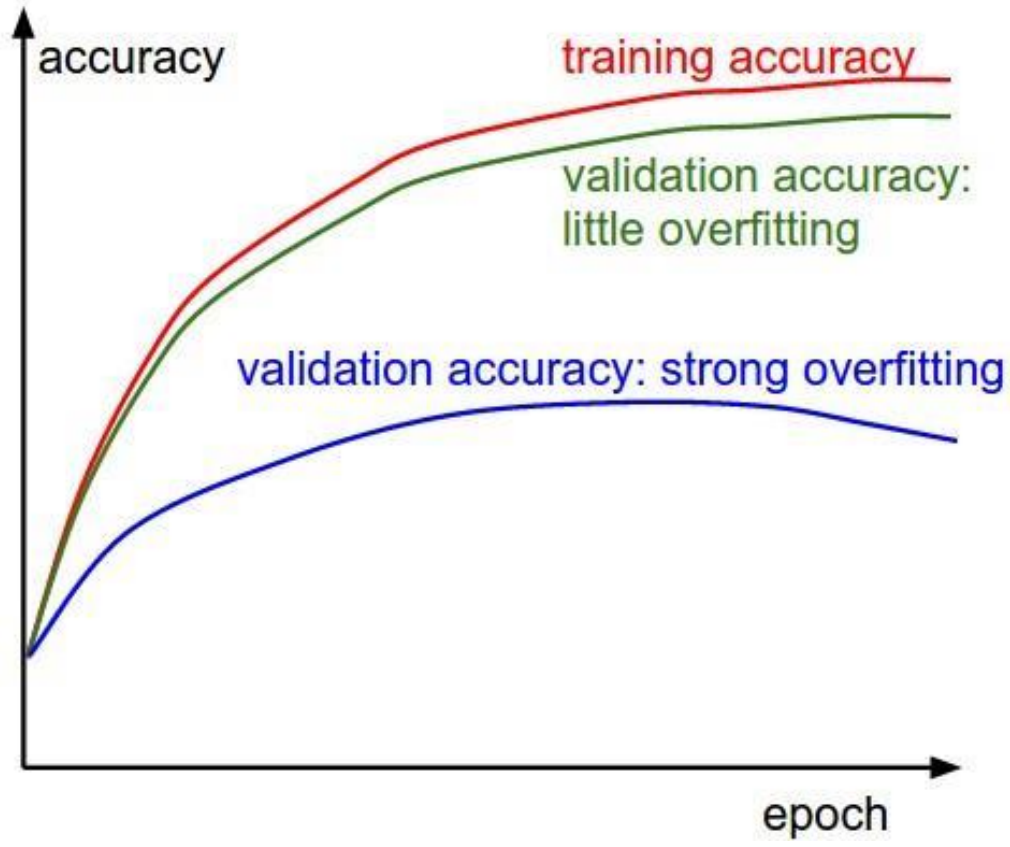
- L_2 parameter norm penalty (or weight decay)

$$J(\theta) = L(\theta) + \alpha\|\theta\|_2^2$$

- L_1 parameter norm penalty

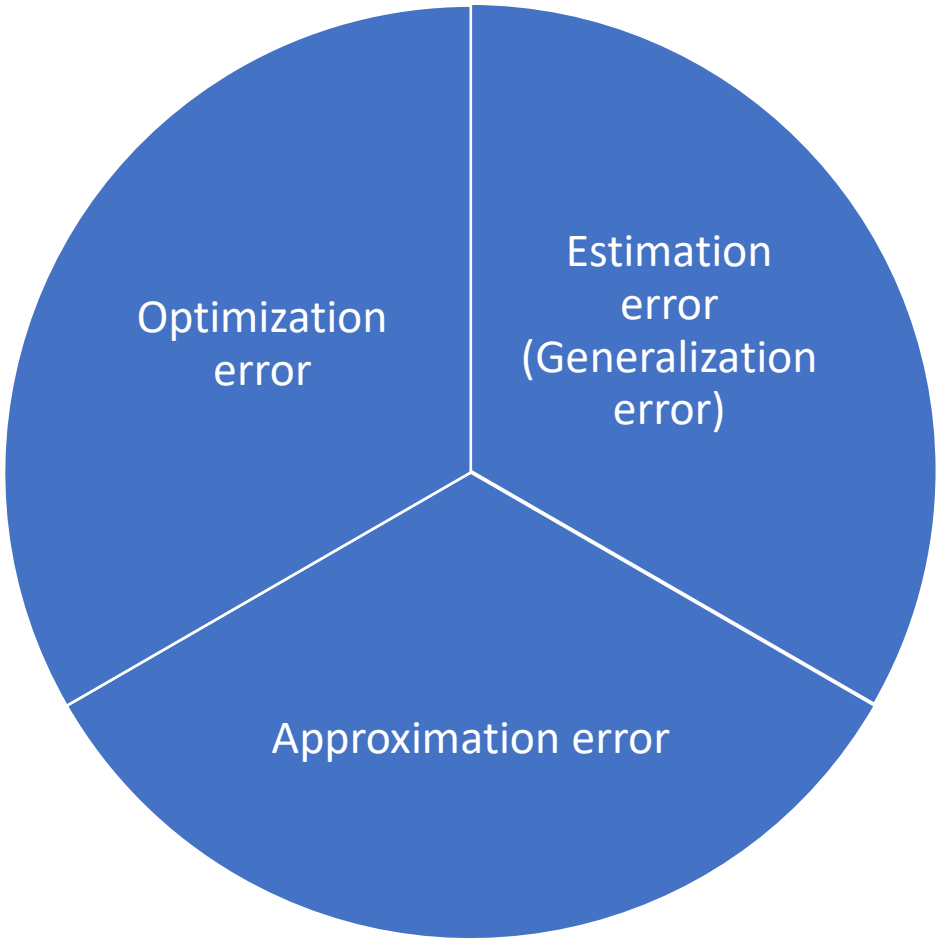
$$J(\theta) = L(\theta) + \alpha\|\theta\|_1$$

- Early stopping
  - Terminate the training when the validation error does not drop for a certain iterations



Deep Learning Theory

Deep Learning Theory

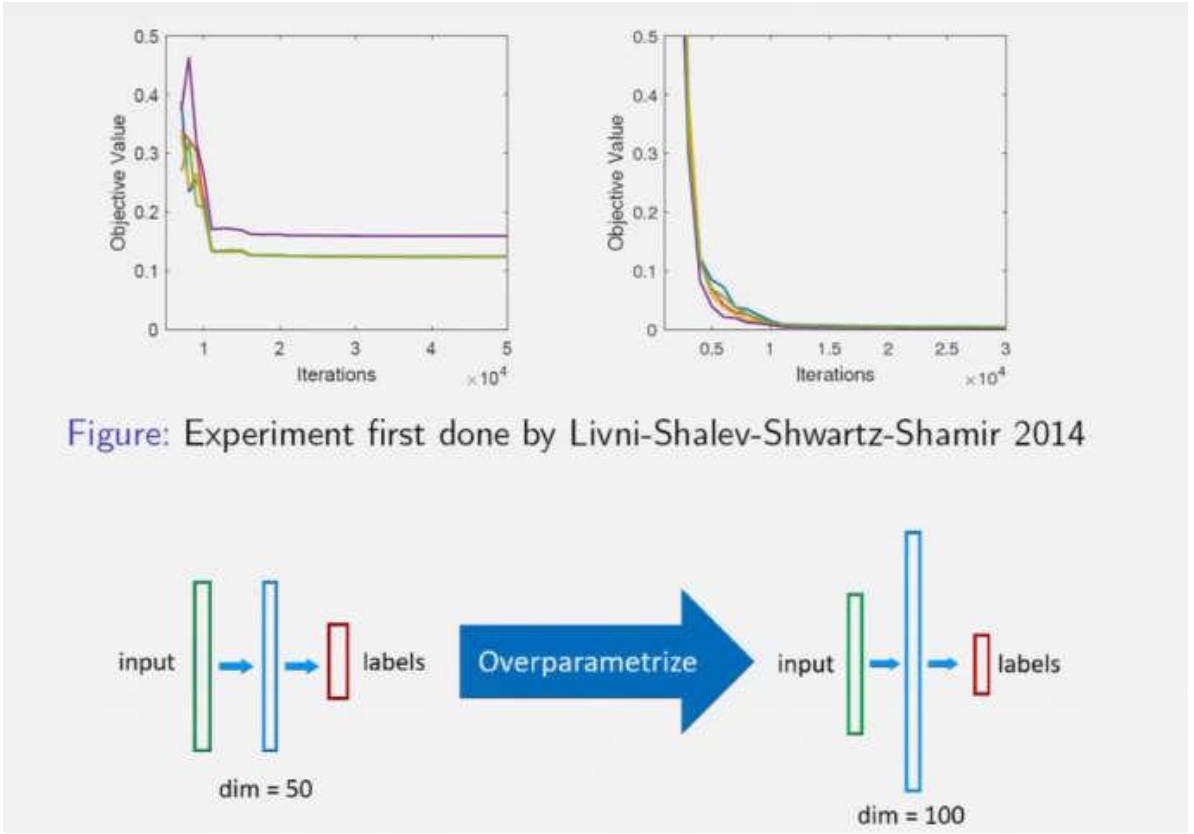


Theoretical Challenge of Deep Learning

- Optimization: Non-convex with exponentially many critical points (gradient=0)
- Generalization: Successful deep networks are big models (more parameters than samples).
- They are intertwined together:
  - The choice of optimization algorithm affects the generalization error
  - Regularizers for generalization also change the algorithm dynamics and loss landscape.
  - Practical observation: gradient methods find high quality solution.
- Approximation: show the benefit of depth.

Over-parameterization Eases Optimization

- Understand how over-parametrization helps training
  - Though the loss of DNN is nonconvex, overparameterized NN behaves like “convex” near the initialization
  - Coincide with practical observation: gradient methods find high quality solutions for wide DNN
  - Gradient descent converges to global minima for training over-parametrized neural network



Rademacher Average of Deep Neural Networks

- Let $\mathcal{F}_{L,W}$ denote the functional class of L-layer neural network parameterized by $W = \{W_1, \dots, W_L\}$, i.e., $\forall f \in \mathcal{F}_{L,W}$

$$f(x) = W_L \sigma(W_{L-1}(\dots \sigma(W_1 x)))$$

- where $\|W_i\|_F \leq B$, then Rademacher Average has an upper bound (Neyshabur et al. 2015):

$$RA(\mathcal{F}_{L,W}; n) \leq O\left(\frac{B^L}{\sqrt{n}}\right)$$

- Not rely on the number of parameters.
- Grow exponentially with L.

Reference

- Z. Allen-Zhu, Y. Li and Z. Song. Convergence theory for learning DNN via over-parameterization. ICML 2019
- H. Zhang, D. Yu, M. Yi, W. Chen and T-Y Liu. Convergence theory for learning over-parameterized ResNet: a full characterization. ArXiv 2019
- Baum, Eric B., and David Haussler. "What size net gives valid generalization?." Advances in neural information processing systems. 1989.
- Bartlett, Peter L., Vitaly Maiorov, and Ron Meir. "Almost linear VC dimension bounds for piecewise polynomial networks." Advances in Neural Information Processing Systems. 1999.
- B. Neyshabur, R. Tomioka, and N. Srebro. Norm-based capacity control in neural networks. COLT, 2015
- Bartlett P L, Harvey N, Liaw C, et al. Nearly-tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks[J]. Journal of Machine Learning Research, 2019, 20(63): 1-17.
- Bartlett, Peter L., Dylan J. Foster, and Matus J. Telgarsky. "Spectrally-normalized margin bounds for neural networks." NIPS 2017.
- John Shawe-Taylor and Omar Ravisplata. Statistical Learning Theory: A Hitchhiker’s Guide. NIPS 2018.
- Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators[J]. Neural networks, 1989, 2(5): 359-366.
- Barron A R. Universal approximation bounds for superpositions of a sigmoidal function[J]. IEEE Transactions on Information theory, 1993, 39(3): 930-945.
- S. Liang and R. Srikant. WHY DEEP NEURAL NETWORKS FOR FUNCTION APPROXIMATION? ICLR 2017
- Nadav Cohen, Or Sharir and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. COLT, 2016.
- Matus Telgarsky. Benefits of depth in neural networks. JMLR 2016
- Z. Lu, H. Pu, F. Wang, Z. Hu and L. Wang. The Expressive Power of Neural Networks: A View from the Width, NIPS 2017

Thanks!

http://web.ee.tsinghua.edu.cn/wqzhang