NYU TANDON SCHOOL OF ENGINEERING

AI4CE

Robot Perception

Introduction to Deep Learning

Dr. Chen Feng

cfeng@nyu.edu

ROB-GY 6203, Fall 2023

Overview

+ Overview of Deep Learning

++ Computational Graph

++ Back Propagation

+ DNN Architectures
    RNN, CNN, ResNet, VAE

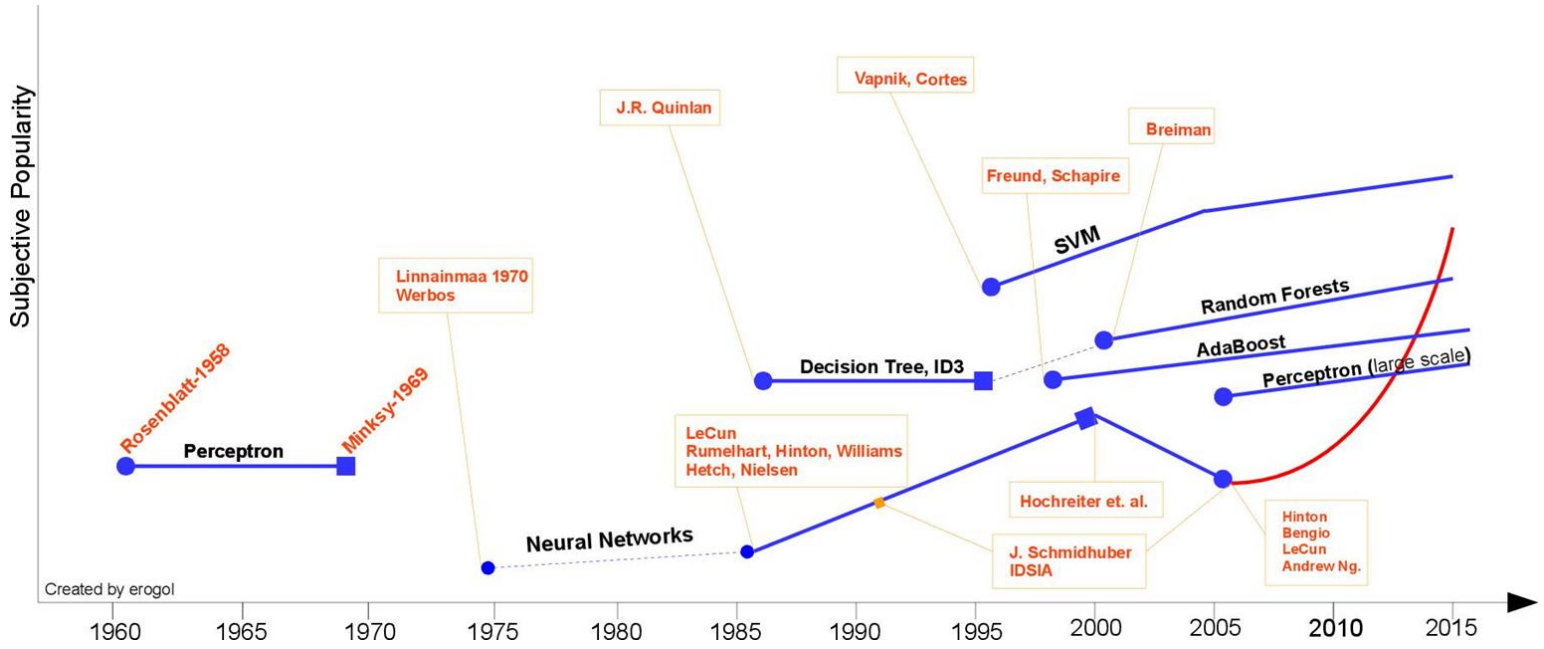
+ Common Practices

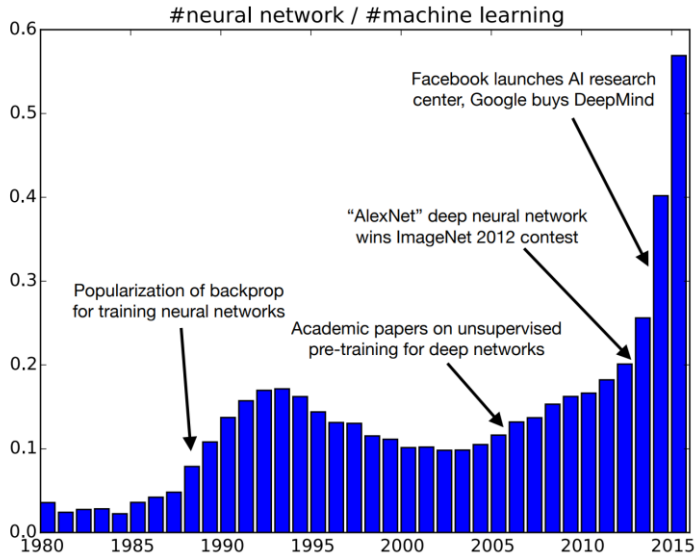
*: know how to code
++: know how to derive
+: know the concept

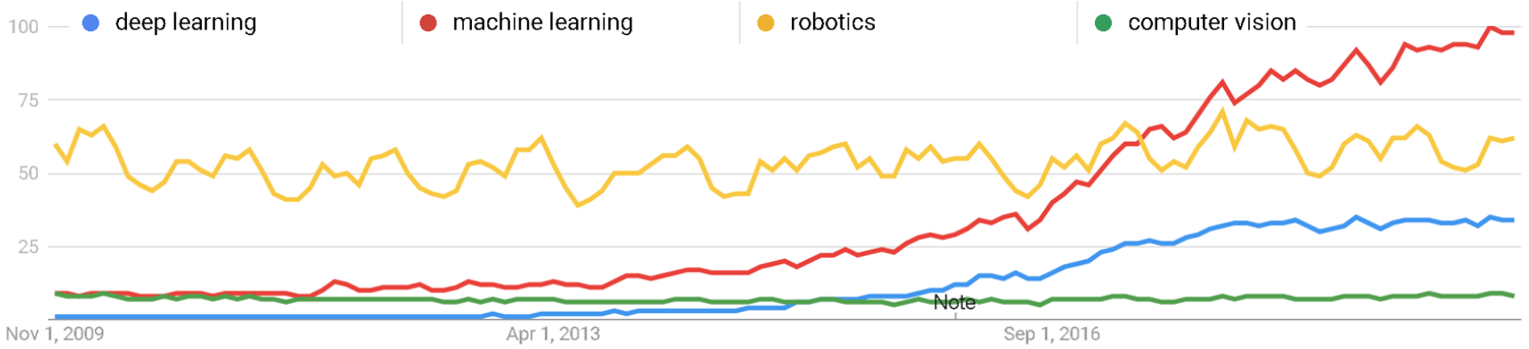
What to expect in this lecture?

- This lecture is about:
  - Quick overview/reminder of some of the basics of deep learning
  - Establishing a common ground of understanding for future weeks
    - Especially for engineering students who have not taken ML courses before
- This lecture is NOT about:
  - Replacing a full machine learning course
    - But we hope this lecture is enough for you to get started on reading and using these tools
  - A comprehensive review nor an in-depth analysis of deep learning
    - But we hope this lecture opens the doors for those who are interested

Recent History in Machine Learning

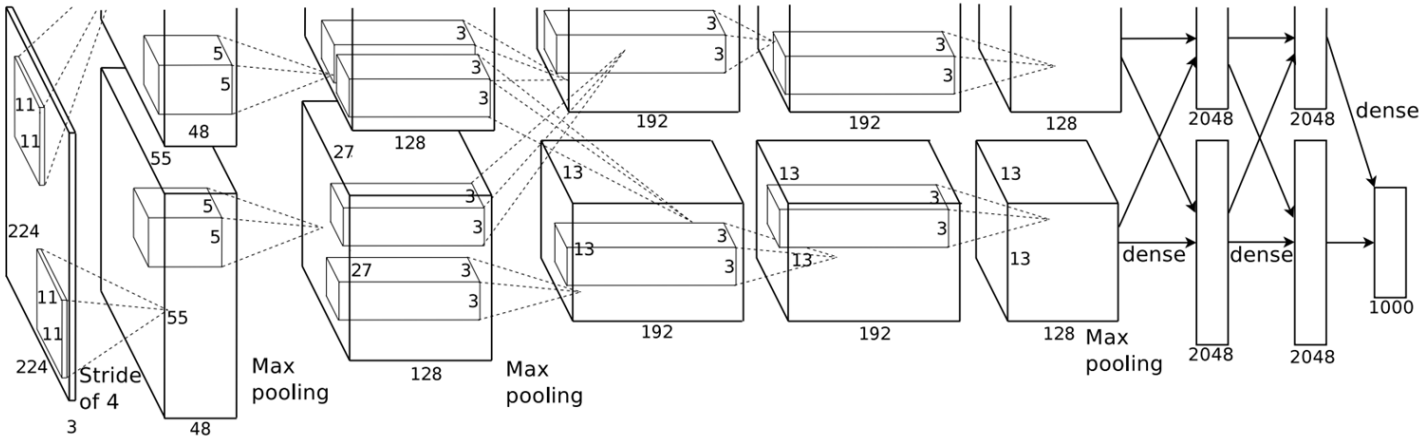




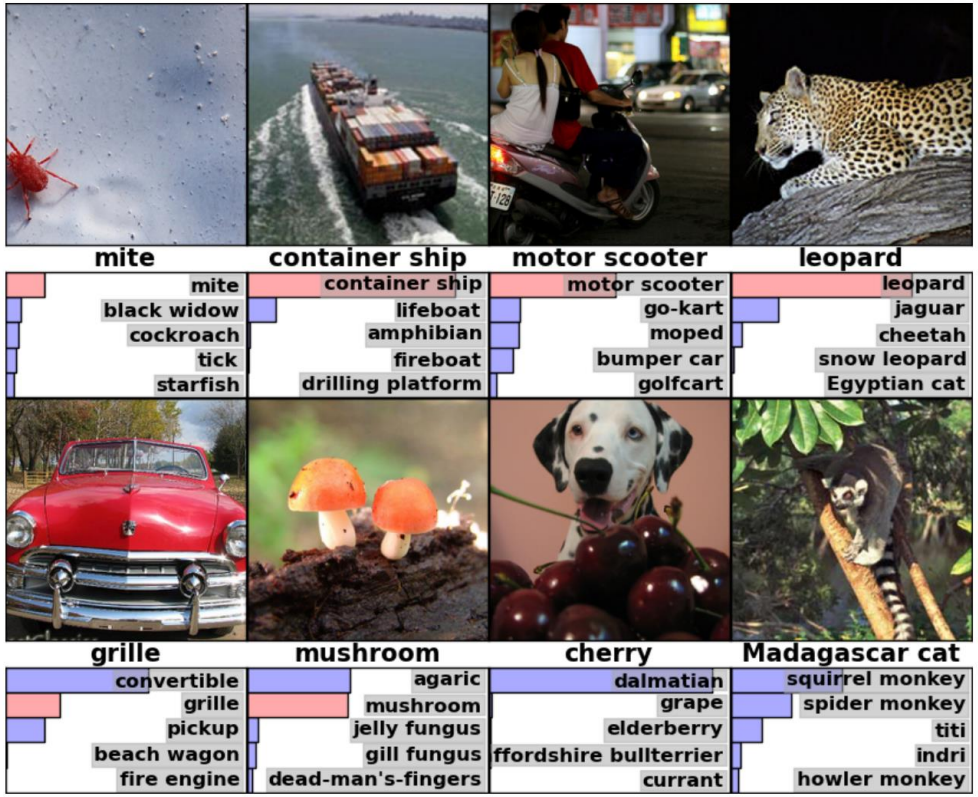


Slides from J. Zico Kolter 2017, and http://www.erogol.com/brief-history-machine-learning/

AlexNet and ImageNet



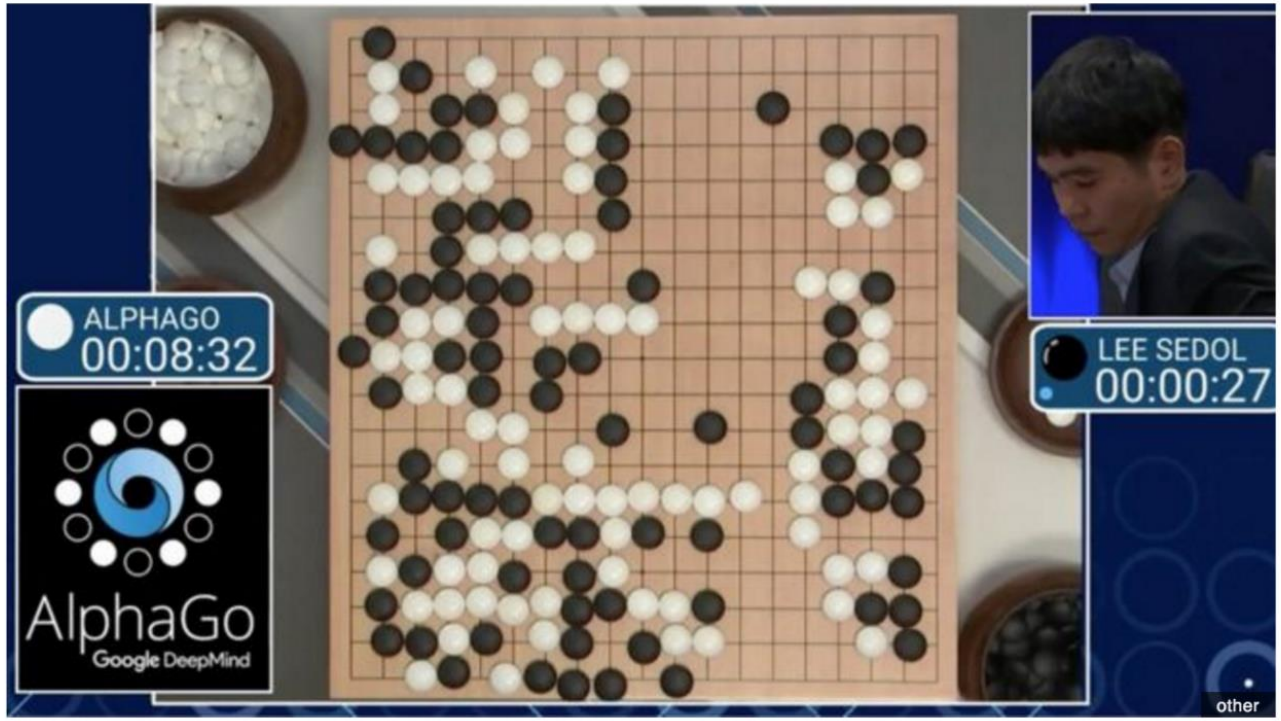
"AlexNet" (Krizhevsky et al., 2012), winning entry of ImageNet 2012 competition with a Top-5 error rate of 15.3% (next best system with highly engineered features based got 26.1% error)



AlphaGo

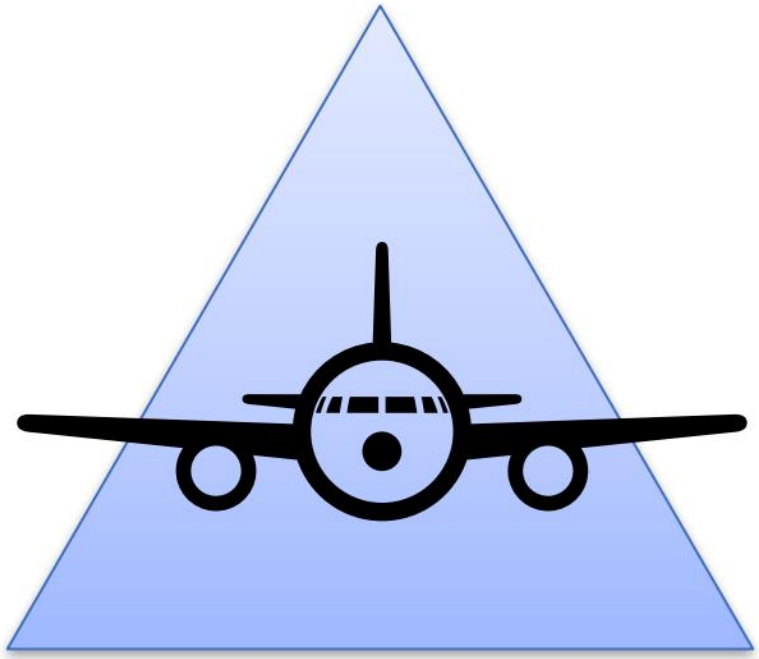
Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol

12 March 2016 | Technology



Why Deep Learning Now?

High capacity models (i.e., large VC dimension)



Lots of computing power

Lots of data

Slides from J. Zico Kolter 2017

Neural Networks for Machine Learning

- Three Components of Machine Learning
  - **Hypothesis Class**: what does the search space look like?
  - **Loss Function**: how good is a hypothesis?
  - **Optimization**: how to find a good hypothesis?

- Neural Network is just a class of hypothesis functions in machine learning
  - Composition of non-linear functions
  - Typically, these functions should be continuous and differentiable at *almost everywhere* between input and output

Before Deep Learning

$$h_\theta(x) = \theta^T x$$

Compose Multiple Hypothesis

Deep Learning

$$h_\theta(x) = f_2(W_2 f_1(W_1 x + b_1) + b_2)$$

How to Compose Multiple Linear Hypothesis Meaningfully?

Given $x \in \mathbb{R}^n$, suppose I run two machine learning algorithms with the hypothesis functions $h, \tilde{h} : \mathbb{R}^n \rightarrow \mathbb{R}$:

$$h(x) = W_2(W_1 x + b_1) + b_2 \qquad \tilde{h}(x) = W_3 x + b_3$$
$$W_1 \in \mathbb{R}^{k \times n}, b_1 \in \mathbb{R}^k, W_2 \in \mathbb{R}^{1 \times k}, b_2 \in \mathbb{R} \qquad W_3 \in \mathbb{R}^{1 \times n}, b_2 \in \mathbb{R}$$

Suppose we use the same data, minimize the same loss function, and are (somehow) able to achieve the *global* optima of both problems (assumed to be unique), which of the following will be true:

1. h achieves lower training loss but higher validation loss than $\tilde{h}$
2. h achieves lower training loss *and* lower validation loss than $\tilde{h}$
3. $\tilde{h}$ achieves lower training loss *and* lower validation loss than h
4. They will both perform identically
5. The performance depends on the data or choice of loss function

Slides from J. Zico Kolter 2017

Neural Networks Are Universal Function Approximators

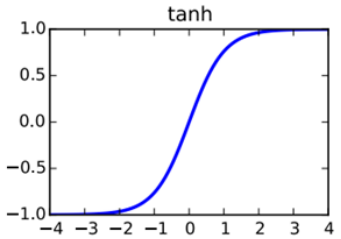
Neural networks are a simple extension of this idea, where we additionally apply a non-linear function after each linear transformation

$$h_\theta(x) = f_2(W_2 f_1(W_1 x + b_1) + b_2)$$

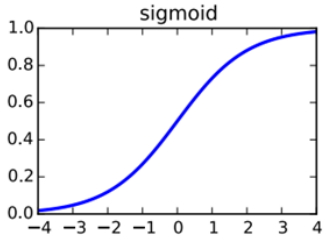
where $f_1, f_2 : \mathbb{R} \to \mathbb{R}$ are a non-linear functions (applied elementwise)

Common choices of f_i:

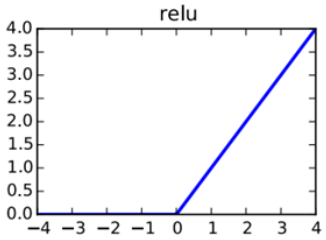
Hyperbolic tangent: $f(x) = \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$



Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

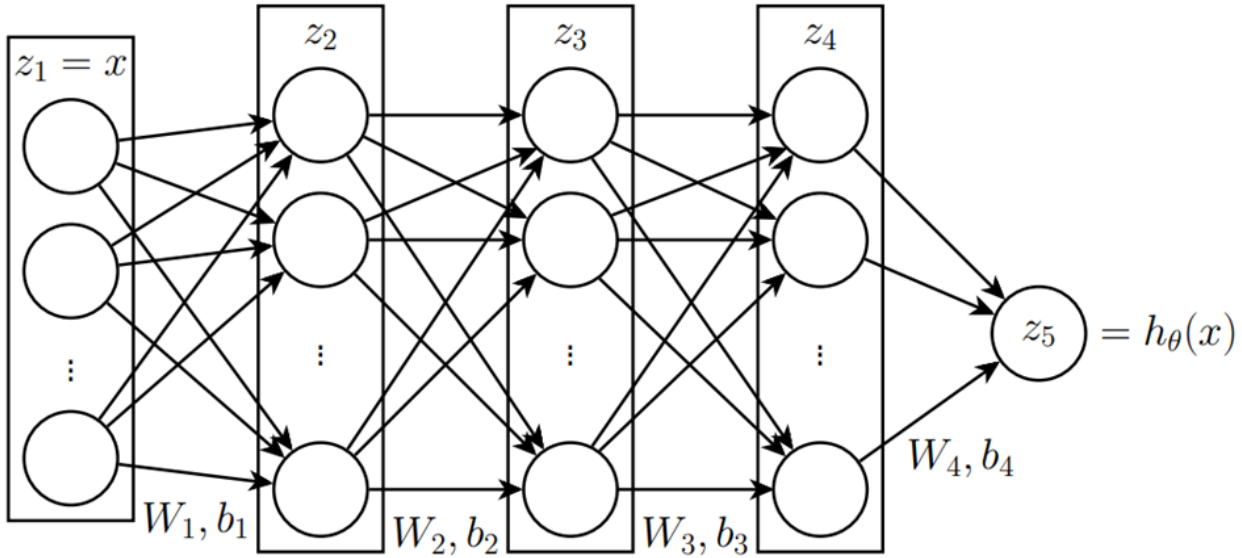


Rectified linear unit (ReLU): $f(x) = \max\{x, 0\}$



Deep Learning

"Deep learning" refers (almost always) to machine learning using neural network models with multiple hidden layers



Hypothesis function for k-layer network

$$z_{i+1} = f_i(W_i z_i + b_i), \qquad z_1 = x, \qquad h_\theta(x) = z_k$$

(note the z_i here refers to a vector, not an entry into vector)

Training Deep Networks

How do we solve the optimization problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

Not a convex problem, so we don't expect to find global optimum, but we will instead be content with *local* solutions

Just use gradient descent as normal (or rather, a version called stochastic gradient descent)

Stochastic Gradient Descent

Key challenge for neural networks: often have very large number of samples, computing gradients can be computationally intensive.

Traditional gradient descent computes the gradient with respect to the sum over *all examples*, then adjusts the parameters in this direction

$$\theta \leftarrow \theta - \alpha \nabla_\theta \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}, y^{(i)}) = \theta - \alpha \sum_{i=1}^{m} \nabla_\theta \ell(h_\theta(x^{(i)}, y^{(i)})$$

Alternative approach, *stochastic gradient descent* (SGD): adjust parameters based upon just *one* sample

$$\theta \leftarrow \theta - \alpha \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$$

and then repeat these updates for all samples

Gradient Descent vs. SGD

Gradient descent, repeat:

- For $i = 1, \ldots, m$:

$$g^{(i)} \leftarrow \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

- Update parameters:

$$\theta \leftarrow \theta - \alpha \sum_{i=1}^{m} g^{(i)}$$

Stochastic gradient descent, repeat:

- For $i = 1, \ldots, m$:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

In practice, stochastic gradient descent uses a small collection of samples, not just one, called a *minibatch*

Slides from J. Zico Kolter 2017

A Unified Way to Understand All Existing Deep Neural Networks

- Let's start from the simplest neural network example
  - Univariate logistic least squares (logistic regression)

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

  - x: **input**
  - w,b: learnable parameters of the model, i.e., **weights** of the model
  - z: learned **feature**
  - σ: sigmoid function, a **non-linear activation**
  - y: **output** of the model
  - t: **ground truth**, aka, target
  - L: MSE/L2 **loss**
- Let's compute the loss' derivative w.r.t. the weights

Chain Rule

How you would have done it in calculus class

$$\mathcal{L} = \frac{1}{2}(\sigma(wx+b)-t)^2$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx+b)-t)^2 \\
&= (\sigma(wx+b)-t)\frac{\partial}{\partial w}(\sigma(wx+b)-t) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial w}(wx+b) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)x
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial}{\partial b}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial b}(\sigma(wx+b)-t)^2 \\
&= (\sigma(wx+b)-t)\frac{\partial}{\partial b}(\sigma(wx+b)-t) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial b}(wx+b) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)
\end{aligned}$$

What are the disadvantages of this approach?

A More Structured Way to Apply Chain Rule

Computing the loss:

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

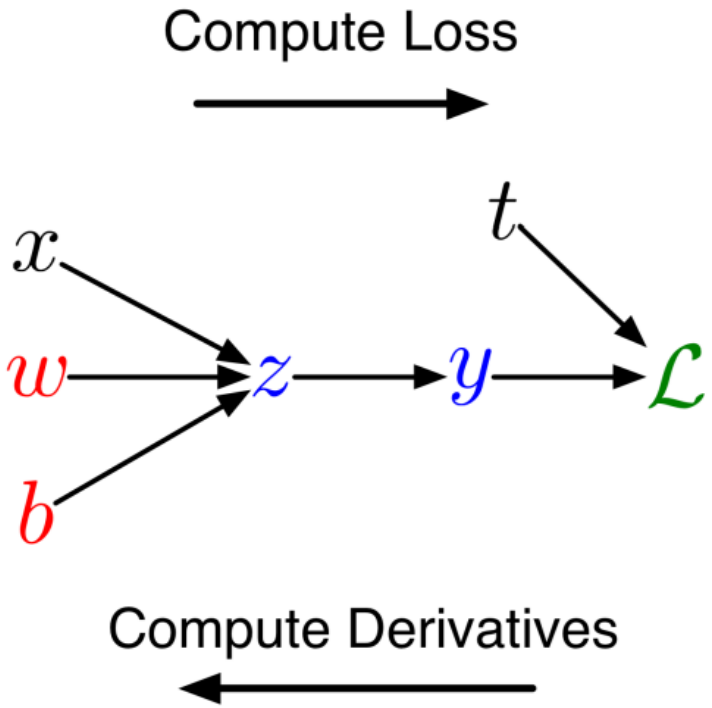
Computing the derivatives:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = y - t$$
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} \cdot \sigma'(z)$$
$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} \cdot x$$
$$\frac{\partial\mathcal{L}}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}$$

Remember, the goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.

A Symbolic Computation Graph

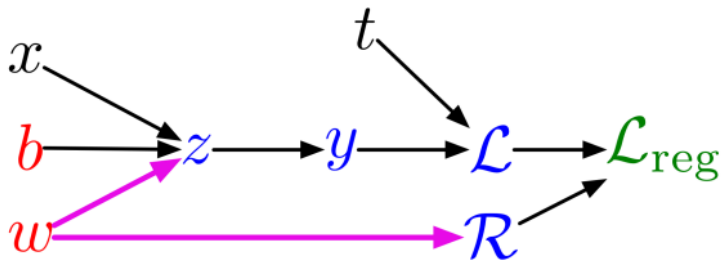
- We can diagram out the computations using a computation graph.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes.



A Symbolic Computation Graph

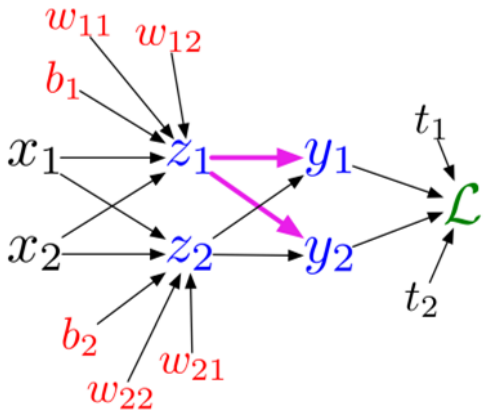
Problem: what if the computation graph has fan-out > 1?
This requires the multivariate Chain Rule!

L_2-Regularized regression



$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Multclass logistic regression

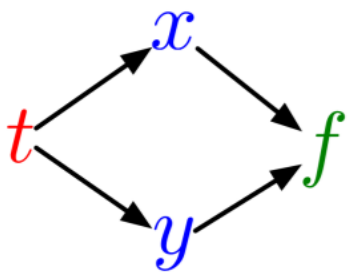


$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$
$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$
$$\mathcal{L} = -\sum_k t_k \log y_k$$

A Symbolic Computation Graph

- Suppose we have a function $f(x, y)$ and functions $x(t)$ and $y(t)$. (All the variables here are scalar-valued.) Then

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$



- Example:

$$f(x, y) = y + e^{xy}$$
$$x(t) = \cos t$$
$$y(t) = t^2$$

- Plug in to Chain Rule:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$
$$= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t$$

A Symbolic Computation Graph

- In the context of backpropagation:

Mathematical expressions to be evaluated

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

Values already computed by our program

- In our notation:

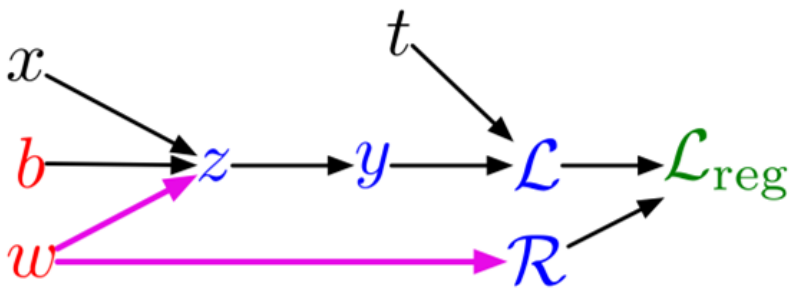
$$\bar{t} = \bar{x} \cdot \frac{\mathrm{d}x}{\mathrm{d}t} + \bar{y} \cdot \frac{\mathrm{d}y}{\mathrm{d}t}$$

Simplify notation (because there is always only one final loss function in DL)**:**

$$\frac{\partial\, \boldsymbol{Loss}}{\partial\, \boldsymbol{Variable}} \Leftrightarrow \overline{\boldsymbol{Variable}}$$

A Symbolic Computation Graph

Example: univariate logistic least squares regression



Forward pass:

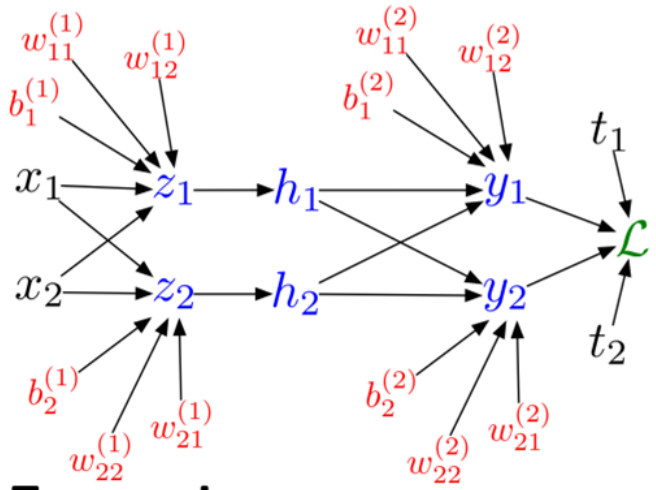
$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Backward pass:

$$\overline{\mathcal{L}_{\text{reg}}} = 1$$
$$\overline{\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\,\lambda$$
$$\overline{\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}$$
$$\overline{y} = \overline{\mathcal{L}}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = \overline{\mathcal{L}}\,(y - t)$$
$$\overline{z} = \overline{y}\frac{\mathrm{d}y}{\mathrm{d}z} = \overline{y}\,\sigma'(z)$$
$$\overline{w} = \overline{z}\frac{\partial z}{\partial w} + \overline{\mathcal{R}}\frac{\mathrm{d}\mathcal{R}}{\mathrm{d}w} = \overline{z}\,x + \overline{\mathcal{R}}\,w$$
$$\overline{b} = \overline{z}\frac{\partial z}{\partial b} = \overline{z}$$

A Symbolic Computation Graph

Multilayer Perceptron (multiple outputs):



Forward pass:

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (y_k - t_k)^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{y_k} = \overline{\mathcal{L}} (y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k} h_i$$

$$\overline{b_k^{(2)}} = \overline{y_k}$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

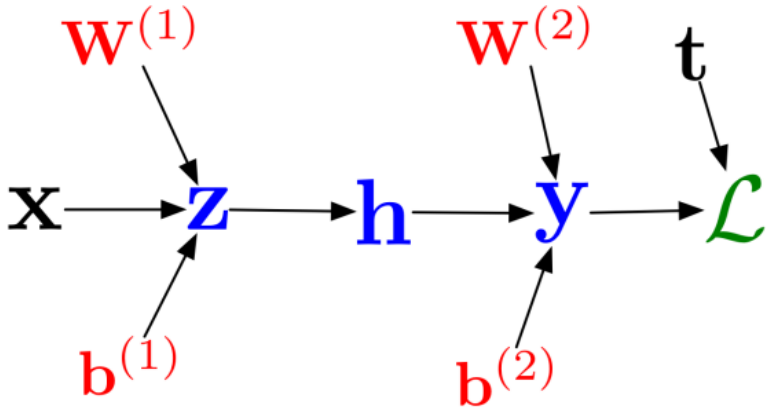
$$\overline{z_i} = \overline{h_i} \sigma'(z_i)$$

$$\overline{w_{ij}^{(1)}} = \overline{z_i} x_j$$

$$\overline{b_i^{(1)}} = \overline{z_i}$$

A Symbolic Computation Graph

In vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{h} = \sigma(\mathbf{z})$$
$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

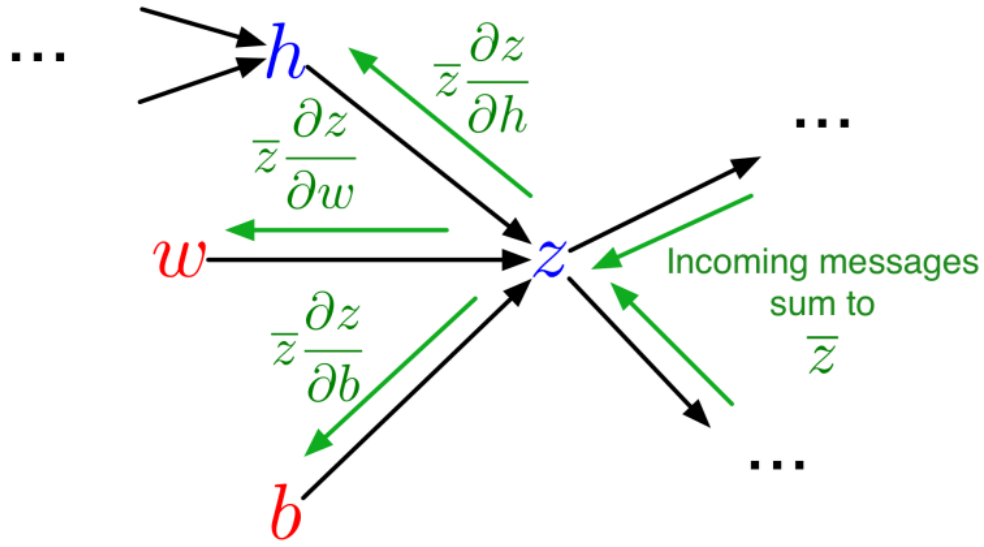
Backward pass:

$$\overline{\mathcal{L}} = 1$$
$$\overline{\mathbf{y}} = \overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$
$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$
$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$
$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$
$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \cdot \sigma'(\mathbf{z})$$
$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$
$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Slides from Roger Grosse

A Symbolic Computation Graph

Backprop as message passing:



- Each node receives a bunch of messages from its children, which it aggregates to get its error signal. It then passes messages to its parents.
- This provides modularity, since each node only has to know how to compute derivatives with respect to its arguments, and doesn't have to know anything about the rest of the graph.

Slides from Roger Grosse

A Symbolic Computation Graph

Full backpropagation algorithm:

Let $v_1, \ldots, v_N$ be a topological ordering of the computation graph (i.e. parents come before children.)

v_N denotes the variable we're trying to compute derivatives of (e.g. loss).

forward pass

For $i = 1, \ldots, N$

Compute v_i as a function of $\text{Pa}(v_i)$

backward pass

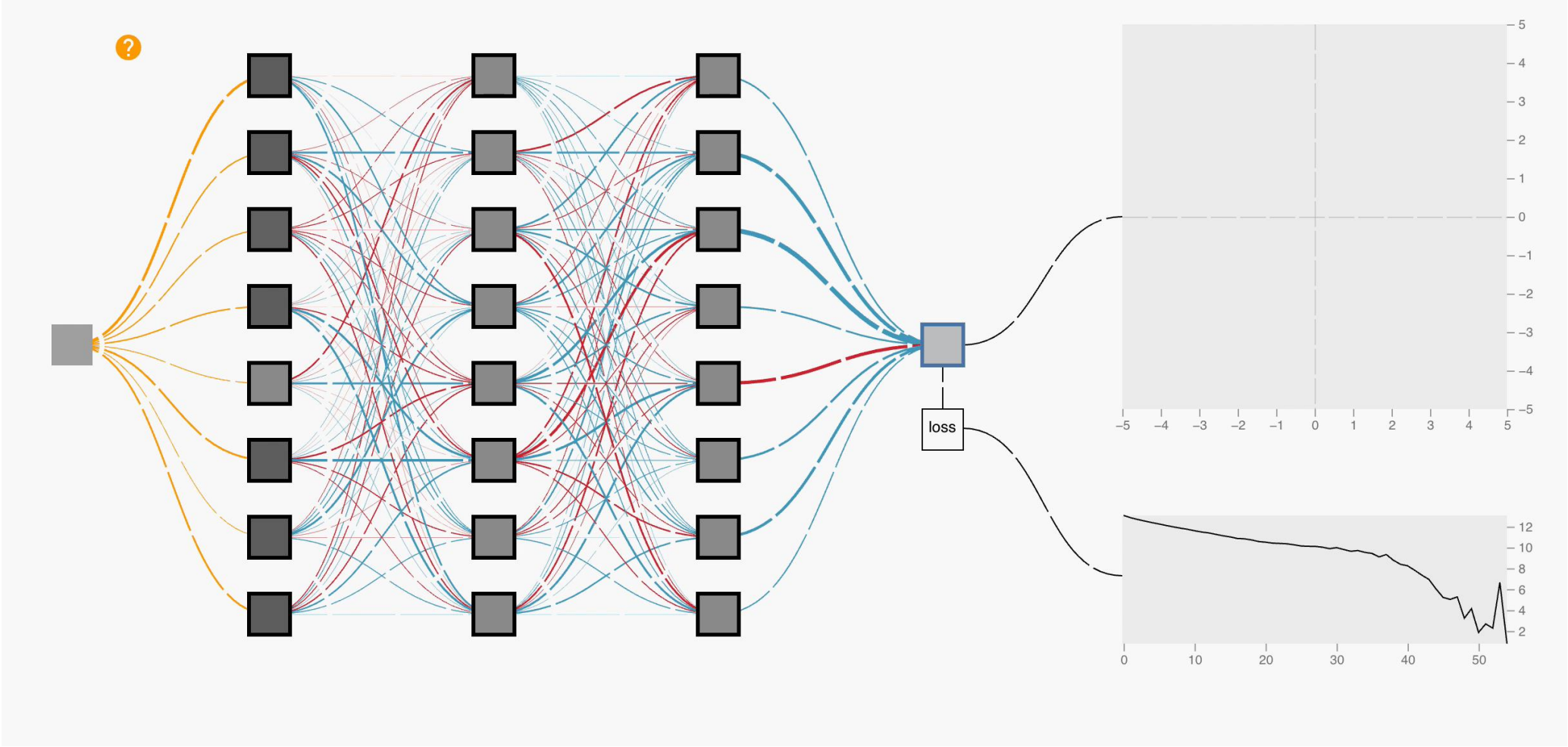
$\overline{v_N} = 1$

For $i = N - 1, \ldots, 1$

$$\overline{v_i} = \sum_{j \in \text{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$$

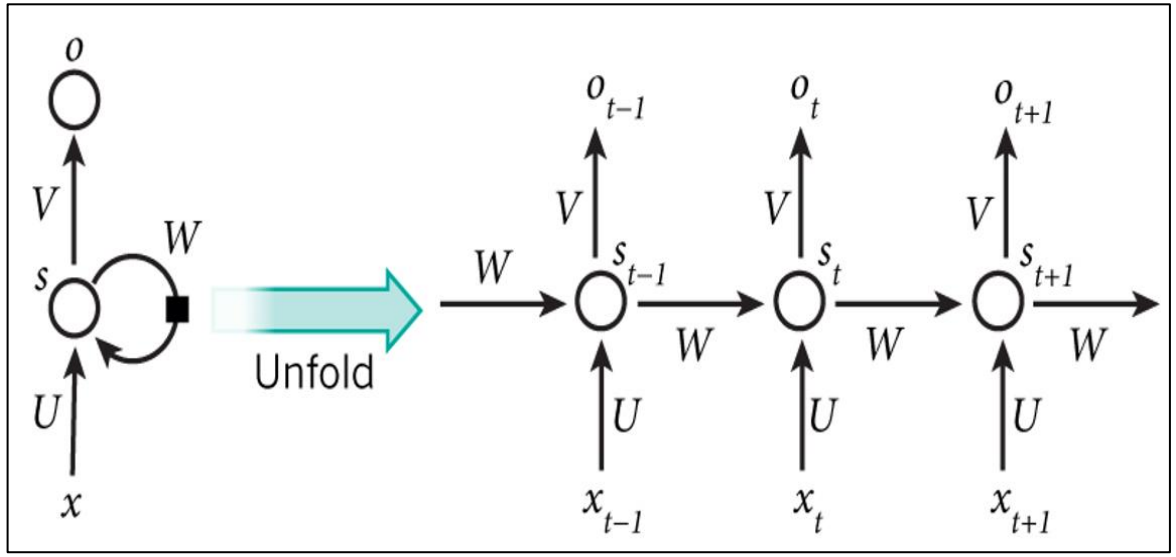
- Use $\overline{y}$ to denote the derivative $\mathrm{d}\mathcal{L}/\mathrm{d}y$, sometimes called the error signal.

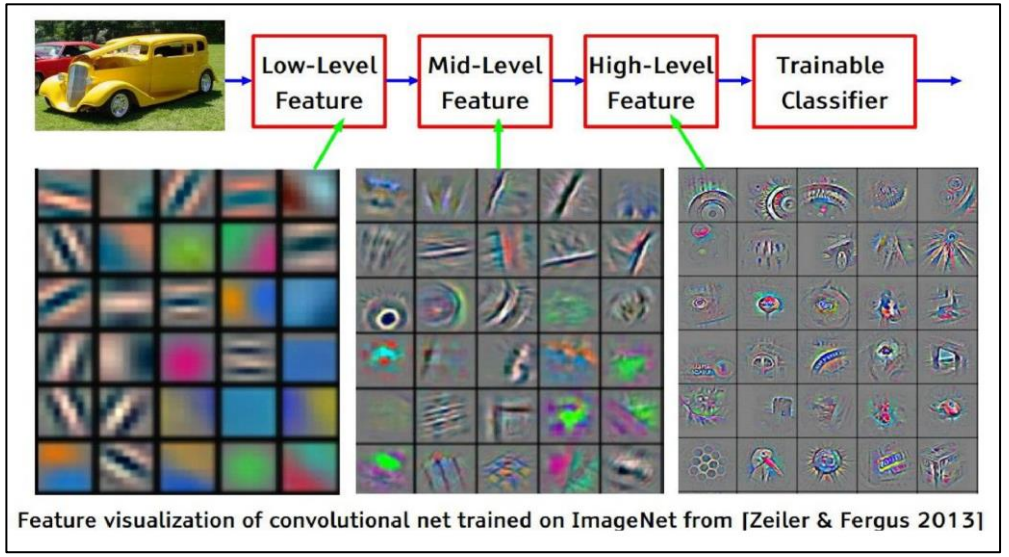
The Whole Picture

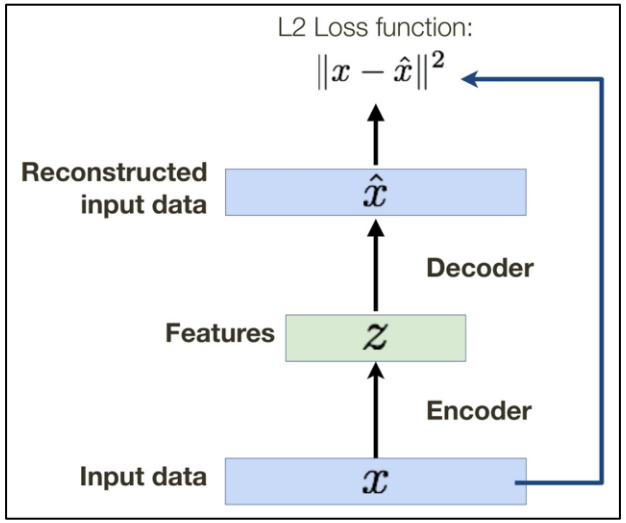


Donald Bertucci and Minsuk Kahng - https://xnought.github.io/backprop-explainer/

A Crash Course on Various Deep Nets







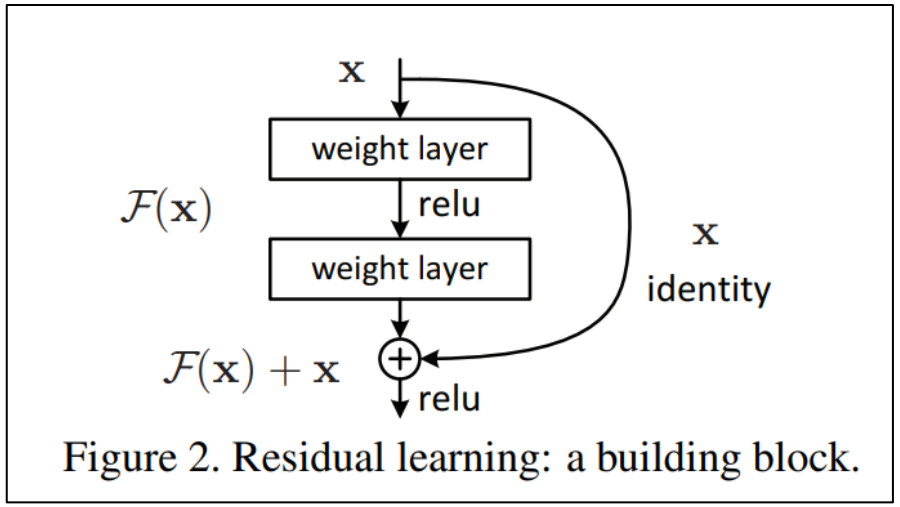


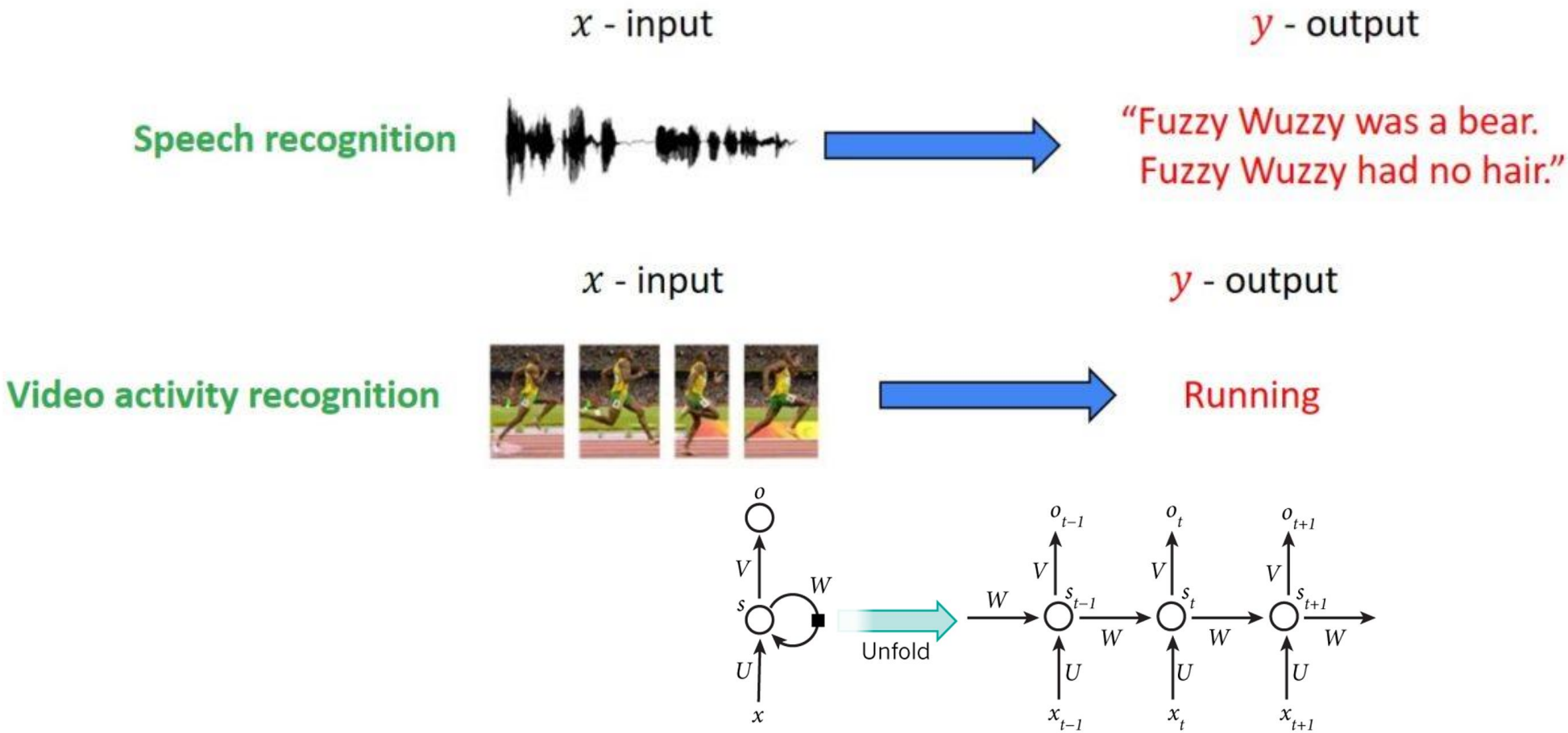
Figure 2. Residual learning: a building block.

Recurrent Neural Network

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Basics of RNN

Recurrent Neural networks (RNN) are a class of neural networks that possess the capability to analyse **sequential data which have dependencies** within them



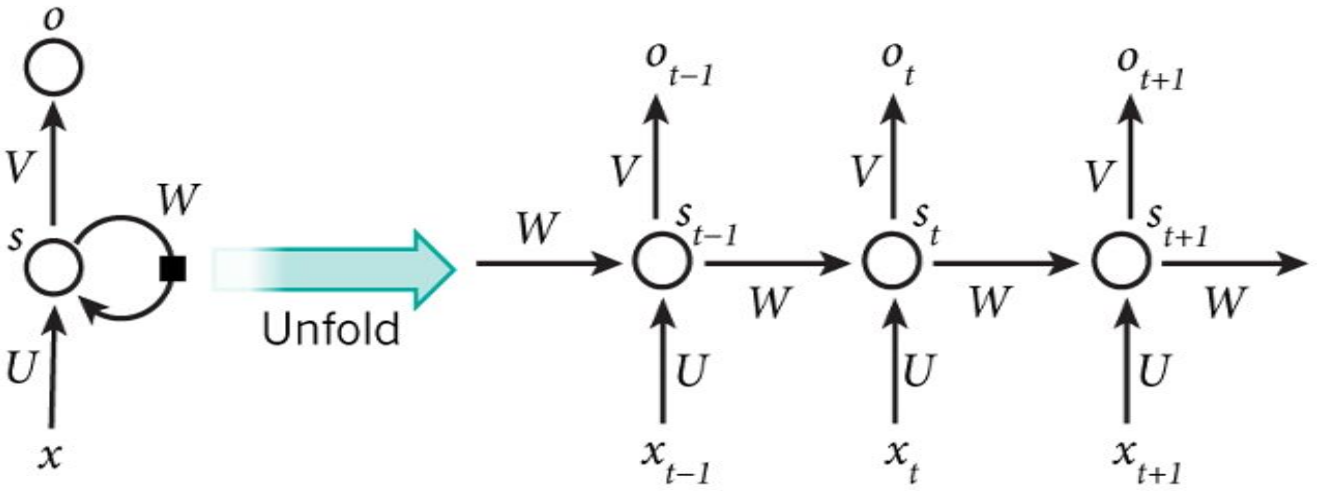
Basics of RNN: a weight-shared MLP between different time steps

- x_t is the **input** at time step t
- s_t is the **hidden state** at time step t

$$s_t = f(\boldsymbol{U}x_t + \boldsymbol{W}s_{t-1})$$

- f is a non-linear activation function (e.g.: tanh)
- o_t is the output at step t

$$o_t = softmax(\boldsymbol{V}s_t)$$



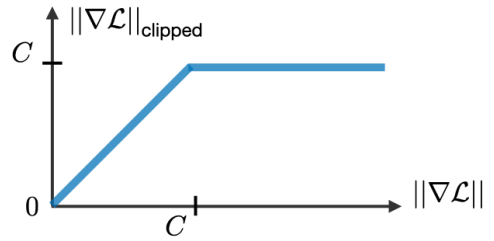
A Problem in Vanilla RNN: Long Term Dependency

- Computing gradients involve long chain of bounded activation value multiplications that lead to **vanishing gradients**.

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} = (\hat{y}_3 - y_3) \otimes s_3$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

- Vanishing/exploding gradient
  - Often encountered in the context of RNNs.
  - Because it is difficult to capture long term dependencies due to **multiplicative gradient** that can be exponentially decreasing/increasing with respect to the number of layers.
- Exploding gradient problem can be alleviated using gradient clipping



One Solution: Long Short Term Memory Networks (LSTM)

- LSTMs are special types of RNNs where the innate structure possesses the capability to handle long term dependency.
- This is achieved by using multiple **gates** that pass information from one time step to another with certain restrictions and modifications.
- This process allows the network to keep in memory the essential information and discard the irrelevant information that is not useful for prediction.

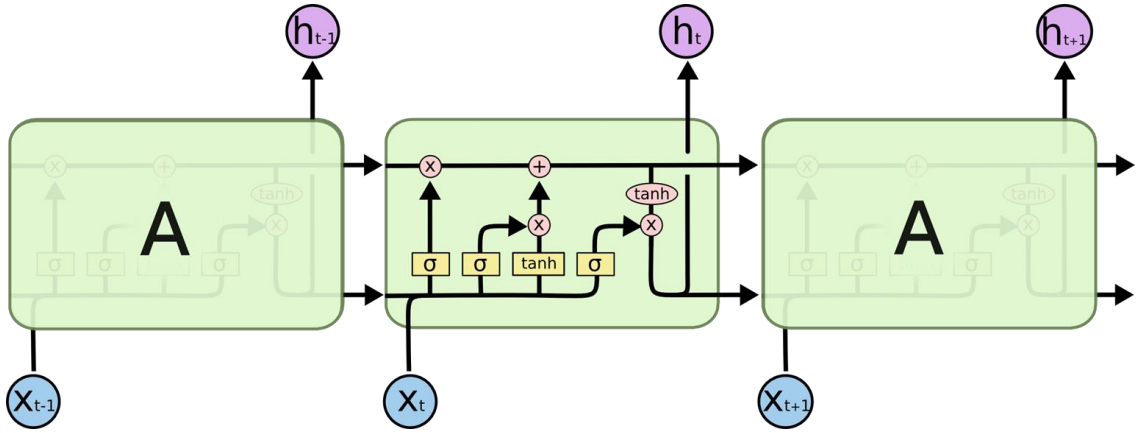
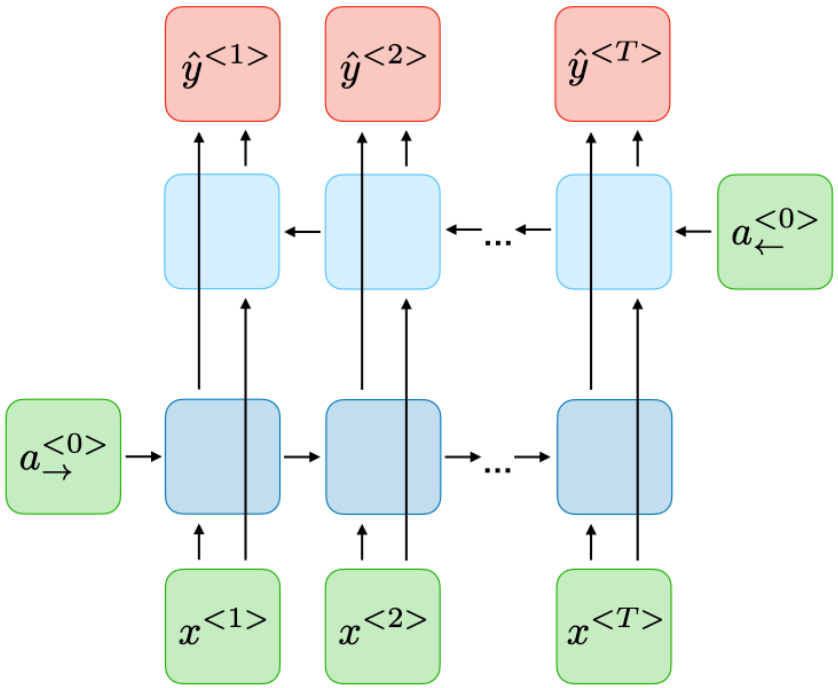


Image from Colah : https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Variants of RNN

Bidirectional (BRNN)



Deep (DRNN) a.k.a., **stacked RNN**

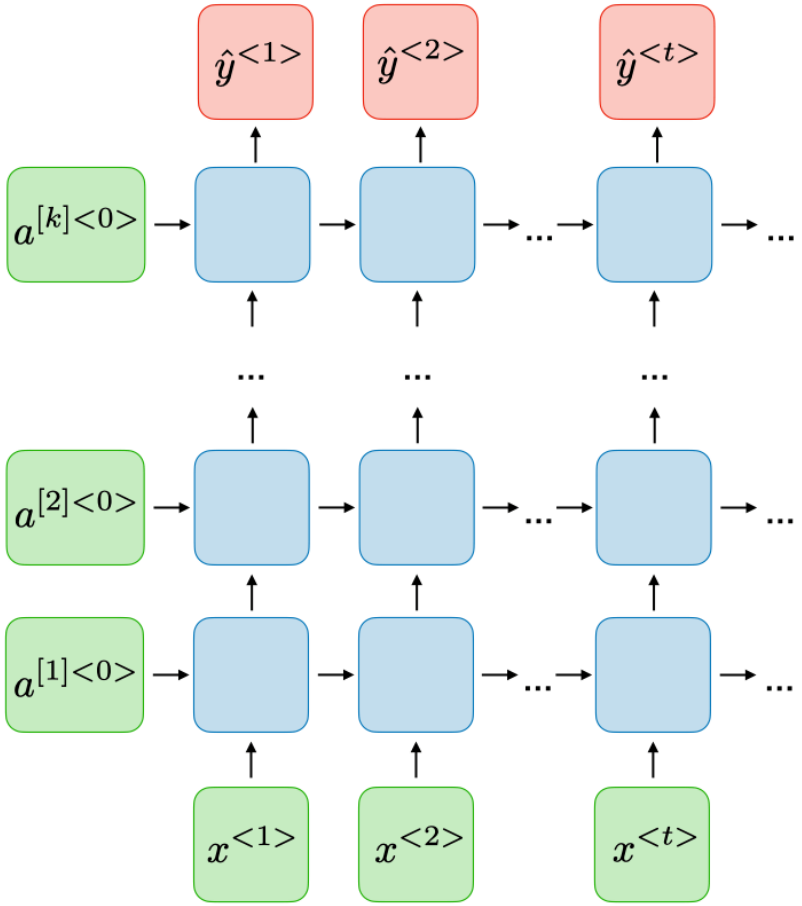


Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Applications of RNN

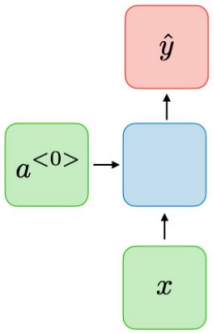
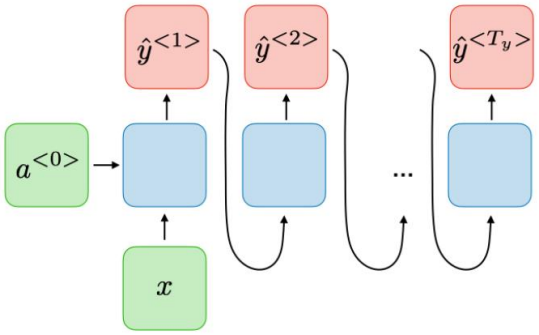
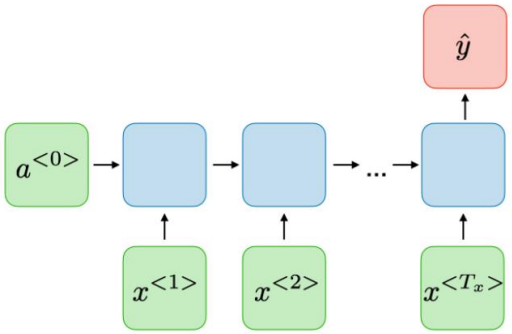
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification

Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Applications of RNN

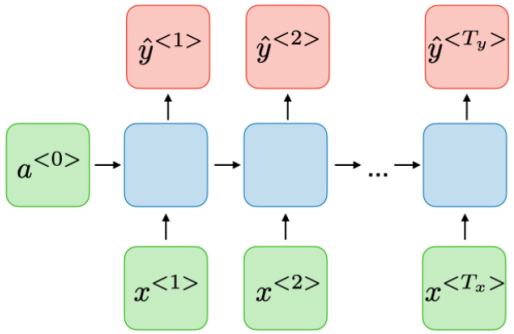
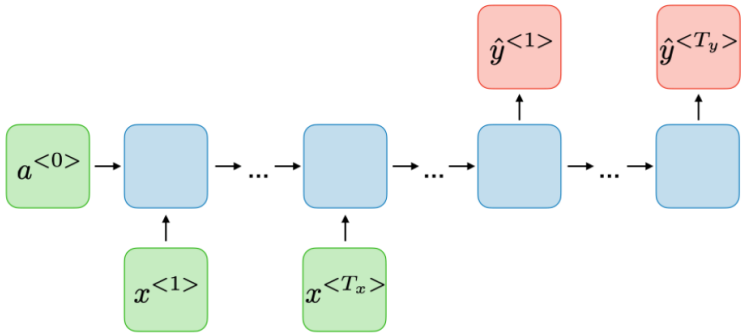
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Convolutional Neural Networks

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Basics of CNN

Convolutional Neural networks (CNN) are a class of neural networks that uses **convolution** to analyse data with **spatial structures/arrangement/neighborhood**

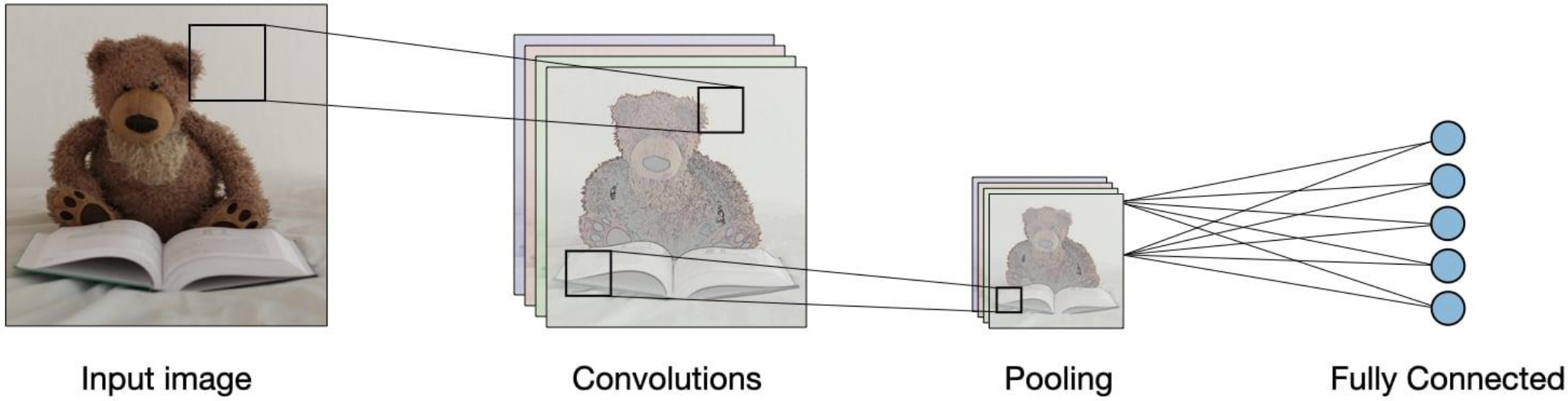


Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Basics of CNN: Convolution Layer

The convolution layer (CONV) uses filters that perform convolution (in fact, correlation) operations as it is scanning the input with respect to its dimensions.

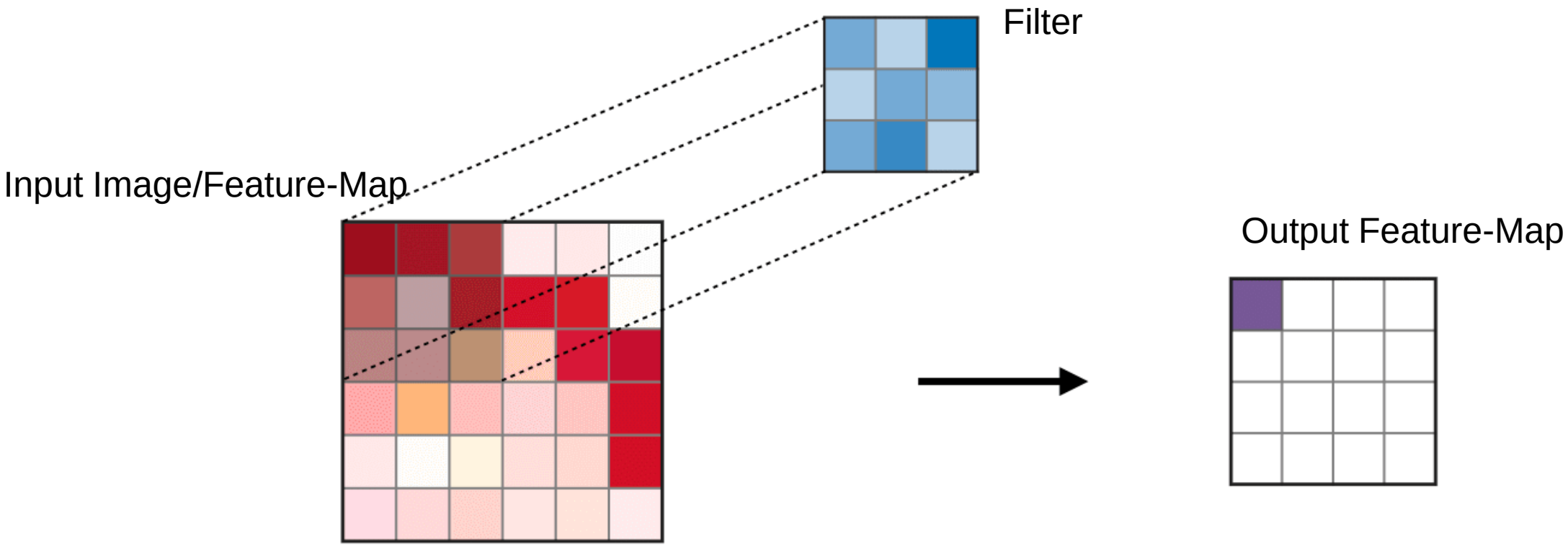
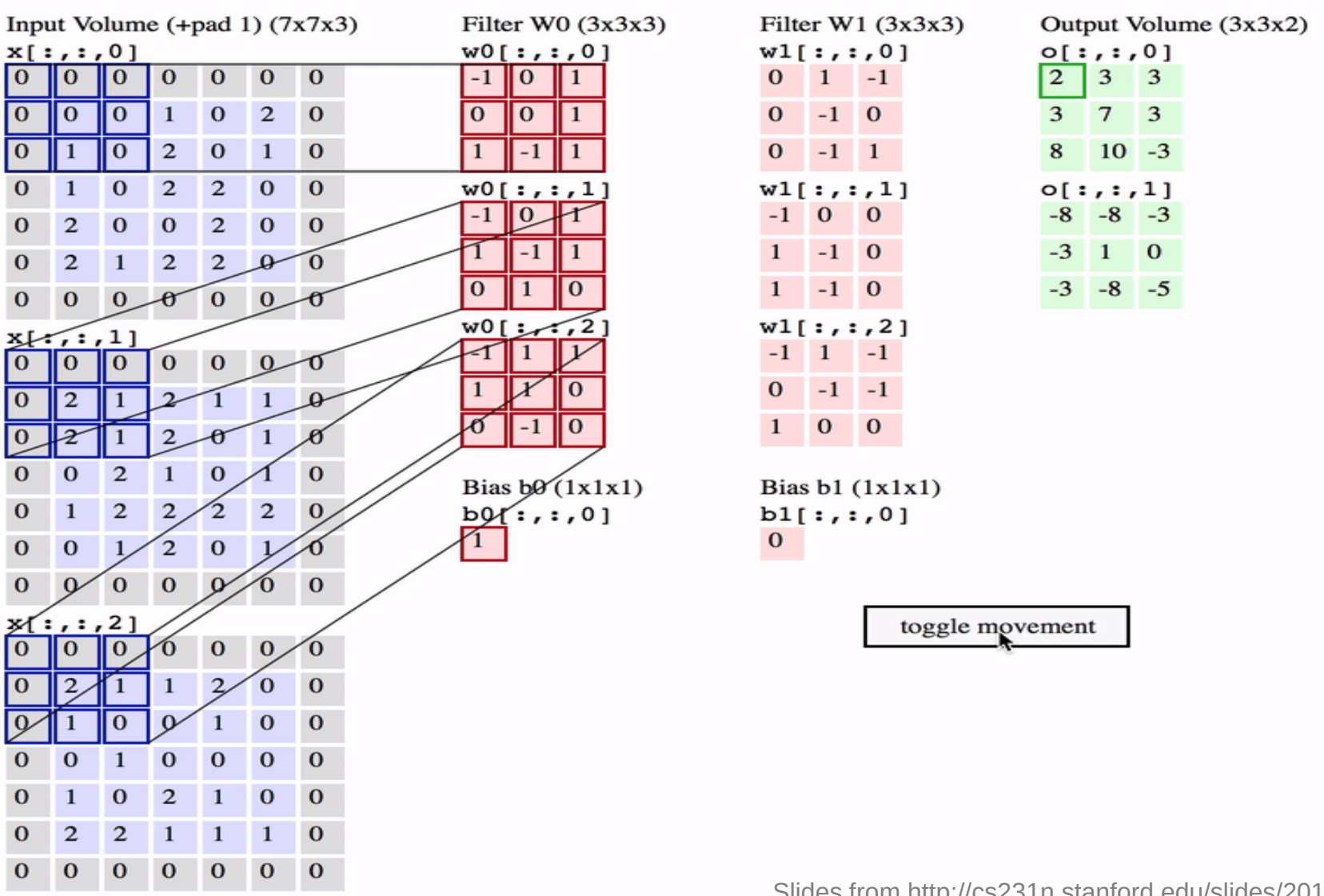


Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

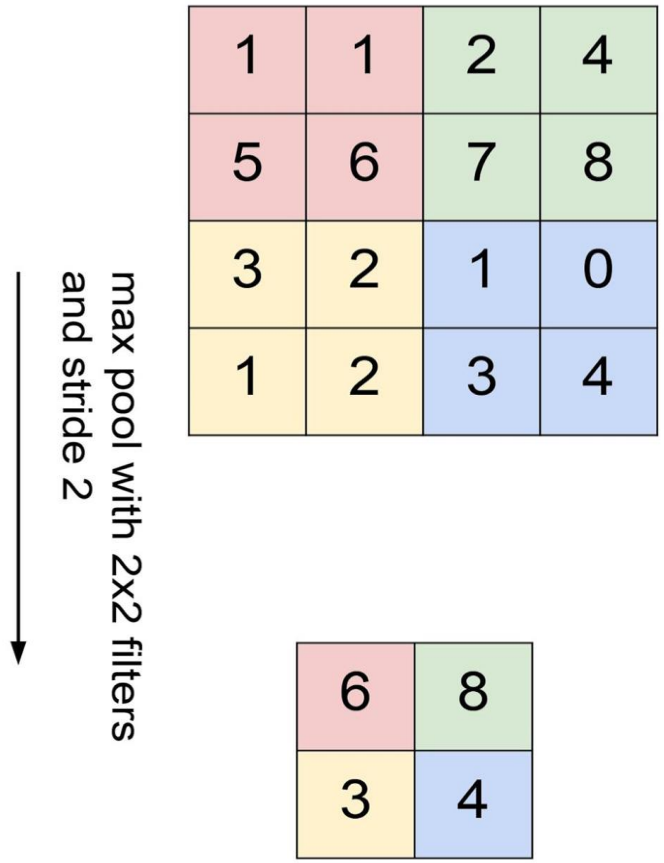
Convolution Operation Animation



Slides from http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Basics of CNN: Pooling Layer

The pooling layer (POOL) is a **downsampling** operation, typically applied after a convolution layer, which does some spatial invariance.



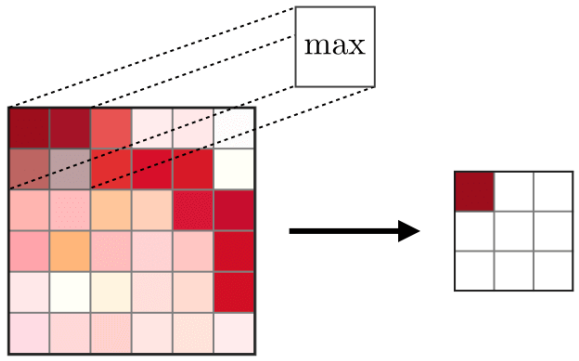
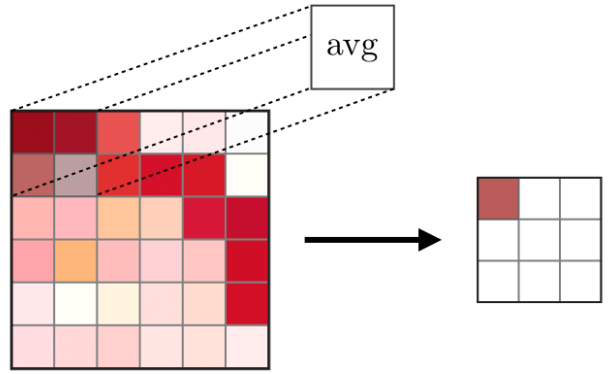
Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	• Preserves detected features • Most commonly used	• Downsamples feature map • Used in LeNet

Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Basics of CNN: Fully Connected Layer

The fully connected layer (FC) operates on **a flattened input** where each input is connected to all neurons.

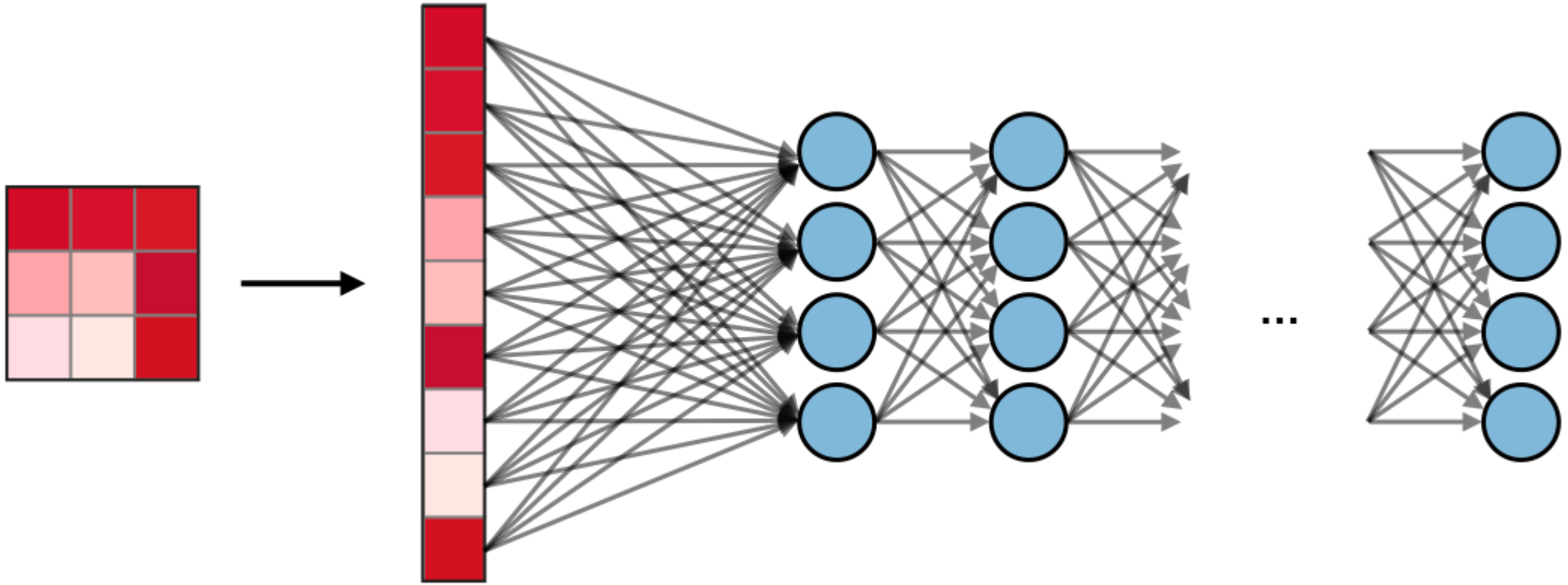


Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Basics of CNN: Relationship between Input, Output, & Parameters

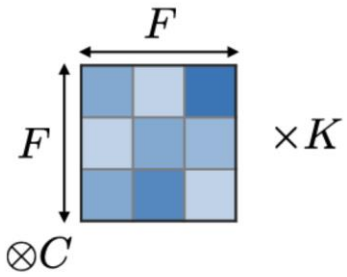
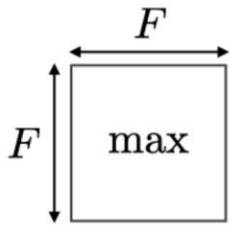
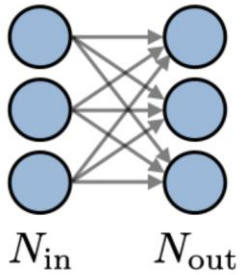
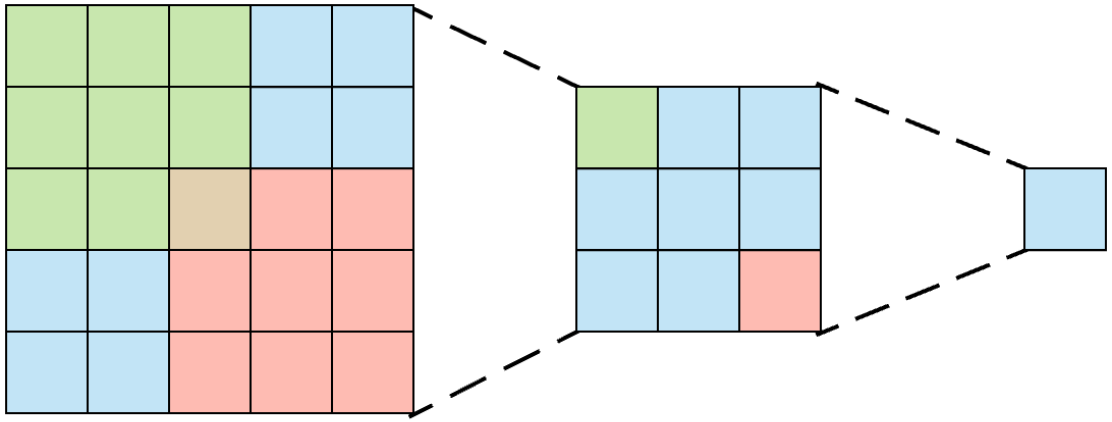
	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks	• One bias parameter per filter • In most cases, $S < F$ • A common choice for K is $2C$	• Pooling operation done channel-wise • In most cases, $S = F$	• Input is flattened • One bias parameter per neuron • The number of FC neurons is free of structural constraints

Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

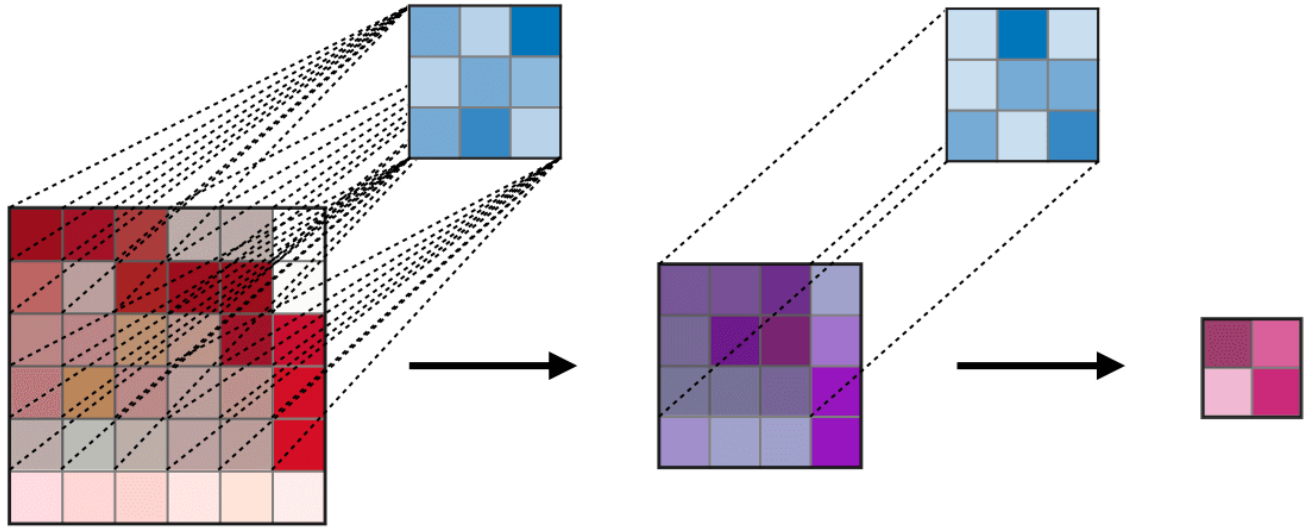
Basics of CNN: Receptive Field

- The **receptive field** in Convolutional Neural Networks (**CNN**) is the region of the input space that affects a particular unit of the network



5x5 receptive field

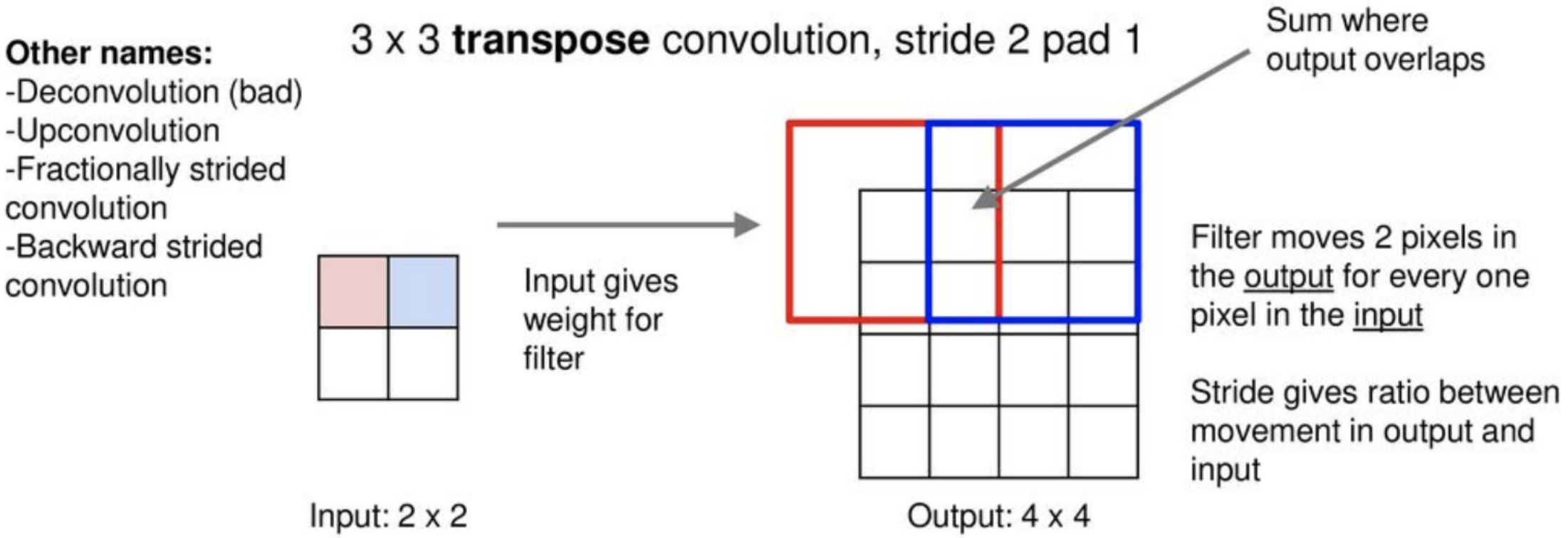
3x3 receptive field



Slides from http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

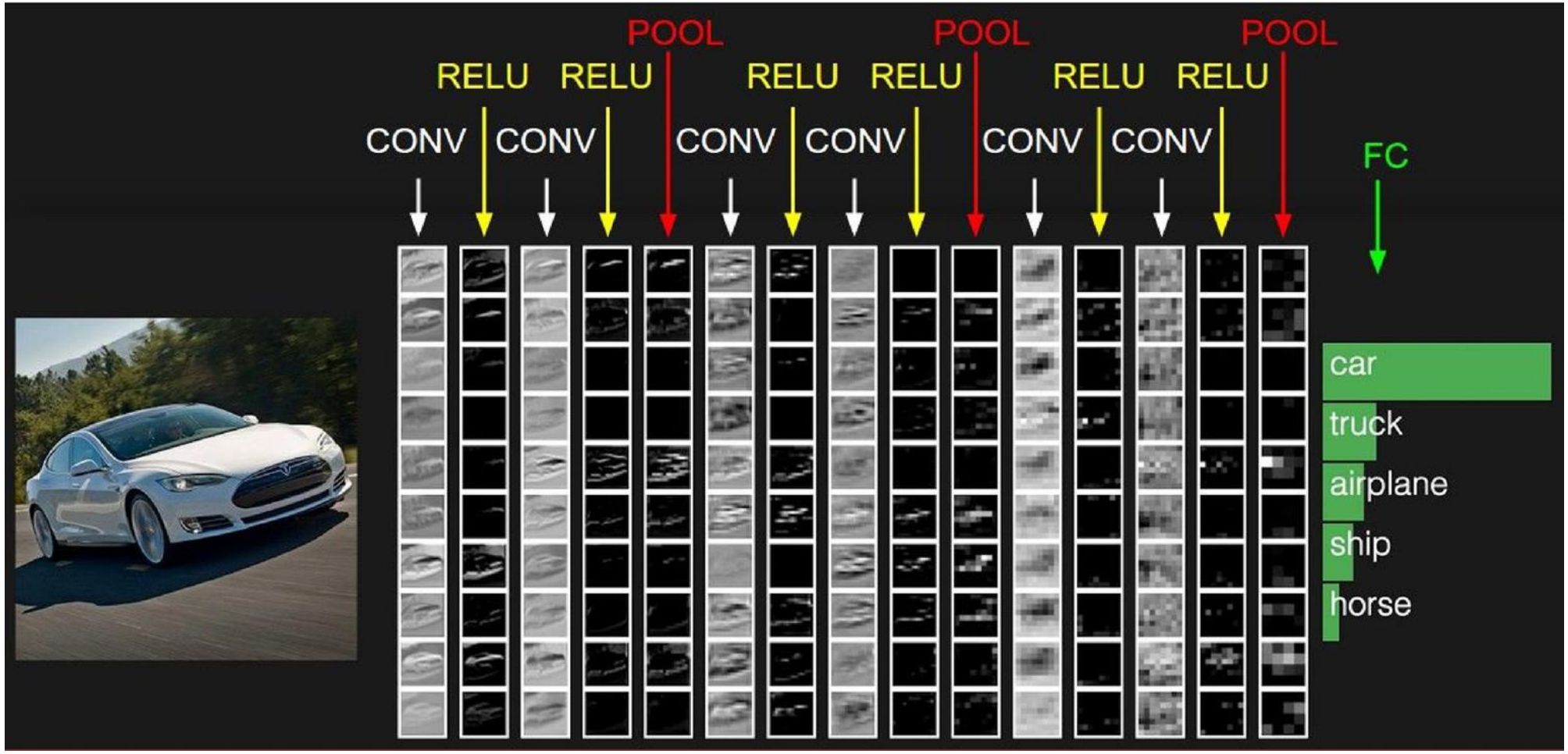
Basics of CNN: Transposed Convolution (Learnable Upsampling)

- Used sometimes in Semantic Segmentation and Image Generation
- **Can be replaced** by *Bilinear Upsampling + Regular Convolution*



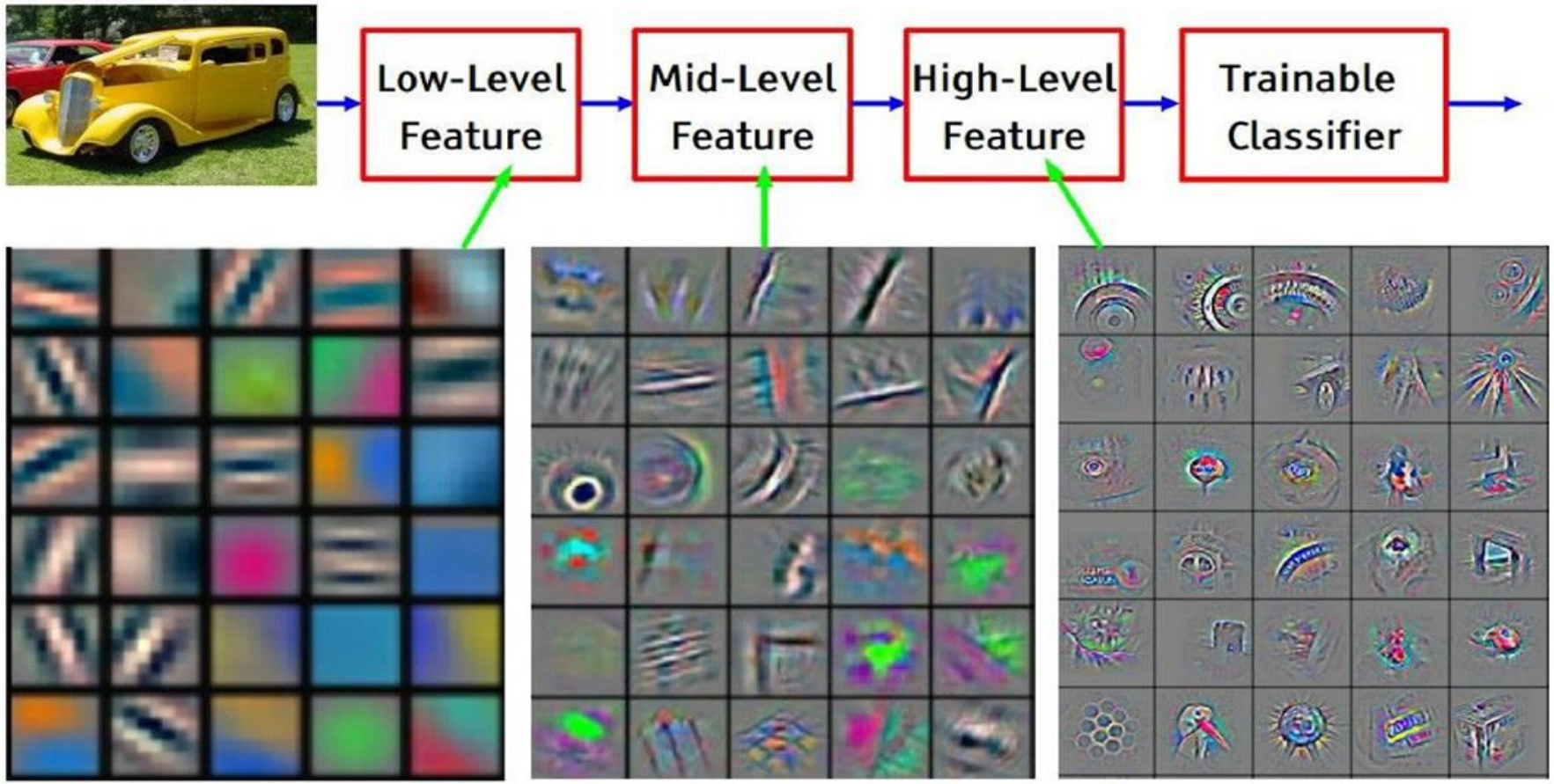
Slides from http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

CNN – All layers



Slides from http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slides from http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Residual Networks

One of the most popular CNN

Is Learning Better Networks as Easy as Stacking More Layers?

1. There is a common trend in the research community that network architecture needs to go **deeper**.
2. Simply **stacking more layers** leads to the problem of **vanishing gradients**, which hamper convergence from the beginning.
3. Meanwhile, **degradation** problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly.

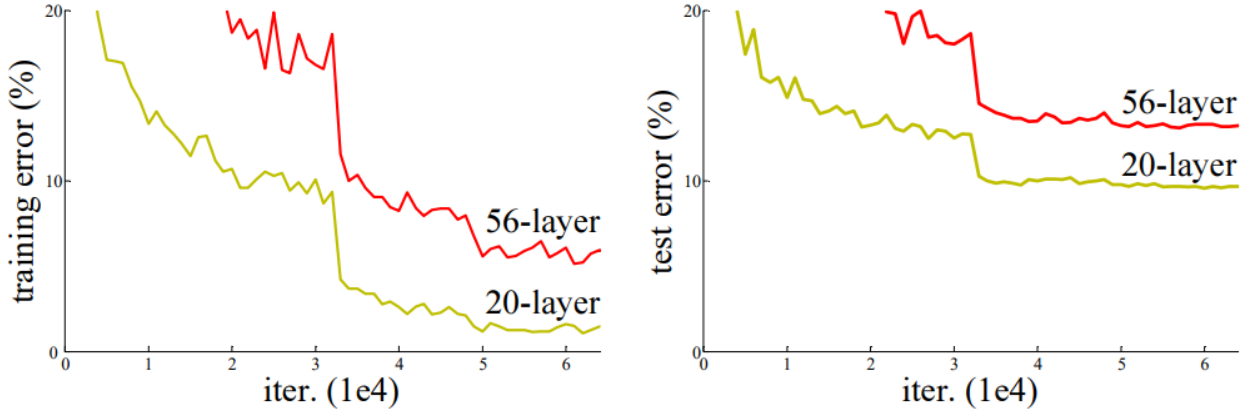


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Residual Learning

Residual learning block is formulated as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \qquad (1)$$

Here x and y are the input and output vectors of the layers considered. The function $\mathcal{F}(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned. For the example in Fig. 2 that has two layers, $\mathcal{F} = W_2\sigma(W_1\mathbf{x})$ in which σ denotes ReLU

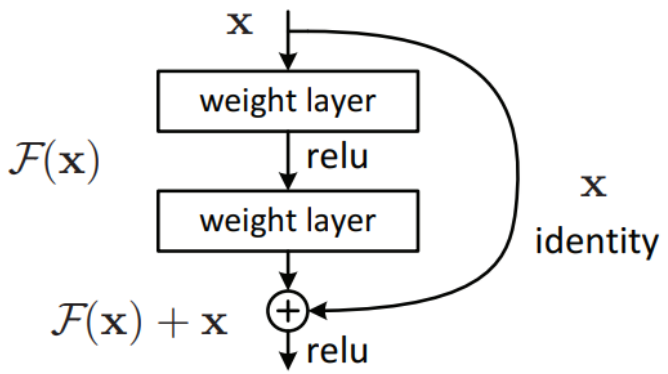


Figure 2. Residual learning: a building block.

Network architecture

Two types of architecture are tested: plain network similar as VGG-19 and residual network.

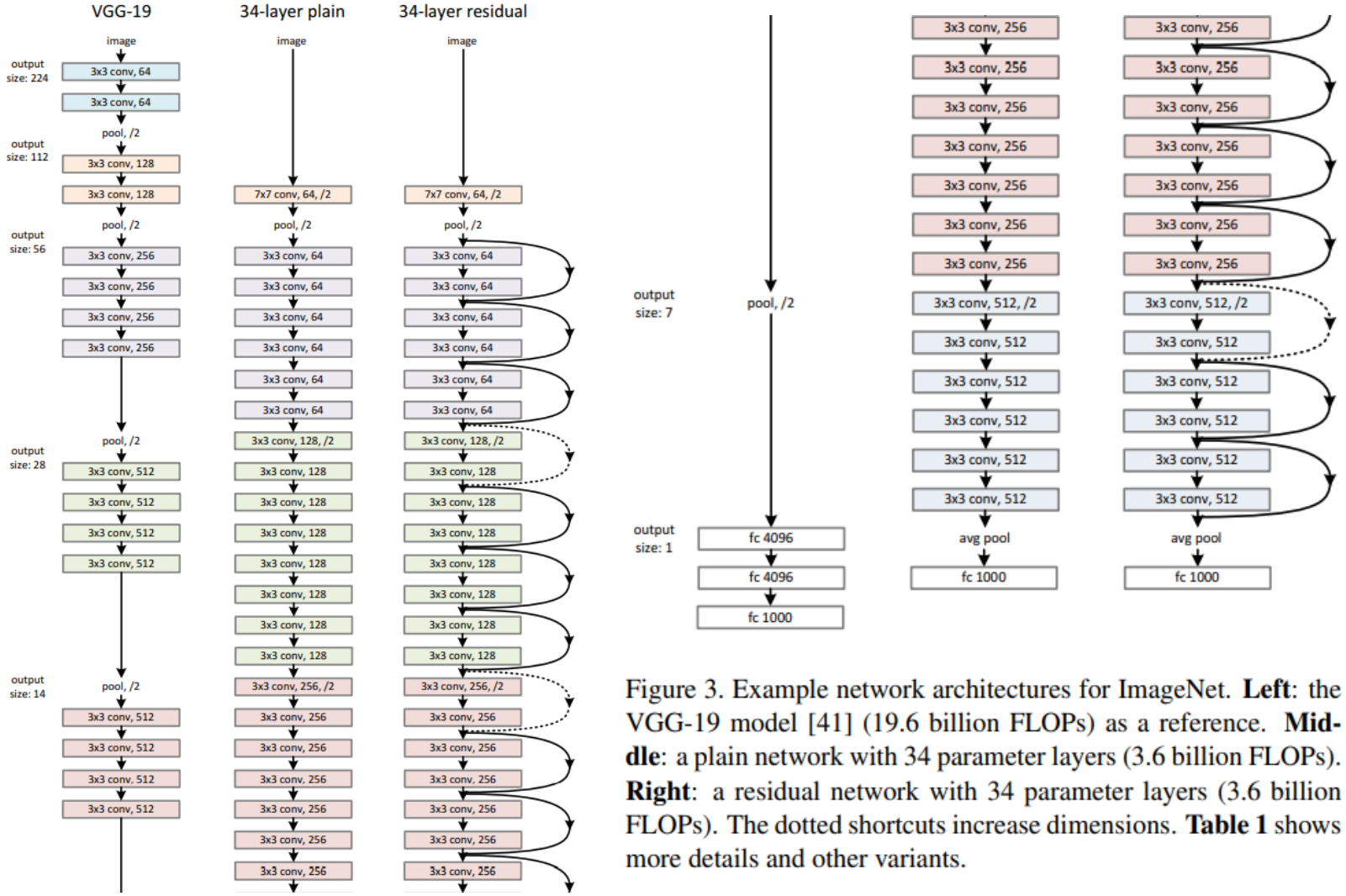


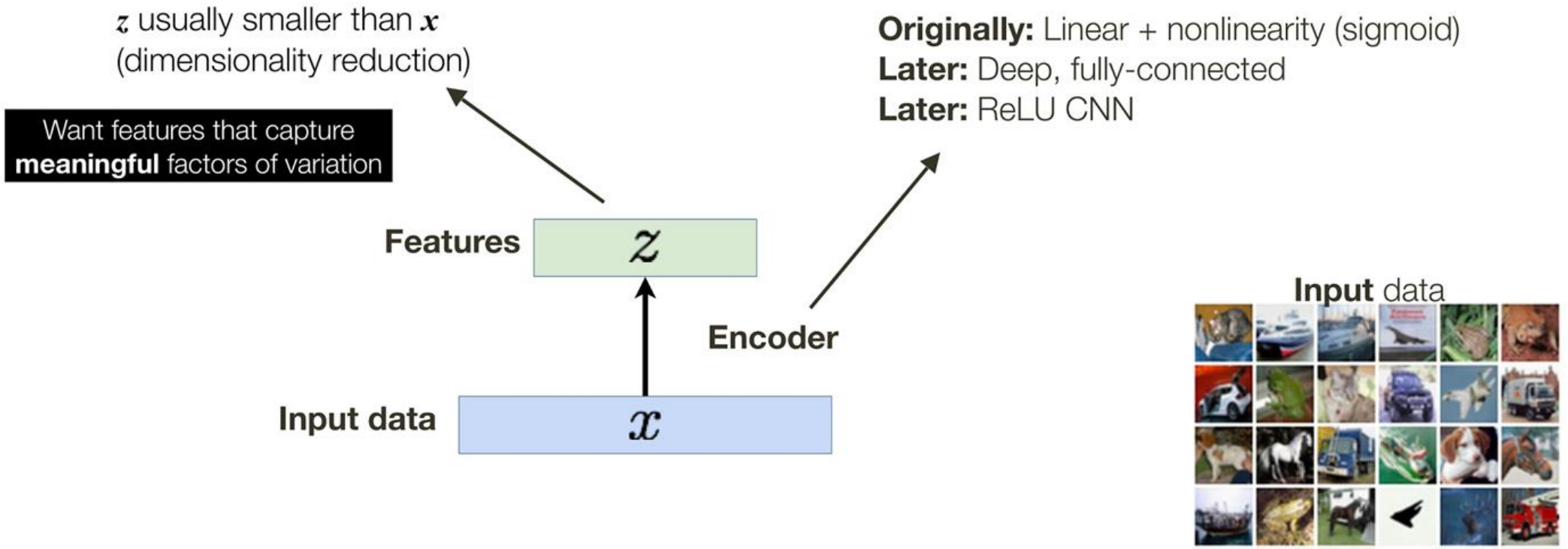
Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Generative Models

Variational Autoencoders (VAE)

Autoencoders (AE)

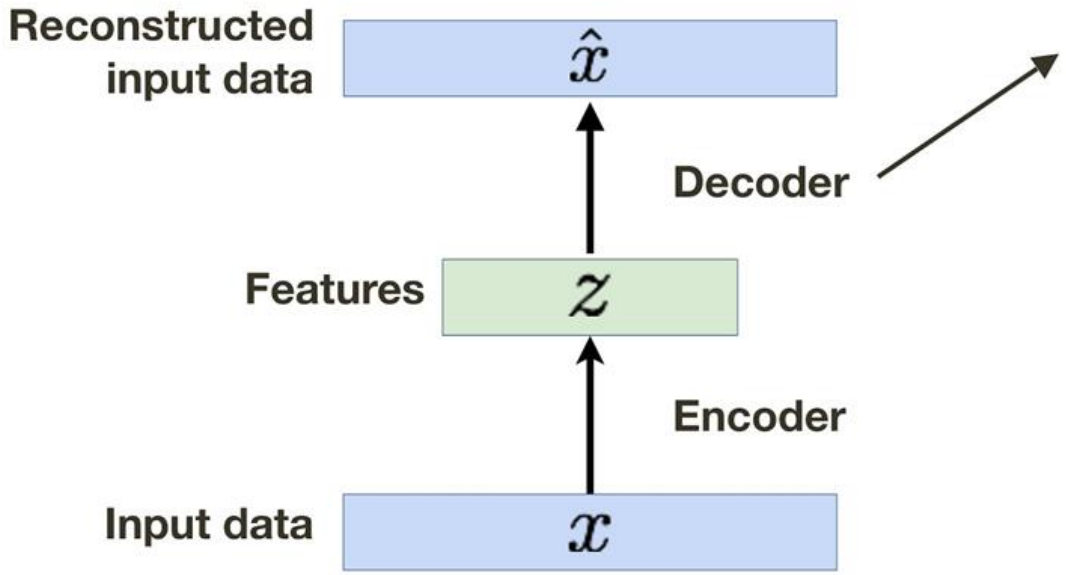
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



* slide from Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

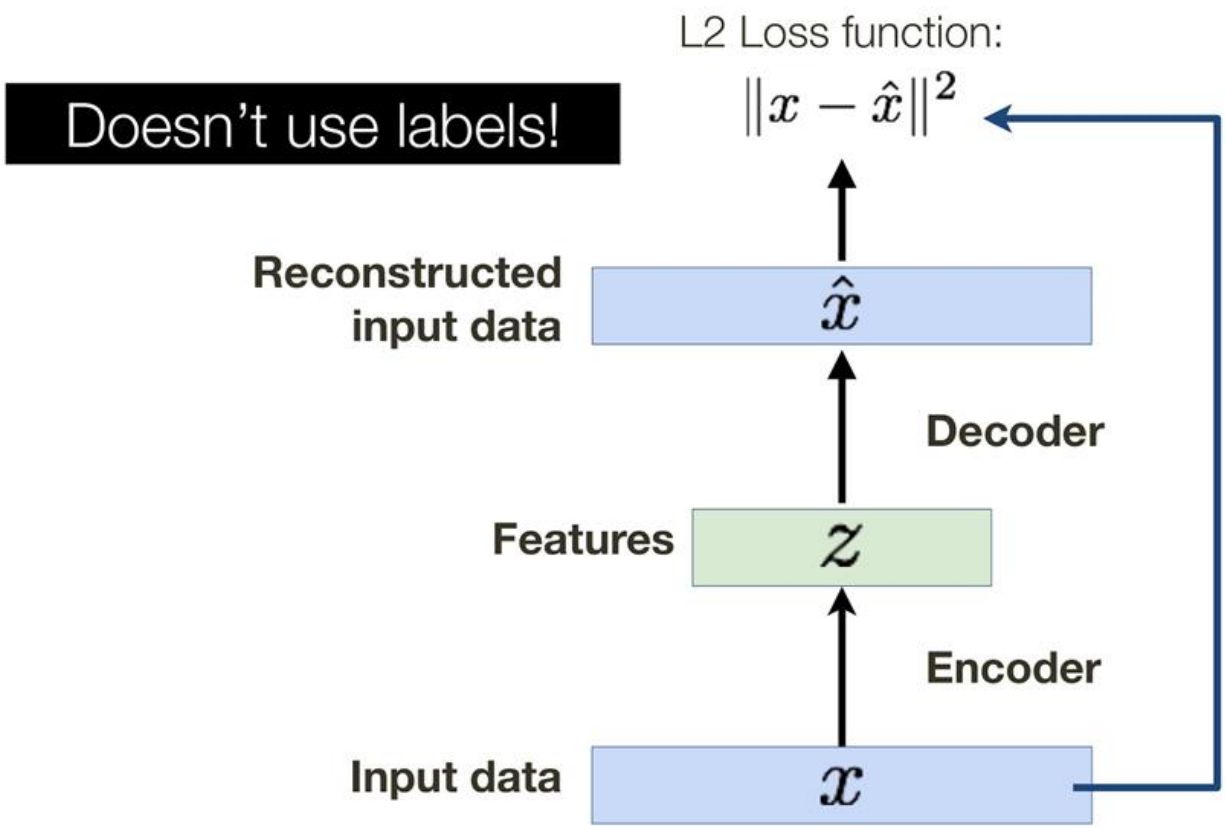
Autoencoders (AE)

Train such that features can reconstruct original data best they can



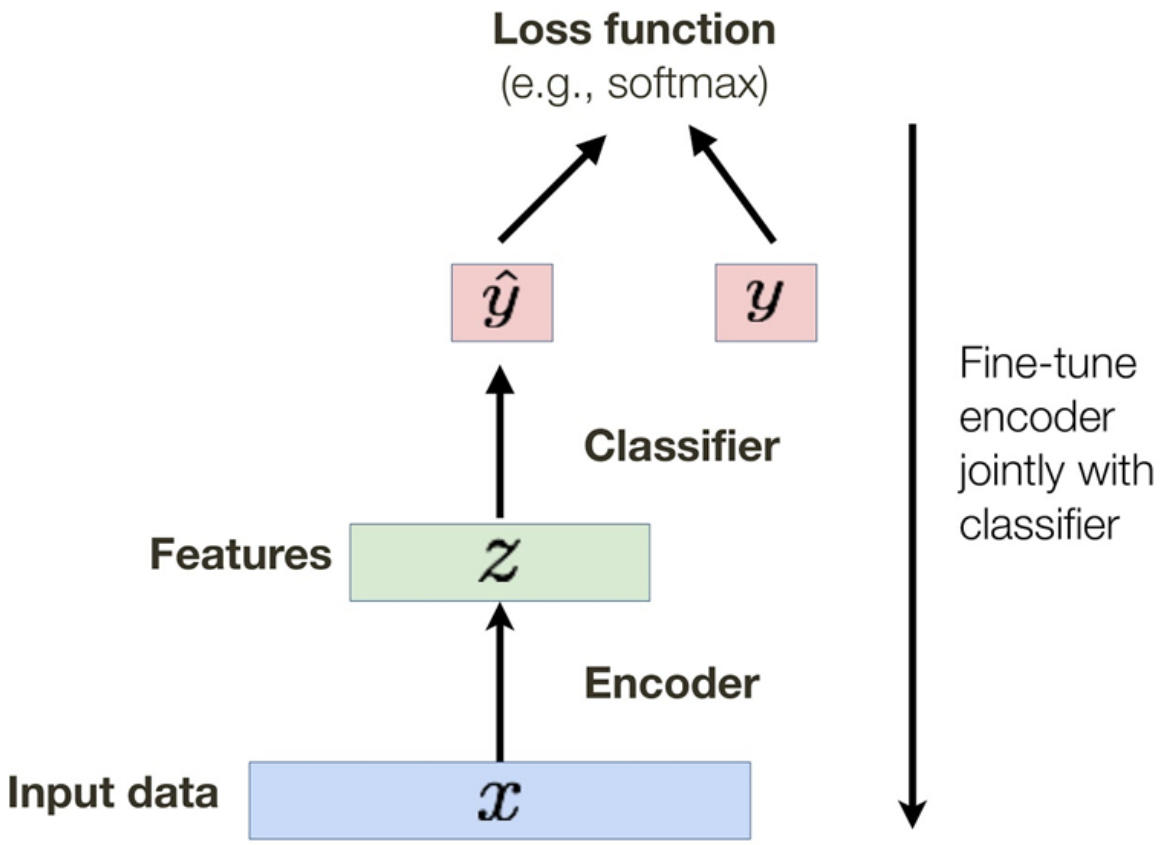
* slide from Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Autoencoders (AE)



* slide from Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Autoencoders (AE)



Train for **final task**
(sometimes with small data)

bird plane
dog deer truck

* slide from Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Variational Autoencoders (VAE)

Probabilistic spin on autoencoder - will let us sample from the model to generate

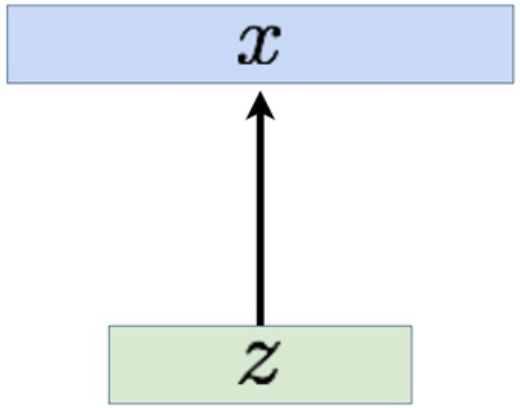
Assume training data is generated from underlying unobserved (latent) representation z

Sample from true **conditional**

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true **prior**

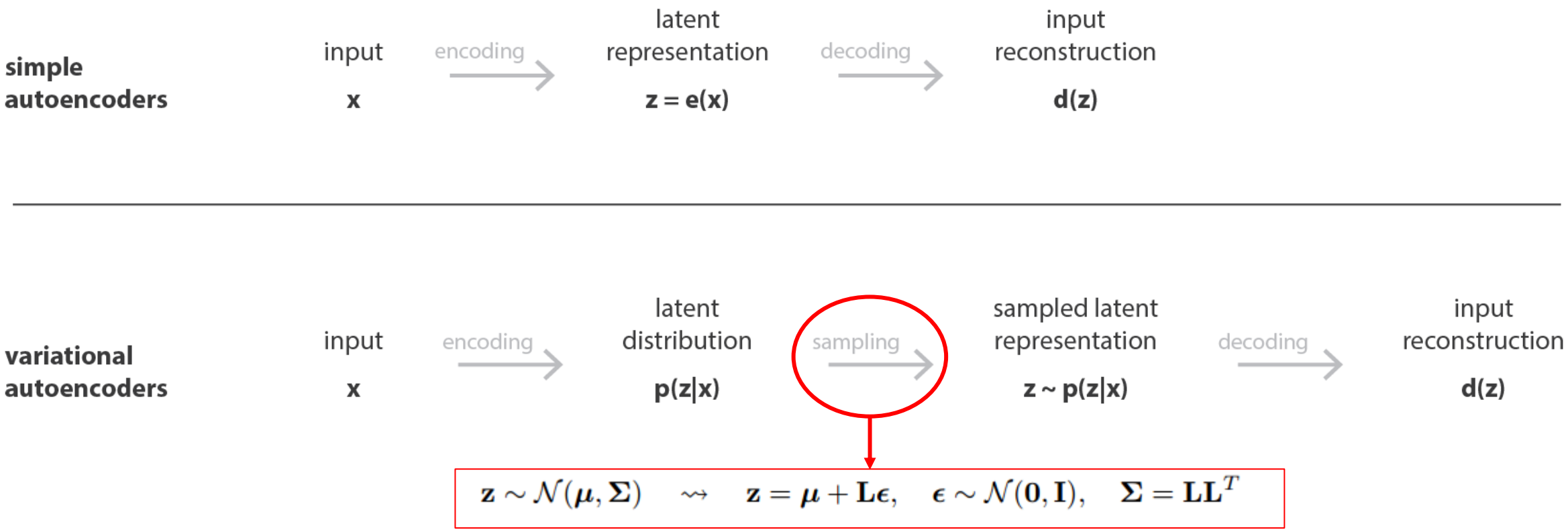
$p_{\theta^*}(z)$



Intuition: x is an image, z is latent factors used to generate x (e.g., attributes, orientation, *etc.*)

* slide from Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

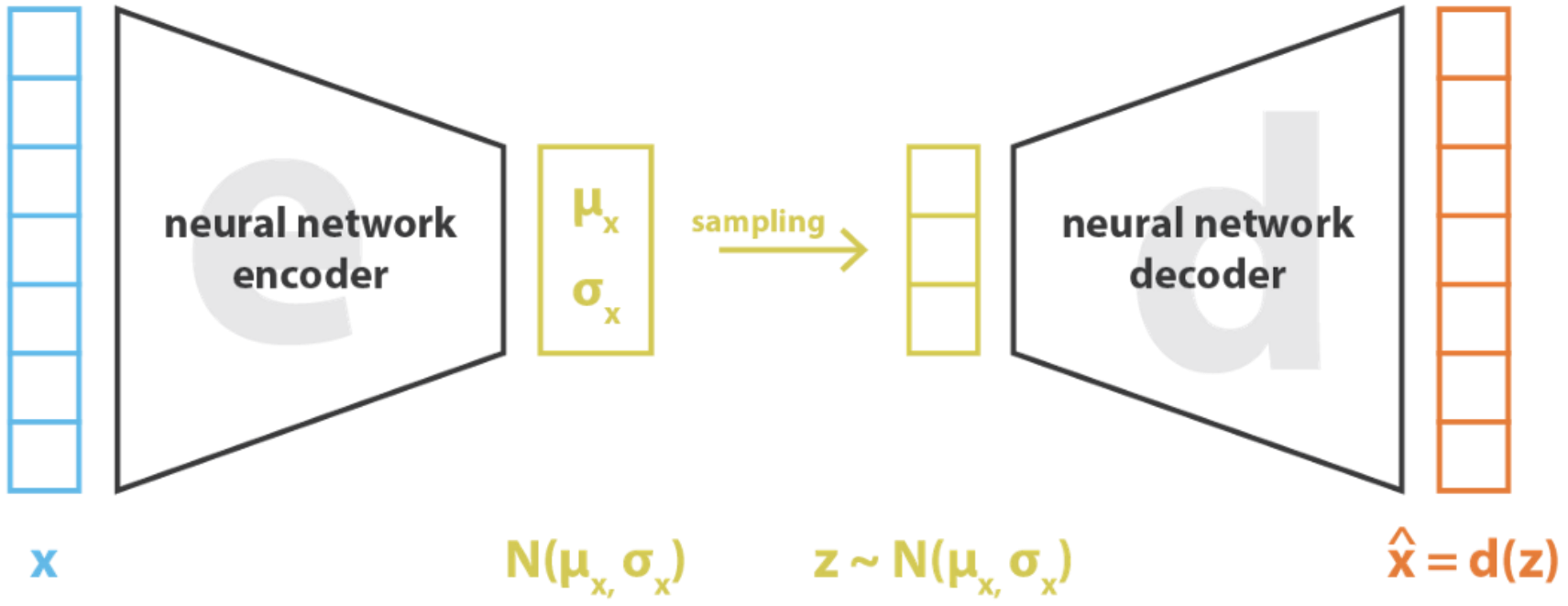
Variational Autoencoders: Probabilistically Reparameterized AE



[Reparameterization trick: sampling from multidimensional Gaussian]

Image modified from: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Variational Autoencoders – Probabilistic AE + KL Regularization



$$\text{loss} = ||x - \hat{x}||^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = ||x - d(z)||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

Image from: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Variational Autoencoders – With vs Without KL Regularization

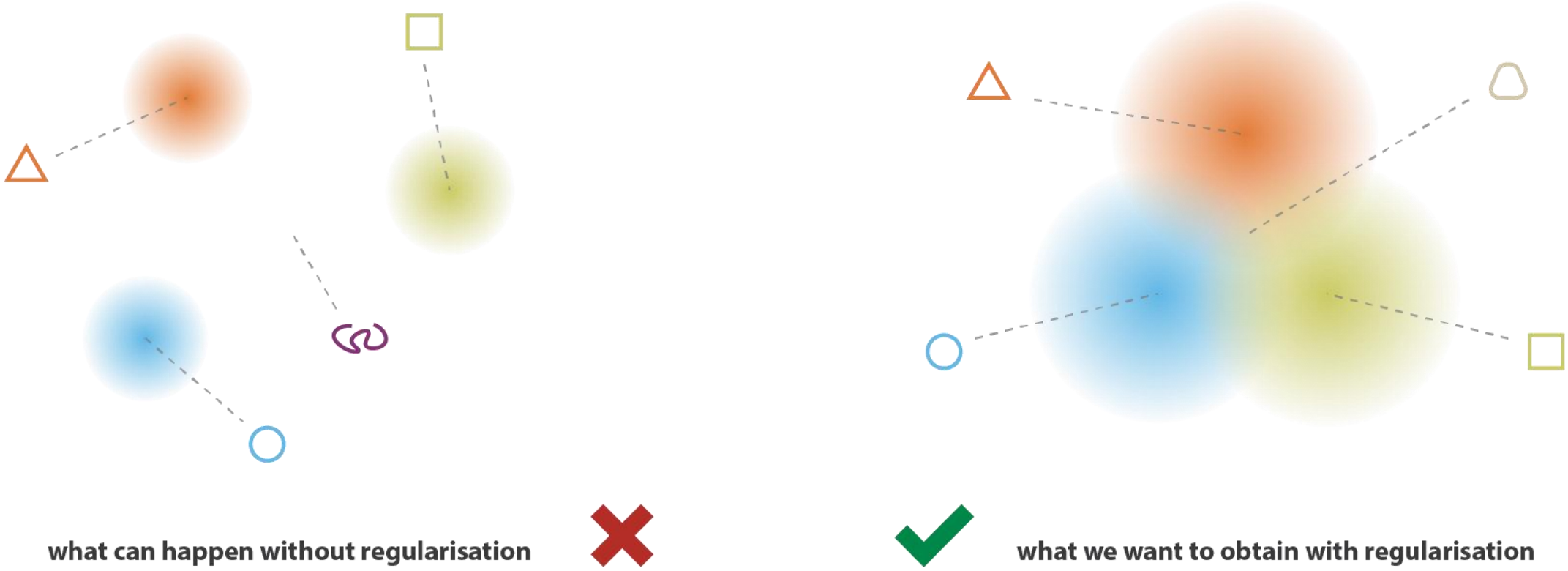


Image from: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Variational Autoencoders – Computation Graph

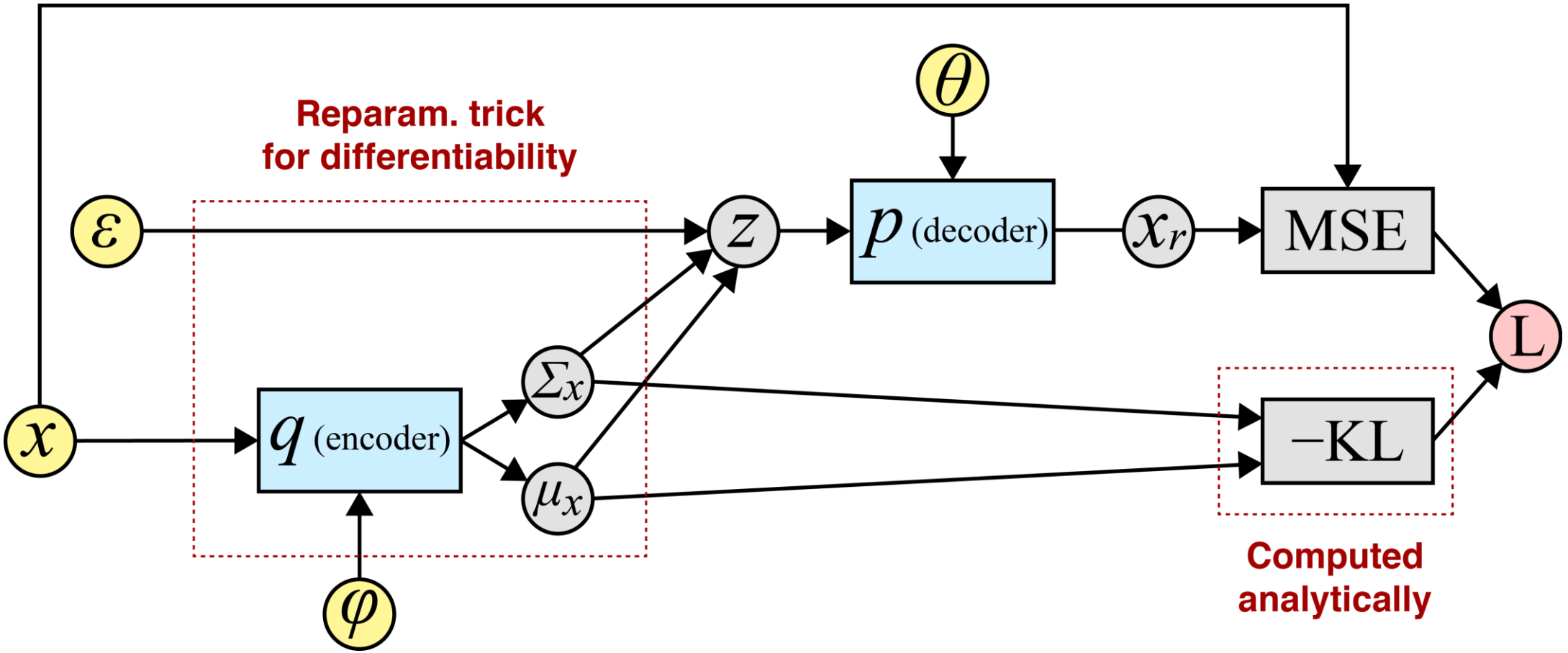
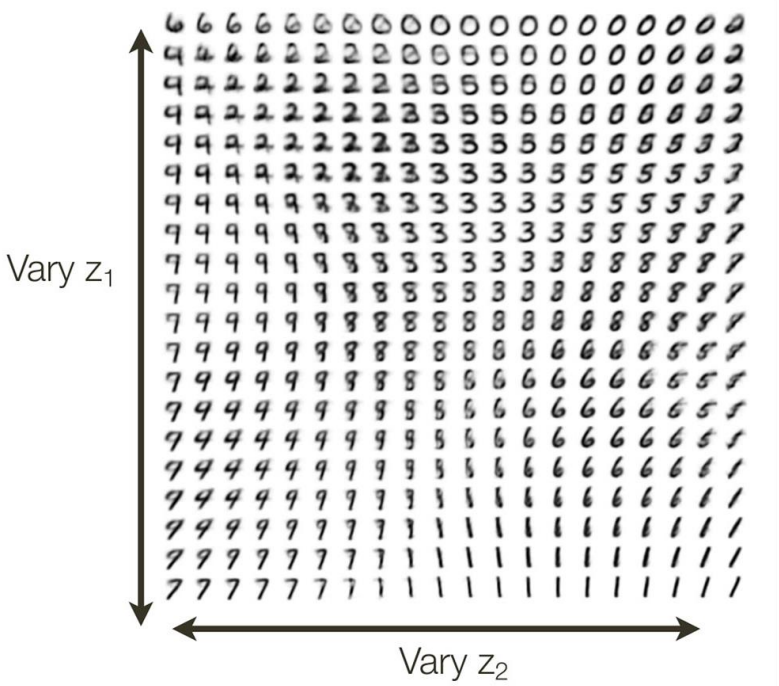
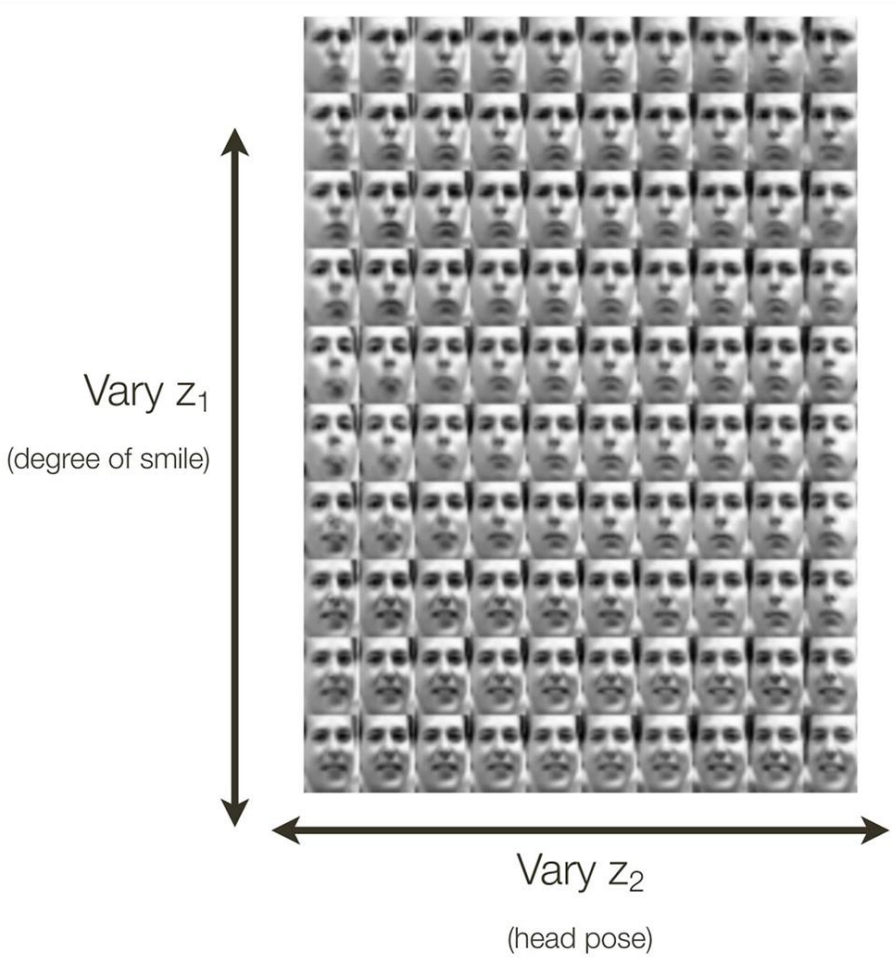


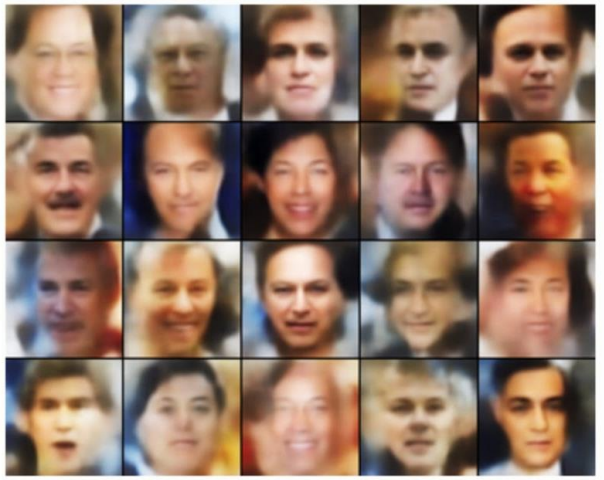
Image from: https://gregorygundersen.com/blog/2018/04/29/reparameterization/

VAE – Data Generation Examples





32x32 CIFAR-10



Labeled Faces in the Wild

How To Train Deep Networks Successfully in Practice?

- Data Augmentation
- Weight Initialization
- Learning Rate Tuning
- Other Common Practices

Data Augmentation

- Deep learning models usually need a lot of data to be properly trained.
- It is often useful to **get more data from the existing ones**.
- Usually augmented on the fly during training.

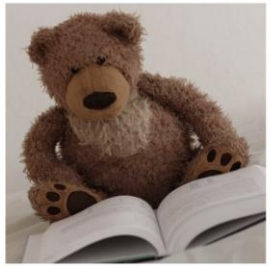
Original	Flip	Rotation	Random crop	Color shift	Noise addition	Information loss	Contrast change
							
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row	• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Weight Initialization

- Train from scratch
  - Randomly initialize weights
  - Different random initialization
    - Xavier
    - Kaiming
- Transfer Learning
  - Training need lots of data and time.
  - Leverage pre-trained weights on huge datasets towards your own use case.

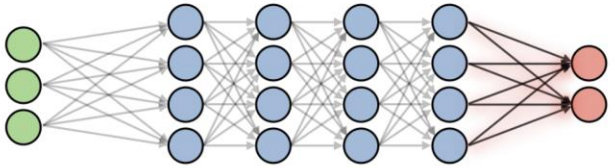
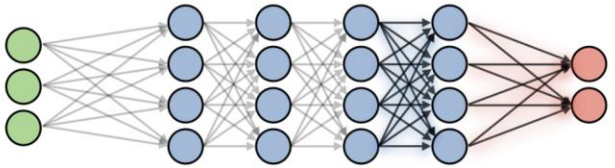
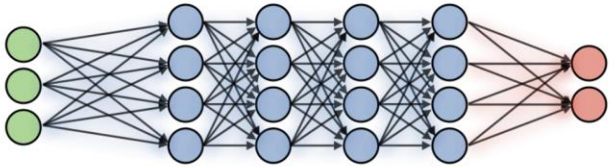
Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Learning Rate Tuning

- The learning rate is the most important hyperparameter in deep learning.
  - If bound by time, focus on tuning it.
- Tuning by monitoring the loss curve that plot the objective function over time.
  - Start with a small learning rate
  - Increase it until loss starts to diverge/increase/NaN
- Tune it on a validation set!!!
  - Commonly split all your data into train-validation-test by ratio: 8:1:1
- Grid search if you have lots of GPU
- Schedule your learning rate, or
- …use Adam instead of SGD solver

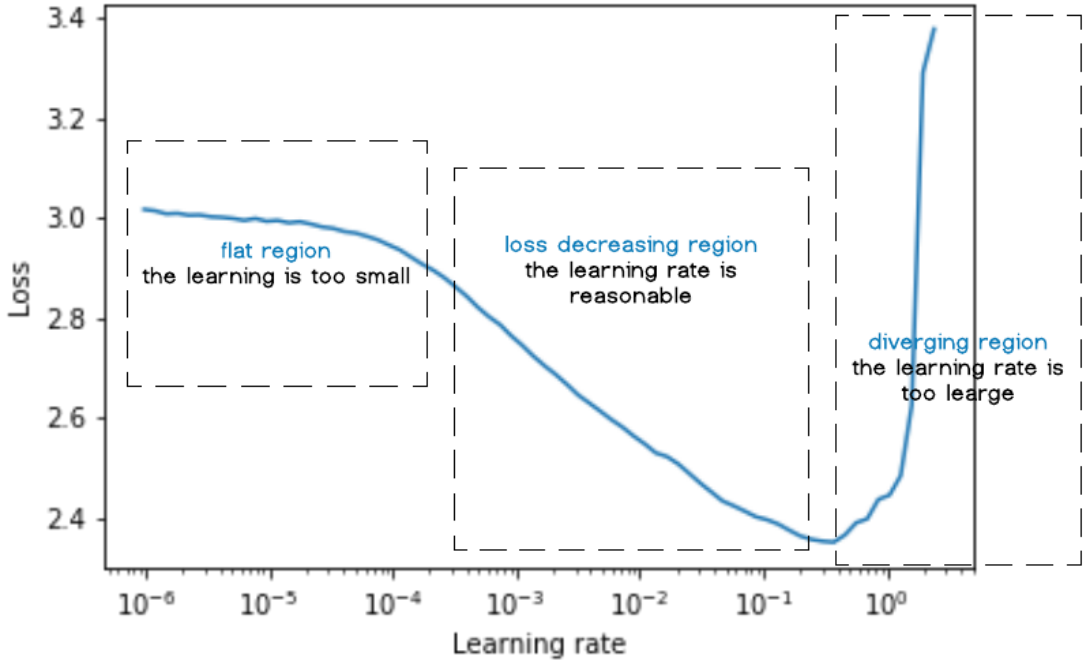
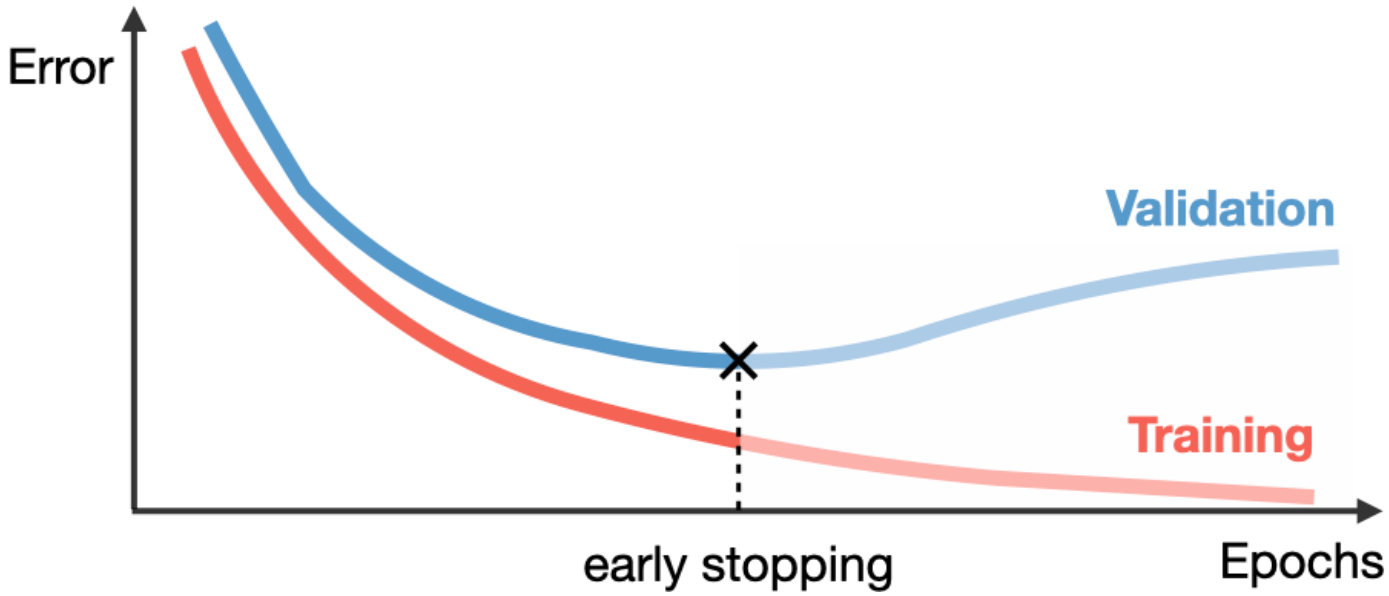


Image from https://medium.com/deep-learning-hk/some-techniques-in-deep-learning-optimization-1-learning-rate-b4669d5bb568

Other Common Practices

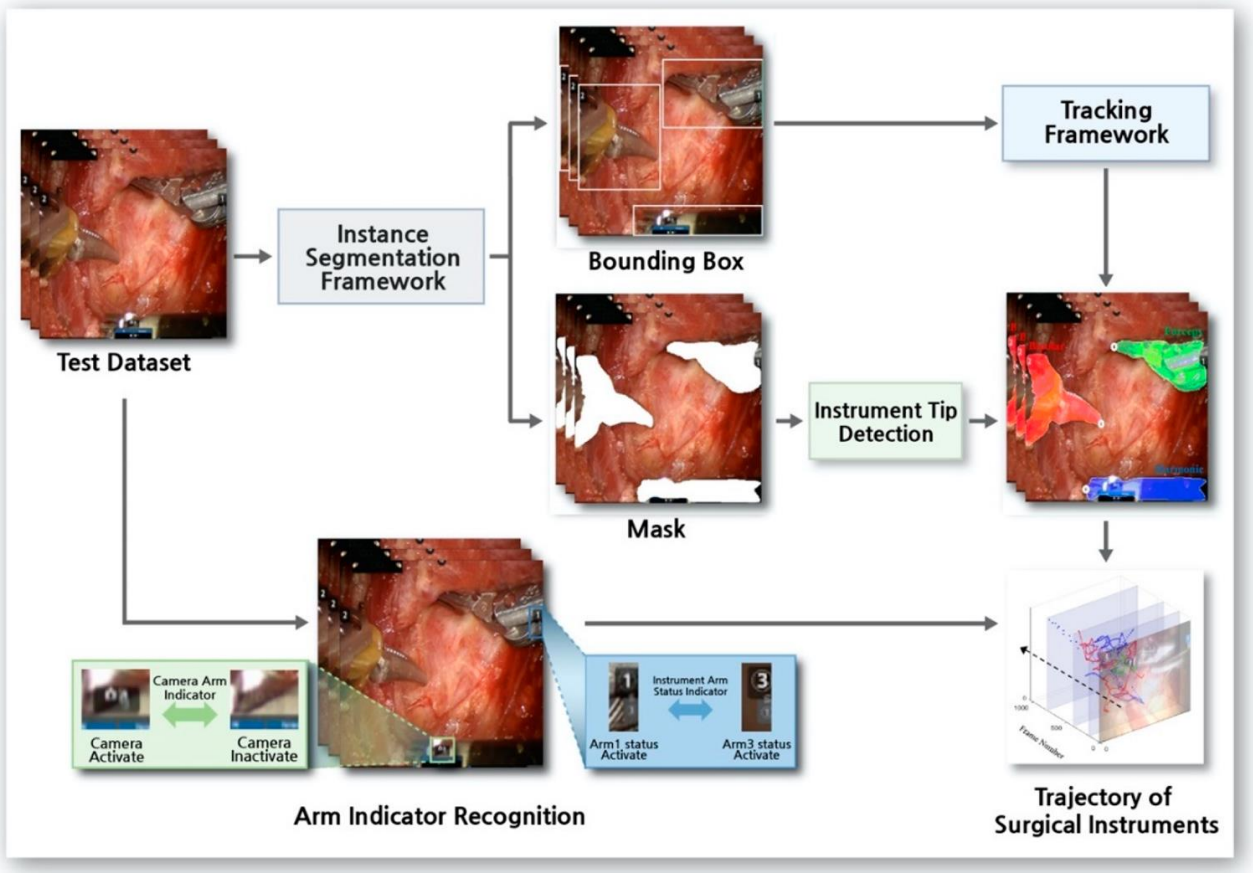


- Early Stopping
- Check gradient
  - Magnitude too small/large?
- Overfitting a small batch first
- Add regularization if
  - Overfitting: training loss is small, but validation loss is large
  - Dropout, L2 regularization of weights
- Normalization!
  - Normalize your data
  - Use normalization layers like BatchNorm/LayerNorm/etc.

Image from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

When Do Deep Learning Methods Face Challenges?

1. When it's **hard to collect data** for the task - simulated or real.
2. When it's hard to obtain **reliable annotation** and self-supervision is not an option.



Example

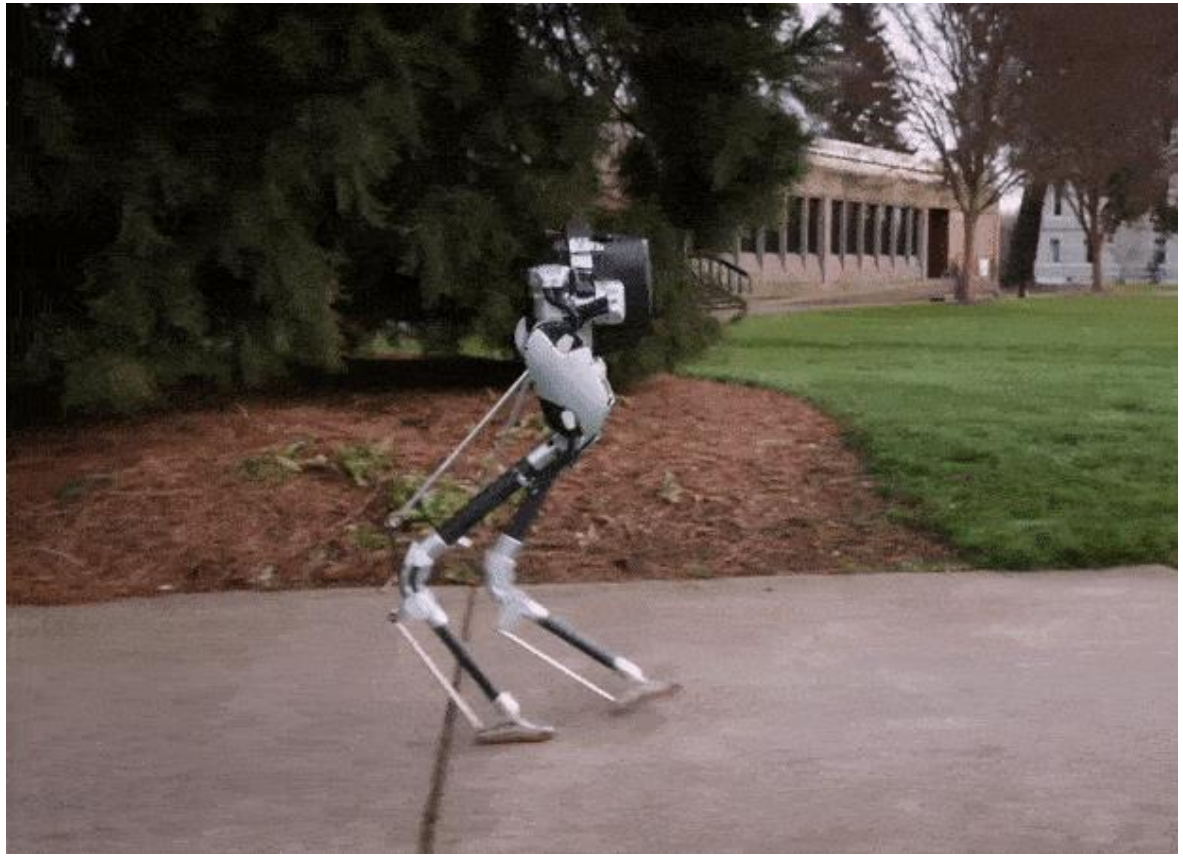
- Robots for surgical assistance
- Pharmaceutical studies - e.g. Simulation of biochemical reactions of new drugs with unknown activation mechanisms

Above example use cases are NOT impossible to tackle via Deep Learning - just that there may be simpler or better ways to approach them in the absence of a proper large-scale dataset.

Learning-Based Multiple Surgical Instrument Tracking in Training and Actual Operations, Donghae, Dongheon Lee et. al., 2020

When Do Deep Learning Methods Face Challenges?

3. When the function to be optimized is **NOT differentiable**



Example

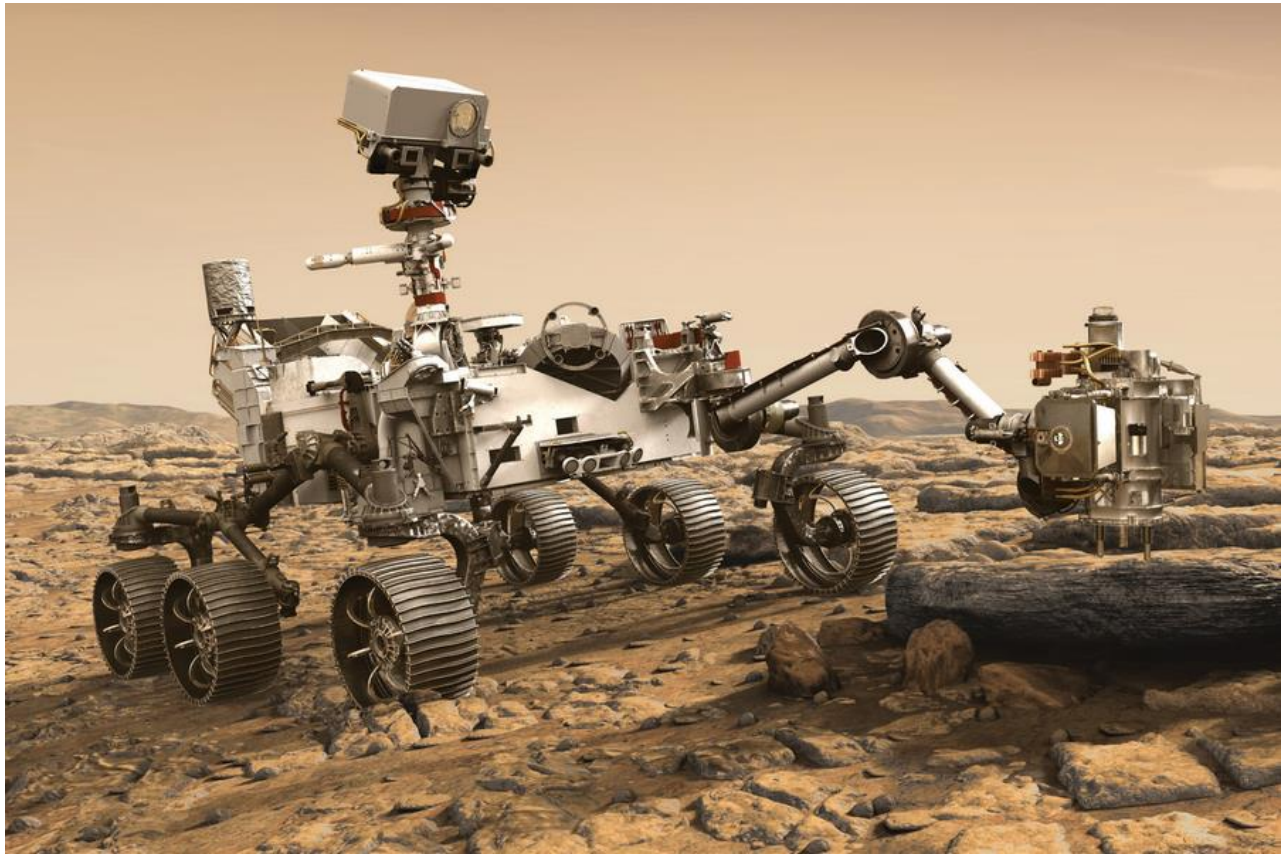
- Legged robots locomotion
- Game playing - e.g. Atari, Go, etc

"Typically" problems that are tackled via RL usually have optimization functions that are hard to define in a differentiable manner.

Cassie, Agility Robotics, 2017 - https://www.youtube.com/watch?v=Is4JZqhAy-M

When Do Deep Learning Methods Face Challenges?

4. When the data is of **lower dimension and complexity** - e.g. in tasks where simpler algorithms suffice.
5. When **computational constraint** is the main bottleneck - e.g. on a Raspberry Pi.



Example

- Edge detection for non-critical processes (e.g. Canny detector may suffice)
- Certain vision task on the Mars rover (where onboard compute and power is limited)

Figure from https://news.mit.edu/2021/perseverance-rover-moxie-0217

Next Week

+ AprilTags

* Homography Estimation

++ Ax=b, Ax=0

* Camera Calibration, Zhang's method

+ DLT

+ Vanishing Points & Lines

*: know how to code
++: know how to derive
+: know the concept

References for Next Week

- HZ2003:
  - Section 2.3, 4.1, 4.4, 7.1, 7.2, 7.4, 8.6
- FP2011:
  - Section 1.2, 1.3, 12.1
- Sz2022:
  - Section 11.1, 11.4.5
- Co2011:
  - Section 11.2, 11.1
- Linear algebra:
  - Sz2011: section A.1.1, A.1.2, A.1.3, A.2, A.2.1
  - HZ2003: A5.1, A5.2, A5.3