Deep Learning for Big Visual Data

Jenq-Neng Hwang (黃正能), Professor
Department of Electrical & Computer Engineering
University of Washington, Seattle WA
hwang@uw.edu

BUPT Summer Intensive Course, 2024

Goals and Grading

- **Goals:** provide students with significant machine learning and deep learning theoretical foundations and the practical applications for solving real-world visual tasks.
- **Prerequisites by Topics:**
  - Discrete Time Signals and LTI Systems, Probability and Statistics
  - Linear Algebra and Basic Optimization Techniques
- **Grading:** 3 Homework assignments (20%/each), 1 final HW+paper review report (30%), attendance and in-class interaction (10%)
- TA: Haiqing Du

Course Contents

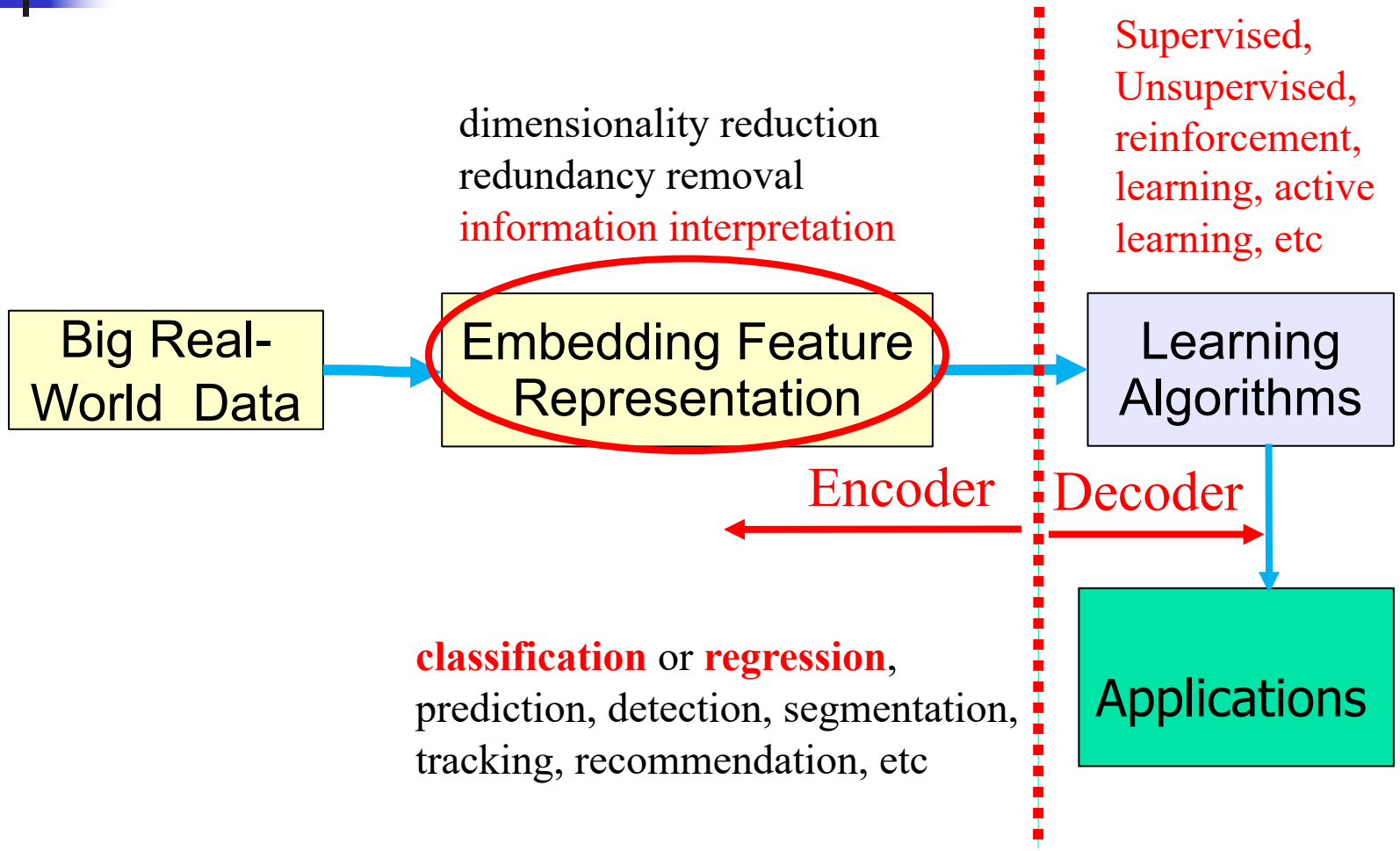
- Introduction to Machine Learning and Deep Learning
- Convolution Neural Networks (CNNs) and Object Classification
- Practical Applications of Deep Learning Object Classification
- Image Object Detection and Video Object Tracking
- Semantic and Instance Segmentation and Human Pose Estimation
- Transformer for Language and Visual Applications
- Generative Adversarial Learning (GANs) for Image Generation
- Diffusion for Image and Video Generation

Introduction to Machine Learning and Deep Learning

Jenq-Neng Hwang (黃正能), Professor
Department of Electrical & Computer Engineering
University of Washington, Seattle WA
hwang@uw.edu

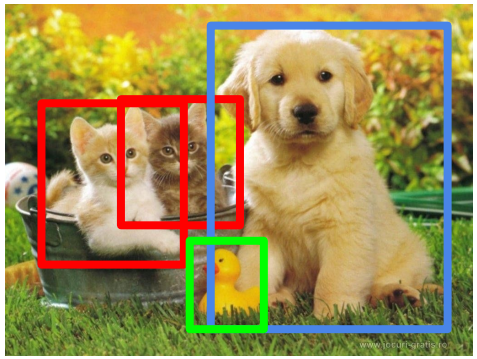
Cameras (image/video) Everywhere

Machine Learning for Big Real-World Data



Visual Data Machine Learning Tasks

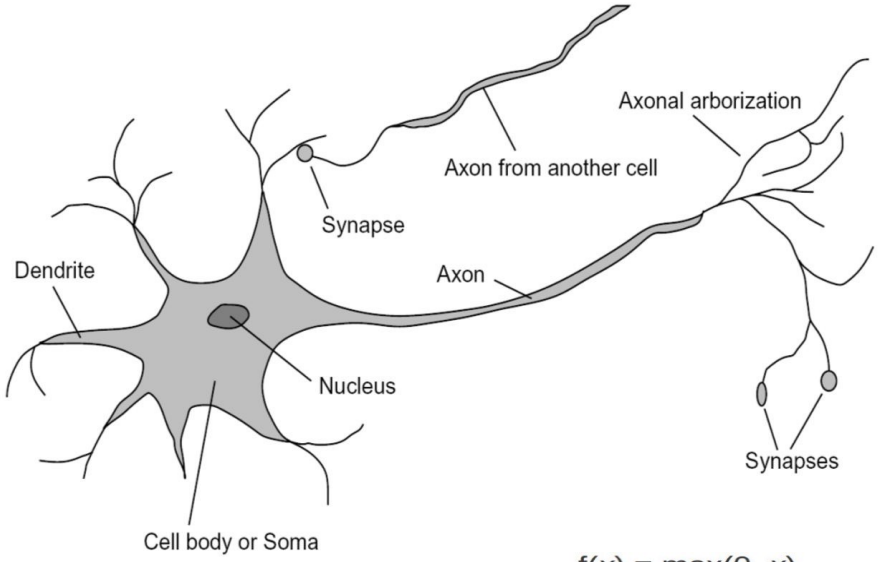
Can also be applied to speech, text, and other sensing data!

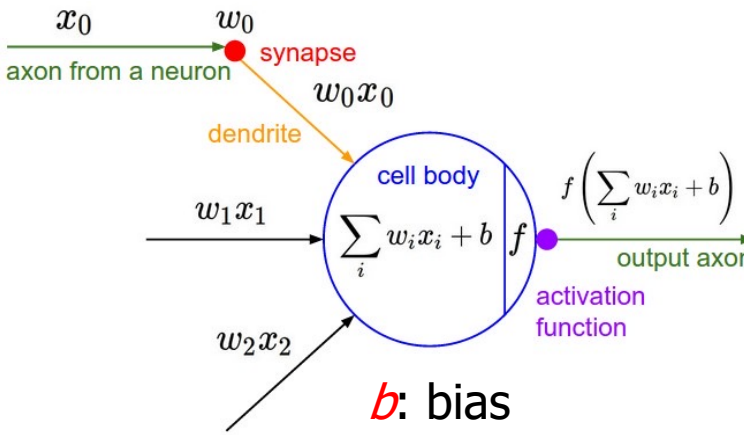
Classification	Classification + Localization	Object Detection	Instance Segmentation
			
CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK
Single object		Multiple objects	

Fei-Fei Li et al., "CS 231 lecture Note," Stanford Univ. 2018

Biological & Artificial Neurons of Human

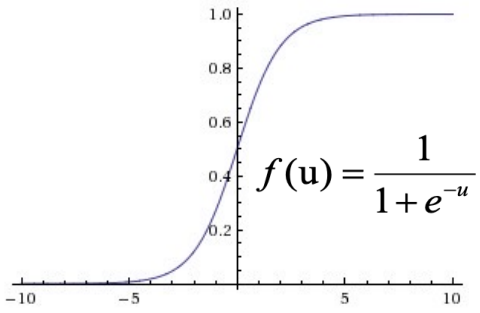
- 10^{11} neurons in our brain and 10^3 synapses per neuron.
- Neuron is a complex electrochemical device with membrane potential.

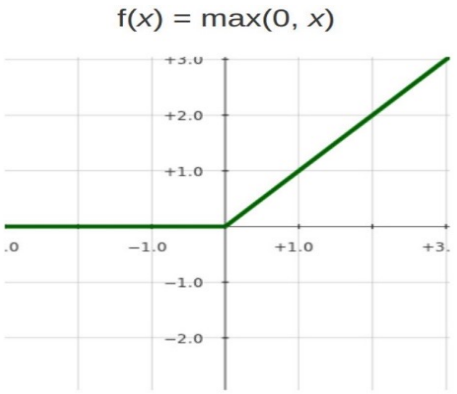




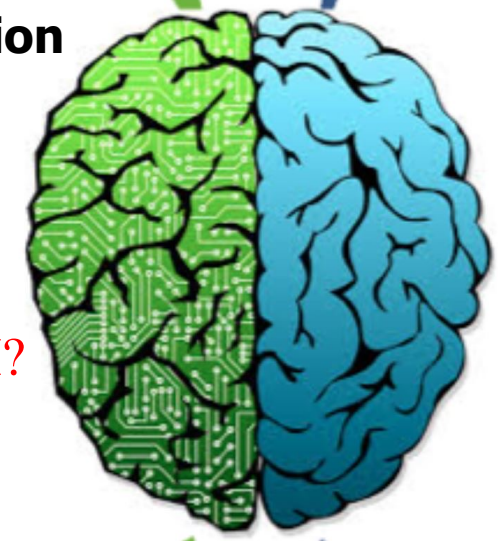
b: bias

voltage x conductance = current

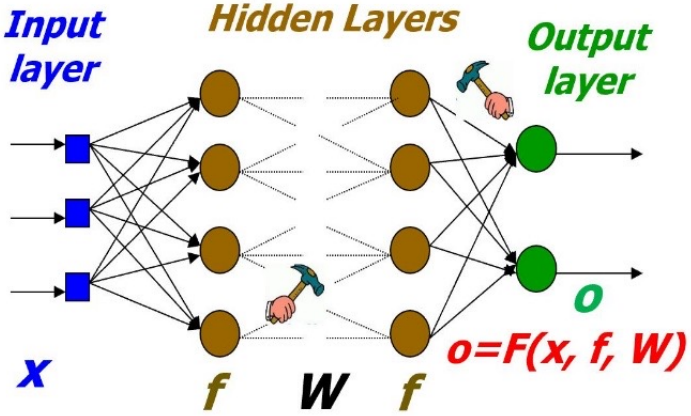




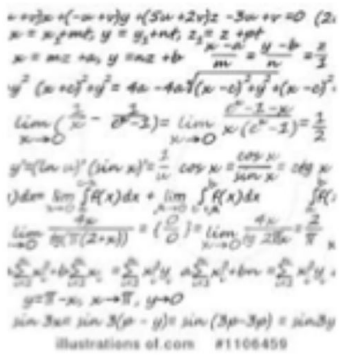
Multilayer Perceptron and Artificial Intelligence (AI)



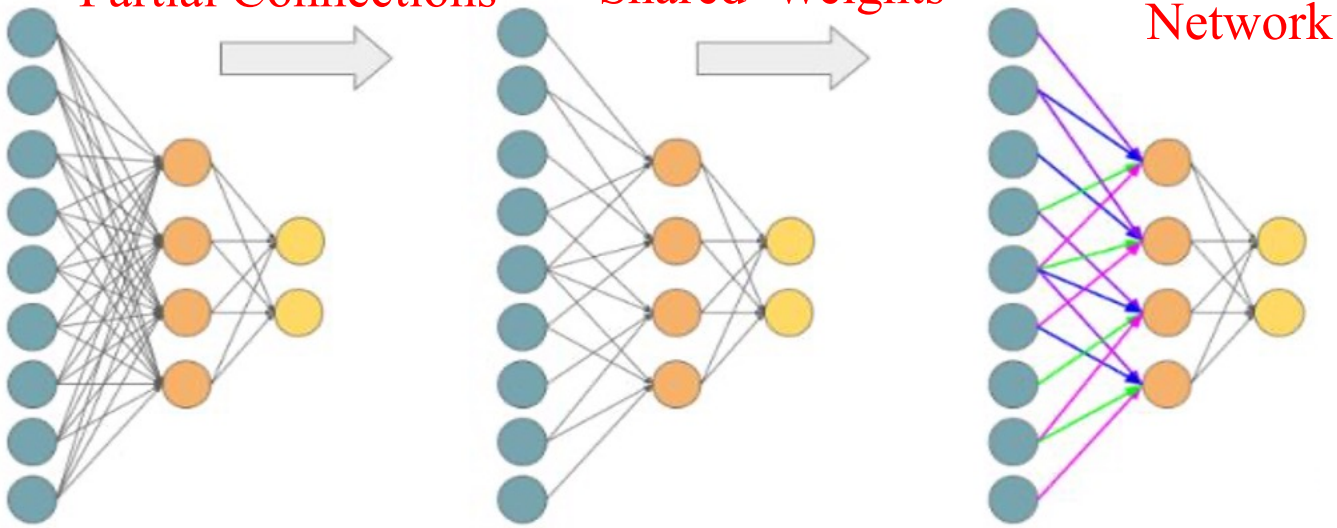
Multlayer Perceptron (MLP)
Learning = weight adjustment



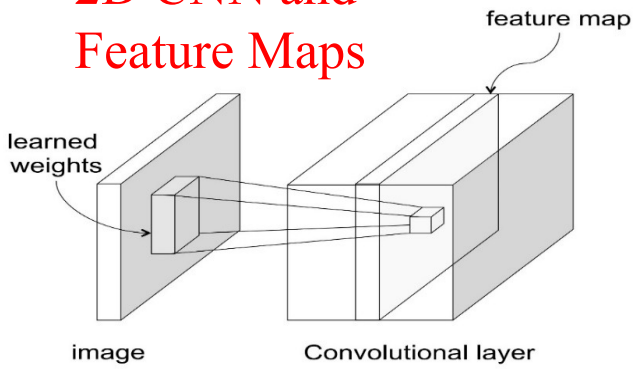
Large Language Model (LLM)?
Generative Visual AI?



Convolution Neural Network



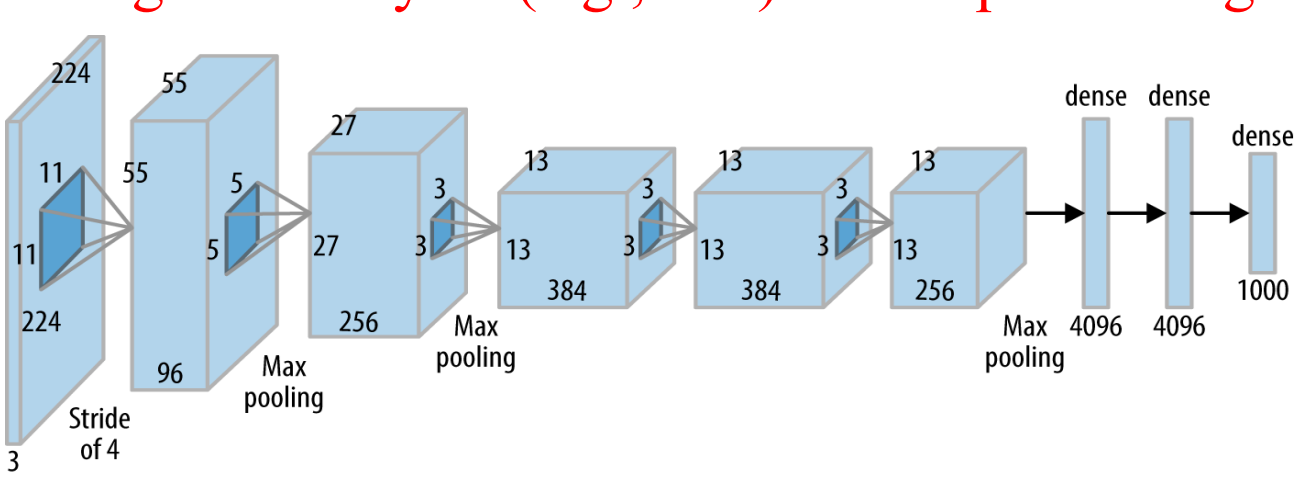
2D CNN and Feature Maps



Backpropagation Learning for MLP can still be applied

AlexNet for ImageNet (2012)

Large # of Layers (e.g., 152) → Deep Learning



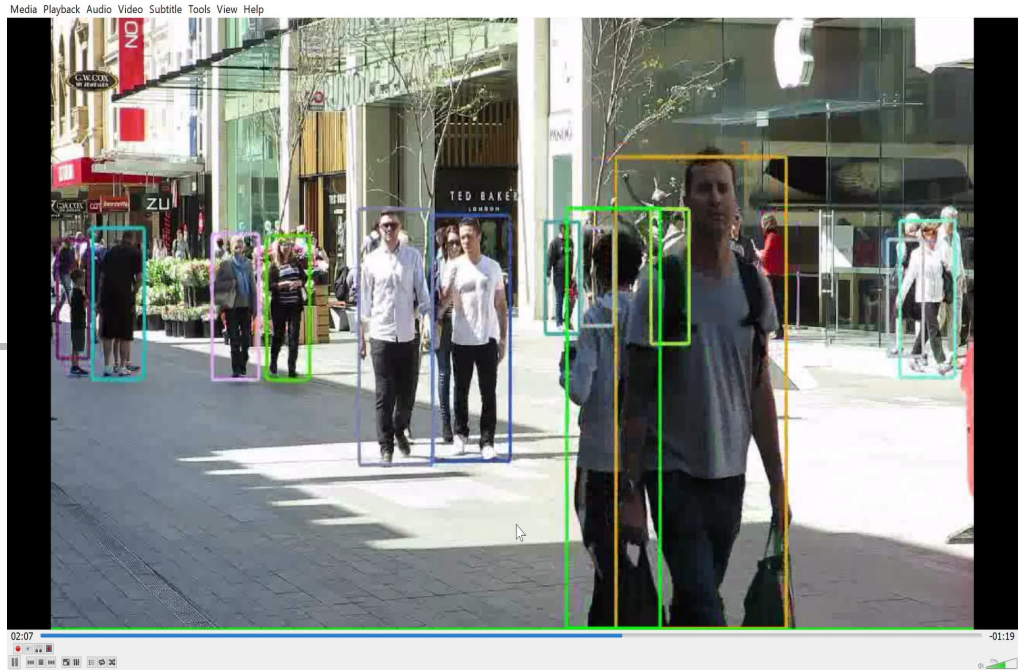
1000 object classes, 1.3M labelled

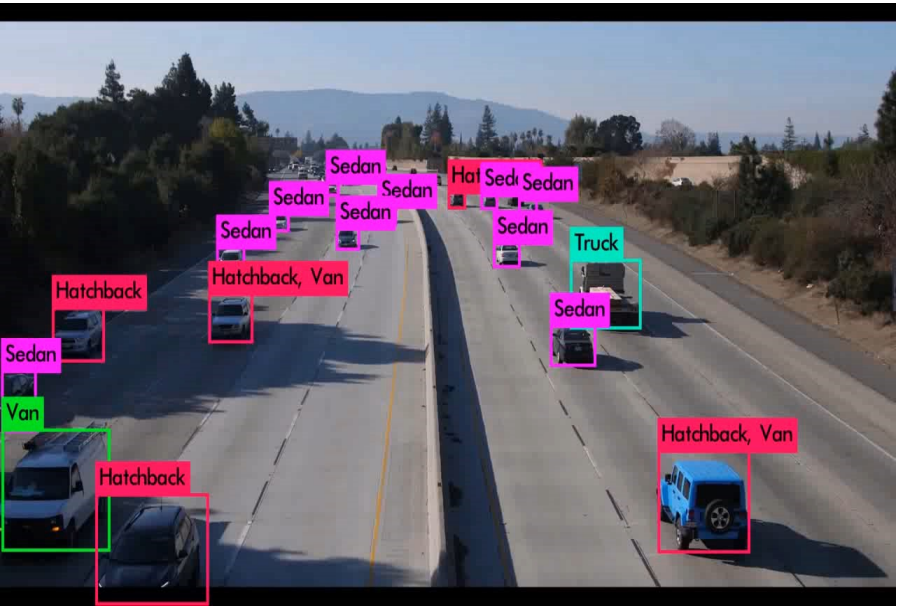
Feature Extraction & Representation

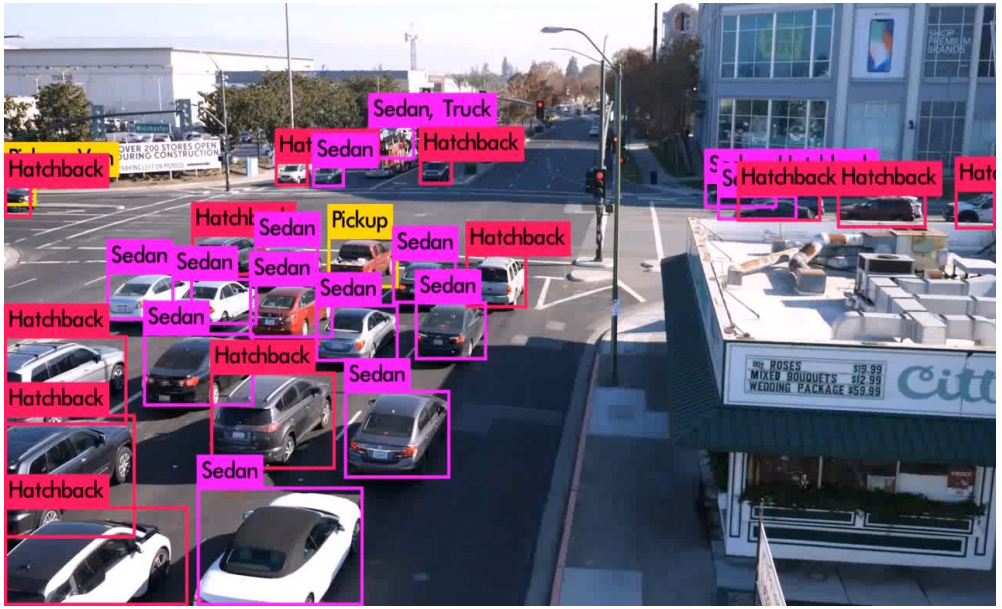
Classification

A single-stage (end-to-end) joint optimization based ML (enough big training data, powerful computing resources)

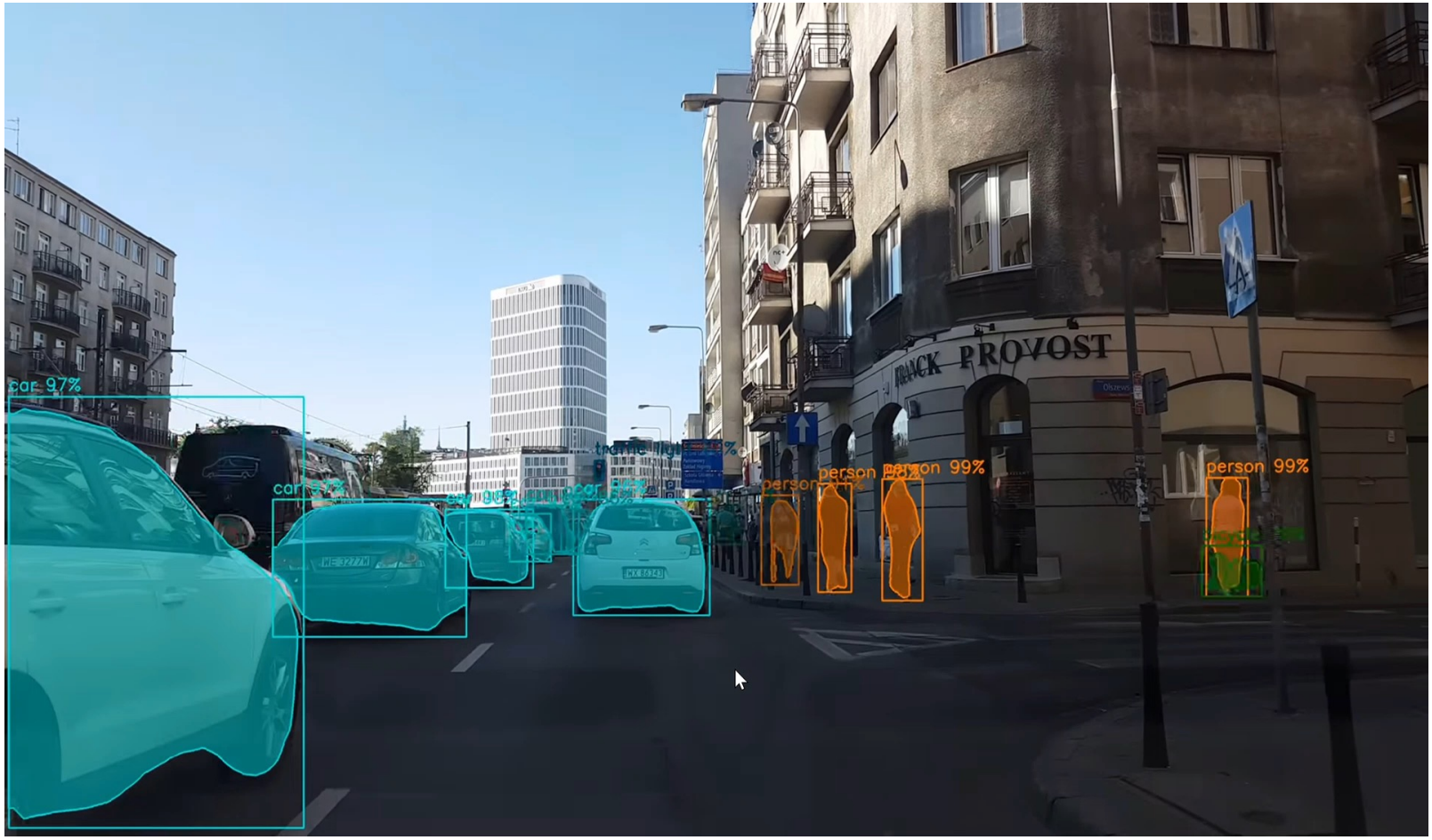
Deep Learning based Object Detectors







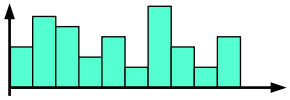
Simultaneous Detection & Instance Segmentation

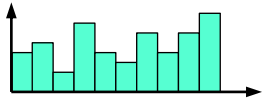


Tracking by Detection

tracking-by-detection

video frames → object detection → object tracking

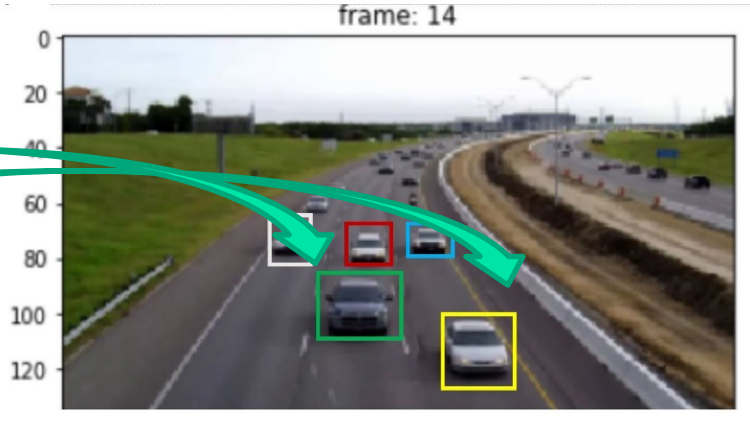




Embedding (Re-ID) Features

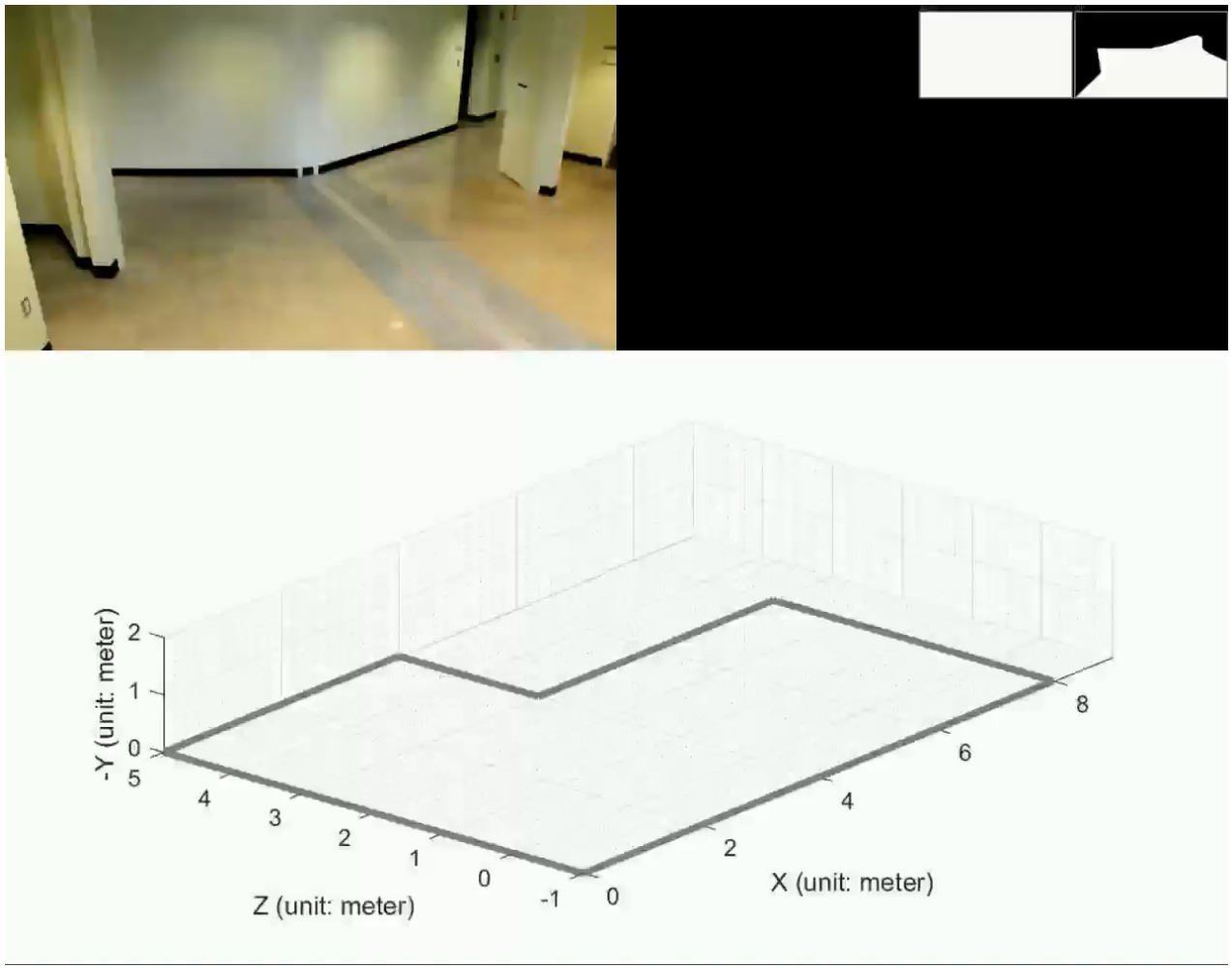
Detection at **t**

Association

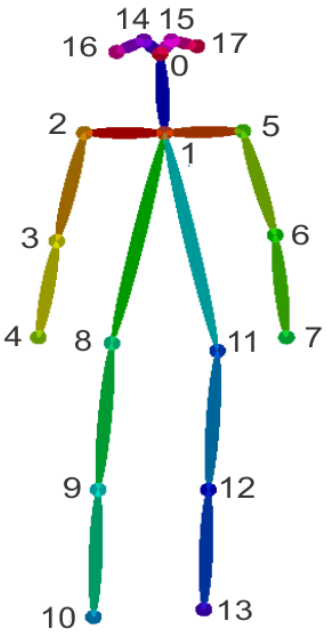


Detection at **t+1**

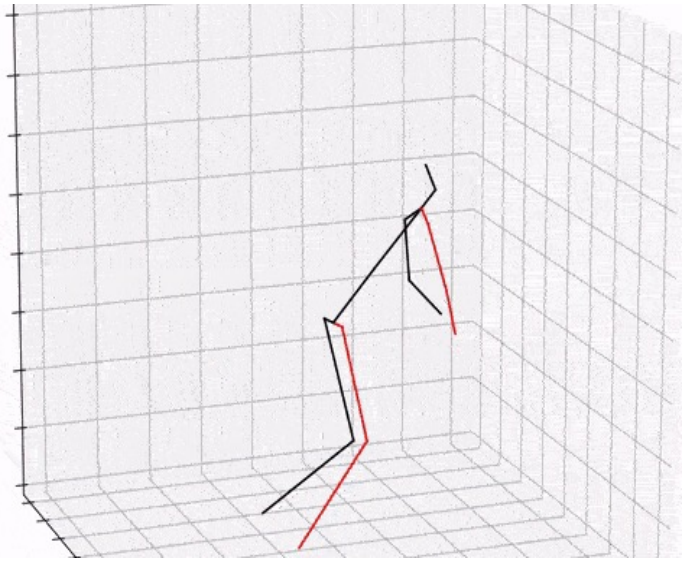
3D Localization of Tracked Objects and Speed Inference



2D/3D Human Pose Estimation (HPE)

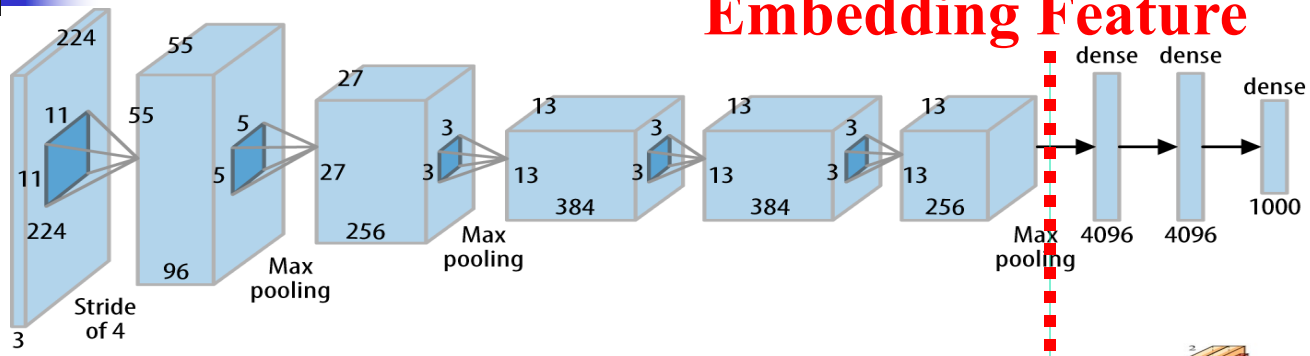


2D body keypoints



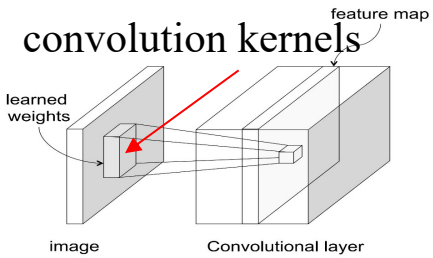
CNN based Embedding

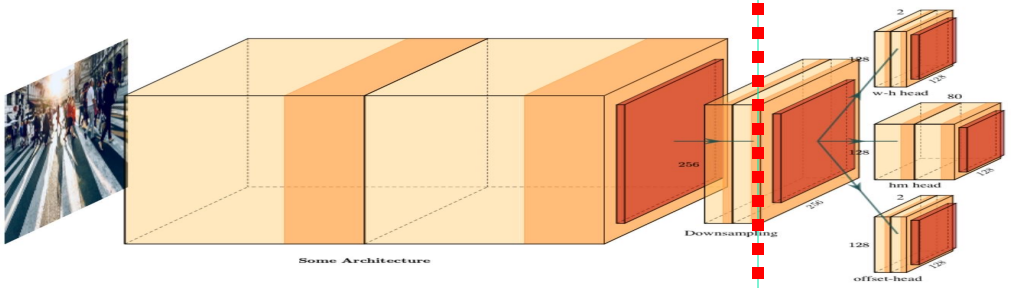
Embedding Feature



Classification

Pattern search via convolution kernels



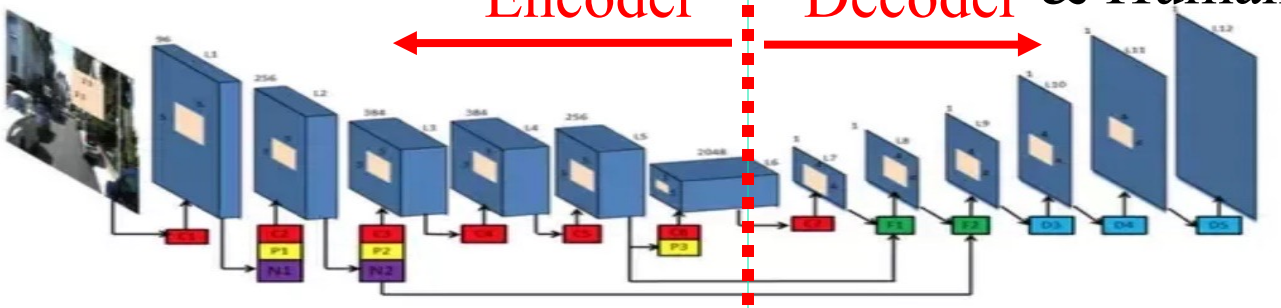


Detection

Segmentation & Human Pose

Encoder

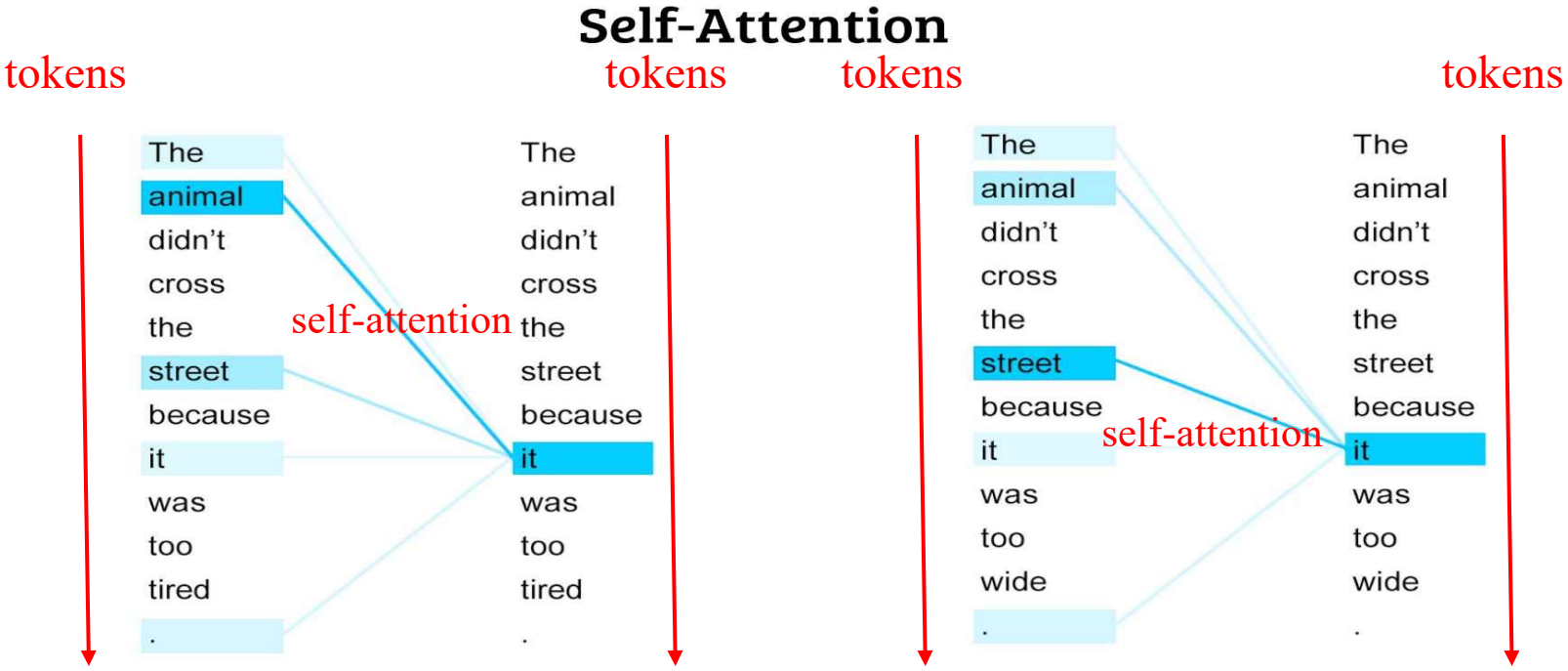
Decoder



Any other way to get better embedding features?

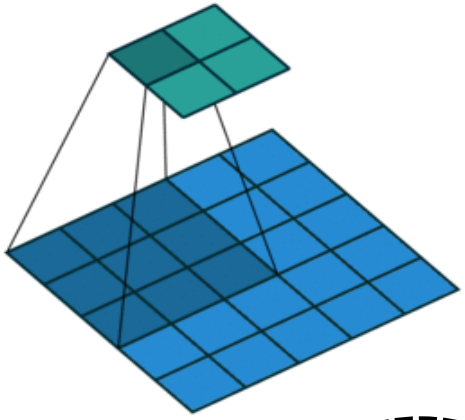
Exploring Self-Attention in Natural Language Processing

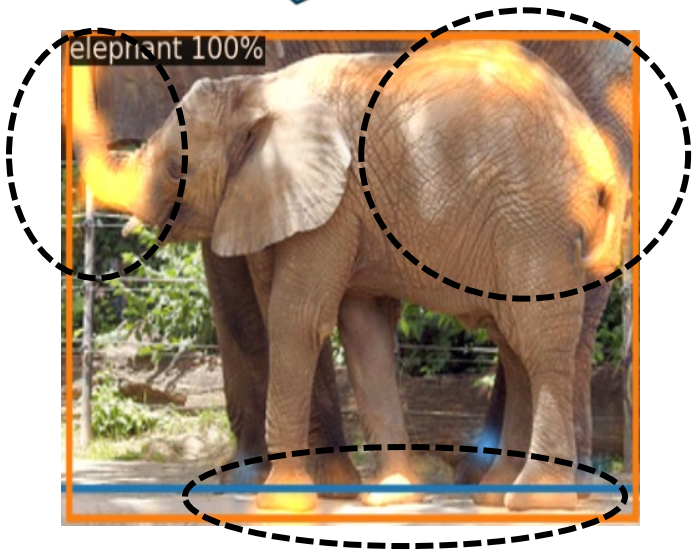
- Self-attention obtained by transformers captures long-range semantic dependencies
  - Creates global context into the output embeddings



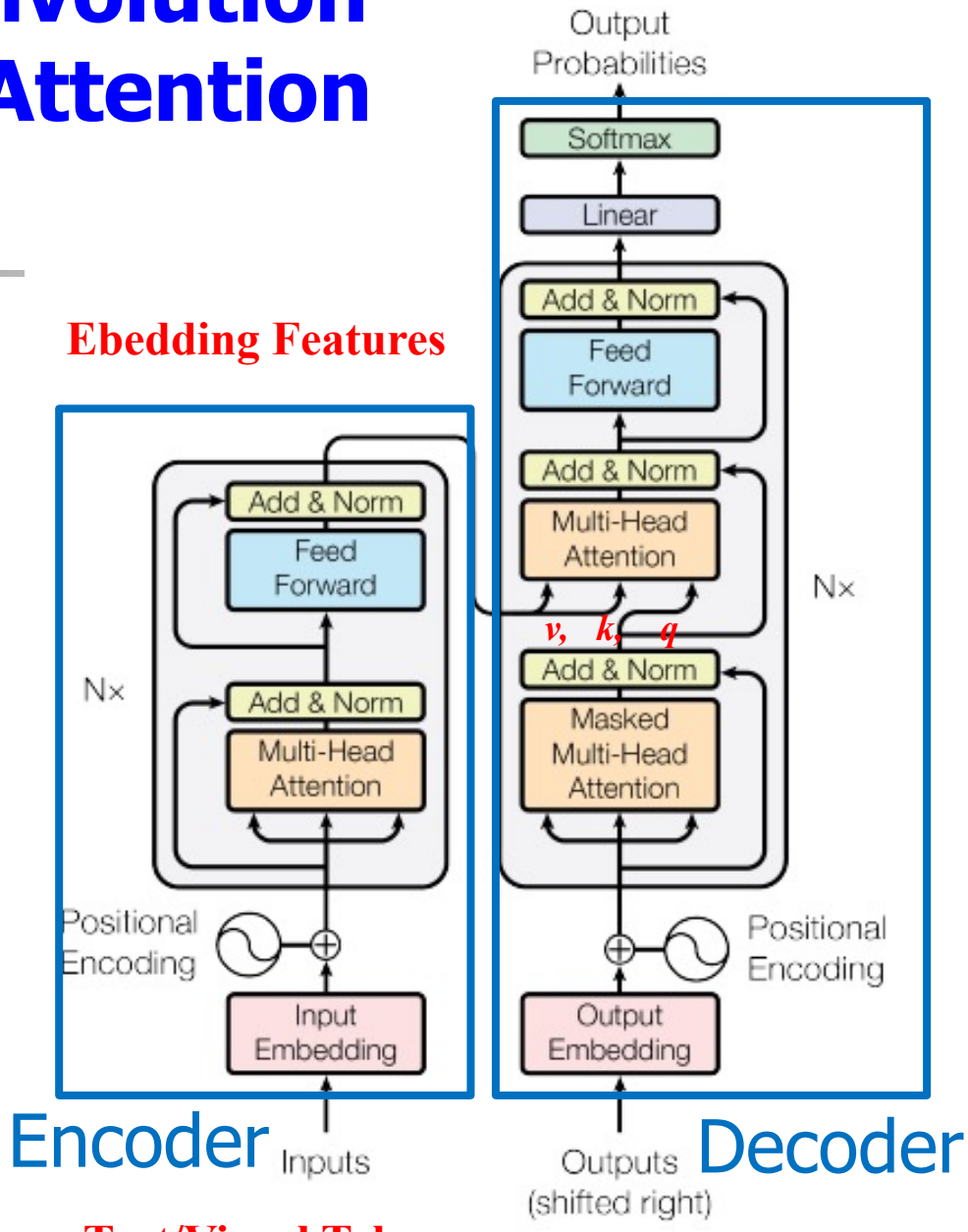
Ashish Vaswani, et al., "Attention Is All You Need," NIPS 2017, Google, https://arxiv.org/abs/1706.03762

From Local Convolution to Global Self-Attention (Transformer)





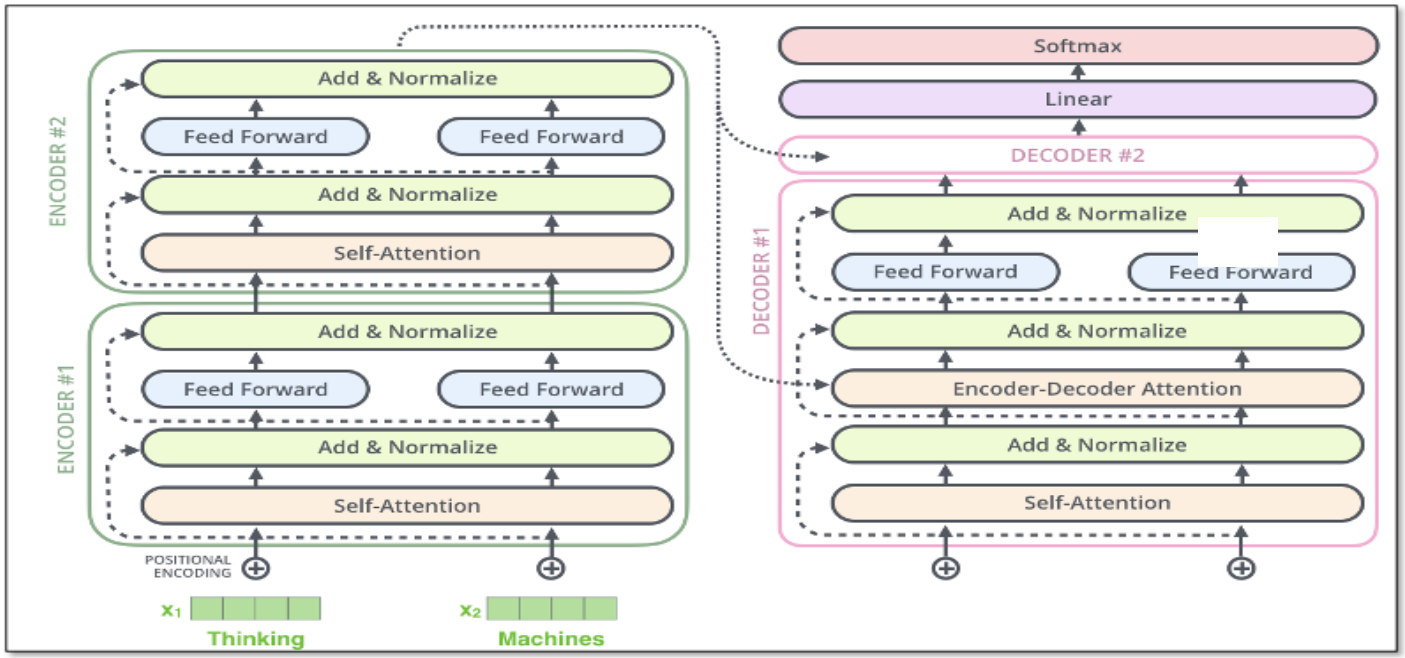
Ebedding Features



Text/Visual Tokens

Large Language Models (LLM) and Artificial General Intelligence (AGI)

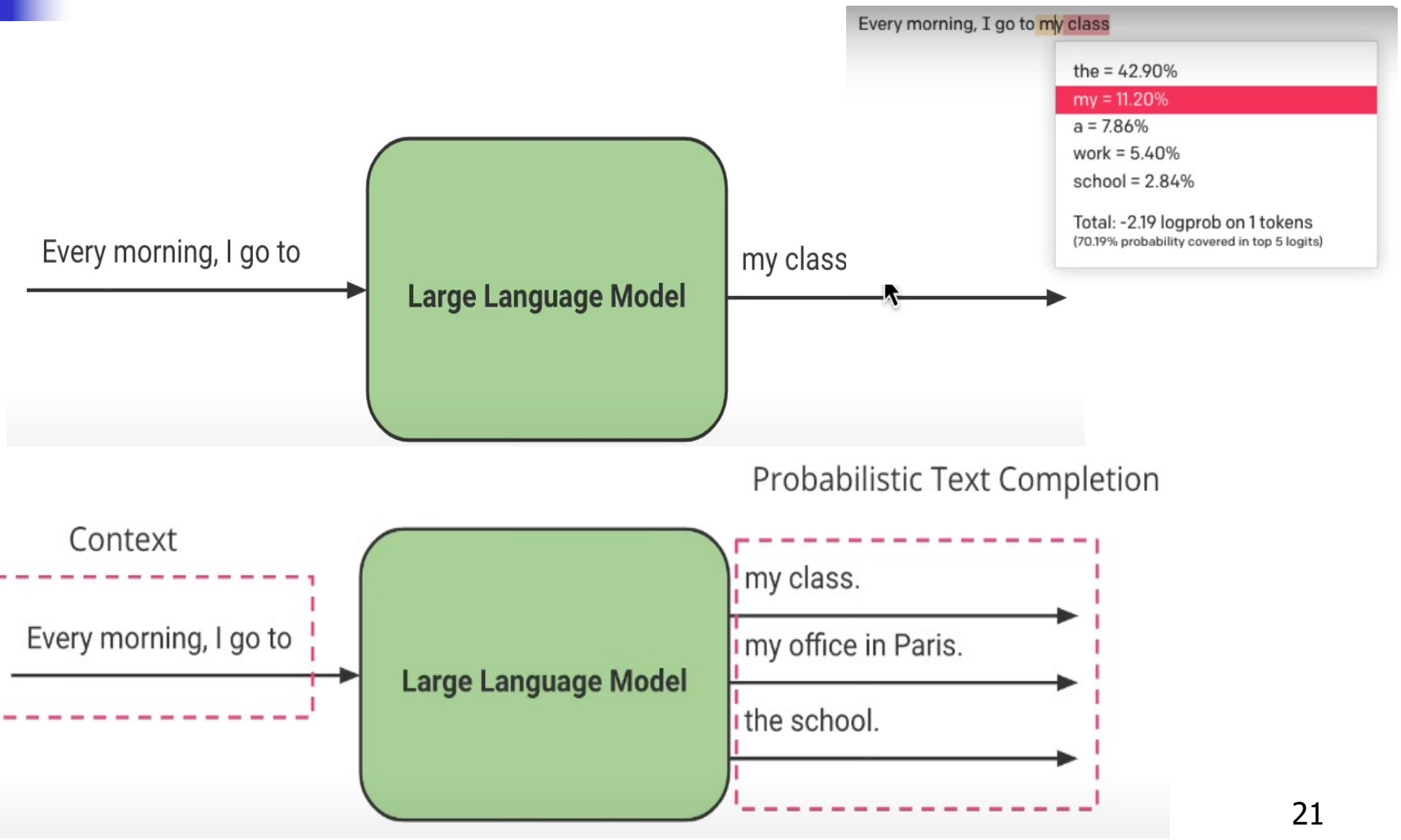
- Encoder only transformer for embedding features extraction, e.g., BERT
- A pre-trained encoder to provide information and allow decoder to output "next word" one-by-one, that gives us GPT, LaMDA/PaLM, LLaMA, Alpaca, etc



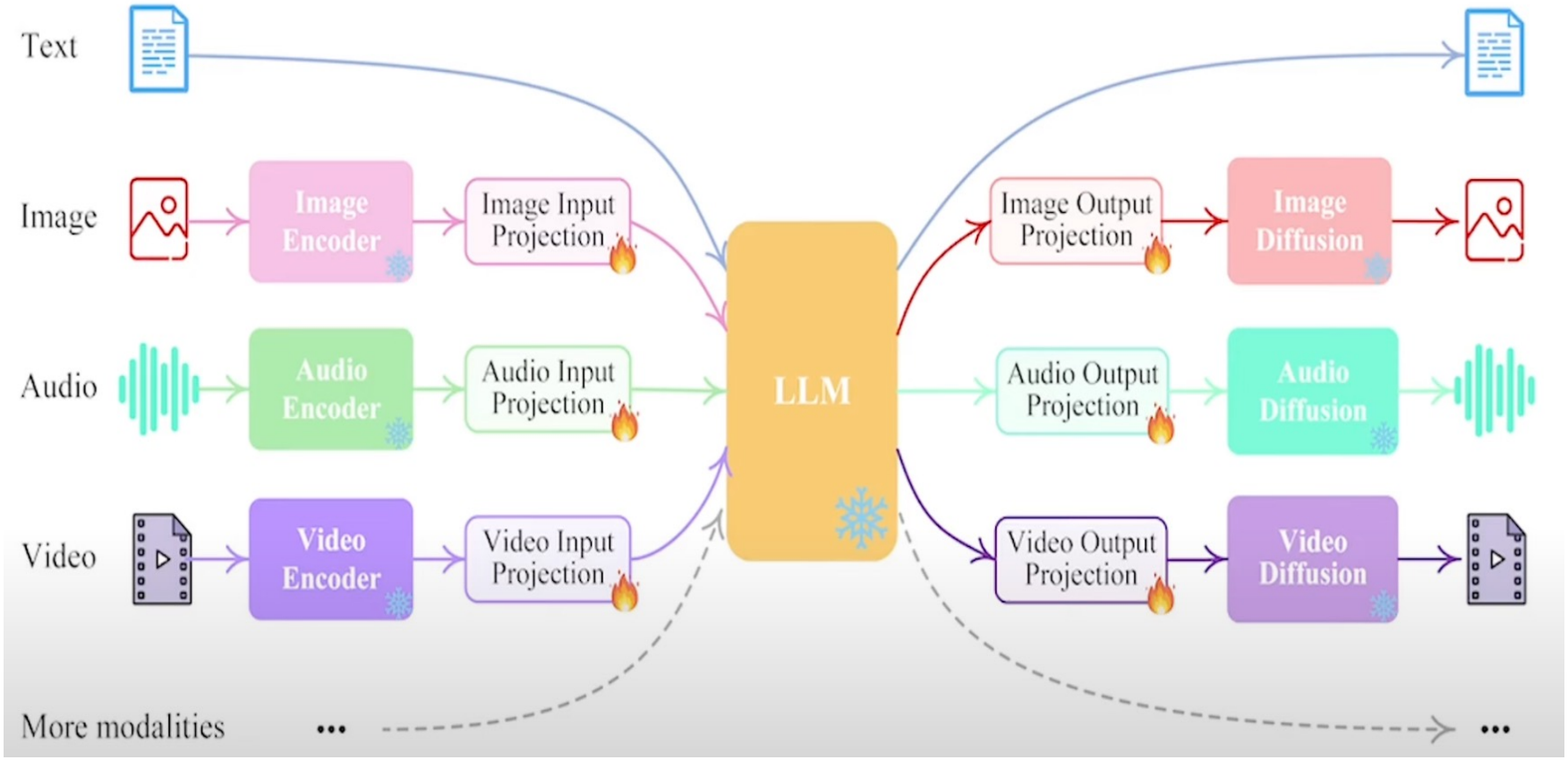
Encoder (e.g., BERT) Decoder (e.g., GPT)

Generative Pre-trained Transformer

ChatGPT: Predicting the Next Word: Probabilistic Text Solitaire

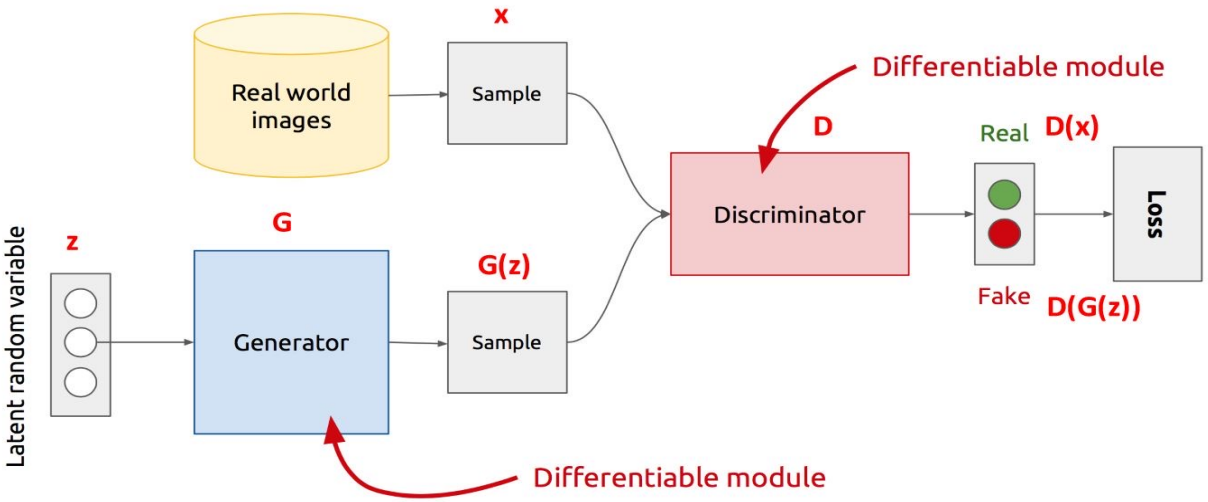


Multimodality LLMs

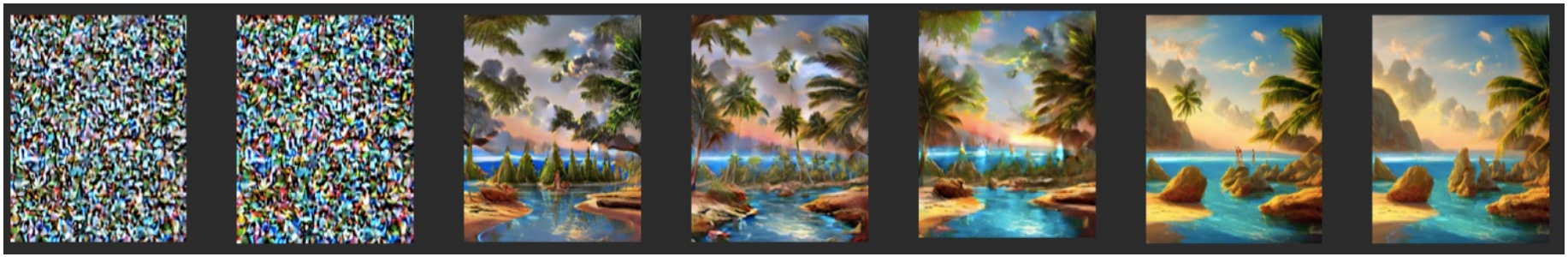


GPT4-o, Gemini 1.5 Pro

From Generative Adversarial Networks (GANs) to Diffusion

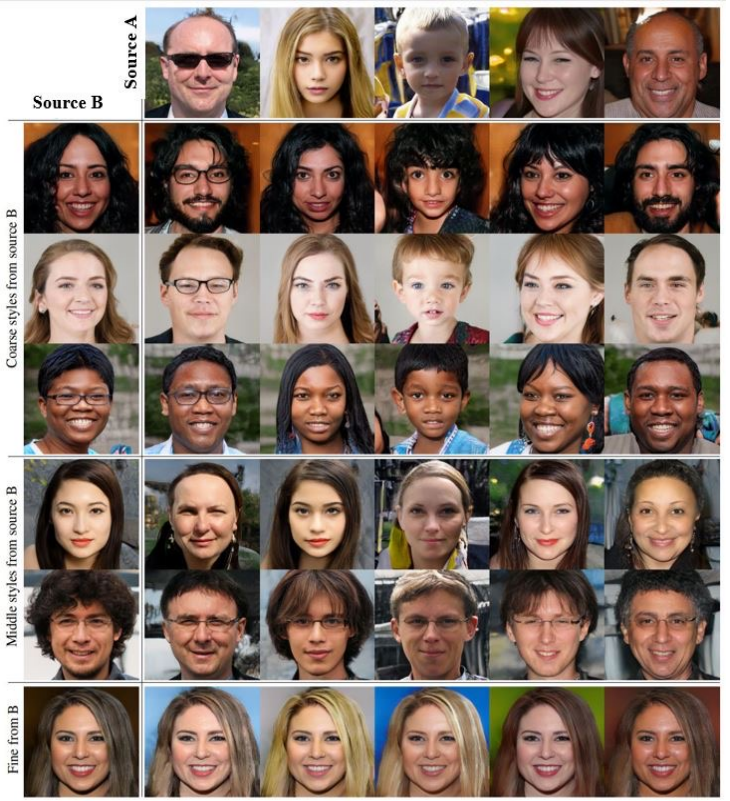


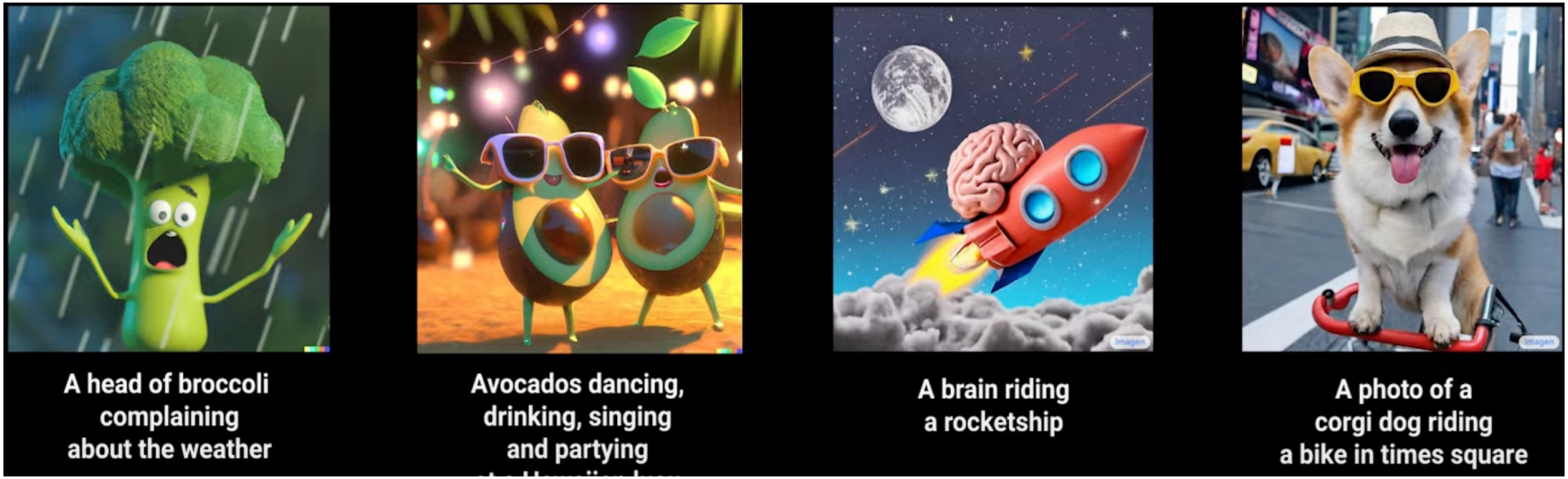
Reverse Diffusion



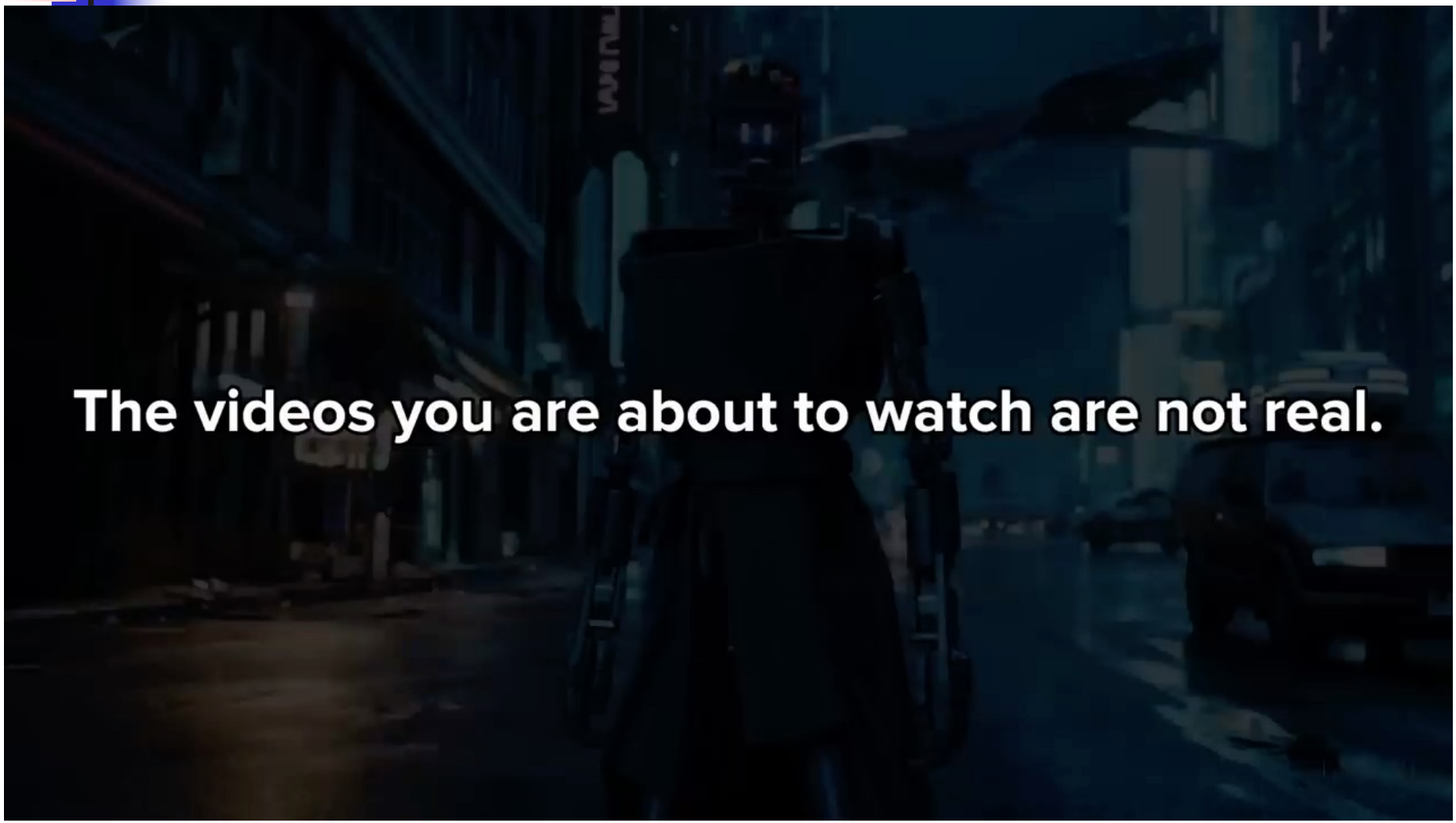
Forward Diffusion

Image Generation: from Noise and Text (GAN & Diffusion)





Sora: Text-to-Video



Machine Learning Paradigms

- Supervised Learning

Given $D = \{x_i, y_i\}$; Learn $y_i = f(x_i)$; s.t., $x_{new} \rightarrow y_{new}$

- Unsupervised Learning, Self-Supervised Learning

Given $D = \{x_i\}$; Learn $y_i = f(x_i)$; s.t., $x_{new} \rightarrow y_{new}$

- Semi-Supervised (Inductive) Learning

Given $D = \{x_i, y_i\}$ and $\{x_j^u\}$; Learn $y_i = f(x_i)$; s.t., $x_{new} \rightarrow y_{new}$

- Reinforcement Learning (game, control, operation)

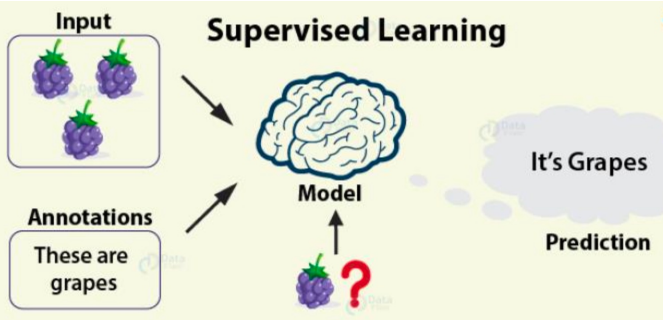
$\{s_t, a_t, s_{t+1}\} \rightarrow r_t$ (under a policy); choose $\{a_1, a_2, \ldots, a_T\}$; s.t., $\max \sum_{t=1}^{T} r_t$

- Query (Active) Learning

Given $D = \{x_i, y_i\}$ and $\{x_j^u\}$; Query $x_j^u \rightarrow y_j^*$; Learn $y = f(x)$; s.t., $x_{new} \rightarrow y_{new}$

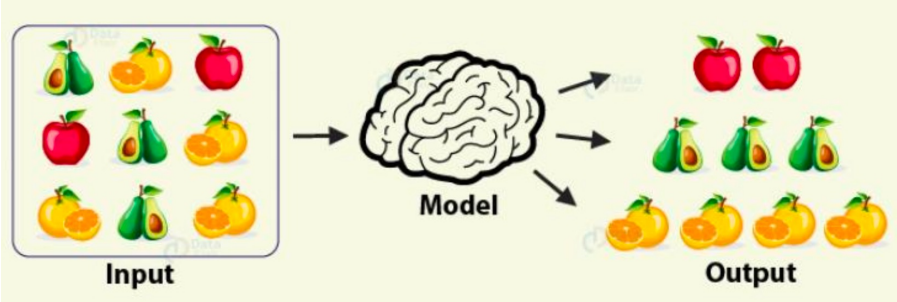
Supervised & Unsupervised Learning

- Supervised Learning: needs large amounts of labeled data to train system for specific use cases



expensive in data annotation

- Unsupervised Learning: find implicit patterns of data without being trained on labeled data



clustering or grouping (feature space)

Supervised & Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Unsupervised Learning

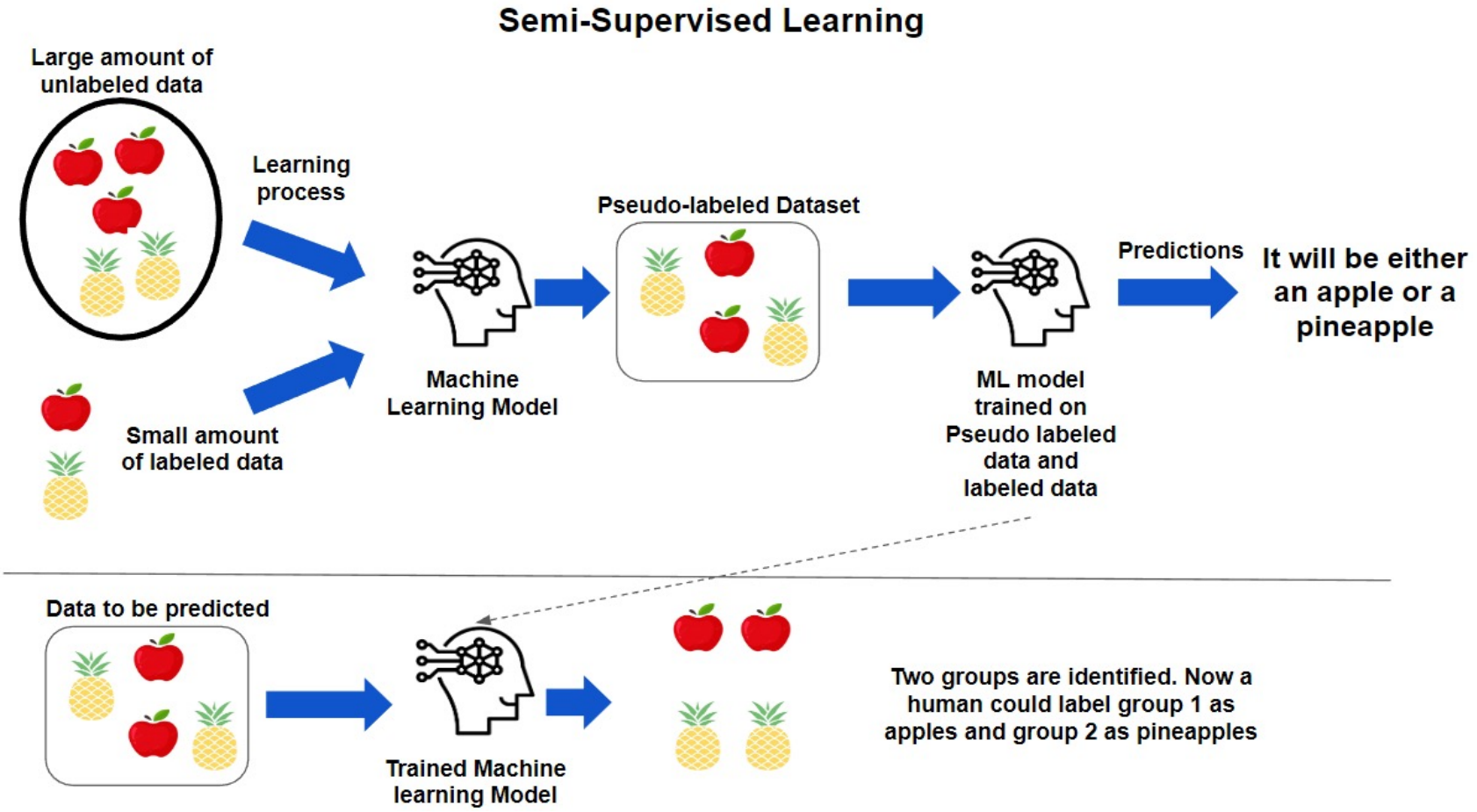
Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Semi-Supervised Learning

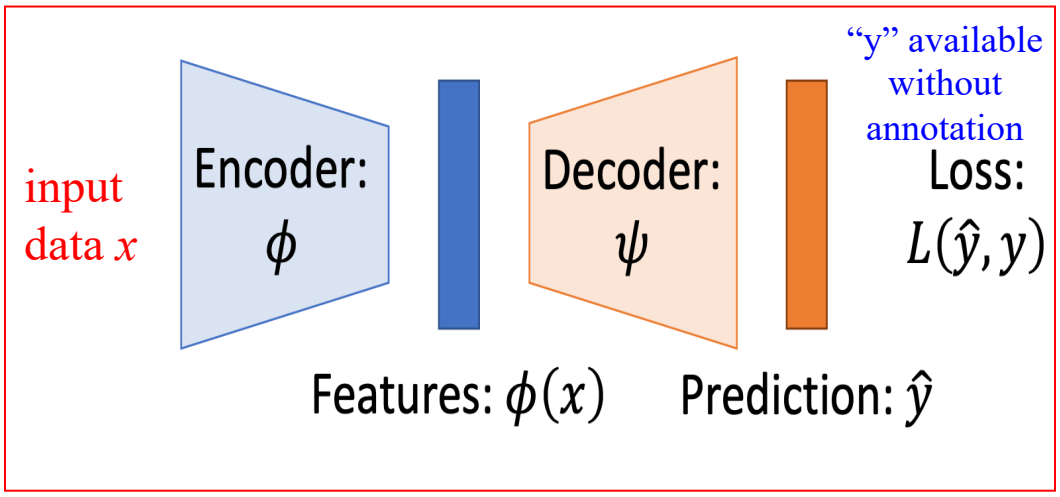


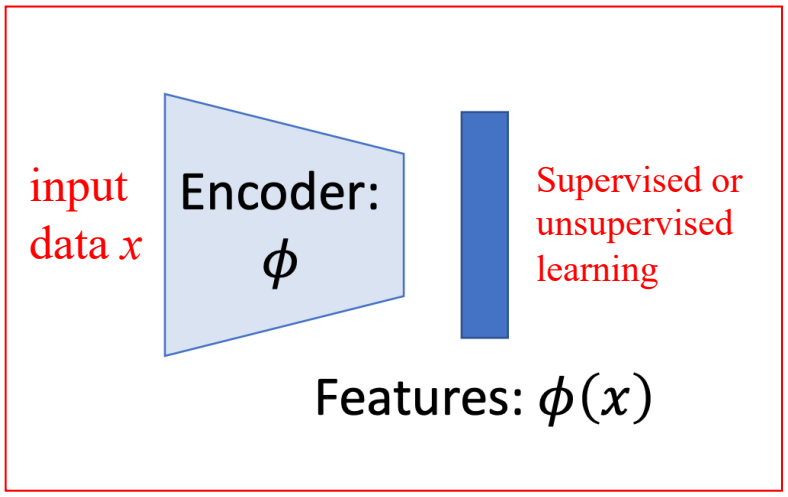
Self-Supervised Feature Learning

- **Step 1:** Pretrain a model to obtain effective features that doesn't require supervision (e.g., PCA, Auto-Encoder, …)

$$L(x) = R(x, \hat{x})$$
$$= \|x - \hat{x}\|_2^2$$

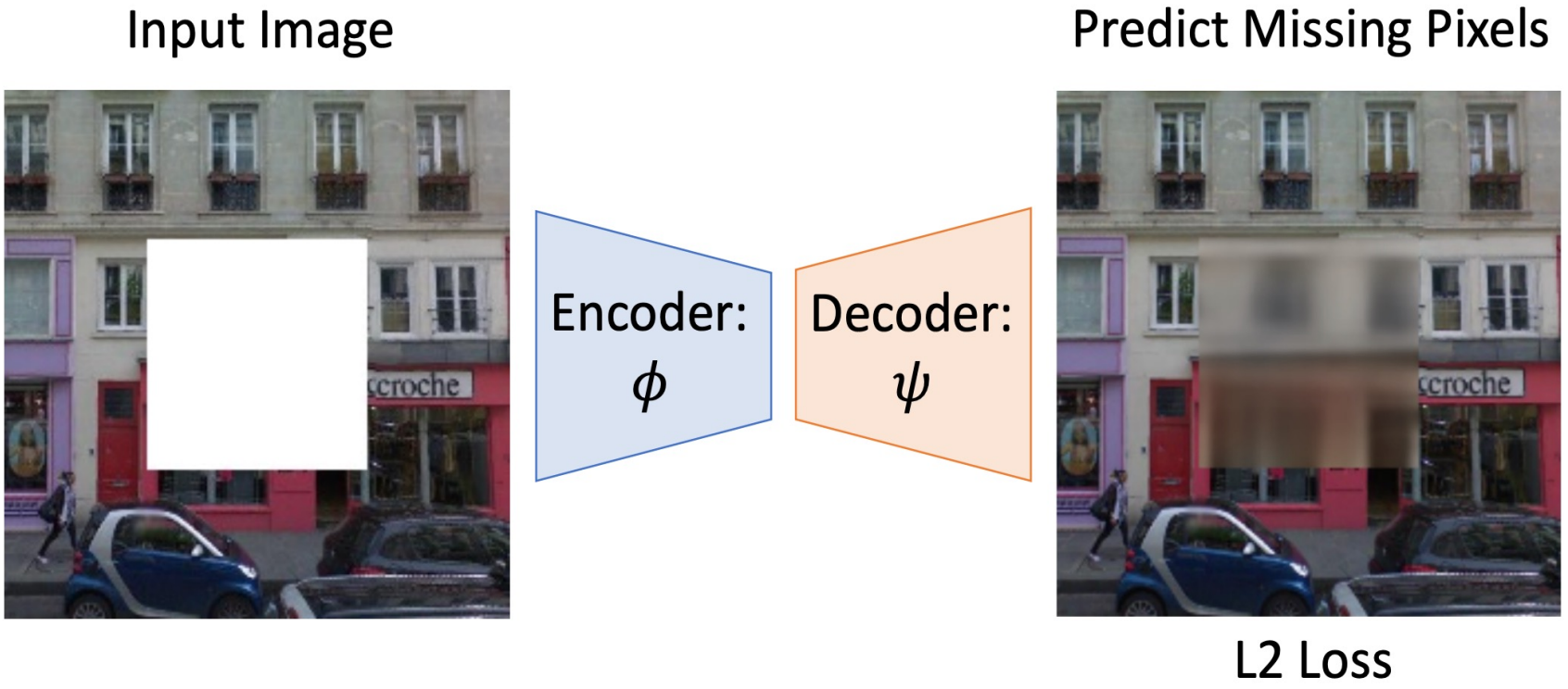
- **Step 2:** Transfer encoder to downstream tasks via supervised or unsupervised learning





Step 1 **Step 2**

Deep Learning based Self-Supervised Feature Learning



Pathak et al, “Context Encoders: Feature Learning by Inpainting”, CVPR 2016

Deep Learning based Self-Supervised Feature Learning

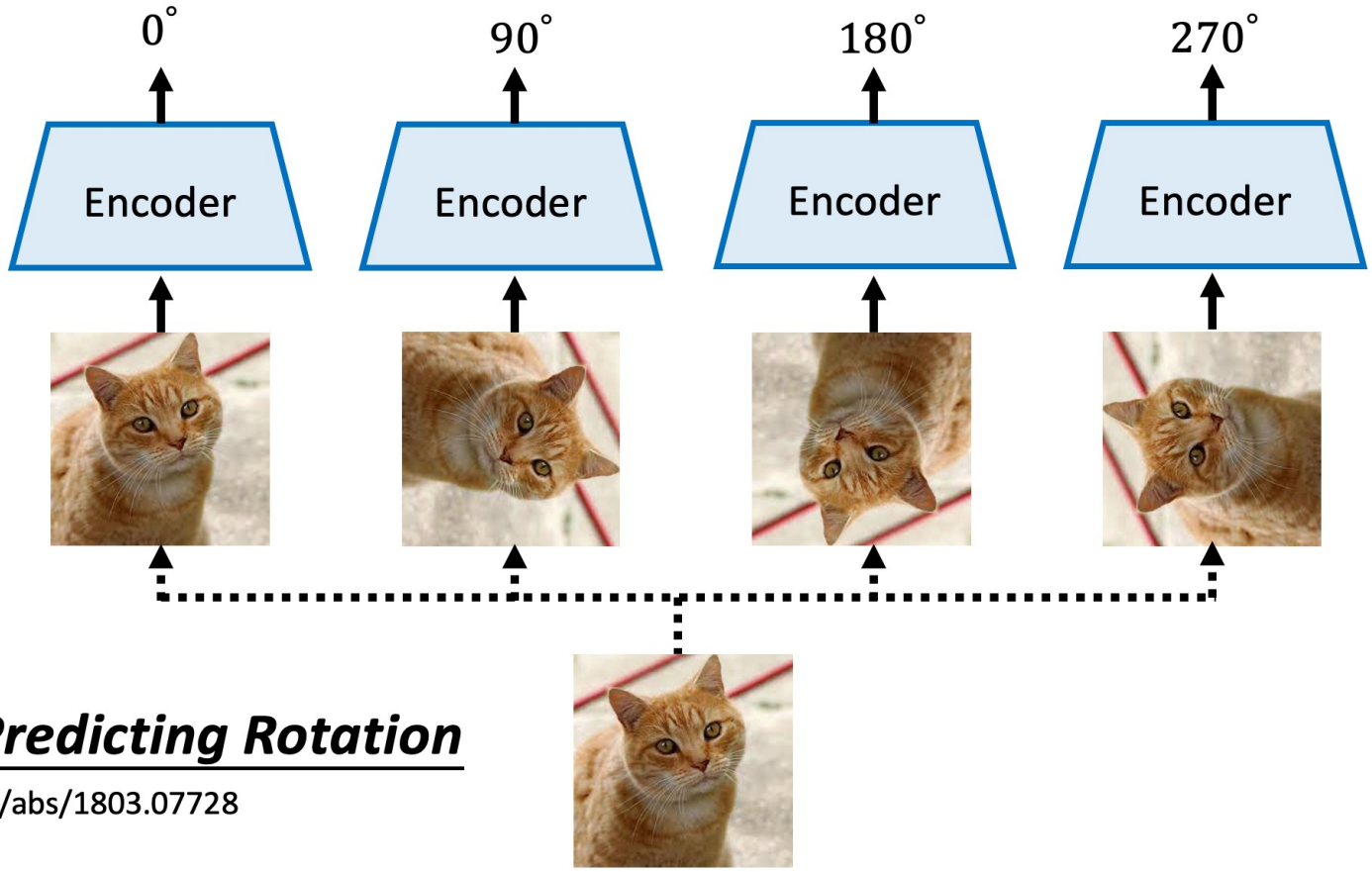
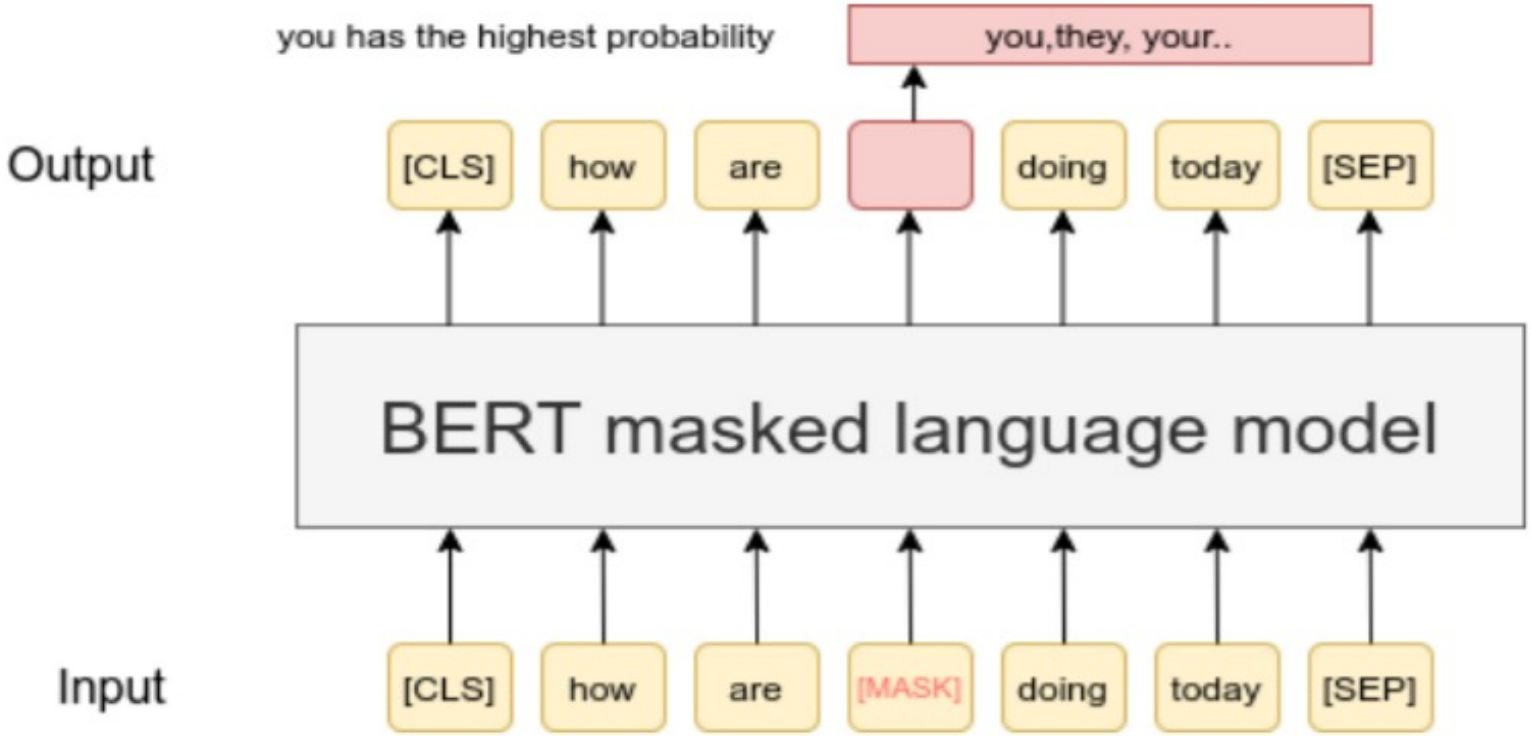


Image - Predicting Rotation

https://arxiv.org/abs/1803.07728

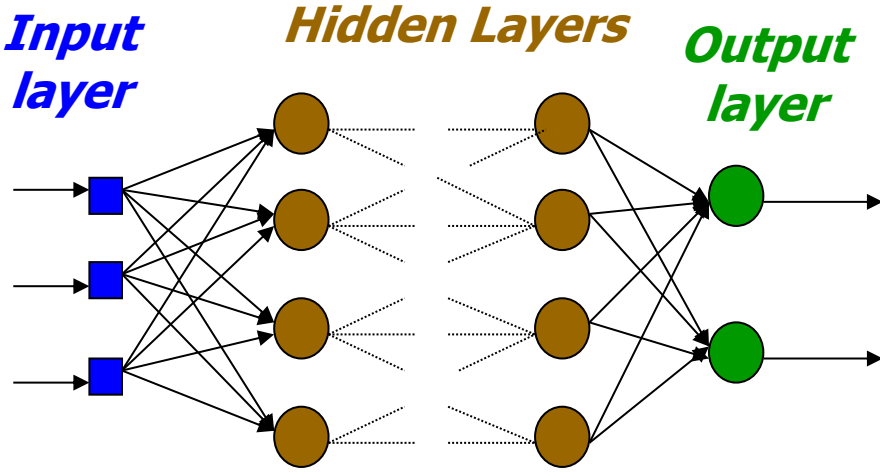
Deep Learning based Self-Supervised Feature Learning

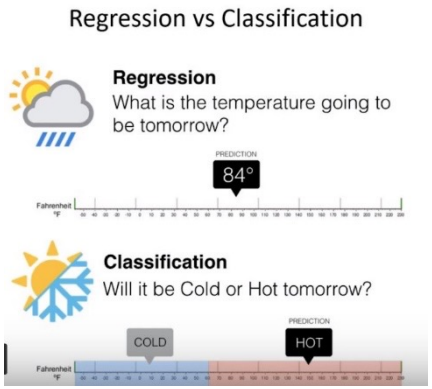
Predict the Missing/Masked Words

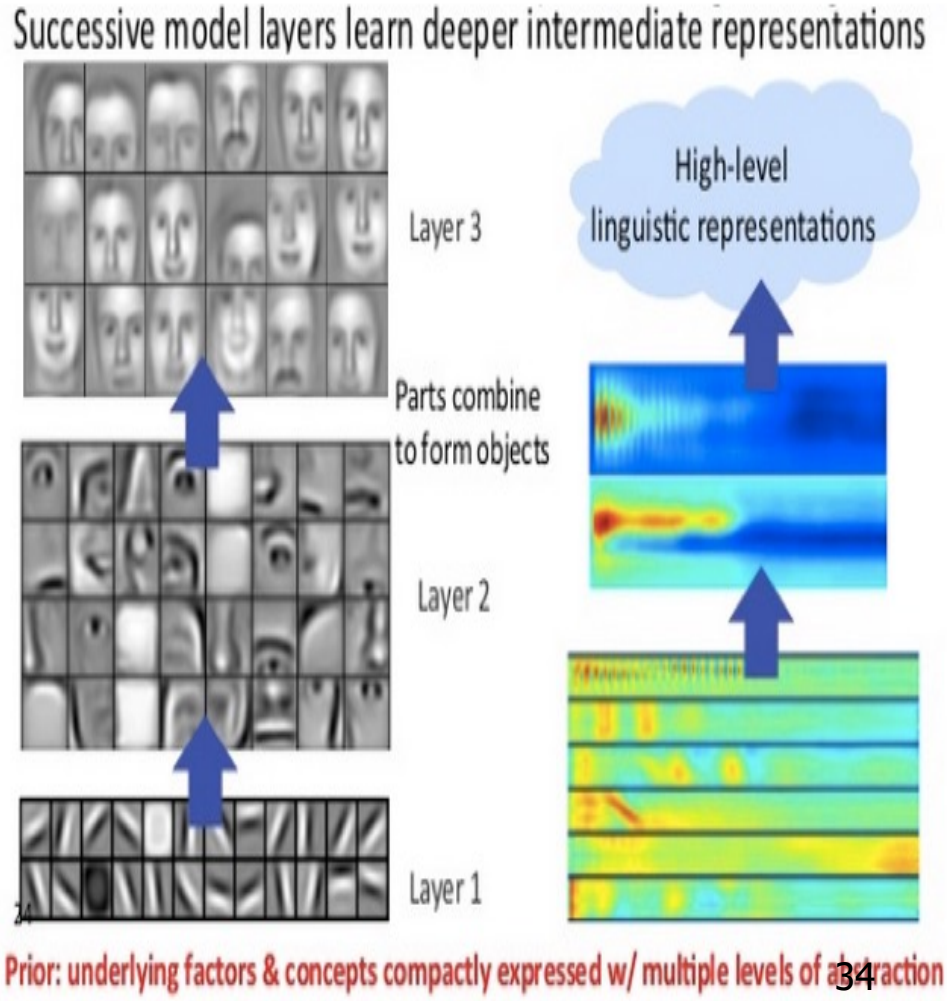


Jacob Devlin, et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arxiv.org/abs/1810.04805

A Multilayer Perceptron (MLP) Neural Network



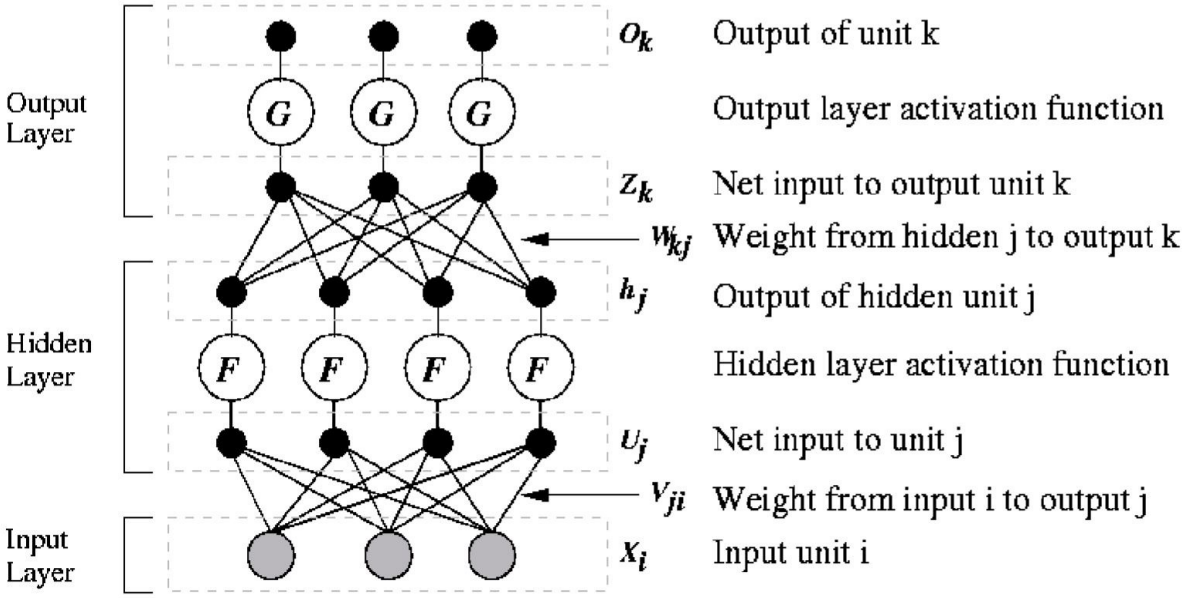




BackPropagation (BP) Learning for An MLP

$$h_j(\mathbf{x}) \overset{f(u_j)}{=} f\left(w_{j0} + \sum_{i=1}^{D} x_i v_{ji}\right)$$

$$o_k(\mathbf{x}) \overset{g(z_k)}{=} g\left(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}\right)$$



R. Urtasun & R. Zemel, "CSC 411, Lecture Note 10", University of Toronto, 2015

Sum/mean squared error (SSE/MSE) for regression applications

$$E = \frac{1}{2}\sum_k (o_k - t_k)^2; \qquad \frac{\partial E}{\partial o_k} = o_k - t_k$$

Stochastic gradient descent

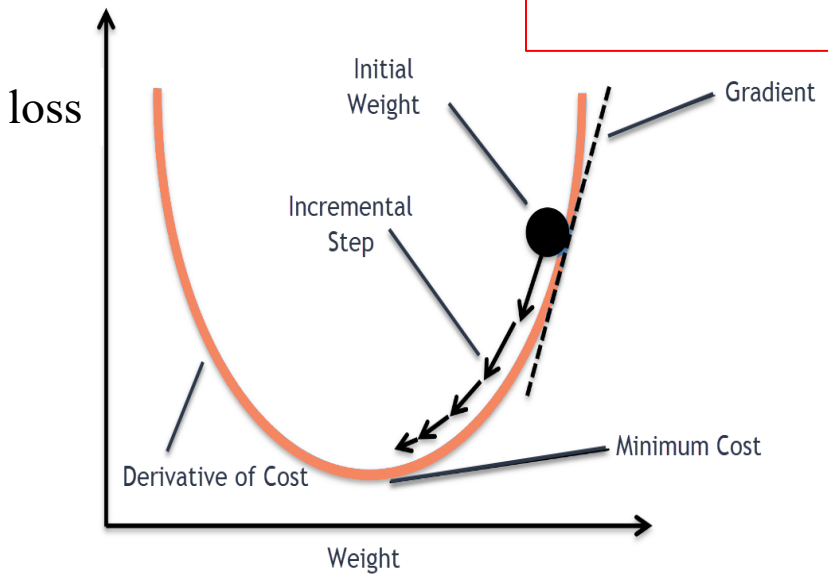
$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}$$

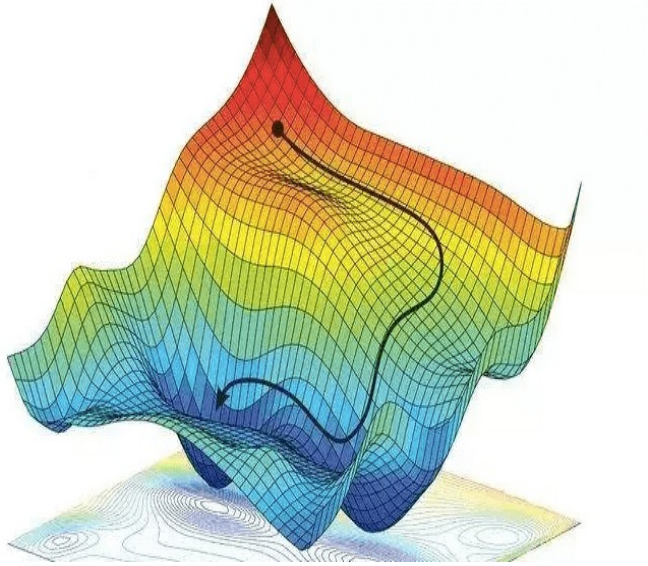
$$v_{ji} \leftarrow v_{ji} - \eta \frac{\partial E}{\partial v_{ji}}$$

η: learning rate

Gradient Descent Search

$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}$$





Rumelhart, D. E., G. E. Hinton & R. J. Williams. (1986b). "Learning Representations by Back-Propagation Error." Nature, 323:533 - 536.

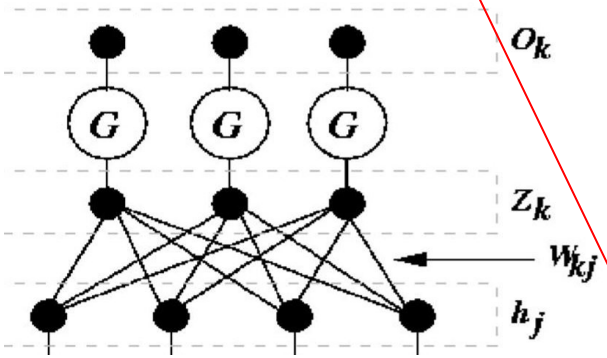
Gradients of Output Layer

- Assuming the error function is mean-squared error (MSE), on a single training example n, we have

$$o_k(\mathbf{x}) = g(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj})$$

$g(z_k)$

$$\frac{\partial E}{\partial o_k^{(n)}} = o_k^{(n)} - t_k^{(n)}$$



Using logistic activations

$$o_k^{(n)} = g(z_k^{(n)}) = (1 + \exp(z_k^{(n)}))^{-1}$$

$$\frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} = o_k^{(n)}(1 - o_k^{(n)})$$

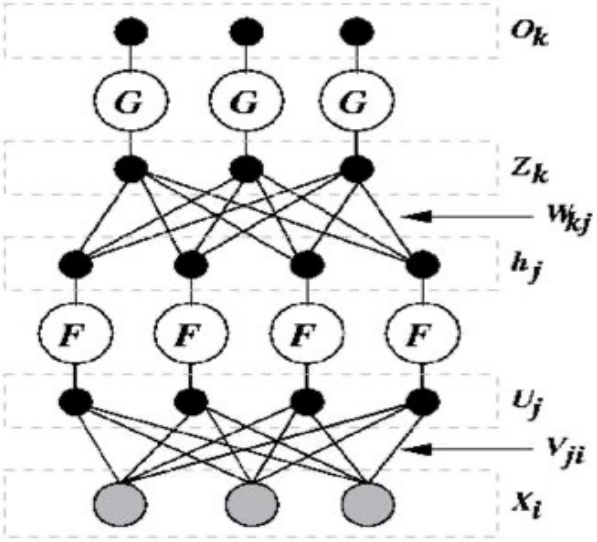
- The error gradient is then:

$$\frac{\partial E}{\partial w_{kj}} = \sum_{n=1}^{N} \frac{\partial E}{\partial o_k^{(n)}} \frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} \frac{\partial z_k^{(n)}}{\partial w_{kj}} = \sum_{n=1}^{N} (o_k^{(n)} - t_k^{(n)}) o_k^{(n)} (1 - o_k^{(n)}) h_j^{(n)}$$

- The gradient descent update rule is given by:

$$w_{kj} \leftarrow w_{kj} - \eta \frac{\partial E}{\partial w_{kj}} = w_{kj} - \eta \sum_{n=1}^{N} (o_k^{(n)} - t_k^{(n)}) o_k^{(n)} (1 - o_k^{(n)}) h_j^{(n)}$$

Gradients of Hidden Layers



- The output weight gradients for a multi-layer network are the same as for a single layer network

$$\frac{\partial E}{\partial w_{kj}} = \Sigma_n \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial w_{kj}} = \Sigma_n \delta_k \frac{\partial o_k}{\partial z_k}\frac{\partial z_k}{\partial w_{kj}}$$

where δ_k is the error w.r.t. the output for unit k

- Hidden weight gradients are then computed via back-prop:

$g(z_k)$

$$o_k(\mathbf{x}) = g\left(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}\right)$$

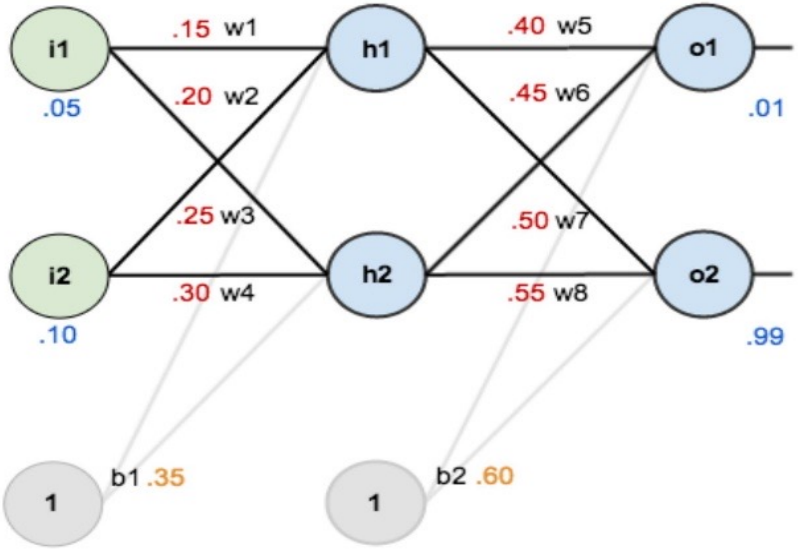
$$\frac{\partial E}{\partial h_j} = \delta_j = \Sigma_k \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial h_j} = \Sigma_k \delta_k \frac{\partial o_k}{\partial z_k}\frac{\partial z_k}{\partial h_j} = \Sigma_k \delta_k \, o_k(1 - o_k) w_{kj}$$

$$h_j(\mathbf{x}) = f\left(w_{j0} + \sum_{i=1}^{D} x_i v_{ji}\right) \quad f(u_j)$$

$$\frac{\partial E}{\partial v_{ji}} = \Sigma_n \frac{\partial E}{\partial h_j}\frac{\partial h_j}{\partial v_{ji}} = \Sigma_n \left(\Sigma_k \delta_k \frac{\partial o_k}{\partial h_j}\right) h_j(1 - h_j) x_i$$

$$\frac{\partial h_j}{\partial u_j}\frac{\partial u_j}{\partial v_{ji}}$$

A Simple Example of BP



$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

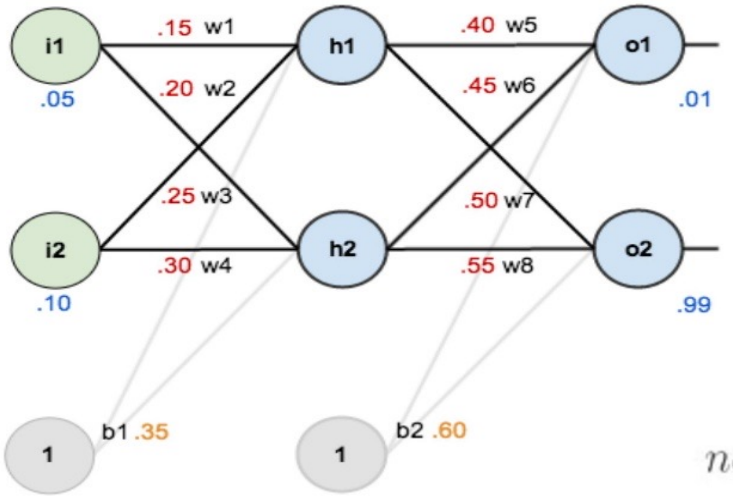
$$E_{o2} = 0.023560026$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

$$out_{h2} = 0.596884378$$

A Simple Example of BP



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

How about update w2?

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Cross-Entropy Loss for BP

- For classification, if it is a binary (one output, 2-class) problem, then cross-entropy loss function often does better

$$-\sum_{i=1}^{n} p(x_i) \times \log q(x_i)$$

log=log₂

$$E = -\sum_{n=1}^{N} t^{(n)} \log o^{(n)} + (1 - t^{(n)}) \log(1 - o^{(n)})$$

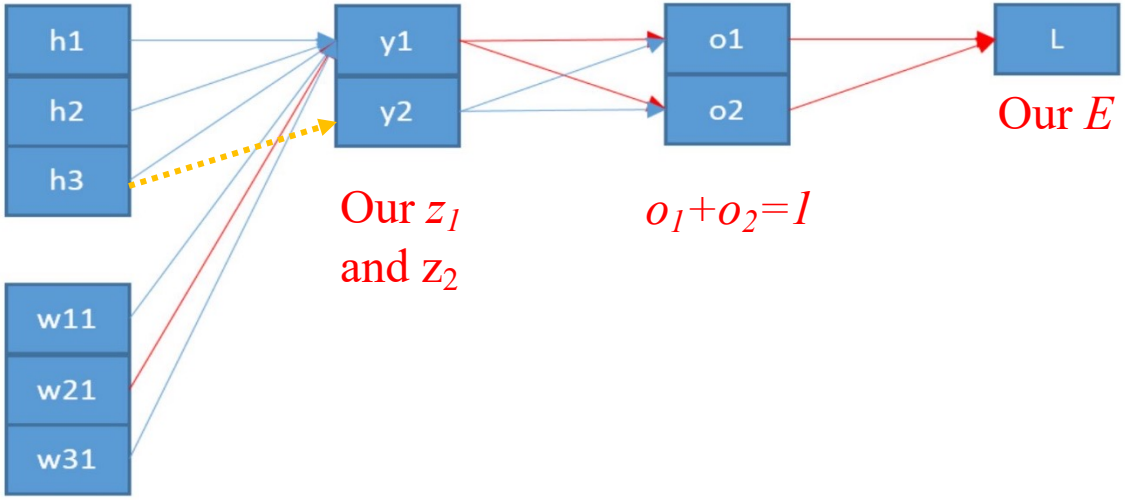
$$o^{(n)} = (1 + \exp(-z^{(n)})^{-1} \quad \text{sigmoid}$$

- For multi-class classification problems, use the softmax activation (prob.)

$$E = -\sum_{n} \sum_{k} t_k^{(n)} \log o_k^{(n)}$$

$$o_k^{(n)} = \frac{\exp(z_k^{(n)})}{\sum_j \exp(z_j^{(n)})}$$

Gradient for Cross-Entropy



$$L = -t_1 \log o_1 - t_2 \log o_2$$

$$o_1 = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)}$$

$$o_2 = \frac{\exp(y_2)}{\exp(y_1) + \exp(y_2)}$$

$$y_1 = w_{11}h_1 + w_{21}h_2 + w_{31}h_3$$

$$y_2 = w_{12}h_1 + w_{22}h_2 + w_{32}h_3$$

$$\frac{\partial L}{\partial o_1} = -\frac{t_1}{o_1}$$

$$\frac{\partial L}{\partial o_2} = -\frac{t_2}{o_2}$$

$$\frac{\partial o_1}{\partial y_1} = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)} - \left(\frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)}\right)^2 = o_1(1 - o_1)$$

$$\frac{\partial o_2}{\partial y_1} = \frac{-\exp(y_2)\exp(y_1)}{(\exp(y_1) + \exp(y_2))^2} = -o_2 o_1$$

$$\frac{\partial y_1}{\partial w_{21}} = h_2$$

$$\begin{aligned}
\frac{\partial L}{\partial w_{21}} &= \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} \\
&= \frac{-t_1}{o_1}[o_1(1 - o_1)]h_2 + \frac{-t_2}{o_2}(-o_2 o_1)h_2 \\
&= h_2(t_2 o_1 - t_1 + t_1 o_1) \\
&= h_2(o_1(t_1 + t_2) - t_1) \\
&= h_2(o_1 - t_1)
\end{aligned}$$

O_1 contains y_1 and y_2 information

$t_1+t_2=1$, because the vector $\boldsymbol{t}=(t_1, t_2)$ is a one-hot vector

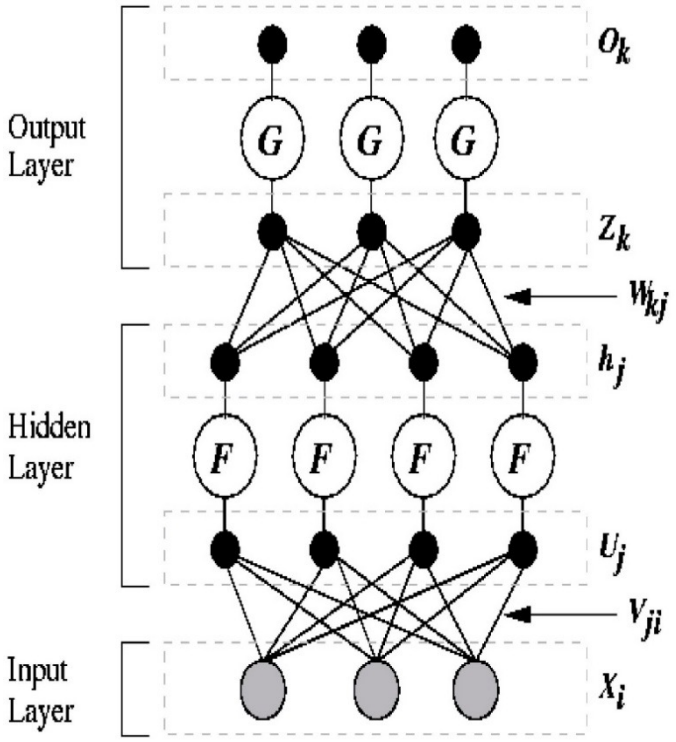
Please show $\frac{\partial L}{\partial w_{32}} = h_3(o_2 - t_2)$

Activation and Loss Function

	Outputs	
	Real Values	Probabilities
Output Activation	Linear	Softmax
Loss Function	Squared Error	Cross Entropy

- The activations used in the middle of hidden layers can be either sigmoid, rectified linear, hyperbolic tangent, etc

Gradient Descent Updates



- How often to update
  - After each training sample ($N=1$) or
  - After a randomly selected mini-batch of N training patterns (e.g., N=32, 64, 128, 256,

$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}} = w_{ki} - \eta \sum_{n=1}^{N} (o_k^{(n)} - t_k^{(n)}) o_k^{(n)} (1 - o_k^{(n)}) x_i^{(n)}$$

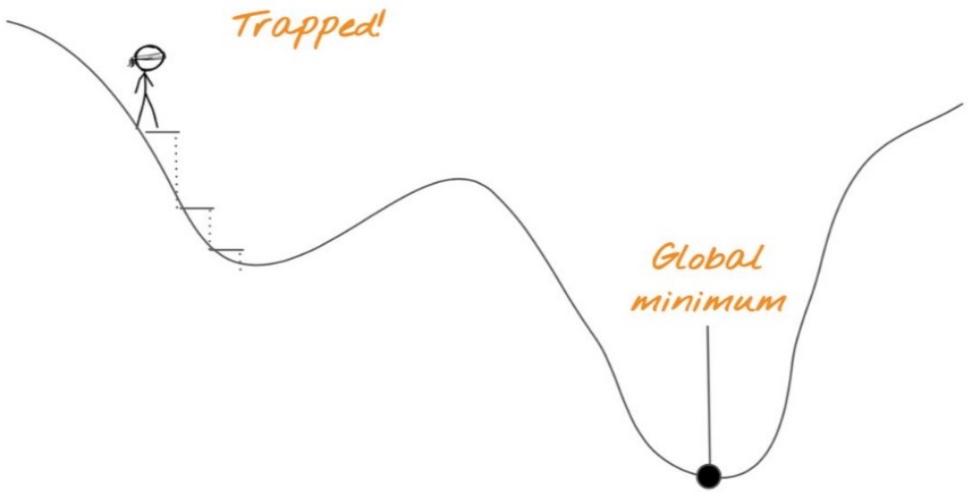
- How much to update
  - Use a fixed or an adaptive learning rate
  - Add a momentum term with leakage

$$V_t = \mu V_{t-1} - \alpha \nabla L_t(W_{t-1})$$
$$W_t = W_{t-1} + V_t$$

- $\alpha > 0$ – *learning rate* (typical choices: 0.01, 0.1)
- $\mu \in [0, 1)$ – *momentum* (typical choices: 0.9, 0.95, 0.99)

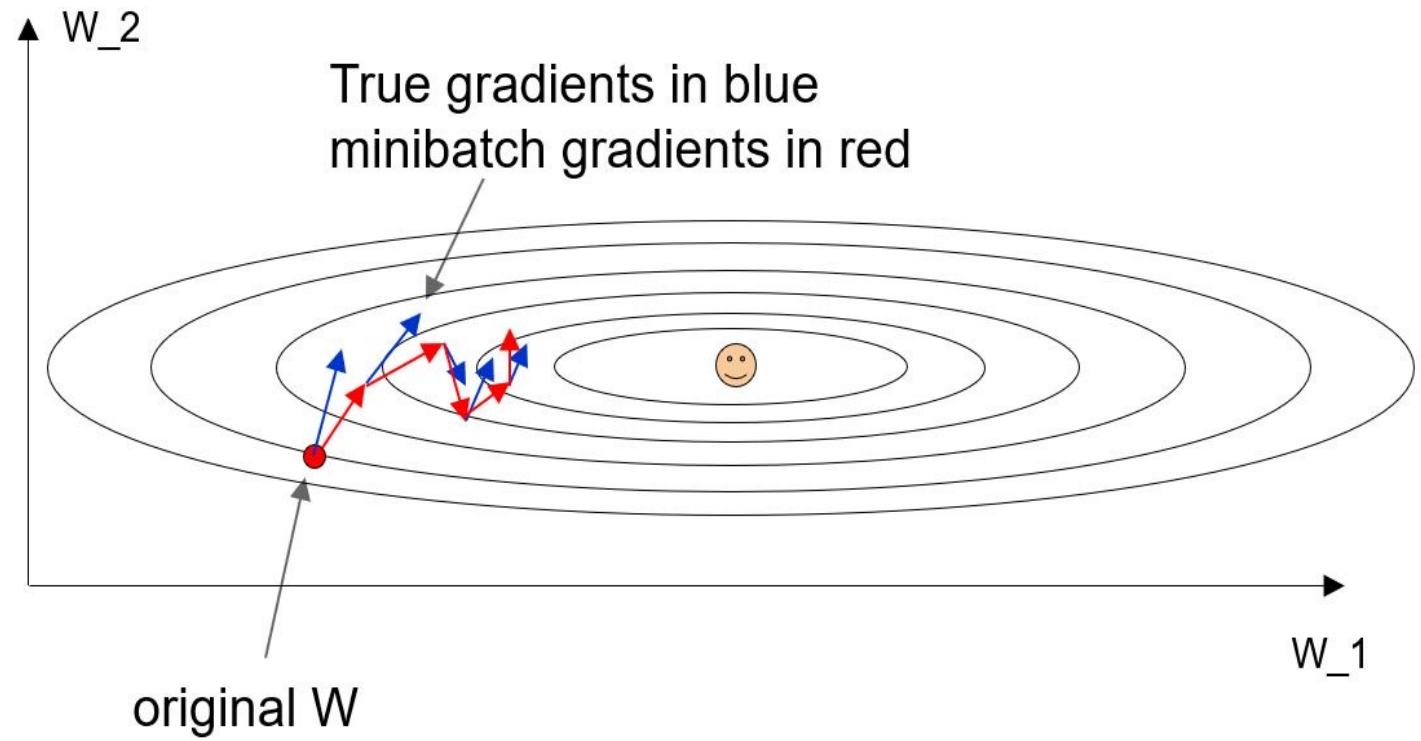
Momentum smooths updates, enhancing stability and speed.

Stochastic Gradient Descent



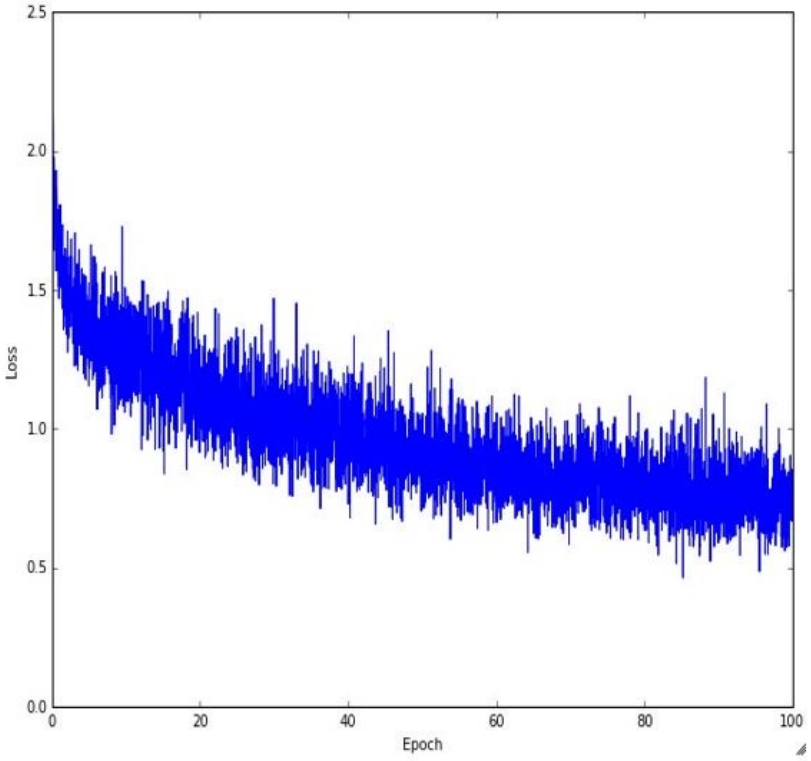
Stochastic gradient descent (SGD) can be regarded as a stochastic approximation (stochastic sampling of each mini-batch) of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a "randomly selected subset" of the data, mini-batch)

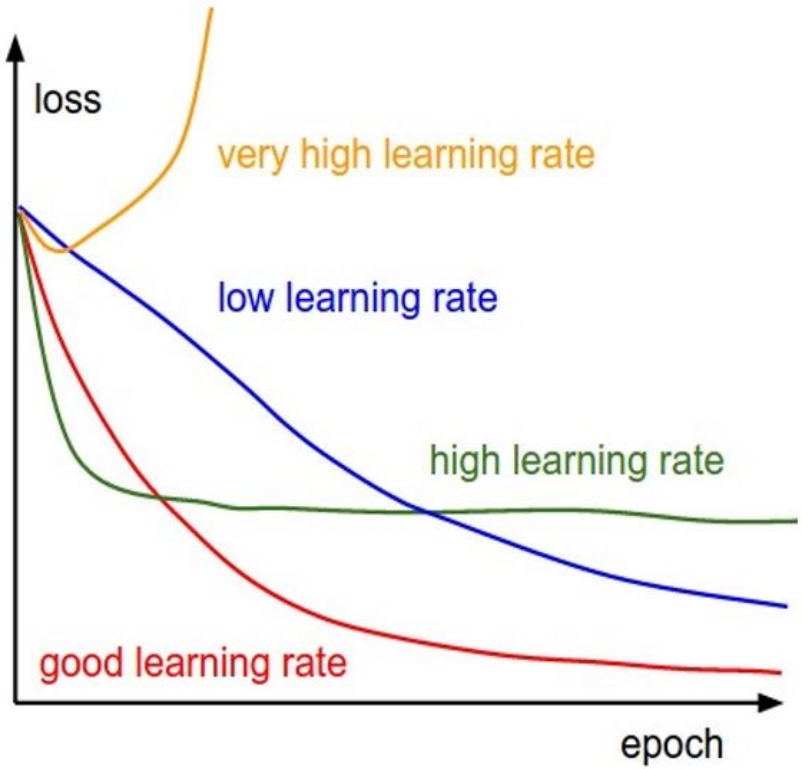
Mini Batch Gradients



Gradients are noisy but still make good progress on average

Choice of Learning Rate





More Variants of Updates

- Adaptive Gradient (AdaGrad)

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{\sum_{t'=1}^{t} \nabla L_{t'}(W_{t'-1})^2}}$$

- Root Mean Square Propagation (RMSProp)

$$\begin{aligned} R_t &= \gamma R_{t-1} + (1-\gamma)\nabla L_t(W_{t-1})^2 \\ W_t &= W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{R_t}} \end{aligned}$$

Adaptive Moment (Adam) Estimation Updates

- Combine the advantages of:
  - AdaGrad – works well with sparse gradients
  - RMSProp – works well in non-stationary settings
- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

$M_0 = \mathbf{0}, R_0 = \mathbf{0}$ (Initialization)

For $t = 1, \ldots, T$:

$M_t = \beta_1 M_{t-1} + (1-\beta_1)\nabla L_t(W_{t-1})$ (1st moment estimate)

$R_t = \beta_2 R_{t-1} + (1-\beta_2)\nabla L_t(W_{t-1})^2$ (2nd moment estimate)

$\hat{M}_t = M_t / (1-(\beta_1)^t)$ (1st moment bias correction)

$\hat{R}_t = R_t / (1-(\beta_2)^t)$ (2nd moment bias correction)

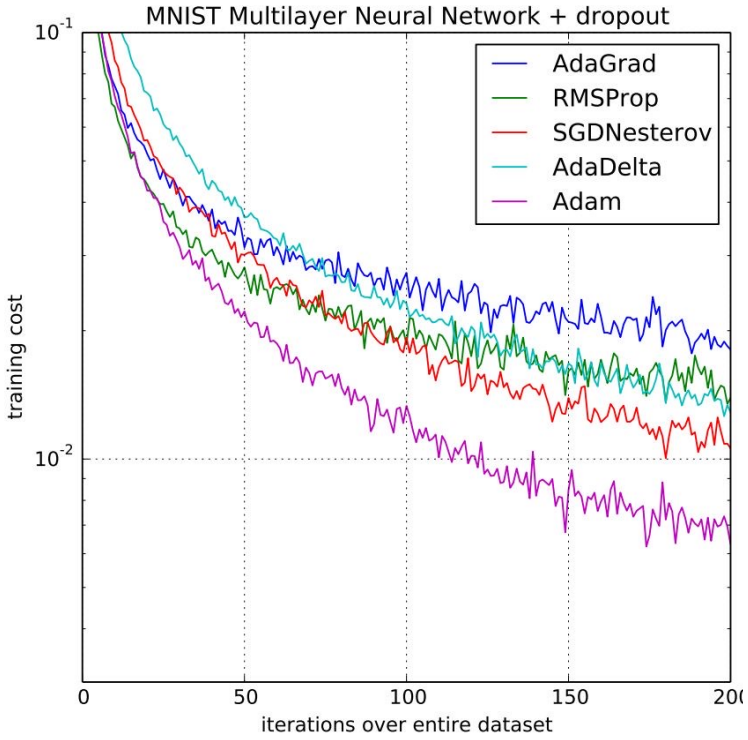
$W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t + \epsilon}}$ (Update)

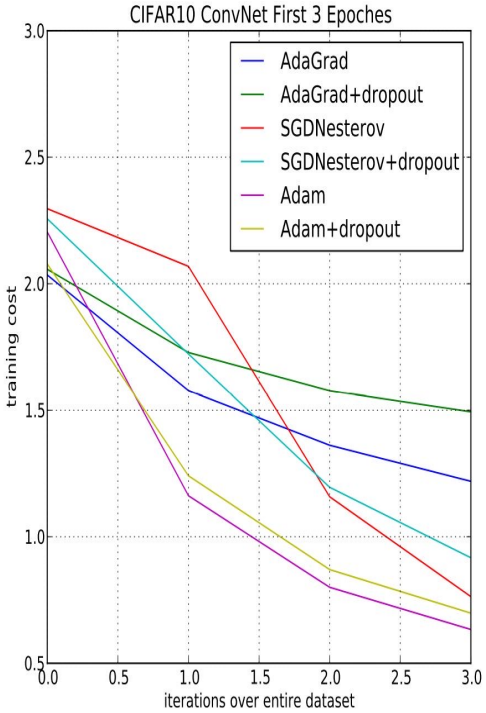
Return W_T

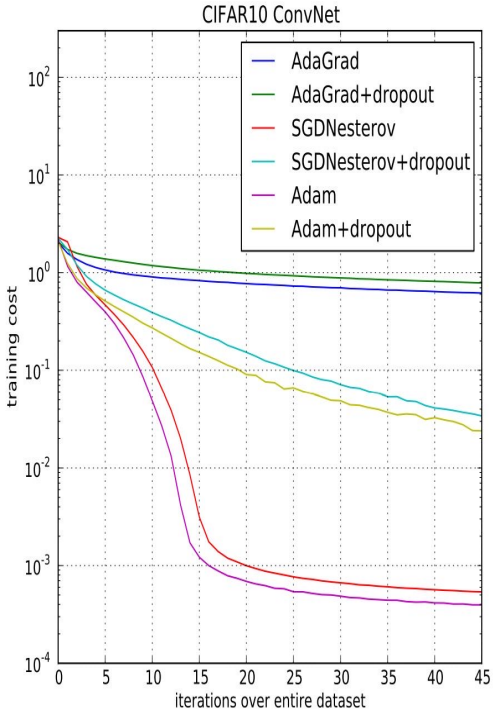
Hyper-parameters:

- $\alpha > 0$ – learning rate (typical choice: 0.001)
- $\beta_1 \in [0,1)$ – 1st moment decay rate (typical choice: 0.9)
- $\beta_2 \in [0,1)$ – 2nd moment decay rate (typical choice: 0.999)
- $\epsilon > 0$ – numerical term (typical choice: 10^{-8})

Adam: A Method for Stochastic Optimization







Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization," ICLR 2015, https://arxiv.org/abs/1412.6980