

Overview of Modern ML Applications: Convolutional Neural Network (CNN)

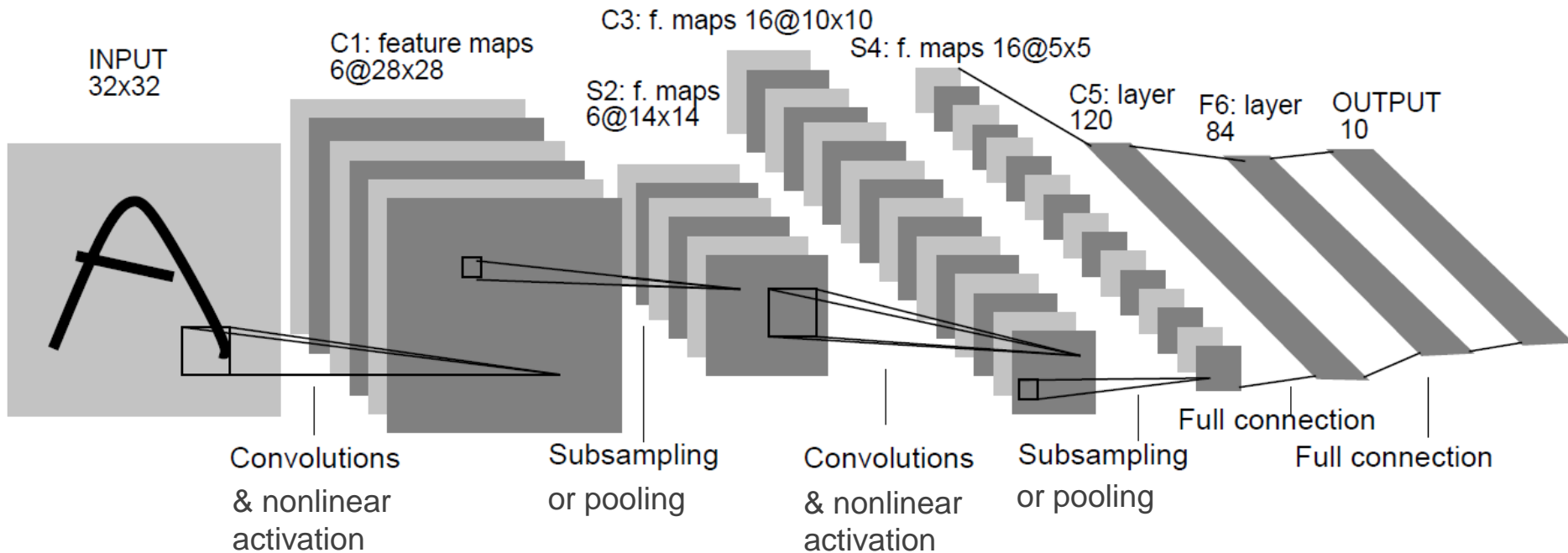
Learning objectives

- Describe the structure of CNN
- Build and train simple CNNs using a deep learning package

(Ref: Ch 9 of [Goodfellow et al. 2016](#))

Convolutional Neural Network (CNN)

The **single** most important technology that fueled the rapid development of **deep learning** and **big data** in the past decade.



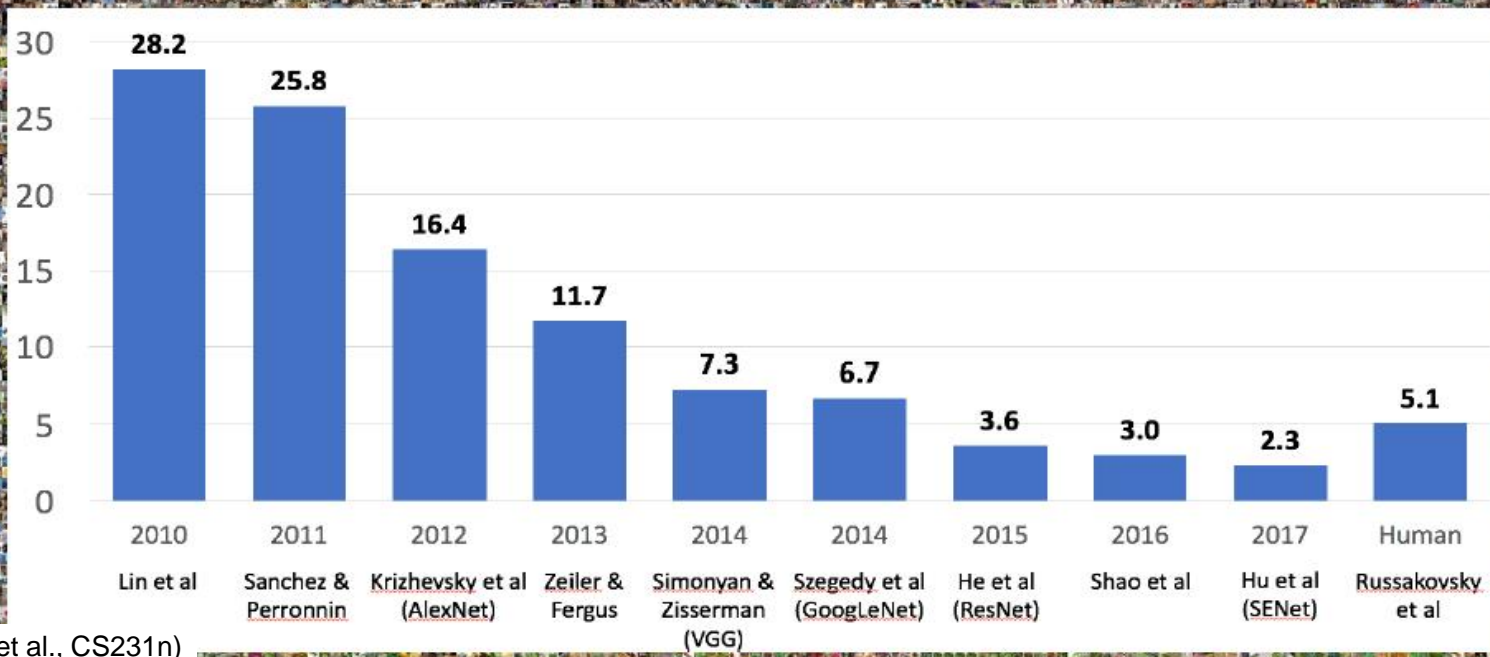
Why is Deep Learning so Successful?

- 1. Improved model:** convolutional layer, more layers (“deep”), simpler activation (i.e., ReLU), skip/residual connection (i.e., ResNet), attention (i.e., Transformer)
 - 2. Big data:** huge dataset, transfer learning
 - 3. Powerful computation:** graphical processing units (GPUs)
- ◆ Example of big data: ImageNet (22K categories, 15M images)



IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:
1,000 object classes
1,431,167 images



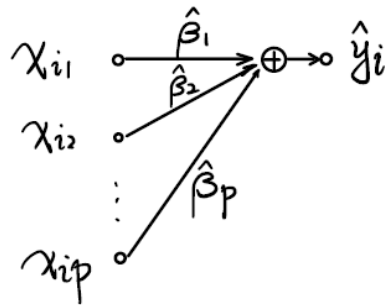
Linear Model to Neural Network

Recall linear model w/ multiple predictors / features / inputs.

$$\underbrace{y_i}_{\text{true output}} = \sum_{j=1}^p x_{ij} \beta_j + e_i = \underbrace{[\beta_1, \dots, \beta_p]}_{\text{true weights}} \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix} + e_i, \quad i=1, \dots, n.$$

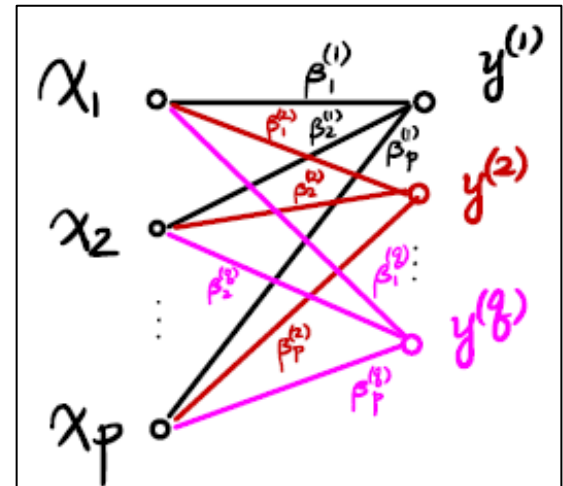
$$\underbrace{\hat{y}_i}_{\text{predicted output}} = \sum_{j=1}^p x_{ij} \hat{\beta}_j = \underbrace{[\hat{\beta}_1, \dots, \hat{\beta}_p]}_{\text{estimated weights}} \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix}, \quad i=n+1, \dots, n+m.$$

Graphically we have:



① Use multiple linear models

② Simplify the notations.



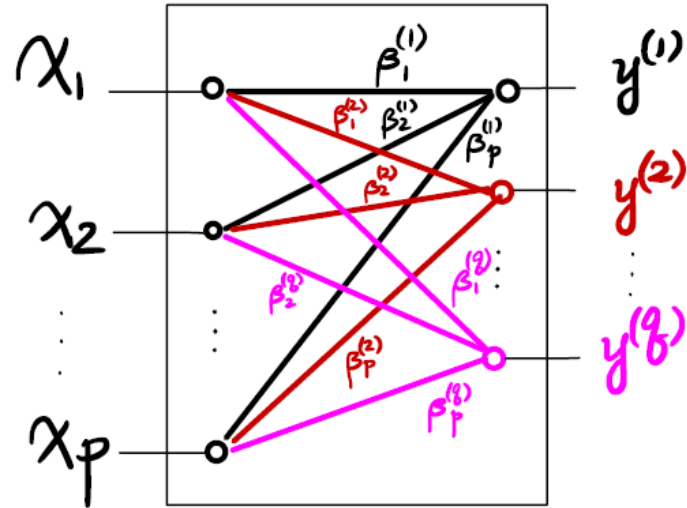
Fully-Connected Layer for 1D Signal

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(g)} \end{bmatrix} = \begin{bmatrix} \beta_1^{(1)} & \beta_2^{(1)} & \dots & \beta_p^{(1)} \\ \beta_1^{(2)} & \beta_2^{(2)} & \dots & \beta_p^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1^{(g)} & \beta_2^{(g)} & \dots & \beta_p^{(g)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$$

layer output, $y \in \mathbb{R}^g$

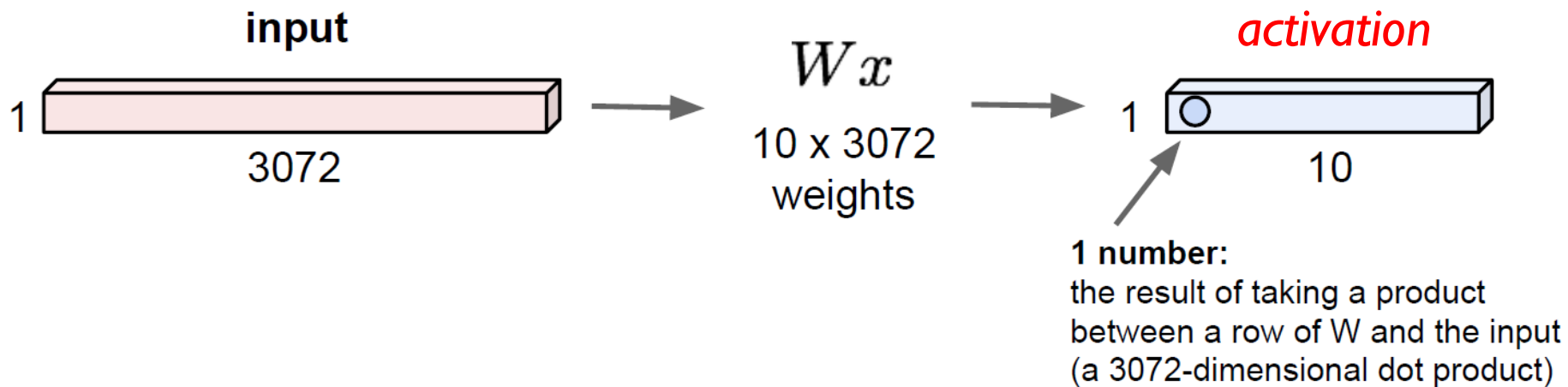
dense weight matrix
 $B \in \mathbb{R}^{g \times p}$

layer input, $x \in \mathbb{R}^p$



Fully-Connected Layer for RGB Image

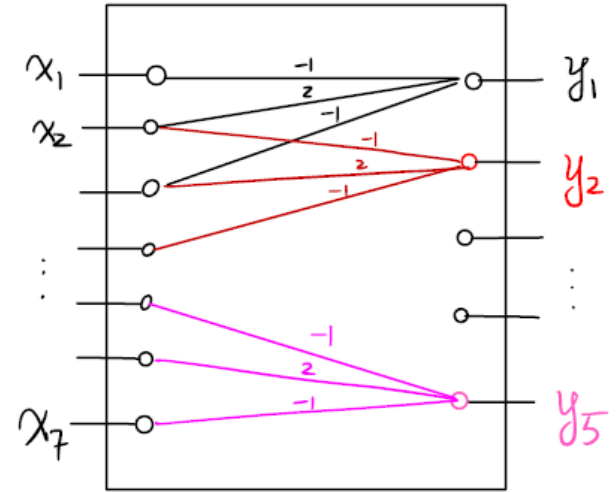
32x32x3 image -> stretch to 3072 x 1



Convolutional Layer for 1D Signal

$$\begin{bmatrix} y_1 \\ \vdots \\ y_5 \end{bmatrix} = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ \vdots & & \vdots \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_7 \end{bmatrix}$$

Sparse weight matrix



Input

x_1	x_2	x_3	x_4	x_5	x_6	x_7
-------	-------	-------	-------	-------	-------	-------

 length 7

Convolution/
filter mask

	*	
-1	2	-1

 → length 3

Output

y_1	y_2	y_3	y_4	y_5
-------	-------	-------	-------	-------

 length $7 - (3 - 1) = 5$
(w/o boundary elements)

Convolutional Layer for 2D Matrix/Image

x_{11}			x_{15}
x_{21}			
x_{51}			x_{55}

Input image

 $*$

$1/4$	$1/4$
$1/4$	$1/4$

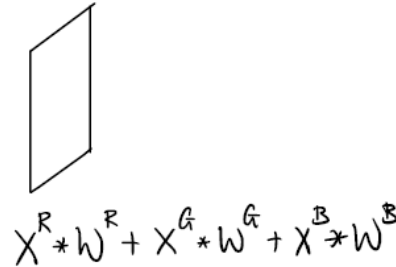
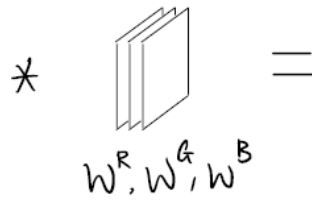
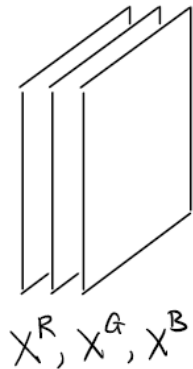
 $=$

filter mask

y_{11}			y_{14}
y_{21}			
y_{41}			y_{44}

Activation map

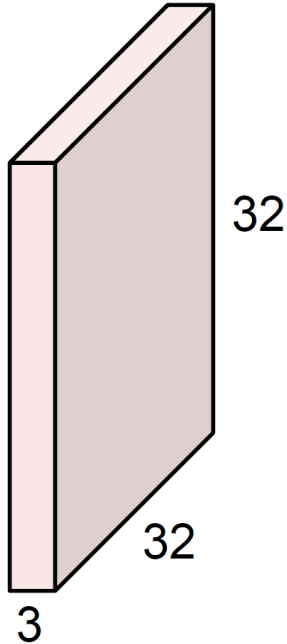
2D Convolution



Multiple color channels need multiple filter masks

Convolutional Layer for RGB Image

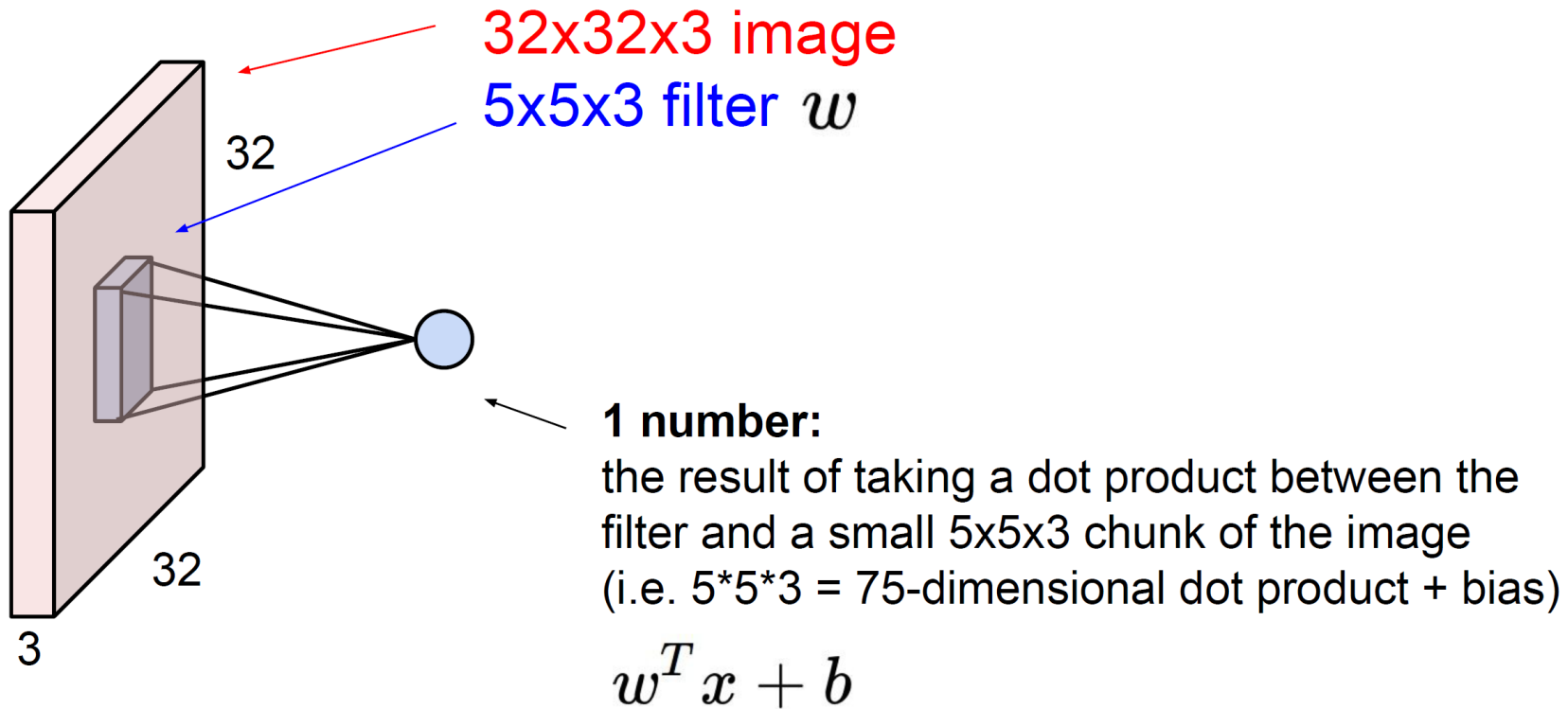
32x32x3 image



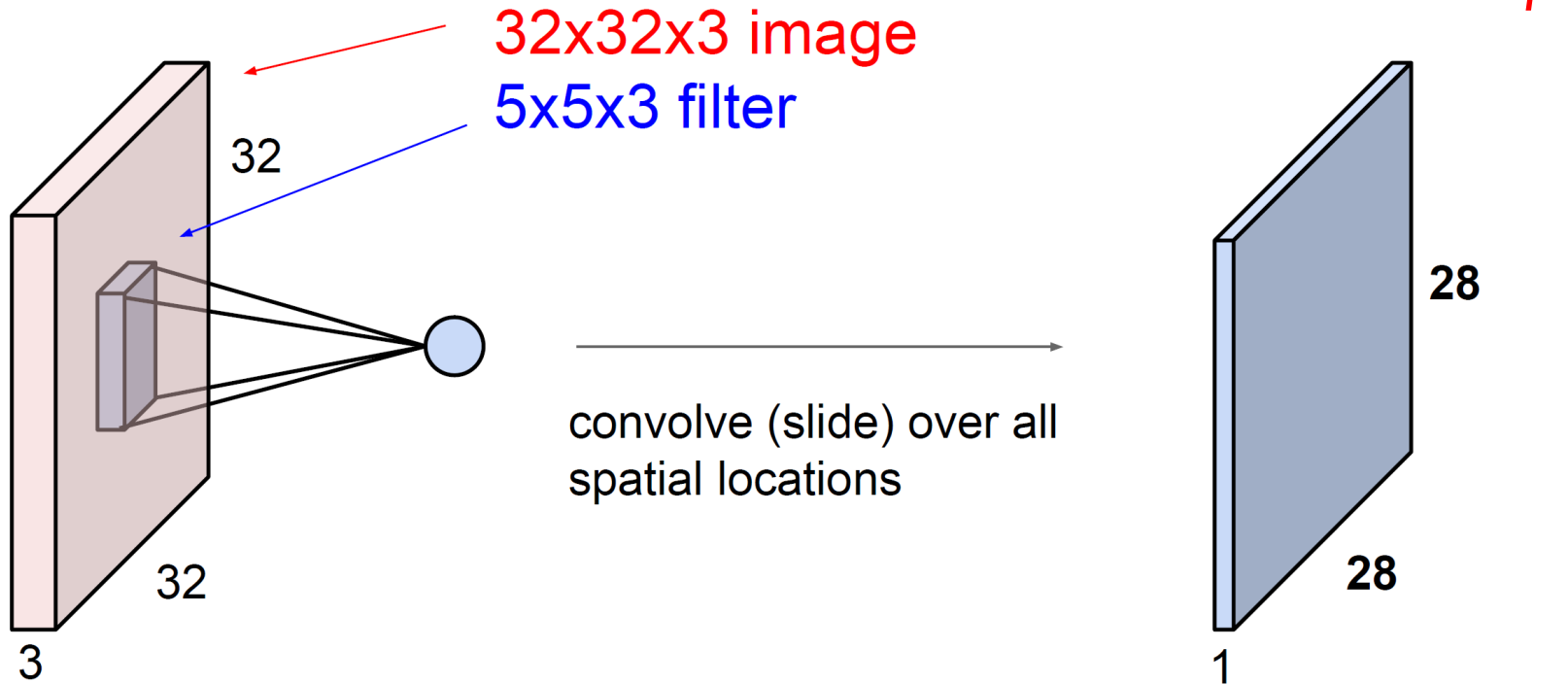
5x5x3 filter



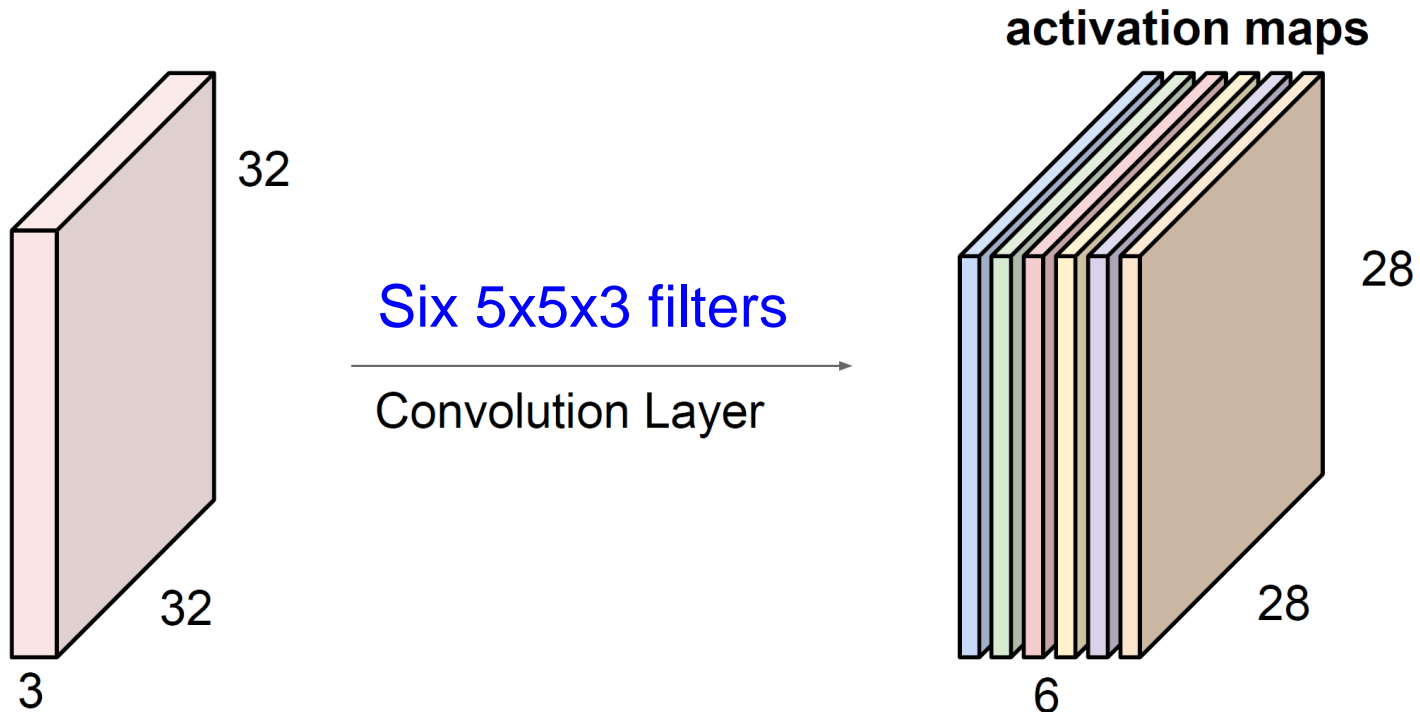
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”



A closer look at spatial dimensions:

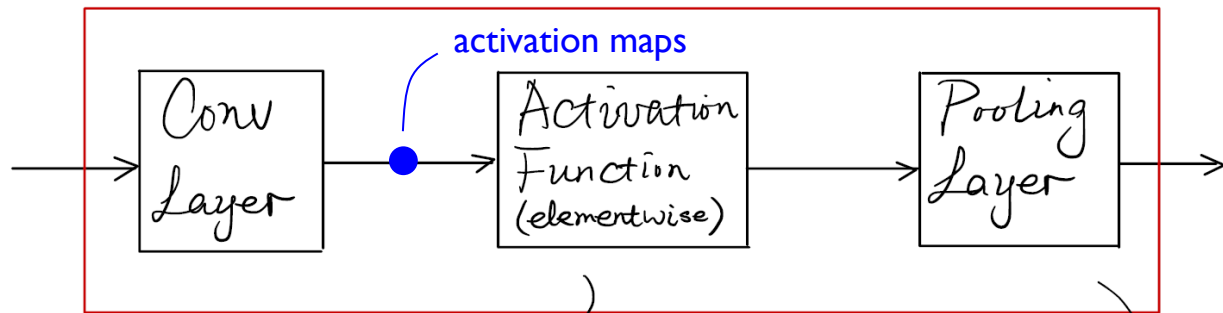


For example, if we had six 5x5 filters, we'll get six separate *activation maps*:

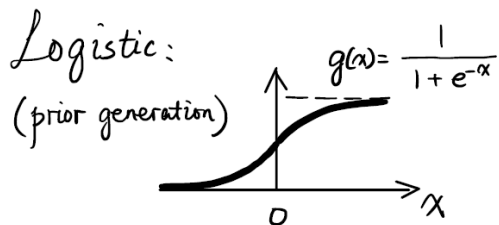
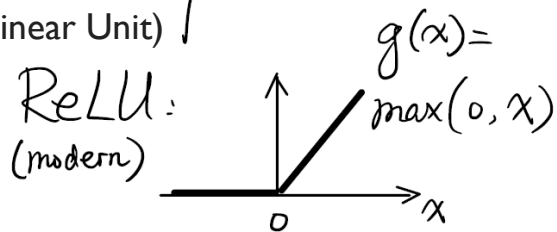


We stack these up to get a “new image” of size 28x28x6!

Building Block for Modern CNN



(Rectified Linear Unit)



Ex: $\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} : \{x_{ij}\}_{i,j=1}^2$

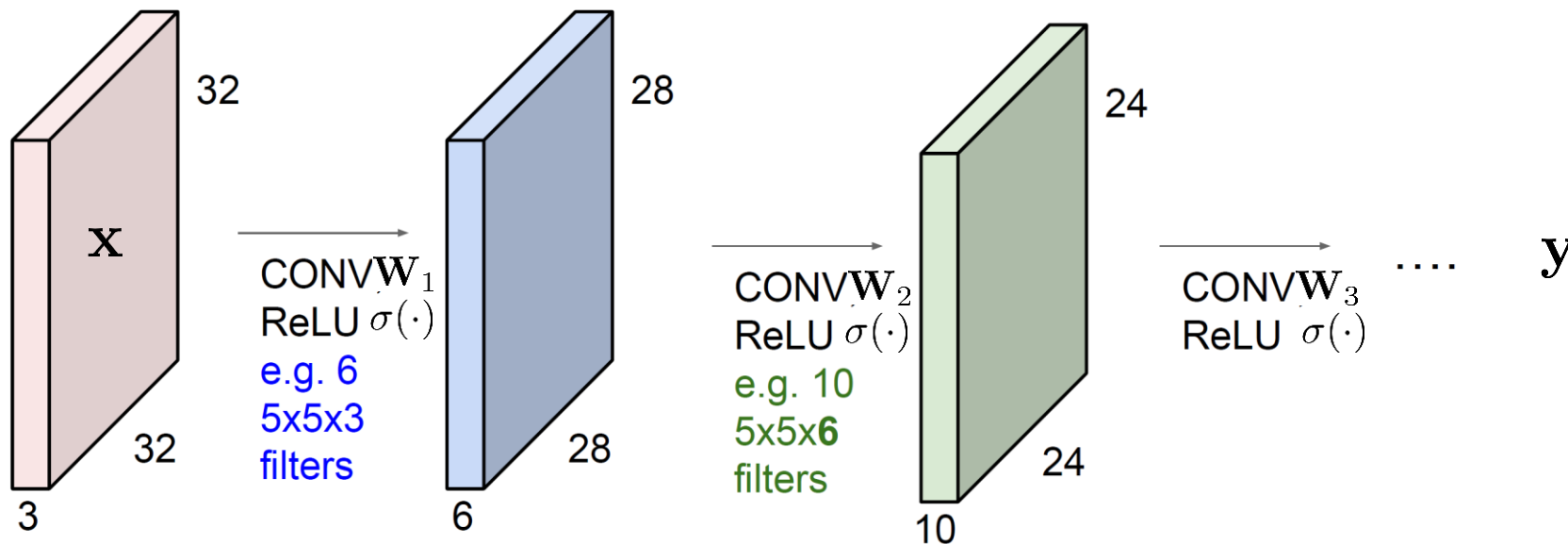
Max pooling:

$$g(\{x_{ij}\}) = \max(\{x_{ij}\})$$

Average pooling:

$$g(\{x_{ij}\}) = \frac{1}{|\{x_{ij}\}|} \sum_{i,j} x_{ij}$$

CNN is composed of a sequence of convolutional layers, interspersed with activation functions (ReLU, in most cases).

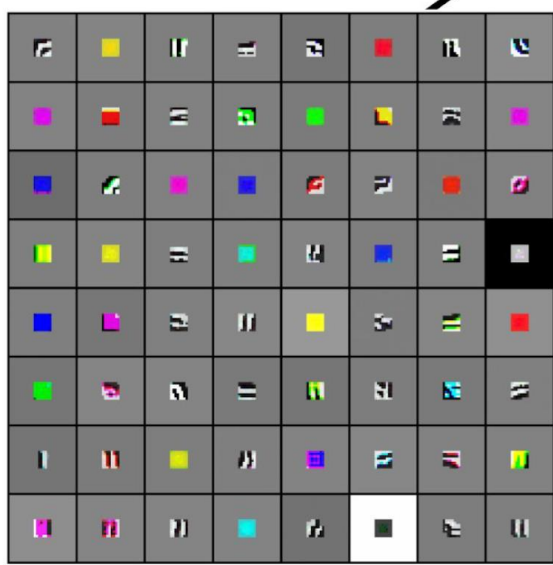


$$\mathbf{y} = \cdots \sigma \left(\mathbf{W}_3 \sigma \left(\mathbf{W}_2 \sigma \left(\mathbf{W}_1 \mathbf{x} \right) \right) \right) \cdots$$

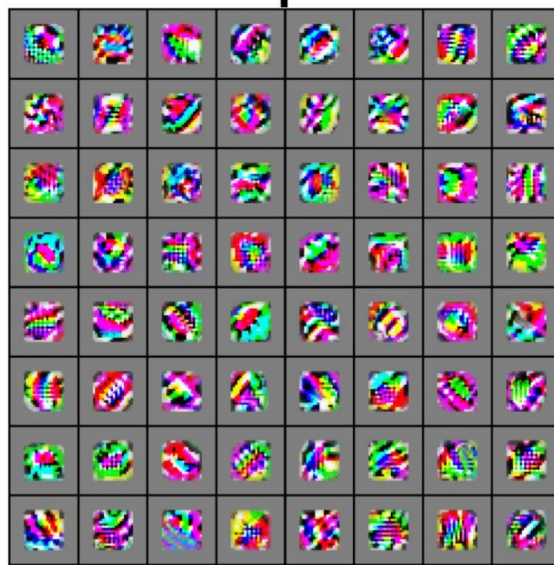
Source of nonlinearity, ReLU: $\sigma(x) := \max(0, x)$

[Zeiler and Fergus 2013]

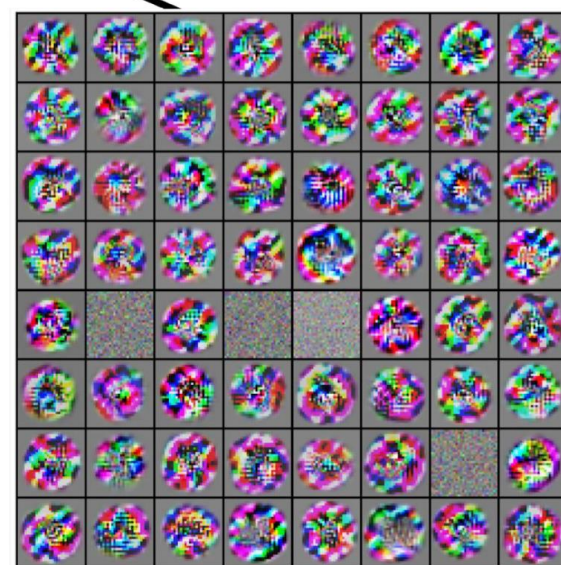
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



VGG-16 Conv1_1



VGG-16 Conv3_2

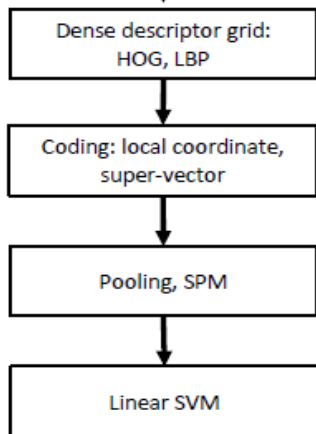


VGG-16 Conv5_3

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC



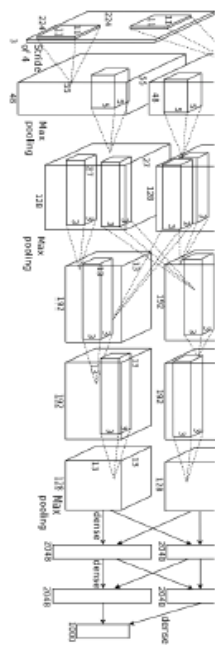
[Lin CVPR 2011]

Lion image by Swissfrog is licensed under [CC BY 3.0](https://creativecommons.org/licenses/by/3.0/)

AlexNet

Year 2012

SuperVision

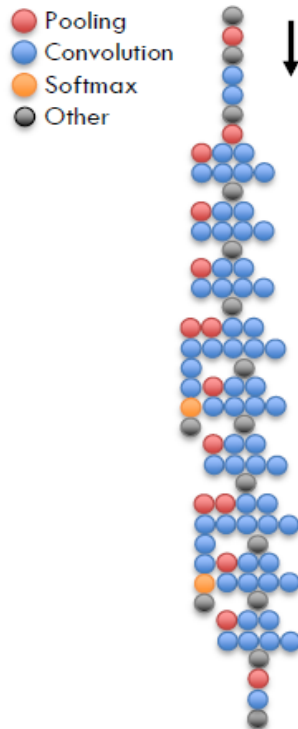


[Krizhevsky NIPS 2012]

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

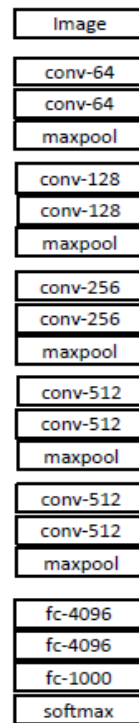
Year 2014

GoogLeNet



[Szegedy arxiv 2014]

VGG

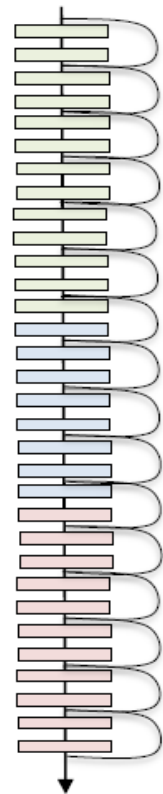


[Simonyan arxiv 2014]

ResNet

Year 2015

MSRA

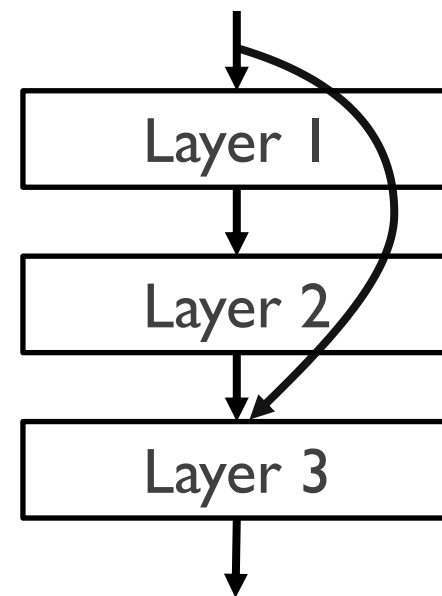


[He ICCV 2015]

(Fei-Fei Li et al., CS231n)

Residual Neural Network (ResNet) (Kaiming He et al., 2015)

- ◆ **Skip connections** or **shortcuts** are added.
- ◆ They can
 - ★ avoid “vanishing gradients”, and
 - ★ make optimization landscape flatter.
- ◆ From Taylor expansion perspective, the neural network only learns the higher-order error terms beyond the linear term \mathbf{x} .
- ◆ Has interpretations in PDE.
- ◆ Preferred modern NN structure.



$$y = \cdots \sigma \left(\mathbf{W}_3 \underbrace{[\mathbf{x} + \sigma(\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x}))]}_{g(\mathbf{x})} \right) \cdots$$

When Output is Categorical / Qualitative

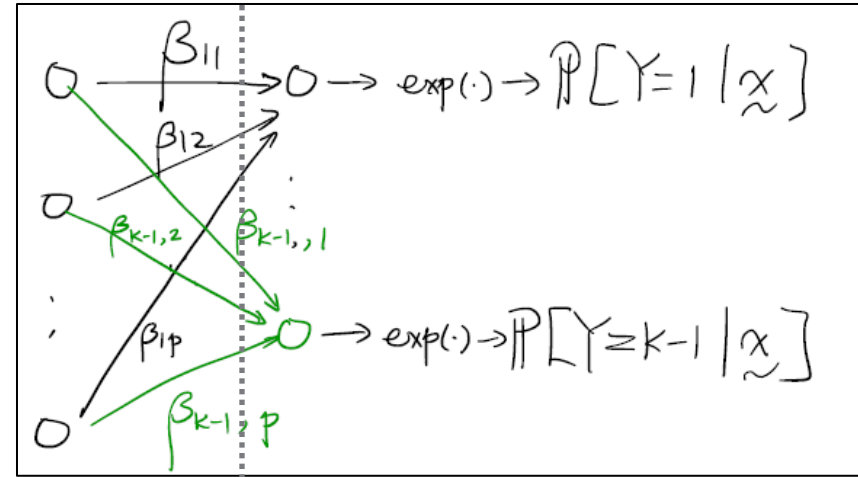
◆ A **softmax layer** is needed:

◆ Softmax function:

$$\sigma_i(\underline{z}) = \frac{e^{\beta z_i}}{\sum_{j=1}^k e^{\beta z_j}}$$

◆ Ex:

$$\begin{aligned}
 K=2 \quad \sigma_1 &= \frac{e^{\beta z_1}}{e^{\beta z_1} + e^{\beta z_2}} \\
 &= \frac{1}{1 + e^{\beta(z_2 - z_1)}}
 \end{aligned}$$



conv/fully connected ← → softmax layer

When β very large,

$z_2 > z_1$ leads to $\begin{cases} \sigma_1 = 0 \\ \sigma_2 = 1 \end{cases}$

Winner takes all!

Other Essential Aspects of CNN

- ◆ Due to time constraints, this overview lecture covered only the structural elements of CNNs. Other essential aspects are:
 - ✦ Cost function/loss, e.g., MSE, cross entropy.
 - ✦ How to train CNNs or estimate the weights (will only give practice code), i.e., *backpropagation* (will cover in the next two lectures).
 - ✦ Practical training considerations including
 - How to determine *number of hidden units/channels* to be used,
 - How to tune *learning rate* and *batch size*, and
 - When to stop training (number of *epochs*).
- ◆ For a more complete treatment on CNN, refer to the dedicate courses such as CS231n CNNs for Visual Recognition or ECE 542/492.

Machine Learning Curriculum

◆ Follow-up machine learning courses:

- ECE 542/492 Neural Networks
- ECE 592-103 Large-Scale ML & Optimization
- ECE 592/792 Adv.Topics ML/DL
- ECE 512 Data Science
- ECE 558 Image Processing
- ECE 759 Pattern Recognition
- ECE 763 Computer Vision
- Any courses/videos on YouTube, Coursera

◆ Top publication venues (theory & application)

- ✦ Machine learning: ICML, NeurIPS, ICLR
- ✦ Computer vision: CVPR, ICCV, ECCV

◆ Data science competitions: [kaggle.com](https://www.kaggle.com)

Neural Network Training: Backpropagation

Acknowledgment: Some graphics and slides were adapted from Profs. Jain (MSU), Min Wu (UMD), Fei-Fei Li (Stanford)

Some figures are from Duda-Hart-Stork textbook, Fei-Fei Li's slides

3-Layer Neural Network Structure

- ◆ A single “bias unit” is connected to each unit in addition to the input units

- ◆ Net activation:
$$net_j = \sum_{i=1}^d x_i w_{ji} + w_{j0} = \sum_{i=0}^d x_i w_{ji} \equiv \mathbf{w}_j^T \mathbf{x},$$

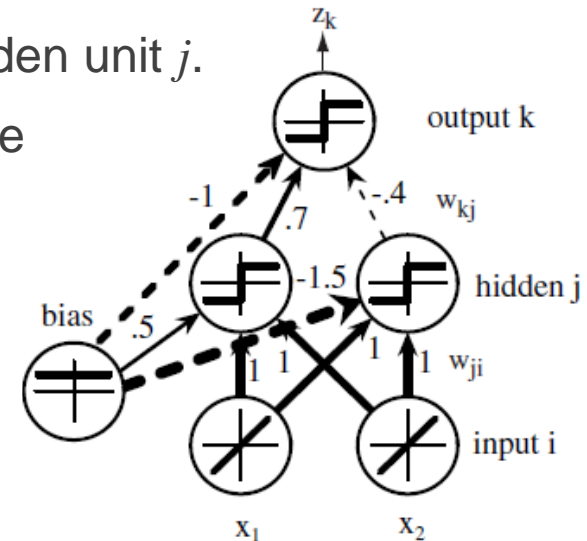
where the subscript i indexes units in the input layer, j indexes units in the hidden layer;

w_{ji} denotes the input-to-hidden layer weights at hidden unit j .

- ◆ In neurobiology, such weights or connections are called “synapses”

- ◆ Each hidden unit emits an output that is a nonlinear function of its activation

$$y_j = \sigma(net_j)$$



Training Neural Networks / Estimating Weights

- Notations: $t_k \sim$ the k th target (or desired) output,
 $z_k \sim$ the k th estimated/computed output with $k = 1, \dots, c$.
 $w_{ij} \sim$ weight of the network

- Squared cost func:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^c (t_k - z_k)^2 = \frac{1}{2} \|\mathbf{t} - \mathbf{z}\|^2$$

$y_k \quad \hat{y}_k$

- Learning based on gradient descent by iteratively updating the weights:

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta \mathbf{w}(m), \quad m = 1, 2, \dots$$

- The weights are initialized with random values,
and updated in a direction to reduce the error.

$$\Delta \mathbf{w} = -\eta \cdot \nabla_{\mathbf{w}} J$$

- **Learning rate**, η , controls the step size of the update in weights.



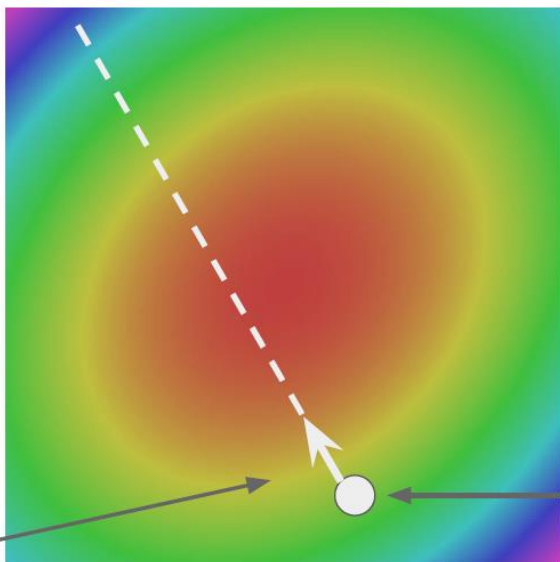
Walking man image is CC0 1.0 public domain

Figure source: Stanford CS231n by Fei-Fei Li

Gradient Descent

Topographic map, color means "height"

W_2



original W

negative gradient direction $-\nabla_{\mathbf{w}} J(\mathbf{w})$

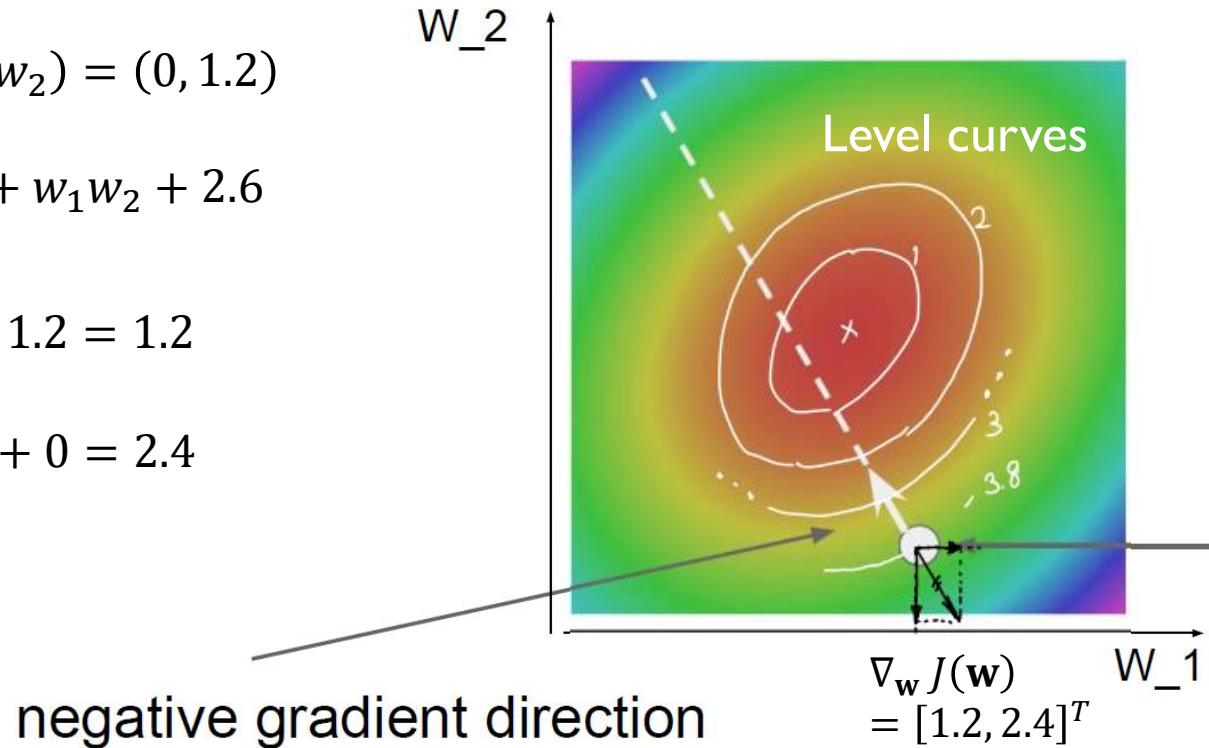
Gradient Descent

Example: Gradient at $(w_1, w_2) = (0, 1.2)$

$$J(w_1, w_2) = w_1^2 + w_2^2 + w_1w_2 + 2.6$$

$$\frac{\partial J}{\partial w_1} = 2w_1 + w_2 = 2 \times 0 + 1.2 = 1.2$$

$$\frac{\partial J}{\partial w_2} = 2w_2 + w_1 = 2 \times 1.2 + 0 = 2.4$$



Efficient Gradient Calculation: Backpropagation

- ◆ Computes $\partial J / \partial w_{ji}$ for a single input-output pair.
- ◆ Exploit the chain rule for differentiation, e.g.,

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

- ◆ Computed by **forward** and **backward** sweeps over the network, keeping track only of quantities local to each unit.
- ◆ Iterate backward one unit at a time from last layer. Backpropagation avoids redundant calculations.

Backpropagation (BP): Simple Example

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

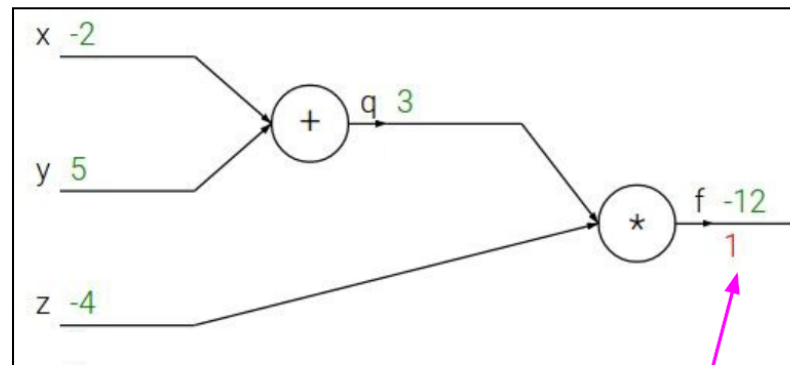
e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Descent step:
$$\begin{bmatrix} x(m+1) \\ y(m+1) \\ z(m+1) \end{bmatrix} = \begin{bmatrix} x(m) \\ y(m) \\ z(m) \end{bmatrix} - \eta \begin{bmatrix} -4 \\ -4 \\ 3 \end{bmatrix}$$



Backpropagation (BP): More Complicated Example

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$

$$\frac{\partial f}{\partial w_i} = ?, \quad i = 0, 1, 2.$$

$$w_0 = 2$$

$$x_0 = -1$$

$$w_1 = -3$$

$$x_1 = -2$$

$$w_2 = -3$$

$$J = \|\mathbf{t} - \mathbf{z}\|^2 / 2 \quad t_k \dots \dots \dots z_k = \sigma(\text{net}_k)$$

Network Learning by BP (cont'd)

- ★ Error on hidden-to-output weight:

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial \text{net}_k} \cdot \frac{\partial \text{net}_k}{\partial w_{kj}} = -\delta_k \cdot y_j$$

- ★ δ_k , the sensitivity of unit k :
describes how the overall loss value changes w/ the activation of the unit's net activation

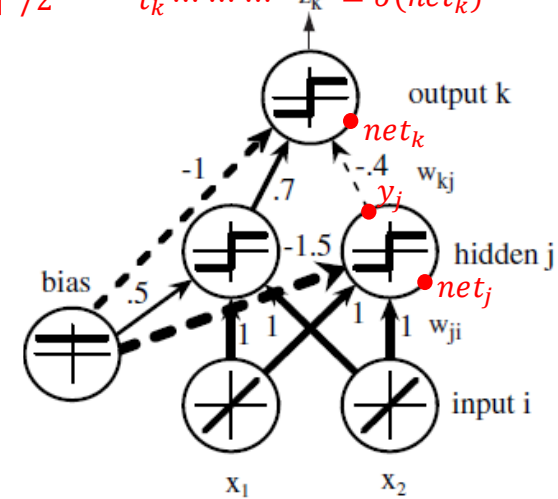
$$\delta_k \equiv -\frac{\partial J}{\partial \text{net}_k}$$

$$\delta_k = -\frac{\partial J}{\partial \text{net}_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial \text{net}_k} = (t_k - z_k) \sigma'(\text{net}_k)$$

- ★ Since $\text{net}_k = \mathbf{w}_k^T \mathbf{y}$, we have $\frac{\partial \text{net}_k}{\partial w_{kj}} = y_j$

- ★ **Summary I:** weight update (or learning rule) for the hidden-to-output weight is:

$$\Delta w_{kj} = -\eta \frac{\partial J}{\partial w_{kj}} = \eta (t_k - z_k) \sigma'(\text{net}_k) y_j = \eta \delta_k y_j$$



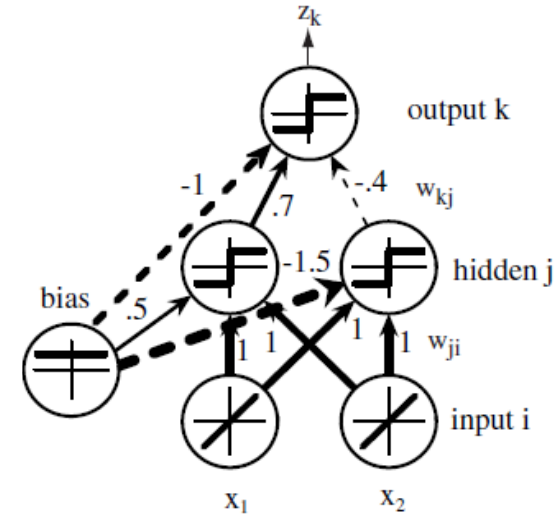
Network Learning by BP (cont'd)

★ Error on input-to-hidden weight:

- chain rule:
$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

- $$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j} \left[\frac{1}{2} \sum_{k=1}^c (t_k - z_k)^2 \right] = - \sum_{k=1}^c (t_k - z_k) \frac{\partial z_k}{\partial y_j}$$

$$= - \sum_{k=1}^c (t_k - z_k) \frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} = - \sum_{k=1}^c (t_k - z_k) \sigma'(net_k) w_{kj}$$



★ Sensitivity of a hidden unit:

(Similarly defined as earlier)

$$\delta_j \equiv \frac{\partial J}{\partial net_j} = \sigma'(net_j) \sum_{k=1}^c w_{kj} \delta_k$$

★ **Summary 2:** Learning rule for the input-to-hidden weight is:

$$\Delta w_{ji} = \eta \underbrace{\left[\sigma'(net_j) \sum_k w_{kj} \delta_k \right]}_{\delta_j} x_i = \eta \delta_j x_i$$

Sensitivity at Hidden Node

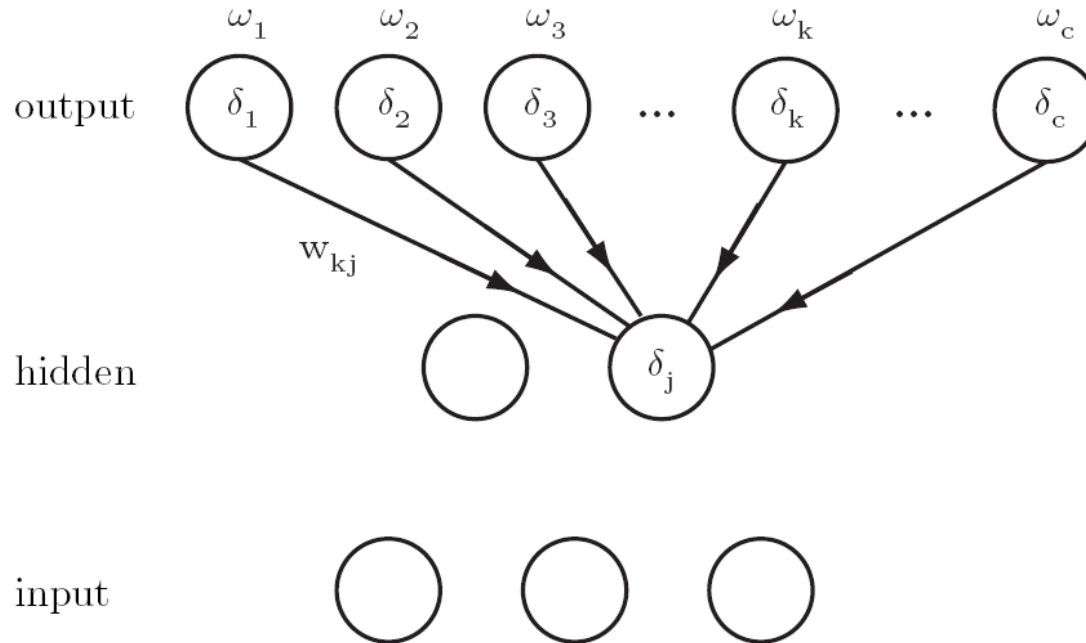


Figure 6.5: The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units: $\delta_j = \sigma'(net_j) \sum_{k=1}^c w_{kj} \delta_k$. The output unit sensitivities are thus propagated “back” to the hidden units.

BP Algorithm: Training Protocols

1. **Gradient descent:** Use **all** training examples to update weights.
 2. **Stochastic gradient descent (SGD):** Use **one** randomly selected training example to iteratively update weights.
 3. **Mini-batch gradient descent:** Separate training examples into disjoint **mini-batches**. Use mini-batches to update weights. One “epoch” = cycling through all mini-batches.
- ◆ Making several complete passes of the training data helps the network to learn.
 - ◆ Large language models (LLMs) may use less than 1 epoch due to huge dataset size.



BP Algorithm: Stopping Criteria

- ◆ Basic idea: Terminating training when the value or the change of value of the following quantities is below some preset threshold:
 - ✦ Loss function $J(\mathbf{w})$,
 - ✦ Gradient norm $\|\nabla_{\mathbf{w}} J\|$.
- ◆ Modern deep learning stopping criteria:
 - ✦ Lowest validation error,
 - ✦ For large models, training as many iterations as resources is available.

Learning Curves

- ★ Before training starts, the training error is high; as the learning proceeds, **training error becomes smaller**
- ★ Error per epoch depends on the amount of training data and expressive power (e.g., # of weights) of the network
- ★ Average error on an independent validation/test set is always higher than on the training set, and it can **decrease or increase**
- ★ A validation set could be used for **model selection**. It can avoid overfitting and can ensure the generalization capability of the learned network: “Select the model whose error on the validation set is minimum”
- ★ A test set is for **performance reporting ONLY**.

loss function value J

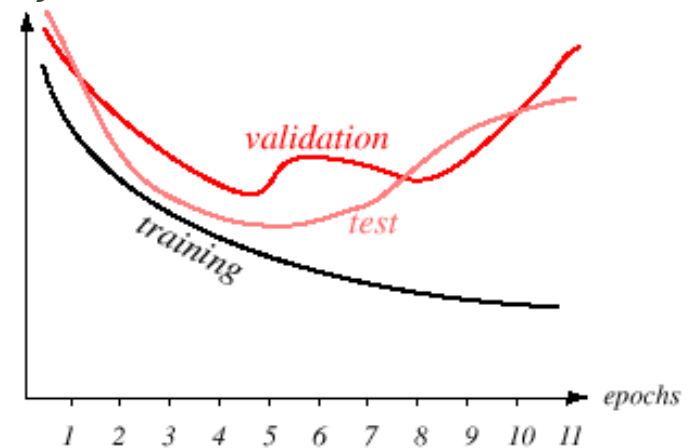


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^n J_p$. The validation error and the test or generalization error per pattern are virtually always higher than the training error. In some protocols, training is stopped at the first minimum of the validation set. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Practical Considerations: Learning Rate

◆ Learning Rate

- ★ Small learning rate: slow convergence
- ★ Large learning rate: high oscillation and slow convergence

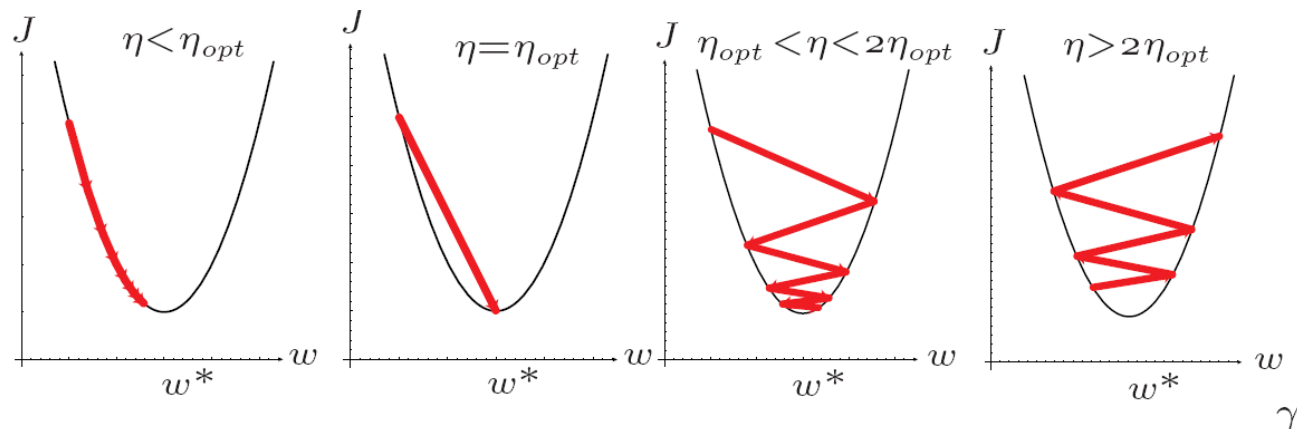
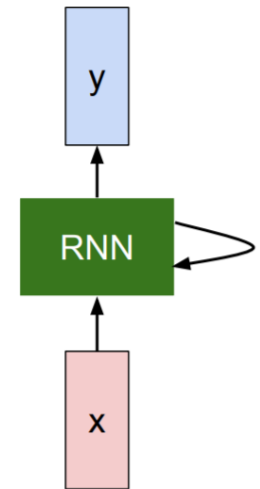


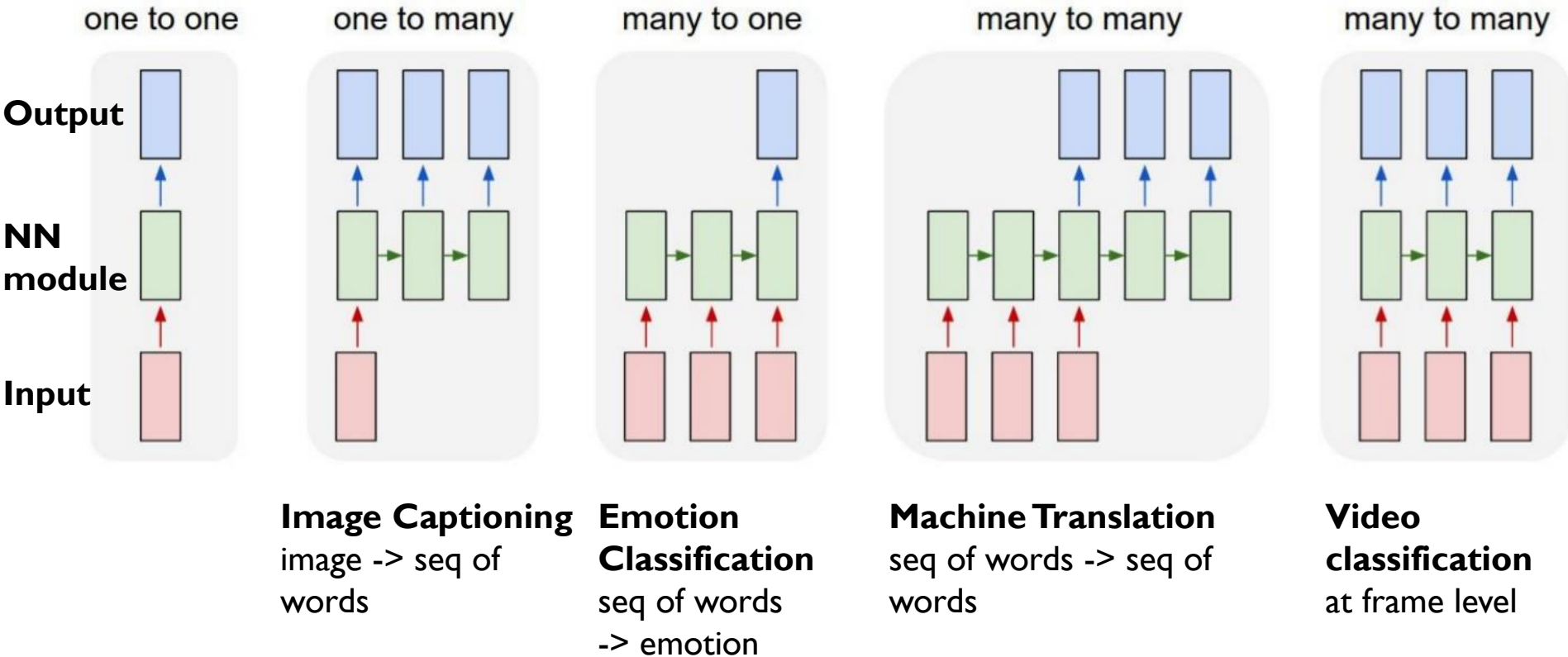
Figure 6.18: Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges.

Overview of Modern ML Applications: Recurrent Neural Network (RNN) and LSTM



A Sequence of **Identical** Neural Network Modules

Force the neural nets (in green) to be the same to lower the complexity!



Recurrent Neural Network (RNN): Definition

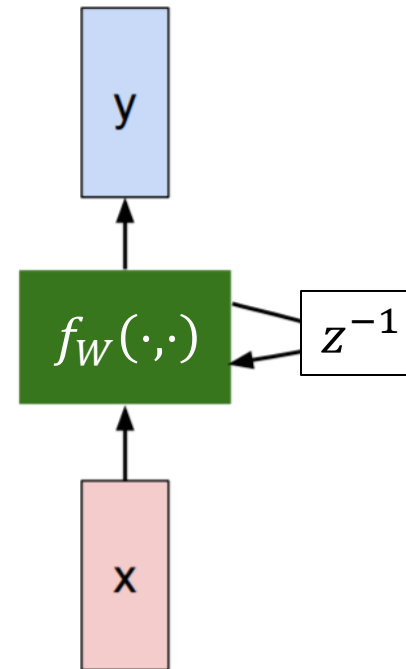
We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t) \quad t = 1, 2, \dots$$

h_t : state, x_t : input

f_W : neural network

$$y_t = W_{hy} h_t$$



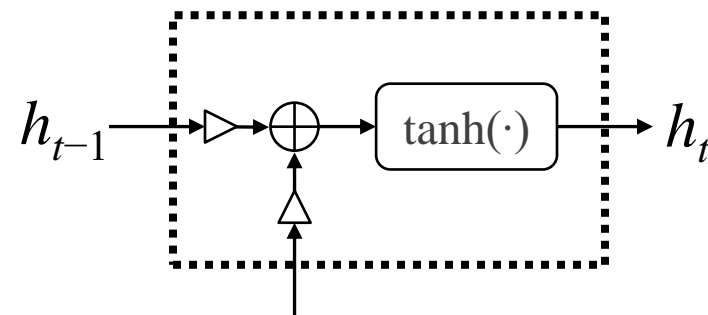
Recurrent Neural Network (RNN): Implementation

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



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

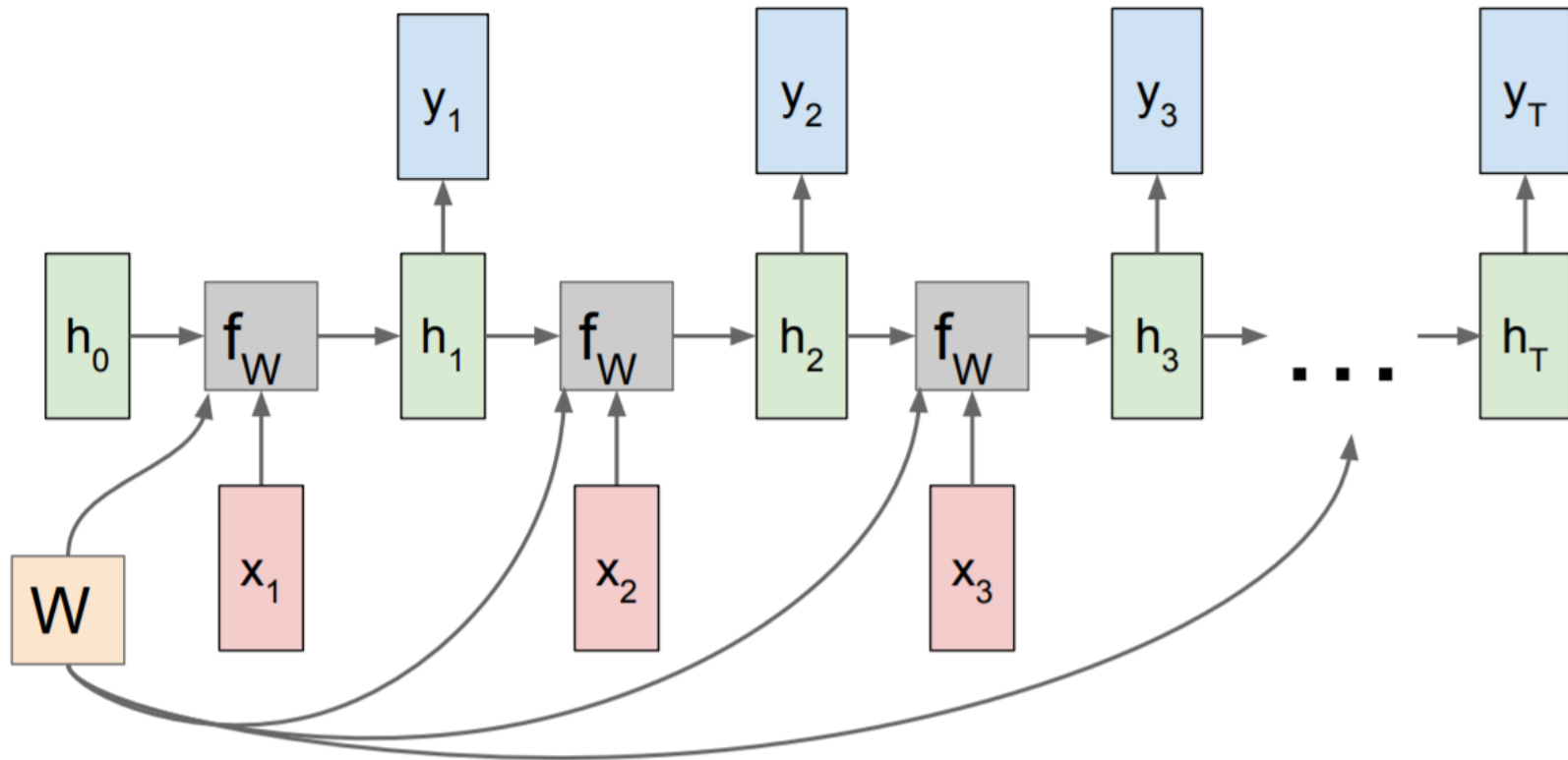
Diagram for f_W .
Can you label the details?



f_W is implemented via

- linear transforms W_{hh} and W_{xh} and
- elementwise **nonlinear** function $\tanh(\cdot)$

Recurrent Neural Network (RNN): Unrolled



Example: Character-Level Language Model

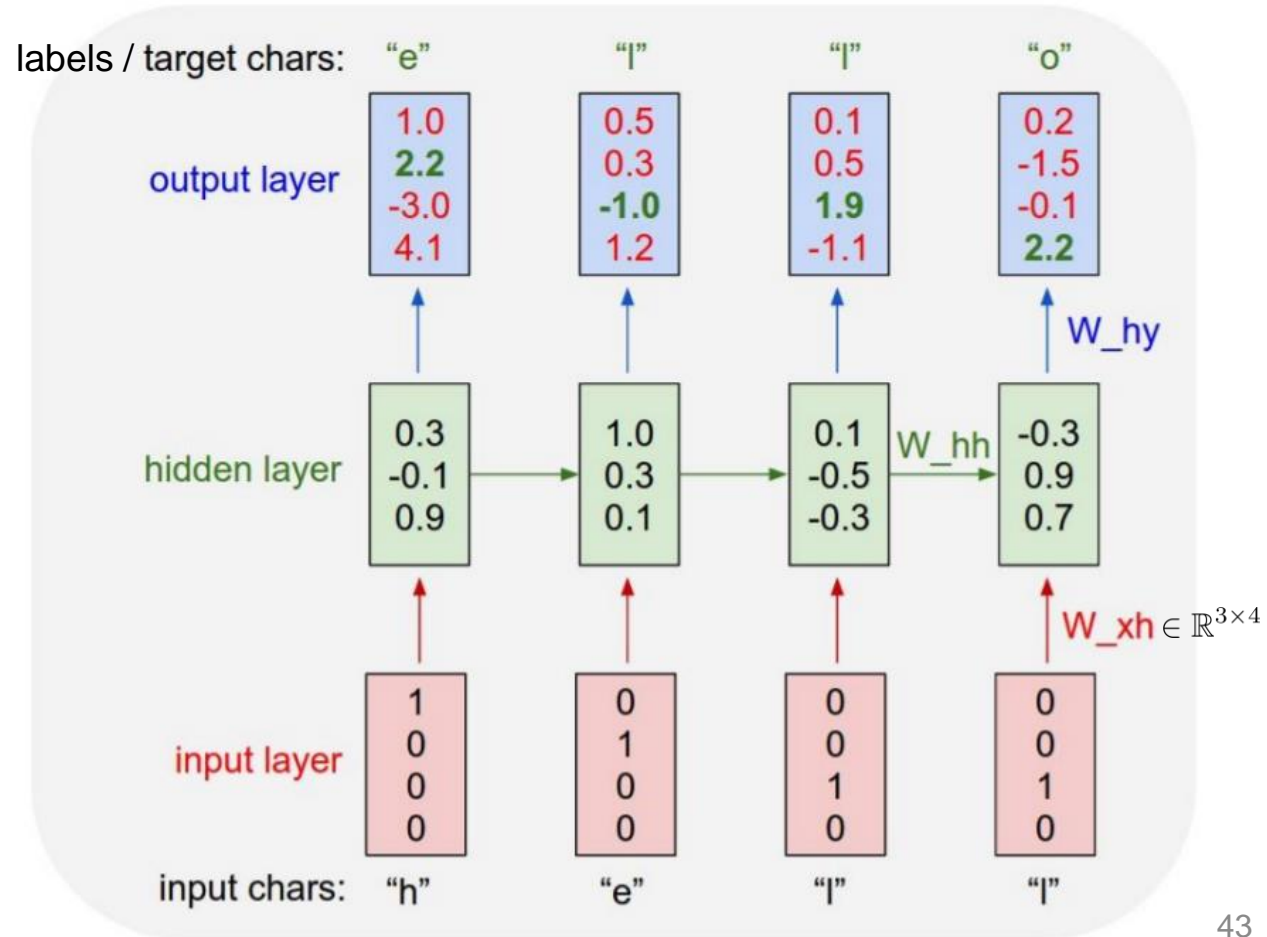
Vocabulary:

[h, e, l, o]

Embeddings (one-hot):

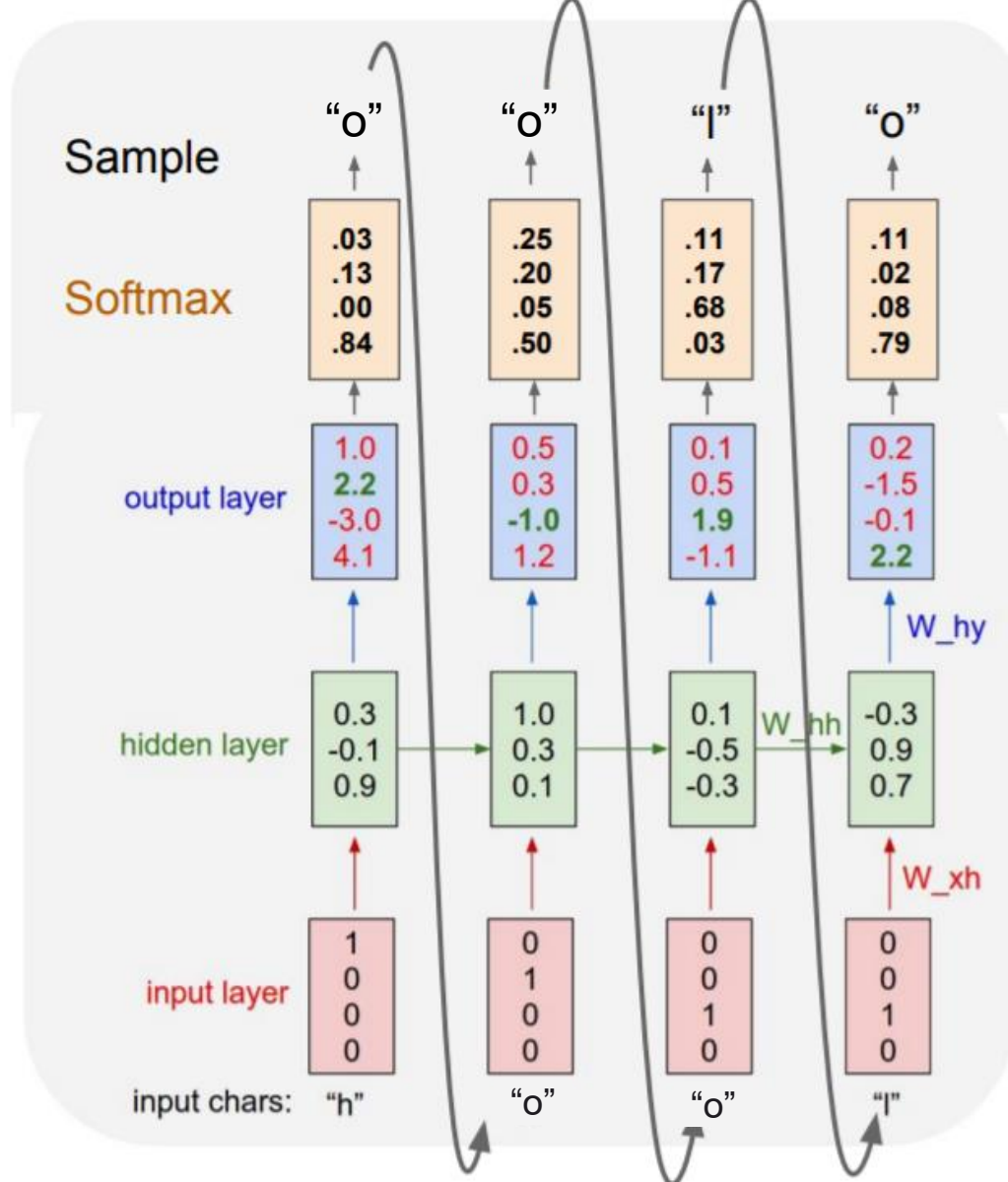
Example training sequence: "hello".

Supervised pairs:



Vocabulary:
[h, e, l, o]

At test time, sample characters one at a time, feed back to model



Long Short-Term Memory (LSTM) Network

- ◆ RNN has the “vanishing gradient” problem!
- ◆ Resolved by long short-term memory (LSTM) units.

RNN

$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

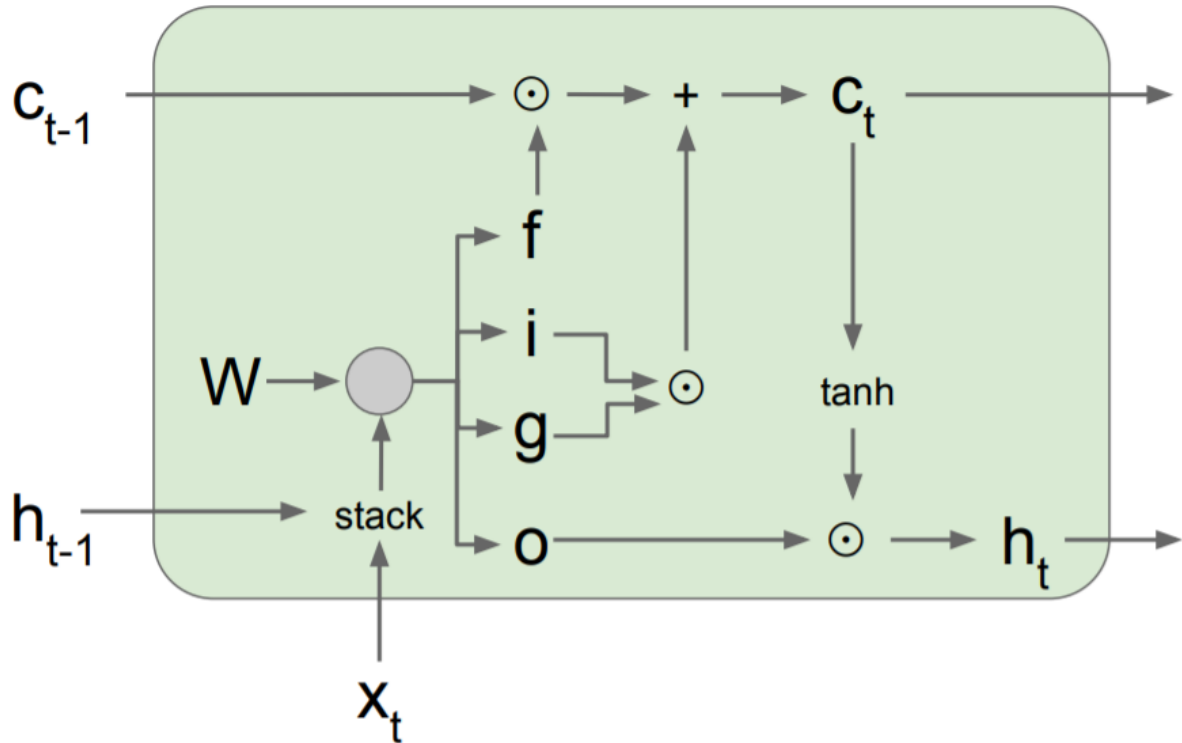
LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

An LSTM Unit [Hochreiter et al., 1997]

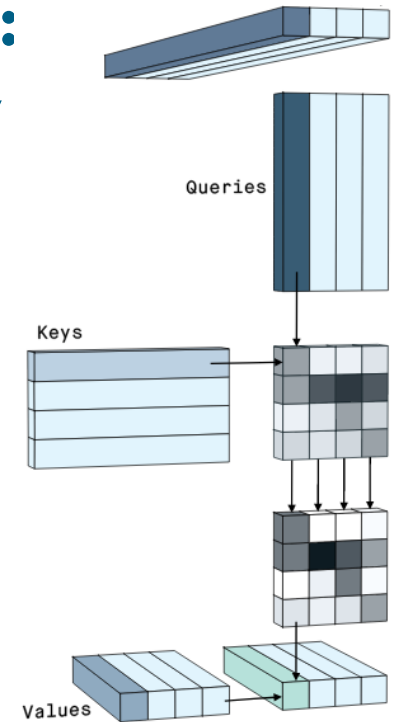


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Overview of Modern ML Applications: Transformer (Underlying technology of BERT & GPT)



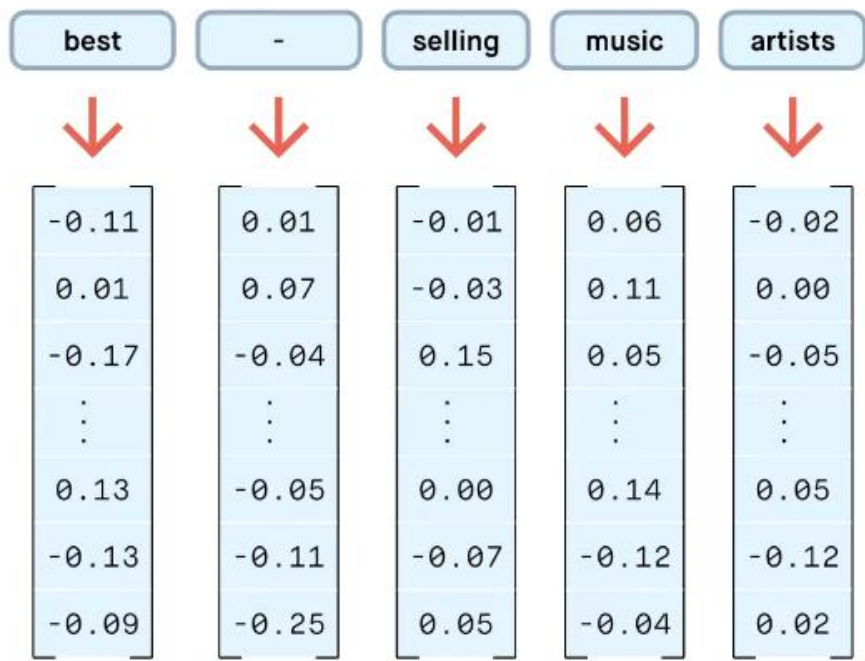
Acknowledgment: Some graphics and slides were adapted from <https://peltarion.com/blog/data-science/self-attention-video>

How to make good sense of language?

- ◆ Reading comprehension: If you were Google, what result(s) should you return for “brazil traveler to usa need a visa”?
 1. A webpage on U.S. citizens traveling to Brazil
 2. A webpage of the U.S. embassy/consulate in Brazil
- ◆ Contextualization/attention is the key!
 - ✦ A nice walk by the river **bank**.
 - ✦ Walk to the **bank** and get cash.
- ◆ The transformer paper (Vaswani et al., 2017) showed that only attention is needed; **convolution** and **recurrence** aren't needed.

Word Embeddings in Natural Language Processing (NLP)

WordPiece tokens:

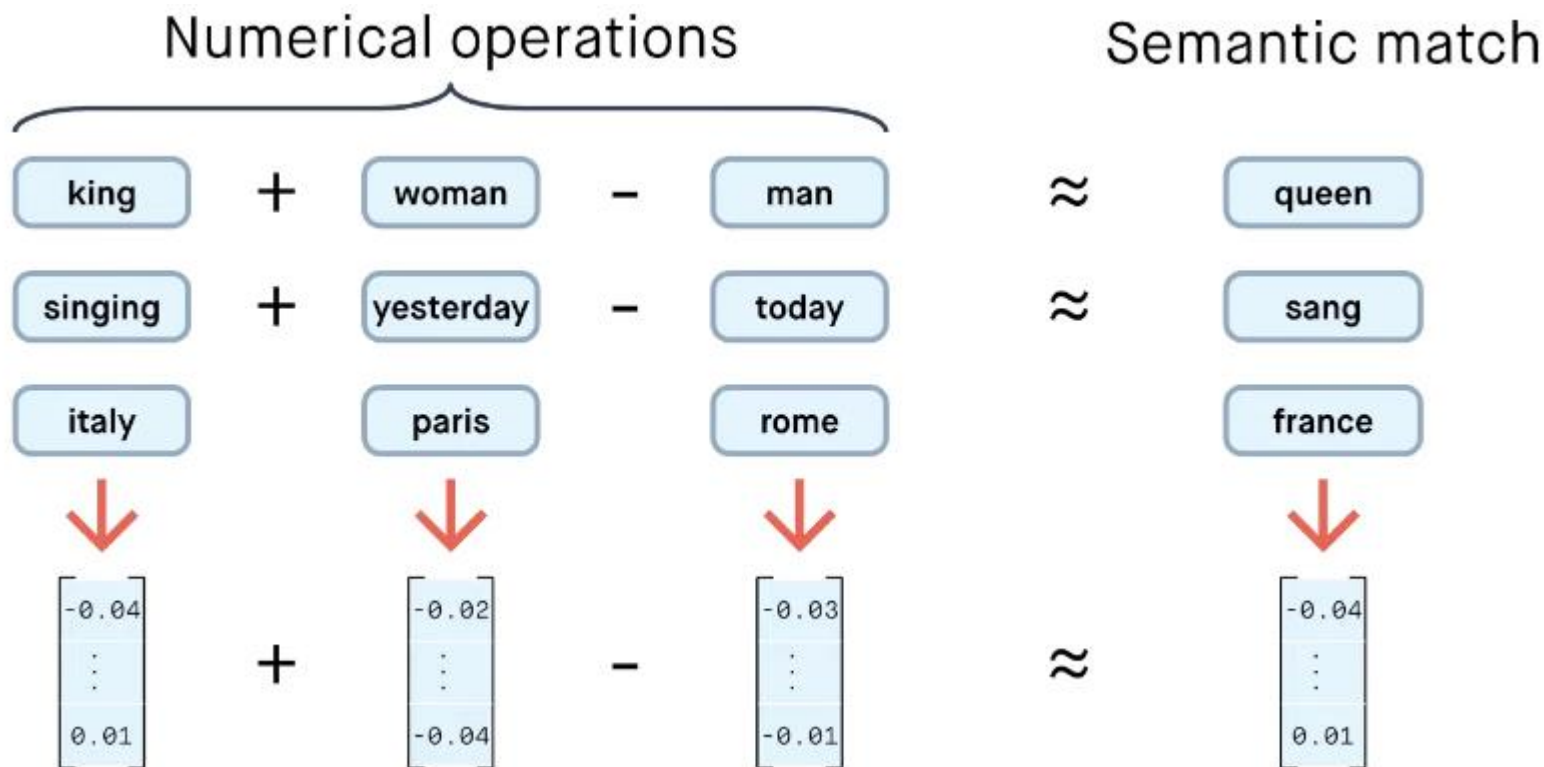


Embedding vectors:

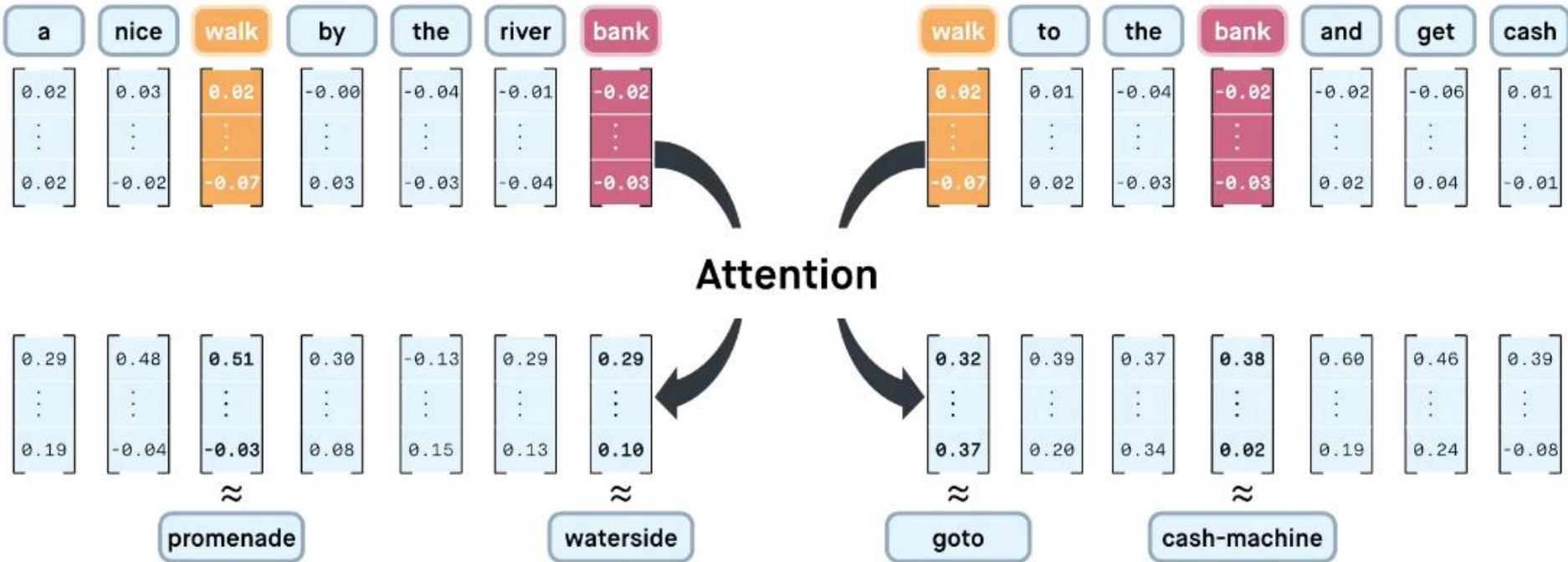
Values are *pretrained*

Embedding examples: Bag of Words (BoW), Word2Vec, ...

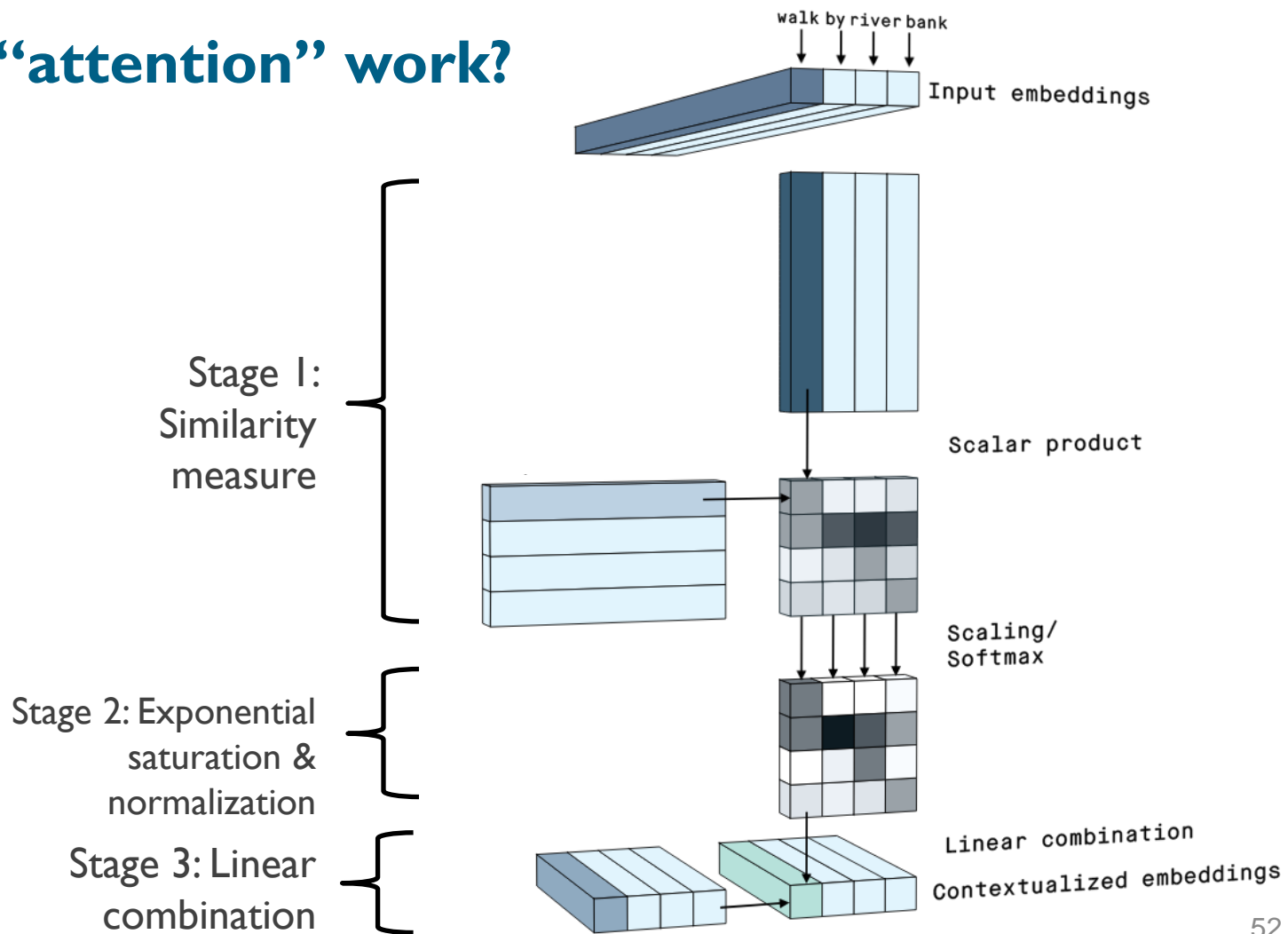
Word Embeddings are Meaningful Under “+” and “-”



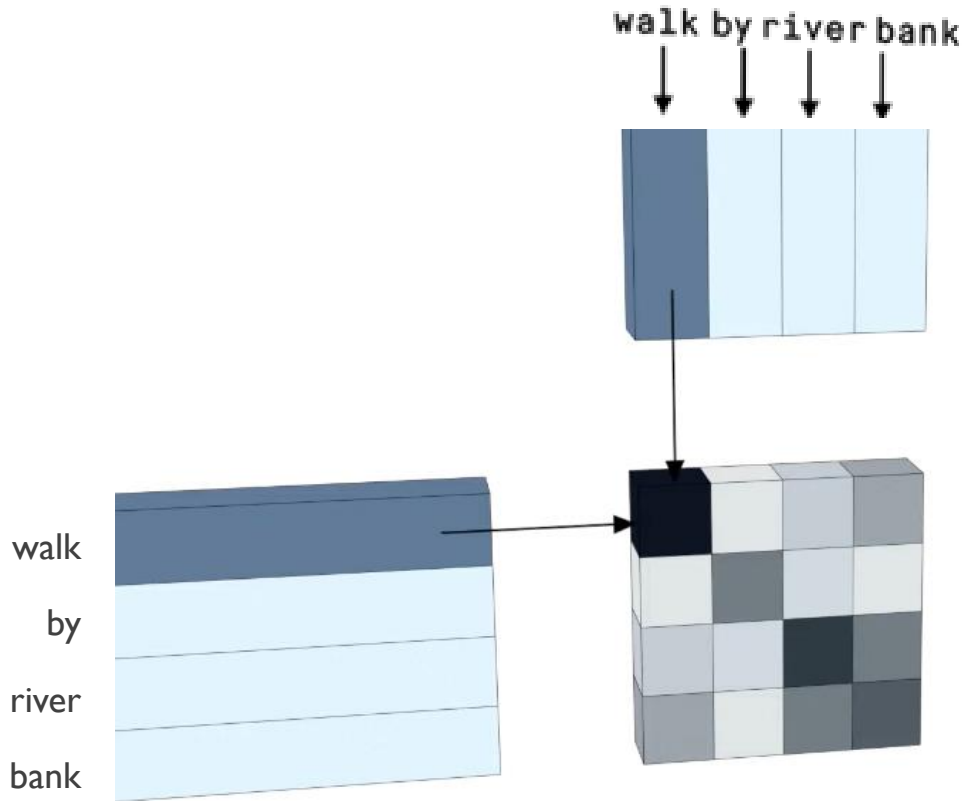
Contextualization by “Attention”



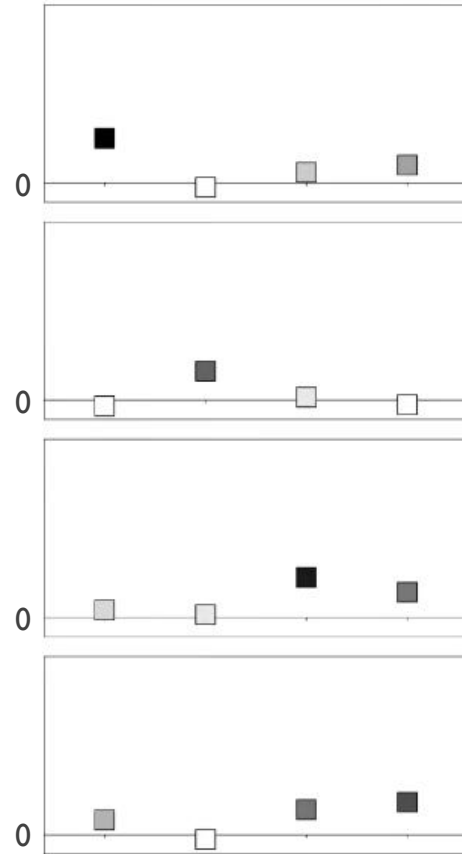
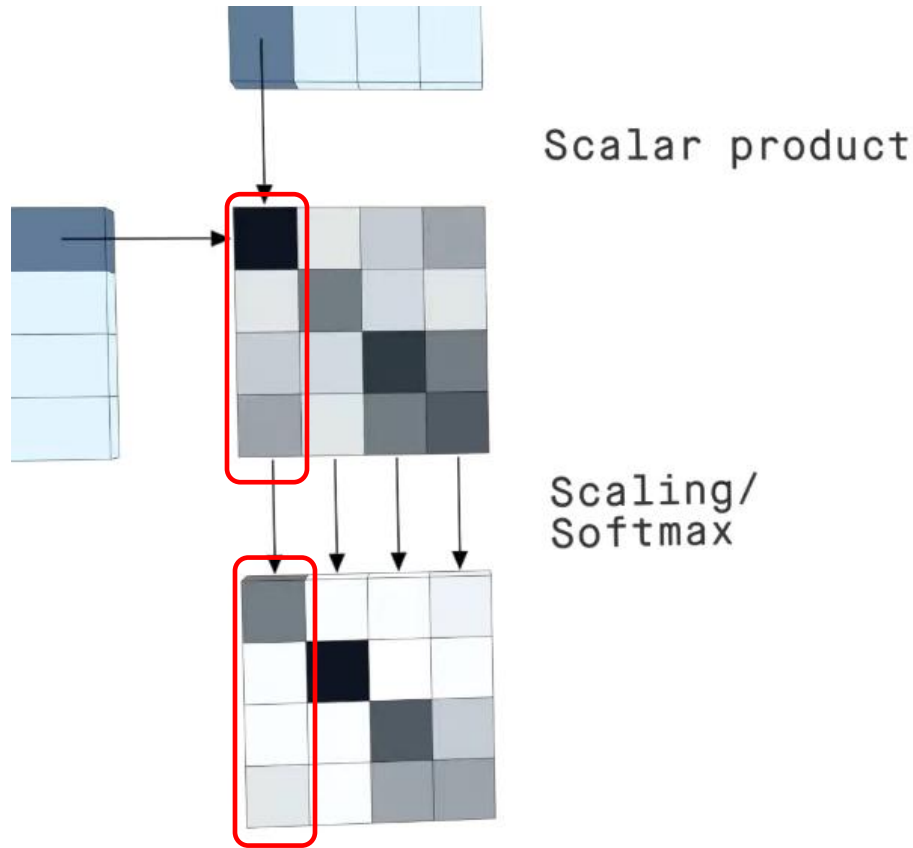
How does “attention” work?



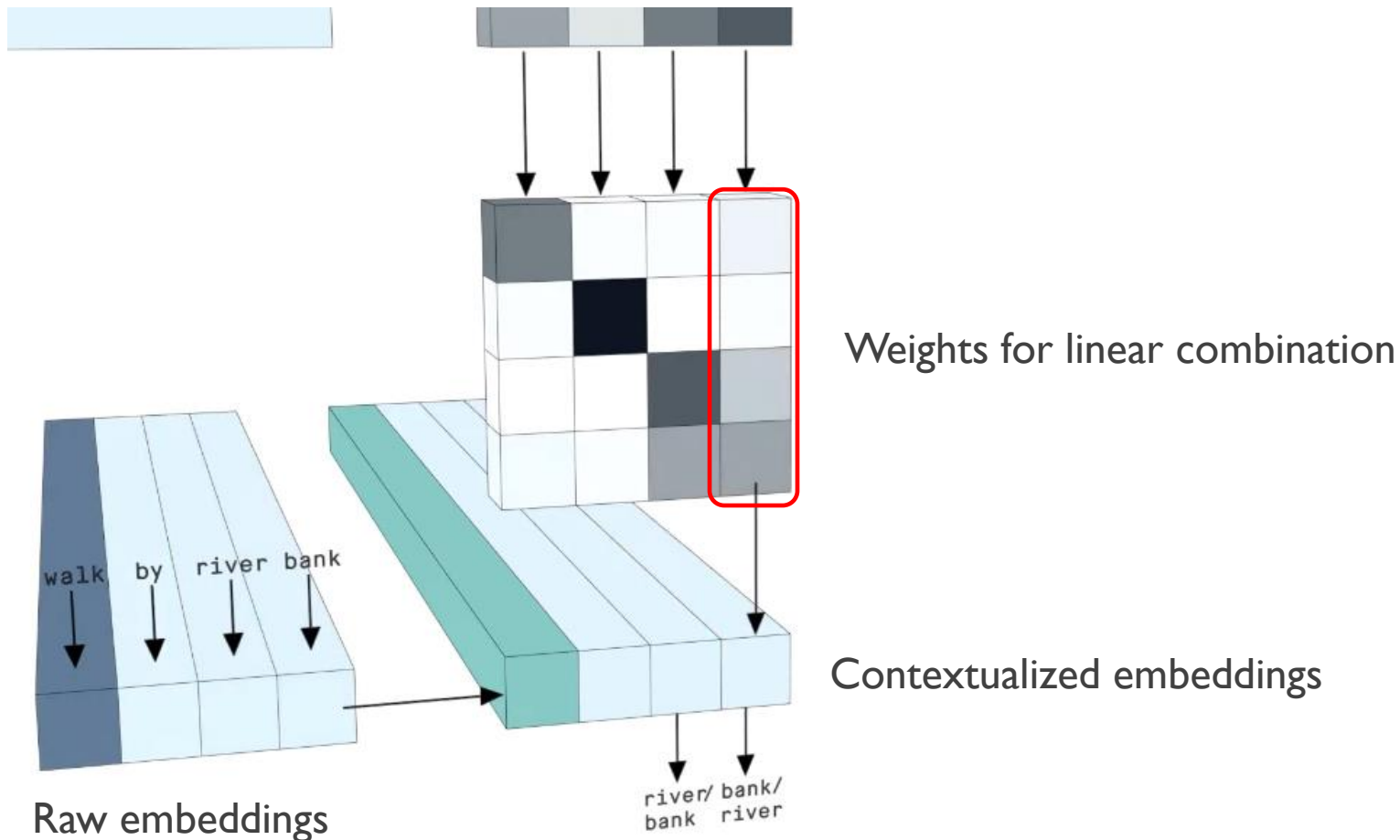
Stage I: Similarity Measure via Inner/Dot/Scalar Product



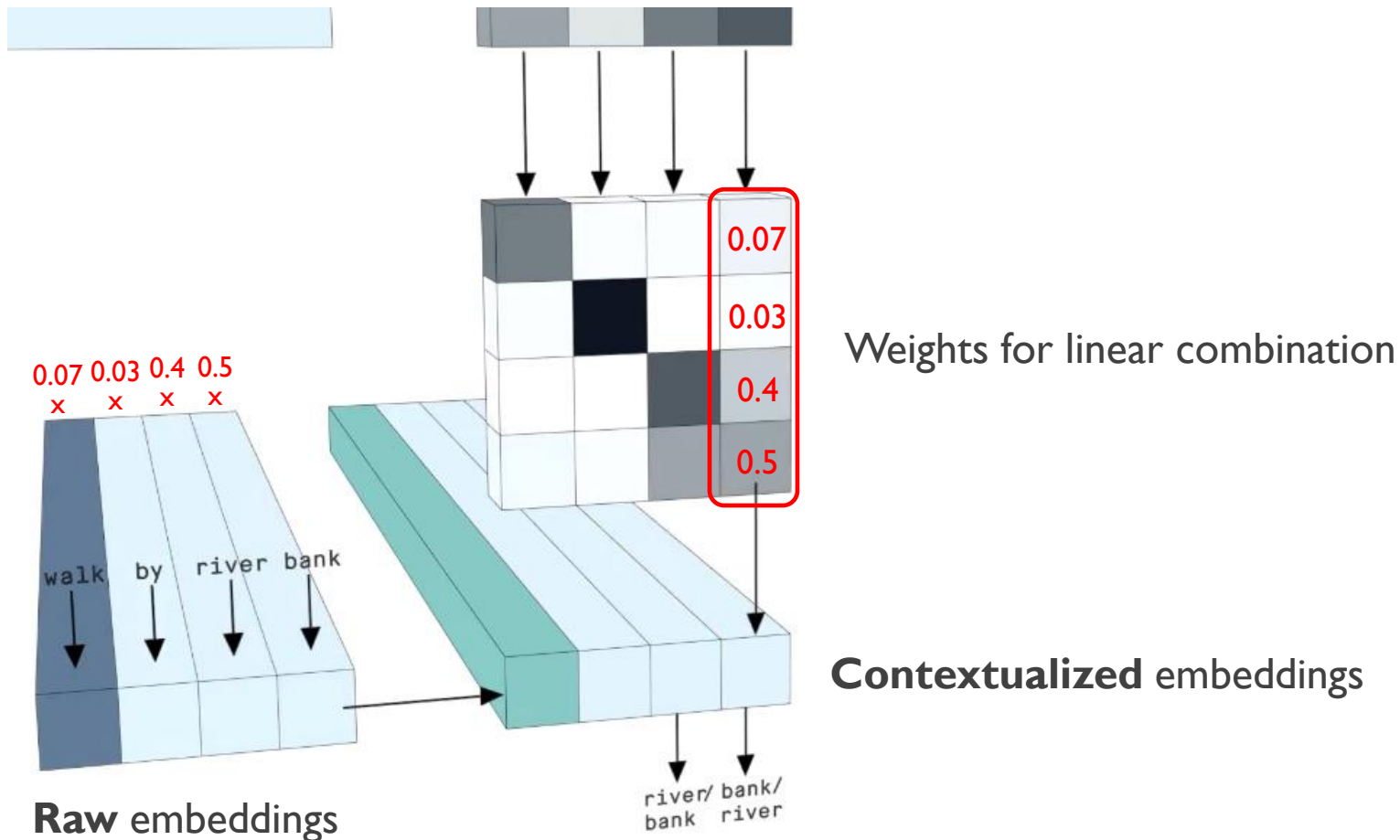
Stage 2: Exponential Saturation & Normalization via Softmax



Stage 3: Contextualization via Linear Combination

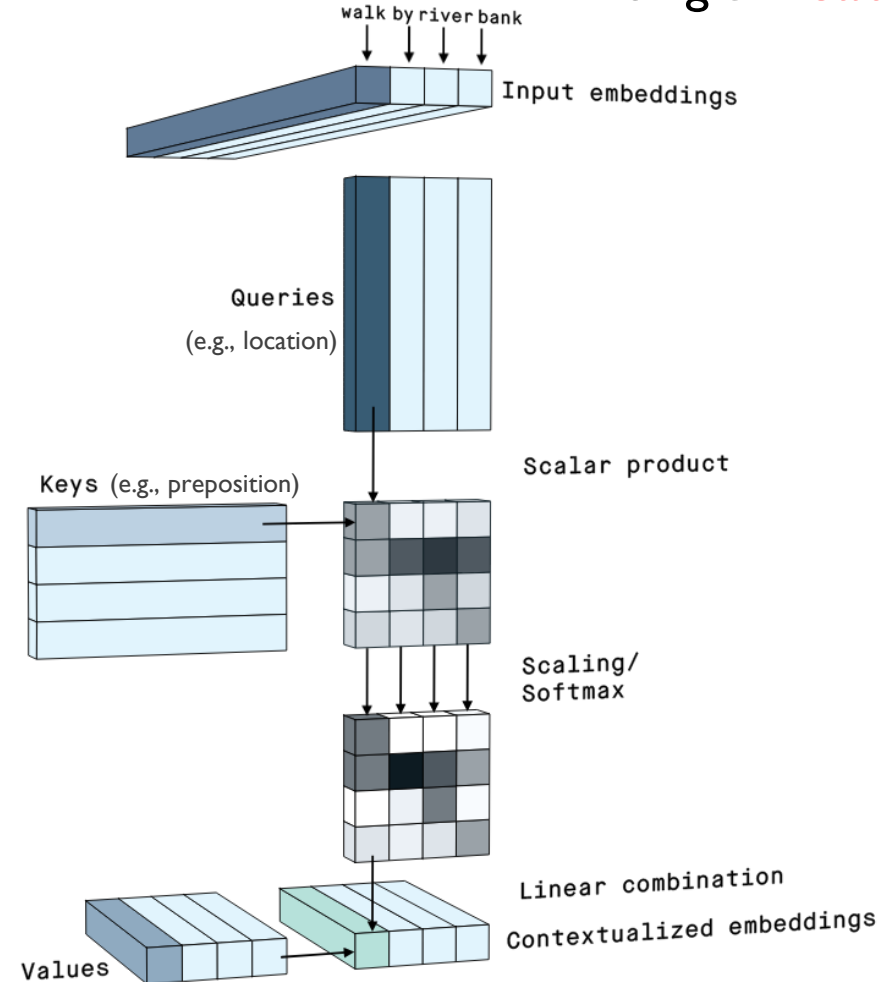


Stage 3: Contextualization via Linear Combination

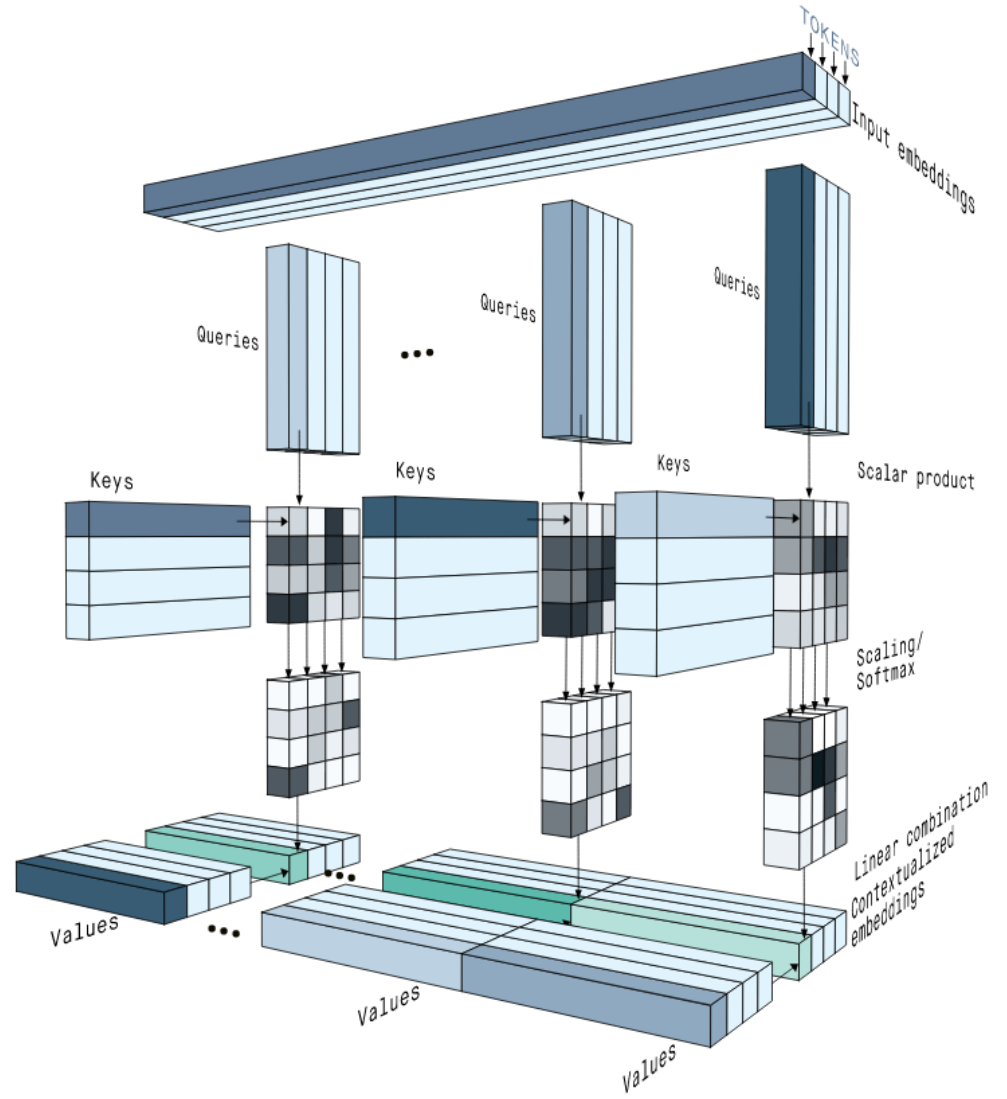


Key, Value, and Query

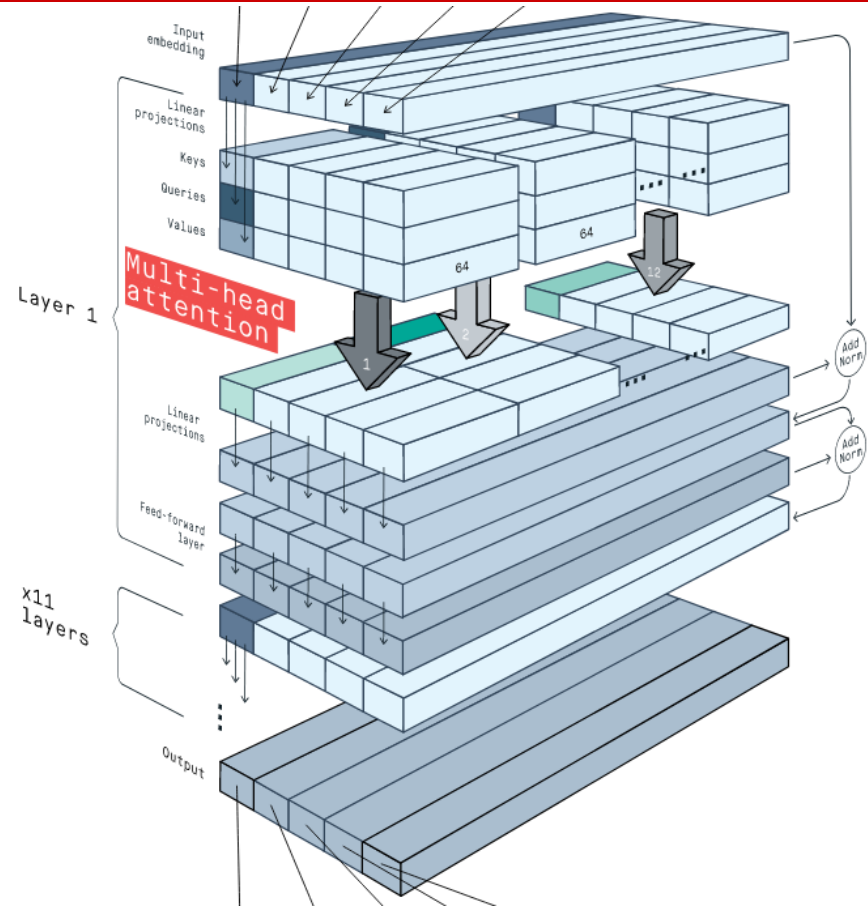
- ◆ “Key”, “value”, and “query” are three projections of an input embedding to three vector subspaces.
- ◆ Each subspace represents a unique **semantic** aspect.
- ◆ The projection operators / matrices provide **trainable** parameters for transformer neural networks.



Multi-Head Attention



Bidirectional Encoder Representations from Transformers (BERT)



A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6000–10, 2017. [\[Transformer paper\]](#)

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *North American Chapter of the Association for Computational Linguistics*, 2019. [\[BERT paper\]](#)

BERT's strategies to train a language model

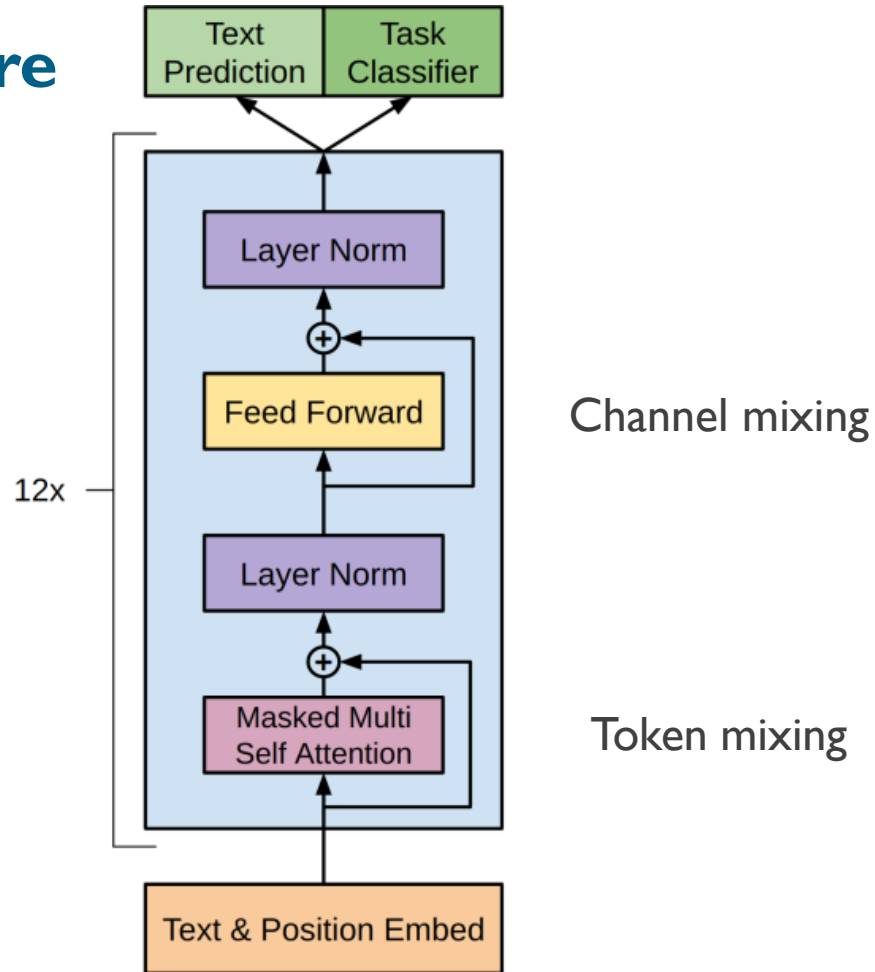
- ◆ Self-supervised task #1: **Masked language model (MLM)**
 - ✦ Mask 15% of all tokens in each sequence.
 - ✦ Predict masked tokens.

- ◆ Self-supervised task #2: **Next sentence prediction (NSP)**
 - ✦ Generate sentence pairs (X, Y). 50% chance Y is the actual next sentence that follows X, and 50% chance Y is a random sentence.
 - ✦ Predict whether Y is the actual next sentence of X.

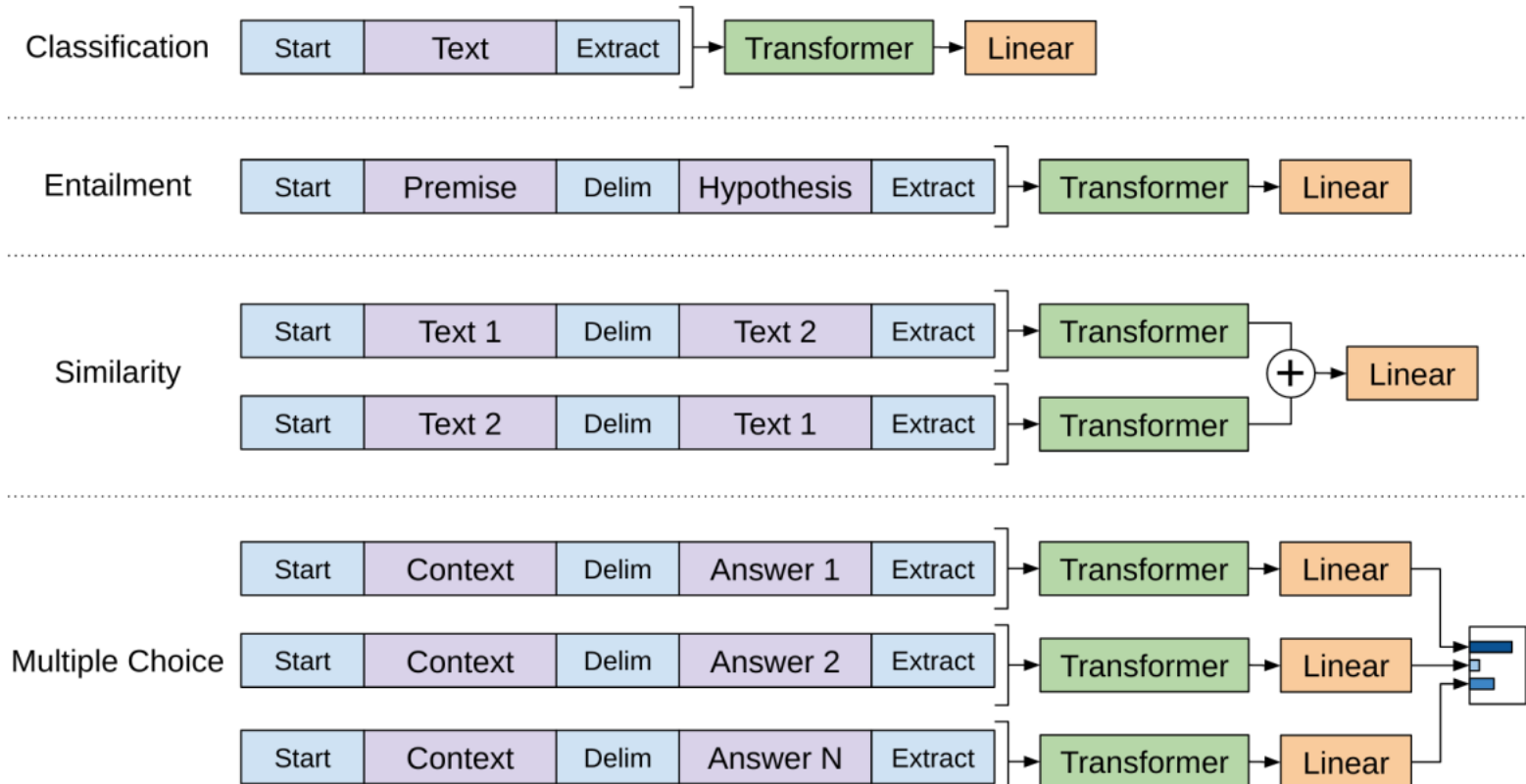
Generative pretrained transformers (GPT) by OpenAI

- ◆ Training strategy: Next token prediction → Key! Autoregressive modeling (with finite Markovian order) is close to modeling the joint distribution of all tokens:
 - ◆ GPT-3 also trained on computer source code.
 - ◆ GPT-3 improved via *reinforcement learning from human feedback (RLHF)*.
 - ◆ ChatGPT fine-tuned with conversational interaction with human users.
 - ◆ Model scale in billion of parameters: GPT-1 (0.1), GPT-2 (1.5), GPT-3 (175), GPT-4 (~10x more); BERT (0.1–0.3).

GPT 1's Structure



Downstream Tasks Using GPT

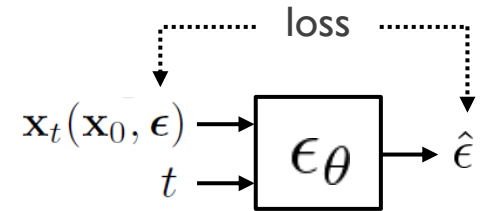


Other Large Language Models (LLMs)

- ◆ Large Language Model Meta AI (Llama). Open source!
 - ✦ Llama 2 (July 2023). Sizes: 7, 13, and 70 bn. Llama Chat available.
 - ✦ Llama 3.1 (July 2024). Sizes: 8, 70, and 405 bn. Context length ↗ 128K.

Rank* (UB)	Model	Arena Score	95% CI	Votes	Organization	License	Knowledge Cutoff
1	o1-preview	1355	+12/-11	2991	OpenAI	Proprietary	2023/10
2	ChatGPT-4o-latest (2024-09-03)	1335	+5/-6	10213	OpenAI	Proprietary	2023/10
2	o1-mini	1324	+12/-9	3009	OpenAI	Proprietary	2023/10
4	Gemini-1.5-Pro-Exp-0827	1299	+5/-4	28229	Google	Proprietary	2023/11
4	Grok-2-08-13	1294	+4/-4	23999	xAI	Proprietary	2024/3
6	GPT-4o-2024-05-13	1285	+3/-3	90695	OpenAI	Proprietary	2023/10
7	GPT-4o-mini-2024-07-18	1273	+3/-3	30434	OpenAI	Proprietary	2023/10
7	Claude-3.5-Sonnet	1269	+3/-3	62977	Anthropic	Proprietary	2024/4
7	Gemini-1.5-Flash-Exp-0827	1269	+4/-4	22264	Google	Proprietary	2023/11
7	Grok-2-Mini-08-13	1267	+4/-5	22041	xAI	Proprietary	2024/3
7	Gemini-Advanced-App (2024-05-14)	1267	+3/-3	52218	Google	Proprietary	Online
7	Meta-Llama-3.1-405b-Instruct-fp8	1266	+4/-4	31280	Meta	Llama 3.1 Community	2023/12

Community benchmarking, e.g., LMSYS Chatbot Arena Leaderboard (as of 9/18/2024)



Overview of Modern ML Applications: Diffusion Models

Learning objectives:

- Be able to explain the principle of diffusion models and name common applications.
- Be able to follow the key derivation steps of diffusion models.

Acknowledgment: This slide deck was adapted from [this CVPR 2023 tutorial](#) by Song, Meng, and Vahdat. [\[Video recording\]](#)

Applications for Generative Models

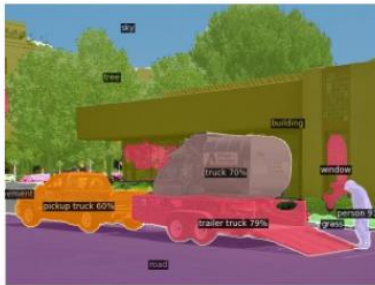
Art & Design



Content Generation



Representation Learning



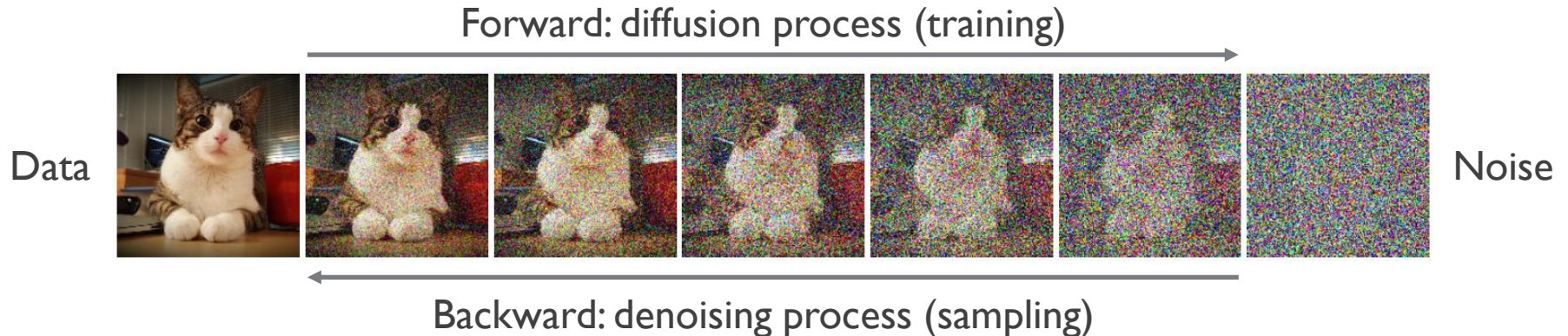
Entertainment



Playground Demo

Diffusion Model: Basic Idea

- ◆ Goal: Learning to generate by denoising.
- ◆ Diffusion model contains two processes:
 - ★ Forward diffusion: Gradually adds noise and learns a denoising net.
 - ★ Backward denoising: Reconstruct data via learned denoising net.

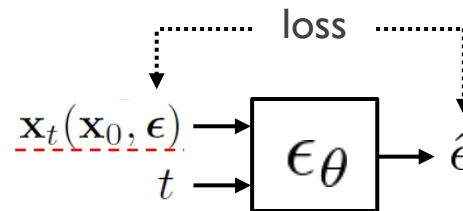


Diffusion Model: Algorithmic Perspective

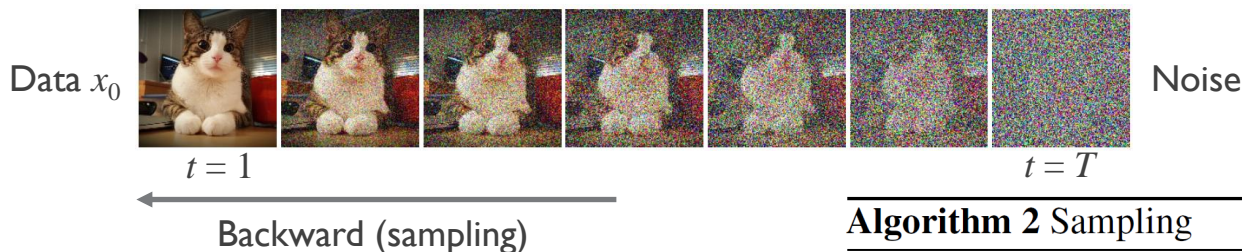
Algorithm 1 Training

- 1: **repeat**
 - 2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
 - 3: $t \sim \text{Uniform}(\{1, \dots, T\})$
 - 4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
 - 5: Take gradient descent step on

$$\nabla_{\theta} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t) \right\|^2$$
 - 6: **until** converged
-



Forward (training) →

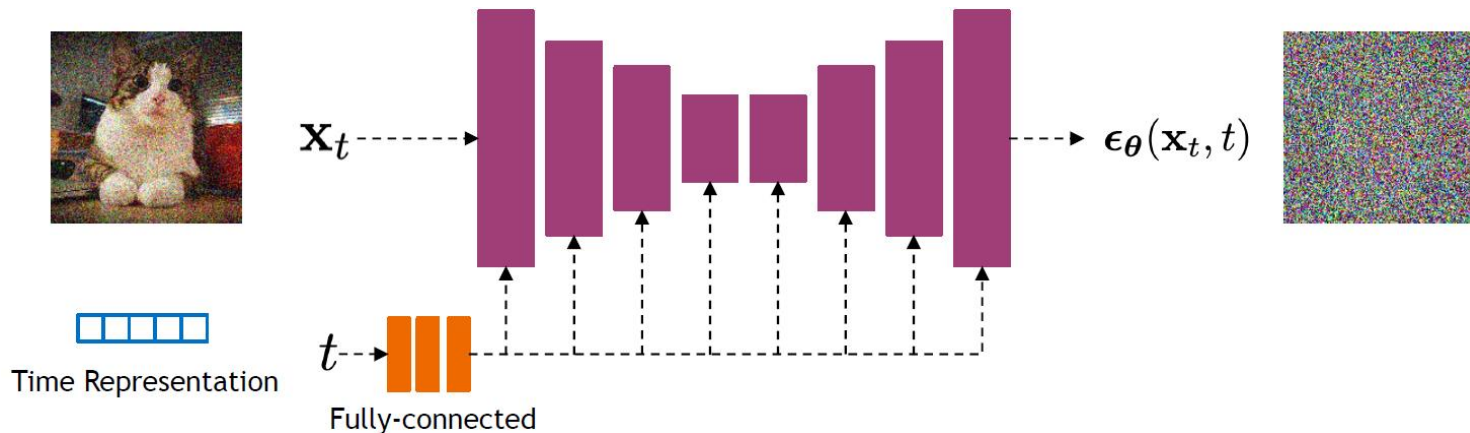


Algorithm 2 Sampling

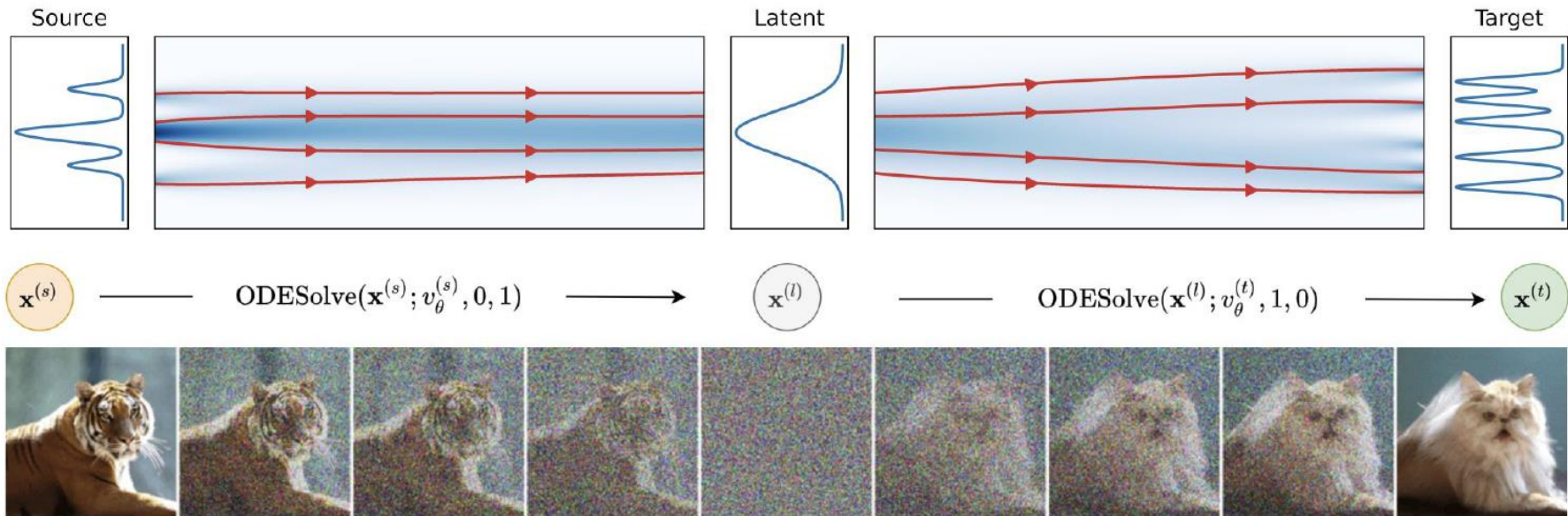
- 1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 2: **for** $t = T, \dots, 1$ **do**
- 3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
- 4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
- 5: **end for**
- 6: **return** \mathbf{x}_0

Diffusion Model: Implementation Details

- ◆ Diffusion models often use U-Net with ResNet blocks and self-attention layers.
- ◆ Time representation: Sinusoidal positional embeddings or random Fourier features.
- ◆ Time is fed to the residual blocks using (i) simple spatial addition or (ii) adaptive group normalization layers (Dhariwal & Nichol, 2021).



Ex: Style Transfer



Ex: Semantic Editing with Mask Guidance (DiffEdit)



User-provided mask **not** needed: Model generates a mask based on caption & query.

Ex: Prompt-to-Prompt Image Editing w/ Cross Attention Control



“The boulevards are crowded today.”



“Photo of a cat riding on a bicycle.”



“Landscape with a house near a river and a rainbow in the background.”



“My fluffy bunny doll.”



“a cake with decorations.”



“Children drawing of a castle next to a river.”

Ex: Personalization with Diffusion Models



Input images



in the Acropolis



swimming



sleeping



in a doghouse



in a bucket




getting a haircut

Ex: Optimizing Text Embedding (Textual Inversion)

Input samples $\xrightarrow{\text{invert}}$ " S_* "



→



"An oil painting of S_* "



"App icon of S_* "




"Elmo sitting in the same pose as S_* "




"Crochet S_* "

Input samples $\xrightarrow{\text{invert}}$ " S_* "



→




"Painting of two S_* fishing on a boat"



"A S_* backpack"



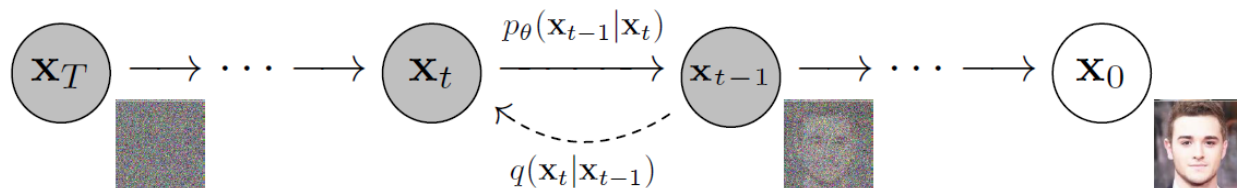
"Banksy art of S_* "



"A S_* themed lunchbox"

Mathematical / Probabilistic Formulation

- ◆ Raw data model: $q(\mathbf{x}_0)$, where q denotes a PDF
- ◆ Diffusion model (parameterized by θ): $p_\theta(\mathbf{x}_0) := \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$
Note $\mathbf{x}_{0:T} = \mathbf{x}_0, \dots, \mathbf{x}_T$



$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

\mathbf{x}_T is Gaussian distributed
w/ mean $\mathbf{0}$ and covariance \mathbf{I}

Note the pipeline is horizontally flipped.

Forward Diffusion: Single Step



x_0 x_1 x_2 x_3 x_4 ... x_T



$$\mathbf{e}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\mathbf{x}_1 = \sqrt{\alpha_1}\mathbf{x}_0 + \sqrt{\beta_1}\mathbf{e}_1 \quad \dots \quad \mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\mathbf{e}_t \quad \dots \quad \mathbf{x}_T = \sqrt{\alpha_T}\mathbf{x}_{T-1} + \sqrt{\beta_T}\mathbf{e}_T$$

Intuition: Image intensity is scaled down and white Gaussian noise is added to corrupt the image. Variance of the raw image is \mathbf{I} . Variances of the noisy images are maintained to be \mathbf{I} .

$$\alpha_t, \beta_t \in (0, 1)$$

$$\alpha_t + \beta_t = 1$$

Alternatively, one-step conditional distribution (1st-order Markov chain) can be written as:

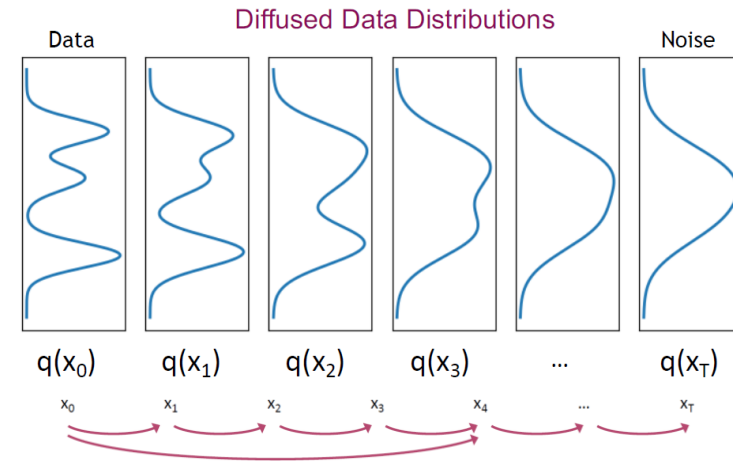
$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Forward Diffusion: Arbitrary Steps

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{(1 - \bar{\alpha}_t)} \epsilon \quad \text{where } \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \bar{\alpha}_t = \prod_{i=1}^t \alpha_i$$

Validation for \mathbf{x}_2 :

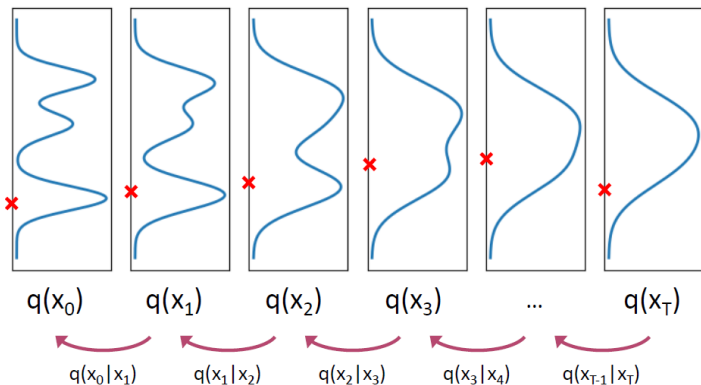
$$\begin{aligned} \mathbf{x}_2 &= \sqrt{\alpha_2} \mathbf{x}_1 + \sqrt{\beta_2} \mathbf{e}_2 \\ &= \sqrt{\alpha_2} \left(\sqrt{\alpha_1} \mathbf{x}_0 + \sqrt{\beta_1} \mathbf{e}_1 \right) + \sqrt{\beta_2} \mathbf{e}_2 \\ &= \sqrt{\alpha_2} \sqrt{\alpha_1} \mathbf{x}_0 + \left(\sqrt{\alpha_2} \sqrt{\beta_1} \mathbf{e}_1 + \sqrt{\beta_2} \mathbf{e}_2 \right) \\ &= \sqrt{\bar{\alpha}_2} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_2} \mathbf{e}'_2 \end{aligned}$$



Alternatively, one can write: $q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I})$

$\beta_t \in (0, 1)$ ensures that for large T (e.g., $T = 1000$), $\bar{\alpha}_T \rightarrow 0$ and $q(\mathbf{x}_T | \mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Backward Denoising



Algorithm 2 Sampling

- 1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
 - 2: **for** $t = T, \dots, 1$ **do**
 - 3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
 - 4: $\mathbf{x}_{t-1} = \mu_{\theta}(\mathbf{x}_t, t) + \sigma_t \mathbf{z}$ Step-by-step reconstruction
 - 5: **end for**
 - 6: **return** \mathbf{x}_0
-

Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1} | \mathbf{x}_t)}_{\text{True Denoising Dist.}}$

True Denoising Dist.

Can we approximate $q(\mathbf{x}_{t-1} | \mathbf{x}_t)$? Yes, we can use a **Normal distribution** if β_t is small in each forward diffusion step.

$$p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_{\theta}(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})$$

Use NN as a function approximator

Note: 1. If random variables are jointly Gaussian, then any conditional distribution is also Gaussian.
 2. \mathbf{x}_0 is not Gaussian, so approximation is needed.

What and How to Train?

- ◆ Due to the following relation, may use another network to approximate $\epsilon_\theta(\mathbf{x}_t, t)$ instead of $\mu_\theta(\mathbf{x}_t, t)$:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right)$$

- ◆ Loss function:

$$l = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), t \sim U[1, T], \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left\| \epsilon - \epsilon_\theta \left(\underbrace{\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon}_{= \mathbf{x}_t}, t \right) \right\|_2^2$$

Forward Diffusion: Forming a Training Data Pair

Step 1: Draw an image \mathbf{x}_0 from $q(\cdot)$

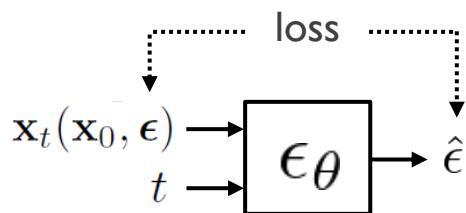
Step 2: Pick a time step t

Step 3: Create a noisy image $\mathbf{x}_t \sim q(\mathbf{x}_t \mid \mathbf{x}_0)$ by fast-forwarding t steps via

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$$

Algorithm 1 Training

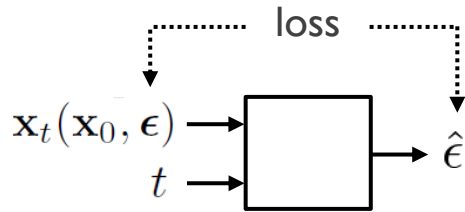
- 1: **repeat**
 - 2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
 - 3: $t \sim \text{Uniform}(\{1, \dots, T\})$
 - 4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
 - 5: Take gradient descent step on
 $\nabla_{\theta} \|\epsilon - \epsilon_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t)\|^2$
 - 6: **until** converged
-



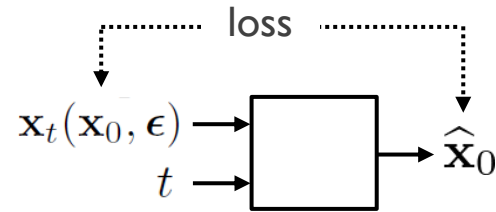
Data: $(\mathbf{x}_t(\mathbf{x}_0, \epsilon), t)$

Label: ϵ

BTW, OpenAI Uses a Slightly Different Loss



DDPM (Ho et al., 2020)

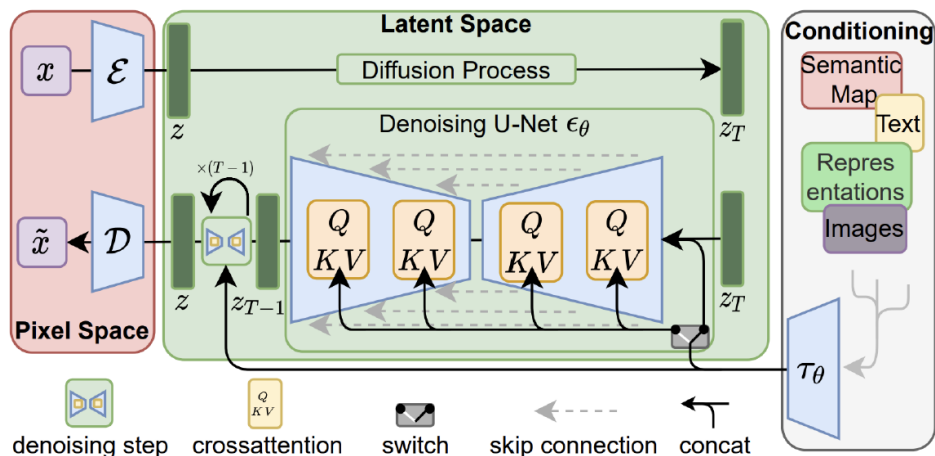


DALL-E 2 (Ramesh et al., 2022)

One-step denoising w/ knowledge of step t

Latent/Stable Diffusion

- ◆ Idea: Use an encoder to map the input data to an embedding space so that denoising diffusion is done in the latent space.



- ◆ Advantages:

- ★ Embeddings are closer to normal distribution => More correct modeling assumption, simpler denoising, faster synthesis.
- ★ Latent space => More expressivity and flexibility in design.
- ★ Tailored Autoencoders => Application to any data type (graphs, text, 3D data, etc.)

Conditioning the Diffusion Models

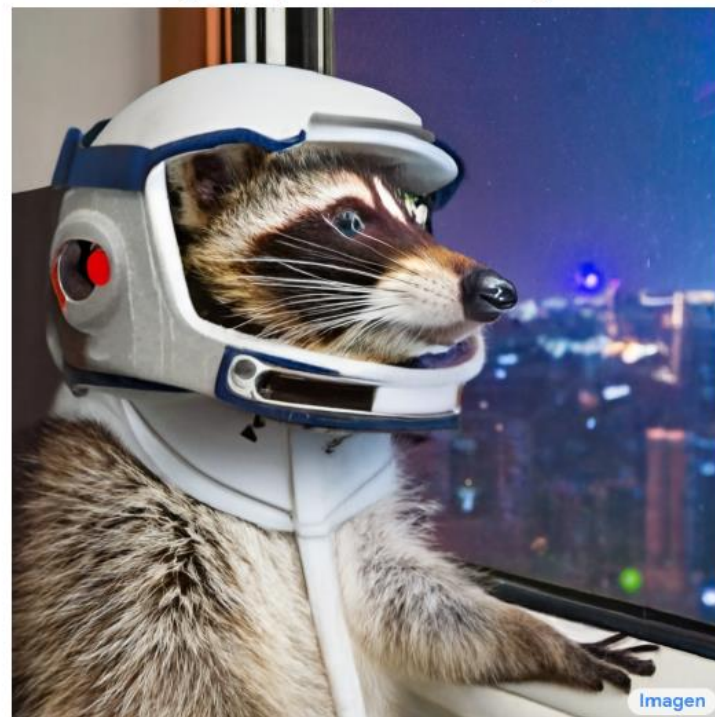
DALL·E 2

“a propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese”

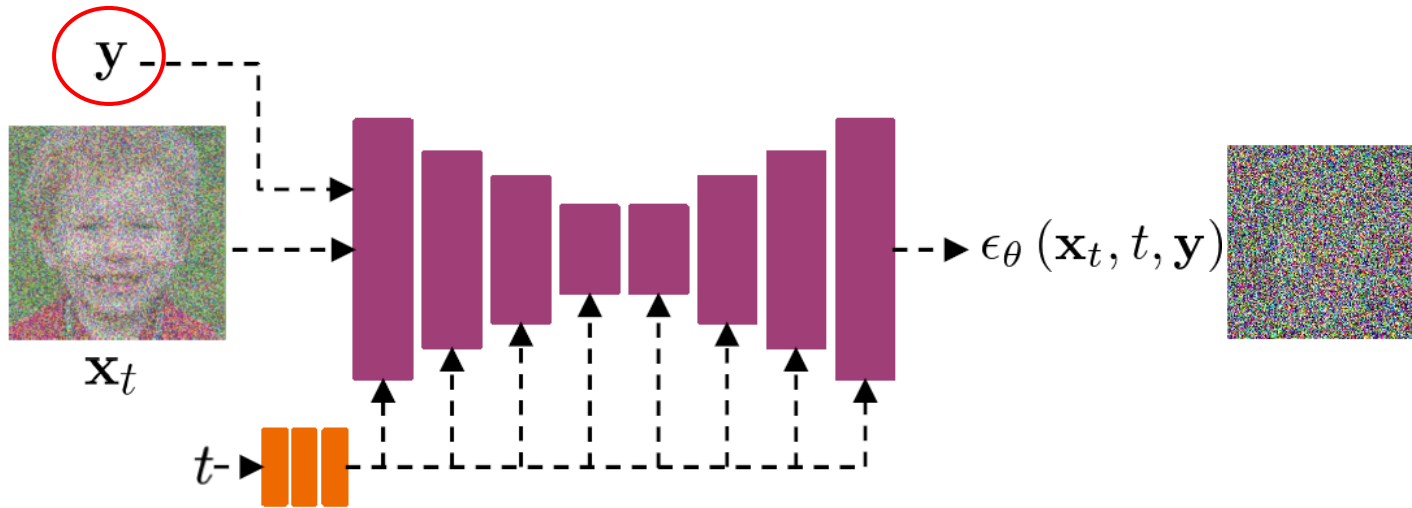


IMAGEN

“A photo of a raccoon wearing an astronaut helmet, looking out of the window at night.”



Treating Side Information \mathbf{y} as Another Input



$$\ell = \mathbb{E}_{(\mathbf{x}_0, \mathbf{y}) \sim q(\mathbf{x}_0, \mathbf{y}), t \sim U[1, T], \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \|\epsilon - \epsilon_\theta(\mathbf{x}_t, t, \mathbf{y})\|_2^2$$