STAT 453: Introduction to Deep Learning and Generative Models

Ben Lengerich

Lecture 26: Review

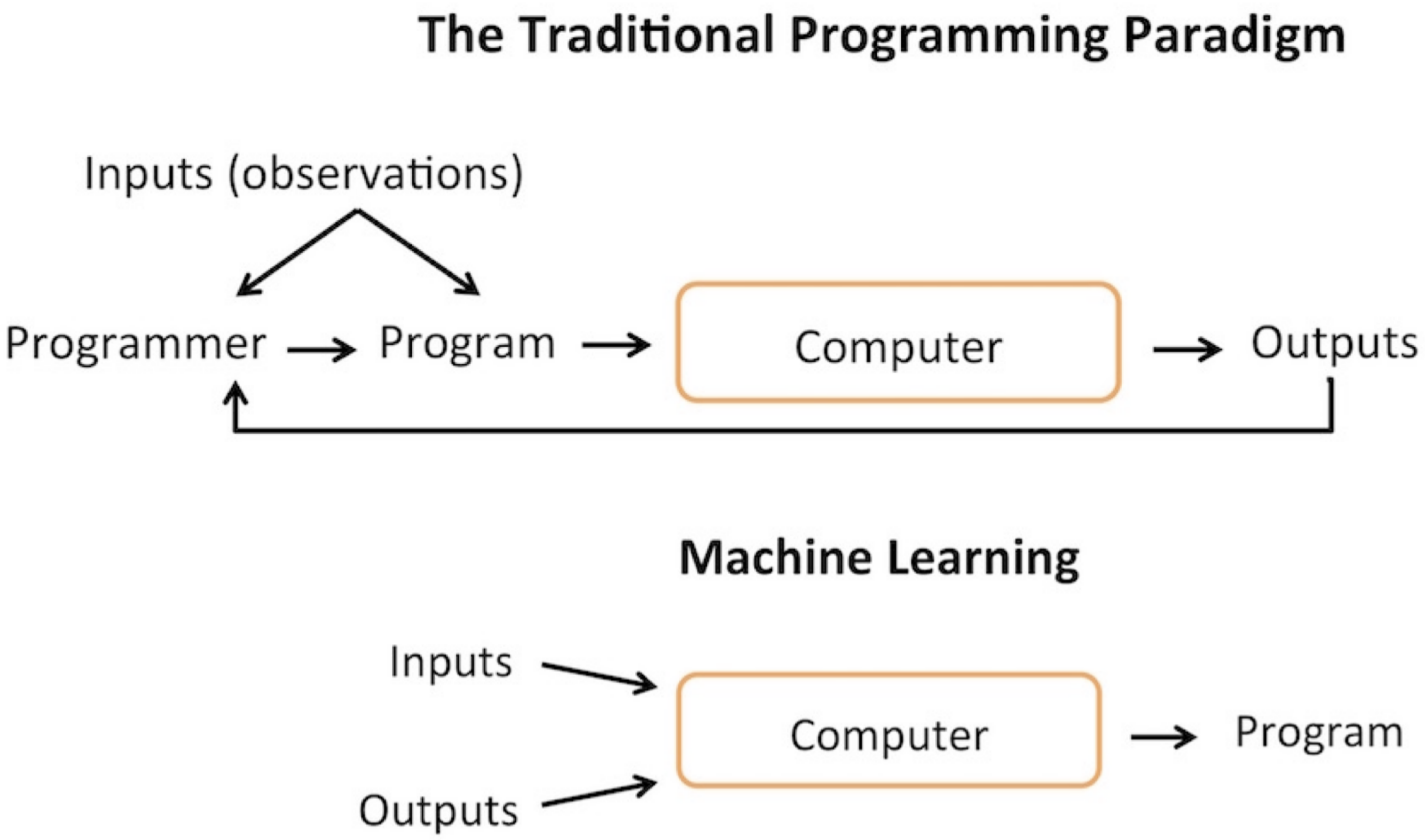
December 3, 2025

Course Schedule / Calendar

Week	Lecture Dates	Topic	Assignments
Module 1: Introduction and Foundations			
1	9/3	Course Introduction	
2	9/8, 9/10	A Brief History of DL, Statistics / linear algebra / calculus review	HW1
3	9/15, 9/17	Single-layer networks Parameter Optimization and Gradient Descent	
4	9/22, 9/24	Automatic differentiation with PyTorch, Cluster and cloud computing resources	HW 2
Module 2: Neural Networks			
5	9/29, 10/1	Multinomial logistic regression, Multi-layer perceptrons and backpropagation	
6	10/6, 10/8	Regularization Normalization / Initialization	HW 3
7	10/13, 10/15	Optimization, Learning Rates CNNs	Project Proposal
8	10/20, 10/22	Review, **Midterm Exam**	In-class Exam

Week	Lecture Dates	Topic	Assignments
Module 3: Intro to Generative Models			
9	10/27, 10/29	A Linear Intro to Generative Models, Factor Analysis, Autoencoders, VAEs	
10	11/3, 11/5	Generative Adversarial Networks, Diffusion Models	Project Midway Report
Module 4: Large Language Models			
11	11/10, 11/12	Sequence Learning with RNNs Attention, Transformers	HW4
12	11/17, 11/19	GPT Architectures, Unsupervised Training of LLMs	
13	11/24, 11/26	Supervised Fine-tuning of LLMs, Prompts and In-context learning	HW5
14	12/1, 12/3	Foundation models, alignment, explainability Open directions in LLM research	
15	12/8, 12/10	**Project Presentations**	Project Final Report
16	12/17	**Final Exam**	Final Exam

What is Machine Learning?



What is Machine Learning?

Formally, a computer program is said to **learn** from experience $\mathcal{E}$ with respect to some task $\mathcal{T}$ and performance measure $\mathcal{P}$ if its **performance at $\mathcal{T}$ as measured by $\mathcal{P}$ improves with** $\mathcal{E}$.

Supervised Learning
- Labeled data
- Direct feedback
- Predict outcome/future

- Task $\mathcal{T}$: Learn a function $h: \mathcal{X} \rightarrow \mathcal{Y}$
- Experience $\mathcal{E}$: Labeled samples $\{(\mathrm{x_i}, \mathrm{y_i})\}_{\mathrm{i}=1}^{n}$
- Performance $\mathcal{P}$: A measure of how good h is

Unsupervised Learning
- No labels/targets
- No feedback
- Find hidden structure in data

- Task $\mathcal{T}$: Discover structure in data
- Experience $\mathcal{E}$: Unlabeled samples $\{\mathrm{x_i}\}_{\mathrm{i}=1}^{n}$
- Performance $\mathcal{P}$: Measure of fit or utility

Reinforcement Learning
- Decision process
- Reward system
- Learn series of actions

- Task $\mathcal{T}$: Learn a policy $\pi: S \rightarrow A$
- Experience $\mathcal{E}$: Interaction with environment
- Performance $\mathcal{P}$: Expected reward

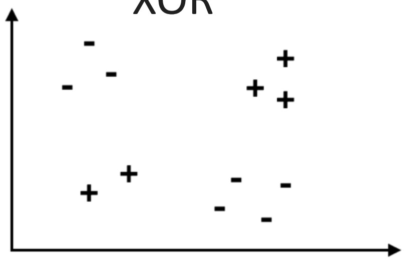
Source: Raschka and Mirjalily (2019). *Python Machine Learning, 3rd Edition*

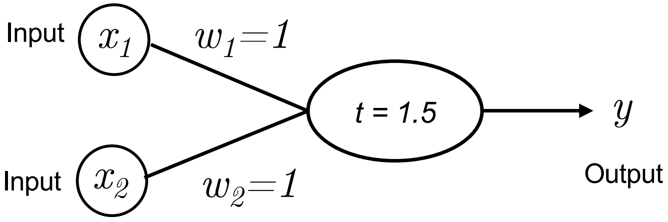
The building blocks of Deep Learning

McCulloch & Pitt's neuron model (1943)

- McCulloch & Pitts neuron: Threshold and (+1, -1) weights
- Can represent "AND", "OR", "NOT"
- But not "XOR"

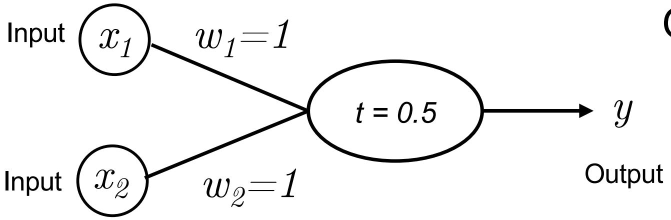
XOR





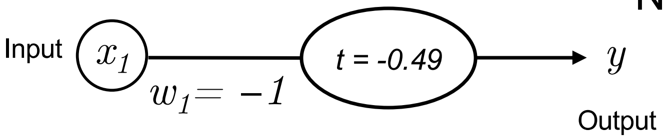
AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

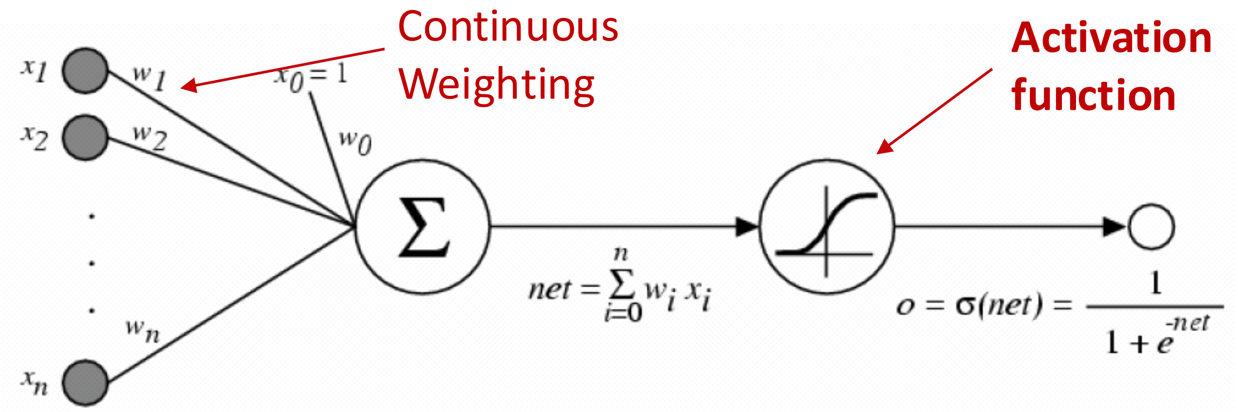


NOT

x_1	y
0	1
1	0

Perceptron

Perceptrons generalize MP Neurons

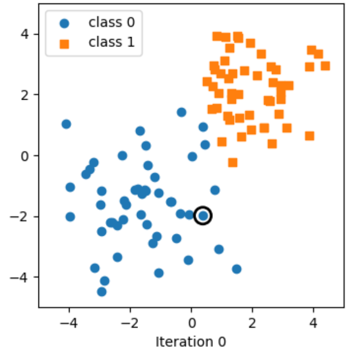


threshold function → **Classic Rosenblatt Perceptron**

sigmoid → **DL "Perceptron" / sigmoid unit**

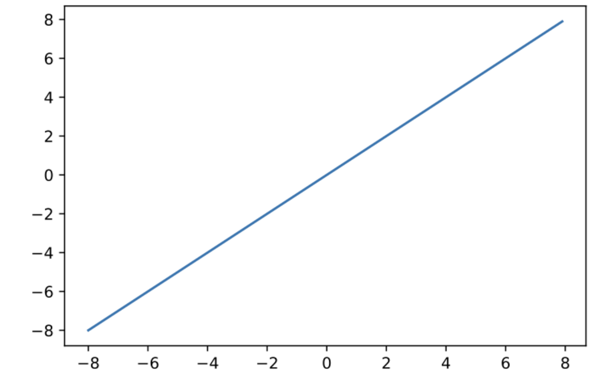
- Many activation functions:
    - Threshold function (perceptron, 1950+)
    - Sigmoid function (before 2000)
    - ReLU function (popular since CNNs)
    - Many variants of ReLU, e.g. leaky ReLU, GeLU

- Unique learning rule for Rosenblatt's Perceptron (but guaranteed convergence in nice settings)
- Does NOT represent XOR

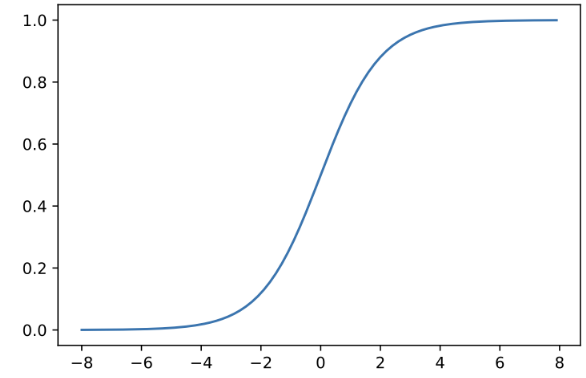


A Selection of Common Activation Functions

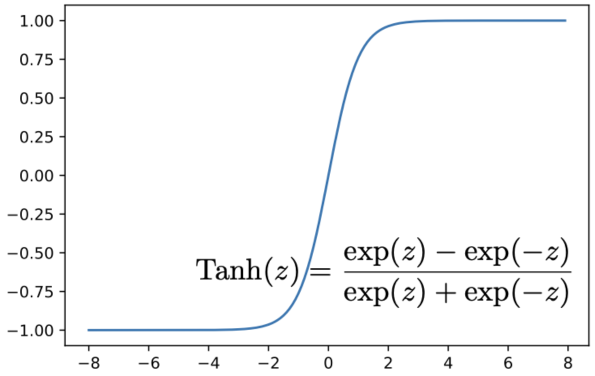
Identity



(Logistic) Sigmoid

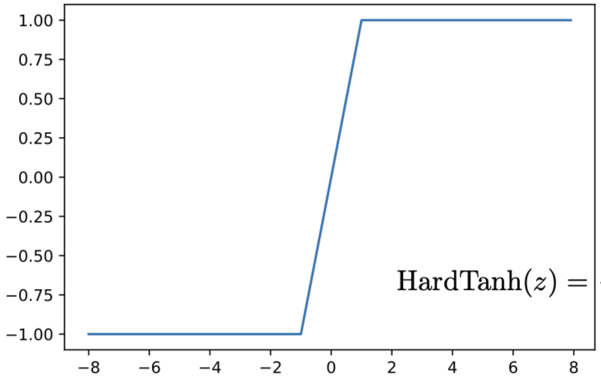


Tanh ("tanH")



$$\mathrm{Tanh}(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

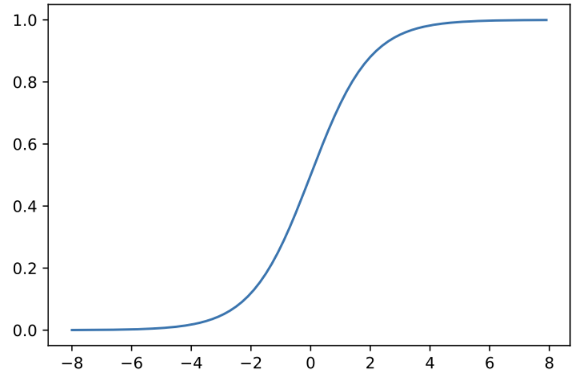
Hard Tanh



$$\mathrm{HardTanh}(z) = \begin{cases} 1 & \text{if } z > 1 \\ -1 & \text{if } z < -1 \\ z & \text{otherwise} \end{cases}$$

A Selection of Common Activation Functions

(Logistic) Sigmoid



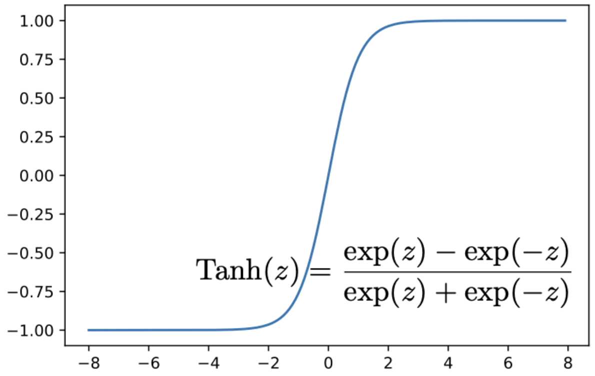
Advantages of Tanh

- Mean centering
- Positive and negative values
- Larger gradients

Also simple derivative:

$$\frac{d}{dz}Tanh(z) = 1 - Tanh(z)^2$$

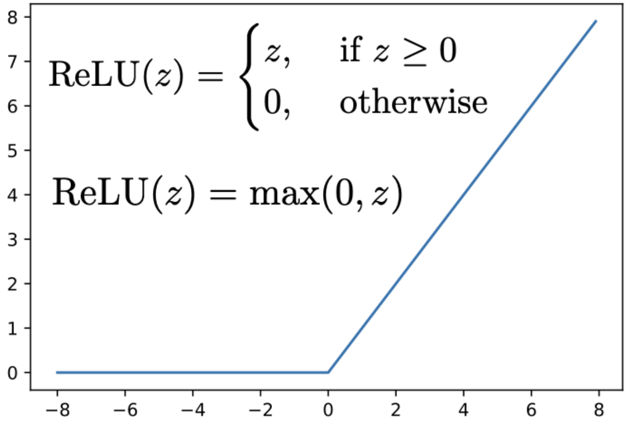
Tanh ("tanH")



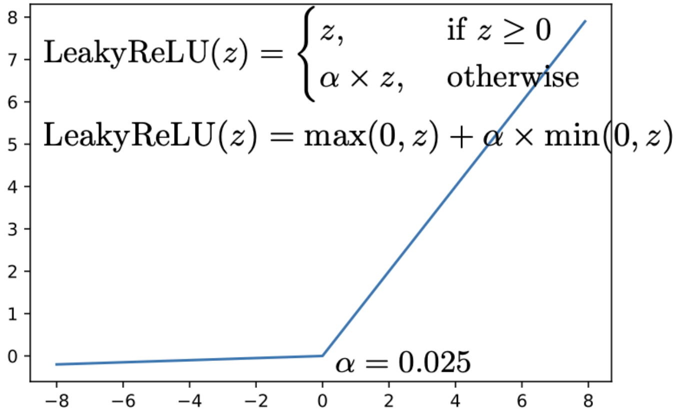
Important to normalize inputs to mean zero and use random weight initialization with **avg. weight centered at zero**

A Selection of Common Activation Functions (cont.)

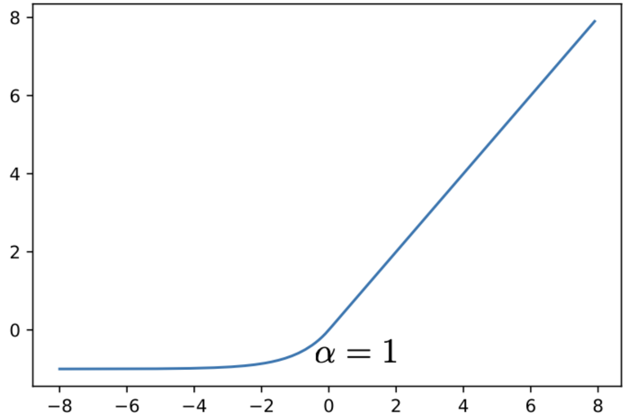
ReLU (Rectified Linear Unit)



Leaky ReLU



ELU (Exponential Linear Unit)



$$\mathrm{ELU}(z) = \max(0, z) + \min(0, \alpha \times (\exp(z) - 1))$$

PReLU (Parameterized Rectified Linear Unit)

here, alpha is a trainable parameter

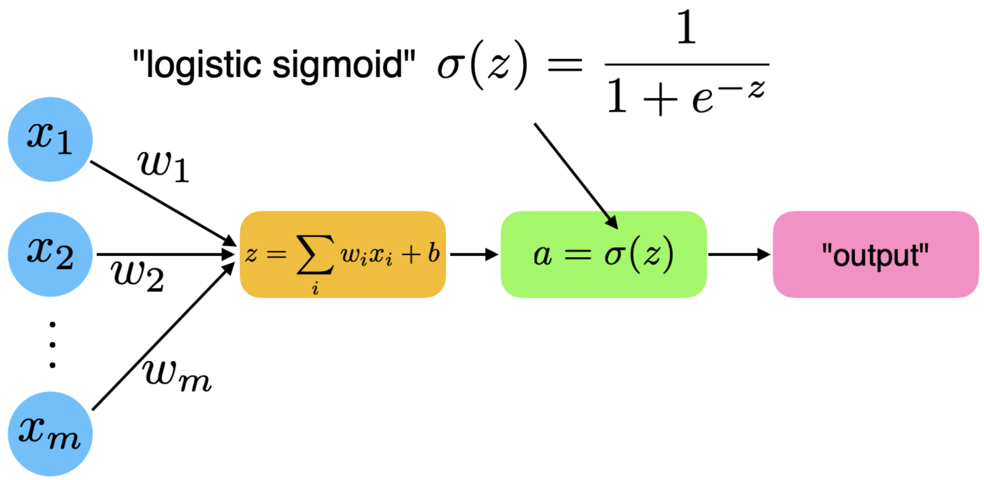
$$\mathrm{PReLU}(z) = \begin{cases} z, & \text{if } z \geq 0 \\ \alpha z, & \text{otherwise} \end{cases}$$

$$\mathrm{PReLU}(z) = \max(0, z) + \alpha \times \min(0, z)$$

Logistic Regression: A Bridge from Perceptron to Probabilistic Model

Logistic Regression Neuron

- For binary classes $y \in \{0, 1\}$



Estimation:

- Given the probability:

$$P(y|\mathbf{x}) = a^y(1-a)^{(1-y)}$$

- Under MLE estimation, we would like to maximize the multi-sample likelihood:

$$P\left(y^{[i]}, \ldots, y^{[n]} | \mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[n]}\right) = \prod_{i=1}^{n} P\left(y^{[i]} | \mathbf{x}^{[i]}\right)$$

$$= \underbrace{\prod_{i=1}^{n} \left(\sigma\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}}}_{\text{Likelihood}}$$

- We are going to optimize via gradient descent, so let's apply the logarithm to separate components:

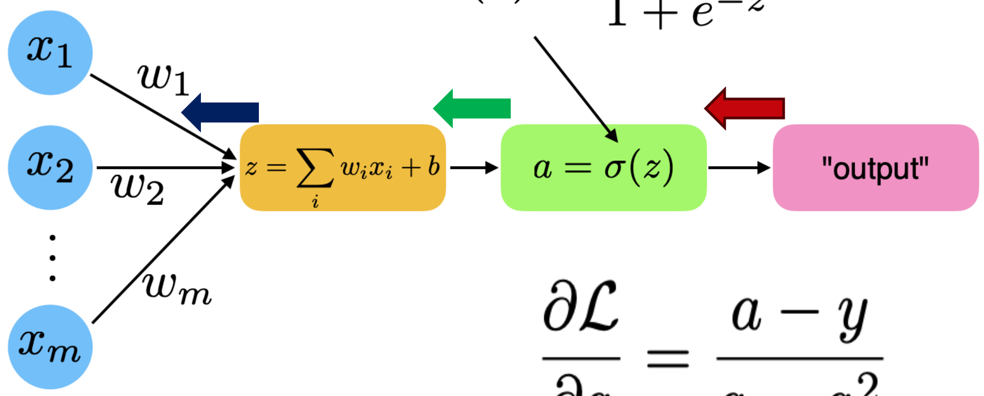
$$l(\mathbf{w}) = \log L(\mathbf{w})$$

$$= \underbrace{\sum_{i=1}^{n} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right)\right]}_{\text{Log-Likelihood}}$$

Logistic Regression: Gradient Descent Learning Rule

- For binary classes $y \in \{0, 1\}$

"logistic sigmoid" $\sigma(z) = \dfrac{1}{1+e^{-z}}$



$$\frac{\partial \mathcal{L}}{\partial w_j} = \boxed{\frac{\partial \mathcal{L}}{\partial a}}\boxed{\frac{da}{dz}}\boxed{\frac{\partial z}{\partial w_j}}$$

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{a-y}{a-a^2}$$

$$\frac{da}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = a \cdot (1-a)$$

$$\frac{\partial z}{w_j} = x_j$$

$$\frac{\partial \mathcal{L}}{\partial z} = a - y$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = (a-y)x_j$$

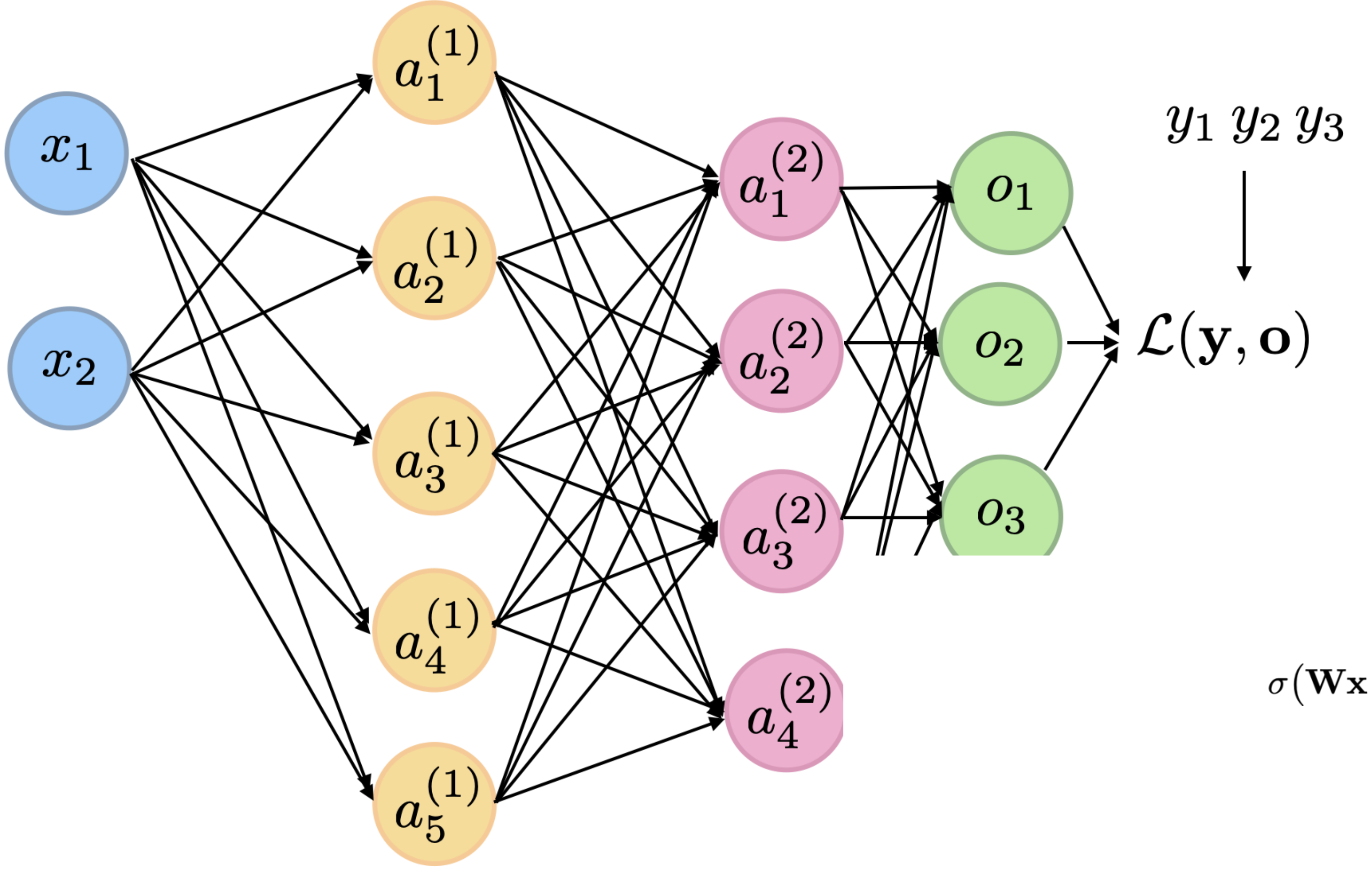
Gradient Descent updates:

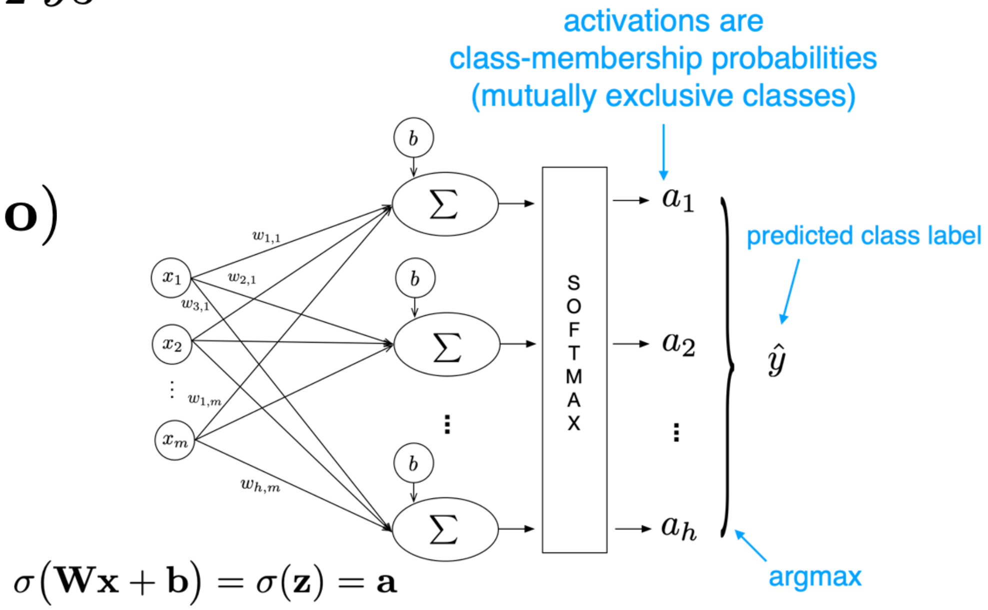
$$\nabla_{\mathbf{w}} \mathcal{L} = -\left(y^{[i]} - \hat{y}^{[i]}\right)\mathbf{x}^{[i]}$$
$$\nabla_b \mathcal{L} = -\left(y^{[i]} - \hat{y}^{[i]}\right)$$

$$\mathbf{w} := \mathbf{w} + \eta \times (-\nabla_{\mathbf{w}} \mathcal{L})$$
$$b := b + \eta \times (-\nabla_b \mathcal{L})$$

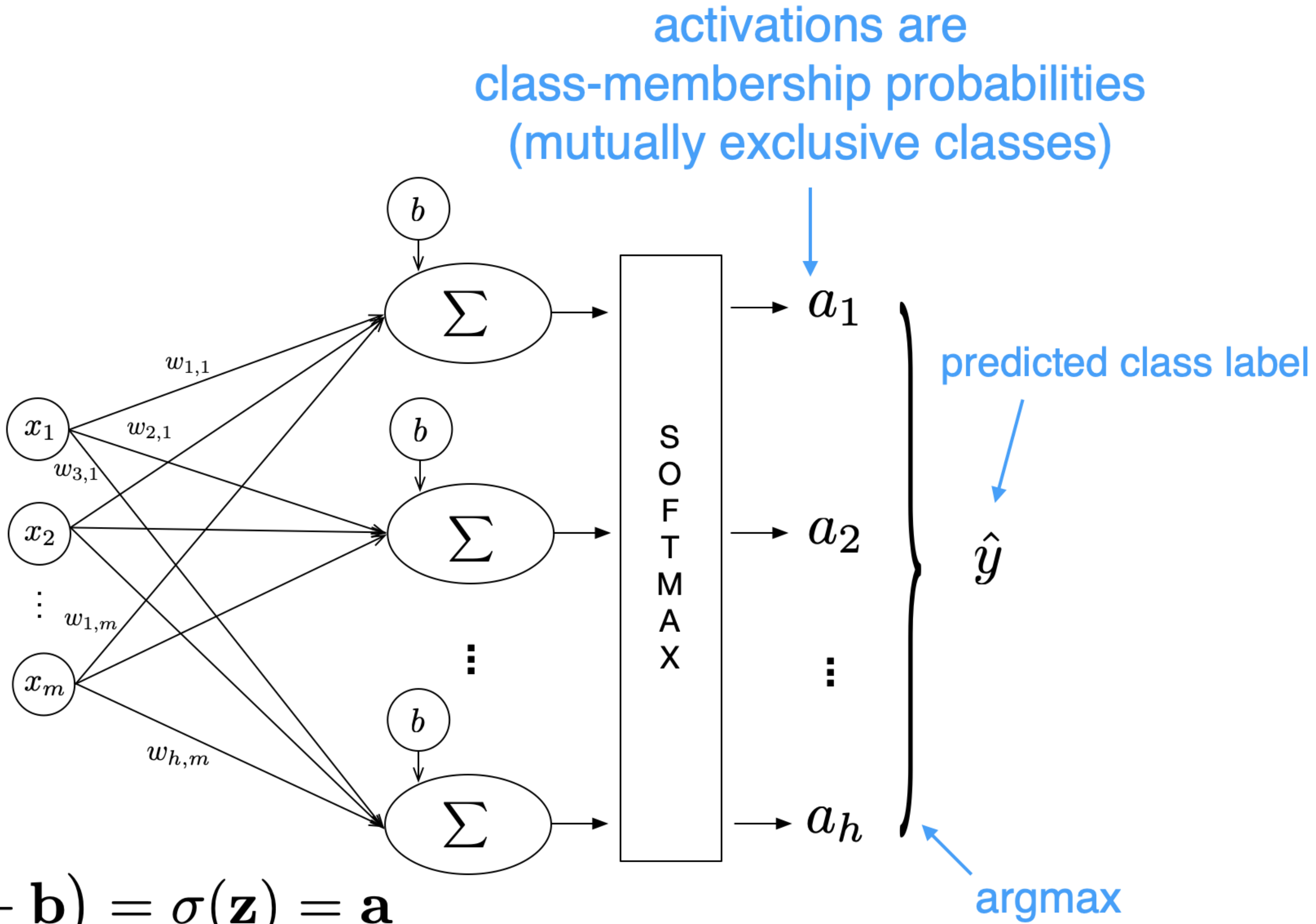
Multilayer Perceptron

- Computation Graph with Multiple Fully-Connected Layers





Multinomial ("Softmax") Logistic Regression



$$\sigma\big(\mathbf{W}\mathbf{x} + \mathbf{b}\big) = \sigma(\mathbf{z}) = \mathbf{a}$$

“Softmax”

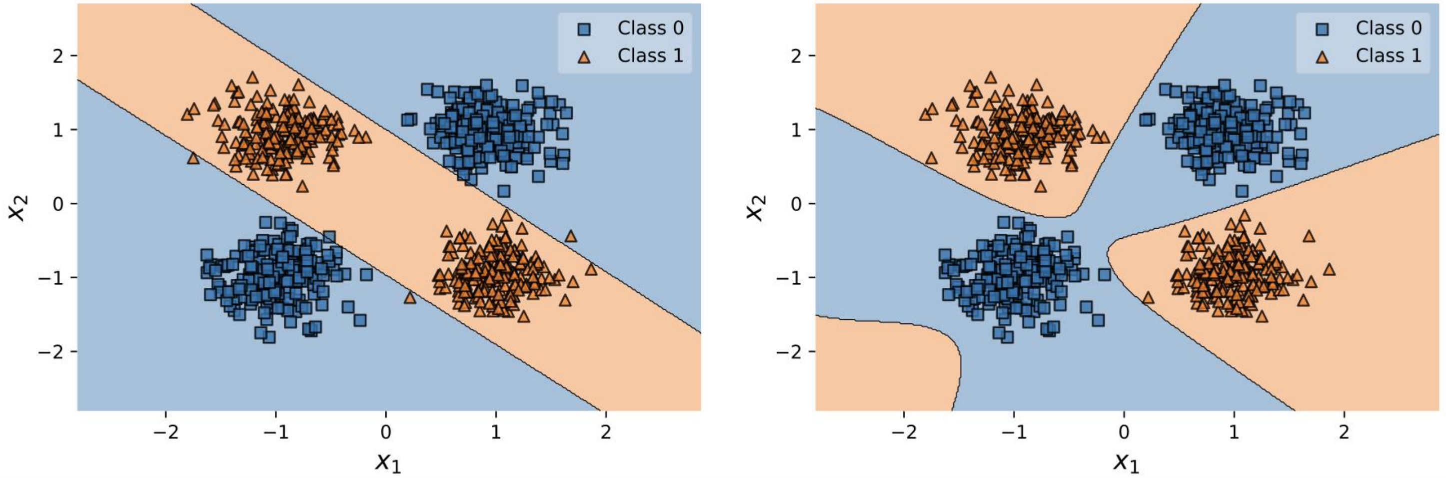
$$P(y = t \mid z_t^{[i]}) = \sigma_{\text{softmax}}(z_t^{[i]}) = \frac{e^{z_t^{[i]}}}{\sum_{j=1}^{h} e^{z_j^{[i]}}}$$

$$t \in \{j \ldots h\}$$

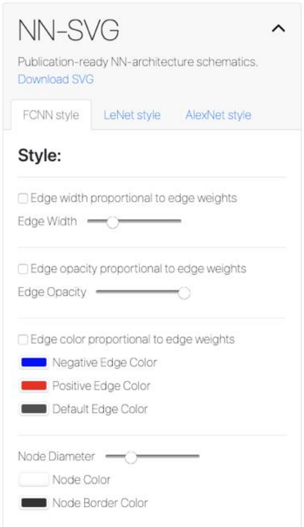
h is the number of class labels

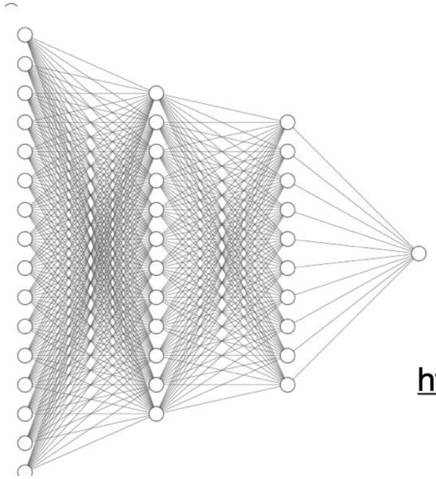
A “soft” (differentiable) version of ”max”

Multilayer Perceptrons Can Solve XOR



Decision boundaries of two different multilayer perceptrons on simulated data solving the XOR problem





https://alexlenail.me/NN-SVG/index.html

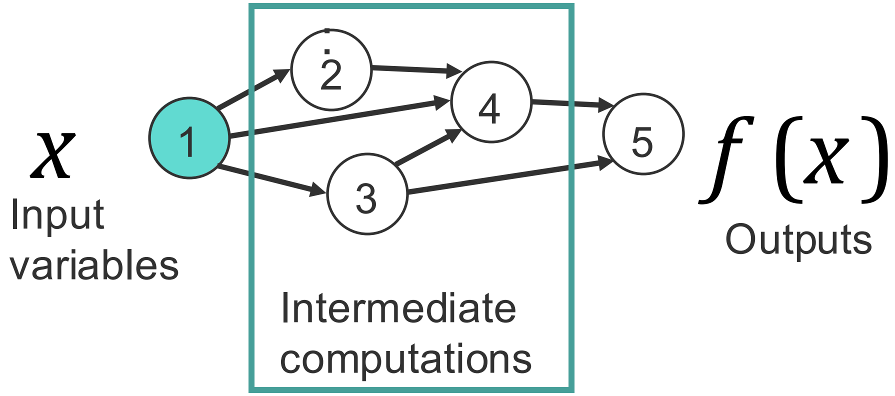
A new problem: Training

- How can we train a multilayer model?
  - No targets / ground truth for the hidden nodes
- Solution: Backpropagation

An algorithm to train models with hidden variables

Backpropagation

- Neural networks are function compositions that can be represented as computation graphs:



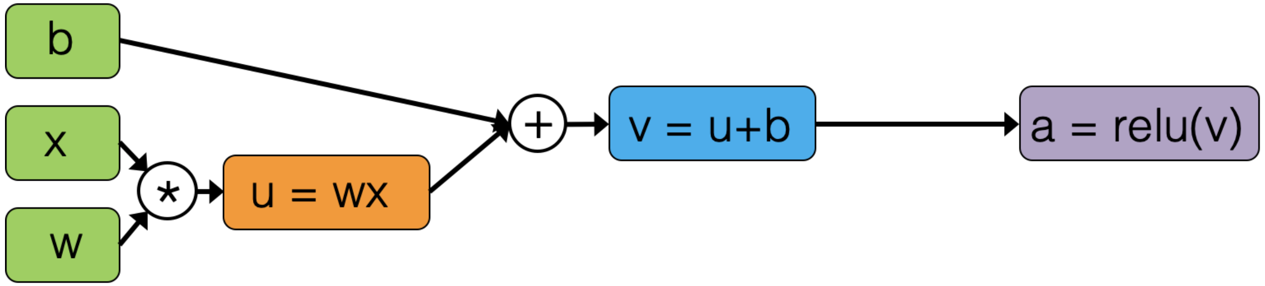
$$\frac{\partial f_n}{\partial x} =$$

- By applying the chain rule, and working in reverse order, we get:

$$\frac{\partial f_n}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \frac{\partial f_{i_1}}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \sum_{i_2 \in \pi(i_1)} \frac{\partial f_{i_1}}{\partial f_{i_2}} \frac{\partial f_{i_1}}{\partial x} = \dots$$

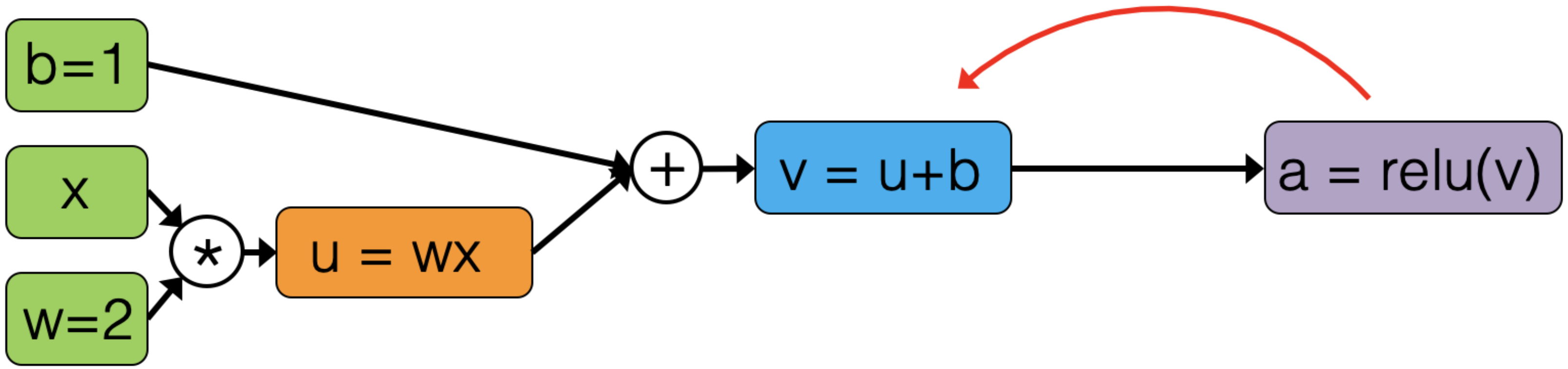
Example: Backpropagation through ReLU

$$a(x, w, b) = relu(\underbrace{\underbrace{w \cdot x}_{u} + b}_{v})$$

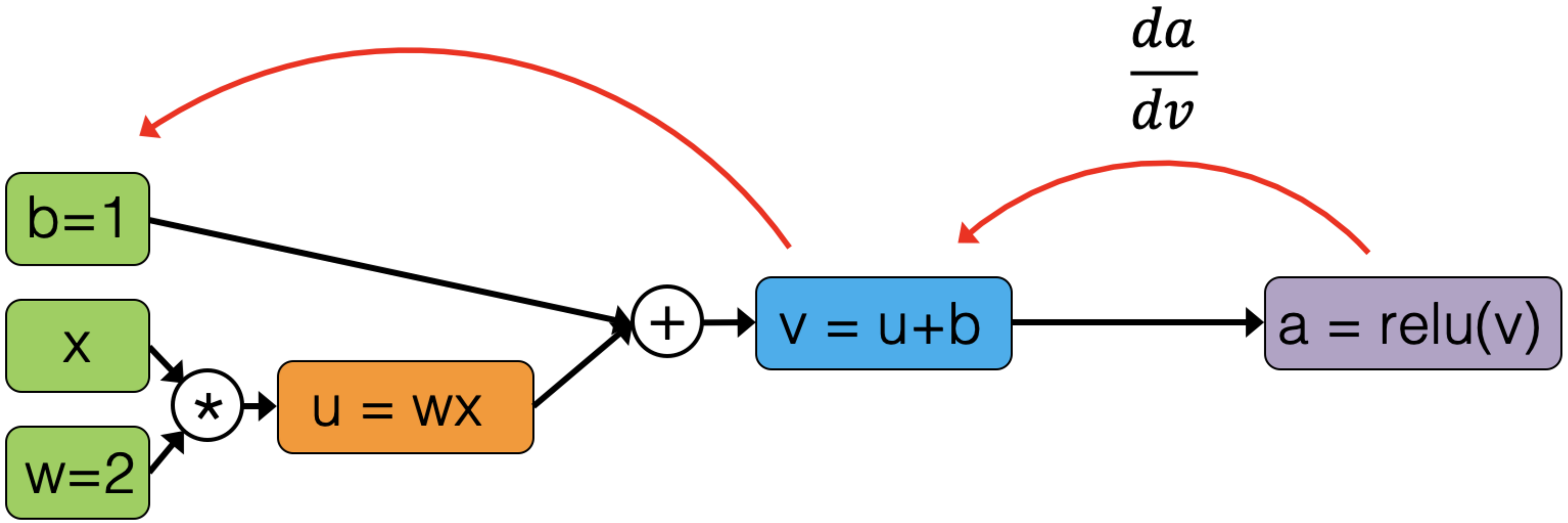


Example: Backpropagation through ReLU

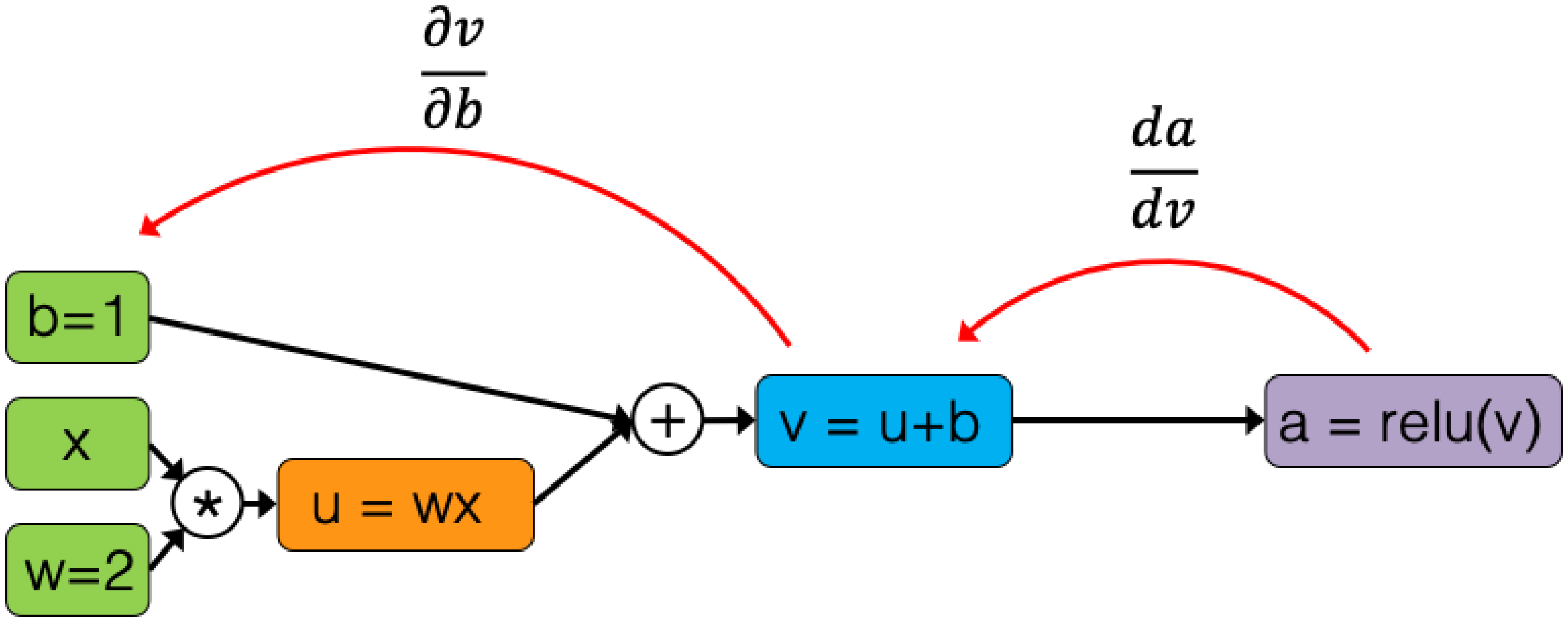
$$\frac{da}{dv}$$



Example: Backpropagation through ReLU

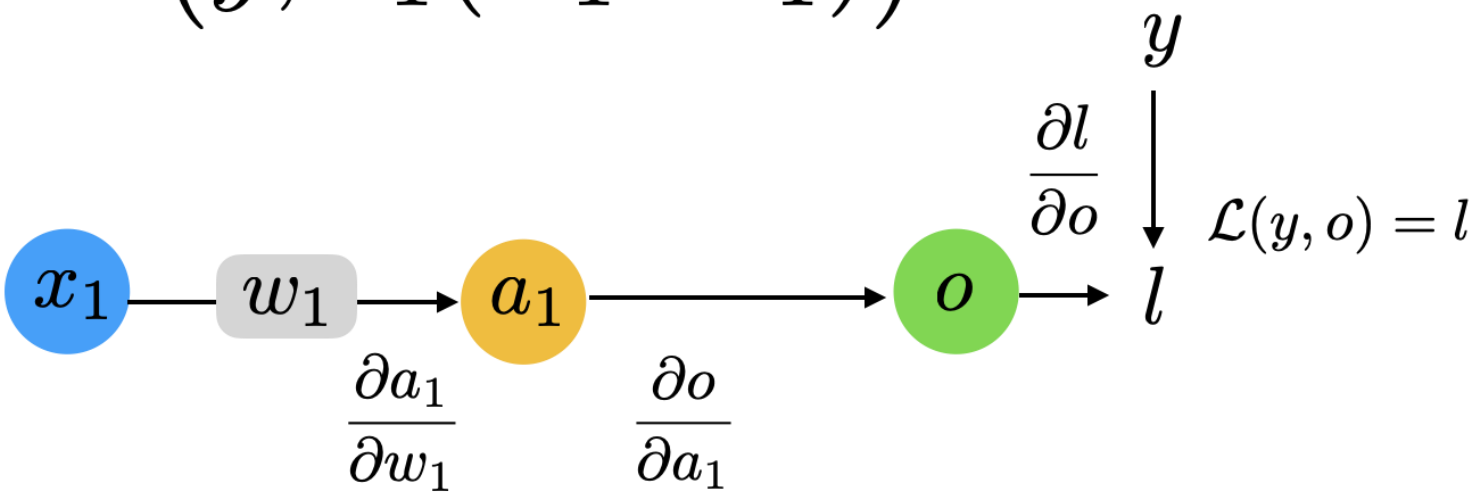


Example: Backpropagation through ReLU



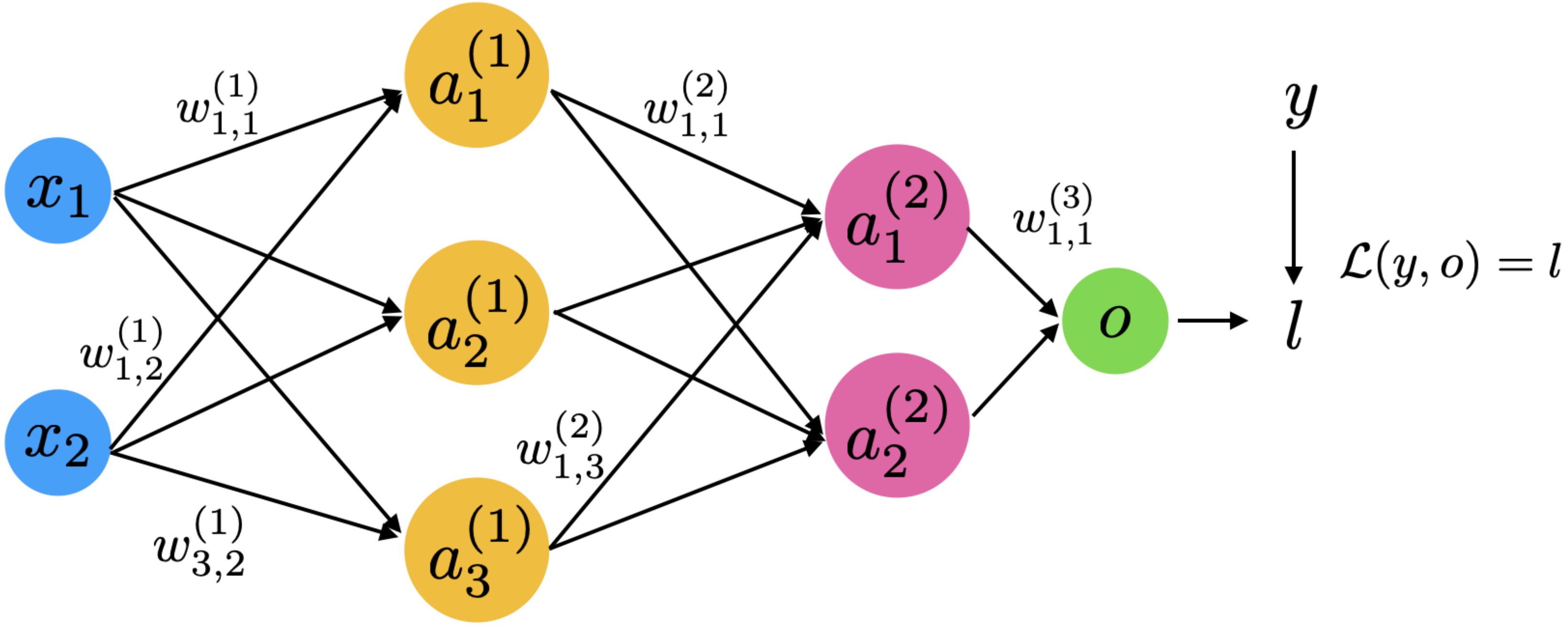
Backpropagation through chains

$$\mathcal{L}(y, \sigma_1(w_1 \cdot x_1))$$



$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$ (univariate chain rule)

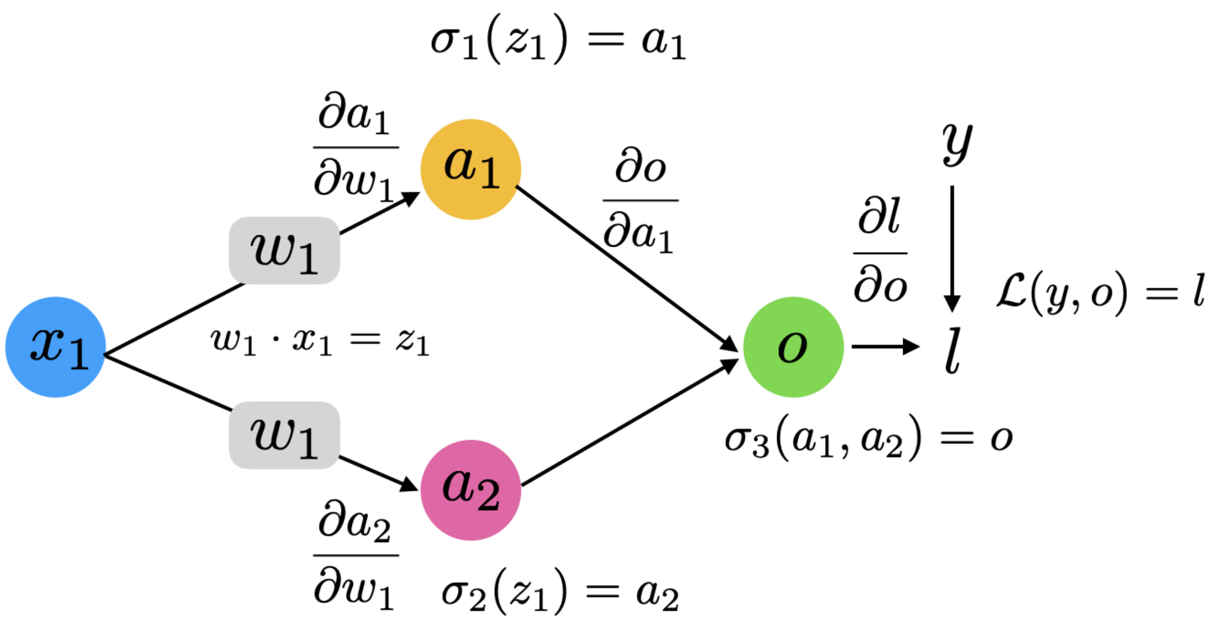
Backpropagation through fully-connected net



$$\frac{\partial l}{\partial w_{1,1}^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$

Backpropagation through weight-sharing archs

$$\mathcal{L}\left(y, \sigma_3\left[\sigma_1(w_1 \cdot x_1), \sigma_2(w_1 \cdot x_1)\right]\right)$$



Upper path

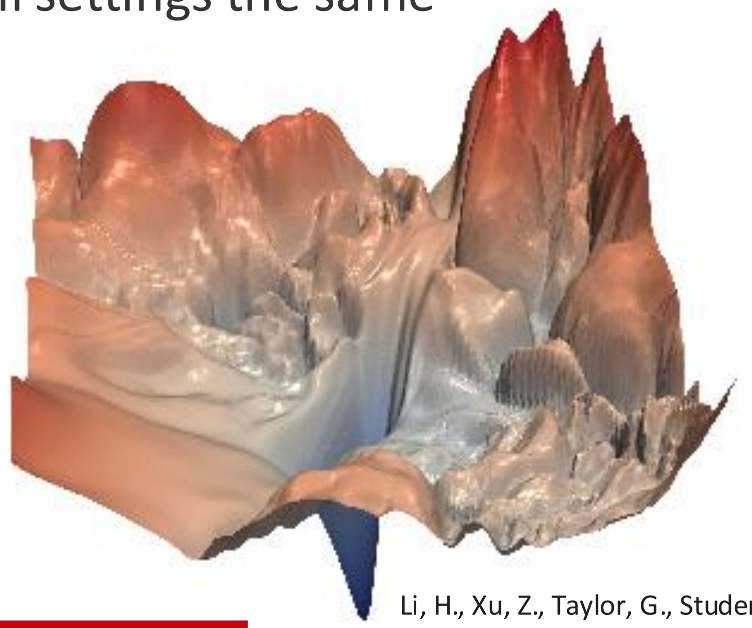
$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_1}$$ (multivariable chain rule)

Lower path

Improvements to optimization

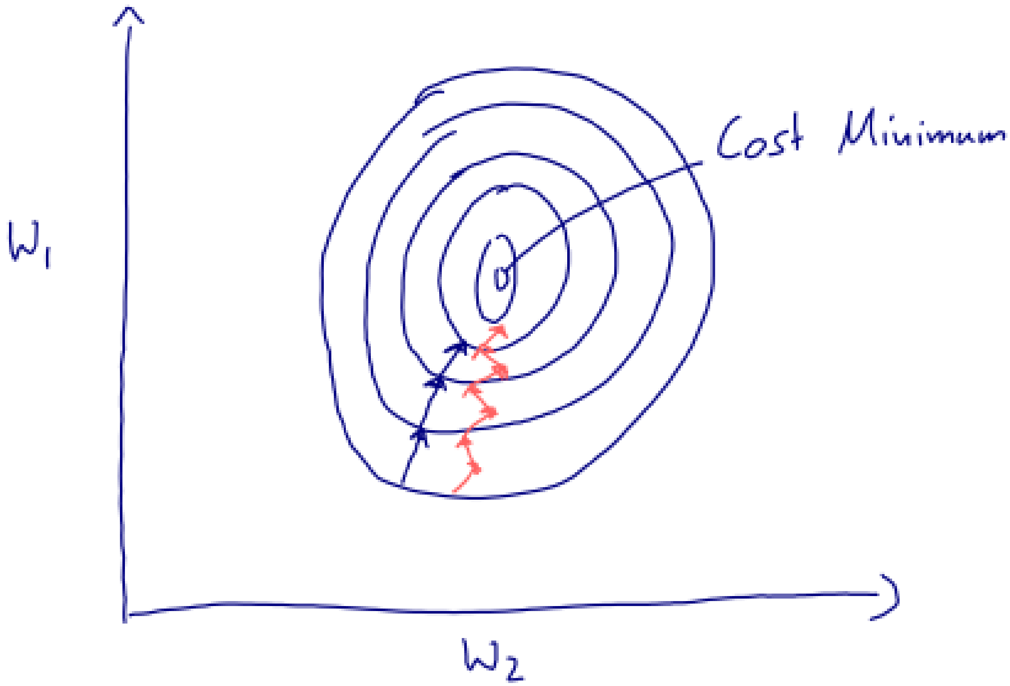
Our Loss is Not Convex Anymore

- Linear regression, Adaline, Logistic Regression, and Softmax Regression have convex loss functions
- But our deep loss is no longer convex (most of the time)
  - In practice, we usually end up at different local minima if we repeat the training (e.g. by changing the random seed for weight initialization or shuffling the dataset while leaving all settings the same



Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T., 2018. Visualizing the loss landscape of neural nets. In Advances in Neural Information Processing Systems (pp. 6391-6401).

Minibatch Training Recap



- Minibatch learning is a form of stochastic gradient descent
- Each minibatch can be considered a sample drawn from the training set (where the training set is in turn a sample drawn from the population)
- Hence, the gradient is **noisier**

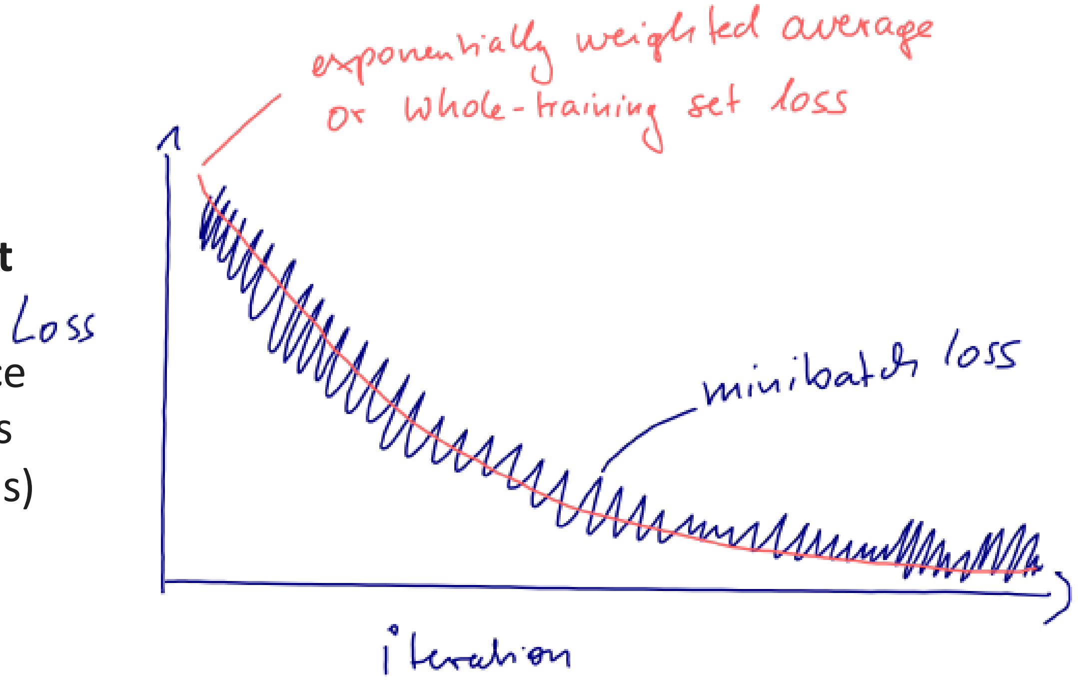
A **noisy** gradient can be:

- **good**: chance to escape local minima
- **bad**: can lead to extensive oscillation

Learning Rate Decay

- Batch effects -- minibatches are samples of the training set, hence minibatch loss and gradients are approximations
- Hence, we usually get oscillations
- To dampen oscillations towards the end of the training, we can **decay the learning rate**

- Danger of learning rate is to decrease the learning rate too early
- Practical tip: try to **train the model without learning rate decay first**, then add it later
- You can also use the validation performance (e.g., accuracy) to judge whether lr decay is useful (as opposed to using the training loss)



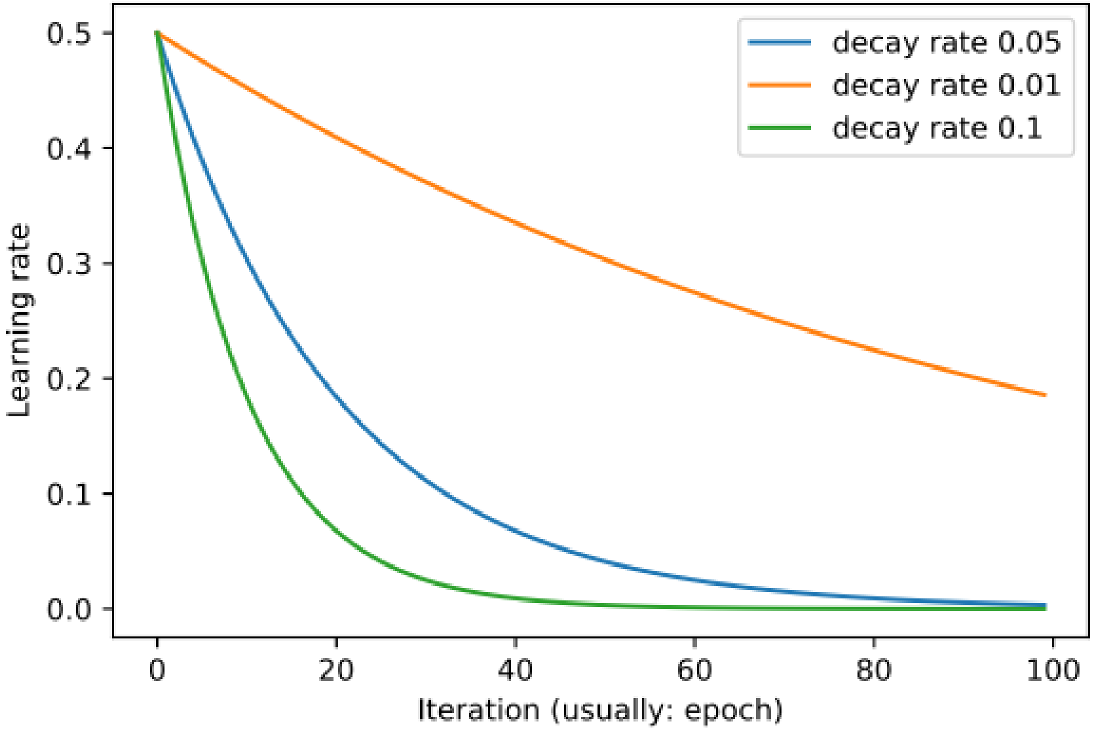
Learning Rate Decay

Most common variants for lr decay:

1. Exponential Decay:

$$\eta_t \coloneqq \eta_0 \mathrm{e}^{-\mathrm{k}\cdot t}$$

where k is the decay rate



Learning Rate Decay

Most common variants for lr decay:

1. Exponential Decay:

$$\eta_t \coloneqq \eta_0 \mathrm{e}^{-\mathrm{k}\cdot t}$$

where k is the decay rate

2. Halving the learning rate:

$$\eta_t \coloneqq \eta_{t-1}/2$$

where t is a multiple of T_0 (e.g. $T_0 = 100$)

3. Inverse decay:

$$\eta_t \coloneqq \frac{\eta_0}{1 + k \cdot t}$$

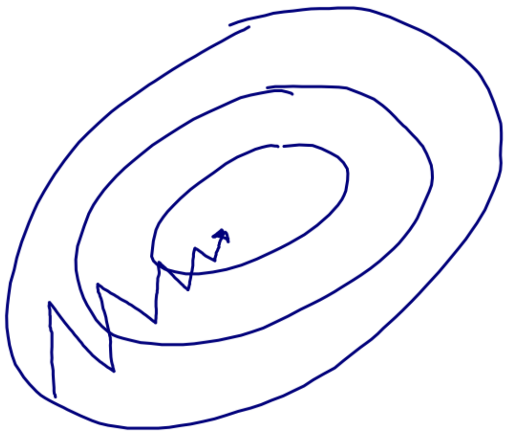
Training with "Momentum"

- Main idea: Let's dampen oscillations by using "velocity" (the speed of the "movement" from previous updates)

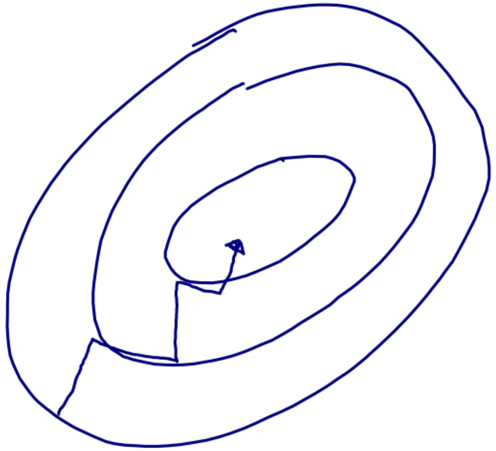
https://www.asherworldturns.com/zorbing-new-zealand/

Training with “Momentum”

- Main idea: Let’s dampen oscillations by using “velocity” (the speed of the “movement” from previous updates)



Without momentum



With momentum

Key take-away: Not only move in the (opposite) direction of the gradient, but also move in the “**weighted averaged**" direction of the last few updates

Training with “Momentum”

Often referred to as "velocity" ***V***

"velocity" from the previous iteration

$$\Delta w_{i,j}(t) := \alpha \cdot \Delta w_{i,j}(t-1) + \eta \cdot \frac{\partial \mathcal{L}}{\partial w_{i,j}}(t)$$

Usually, we choose a momentum rate between 0.9 and 0.999; you can think of it as a "friction" or "dampening" parameter

Regular partial derivative/ gradient multiplied by learning rate at current time step *t*

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural Networks : The Official Journal of the International Neural Network Society, 12(1), 145–151. http://doi.org/10.1016/S0893-6080(98)00116-6

Nesterov: A Better Momentum

We already know where the momentum part will push us in this step. Let's calculate the **new gradient** with that update in mind:

Before:

$$\Delta \mathbf{w}_t := \alpha \cdot \Delta \mathbf{w}_{t-1} + \eta \cdot \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t)$$
$$\mathbf{w}_{t+1} := \mathbf{w}_t - \Delta \mathbf{w}_t$$

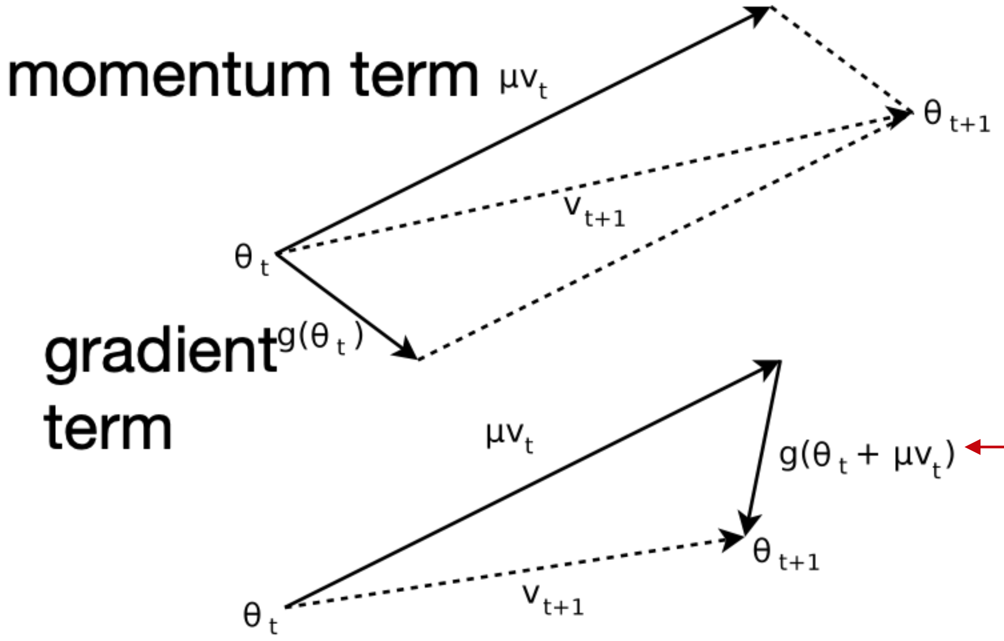
Nesterov:

$$\Delta \mathbf{w}_t := \alpha \cdot \Delta \mathbf{w}_{t-1} + \eta \cdot \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t - \alpha \cdot \Delta \mathbf{w}_{t-1})$$
$$\mathbf{w}_{t+1} := \mathbf{w}_t - \Delta \mathbf{w}_t$$

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543– 547.

Sutskever, I., Martens, J., Dahl, G. E., & Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. ICML (3), 28(1139-1147), 5.

Nesterov: A Better Momentum



correction term (gradient of the point where you would have ended up via the standard momentum method)

Figure 1. **(Top)** Classical Momentum **(Bottom)** Nesterov Accelerated Gradient

Sutskever, I., Martens, J., Dahl, G. E., & Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. ICML (3), 28(1139-1147), 5.

Adaptive Learning Rates

Many different flavors of adapting the learning rate

Rule of thumb:

1. decrease learning if the gradient changes its direction
2. increase learning if the gradient stays consistent

RMSProp

- Unpublished (but very popular) algorithm by Geoff Hinton
- Based on Rprop [1]
- Very similar to another concept called AdaDelta
- **Main idea:** divide learning rate by an exponentially decreasing moving average of the squared gradients
  - **RMS = "Root Mean Squared"**
  - Takes into account that gradients can vary widely in magnitude
  - Damps oscillations like momentum (in practice, works better)

[1] Igel, Christian, and Michael Hüsken. "Improving the Rprop learning algorithm." Proceedings of the Second International ICSC Symposium on Neural Computation (NC 2000). Vol. 2000. ICSC Academic Press, 2000.

ADAM (Adaptive Moment Estimation)

- Probably the most widely used optimization algorithm in DL
- Combination of momentum + RMSProp

Momentum-like term:

$$\underset{m_t}{\cancel{\Delta w_{i,j}(t)}} := \alpha \cdot \overset{m_{t-1}}{\cancel{\Delta w_{i,j}(t-1)}} + \eta \cdot \frac{\partial \mathcal{L}}{\partial w_{i,j}}(t)$$

$$m_t := \alpha \cdot m_{t-1} + (1-\alpha) \cdot \frac{\partial \mathcal{L}}{\partial w_{i,j}}(t)$$

RMSProp term:

$$r := \beta \cdot MeanSquare(w_{i,j}, t-1) + (1-\beta)\left(\frac{\partial \mathcal{L}}{\partial w_{i,j}(t)}\right)^2$$

ADAM update:

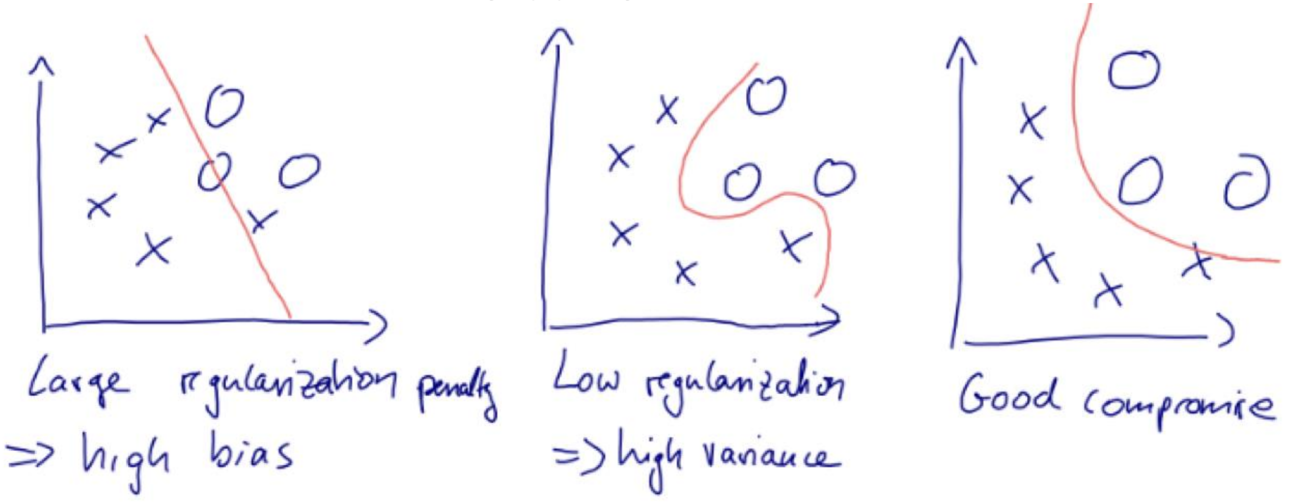
$$w_{i,j} := w_{i,j} - \eta \frac{m_t}{\sqrt{r} + \epsilon}$$

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Where we are…

- Good news: We can solve non-linear problems!
- Bad news: Our multilayer neural networks have lots of parameters and it's easy to overfit the data…

Next time:



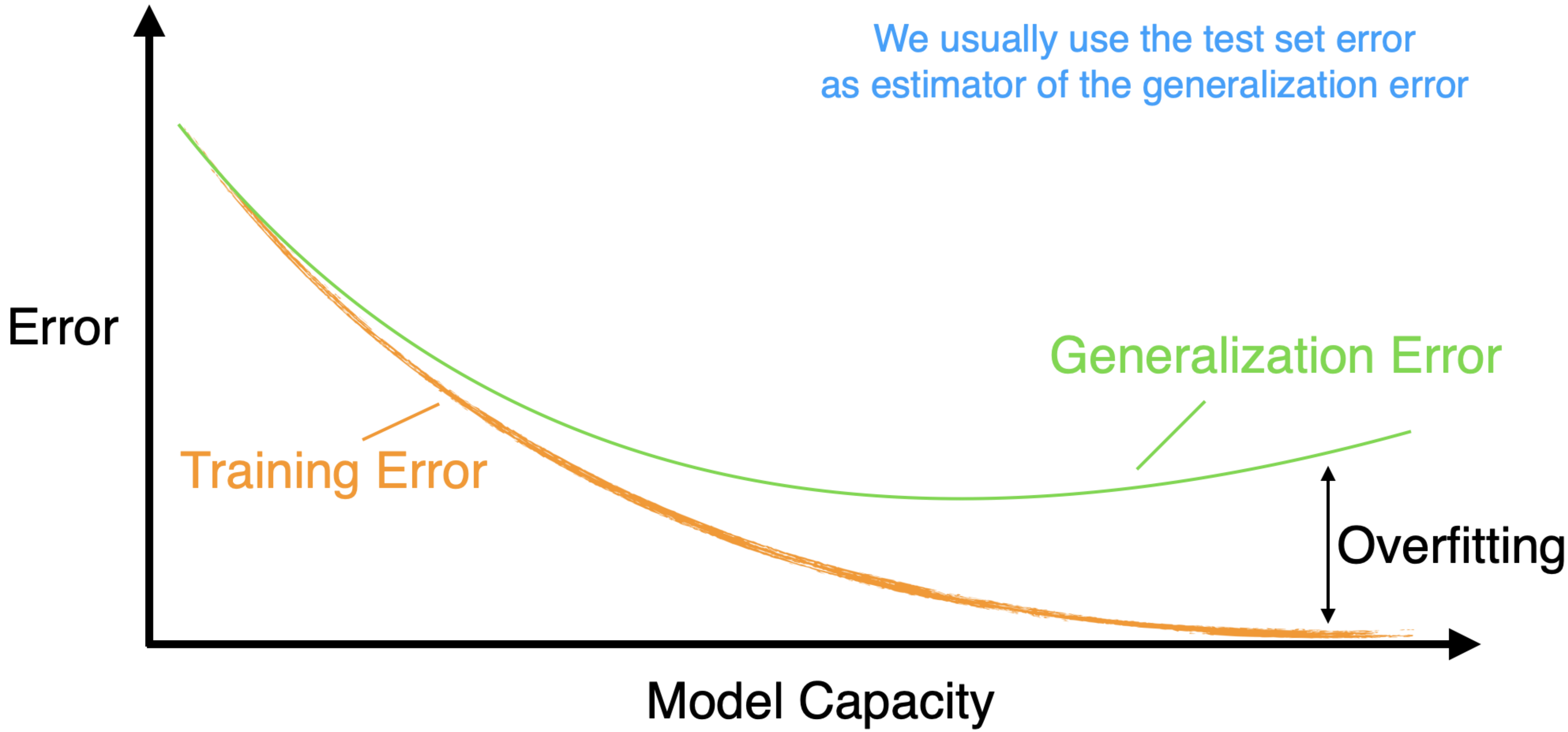
Regularization

Parameters vs Hyperparameters

weights (weight parameters)
biases (bias units)

minibatch size
data normalization schemes
number of epochs
number of hidden layers
number of hidden units
learning rates
(random seed, why?)
loss function
various weights (weighting terms)
activation function types
regularization schemes (more later)
weight initialization schemes (more later)
optimization algorithm type (more later)
...

Overfitting and Underfitting



Bias-Variance Decomposition

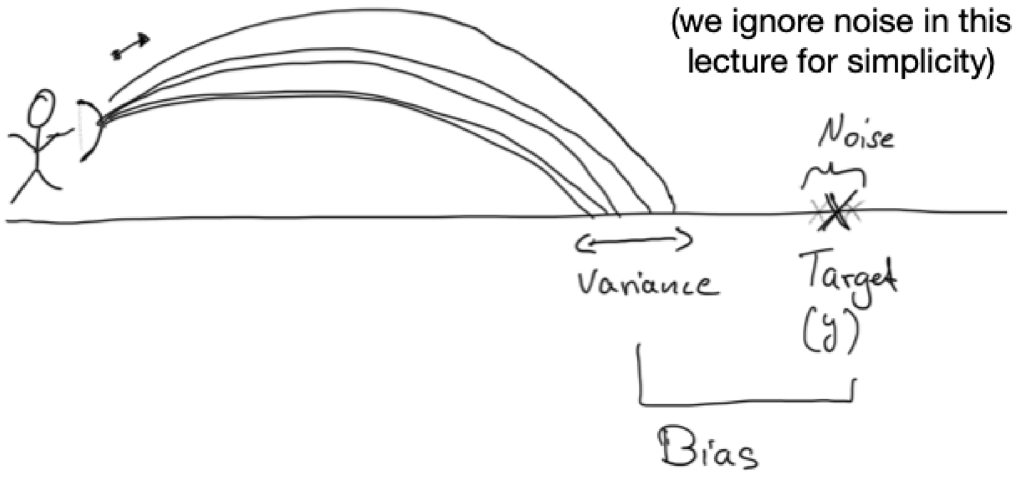
General Definition:

$$\text{Bias}_\theta[\hat{\theta}] = E[\hat{\theta}] - \theta$$

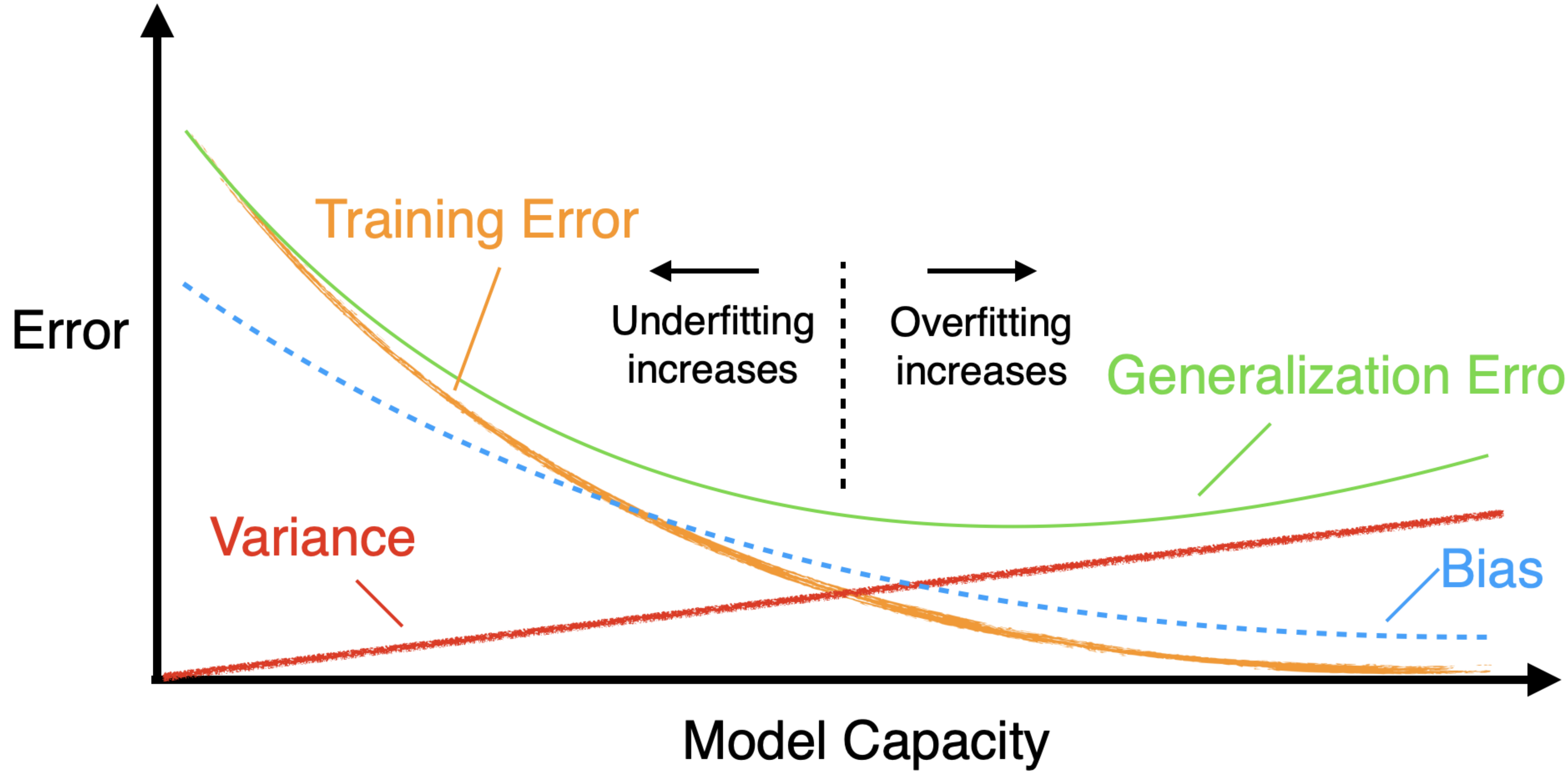
$$\text{Var}_\theta[\hat{\theta}] = E\left[\hat{\theta}^2\right] - (E[\hat{\theta}])^2$$

Intuition:

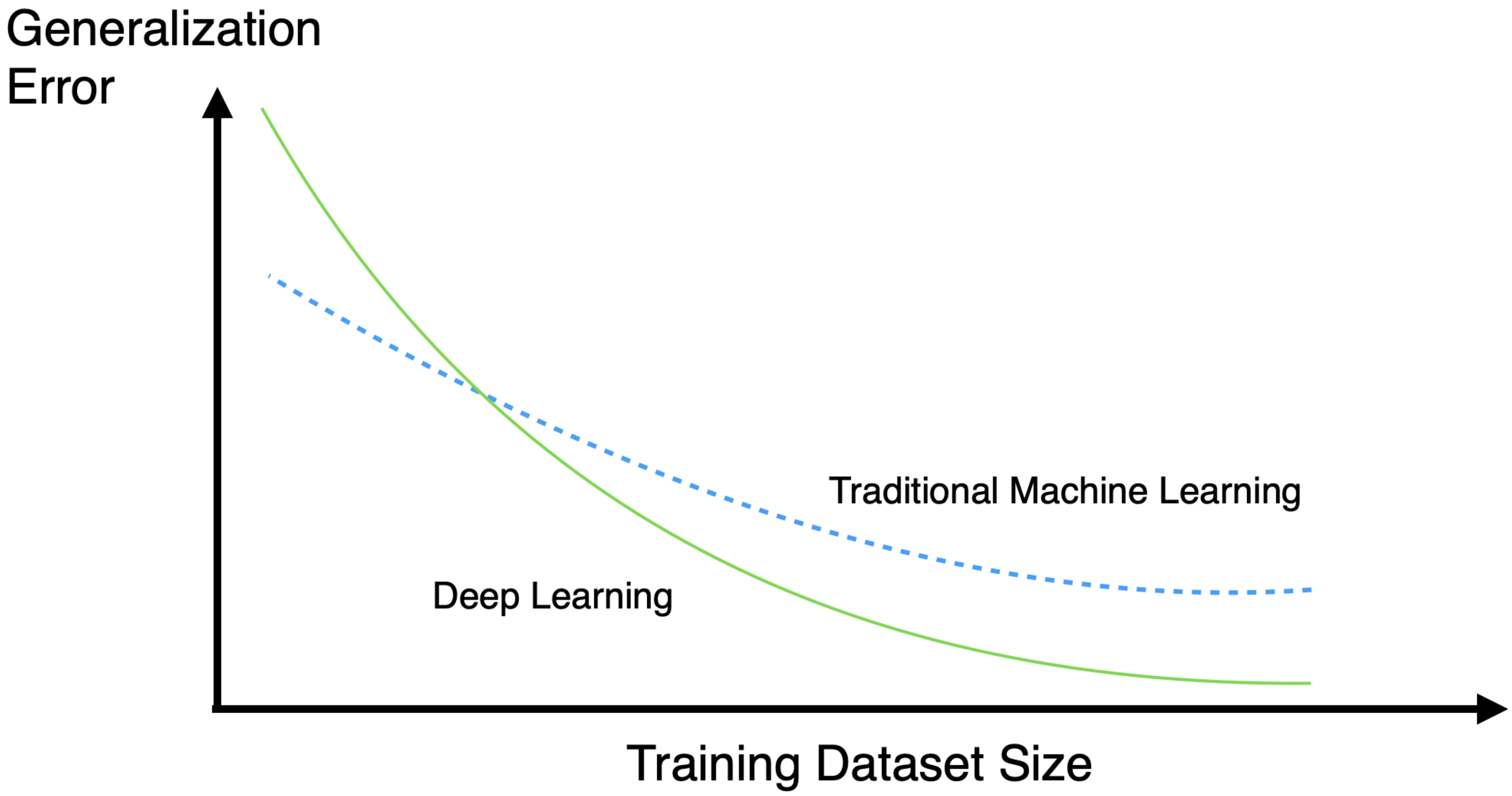
(we ignore noise in this lecture for simplicity)



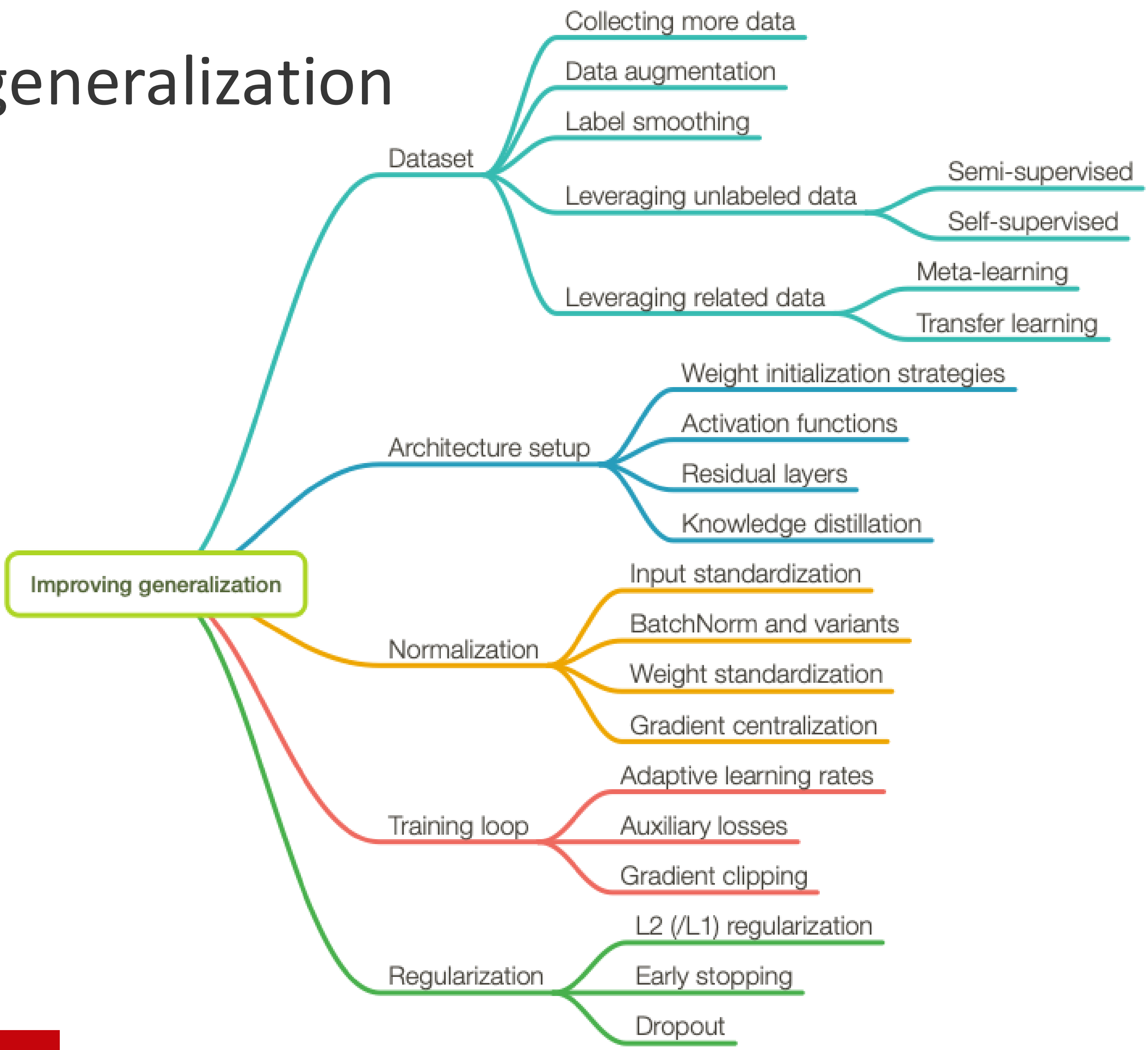
Bias-Variance & Overfitting-Underfitting



Deep Learning works best with large datasets



Many ways to improve generalization



General Strategies to Avoid Overfitting

- Collecting more data, especially high-quality data, is best & always recommended
  - Alternatively: semi-supervised learning, transfer learning, and self-supervised learning
- Data augmentation is helpful
  - Usually requires prior knowledge about data or tasks
- Reducing model capacity can help

Data Augmentation

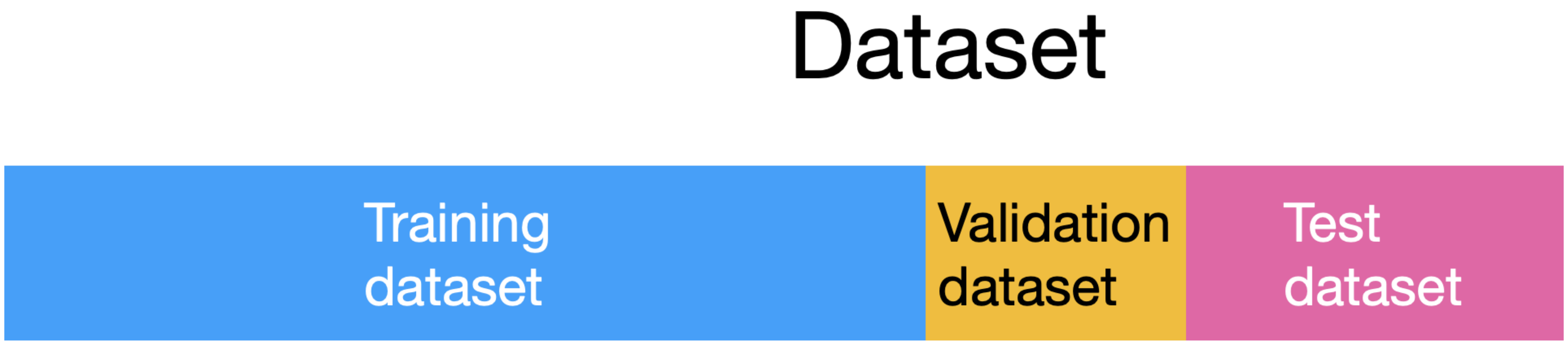
- **Key Idea:** If we know the label shouldn't depend on a transformation h(x), then we can generate new training data $h(x^i), y^i$
- But we must already know something that our outcome doesn't depend on
- Example: image classification
  - rotation, zooming, sepia filter, etc.

Reduce Network's Capacity

- **Key Idea:** The simplest model that matches the outputs should generalize the best
- Choose a smaller architecture: fewer hidden layers & units, add dropout, use ReLU + L1 penalty to prune dead activations,e tc.
- Enforce smaller weights: Early stopping, L2 norm penalty
- Add noise: Dropout
- Note: With recent LLMs and foundation models, it's possible to use a large pretrained model and perform efficient **fine-tuning** (updating small number of parameters of a large model)

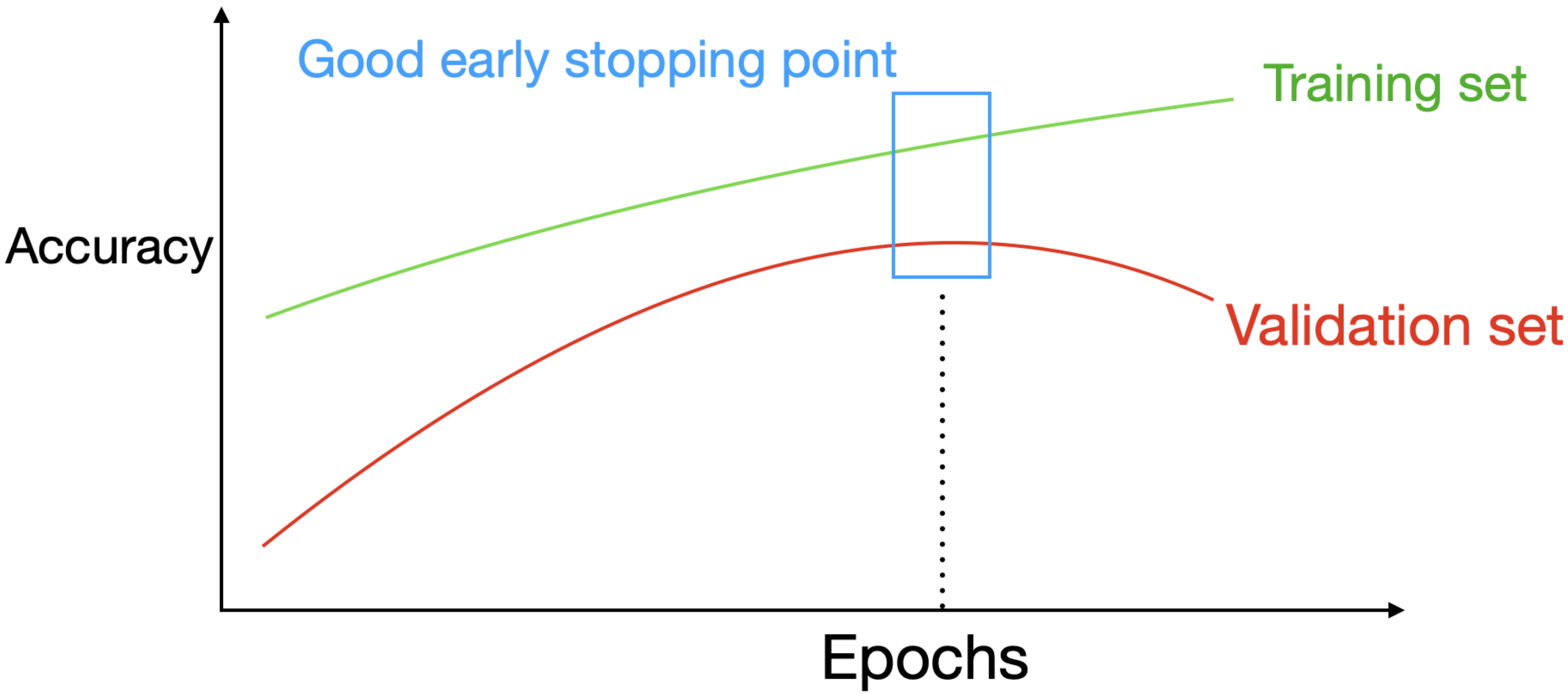
Early Stopping

- **Step 1:** Split your dataset into 3 parts (as always)
  - Use test set only once at the end
  - Use validation accuracy for tuning



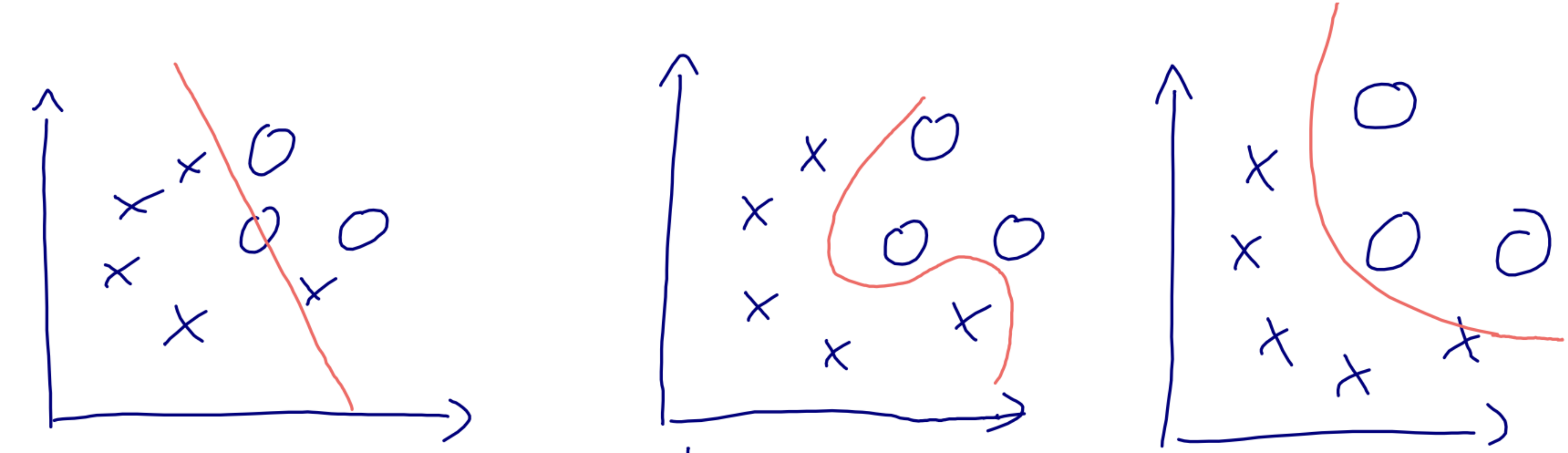
Early Stopping

- **Step 2:** Stop training early
  - Reduce overfitting by observing the training/validation accuracy gap during training and then stop at the “right” point



Effect of Regularization on Decision Boundary

Assume a nonlinear model



Large regularization penalty
=> high bias

Low regularization
=> high variance

Good compromise

L2 regularization for Multilayer Neural Networks

$$\text{L2-Regularized-Cost}_{\mathbf{w},\mathbf{b}} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y^{[i]}, \hat{y}^{[i]}) + \frac{\lambda}{n}\sum_{l=1}^{L}||\mathbf{w}^{(l)}||_F^2$$

sum over layers

where $||\mathbf{w}^{(l)}||_F^2$ is the Frobenius norm (squared):

$$||\mathbf{w}^{(l)}||_F^2 = \sum_i \sum_j (w_{i,j}^{(l)})^2$$

L2 regularization for Multilayer Neural Networks

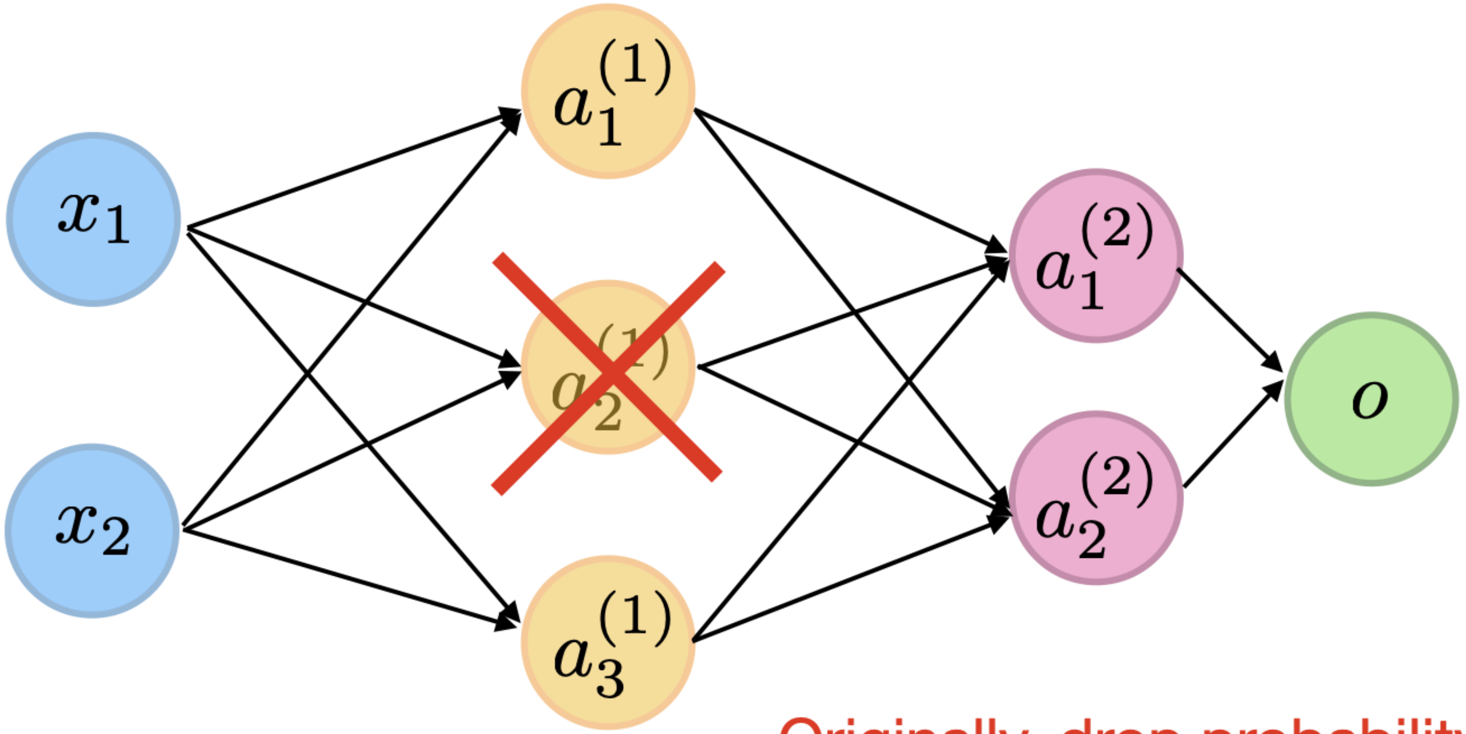
Regular gradient descent update:

$$w_{i,j} := w_{i,j} - \eta \frac{\partial \mathcal{L}}{\partial w_{i,j}}$$

Gradient descent update with L2 regularization:

$$w_{i,j} := w_{i,j} - \eta \left(\frac{\partial \mathcal{L}}{\partial w_{i,j}} \boxed{+ \frac{2\lambda}{n} w_{i,j}} \right)$$

Dropout



Originally, drop probability 0.5

(but 0.2-0.8 also common now)

Dropout

- How do we drop node activations practically / efficiently?

<u>Bernoulli Sampling (during training):</u>

- p := drop probability
- v := random sample from uniform distribution in range [0, 1]
- $\forall i \in \mathbf{v} : v_i := 0 \text{ if } v_i < p \text{ else } 1$
- $\mathbf{a} := \mathbf{a} \odot \mathbf{v}$ *($p \times 100\%$ of the activations a will be zeroed)*

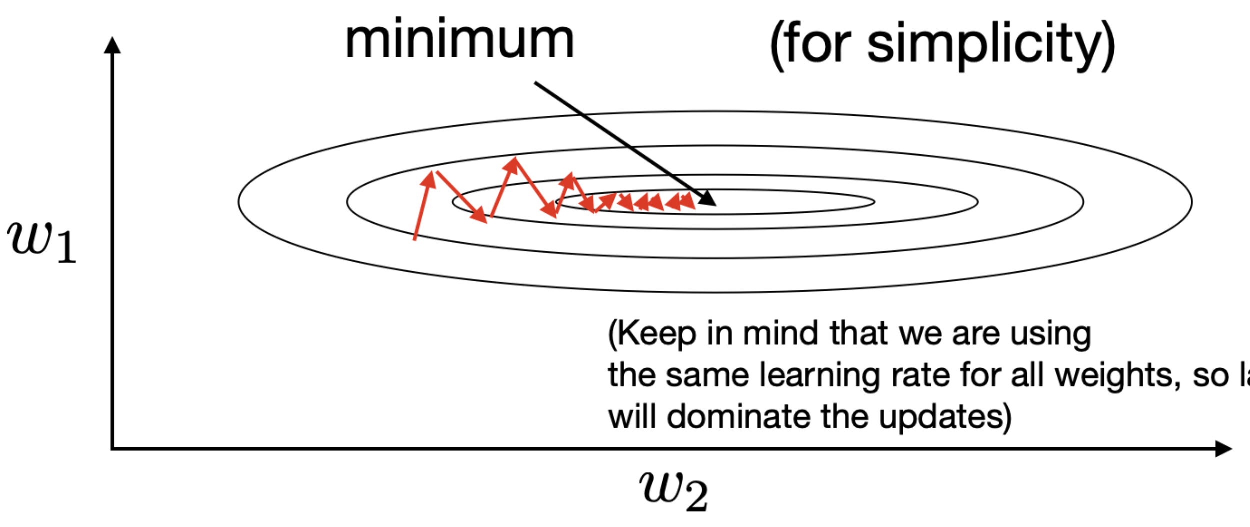
Then, after training when making predictions (during "inference")

scale activations via $\mathbf{a} := \mathbf{a} \odot (1 - p)$

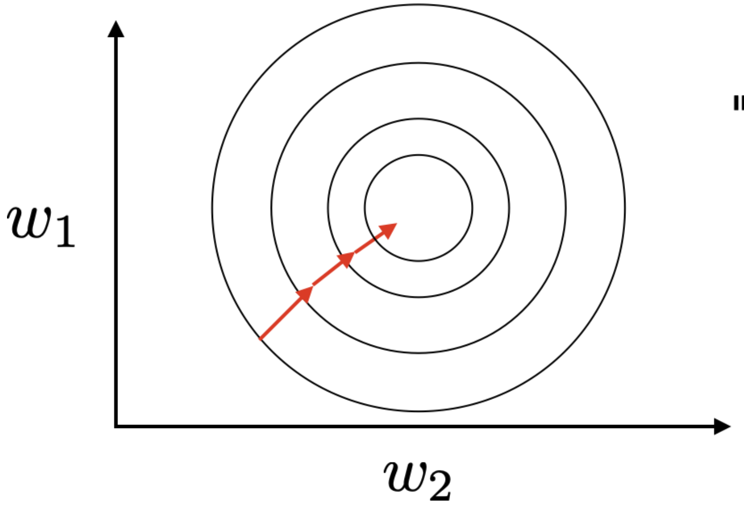
Normalization

Normalization and gradient descent

Surface of a convex cost function
(for simplicity)



(Keep in mind that we are using
the same learning rate for all weights, so large parameters
will dominate the updates)



"Standardization" of input features

$$x'^{[i]}_j = \frac{x_j^{[i]} - \mu_j}{\sigma_j}$$

(scaled feature will have zero mean,
unit variance)

In deep models…

Normalizing the **inputs** only affects the first hidden layer…what about the rest?

Batch Normalization (“BatchNorm”)

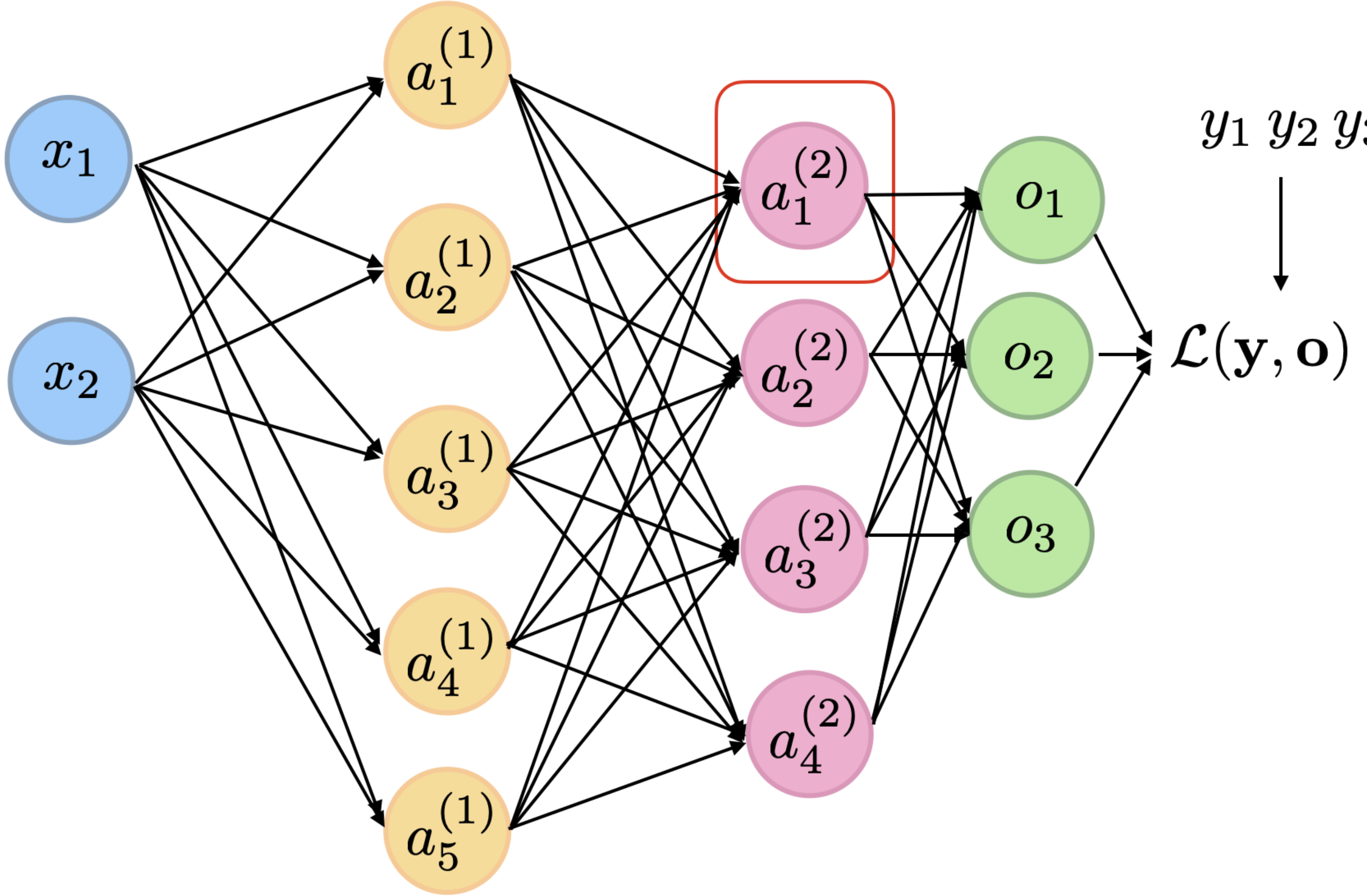
Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning* (pp. 448-456).

http://proceedings.mlr.press/v37/ioffe15.html

- Normalizes hidden layer inputs
- Helps with exploding/vanishing gradient problems
- Can increase training stability and convergence rate
- Can be understood as additional (normalization) layers (with additional parameters)

Batch Normalization (“BatchNorm”)

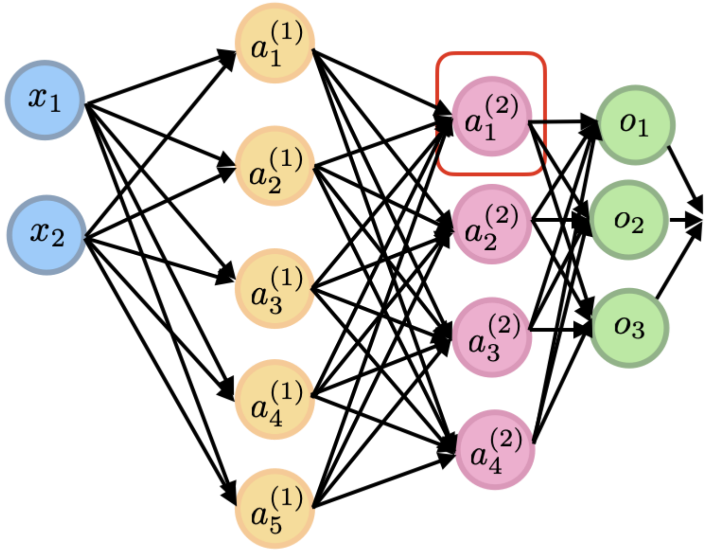
Suppose, we have net input $z_1^{(2)}$
associated with an activation in the 2nd hidden layer



Batch Normalization ("BatchNorm")

Now, consider all examples in a minibatch such that the net input of a given training example at layer 2 is written as $z_1^{(2)[i]}$

where $i \in \{1, ..., n\}$



In the next slides, let's omit the layer index, as it may be distracting...

BatchNorm Step 1: Normalize Net Inputs

$$\mu_j = \frac{1}{n} \sum_i z_j^{[i]}$$

$$\sigma_j^2 = \frac{1}{n} \sum_i (z_j^{[i]} - \mu_j)^2$$

$$z'^{[i]}_j = \frac{z_j^{[i]} - \mu_j}{\sigma_j}$$

<u>In practice:</u>

$$z'^{[i]}_j = \frac{z_j^{[i]} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

For numerical stability, where epsilon is a small number like 1E-5

BatchNorm Step 2: Pre-Activation Scaling

$$z'^{[i]}_j = \frac{z^{[i]}_j - \mu_j}{\sigma_j}$$

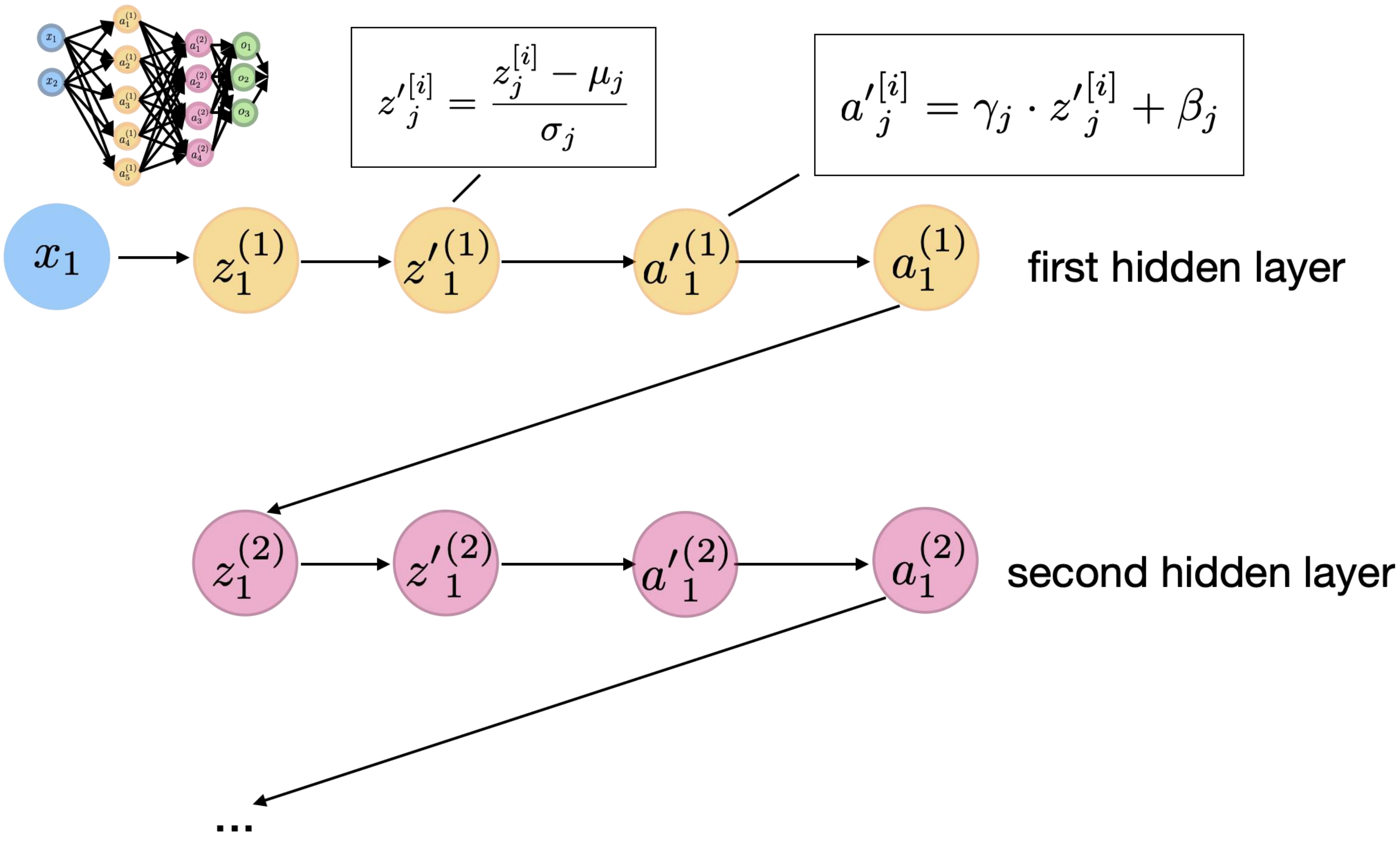
$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$

Controls the mean

Controls the spread or scale

Technically, a BatchNorm layer could learn to perform
"standardization" with zero mean and unit variance

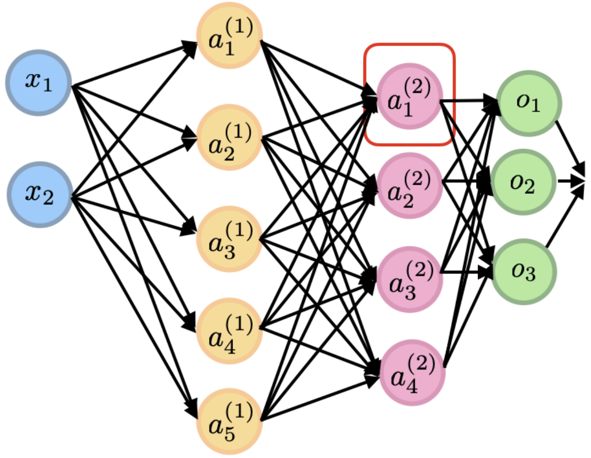
BatchNorm Steps 1+2 Together



BatchNorm Steps 1+2 Together

$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$

This parameter makes the bias units redundant



Also, note that the batchnorm parameters are vectors with the same number of elements as the bias vector

BatchNorm at Test-Time

- Use exponentially weighted average (moving average) of mean and variance

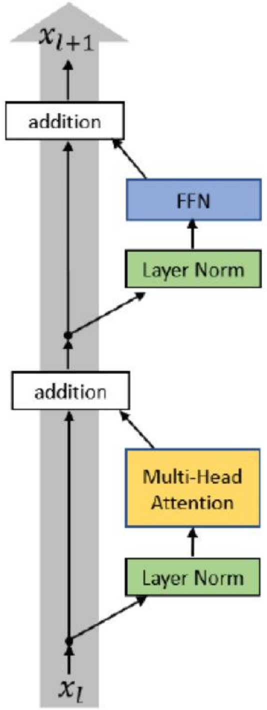
running_mean = momentum * running_mean + (1 - momentum) * sample_mean

(where momentum is typically ~0.1; and same for variance)

- Alternatively, can also use global training set mean and variance

Related: LayerNorm

- Layer normalization (LN)
- BN calculates mean/std based on a mini batch, whereas LN calculates mean/std based on feature/embedding vectors
- In the stats language, BN zero mean unit variance, whereas LN projects feature vector to **unit sphere**
- LN in Transformers



Pre-LN Transformer

$$
\begin{aligned}
x_{l,i}^{pre,1} &= \mathrm{LayerNorm}(x_{l,i}^{pre}) \\
x_{l,i}^{pre,2} &= \mathrm{MultiHeadAtt}(x_{l,i}^{pre,1}, [x_{l,1}^{pre,1}, \cdots, x_{l,n}^{pre,1}]) \\
x_{l,i}^{pre,3} &= x_{l,i}^{pre} + x_{l,i}^{pre,2} \\
x_{l,i}^{pre,4} &= \mathrm{LayerNorm}(x_{l,i}^{pre,3}) \\
x_{l,i}^{pre,5} &= \mathrm{ReLU}(x_{l,i}^{pre,4} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l} \\
x_{l+1,i}^{pre} &= x_{l,i}^{pre,5} + x_{l,i}^{pre,3}
\end{aligned}
$$

Final LayerNorm: $x_{Final,i}^{pre} \leftarrow \mathrm{LayerNorm}(x_{L+1,i}^{pre})$

Normalize everything?

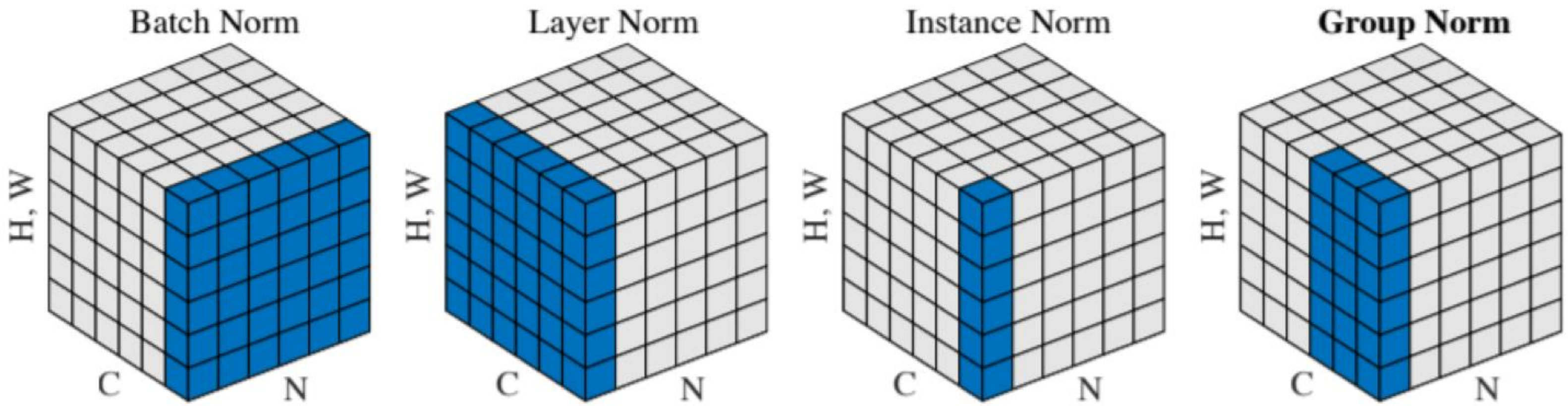


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Wu, Y., & He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 3-19).

Initialization

Weight initialization

- Recall: Can’t initialize all weights to 0 (symmetry problem)
- But we want weights to be relatively small.
  - Traditionally, we can initialize weights by sampling from a random uniform distribution in range [0, 1], or better, [-0.5, 0.5]
  - Or, we could sample from a Gaussian distribution with mean 0 and small variance (e.g., 0.1 or 0.01)

Xavier Initialization

Method:

- **Step 1:** Initialize weights from Gaussian or uniform distribution
- **Step 2:** Scale the weights proportional to the number of inputs to the layer
  - For the first hidden layer, that is the number of features in the dataset; for the second hidden layer, that is the number of units in the 1st hidden layer, etc.

Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.

Xavier Initialization

<u>Rationale behind this scaling:</u>
Variance of the sample (between data points, not variance of the mean) linearly increases as the sample size increases (variance of the sum of independent variables is the sum of the variances); square root for standard deviation

$$\mathrm{Var}\left(z_j^{(l)}\right) = \mathrm{Var}\left(\sum_{j=1}^{m_{l-1}} W_{jk}^{(l)} a_k^{(l-1)}\right)$$

$$= \sum_{j=1}^{m^{(l-1)}} \mathrm{Var}\left[W_{jk}^{(l)} a_k^{(l-1)}\right] = \sum_{i=1}^{m^{(l-1)}} \mathrm{Var}\left[W_{jk}^{(l)}\right] \mathrm{Var}\left[a_k^{(l-1)}\right]$$

$$= \sum_{j=1}^{m^{(l-1)}} \mathrm{Var}\left[W^{(l)}\right] \mathrm{Var}\left[a^{(l-1)}\right] = m^{(l-1)} \mathrm{Var}\left[W^{(l)}\right] \mathrm{Var}\left[a^{(l-1)}\right]$$

He Initialization

- Assuming activations with mean 0, which is reasonable, Xavier Initialization assumes a derivative of 1 for the activation function (which is reasonable for tanH)
- For ReLU, the **activations are not centered at zero**
- He initialization takes this into account
- The result is that we add a scaling factor of $\sqrt{2}$

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} \cdot \sqrt{\frac{2}{m^{(l-1)}}}$$

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In *Proceedings of the IEEE international conference on computer vision*, pp. 1026-1034. 2015.

Convolutional Neural Networks

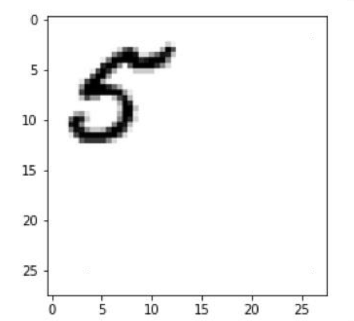
Images are hard

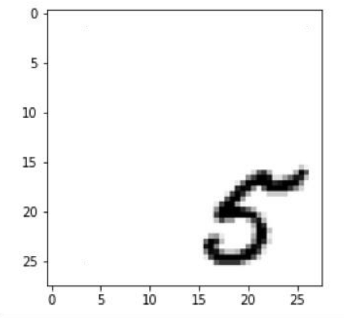
Different lighting, contrast, viewpoints, etc.

Image Source: twitter.com%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=1553887775741551

Image Source: https://www.123rf.com/photo_76714328_side-view-of-tabby-cat-face-over-white.html

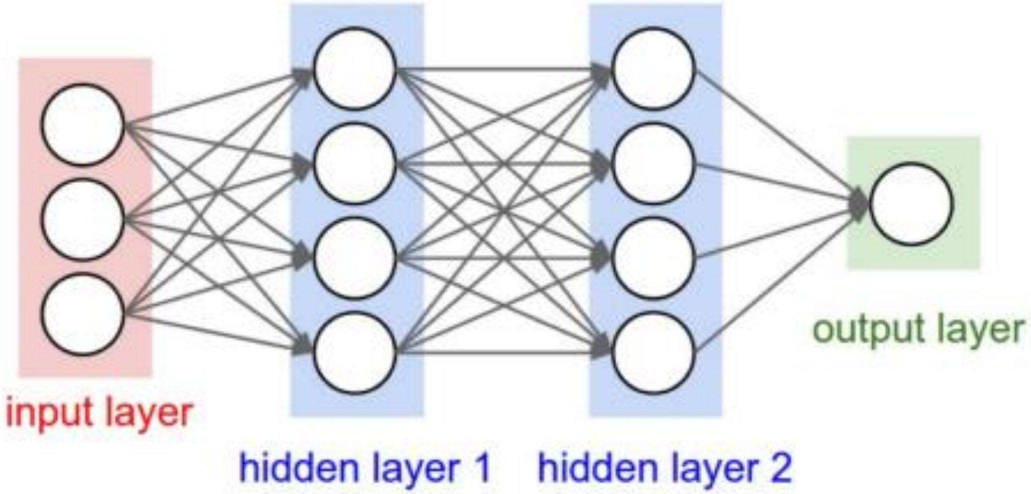
Or even simple translation





Do deep fully-connected nets solve this?

Images are hard

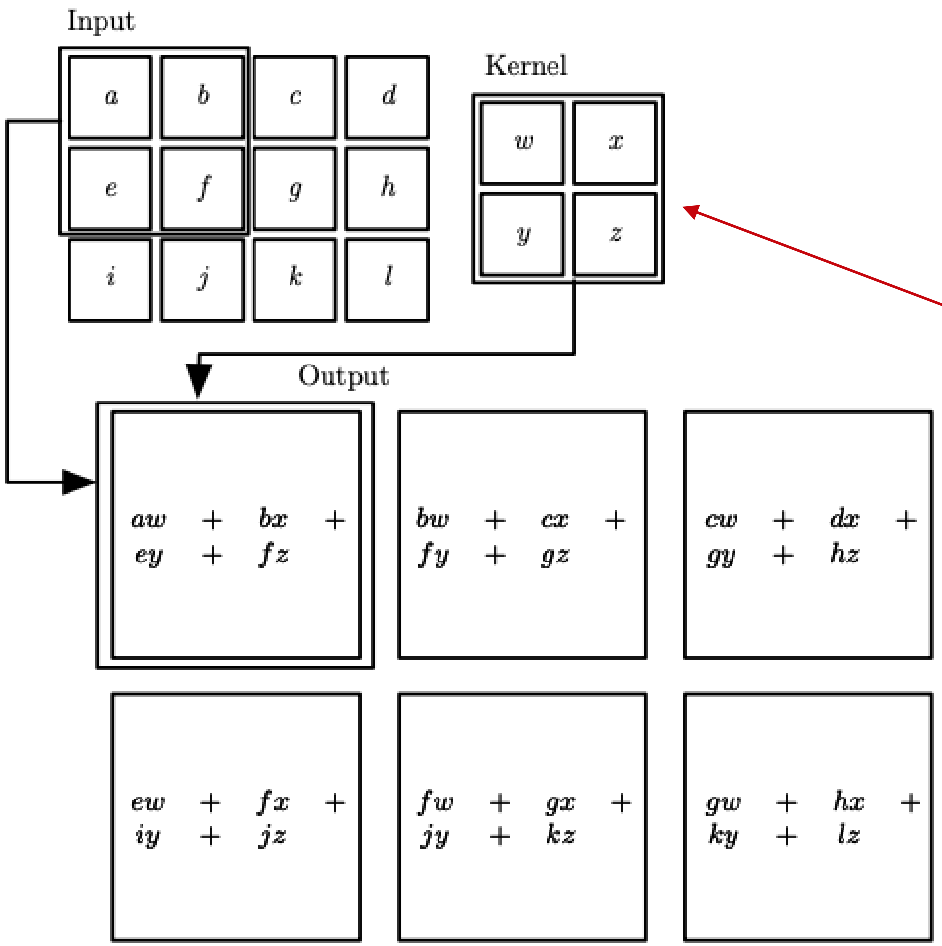


- 3x200x200 images imply **120,000** weights per neuron in first hidden layer

Convolutional Neural Networks [LeCun 1989]

- Let’s share parameters.
- Instead of learning position-specific weights, learn weights defined for **relative positions**
  - Learn “filters” that are reused across the image
  - Generalize across spatial translation of input
- Key idea:
  - Replace matrix multiplication in neural networks with a <u>convolution</u>
- Later, we will see that this can work for any graph-structured data, not just images.

Weight sharing in kernels

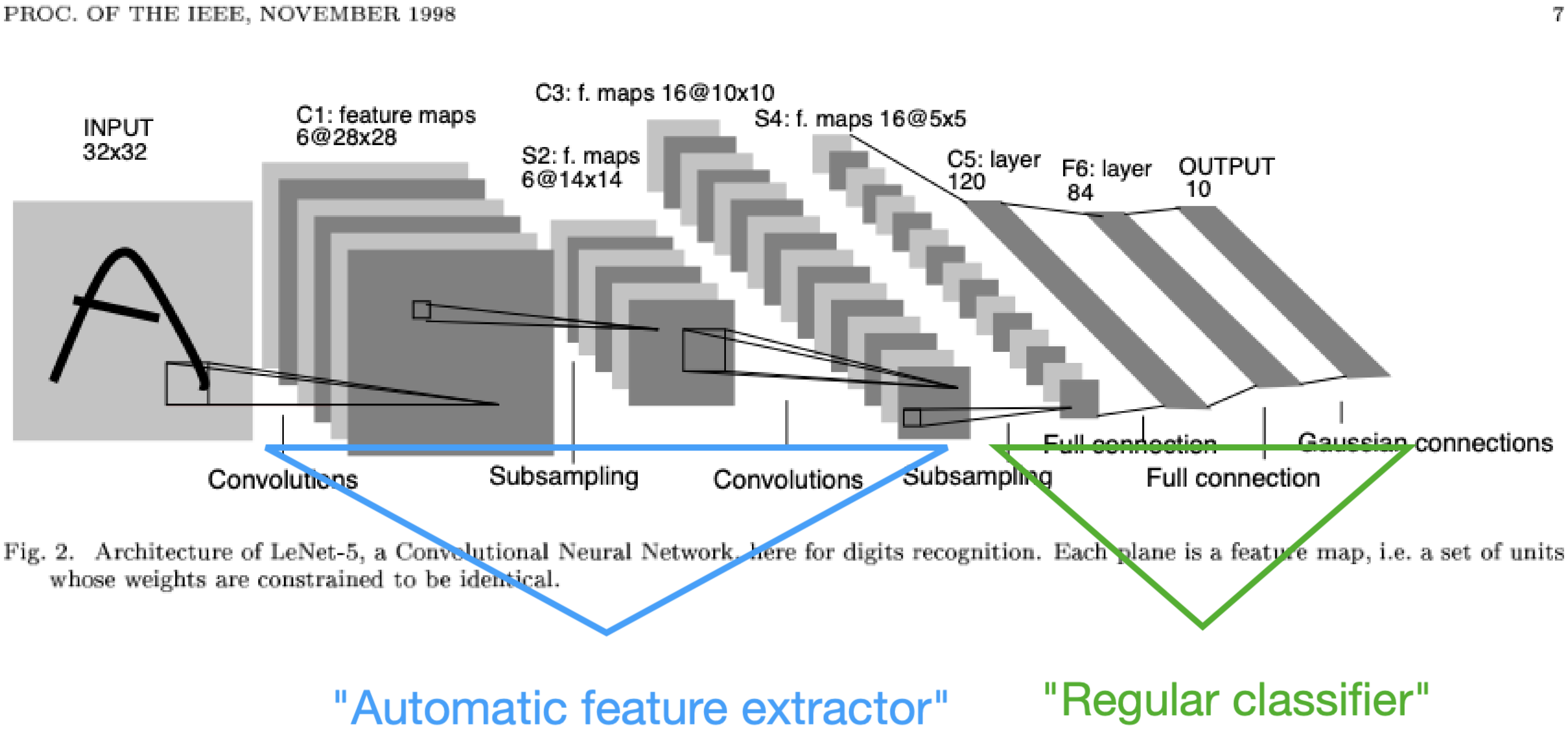


Sliding filters (kernels)

Reused weights (small)!

Fig. Goodfellow et al. 2016

Convolutional Neural Networks [LeCun 1989]



Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner: Gradient Based Learning Applied to Document Recognition, Proceedings of IEEE, 86(11):2278–2324, 1998.

Convolutional Neural Networks [LeCun 1989]

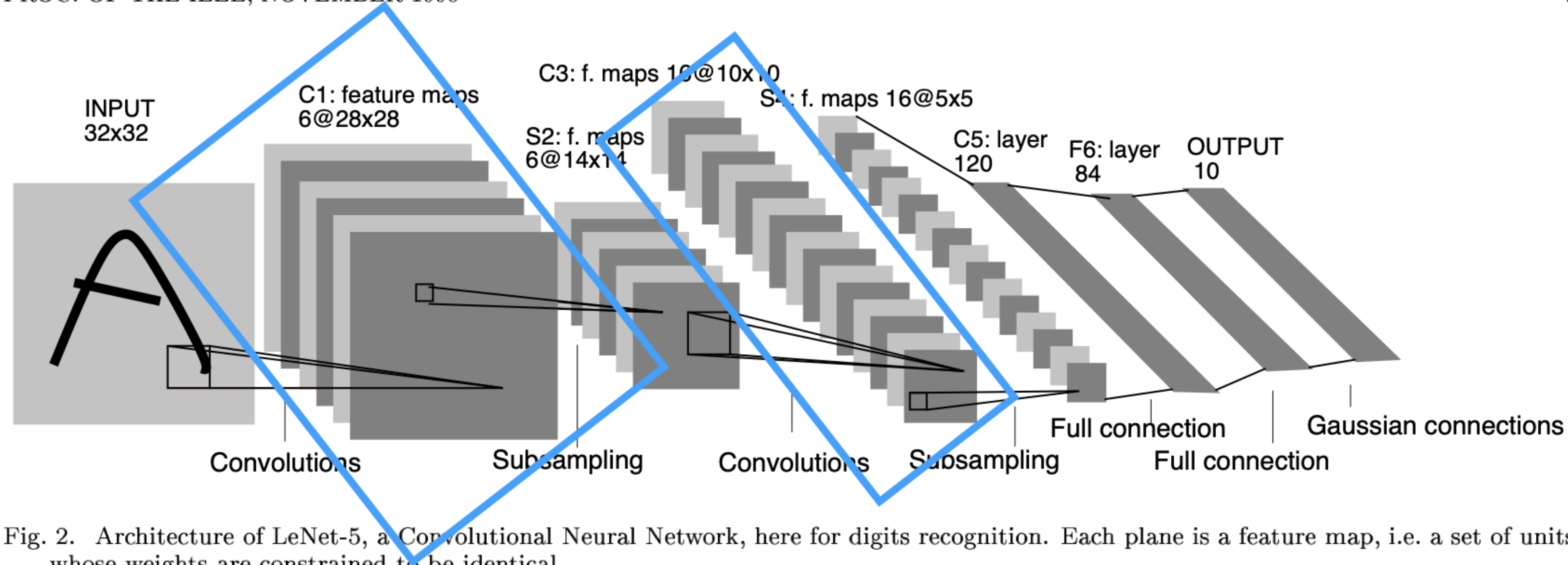


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Each "bunch" of feature maps represents one hidden layer in the neural network.

Counting the FC layers, this network has **5** layers

Convolutional Neural Networks [LeCun 1989]

PROC. OF THE IEEE, NOVEMBER 1998

Number of feature detectors

Size of the resulting layers

Multi-layer perceptron

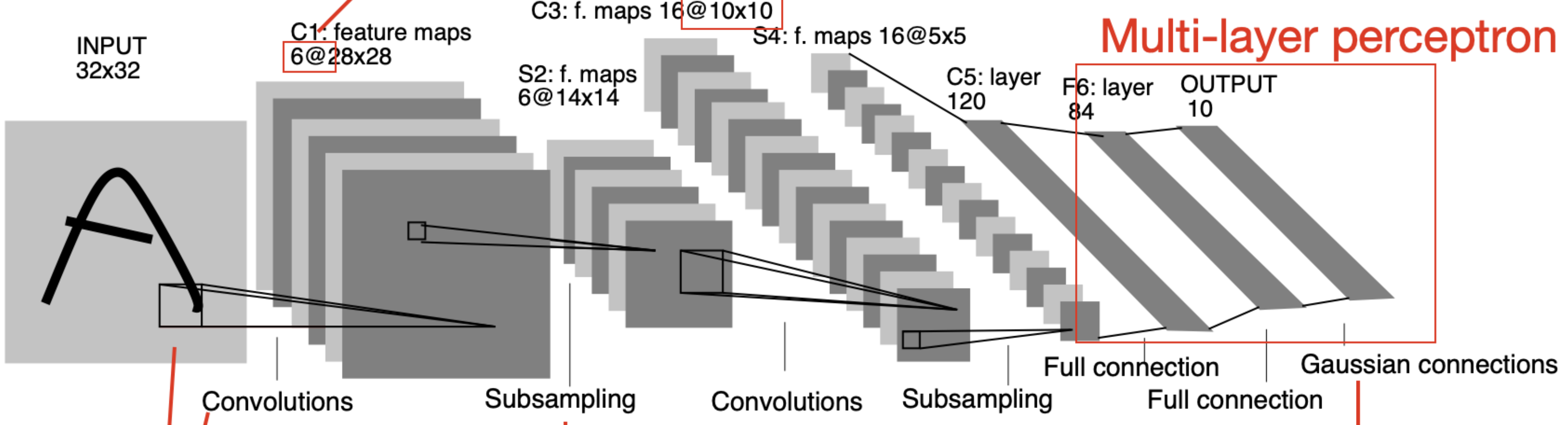


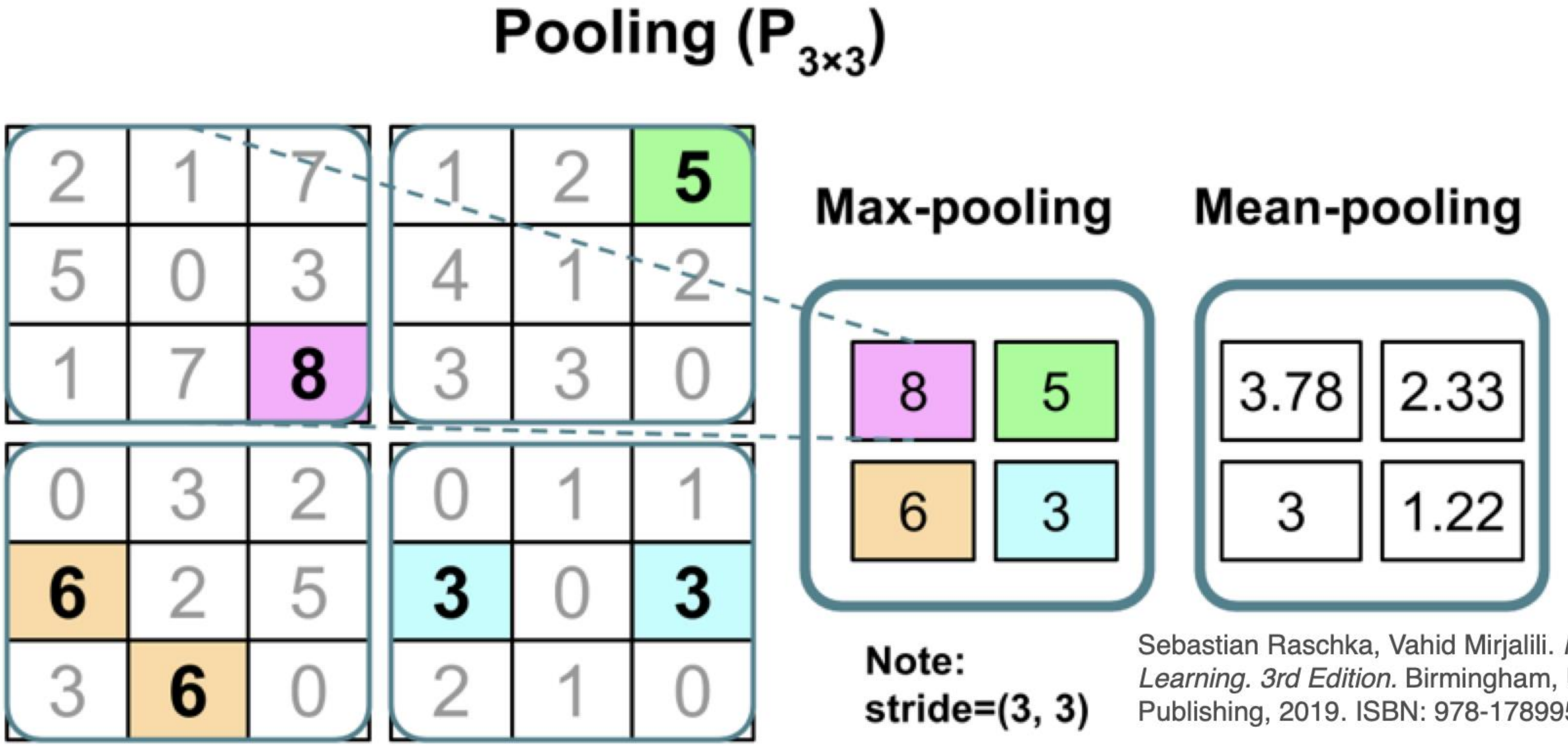
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

nowadays called "pooling"

"Feature detectors" (weight matrices)
that are being reused ("weight sharing")
=> also called "kernel" or "filter"

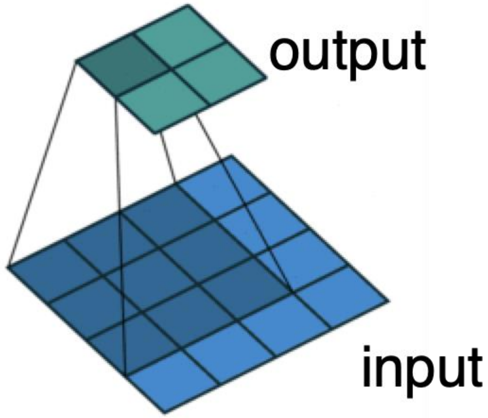
basically a fully-connected
layer + MSE loss
(nowadays common to use
fc-layer + softmax
+ cross entropy)

“Pooling”: lossy compression

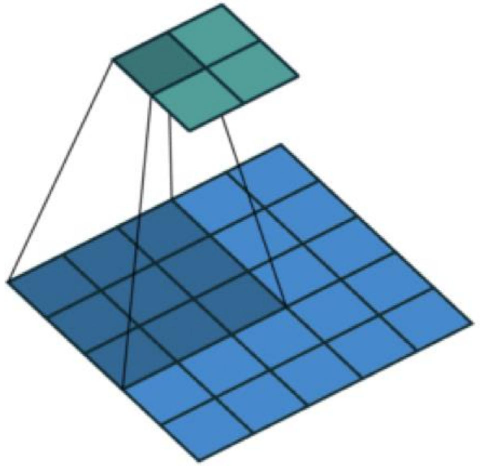


Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019. ISBN: 978-1789955750

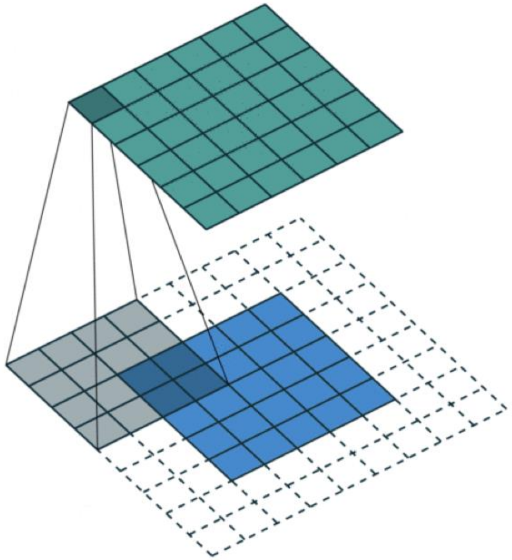
Padding



No padding, stride=1



No padding, stride=2



padding=2, stride=1

Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

Main ideas of CNNs

- **Sparse-connectivity:** A single element in the feature map is connected to only a small patch of pixels. (This is very different from connecting to the whole input image, in the case of multi-layer perceptrons.)
- **Parameter-sharing:** The same weights are used for different patches of the input image.
- **Many layers:** Combining extracted local patterns to global patterns

Impact of convolutions on size

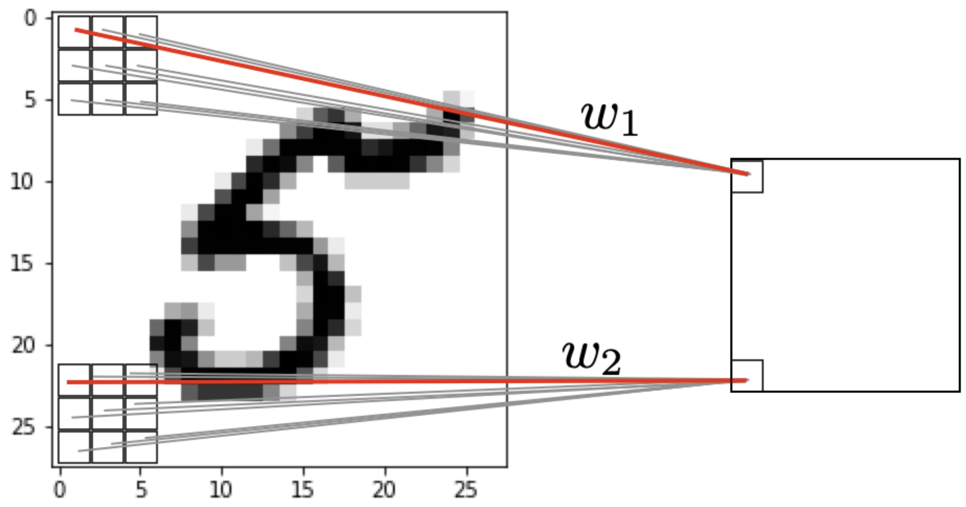
Feature map size:

$$\underbrace{O}_{\text{output width}} = \frac{\overbrace{W}^{\text{input width}} - \overbrace{K}^{\text{kernel width}} + 2\overbrace{P}^{\text{padding}}}{\underbrace{S}_{\text{stride}}} + 1$$

Backpropagation in CNNs

- Same concept as before: Multivariable chain rule, and now with an additional weight-sharing constraint

Due to weight sharing: $w_1 = w_2$



Optional averaging

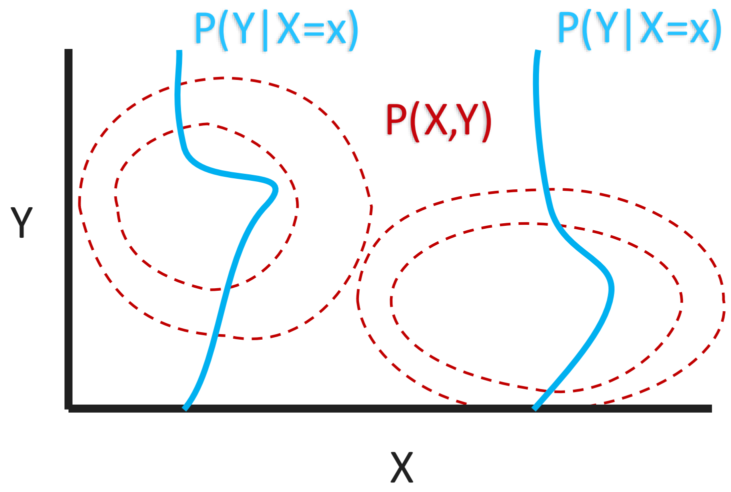
weight update:

$$w_1 := w_2 := w_1 - \eta \cdot \frac{1}{2}\left(\frac{\partial \mathcal{L}}{\partial w_1} + \frac{\partial \mathcal{L}}{\partial w_2}\right)$$

Generative Models

Generative and Discriminative Models

- **Generative**:
  - Models the joint distribution $\mathrm{P}(X, Y)$.
- **Discriminative**:
  - Models the conditional distribution $\mathrm{P}(Y|X)$.



Two paths to P(Y|X)

- **Discriminative**:

Observe X, Y → Learn P(Y|X)

- **Generative**:

  - Learn $\mathrm{P}(X|Y), \mathrm{P}(\mathrm{Y})$
  - Calculate $P(X) = \int_Y P(X,Y)dY$

Observe X, Y

$$\mathrm{P(Y|X)} = \frac{\mathrm{P(X|Y)P(Y)}}{P(X)}$$

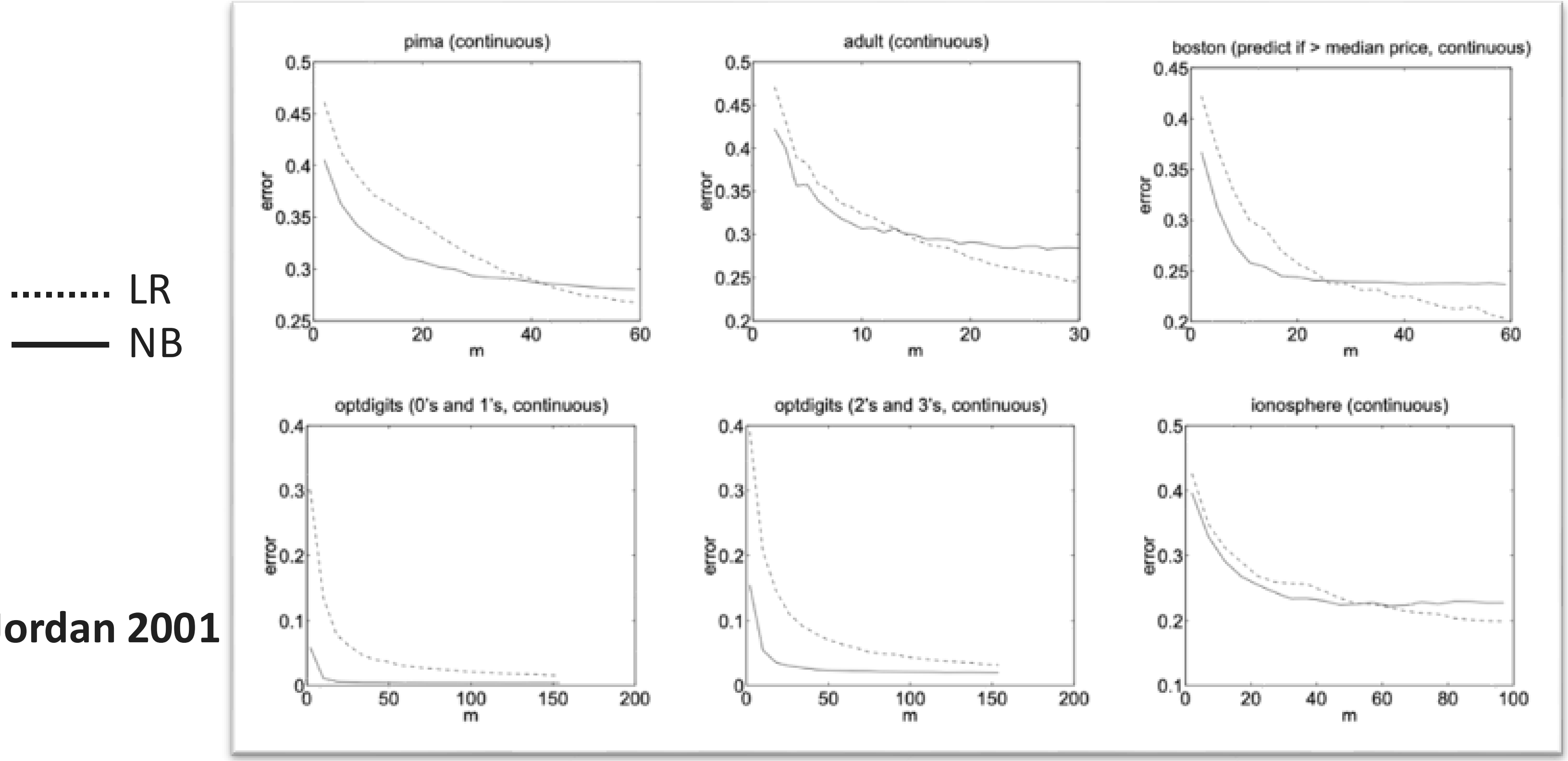
Example: Logistic Regression vs Naïve Bayes

Logistic Regression	Naïve Bayes
Discriminative	Generative
Defines $P(Y\|X;\theta)$	Defines $P(X,Y;\theta)$
Estimates $\widehat{\theta_{lr}} = \text{argmax}_\theta P(Y\|X;\theta)$	Estimates $\widehat{\theta_{nb}} = \text{argmax}_\theta P(X,Y,\theta)$
Lower asymptotic error on classification	Higher asymptotic error on classification
Slower convergence in terms of samples	Faster convergence in terms of samples

Discriminative vs Generative: A Proposition

- “While discriminative learning has lower asymptotic error, a generative classifier may also approach its (higher) asymptotic error much faster.”

Why?



Ng & Jordan 2001

Discriminative vs Generative: A Proposition

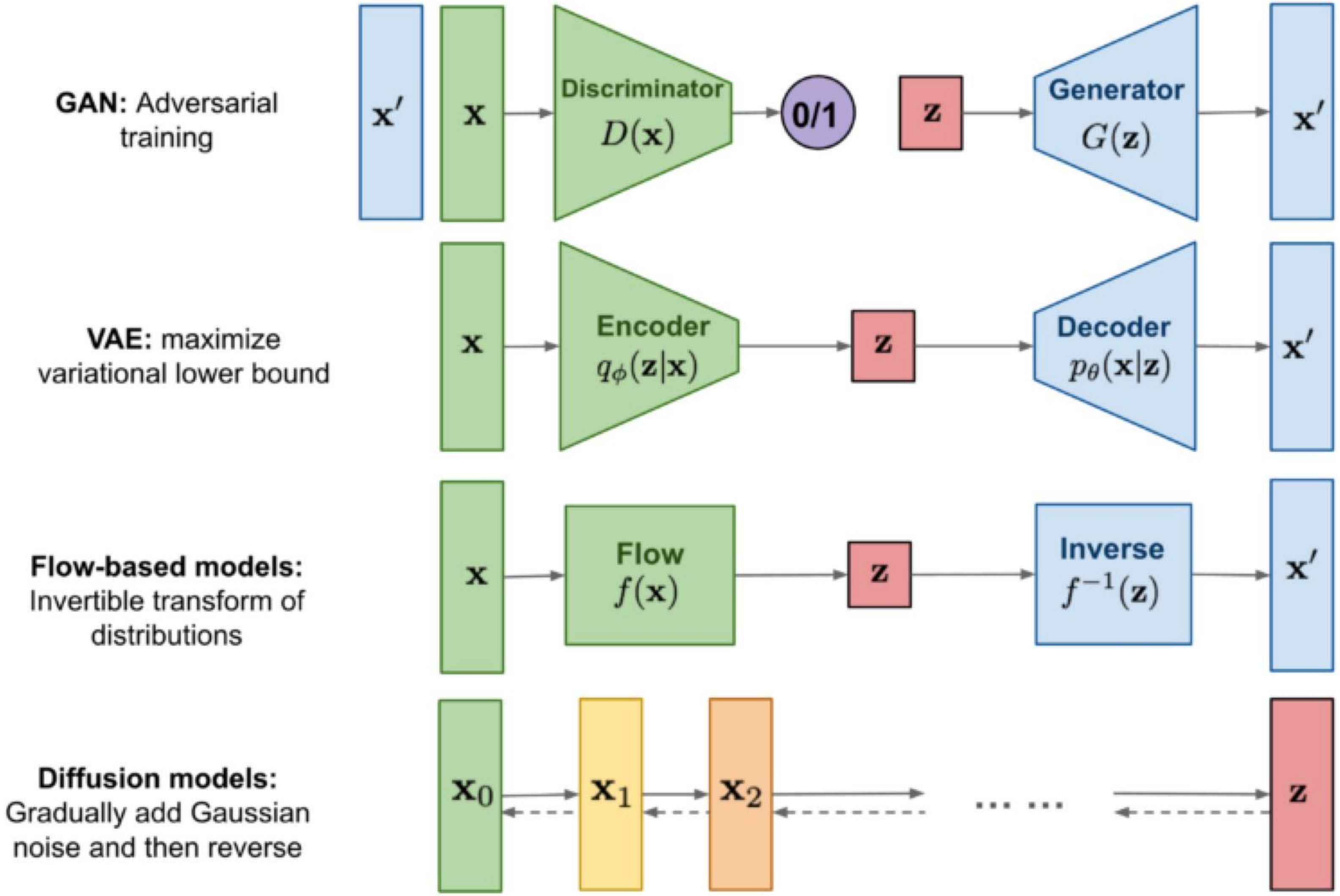
- "While discriminative learning has lower asymptotic error, a generative classifier may also approach its (higher) asymptotic error much faster."
- Underlying assumption of this statement:
  - Generative models of the form $\boldsymbol{P(X, Y, \theta)}$ make **more simplifying assumptions** than do discriminative models of the form $\boldsymbol{P(Y|X, \theta)}$.
  - **Not always true**
  - "So far there is no theoretically correct, general criterion for choosing between the discriminative and the generative approaches to classification of an observation $\mathbf{x}$ into a class y; the choice depends on the relative confidence we have in the correctness of the specification of either $p(y|\mathbf{x})$ or $p(\mathbf{x}, y)$ for the data."

Xue & Tittering 2008

Modern Deep Generative Models (DGMs)

- Goal: Generative models of the form $\boldsymbol{P(X, Y, \theta)}$ without strong simplifying assumptions.
- Hidden structure z that explains high-dim. x
- Fundamental challenge: We never observe z
- This makes two core computations difficult:
  - **Marginal likelihood:** $p_\theta(x) = \int p_\theta(x, z) dz$
  - **Posterior inference:** $p_\theta(z \mid x) \propto p_\theta(x \mid z) p(z)$
- Each type of DGM makes a tradeoff

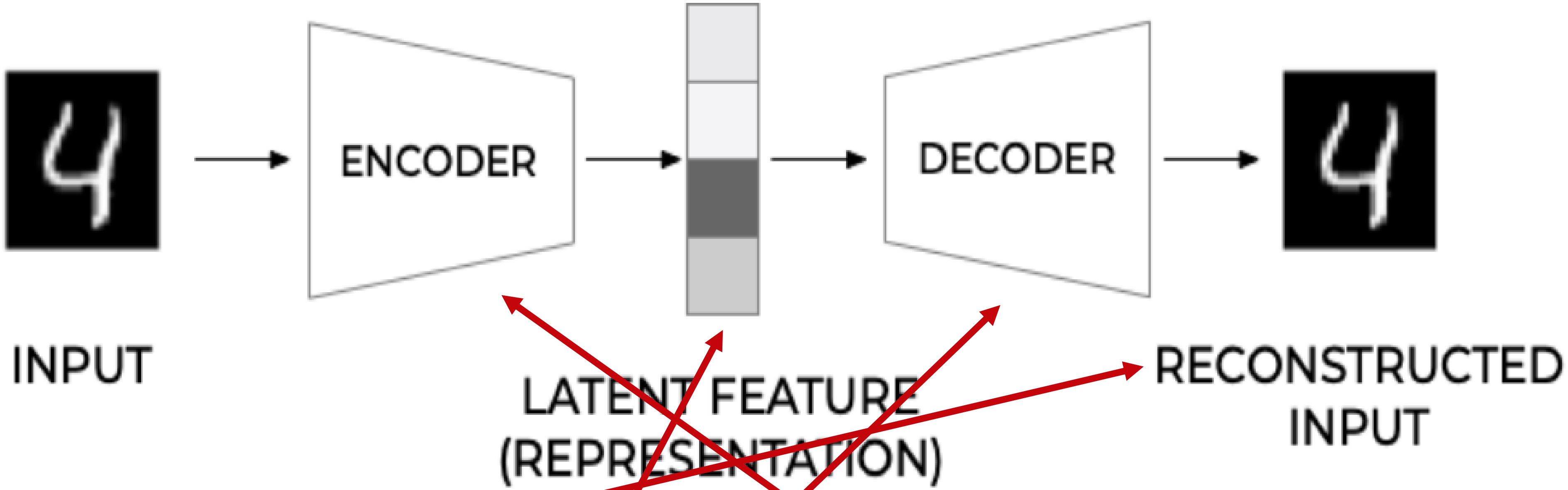
Overview and comparison of generative models



Source: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Autoencoders

Autoencoders

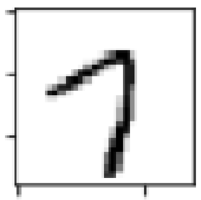


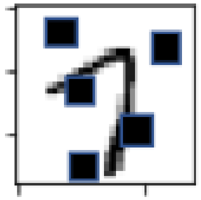
$$\tilde{x} = f(h) = f\big(g(x)\big)$$

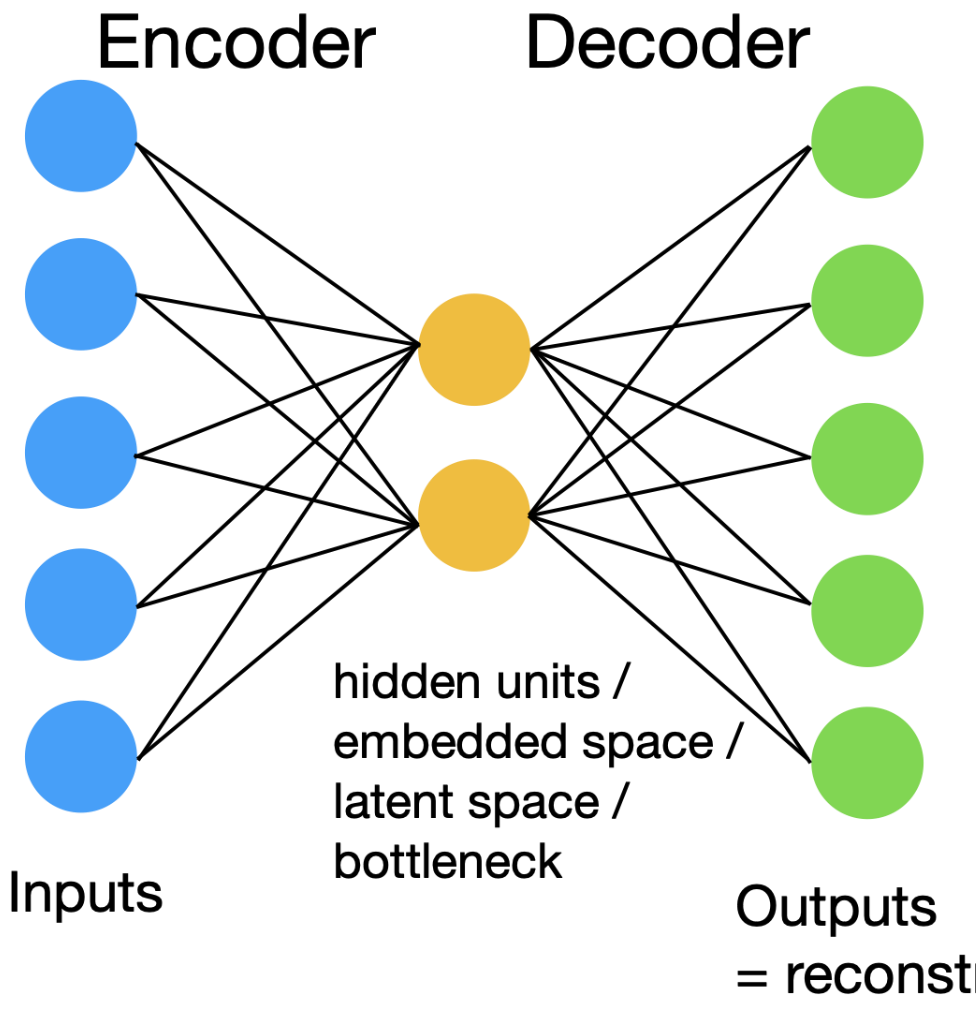
[Michelucci 2022]

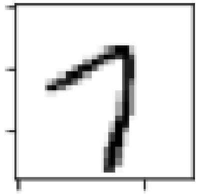
Denoising Autoencoders

Add dropout after the input, or add noise to the input to learn to denoise inputs



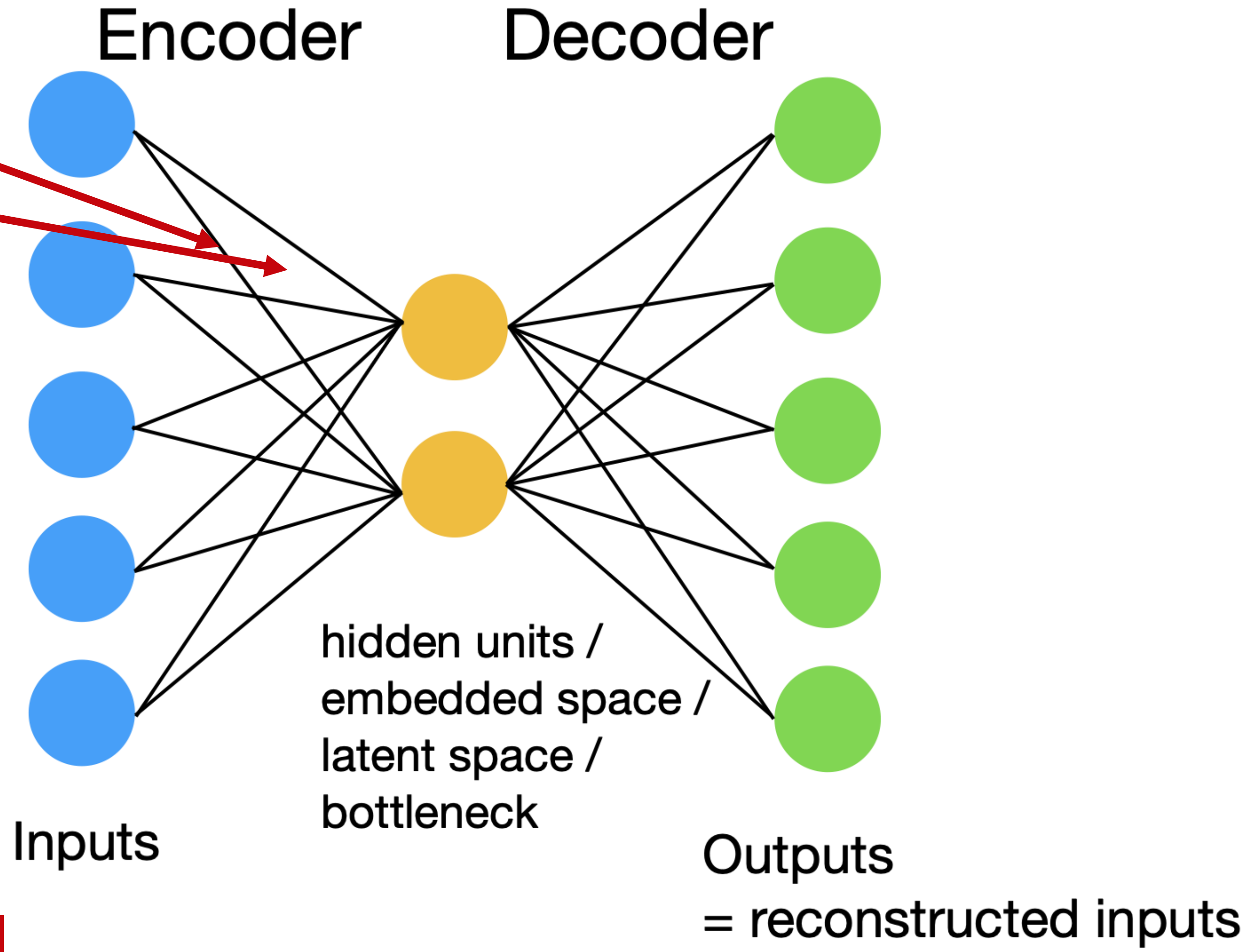






Autoencoders and Dropout

Add dropout layers to force the network to learn redundant features

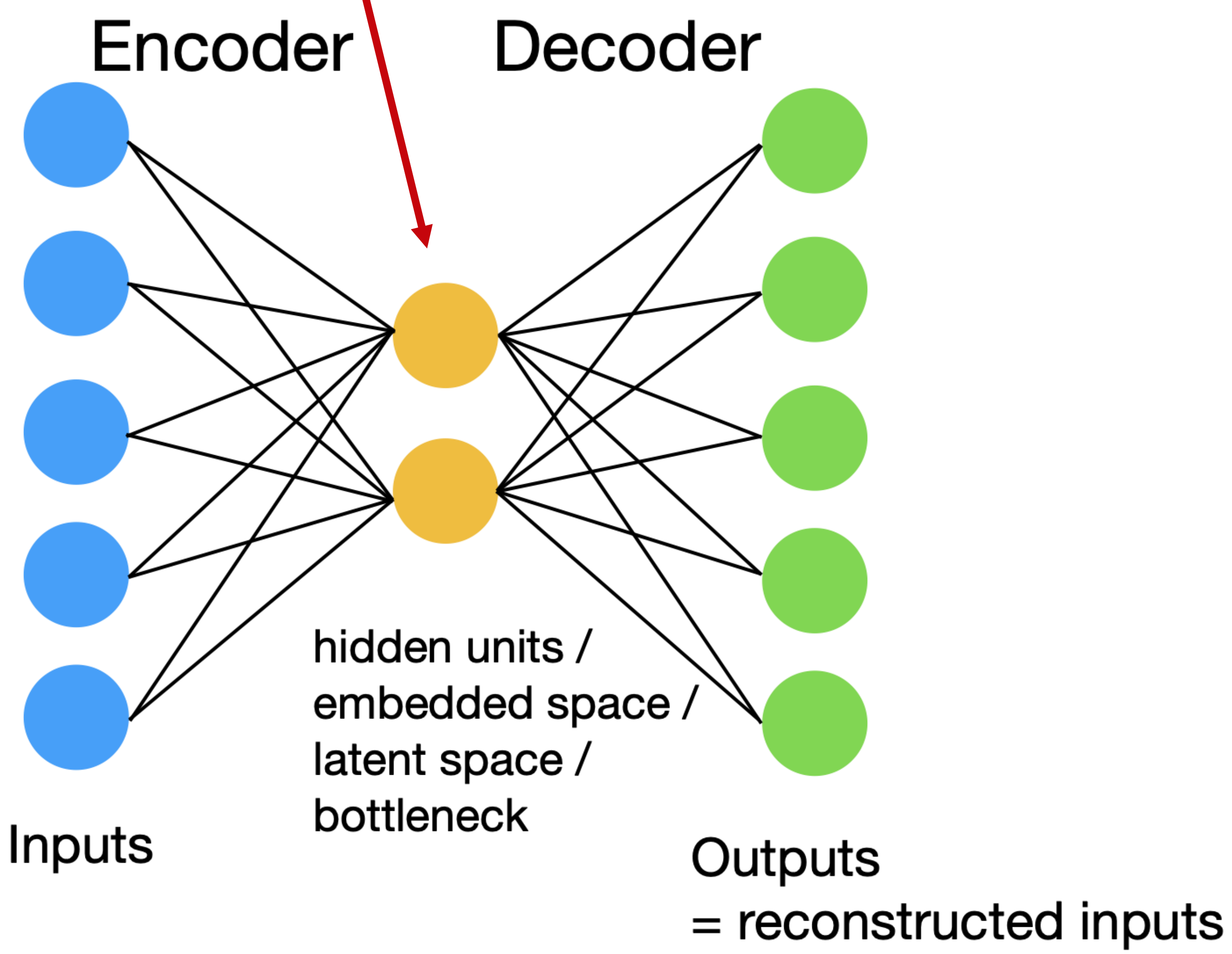


Sparse Autoencoders

Add L1 penalty to the loss to learn sparse feature representations

$$\sum_i |Enc_i(\mathbf{x})|$$

$$\mathcal{L} = ||\mathbf{x} - Dec(Enc(\mathbf{x}))||_2^2 + \sum_i |Enc_i(\mathbf{x})|$$



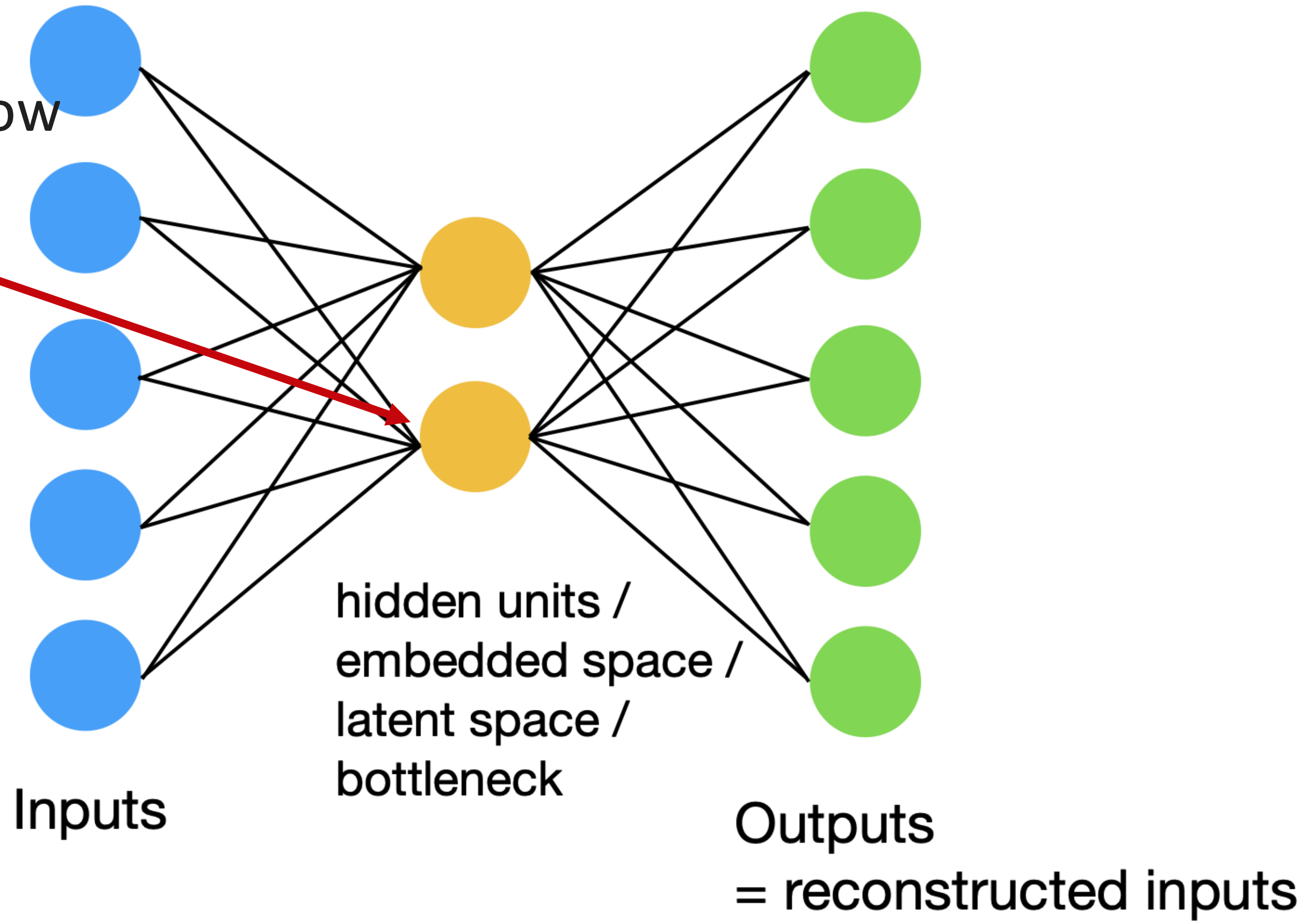
Variational Autoencoders

Kullback-Leibler divergence term where $p(z) = \mathcal{N}\left(\mu = 0, \sigma^2 = 1\right)$

$$L^{[i]} = -\mathbb{E}_{z \sim q_w(z \mid x^{[i]})}\left[\log p_w\left(x^{[i]} \mid z\right)\right] + \mathbf{KL}\left(q_w\left(z \mid x^{[i]}\right) \| p(z)\right)$$

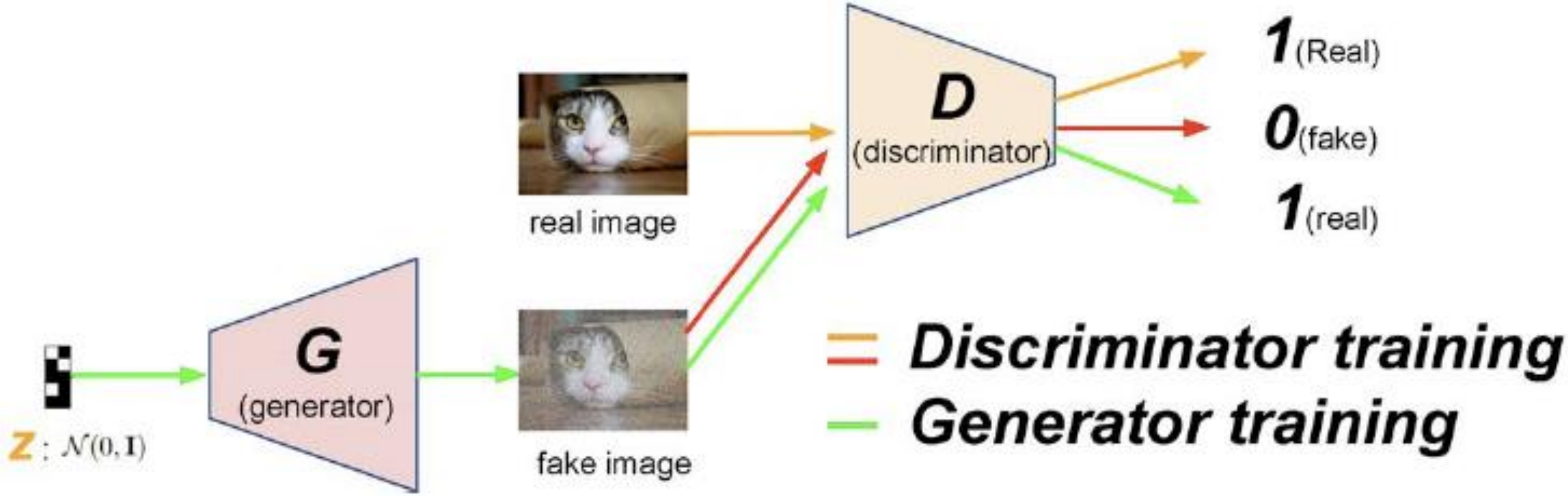
Define latent dim to follow Normal distribution

Enables sampling



Generative Adversarial Networks (GANs)

Generative Adversarial Networks



Discriminator: $$\max_D \mathcal{L}_D = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})} [\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim G(\boldsymbol{z}), \boldsymbol{z} \sim p(\boldsymbol{z})} [\log(1 - D(\boldsymbol{x}))]$$

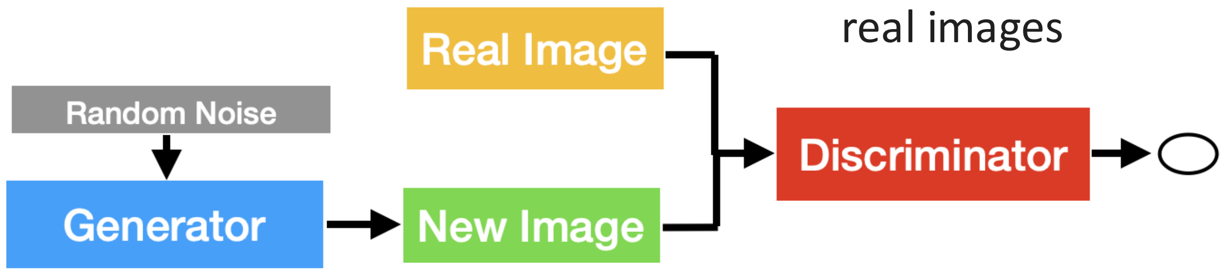
Generator: $$\min_G \mathcal{L}_G = \mathbb{E}_{\boldsymbol{x} \sim G(\boldsymbol{z}), \boldsymbol{z} \sim p(\boldsymbol{z})} [\log(1 - D(\boldsymbol{x}))] .$$

GAN Training – Putting it together

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Discriminator gradient for update (gradient ascent):

$$\nabla_{\mathbf{w}_D} \frac{1}{n} \sum_{i=1}^{n} \Big[\underbrace{\log D\left(\boldsymbol{x}^{(i)}\right)}_{\text{want large probability on real images}} + \underbrace{\log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)}_{\text{want small probability on generated images}} \Big]$$

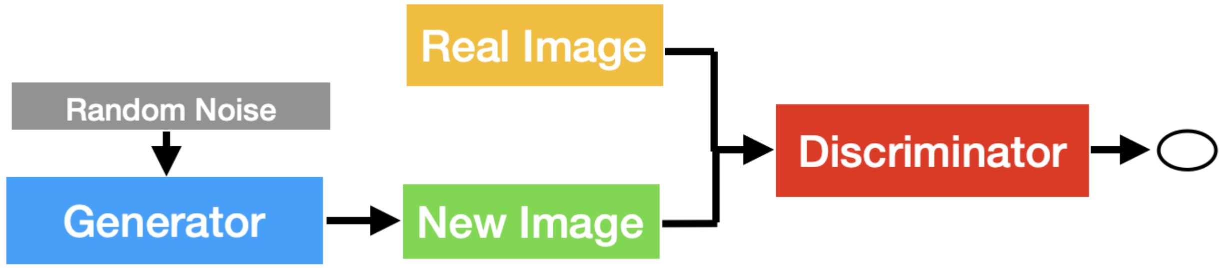


GAN Training – Putting it together

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Generator gradient for update (gradient descent):

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 - \underbrace{D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)}_{\text{Want discriminator to predict poorly on fake images}}\right)$$



GAN Training Problems

- Oscillation between generator and discriminator loss
- Mode collapse (generator produces examples of a particular kind only)
- Discriminator is too strong, such that the gradient for the generator vanishes and the generator can't keep up

Instead of gradient descent with

$$\nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

Do gradient ascent with

"Non-saturating" GAN

$$\nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$$

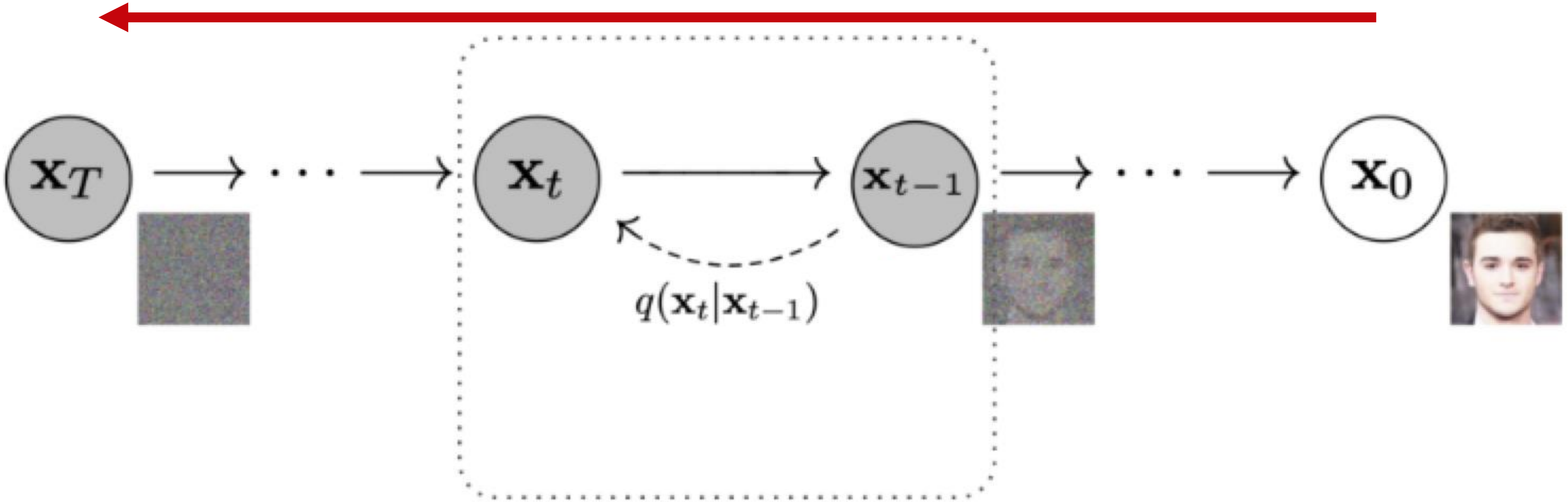
GAN Training Problems

- Oscillation between generator and discriminator loss
- Mode collapse (generator produces examples of a particular kind only)
- Discriminator is too strong, such that the gradient for the generator vanishes and the generator can't keep up
- Discriminator is too weak, and the generator produces non-realistic images that fool it too easily (rare problem, though)
- Sensitive to learning rate and other hyper parameters

Diffusion Models

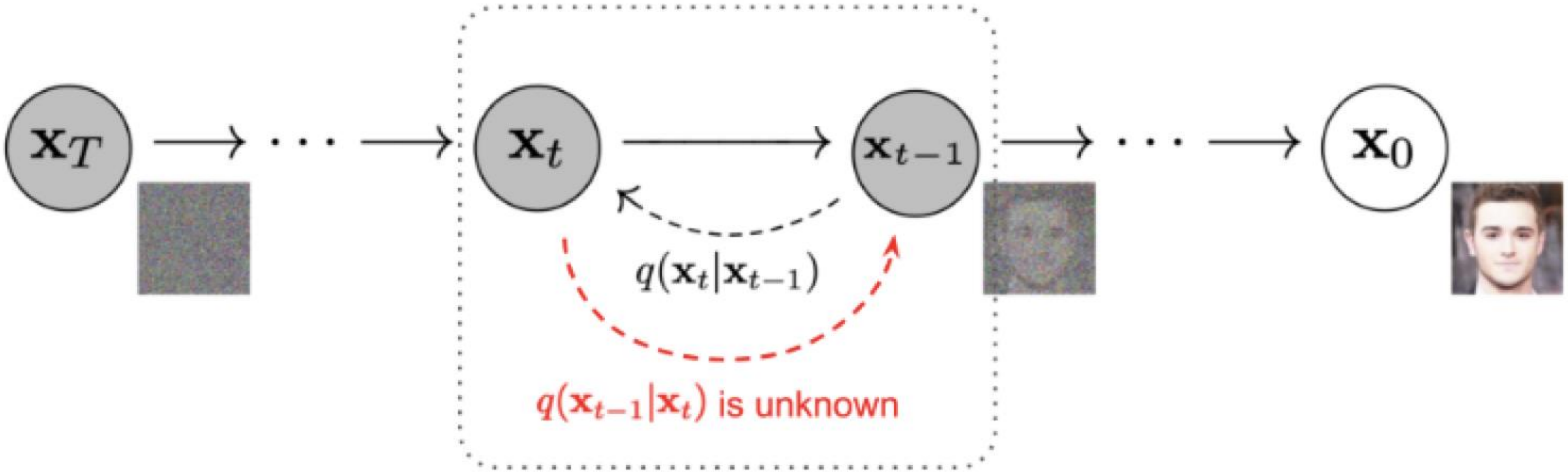
Diffusion models: forward pass

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$



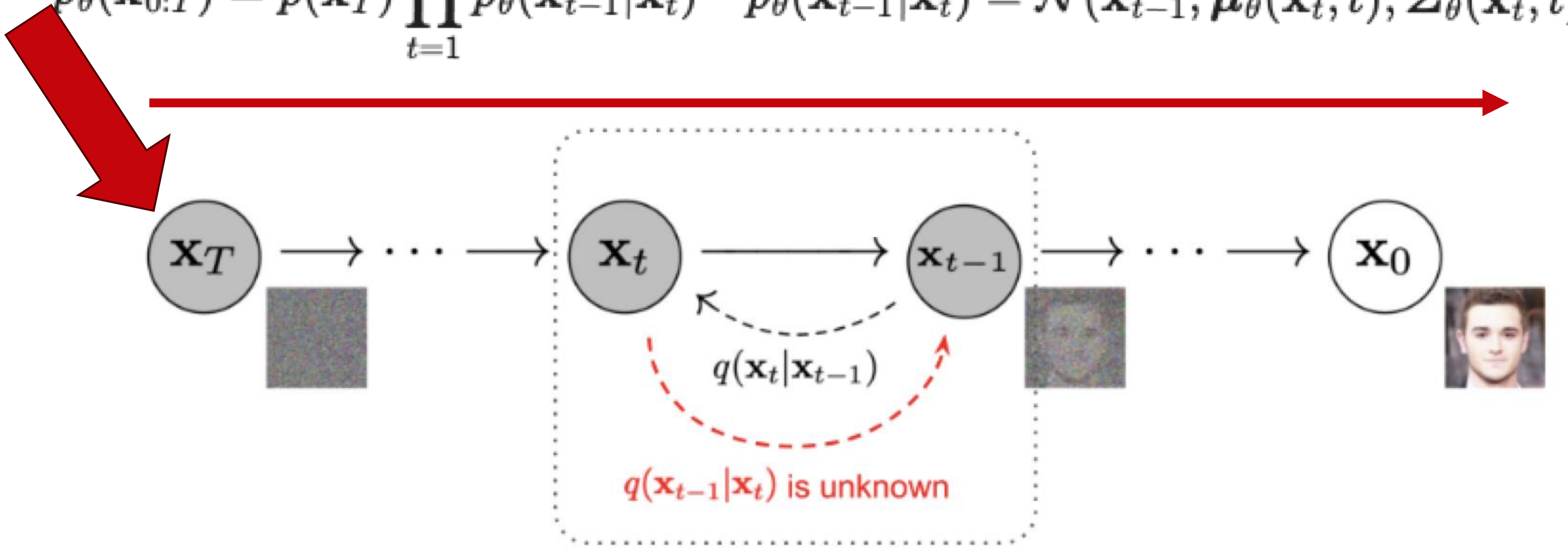
Diffusion models: reverse pass

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$



Diffusion models: generating a new sample

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

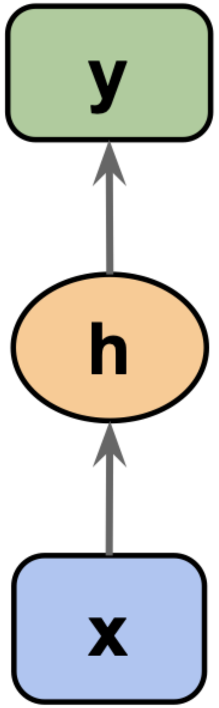


Property	VAE	GAN	Diffusion
What we specify	Prior p(z), Likelihood $p_\theta(x \mid z)$	Prior p(z), Generator $G_\theta(z)$	Fixed **forward noising** $q(x_t \mid x_{\{t-1\}})$; learn reverse $p_\theta(x_{t-1} \mid x_t)$
Induced $p(x)$	$p_\theta(\mathrm{x}) = \int_z p_\theta(X \mid z) p(z) dz$	$p_\theta(\mathrm{x}) = \int_z p_\epsilon(x - \mathrm{G}_\theta(\mathrm{z}))\mathrm{p}(\mathrm{z})\mathrm{dz}$	$p_\theta(\mathrm{x}) = \int p(x_T) \prod_t p_\theta(x_{t-1} \mid x_t) dx$
Simplifying assumption	Choose a **restricted** variational posterior $q_\phi(z \mid x)$	**Replace NLL** with a **distributional discrepancy** on samples (adversarial/IPM).	Fix forward noise q; and optimize a **variational bound** on $-\log p_\theta(x_0)$.
Training objective	ELBO: $E_q[\log p_\theta(x \mid z)] - KL(q_\phi(z \mid x) \| p(z))$	Minimax fooling discriminator	**VLB / score matching**: with Gaussian schedules reduces to $\mathbb{E}_{t,x_0,\epsilon}[w(t) \ \|\epsilon - \epsilon_\theta(x_t, t)\|^2]$
What's ignored from $p_\theta(x)$	$KL(q_\phi(z \mid x) \| p_\theta(z \mid x))$	**All of NLL**: $\log p_\theta(x)$ isn't evaluated or maximized.	Exact NLL not computed; optimize a **variational upper bound on NLL** (equivalently lower bound on $\log p$;(practical losses often **reweight** or drop constants from the exact VLB.
Modes	Covering	Collapse	Covering

Sequence Models

Recurrent Neural Networks (RNNs)

Networks we used previously: also called feedforward neural networks



Recurrent Neural Network (RNN)

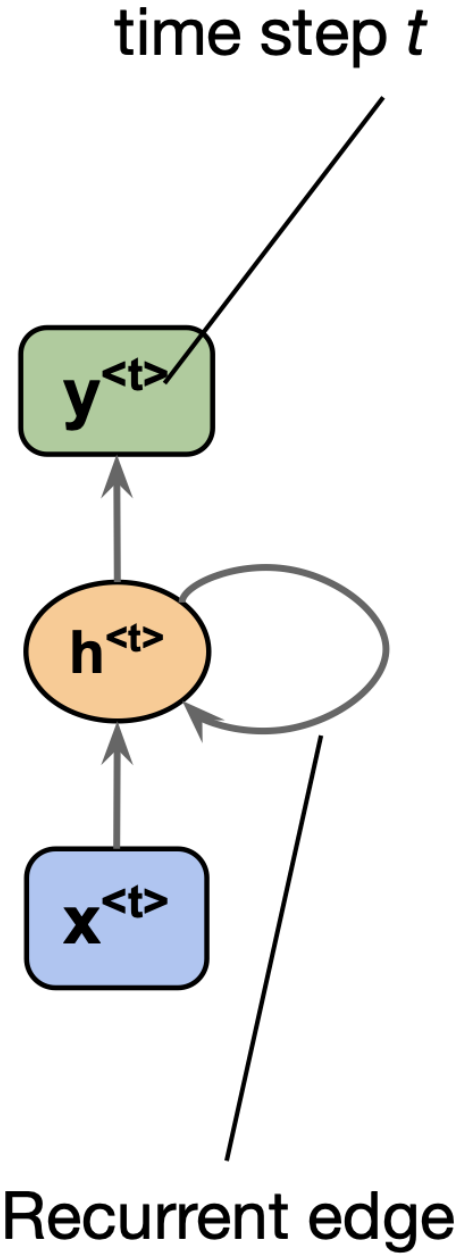


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Recurrent Neural Networks (RNNs)

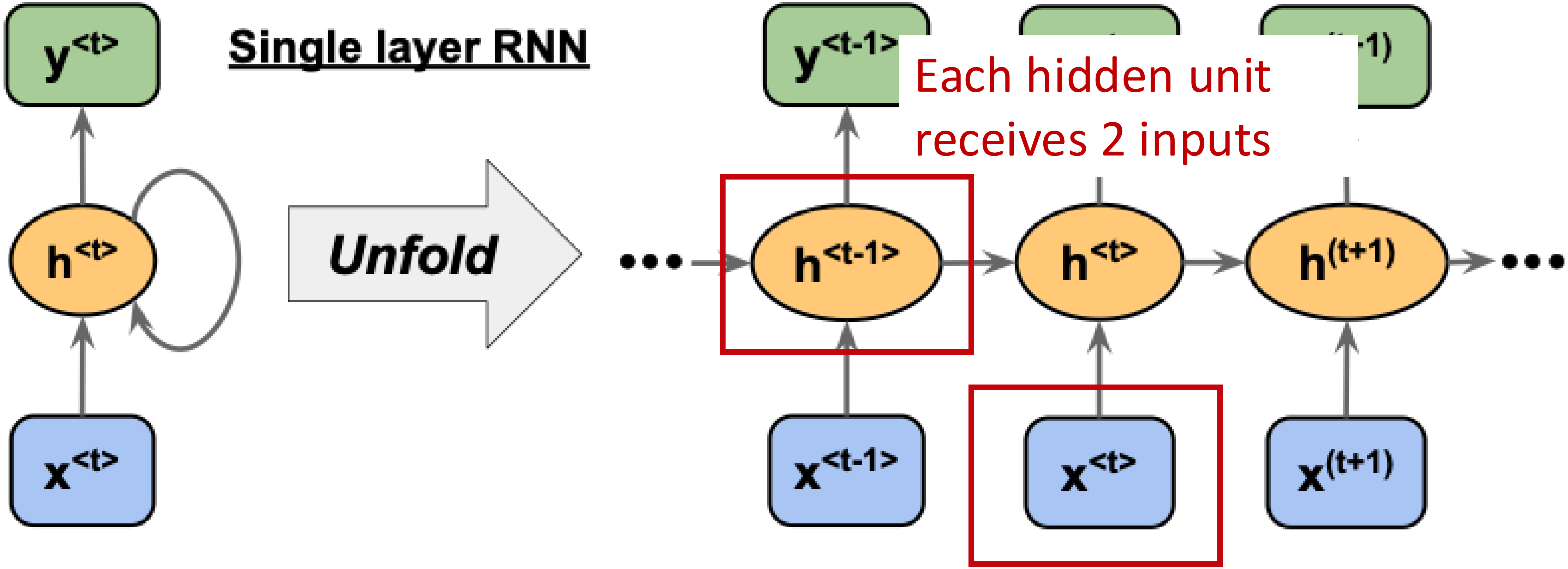


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Multilayer RNNs

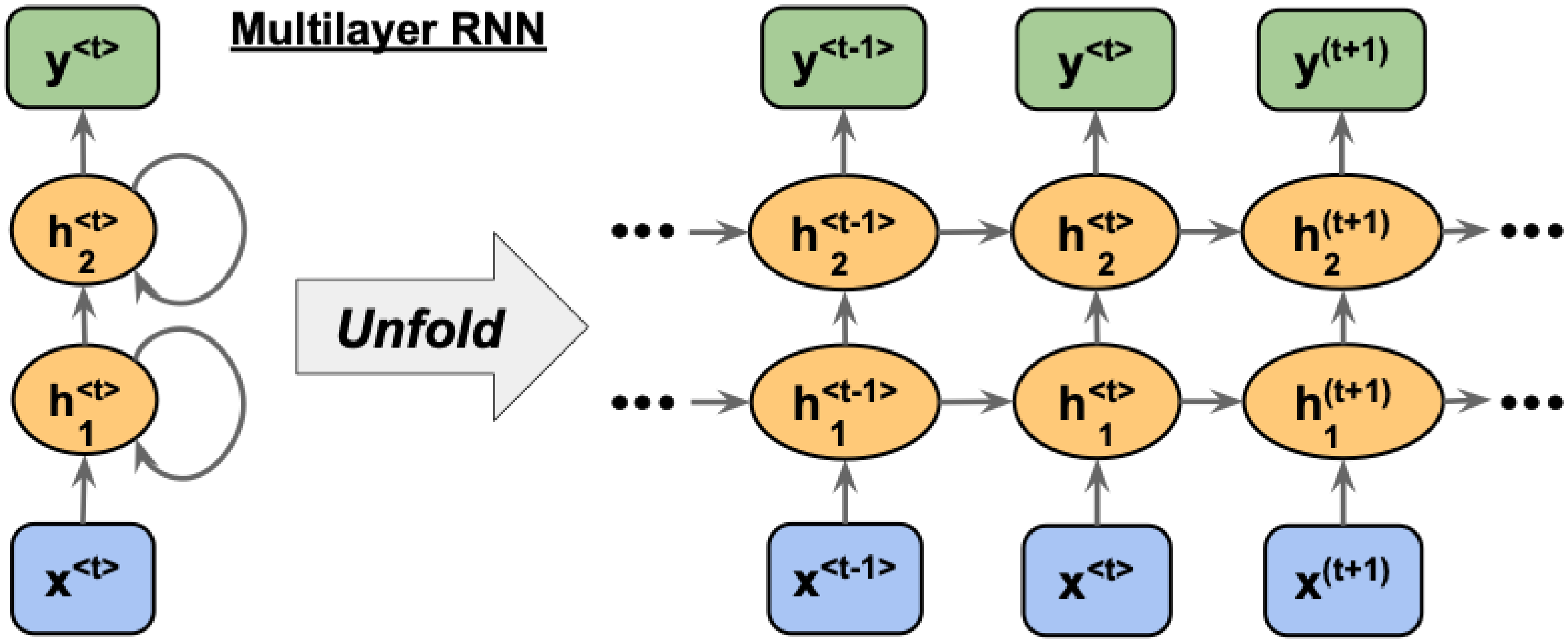
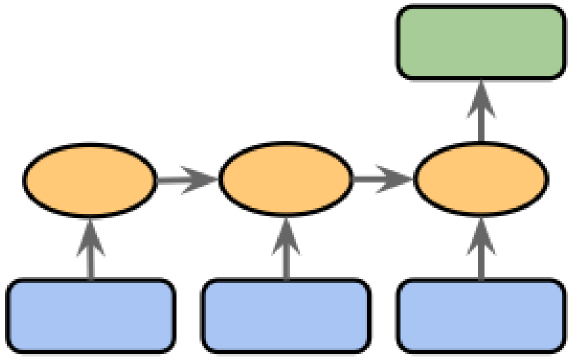
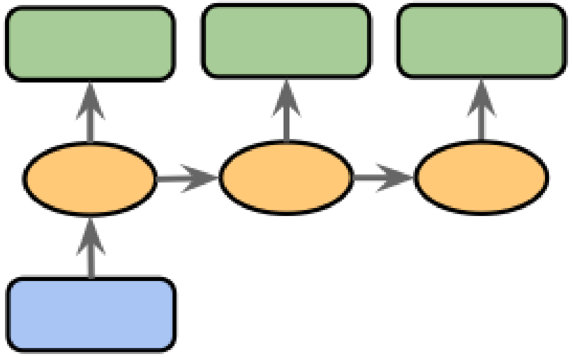


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

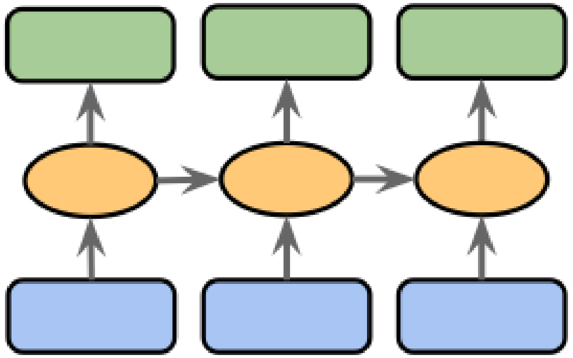
Recurrence unlocks many types of sequence tasks



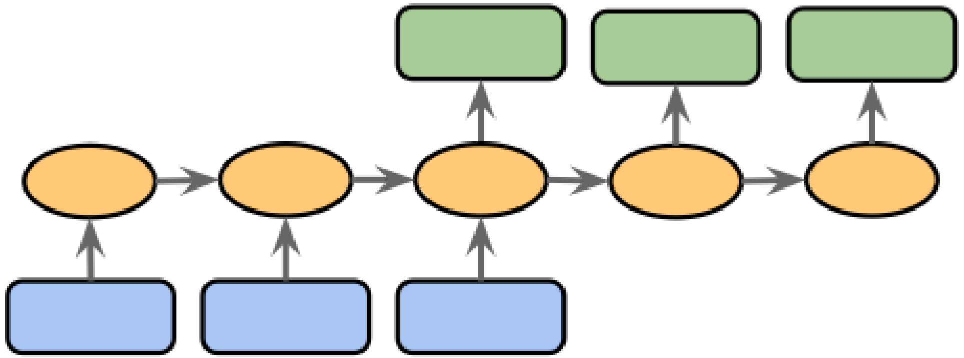
many-to-one



one-to-many



many-to-many

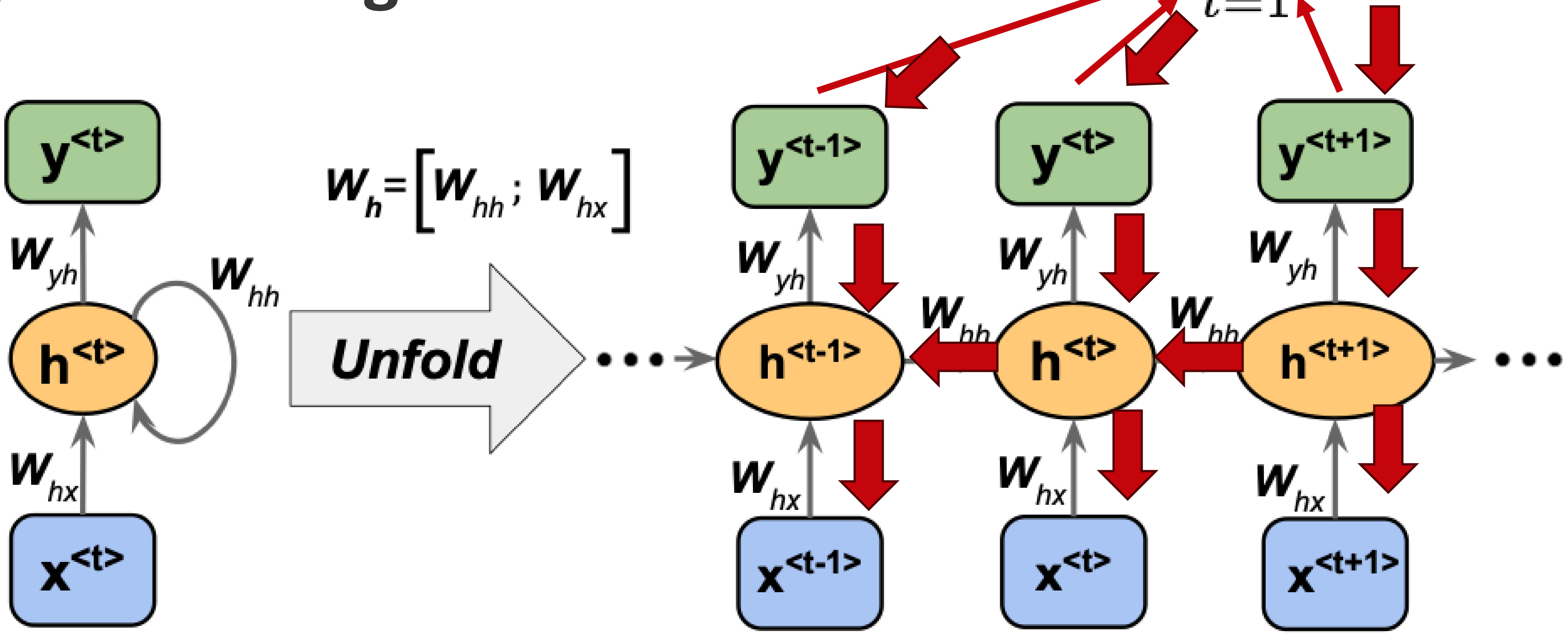


many-to-many

Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Backpropagation through time

$$L = \sum_{t=1}^{T} L^{\langle t \rangle}$$

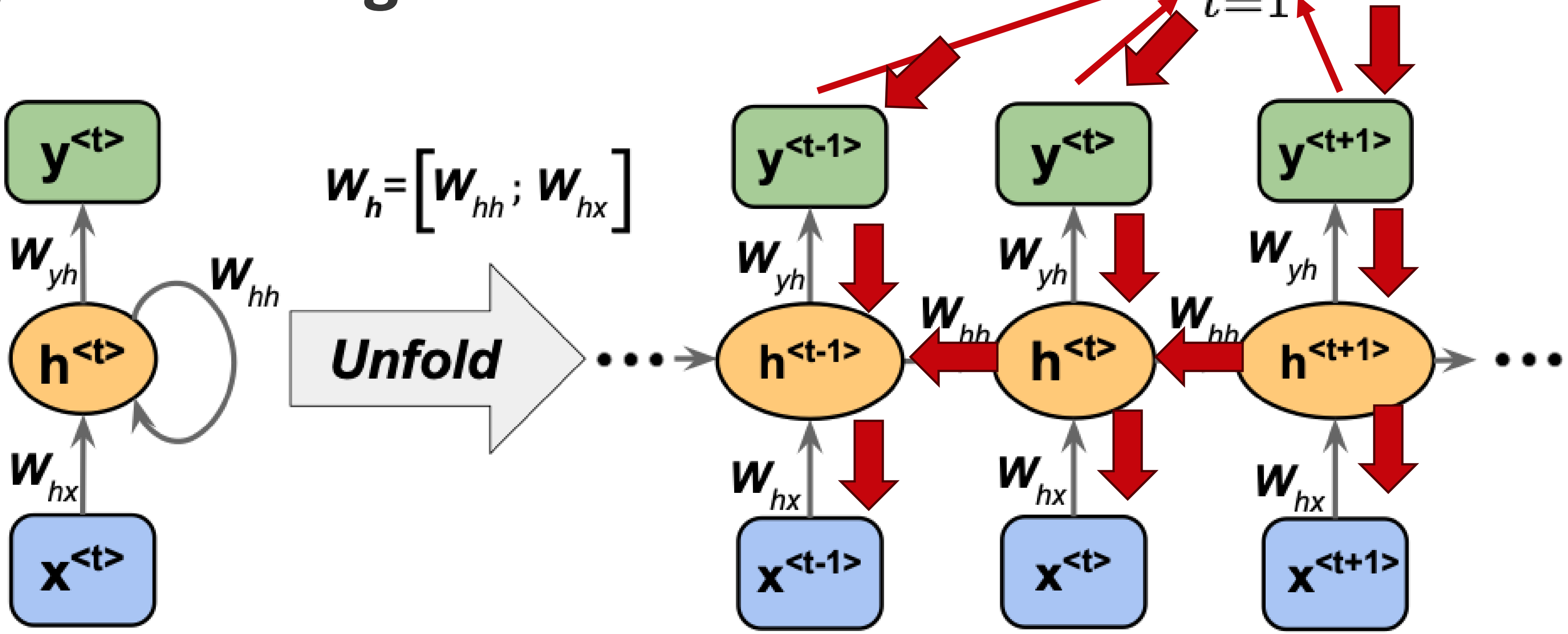


The overall loss can be computed as the sum over all time steps

Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Backpropagation through time

$$L = \sum_{t=1}^{T} L^{\langle t \rangle}$$

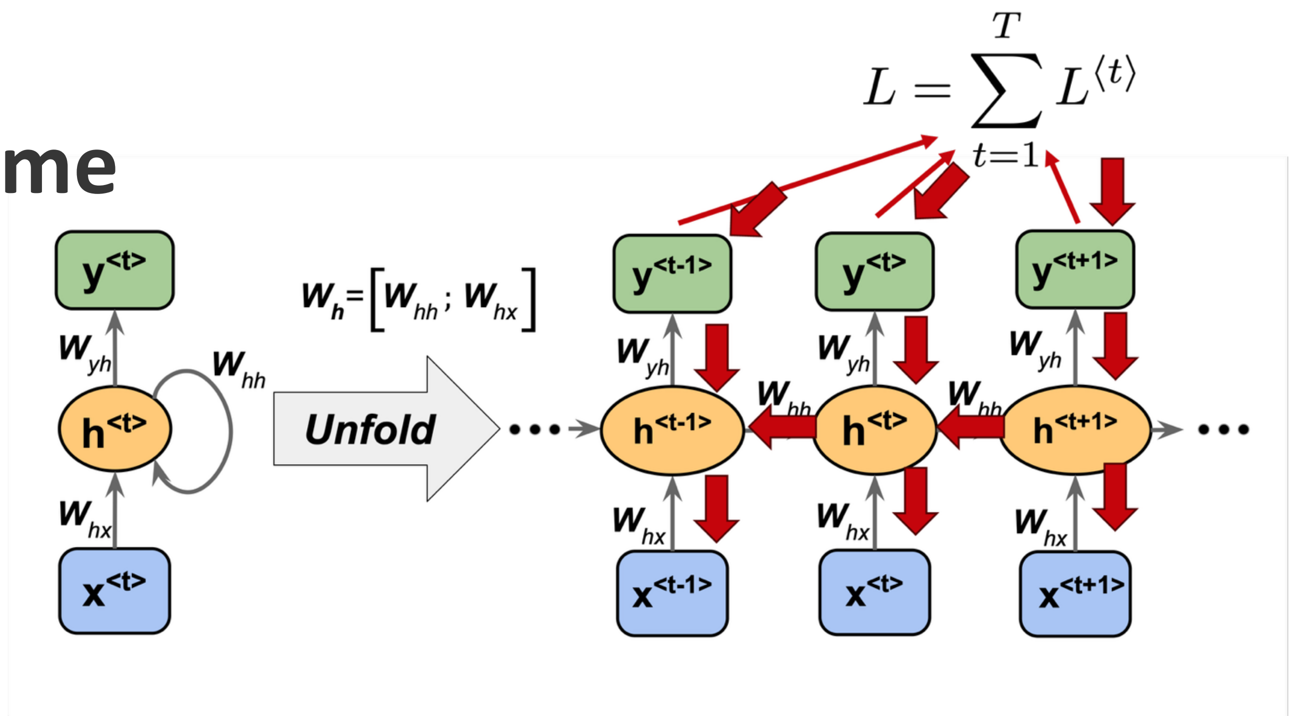


$$L = \sum_{t=1}^{T} L^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Backpropagation through time



Computed as a multiplication of adjacent time steps:

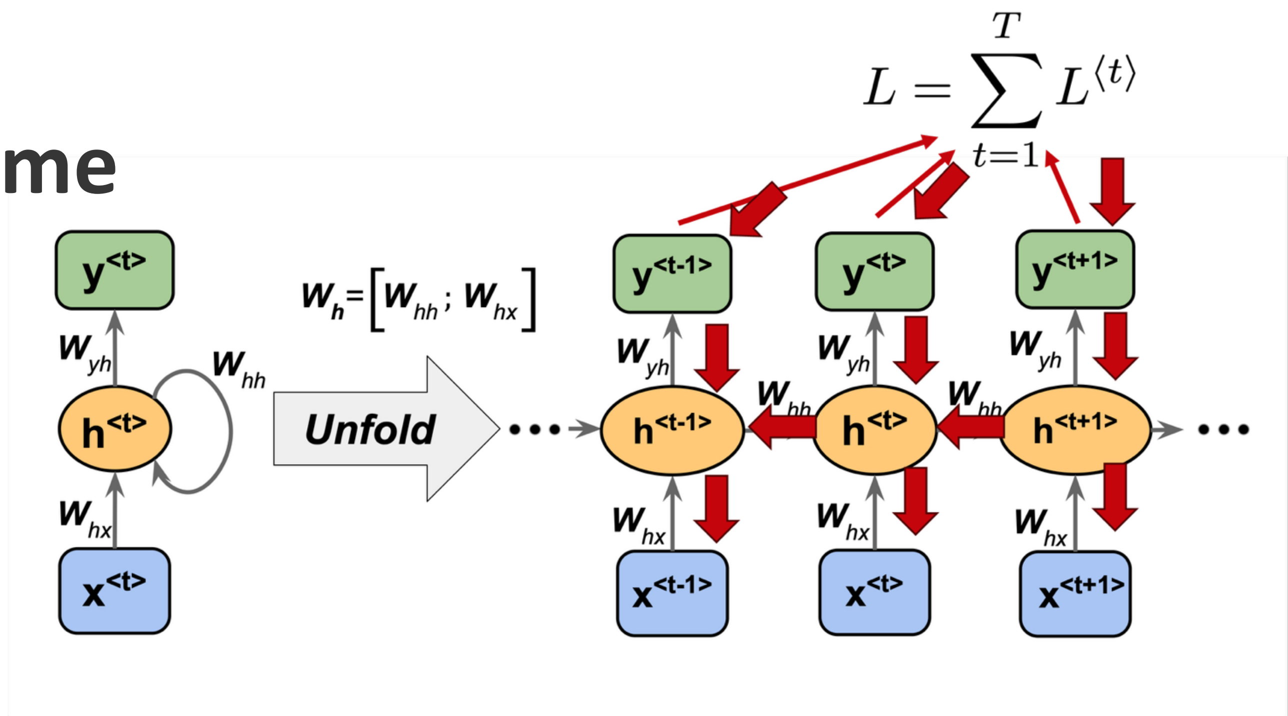
$$L = \sum_{t=1}^{T} L^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \boxed{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Backpropagation through time



Straightforward, but problematic:
vanishing / exploding gradients!

Computed as a multiplication of adjacent time steps:

$$L = \sum_{t=1}^{T} L^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \boxed{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Long-short term memory (LSTM)

- Not an oxymoron: **2 paths** of memory

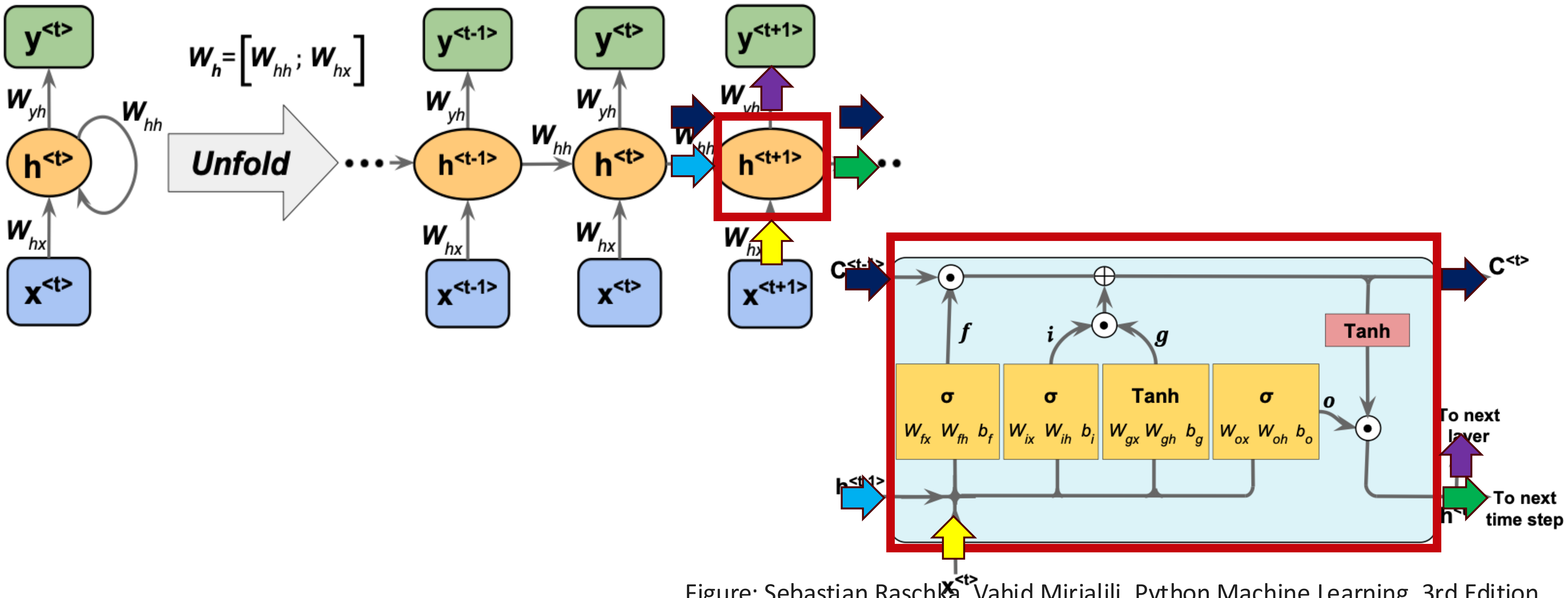


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

From RNN…

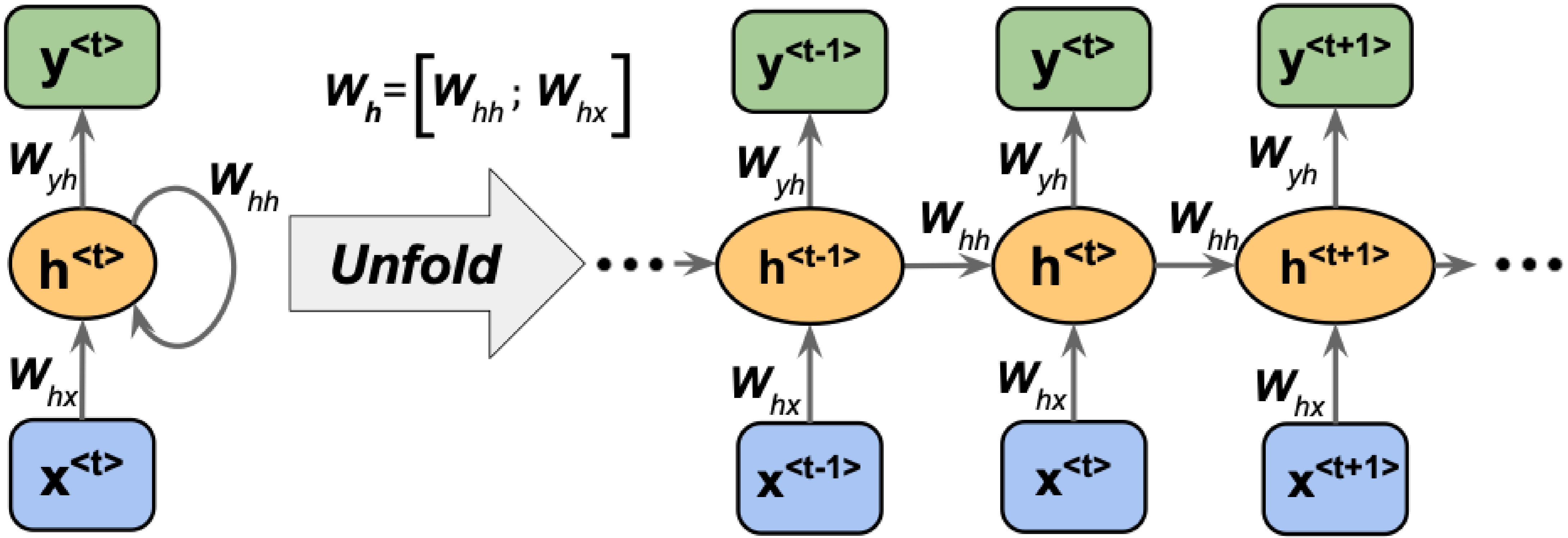


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

From RNN…to GPT

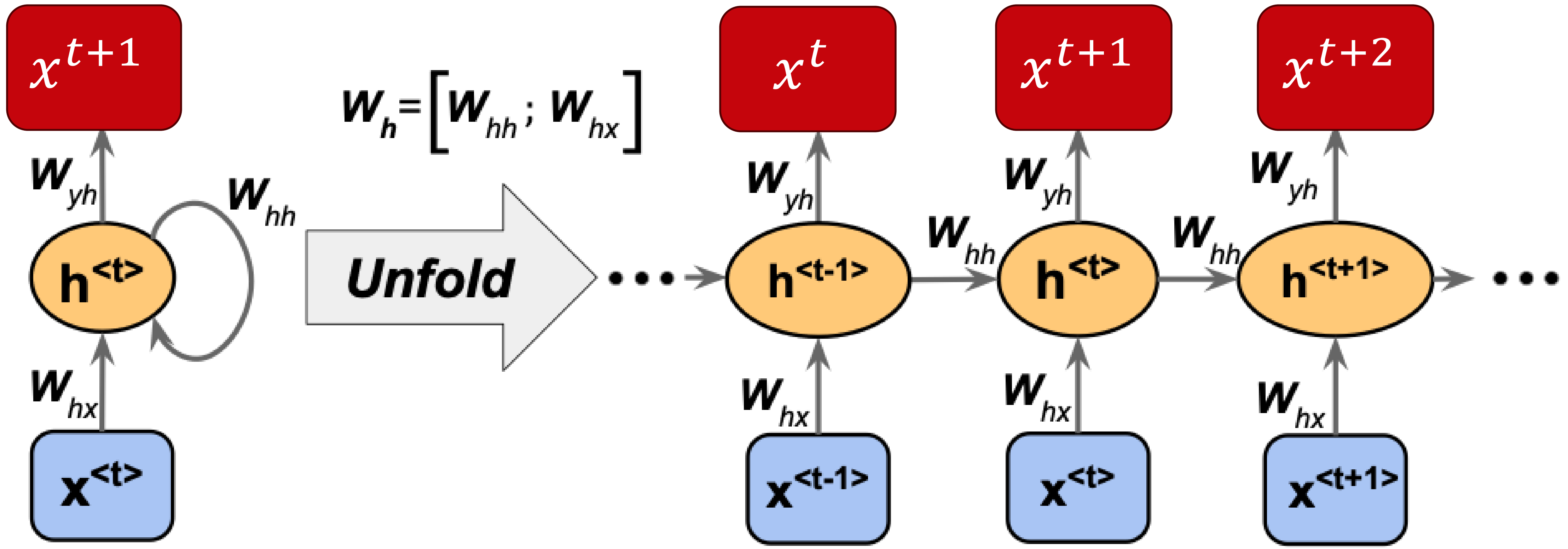
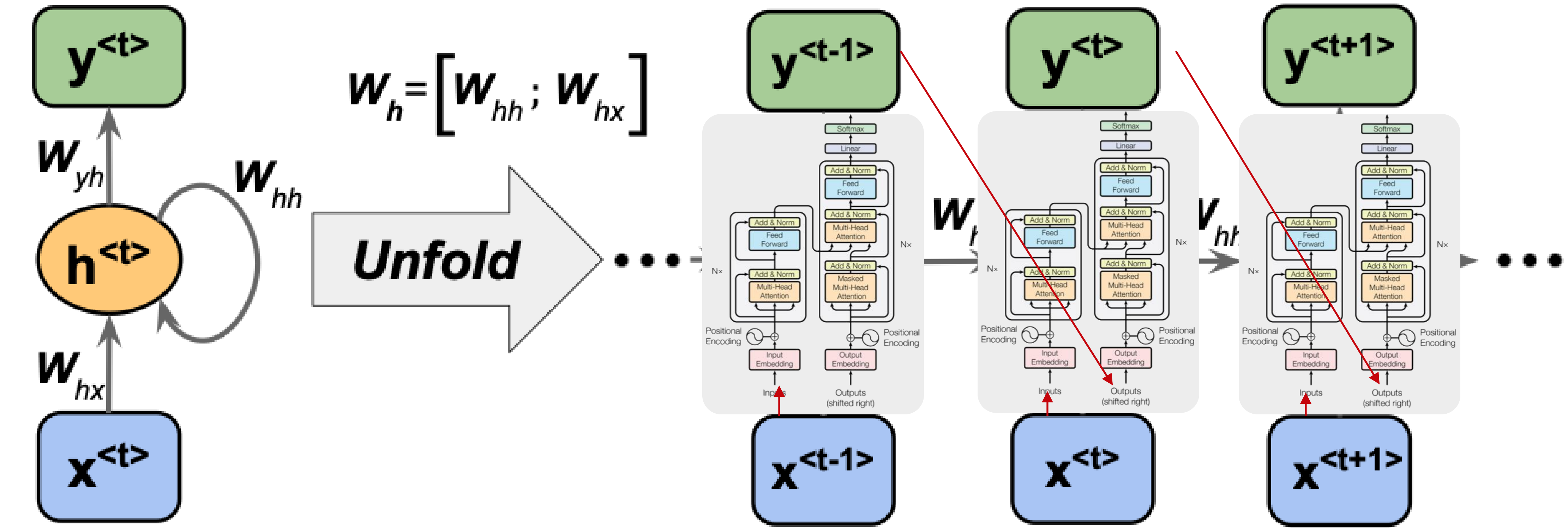


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

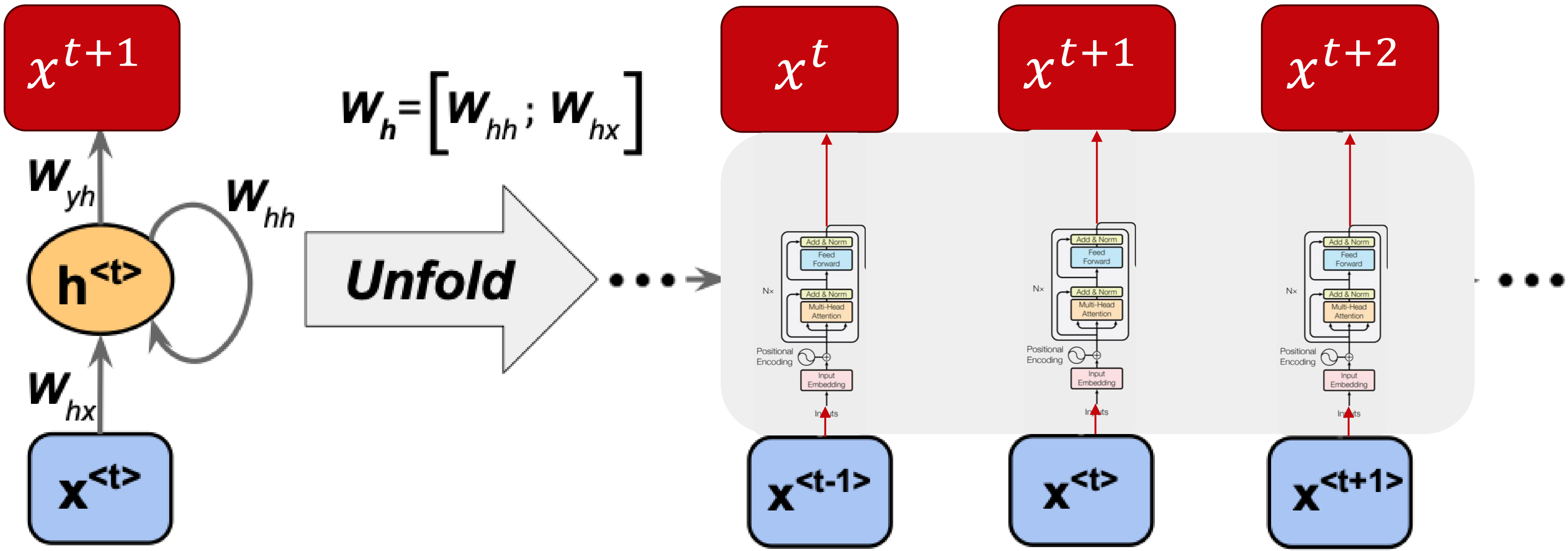
From RNN…to GPT…by Transformers



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

From RNN…to GPT…by Transformers



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

The Attention Mechansism

"Attention"

Main idea:

Assign attention weight to each word, to know how much "attention" the model should pay to each word

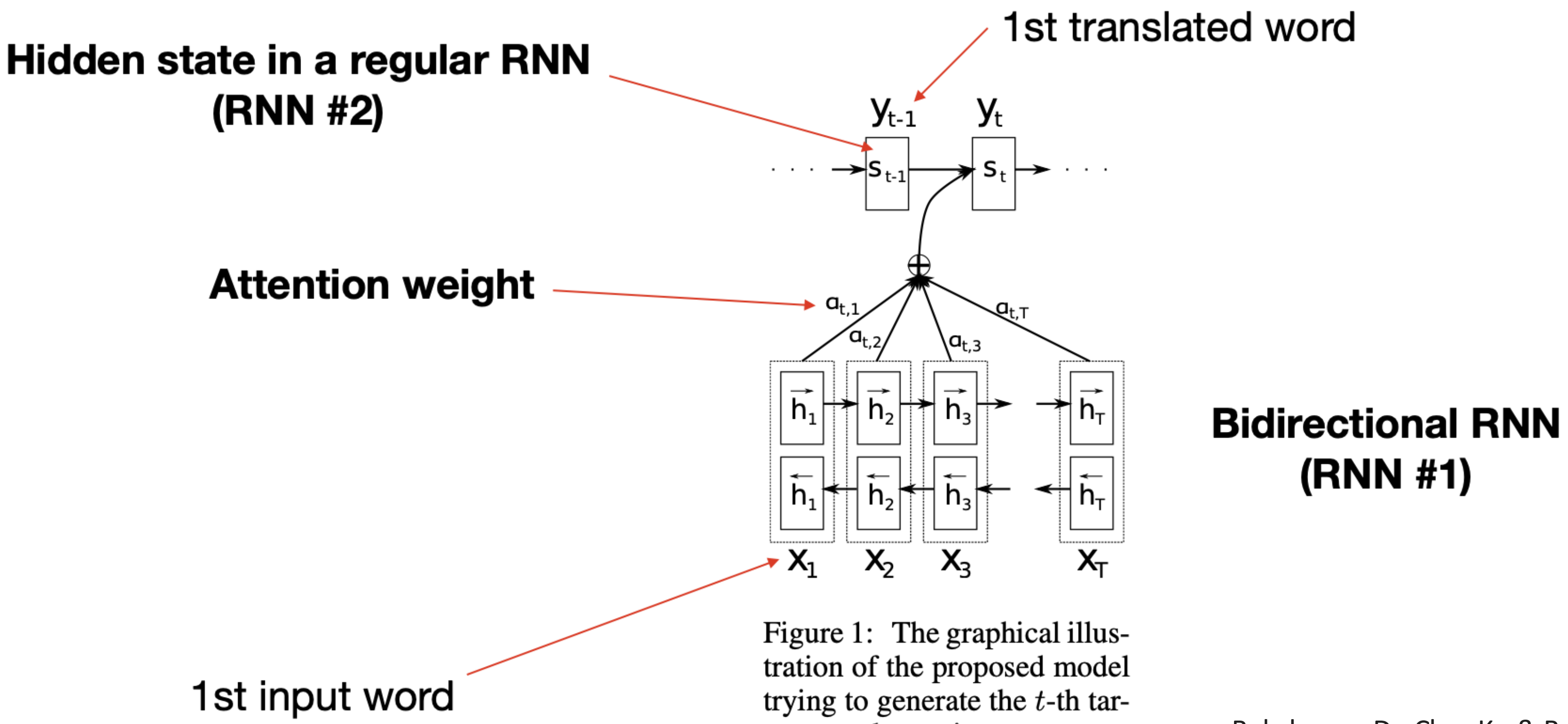
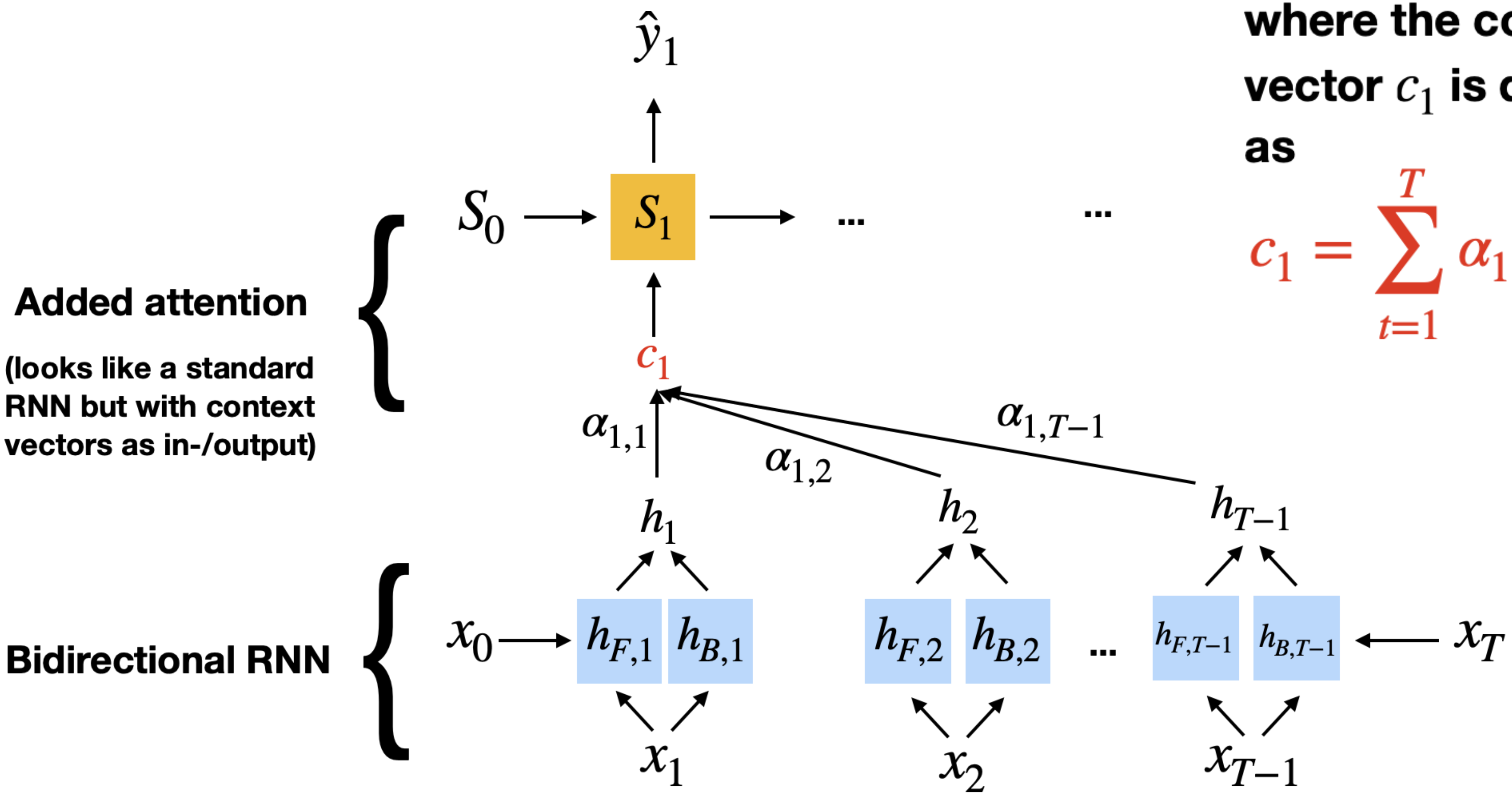


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473

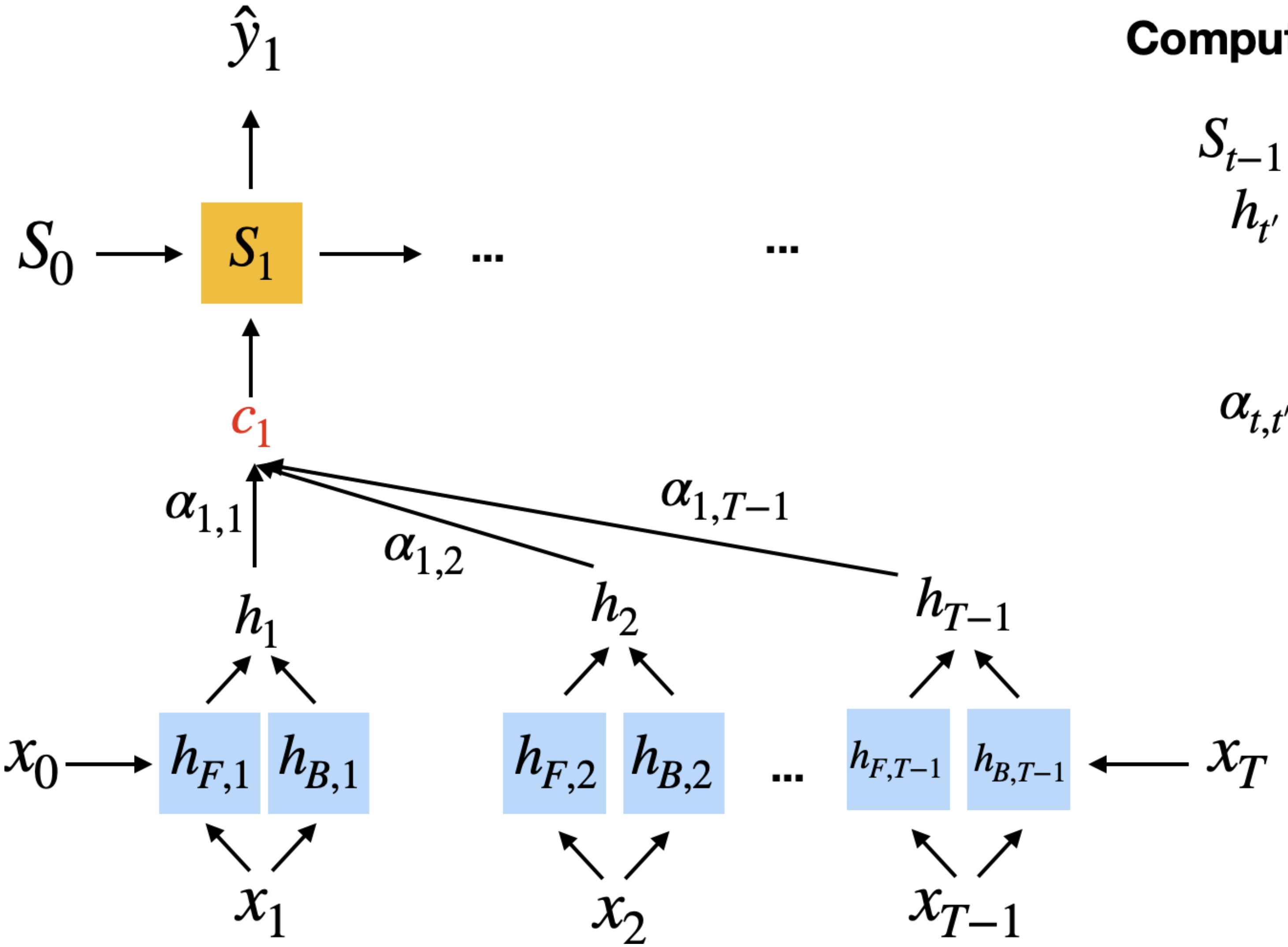
Soft attention



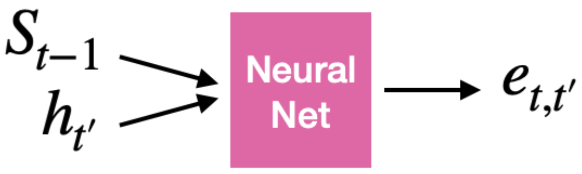
where the context vector c_1 is defined as

$$c_1 = \sum_{t=1}^{T} \alpha_{1,t}\, h_t$$

Soft attention



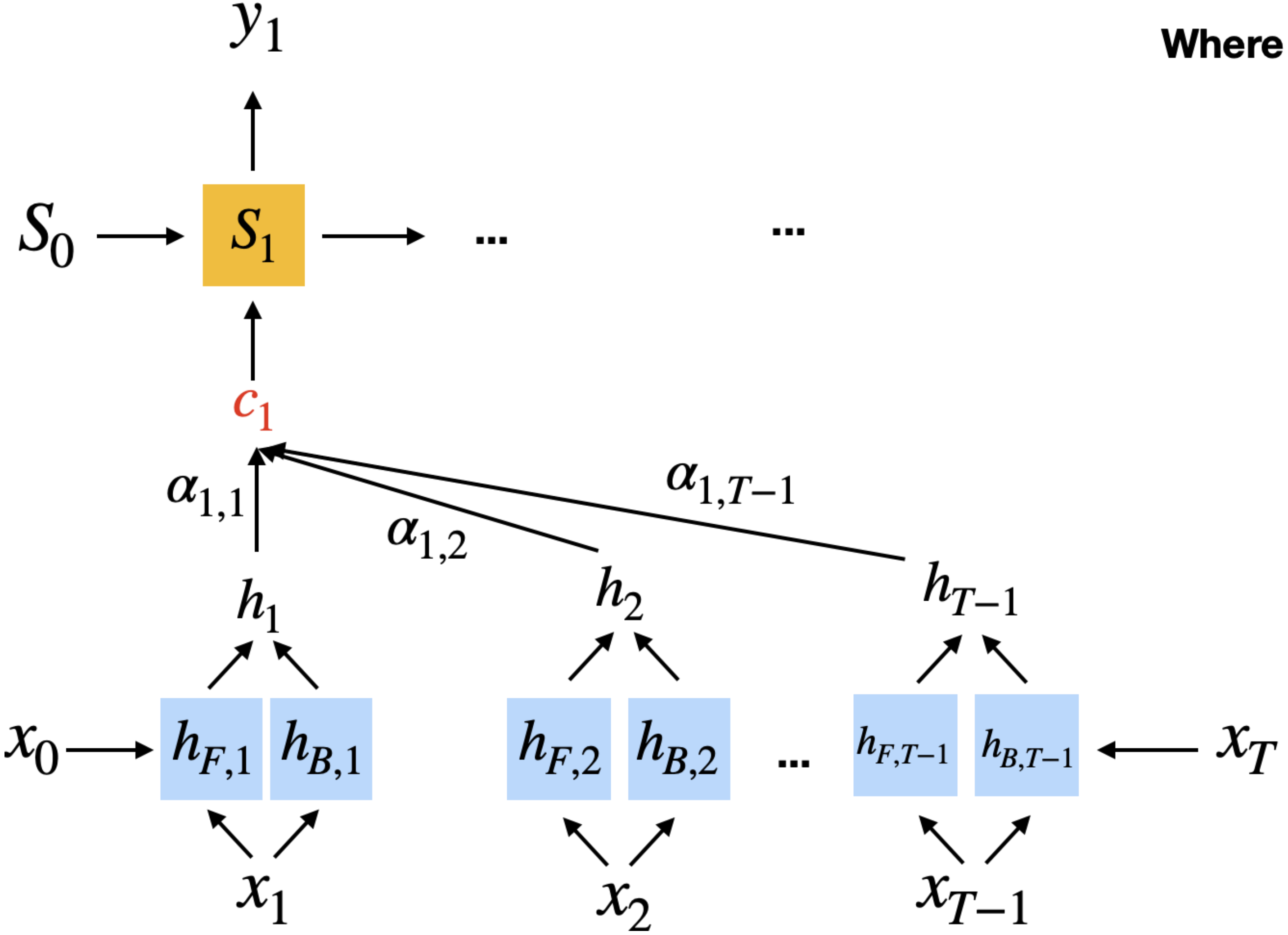
Computing attention weights



$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$

Self-Attention

”Original” (RNN) Attention Mechanism

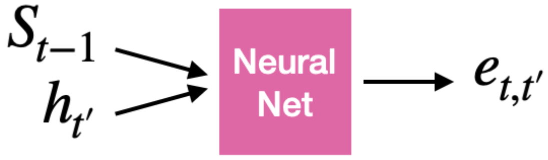


Where the context vector c_1 is defined as

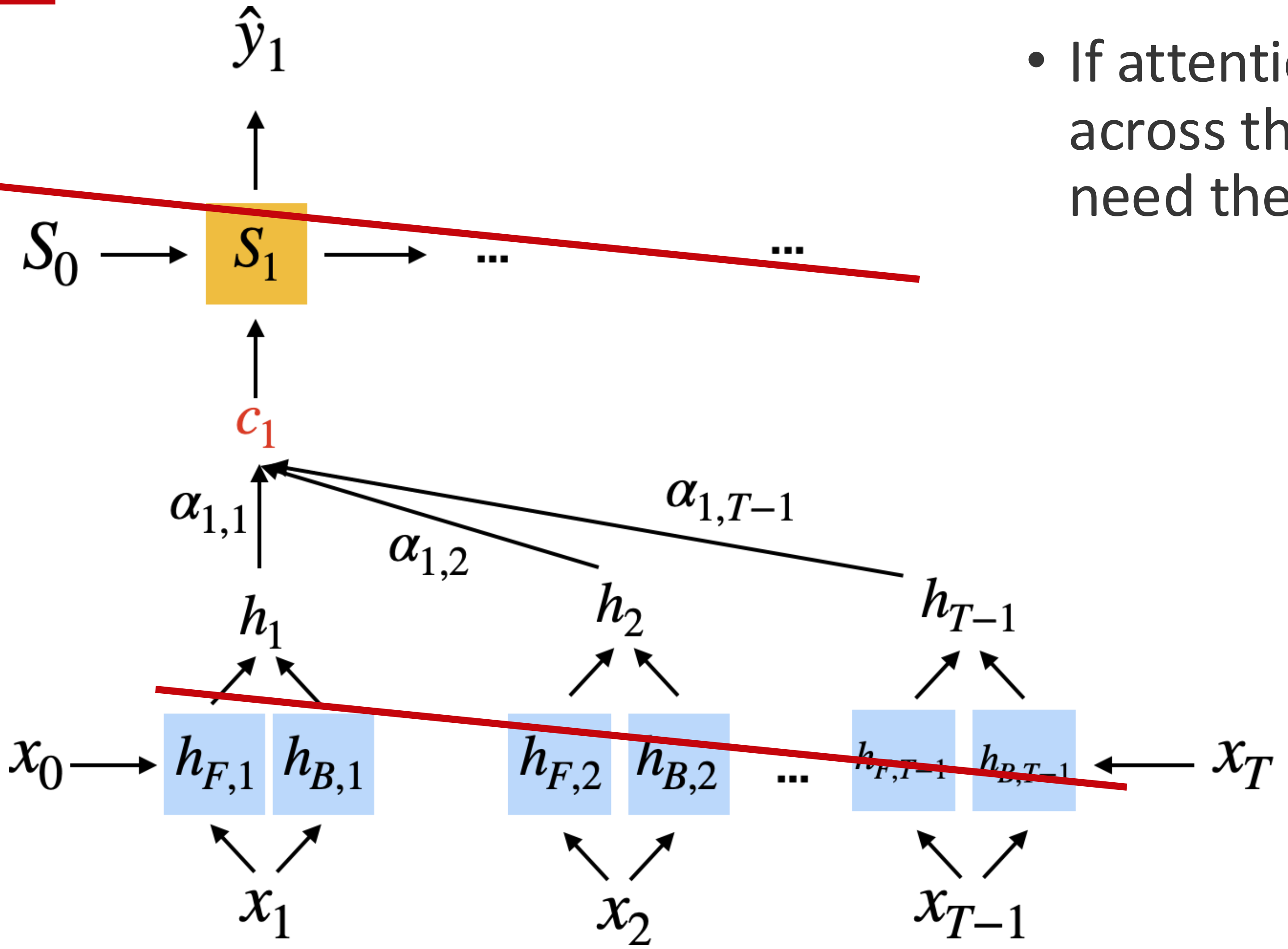
$$c_1 = \sum_{t=1}^{T} \alpha_{1,t}\, h_t$$

And the attention weights are

$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$



Can we get rid of the sequential parts?



- If attention already ties inputs across the sequence, do we really need the recurrence?

Self-attention (very basic form)

No learnable parameters?

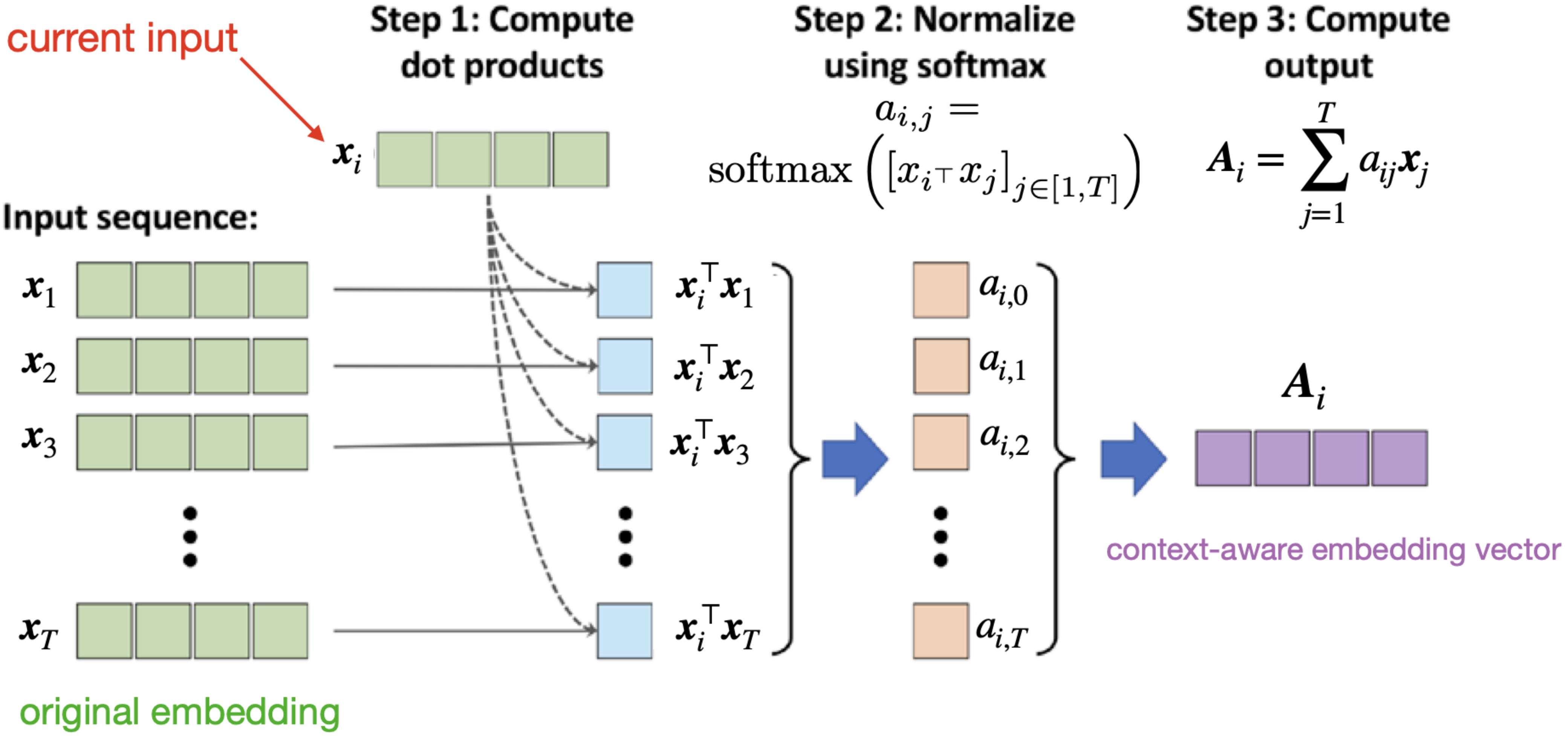


Image source: Raschka & Mirjalili 2019. Python Machine Learning, 3rd edition

Learnable Self-attention

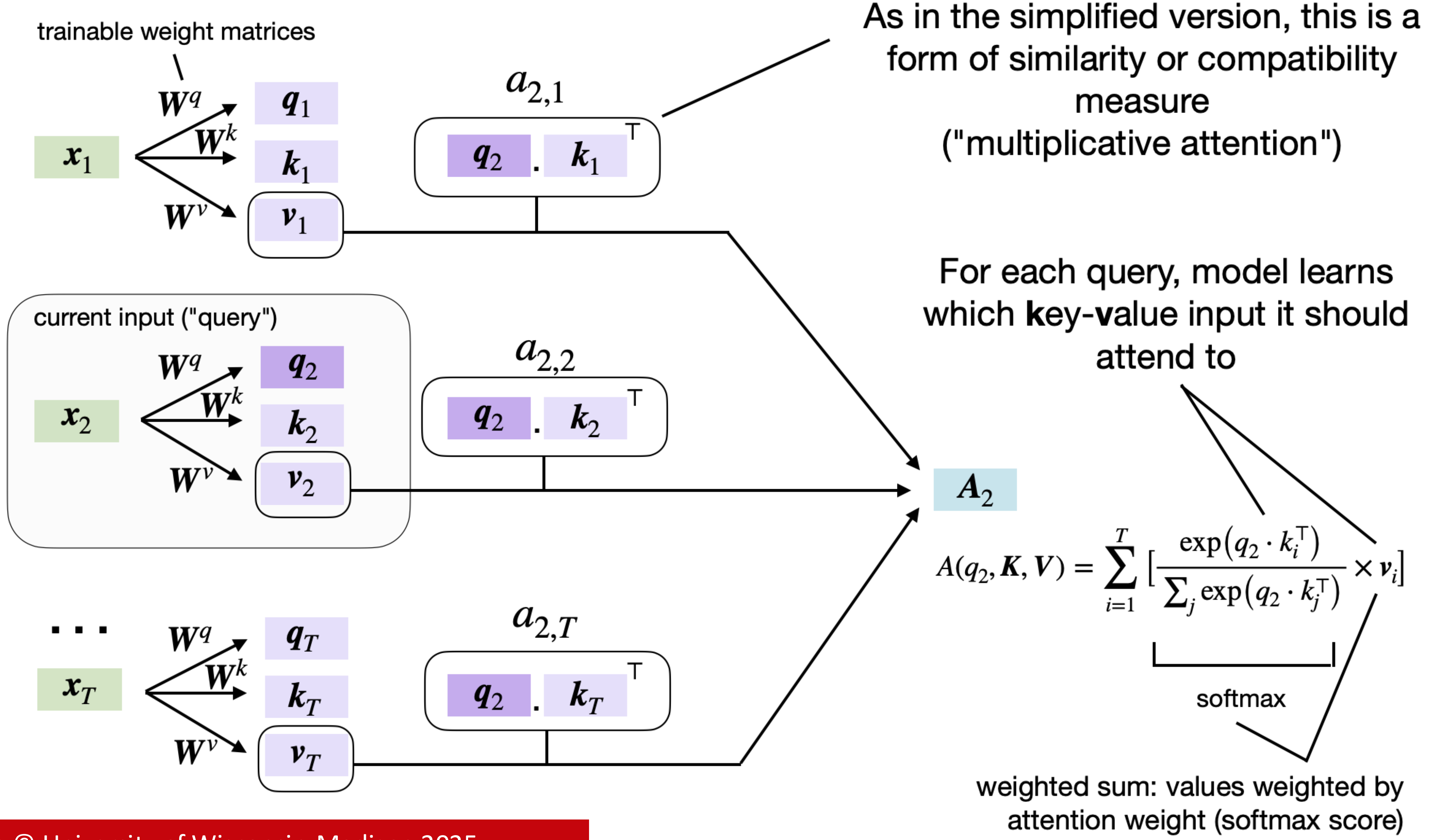
- Previous basic version did not involve any learnable parameters, so not very useful for learning a language model
- We are now adding 3 trainable weight matrices that are multiplied with the input sequence embeddings

$$\text{query} = \boldsymbol{W}^q \boldsymbol{x}_i$$

$$\text{key} = \boldsymbol{W}^k \boldsymbol{x}_i$$

$$\text{value} = \boldsymbol{W}^v \boldsymbol{x}_i$$

Learnable Self-attention



The Transformer

The ”Transformer”

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Attention is **all you need**

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

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

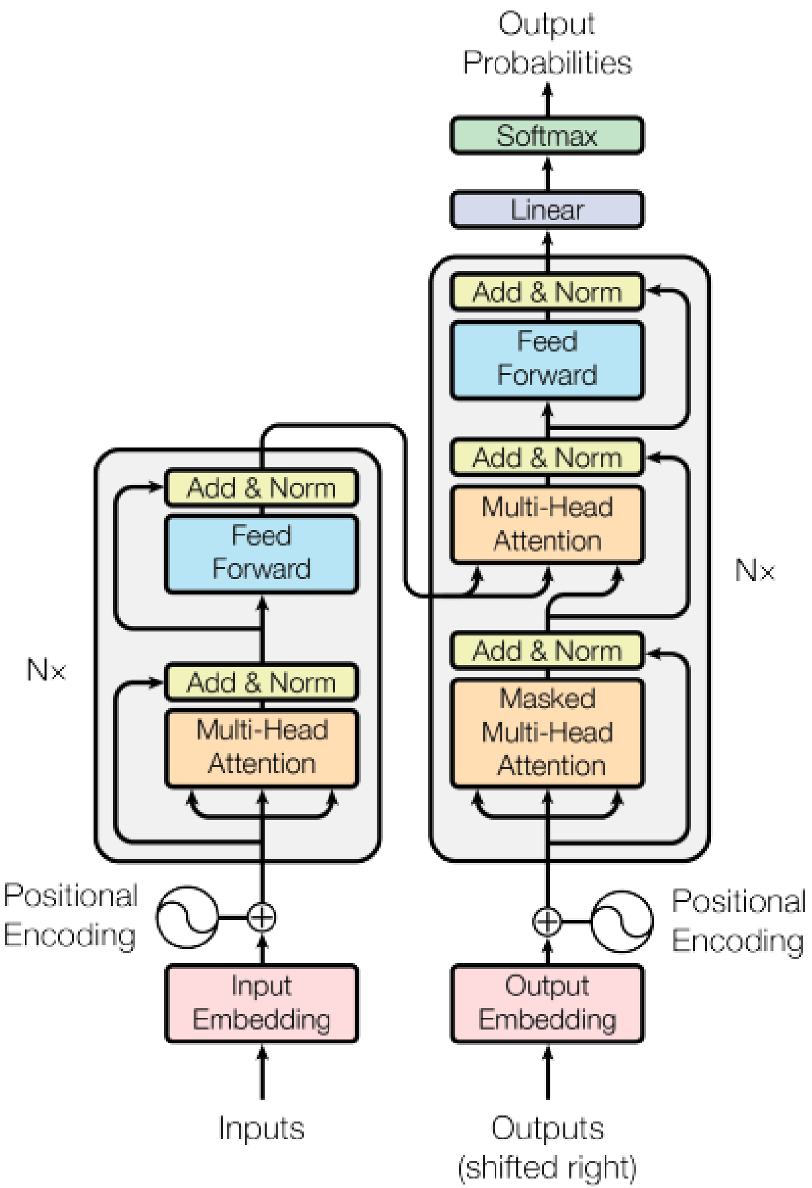
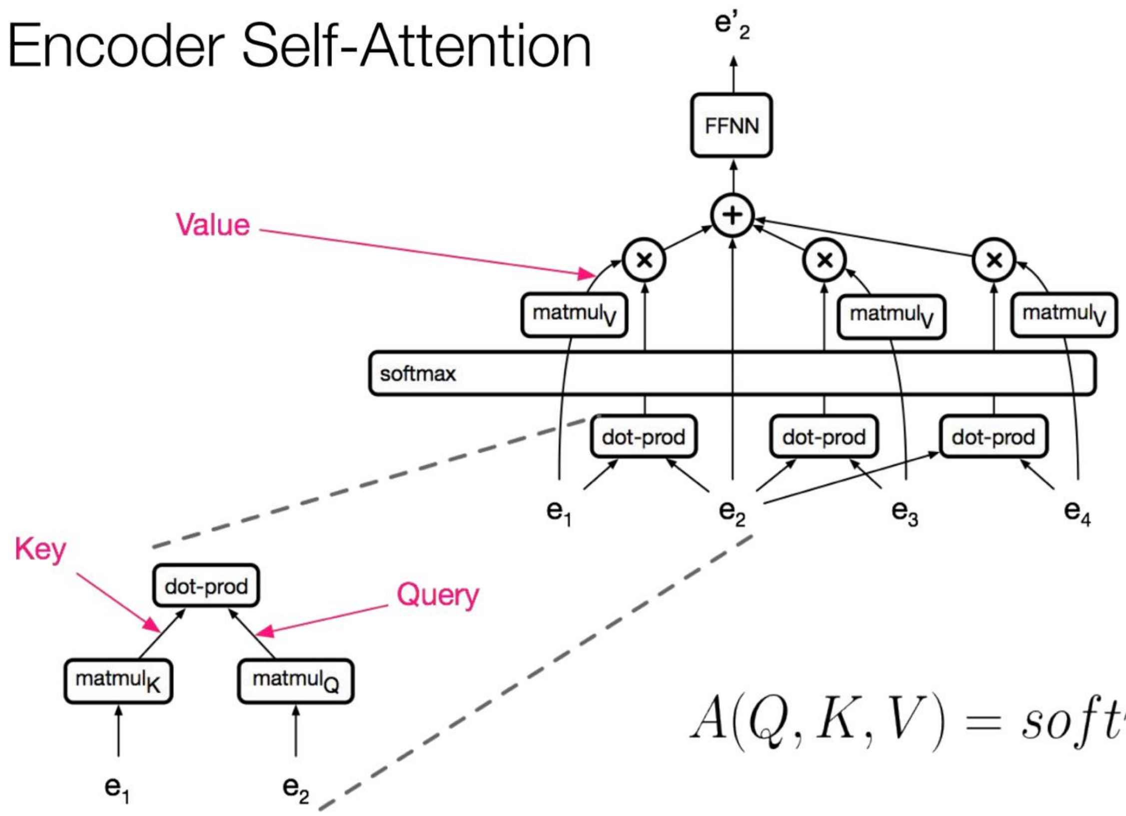


Figure 1: The Transformer - model architecture.

The "Transformer": Encoder



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

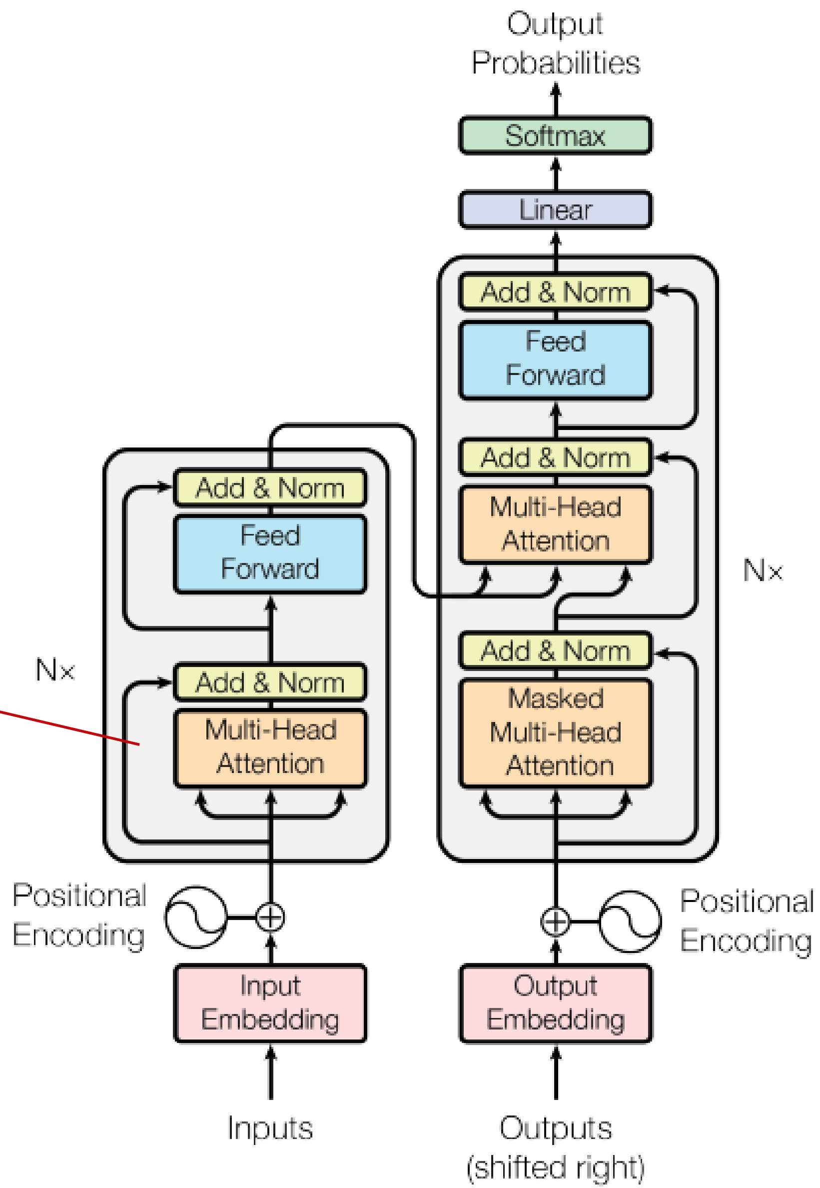
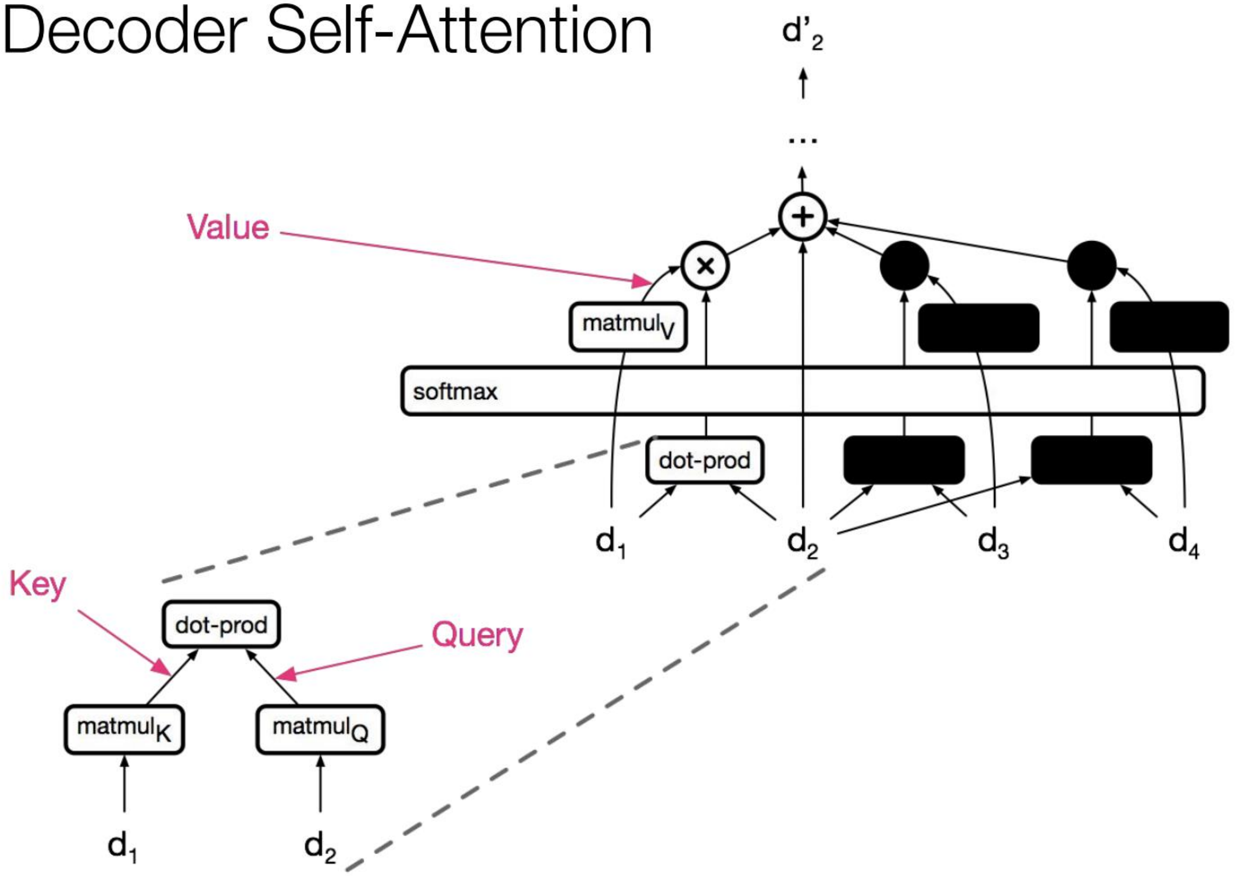


Figure 1: The Transformer - model architecture.

Vasvani "Self-Attention for Generative Models"

The ”Transformer”: Decoder

Decoder Self-Attention



Vasvani [“Self-Attention for Generative Models”](#)

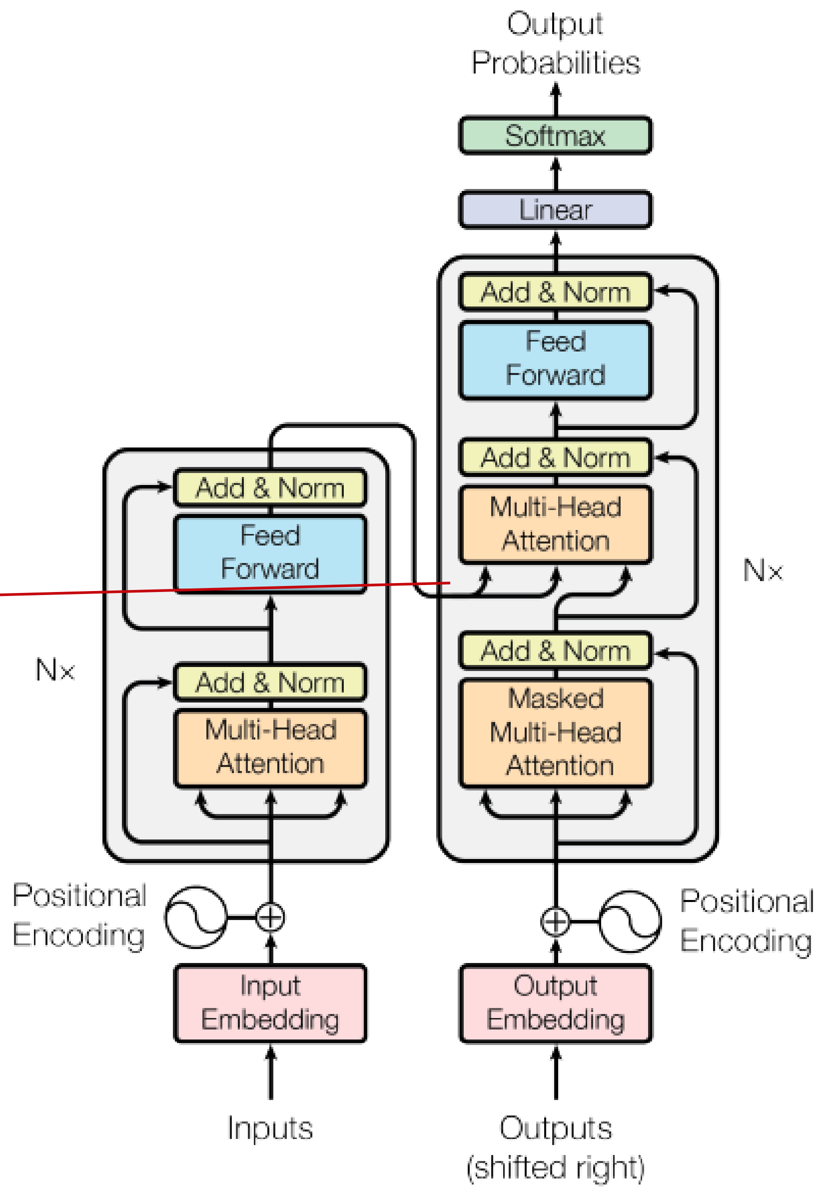


Figure 1: The Transformer - model architecture.

Foundation Models take the field

2017 - Transformer

2018 - GPT

2019 – GPT2

2020 – GPT3

2024 – GPT4

2024 – GPT5

117M params

1.5B params

175B params

>1T params

>?? params

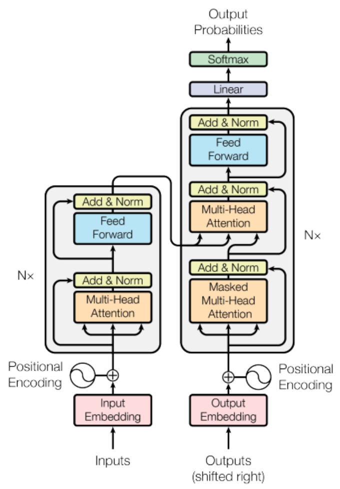
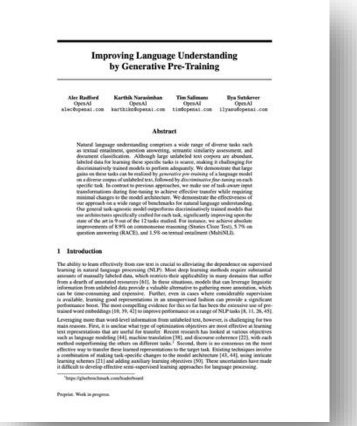


Figure 1: The Transformer - model architecture.

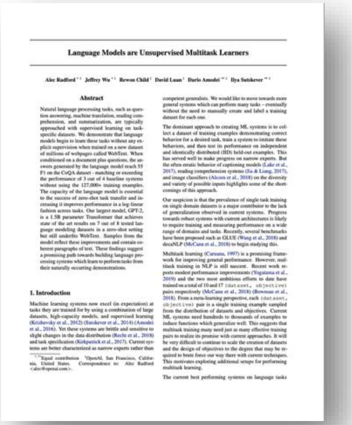


Improving Language Understanding by Generative Pre-Training

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever

Abstract

1 Introduction



Language Models are Unsupervised Multitask Learners

Abstract

1. Introduction

Language Models are Few-Shot Learners

arXiv:2005.14165v4 [cs.CL] 22 Jul 2020

OpenAI

Abstract

GPT-4 Technical Report

arXiv:2303.08774v6 [cs.CL] 4 Mar 2024

OpenAI

Abstract

1 Introduction

Generative Pre-trained Transfomers (GPT)

From Sequence Transduction to Sequence Modeling

- **Original Transformer** (Vaswani et al., 2017):

$$P(Y \mid X) = \prod_t P(Y_t \mid Y_{<t}, X)$$

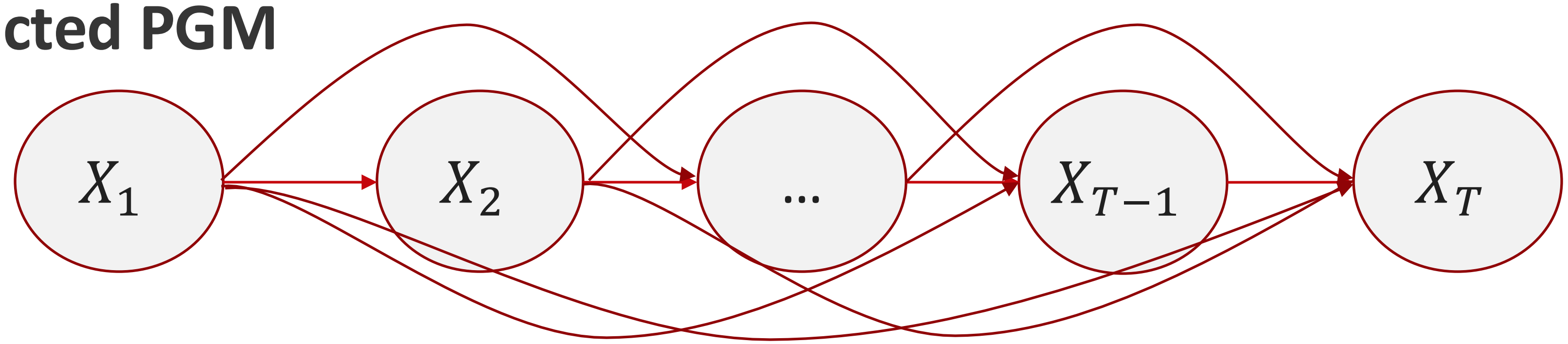
  - **Conditional** sequence model for tasks like translation (input → output)
- **Generative Pretrained Transformer (GPT)** Models:

$$P(X) = \prod_t P(X_t \mid X_{<t})$$

  - **Unconditional** generative model over raw text
- Architectural consequence: **no encoder**, only a decoder with causal structure

GPT = Probabilistic Model + Transformer Decoder

- **Directed PGM**



$$P_\theta(X) = \prod_i \prod_t P_\theta(X_{i,t} \mid X_{i,<t})$$

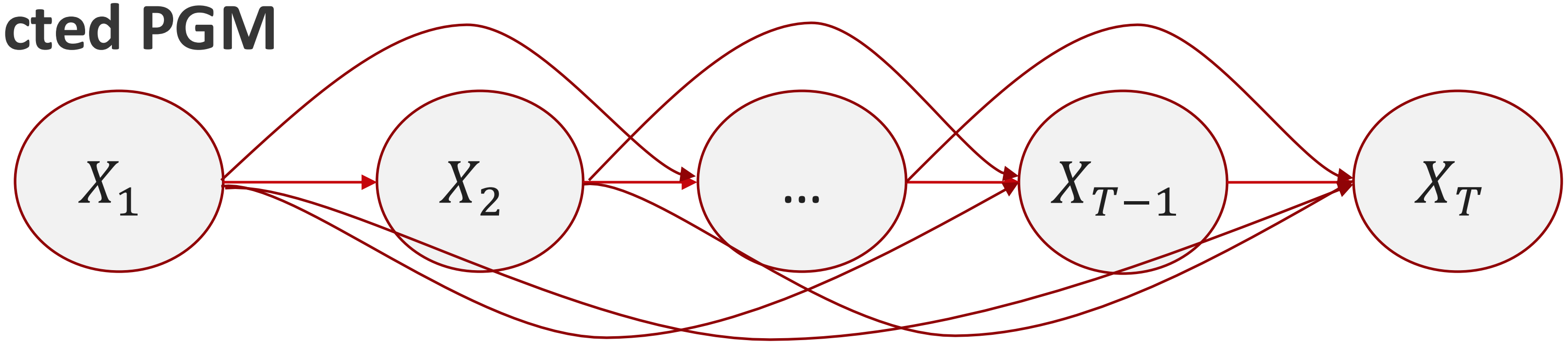
- **Probabilistic objective:** Max log-likelihood of observed seqs

$$\max_\theta \sum_i \sum_t \log P_\theta(X_{i,t} \mid X_{i,<t})$$

[Radford et al., Improving Language Understanding by Generative Pre-Training]

GPT = Probabilistic Model + Transformer Decoder

- **Directed PGM**



$$P_\theta(X) = \prod_i \prod_t P_\theta(X_{i,t} \mid X_{i,<t})$$

- **Model structure:**
  - Input: token embeddings + positional encodings
  - Masked multi-head attention: Enforces "causality"
  - Decoder stack: Learns $P(X_t \mid X_{<t})$
  - Output: softmax over vocabulary

[Radford et al., Improving Language Understanding by Generative Pre-Training]

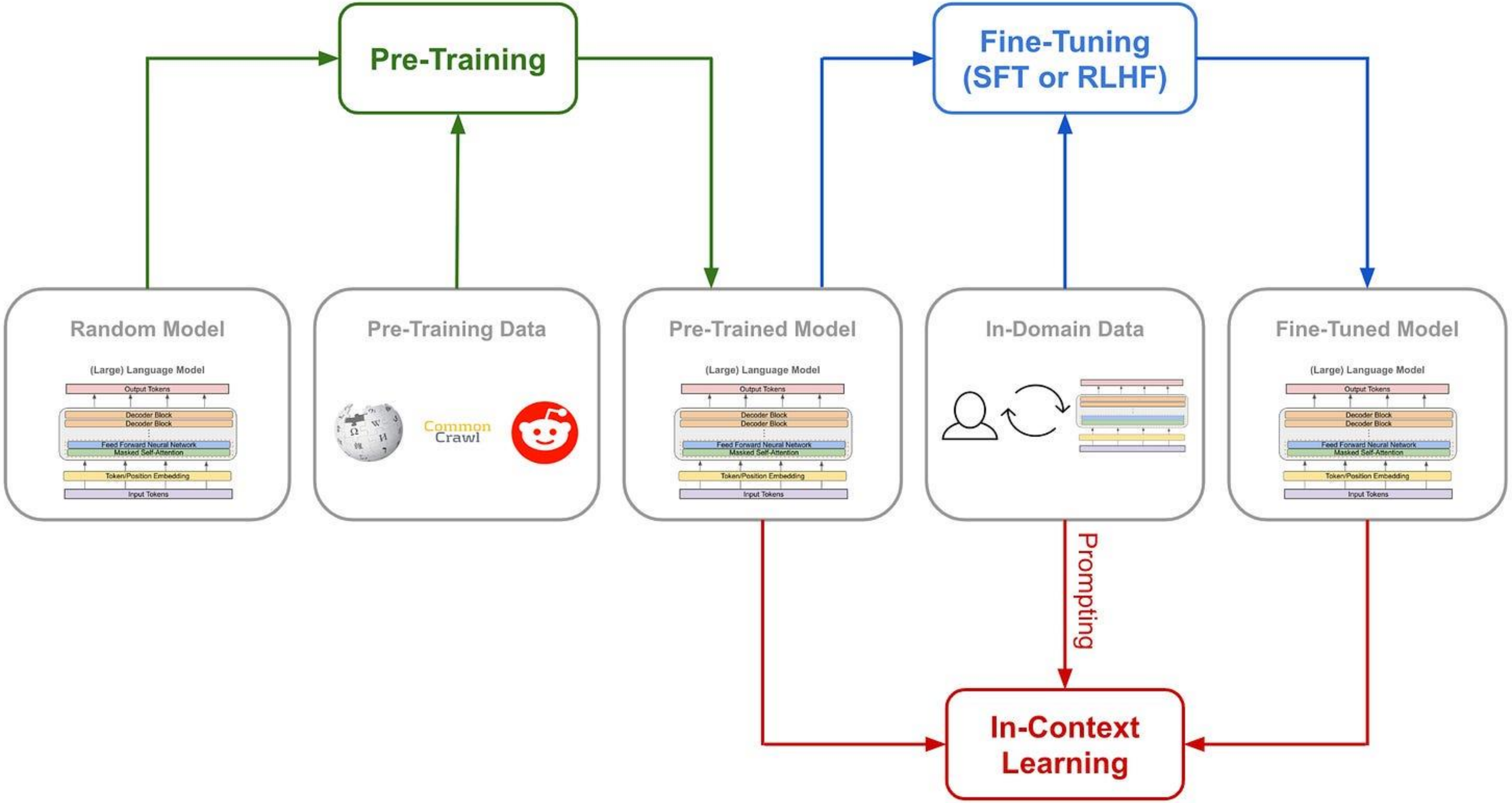
Summary: From Transformer to GPT

Component	Transformer	GPT
Architecture	Encoder-decoder (full)	Decoder-only
Attention	Full self-attention	Masked (causal) self-attention
Positional encoding	Sinusoidal (original)	Learned positional embeddings
Output	Task-specific	Next-token prediction
Training objective	Flexible (e.g., translation)	Language modeling (autoregressive)
Inference	Depends on task	Greedy / sampling for text gen

Summary: From GPT-1 to GPT-4

- **Architecture:**
  - **Scale:** Variety of options, with biggest (1.5B params → >1T params):
    - Block size (max context): 512 → 128k
    - Layers: 12 → >96
    - Attention Heads: 12 → >96
    - Embedding Dim: 768 → >12,288
    - Vocab: 40k → >50k tokens
  - Tokenizer: Includes image patches for multimodal
  - **Mixture-of-Experts**
- **Training:**
  - Dataset: BookCorpus (5GB) → Private 13T tokens (~50TB)
  - Reinforcement learning for alignment

Three phases of training

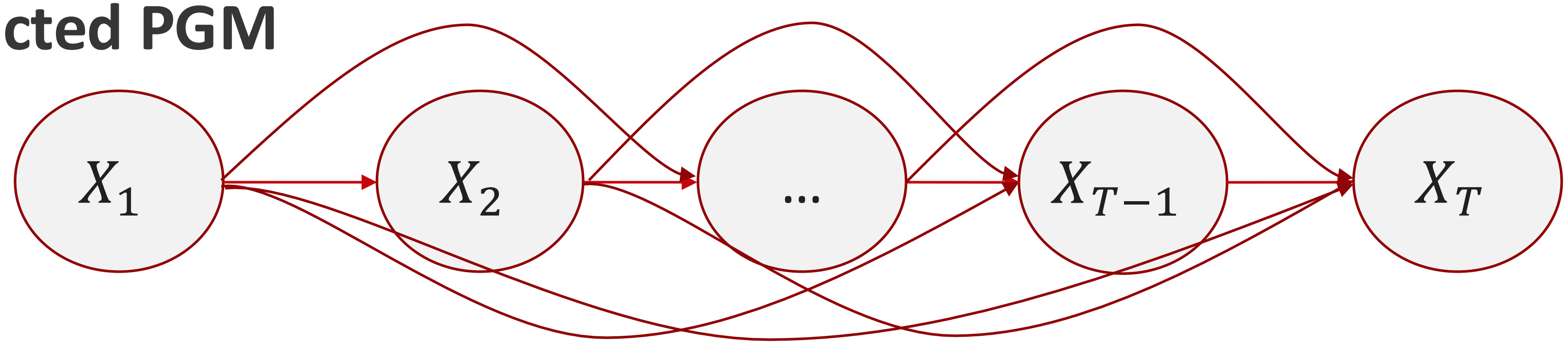


https://cameronrwolfe.substack.com/p/understanding-and-using-supervised

Unsupervised Training of LLMs

Recall GPT training objective: MLE

- **Directed PGM**



$$P_\theta(X) = \prod_i \prod_t P_\theta(X_{i,t} \mid X_{i,<t})$$

- **Probabilistic objective:** Max log-likelihood of observed seqs

$$\max_\theta \sum_i \sum_t \log P_\theta\left(X_{i,t} \mid X_{i,<t} \right)$$

[Radford et al., Improving Language Understanding by Generative Pre-Training]

What happens as we scale training?

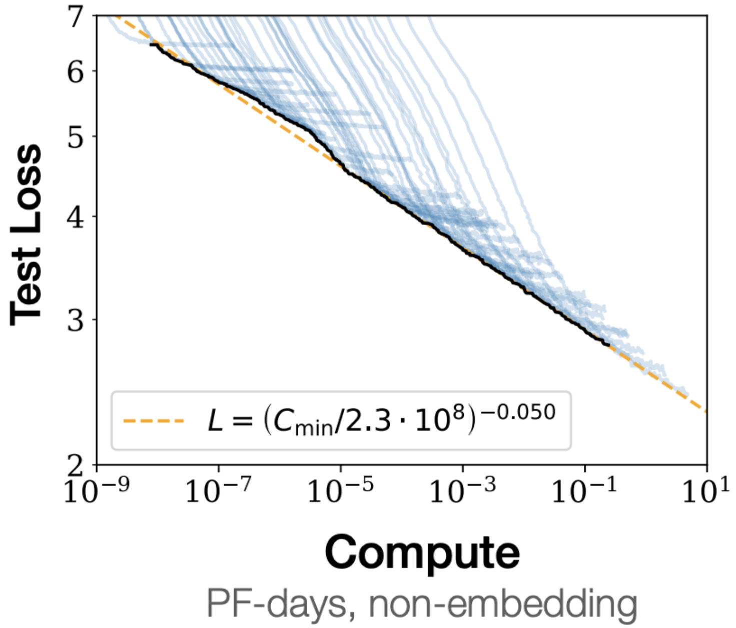


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

"Scaling Laws for Neural Language Models". Kaplan et al 2021

What happens as we scale training?

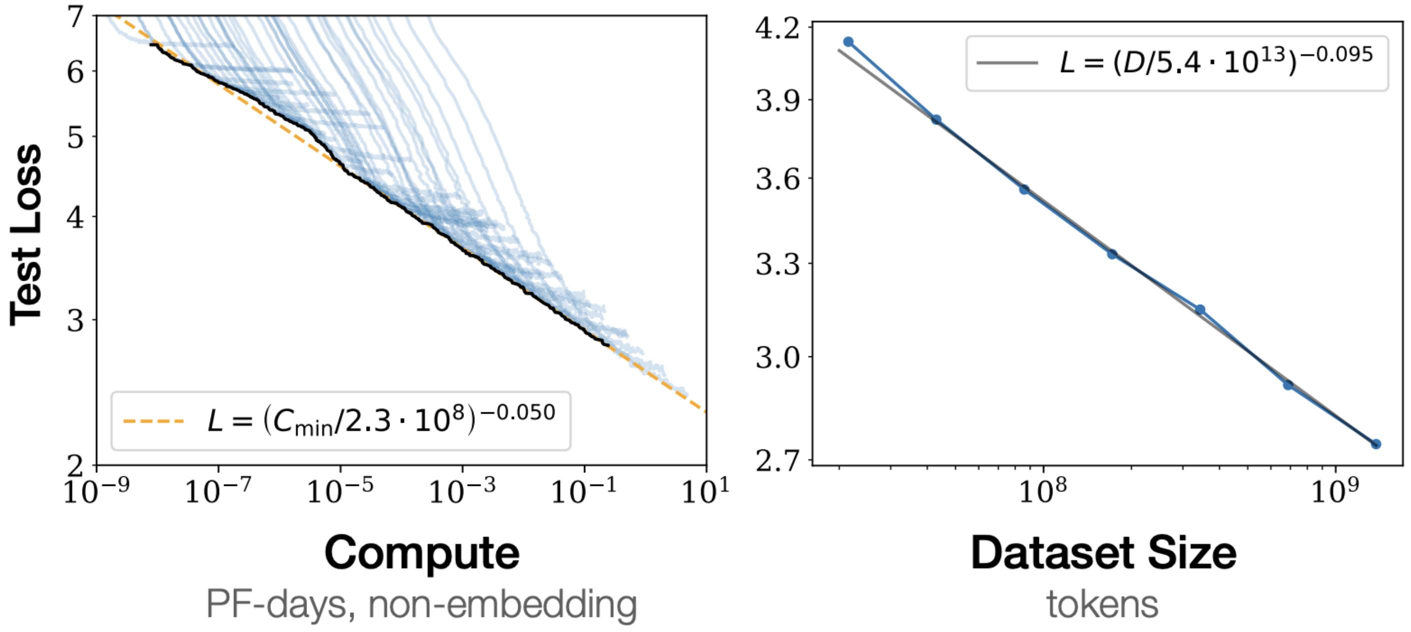


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

"Scaling Laws for Neural Language Models". Kaplan et al 2021

What happens as we scale training?

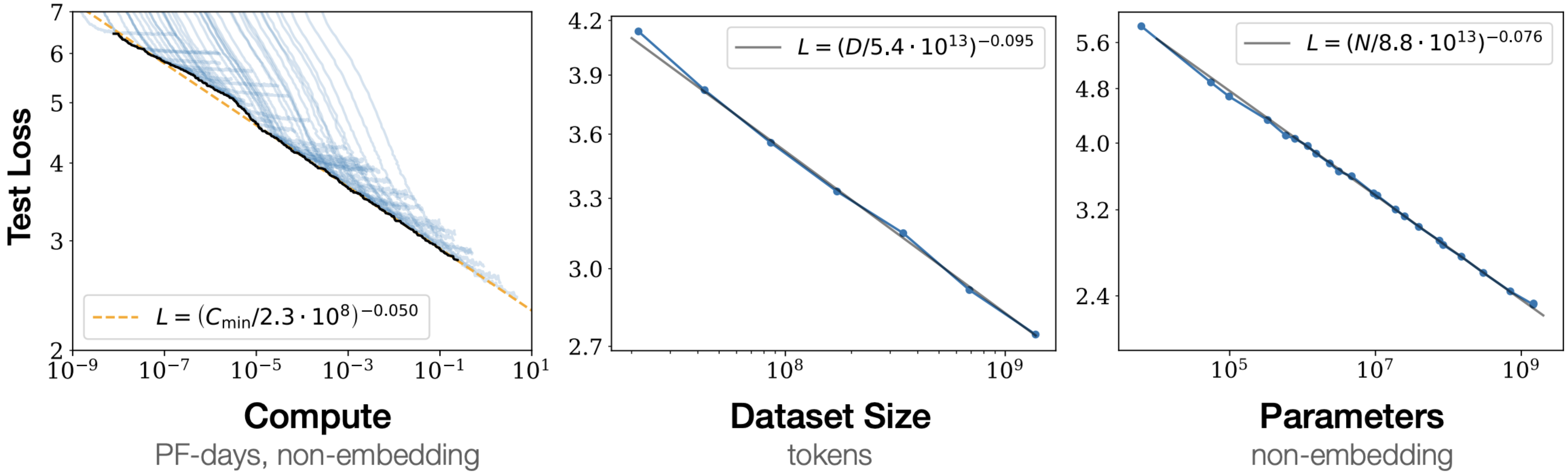
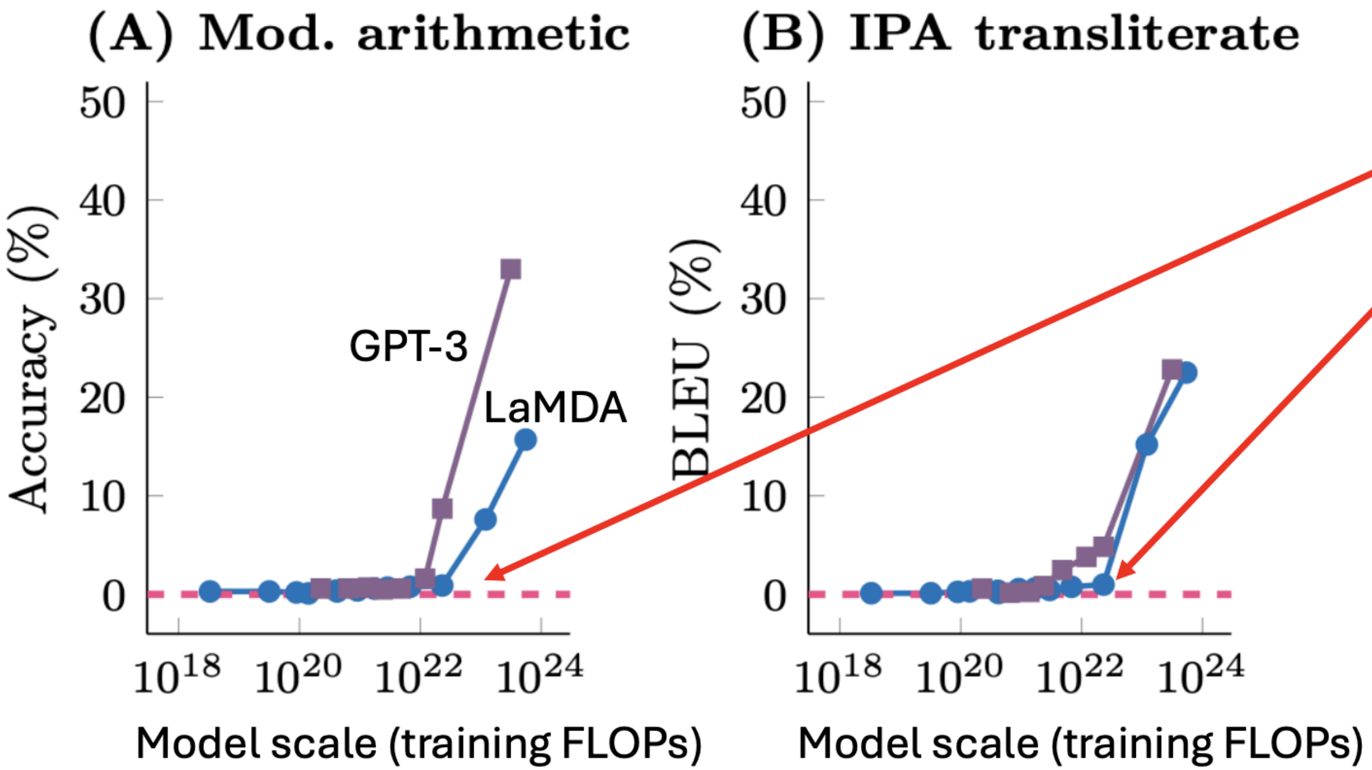


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

"Scaling Laws for Neural Language Models". Kaplan et al 2021

Smooth improvements → sharp emergent ability?



An ability is emergent if it is not present in smaller models but is present in larger models [Wei, et al (2022). Emergent Abilities of Large Language Models

What does MLE not do?

- No **task goals**
- No **explicit reward**
- No utility
- Dataset selection drives everything

Can we fine-tune our model to be **useful** after learning unsupervised P(X) learning?

Supervised Fine-Tuning of LLMs

Supervised Fine-Tuning (SFT)

- Show the language model how to appropriately respond to prompts of different types
- “Behavior cloning”
- InstructGPT

Training language models to follow instructions with human feedback

Long Ouyang* **Jeff Wu*** **Xu Jiang*** **Diogo Almeida*** **Carroll L. Wainwright***

Pamela Mishkin* **Chong Zhang** **Sandhini Agarwal** **Katarina Slama** **Alex Ray**

John Schulman **Jacob Hilton** **Fraser Kelton** **Luke Miller** **Maddie Simens**

Amanda Askell[†] **Peter Welinder** **Paul Christiano***[†]

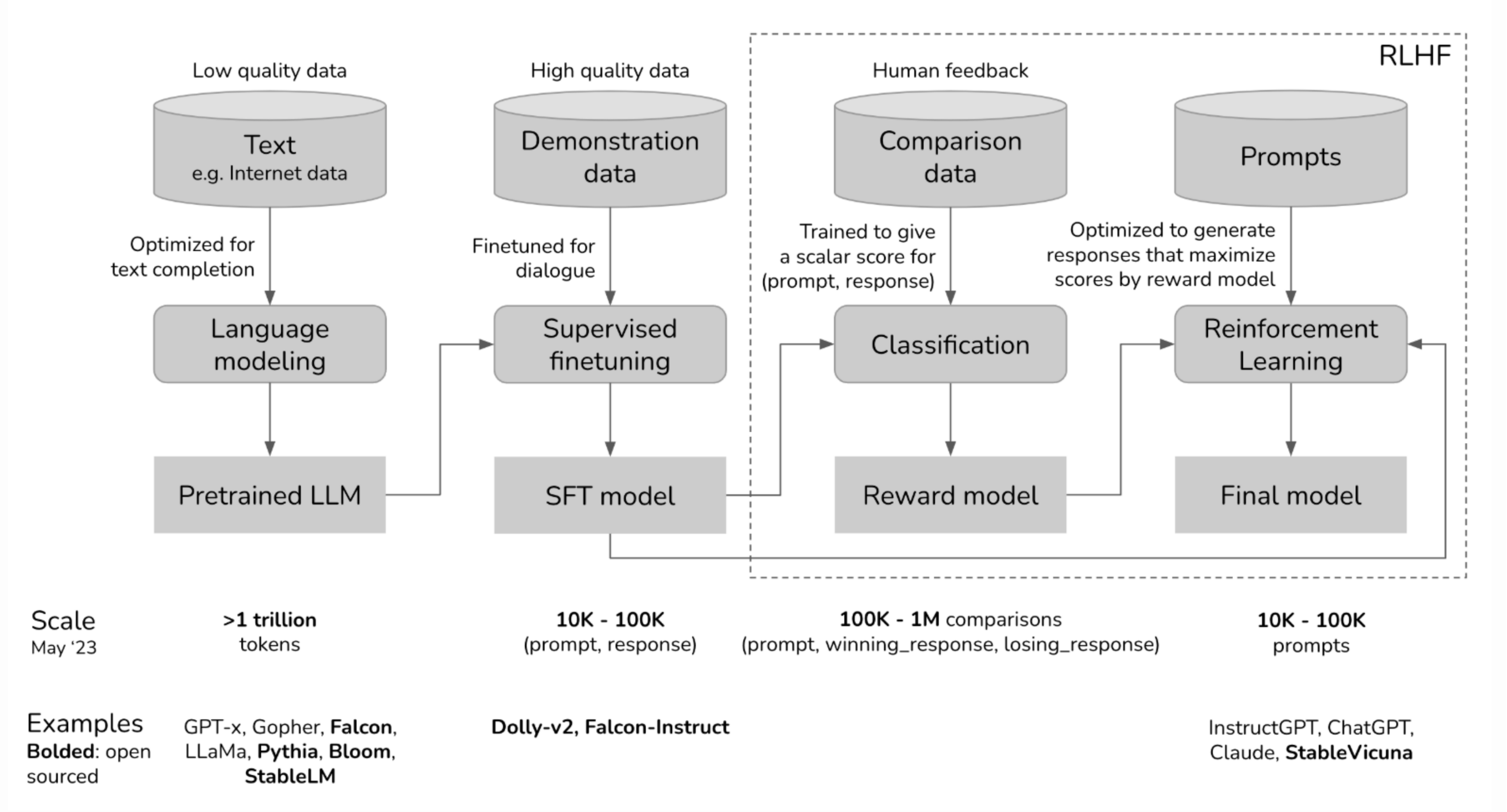
Jan Leike* **Ryan Lowe***

OpenAI

Prompt	Response
Serendipity means the occurrence and development of events by chance in a happy or beneficial way. Use the word in a sentence.	Running into Margaret and being introduced to Tom was a fortunate stroke of serendipity.
ELI5: What's the cause of the "anxiety lump" in our chest during stressful or disheartening experiences?	The anxiety lump in your throat is caused by muscular tension keeping your glottis dilated to maximize airflow. The clenched chest or heartache feeling is caused by the vagus nerve which tells the organs to pump blood faster, stop digesting, and produce adrenaline and cortisol.
Create a shopping list from this recipe: Trim the ends off zucchini. Cut zucchini in half lengthwise; scoop out pulp, leaving 1/2-in. shells. Finely chop pulp. In a skillet, cook beef, zucchini pulp, onion, mushrooms and peppers over medium heat until meat is no longer pink; drain. Remove from the heat. Add 1/2 cup cheese, ketchup, salt and pepper; mix well. Spoon into the zucchini shells. Place in a greased 13x9-in. baking dish. Sprinkle with remaining cheese.	Zucchini, beef, onion, mushroom, peppers, cheese, ketchup, salt, pepper

https://huyenchip.com/2023/05/02/rlhf.html

Reinforcement Learning with Human Feedback



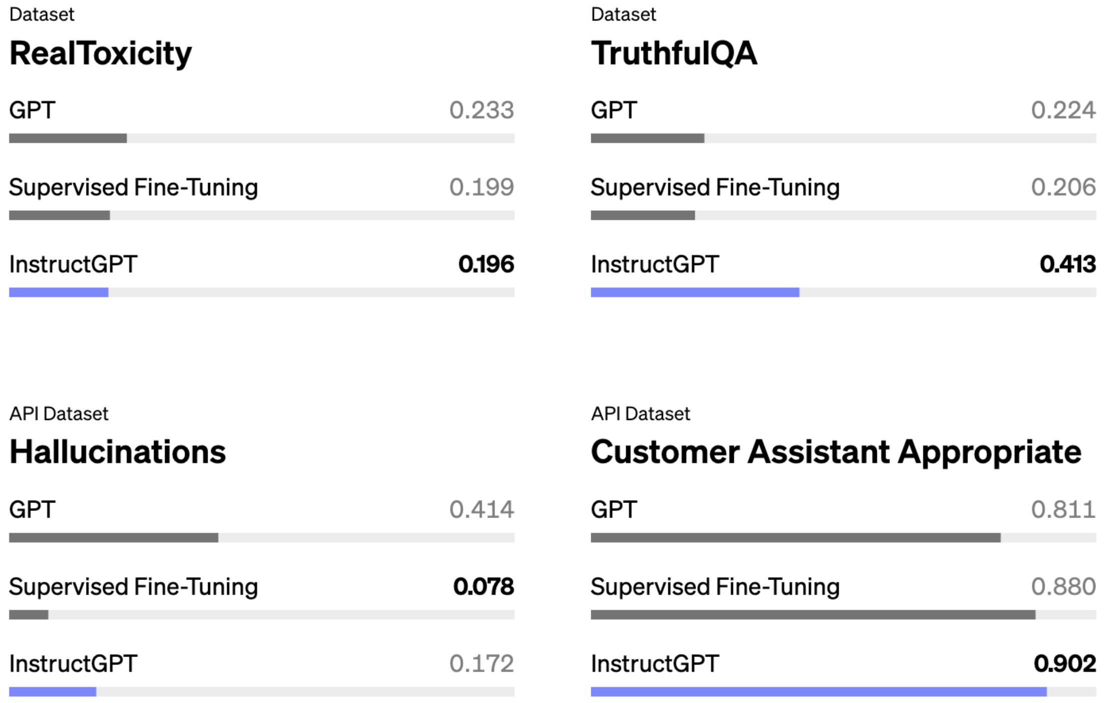
https://huyenchip.com/2023/05/02/rlhf.html

Does human feedback reduce model hallucinations?

How to Fix with RL

- 1) Adjust output distribution so model is allowed to express uncertainty, challenge premise, admit error. (Can use behavior cloning.)
- 2) Use RL to precisely learn behavior boundary.
  - Reward(x) = {
    1 if unhedged correct (The answer is y)
    0.5 if hedged correct (The answer is likely y)
    0 if uninformative (I don't know)
    -2 if hedged wrong (The answer is likely z)
    -4 wrong (The answer is z)
    }
  - This reward is similar to log loss, or a proper scoring rule

John Schulman 2023



Evaluating InstructGPT for toxicity, truthfulness, and appropriateness. Lower scores are better for toxicity and hallucinations, and higher scores are better for TruthfulQA and appropriateness. Hallucinations and appropriateness are measured on our API prompt distribution. Results are combined across model sizes.

https://huyenchip.com/2023/05/02/rlhf.html

Reinforcement Learning with Verifiable Rewards

- RLVR
- Better than human feedback: verifiable truth
- Examples:
  - Code generation (verify: does it run correctly?)
  - Math questions (verify: did you solve it?)
  - Formatting-specifics (verify: did output match format requirements?)

Parameter Efficient Fine-Tuning

Low-Rank Adaptation (LoRA)

- Hypothesis: The change in weights during model adaptation has a low "***intrinsic rank***."

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

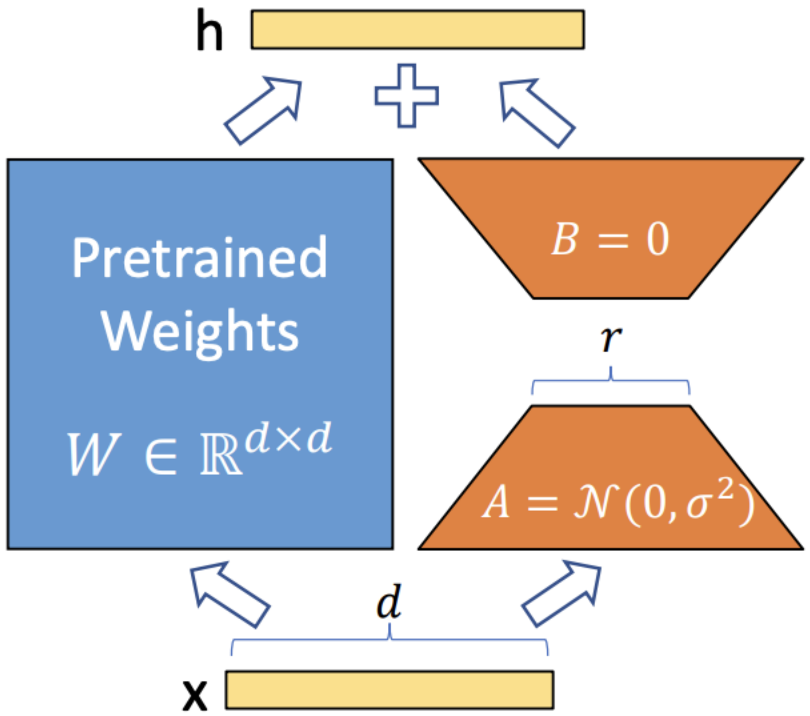
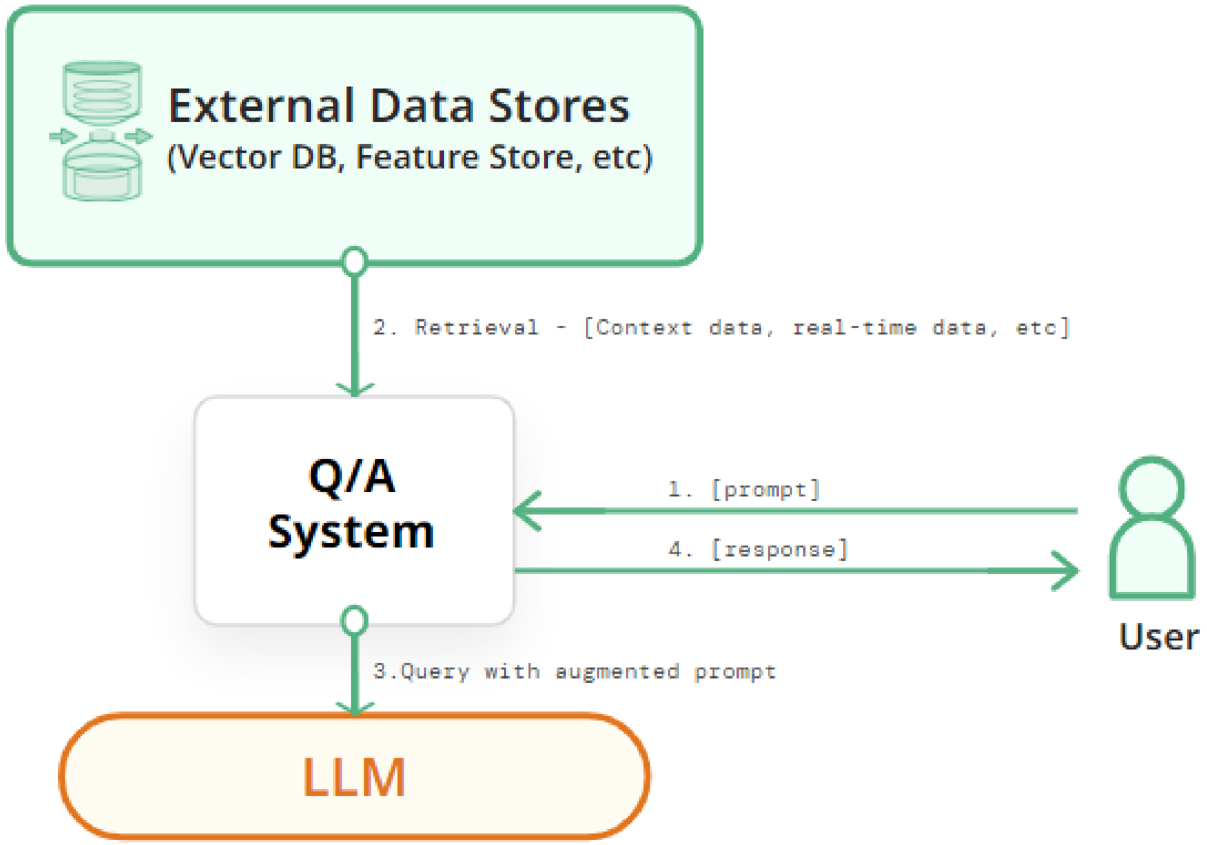


Figure 1: Our reparametrization. We only train A and B.

Retrieval-Augment Generation

- Resource access enables personalization



https://www.hopsworks.ai/dictionary/retrieval-augmented-generation-llm

Prompting

Few-Shot / Zero-shot learning

One key emergent ability in GPT-2 is **zero-shot learning**: the ability to do many tasks with **no examples,** and **no gradient updates,** by simply:

- Specifying the right sequence prediction problem (e.g. question answering):

  `Passage: Tom Brady... Q: Where was Tom Brady born? A: ...`

- Comparing probabilities of sequences (e.g. Winograd Schema Challenge [Levesque, 2011]):

  `The cat couldn't fit into the hat because it was too big.`
  Does `it = the cat` or `the hat?`

  ≡ Is `P(...because` **`the cat`** `was too big) >=`
  `P(...because` **`the hat`** `was too big)?`

[Radford et al., 2019]

Few-Shot / Zero-shot learning

GPT-2 beats SoTA on language modeling benchmarks with no task-specific fine-tuning

Context: "Why?" "I would have thought you'd find him rather dry," she said. "I don't know about that," said Gabriel. "He was a great craftsman," said Heather. "That he was," said Flannery.
Target sentence: "And Polish, to boot," said ______.
Target word: Gabriel

LAMBADA (language modeling w/ long discourse dependencies) [Paperno et al., 2016]

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**

[Radford et al., 2019]

Few-Shot / Zero-shot learning

You can get interesting zero-shot behavior if you're creative enough with how you specify your task!

Summarization on CNN/DailyMail dataset [See et al., 2017]:

```
SAN FRANCISCO,
California (CNN) --
A magnitude 4.2
earthquake shook
the San Francisco
...
overturn unstable
objects. TL;DR:
```

"Too Long, Didn't Read"

"Prompting"?

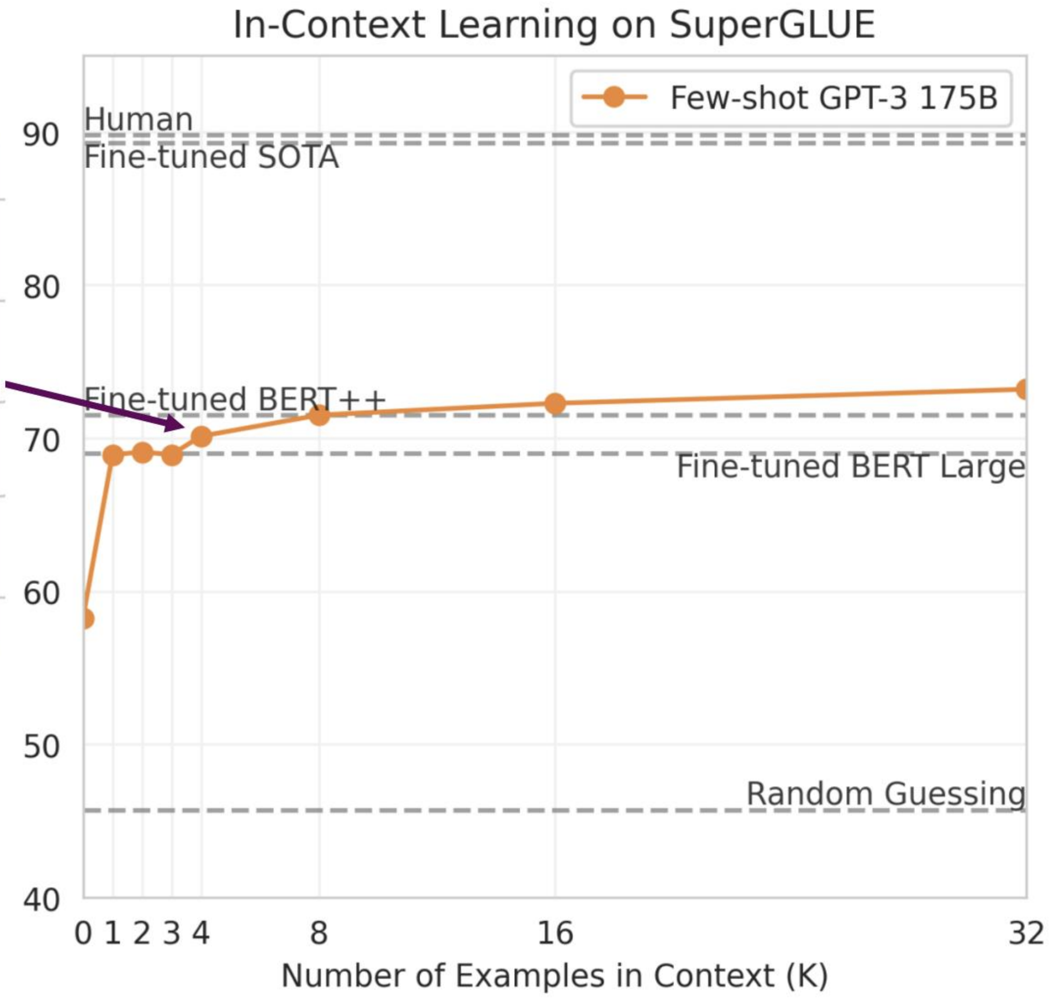
		ROUGE		
		R-1	R-2	R-L
2018 SoTA	Bottom-Up Sum	**41.22**	**18.68**	**38.34**
	Lede-3	40.38	17.66	36.62
Supervised (287K)	Seq2Seq + Attn	31.33	11.81	28.83
	GPT-2 TL;DR:	29.34	8.27	26.58
Select from article	Random-3	28.78	8.63	25.52

[Radford et al., 2019]

“In-Context Learning”

Few-shot

```
1  Translate English to French:
2  sea otter => loutre de mer
3  peppermint => menthe poivrée
4  plush girafe => girafe peluche
5  cheese =>
```



[Brown et al., 2020]

Open Problems

Alignment: What did the model learn to optimize?

- Connect probabilistic objectives to value-based objectives
- Outer vs inner alignment:
  - **Outer alignment:** Is the loss function we train on actually aligned with human goals?
  - **Inner alignment:** Given that loss, does the trained model's internal representation faithfully implement that goal, even off-distribution?

More open problems

- RL (how to effectively train at scale with distant reward signals)
- Scaling verifiable rewards
- Combining LLMs with symbolic reasoning
- Combining LLMs with graphical models
- Continual learning
- Formal theory of alignment.
- Post-hoc interpretability of large models.
- Ante-hoc interpretable-by-design large models.
- Ethical and technical fusion: aligning not just models, but the human-model system.

https://www.youtube.com/watch?v=aR20FWCCjAs

Some open problems from Ilya

- Models show impressive eval performance but lack real-world economic impact and exhibit jaggedness, like repeating bugs in coding tasks.
- Human emotions serve as robust value functions? Current AI lacks similar mechanisms.
- Pre-training scales uniformly but hits data walls; RL consumes more compute but needs better efficiency via value functions.
- Humans generalize better than models with fewer samples and unsupervised learning.
- Alignment involves designing AI to care for sentient life, including AIs, for broader empathy over human-centric values?

Some open problems from Ilya

- Models show impressive eval performance but lack real-world economic impact and exhibit jaggedness, like repeating bugs in coding tasks.
- H
  si
- Pr
  compute but needs better efficiency via value functions.

You all now have the tools and vocabulary to discuss SOTA research that is worth billions of $.

- Humans generalize better than models with fewer samples and unsupervised learning.
- Alignment involves designing AI to care for sentient life, including AIs, for broader empathy over human-centric values?

Questions?

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON