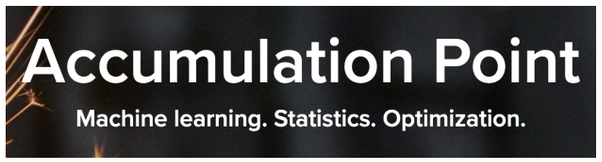
The Mathematical Engineering of Deep Learning

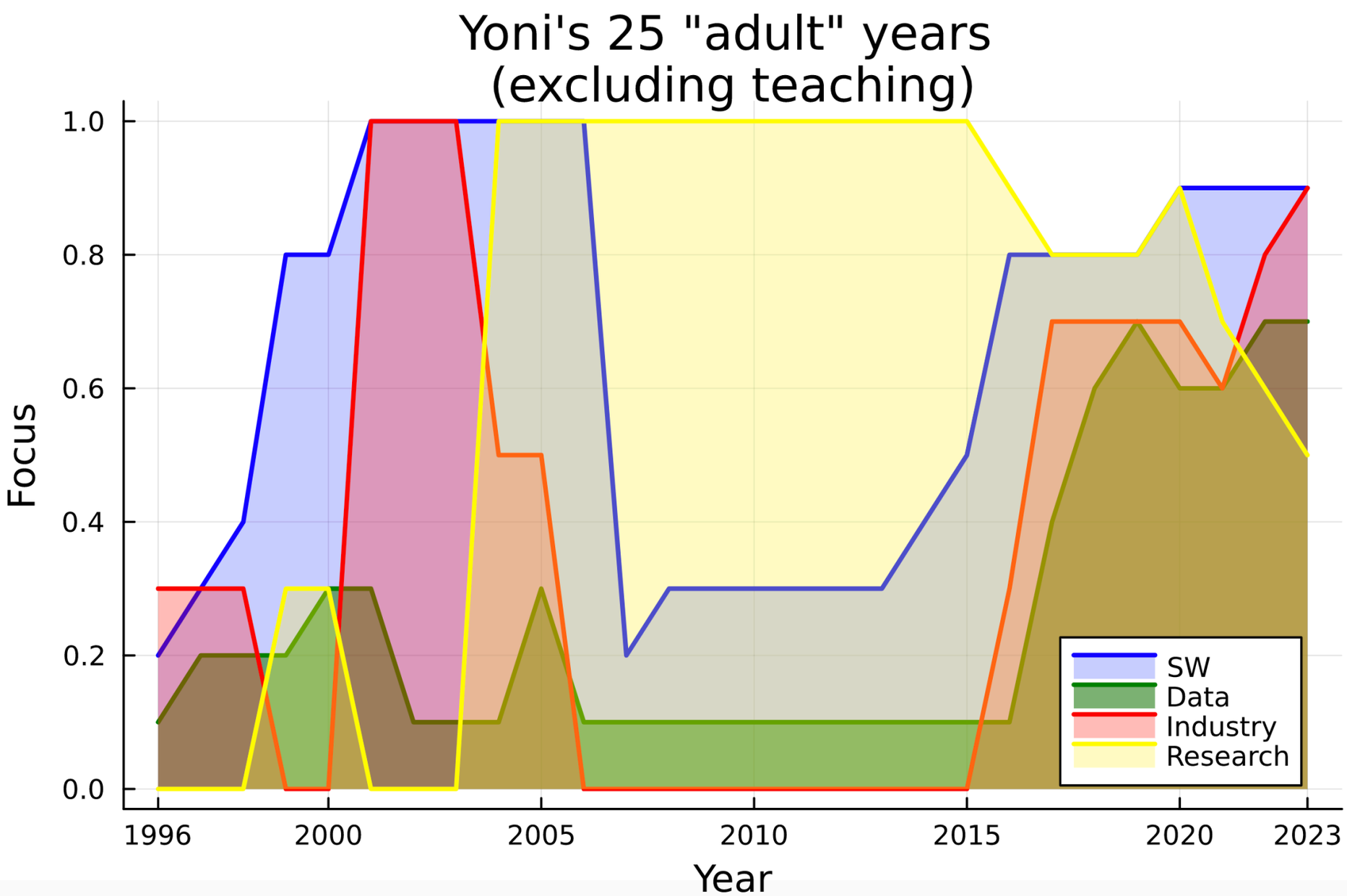
Presented at the HDR Student Research Symposium,
The School of Mathematics, Physics, and Computing,
The University of Southern Queensland

September 15, 2023

Yoni Nazarathy



About the speaker…

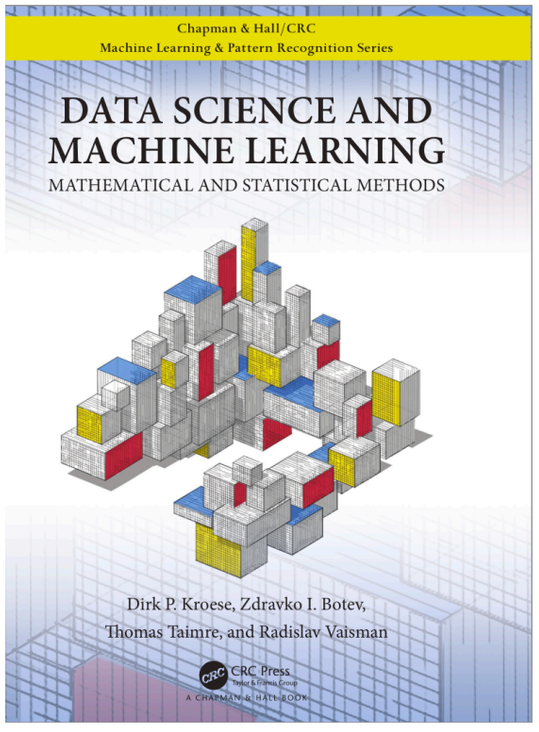


Abstract: So much has happened in the space of AI that it is hard to keep up. A bit more than a decade ago deep learning moved to the forefront of machine learning and AI, and since then advancements are arriving at phenomenal pace. The latest massive leap is in large language models (e.g. ChatGPT), but just before that (in 2022) diffusion models for image generation were a hit. Before that, success with reinforcement learning (e.g. the game of Go) was a highlight and all throughout the last decade, image analysis developed to incredible strengths. Within the course of these developments one important milestone was the development of Generative Adversarial Networks (GANs). These were the first models that could effectively create “real looking” but “fake” images at scale.

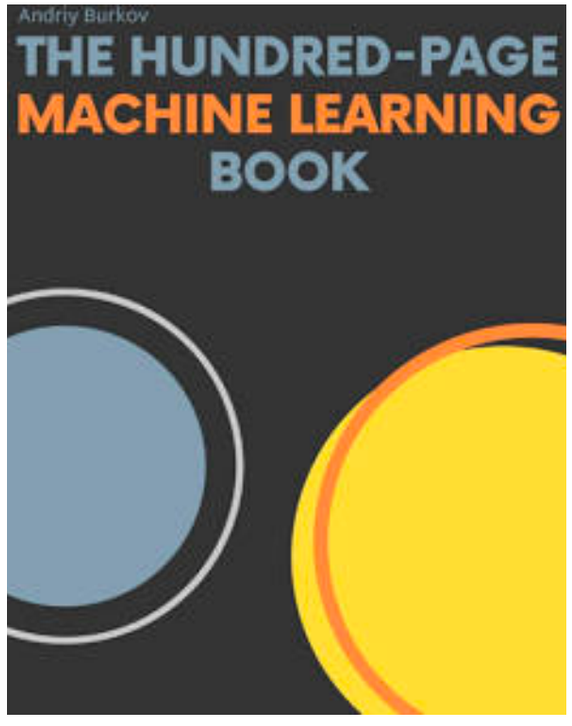
In this talk we briefly outline the recent history of neural networks and AI, focusing on a “Mathematical Engineering” perspective where formulas, mathematical transformations, and algorithms are key. With this approach software and actual implementation is only peripheral. We then focus on GANs to explore how they work, and why there may be a future to this paradigm in the years to come.

Some resources

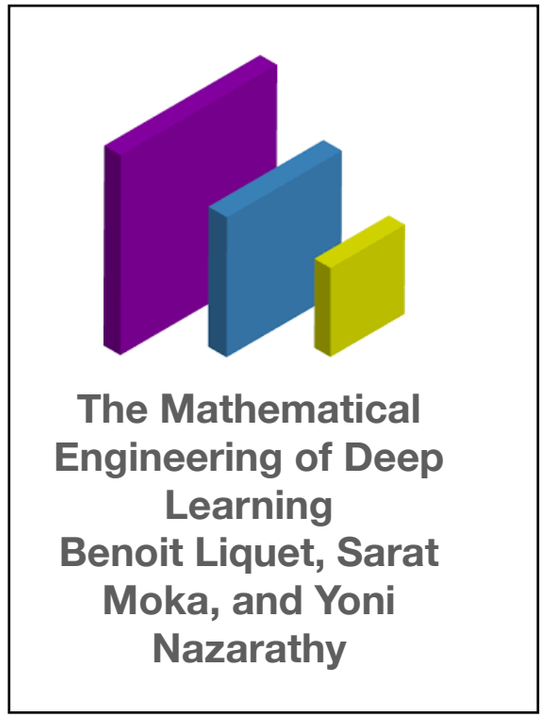
https://people.smp.uq.edu.au/DirkKroese/DSML/



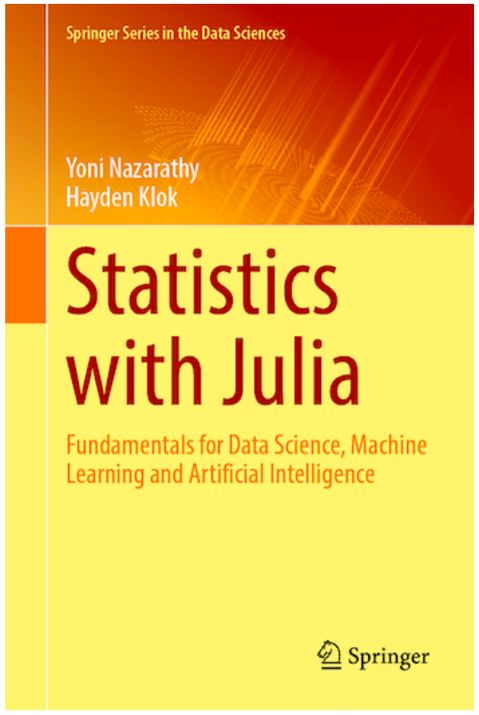
http://themlbook.com/



https://deeplearningmath.org/



https://statisticswithjulia.org/



Data Science

Computer Engineering

Artificial Intelligence

Machine Learning

Deep Learning

Computer Science

Signal (and image) Processing

Statistics

Neuroscience

Statistical Learning

Data Mining

Image Data

Text Data

Sound Data

Genomic Data

CSV Tabular Data

Social Networks Data

.....

Training

Deep Learning

Production

Agriculture

Robotics

Education

Medicine

Security and Safety

Business Analytics

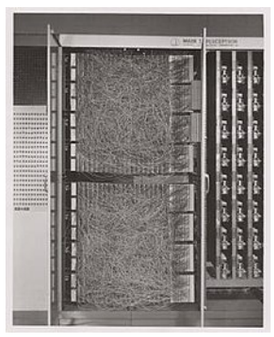
Social Networks

.....

A bit of history

Timeline of weak (narrow) AI

Weak AI
`Explosion' and Deep Learning



~2010 Machine Learning
Renaissance begins

~2000 Revival of machine learning

1980's Reinforcement Learning

1970 Reverse Mode Automatic Differentiation

1960's Neural Networks

1950 Turing Test Proposed

Timeline of strong AI (Artificial General Intelligence)

???
ChatGPT

2010+ Weak AI Hype…

1970’s-80’s Expert Systems

1970's Thoughts and definitions about AGI

1950 Turing Test Proposed

Timeline of Deep Learning (prehistory)

"Deep Learning" is a thing!
Deep Learning = AI?

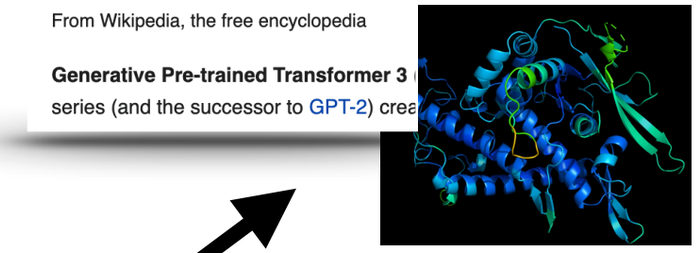
2012 (**AlexNet**): ImageNet Classification with Deep Convolutional Neural Networks, by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton.

1998: (**Convolutional nets**): Gradient based learning applied to document recognition, by Yann Lecun, Leon Bottou, Yoshua Bengio and Patrick Haffne.

1982 (**Hopfield nets**): Neural networks and physical systems with emergent collective computational abilities, by John Hopfield.

1958 (**Perceptron**): The perceptron: a probabilistic model for information storage and organization in the brain, by Frank Rosenblatt.

Current Deep Learning



2021/2 (**Diffusion models explosion**)

2017 (**Transformers**): Attention is all you need, by Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser and Illia Polosukhin.

2016 (**Deep RL vs. Go**): Mastering the game of Go with deep neural networks and tree search, by David Silver, Aja Huang, Chris Maddison and others.

2015 (**ResNet**): Deep residual learning for image recognition, by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun.

2015 (**Inception**): Going deeper with convolutions, by Christian Szegedy et. al.

2015 (**VGG**): Very deep convolutional networks for large-scale image recognition, by Karen Simonyan and Andrew Zisserman.

2014: (**GAN**): Generative adversarial nets, by Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio.

2014: (**ADAM**): Adam: A method for stochastic optimization, by Diederik Kingma and Jimmy Ba.

2013: (**Deep RL**): Playing Atari with Deep Reinforcement Learning by Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller.

2013: (**Inner layers**): Visualizing and Understanding Convolutional Networks, by Matthew Zeiler and Rob Fergus

2012 (**AlexNet**): ImageNet Classification with Deep Convolutional Neural Networks, by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton.

Shop talk…

Dropout

Learning rate

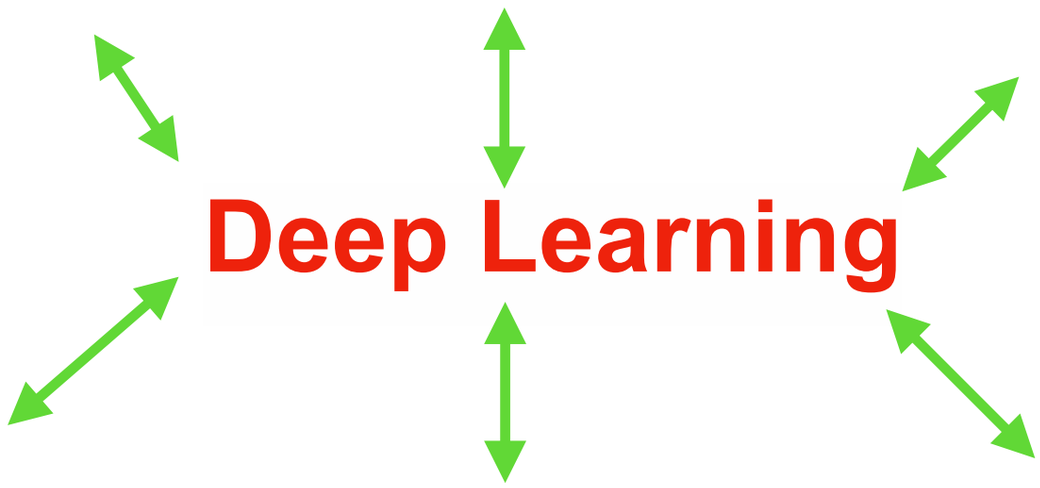
Minibatch

Cross entropy loss

Batch normalization

L2 loss

Transformers



Transfer learning

Early stopping

Channels

Hyperparameters

Residual connections

Activation functions

Fully connected layers

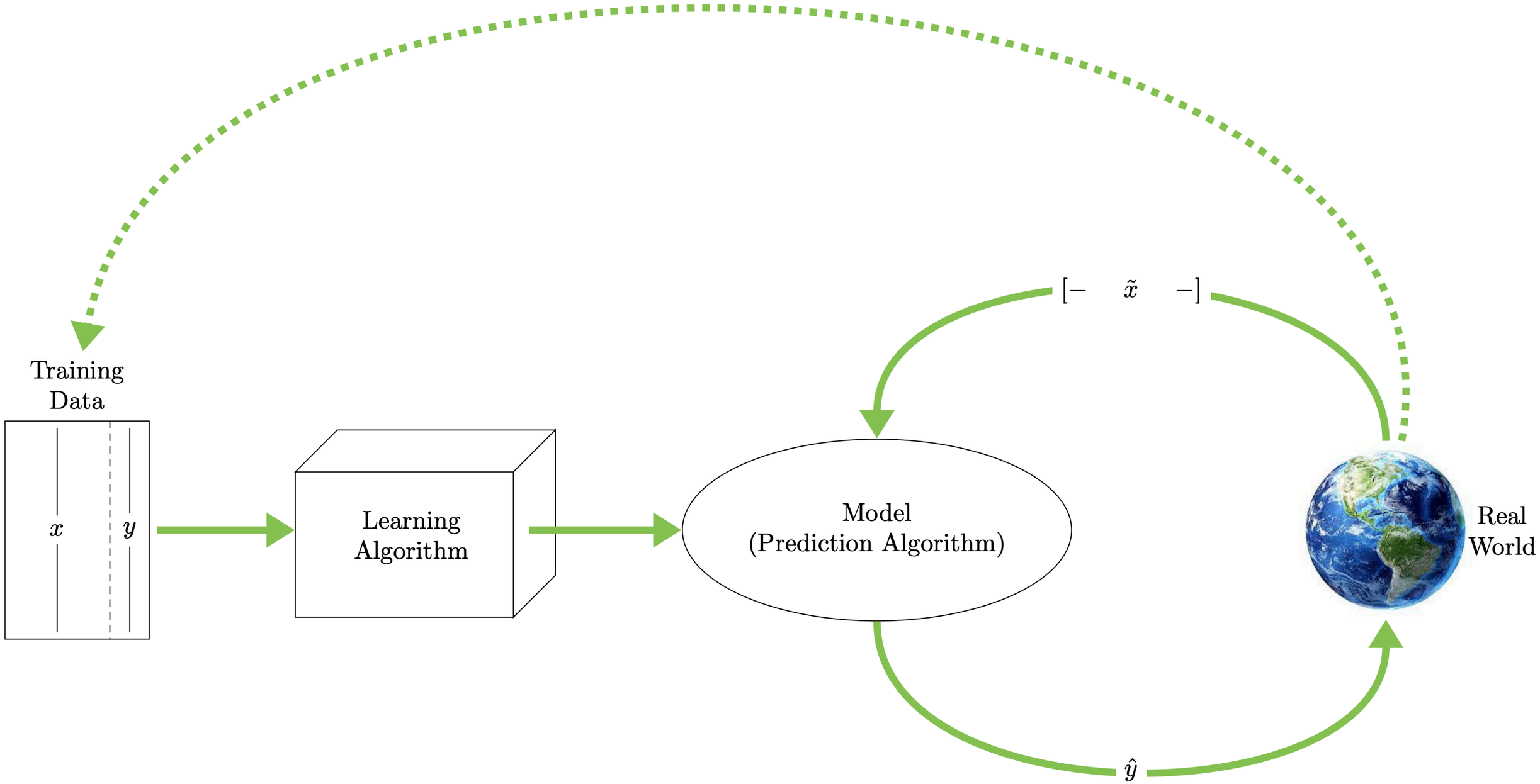
Max pooling

Convolutional layers

Main activities of machine learning

- Supervised learning (Y = labels, X = features) → Regression / Classification
- Unsupervised learning (only X)
- Semi-supervised learning (hybrid)
- Reinforcement Learning (a time component)
- Generative modeling (generating new X’s)
- Dealing with sequence data

Our focus today is supervised Learning



How does it work?

(oversimplified linear model example below)

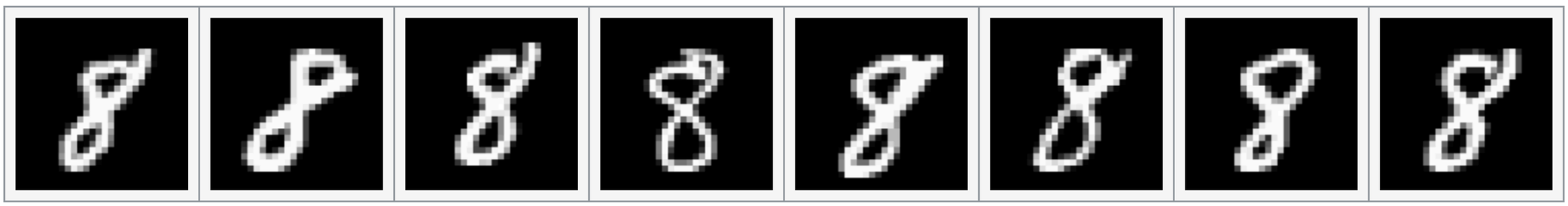
A linear (binary) classifier

Positive (+1)

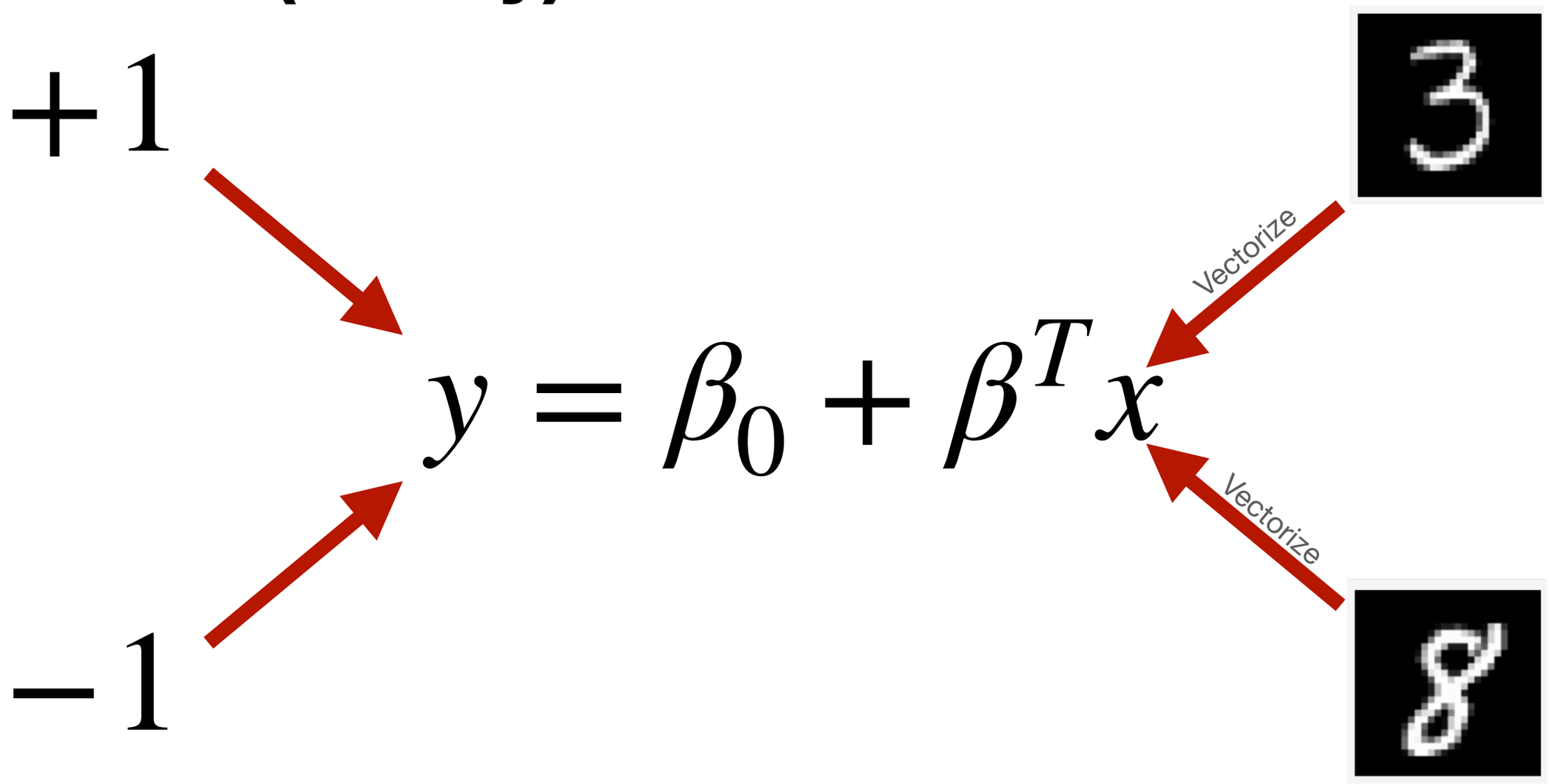
```
imgs[labels .== 3][1:8]
```

Negative (-1)

```
imgs[labels .== 8][1:8]
```



A linear (binary) classifier



Minimize: $\text{Loss}(x, y) = \sum_{i=1}^{m} (y_i - \beta_0 - \beta^T x_i)^2$

Classifier: $\hat{f}(x) = \text{sign}(\hat{\beta}_0 + \hat{\beta}^T x)$

A linear (binary) classifier

Minimize: $\text{Loss}(x, y) = \sum_{i=1}^{m} (y_i - \beta_0 - \beta^T x_i)^2 = ||y - A\beta||^2$

Sometimes

Option 1: $\hat{\beta} = A^{\dagger} y$ $\qquad A^{\dagger} = (A^T A)^{-1} A^T$

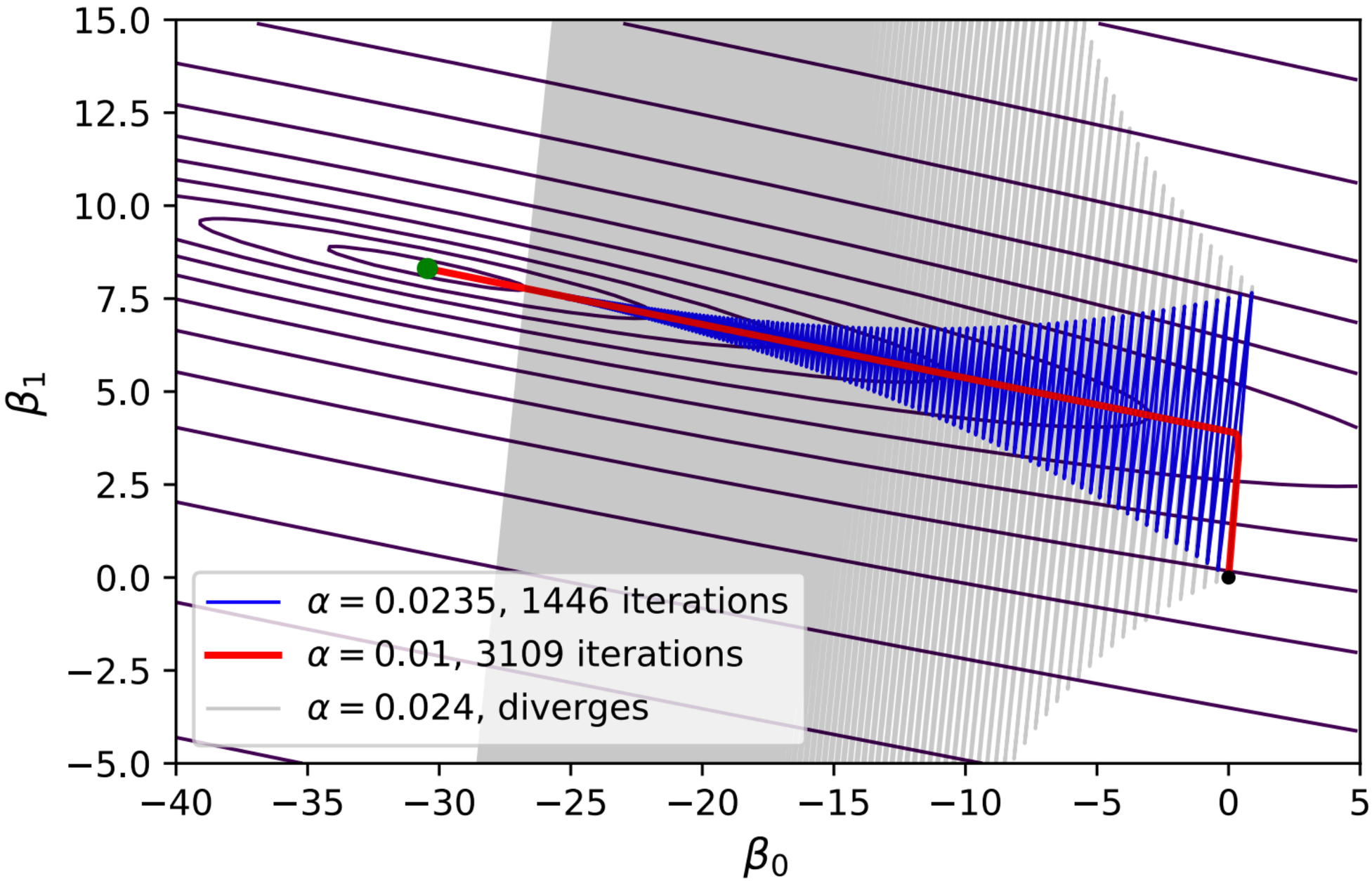
Option 2: $\hat{\beta}(0), \hat{\beta}(1), \hat{\beta}(2), \hat{\beta}(3), \ldots$ With (some form of) gradient descent

$$\hat{\beta}(t+1) = \hat{\beta}(t) - \eta \nabla L\left(\hat{\beta}(t)\right)$$

$$\nabla L(\beta) = 2A^T(A\beta - y)$$

$\theta \leftarrow \theta_{\text{init}}$
repeat
  Compute the gradient $\nabla C(\theta)$
  Determine the *learning rate* α
  $\theta \leftarrow \theta - \alpha \nabla C(\theta)$
until θ satisfies a *termination condition*
return θ

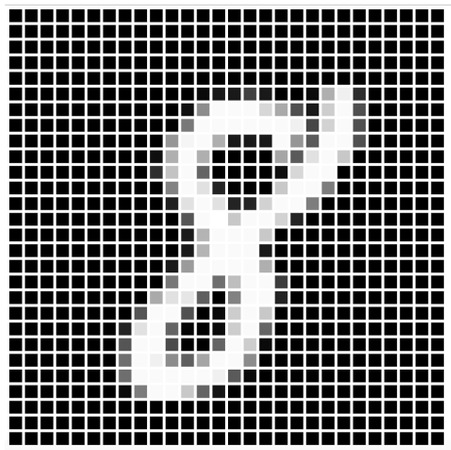
Gradient descent



$$\alpha < \frac{n}{\lambda_{\max}}$$

Multi-class: One vs. rest

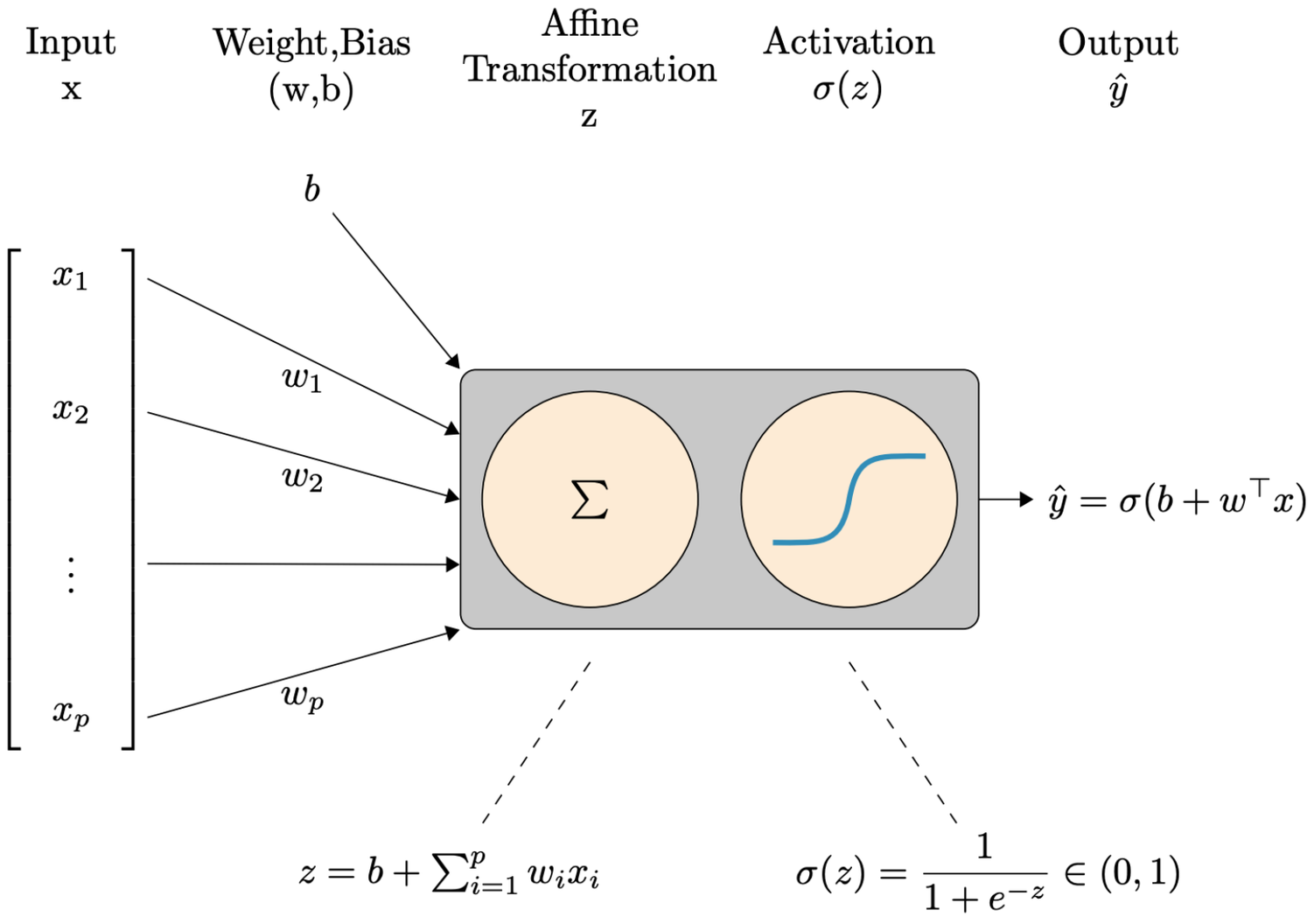
```
using Flux.Data.MNIST, PyPlot, LinearAlgebra
using Flux: onehotbatch

imgs   = MNIST.images()
labels = MNIST.labels()
nTrain = length(imgs)

trainData = vcat([hcat(float.(imgs[i])...) for i in 1:nTrain]...);
trainLabels = labels[1:nTrain];

testImgs = MNIST.images(:test)
testLabels = MNIST.labels(:test)
nTest = length(testImgs)

testData = vcat([hcat(float.(testImgs[i])...) for i in 1:nTest]...);

A = [ones(nTrain) trainData];
Adag = pinv(A);
tfPM(x) = x ? +1 : -1
yDat(k) = tfPM.(onehotbatch(trainLabels,0:9)'[:,k+1])
bets = [Adag*yDat(k) for k in 0:9];

classify(input) = findmax([([1 ; input])'*bets[k] for k in 1:10])[2]-1

predictions = [classify(testData[k,:]) for k in 1:nTest]
confusionMatrix = [sum((predictions .== i) .& (testLabels .== j))
                   for i in 0:9, j in 0:9]
accuracy = 100*sum(diag(confusionMatrix))/nTest

println("Accuracy: ", accuracy,"%")
confusionMatrix
```



Basic statistics
(least squares/regression)

```
Accuracy: 86.03%

10×10 Array{Int64,2}:
 944     0    18     4     0    23    18     5    14    15
   0  1107    54    17    22    18    10    40    46    11
   1     2   813    23     6     3     9    16    11     2
   2     2    26   880     1    72     0     6    30    17
   2     3    15     5   881    24    22    26    27    80
   7     1     0    17     5   659    17     0    40     1
  14     5    42     9    10    23   875     1    15     1
   2     1    22    21     2    14     0   884    12    77
   7    14    37    22    11    39     7     0   759     4
   1     0     5    12    44    17     0    50    20   801
```

Logistic Regression



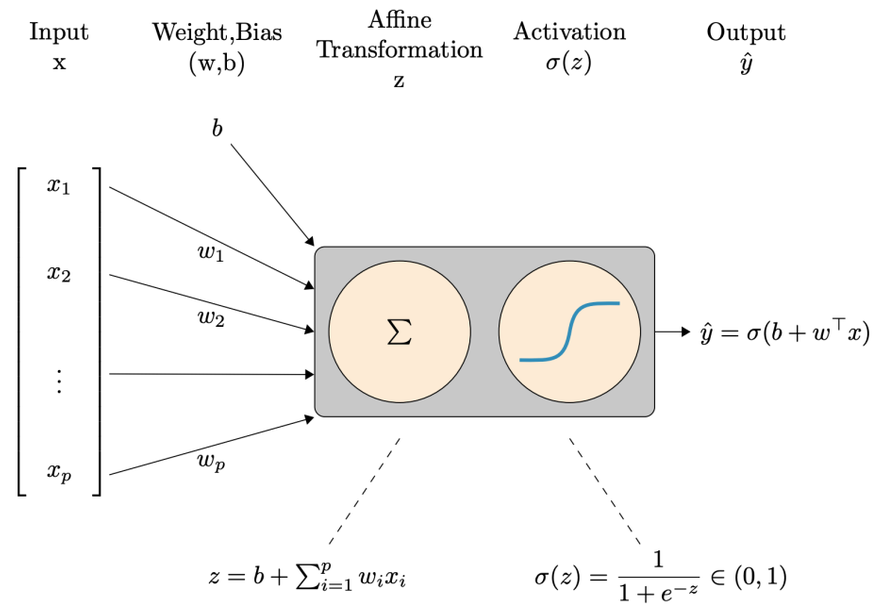
$$\hat{y} = \underbrace{\sigma\left(\overbrace{b + w^\top x}^{z}\right)}_{a}$$

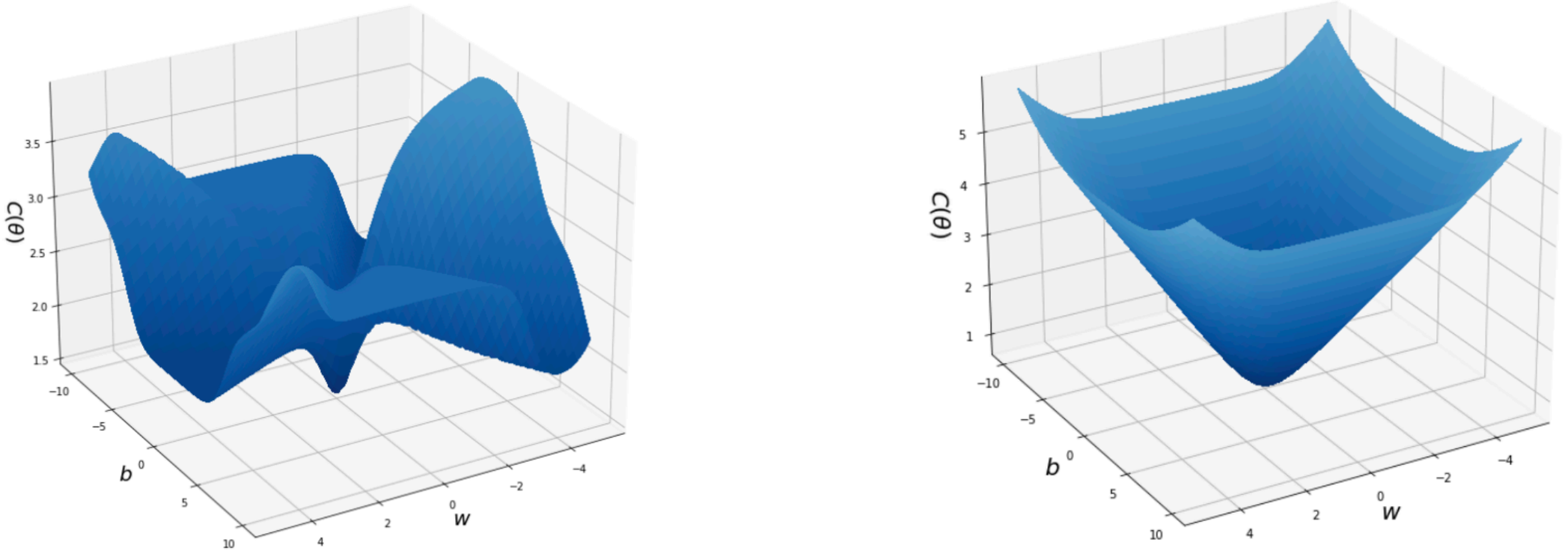
Loss for Logistic Regression (MLE motivated)

$$C(\theta\,;\mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n}\left[-\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + (1-y^{(i)})\log\left(1-\hat{y}^{(i)}\right)\right)\right]$$

$$\nabla C(\theta) = \frac{1}{n}\sum_{i=1}^{n}\nabla C_i(\theta)$$

$$\nabla C_i\,(\theta) = \left(\sigma_{\text{Sig}}(w^\top x^{(i)} + b) - y^{(i)}\right)\begin{bmatrix}1\\ x^{(i)}\end{bmatrix}$$





(a) **(b)**

Figure 3.4.: The loss landscape of logistic regression for a synthetic dataset. (a) Using the squared loss $C_i(\theta) = (y^{(i)} - \hat{y}^{(i)})^2$. (b) Using the binary cross entropy loss $C_i(\theta) = \text{CE}_{\text{binary}}(y^{(i)}, \hat{y}^{(i)})$.

Logistic Regression with Multiple Classes

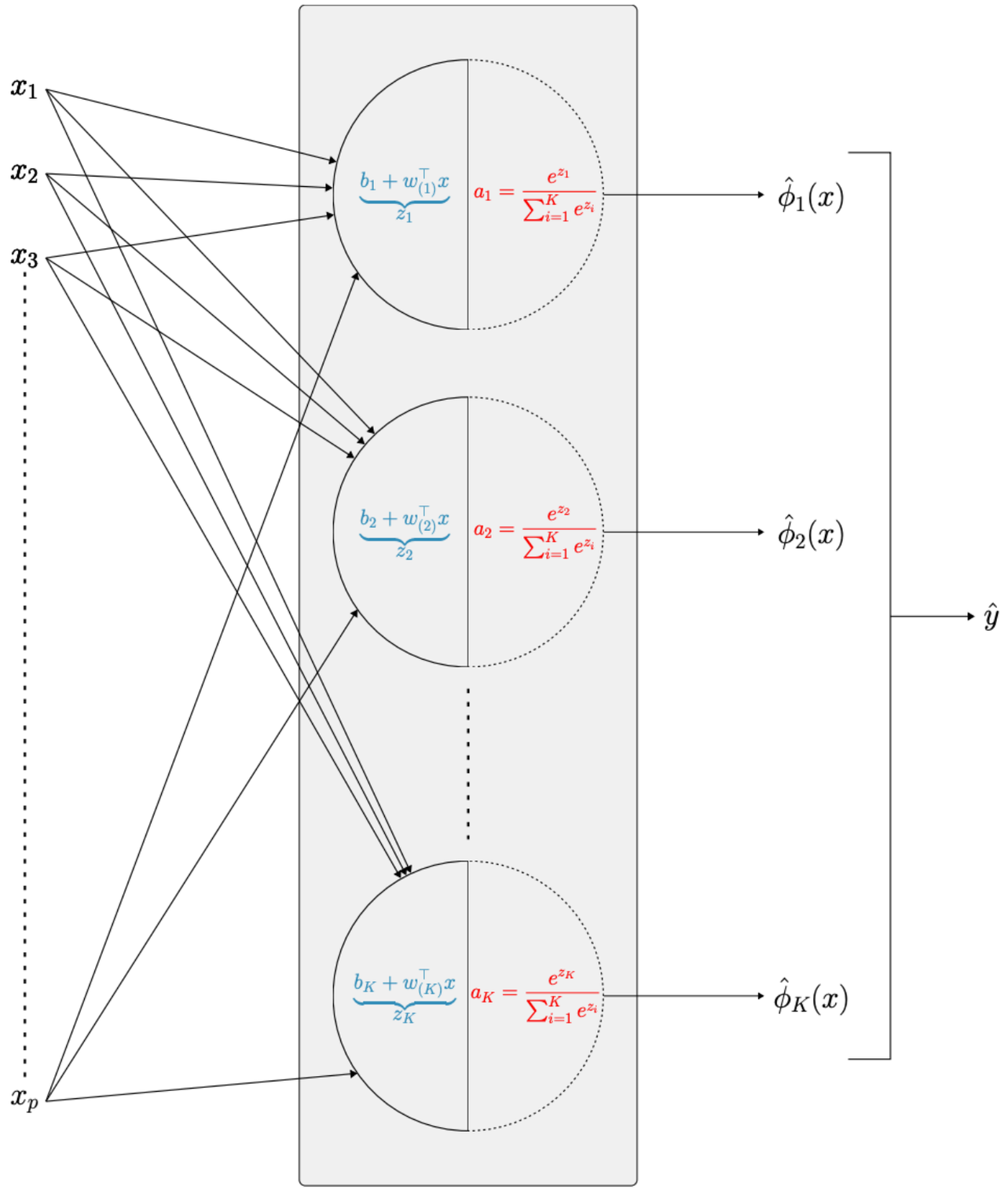
$$\hat{y} = \underbrace{S_{\text{softmax}}\big(\overbrace{b + Wx}^{z \in \mathbb{R}^K}\big)}_{a \in \mathbb{R}^K}$$

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix}, \quad \text{and} \quad W = \begin{bmatrix} — & w_{(1)}^\top & — \\ — & w_{(2)}^\top & — \\ & \vdots & \\ — & w_{(K)}^\top & — \end{bmatrix}$$

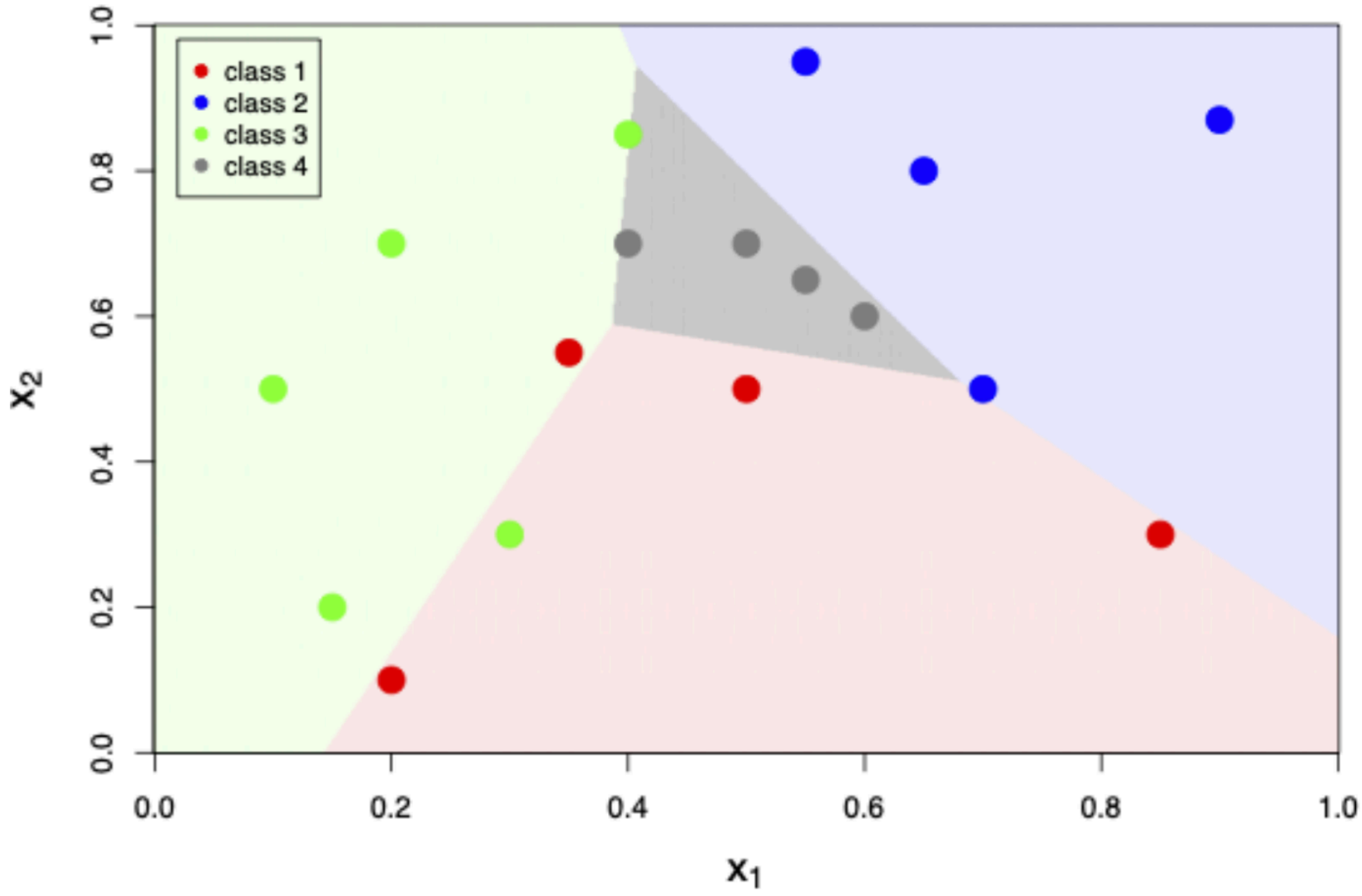
$$S_{\text{softmax}}(z) = \frac{1}{\sum_{i=1}^K e^{z_i}} \begin{bmatrix} e^{z_1} \\ \vdots \\ e^{z_K} \end{bmatrix}$$

$$\text{CE}_{\text{categorical}}(y, \hat{y}) = -\sum_{k=1}^{K} \mathbf{1}\{y = k\} \log \hat{y}_k = -\log \hat{y}_y$$

$$C_i(\theta) = \text{CE}_{\text{categorical}}\big(y^{(i)}, S_{\text{softmax}}(b + Wx^{(i)})\big)$$



Linear Decision Boundaries



Going beyond linear decision boundaries

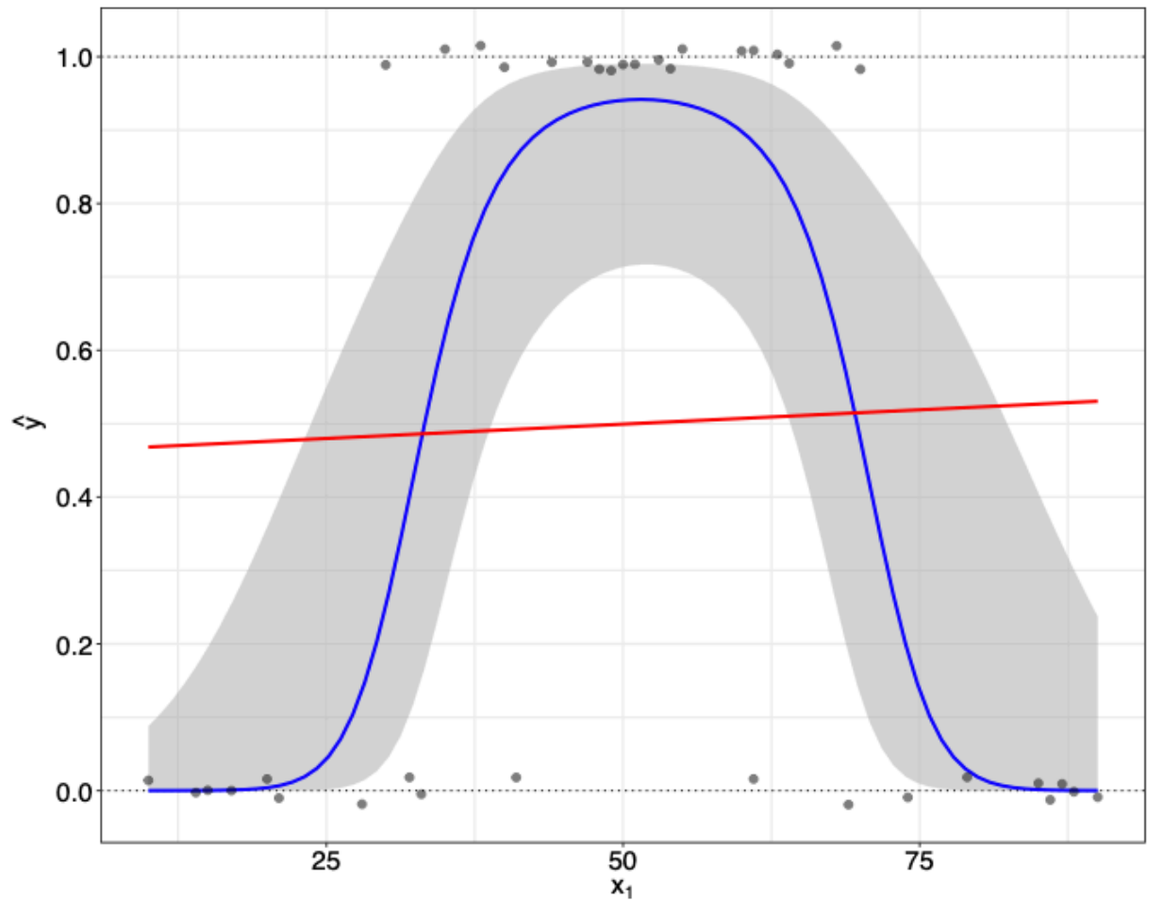
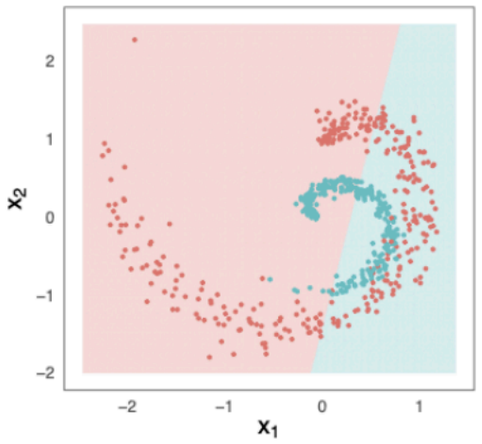
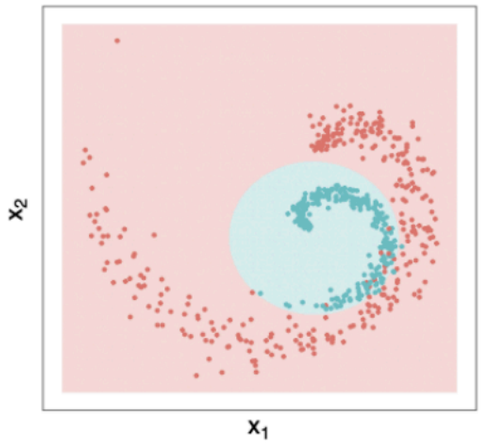


Figure 3.8.: Logistic regression fit to a feature engineered model with the feature $x_2 = x^2$ and a confidence band. The response curve is plotted in red together with confidence bounds. The blue curve is a sigmoidal function fit to the single feature x.

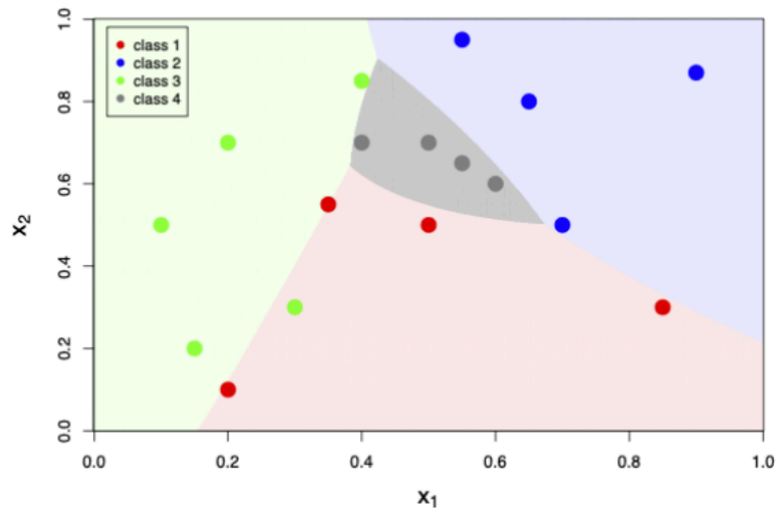
Adding higher monomial terms

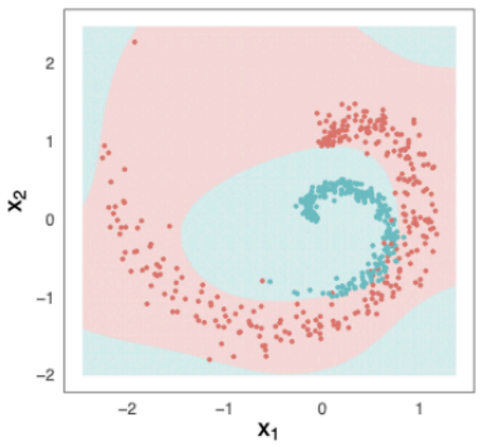


(a)

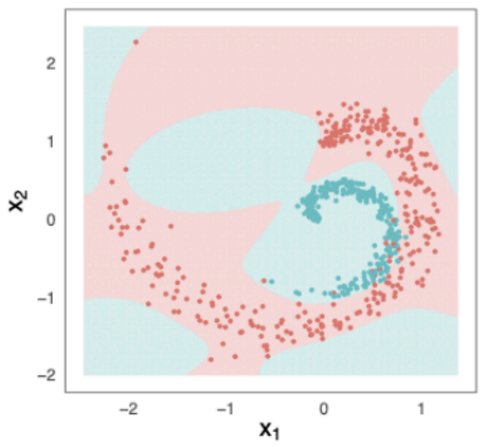


(b)

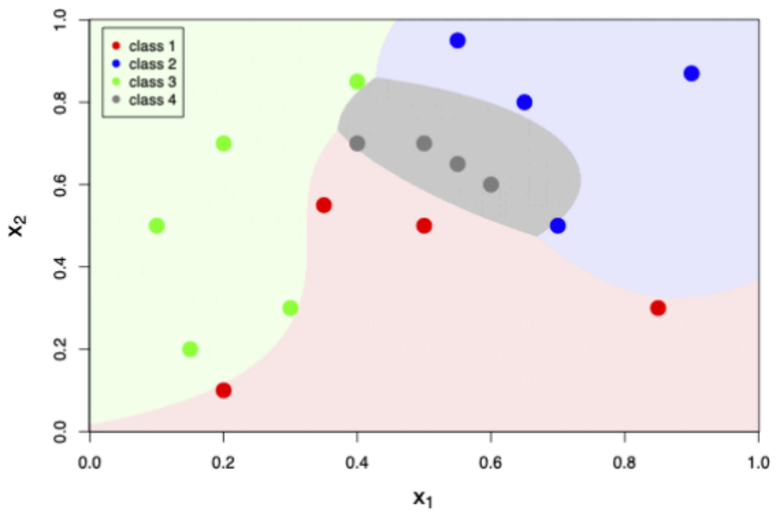




(c)



(d)



A digression to optimization

Loss as a proxy for performance

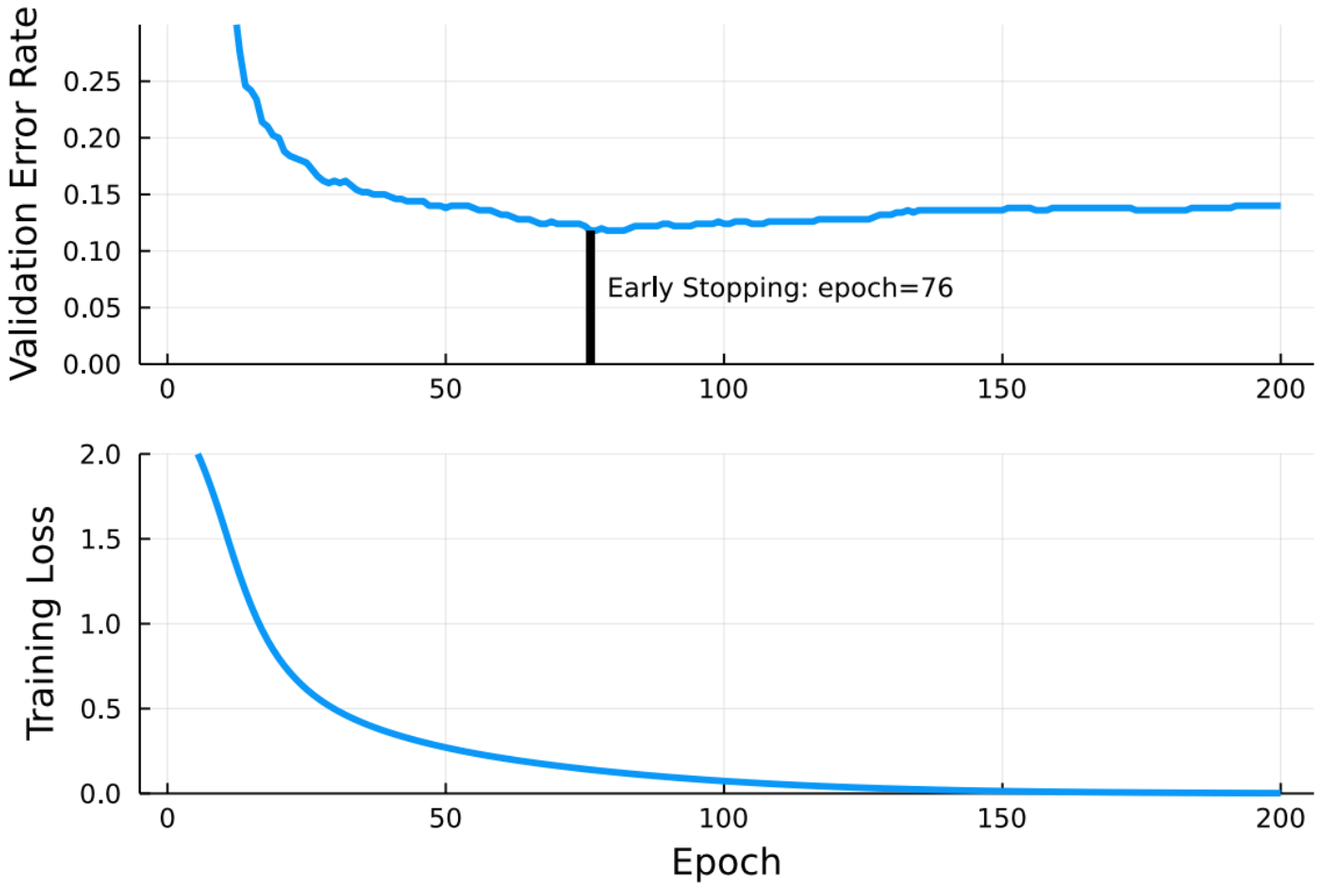
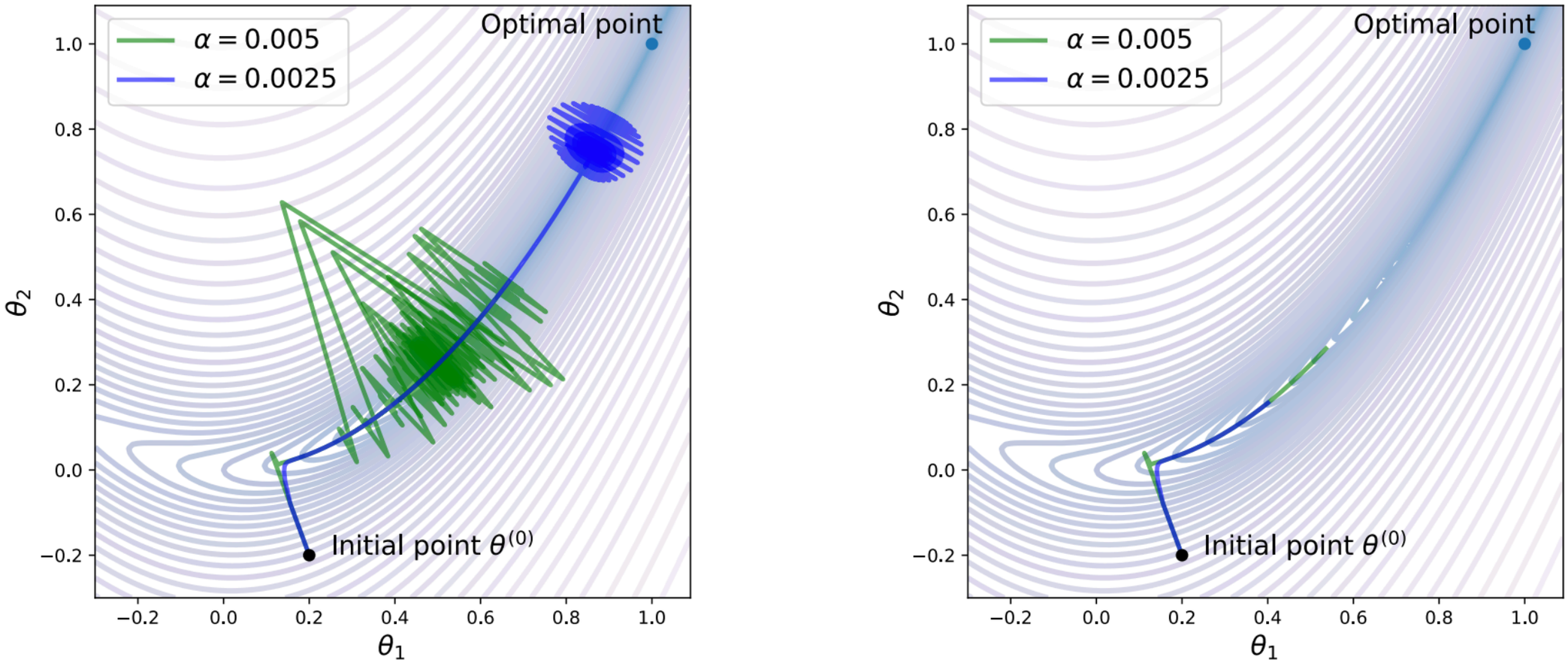


Figure 4.4.: Training on a small subset of the MNIST dataset. The validation error rate is tracked together with the training loss. The early stopping strategy uses the model from epoch 76 where the validation performance is best.

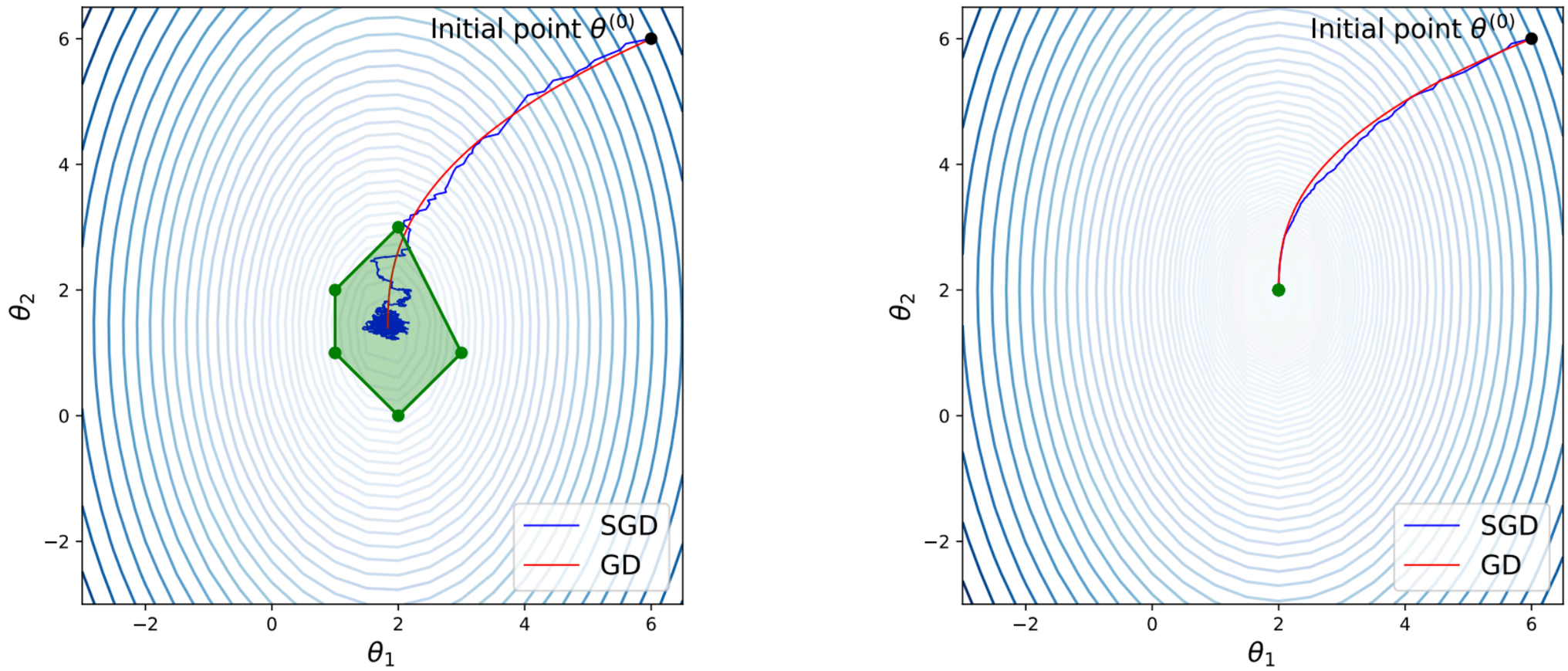
Making gradient descent work is hard



(a) **(b)**

Figure 4.2.: Attempting minimization of a Rosenbrock function with a minimum at $\theta = (1, 1)$ via basic gradient descent for two values of α with 10^5 iterations and $\theta_{\text{init}} = (0.2, -0.2)$. (a) Fixed learning rate: $\alpha^{(t)} = \alpha$. (b) Exponentially decaying learning rate: $\alpha^{(t)} = \alpha\, 0.99^t$.

Stochastic gradient descent



(a) **(b)**

Figure 4.3.: An hypothetical example where $C(\theta)$ is composed of individual $C_i(\theta)$ as in (4.12). Comparison of gradient descent (GD) and stochastic gradient descent (SGD). (a) The minimizers $u_1, \ldots, u_n$ of $C_i(\theta)$ are different. (b) The minimizers $u_1, \ldots, u_n$ are the same.

Momentum

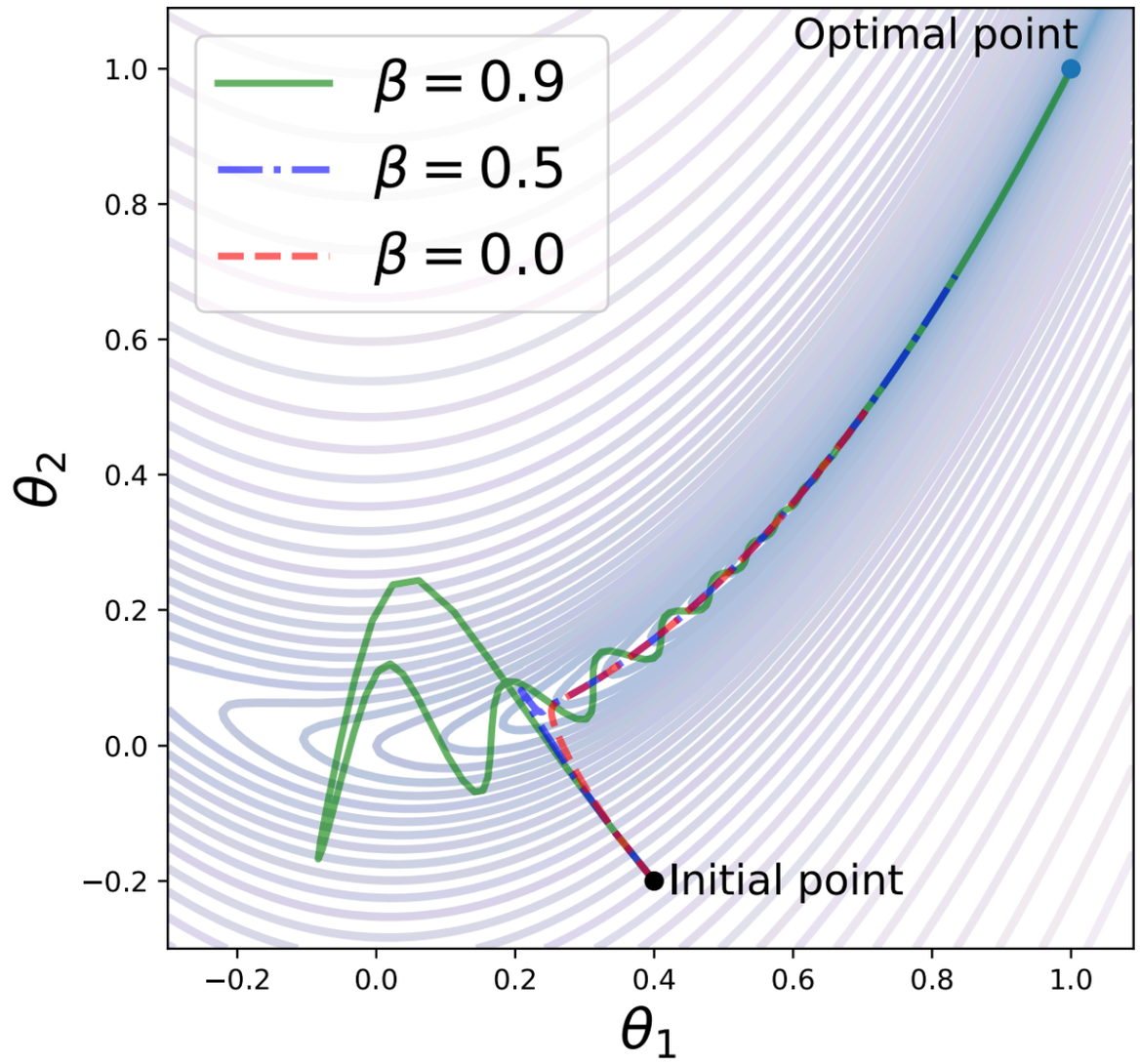


Figure 4.6.: Application of momentum to the Rosenbrock function (4.9) for three different values of β with learning rate $\alpha = 0.001/(1-\beta)$ and a fixed number of iterations. Note that $\beta = 0$ is basic gradient descent.

The most common algorithm: ADAM

Algorithm 4.2: ADAM Algorithm

Input: Dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$,
objective function $C(\cdot) = C(\cdot\,; \mathcal{D})$, and
initial parameter vector θ_{init}

Output: An *optimal* θ

$t \leftarrow 0$ (Initialize iteration counter)

$\theta \leftarrow \theta_{\text{init}}$, $v \leftarrow 0$, and $s \leftarrow 0$ (Initialize state vectors)

repeat

  $g \leftarrow \nabla C(\theta)$ (Compute gradient)

  $v \leftarrow \beta v + (1 - \beta)\, g$ (Momentum update)

  $s \leftarrow \gamma s + (1 - \gamma)\, (g \odot g)$ (Second moment update)

  $\hat{v} \leftarrow \dfrac{v}{1 - \beta^{t+1}}$ (Bias correction)

  $\hat{s} \leftarrow \dfrac{s}{1 - \gamma^{t+1}}$ (Bias correction)

  $\theta \leftarrow \theta - \alpha \dfrac{1}{\sqrt{\hat{s}} + \varepsilon} \hat{v}$ (Update parameters)

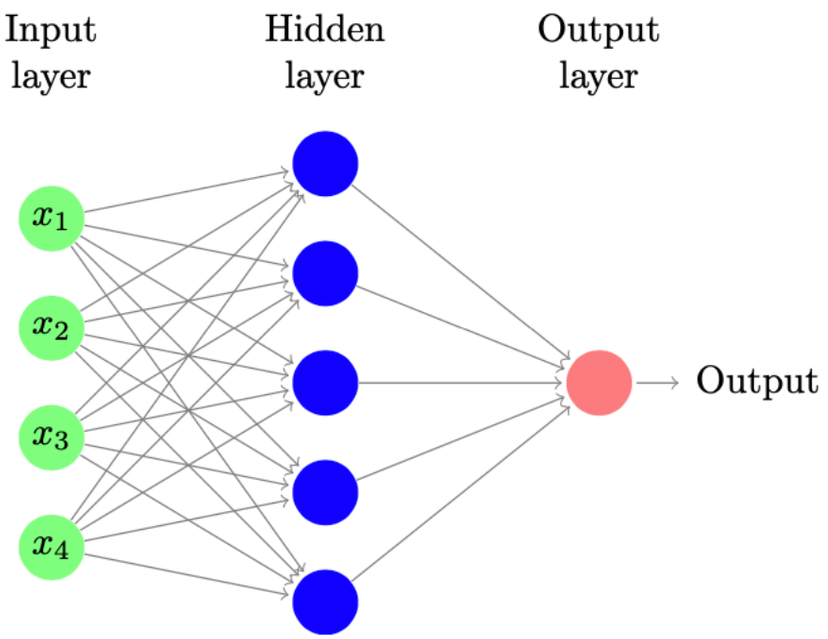
  $t \leftarrow t + 1$

until *termination condition* is satisfied

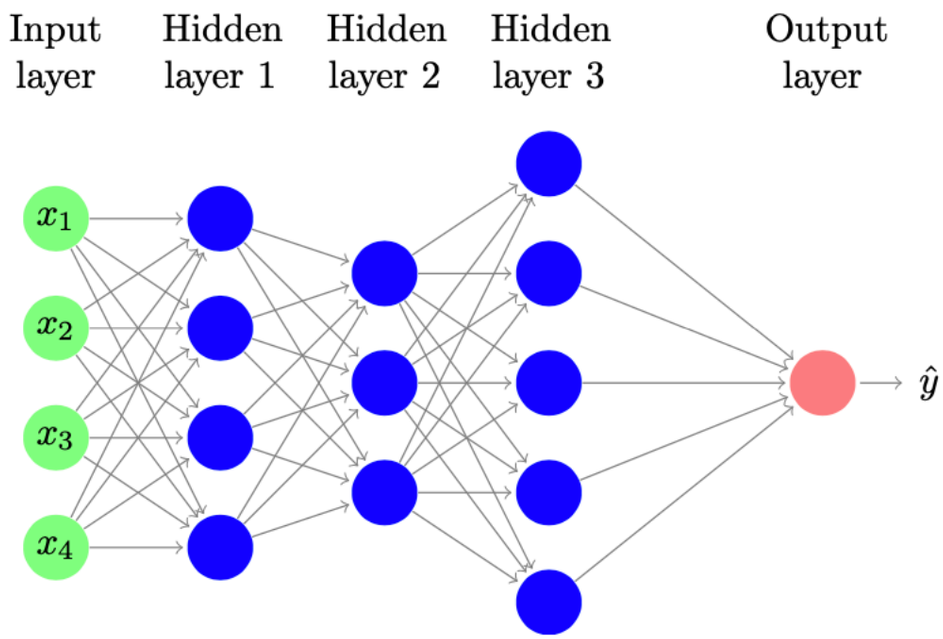
return θ

Feedforward Neural Networks

Adding activations and layers



(a) One-layer hidden Network



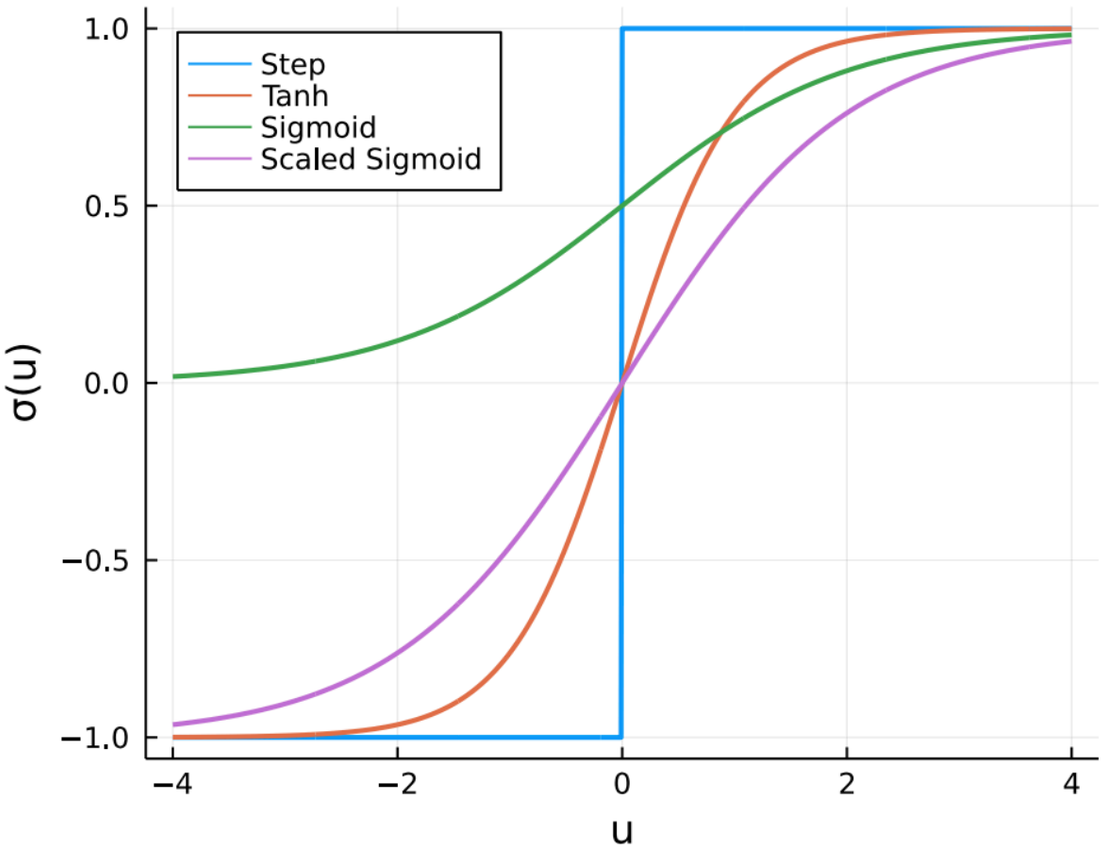
(b) Deep Neural Networks

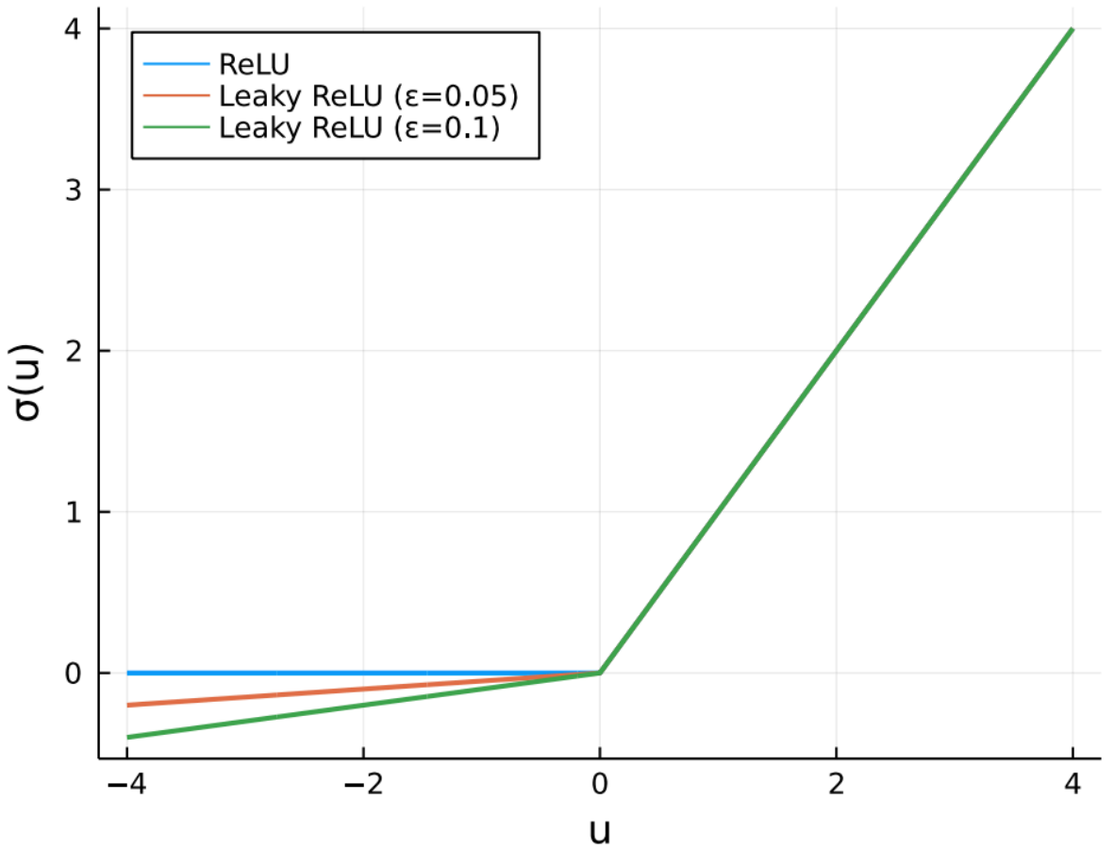
$$f_\theta(x) = f^{[L]}_{\theta^{[L]}}(f^{[L-1]}_{\theta^{[L-1]}}(\ldots(f^{[1]}_{\theta^{[1]}}(x))\ldots))$$

$$a^{[\ell-1]} \xrightarrow{\text{Affine Transformation}} z^{[\ell]} := W^{[\ell]}a^{[\ell-1]} + b^{[\ell]} \xrightarrow{\text{Activation}} a^{[\ell]} := S^{[\ell]}(z^{[\ell]})$$

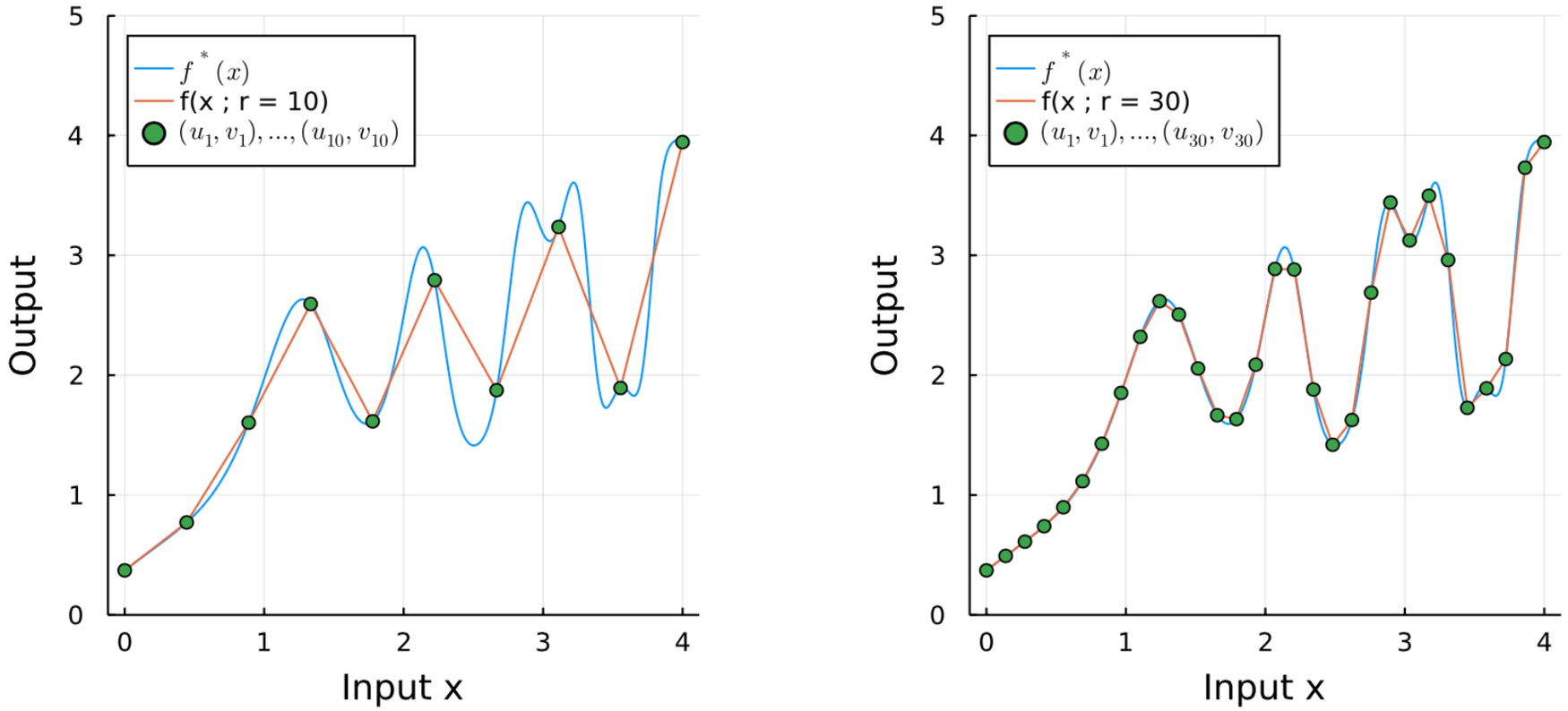
$$f^{(\ell)}_{\theta^{[\ell]}}$$

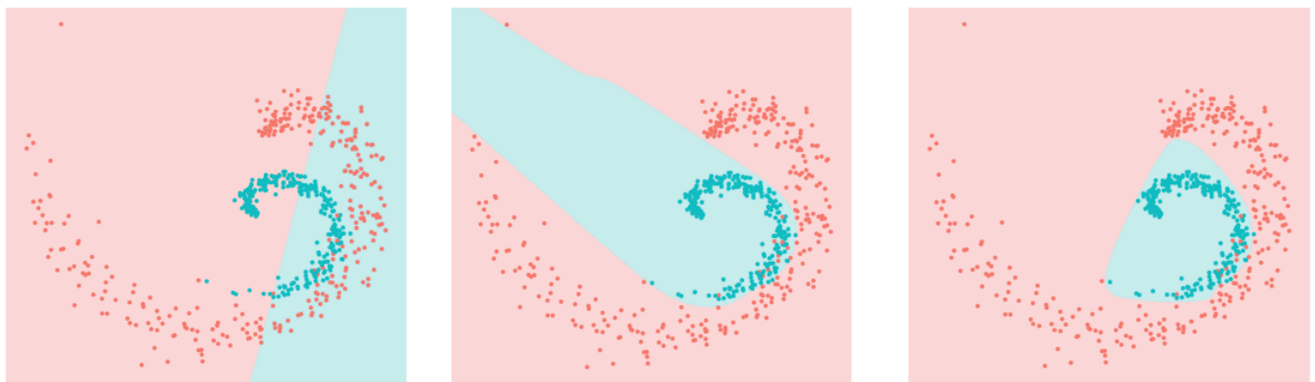
Common activation functions





Neural Networks are Expressive





(a) Sigmoid model ($L = 1$).

(b) One-hidden layer ($L = 2$) $N_1 = 4$ neurons.

(c) One-hidden layer ($L = 2$) $N_1 = 10$ neurons.

Feedforward equations

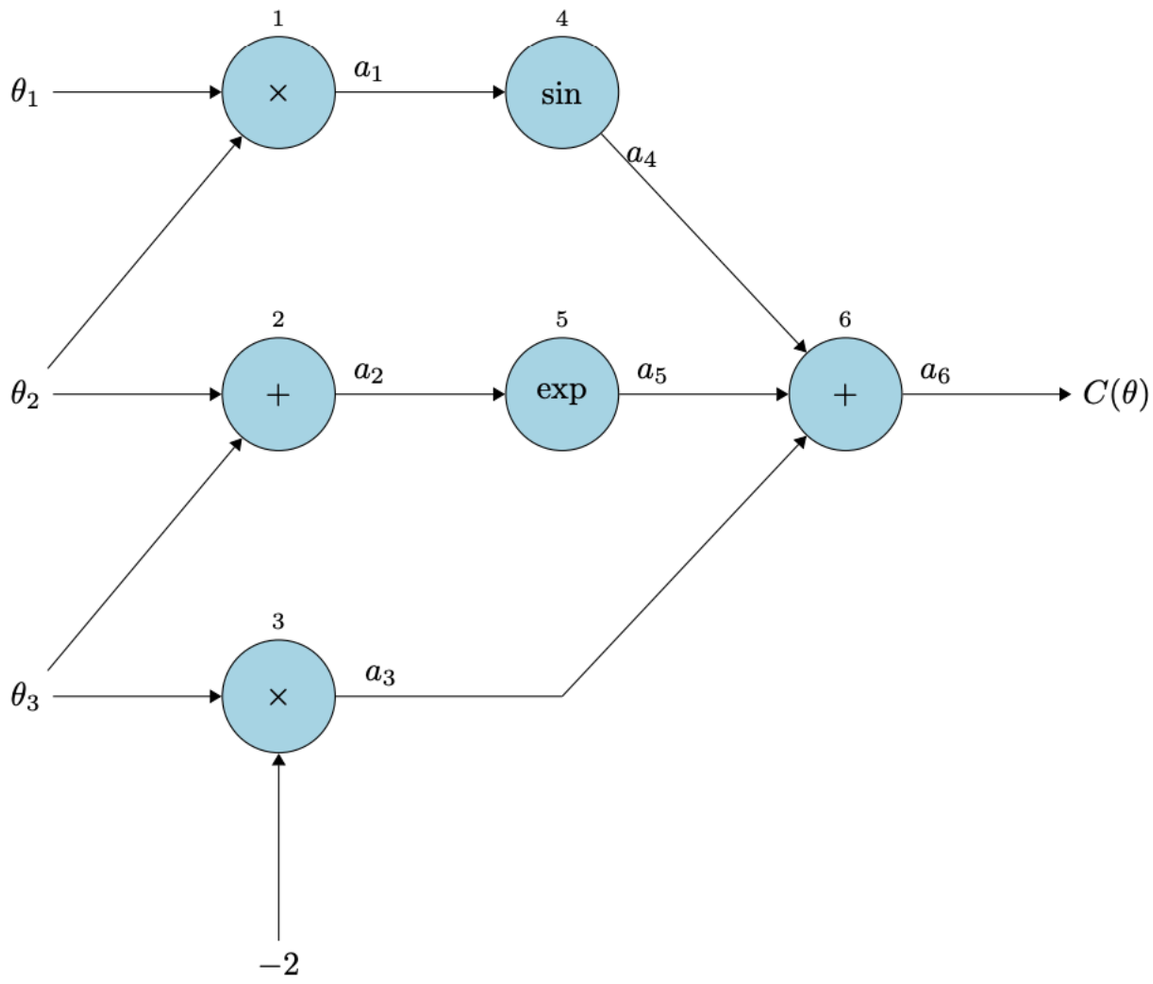
$$S^{[\ell]}(z) = \left[\sigma^{[\ell]}(z_1), \ldots, \sigma^{[\ell]}(z_{N_\ell})\right]^T$$

$$f_\theta(x) = \overbrace{S^{[2]}\left(W^{[2]}\underbrace{S^{[1]}(\overbrace{W^{[1]}x + b^{[1]}}^{z^{[1]}})}_{a^{[1]}} + b^{[2]}\right)}^{z^{[2]}},$$

$$\text{Affine Transformation}: \begin{cases} z_1^{[\ell]} &= {w_1^{[\ell]}}^\top a^{[\ell-1]} + b_1^{[\ell]} \\ z_2^{[\ell]} &= {w_2^{[\ell]}}^\top a^{[\ell-1]} + b_2^{[\ell]} \\ &\vdots \\ z_{N_\ell}^{[\ell]} &= {w_{N_\ell}^{[\ell]}}^\top a^{[\ell-1]} + b_{N_\ell}^{[\ell]} \end{cases} \Rightarrow \text{Activation Step}: \begin{cases} a_1^{[\ell]} &= \sigma\left(z_1^{[\ell]}\right) \\ a_2^{[\ell]} &= \sigma\left(z_2^{[\ell]}\right) \\ &\vdots \\ a_{N_\ell}^{[\ell]} &= \sigma\left(z_{N_\ell}^{[\ell]}\right) \end{cases}$$

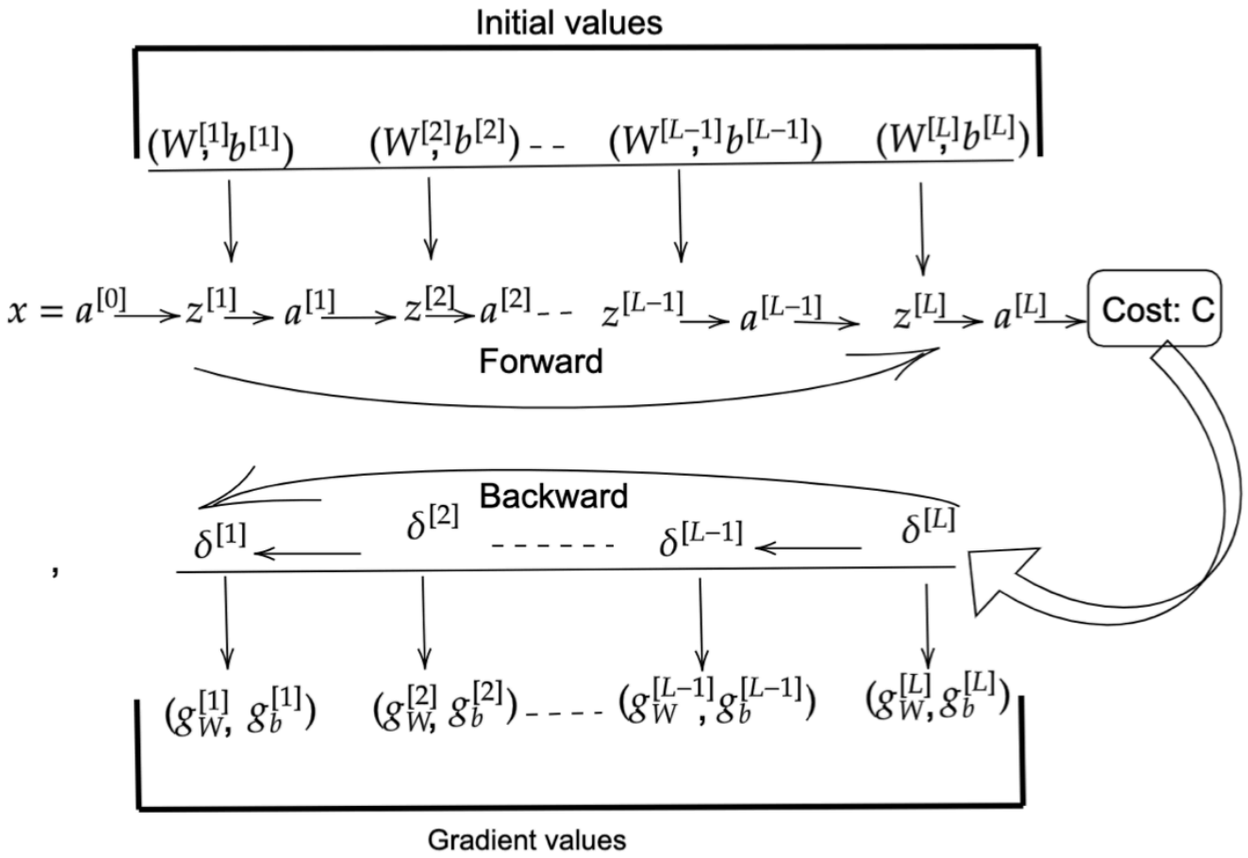
You have to _compute_ the gradient

Automatic differentiation



$$C(\theta) = \sin(\theta_1\,\theta_2) + \exp(\theta_2 + \theta_3) - 2\theta_3$$

The devil is in the details: Backpropagation



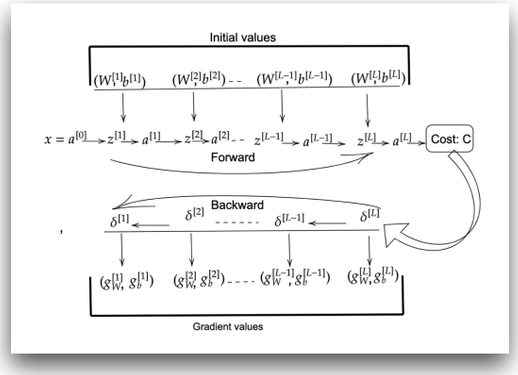
Algorithm 5.1: Back-Propagation for the general recursive model

Input: a sample (x, y) and parameter values $\theta = (\theta^{[1]}, \ldots, \theta^{[L]})$
Output: gradients of the loss $g_\theta^{[1]}, \ldots, g_\theta^{[L]}$

1 Compute $a^{[\ell]}$ for $\ell = 1, \ldots, L$ using (5.21) (Forward pass)
2 Compute $\zeta^{[L]} = \dot{C}(a^{[L]})$
3 Compute $g_\theta^{[L]} = \dot{f}_\theta^{[L]}(\theta^{[L]})\, \zeta^{[L]}$
4 **for** $\ell = L-1, \ldots, 1$ **do**
5 compute $\zeta^{[\ell]} = \dot{f}_a^{[\ell+1]}(a^{[\ell]})\, \zeta^{[\ell+1]}$
6 compute $g_\theta^{[\ell]} = \dot{f}_\theta^{[\ell]}(\theta^{[\ell]})\, \zeta^{[\ell]}$

(Backward mode automatic differentiation)

Deep learning frameworks



Keras

TensorFlow

fast.ai

PYTORCH

flux

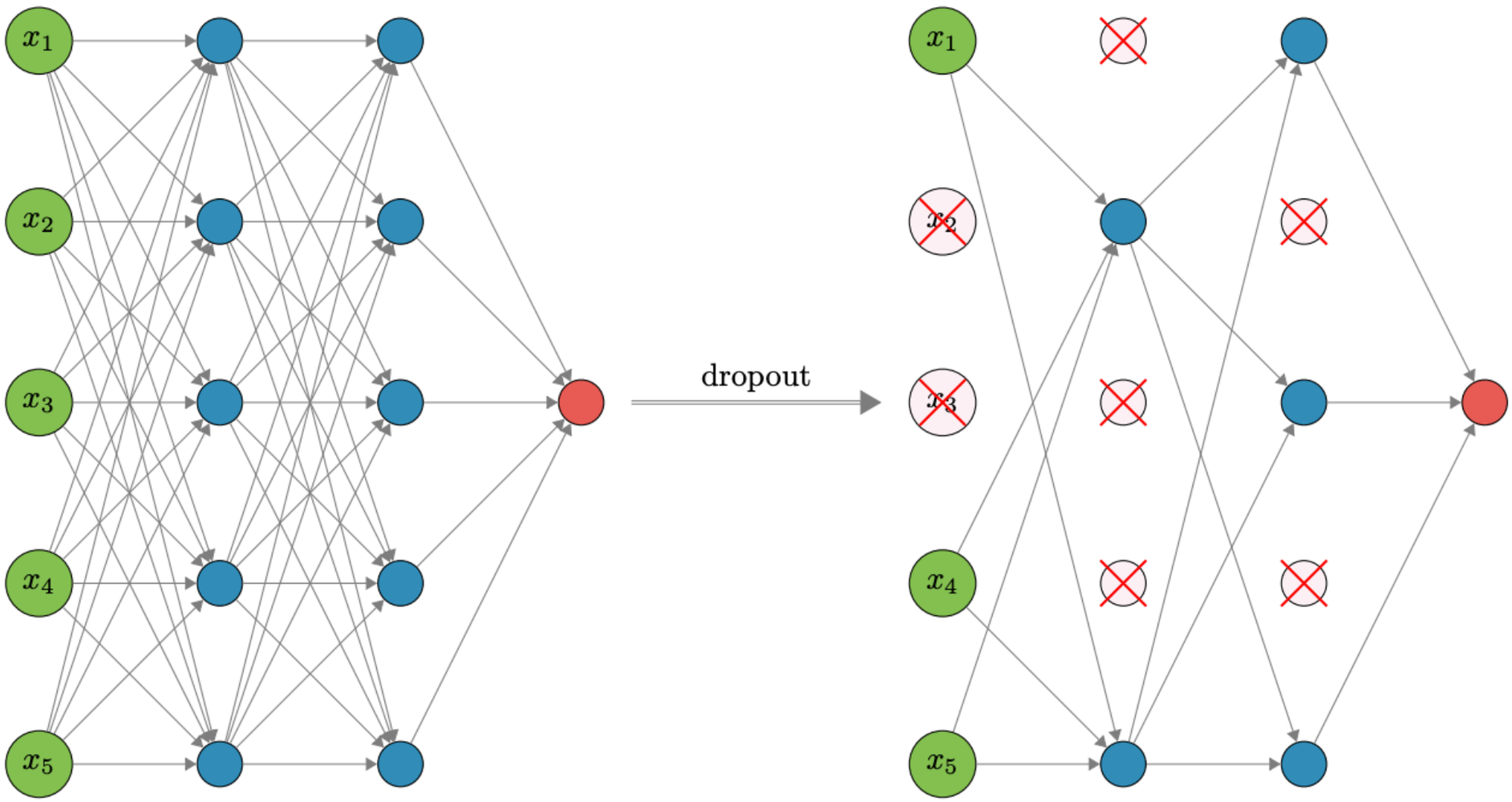
julia

cuDNN

NVIDIA CUDA

Common tricks

Dropout as a form of regularization



Batch normalization

$$\hat{\mu}_j^{[\ell]} = \frac{1}{n_b}\sum_{i=1}^{n_b} z_j^{[\ell](i)} \qquad \text{and} \qquad \hat{\sigma}_j^{[\ell]} = \sqrt{\frac{1}{n_b}\sum_{i=1}^{n_b}(z_j^{[\ell](i)} - \hat{\mu}_j^{[\ell]})^2}$$

$$\bar{z}_j^{[\ell](i)} = \frac{z_j^{[\ell](i)} - \hat{\mu}_j^{[\ell]}}{\sqrt{(\hat{\sigma}_j^{[\ell]})^2 + \varepsilon}}$$

Training:

$$\tilde{z}_j^{[\ell](i)} = \gamma_j^{[\ell]}\,\bar{z}_j^{[\ell](i)} + \beta_j^{[\ell]}$$

Production:

$$\tilde{z}_j^{[\ell]} = \gamma_j^{[\ell]}\,\frac{z_j^{[\ell]} - \hat{\bar{\mu}}_j^{[\ell]}}{\sqrt{\left(\hat{\bar{\sigma}}_j^{[\ell]}\right)^2 + \varepsilon}} + \beta_j^{[\ell]}$$

Weight initialization

$$a_i^{[\ell]} = \sigma\left(z_i^{[\ell]}\right), \qquad \text{where} \qquad z_i^{[\ell]} = \sum_{j=1}^{N_{\ell-1}} w_{ij}^{[\ell]} a_j^{[\ell-1]} + b_i^{[\ell]}$$

Want similarly distributed…

Images and CNN

The old trade of filtering…

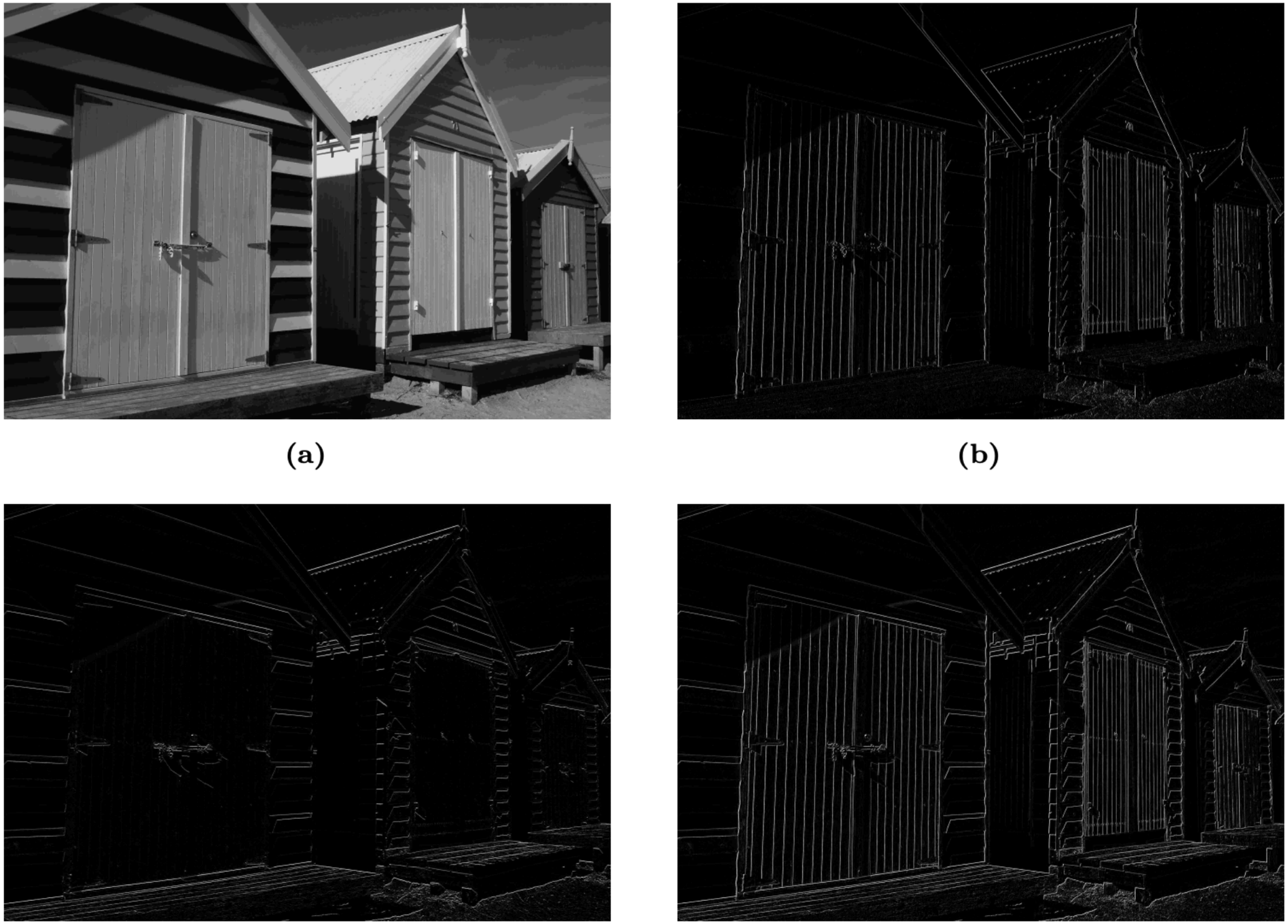


Figure 6.1.: Edge detection using the Sobel filter.

Convolutional Neural Networks

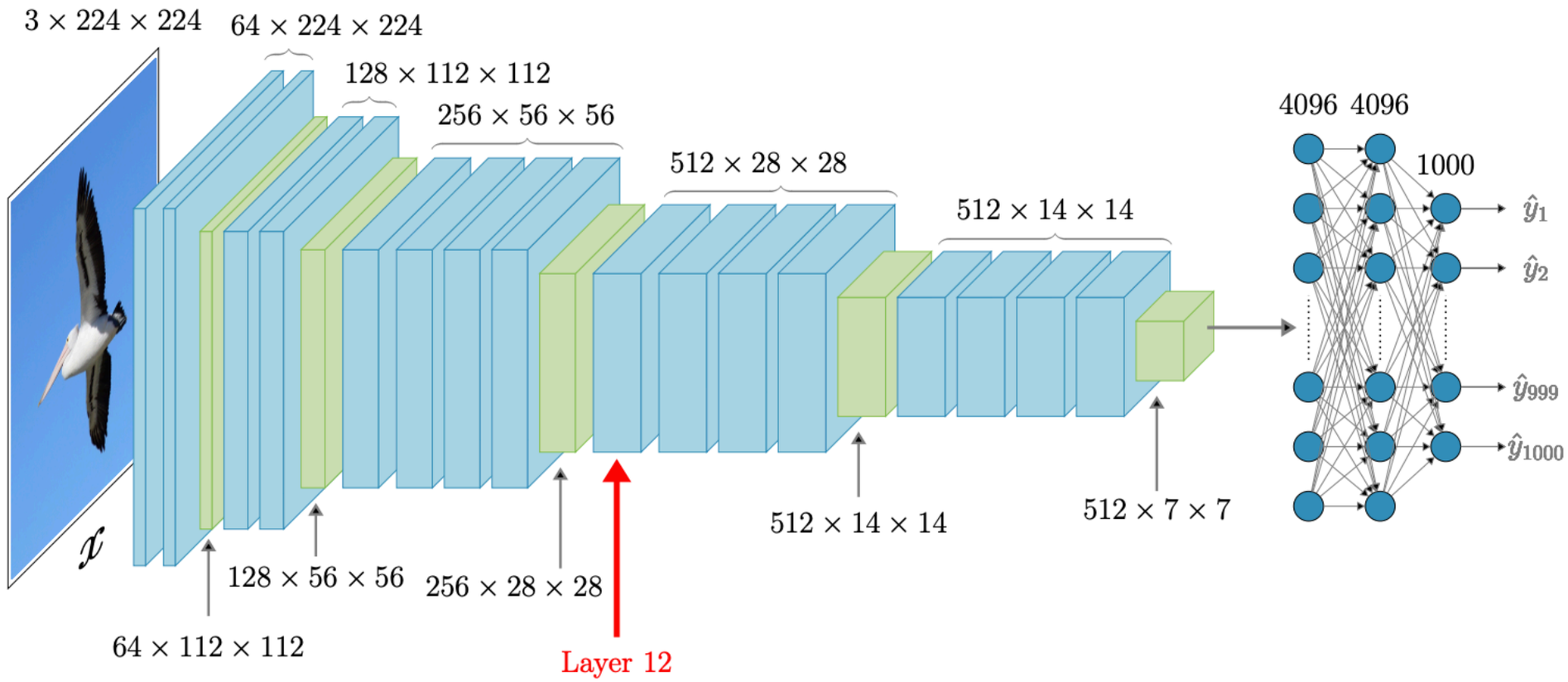


Figure 6.2.: The VGG19 network architecture. An input x is a $3 \times 224 \times 224$ color image. It is processed through a series of convolutional layers followed by fully connected layers. The resulting output, $\hat{y}_1, \ldots, \hat{y}_{1000}$ is a vector of probabilities indicating the class of the image.

The convolution

$$(f \star g)(t) = \sum_{\tau \in \mathbb{Z}} f(t-\tau) g(\tau)$$

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} = \underbrace{\begin{bmatrix} f_0 & 0 & 0 & 0 & 0 & 0 \\ f_1 & f_0 & 0 & 0 & 0 & 0 \\ f_2 & f_1 & f_0 & 0 & 0 & 0 \\ 0 & f_2 & f_1 & f_0 & 0 & 0 \\ 0 & 0 & f_2 & f_1 & f_0 & 0 \\ 0 & 0 & 0 & f_2 & f_1 & f_0 \\ 0 & 0 & 0 & 0 & f_2 & f_1 \\ 0 & 0 & 0 & 0 & 0 & f_2 \end{bmatrix}}_{T(f)} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \end{bmatrix}$$

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$
$w_{3,1}$	$w_{3,2}$	$w_{3,3}$

W

$\star$

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$	$x_{1,6}$	$x_{1,7}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$	$x_{2,6}$	$x_{2,7}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$	$x_{3,5}$	$x_{3,6}$	$x_{3,7}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$	$x_{4,5}$	$x_{4,6}$	$x_{4,7}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$	$x_{5,5}$	$x_{5,6}$	$x_{5,7}$
$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$	$x_{6,5}$	$x_{6,6}$	$x_{6,7}$

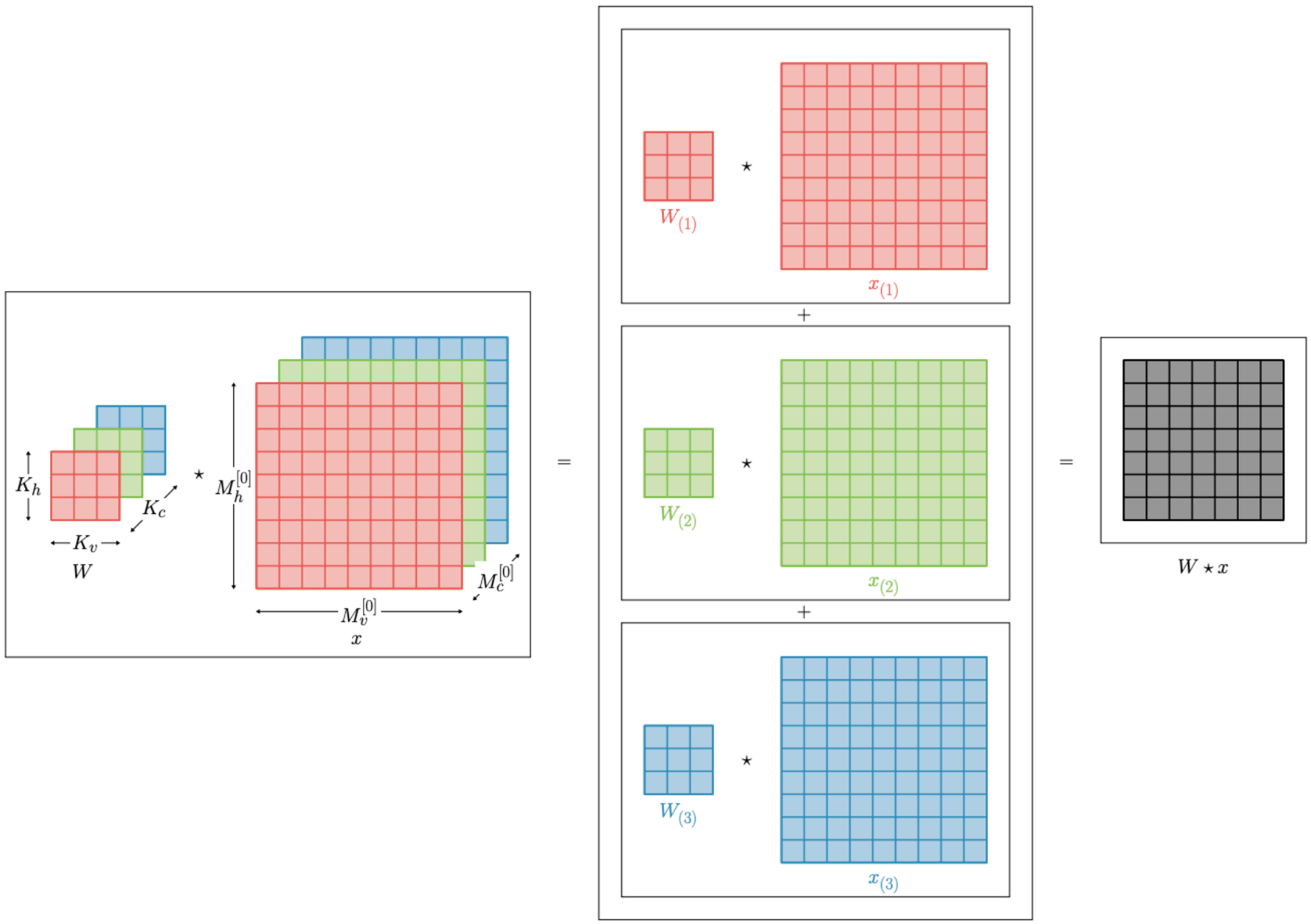
x

$=$

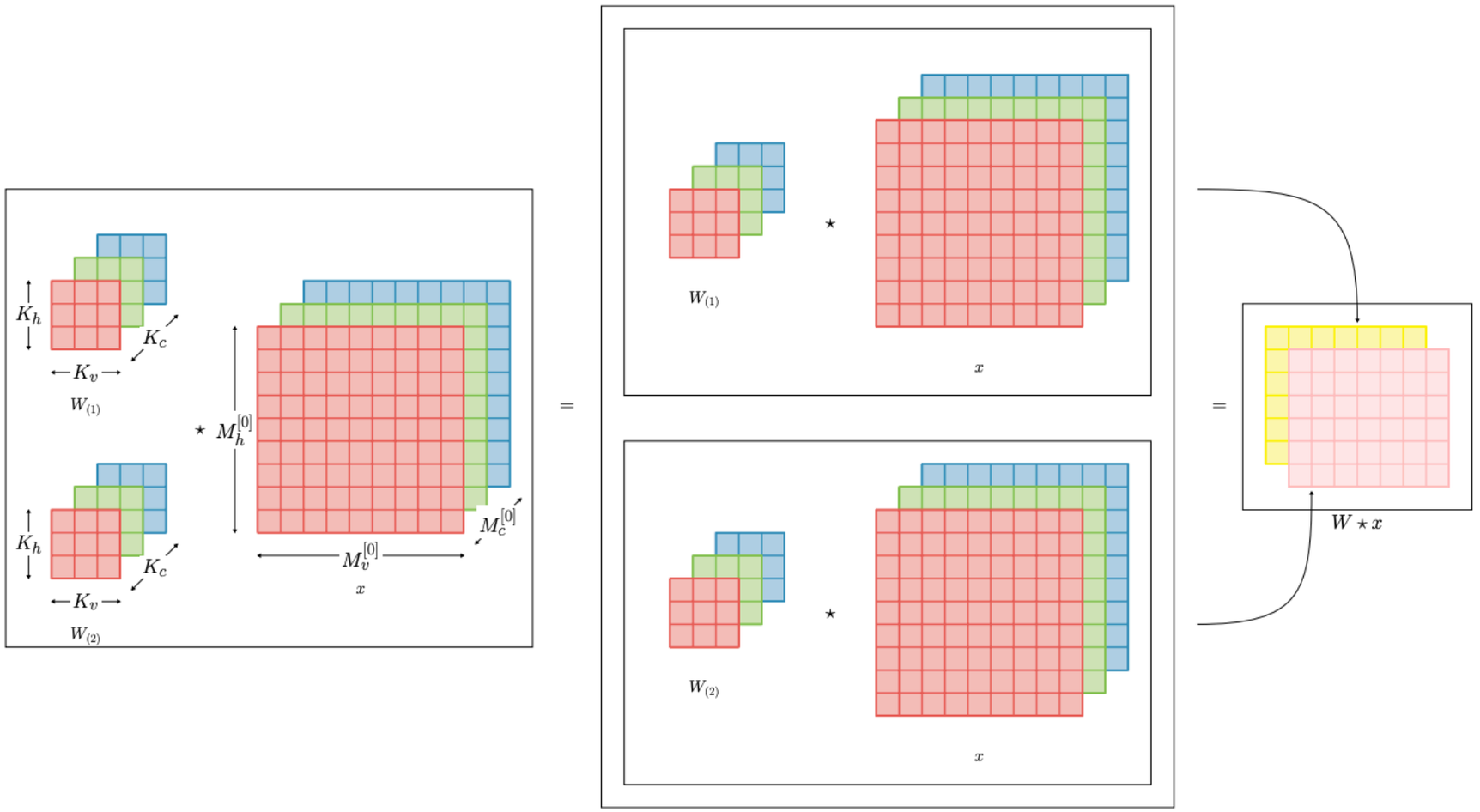
$z_{1,1}$	$z_{1,2}$	$z_{1,3}$	$z_{1,4}$	$z_{1,5}$
$z_{2,1}$	$z_{2,2}$	$z_{2,3}$	$z_{2,4}$	$z_{2,5}$
$z_{3,1}$	$z_{3,2}$	$z_{3,3}$	$z_{3,4}$	$z_{3,5}$
$z_{4,1}$	$z_{4,2}$	$z_{4,3}$	$z_{4,4}$	$z_{4,5}$

z

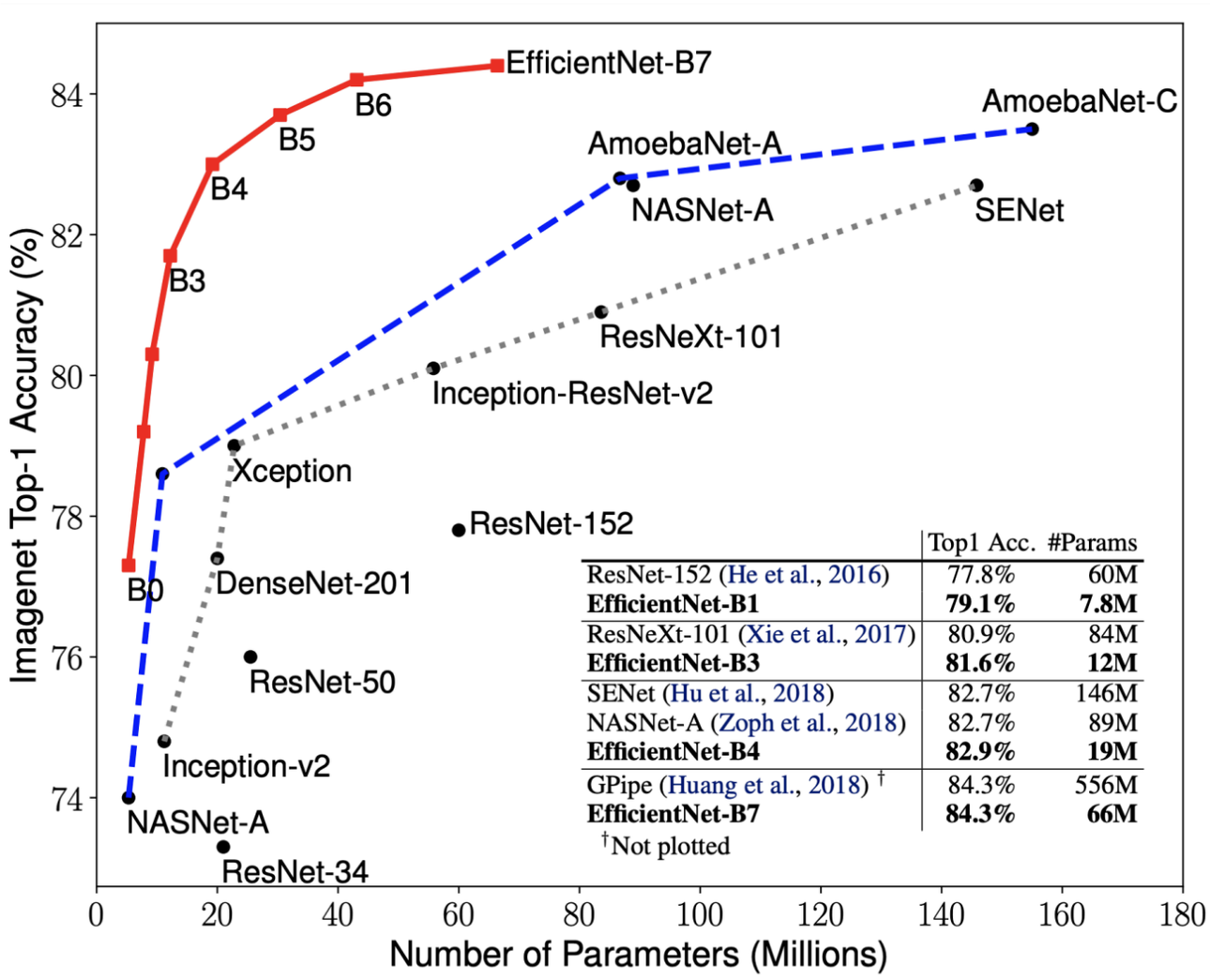
Volume convolution



Multiple output channels



Near state of the art in classification



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

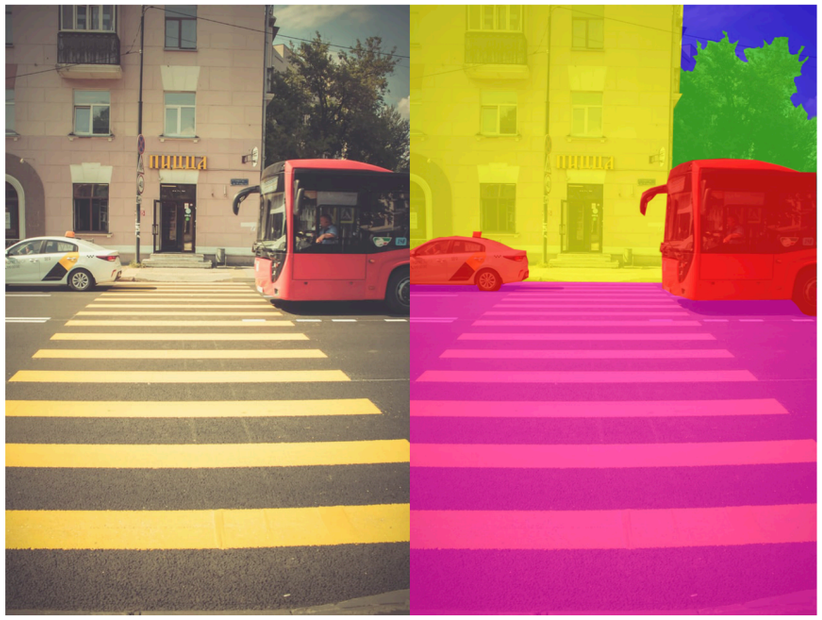
[Submitted on 28 May 2019 (v1), last revised 11 Sep 2020 (this version, v5)]

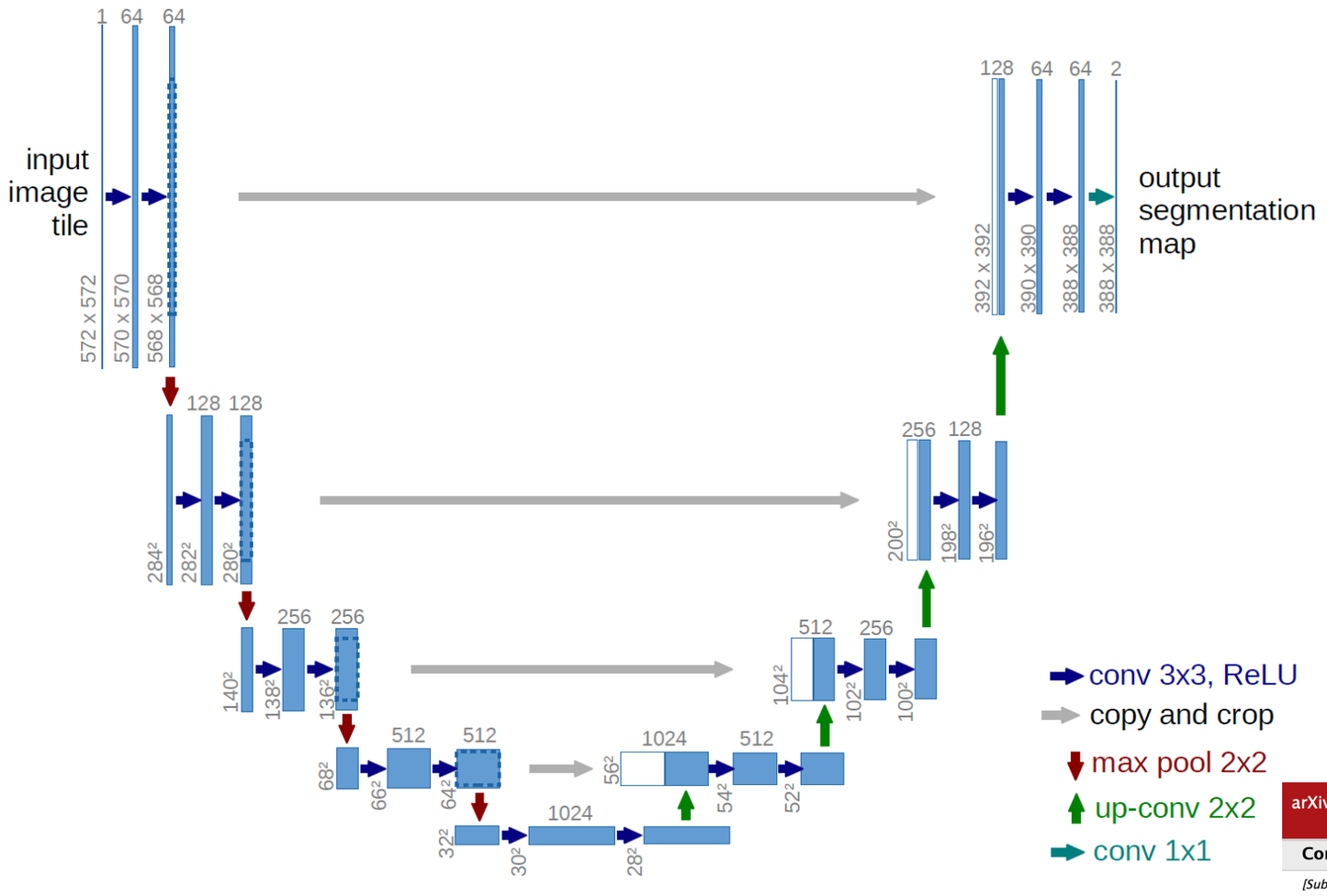
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan, Quoc V. Le

Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more re

Beyond classification: e.g. segmentation





arXiv.org > cs > arXiv:1505.04597

Computer Science > Computer Vision and Pattern Recognition

[Submitted on 18 May 2015]

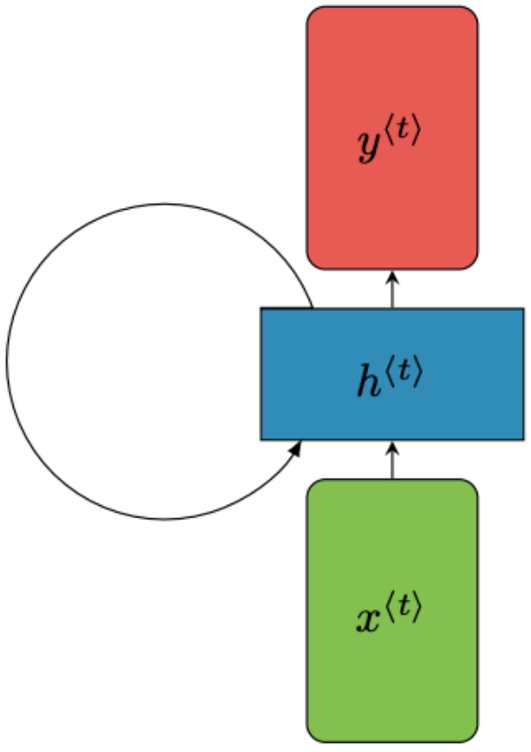
U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, Thomas Brox

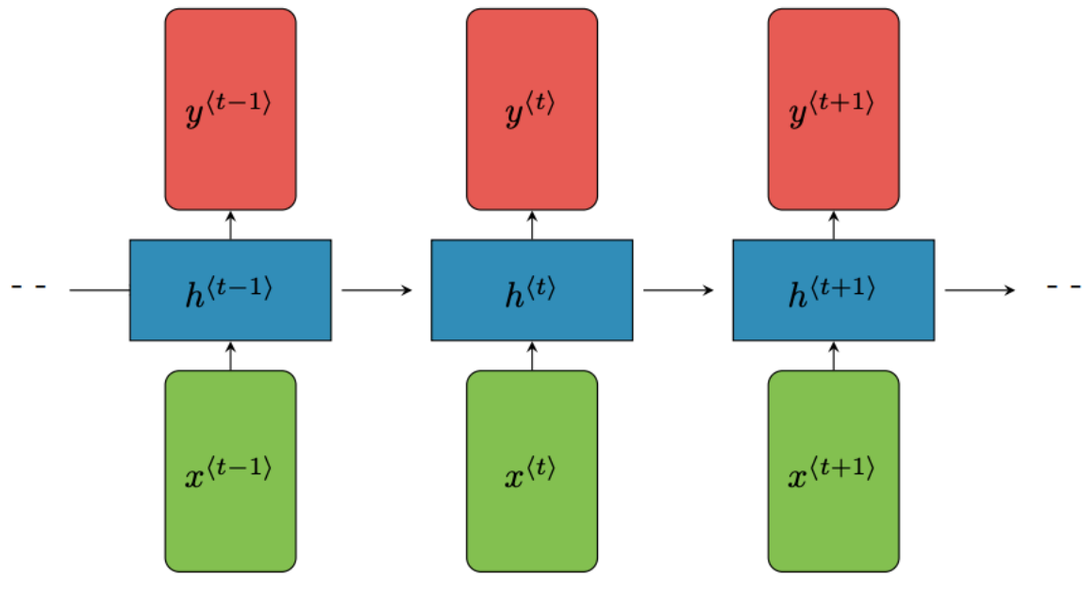
Text: RNN, Attention, and Transformers

Recurrent neural networks (RNN)

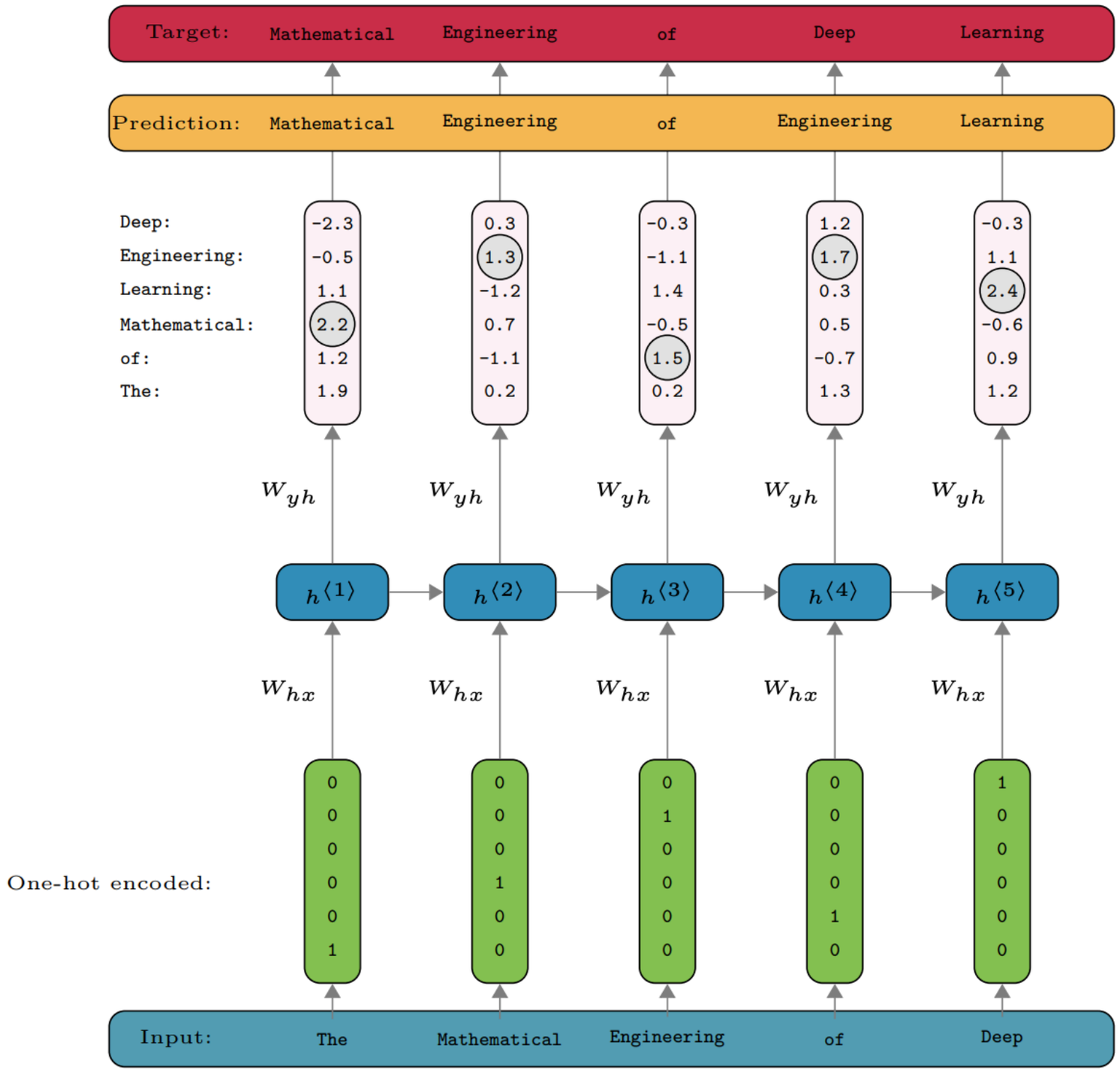
$$\begin{cases} h^{\langle t \rangle} = S_h(W_{hh}h^{\langle t-1 \rangle} + W_{hx}x^{\langle t \rangle} + b_h) \\ \hat{y}^{\langle t \rangle} = S_y(W_{yh}h^{\langle t \rangle} + b_y). \end{cases}$$



$$h^{\langle t \rangle} = f(\ldots f(\underbrace{f(\underbrace{f(h^{\langle 0 \rangle}, x^{\langle 1 \rangle})}_{h^{\langle 1 \rangle}}, x^{\langle 2 \rangle})}_{h^{\langle 2 \rangle}}, x^{\langle 3 \rangle})\ldots, x^{\langle t \rangle}).$$



Predictive auto-regressive training



Handle long term memory with more complex structures…

GRU

$$\begin{cases} g_r = S_{\text{Sig}}\big(W_{rh}h^{\langle t-1\rangle} + W_{rx}x^{\langle t\rangle} + b_r\big), \\ g_u = S_{\text{Sig}}\big(W_{uh}h^{\langle t-1\rangle} + W_{ux}x^{\langle t\rangle} + b_u\big). \end{cases}$$

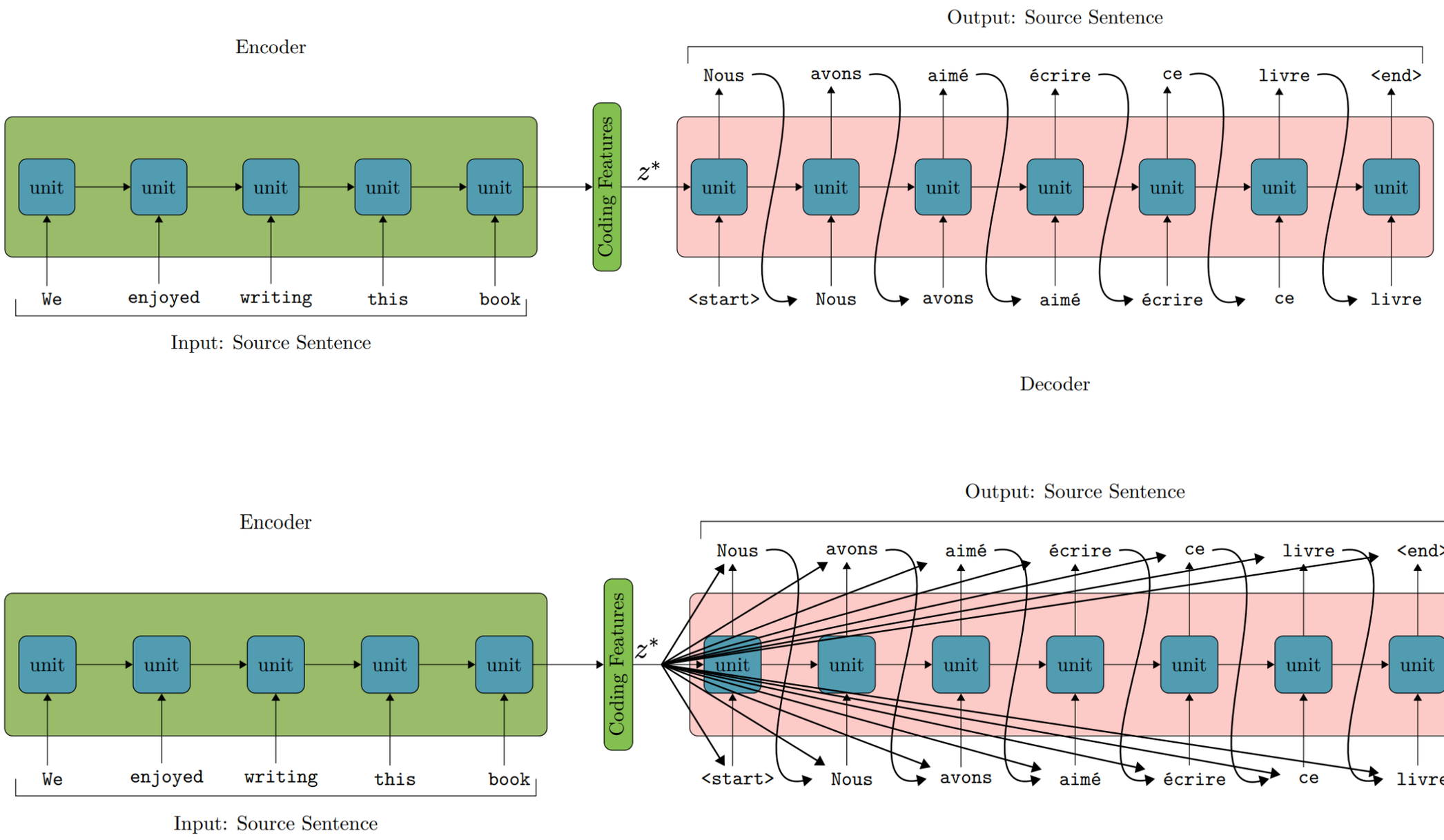
$$\begin{cases} h^{\langle t\rangle} = (1-g_u)\odot h^{\langle t-1\rangle} \;+\; g_u \odot \underbrace{S_{\text{Tanh}}\left(W_{\tilde{h}h}\,(g_r \odot h^{\langle t-1\rangle}) + W_{\tilde{h}x}\,x^{\langle t\rangle} + b_{\tilde{h}}\right)}_{\tilde{h}^{\langle t\rangle}}, \\ \\ \hat{y}^{\langle t\rangle} = S_y(W_{yh}h^{\langle t\rangle} + b_y), \end{cases}$$

LSTM

$$\begin{cases} c^{\langle t\rangle} = g_f \odot c^{\langle t-1\rangle} \;+\; g_i \odot \underbrace{S_{\text{Tanh}}\left(W_{\tilde{c}h}\,h^{\langle t-1\rangle} + W_{\tilde{c}x}\,x^{\langle t\rangle} + b_{\tilde{c}}\right)}_{\tilde{c}^{\langle t\rangle}} \\ \\ h^{\langle t\rangle} = g_o \odot S_{\text{Tanh}}(c^{\langle t\rangle}) \\ \\ \hat{y}^{\langle t\rangle} = S_y(W_{yh}h^{\langle t\rangle} + b_y), \end{cases}$$

$$\begin{cases} g_f = S_{\text{Sig}}\big(W_{fh}h^{\langle t-1\rangle} + W_{fx}x^{\langle t\rangle} + b_f\big) & \text{(forget gate)} \\ g_i = S_{\text{Sig}}\big(W_{ih}h^{\langle t-1\rangle} + W_{ix}x^{\langle t\rangle} + b_i\big) & \text{(input gate)} \\ g_o = S_{\text{Sig}}\big(W_{oh}h^{\langle t-1\rangle)} + W_{ox}x^{\langle t\rangle} + b_o\big) & \text{(output gate)} \end{cases}$$

Encoder decoder architectures…



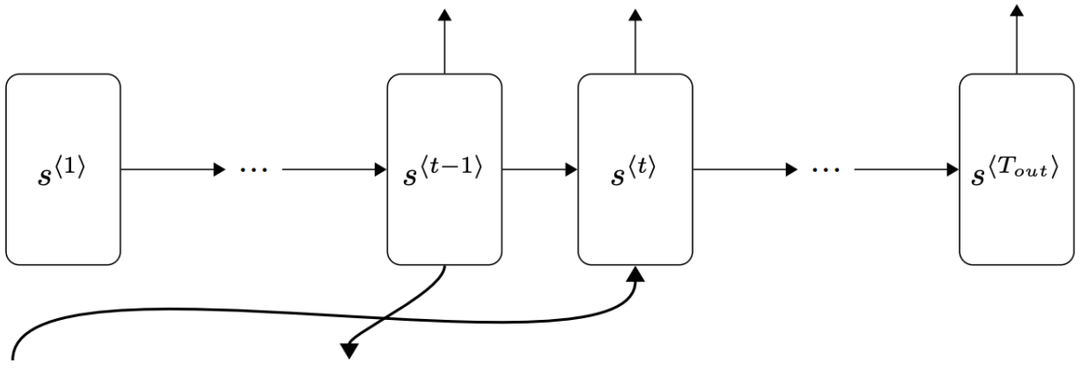
The attention mechanism

$$
\begin{aligned}
s^{\langle t \rangle} &= f_{\text{decoder}}\left(s^{\langle t-1 \rangle}, \hat{y}^{\langle t-1 \rangle}, u^{\langle t \rangle}\right), \\
\hat{y}^{\langle t \rangle} &= f_{\text{decoder-out}}\left(s^{\langle t \rangle}\right),
\end{aligned}
$$

$$
\begin{cases}
u^{\langle t \rangle} = \sum_{\tau=1}^{T_{\text{in}}} \alpha_{\tau}^{\langle t \rangle} v^{\langle \tau \rangle}, & \text{(Non-causal attention)} \\
\qquad\qquad \text{or} \\
u^{\langle t \rangle} = \sum_{\tau=1}^{t} \alpha_{\tau}^{\langle t \rangle} v^{\langle \tau \rangle}. & \text{(Causal attention)}
\end{cases}
$$

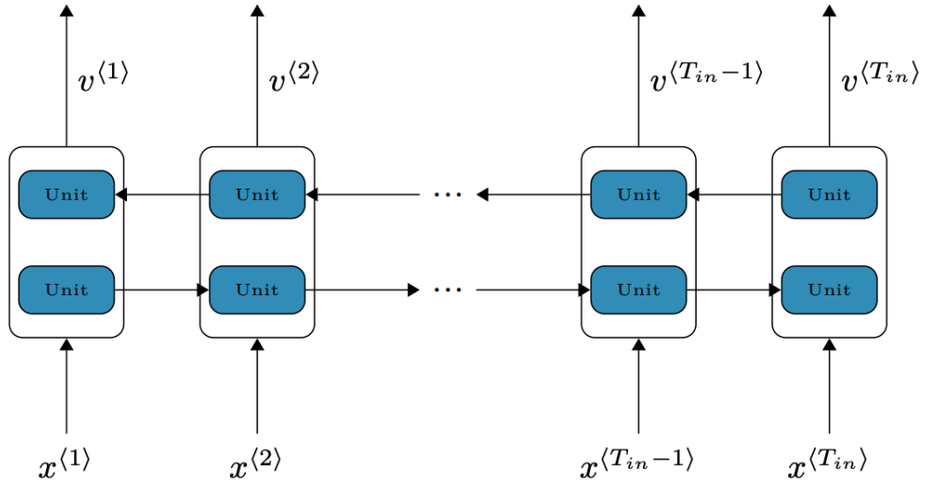
$$
\alpha^{\langle t \rangle} = S_{\text{softmax}}\left(\begin{bmatrix} s(z_q^{\langle t \rangle}, z_k^{\langle 1 \rangle}) \\ s(z_q^{\langle t \rangle}, z_k^{\langle 2 \rangle}) \\ \vdots \\ s(z_q^{\langle t \rangle}, z_k^{\langle T_{\text{in}} \rangle}) \end{bmatrix}\right), \qquad \text{for} \qquad t = 1, \ldots, T_{\text{out}}.
$$

Decoder



$$
u^{\langle t \rangle} = \text{Attention}_t[z_q = s^{\langle t-1 \rangle}, z_k = (v^{\langle 1 \rangle}, \ldots, v^{\langle T_{in} \rangle}), v = (v^{\langle 1 \rangle}, \ldots, v^{\langle T_{in} \rangle})]
$$

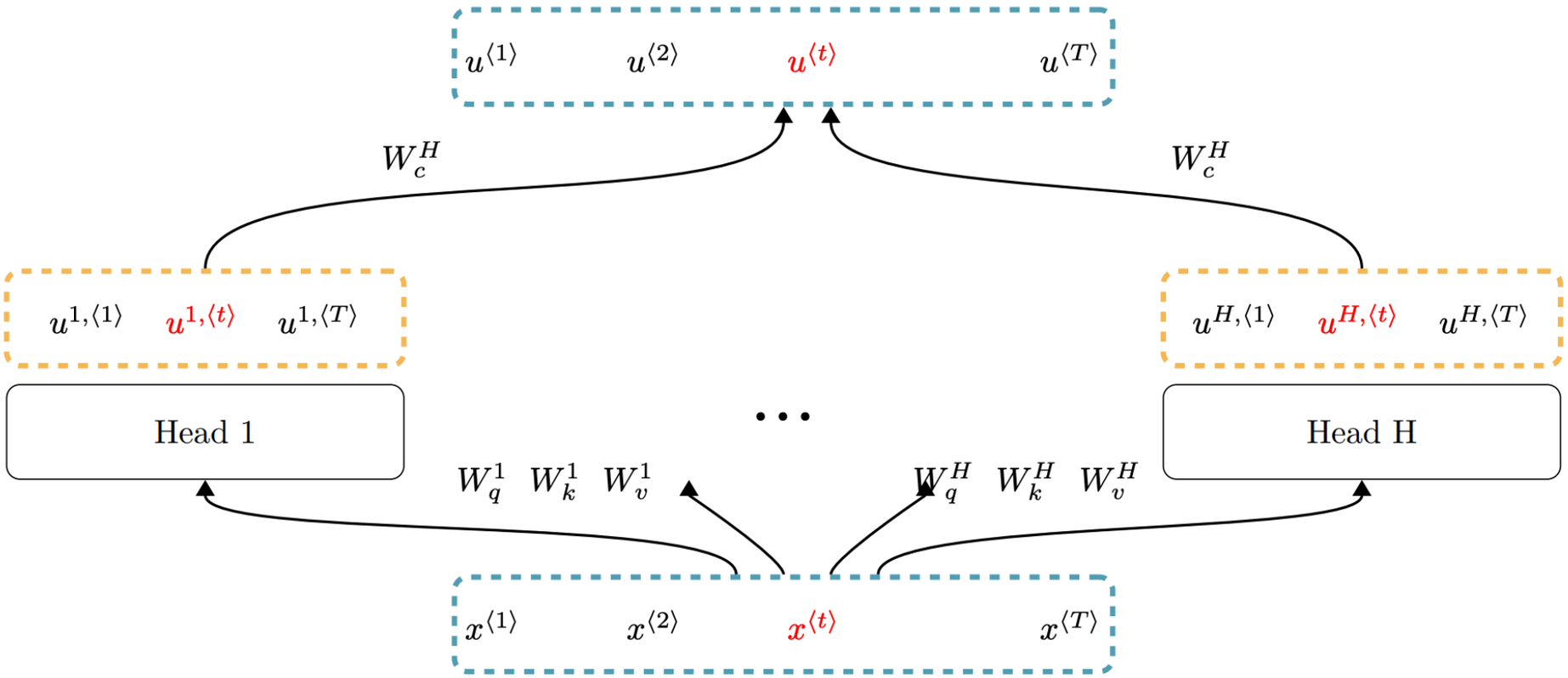
Encoder



$[\alpha_{\tau}^{\langle t \rangle}] =$

	We $\tau=1$	love $\tau=2$	deep $\tau=3$	learning $\tau=4$	<stop> $\tau=5$	
nous	0.93	0.02	0.01	0.02	0.02	t=1
aimons	0.44	0.46	0.04	0.05	0.01	t=2
l'	0.23	0.26	0.23	0.25	0.03	t=3
apprentissage	0.01	0.04	0.45	0.47	0.03	t=4
en	0.01	0.02	0.39	0.55	0.03	t=5
profondeur	0.02	0.01	0.67	0.28	0.02	t=6
<stop>	0.01	0.01	0.01	0.01	0.96	t=7

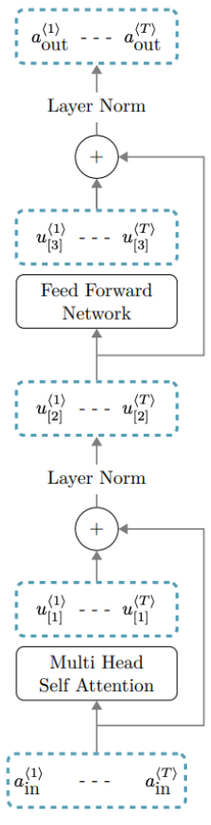
Multi-head self attention block…



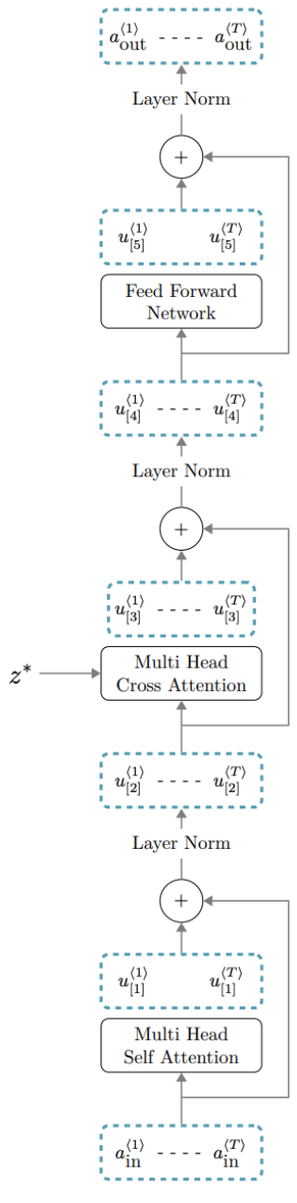
$$u^{h,\langle t\rangle} = \sum_{\tau=1}^{T} \alpha_\tau^{h,\langle t\rangle} W_v^h x^{\langle \tau\rangle}, \qquad \text{with} \qquad \alpha_\tau^{h,\langle t\rangle} = \frac{e^{s(W_q^h x^{\langle t\rangle}, W_k^h x^{\langle \tau\rangle})}}{\sum_{t'=1}^{t} e^{s(W_q^h x^{\langle t\rangle}, W_k^h x^{\langle t'\rangle})}}. \qquad u^{\langle t\rangle} = \sum_{h=1}^{H} W_c^h u^{h,\langle t\rangle},$$

Transformer architectures

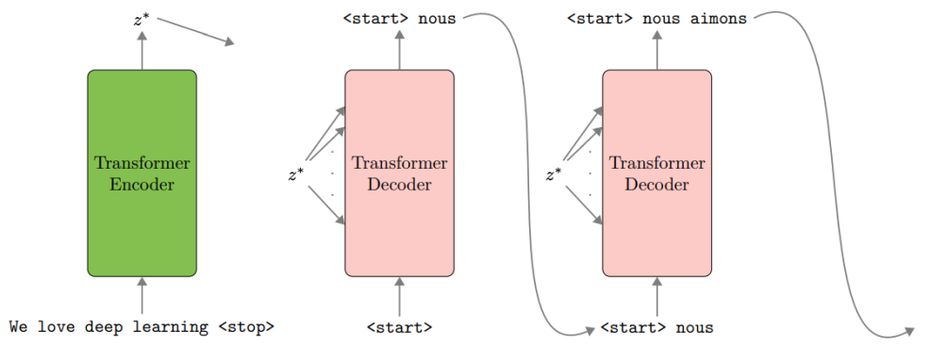
Encoder

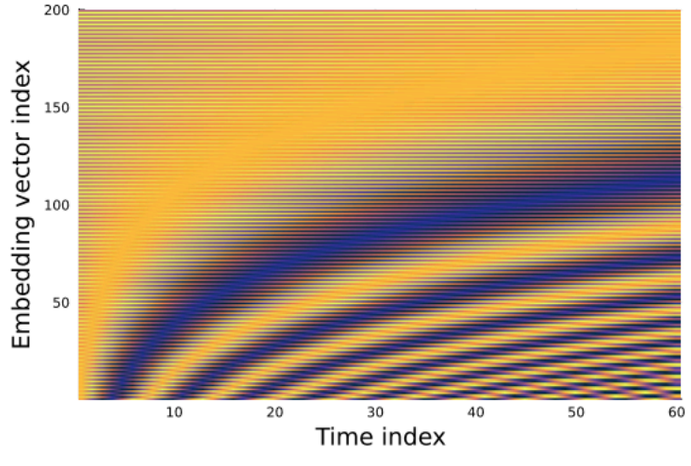


Decoder



$$
\begin{cases}
\hat{y}^{\langle 2\rangle} = f_{\text{decoder}}\left(z_*, (\tilde{y}^{\langle 1\rangle})\right), \\
\hat{y}^{\langle 3\rangle} = f_{\text{decoder}}\left(z_*, (\tilde{y}^{\langle 1\rangle}, \tilde{y}^{\langle 2\rangle})\right), \\
\hat{y}^{\langle 4\rangle} = f_{\text{decoder}}\left(z_*, (\tilde{y}^{\langle 1\rangle}, \tilde{y}^{\langle 2\rangle}, \tilde{y}^{\langle 3\rangle})\right), \\
\quad \vdots \\
\hat{y}^{\langle T\rangle} = f_{\text{decoder}}\left(z_*, (\tilde{y}^{\langle 1\rangle}, \tilde{y}^{\langle 2\rangle}, , \ldots, \tilde{y}^{\langle t-1\rangle})\right).
\end{cases}
$$





Generative Adversarial Networks, GANs

arXiv > stat > arXiv:1406.2661

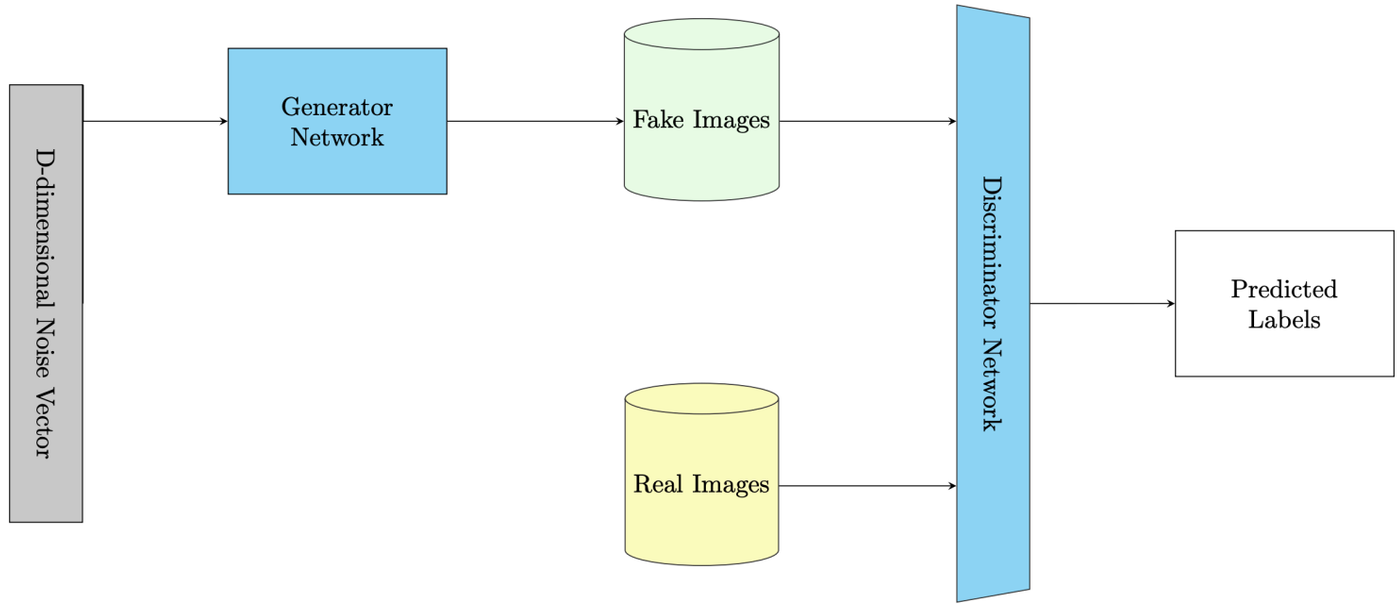
Statistics > Machine Learning

[Submitted on 10 Jun 2014]

Generative Adversarial Networks

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio

GAN basics



2019: A style-based generator architecture for generative adversarial networks by Tero Karras, Samuli Laine and Timo Aila.

2018: Large scale GAN training for high fidelity natural image synthesis by Andrew Brock, Jeff Donahue, and Karen Simonyan.

A sort of “game” between G and D

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Where to next with your next project?

As you embark on the project, ask yourself…

Am I trying to mimic human level performance, or am I building predictive models, or something else?

Do I have the data that I need? Is it **labelled**?

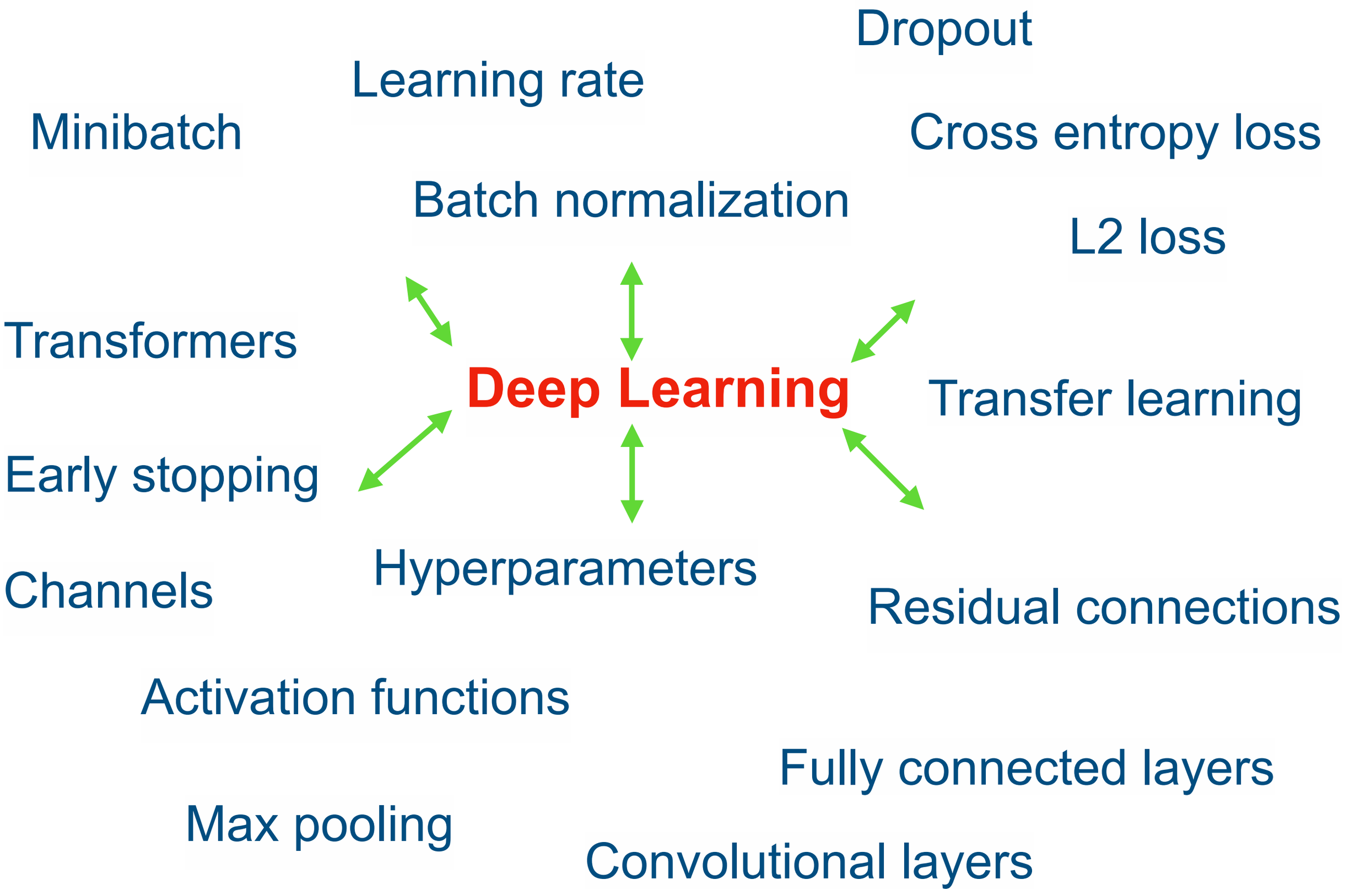
Where will I compute?

How will the model be deployed?

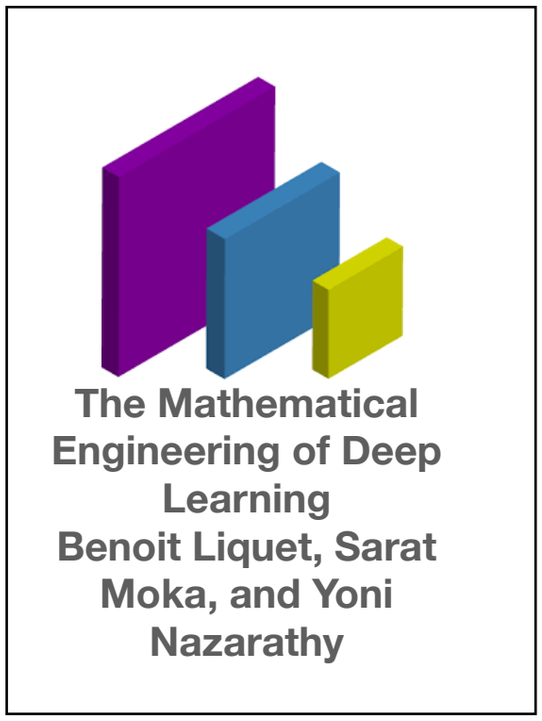
Did I spend enough time searching who has done something similar? What can I reuse?

How long is this going to take? Do I have time to refine iterations? Do I have organized workflows?

Don’t be afraid of the terminology…



Thank You



https://deeplearningmath.org/