ROB 430/599: Deep Learning for Robot Perception and Manipulation (DeepRob)

Lecture 4 Optimization (cont'd)

Lecture 5: Neural Networks

01/26/2026

ROBOTICS

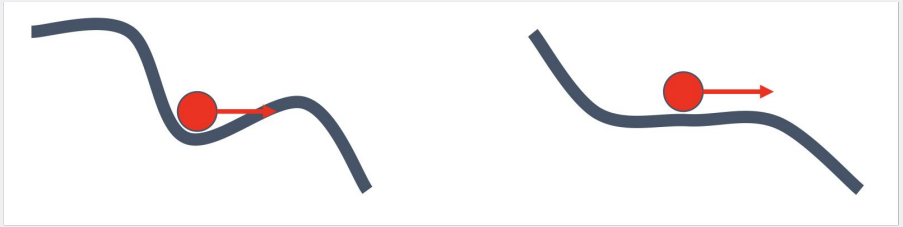
Today

- Wrap up Optimization (30 min)
- Start Neural Networks
  - Image Features (20min)
  - Neural Networks, Activation Functions (20min)
- Summary and Takeaways (5min)

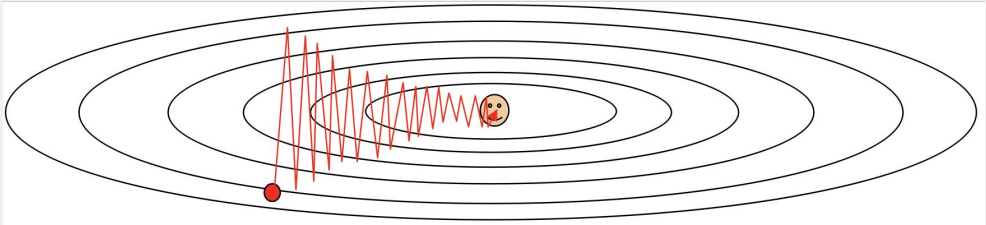
Recap: SGD + Momentum

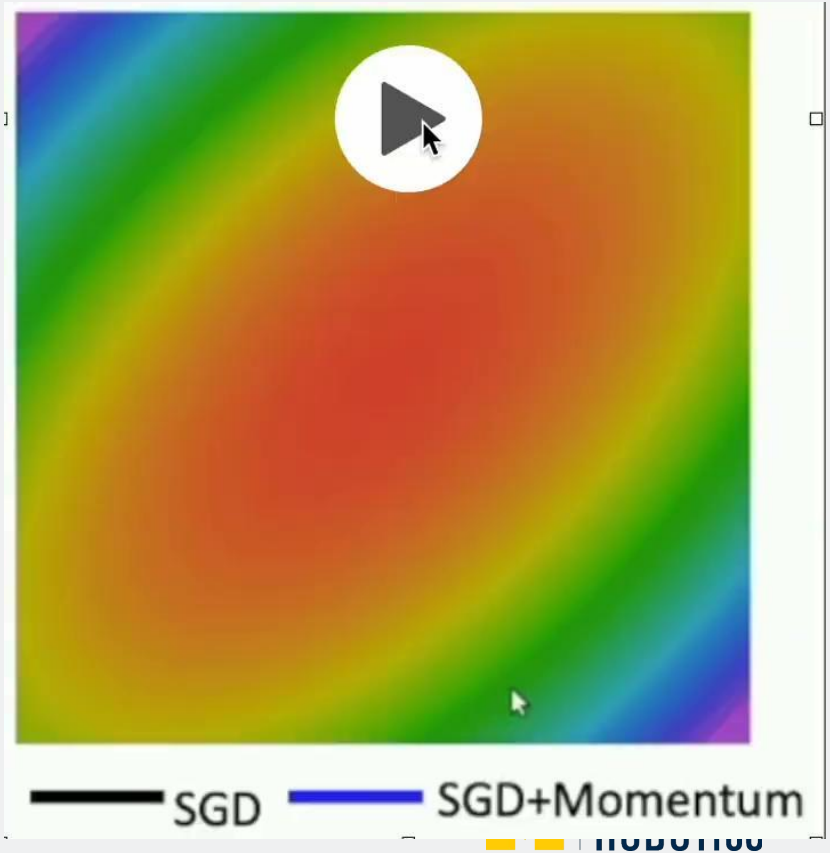
Local Minima

Saddle Points



Poor Conditioning





AdaGrad

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

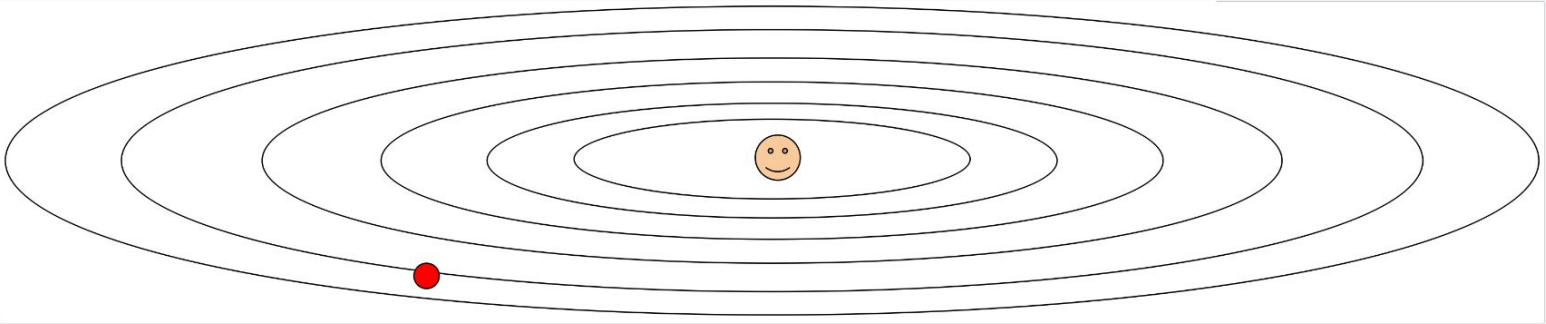
- Added element-wise scaling of the gradient based on the historical sum of squares in each dimension
- “Per-parameter learning rates” or “adaptive learning rates”

M | ROBOTICS

AdaGrad

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

Problem: AdaGrad will slow over many iterations



Q: What happens with AdaGrad?

Progress along “steep” directions is damped; progress along “flat” directions is accelerated

Duchi et al, “Adaptive sub gradient methods for online learning and stochastic optimization,” JMLR 2011

RMSProp: “Leaky AdaGrad”

AdaGrad

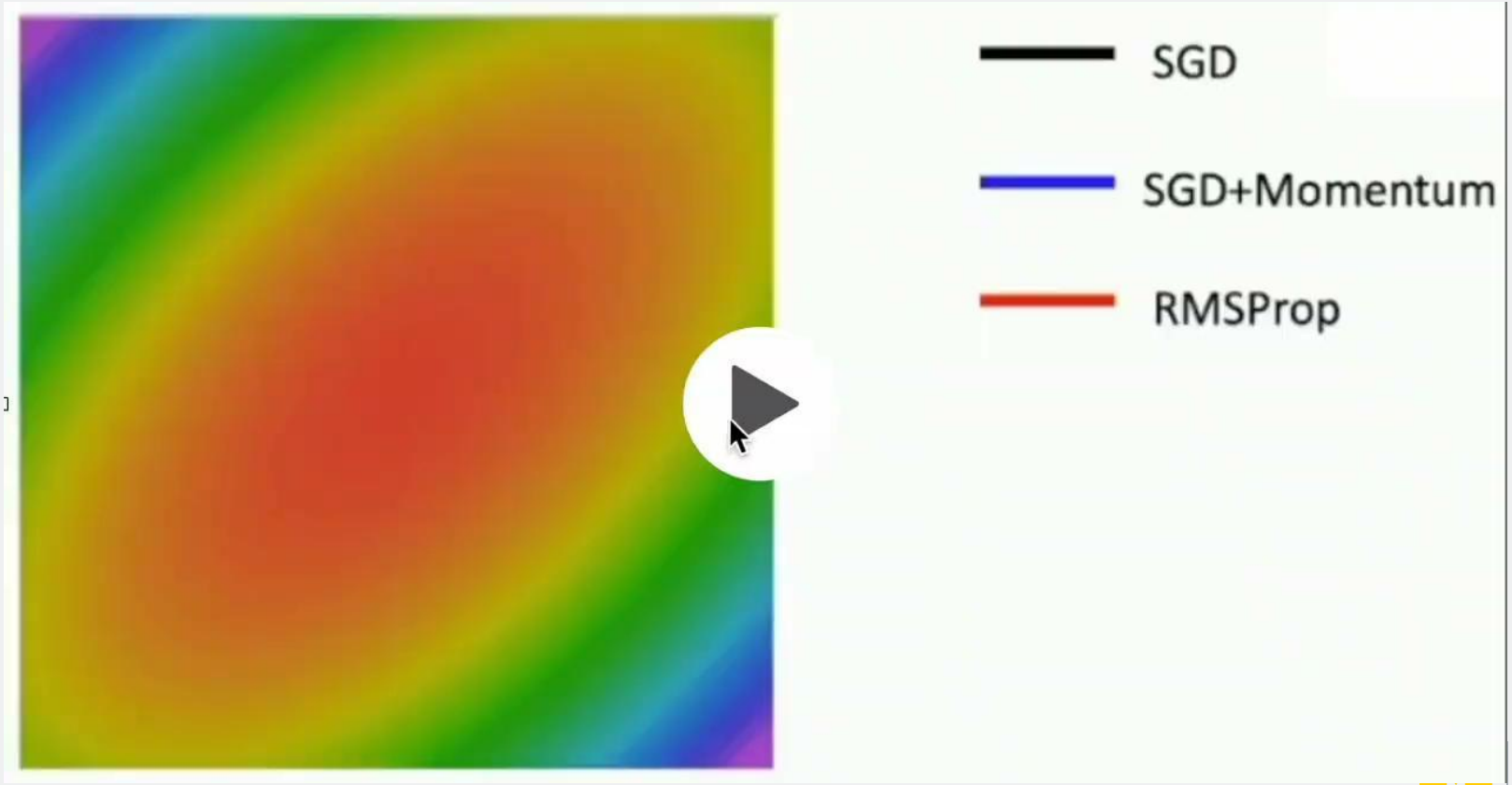
```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

RMSProp

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

M | ROBOTICS

RMSProp: “Leaky AdaGrad”



RMSProp + Momentum (“Almost” Adam)

```
moment1 = 0
moment2 = 0
for t in range(1, num_steps + 1):  # Start at t = 1
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

RMSProp + Momentum (“Almost” Adam)

```
moment1 = 0
moment2 = 0
for t in range(1, num_steps + 1):  # Start at t = 1
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Adam

Momentum

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

SGD+Momentum

M | ROBOTICS

RMSProp + Momentum ("Almost" Adam)

```
moment1 = 0
moment2 = 0
for t in range(1, num_steps + 1):  # Start at t = 1
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Adam

Momentum

AdaGrad / RMSProp

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

RMSProp

Kingma and Ba, "Adam: A method for stochastic optimization," ICLR 2015

RMSProp + Momentum ("Almost" Adam)

```
moment1 = 0
moment2 = 0
for t in range(1, num_steps + 1):  # Start at t = 1
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Adam

Momentum

AdaGrad / RMSProp

Q: What happens at t=1?
(Assume beta2 = 0.999)

Aha Slides
(In-class participation)

https://ahaslides.com/TA5NG

RMSProp + Momentum ("Almost" Adam)

```
moment1 = 0
moment2 = 0
for t in range(1, num_steps + 1):  # Start at t = 1
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  moment1_unbias = moment1 / (1 - beta1 ** t)
  moment2_unbias = moment2 / (1 - beta2 ** t)
  w -= learning_rate * moment1_unbias / (moment2_unbias.sqrt() + 1e-7)
```

Momentum

AdaGrad / RMSProp

Bias correction

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3, 5e-4, 1e-4 is a great starting point for many models!

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

Adam: Very common in practice!

for input to the CNN; each colored pixel in the image yields a 7D one-hot vector. Following common practice, the network is trained end-to-end using stochastic gradient descent with the Adam optimizer [22]. We anneal the learning rate to 0 using a half cosine schedule without restarts [28].

Bakhtin, van der Maaten, Johnson, Gustafson, and Girshick, NeurIPS 2019

We train all models using Adam [23] with learning rate 10^{-4} and batch size 32 for 1 million iterations; training takes about 3 days on a single Tesla P100. For each mini-batch we first update f, then update D_{img} and D_{obj}.

Johnson, Gupta, and Fei-Fei, CVPR 2018

ganized into three residual blocks. We train for 25 epochs using Adam [27] with learning rate 10^{-4} and 32 images per batch on 8 Tesla V100 GPUs. We set the `cubify` thresh-

Gkioxari, Malik, and Johnson, ICCV 2019

sampled with each bit drawn uniformly at random. For gradient descent, we use Adam [29] with a learning rate of 10^{-3} and default hyperparameters. All models are trained with batch size 12. Models are trained for 200 epochs, or 400 epochs if being trained on multiple noise layers.

Zhu, Kaplan, Johnson, and Fei-Fei, ECCV 2018

16 dimensional vectors. We iteratively train the Generator and Discriminator with a batch size of 64 for 200 epochs using Adam [22] with an initial learning rate of 0.001.

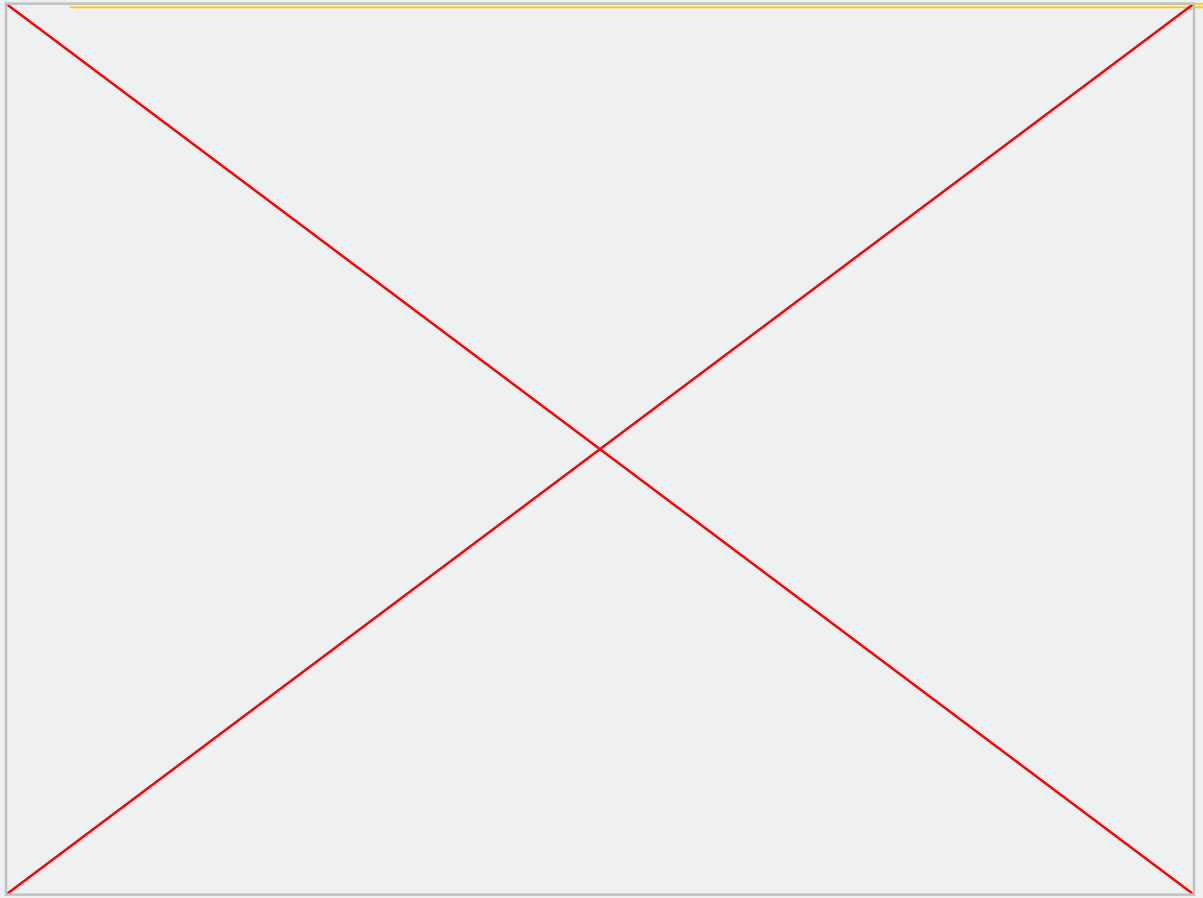
Gupta, Johnson, et al, CVPR 2018

➡ Adam with beta1 = 0.9,

beta2 = 0.999, and learning_rate = 1e-3, 5e-4, 1e-4 is a great starting point for many models!

M | ROBOTICS

Adam: Very common in practice!



Additional References:

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Optimization Algorithms Comparison

Algorithm	Tracks first moments (Momentum)	Tracks second moments (Adaptive learning rates)	Leaky second moments	Bias correction for moment estimates
SGD	✗	✗	✗	✗
SGD+Momentum	✓	✗	✗	✗
Nesterov	✓	✗	✗	✗
AdaGrad	✗	✓	✗	✗
RMSProp	✗	✓	✓	✗
Adam	✓	✓	✓	✓

L2 Regularization vs. Weight Decay

Optimization Algorithm

$$L(w) = L_{data}(w) + L_{reg}(w)$$
$$g_t = \nabla L(w_t)$$
$$s_t = optimizer(g_t)$$
$$w_{t+1} = w_t - \alpha s_t$$

L2 Regularization and Weight Decay are equivalent for SGD, SGD+Momentum so people often use the terms interchangeably!

But they are not the same for adaptive methods (AdaGrad, RMSProp, Adam, etc)

Loshchilov and Hunter, "Decoupled Weight Decay Regularization," ICLR 2019

L2 Regularization

$$L(w) = L_{data}(w) + \lambda |w|^2$$
$$g_t = \nabla L(w_t) = \nabla L_{data}(w_t) + 2\lambda w_t$$
$$s_t = optimizer(g_t)$$
$$w_{t+1} = w_t - \alpha s_t$$

Weight decay

$$L(w) = L_{data}(w)$$
$$g_t = \nabla L_{data}(w_t)$$
$$s_t = optimizer(g_t) + 2\lambda w_t$$
$$w_{t+1} = w_t - \alpha s_t$$

M | ROBOTICS

AdamW: Decouple Weight Decay

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Loshchilov and Hunter, "Decoupled Weight Decay Regularization," ICLR 2019

AdamW: Decouple Weight Decay

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

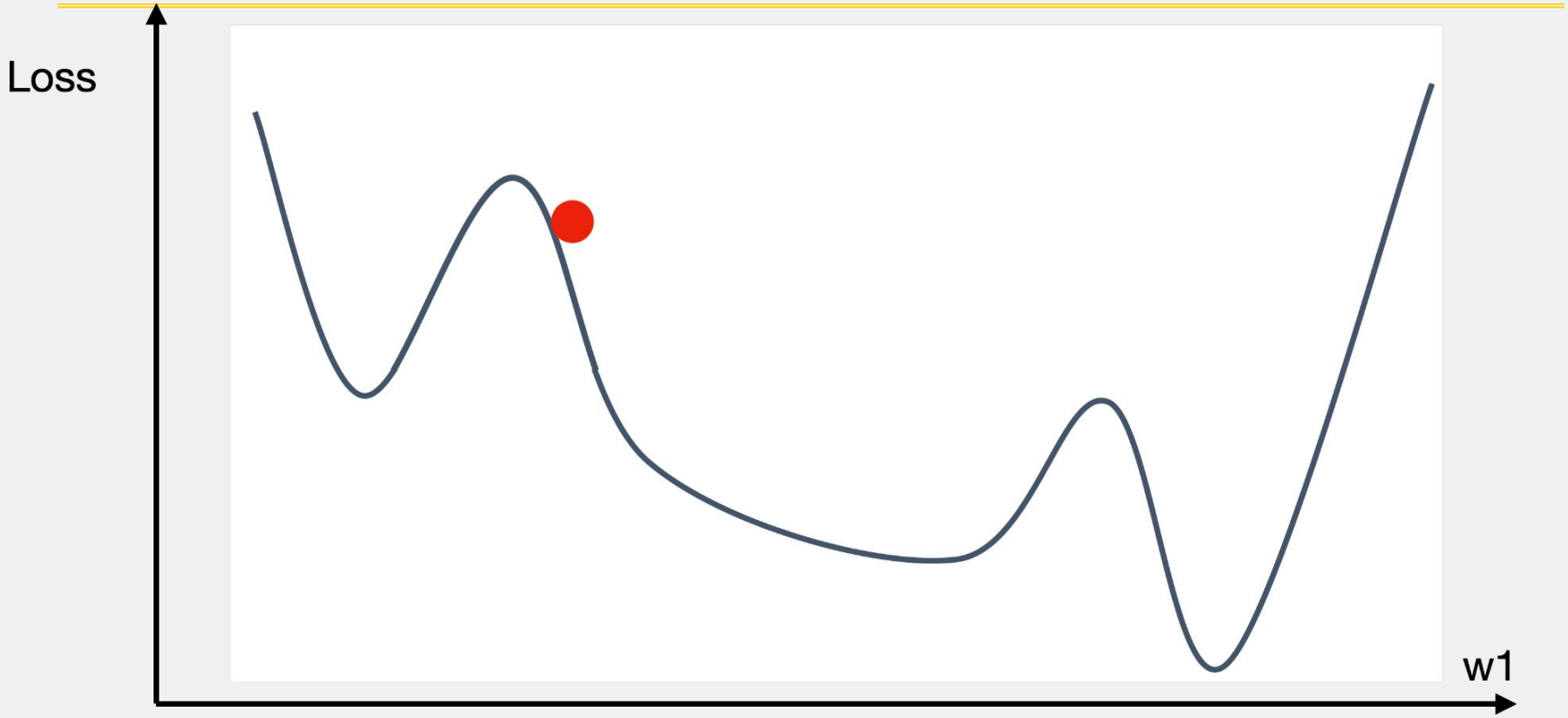
1: **given** $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
2: **initialize** time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3:
4:
5:
6:
7:
8:
9:
10:
11: $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
12: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
13: **until** *stopping criterion is met*
14: **return** optimized parameters $\boldsymbol{\theta}_t$

AdamW should/could probably be your "default" optimizer for new problems

Loshchilov and Hunter, "Decoupled Weight Decay Regularization," ICLR 2019

Second-Order Optimization

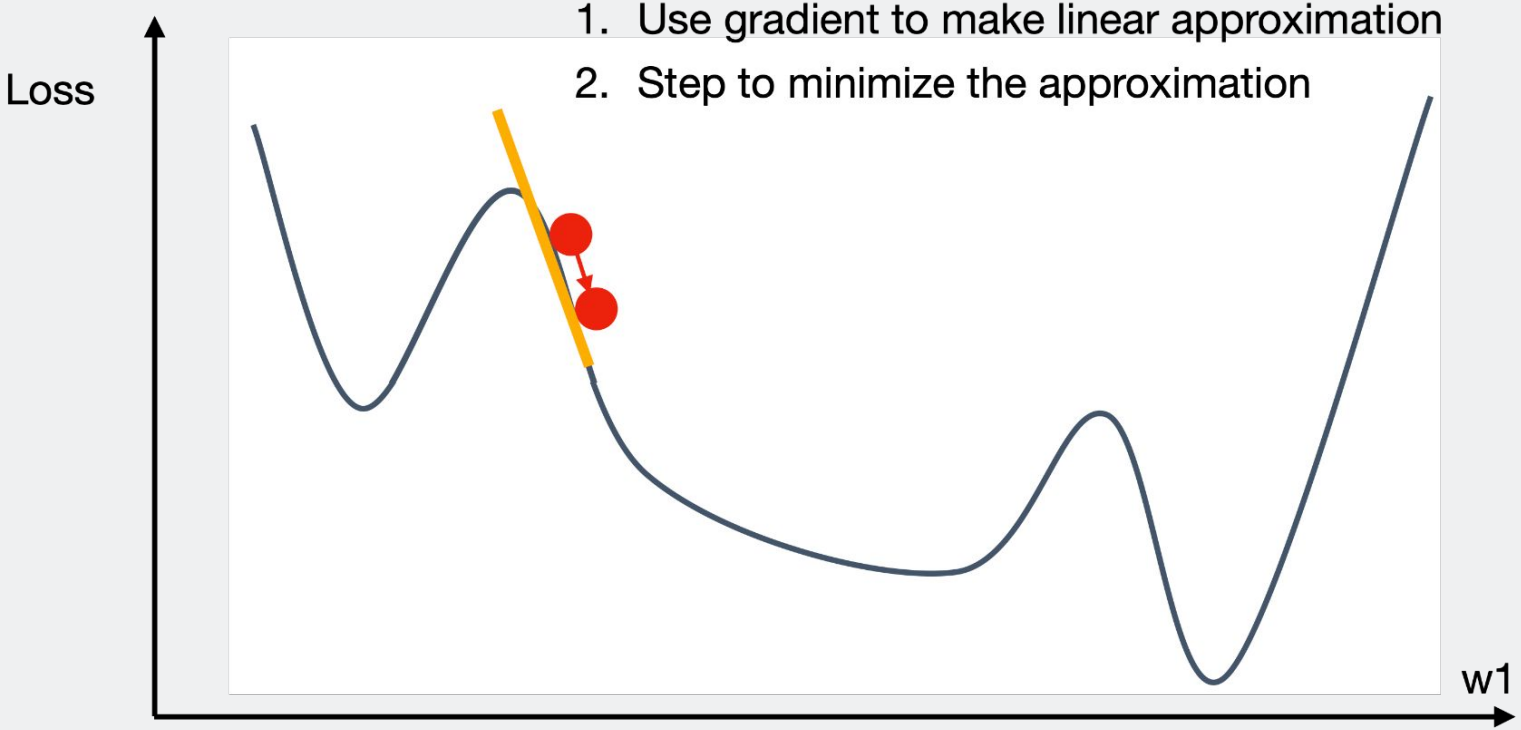
So Far: First-Order Optimization



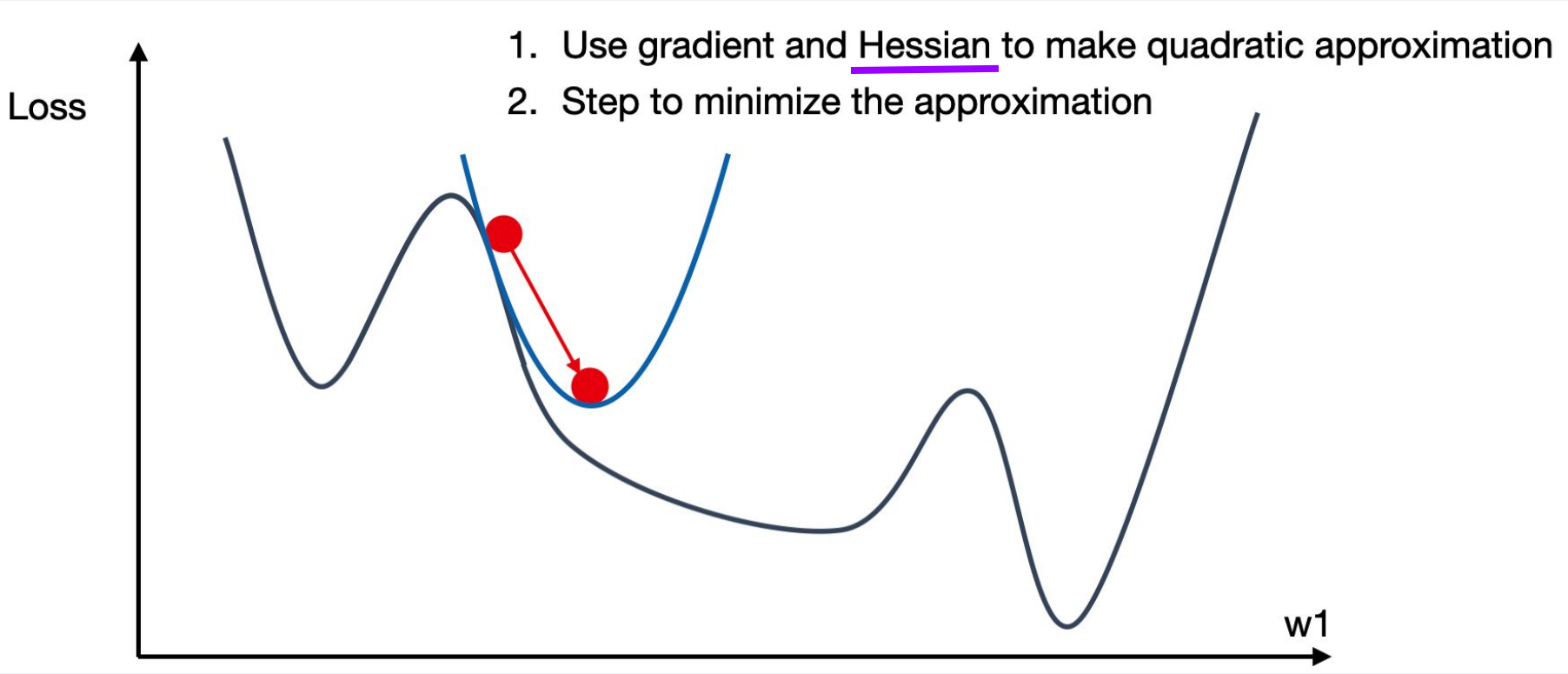
ROBOTICS

So Far: First-Order Optimization

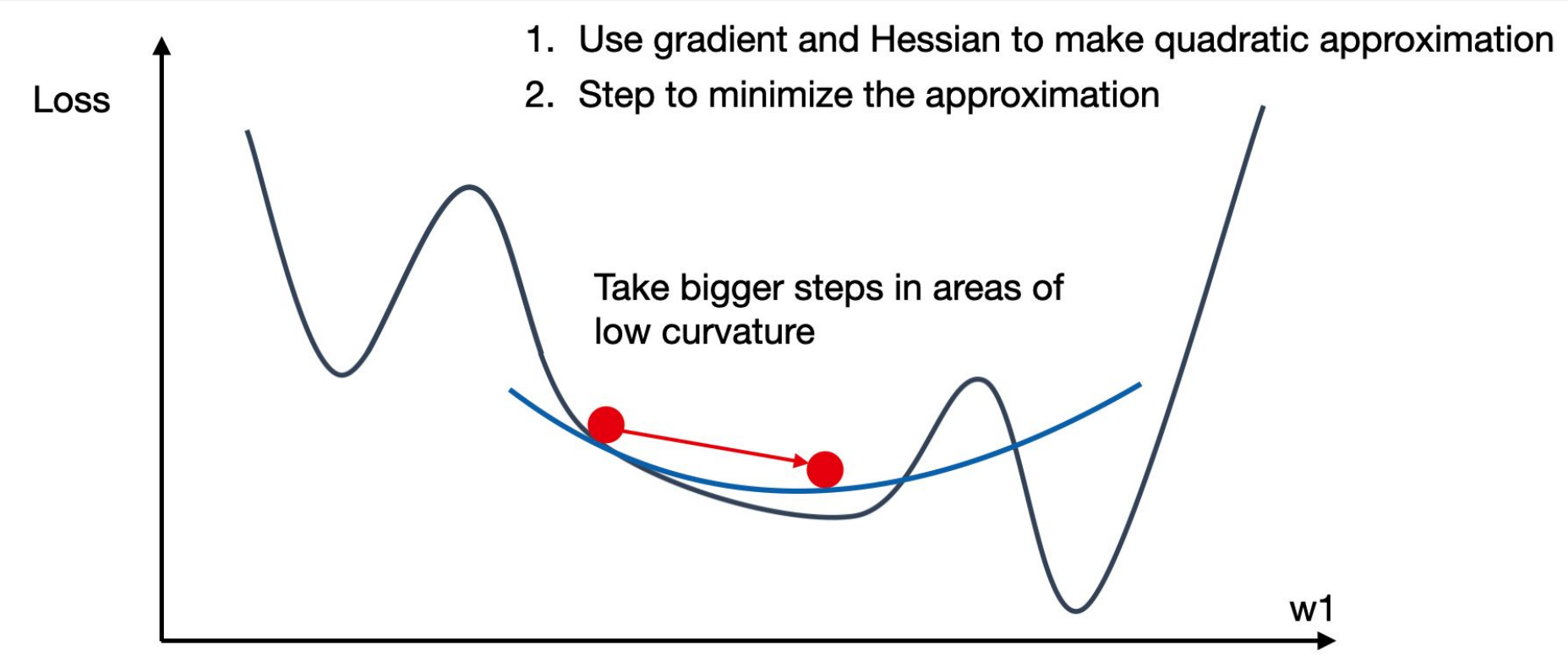
1. Use gradient to make linear approximation
2. Step to minimize the approximation



Second-Order Optimization



Second-Order Optimization



Second-Order Optimization

Second-order Taylor Expansion:

$$L(w) \approx L(w_0) + (w - w_0)^T \nabla_w L(w_0) + \frac{1}{2}(w - w_0)^T H_w L(w_0)(w - w_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$w^* = w_0 - \mathbf{H}_w L(w_0)^{-1} \nabla_w L(w_0)$$

Q: Why is this impractical?

Hessian has O(N^2) elements

Inverting takes O(N^3)

N = (Tens or Hundreds of) Millions

Second-Order Optimization

$$w^* = w_0 - \mathbf{H}_w L(w_0)^{-1} \nabla_w L(w_0)$$

- Quasi-Newton methods (BGFS most popular): *instead of inverting the Hessian ((O(n^3)), approximate inverse Hessian with rank 1 updates over time (O(n^2) each).*

- **L-BFGS** (Limited memory BFGS): *Does not form/store the full inverse Hessian*

Second-Order Optimization: L-BFGS

- **Usually works very well in full batch, deterministic mode** i.e. if you have a single, deterministic f(x) then L-BFGS will probably work very nicely.

- **Does not transfer very well to mini-batch setting.** Gives bad results. Adapting second-order methods to large-scale, stochastic setting is an active area of research.

Le et al, “On optimization methods for deep learning,” ICML 2011
Ba et al, “Distributed second-order optimization using Kronecker-factored approximations,” ICLR 2017

In-Practice (Take-aways)

- Adam/AdamW is a good default choice in many cases. SGD+Momentum can outperform Adam but may require more tuning.
- If you can afford to do full batch updates then try out L-BFGS (and don't forget to disable all sources of noise)

State-of-the-Art

(2024 ICLR accepted papers - example)

- Large Language Models as Optimizers

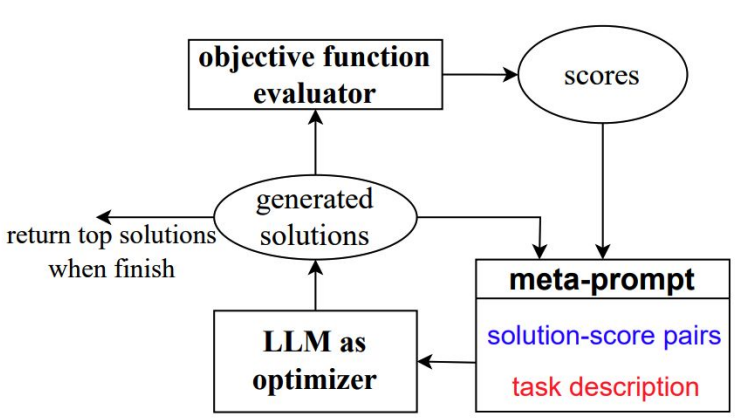


Table 1: Top instructions with the highest GSM8K zero-shot test accuracies from prompt optimization with different optimizer LLMs. All results use the pre-trained `PaLM 2-L` as the scorer.

Source	Instruction	Acc
Baselines		
(Kojima et al., 2022)	Let's think step by step.	71.8
(Zhou et al., 2022b)	Let's work this out in a step by step way to be sure we have the right answer.	58.8
	(empty string)	34.0
Ours		
`PaLM 2-L-IT`	Take a deep breath and work on this problem step-by-step.	**80.2**
`PaLM 2-L`	Break this down.	79.9
`gpt-3.5-turbo`	A little bit of arithmetic and a logical approach will help us quickly arrive at the solution to this problem.	78.5
`gpt-4`	Let's combine our numerical command and clear thinking to quickly and accurately decipher the answer.	74.5

"meta-prompt"

Large Language Models as Optimizers. Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, Xinyun Chen. https://iclr.cc/virtual/2024/poster/19209

M | ROBOTICS

State-of-the-Art

(2024 ICLR accepted papers - example)

- Neural Topic Modeling as Multi-Objective Contrastive Optimization

Table 1: Illustration of the effects of peculiar words on topic representations. We record the cosine similarity of the *input* and the *document instances*' topic representations generated by NTM+CL (Nguyen & Luu, 2021) and our neural topic model. As shown with underlined numbers, inserting unusual term, such as *zeppelin* or *scardino*, unexpectedly raises the similarity of the input with the document instance, even surpassing the similarity of the semantically close pair.

Input	Document Instance	Cosine Similarity	
		NTM+CL	Our Model
shuttle lands on planet	job career ask development	0.0093	0.0026
	star astronaut planet light moon	0.8895	0.9178
shuttle lands on planet *zeppelin/scardino*	job career ask development *zeppelin/scardino*	0.9741/0.9413	0.0064/0.0080
	star astronaut planet light moon	0.1584/0.2547	0.8268/0.7188

https://openreview.net/pdf?id=HdAoLSBYXj

M | ROBOTICS

State-of-the-Art

(2024 ICLR accepted papers - example)

- Neural Topic Modeling as Multi-Objective Contrastive Optimization

$$\min_{\alpha} \left\{ \left\| \alpha \nabla_\theta \mathcal{L}_{\text{InfoNCE}}(\theta) + (1-\alpha) \nabla_\theta \mathcal{L}_{\text{ELBO}}(\theta, \phi) \right\|_2^2 \middle| \alpha \geq 0 \right\}$$

$$f(\mathbf{x}, \mathbf{y}) = \frac{g_\varphi(\mathbf{x})^T g_\varphi(\mathbf{y})}{\| g_\varphi(\mathbf{x}) \| \| g_\varphi(\mathbf{y}) \|} / \tau$$

$$\min_{\theta,\phi} \mathcal{L}_{\text{ELBO}} = -\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})} \left[\log p_\phi(\mathbf{x}|\mathbf{z}) \right] + \mathbb{KL} \left[q_\theta(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}) \right]$$

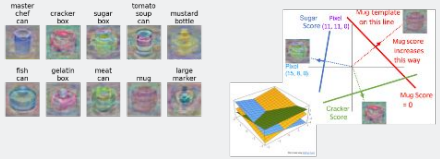
https://openreview.net/pdf?id=HdAoLSBYXj

M | ROBOTICS

Summary

- Use **Linear Models** for image classification problems.
- Use **Loss Functions** to express preferences over different choices of weights.
- Use **Regularization** to prevent overfitting to training data.
- Use **Stochastic Gradient Descent** to minimize our loss functions and train the model.

$$s = f(x; W) = Wx$$

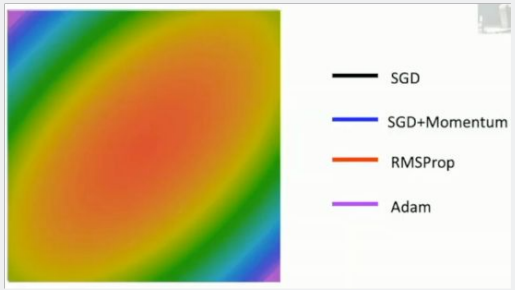


$$L_i = -\log\left(\frac{\exp^{s_{y_i}}}{\sum_i \exp^{s_j}}\right)$$ **Softmax**

$$L_i = \sum_{j \neq y_i} \max(0, s_j = -s_{y_i} + 1)$$ **SVM**

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```



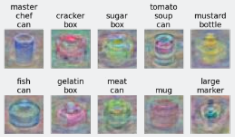
Next up: Neural Networks

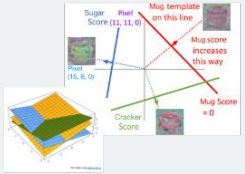
Recap

P1 Deadline: Feb. 2, 2025

- Use **Linear Models** for image classification problems.
- Use **Loss Functions** to express preferences over different choices of weights.
- Use **Regularization** to prevent overfitting to training data.
- Use **Stochastic Gradient Descent** to minimize our loss functions and train the model.

$$s = f(x; W) = Wx$$



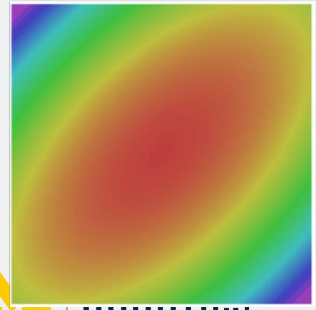


$$L_i = -\log\left(\frac{\exp^{s_{y_i}}}{\sum_i \exp^{s_j}}\right)$$ **Softmax**

$$L_i = \sum_{j \neq y_i} \max(0, s_j = -s_{y_i} + 1)$$ **SVM**

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$

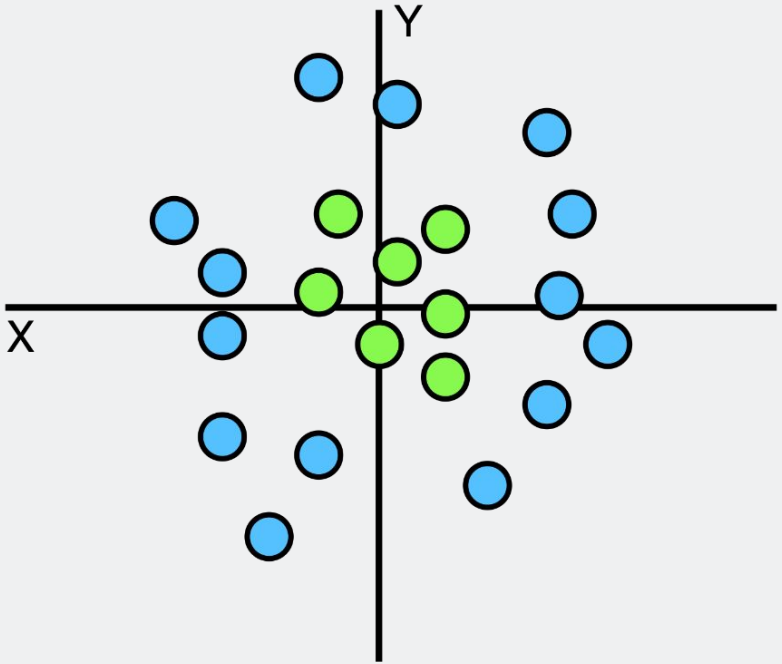
```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```



Neural Networks

Problem: Linear Classifiers aren't that powerful

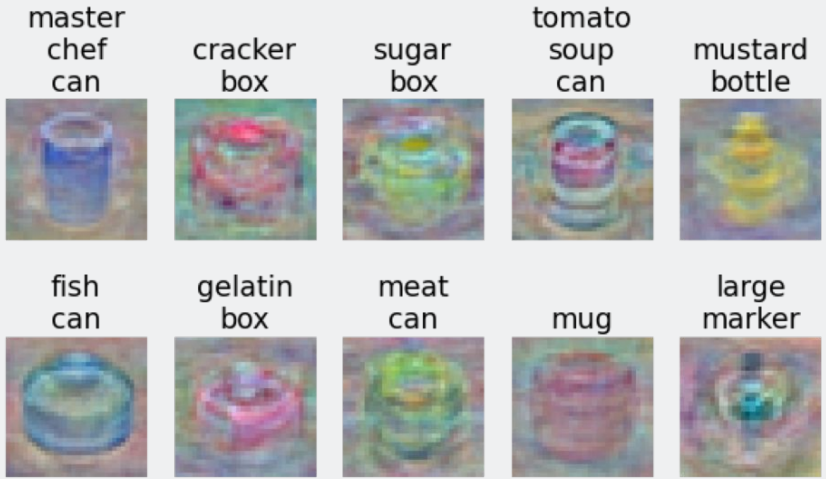
Geometric Viewpoint



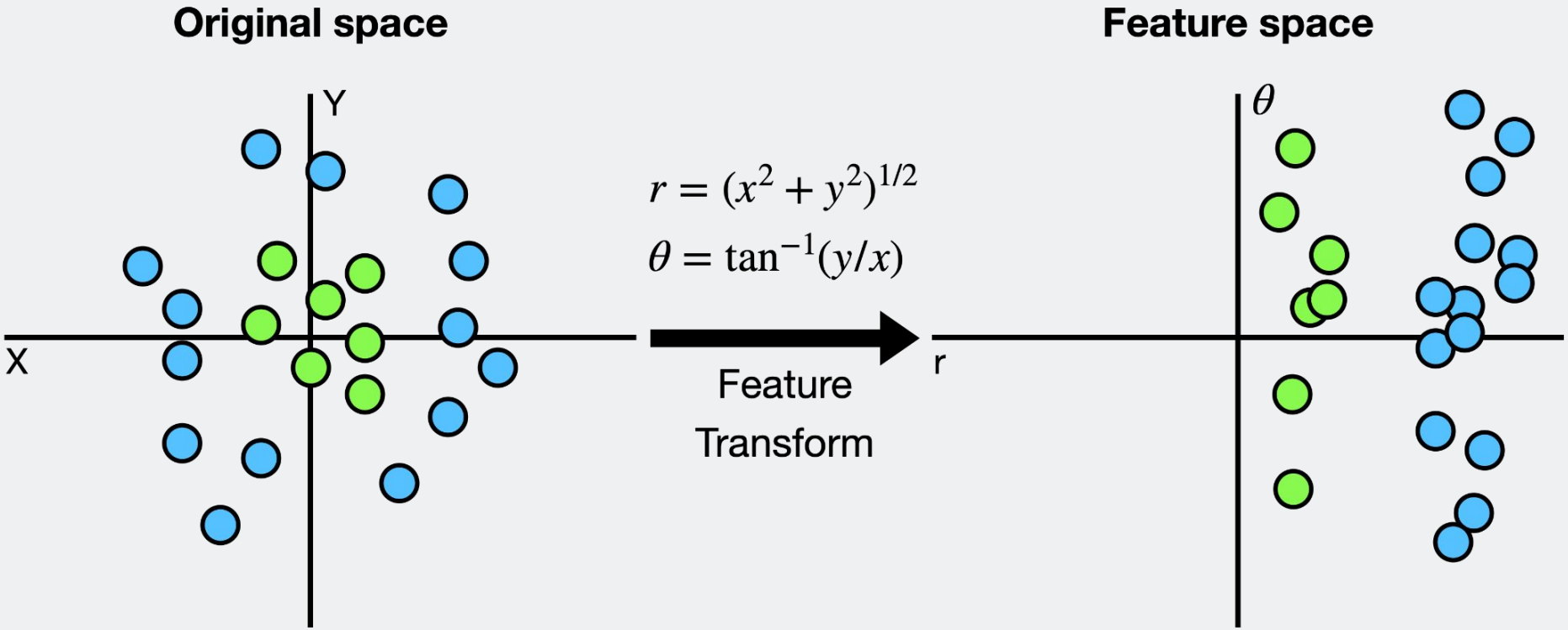
Visual Viewpoint

One template per class:

Can't recognize different modes of a class

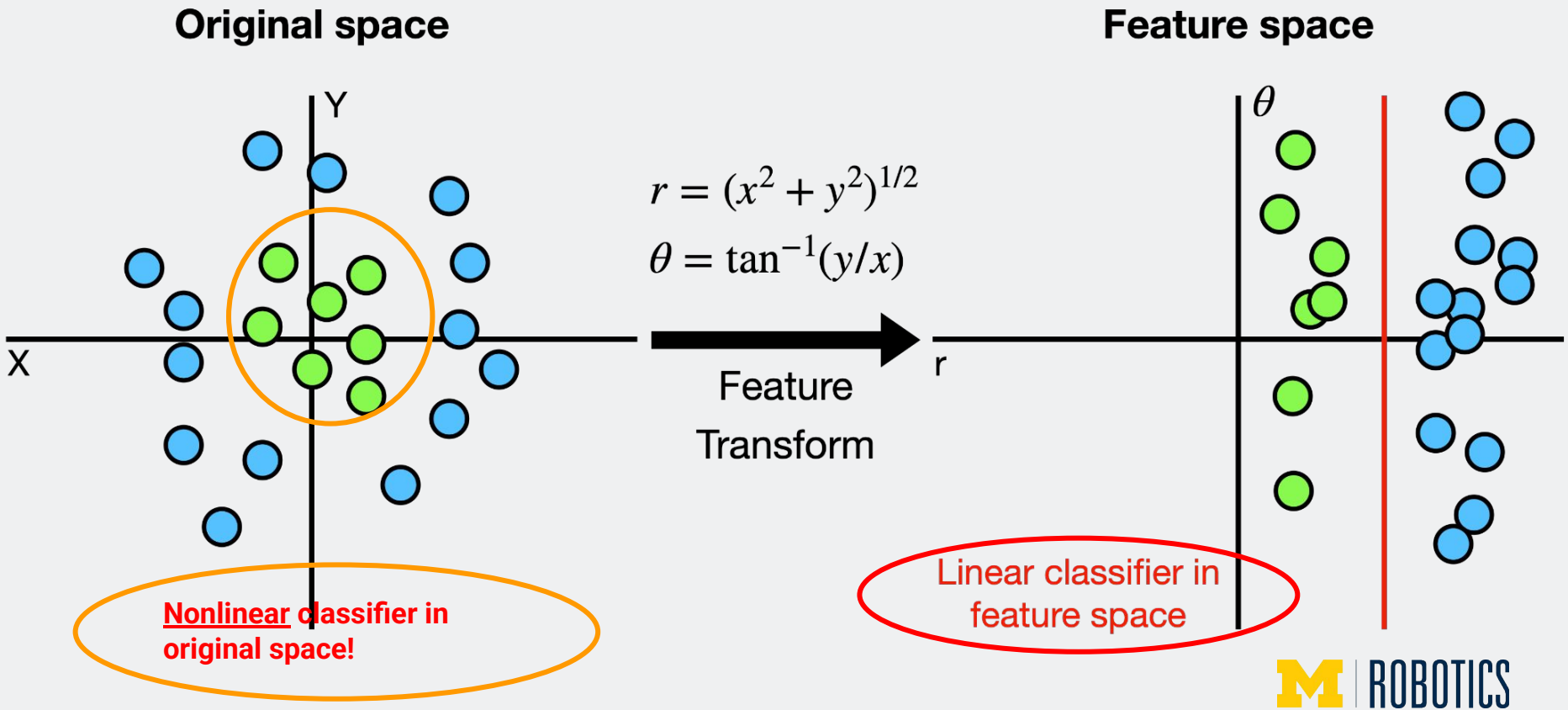


One Solution: Feature transforms



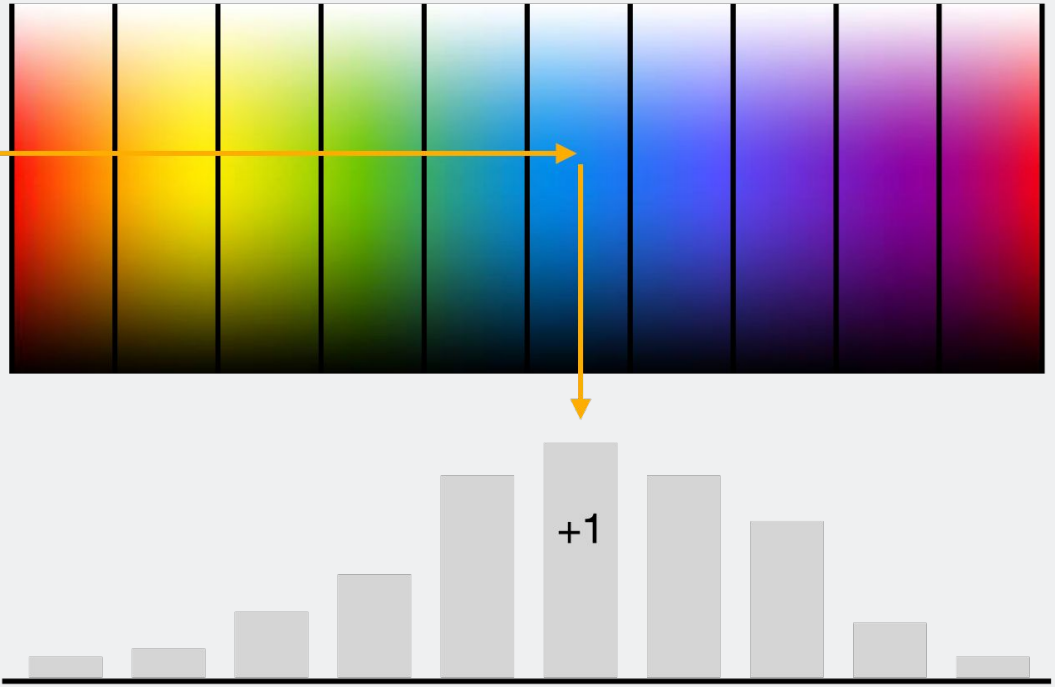
M | ROBOTICS

One Solution: Feature transforms



M | ROBOTICS

Image Feature: Color Histogram



Ignores texture,
spatial positions

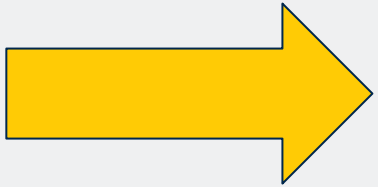
Frog image is in the public domain

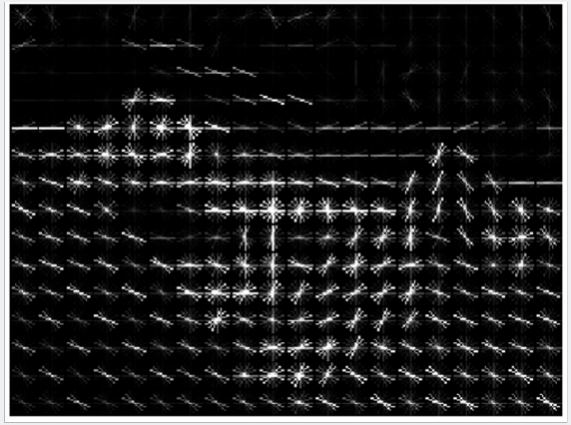
Image Feature: HoG (Histogram of Oriented Gradients)

1. Compute edge direction/ strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge direction weighted by edge strength

Lowe, “Object recognition from local scale-invariant features,” ICCV 1999
Dalal and Triggs, “Histograms of oriented gradients for human detection,” CVPR 2005

Image Feature: HoG (Histogram of Oriented Gradients)





1. Compute edge direction/ strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge direction weighted by edge strength

Example: 320x240 image gets divided into 40x30 bins;

9 directions per bin;

feature vector has 30*40*9 = 10,800 numbers

Lowe, “Object recognition from local scale-invariant features,” ICCV 1999
Dalal and Triggs, “Histograms of oriented gradients for human detection,” CVPR 2005

Image Feature: HoG (Histogram of Oriented Gradients)

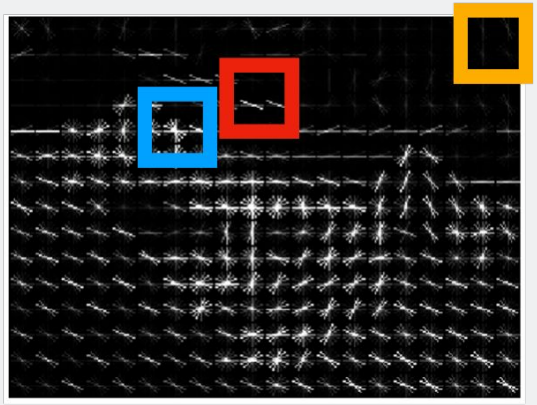
1. Compute edge direction/ strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge direction weighted by edge strength

Weak edges

Strong diagonal edges

Edges in all directions

Capture texture and position, robust to small image changes



Example: 320x240 image gets divided into 40x30 bins;

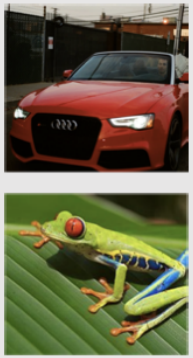
9 directions per bin;

feature vector has 30*40*9 = 10,800 numbers

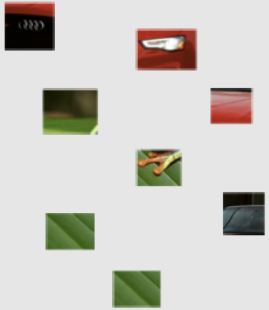
Lowe, “Object recognition from local scale-invariant features,” ICCV 1999
Dalal and Triggs, “Histograms of oriented gradients for human detection,” CVPR 2005

Image Feature: Bag of Words (Data Driven)

Step 1: Build codebook



Extract random patches



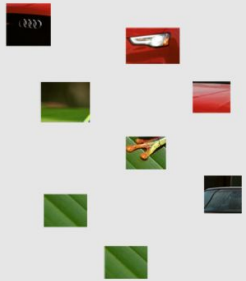
Cluster patches to form “codebook” of “visual words”

Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories,” CVPR 2005

Image Feature: Bag of Words (Data Driven)

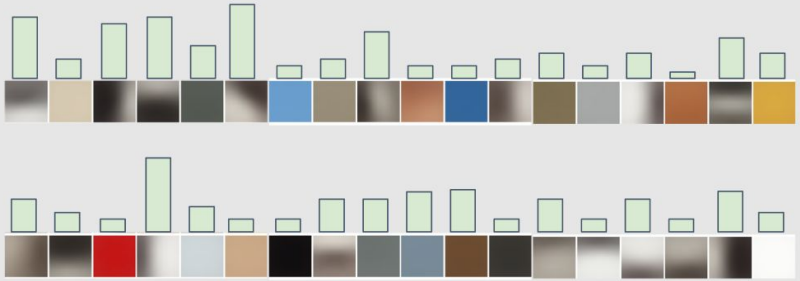
Step 1: Build codebook

Extract random patches



Cluster patches to form “codebook” of “visual words”

Step 2: Encode images



Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories,” CVPR 2005

M | ROBOTICS

Image Features

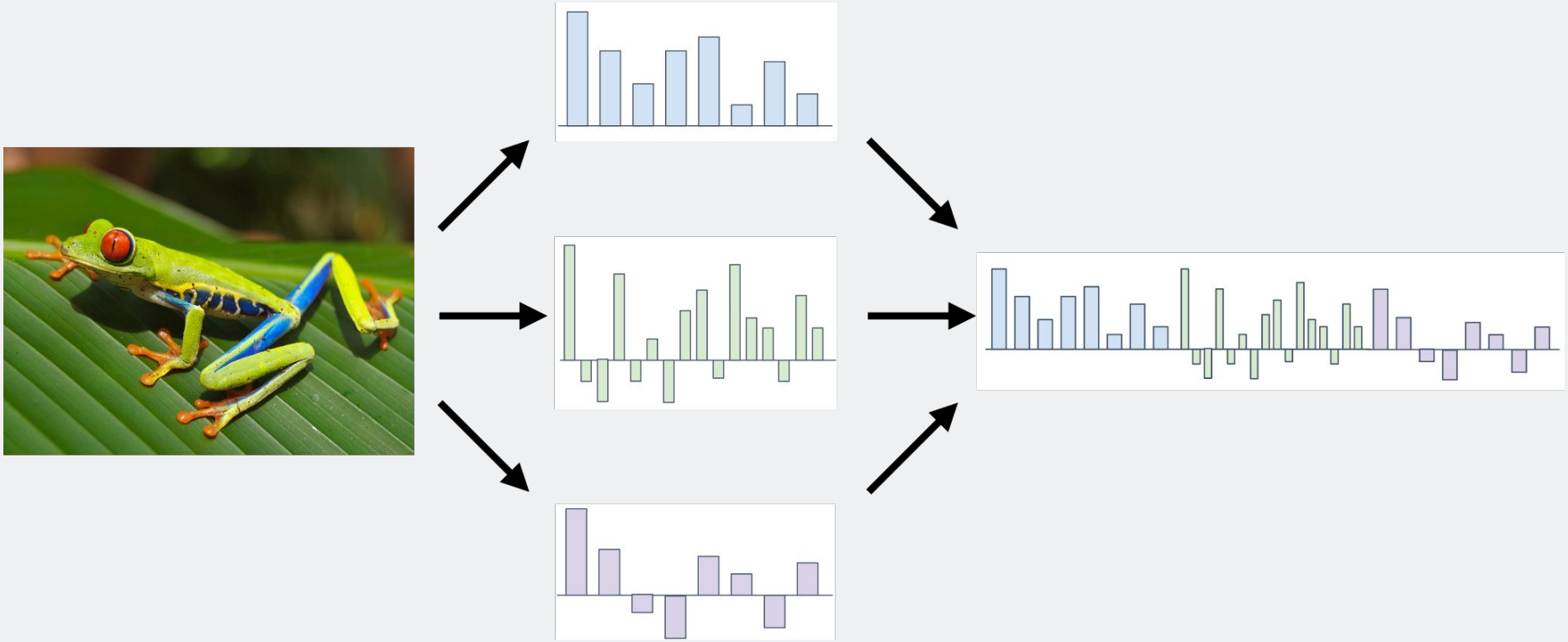
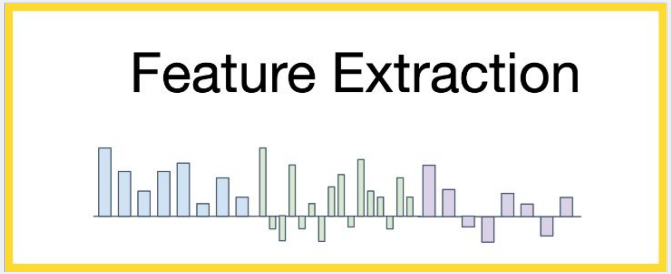
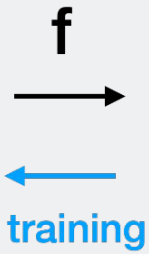


Image Features





10 numbers giving scores for classes

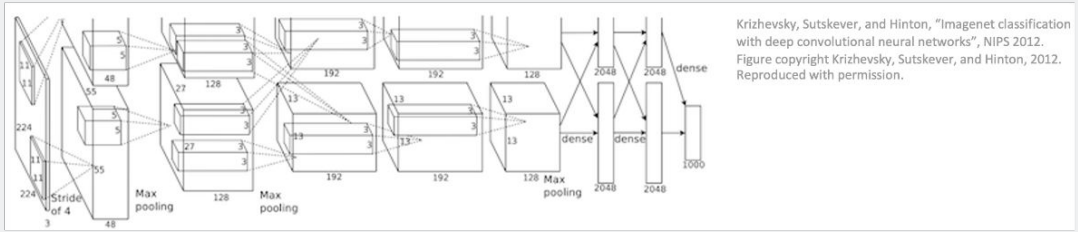
Image Features vs. Neural networks

Feature Extraction

f

training

10 numbers giving scores for classes



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

10 numbers giving scores for classes

training

M | ROBOTICS

Example: Winner of 2011 ImageNet Challenge

Low-level feature extraction ≈ 10k patches per image

- SIFT: 128-dims
- Color: 96-dim

} Reduced to 64-dim with PCA

FV extraction and compression:

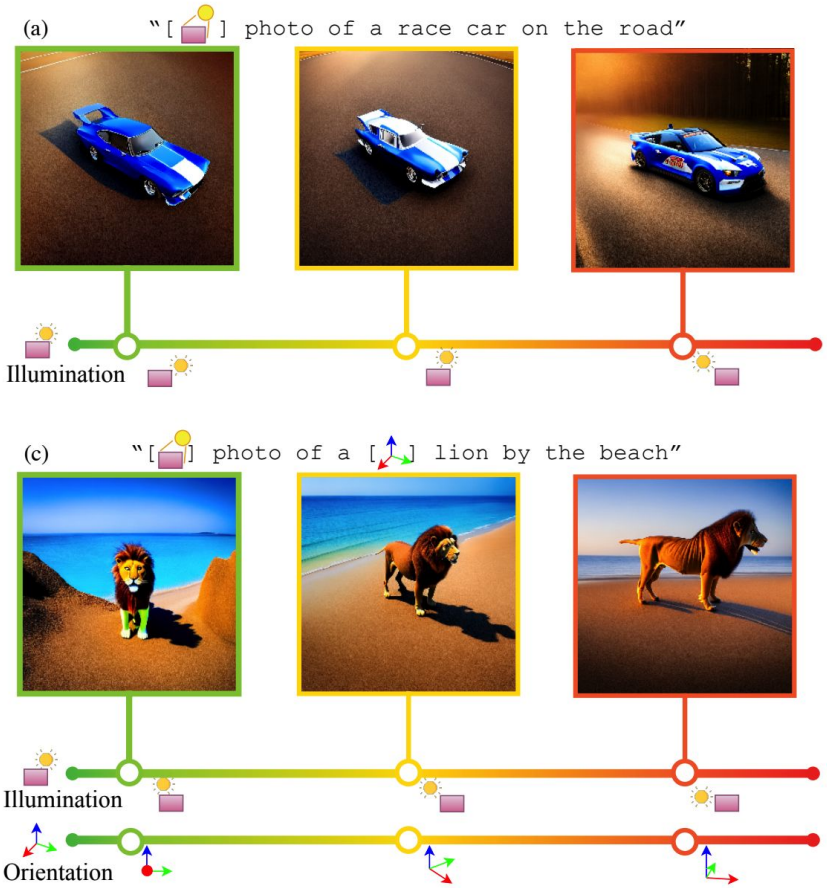
- N=1024 Gaussians, R=4 regions → 520K dim x 2
- Compression: G=8, b=1 bit per dimension

One-vs-all SVM learning with SGD

Late fusion of SIFT and color systems

F. Perronnin, J. Sánchez, “Compressed Fisher vectors for LSVRC”, PASCAL VOC / ImageNet workshop, ICCV, 2011.

Example: 2024 CVPR accepted paper



"Text-to-image" 3D Words

Cheng, T. Y., Gadelha, M., Groueix, T., Fisher, M., Mech, R., Markham, A., & Trigoni, N. (2024). Learning Continuous 3D Words for Text-to-Image Generation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 6753-6762).

M | ROBOTICS

Example: 2024 CVPR accepted paper

Figure 1. Visual comparison of 500 superpixels resulting from (a, c) ETPS [previous], (b, d) HHTS [proposed] segmentation.

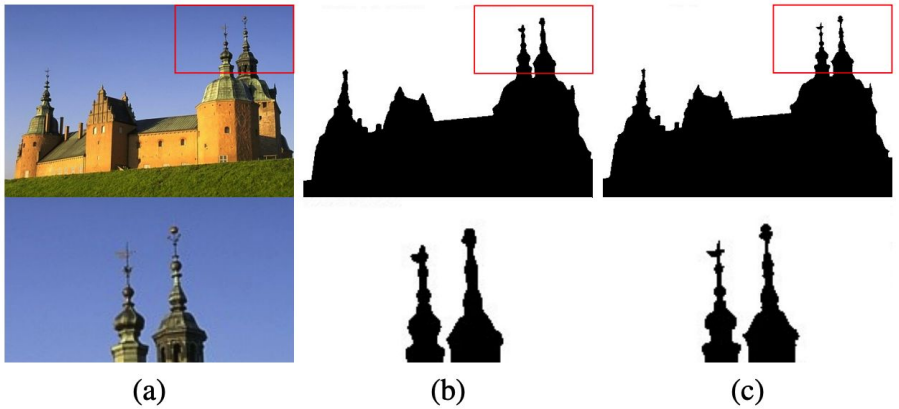


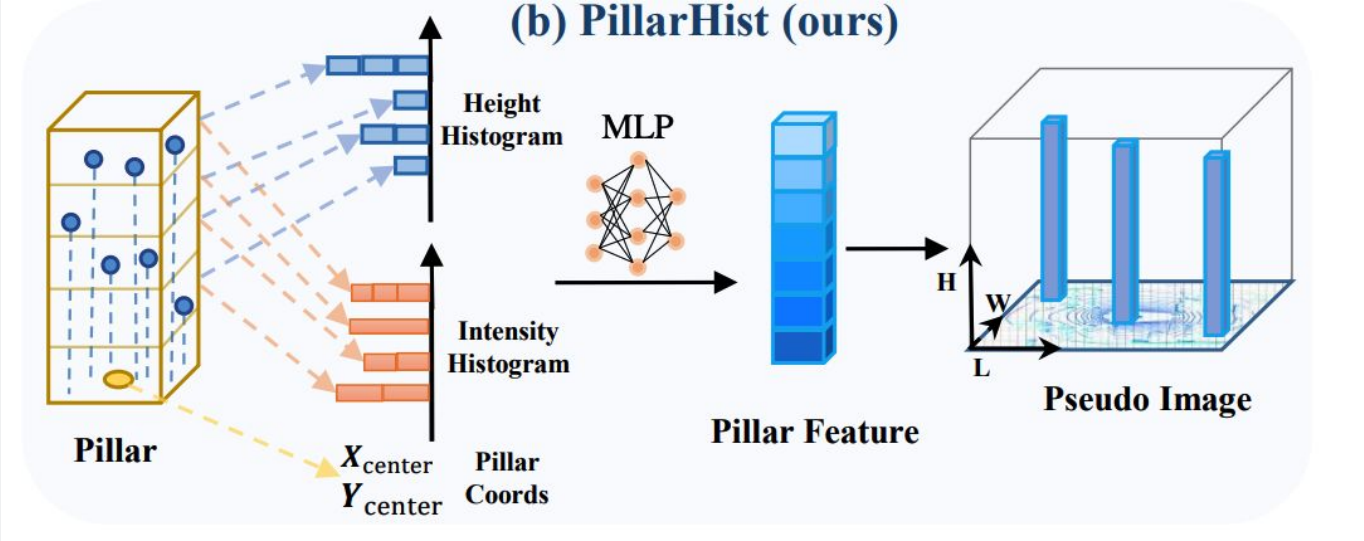
Figure 2. Visual comparison of semantic segment masks (a) original image, (b) semantic segment (SAM ViT-H) [previous] and (c) refined semantic segment (SAM + HHTS) [proposed]

- Hierarchical Histogram Threshold Segmentation
- “Fine-tuning” segmentation masks

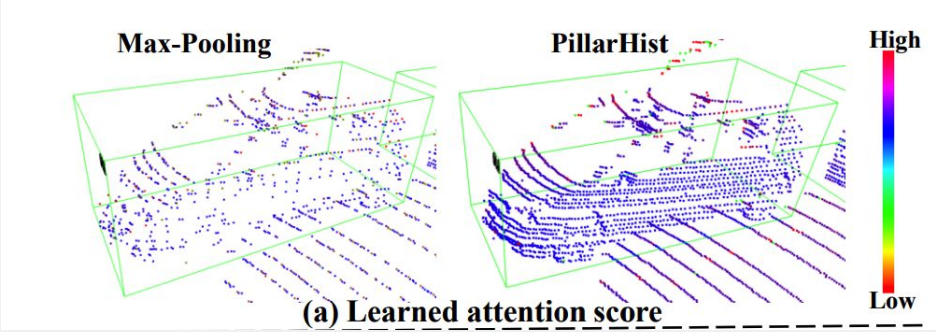
Chang, T. V., Seibt, S., & von Rymon Lipinski, B. (2024). Hierarchical Histogram Threshold Segmentation-Auto-terminating High-detail Oversegmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 3195-3204).

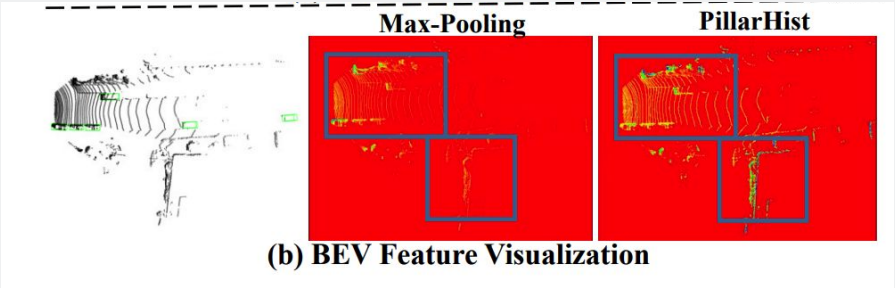
M | ROBOTICS

Example: 2025 CVPR accepted paper



Zhou, Sifan, et al. "Pillarhist: A quantization-aware pillar feature encoder based on height-aware histogram." Proceedings of the Computer Vision and Pattern Recognition Conference. 2025.





M | ROBOTICS

Example: World Labs (2025)

“Spatial Intelligence”

https://www.worldlabs.ai/

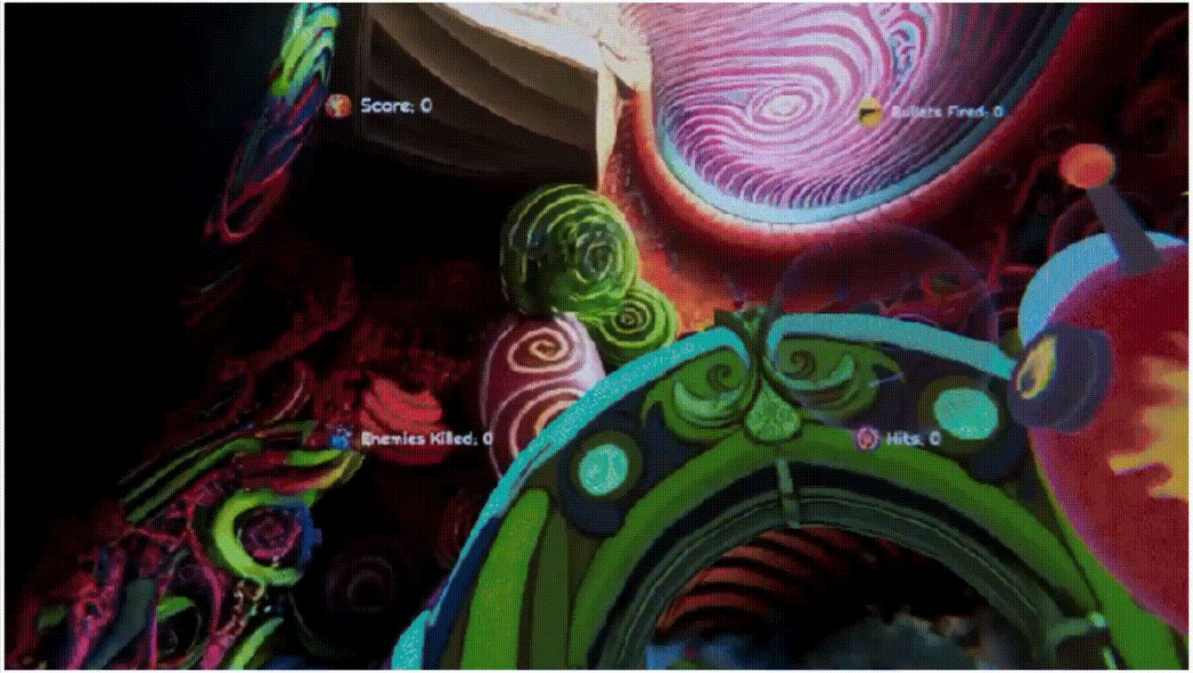


IMAGE CREDITS:WORLD LABS

Marble: https://marble.worldlabs.ai/?utm_source=media&utm_medium=referral&utm_campaign=marble_launch

News: https://techcrunch.com/2025/11/12/fei-fei-lis-world-labs-speeds-up-the-world-model-race-with-marble-its-first-commercial-product/

M | ROBOTICS

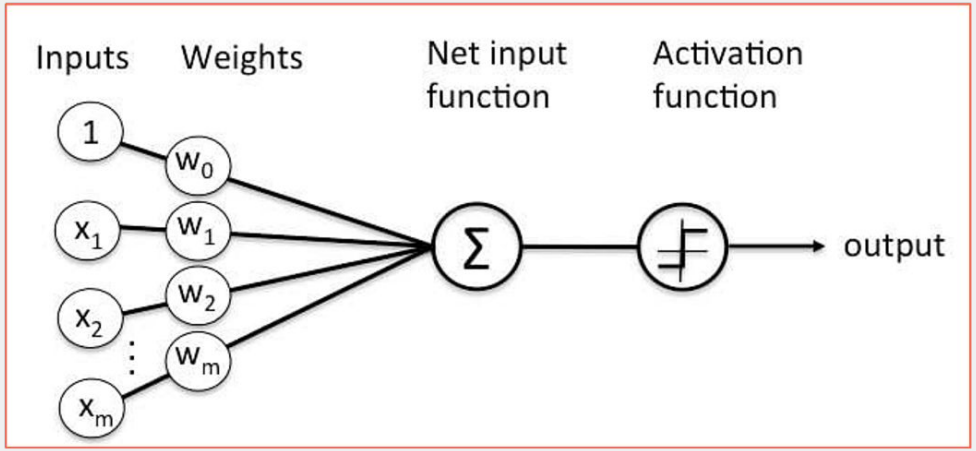
Neural Networks (Overview)

Input: $x \in \mathbb{R}^D$

Output: $f(x) \in \mathbb{R}^C$

Rosenblatt's Perceptron

- A set of *synapses* each of which is characterized by a *weight* (which includes a *bias*).
- An *adder*
- An *activation function* (e.g., Rectified Linear Unit/ReLU, Sigmoid function, etc.)



$$y_k = \phi\left(\sum_{j=1}^{m} w_{kj}x_j + b_k\right)$$

Neural Networks

Input: $x \in \mathbb{R}^D$ **Output:** $f(x) \in \mathbb{R}^C$

Before: Linear Classifier: $f(x) = Wx + b$

Learnable parameters: $W \in \mathbb{R}^{D \times C}, b \in \mathbb{R}^C$

Neural Networks

Input: $x \in \mathbb{R}^D$ **Output:** $f(x) \in \mathbb{R}^C$

Before: Linear Classifier: $f(x) = Wx + b$

Learnable parameters: $W \in \mathbb{R}^{D \times C}, b \in \mathbb{R}^C$

Feature Extraction

Linear Classifier

Now: Two-Layer Neural Network: $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

Learnable parameters: $W_1 \in \mathbb{R}^{H \times D}, b_1 \in \mathbb{R}^H, W_2 \in \mathbb{R}^{C \times H}, b_2 \in \mathbb{R}^C$

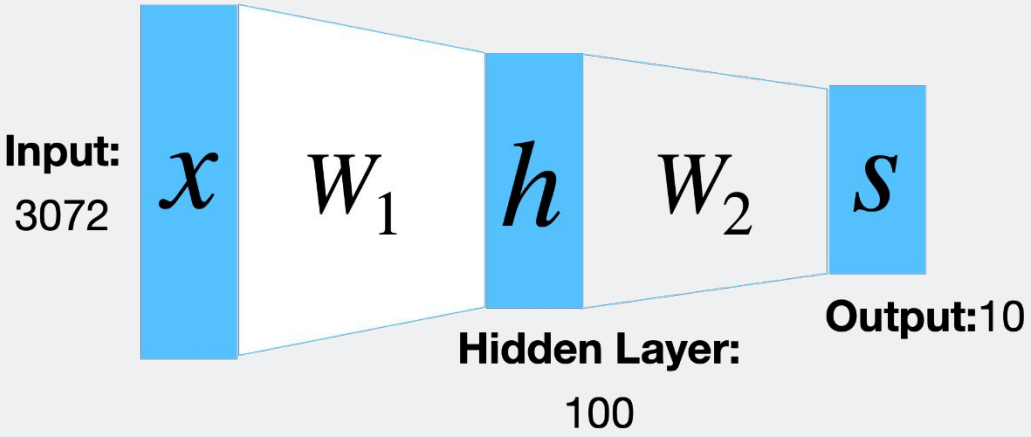
Or Three-Layer Neural Network:

$$f(x) = W_3 \max(0, W_2 \max(0, W_1 x + b_1) + b_2) + b_3$$

Neural Networks

Before: Linear Classifier: $f(x) = Wx + b$

➡ **Now:** Two-Layer Neural Network: $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

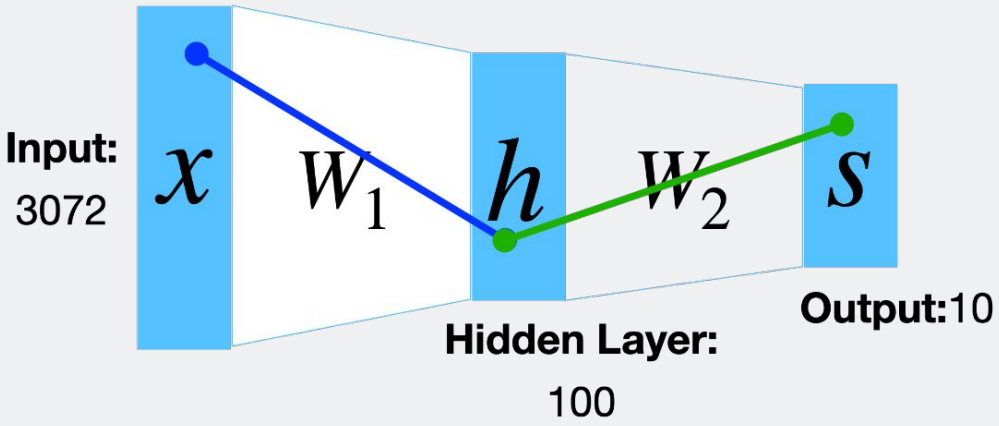
M | ROBOTICS

Neural Networks

Before: Linear Classifier: $f(x) = Wx + b$

Now: Two-Layer Neural Network: $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

Element (i, j) of W_1 gives the effect on h_i from x_j



Element (i, j) of W_2 gives the effect on s_i from h_j

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

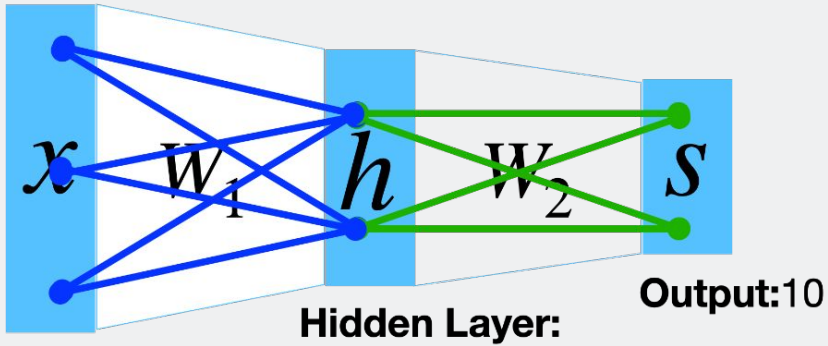
Neural Networks

Before: Linear Classifier: $f(x) = Wx + b$

Now: Two-Layer Neural Network: $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

Element (i,j) of W_1 gives the effect on h_i from x_j

All elements of x affect all elements of h



Element (i,j) of W_2 gives the effect on s_i from h_j

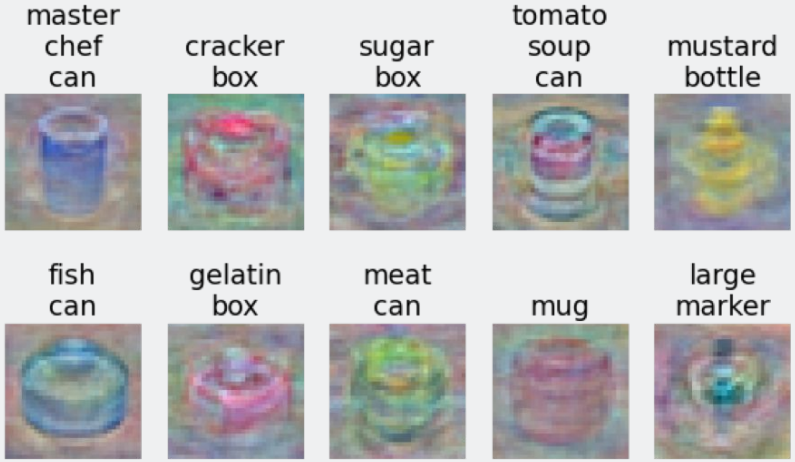
All elements of h affect all elements of s

Fully-connected neural network also "Multi-Layer Perceptron" (MLP)

Neural Networks

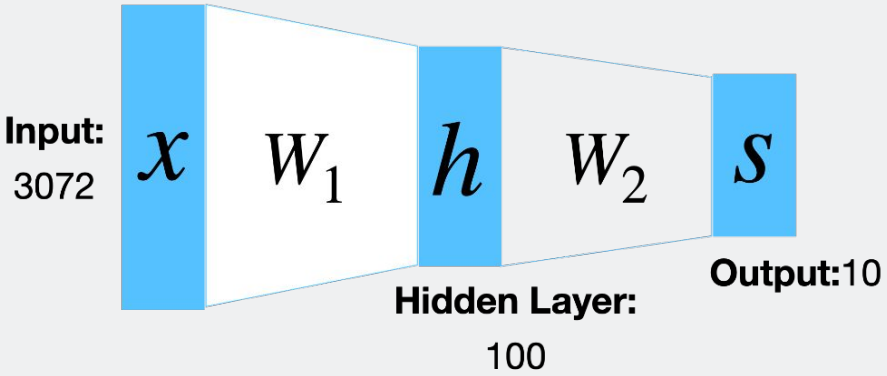
Recall:

Linear classifier: One template per class



Before: Linear score function

Now: Two-Layer Neural Network:



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

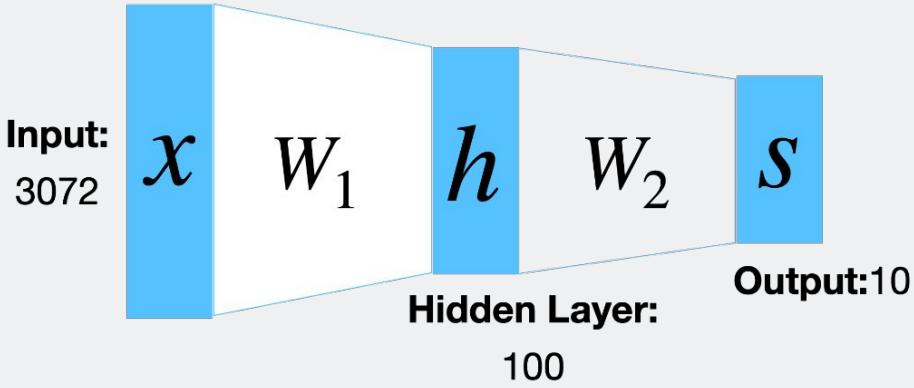
M | ROBOTICS

Neural Networks

Neural net: first layer is bank of templates;
Second layer recombines templates

Before: Linear score function

Now: Two-Layer Neural Network:



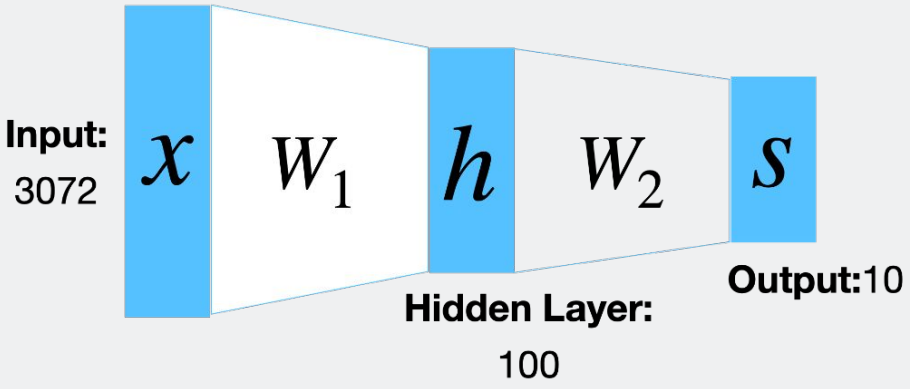
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Neural Networks

Can use different templates to cover multiple modes of a class!

Before: Linear score function

Now: Two-Layer Neural Network:



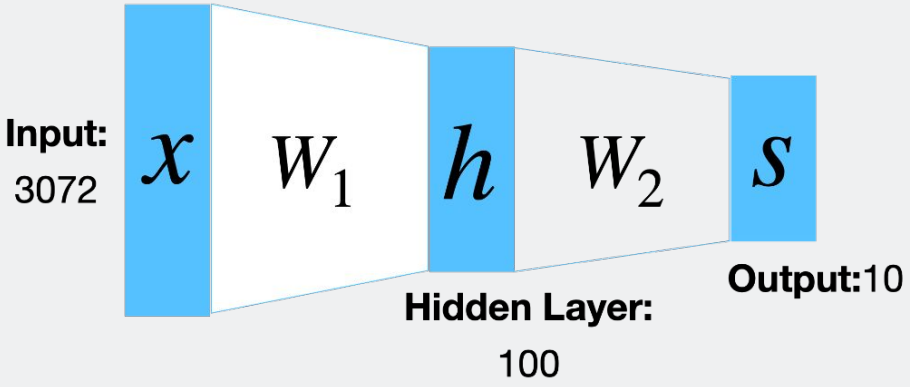
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Neural Networks

Can use different templates to cover multiple modes of a class!

Before: Linear score function

Now: Two-Layer Neural Network:



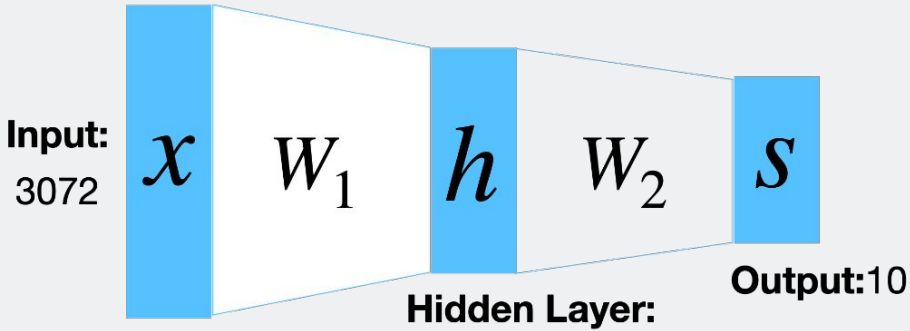
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Neural Networks

"Distributed representation": Most templates not interpretable!

Before: Linear score function

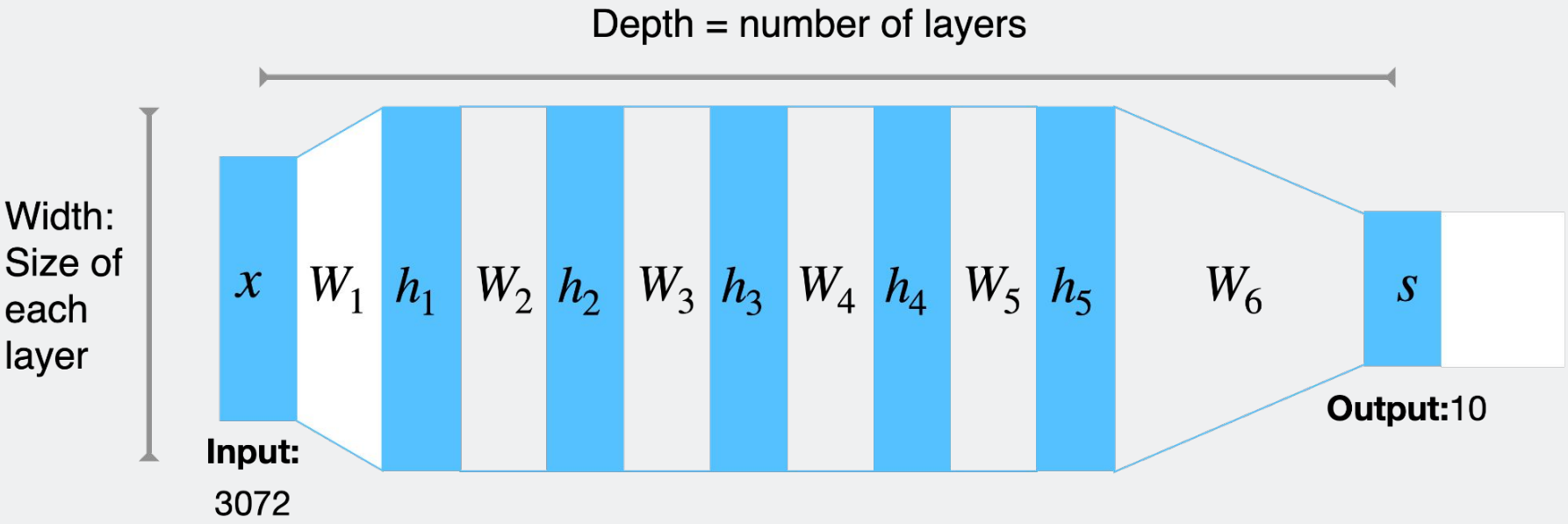
Now: Two-Layer Neural Network:



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

M | ROBOTICS

Deep Neural Networks



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

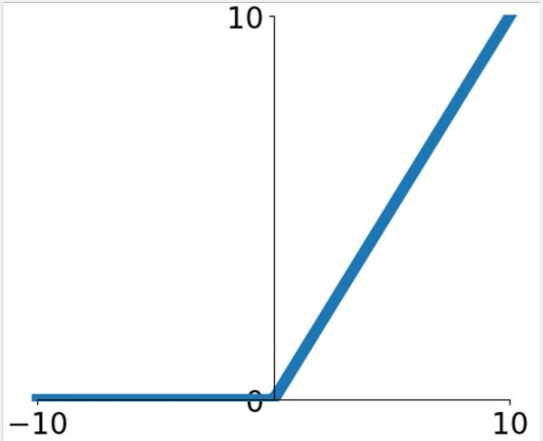
Neural Networks: Activation Functions

2-Layer Neural Network

$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$

The auction $ReLU(z) = \max(0, z)$ is called “Rectified Linear Unit”

This is called the **activation function** of the neural network



Q: What happens if we build a neural network with no activation function?

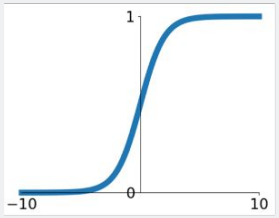
Aha Slides
(In-class participation)

https://ahaslides.com/TA5NG

Activation Functions

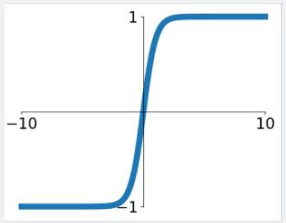
Sigmoid

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



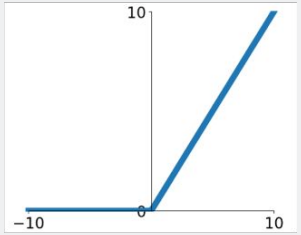
tanh

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



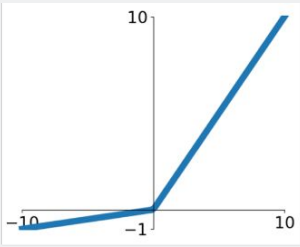
ReLU

$$\max(0,x)$$



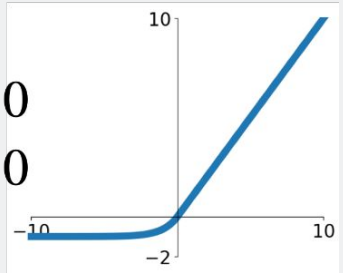
Leaky ReLU

$$\max(0.2x, x)$$



ELU

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) -), & x \leq 0 \end{cases}$$

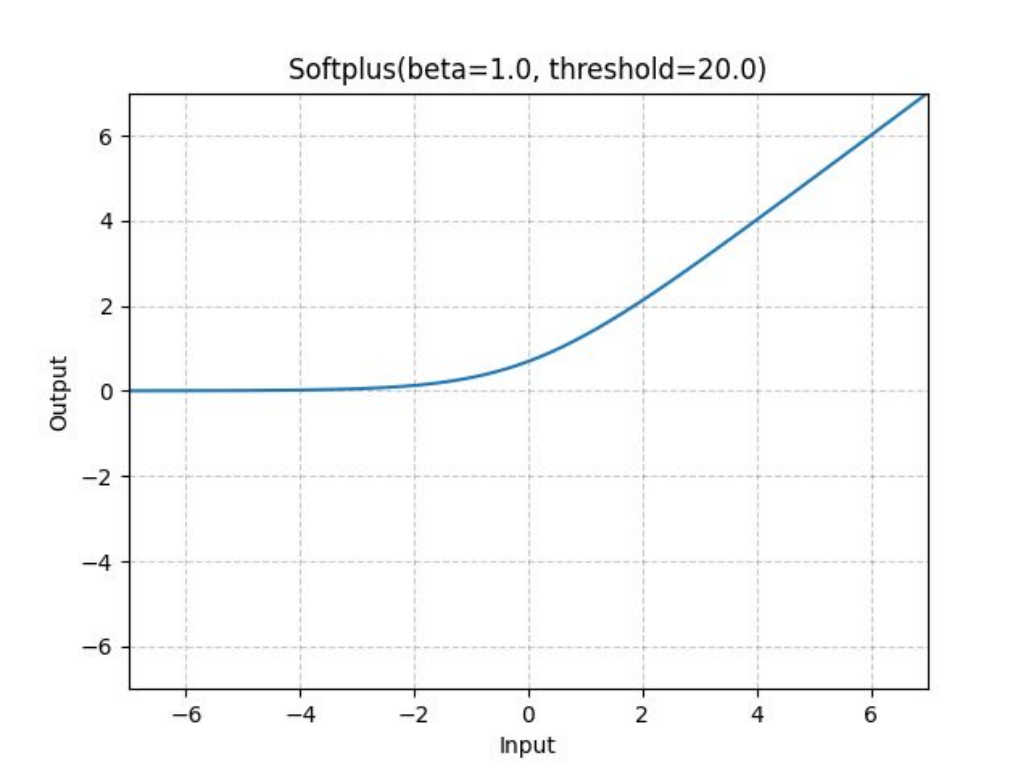


M | ROBOTICS

Activation Functions

Softplus

$$\log(1 + \exp(x))$$



https://pytorch.org/docs/stable/generated/torch.nn.Softplus.html

Activation Functions

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

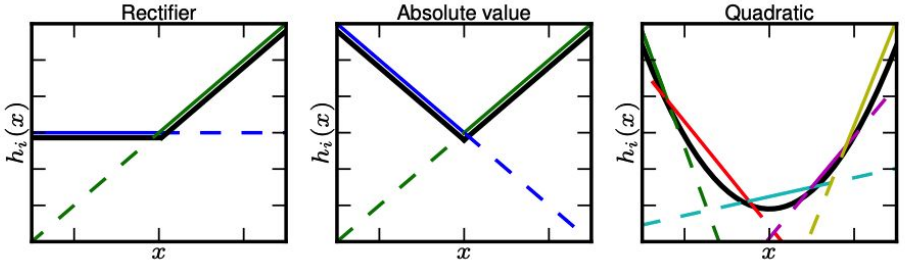


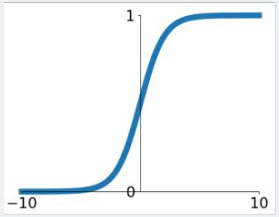
Figure 1. Graphical depiction of how the maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. This diagram is 2D and only shows how maxout behaves with a 1D input, but in multiple dimensions a maxout unit can approximate arbitrary convex functions.

https://proceedings.mlr.press/v28/goodfellow13.pdf

Activation Functions

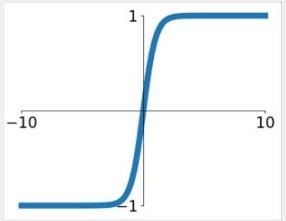
Sigmoid

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



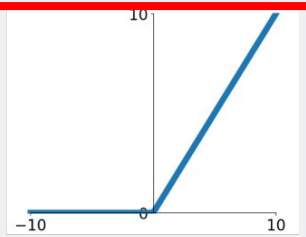
tanh

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



ReLU

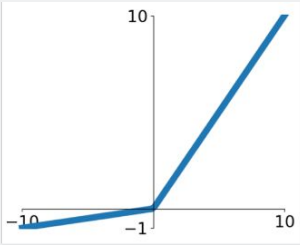
$$\max(0,x)$$



ReLU is a good default choice for most problems

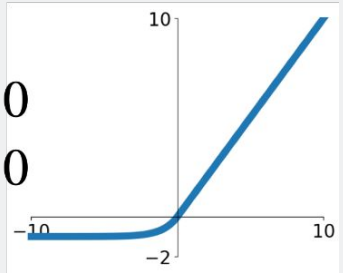
Leaky ReLU

$$\max(0.2x, x)$$



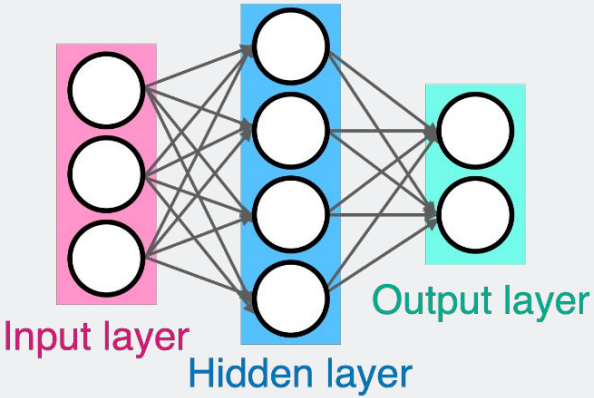
ELU

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) -), & x \leq 0 \end{cases}$$



M | ROBOTICS

Neural Network in 20 Lines



Initialize weights and data

Compute loss (Sigmoid activation, L2 loss)

Compute gradients

SGD step

```
1  import numpy as np
2  from numpy.random import randn
3
4  N, Din, H, Dout = 64, 1000, 100, 10
5  x, y = randn(N, Din), randn(N, Dout)
6  w1, w2 = randn(Din, H), randn(H, Dout)
7  for t in range(10000):
8    h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9    y_pred = h.dot(w2)
10   loss = np.square(y_pred - y).sum()
11   dy_pred = 2.0 * (y_pred - y)
12   dw2 = h.T.dot(dy_pred)
13   dh = dy_pred.dot(w2.T)
14   dw1 = x.T.dot(dh * h * (1 - h))
15   w1 -= 1e-4 * dw1
16   w2 -= 1e-4 * dw2
```

M | ROBOTICS