



LECTURE 4

DEEP REPRESENTATION LEARNING I

What is Learning?

PRESS **→** OR SPACE. 

© Marc Rußwurm

Licensed under [Creative Commons Attribution-NonCommercial 4.0 International \(CC BY-NC 4.0\)](#).

You may share and adapt these slides for teaching, research, and other non-commercial educational purposes with attribution.

Commercial training, paid workshops, consulting seminars, or incorporation into commercial course products require prior permission.

Two Questions for Lectures 3–4

LECTURE 3

How does a neural network learn?

LECTURE 4

**Why should what it learns work
beyond the training data?**

Learning Outcomes Roadmap

Learning outcome	Block
☐ Explain why representation choice determines task difficulty.	Why Learn Representations?
☐ Interpret logistic regression as a linear decision boundary in feature space.	Why Learn Representations?
☐ Explain how an MLP learns transformations of feature space.	Multilayer Perceptrons
☐ Distinguish representation learner vs final decision layer in a network.	Multilayer Perceptrons
☐ Explain the roles of loss, gradient descent, and backpropagation.	Gradient Descent & Backpropagation

Today: Lecture Structure

Three major blocks guide the narrative.

1

Why Learn Representations?

2

Multilayer Perceptrons

3

**Gradient Descent &
Backpropagation**

Start with representation quality and geometric separability.

Good representations make hard problems easy.

Why Learn Representations?

A difficult task is easy in the right form.

Recap Lecture 1

CHINESE

□□□□□□□□□□□□□□

ARABIC

لقد تعلّم عقلنا أن يفهم هذه الجملة.

HEBREW

המוח שלנו למד להבין את המשפט הזה.

KOREAN

□□□ □□□ □ □□□ □□□□ □□ □□□□□.

LATIN

Our mind has learned to understand this sentence.



Photo by Vitaly Gariev on Pexels

Choosing Suitable Data Representations

Recap Lecture 1

Images

255	255	255	255	255	88	88	255	255	255	255	255
255	255	255	255	88	88	255	88	88	255	255	255
255	255	255	70	170	170	70	170	170	70	255	255
255	255	70	200	200	200	200	200	200	70	255	255
255	70	200	200	200	200	200	200	200	200	70	255
255	70	200	200	200	200	200	200	200	200	70	255
255	70	200	200	200	200	200	200	200	200	70	255
255	70	200	200	200	200	200	200	200	200	70	255
255	255	70	200	200	200	200	200	200	70	255	255
255	255	70	200	200	200	200	200	200	70	255	255
255	255	255	70	200	200	200	200	200	70	255	255
255	255	255	255	70	70	70	70	70	255	255	255

Math

$$\text{XLVII} + \text{LXXVIII} = ?$$

$$47 + 78 = 125$$

Places

COORDINATES

50°43'57.73" N,
7°06'16.63" E

PLACE

**Hofgarten, Bonn,
Germany**

Machine Learning needs good Features

Recap Lecture 3

Input Tensor

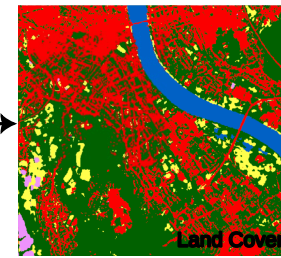


Feature Design

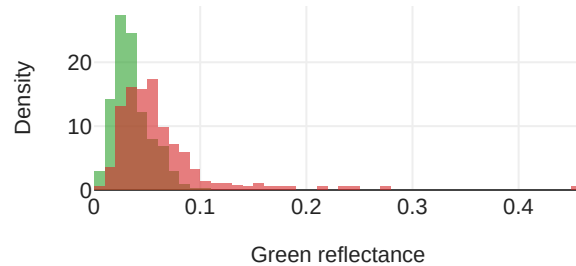


Model

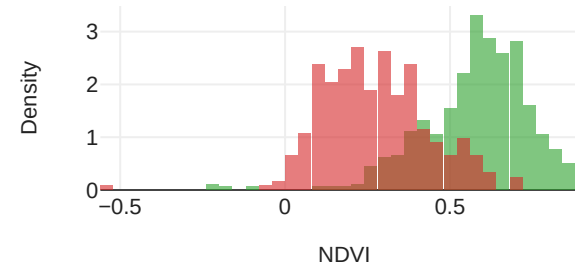
Output Tensor



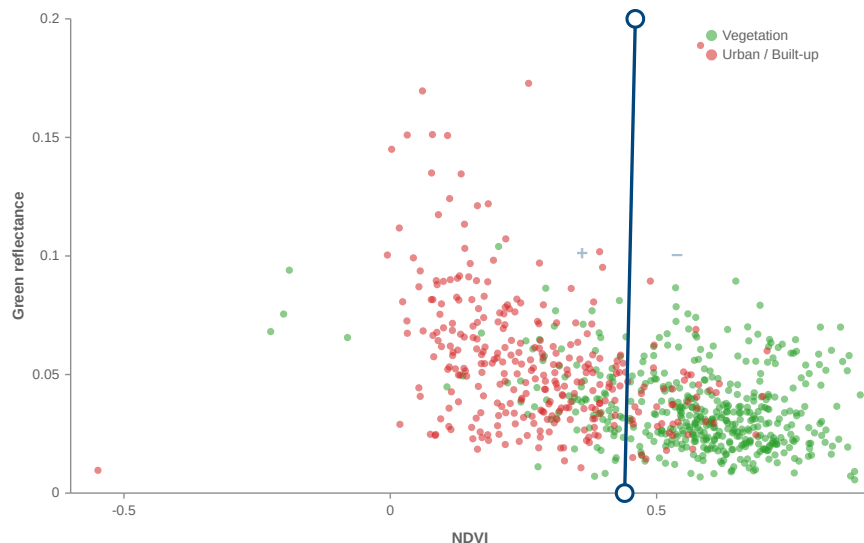
Raw Band: Green



Engineered Feature: NDVI



Good Representations allow for Simple Models



LINEAR REGRESSION MODEL

$$y = w_1 \text{NDVI} + w_2 \text{Green} + b$$

$$y = \mathbf{w}^\top \mathbf{x} + b$$

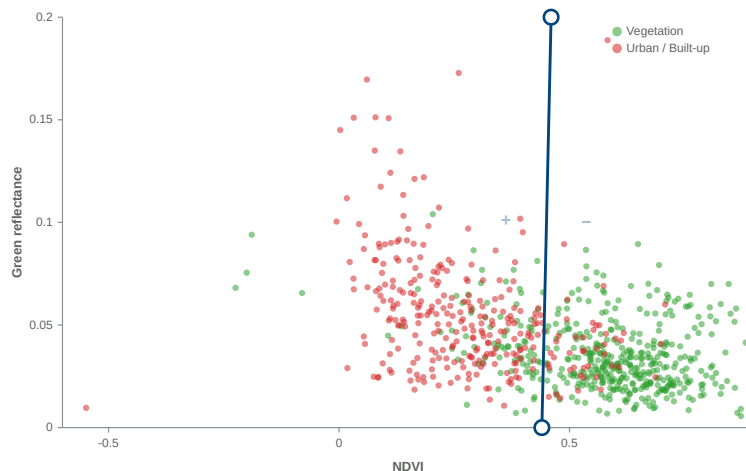
CURRENT BOUNDARY

$$y = -1.00 \text{NDVI} + 0.10 \text{Green} + 0.44$$

Accuracy **82.9%** (759 points)

Drag either handle to reposition the line. Hover the plot to measure distance to the boundary.

Logistic Regression for Classification



LOGISTIC REGRESSION MODEL $P(y | \mathbf{x})$

$$P(y=1 | \mathbf{x}) = \sigma(w_1 \text{NDVI} + w_2 \text{Green} + b)$$

$$P(y=1 | \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$

where $\sigma(z) = \frac{1}{1 + e^{-z}}$ squashes the score into a probability in (0, 1).

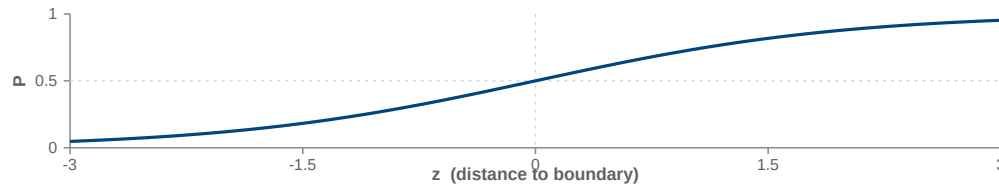
DECISION BOUNDARY · $P = 0.5$

$$z = -1.00 \text{NDVI} + 0.10 \text{Green} + 0.44$$

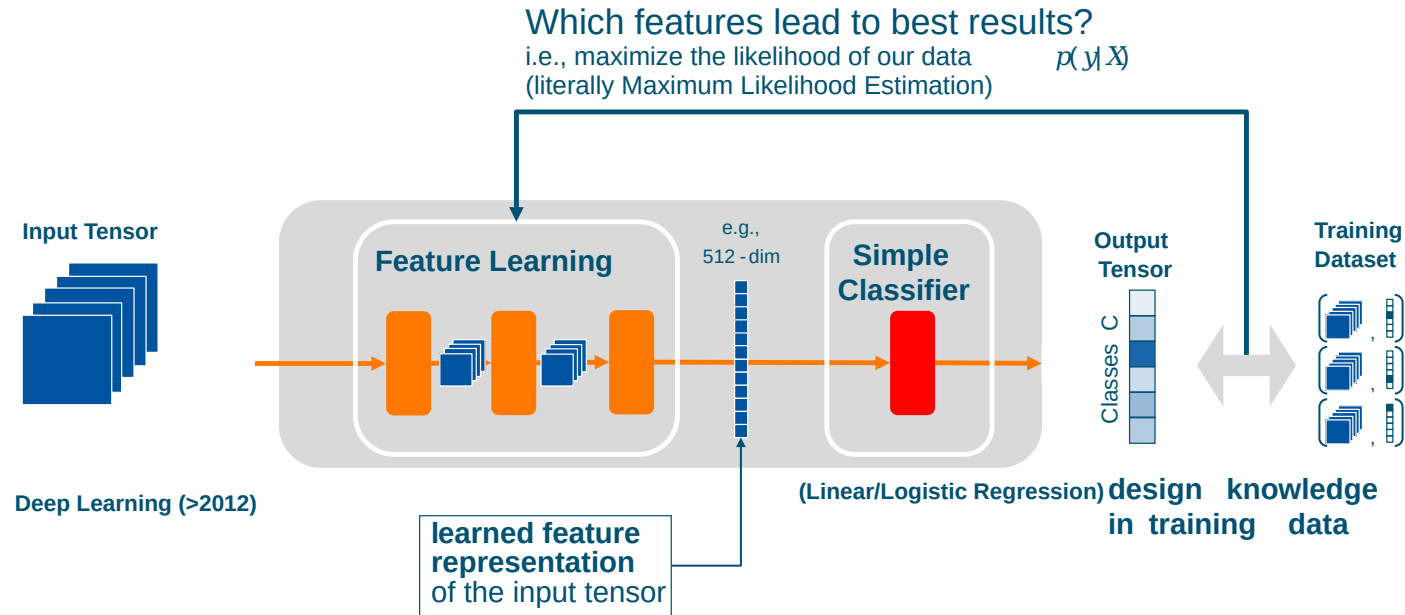
Accuracy **82.9%** (759 points)

Drag either handle to reposition the boundary. Hover the plot to read the predicted probability.

The logistic (sigmoid) function — squashing the score z into a probability



Learning Feature Representations



Timeline and Evolution of Deep Learning

<2012	2015 onwards	>2020 →
Classic Machine Learning	Supervised Deep Learning	Self-Supervised Learning
Feature Design: Which features extract the most information?	Model Design: Which architecture is best for my tensors?	Loss Design: Which loss function or training strategy?
Data: small labelled	Data: large labelled	Data: large unlabelled
Models: Random Forest, Boosting, Lin. Regression	Models: MLPs, CNNs, RNNs	Model: Transformers
<i>Covered in Lectures 2 & 3</i>	<i>Covered in Lectures 4 & 5</i>	<i>Covered from Lecture 7 onwards</i>

Image Classification

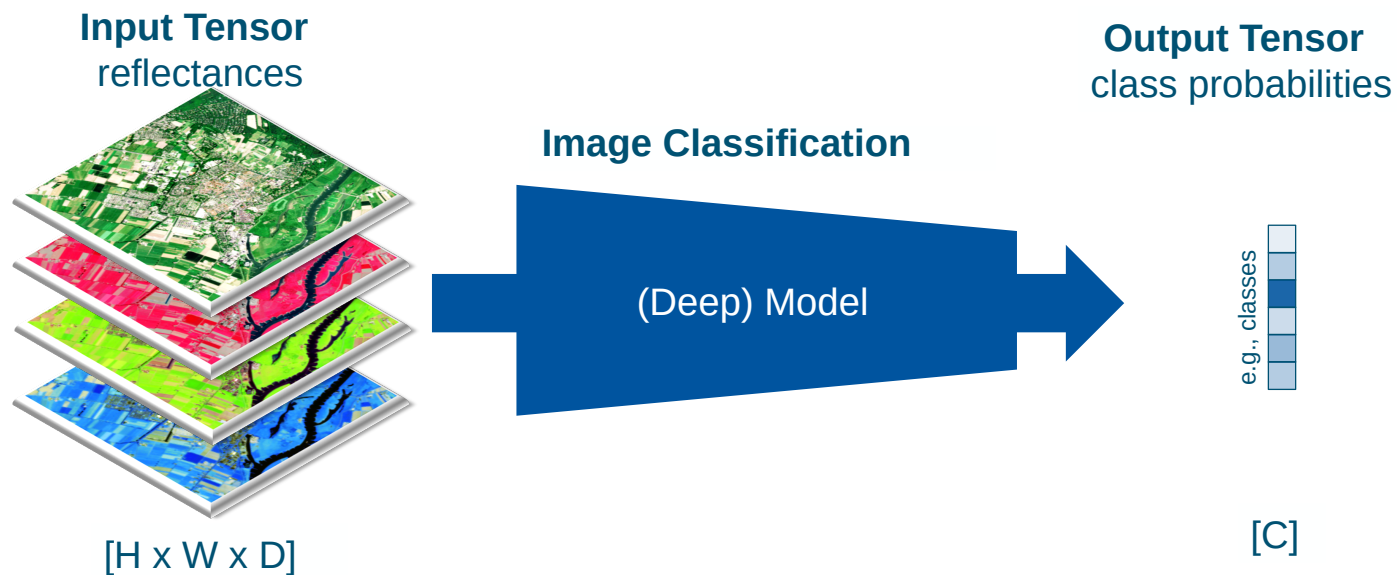
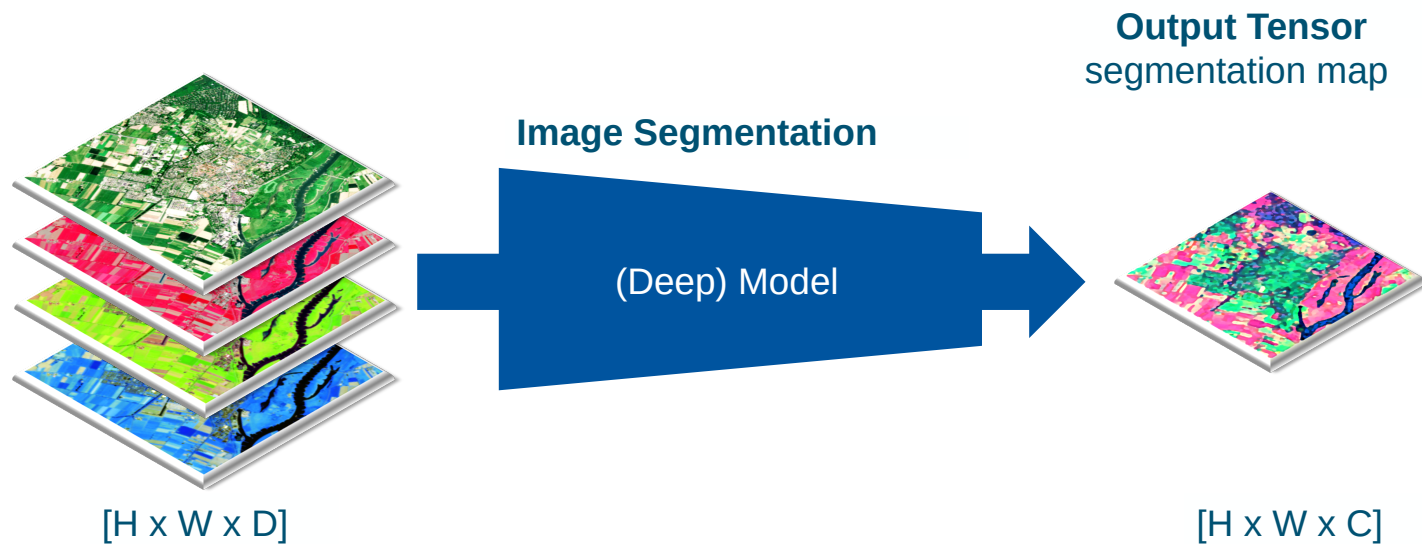


Image Segmentation



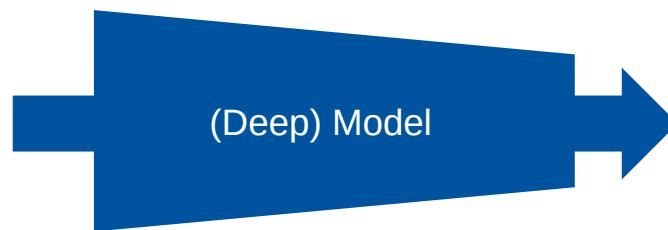
Object Detection

Input Tensor
reflectances



$[H \times W \times 3]$

Object Detection

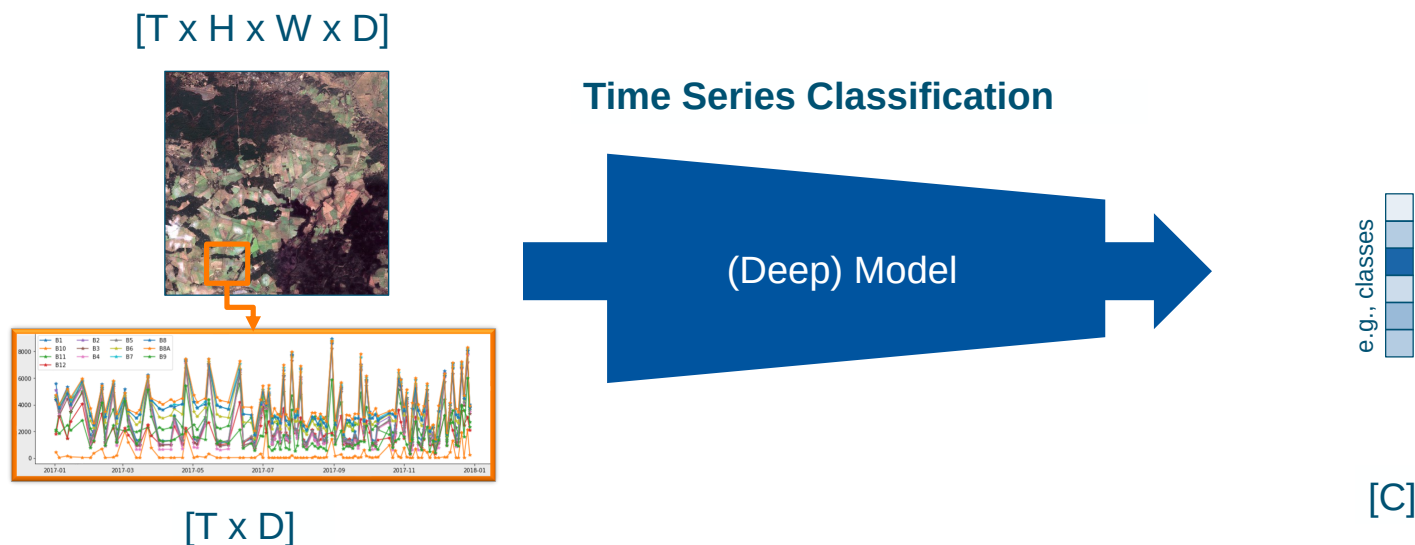


Output Tensor
bounding boxes

X	Y	H	W	
Green	Green	Green	Green	Bbox 1
Red	Red	Red	Red	Bbox 2
Yellow	Yellow	Yellow	Yellow	Bbox 3
Blue	Blue	Blue	Blue	...

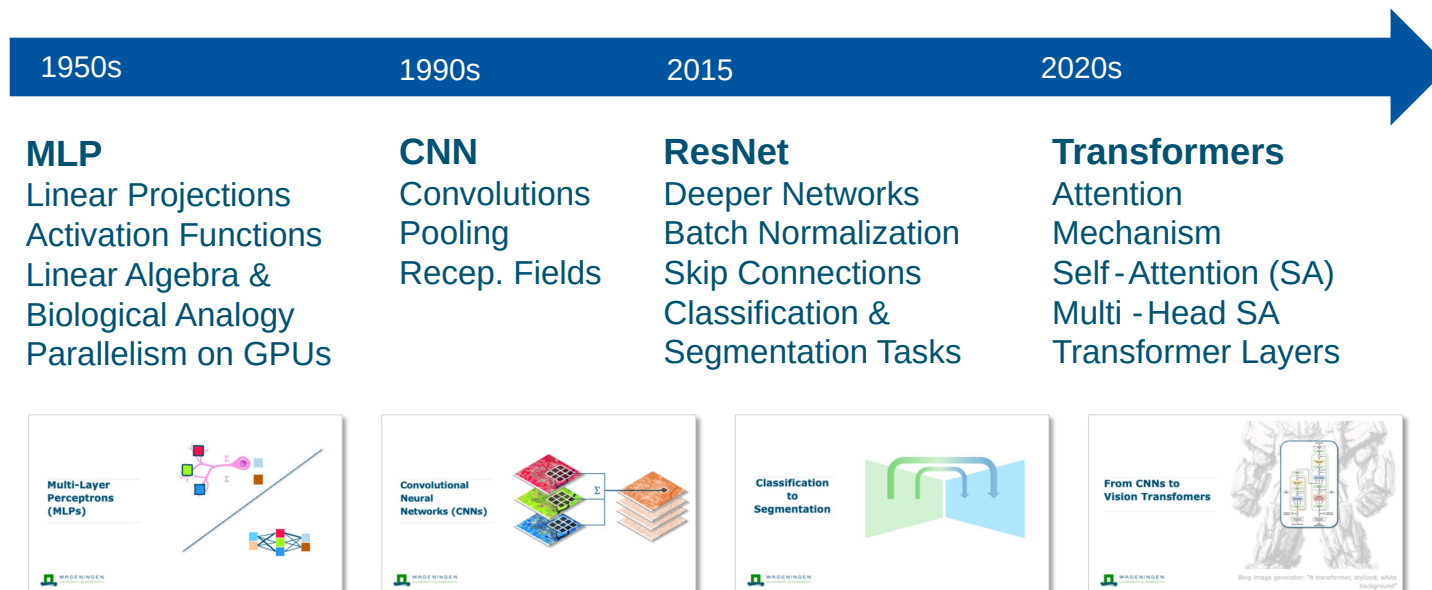
$[N \times 4]$

Time Series Classification



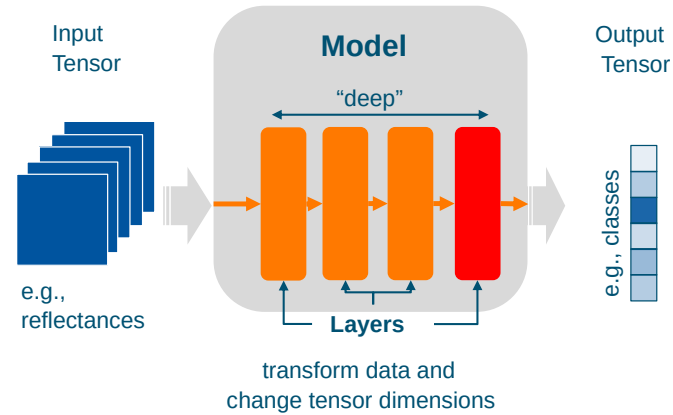
Overview: Deep Model Architectures

Lab 4 Content

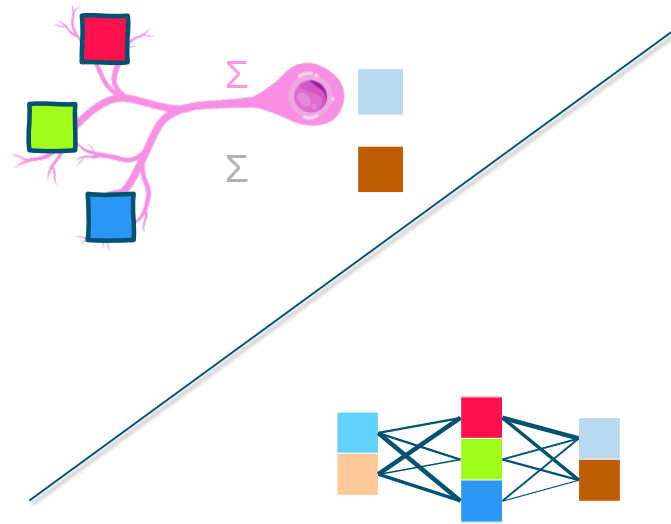


Recap: Learned Representations

- Good representations make difficult problems easier.
- In classic machine learning, we manually select data and engineer features to be a good representation of the problem.
- Deep learning models learn a feature representation that is then well-suited for a final linear classification with logistic regression.

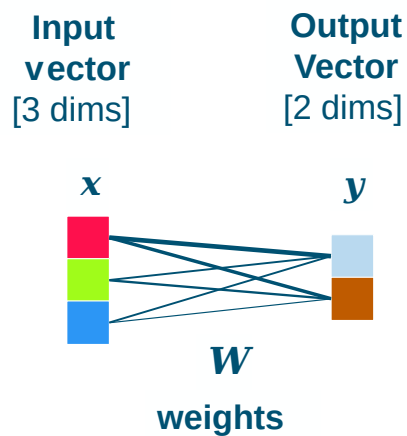


Multilayer Perceptrons



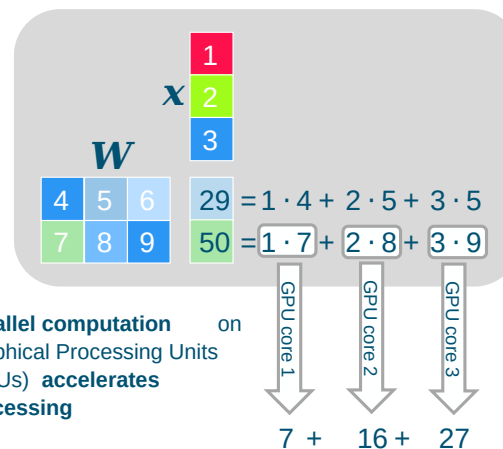
Learning the representation.

Linear Transformation



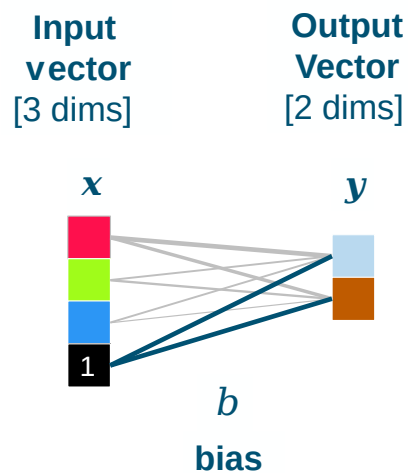
Linear Transformation

$$y = Wx = \sum w_i x_i$$



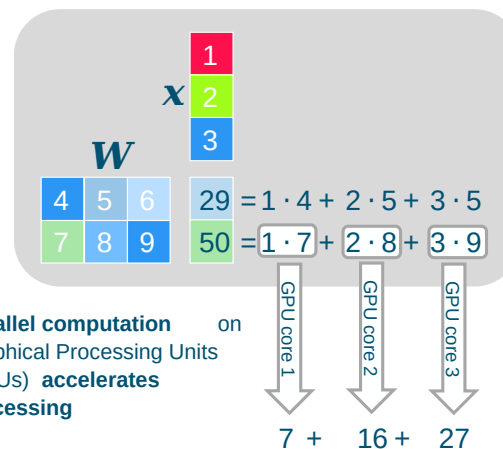
Parallel computation on
Graphical Processing Units
(GPUs) accelerates
processing

Linear Transformation with Bias



Linear Transformation

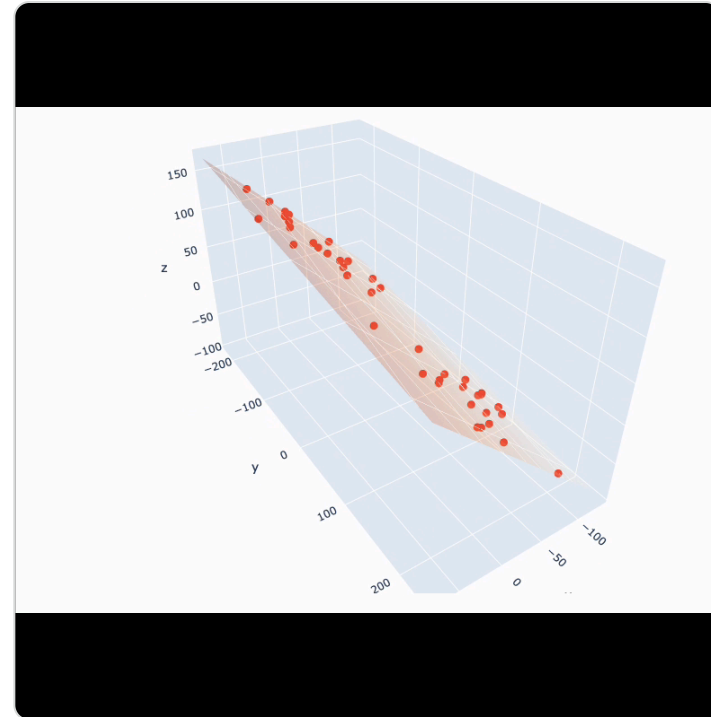
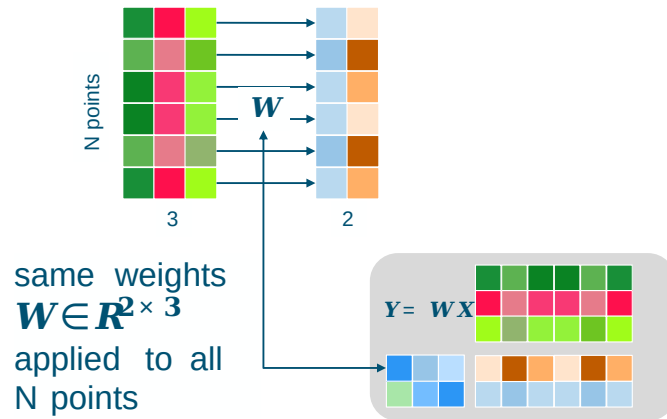
$$y = Wx = \sum w_i x_i + b$$



Parallel computation on
Graphical Processing Units
(GPUs) accelerates
processing

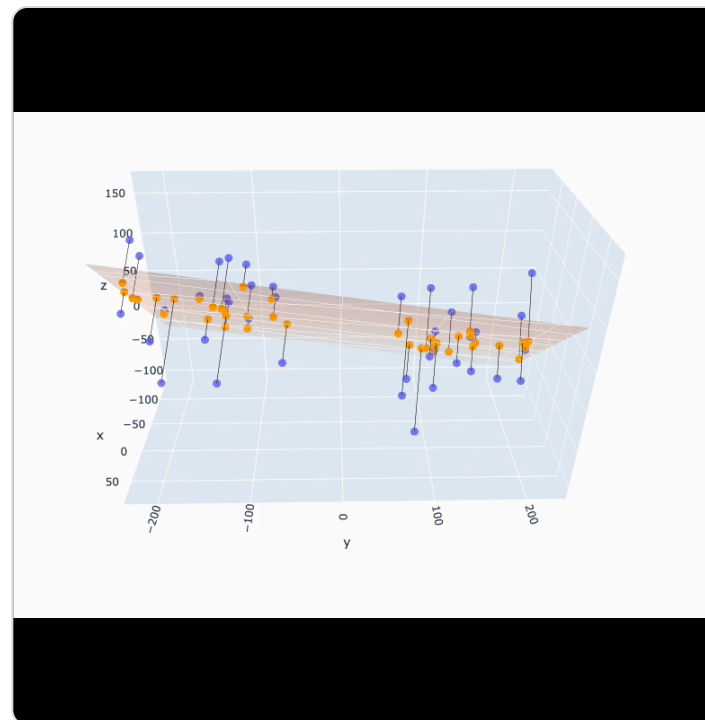
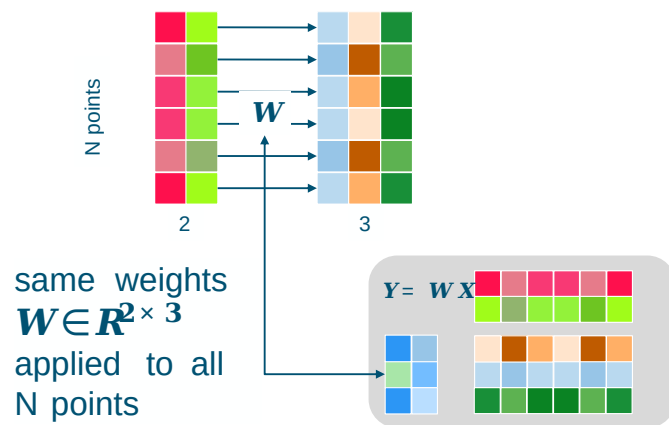
Linear Transformation from 3D to 2D

Linear Transformation
of N points from 3D to 2D



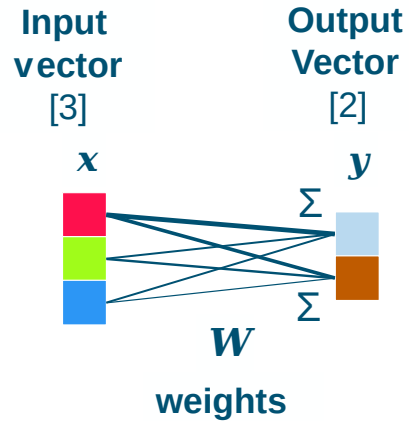
Linear Transformation from 2D to 3D

Linear Transformation $Y = W X$
of N points from 2D to 3D

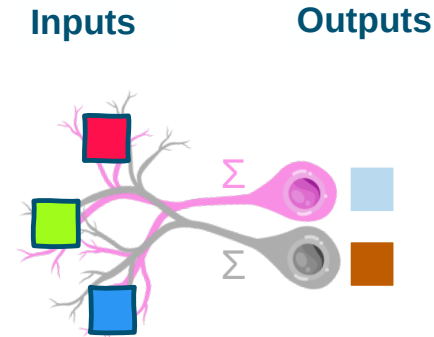


Biological Interpretation of Adding Signals

Matrix vector multiplication



Biological Interpretation

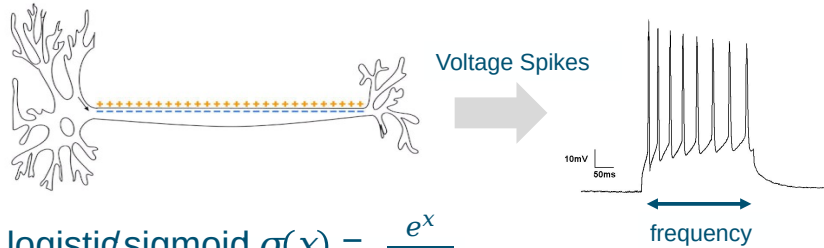


Neuron Image: [Wikipedia](#) by [Hariadhi](#) - Own work

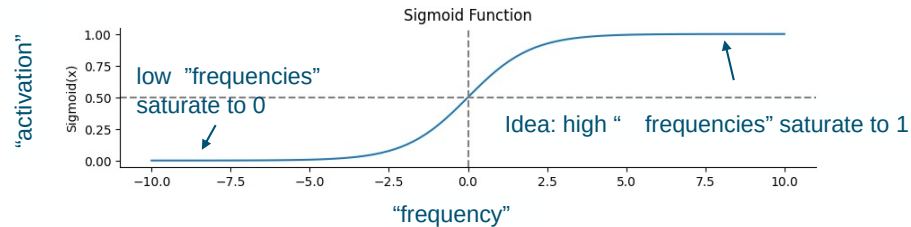
Activation Functions – Biological Interpretation

Image: [Wikipedia](#) [Laurentaylorj](#)
CC BY-SA 3.0

Spikes in voltage
Image: [Wikipedia](#) Bilz0r CC BY - SA 3.0

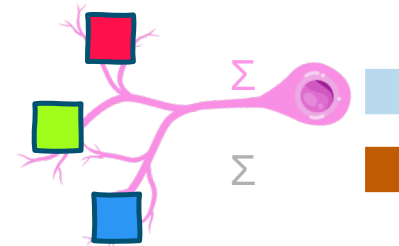


logistic/sigmoid $\sigma(x) = \frac{e^x}{1 + e^x}$



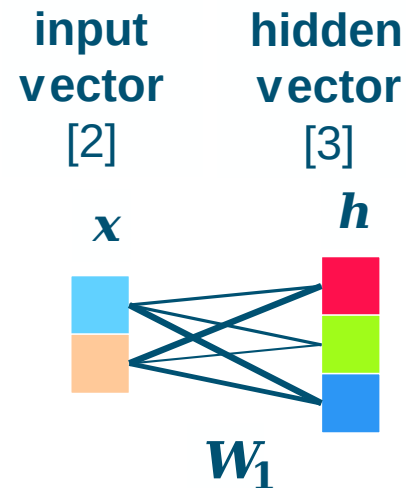
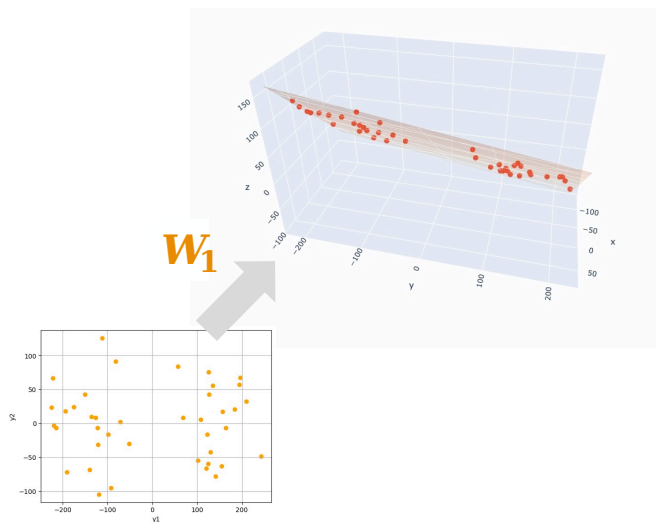
Inputs

Outputs



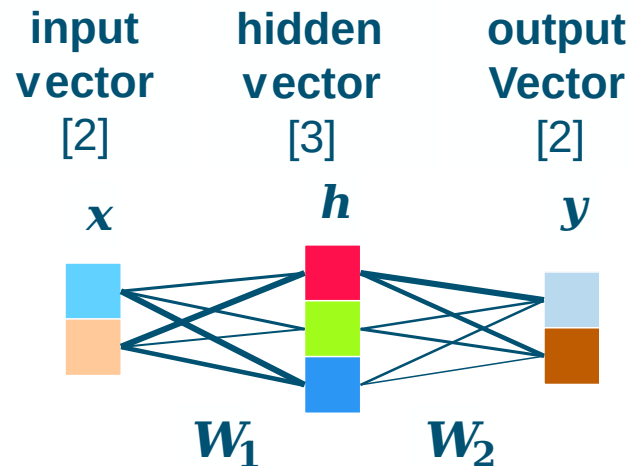
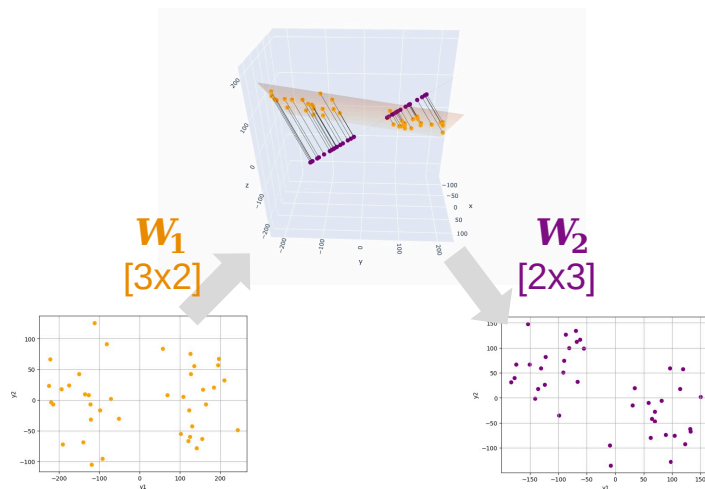
Linear Algebra: Stacking Layers

A $2D \rightarrow 3D$ Linear Projection moves data into 3D representation space



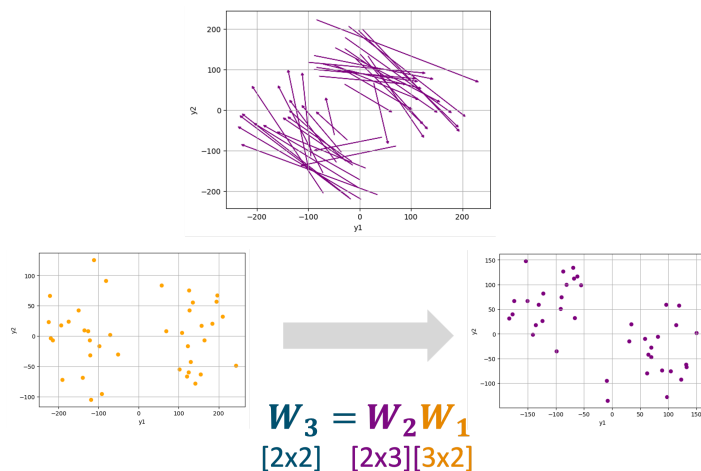
Linear Algebra: Stacking Layers

Two Linear projections $2D \rightarrow 3D \rightarrow 2D$ in a 2-Layer network



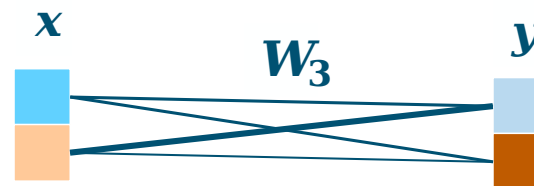
Linear Algebra: Stacking Layers

Two Linear projections $2D \rightarrow 3D \rightarrow 2D$ without non-linearity is a single $2D \rightarrow 2D$ projection



input
vector
[2]

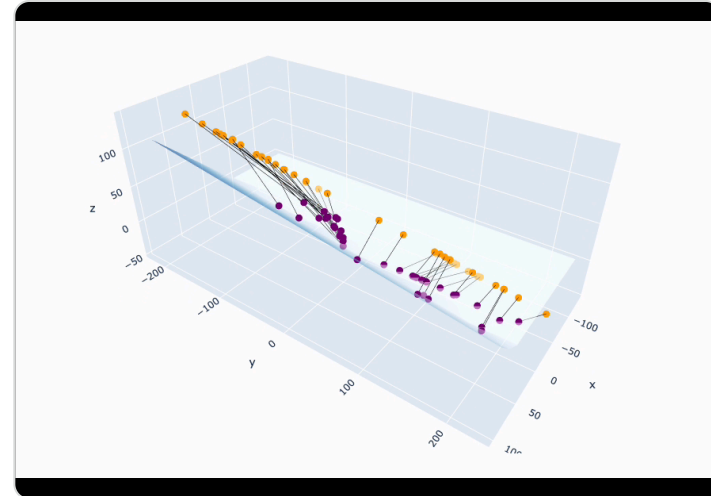
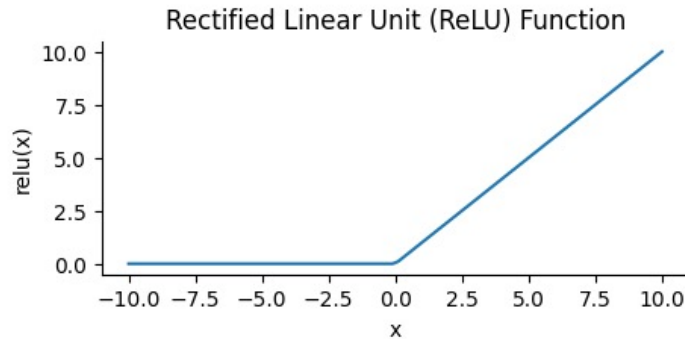
output
Vector
[2]



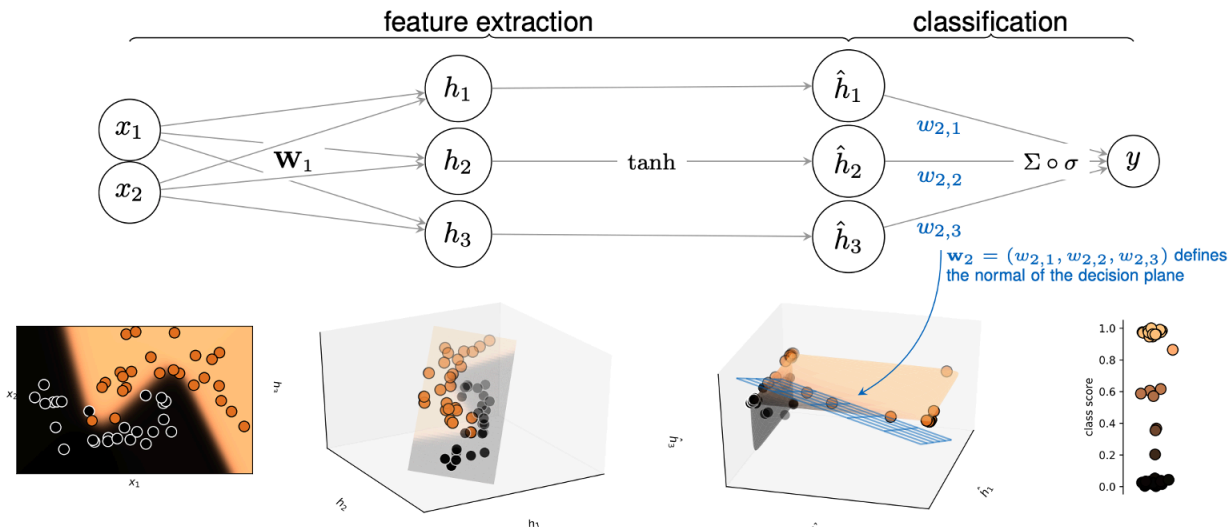
Linear Algebra: Stacking Layers

Non-linear functions like ReLU are needed to transform feature space

Rectified Linear Unit $\text{ReLU}(x) = \max(x, 0)$



More layers → more complex transformations



(a) Input space
 $\mathbf{x} = (x_1, x_2)$

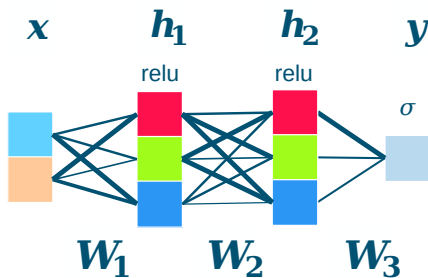
(b) linear projection
 $\mathbf{W}_1^T \mathbf{x}$ into $\mathbb{R}^3$

(c) non-linear distortion
with $\tanh(\mathbf{W}_2^T \mathbf{x})$

(d) Output class probabilities $y \in [0, 1]$

Multi-Layer Perceptron - Implementation

3 Layer MLP Classification Network:

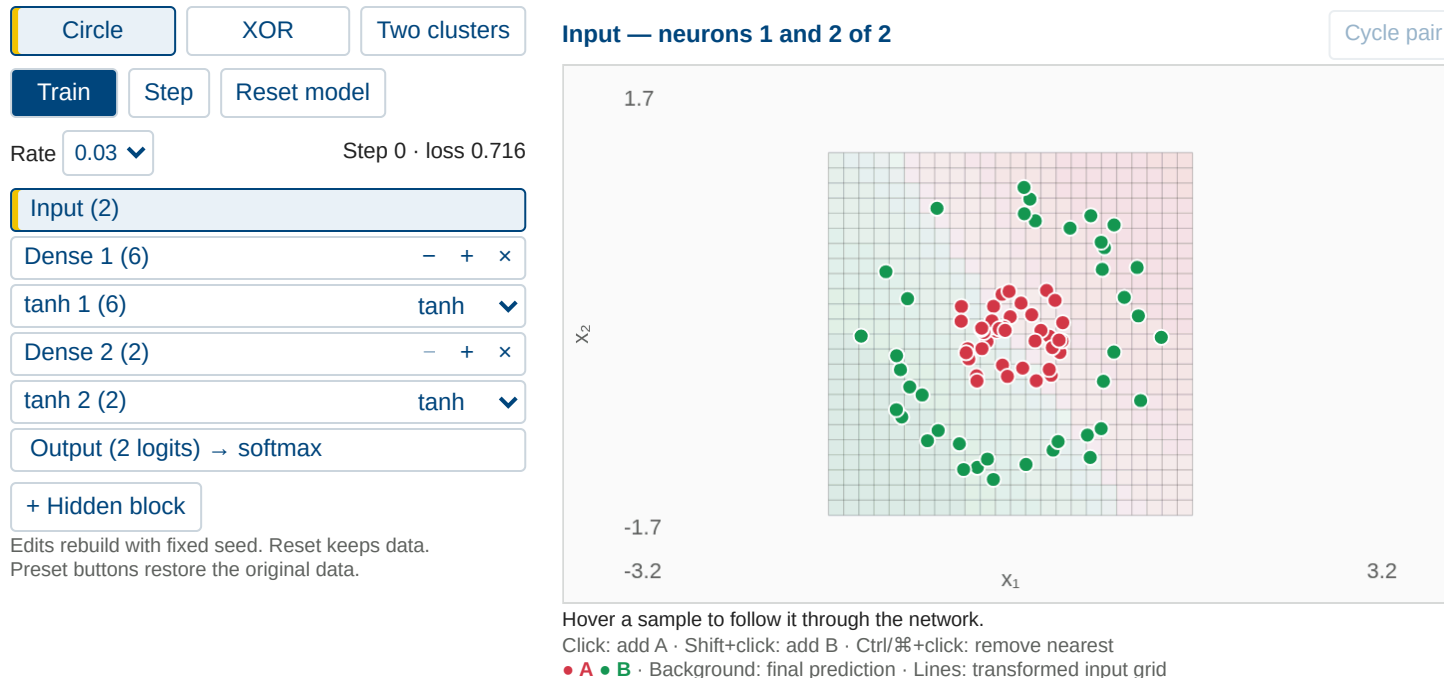


```
import torch
import torch.nn as nn
import torch.nn.functional as F

class MLPClassifier(nn.Module):
    def __init__(
        self, input_dim, hidden_dim, num_classes
    ):
        super(MLPClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, num_classes)

    def forward(self, x):
        x = F.relu(self.fc1(x))      # 1st layer ReLU
        x = F.relu(self.fc2(x))      # 2nd layer ReLU
        x = F.sigmoid(self.fc3(x))   # Output (probs)
        return x
```

Demo: Multi-Layer Perceptron Feature Space



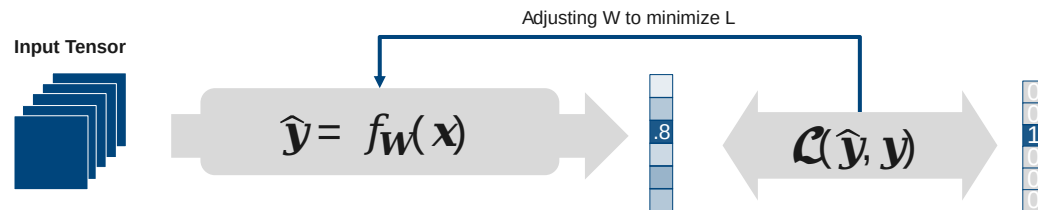
Inspired by Andrej Karpathy's [ConvNetJS Classify2D](#) demo. Implemented with Claude and Codex.

Learning the parameters.

Gradient Descent & Backpropagation

Training: Adjusting Weights to Minimize Loss

Finding good parameters W



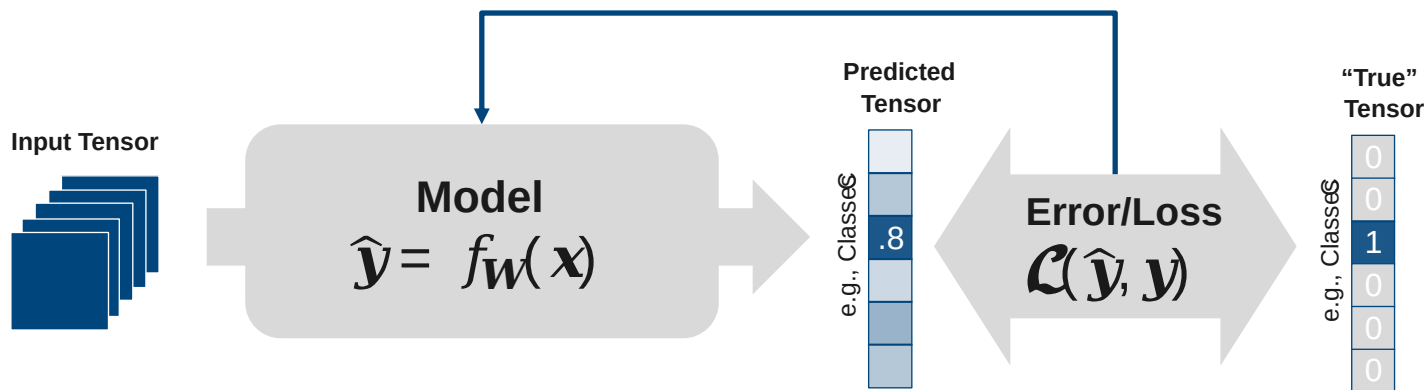
$$\mathcal{L}(\mathcal{D}_{\text{train}}) = \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{D}_{\text{train}}} \mathcal{L}(f(\mathbf{x}), \mathbf{y})$$



The Learning Objective: argmin

Minimize Error/Loss function

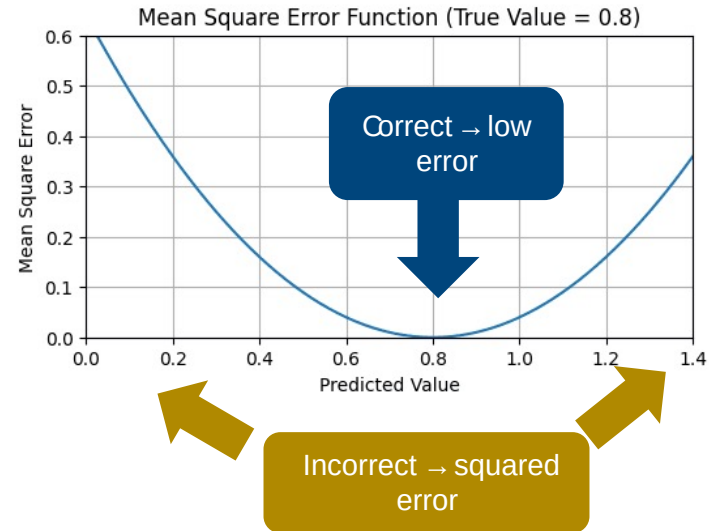
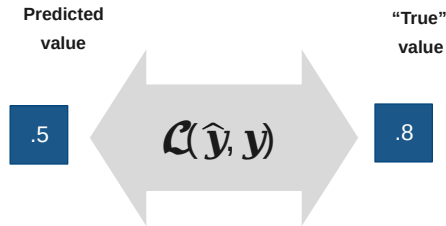
$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \mathcal{L}(f(\mathbf{x}), \mathbf{y})$$



Loss Function: Mean Squared Error

Regression: Mean Squared Error

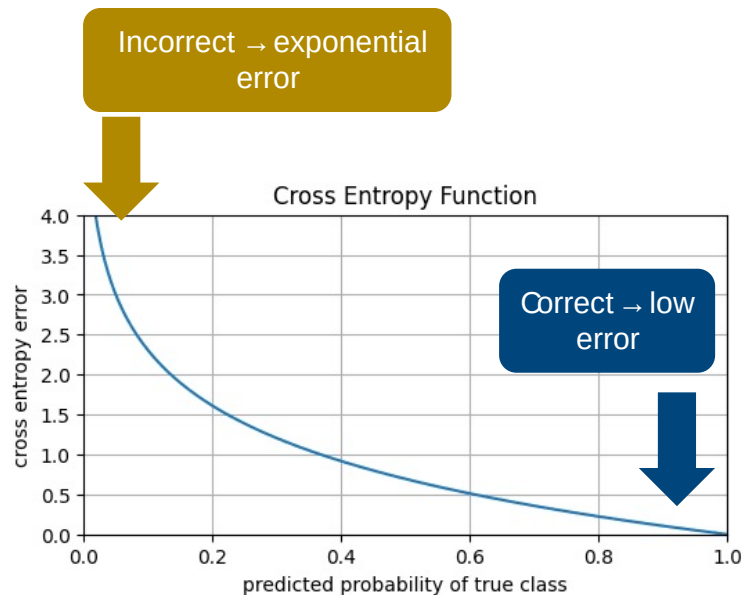
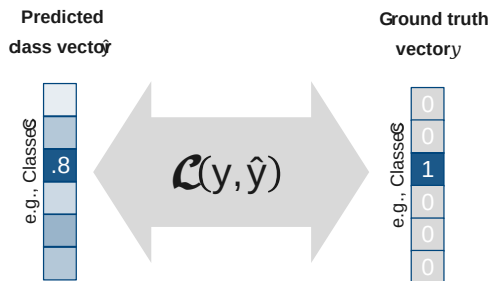
$$\mathcal{L}(y, \hat{y}) = (y - \hat{y})^2$$



Loss Function: Cross-Entropy

Classification: Cross Entropy

$$\mathcal{L}(y, \hat{y}) = \sum_{c \in \text{Classes}} -y_c \log(\hat{y}_c)$$



Loss Function: Cross-Entropy

Minimize Error/Loss Function:

$$W^* = \operatorname{argmin}_W \mathcal{L}(f_W(x), y)$$

Through Iterative Gradient Descent:

$$W^{(i+1)} = W^i - \nabla \mathcal{L}(f_W, \mathcal{D}_{\text{train}})$$

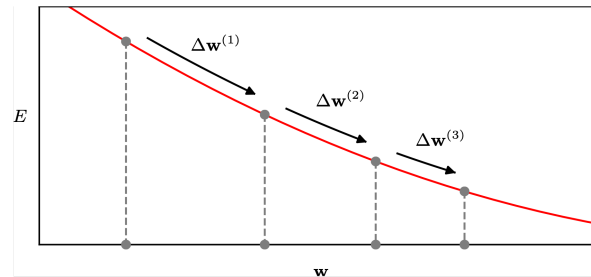
new weights

old weights



Gradients ∇ of the Loss function $\mathcal{L}$ over a “batch” of training data $\mathcal{D}_{\text{train}}$

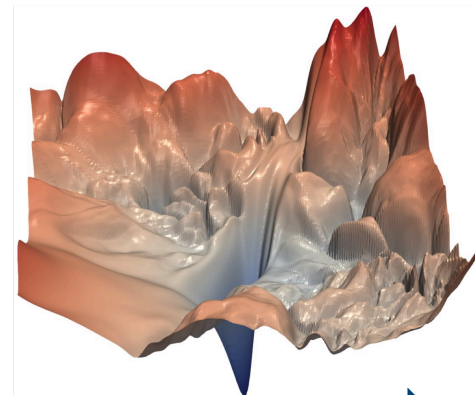
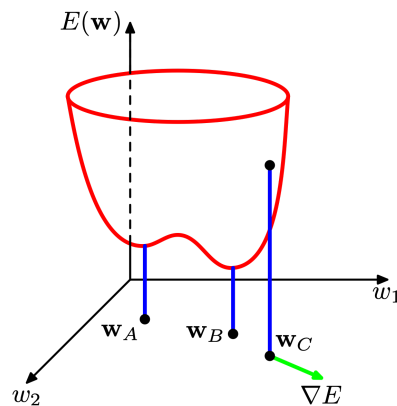
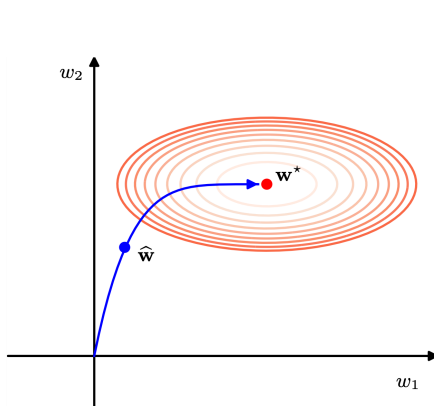
“What weights change the loss the most?”



Loss Surfaces

Minimize Error/Loss Function:

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \mathcal{L}(f_{\mathbf{W}}(\mathbf{x}), \mathbf{y})$$

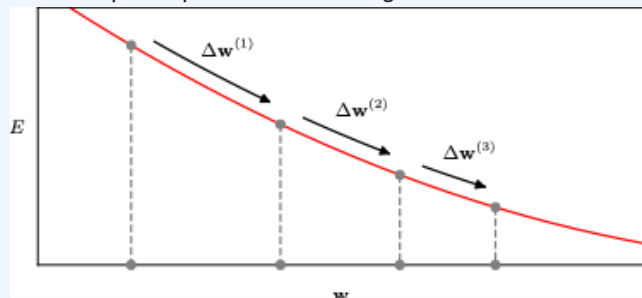


more complex models have more complex loss surfaces

Gradient Descent vs Backpropagation

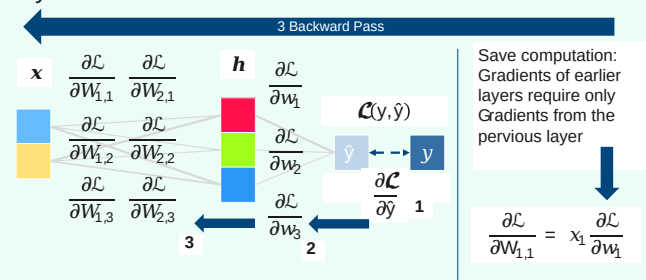
Gradient Descent

How to update parameters once gradients are known.



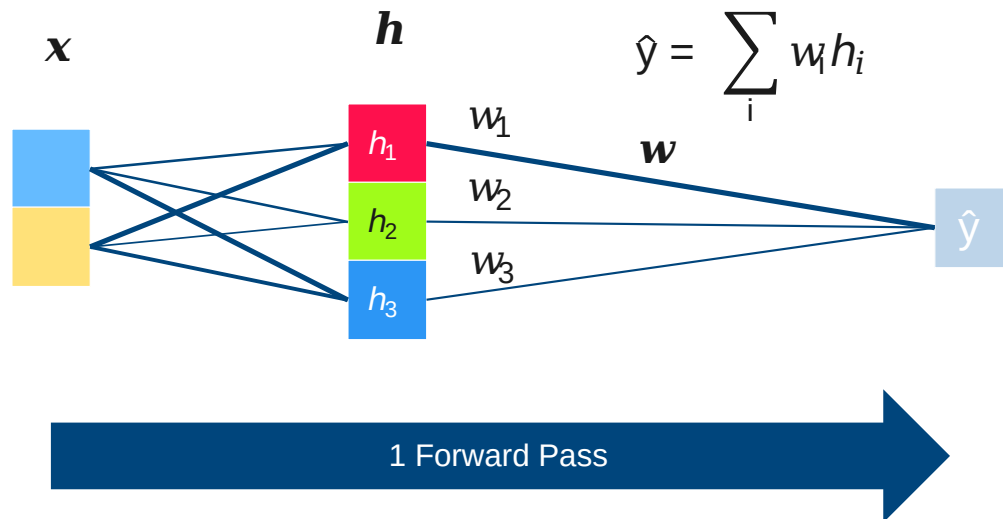
Backpropagation

How to compute gradients efficiently through many layers.

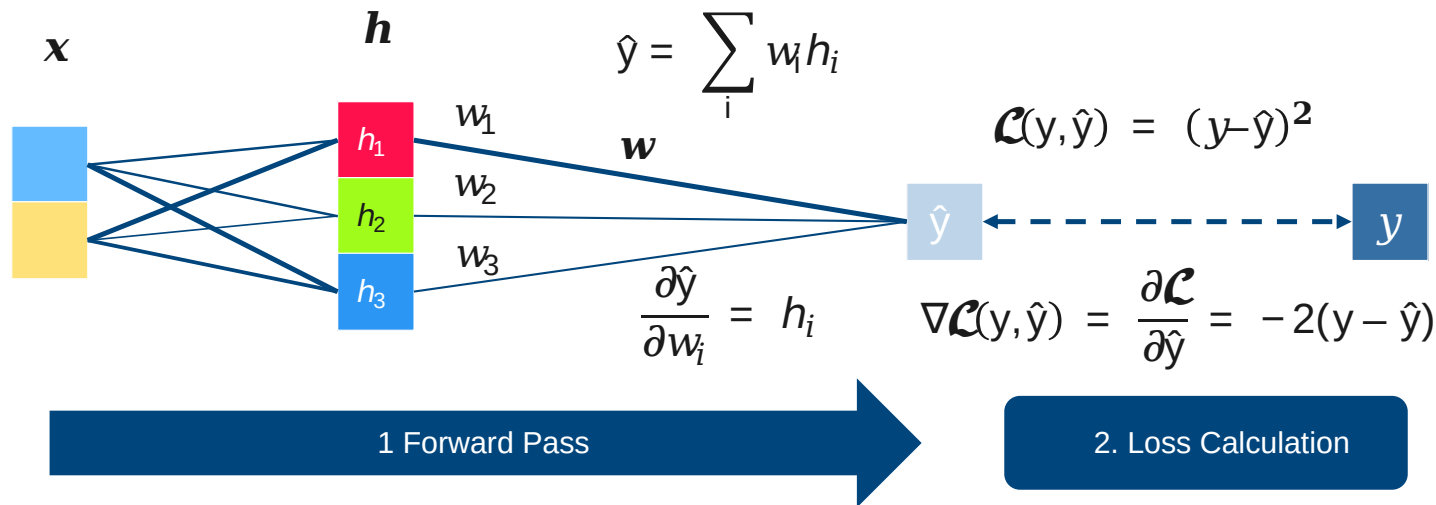


Backprop gives the gradients; gradient descent uses them to step the parameters.

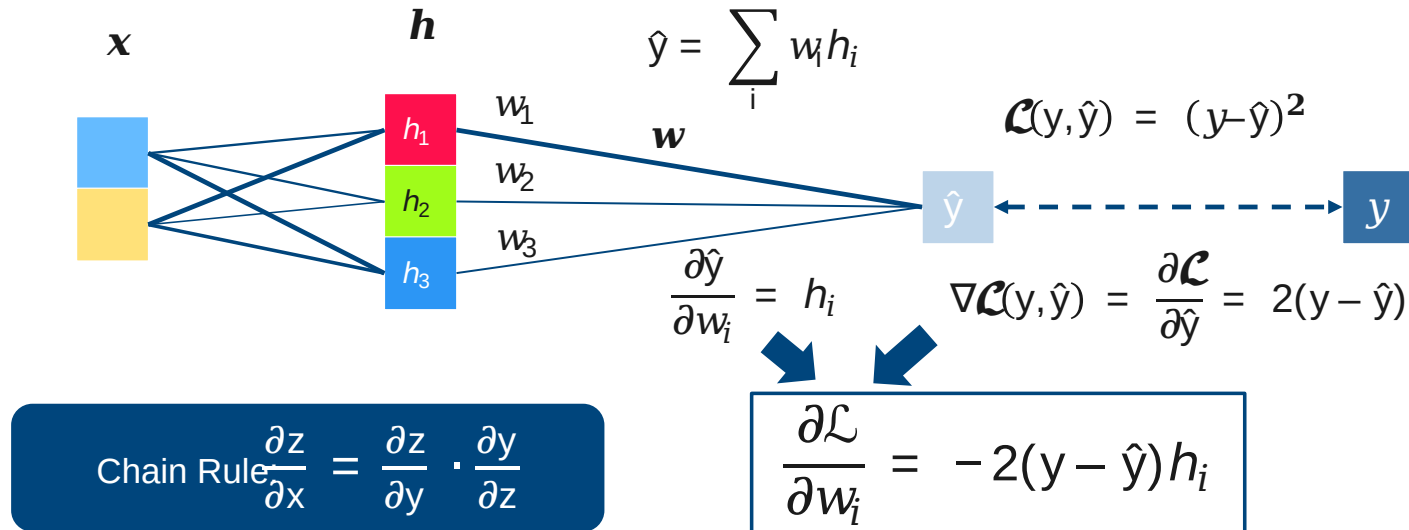
Backpropagation 1



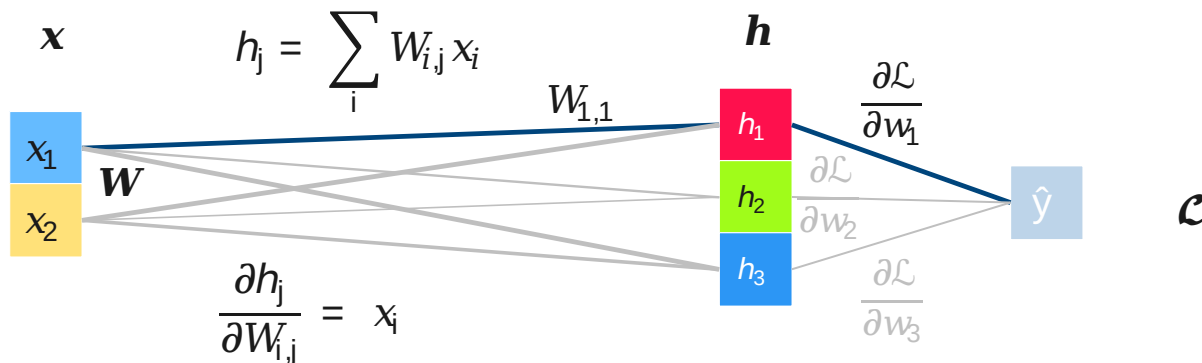
Backpropagation 2



Backpropagation 3

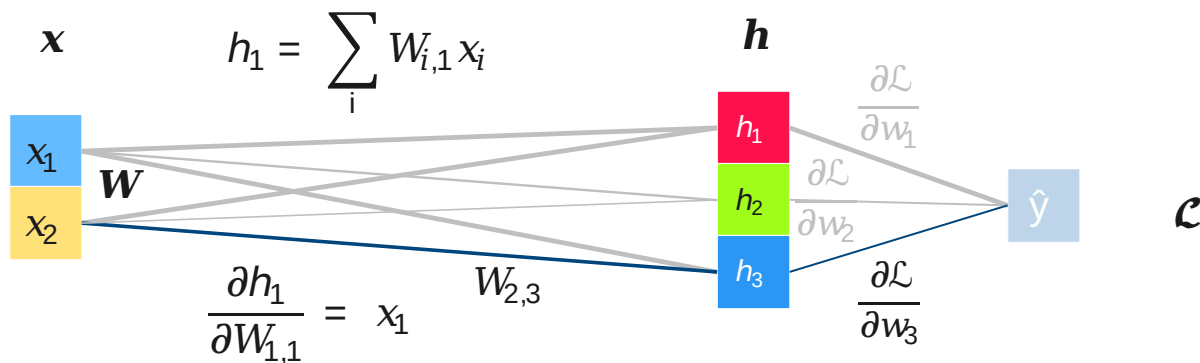


Backpropagation 4



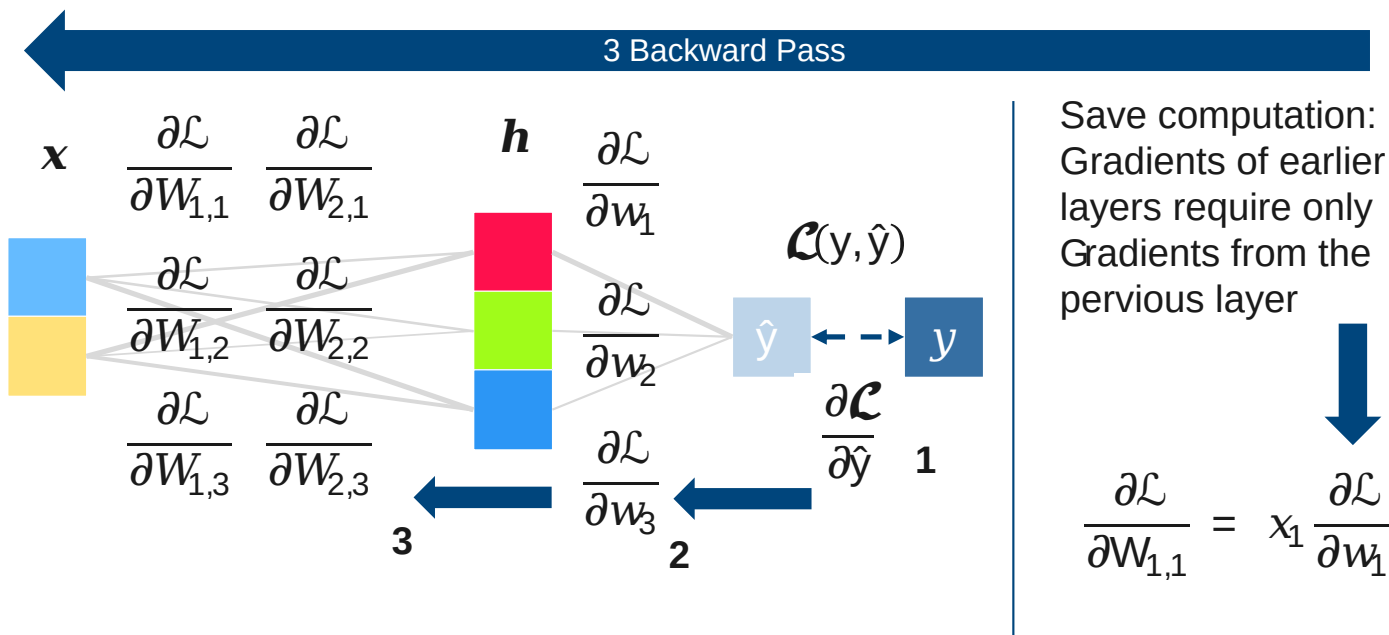
$$\frac{\partial \mathcal{L}}{\partial W_{1,1}} = x_1 \frac{\partial \mathcal{L}}{\partial h_1} = -x_1 2(y - \hat{y})h_1$$

Backpropagation 5

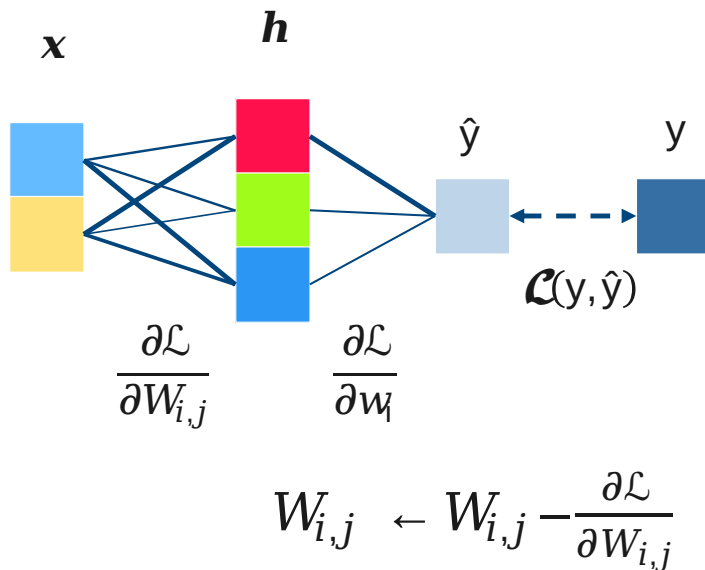


$$\frac{\partial \mathcal{L}}{\partial W_{2,3}} = x_2 \frac{\partial \mathcal{L}}{\partial w_3} = -x_2 2(y - \hat{y}) h_3$$

Backpropagation 6



Backpropagation 7



Forward Pass

```
y_pred = model(x)
```

Loss Computation

```
loss = MSE(y_pred, y)
```

Backward Pass

```
loss.backward()
```

Update Gradients

```
optimizer.step()
```

