

# Applied Deep Learning

---

Neural Networks, Optimization, and Backpropagation

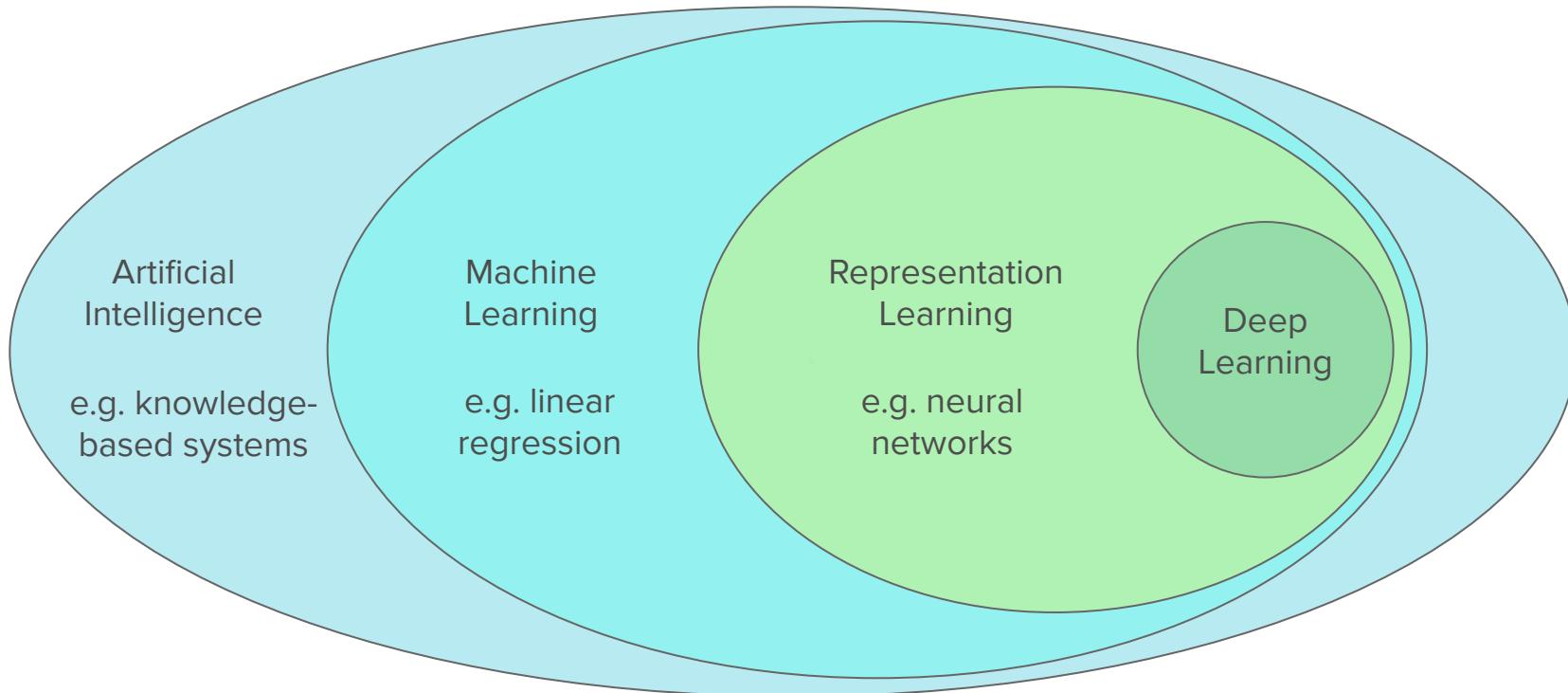
Alexander Pacha - TU Wien

# Recap

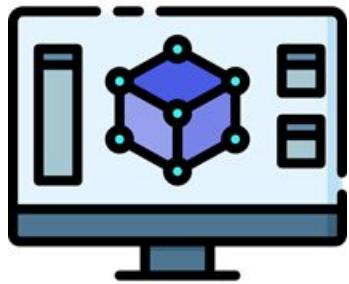
- What is the difference between Artificial Intelligence, Machine Learning, and Deep Learning?
- What are the main ingredients for machine learning?



# Recap - Terminology



# Recap - How Can a Machine Learn?



Model



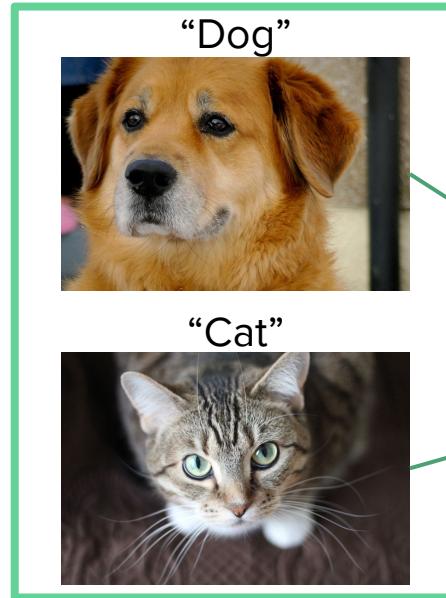
Loss function



Optimization function

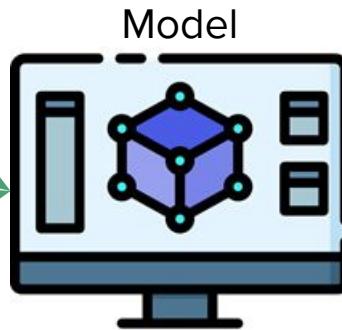
# How does a model work?

Training Set



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

Test Set



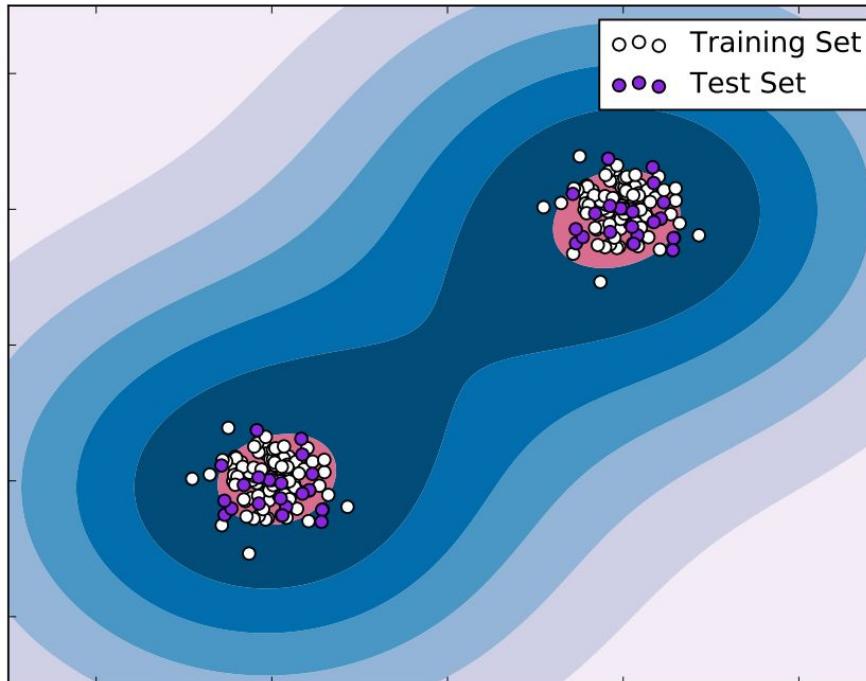
$w \in \{\text{Dog, Cat}\}$   
 $P(w|x)$

Inference

$P(\text{dog}|x) = 0.7$   
 $P(\text{cat}|x) = 0.3$

# How can a model work on unseen data?

Both datasets must have similar distribution



# Does it generalize?



A musical score page from Gustav Mahler's "Das Lied von der Erde". The page is numbered 11. The music is in 2/4 time, B-flat major. The vocal parts include "nim - mer wie - der - wa - ehen!", "ne - er ne - er a - wak - en!", "O Le - hen wie - bist du so - freu - de - ler O Herz, wie - Oh, life, thou art void of all - joy - to me! Oh heart, nigh -", "bist du he - - gen! So wollt' ich es - spü - ten mich sehn, fendl's Meer mit - braken a - - sun - der! Now, would thou coulds draw me, oh, pit - y - ing sea, Thy -", "let - die raus-henden Wo - - gen. (Julius Stern.) wild rushing bil - low deep un - - dee!", and "cresc." markings. The conductor's part shows various dynamics and tempo changes.



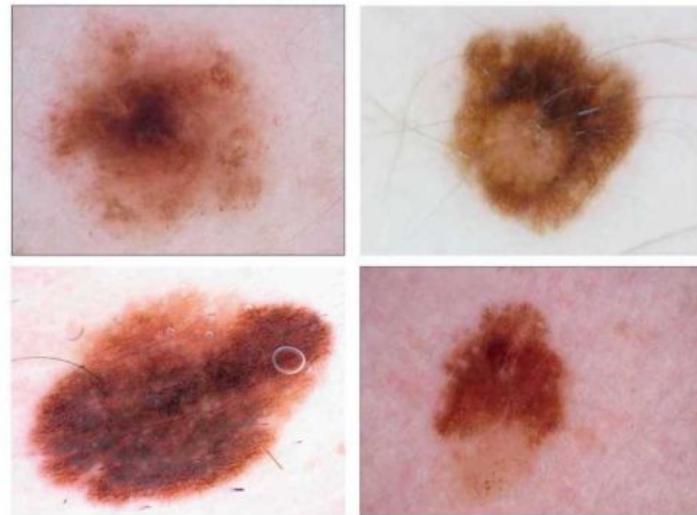
An open handwritten musical manuscript. The top page is labeled "Oboe in 8<sup>va</sup>". It contains several staves of music with various notes and rests. The bottom page contains lyrics in German: "Bijou, yvonne, Schauspielerin, sang, vorspiel, Blau, Enid, willst du blaue". The manuscript is written in ink on aged paper.

# Image Classification

Melanoma

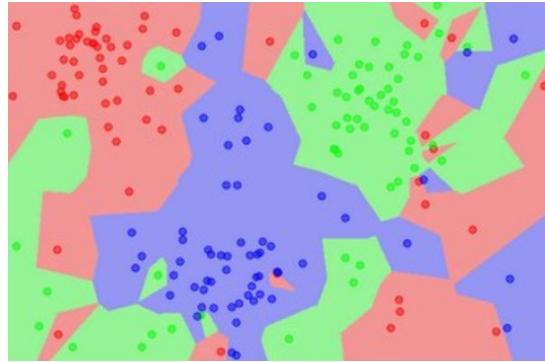


Benign

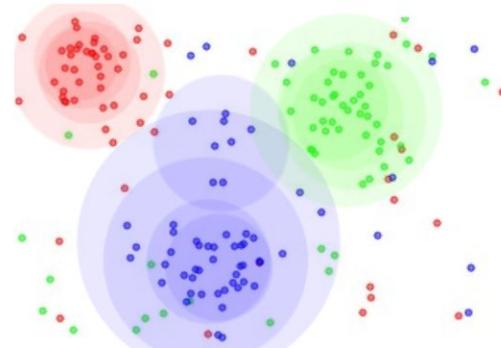


# Classes of models

Discriminative models learn decision boundaries where  $\operatorname{argmax}_w P(w|x)$  changes

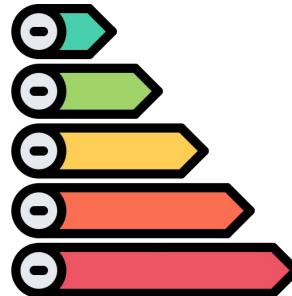


Generative models learn class-conditional densities  $P(x|w)$



# Output

- Classification problem with finite number of different labels



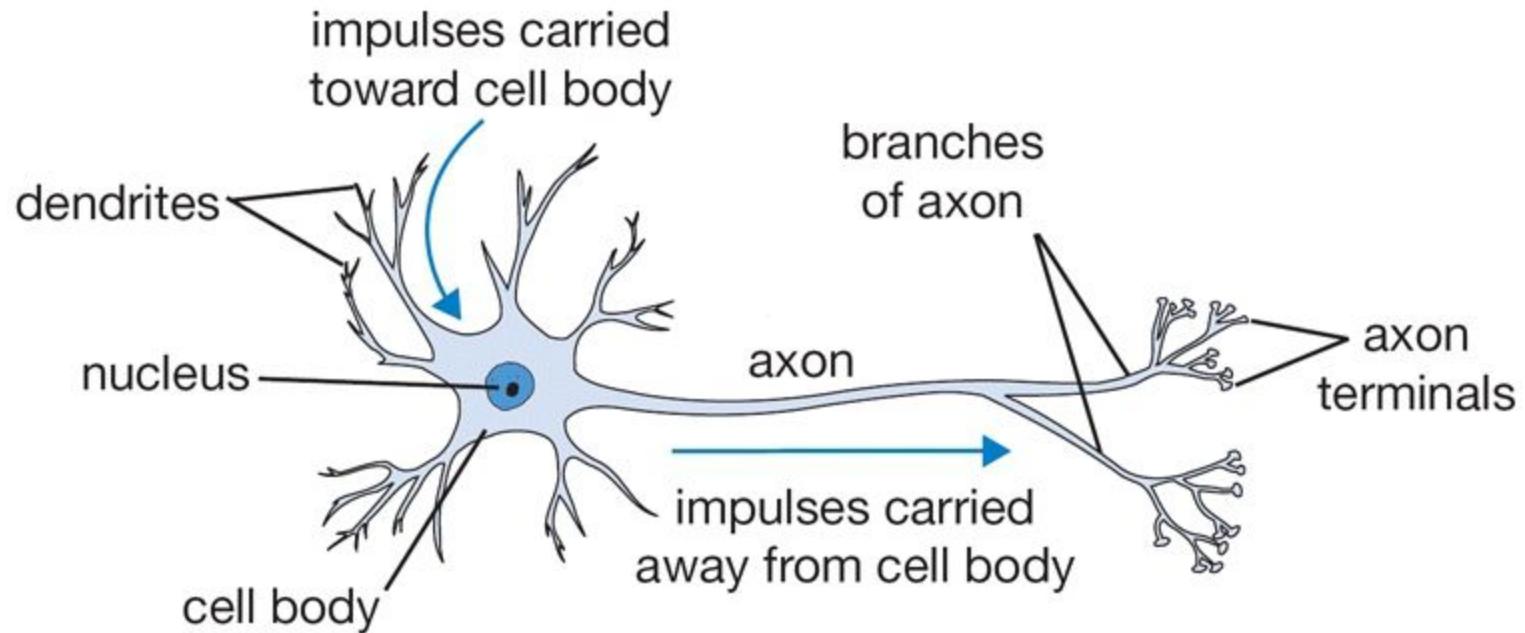
- Regression problem with a continuum of output values



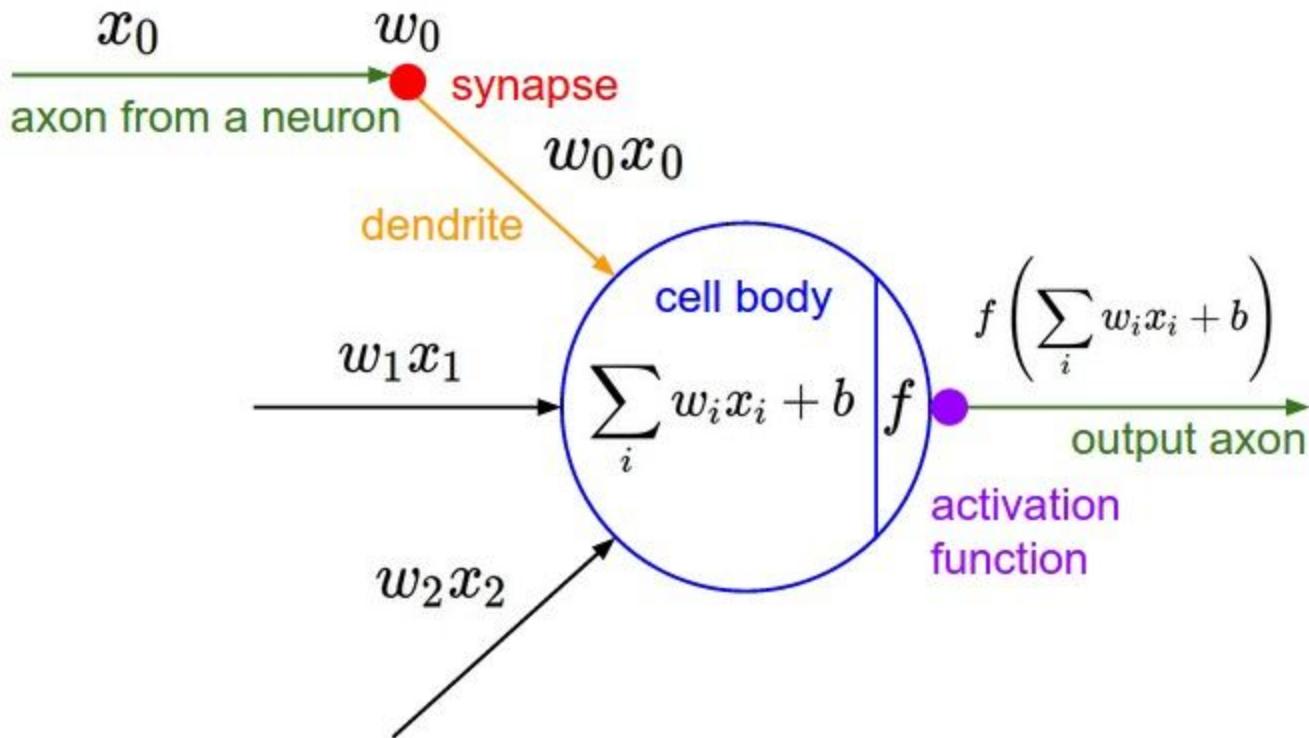
# Neural Networks

---

# Biological Neuron



# Artificial Neuron

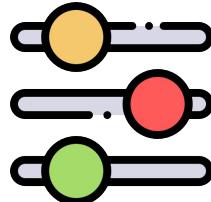


# Parametric Models

Neurons are controlled by parameters:

- Weight matrix  $W$
- Bias vector  $b$

Joining multiple neurons creates a parametric model  $m = f(x; \theta) = Wx + b$   
with parameters  $\theta = (W, b)$



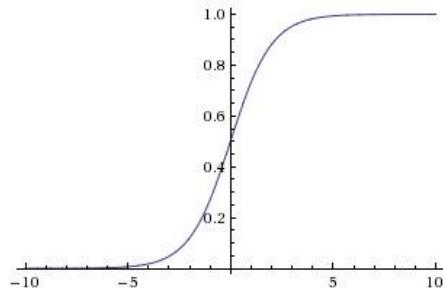
# Activation Function

- Also called non-linearities
- Decide whether neuron should be activated or not

$$Y = \text{Activation}(\sum(\text{weight} * \text{input}) + \text{bias})$$

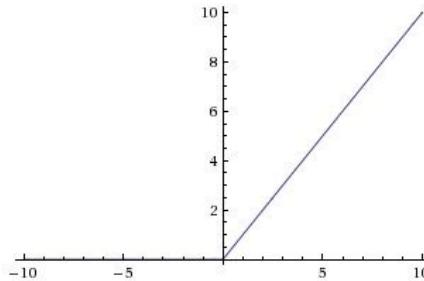
- A neural network without activation function is essentially just a linear regression model.

# Popular Activation Functions



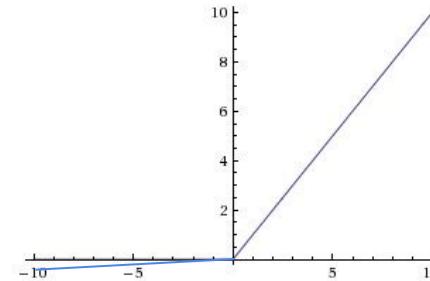
Sigmoid

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



ReLU

$$f(x) = \max(0, x)$$



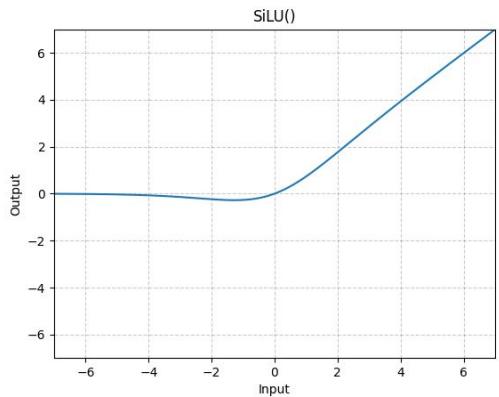
Leaky ReLU

$$f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$$

- Sigmoid inspired by nature
- ReLU very efficient to compute but has problem of dying neurons
- Leaky ReLU tries to mitigate dying neurons issue by propagating a small signal

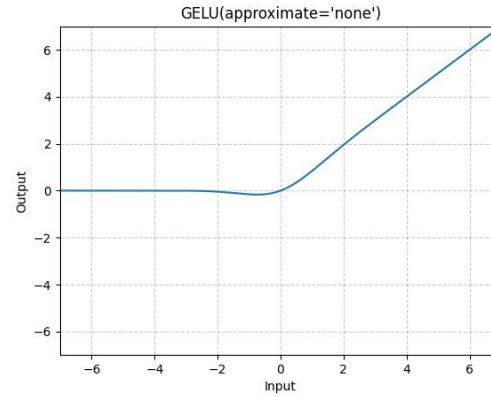
Recommendation: Never use sigmoid, try ReLU, if dying neurons are a concern, use LeakyReLU or other activation functions.

# Other Activation Functions



**SiLU**

$$f(x) = x * \text{sigmoid}(x) = x / (1 + e^{-x})$$



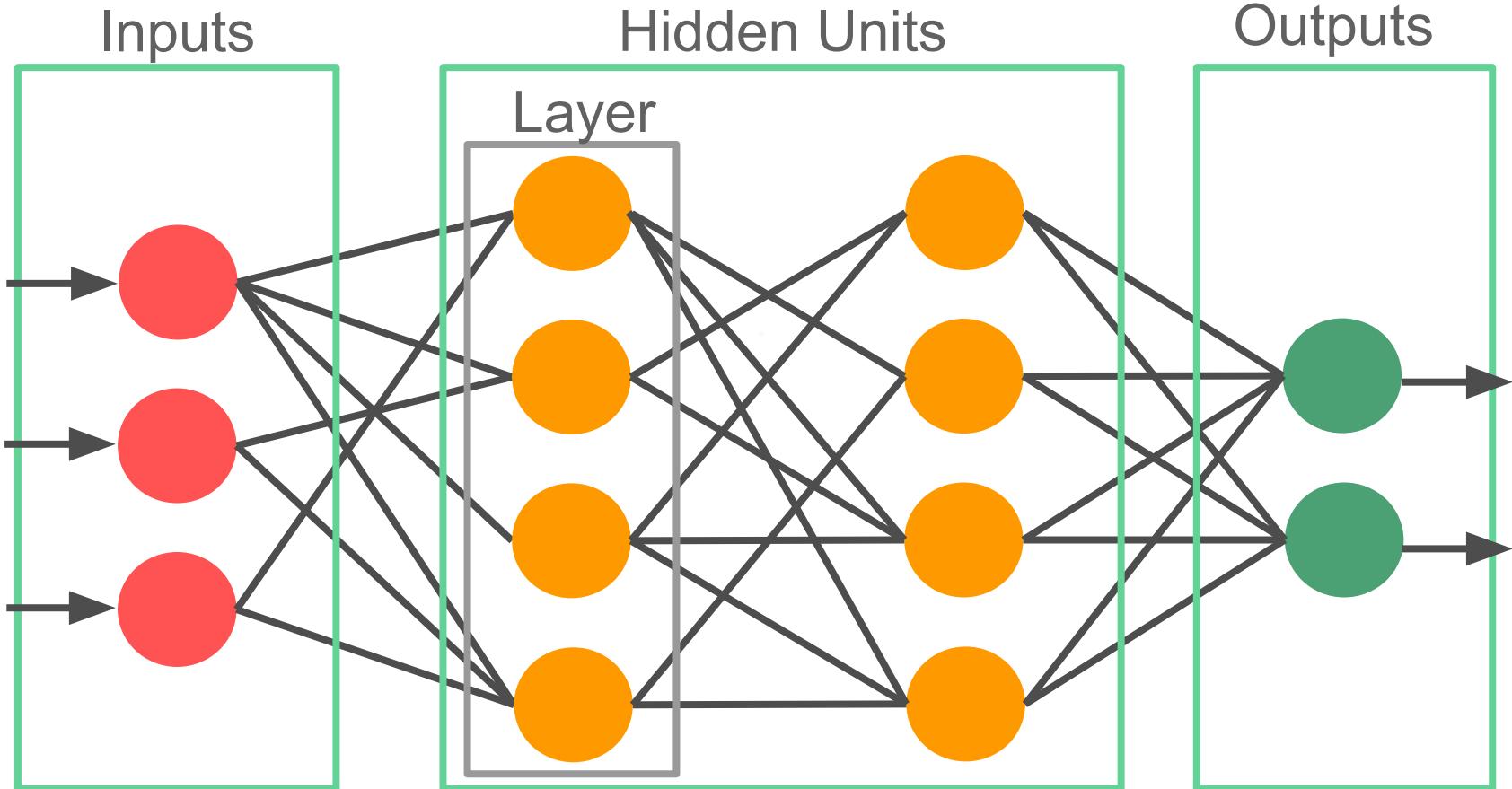
**GELU**

$$f(x) = x * (\Phi(x)) = x / (1 + e^{-1.702x})$$

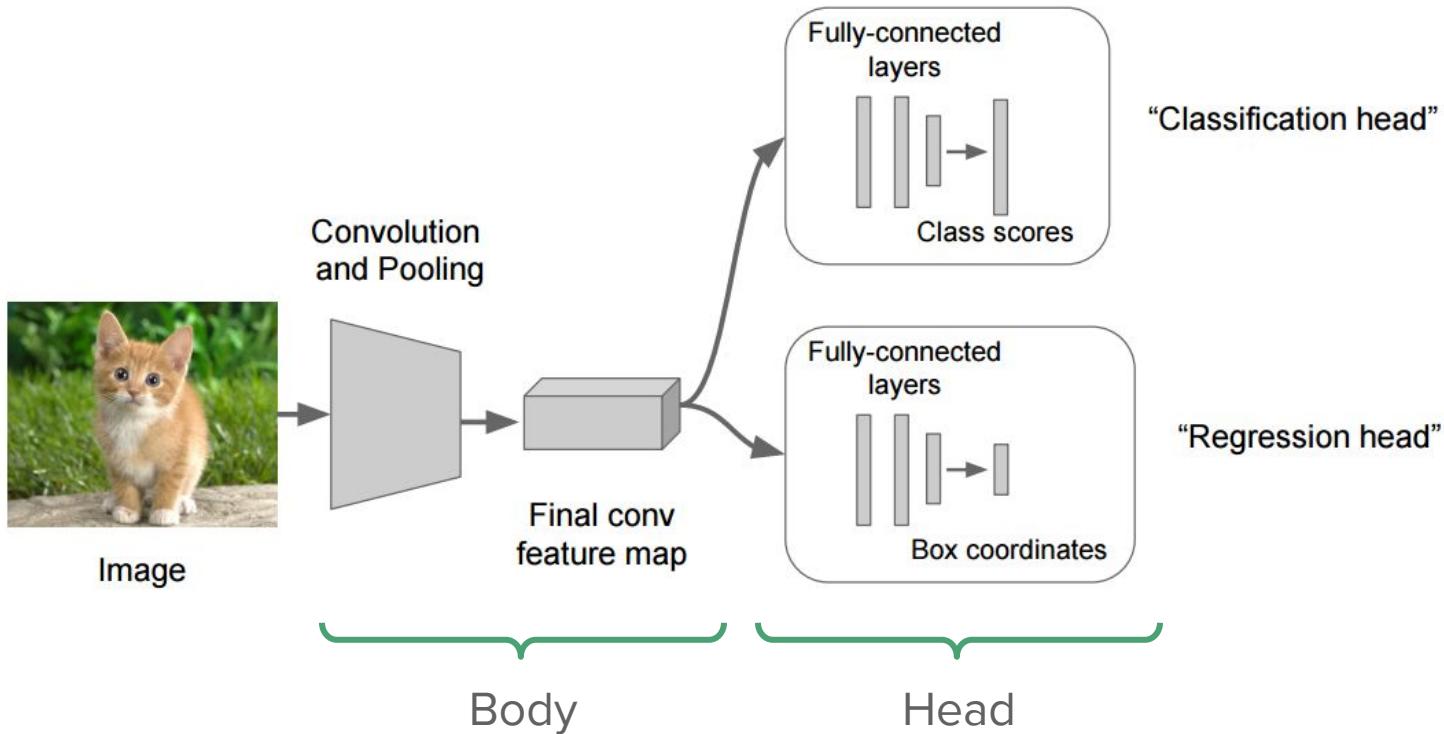
Alternatives to RELU have shown even better results due to smoother loss surface:

- Sigmoid Linear Unit (SiLU), aka. “Swish”
- Gaussian Error Linear Units (GELU)

# Neural Network



# Common Neural Network Structure



# Loss Functions (aka. Costs)



How are we doing?

---

# Loss Function

- Depending on problem, different loss functions are required

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

Mean Squared Error

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

Mean Absolute Error

Regression

$$CrossEntropyLoss = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

Classification

# Cross Entropy Loss Example

$$CrossEntropyLoss = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

```
target = [0.0, 0.0, 0.0, 1.0]
```

```
good_prediction = [0.01, 0.01, 0.01, 0.96]
```

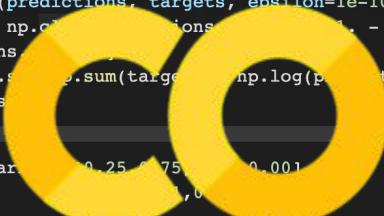
```
poor_prediction = [0.25, 0.25, 0.25, 0.25]
```

```
cross_entropy(good_prediction, target) = 0.04
```

```
cross_entropy(poor_prediction, target) = 1.39
```

# Cross Entropy Loss Example

## Cross-Entropy in Action

A screenshot of a Jupyter Notebook interface titled "02 - Cross Entropy.ipynb". The code cell at index [10] contains a function definition for cross entropy loss. The code cell at index [15] creates sample data for predictions and targets. The final code cell at index [16] calls the function and prints the result. A large graphic of two interlocking yellow rings is centered over the code area.

```
[10]: import numpy as np

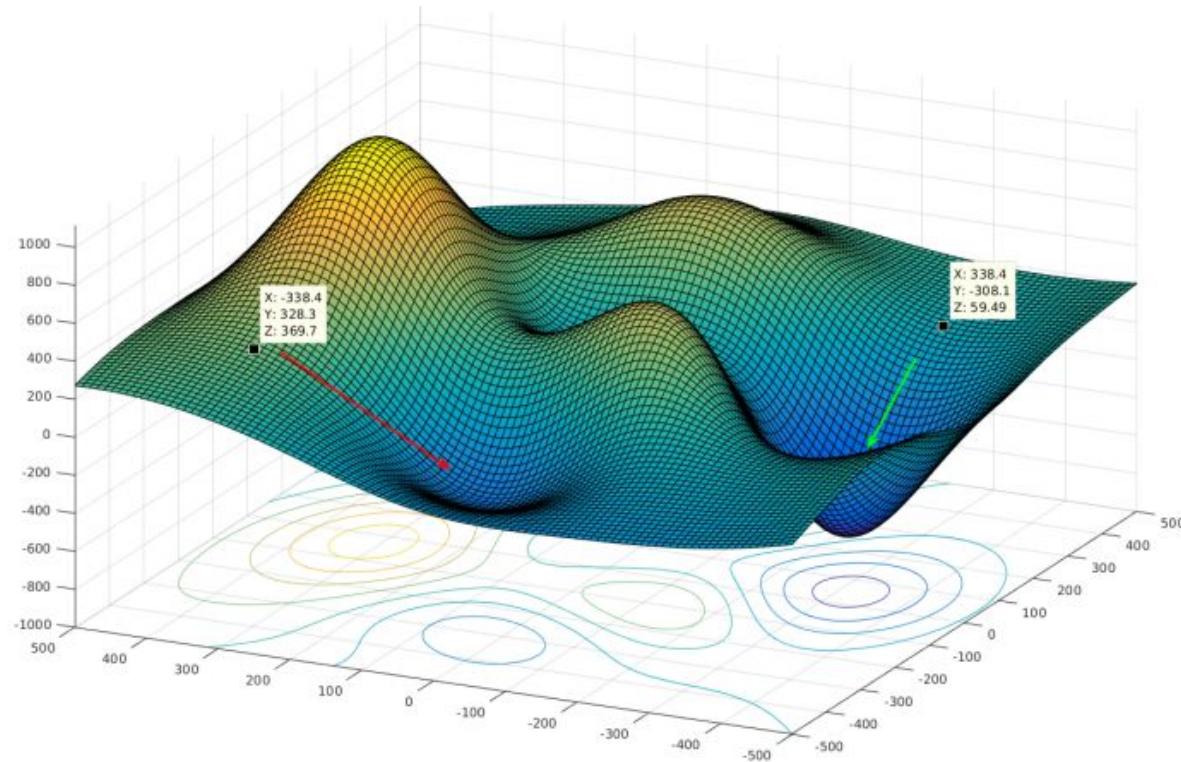
{x}     def cross_entropy(predictions, targets, epsilon=1e-10):
        predictions = np.clip(predictions, epsilon, 1. - epsilon)
        N = predictions.shape[0]
        ce_loss = -np.sum(targets * np.log(predictions + epsilon))/N
        return ce_loss

[15]: predictions = np.array([0.25, 0.75, 0.001, 0.001])
       targets = np.array([[0,0,0,1], [0,0,0,1]])

[16]: cross_entropy_loss = cross_entropy(predictions, targets)
       print ("Cross entropy loss is: " + str(cross_entropy_loss))

Cross entropy loss is: 5.776863521464035
```

# Loss Function



# Optimization



How to get to the valley?

---

# Gradient Descent

- Nonlinear optimization algorithm
- Most popular algorithm in Deep Learning



# Gradients

Let  $f(x_1, x_2, \dots, x_n)$  be a differential, real-valued function

- You can compute the partial derivative  $f_{x_i}$  of  $f$  with respect to  $x_i$ 
  - This encodes how fast  $f$  changes with  $x_i$
- Gradient  $\nabla f$  is vector of all partial derivatives of  $f$ 
  - This encodes how fast  $f$  changes with all arguments at some location  $x$



# Gradient Descent

Iterative algorithm on loss function  $L(\theta)$ :

- Compute gradient  $\theta' = \nabla L(\theta)$
- Update parameters  $\theta = \theta - \alpha\theta'$

with learning rate  $\alpha > 0$

**Requires that loss function is differentiable and real-valued**

# Gradient Descent

Stochastic gradient descent:  $\theta = \theta - \alpha \nabla L(\theta; x^{(i)}, y^{(i)})$

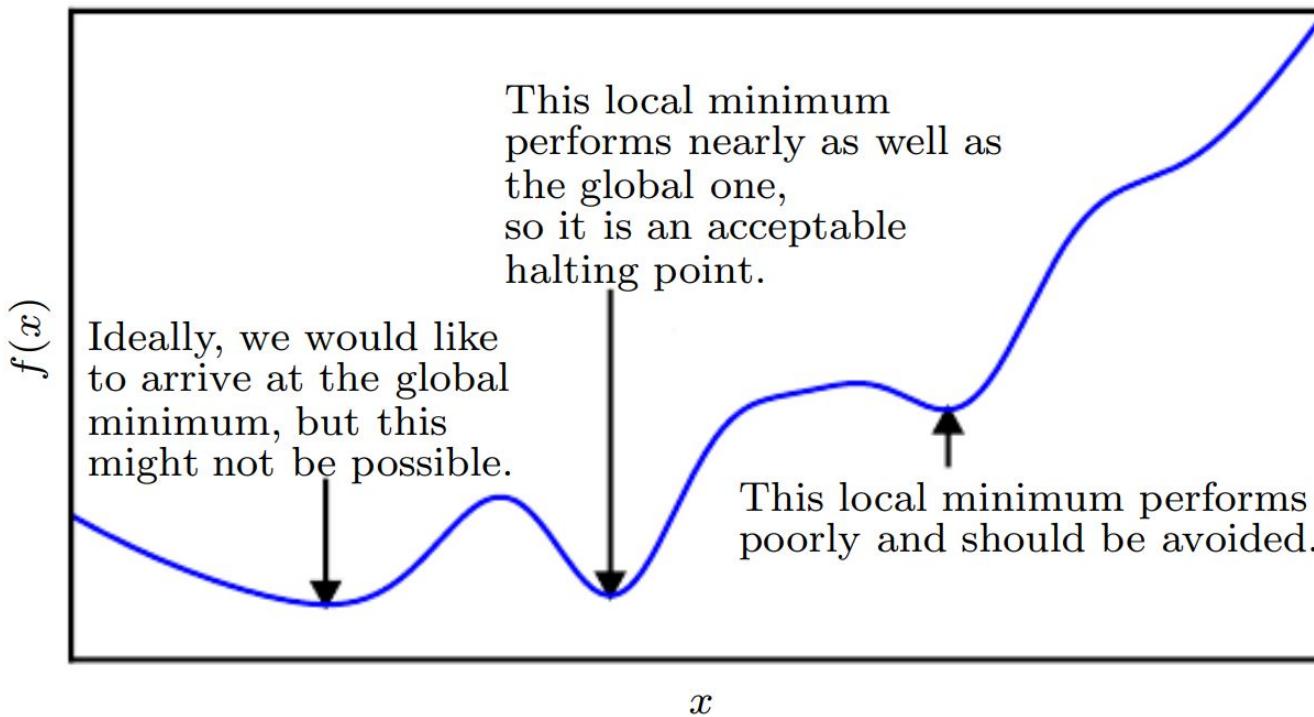
Batch gradient descent:  $\theta = \theta - \alpha \nabla L(\theta)$

Mini-batch gradient descent:  $\theta = \theta - \alpha \nabla L(\theta; x^{(i:i+n)}, y^{(i:i+n)})$

Challenges:

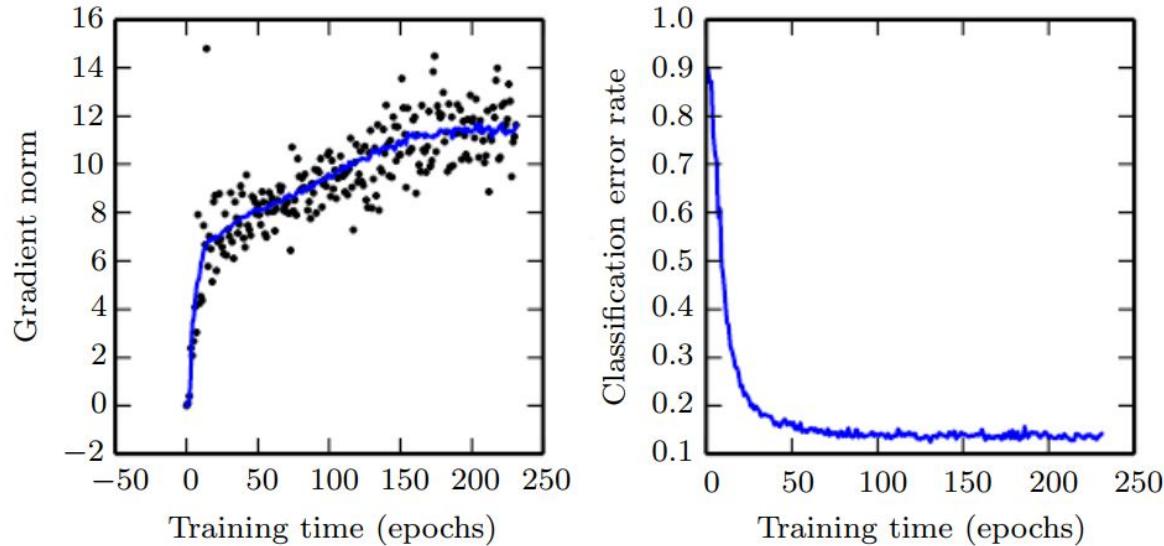
- Finding a proper learning rate
- Same learning rate applied to all parameter updates
- Potential critical points

# Global and Local Minima



# Critical points

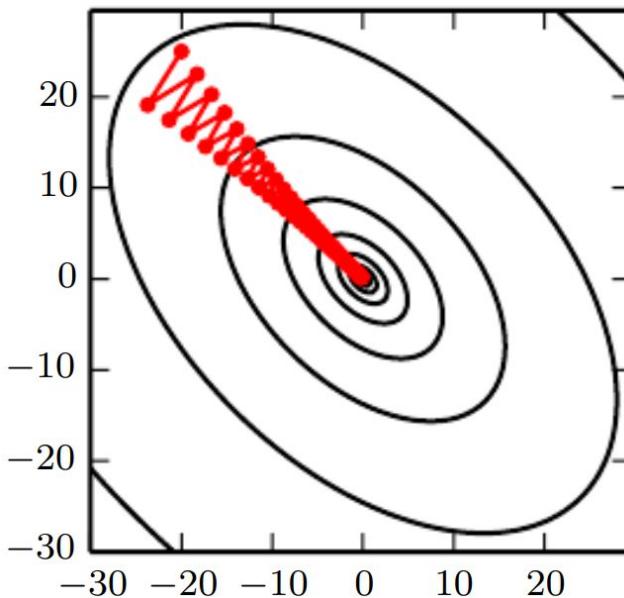
Gradient descent often does not arrive at a critical point of any kind.



# Gradient Descent Optimizations - Momentum

Loss function can have canyon-like curvature

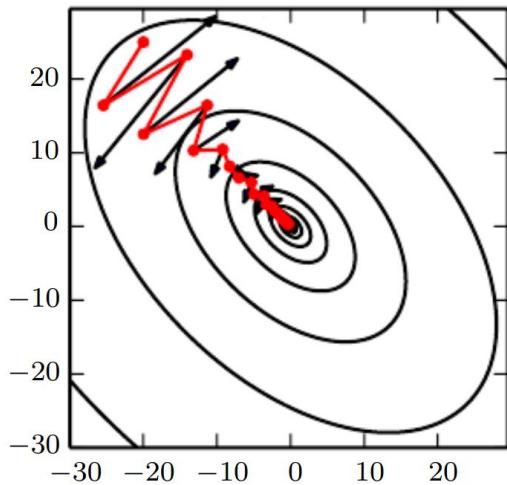
Gradient bounces between canyon walls



# Gradient Descent Optimizations - Momentum

Gradient descent with momentum by introducing a velocity  $v$ :

- Update velocity  $v = \beta v - \alpha \nabla L(\theta)$
- Update parameters  $\theta = \theta + v$
- With momentum  $\beta \in [0, 1)$

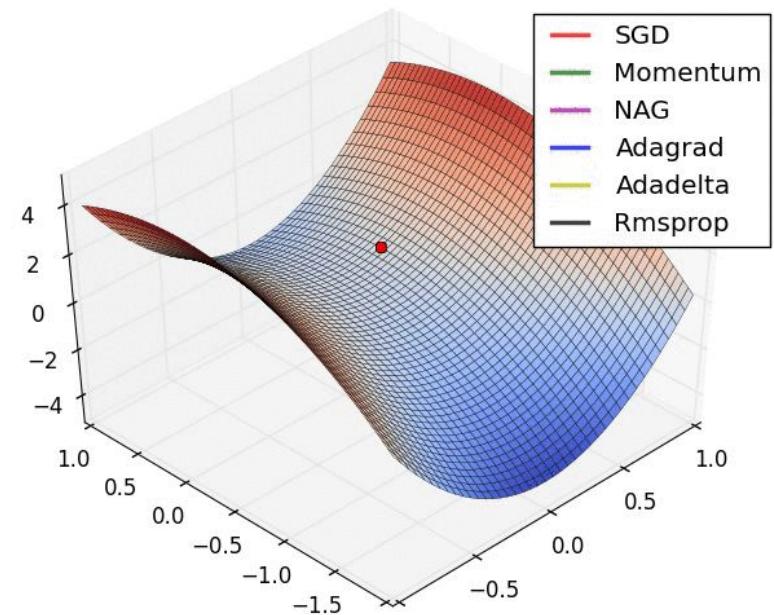
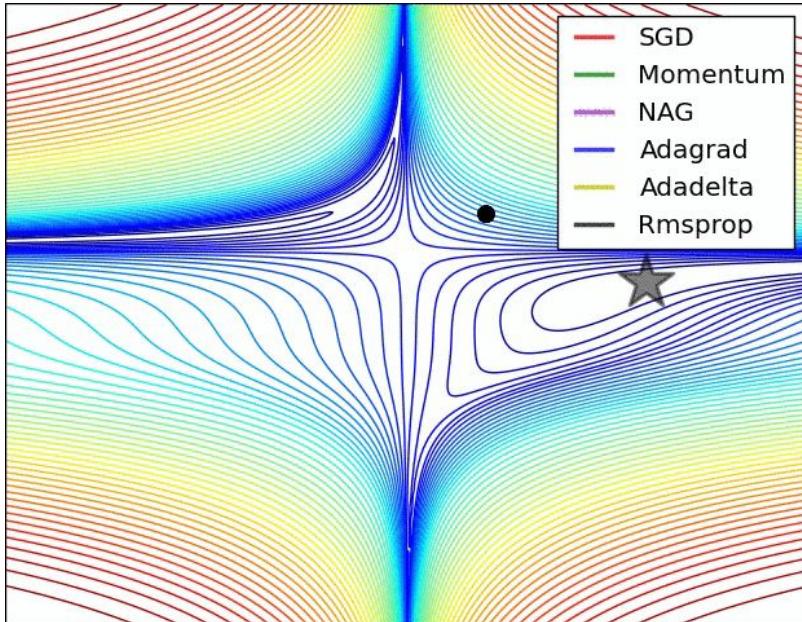


# Adaptive Algorithms for Gradient Descent

- **Adagrad:** Adapts learning rate to the parameters (low learning rate for frequent events, high learning rate for infrequent events)
- **Adadelta:** Extension of Adagrad + aggressively decrease learning rate
- **RMSprop:** Very similar to Adadelta
- **Adam:** Adapts learning rate + stores decaying average of past gradients, similar to momentum
- **AMSGrad:** Adapts learning rate + maximum of past squared gradients instead of exponential average to update parameters

**Beware:** The benefits of adaptive algorithms are still controversial.

# Adaptive Algorithms for Gradient Descent



# Backpropagation

Big question: How to compute  $\nabla L(\theta)$ ?

- Function  $f$  is composed of other functions
- Loss function is again a graph for which we need the derivatives

Backpropagation recursively applies the chain rule to compute the derivatives for that graph.

$$f^1(x_1, x_2) = x_1 x_2$$

$$f_{x_1}^1(x_1, x_2) = x_2 \text{ and } f_{x_2}^1(x_1, x_2) = x_1$$

$$f^2(x_1, x_2) = x_1 + x_2$$

$$f_{x_1}^2(x_1, x_2) = 1 \text{ and } f_{x_2}^2(x_1, x_2) = 1$$

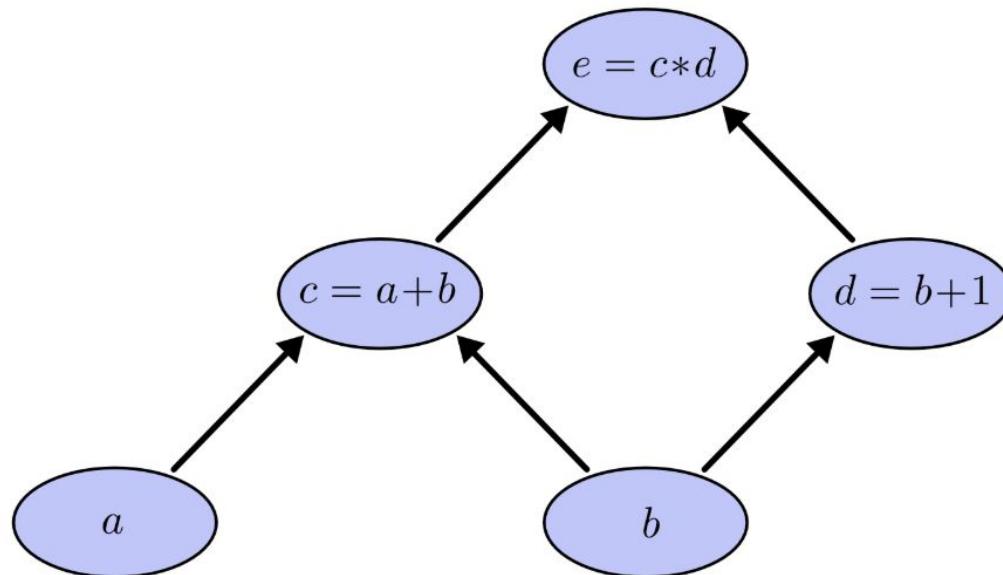
$$f^3(x_1, x_2) = \max(x_1, x_2)$$

$$f_{x_1}^3(x_1, x_2) = 1 \text{ if } x_1 \geq x_2 \text{ else } 0$$

$$f_{x_2}^3(x_1, x_2) = 1 \text{ if } x_2 \geq x_1 \text{ else } 0$$

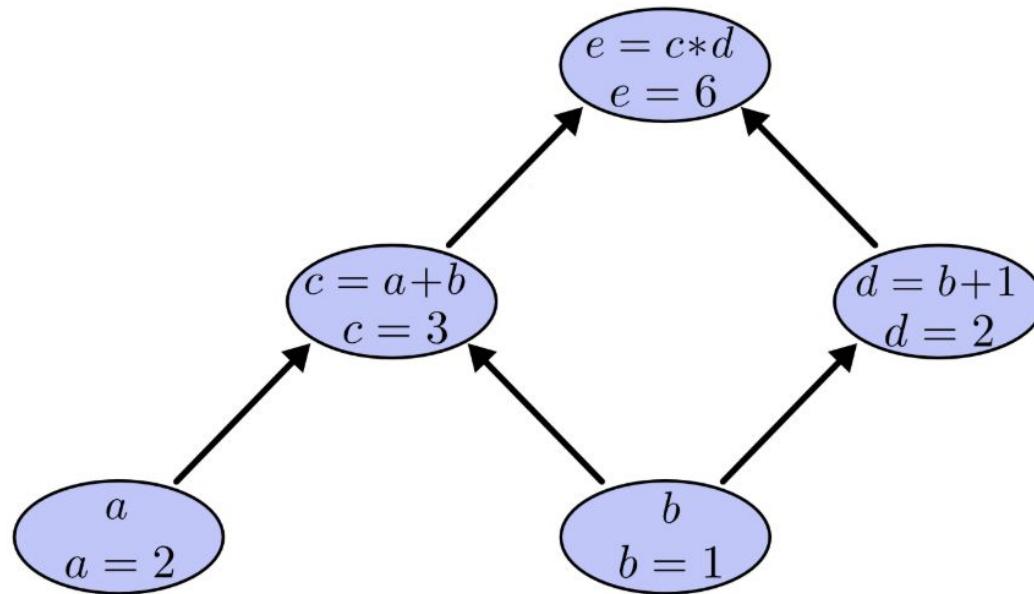
# Backpropagation

Sample expression  $e(a,b) = (a+b)(b+1)$



# Backpropagation

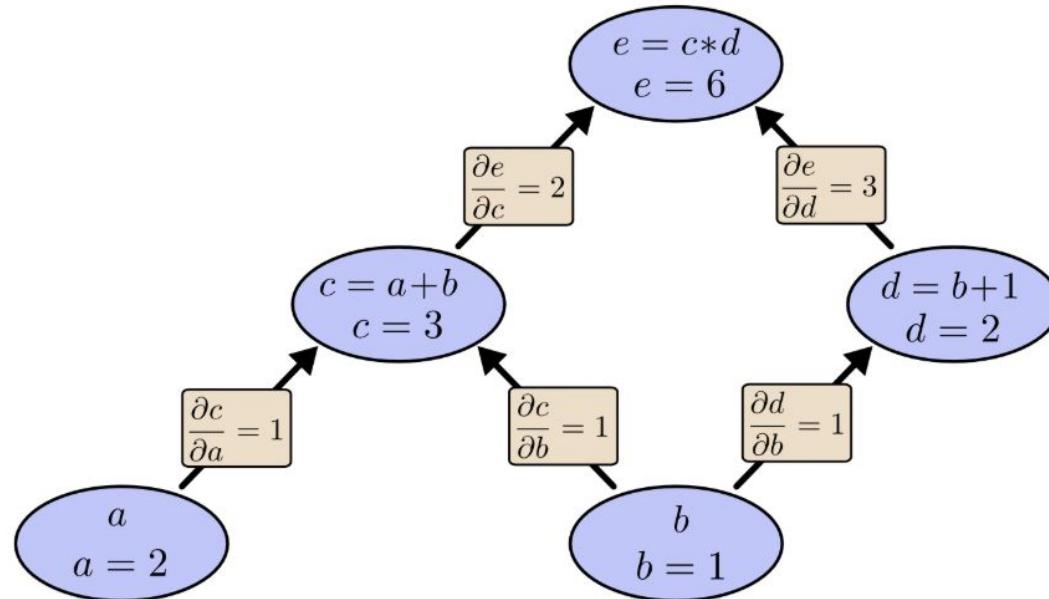
Sample expression  $e(a,b) = (a+b)(b+1)$ , for  $a=2$  and  $b=1$



# Backpropagation

Compute local gradients independently

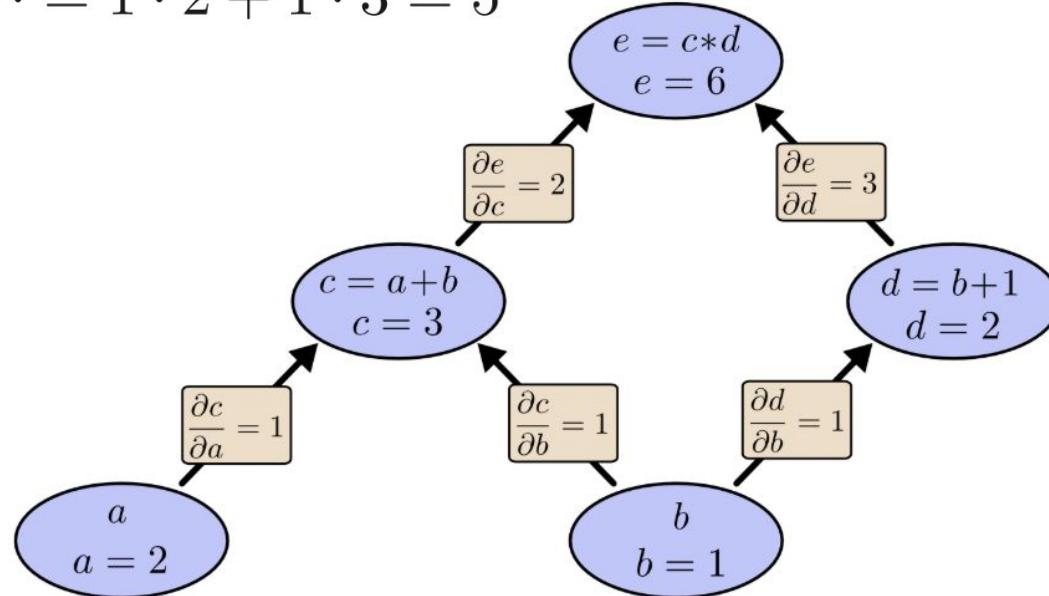
Compute remaining gradients from local ones using multivariate chain rule



# Backpropagation

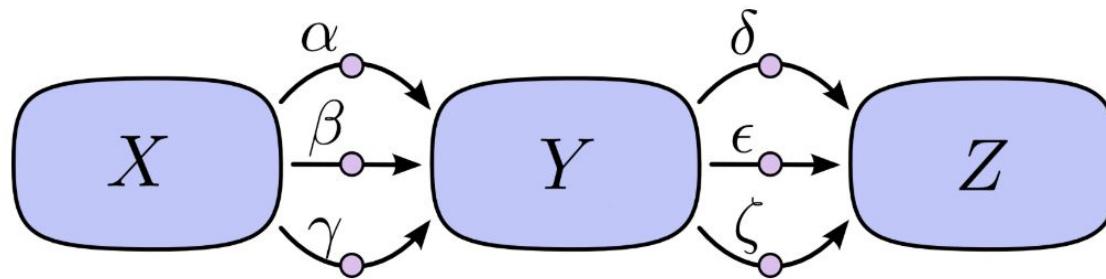
$$e_a(2, 1) = c_a(2, 1) \cdot e_c(2, 1) = 1 \cdot 2 = 2$$

$$e_b(2, 1) = \dots = 1 \cdot 2 + 1 \cdot 3 = 5$$



# Path Factorization

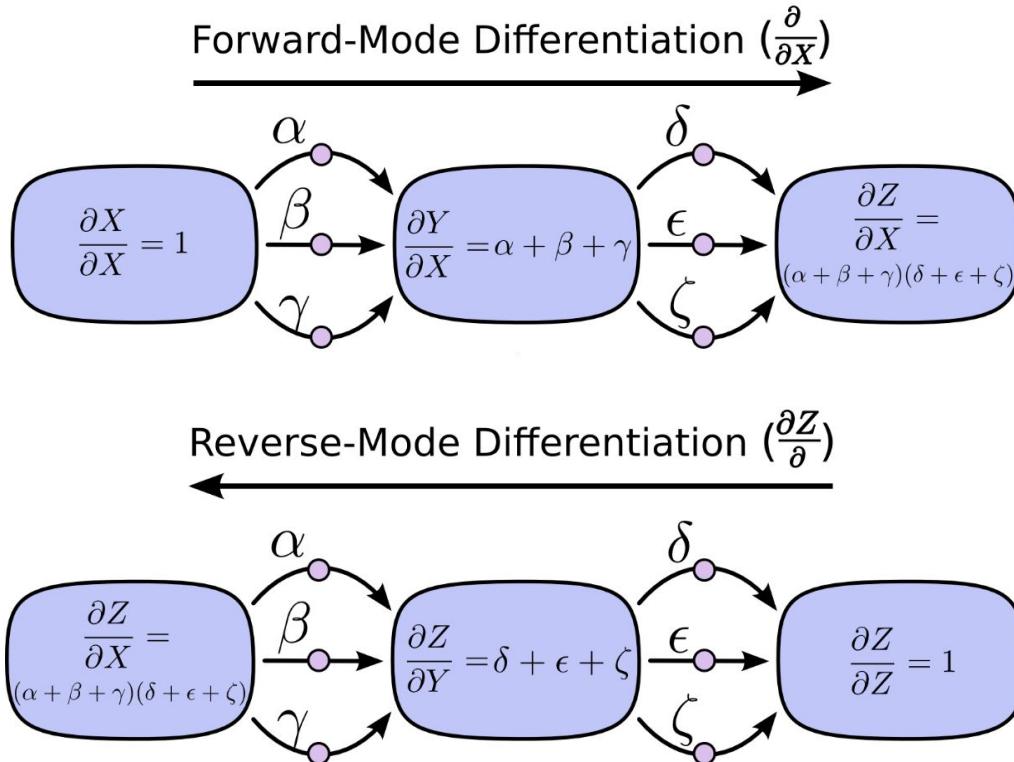
Summing over all paths leads to combinatorial explosion



$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

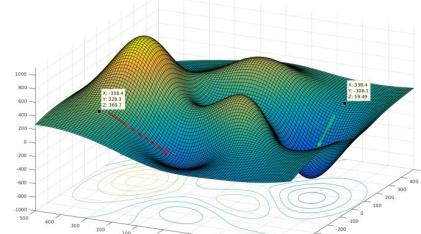
$$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$$

# Forward-mode and Reverse-mode differentiation



# Summary

- Neural Networks are inspired by the way how human brain works
  - Parametric model
  - Put a weight on each input
  - Activation function to introduce non-linearity
- Model learns on training set to predict samples from test set
  - Both sets must have the same underlying distribution
- Loss function tells us “how good we are doing”
  - Computing derivatives wrt. parameters creates topology
- Gradient descent is an efficient way to navigate through that terrain



# Literature

1. Pramerdorfer, [Deep Learning for Visual Computing](#), 2016.
2. Lopez et al. [Skin lesion classification from dermoscopic images using deep learning techniques](#), 2017.
3. Fei fei Li et al. [Deep Learning for Visual Computing](#)
4. Araujo dos Santos, [Artificial Intelligence](#)
5. Deep Learning: <https://www.deeplearningbook.org>
6. Ruder. [An overview of gradient descent optimization algorithms](#)
7. Colah. [Calculus on Computational Graphs: Backpropagation](#), 2015.
8. Ramachandran et al. [Searching for Activation Functions](#), 2017.
9. Hendrycks et al. [Gaussian Error Linear Units \(GELUs\)](#), 2023.

# Icon credits

Free icons from Flaticon:

- [https://www.flaticon.com/free-icon/modeling 1373079](https://www.flaticon.com/free-icon/modeling_1373079)
- [https://www.flaticon.com/free-icon/analytics 166904](https://www.flaticon.com/free-icon/analytics_166904)
- [https://www.flaticon.com/free-icon/landing 1284826](https://www.flaticon.com/free-icon/landing_1284826)
- [https://www.flaticon.com/free-icon/dimmer 1833516](https://www.flaticon.com/free-icon/dimmer_1833516)
- [https://www.flaticon.com/free-icon/energy-class 1833531](https://www.flaticon.com/free-icon/energy-class_1833531)
- [https://www.flaticon.com/free-icon/slider-tool 941588](https://www.flaticon.com/free-icon/slider-tool_941588)
- [https://www.flaticon.com/free-icon/asking 900415](https://www.flaticon.com/free-icon/asking_900415)

Other images:

- Title image: <https://flic.kr/p/28kDqNb>
- Dog: <https://flic.kr/p/9pgKCb>
- Cat: <https://bristolcountyvet.com/wp-content/uploads/2018/11/pexels-photo-min.jpg>
- Dog 2: <https://flic.kr/p/6YZB9B>