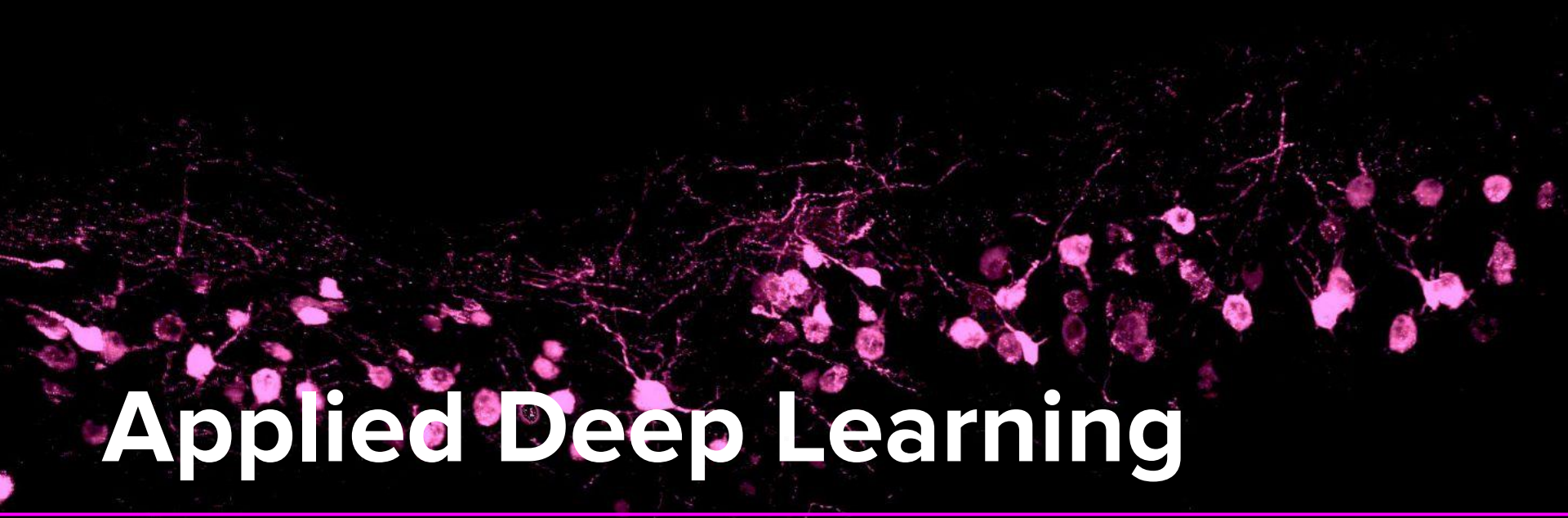


Applied Deep Learning

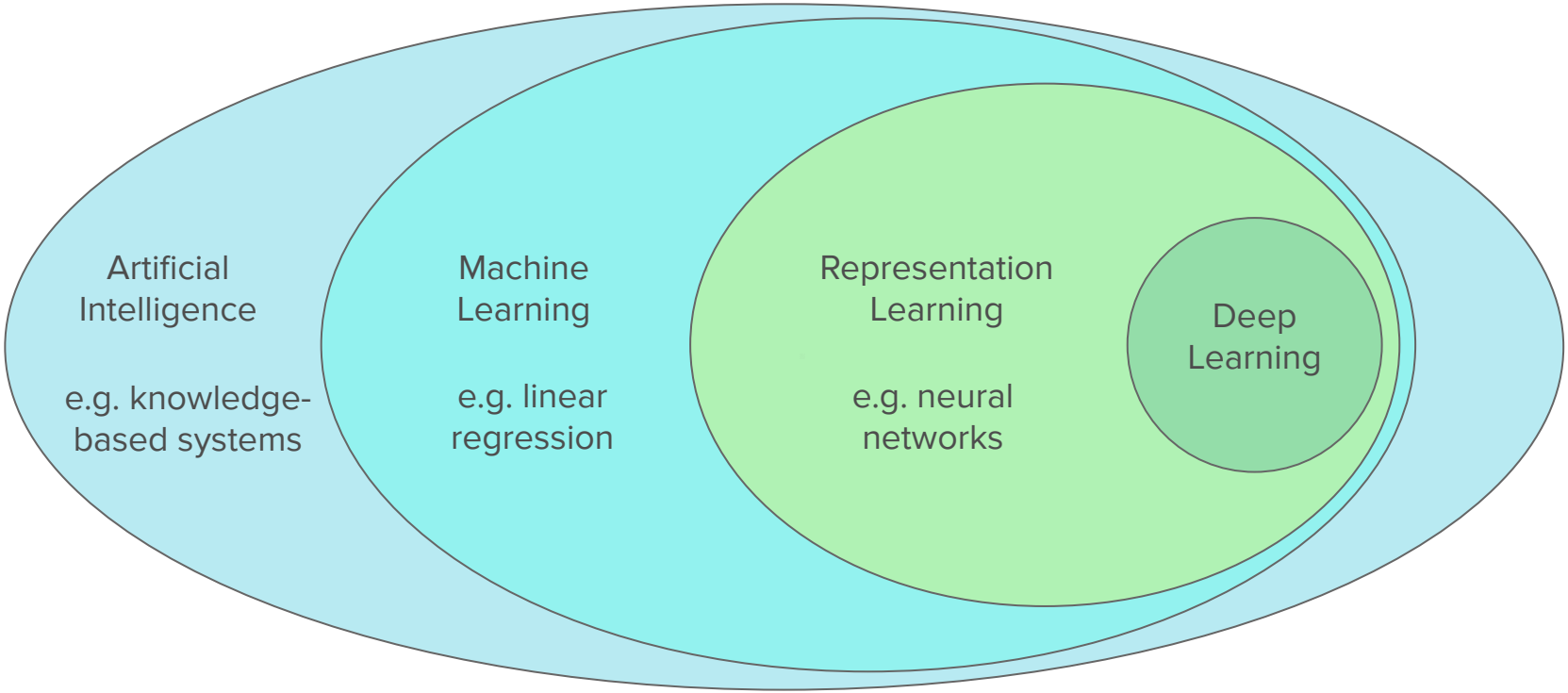
Neural Networks, Optimization, and Backpropagation

Alexander Pacha - TU Wien

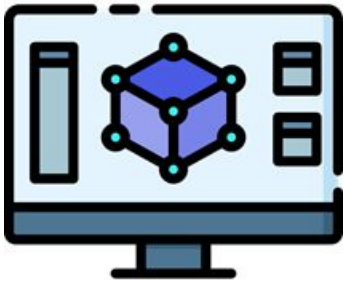
Recap

- 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?

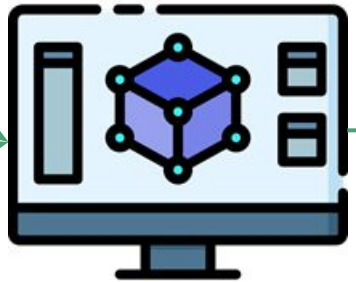
Training Set

"Dog"

"Cat"

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

Model

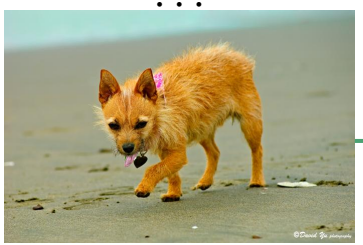


$w \in \{Dog, Cat\}$

$P(w|\mathbf{x})$

Test Set

"???"



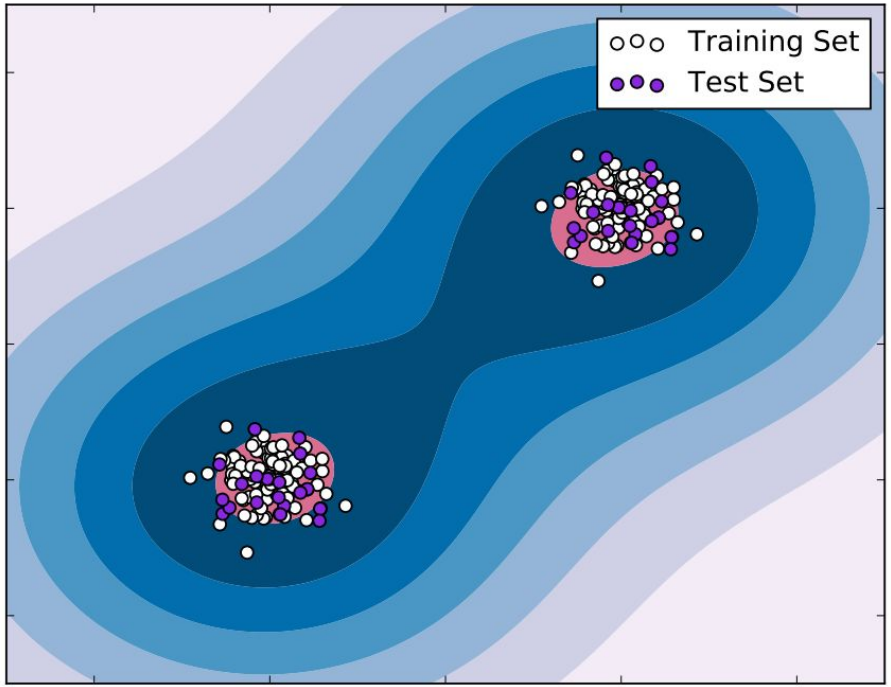
Inference

$P(dog|x) = 0.7$

$P(cat|x) = 0.3$

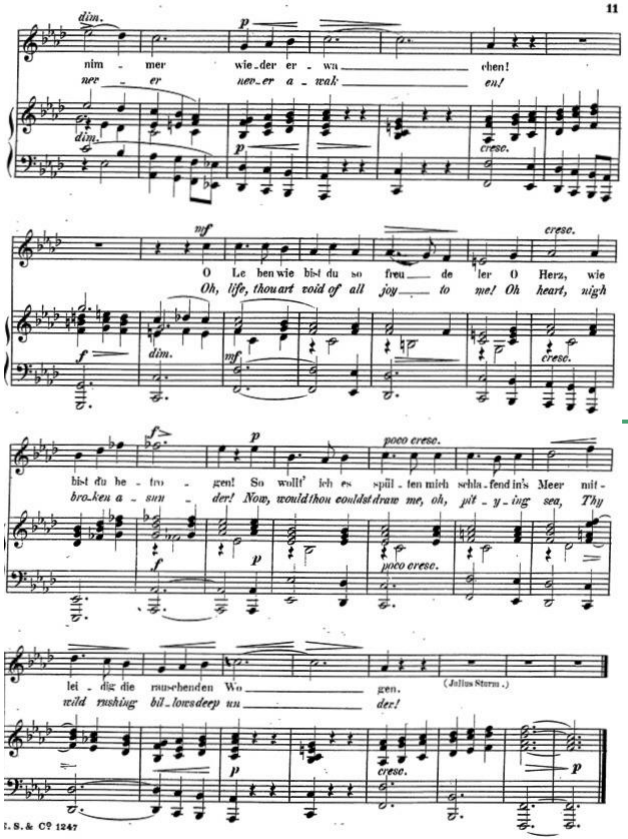
How can a model work on unseen data?

Both datasets must have similar distribution



[1]

Does it generalize?



?

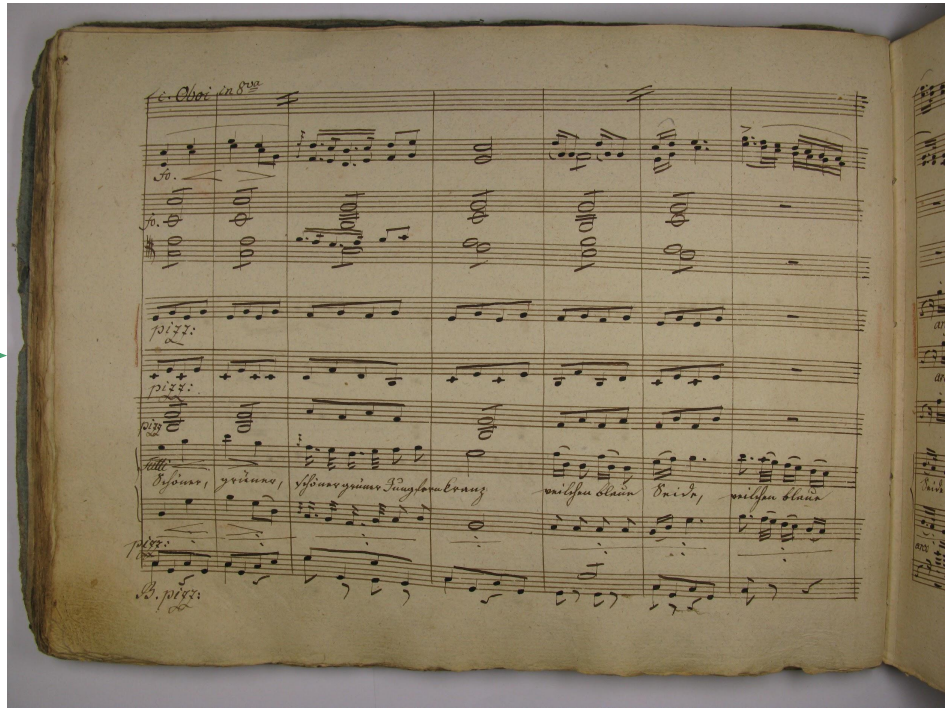
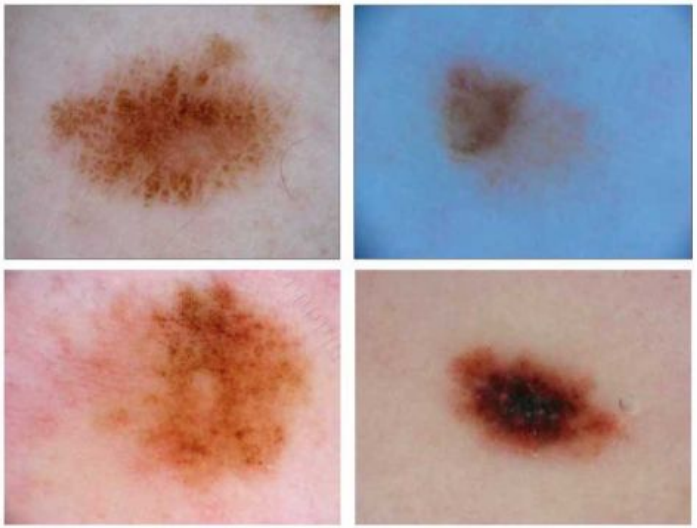
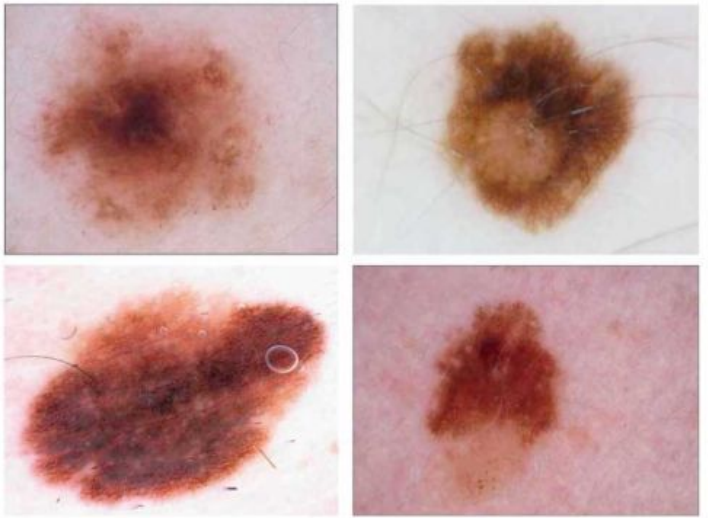


Image Classification

Melanoma



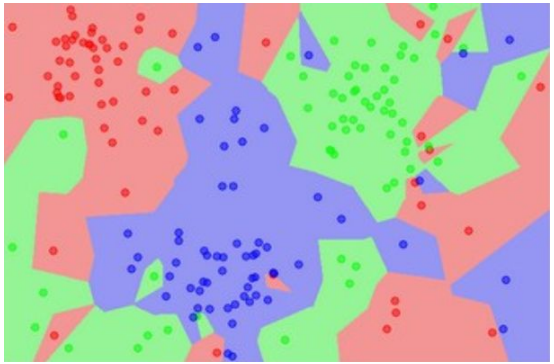
Benign



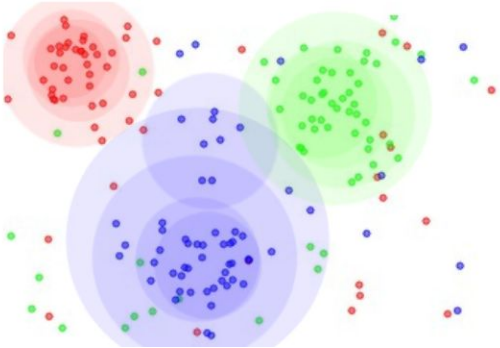
[2]

Classes of models

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



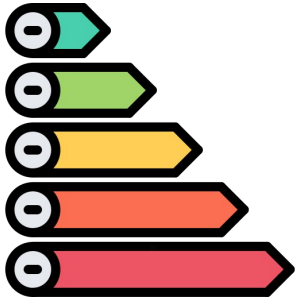
Generative models learn class-conditional densities $P(\mathbf{x}|w)$



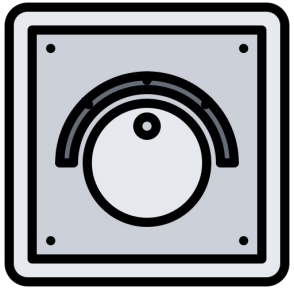
[3]

Output

- Classification problem with finite number of different labels

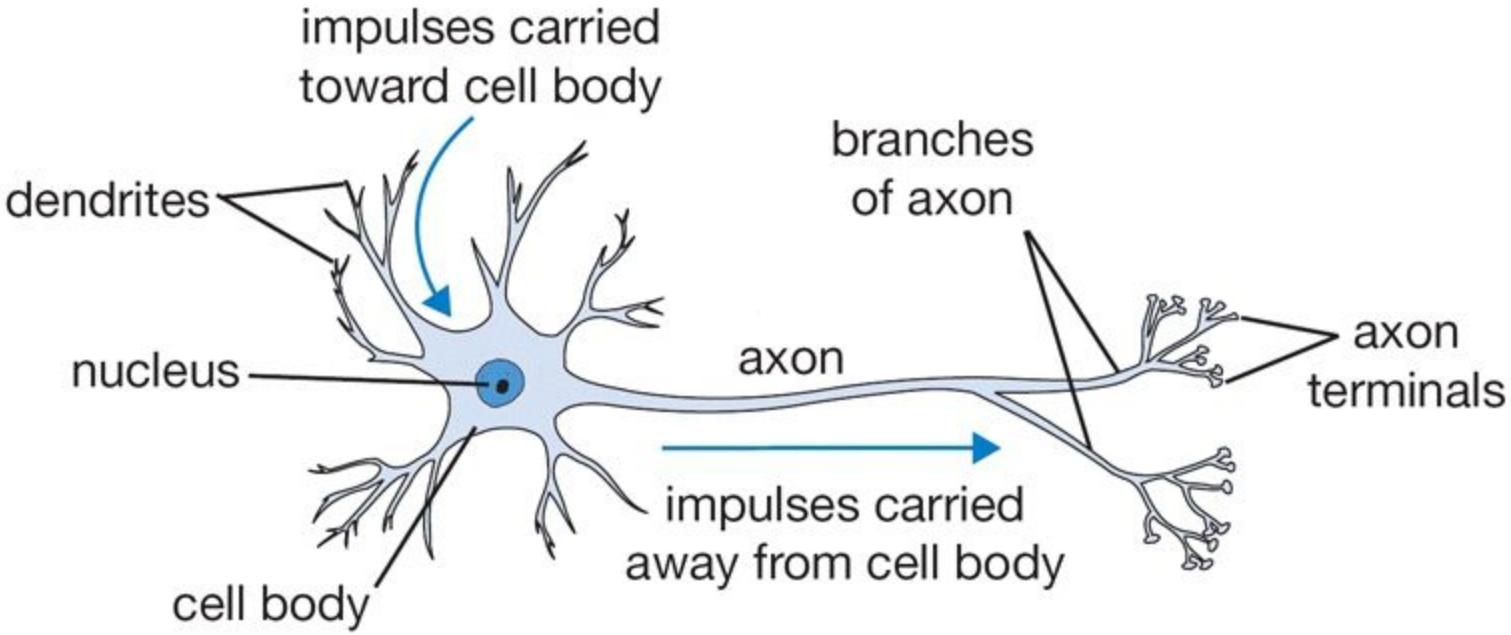


- Regression problem with a continuum of output values



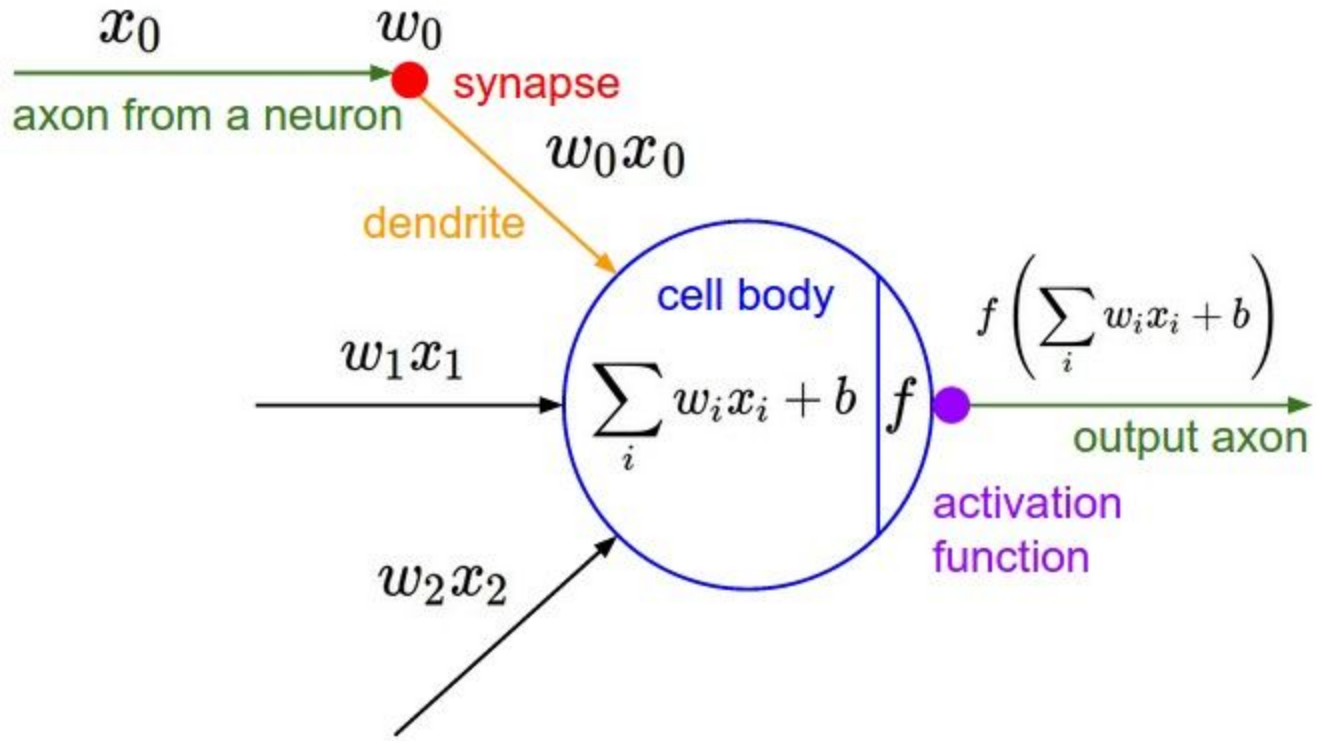
Neural Networks

Biological Neuron



[3]

Artificial Neuron



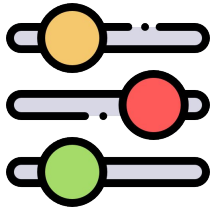
[3]

Parametric Models

Neurons are controlled by parameters:

- Weight matrix $\mathbf{W}$
- Bias vector $\mathbf{b}$

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



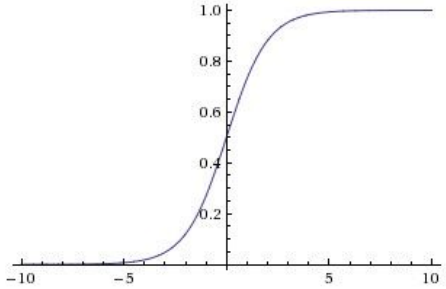
Activation Function

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

$$Y = Activation(\Sigma(weight*input) + bias)$$

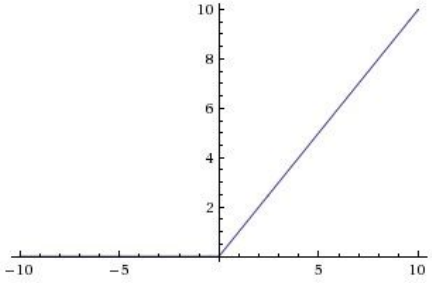
- A neural network without activation function is essentially just a linear regression model.

Popular Activation Functions



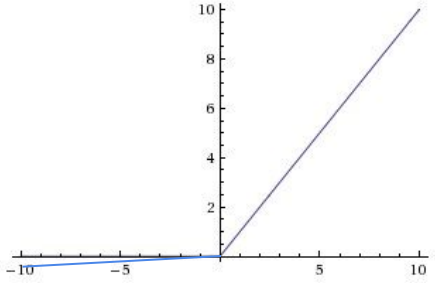
Sigmoid

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



ReLU

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



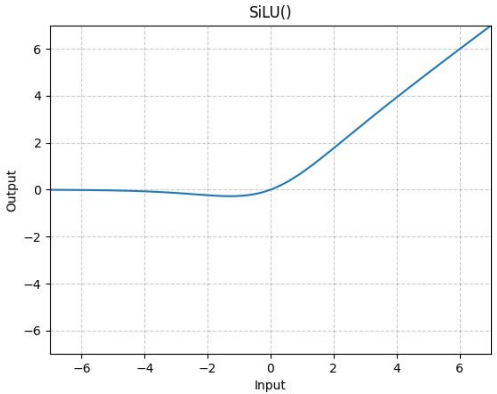
Leaky ReLU

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

- 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

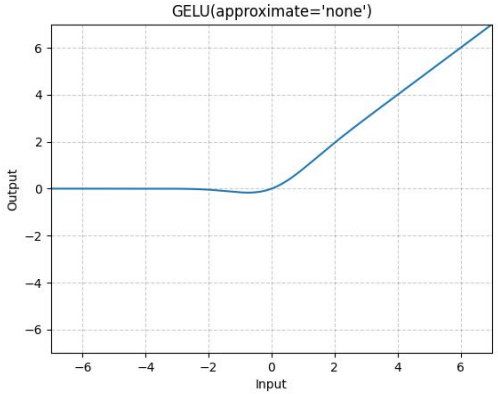
Recommendation: Never use sigmoid, try ReLU, if dying neurons are a concern, use LeakyReLU or other activation functions.

Other Activation Functions



SiLU

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



GELU

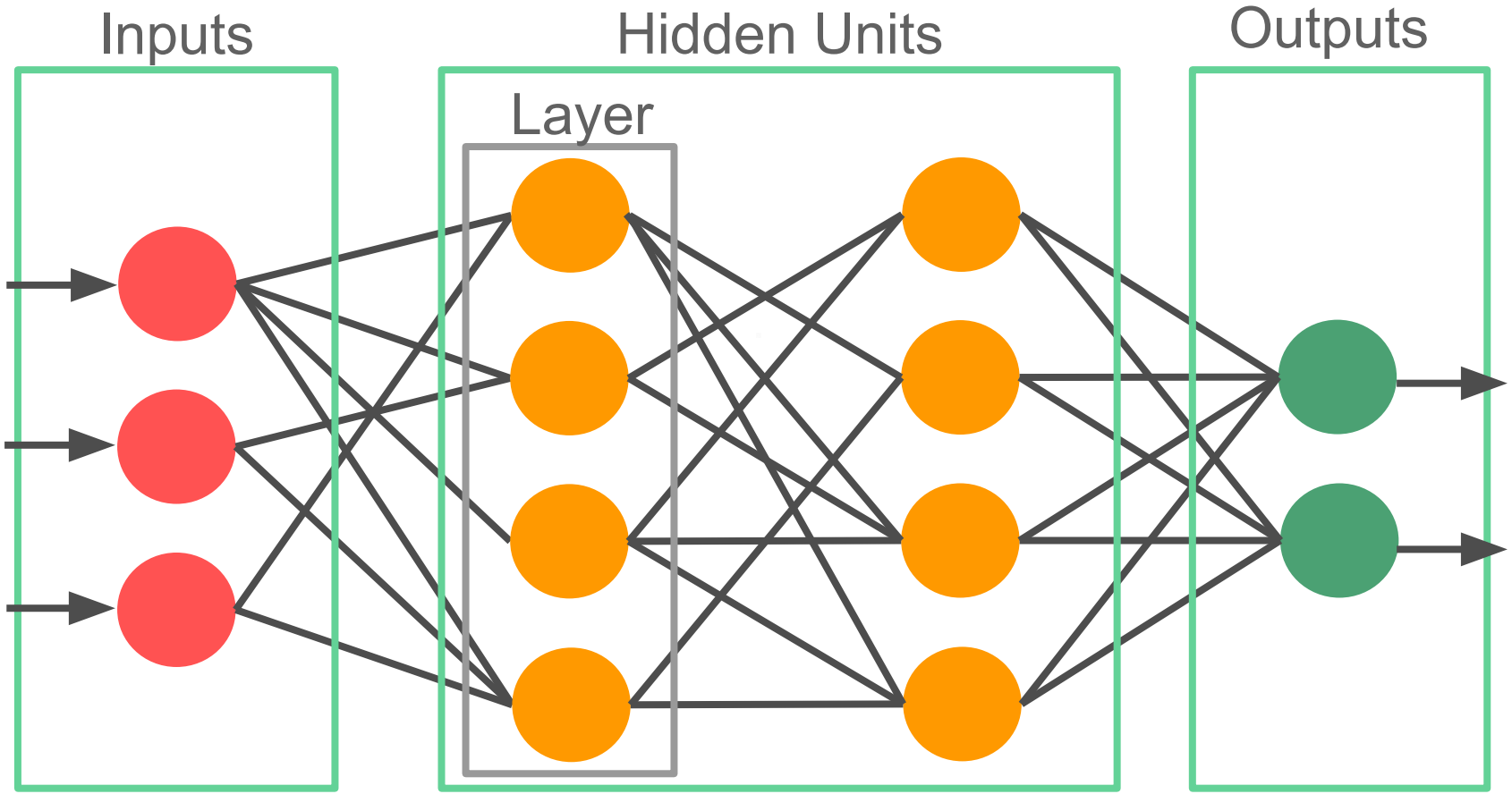
$f(x) = x*(\Phi(x)) = x/(1+e^{-1.702x})$

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

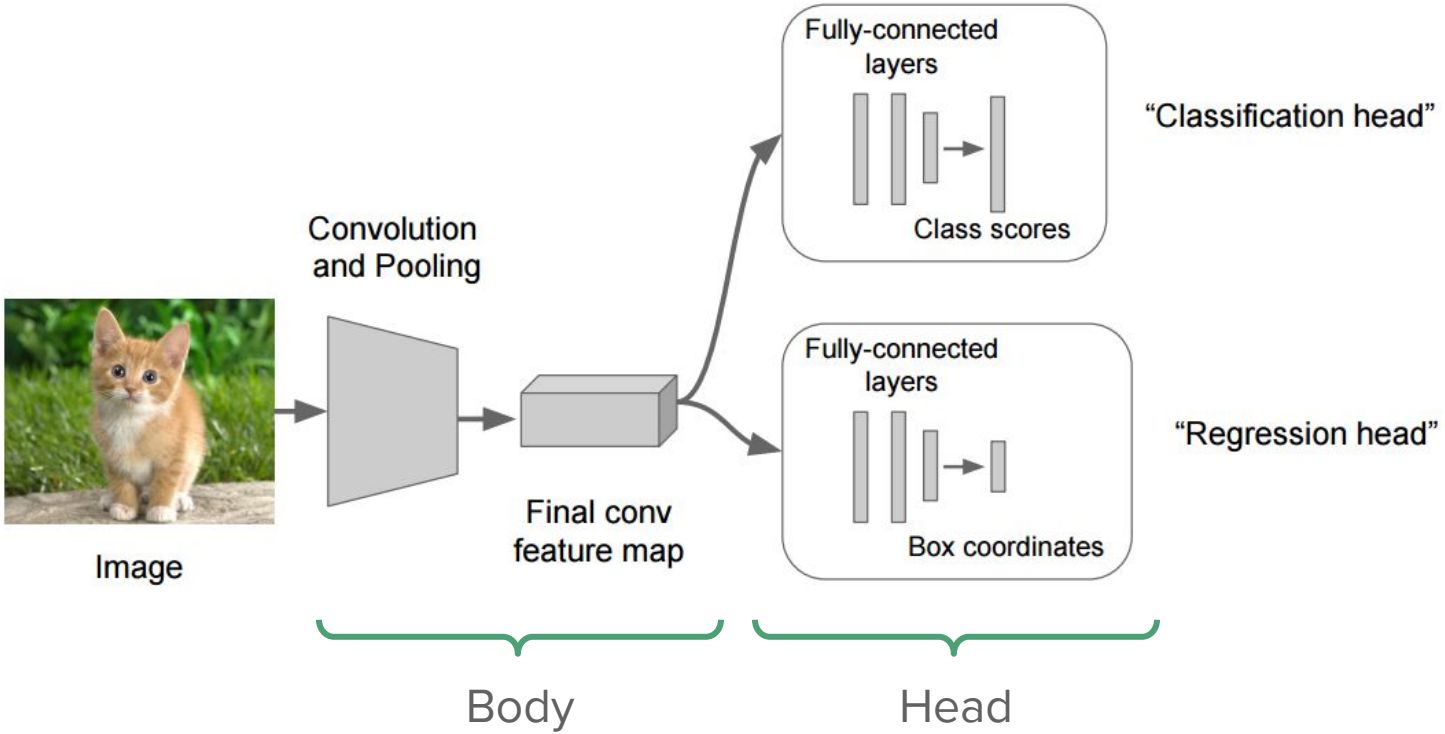
- Sigmoid Linear Unit (SiLU), aka. “Swish”
- Gaussian Error Linear Units (GELU)

[8, 9]

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

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

Mean Absolute Error

Regression

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

Classification

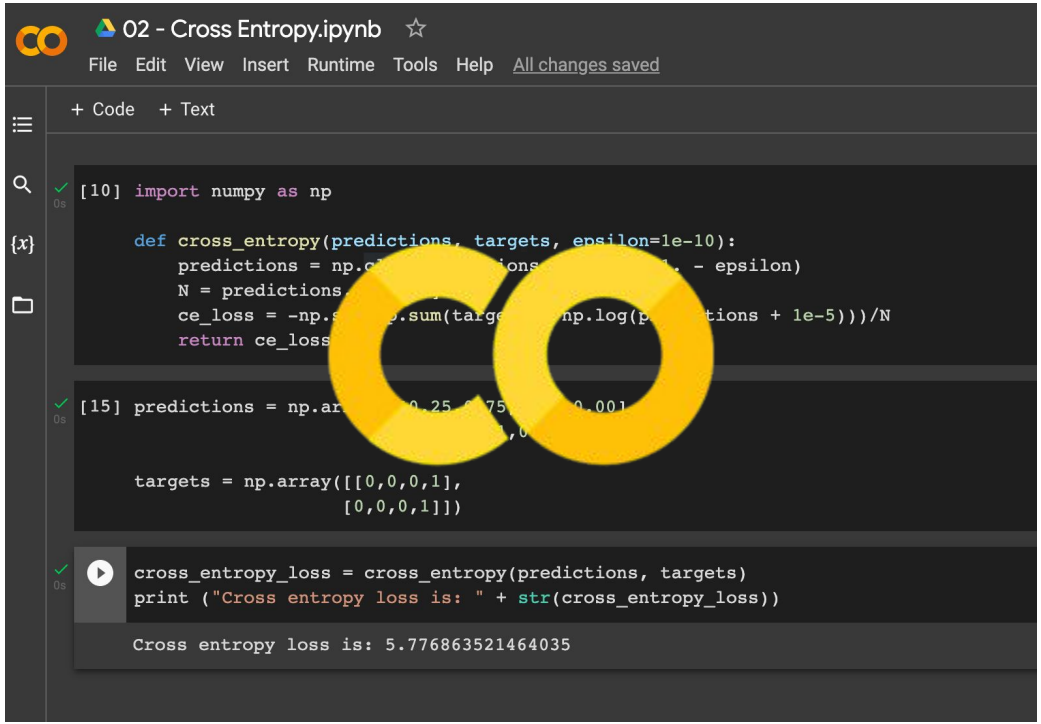
Cross Entropy Loss Example

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

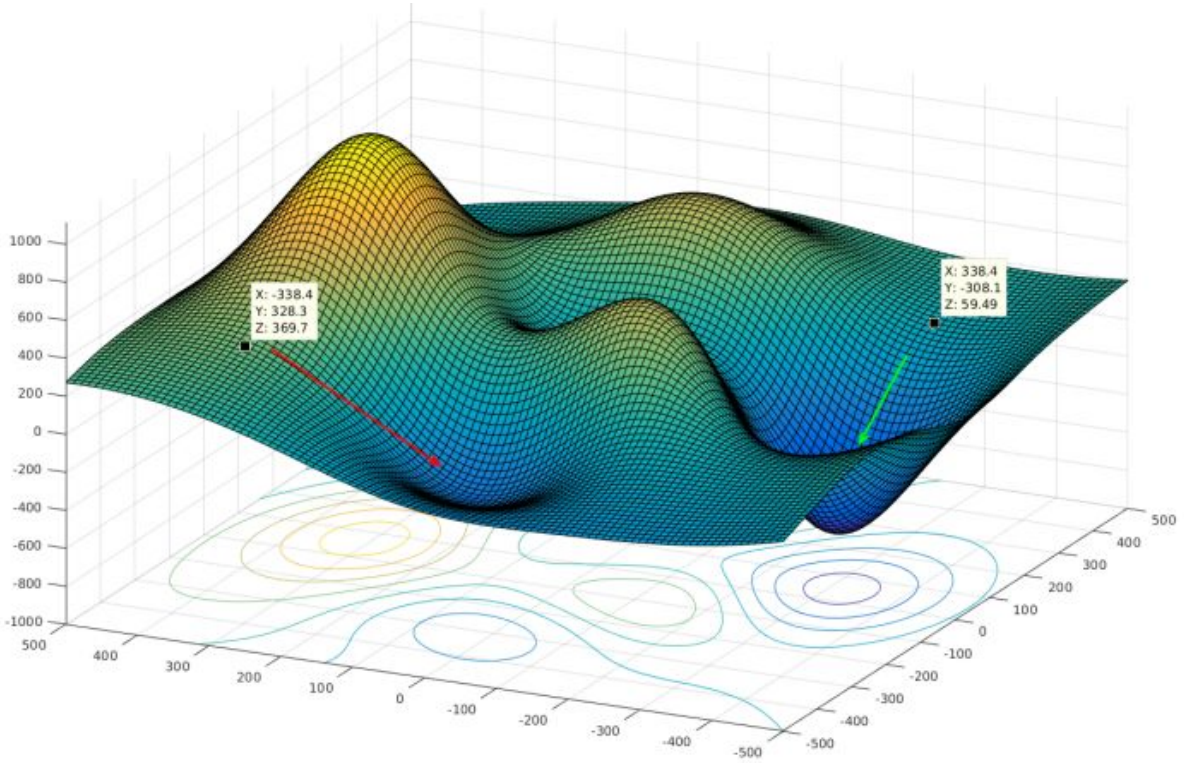
```
target = [0.0, 0.0, 0.0, 1.0]
good_prediction = [0.01, 0.01, 0.01, 0.96]
poor_prediction = [0.25, 0.25, 0.25, 0.25]

cross_entropy(good_prediction, target) = 0.04

cross_entropy(poor_prediction, target) = 1.39
```

Cross Entropy Loss Example

[Cross-Entropy in Action]



Loss Function



[4]

Optimization

How to get to the valley?

Gradient Descent

- Nonlinear optimization algorithm
- Most popular algorithm in Deep Learning

Gradients

Let $f(x_1, x_2, \ldots, x_n)$ 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 ∇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: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla L(\boldsymbol{\theta}; x^{(i)}, y^{(i)})$

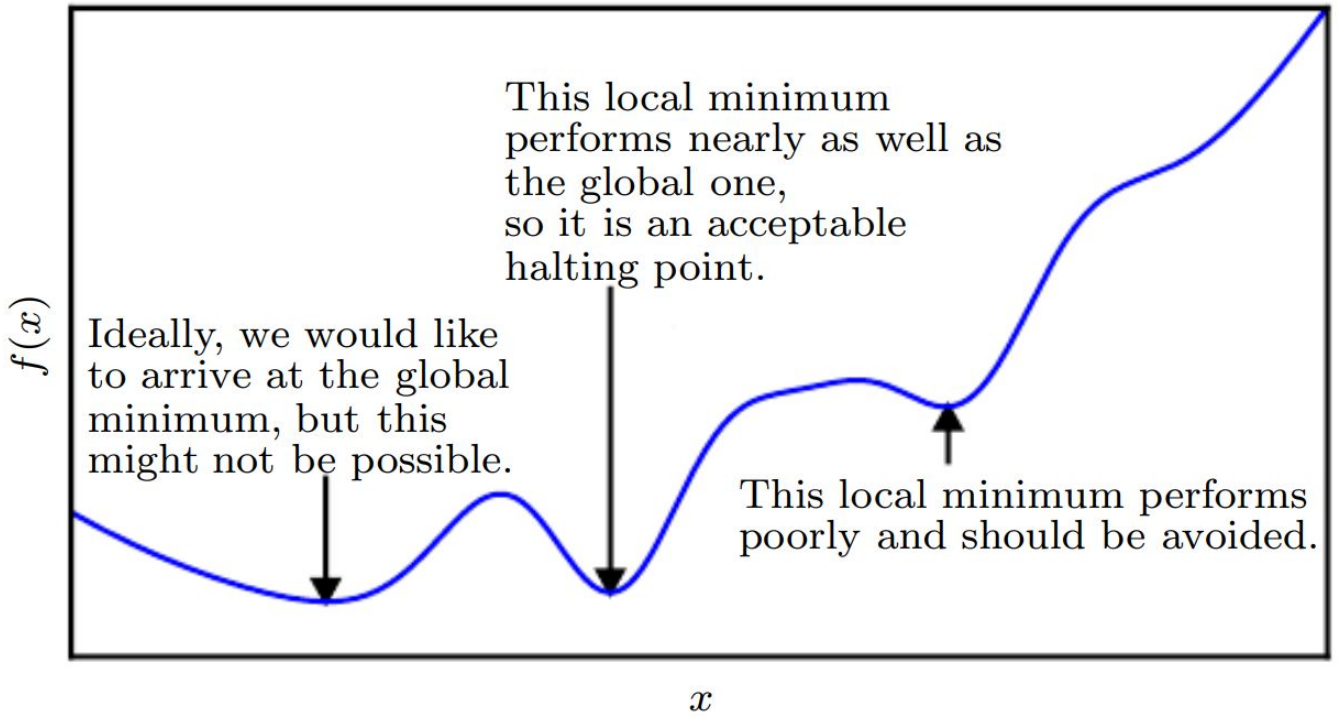
Batch gradient descent: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla L(\boldsymbol{\theta})$

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

Challenges:

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

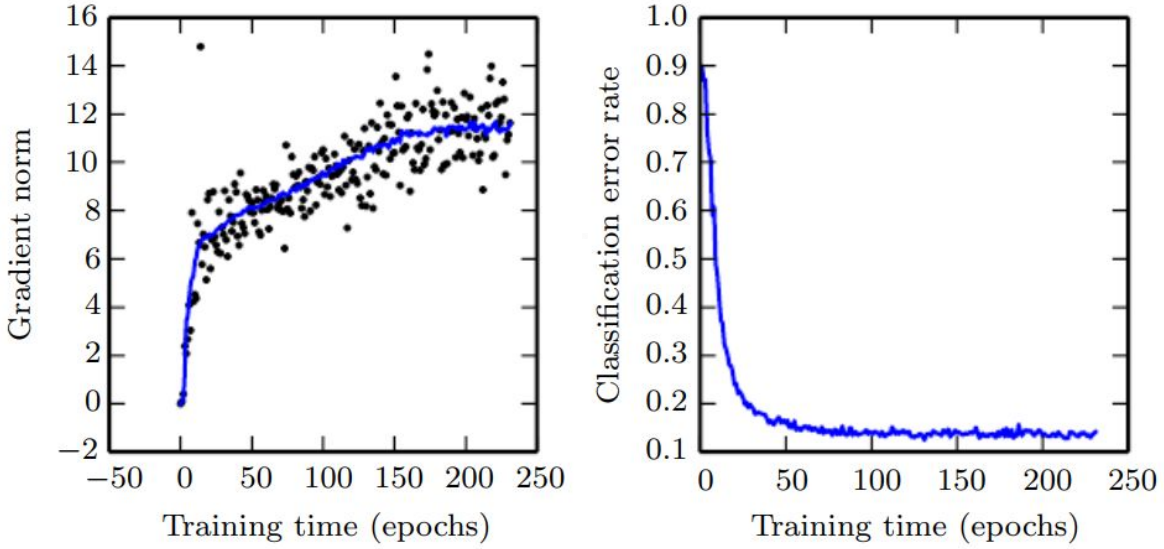
Global and Local Minima



[5]

Critical points

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

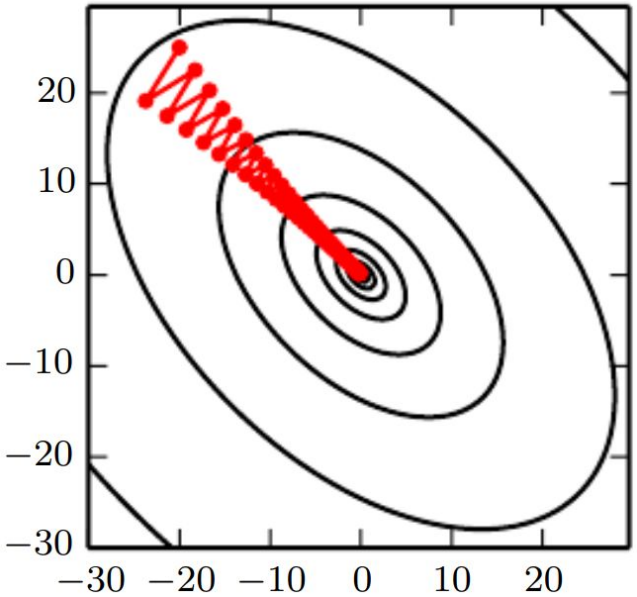


[5]

Gradient Descent Optimizations - Momentum

Loss function can have canyon-like curvature

Gradient bounces between canyon walls

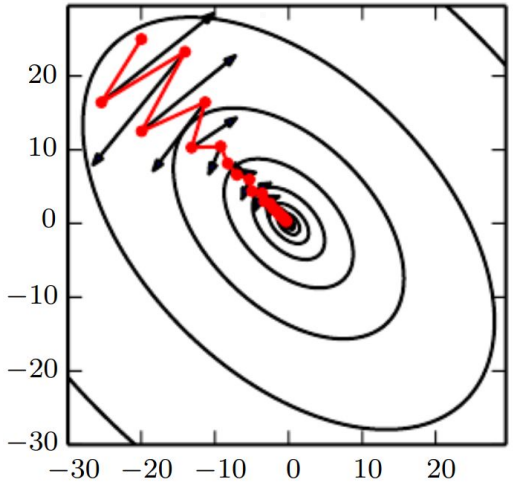


[5]

Gradient Descent Optimizations - Momentum

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

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



[5]

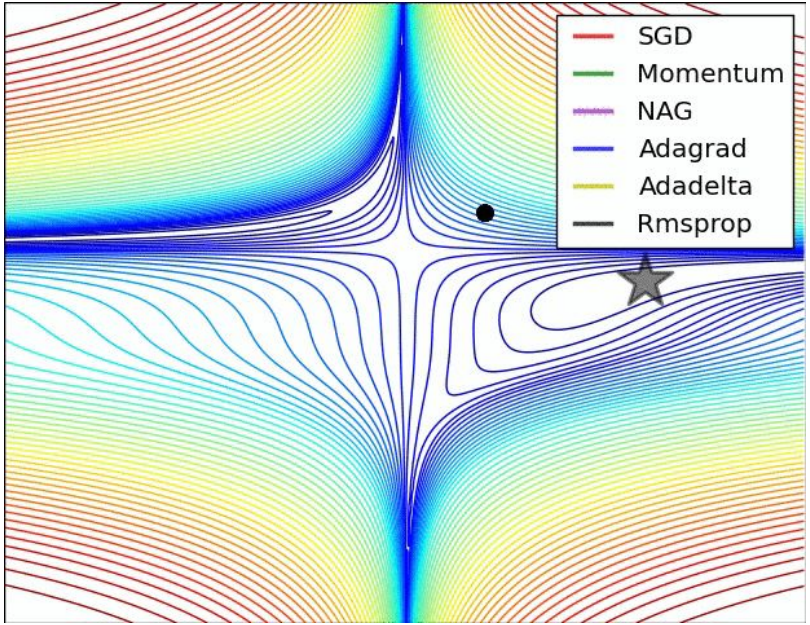
Adaptive Algorithms for Gradient Descent

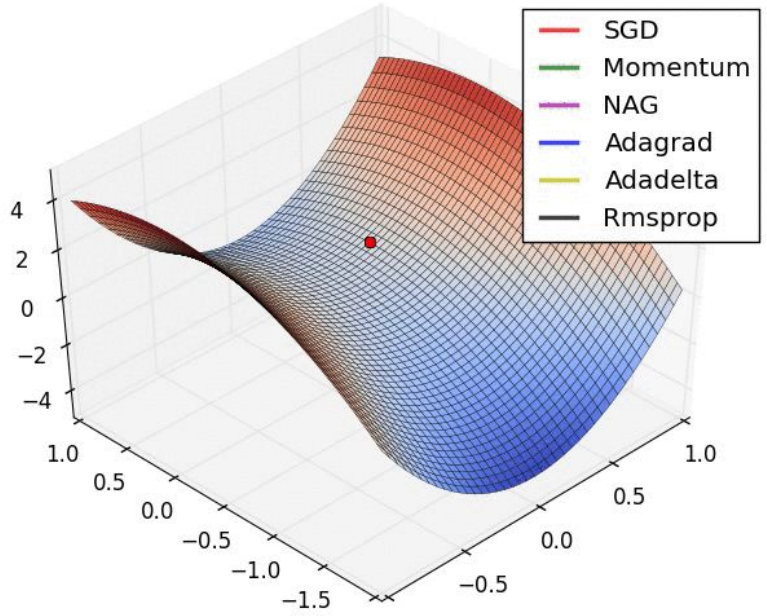
- **Adagrad**: Adapts learning rate to the parameters (low learning rate for frequent events, high learning rate for infrequent events)
- **Adadelta**: Extension of Adagrad + aggressively decrease learning rate
- **RMSprop**: Very similar to Adadelta
- **Adam**: Adapts learning rate + stores decaying 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.

[6]

Adaptive Algorithms for Gradient Descent





[6]

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.

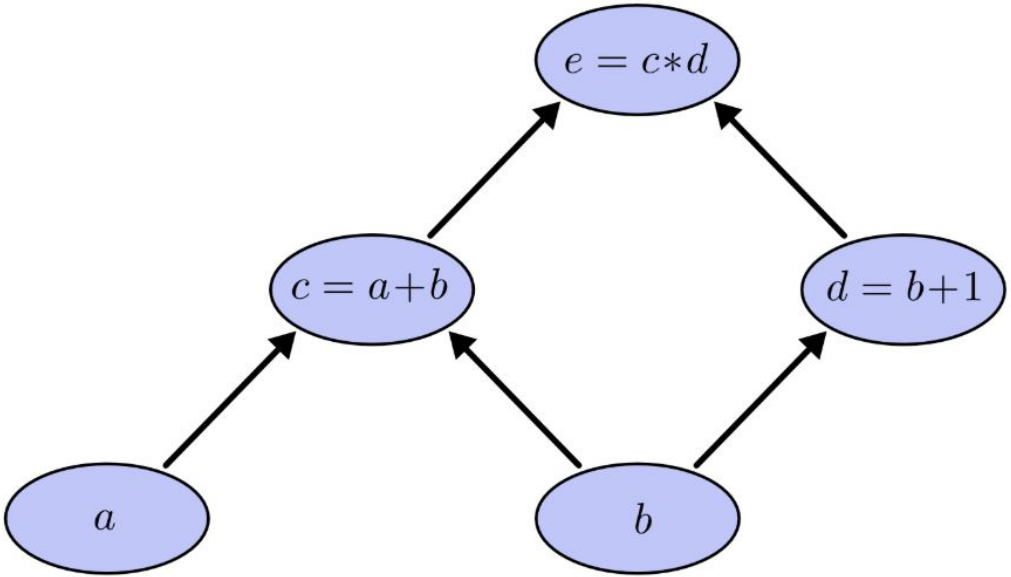
$$f^1(x_1, x_2) = x_1 x_2$$
$$f^2(x_1, x_2) = x_1 + x_2$$
$$f^3(x_1, x_2) = \max(x_1, x_2)$$

$$f^1_{x_1}(x_1, x_2) = x_2 \text{ and } f^1_{x_2}(x_1, x_2) = x_1$$
$$f^2_{x_1}(x_1, x_2) = 1 \text{ and } f^2_{x_2}(x_1, x_2) = 1$$
$$f^3_{x_1}(x_1, x_2) = 1 \text{ if } x_1 \geq x_2 \text{ else } 0$$
$$f^3_{x_2}(x_1, x_2) = 1 \text{ if } x_2 \geq x_1 \text{ else } 0$$

[1]

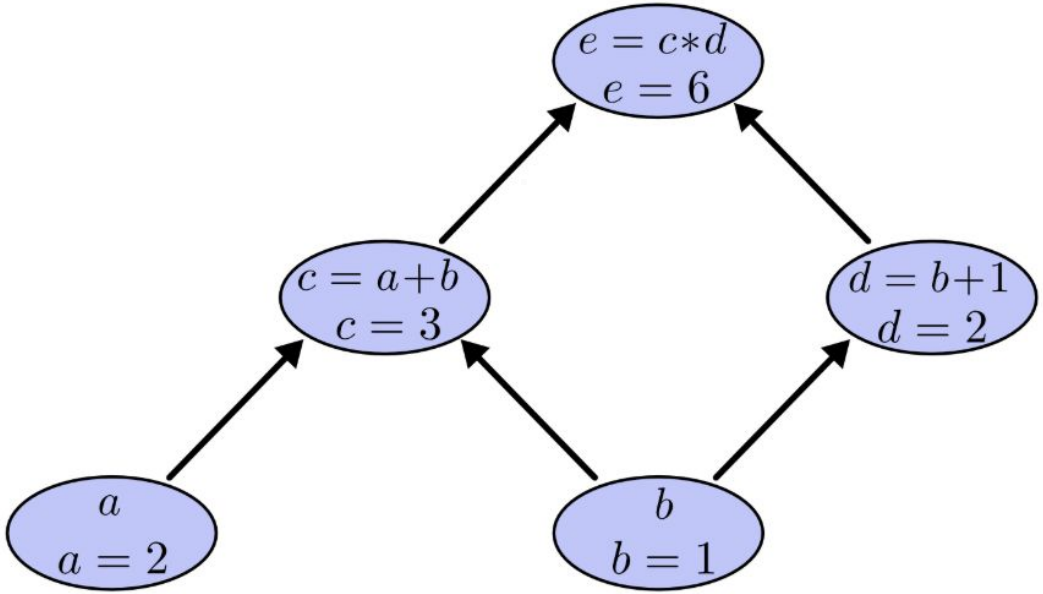
Backpropagation

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`

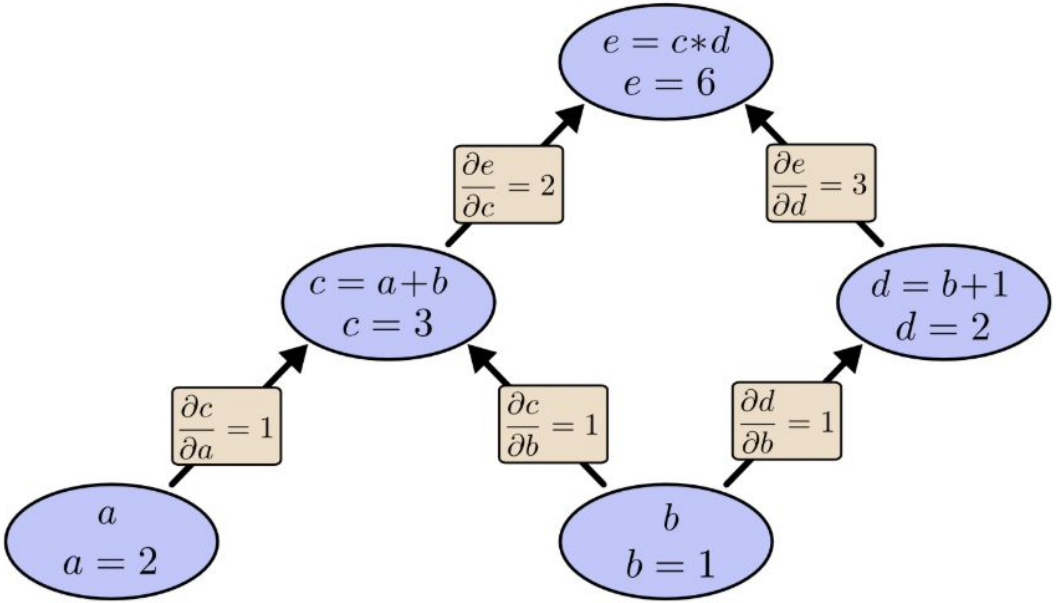


[7]

Backpropagation

Compute local gradients independently

Compute remaining gradients from local ones using multivariate chain rule

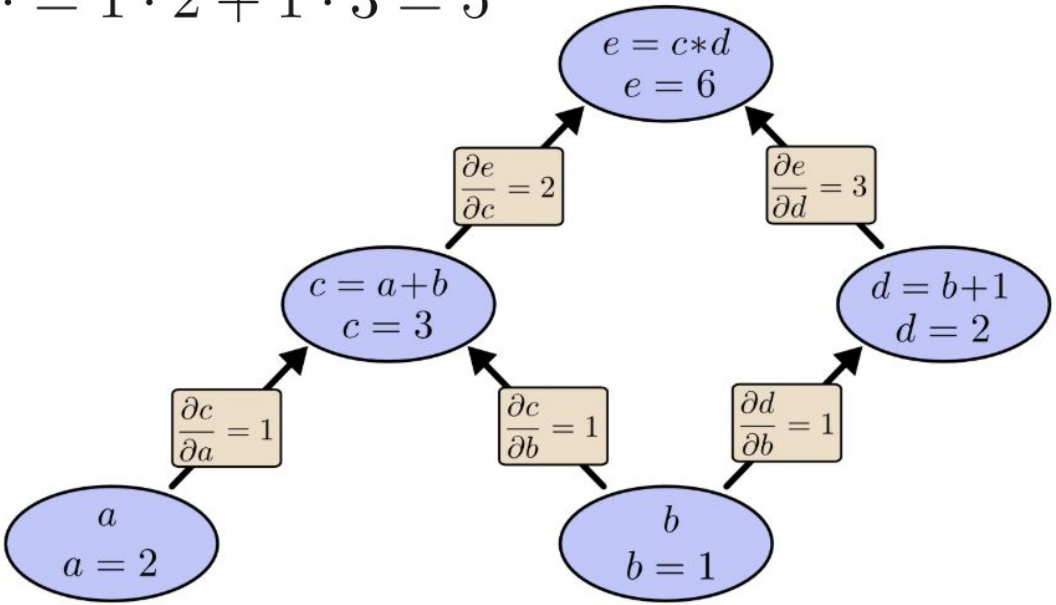


[7]

Backpropagation

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

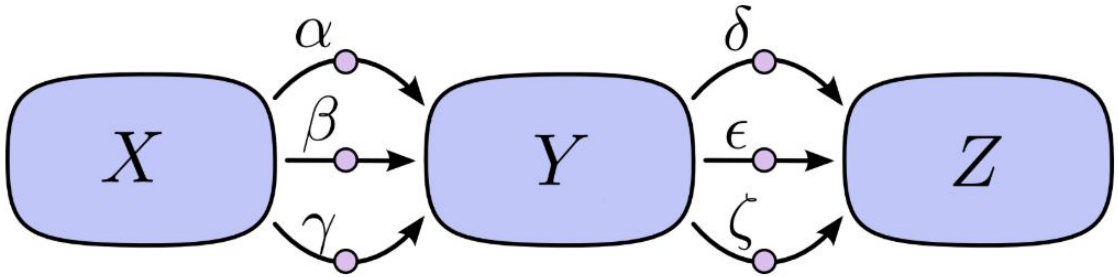
$$e_b(2,1) = \cdots = 1 \cdot 2 + 1 \cdot 3 = 5$$



[7]

Path Factorization

Summing over all paths leads to combinatorial explosion

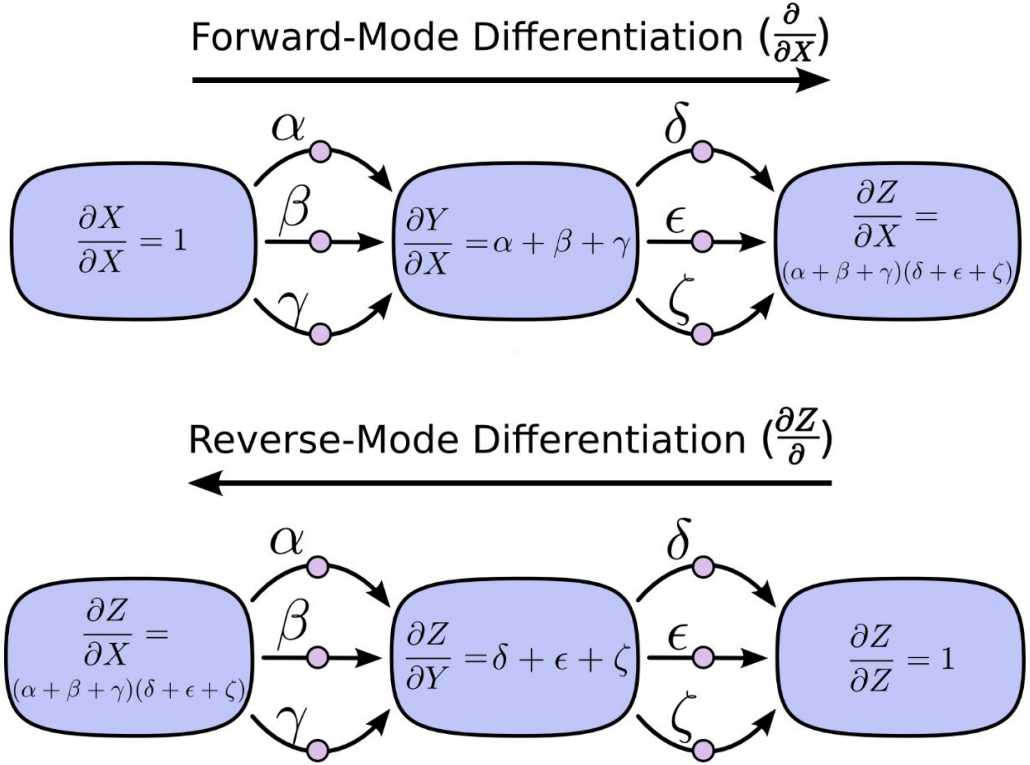


$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

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

[7]

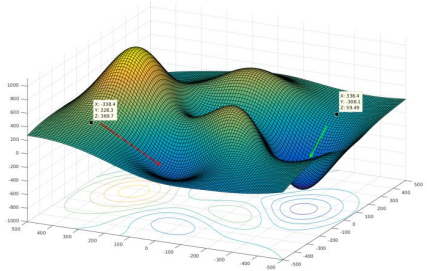
Forward-mode and Reverse-mode differentiation



[7]

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/analytics_166904
- https://www.flaticon.com/free-icon/landing_1284826
- https://www.flaticon.com/free-icon/dimmer_1833516
- https://www.flaticon.com/free-icon/energy-class_1833531
- https://www.flaticon.com/free-icon/slider-tool_941588
- 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