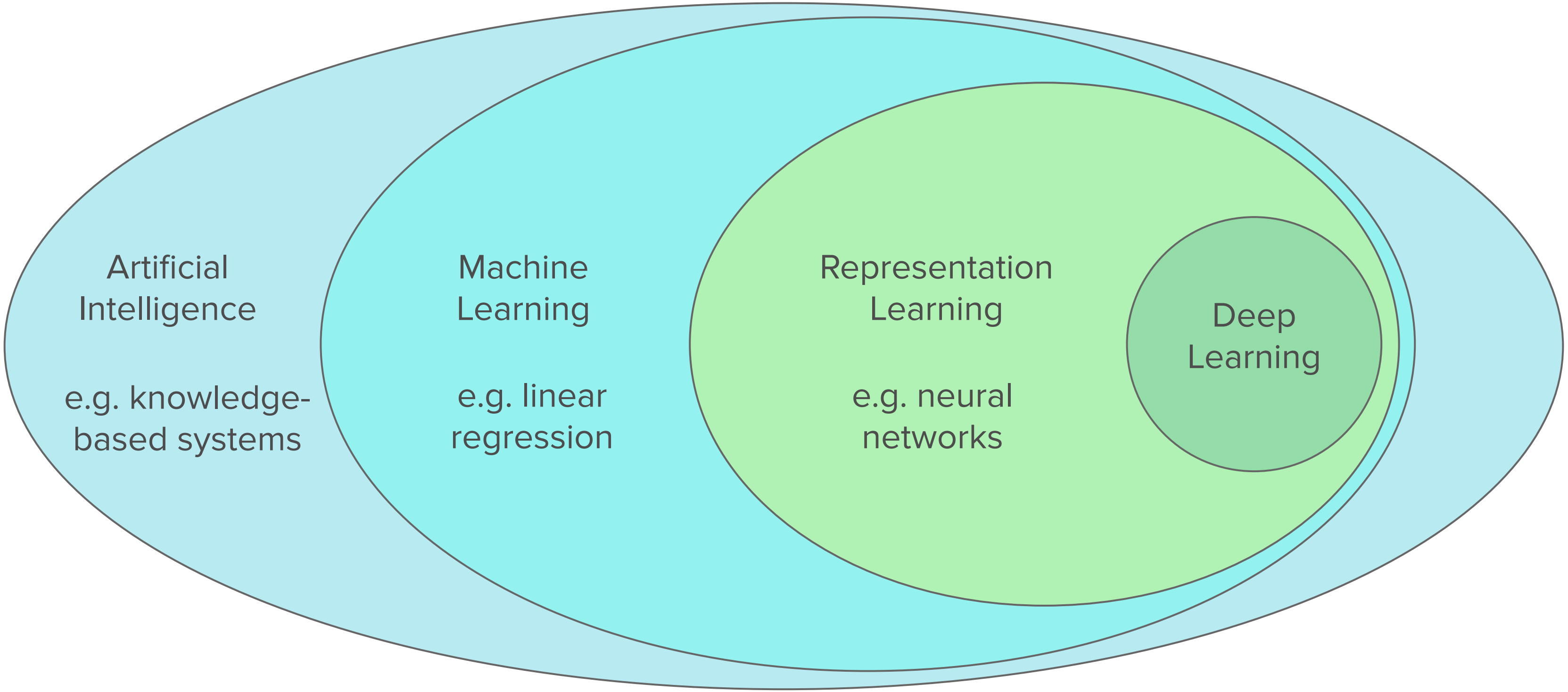
Applied Deep Learning

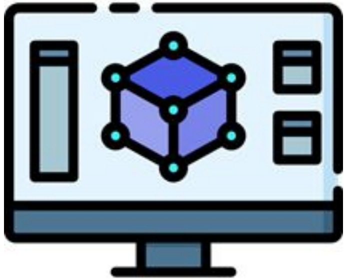
Neural Networks, Optimization and Backpropagation
10.10.2019

Alexander Pacha

Recap - Terminology



Recap - How Can a Machine Learn?



Model

Loss

Optimization function

How does a model work?

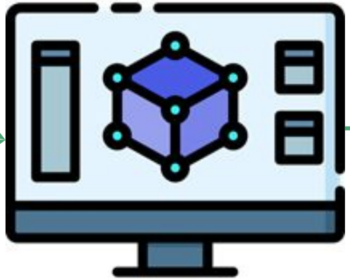
Training Set

"Dog"

"Cat"

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

Model



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

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

Test Set

"???"

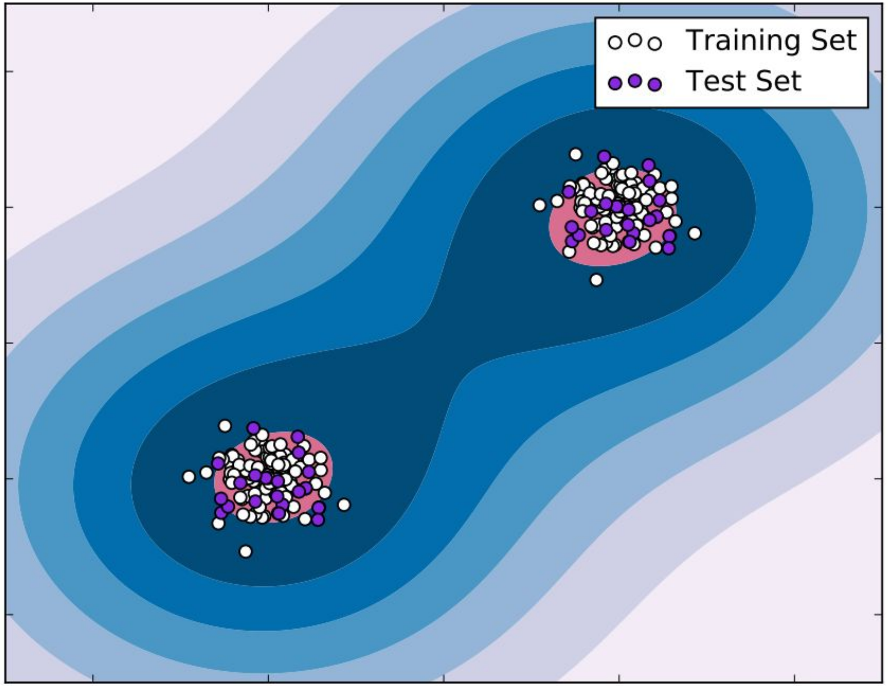
Inference

$P(dog|x) = 0.7$

$P(cat|x) = 0.3$

How can a model work on unseen data?

Both datasets must have similar distribution



Source: https://github.com/cpra/dlvc2016

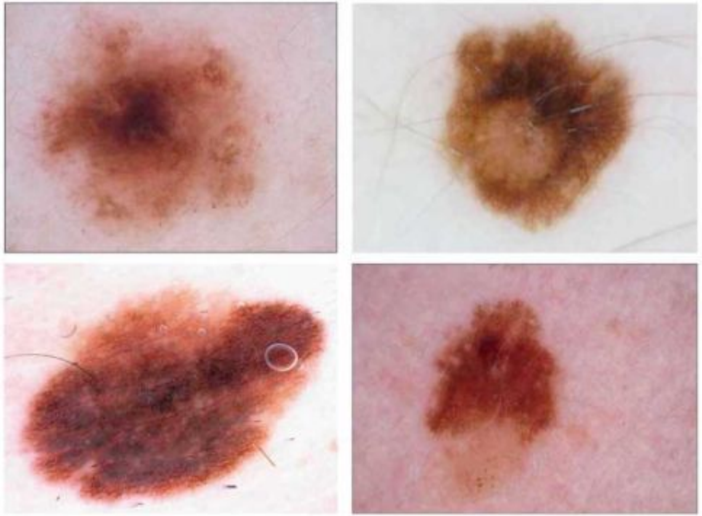
Does it generalize?

?

Image Classification

Melanoma

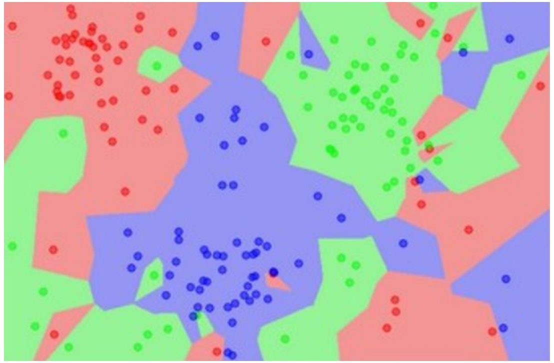
Benign



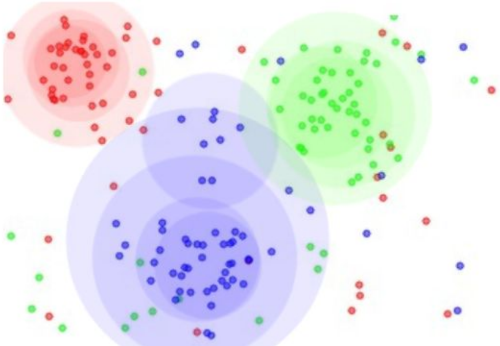
Source: Lopez et al. Skin lesion classification from dermoscopic images using deep learning techniques. 2017

Classes of models

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



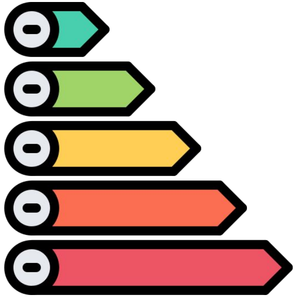
Generative models learn class-conditional densities $\mathtt{P}(\mathbf{x}|\mathtt{w})$



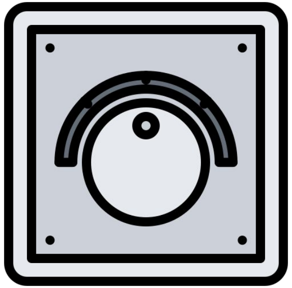
Source: http://cs231n.github.io

Output

- Classification problem with finite number of different labels

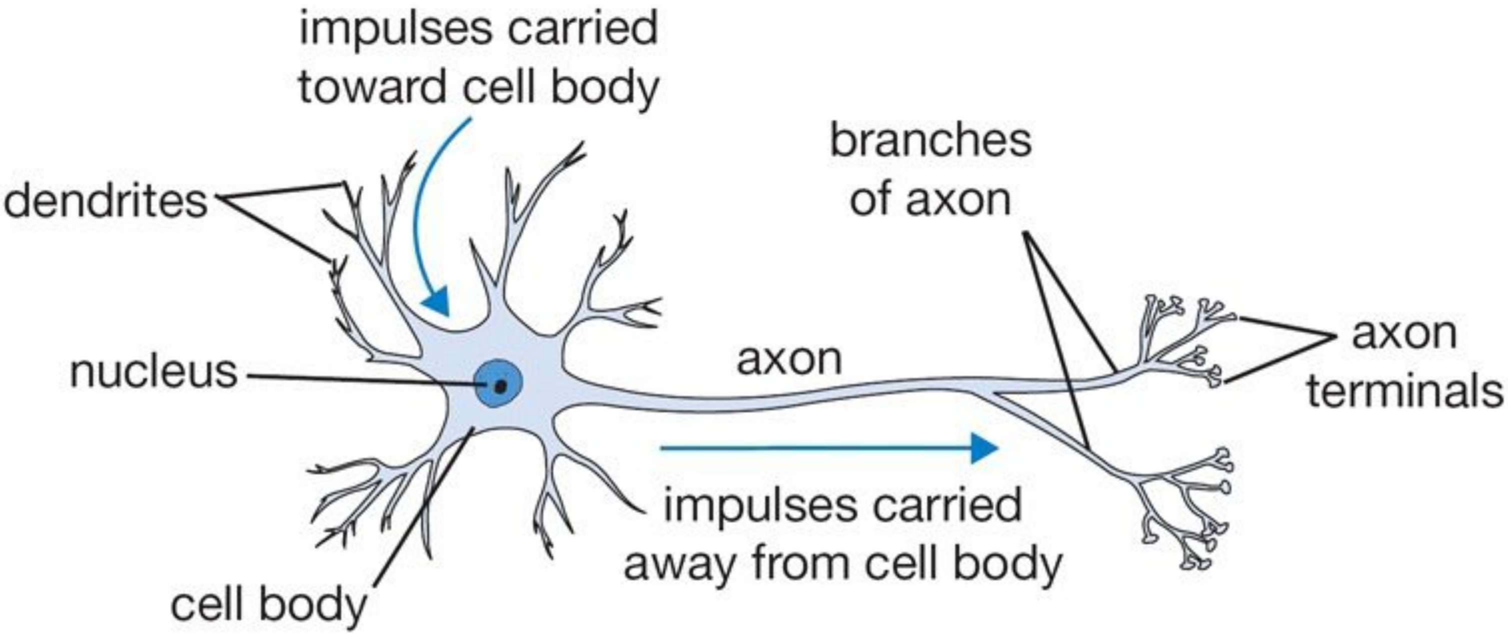


- Regression problem with a continuum of output values



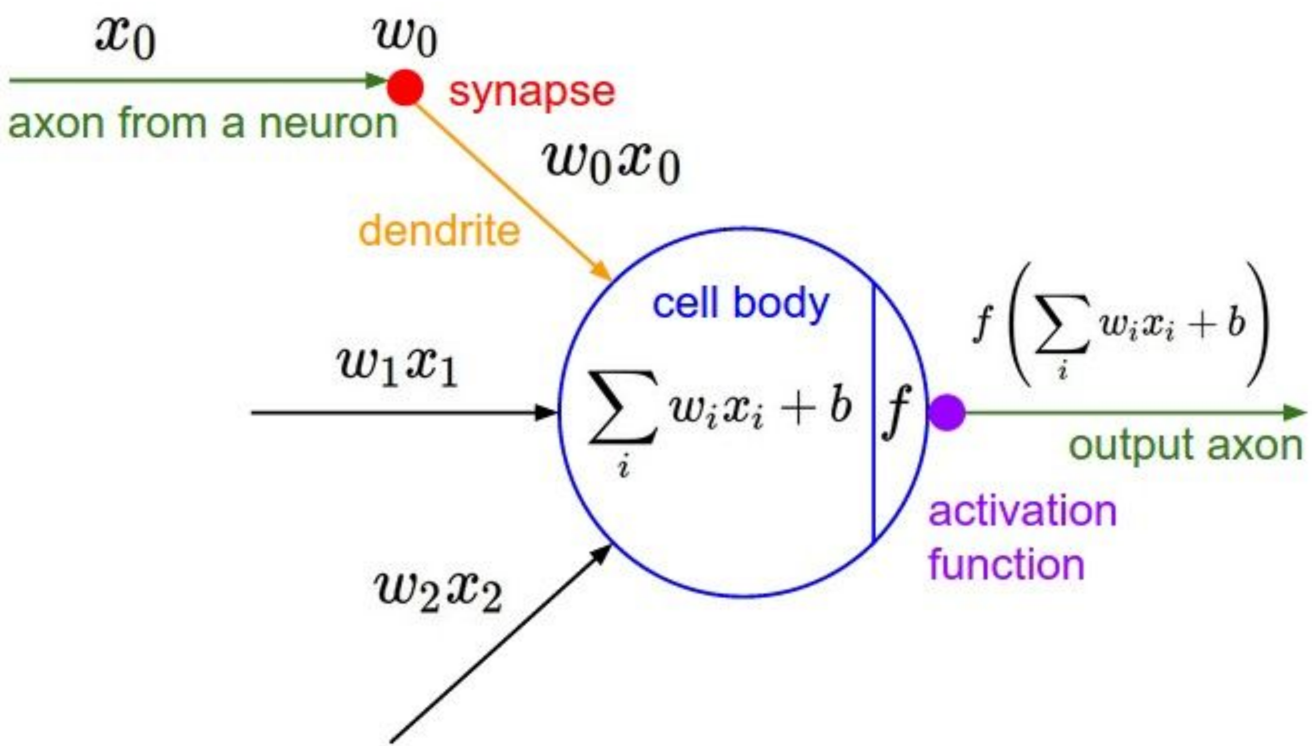
Neural Networks

Biological Neuron



Source: http://cs231n.github.io/neural-networks-1/

Artificial Neuron



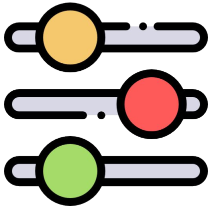
Source: http://cs231n.github.io/neural-networks-1/

Parametric Models, MLP, Feedforward NN

Neurons are controlled by parameters:

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

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



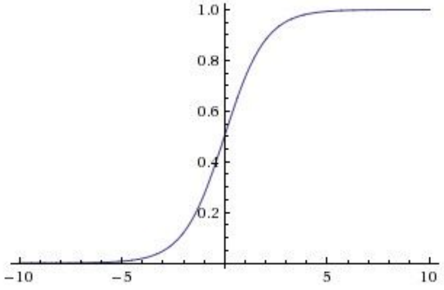
Activation Function

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

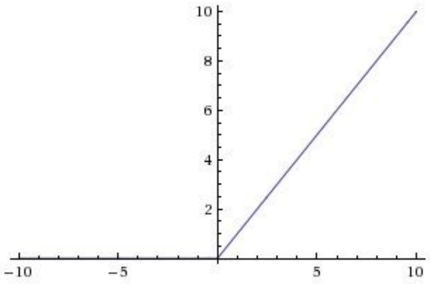
$$Y = \text{Activation}(\Sigma(\text{weight} * \text{input}) + \text{bias})$$

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

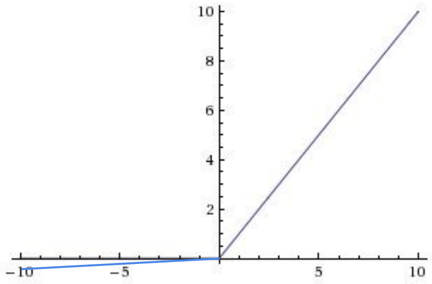
Popular Activation Functions



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



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

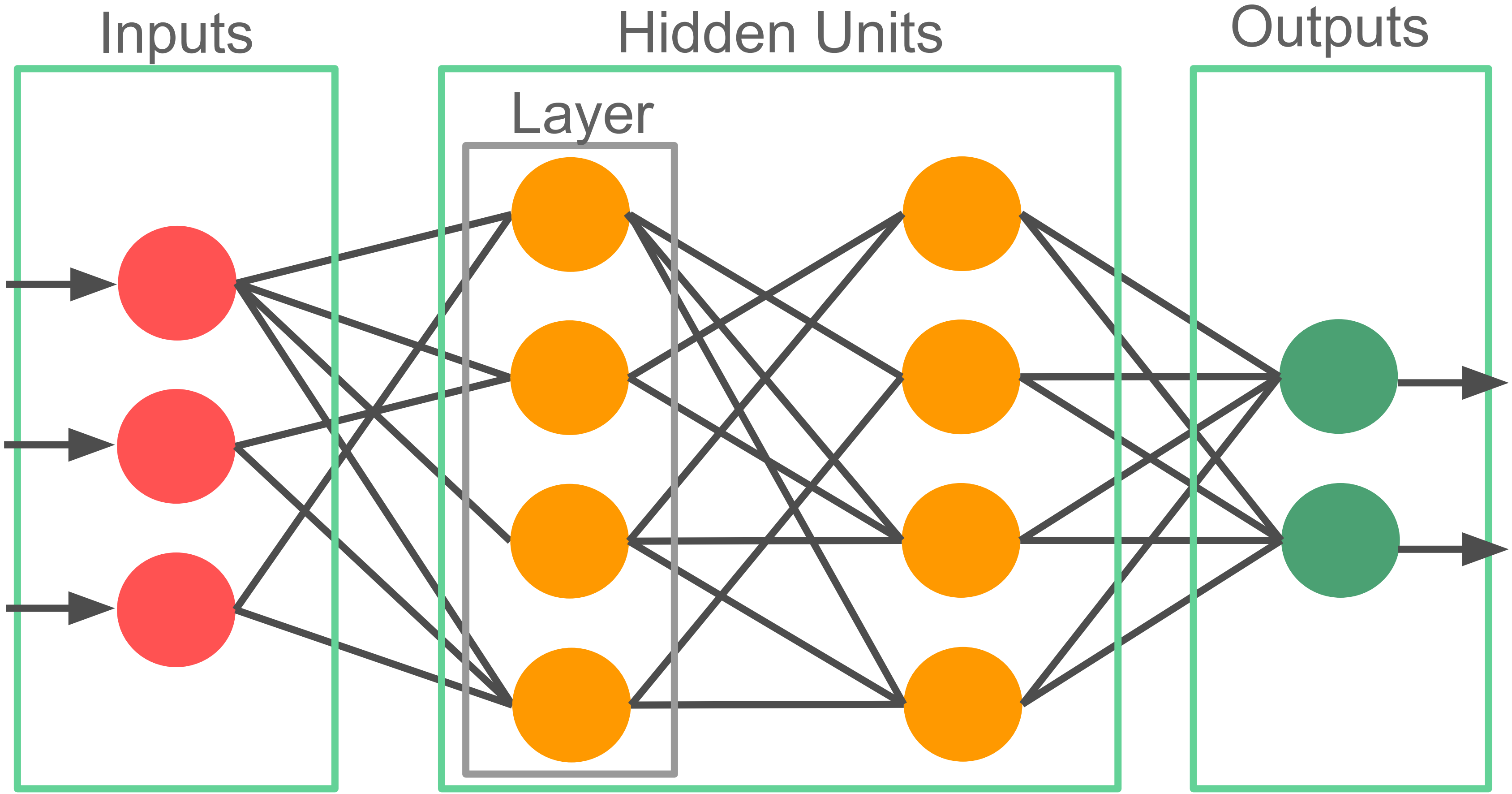


Leaky ReLU
$f(x)=1(x<0)(\alpha x)+1(x>=0)(x)$

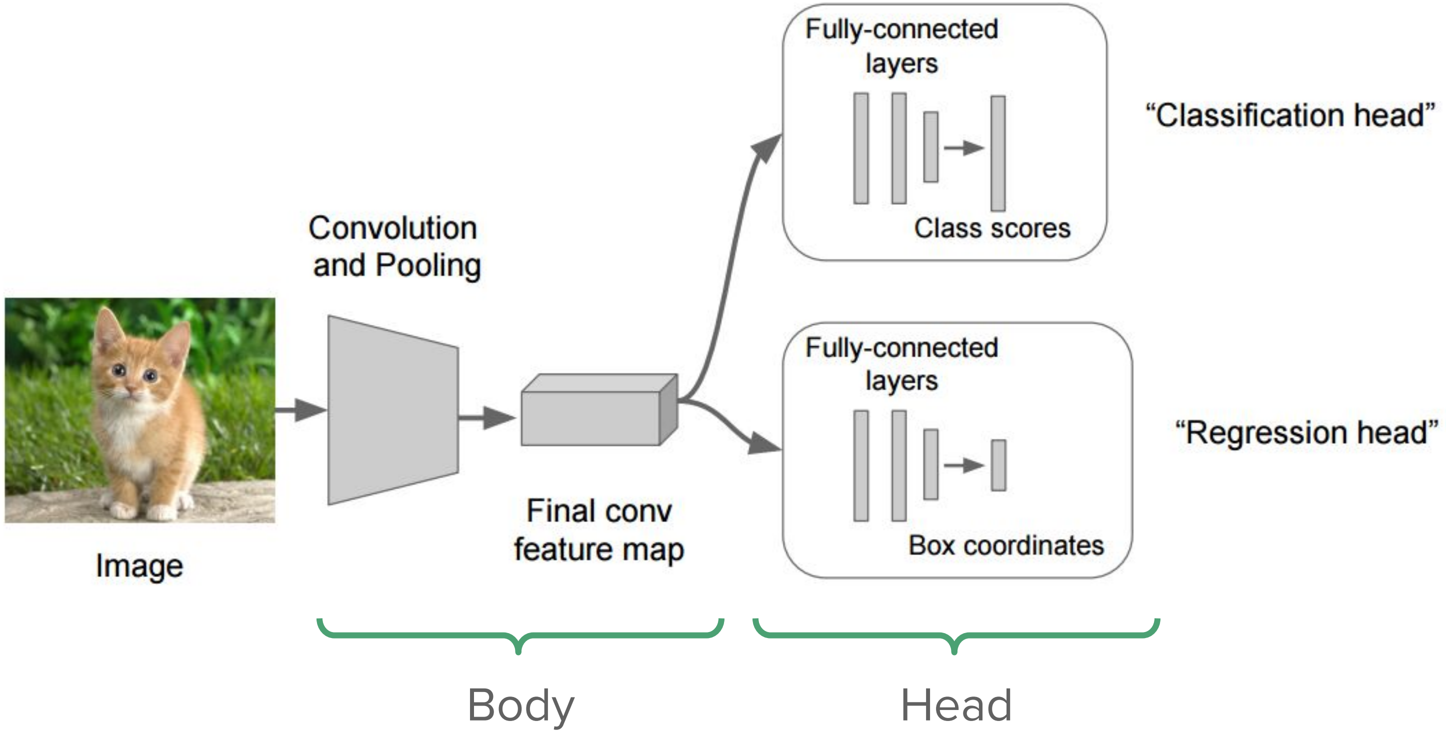
- Sigmoid inspired by nature
- ReLU very efficient to compute but has problem of dying neurons
- Leaky ReLU tries to mitigate dying neurons issue by propagating a small signal

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

Neural Network



Common Neural Network Structure



Loss Functions

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

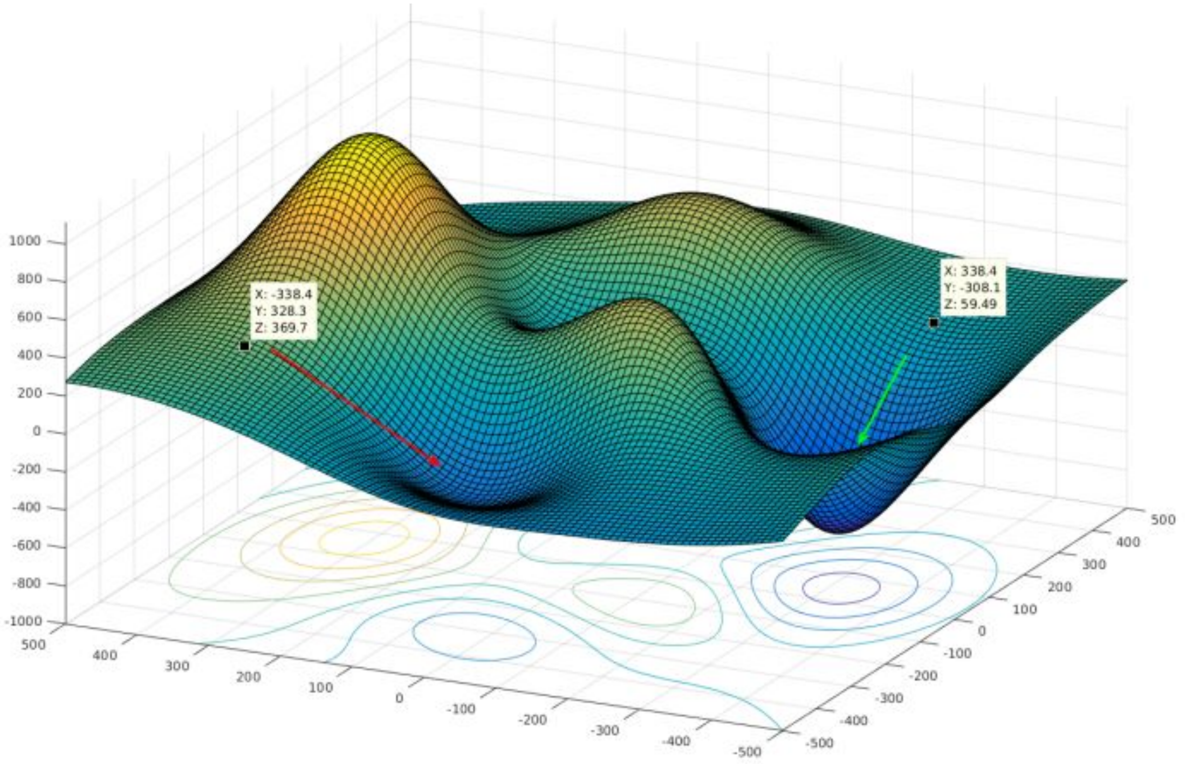
$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

```
target = [0,0,0,1]
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
```

Loss Function



Source: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/model_optimization.html

Optimization

How to get to the valley?

Gradient Descent

- Nonlinear optimization algorithm
- Most popular algorithm in Deep Learning

Gradient Descent

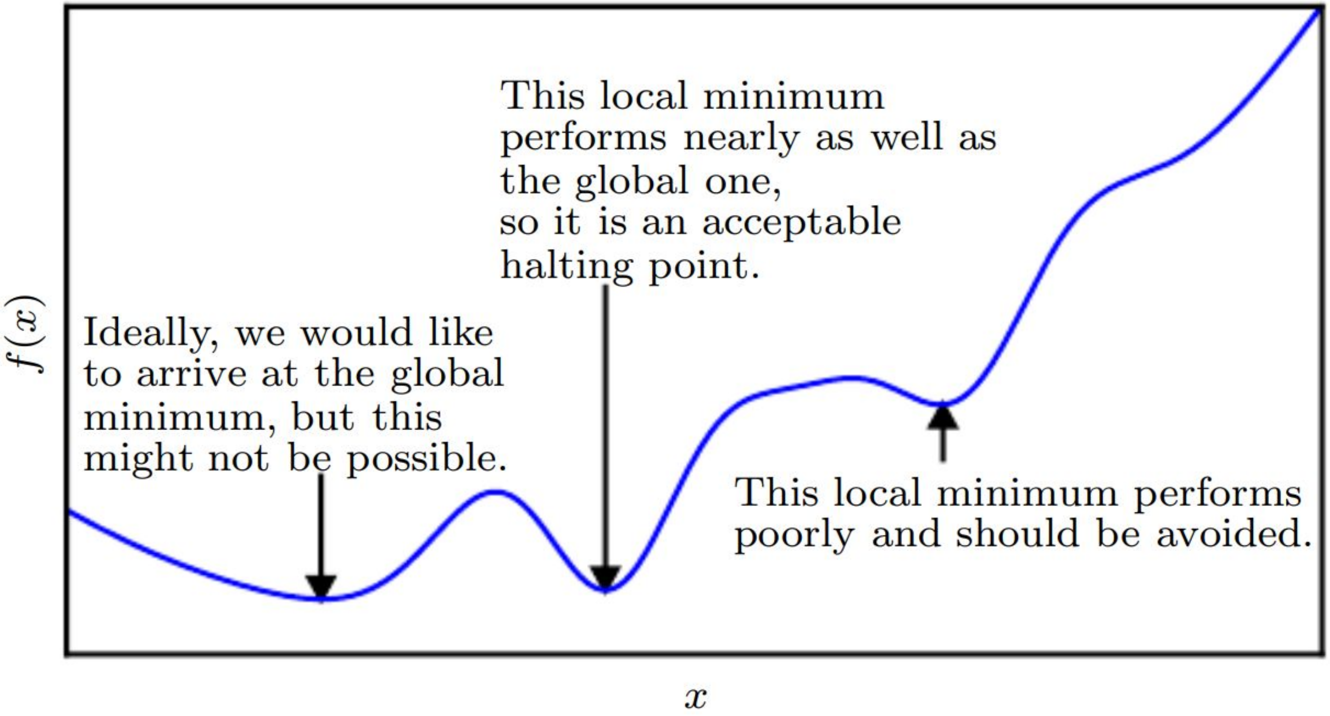
Iterative algorithm on loss function $L(\theta)$:

- 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

Global and Local Minima



Source: http://www.deeplearningbook.org/

Gradient Descent

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

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

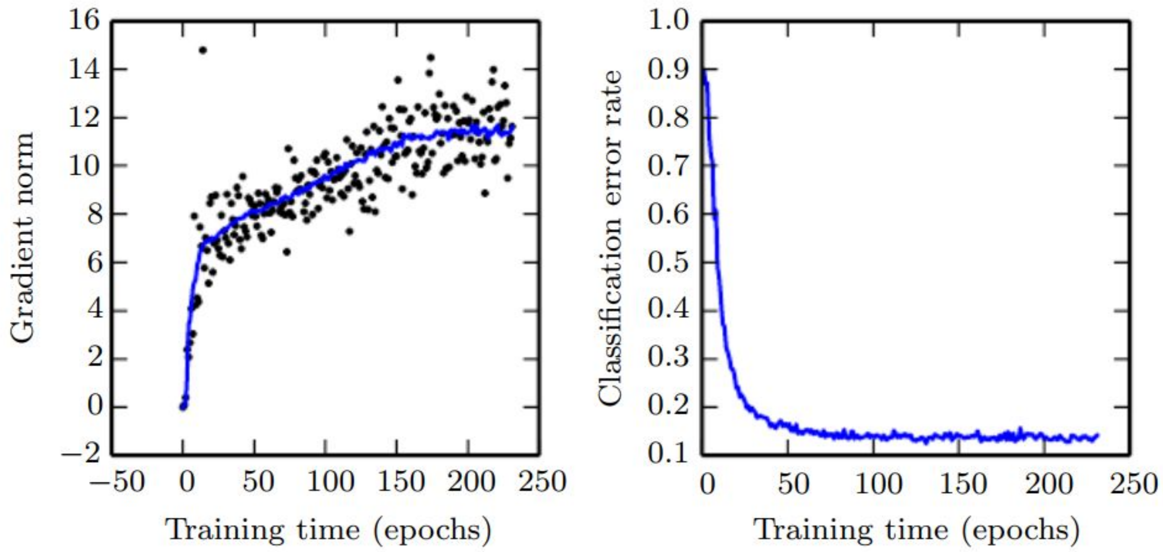
Mini-batch gradient descent: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha\nabla L(\boldsymbol{\theta};\ x^{(i;i+n)},\ y^{(i;i+n)})$

Challenges:

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

Critical points

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

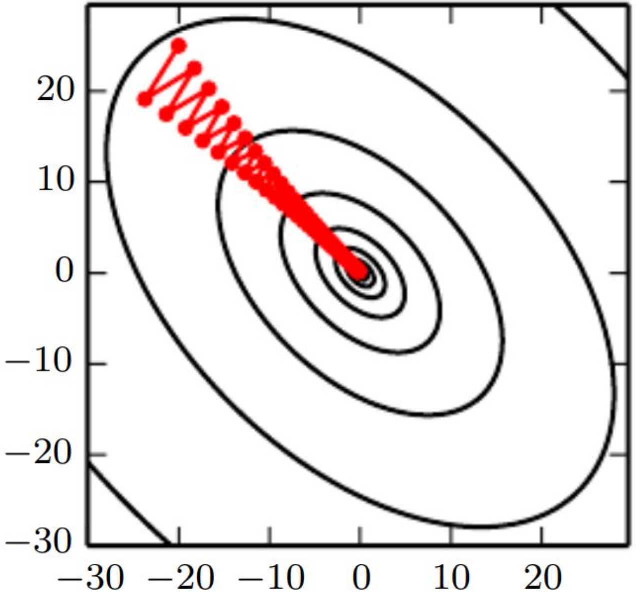


Source: http://www.deeplearningbook.org/

Gradient Descent Optimizations - Momentum

Loss function can have canyon-like curvature

Gradient bounces between canyon walls

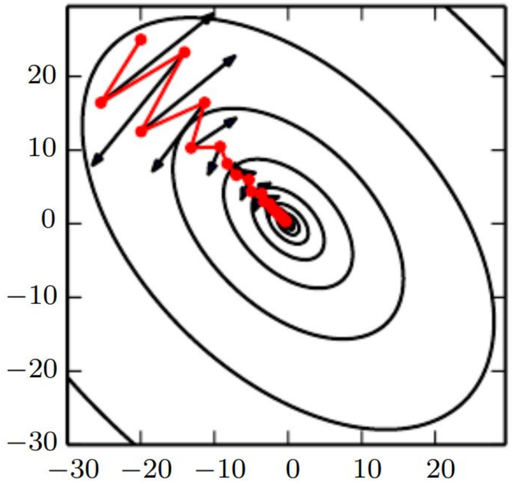


Source: http://www.deeplearningbook.org/

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 $\beta \in [0,1)$



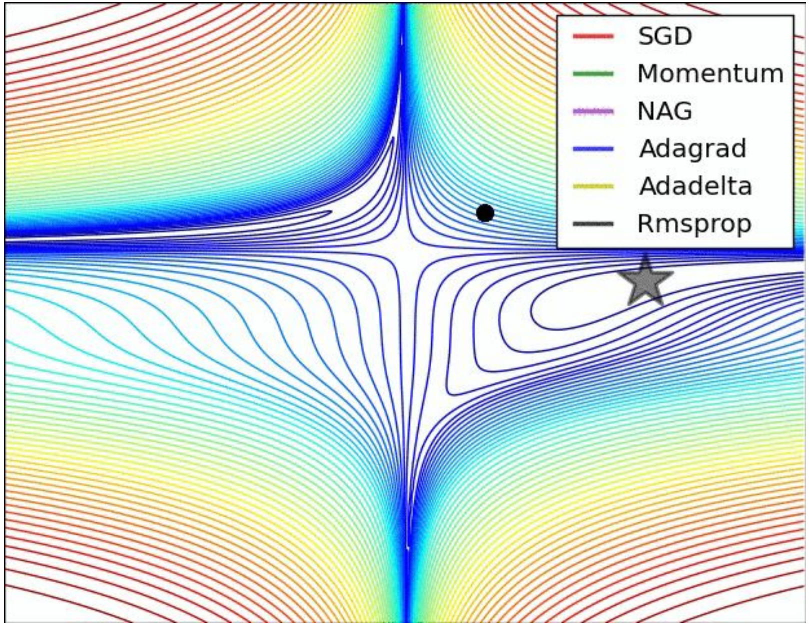
Source: http://www.deeplearningbook.org/

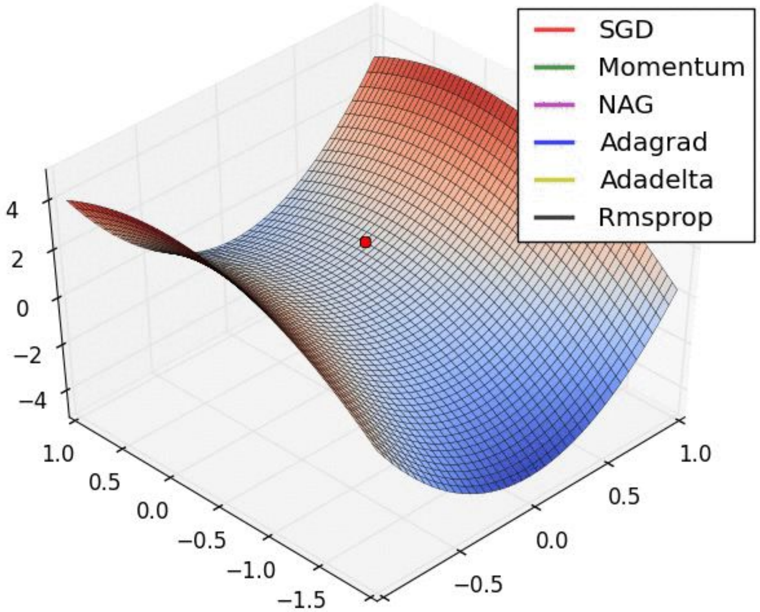
Adaptive Algorithms for Gradient Descent

- **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

Source: http://ruder.io/optimizing-gradient-descent/

Adaptive Algorithms for Gradient Descent





Source: http://ruder.io/optimizing-gradient-descent/

Backpropagation

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

- 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.

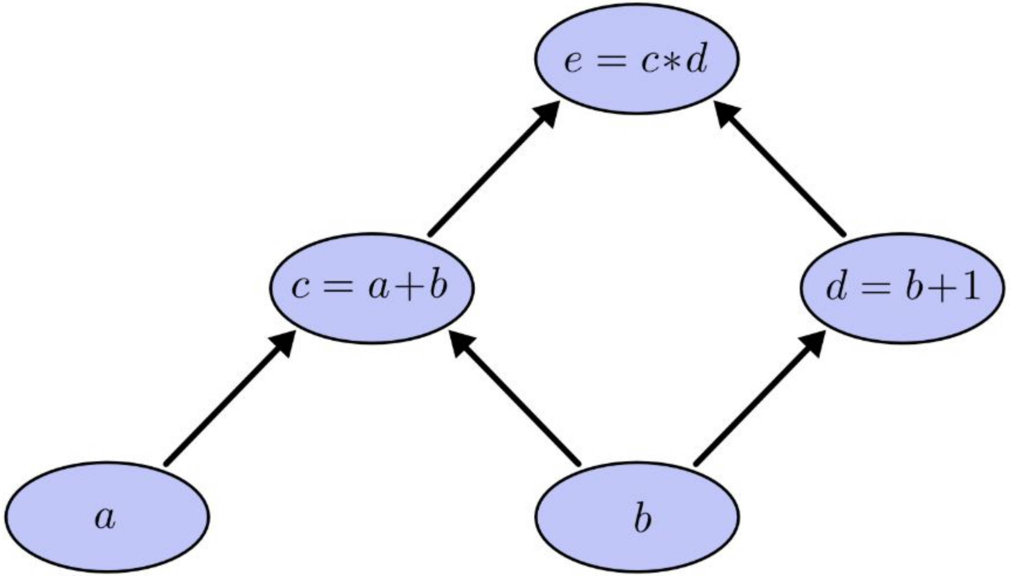
$$f^1(x_1, x_2) = x_1 x_2$$
$$f^2(x_1, x_2) = x_1 + x_2$$
$$f^3(x_1, x_2) = \max(x_1, x_2)$$

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

Source: https://github.com/cpra/dlvc2016

Backpropagation

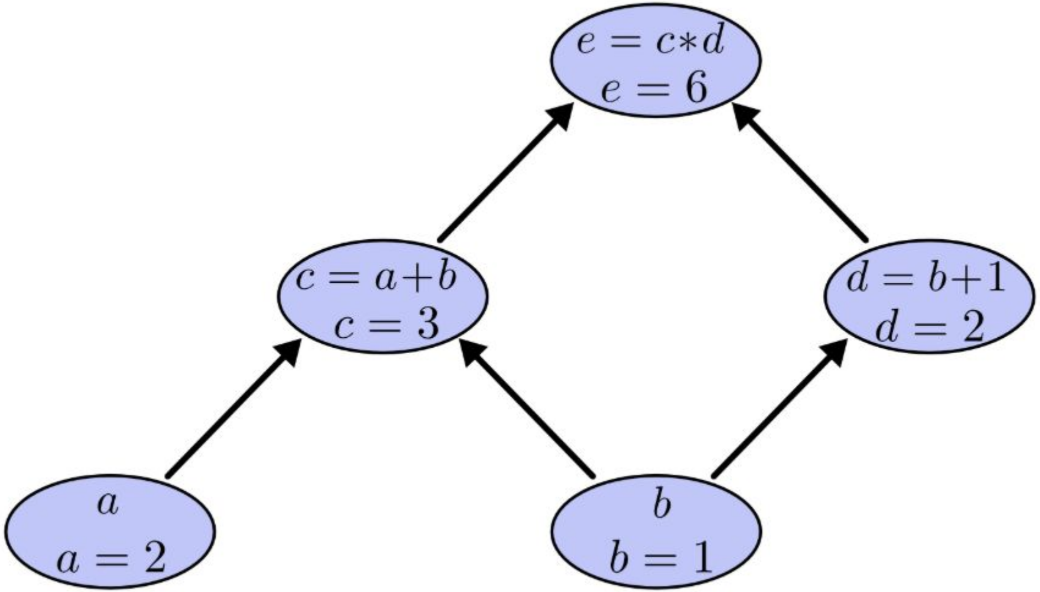
Sample expression `e(a,b) = (a+b)(b+1)`



Source: https://colah.github.io/posts/2015-08-Backprop/

Backpropagation

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

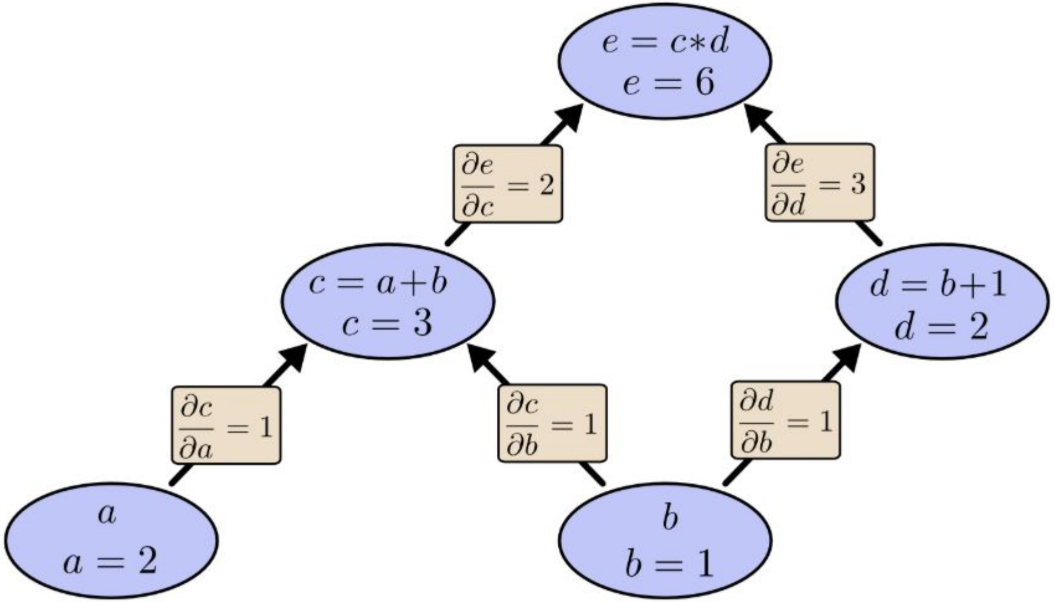


Source: https://colah.github.io/posts/2015-08-Backprop/

Backpropagation

Compute local gradients independently

Compute remaining gradients from local ones using multivariate chain rule

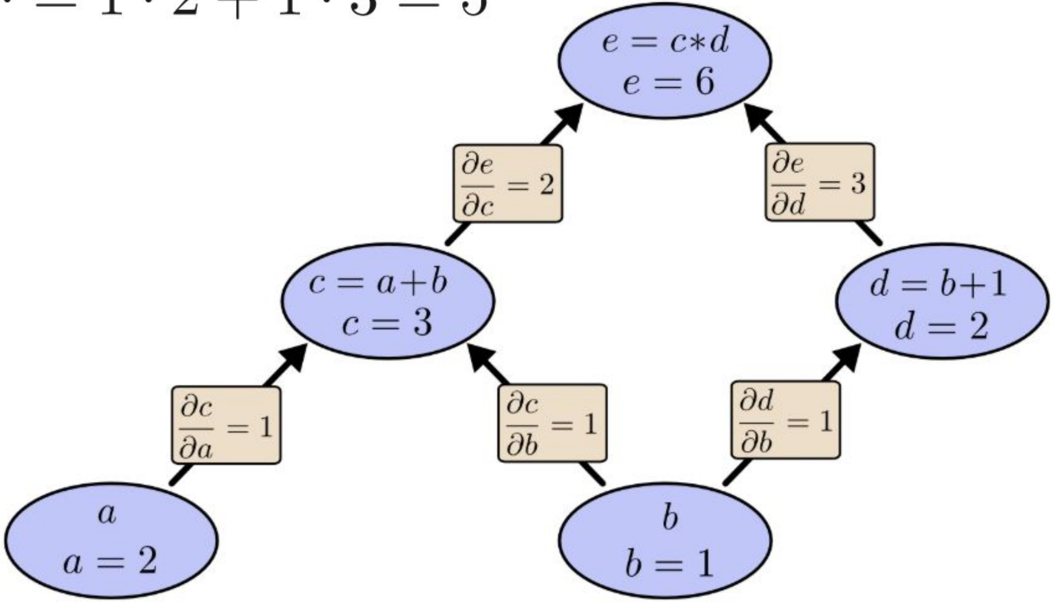


Source: https://colah.github.io/posts/2015-08-Backprop/

Backpropagation

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

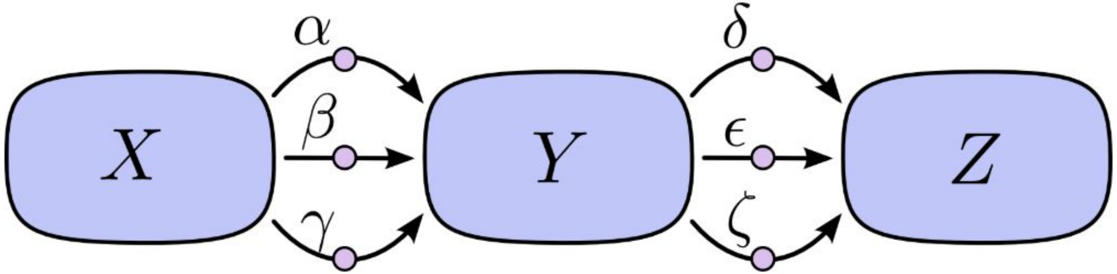
$$e_b(2,1) = \cdots = 1 \cdot 2 + 1 \cdot 3 = 5$$



Source: https://colah.github.io/posts/2015-08-Backprop/

Path Factorization

Summing over all paths leads to combinatorial explosion

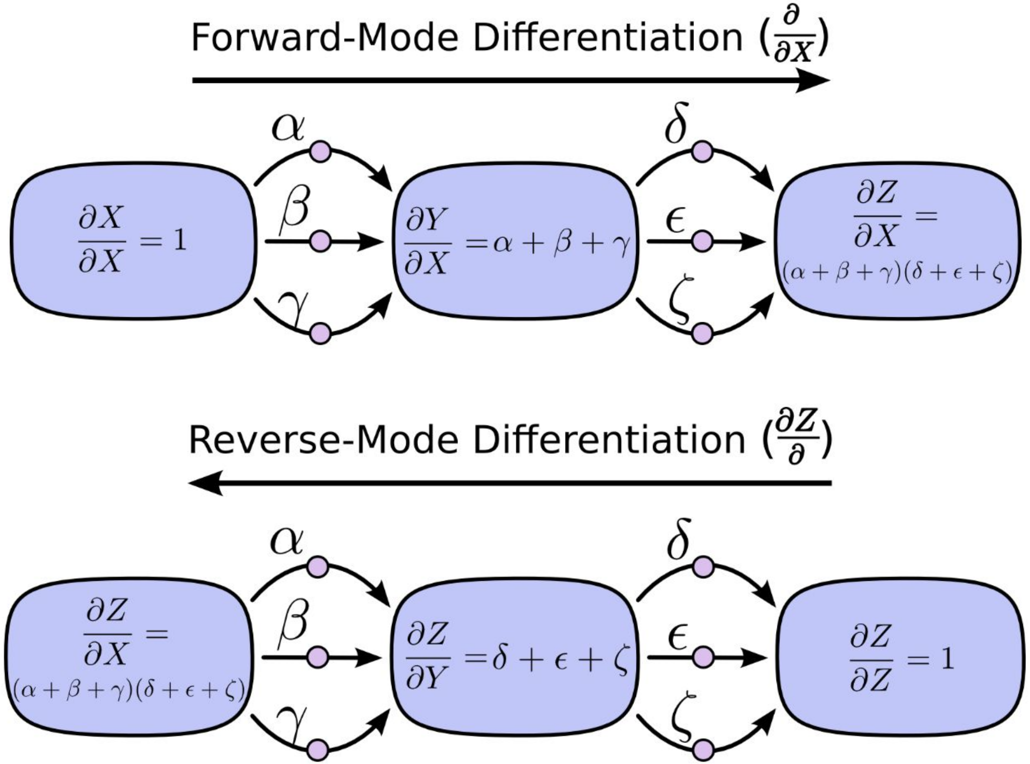


$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

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

Source: https://colah.github.io/posts/2015-08-Backprop/

Forward-mode and Reverse-mode differentiation



Source: https://colah.github.io/posts/2015-08-Backprop/

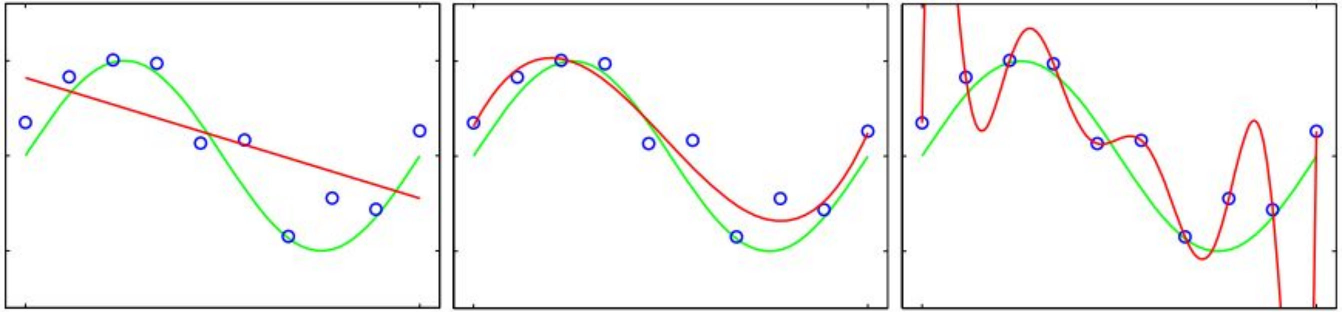
Algorithmic performance

Two factors:

- Ability to minimize training error
- Ability to minimize gap between training and test error

Two challenges:

- Underfitting: Unable to reach low training error
- Overfitting: Large gap between training and test error



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