Training Neural Networks

Florin Gogianu,
Technical University of Cluj-Napoca,
Bitdefender

April 13, 2022

Inspired by CS231n - Stanford, CS421 - University of Toronto, NYU-DLSP21 - University of New York. Some slides adapted from Ștefan Postăvaru.

Before we begin, some useful resources

- Learn X in Y minutes, X=Python
- Learn what tensors are in PyTorch
- Watch this tutorial.

Goals

1. Optimize large, deep neural networks...

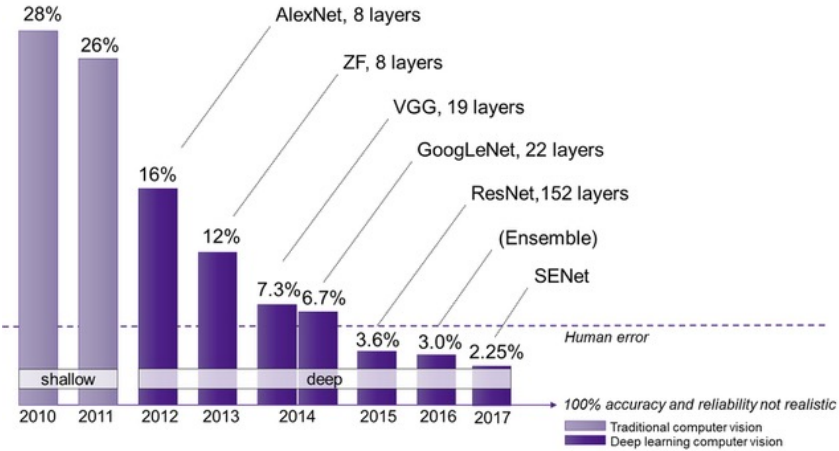
Goals

1. Optimize large, deep neural networks...
2. ...for learning *useful* representations.

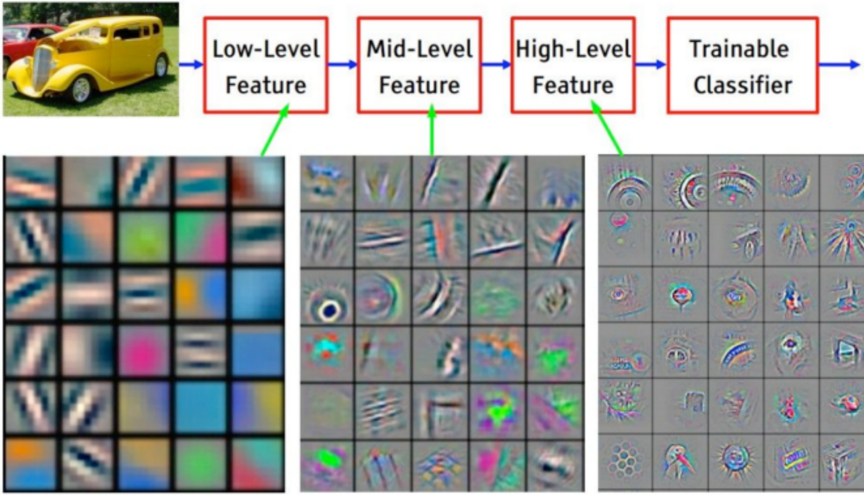
Terminology

- **Supervised learning:** labeled data points of the correct behaviour.
- **Reinforcement learning:** receive some reward signal and try to maximize it by improving the model's behaviour
- **Unsupervised learning:** no labels - the aim is discovering interesing patterns in the data and usefull representations

Supervised Learning



Supervised Learning



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Unsupervised Learning. DALL-E

Adam × DALL·E

“a photo of dog wearing a 1960 science fiction space helmet”

Created with DALL·E, an AI system by OpenAI

Adam × DALL·E

"a 35 millimeter macro photo of a bald eagle sitting on top of a space station"

Created with DALL·E, an AI system by OpenAI

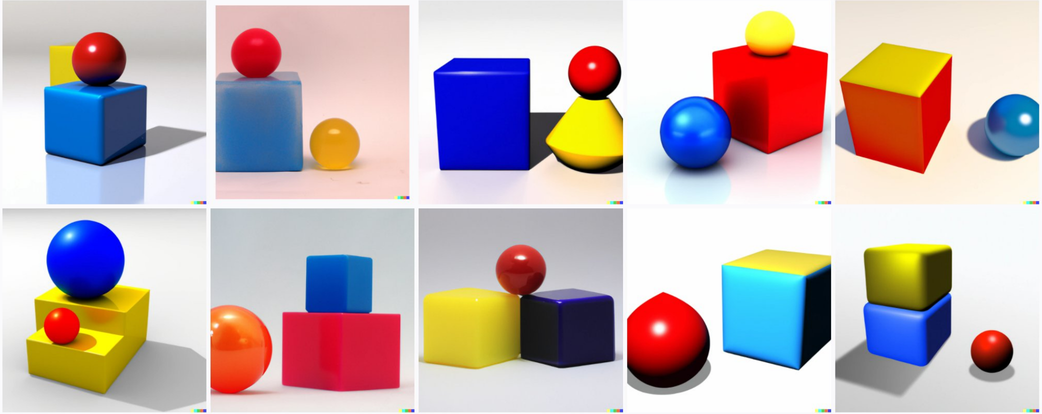
Adam × DALL·E

“a 3D rendering of playing cards with 4 aces being held by a man playing poker”

Created with DALL·E, an AI system by OpenAI

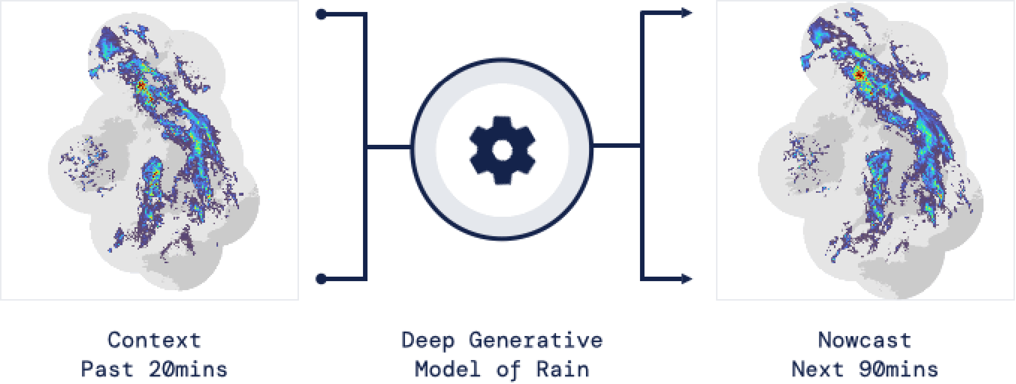
a blue cube on top of a red cube, beside a smaller yellow sphere

Report issue



Unsupervised Learning. Nowcasting

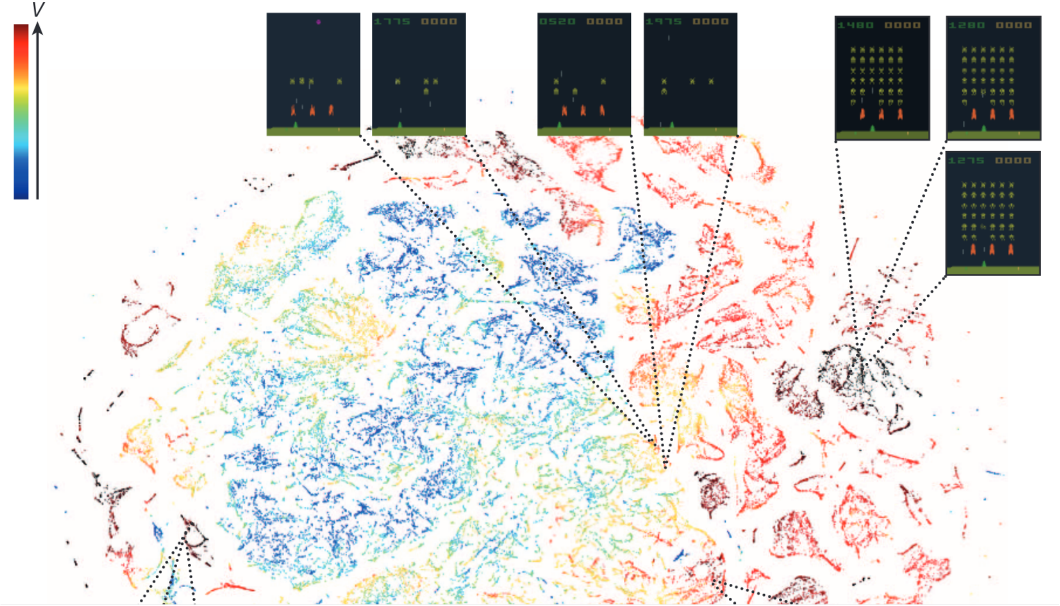
Nowcasting the next hour of rain [1]



[1] DeepMind, 2022

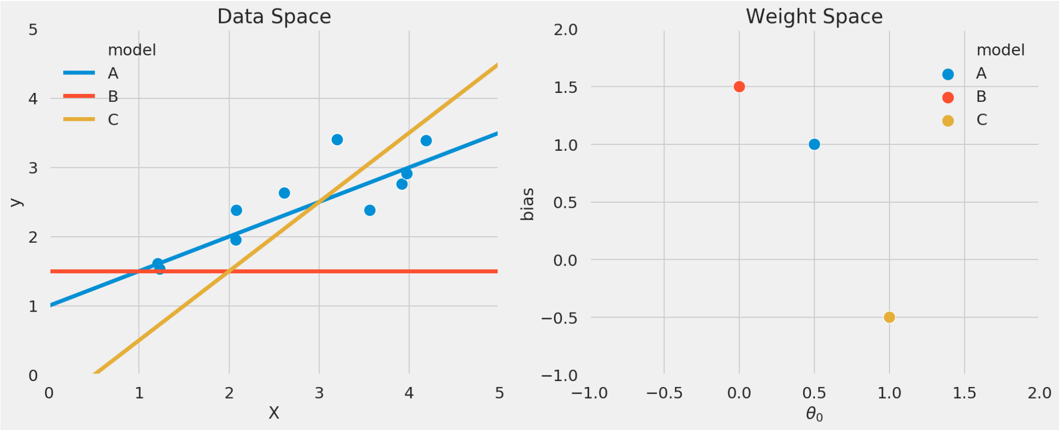
Emergent Tool Use in Hide'n'Seek

Reinforcement Learning



Outline

Linear regression



Three different linear models in the data and weight space [1].

$$y = \boldsymbol{\theta}^\top \boldsymbol{x} + b$$

[1] following Grosse, Ba - CS421, 2019

Linear models

- Linear model $y = \boldsymbol{\theta}^\intercal\, \phi(\boldsymbol{x})$, with `Mean Squared Error` objective function $\mathcal{L}(\boldsymbol{\theta}) = (t - y)^2$

Linear models

- Linear model $y = \boldsymbol{\theta}^\intercal \, \phi(\boldsymbol{x})$, with `Mean Squared Error` objective function $\mathcal{L}(\boldsymbol{\theta}) = (t - y)^2$
- Has a nice closed form solution: $(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{t}$

Linear models

- Linear model $y = \boldsymbol{\theta}^\top \phi(\boldsymbol{x})$, with `Mean Squared Error` objective function $\mathcal{L}(\boldsymbol{\theta}) = (t - y)^2$
- Has a nice closed form solution: $(\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{t}$
- But the solution can also be found iteratively:
  - Compute the gradient of $\mathcal{L}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = (t - y) \phi(\boldsymbol{x})$$

Linear models

- Linear model $y = \boldsymbol{\theta}^\intercal \phi(\boldsymbol{x})$, with `Mean Squared Error` objective function $\mathcal{L}(\boldsymbol{\theta}) = (t - y)^2$
- Has a nice closed form solution: $(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{t}$
- But the solution can also be found iteratively:
  - Compute the gradient of $\mathcal{L}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$:

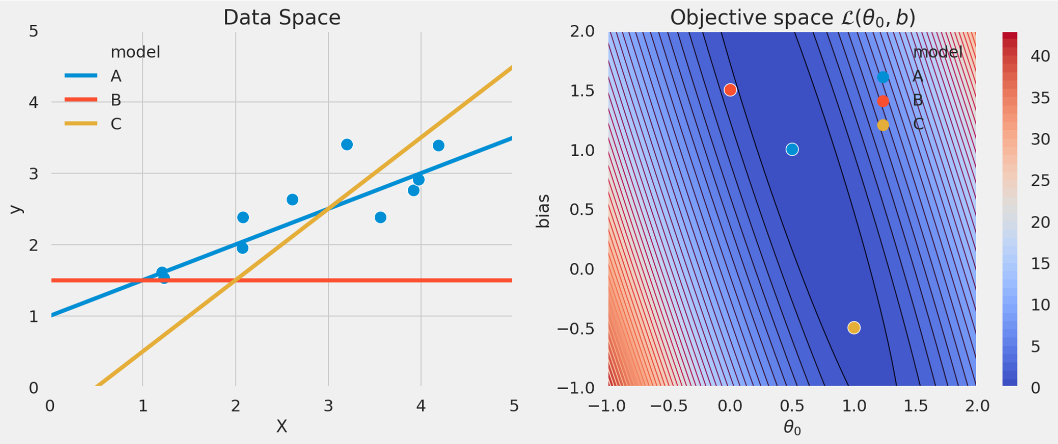
$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = (t - y)\phi(\boldsymbol{x})$$

  - Perform gradient descent:

$$\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \alpha \nabla_{\boldsymbol{\theta}_j} \mathcal{L}$$

- Why do gradient descent if we can find the minimum analytically?

Linear models

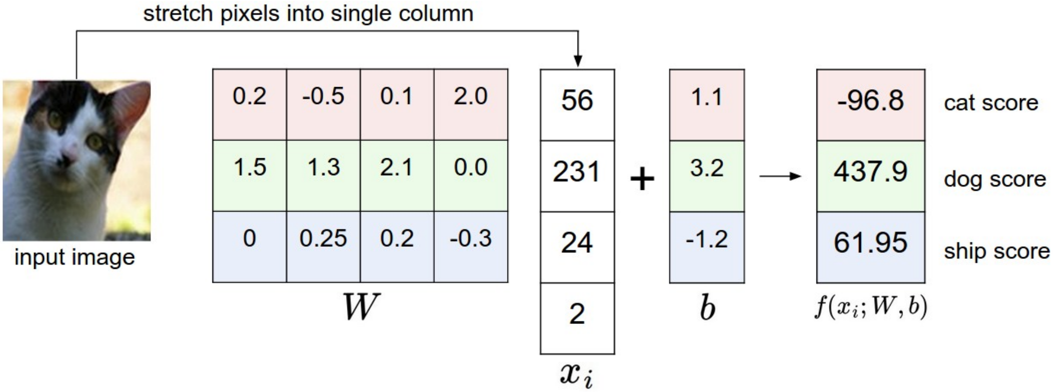


Three different linear models in the data and objective $\mathcal{L}(\theta_0, b) = [t - f(x; \theta_0, b)]^2$ space.

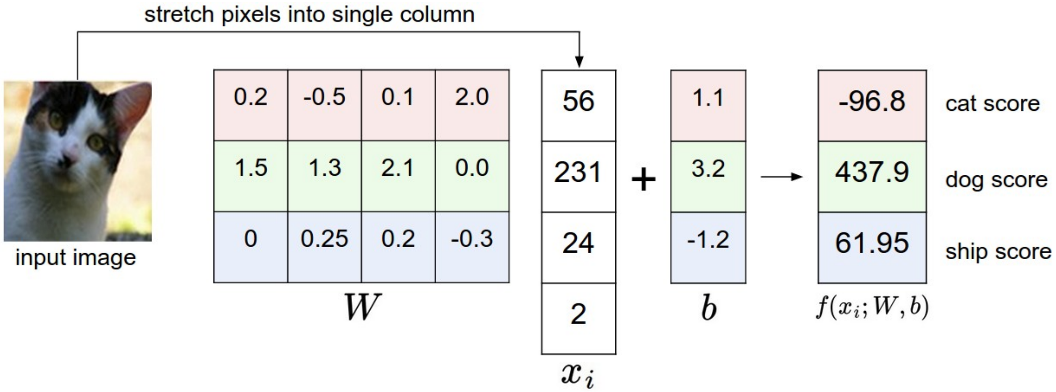
following Grosse, Ba - CS421, 2019

Outline

Linear classification

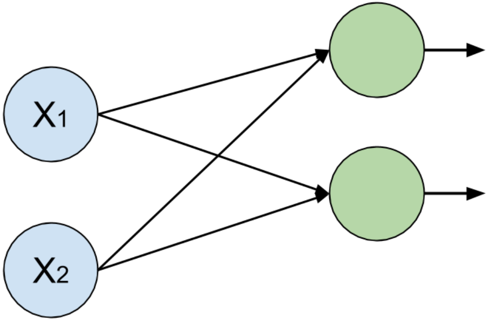


Linear classification



Learned weights:

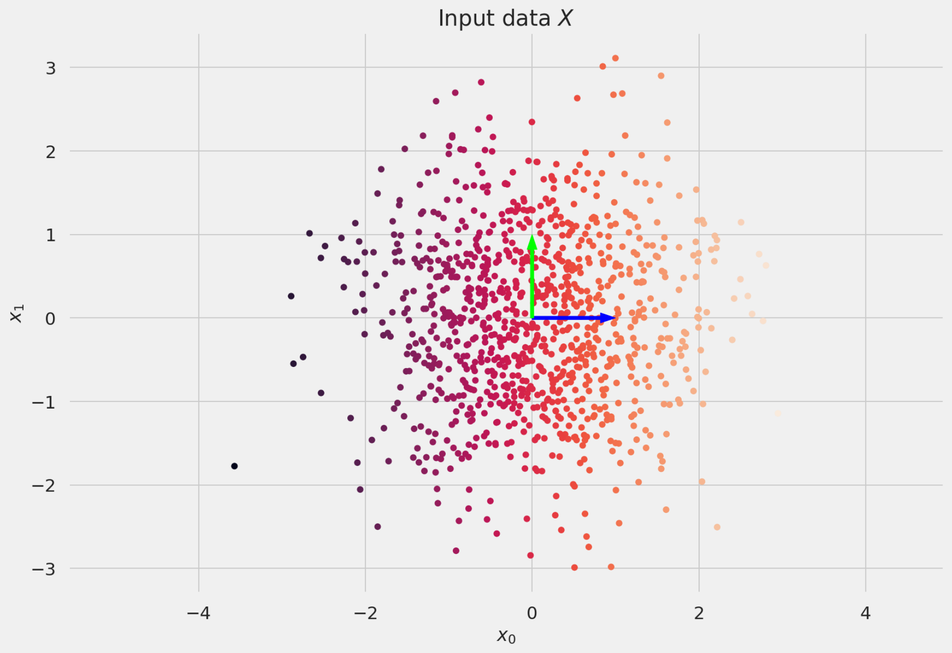
A simple transformation



A linear model with four weights. Also can be seen as two stacked "neurons" without activation functions.

Linear case: $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$

A simple transformation



A simple transformation

- $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$
- What is $\boldsymbol{W}$ actually doing to $\boldsymbol{x}$?

[1] Mathematics for Machine Learning, ch. 4

A simple transformation

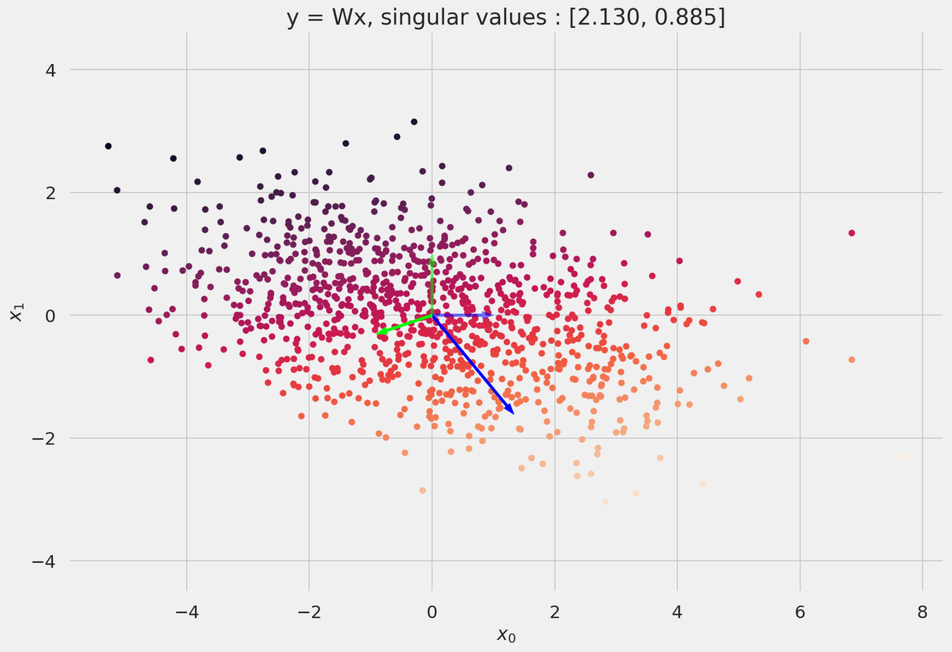
- $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$
- What is $\boldsymbol{W}$ actually doing to $\boldsymbol{x}$?
- Let's perform singular value decomposition[1] for some intuition:

$$\boldsymbol{W} = \boldsymbol{U} \times \boldsymbol{S} \times \boldsymbol{V}^\intercal$$

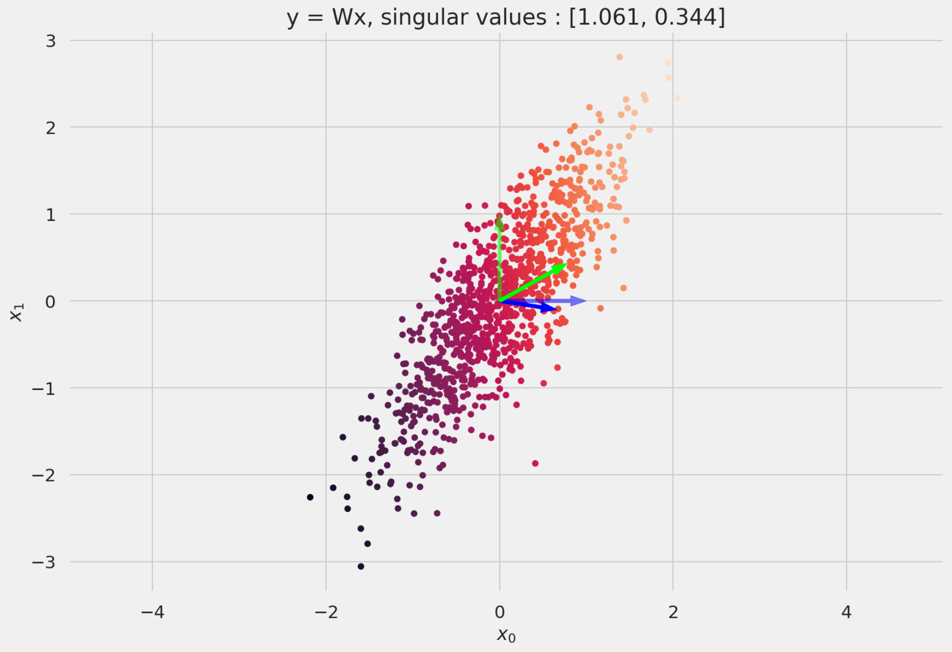
$$\begin{bmatrix} 0.19 & 1.84 \\ -0.97 & -0.93 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.83 & -0.55 \\ -0.55 & -0.83 \end{bmatrix}}_{\text{rotation}} \times \underbrace{\begin{bmatrix} 2.17 & 0.00 \\ 0.00 & 0.74 \end{bmatrix}}_{\text{scale}} \times \underbrace{\begin{bmatrix} 0.32 & 0.94 \\ 0.94 & -0.32 \end{bmatrix}}_{\text{reflection}}$$

[1] Mathematics for Machine Learning, ch. 4

A simple transformation



A simple transformation



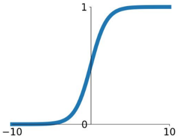
A simple transformation

We can **scale**, **rotate** and **reflect** data. Can we do more?

Non-linear functions[1]

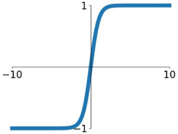
Sigmoid

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



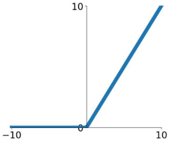
tanh

$\tanh(x)$



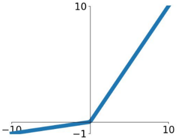
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

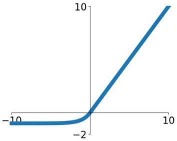


Maxout

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

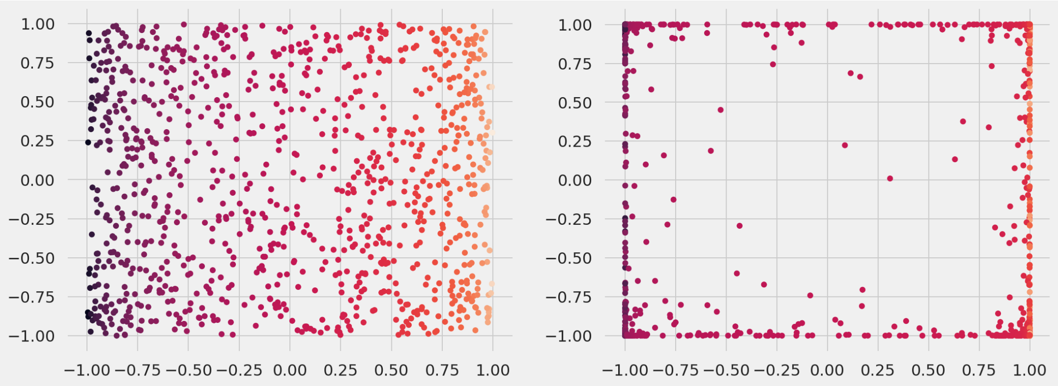
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



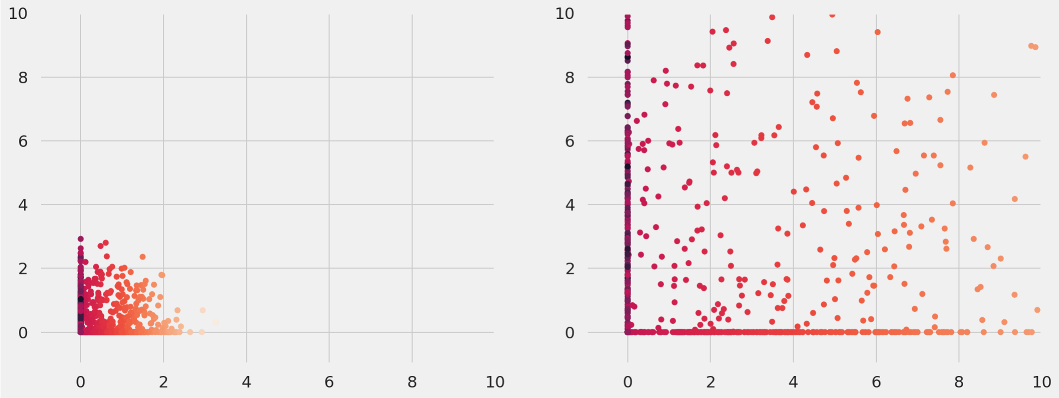
[1] from CS231n, lecture 4, 2019

Non-linear transformation



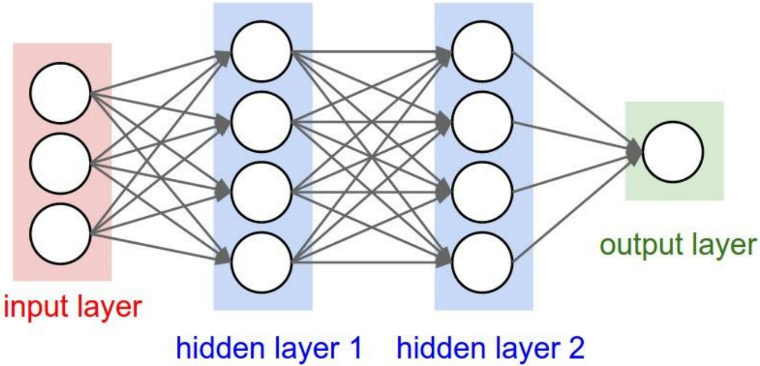
$\mathtt{tanh}(\boldsymbol{Sx})$ activation function with two different scale factors (left=1.0, right=5.0).

Non-linear transformation



$\mathtt{ReLU}(\boldsymbol{Sx})$ activation function with two different scale factors (left=1.0, right=5.0).

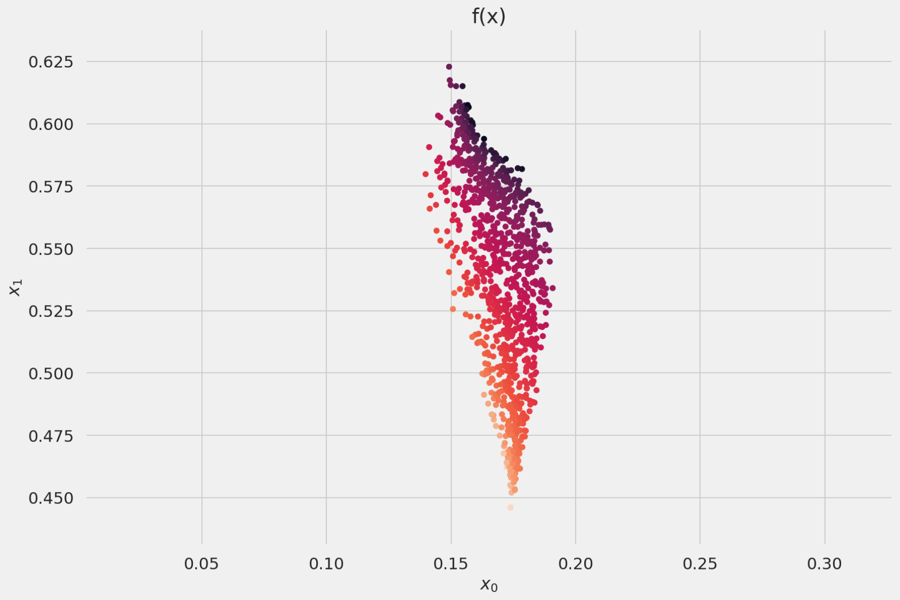
Neural Networks, finally



Typical "three-layer" or "two-hidden-layer" neural network

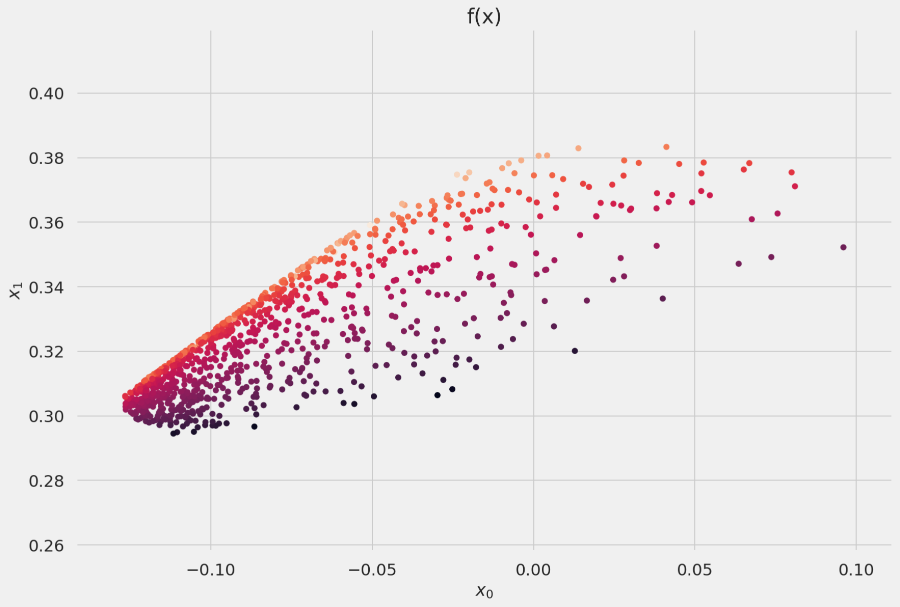
- Linear case: $f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}$
- Neural Net: $f(\boldsymbol{x}) = \boldsymbol{W}_3\,\sigma(\boldsymbol{W}_2\,\sigma(\boldsymbol{W}_1\boldsymbol{x}))$

NN transformation



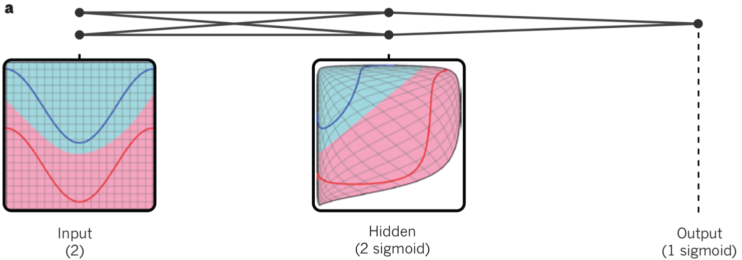
Transformation performed by a 5-layer **random** neural network

NN transformation



Transformation performed by a 5-layer **random** neural network

Warping space. Why is it usefull.

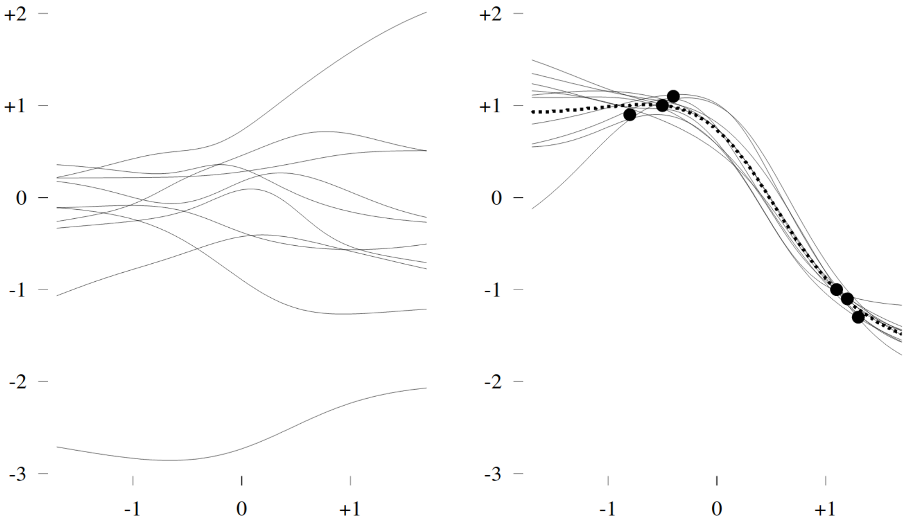


Space warping by a one hidden layer neural network for learning a linearized representation of the decision boundary[1].

[1] from LeCun 2015, Nature.

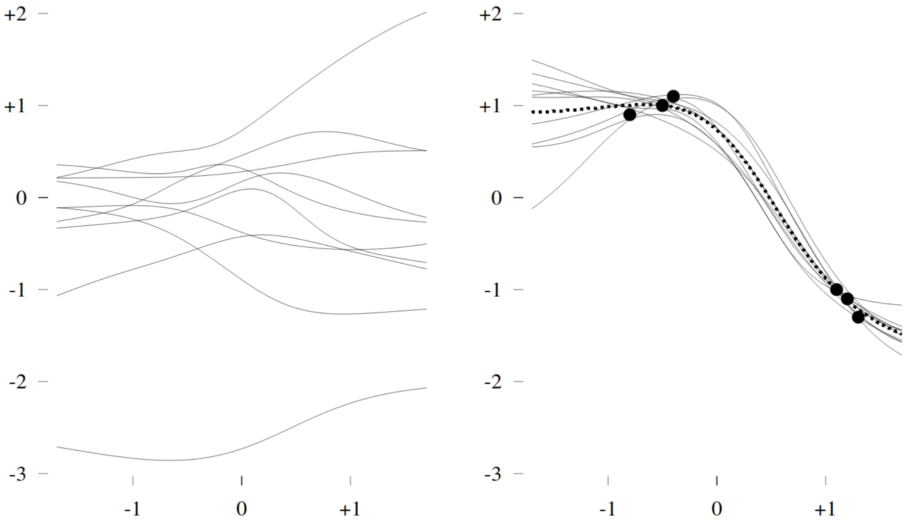
Outline

How to learn θ?



[1] Radford M. Neal, Bayesian Learning for Neural Networks

How to learn θ?



Sample 2.6 million networks and you're done!

[1] Radford M. Neal, Bayesian Learning for Neural Networks

How to learn θ?

- Can we do better?

[1] dsdeepdive.blogspot.com

How to learn θ?

- Can we do better?
- Use the lesson from the iterative linear regression

[1] dsdeepdive.blogspot.com

How to learn θ?

- Can we do better?
- Use the lesson from the iterative linear regression
- Compute the gradient of $\mathcal{L}(\mathcal{D}, \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$,

[1] dsdeepdive.blogspot.com

How to learn θ?

- Can we do better?
- Use the lesson from the iterative linear regression
- Compute the gradient of $\mathcal{L}(\mathcal{D}, \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$,
- Find incremental solutions that better explain the data.

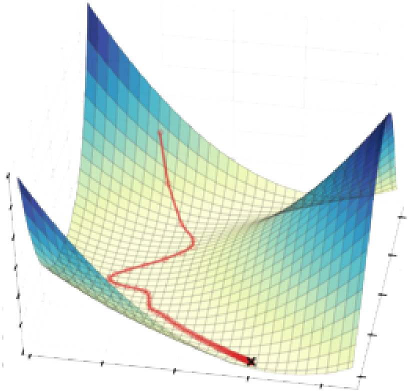
using:

$$\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \alpha \nabla_{\boldsymbol{\theta}_j} \mathcal{L}$$

sequentially find:

$$\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k$$

such that $L(\mathcal{D}, \boldsymbol{\theta})$decreases.



Sequential minimizing $L(\mathcal{D}, \theta)$ [1]

[1] dsdeepdive.blogspot.com

The problem...

Computing the gradient of $\mathcal{L}(\mathcal{D}, \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ can quickly get tedious and inflexible:

- Additional cost functions, regularization loss, etc...
- We would want to quickly experiment with different layers and activations.

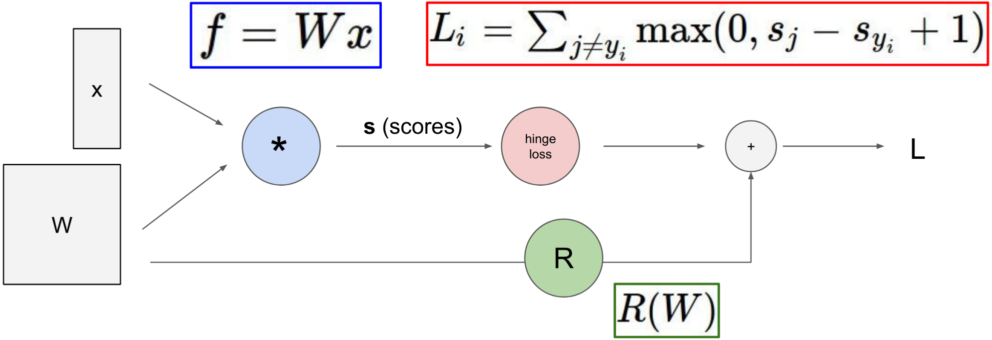
But it's not that complicated[1]

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

[1] from CS231n, lecture 4, 2019

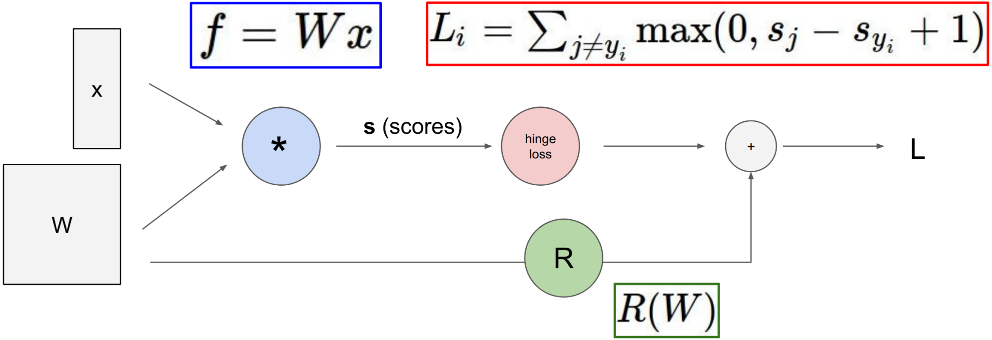
Outline

Computation graphs + automatic differentiation



[1] Grosse, Ba, CS421

Computation graphs + automatic differentiation



Automatic differentiation: takes a program which computes a value, and automatically constructs a procedure for computing derivatives of that value[1].

---

[1] Grosse, Ba, CS421

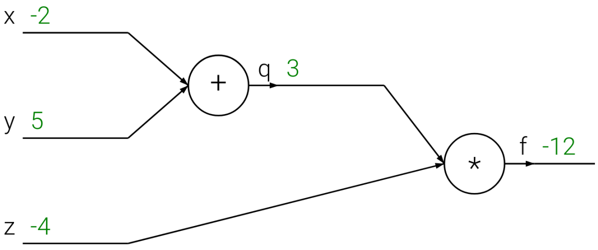
Simple example[1]

$$f(x, y, z) = (x + y)z$$

[1] from CS231n, lecture 4, 2019

Simple example[1]

$$f(x, y, z) = (x + y)z$$



[1] from CS231n, lecture 4, 2019

Simple example[1]

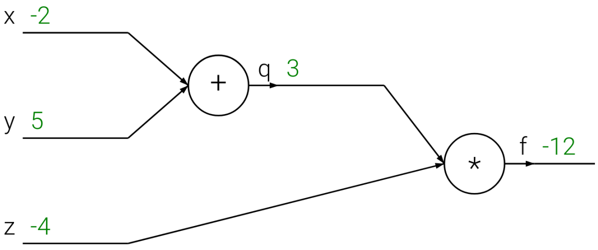
$$f(x, y, z) = (x + y)z$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

We want:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



[1] from CS231n, lecture 4, 2019

Simple example[1]

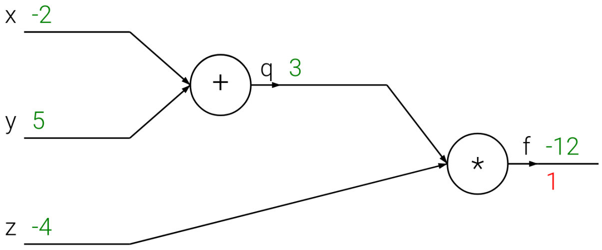
$$f(x, y, z) = (x + y)z$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

We want:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



[1] from CS231n, lecture 4, 2019

Simple example[1]

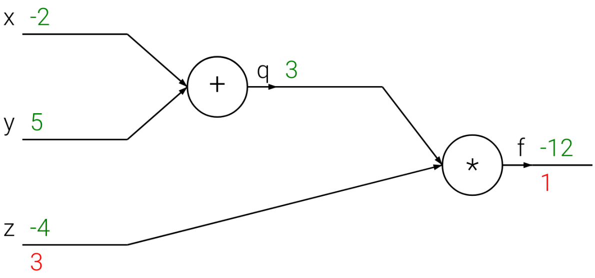
$$f(x, y, z) = (x + y)z$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

We want:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



[1] from CS231n, lecture 4, 2019

Simple example[1]

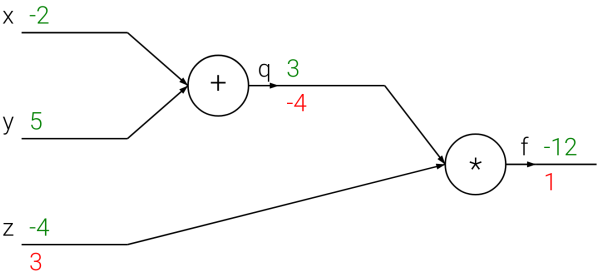
$$f(x, y, z) = (x + y)z$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

We want:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



[1] from CS231n, lecture 4, 2019

Simple example[1]

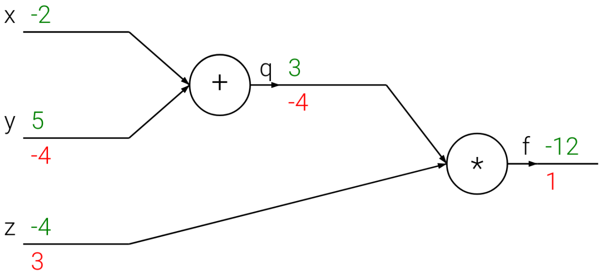
$$f(x, y, z) = (x + y)z$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

We want:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

[1] from CS231n, lecture 4, 2019

Simple example[1]

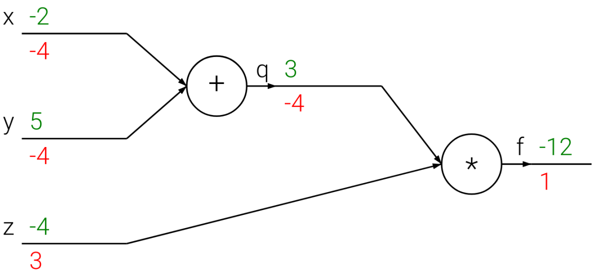
$$f(x, y, z) = (x + y)z$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

We want:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

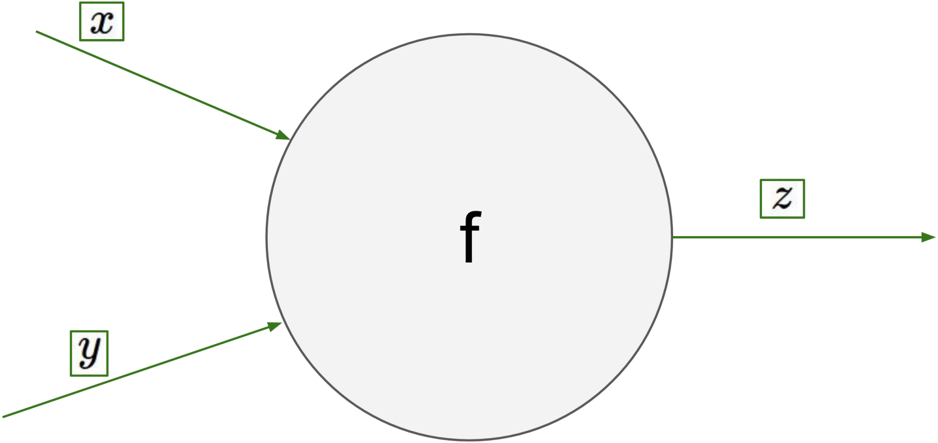


$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

[1] from CS231n, lecture 4, 2019

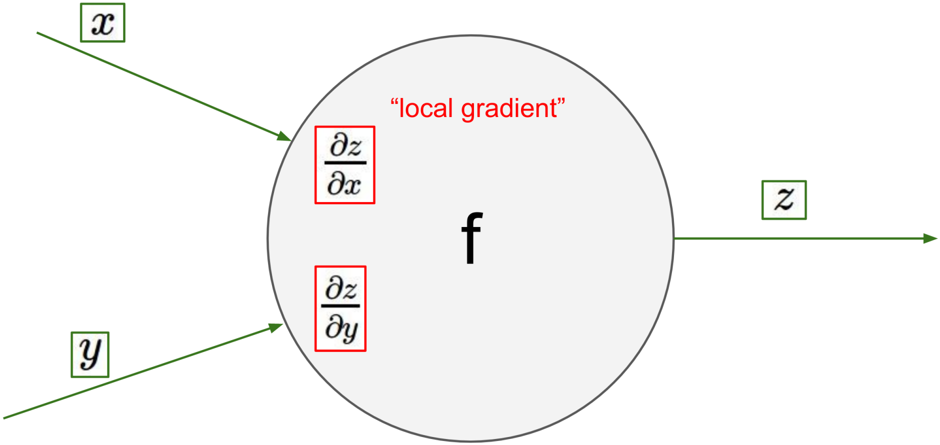
Link: Automatic Differentiation in PyTorch

Backprop[1]



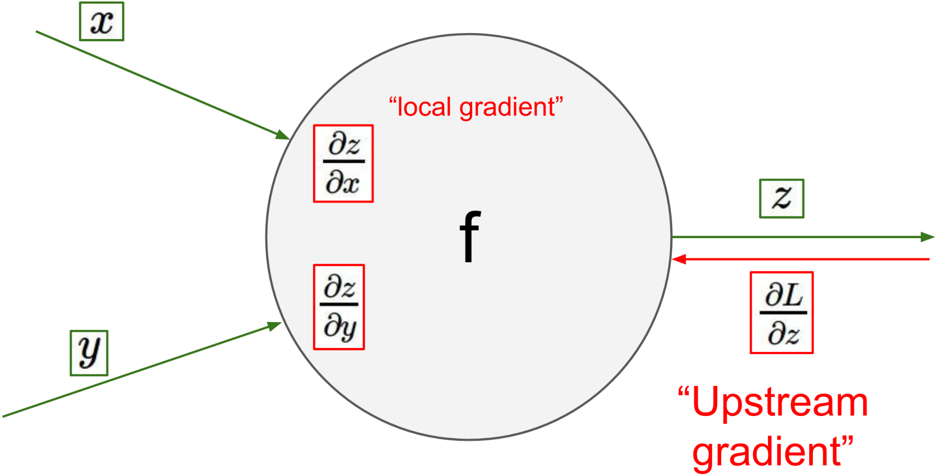
[1] from CS231n, lecture 4, 2019

Backprop[1]



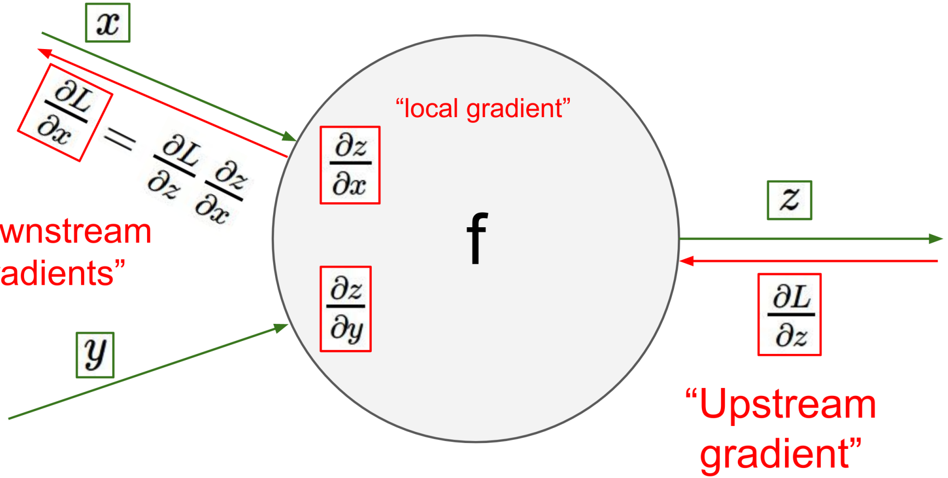
[1] from CS231n, lecture 4, 2019

Backprop[1]



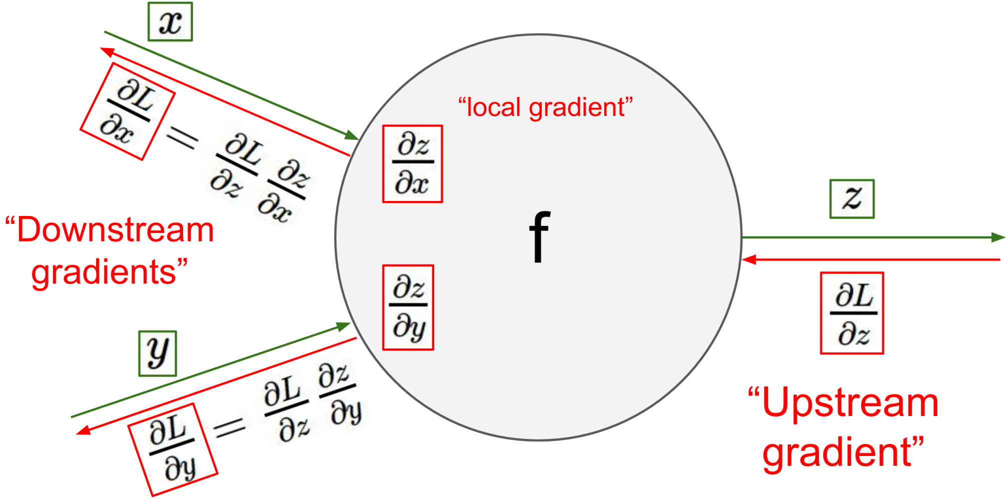
[1] from CS231n, lecture 4, 2019

Backprop[1]



[1] from CS231n, lecture 4, 2019

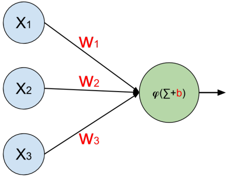
Backprop[1]



[1] from CS231n, lecture 4, 2019

Outline

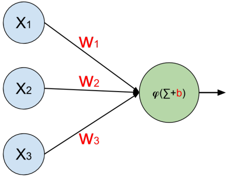
NN = Tensor Operations



Perceptron

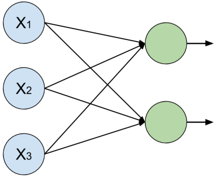
$$\begin{bmatrix} X1 & X2 & X3 \end{bmatrix} \bullet \begin{bmatrix} W1 \\ W2 \\ W3 \end{bmatrix} + \begin{bmatrix} B \end{bmatrix} = \begin{bmatrix} P \end{bmatrix}$$

NN = Tensor Operations



Perceptron

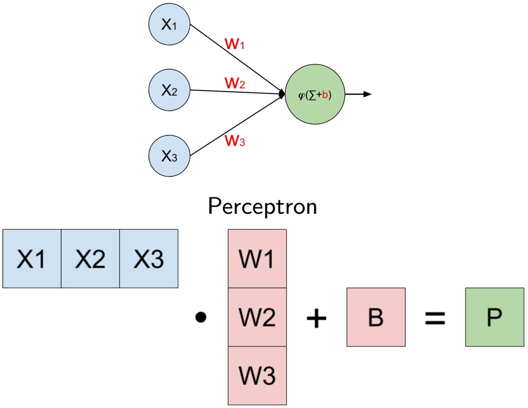
$$\begin{bmatrix} X1 & X2 & X3 \end{bmatrix} \bullet \begin{bmatrix} W1 \\ W2 \\ W3 \end{bmatrix} + \begin{bmatrix} B \end{bmatrix} = \begin{bmatrix} P \end{bmatrix}$$

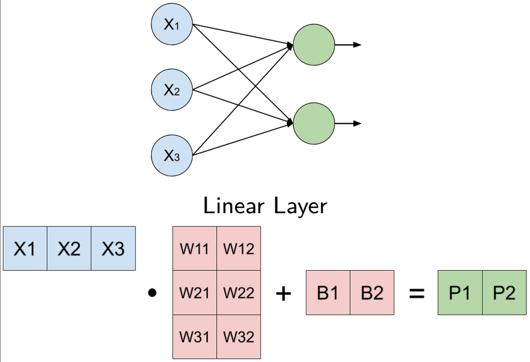


Linear Layer

$$\begin{bmatrix} X1 & X2 & X3 \end{bmatrix} \bullet \begin{bmatrix} W11 & W12 \\ W21 & W22 \\ W31 & W32 \end{bmatrix} + \begin{bmatrix} B1 & B2 \end{bmatrix} = \begin{bmatrix} P1 & P2 \end{bmatrix}$$

NN = Tensor Operations





How about the case with multiple input tensors?

It's straightforward in PyTorch

```
import torch
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(input_dim, 64),
    nn.ReLU(),
    nn.Linear(64,64),
    nn.ReLU(),
    nn.Linear(64,out_dim)
)

dummy_inputs = torch.randn((1, input_dim))
prediction = model(dummy_inputs)
```

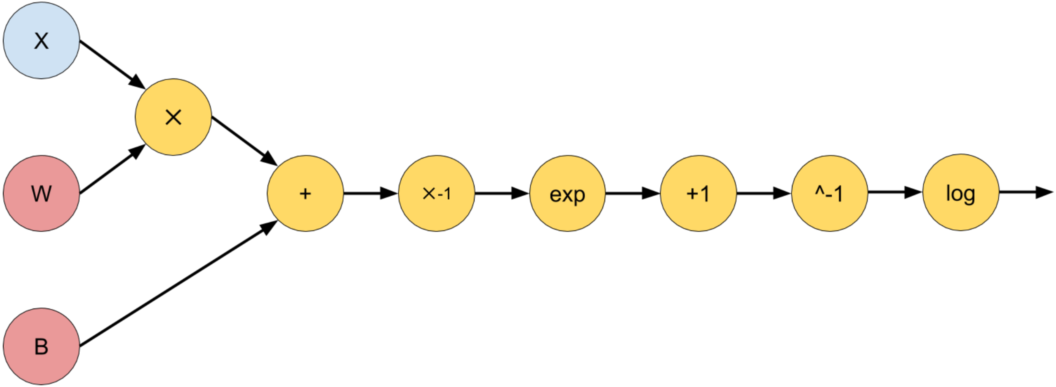
Outline

Computation Graph

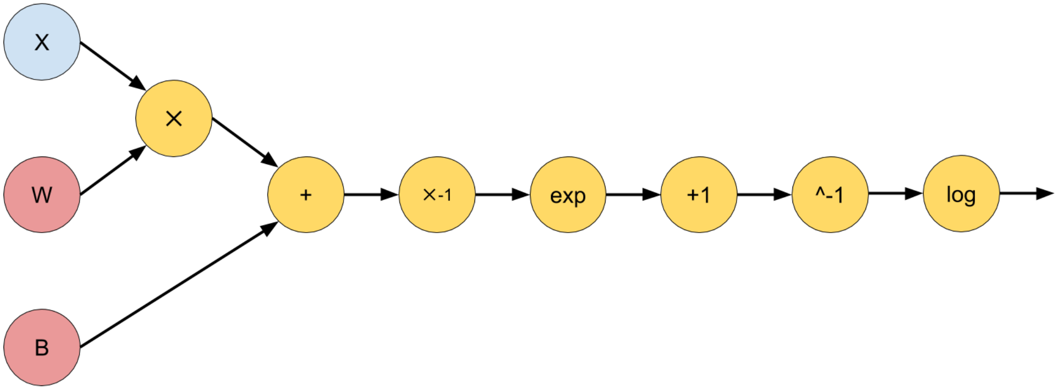
$$L(x, w, b) = log\left(\frac{1}{e^{-(xw+b)} + 1}\right)$$

Computation Graph

$$L(x, w, b) = log\left(\frac{1}{e^{-(xw+b)} + 1}\right)$$

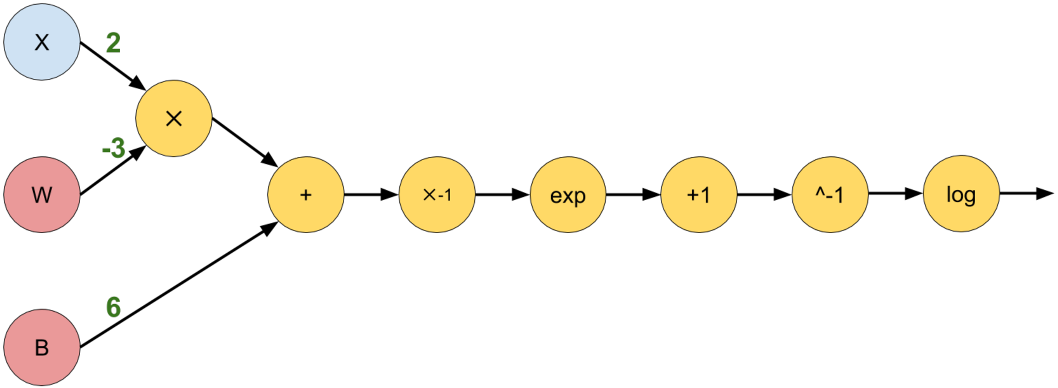


Forward



Task: Compute $L(x, w, b)$

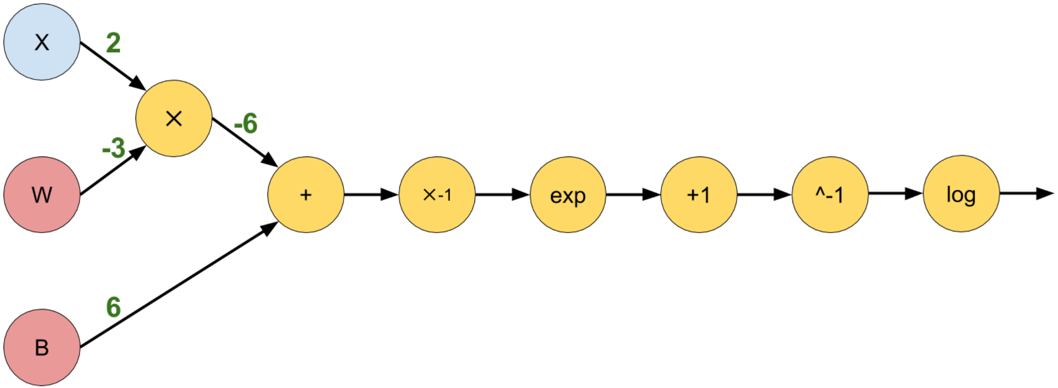
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

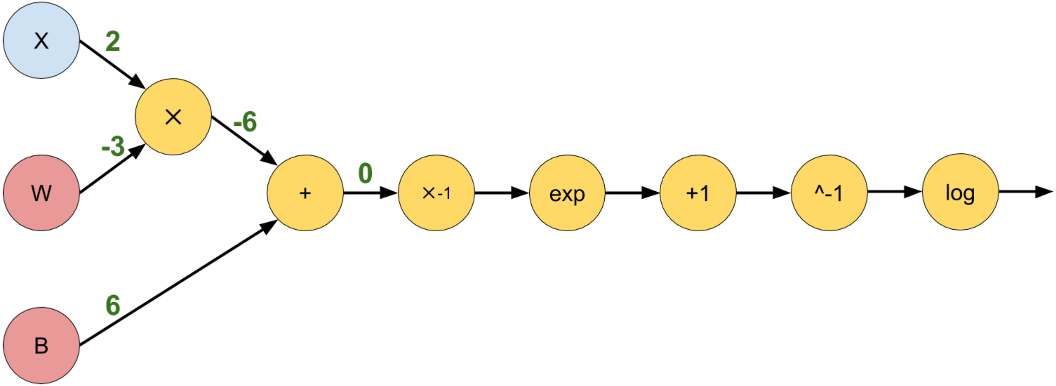
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

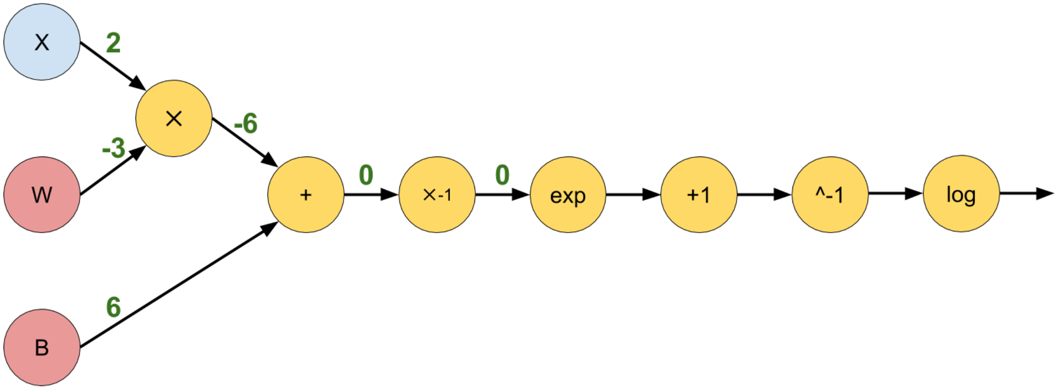
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

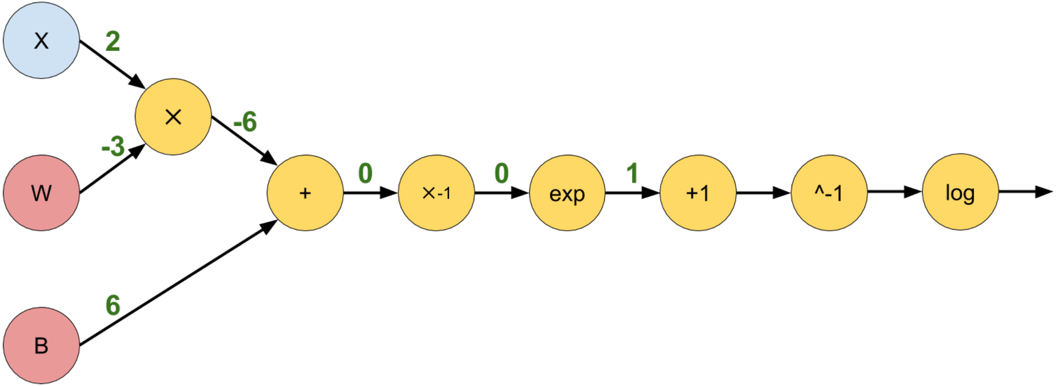
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

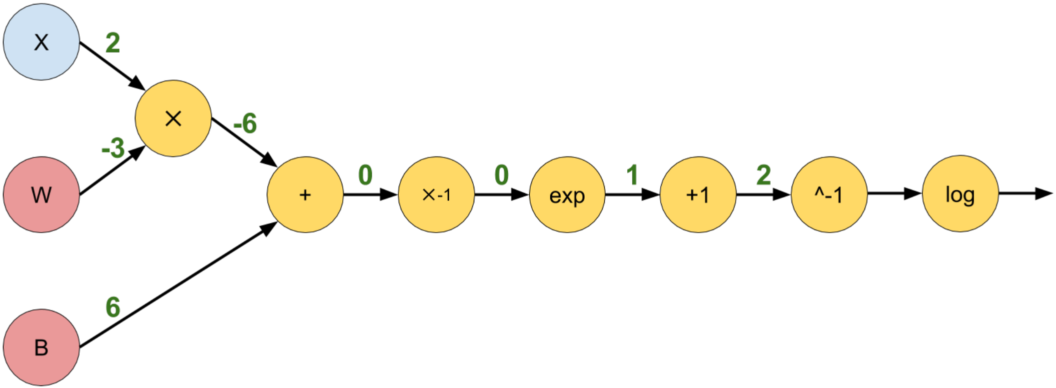
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

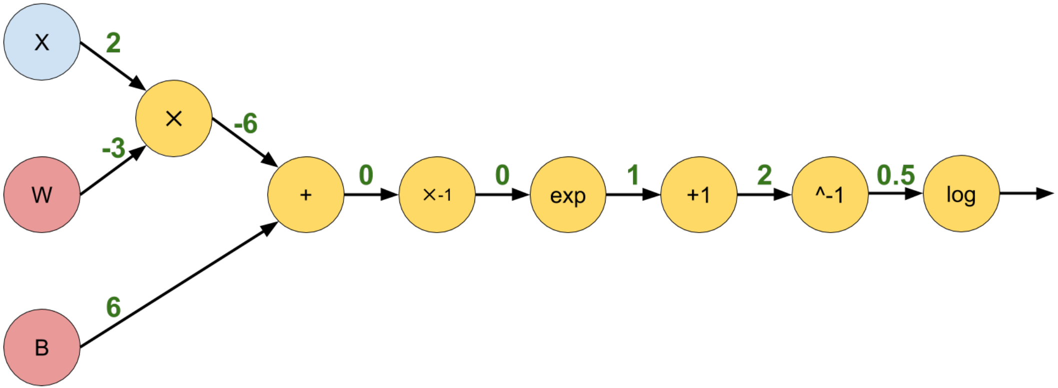
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

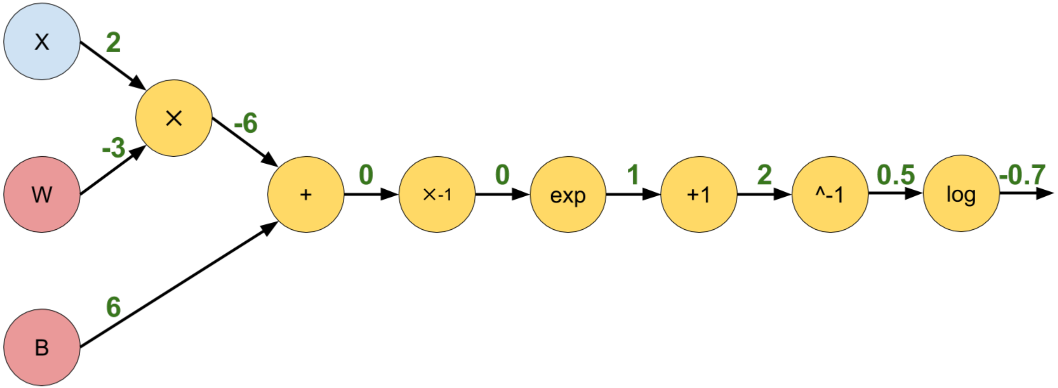
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

Forward

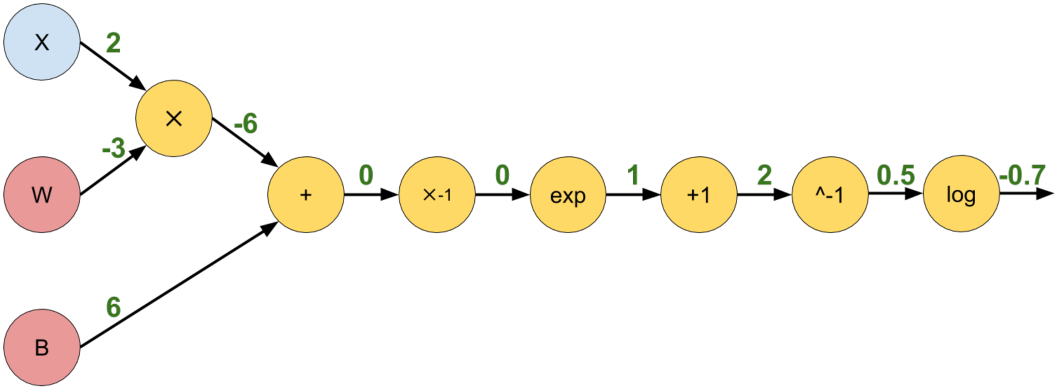


Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

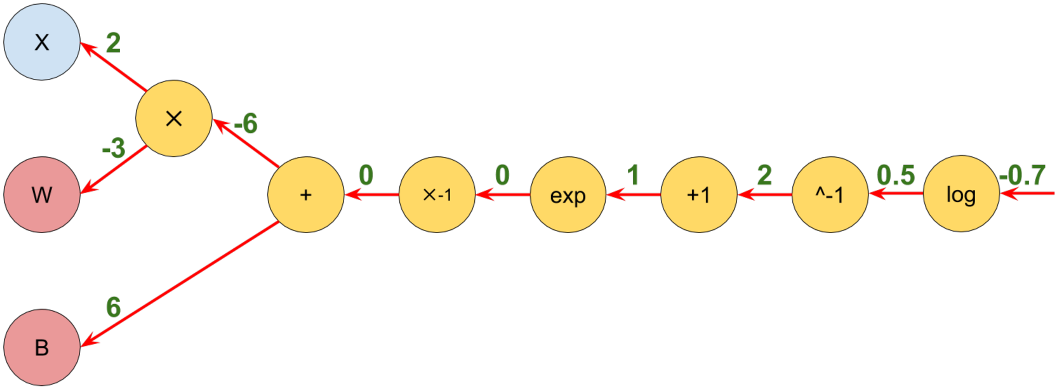
$$L(2, -3, 6) \simeq 0.7$$

Backward



Task: Compute $\nabla L = \left[\frac{\partial L}{\partial x}, \frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right]$

Backward

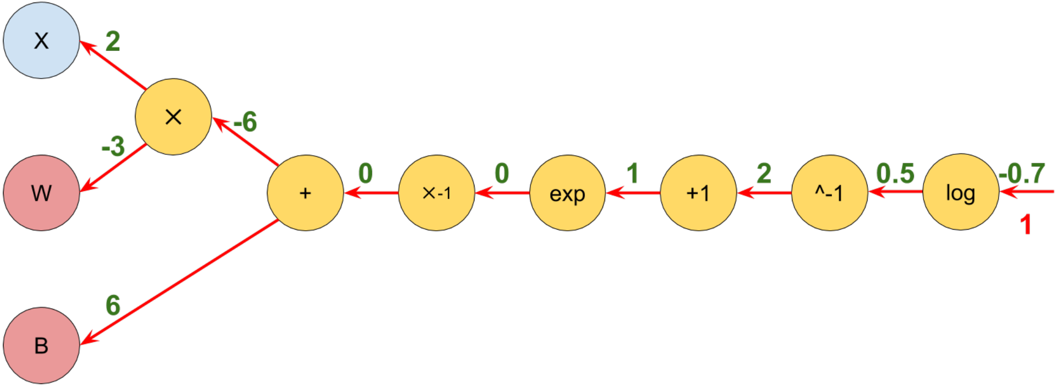


Task: Compute $\nabla L = \left[\frac{\partial L}{\partial x}, \frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right]$

Chain-rule $\implies$ each sub-module M:

- receives $\frac{\partial L}{\partial M(in)}$
- computes $\frac{\partial M(in)}{\partial in_k}$
- returns $\frac{\partial L}{\partial in_k} = \frac{\partial L}{\partial M(in)} \frac{\partial M(in)}{\partial in_k}$

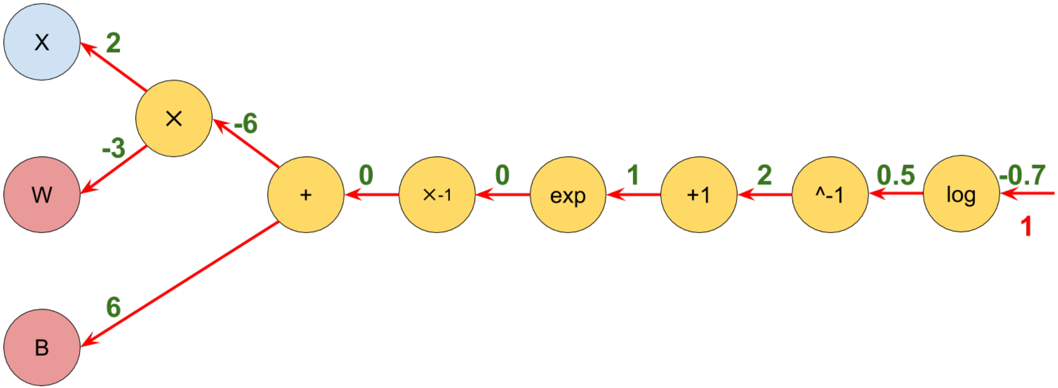
Backward



log:

- receives $\frac{\partial L}{\partial log(in)} = 1$

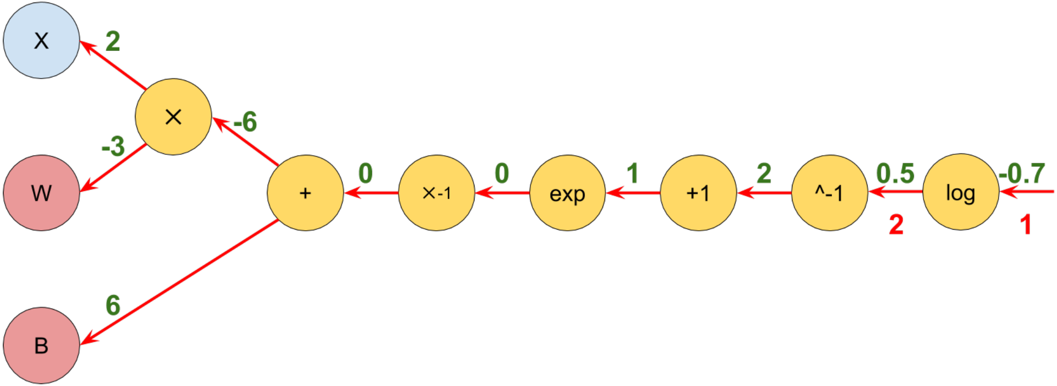
Backward



log:

- receives $\frac{\partial L}{\partial log(in)} = 1$
- computes $\frac{\partial log(in)}{\partial in} = \frac{1}{in} = 2$

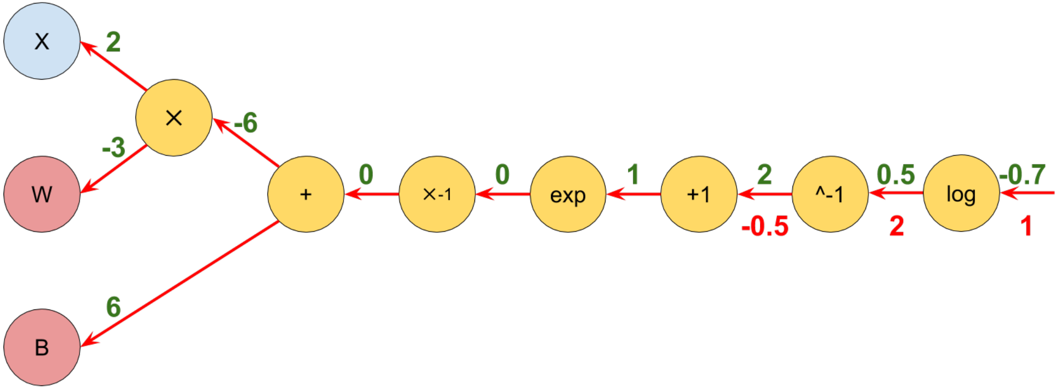
Backward



log:

- receives $\frac{\partial L}{\partial log(in)} = 1$
- computes $\frac{\partial log(in)}{\partial in} = \frac{1}{in} = 2$
- returns $\frac{\partial L}{\partial in} = 1 * 2$

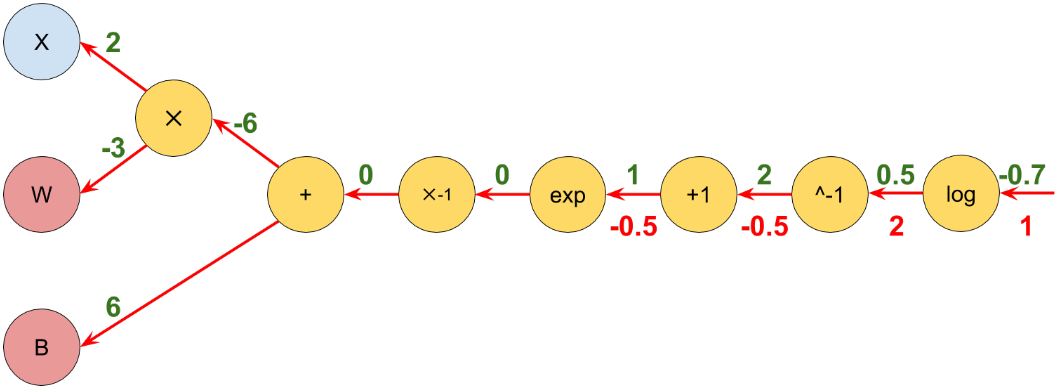
Backward



$\hat{}-1$:

- receives $\frac{\partial L}{\partial (in)^{-1}} = 2$
- computes $\frac{\partial (in)^{-1}}{\partial in} = -(in)^{-2} = -0.25$
- returns $\frac{\partial L}{\partial in} = 2 * -0.25 = -0.5$

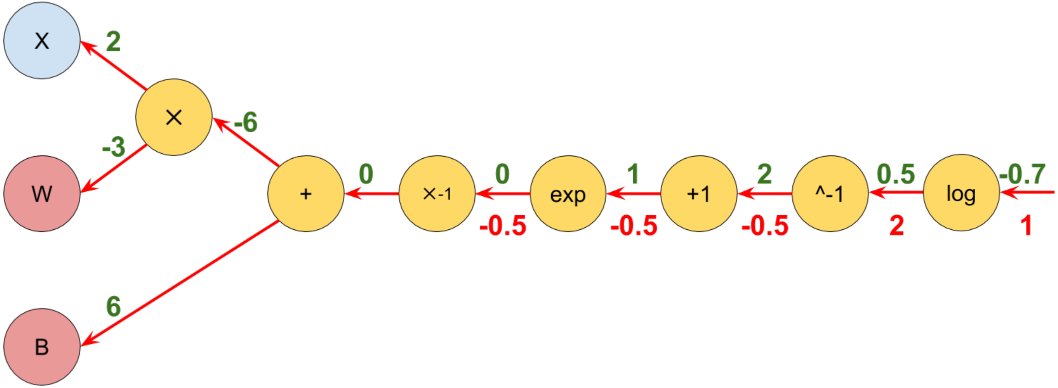
Backward



+1:

- receives $\frac{\partial L}{\partial(in+1)} = -0.5$
- computes $\frac{\partial(in+1)}{\partial in} = 1$
- returns $\frac{\partial L}{\partial in} = -0.5 * 1 = -0.5$

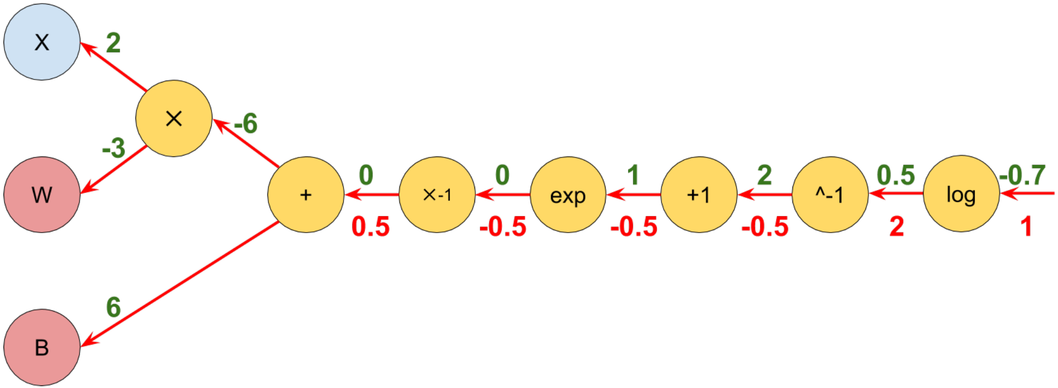
Backward



exp:

- receives $\frac{\partial L}{\partial e^{in}} = -0.5$
- computes $\frac{\partial e^{in}}{\partial in} = e^{in} = 1$
- returns $\frac{\partial L}{\partial in} = -0.5 * 1 = -0.5$

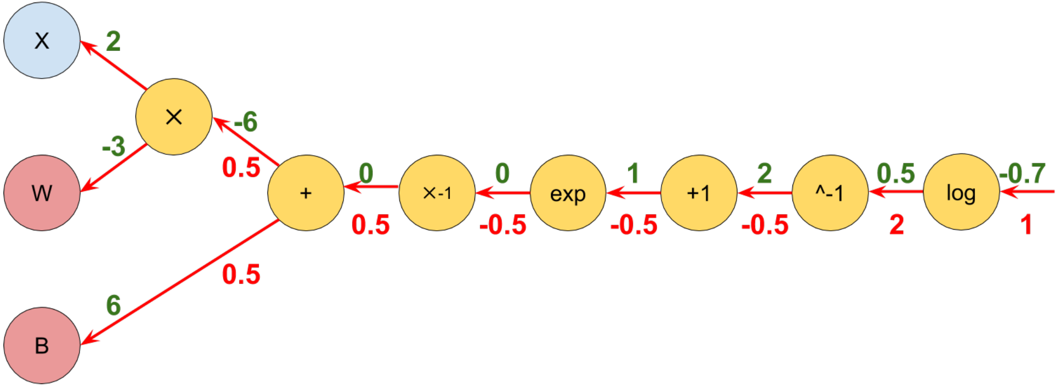
Backward



$\times - 1$:

- receives $\frac{\partial L}{\partial(-in)} = -0.5$
- computes $\frac{\partial(-in)}{\partial in} = -1$
- returns $\frac{\partial L}{\partial in} = -0.5 * -1 = 0.5$

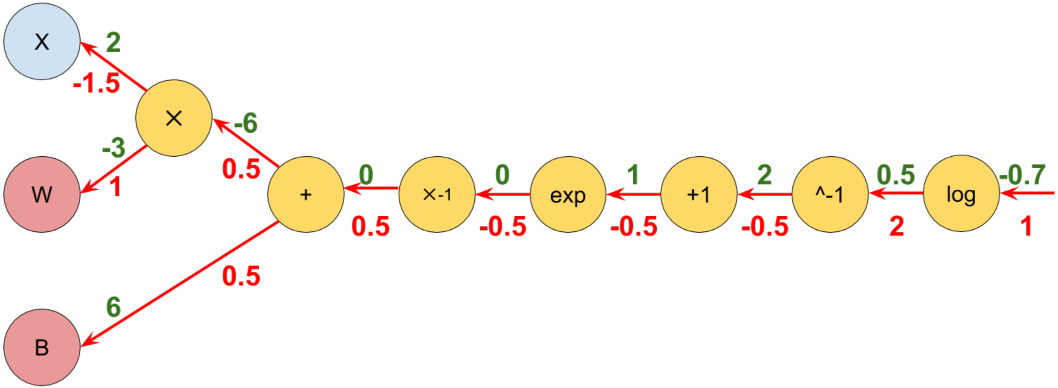
Backward



$+$:

- receives $\frac{\partial L}{\partial(in_1+in_2)} = 0.5$
- computes $\left(\frac{\partial(in_1+in_2)}{\partial in_1}, \frac{\partial(in_1+in_2)}{\partial in_2}\right) = (1, 1)$
- returns $\left(\frac{\partial L}{\partial in_1}, \frac{\partial L}{\partial in_2}\right) = (0.5, 0.5)$

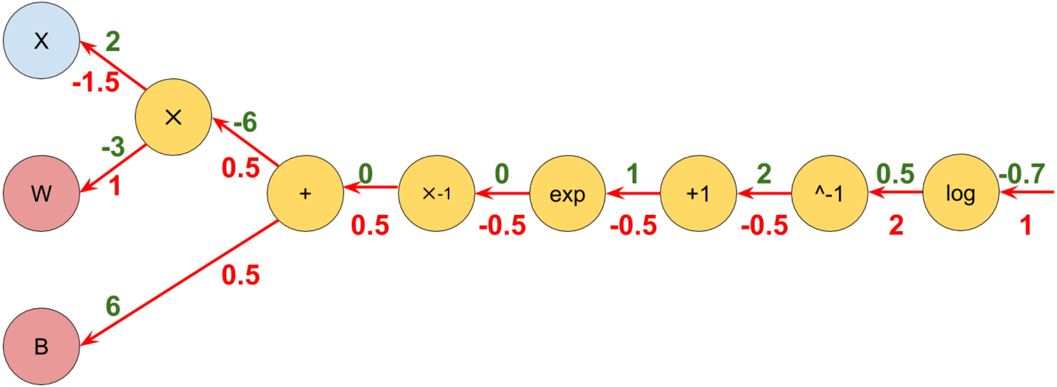
Backward



$\times$:

- receives $\frac{\partial L}{\partial(in_1 \times in_2)} = 0.5$
- computes $\left(\frac{\partial(in_1 \times in_2)}{\partial in_1}, \frac{\partial(in_1 \times in_2)}{\partial in_2}\right) = (in_2, in_1) = (-3, 2)$
- returns $\left(\frac{\partial L}{\partial in_1}, \frac{\partial L}{\partial in_2}\right) = 0.5 * (-3, 2) = (-1.5, 1)$

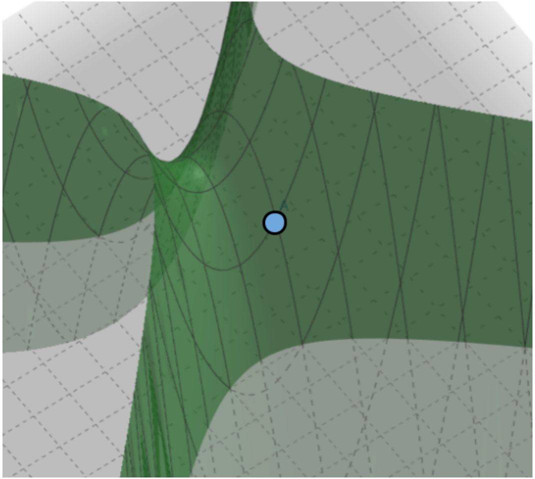
Backward



$$\nabla L = \left[\frac{\partial L}{\partial x}, \frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right] = [-1.5, 1, 0.5]$$

Outline

Gradient



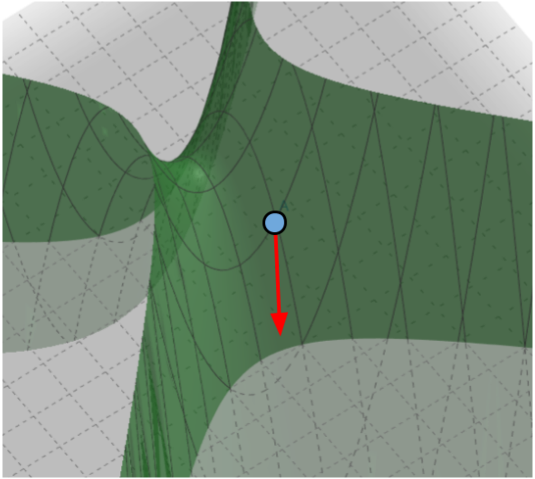
$$\theta_0 = [2, 1]$$

$$L(\theta_0) = 4$$

$$L : \mathbb{R}^2 \to \mathbb{R}$$

$$L(x, y) = x^2 - y^2 - 2x + 5y$$

Gradient



$$\theta_0 = [2, 1]$$

$$L(\theta_0) = 4$$

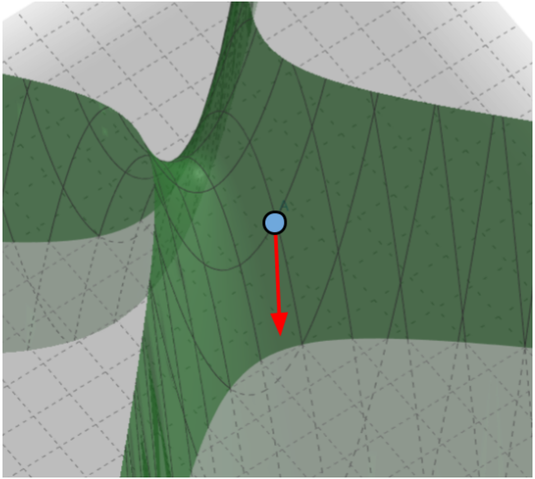
$$\nabla L = \left[\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}\right]$$

$$\frac{\partial L}{\partial x} = 2x - 2, \frac{\partial L}{\partial y} = -2y + 5$$

$$L : \mathbb{R}^2 \to \mathbb{R}$$

$$L(x, y) = x^2 - y^2 - 2x + 5y$$

Gradient



$$L : \mathbb{R}^2 \to \mathbb{R}$$

$$L(x, y) = x^2 - y^2 - 2x + 5y$$

$$\theta_0 = [2, 1]$$

$$L(\theta_0) = 4$$

$$\nabla L = \left[\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}\right]$$

$$\frac{\partial L}{\partial x} = 2x - 2, \frac{\partial L}{\partial y} = -2y + 5$$

$$\nabla L(\theta_0) = [2, 3]$$

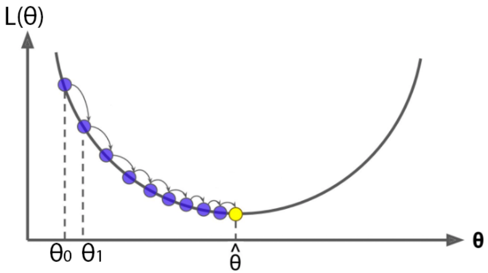
$$\theta_1 = \theta_0 - 0.1 \nabla L(\theta_0) = [1.8, 0.7]$$

$$L(\theta_1) = 2.65$$

Gradient Descent

- $L : \mathbb{R}^k \to \mathbb{R}, min_\theta\{L(\theta)\}$
- choose a learning rate $\eta \in \mathbb{R}$
- sample randomly θ_0
- repeat until *convergence*:

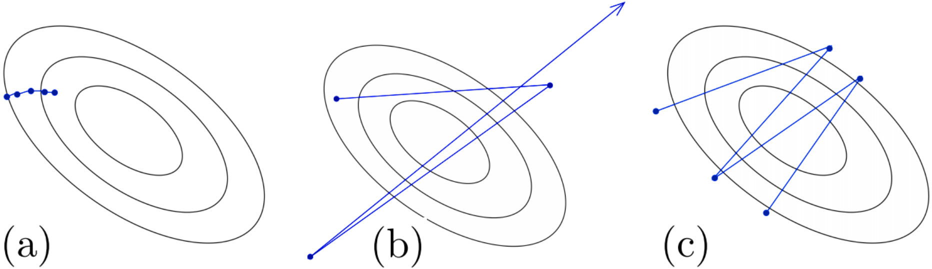
$$\theta_{i+1} = \theta_i - \eta \nabla L(\theta_i)$$



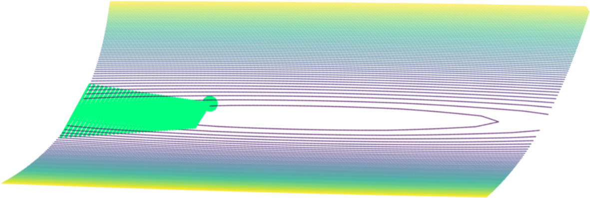
Univariate $L(\theta)$ [1]

[1] tigerthinks.com

Problems



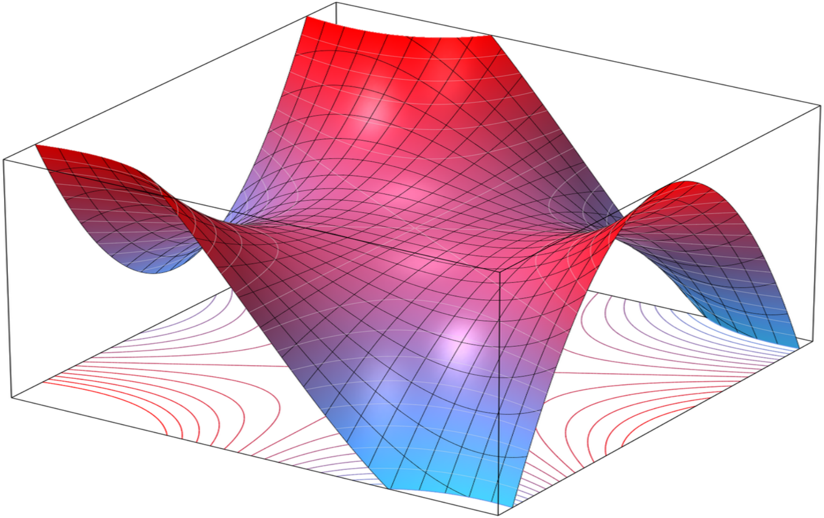
Slow Convergence (small η); Divergence (high η); Oscillations (high η) [1]



Highly different curvature

[1] cs.toronto.edu

Problems



Near-flat Dimensions

Momentum

- Use past information to take the current action
- Consistent small step in a direction $\implies$ encourage a bigger step
- Oscillating moves $\implies$ dampen movement
- Straightforward way to achieve this: use the previous update direction

Plain gradient descent:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}_t} \mathcal{L}$$

With momentum:

$$\mathbf{v}_{t+1} \leftarrow \mu \mathbf{v}_t + \nabla_{\boldsymbol{\theta}_t} \mathcal{L}$$
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \mathbf{v}_{t+1}.$$

Adaptive Learning-Rate. Adam

- Use gradient component $|\nabla L(\theta)_k|$ as signal strength.
- Amplify the parameter update step inversely proportional to signal strength.
- Take into account the entire history, emphasising recent information.

Adaptive Learning-Rate. Adam

- Use gradient component $|\nabla L(\theta)_k|$ as signal strength.
- Amplify the parameter update step inversely proportional to signal strength.
- Take into account the entire history, emphasising recent information.

$$\boldsymbol{v}_t \leftarrow \mu \boldsymbol{v}_{t-1} + (1-\alpha)\nabla_{\boldsymbol{\theta}_t}\mathcal{L}$$
$$\boldsymbol{s}_t \leftarrow \gamma \boldsymbol{s}_{t-1} + (1-\gamma)[\nabla_{\boldsymbol{\theta}_t}\mathcal{L}]^2,$$

The parameter update is then:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\boldsymbol{s}_t + \epsilon}}\boldsymbol{v}_t$$

It's easy in PyTorch

```python
model = YourFancyNeuralNetwork()
optim = torch.optim.Adam(model.parameters(), ...)

for inputs, targets in data_loader:

    predictions = model(inputs)
    loss = torch.mean_squared_error(predictions, targets)

    model.zero_grad()  # ignore this for now

    loss.backward() # computes gradients
    optim.step() # change parameters accordingly
```

In a nutshell

- Compute loss function on training data $L = \frac{1}{n} \sum_i l(h_\theta(x_i), y_i)$
- Backpropagate gradient
- Apply optimization step
- Stop when the error stops decreasing (or any other metric you are interested in stops improving)

Outline

Universal Approximation Theorem

A fully connected network with one hidden layer followed by a non-linear function can approximate any function from one finite-dimensional space to another in the limit of the **width** of the hidden layer (Hornik et al., 1990).

Universal Approximation Theorem

A fully connected network with one hidden layer followed by a non-linear function can approximate any function from one finite-dimensional space to another in the limit of the **width** of the hidden layer (Hornik et al., 1990).

Potentially the hidden layer can get exponentially large. Eg.: no of possible binary functions on vectors $\boldsymbol{v} \in \{0, 1\}^n$ is 2^{2^n}, requiring $O(2^n)$ units, each responding to a single input.

UAT and depth

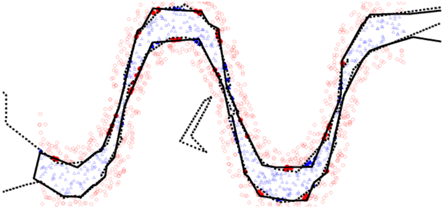
- How does depth relate to UAT?

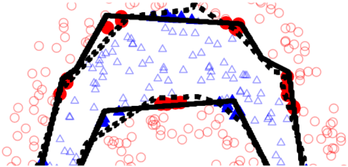
UAT and depth

- How does depth relate to UAT?
- There exist families of functions which can be approximated efficiently (in numbers of parameters) by **deep** networks.

UAT and depth

- How does depth relate to UAT?
- There exist families of functions which can be approximated efficiently (in numbers of parameters) by **deep** networks.
- Montufar et al. 2014:

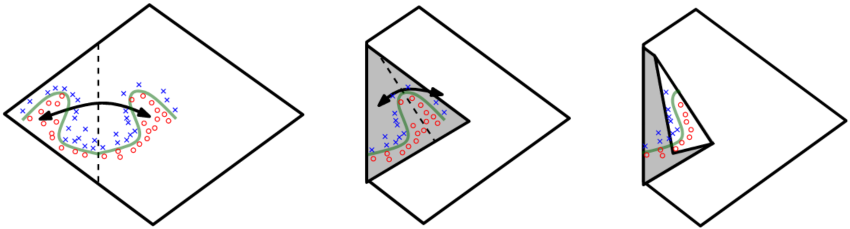




A network with ReLU units learns piece-wise linear functions at each layer and deep networks reuse these computations exponentially more often than shallow networks. Solid line is a 20 units, one layer NN and dotted line is a 2 layers, 10 units each NN.

UAT and depth

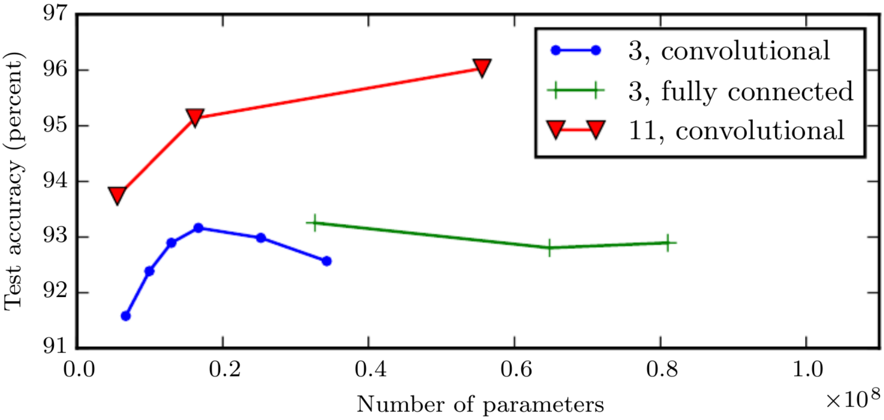
- How does depth relate to UAT?
- There exist families of functions which can be approximated efficiently (in numbers of parameters) by **deep** networks.
- Montufar et al. 2014:



A network with ReLU units creates mirror images of the function computed at the input of some hidden unit resulting in increasingly expressive networks

UAT and depth

- How does depth relate to UAT?
- There exist families of functions which can be approximated efficiently (in numbers of parameters) by **deep** networks.
- Goodfellow et al. 2014:



Generalization correlates with depth, not number of parameters.

The new generalization theory

Is the classic generalization theory still valid?