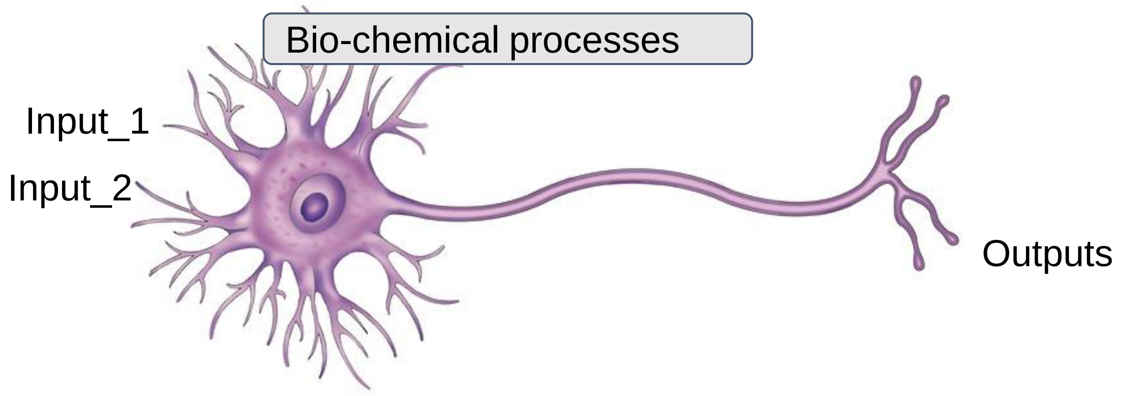
Deep Learning Fundamentals

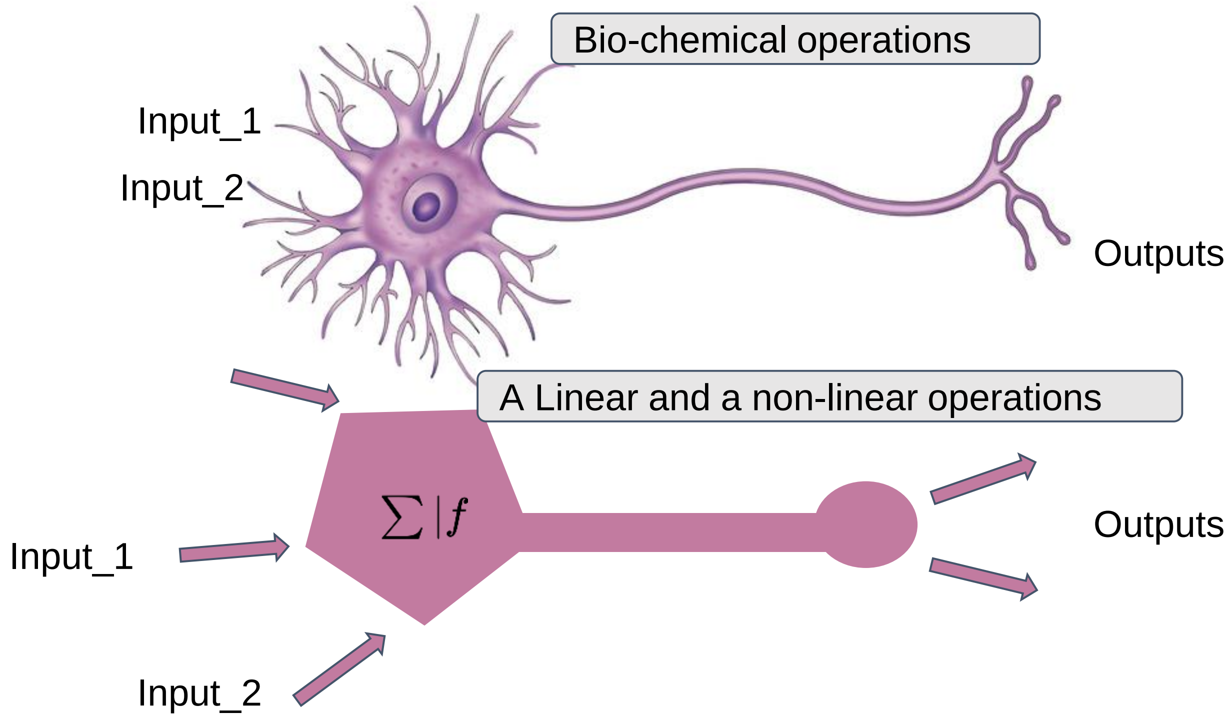
(A crash course)

Danda Pani Paudel
Computer Vision Lab @ ETH Zurich

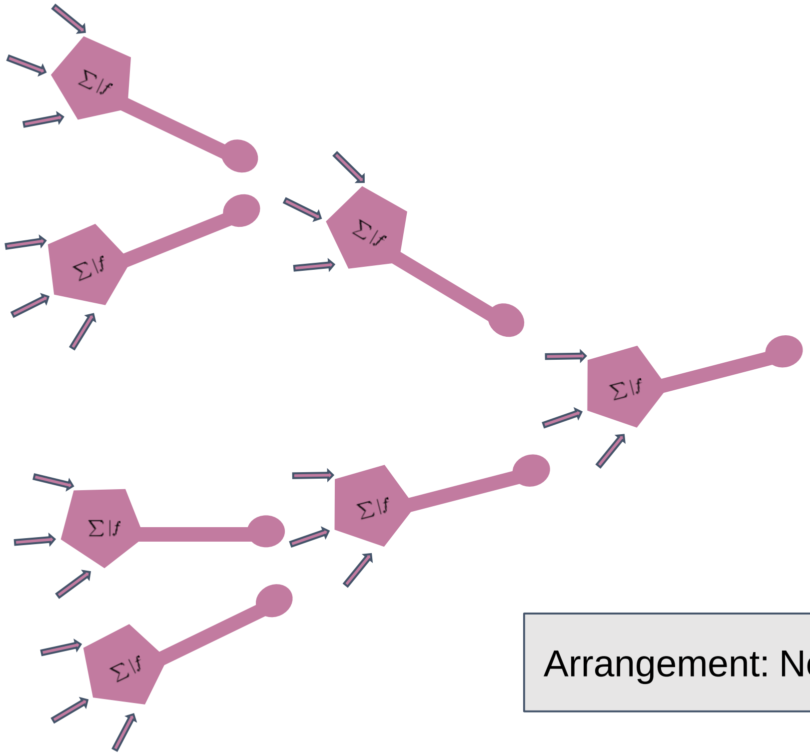
Biological Neurons and Artificial Neurons



Biological Neurons and Artificial Neurons

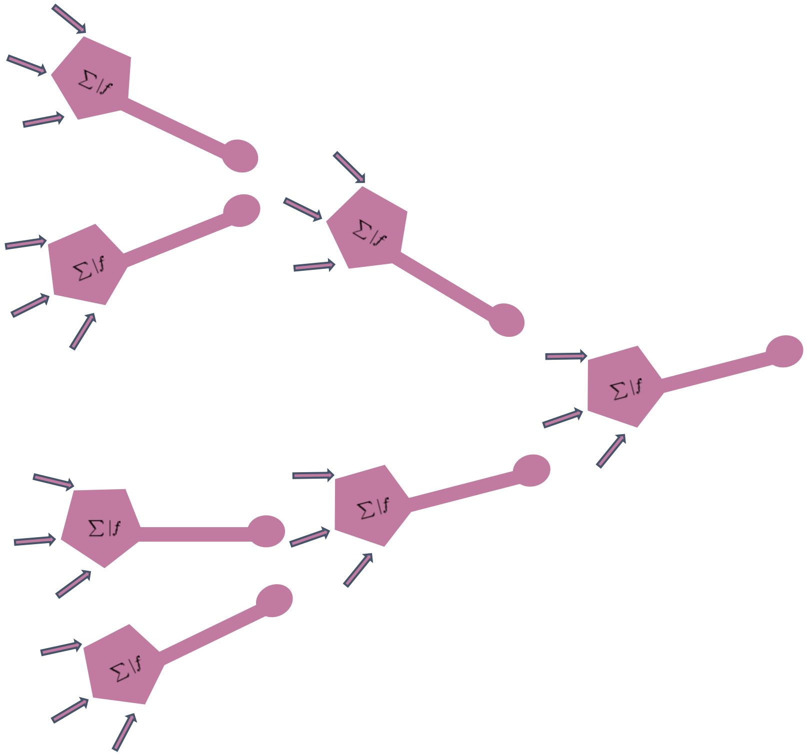


Many Artificial Neurons



Arrangement: Network Architecture

What is so special about it?



Parameters

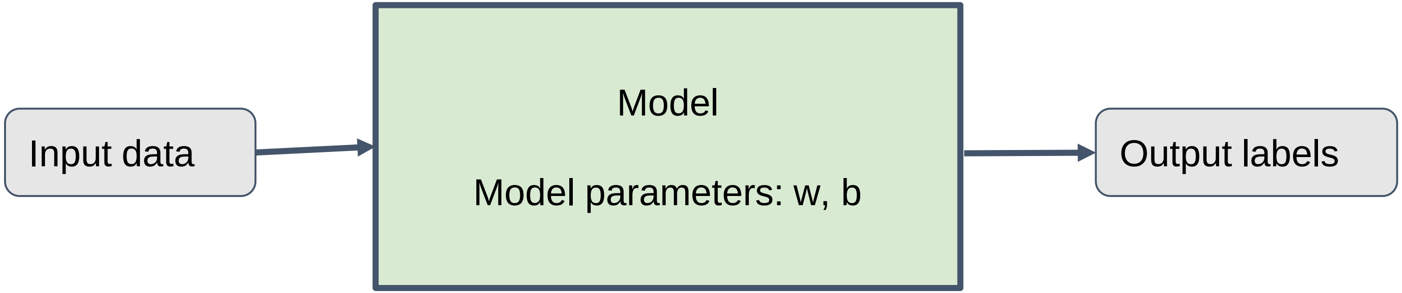
Data/Examples

Input data

Output labels

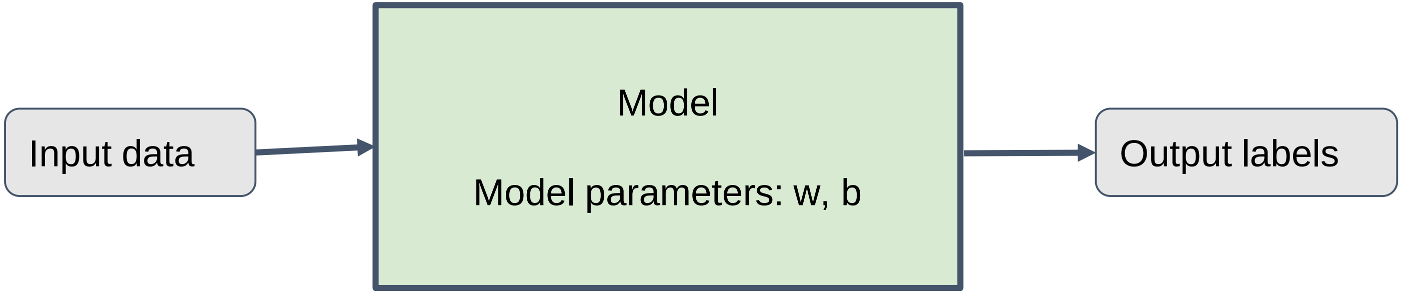
Logistic regression

Binary classification



Logistic regression

Binary classification example

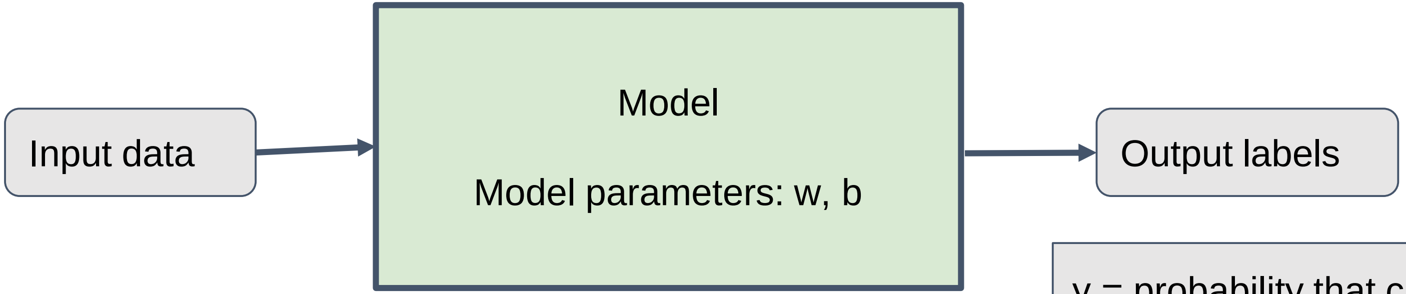


[Fuel mileage (x_1), Acceleration (x_2), Sound Frequency (x_3)]

$$x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top$$

Logistic regression

Binary classification example



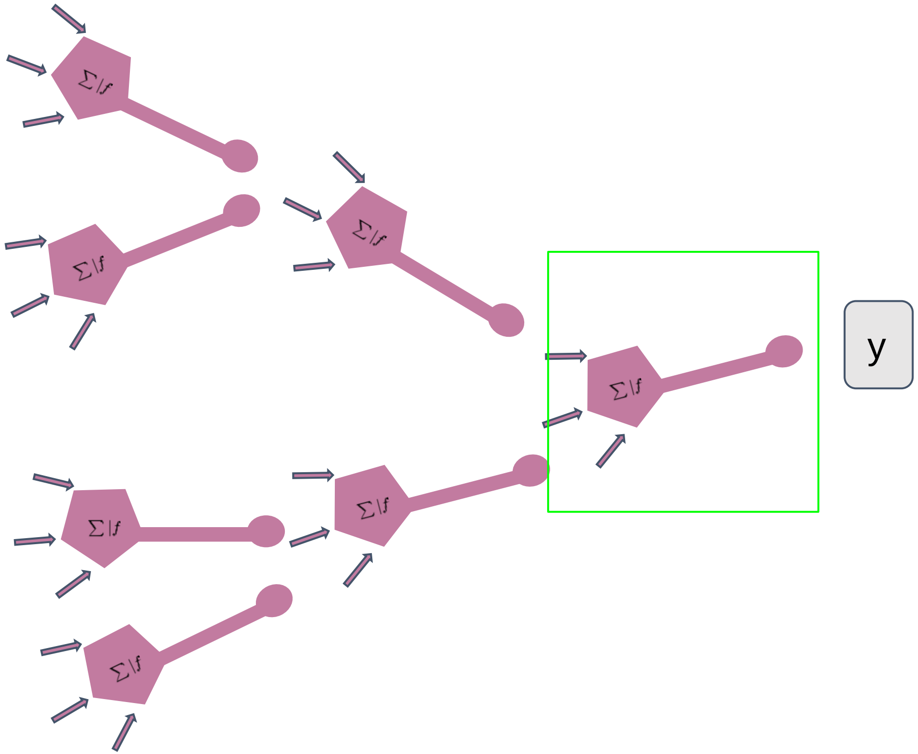
Single data point:

$$x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top$$

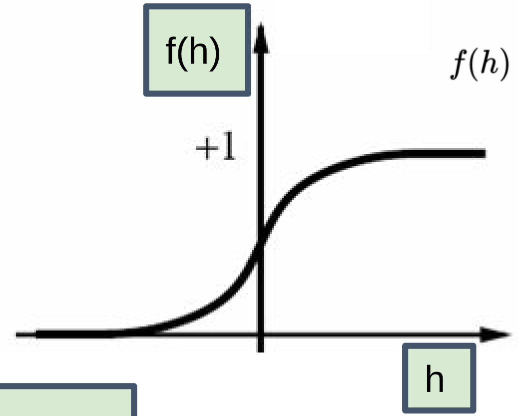
y = probability that car is in good condition
y = 0 => certain that car is broken
y = 1 => certain that car is fine

Logistic Regression

$$x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top$$



Logistic regression



$f(h) = \frac{1}{1+e^{-h}}$

Sigmoid

Binary classification example

$x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top$

Model

f(h)

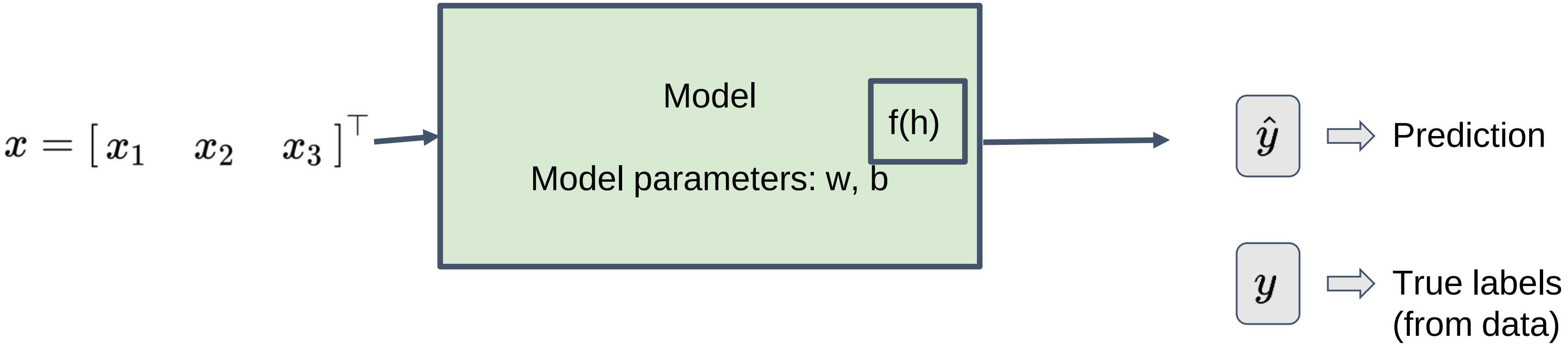
Model parameters: w, b

$\hat{y}$

Sigmoid squeezes the results between 0 and 1

Logistic regression

Binary classification example



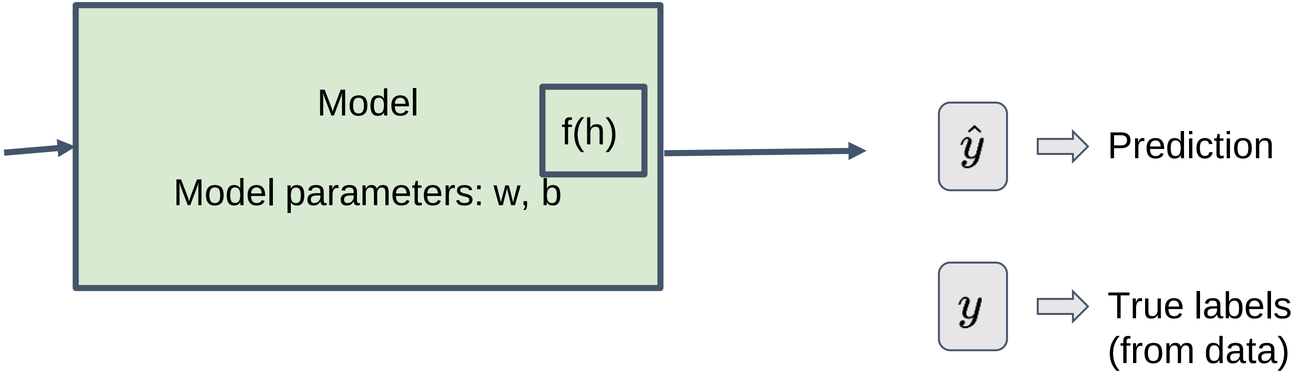
Logistic regression loss: Error of prediction from the true label

$$-\left(y \log \hat{y} + (1-y)\log(1-\hat{y})\right)$$

Logistic regression

Binary classification example

$$x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top$$



Logistic regression loss: Error of prediction from the true label

$$-\left(y \log \hat{y} + (1-y)\log(1-\hat{y})\right)$$

Logistic regression

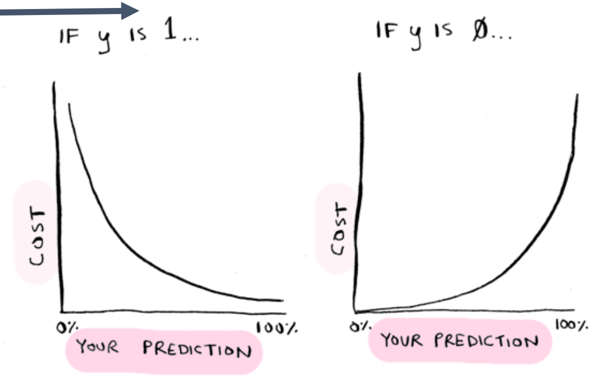
Binary classification example

$x = [x_1 \quad x_2 \quad x_3]^\top$ →

Model

$f(h)$

Model parameters: w, b



$$-(y \log \hat{y} + (1-y)\log(1-\hat{y})) \qquad 0 \rightarrow \infty$$

A Convex Function w.r.t. h!!!

Gradient Descent for Logistic Regression

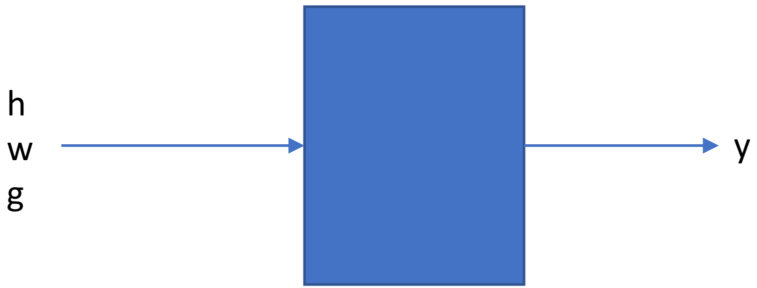
- ❑ Dummy example

Training dataset:

50 people: each person is represented by **height, weight and gender**

50 people: predict ***overweight or not***?

$$J_{\text{train}} = \sum^{50} J$$



- ❑ Gradient Descent evaluates all samples for each update

Stochastic Gradient Descent

- ❑ Dummy example

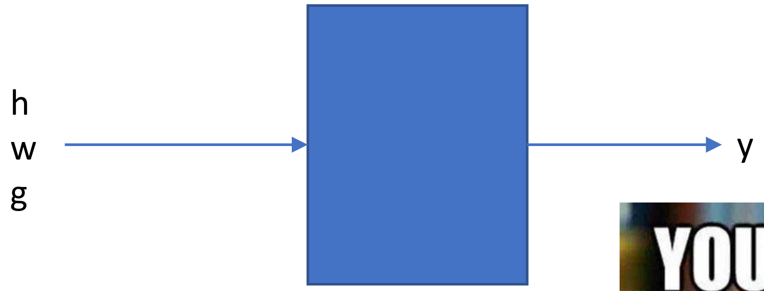
$$J_{\text{sgd}} = J_i$$

$$i = \text{random\_index}$$

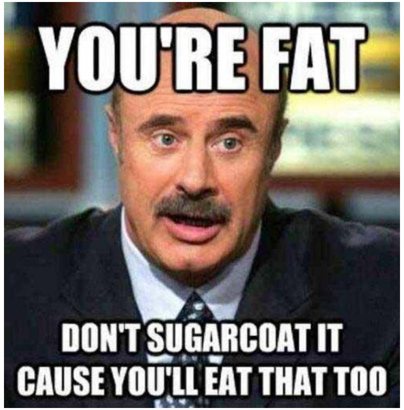
Training dataset:

50 people: each person is represented by height, weight and gender

50 people: predict ***overweight or not***?

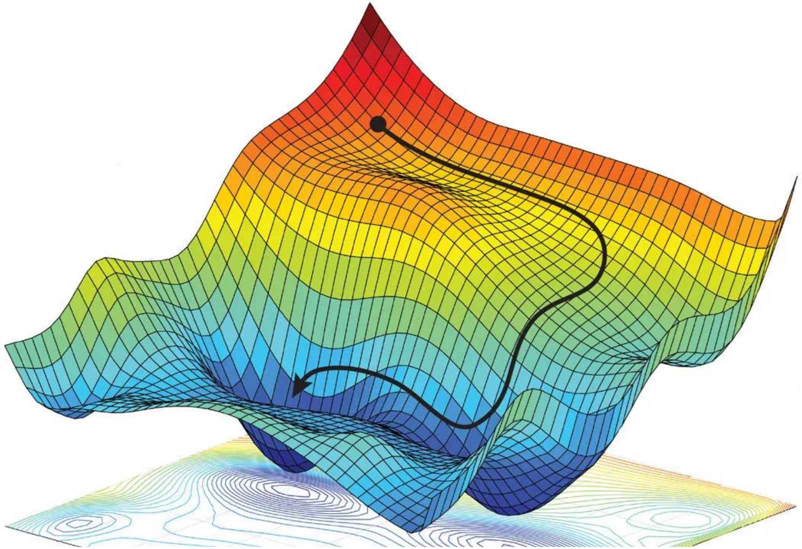


- ❑ SGD takes only one (few) random sample(s)



Why Stochastic Gradient Descent?

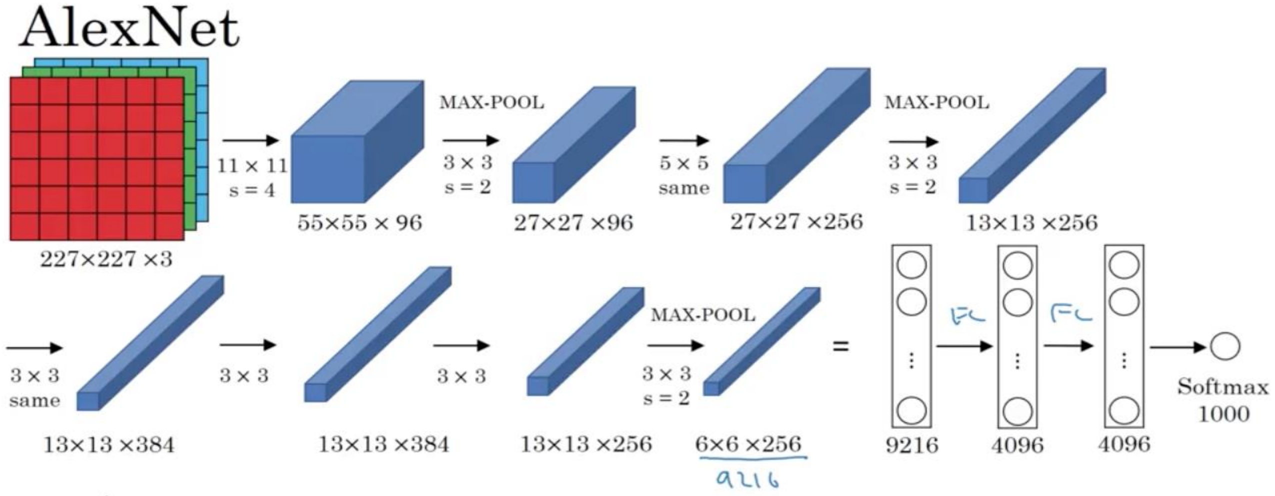
- Because of the memory/computational limitation
- To avoid local minima traps.... hopefully!



Source: https://reconsider.news/2018/05/09/ai-researchers-allege-machine-learning-alchemy/

Sneak - peak into current deep learning

- ImageNet: Learn to classify images using **1M training examples**
- Using a deep neural Network of **60 M parameters**!!

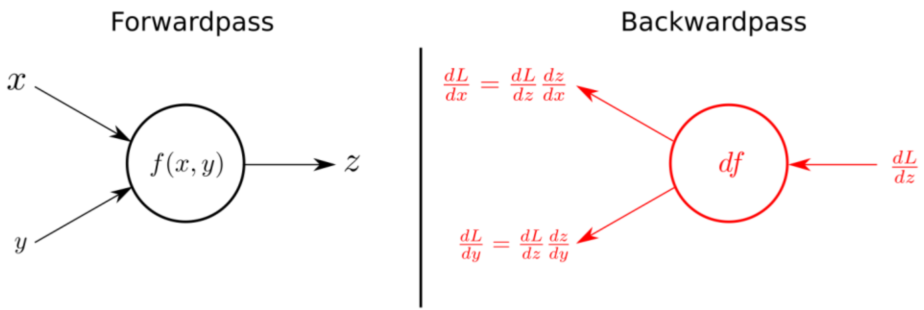


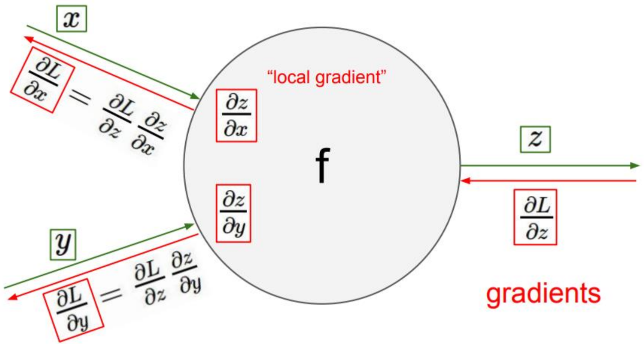
[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

Andrew Ng

How do I optimize millions of parameters?

Backpropagation





Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

©1986 **Nature Publishing Group**

$$E = \frac{1}{2}\sum_c \sum_j (y_{j,c} - d_{j,c})^2 \qquad (3)$$

The backward pass starts by computing $\partial E/\partial y$ for each of the output units. Differentiating equation (3) for a particular case, c, and suppressing the index c gives

$$\partial E/\partial y_j = y_j - d_j \qquad (4)$$

We can then apply the chain rule to compute $\partial E/\partial x_j$

$$\partial E/\partial x_j = \partial E/\partial y_j \cdot \mathrm{d}y_j/\mathrm{d}x_j$$

Differentiating equation (2) to get the value of $\mathrm{d}y_j/\mathrm{d}x_j$ and substituting gives

$$\partial E/\partial x_j = \partial E/\partial y_j \cdot y_j(1-y_j) \qquad (5)$$

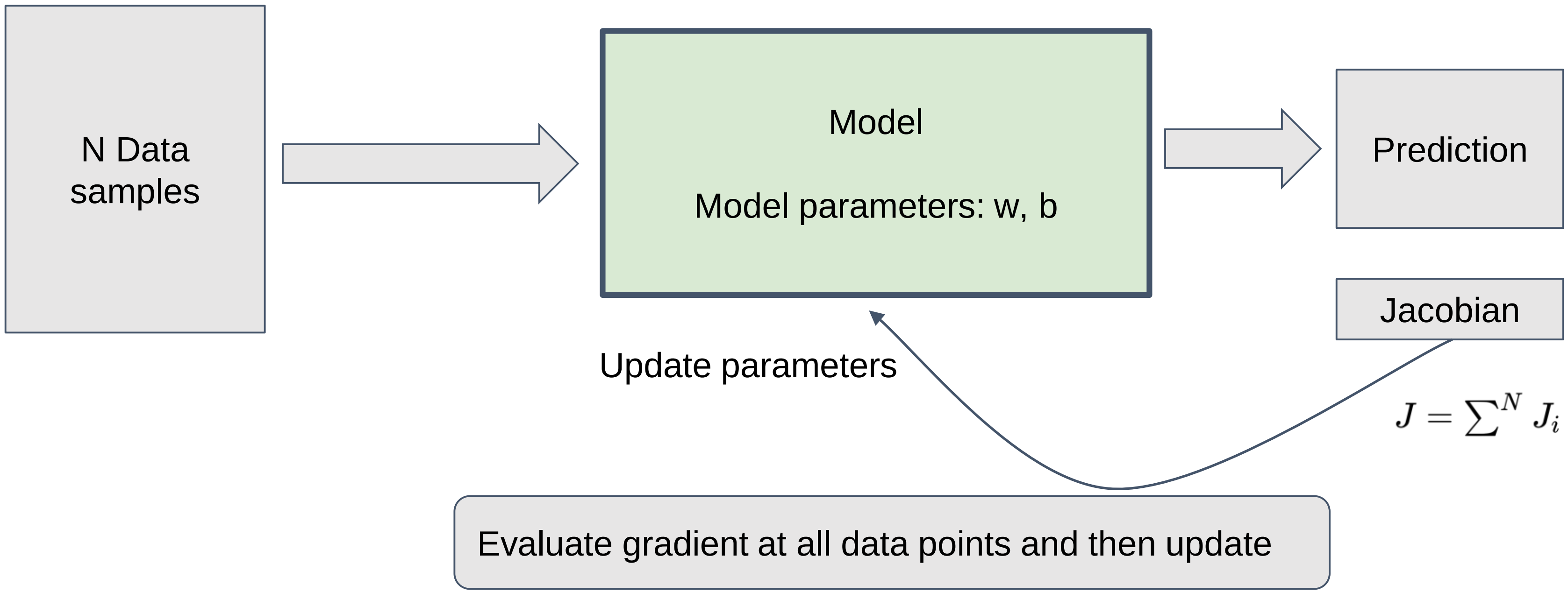
This means that we know how a change in the total input x to an output unit will affect the error. But this total input is just a linear function of the states of the lower level units and it is also a linear function of the weights on the connections, so it is easy to compute how the error will be affected by changing these states and weights. For a weight w_{ji}, from i to j the derivative is

$$\partial E/\partial w_{ji} = \partial E/\partial x_j \cdot \partial x_j/\partial w_{ji}$$
$$= \partial E/\partial x_j \cdot y_i \qquad (6)$$

Source: becominghuman.ai
Rumelhar et al. Nature 198

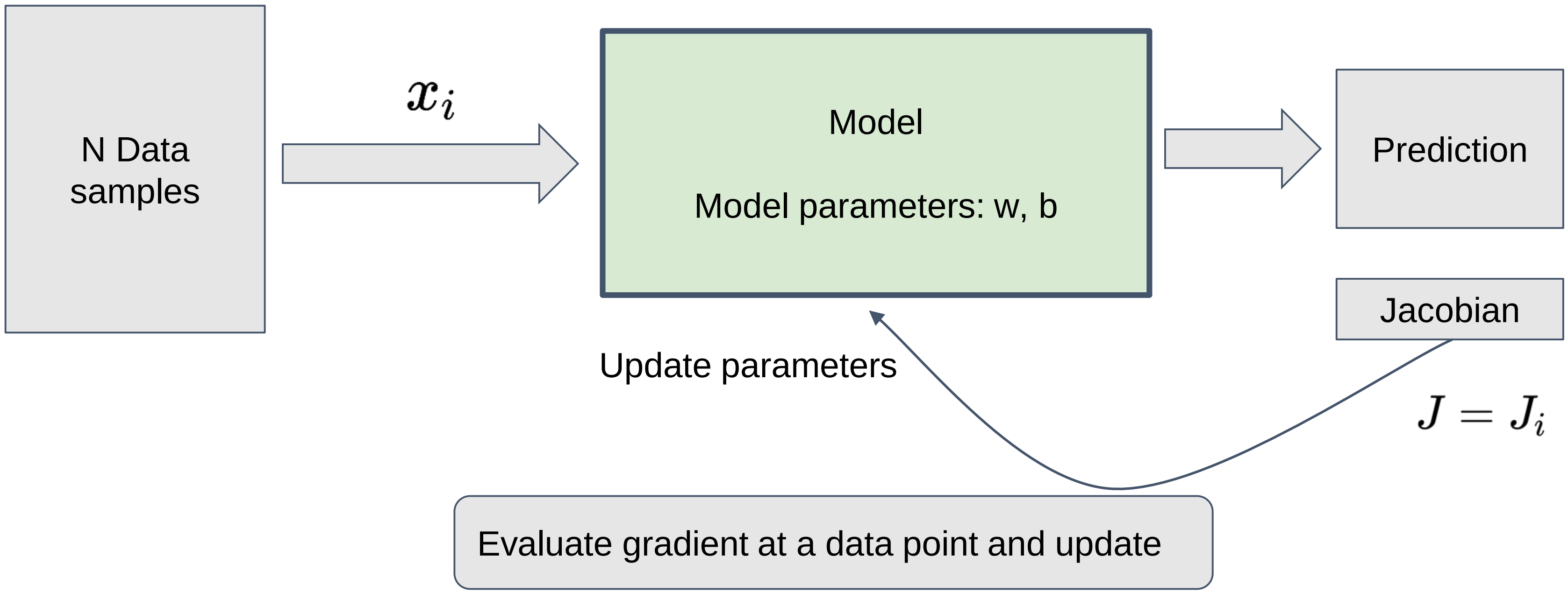
Training in practice

Gradient descent



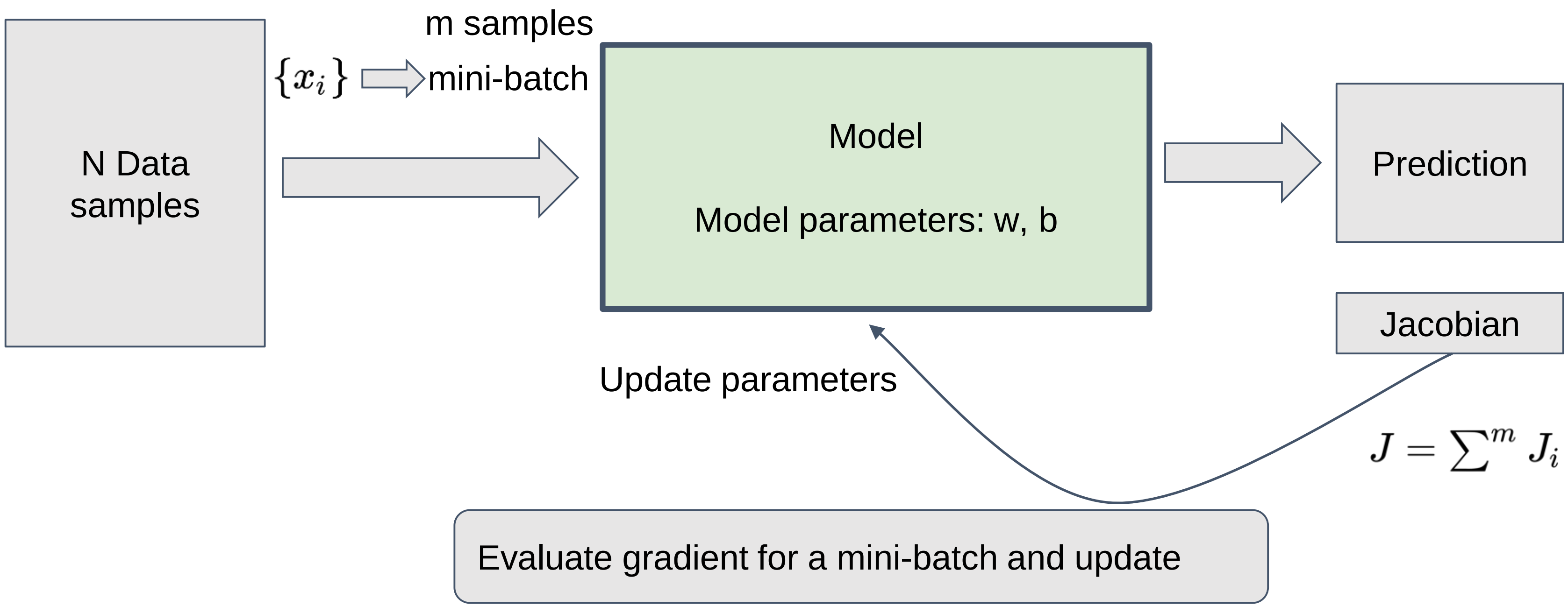
Training in practice

Stochastic Gradient descent



Training in practice

Mini-batch Gradient descent



Mini-batch and Epoch

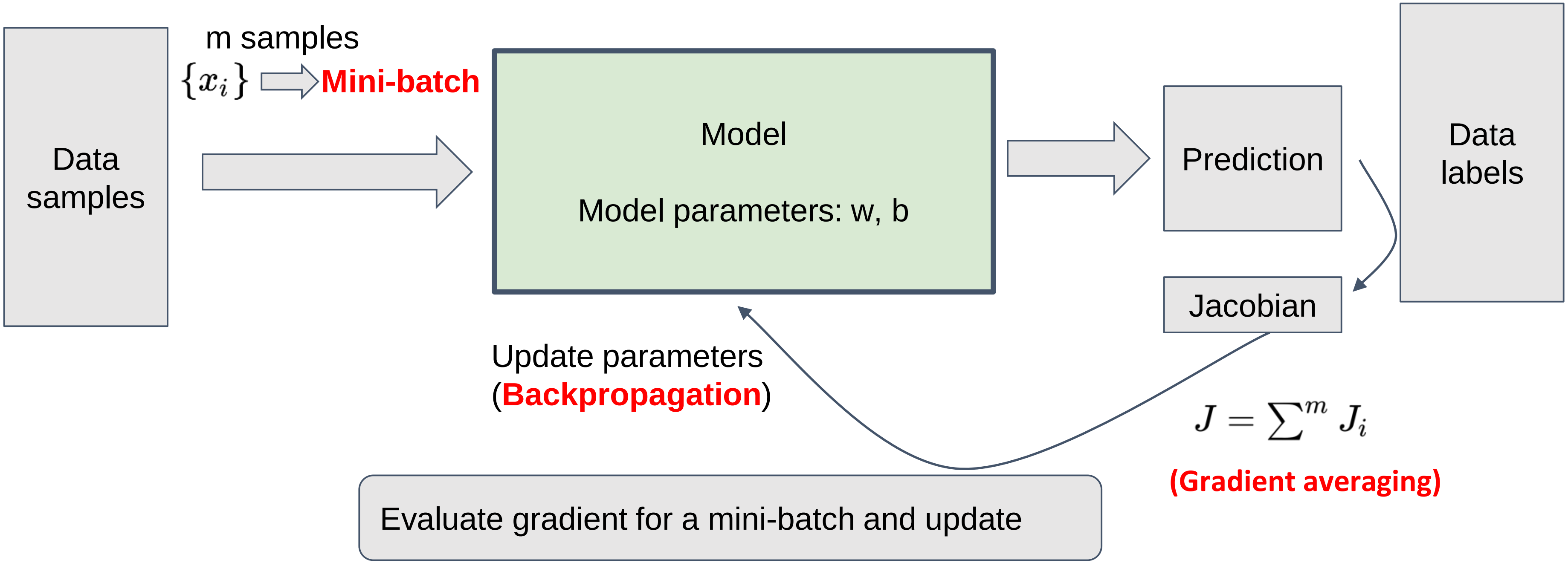
- **Mini-batch:** examples whose gradients are averaged before backpropagation
  - For more stable gradients

- **Epoch:** an ENTIRE dataset is passed forward and backward once
  - A complete training goes through several epoch

Source: https://createmomo.github.io/

Training in practice -- Overview

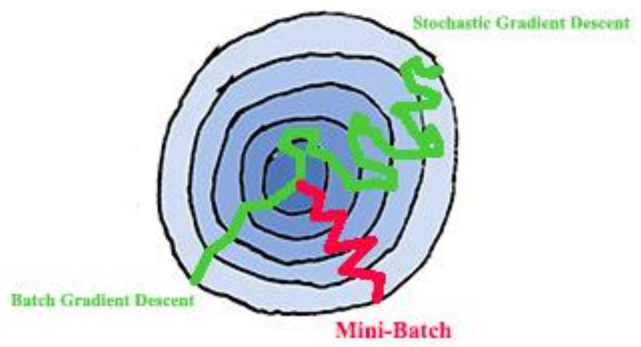
- For multiple epochs,....



Problems in Training?

How good is SGD really?

- In the region of gentle slopes, gradients are very small.
- Near the minima loss are slightly ellipsoidal, where the first order Taylor approximations are almost orthogonal to the true direction of convergence.

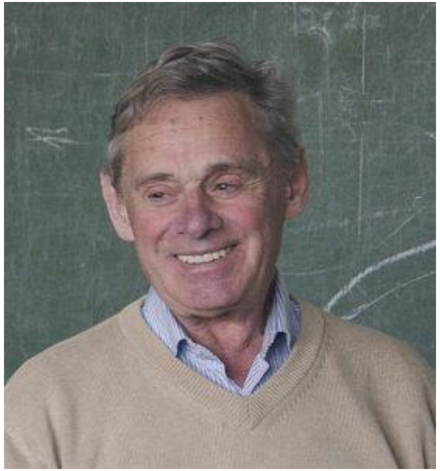


Thought: *Second order Taylor approximation may help.*

But using Newton (or Gauss-Newton) is too expensive.

Back in USSR

Polyak momentum (1964)



Boris Polyak

Nesterov momentum (1983)

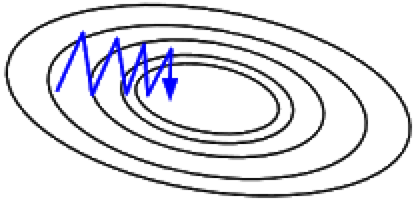
Yurii Nesterov

Polyak Momentum

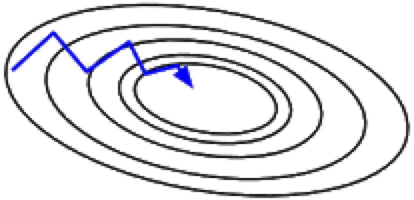
Damping factor **Learning rate**

$$\text{Momentum: } \mathbf{m}_t = \eta \mathbf{m}_{t-1} + \lambda \nabla L(\mathbf{x}_t)$$

$$\text{Update: } \mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{m}_t$$



Stochastic Gradient Descent (SGD) without Momentum



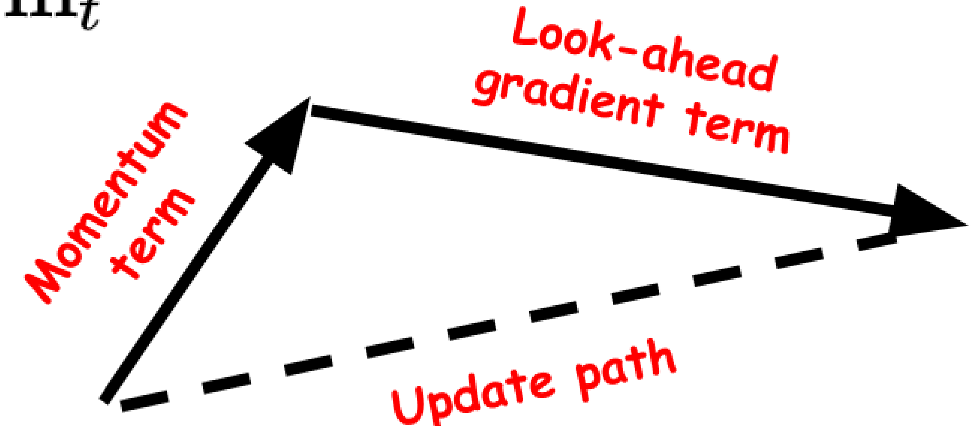
Stochastic Gradient Descent (SGD) with Momentum

Nesterov Momentum

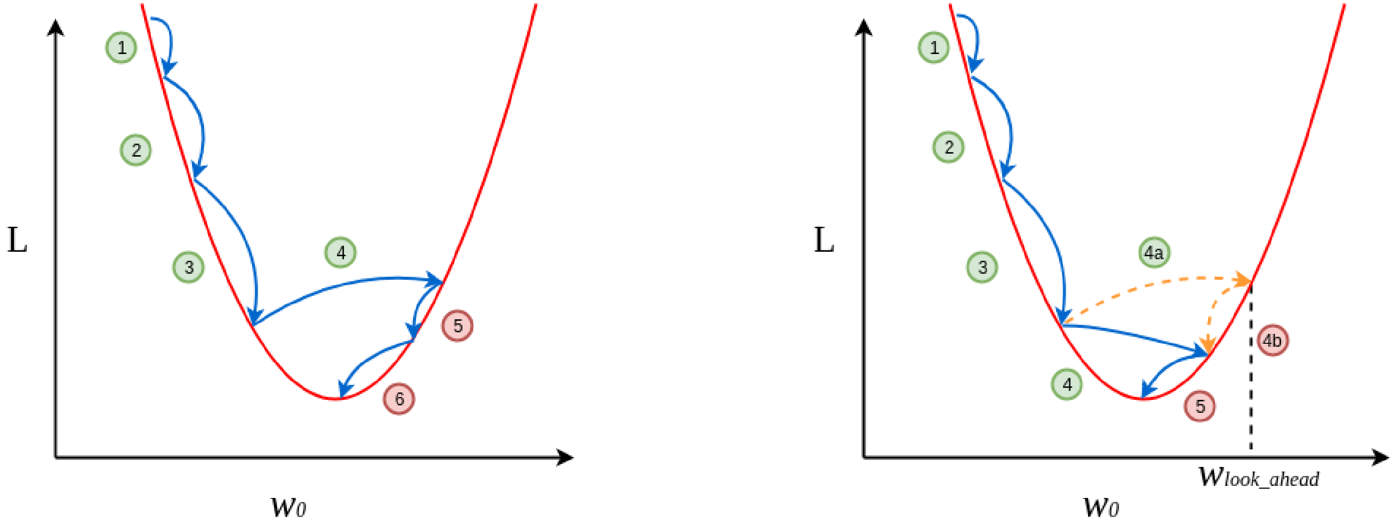
Look ahead: $\mathbf{x}_{temp} = \mathbf{x}_t - \eta \mathbf{m}_{t-1}$

Momentum: $\mathbf{m}_t = \eta \mathbf{m}_{t-1} + \lambda \nabla L(\mathbf{x}_{temp})$

Update: $\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{m}_t$



Polyak vs. Nesterov Momentum



(a) Momentum-Based Gradient Descent

(b) Nesterov Accelerated Gradient Descent

Playing with the learning rate…….

- **AdaGrad:** larger updates for infrequent and smaller updates for frequent parameters

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \frac{\lambda}{\mathrm{RMS}[\nabla L(\mathbf{x}_t)]_t} \nabla L(\mathbf{x}_t)$$

Where $\mathrm{RMS}[\nabla L(\mathbf{x}_t)]_t$ is the sum of the squares of the past gradients.

```
torch.optim.Adagrad(params, lr=0.01, eps=1e-10)
```

- **Major issue:** **rapid decay!**

Fixing AdaGrad......

- **AdaDelta:** monotonically decrease using **running average** within some fixed window.

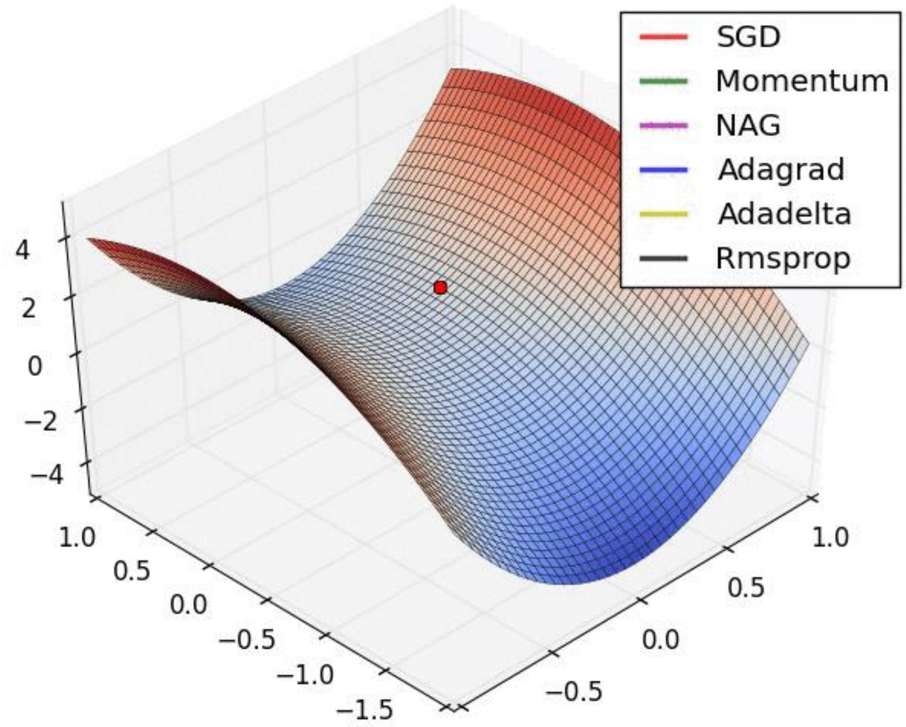
$$\mathbf{x}_t = \mathbf{x}_{t-1} - \frac{\mathrm{RMS}[\Delta \mathbf{x}]_{t-1}}{\mathrm{RMS}[\nabla L(\mathbf{x}_t)]_t} \nabla L(\mathbf{x}_t)$$

- **RMSProp:** decrease learning rate using a moving average of the squared gradients

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\lambda}{\sqrt{v_t + \epsilon}} \nabla L(\mathbf{x}_t) \quad \text{where,} \quad v_t = (1-\alpha)\nabla L(\mathbf{x}_t)^2 + \alpha v_{t-1}$$

```
torch.optim.RMSprop(params, lr=0.01, alpha=0.99, eps=1e-08)
```

So, finally …..what are the possible choices?



Adam: adaptive moment estimation

- Adam is a combination of RMSprop and SGD with momentum

RMSprop

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\lambda}{\sqrt{v_t + \epsilon}} \nabla L(\mathbf{x}_t)$$

SGD with momentum

Momentum: $\mathbf{m}_t = \eta \mathbf{m}_{t-1} + \lambda \nabla L(\mathbf{x}_t)$

Update: $\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{m}_t$

$$\text{Update: } \mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\lambda \hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

$$\text{where, } \hat{m}_t = \frac{m_t}{1 - \beta_1^t} \text{ and } \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

with,

$$m_t = (1 - \beta_1) \nabla L(\mathbf{x}_t) + \beta_1 m_{t-1}$$

$$v_t = (1 - \beta_2) \nabla L(\mathbf{x}_t)^2 + \beta_2 v_{t-1}$$

Moving averages of gradient and squared gradient

```
torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08)
```

Adam: adaptive moment estimation

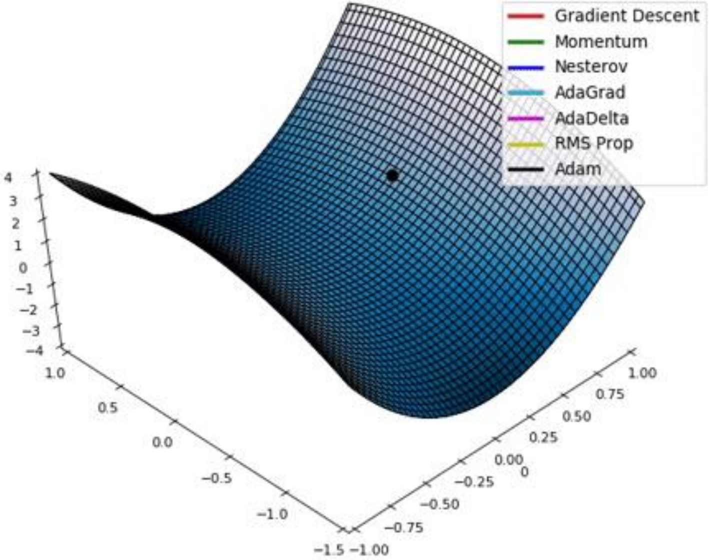
Adam: A Method for Stochastic Optimization

https://arxiv.org › cs ▾

by DP Kingma - 2014 - Cited by 34535 - Related articles

Abstract: We introduce **Adam**, an algorithm for first-order gradient-based **optimization** of stochastic objective functions, based on adaptive estimates of ...

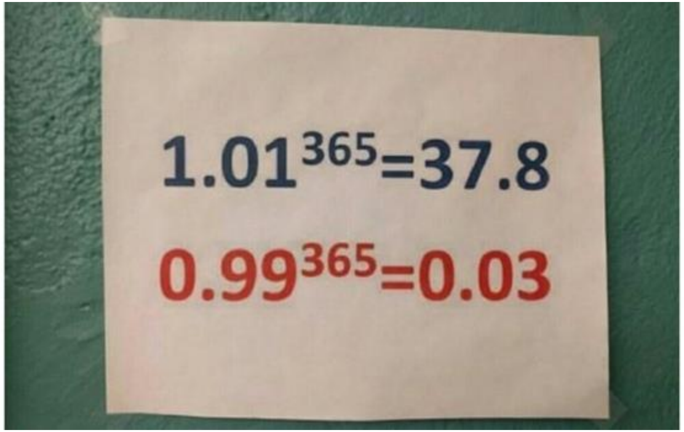
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

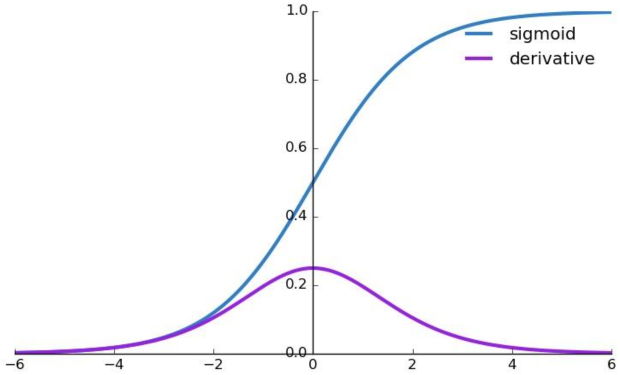


It’s finally done! You understood everything.

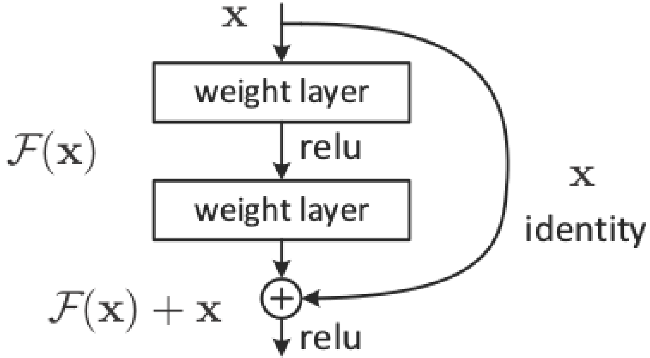

DEEP LEARNING
WHY YOU NO WORK?!

Vanishing/Exploding gradient?



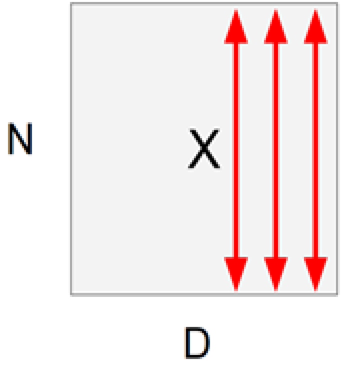


- Treatments
  - Weight initialization
  - Skip connections



Need better gradients…??

- ❑ Batch normalization



1. compute the empirical mean and variance independently for each dimension.

$$\hat{x}_k = \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\mathrm{Var}(x_k)}}$$

```
nn.BatchNorm1d(input_size)
```

Sources: isaacchanghau.github.io
Prof. Elisa Sayrol Clols

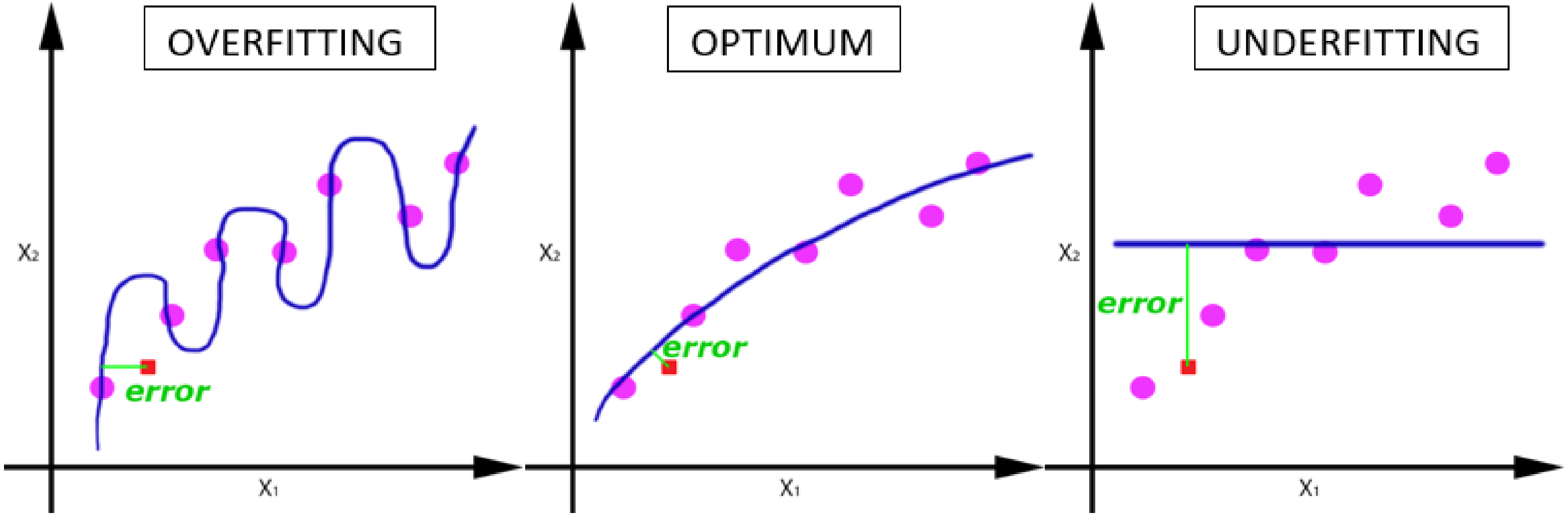
Deep Learning Fundamentals -- Checklist*

- Settings...
  - Optimizers ✓
  - Learning rate ✓
  - Batch size ✓
  - ~~Hyperparameters~~
  - Weight decay
  - Dropouts

- Design...
  - Batch Normalization ✓
  - ~~Layers/Channels~~
  - Activations
  - Data Augmentation
  - Loss functions
  - Softmax

*non exhaustive!

What is Overfitting?

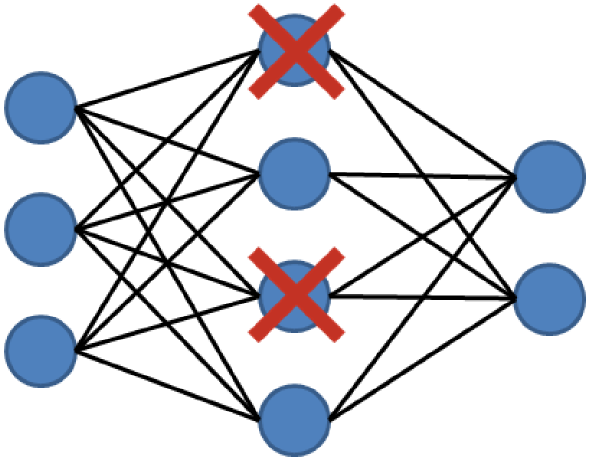


- Treatments:
  - Dropouts
  - Weight decay
  - Data augmentation

Source: https://towardsdatascience.com/

Overfitting?

- Dropouts



```
nn.Dropout(p=0.4)
```

Sources: isaacchanghau.github.io
Prof. Elisa Sayrol Clols

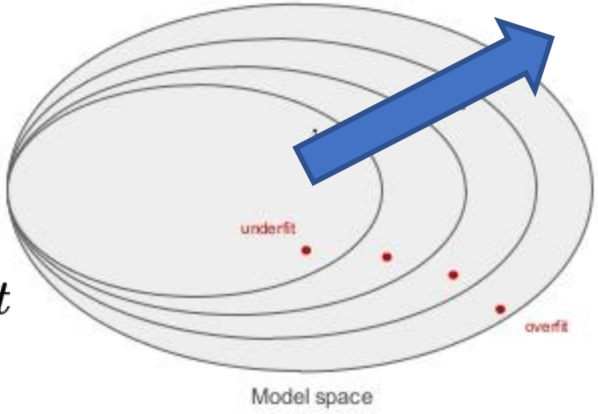
Overfitting??

- Weight decay:
  - Add penalty for large weights

$$L(\mathbf{x}_t) = L_{data}(\mathbf{x}_t) + \frac{\beta}{2}||\mathbf{x}_t||^2$$

$$\text{Update: } \mathbf{x}_{t+1} = \mathbf{x}_t - \lambda \nabla L_{data}(\mathbf{x}_t) - \beta \mathbf{x}_t$$

β

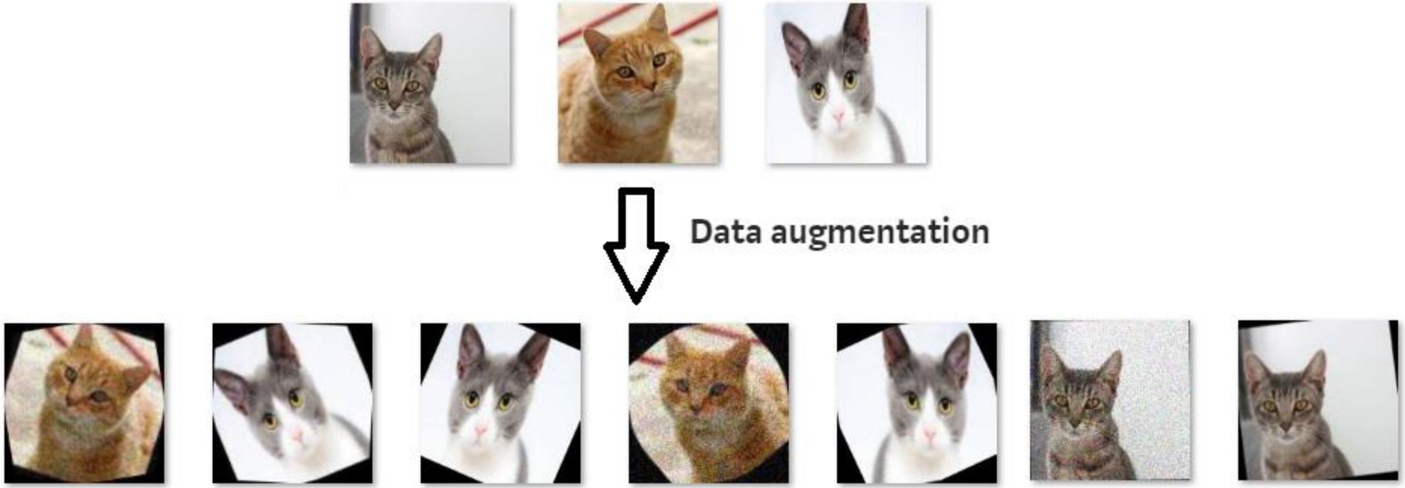



```
torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.01)
```

Sources: isaacchanghau.github.io
Prof. Elisa Sayrol Clols

Overfitting??

- Data Augmentation: make the problem more difficult

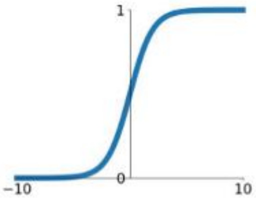



```
transforms.RandomHorizontalFlip();  transforms.CenterCrop(); ts.transforms.Rotate(); …….
```

Activation functions: ReLu, Sigmoid, and more…

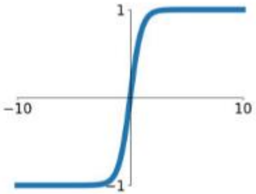
Sigmoid

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



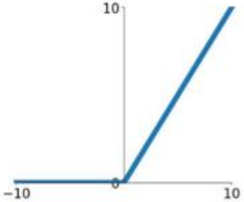
tanh

$\tanh(x)$



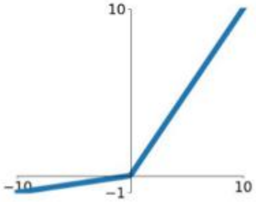
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

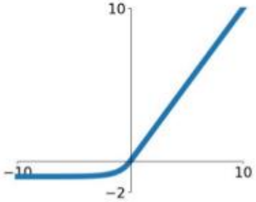


Maxout

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

ELU

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




```
nn.ReLU(); nn.Sigmoid(); nn.Tanh()…….
```

What loss to minimize?

- L1-norm, L2-norm,
- Robust losses.....

$$Least - square := ||\hat{y} - y||^2$$

Prediction **ground-truth**

Name	$\rho(x)$	$\psi(x)$	$\omega(x)$
Least-squares	$x^2/2$	x	1
L_1-norm	$\|x\|$	$sgn(x)$	$1/\|x\|$
L_p-norm	$\|x\|^p/p$	$sgn(x)\|x\|^{p-1}$	$\|x\|^{p-2}$
Fair	$\xi^2(\frac{\|x\|}{\xi} - log(1 + \frac{\|x\|}{\xi}))$	$\frac{x}{1+\|x\|/\xi}$	$\frac{1}{1+\|x\|/\xi}$
Cauchy	$\frac{\xi^2}{2} log(1 + x^2/\xi^2)$	$\frac{x}{(1+x^2/\xi^2)}$	$\frac{1}{(1+x^2/\xi^2)}$
Huber $\begin{cases} \|x\| \leq \xi \\ \|x\| > \xi \end{cases}$	$\begin{cases} x^2/2 \\ \xi(\|x\| - \xi/2) \end{cases}$	$\begin{cases} x \\ \xi sgn(x) \end{cases}$	$\begin{cases} 1 \\ \xi/\|x\| \end{cases}$
Tukey $\begin{cases} \|x\| \leq \xi \\ \|x\| > \xi \end{cases}$	$\begin{cases} \frac{x^6}{6} - \frac{\xi^2 x^4}{2} + \frac{\xi^4 x^2}{2} \\ \frac{\xi^6}{6} \end{cases}$	$\begin{cases} x\left(\xi^2 - x^2\right)^2 \\ 0 \end{cases}$	$\begin{cases} \left(\xi^2 - x^2\right)^2 \\ 0 \end{cases}$

- Cross Entropy → logistic outputs (e.g. classification)
  - In binary classification, the cross-entropy is:

$$-(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

  - If the number of classes M>2 (i.e. multiclass classification), it turns out to be:

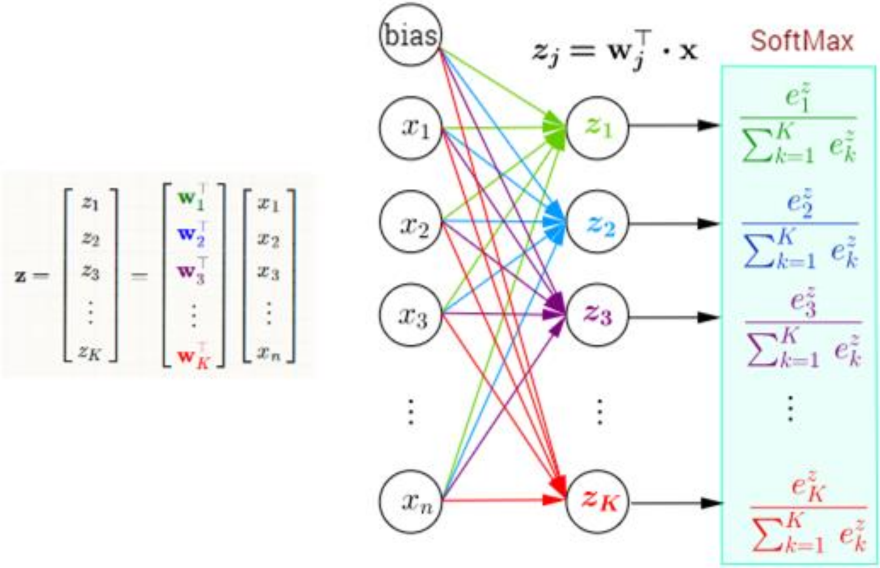
```
loss = nn.CrossEntropyLoss()
```

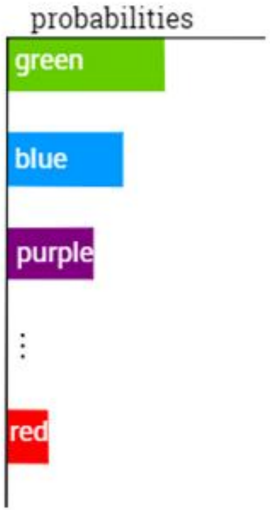
$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Multi-class classificaiton

- Softmax

Multiple outputs of probabilities

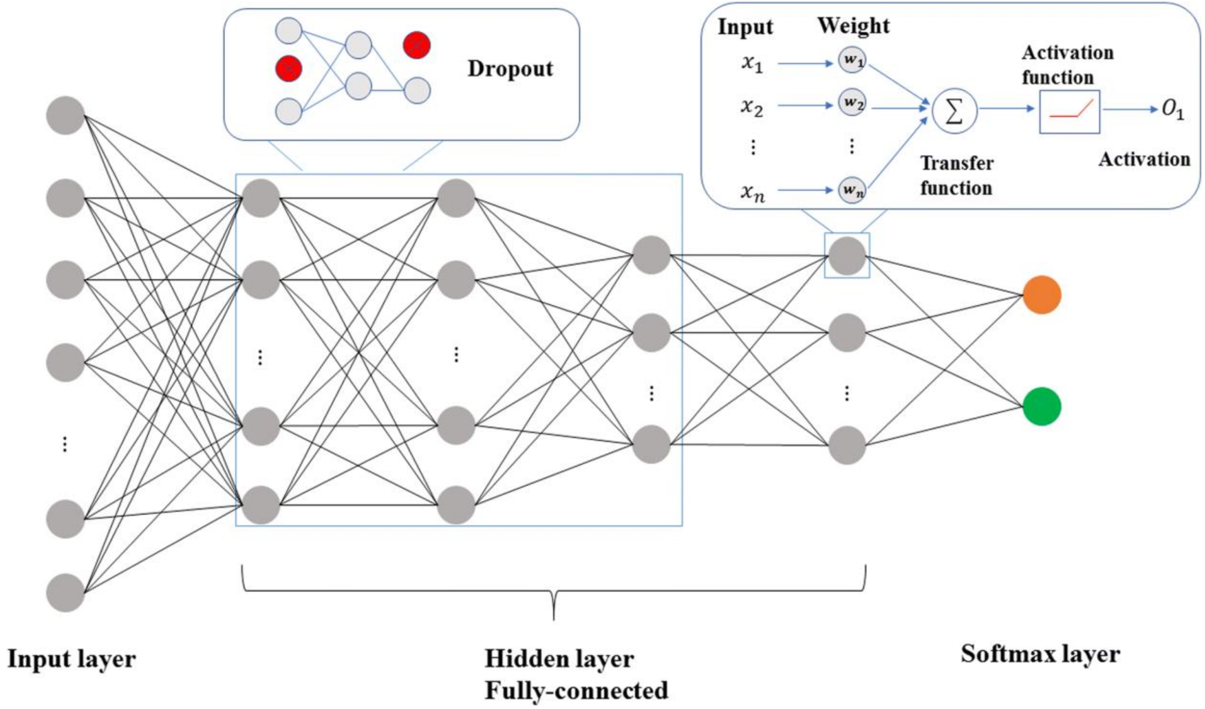





```
nn.functional.softmax(input)
```

Sources: isaacchanghau.github.io
Prof. Elisa Sayrol Clols

A General Overview…..



Lab session...

- Install *pytorch*
- Plot the loss function and accuracy evolution with change in
  - Learning rate
  - Batch size
  - Optimizers

Using .... github code: yunjey/pytorch-tutorial

Link: https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/01-basics/logistic_regression/main.py

- Feel free to try other "pytorch-tutorial" code in the same repository!