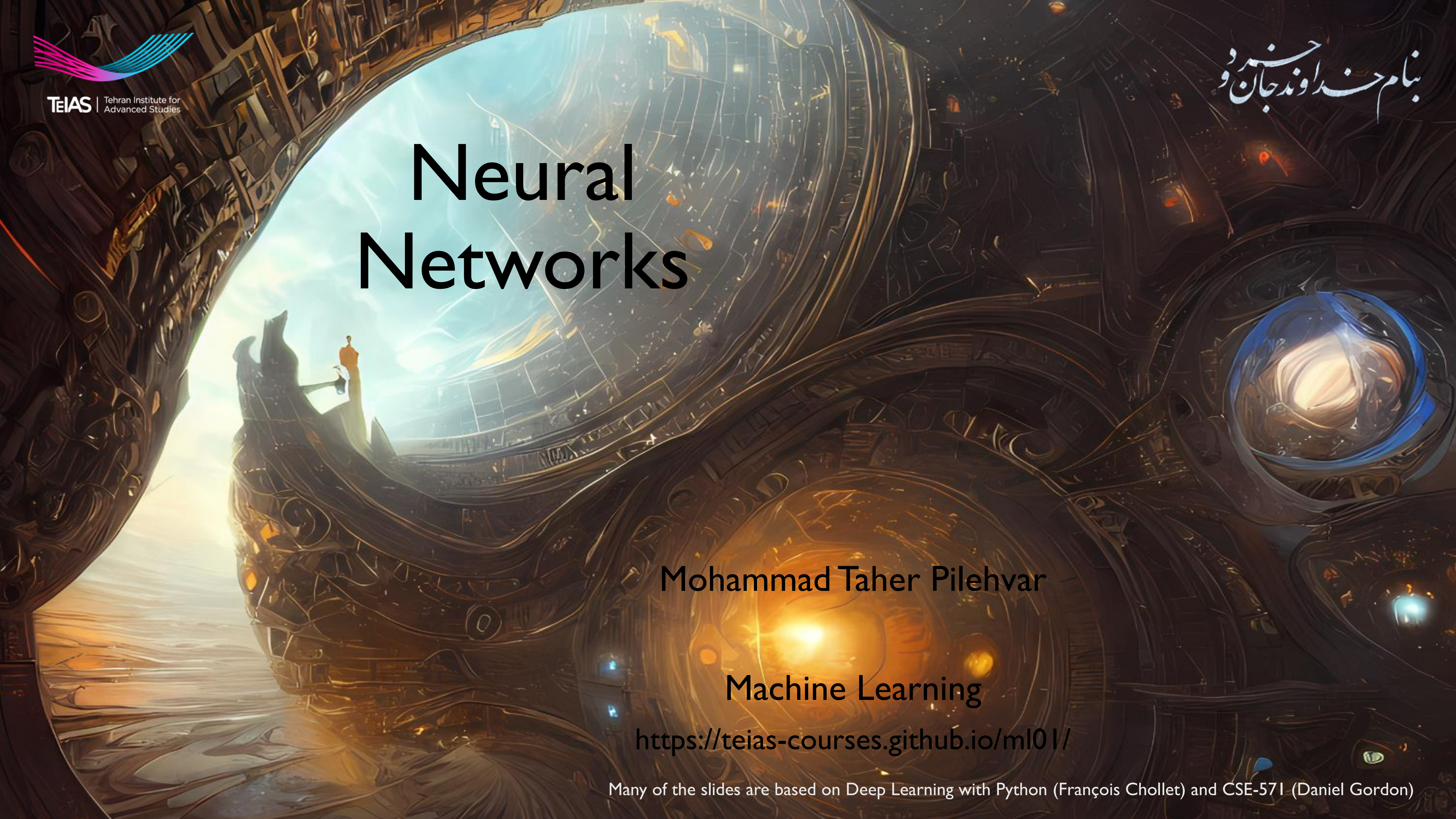


TeIAS | Tehran Institute for Advanced Studies

بنام خداوند جان و خرد

Neural Networks

Mohammad Taher Pilehvar

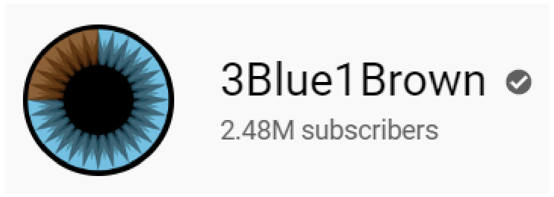
Machine Learning

https://teias-courses.github.io/ml01/

Many of the slides are based on Deep Learning with Python (François Chollet) and CSE-571 (Daniel Gordon)

A quick overview of neural networks

Check “Nerual Network” videos on



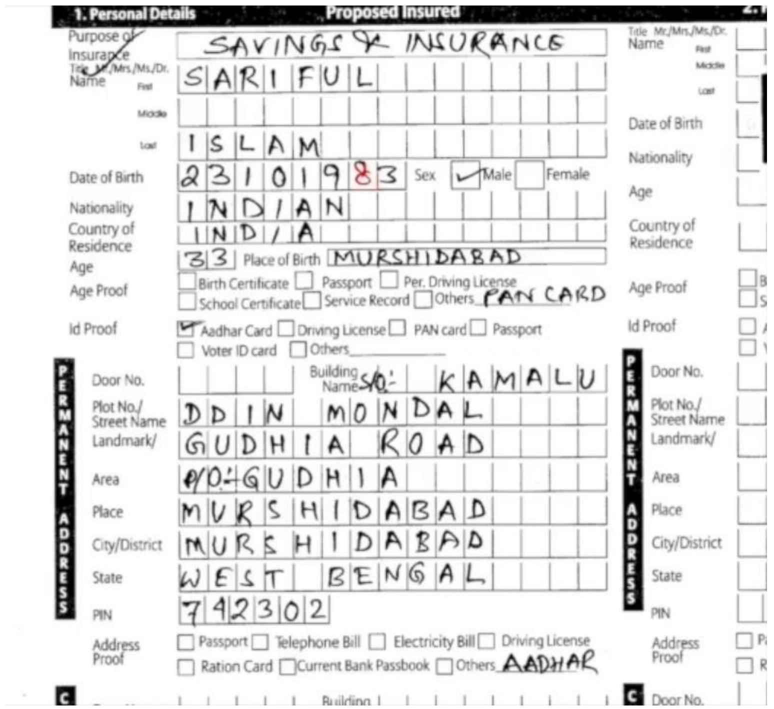
3Blue1Brown

2.48M subscribers

Our first network: handwritten recognition

- Digit classification
  - A 28x28 image into 10 categories (0 to 9)

- MNIST
  - An old classic dataset (1980s)
  - 60K training examples + 10K test



1. Personal Details **Proposed Insured**

Purpose of Insurance	SAVINGS & INSURANCE
Title Mr./Mrs./Ms./Dr. Name First	SARIFUL
Middle	
Last	ISLAM
Date of Birth	23101983 Sex [✓ Male] Female
Nationality	INDIAN
Country of Residence	INDIA
Age	33 Place of Birth MURSHIDABAD
Age Proof	Birth Certificate Passport Per. Driving License School Certificate Service Record Others PAN CARD
Id Proof	[✓ Aadhar Card] Driving License PAN card Passport Voter ID card Others

PERMANENT ADDRESS

Door No.	Building Name S/O:- KAMALU
Plot No./ Street Name	DDIN MONDAL
Landmark/	GUDHIA ROAD
Area	P/O:- GUDHIA
Place	MURSHIDABAD
City/District	MURSHIDABAD
State	WEST BENGAL
PIN	742302
Address Proof	Passport Telephone Bill Electricity Bill Driving License Ration Card Current Bank Passbook Others AADHAR

Title Mr./Mrs./Ms./Dr. Name First Middle Last

Date of Birth

Nationality

Age

Country of Residence

Age Proof

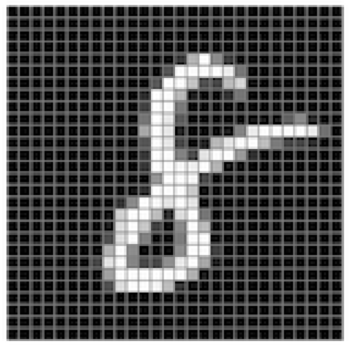
Id Proof

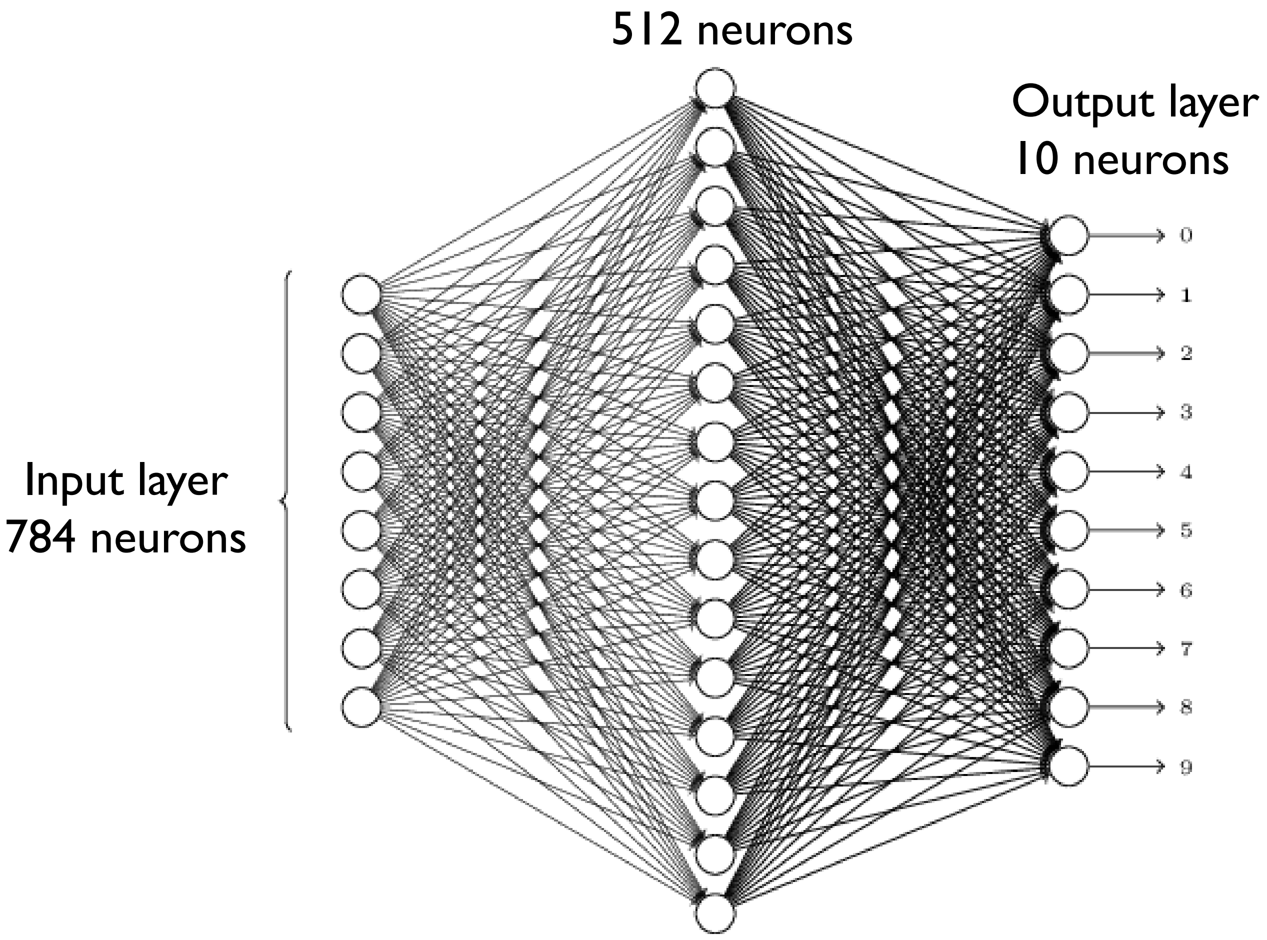
PERMANENT ADDRESS Door No. Plot No./ Street Name Landmark/ Area Place City/District State PIN

Address Proof

C Building Door No.

Dense: MLP




512 neurons
Output layer
10 neurons
Input layer
784 neurons
0
1
2
3
4
5
6
7
8
9

Crushed Granite
Porcelain Sand
Activated Carbon
Quartz Sand
Filter Paper
Filter Paper
Filter Paper

First network implementation

But we need more:

- A *loss function* – to measure performance on the training data, to steer in the right direction
- An *optimizer* – to update the network
- A metric – how is the performance measured?

First network implementation in Keras

Some data preprocessing

- Reshape into the shape the network expects, while scaling to [0, 1]

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

- Categorically encode the labels

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

First network implementation in Keras

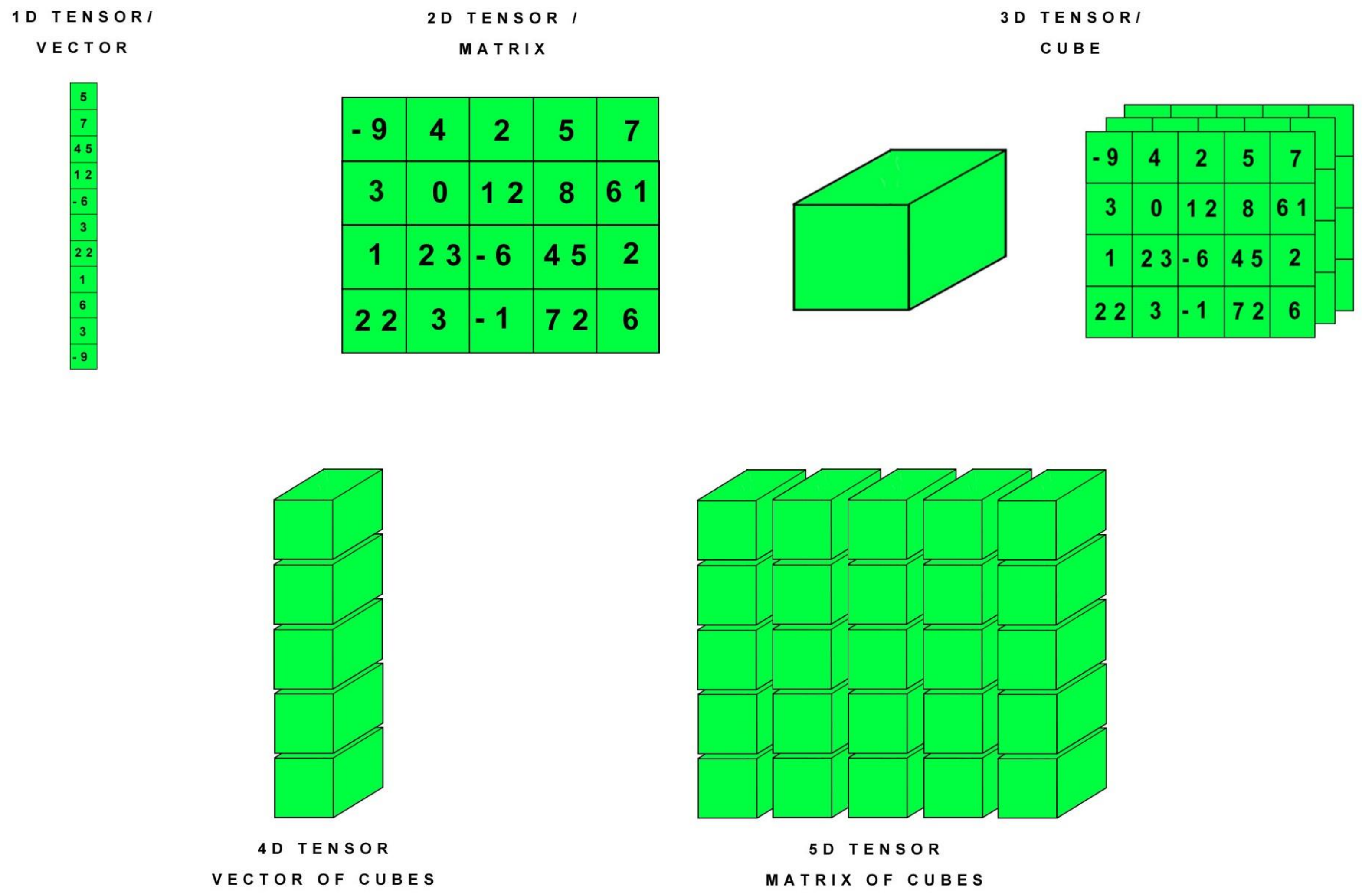
We are ready to train!

```
>>> network.fit(train_images, train_labels, epochs=5, batch_size=128)

Epoch 1/5
60000/60000 [==============================] - 9s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [========================>.....] - ETA: 1s - loss: 0.1035 - acc:
0.9692
```

And we can test the model:

```
>>> test_loss, test_acc = network.evaluate(test_images, test_labels)
>>> print('test_acc:', test_acc)
test_acc: 0.9785
```

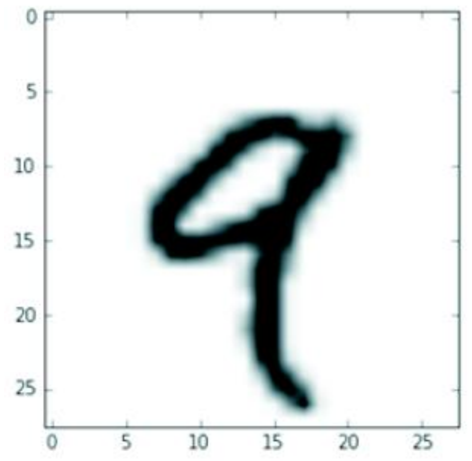
What are tensors?



From hackernoon.com

In our example

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

>>> print(train_images.ndim)
3

>>> print(train_images.shape)
(60000, 28, 28)

>>> print(train_images.dtype)
uint8

digit = train_images[4]
import matplotlib.pyplot as plt
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```



Data tensors

- Vector data - 2D tensors of shape `(samples, features)`
- Timeseries data or sequence data - 3D tensors of shape `(samples, timesteps, features)`
- Images - 4D tensors of shape `(samples, height, width, channels)` or `(samples, channels, height, width)`
- Video - 5D tensors of shape `(samples, frames, height, width, channels)` or `(samples, frames, channels, height, width)`

Vector data

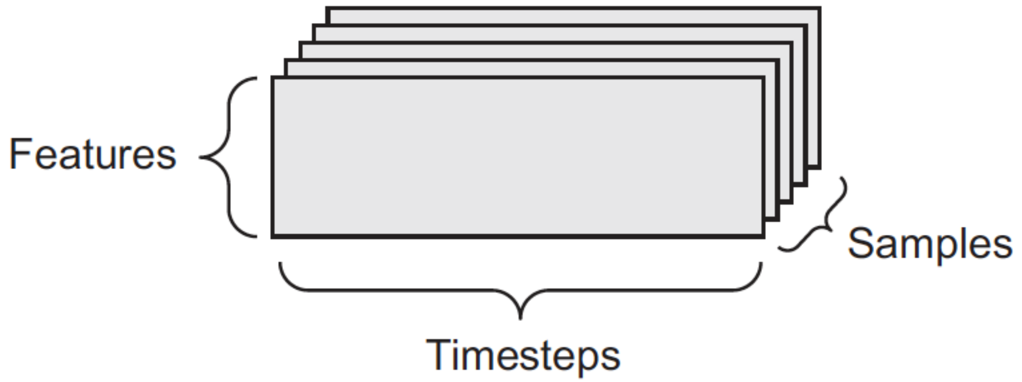
- Each single data point can be encoded as a vector
  - Thus a batch of data will be encoded as a 2D tensor
  - The first axis is the *samples axis* and the second axis is the *features axis*

- Examples:
  - A dataset of 100,000 people, where we consider each person's *age*, *post code*, and *income:* a 2D tensor of shape (100000, 3).

  - A dataset of text documents, where we represent each document by the counts of how many times each word appears in it (out of a dictionary of 20,000 common words): a 2D tensor of (500, 20000).

Timeseries and sequence data

- Whenever time matters in your data (or the notion of sequence order), it makes sense to store it in a 3D tensor with an explicit time axis.



- The time axis is always the second axis (axis of index 1), by convention.

Image data

Images typically have three dimensions: height, width, and color depth.

- A batch of 128 grayscale images of size 256 X 256 could thus be stored in a tensor of shape `(128, 256, 256, 1)`

- A batch of 128 color images could be stored in a tensor of shape `(128, 256, 256, 3)`

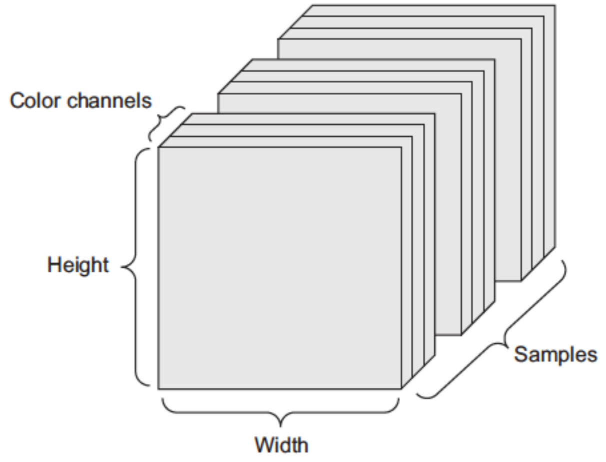


Image data

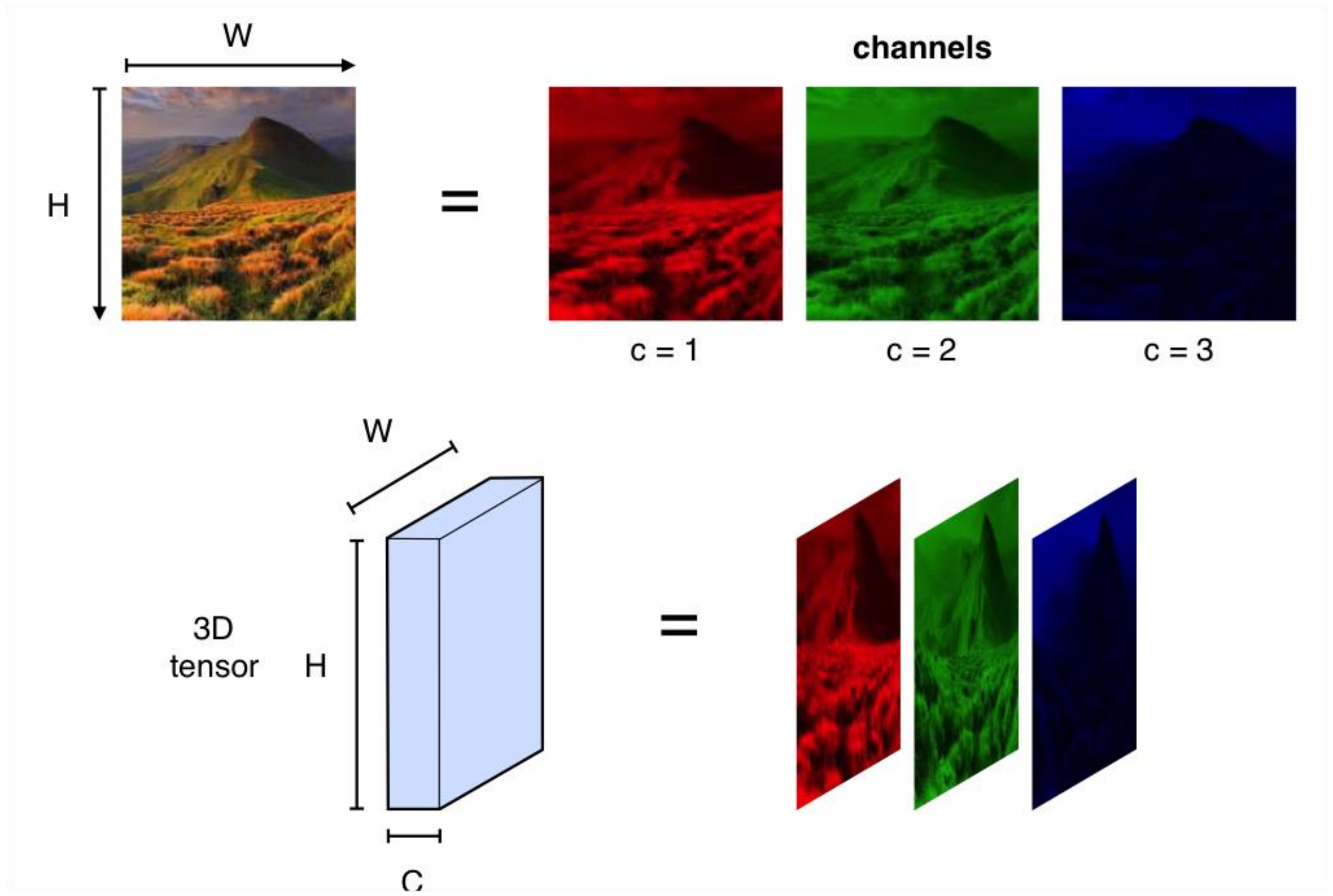


Illustration by Marc Lelarge

Video data

A sequence of frames, each frame being a color image.

```
(samples, frames, height, width, color_depth)
```

- For instance, a batch of 4 YouTube video clips of 60-second, 144 X 256 sampled at 4 frames per second would have 240 frames.

```
(4, 240, 144, 256, 3)
```

- That's a total of 106,168,320 values! If the dtype of the tensor was float32, then each value would be stored in 32 bits, so the tensor would represent 405 MB.

Gardient-based optimization

```
output = relu(dot(W, input) + b)
```

`W` and `b` are *weights* or *trainable parameters* of the layer

- These weights contain the information learned by the network from exposure to training data.

Initially, these are filled with random values (*random initialization*)

But, we gradually adjust these weights based on the feedback signal

Training loop

This gradual adjustment, also called *training*, is basically the learning that machine learning is all about.

1. Draw a batch of training samples **`x`** and corresponding targets **`y`**.
2. Run the network on **`x`** (a step called the *forward pass*) to obtain predictions **`y_pred`**.
3. Compute the *loss* of the network on the batch, a measure of the mismatch between **`y_pred`** and **`y`**.
4. Update all weights of the network in a way that slightly reduces the loss on this batch.

Training loop

But, how to update the weights of the network?

1. Draw a batch of training samples `x` and corresponding targets `y`.
2. Run the network on `x` (a step called the *forward pass*) to obtain predictions `y_pred`.
3. Compute the *loss* of the network on the batch, a measure of the mismatch between `y_pred` and `y`.
4. Update all weights of the network in a way that slightly reduces the loss on this batch.

How to adjust weights?

One solution:

- Freeze all weights in the network except the one scalar coefficient being considered, and try different values for this coefficient.
- Slightly change the value and monitor the *loss*.

- Horribly inefficient!

- A much better solution:
  - All operations in the network are *differentiable*
  - Compute the *gradient* of *loss* with respect to coefficients

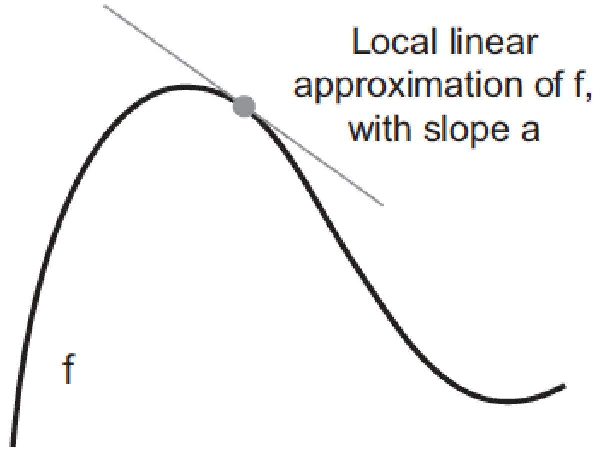
Some basics: derivative

Consider a continuous, smooth function **`f(x) = y`**

- We'd like to see how much **`f(x)`** changes with small changes in **`x`** around point **`p`**

- This can be computed as the slope of **`f(x)`** around point **`p`**

- The slope a is called the *derivative* of **`f`** in **`p`**



Gradient

A *gradient* is the derivative of a tensor operation.

Gradient descent

Gradient points away from the minimum!

df/dx is positive when $\uparrow x = \uparrow f(x)$

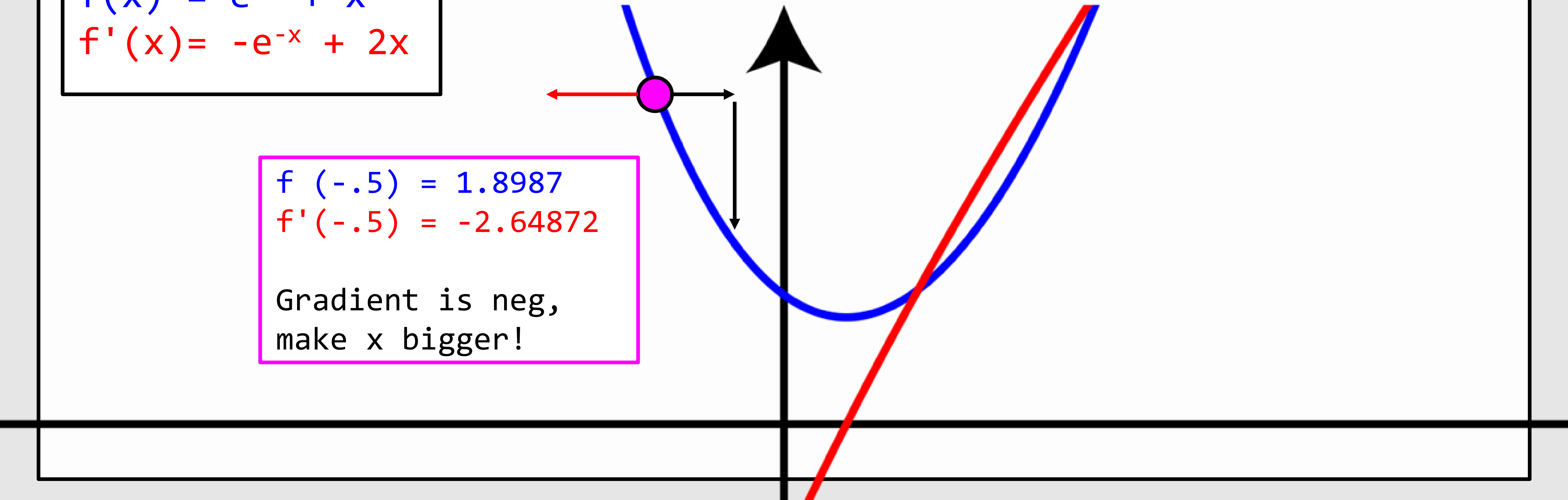
df/dx is negative when $\uparrow x = \downarrow f(x)$

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

```
f(x)  = e^-x + x^2
f'(x)= -e^-x + 2x
```

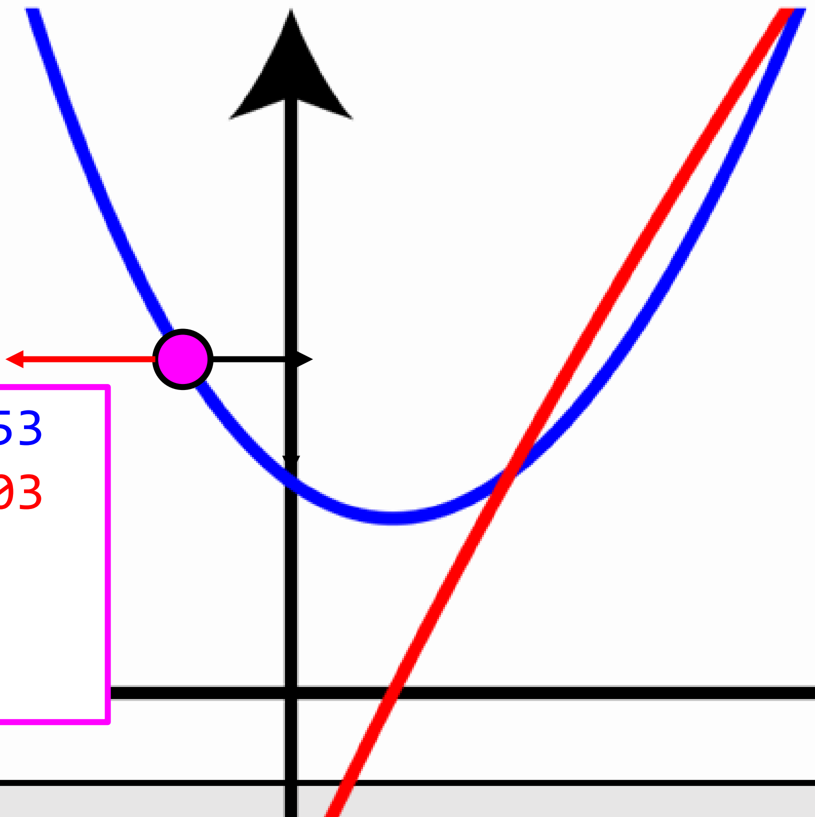


```
f (-.5) = 1.8987
f'(-.5) = -2.64872

Gradient is neg,
make x bigger!
```

Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

```
f(x)  = e^-x + x^2
f'(x)= -e^-x + 2x
```



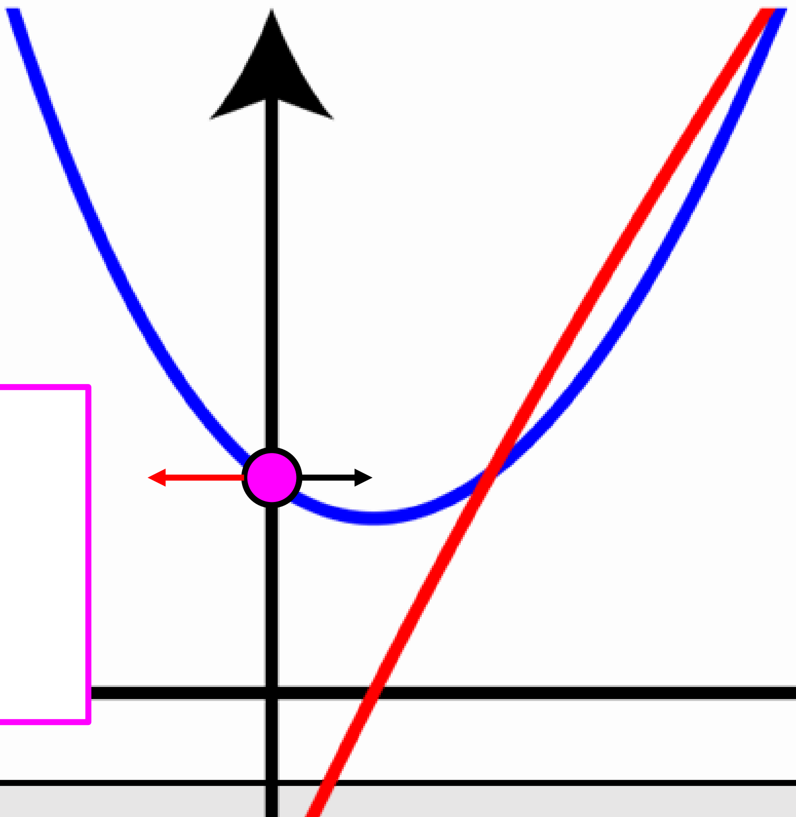
```
f (-.25) = 1.34653
f'(-.25)= -1.78403

Gradient is neg,
make x bigger!
```

Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

```
f(x) = e^{-x} + x^2
f'(x)= -e^{-x} + 2x
```

$f(x) = e^{-x} + x^2$
$f'(x) = -e^{-x} + 2x$

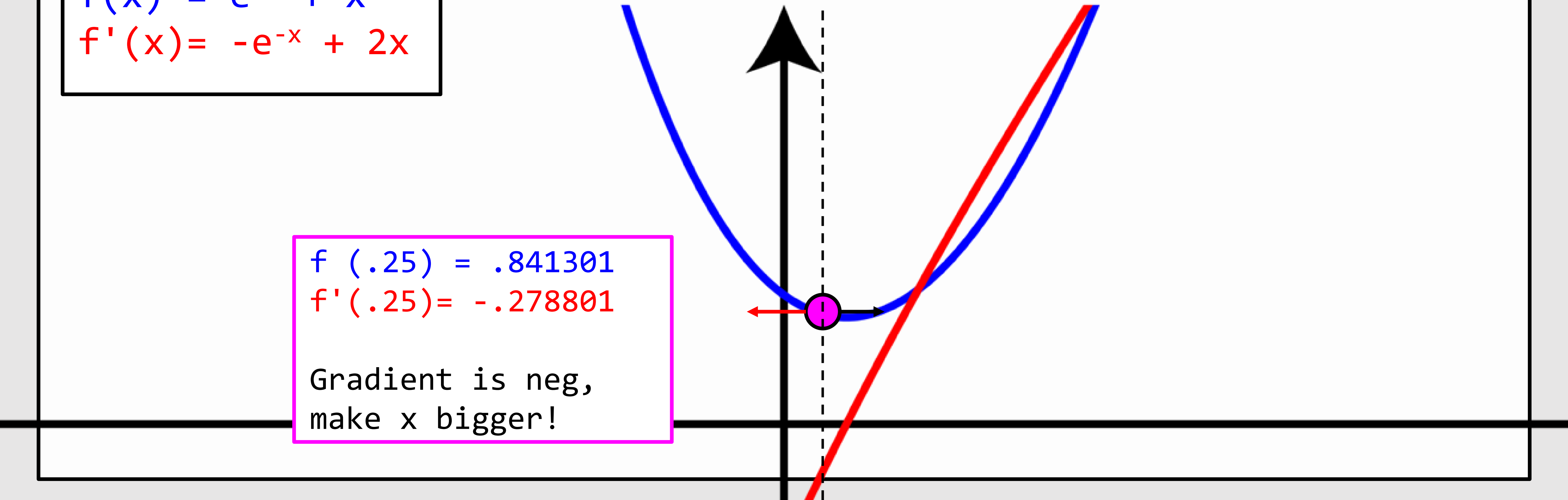


```
f (0) = 1
f'(0)= -1

Gradient is neg,
make x bigger!
```

Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

```
f(x) = e^-x + x^2
f'(x)= -e^-x + 2x
```

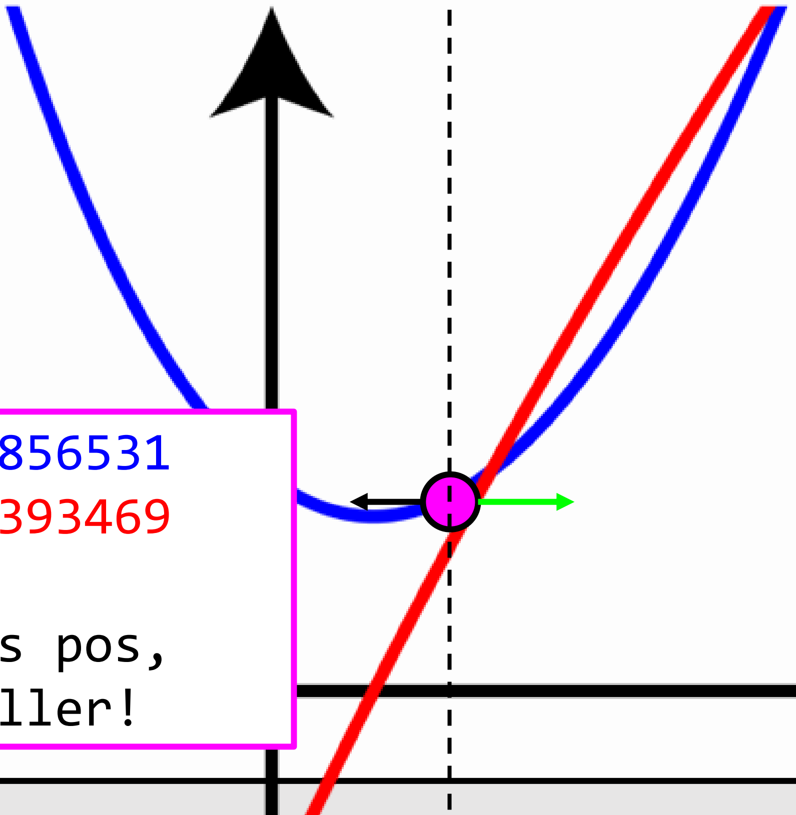


```
f (.25) = .841301
f'(.25)= -.278801

Gradient is neg,
make x bigger!
```

Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

```
f(x) = e^-x + x^2
f'(x)= -e^-x + 2x
```

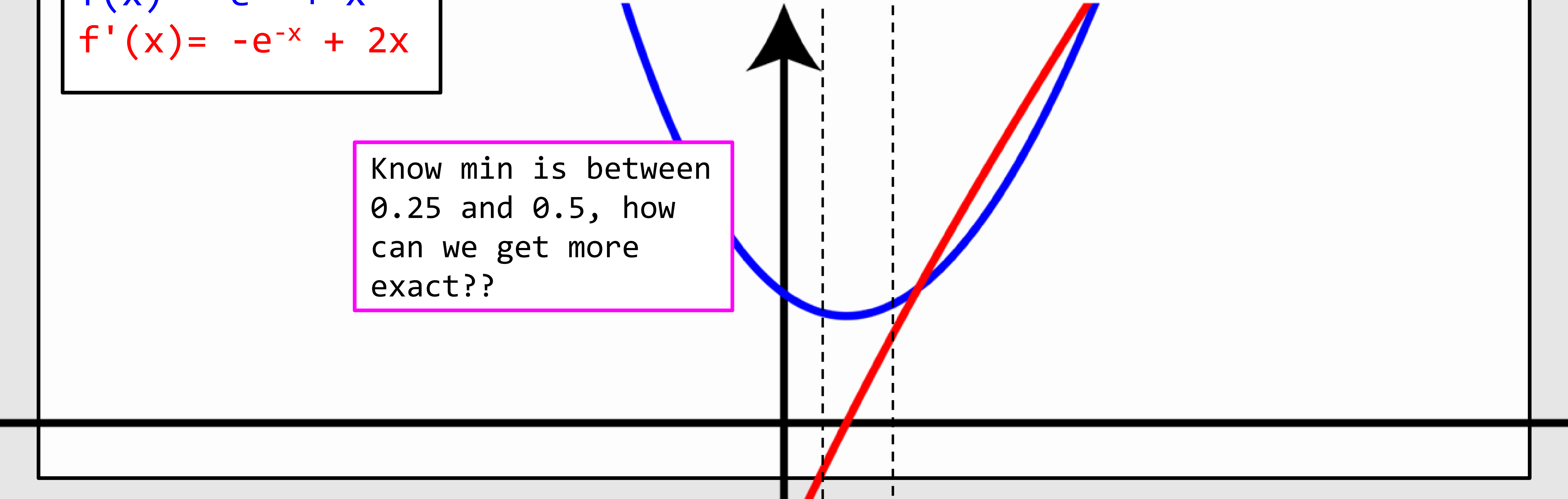


```
f (.5) = .856531
f'(.5) = .393469

Gradient is pos,
make x smaller!
```

Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

$f(x) = e^{-x} + x^2$
$f'(x) = -e^{-x} + 2x$

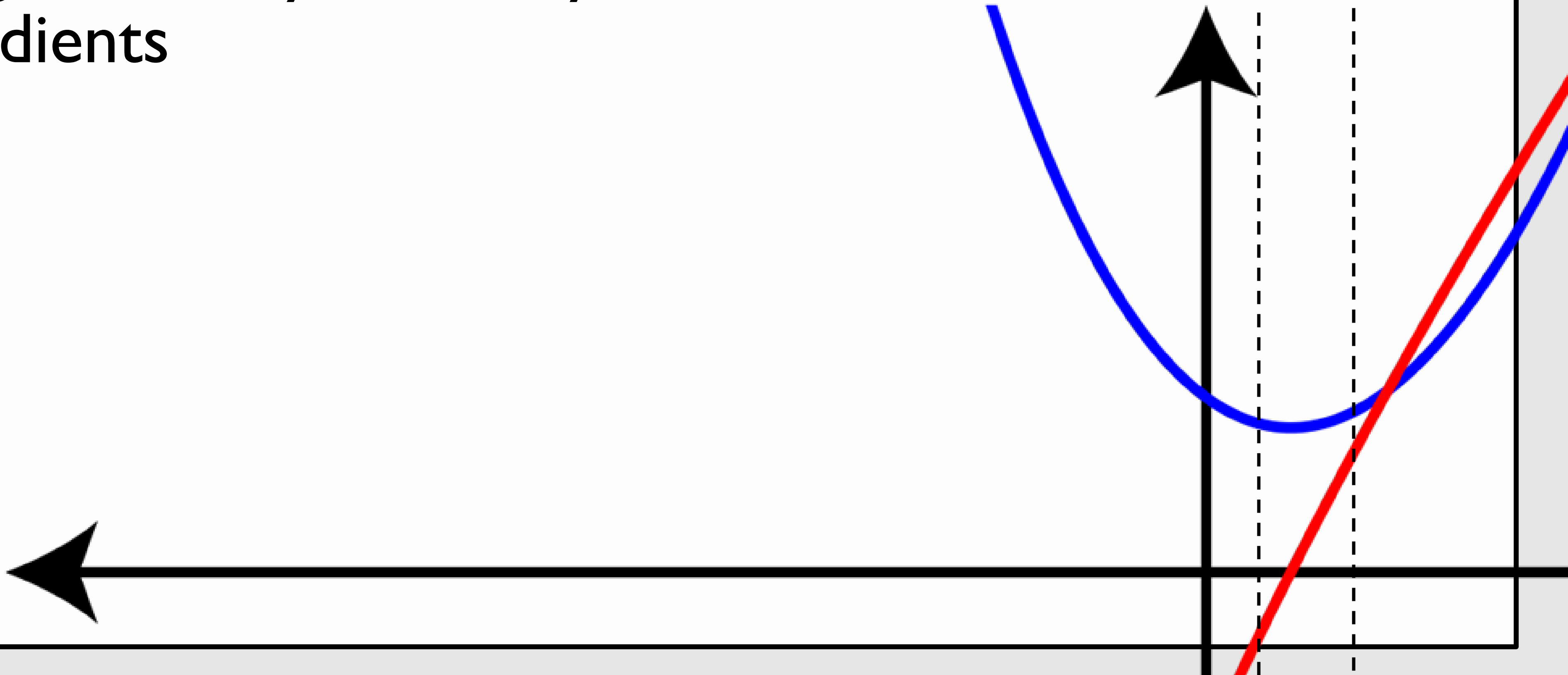


Know min is between 0.25 and 0.5, how can we get more exact??

Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

We were moving around by .25 every time, instead we could move based off the gradients

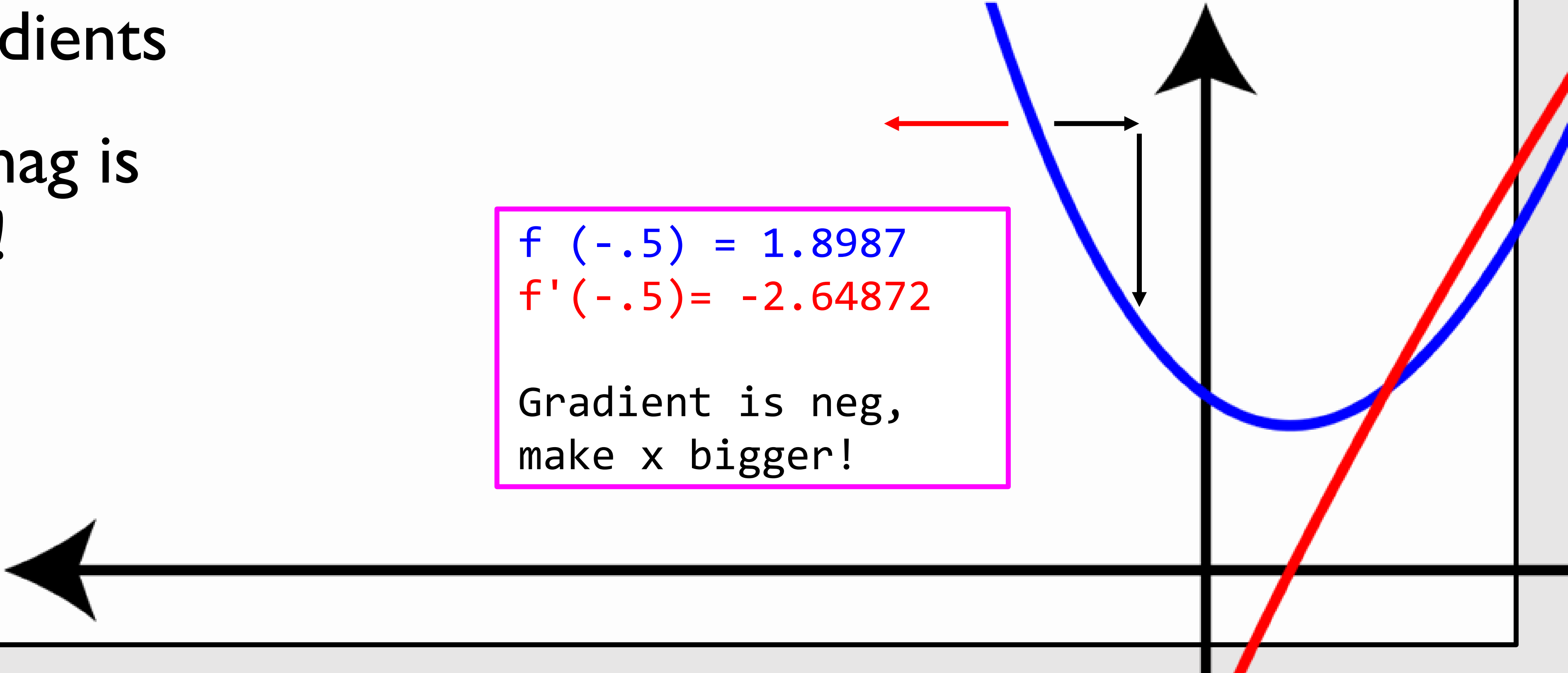


Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

We were moving around by .25 every time, instead we could move based off the gradients

At -.5, gradient mag is large, move a lot!



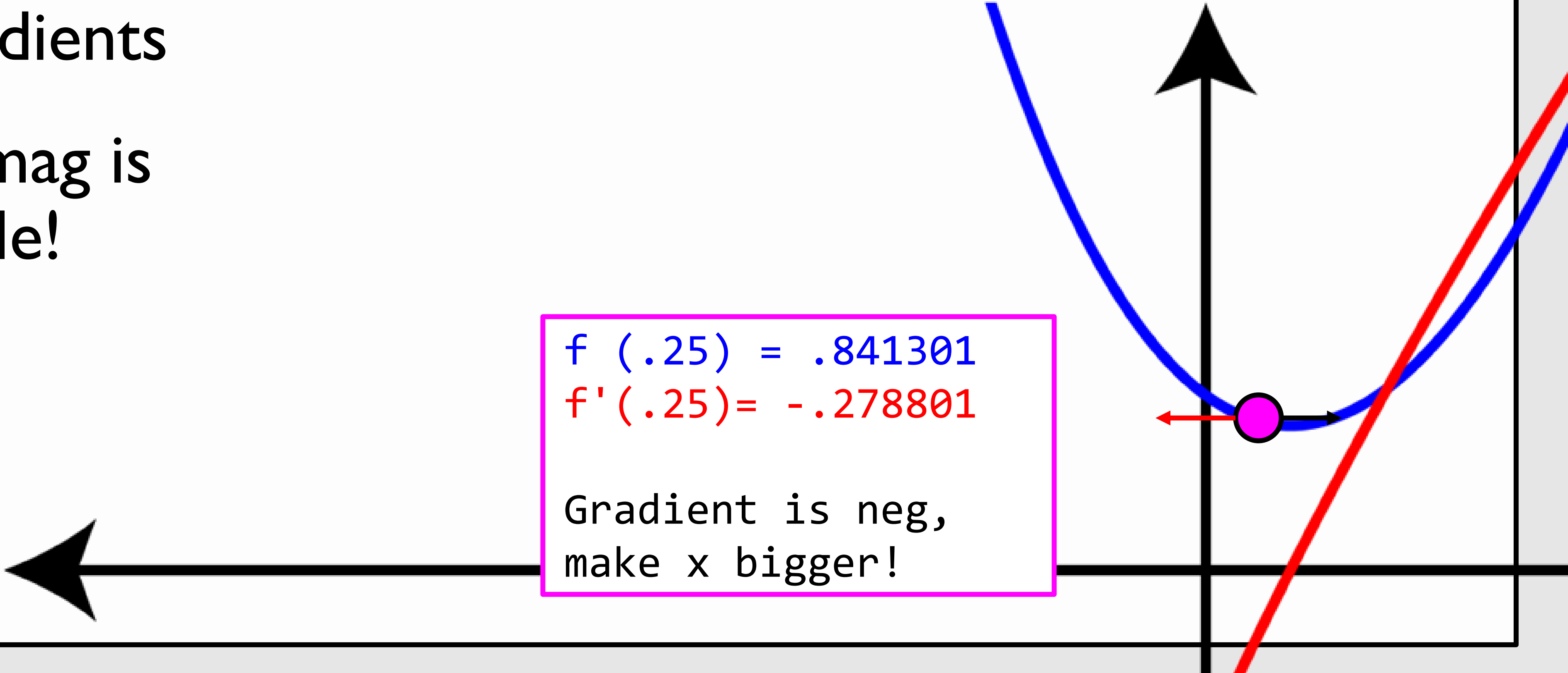
```
f (-.5) = 1.8987
f'(-.5)= -2.64872

Gradient is neg,
make x bigger!
```

Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

We were moving around by .25 every time, instead we could move based off the gradients

At .25, gradient mag is small, move a little!



```
f (.25) = .841301
f'(.25)= -.278801

Gradient is neg,
make x bigger!
```

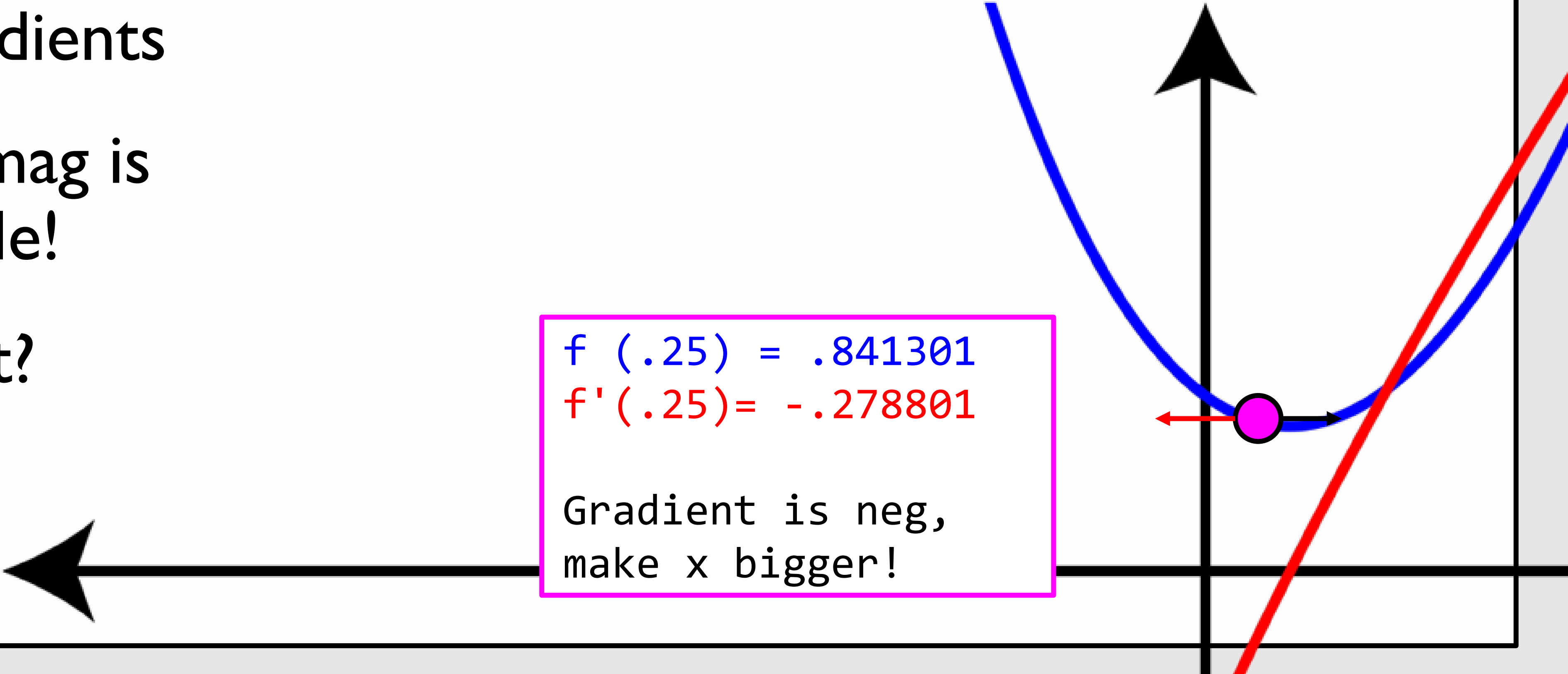
Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

We were moving around by .25 every time, instead we could move based off the gradients

At .25, gradient mag is small, move a little!

Move by gradient?
x = x - ∇f



```
f (.25) = .841301
f'(.25)= -.278801

Gradient is neg,
make x bigger!
```

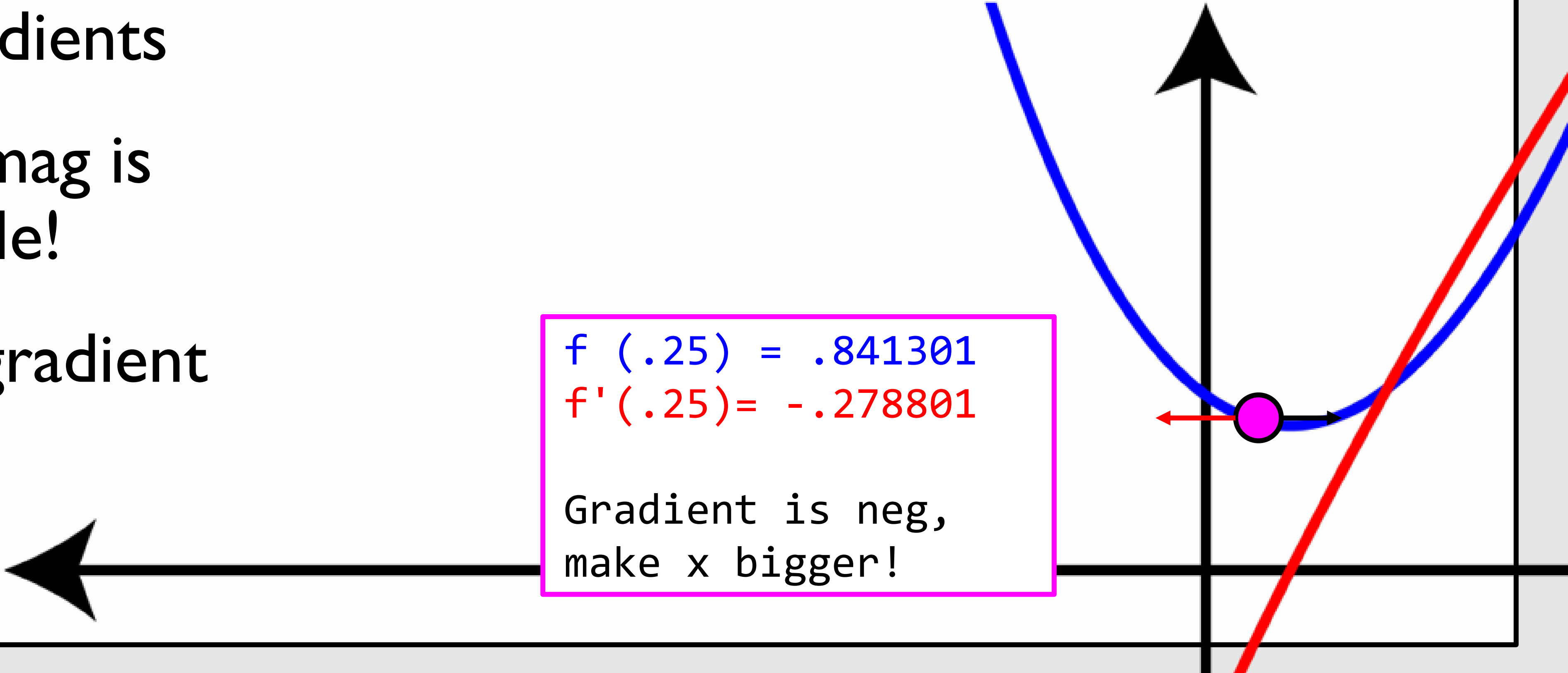
Gradient descent

To minimize $f(x)$, when gradient is positive make x smaller, when negative make x larger!

We were moving around by .25 every time, instead we could move based off the gradients

At .25, gradient mag is small, move a little!

Move by scaled gradient
$x = x - \eta \nabla f$



```
f (.25) = .841301
f'(.25)= -.278801

Gradient is neg,
make x bigger!
```

Gradient Descent Algorithm

To find $\text{argmin}_x f(x)$

Initialize x somehow

Gradient Descent Algorithm

```
To find $\text{argmin}_x f(x)$

Initialize x somehow
Until converged:
        Computer gradient $\nabla f(x)$
```

Gradient Descent Algorithm

To find $\text{argmin}_x f(x)$

```
Initialize x somehow
Until converged:
        Computer gradient $\nabla f(x)$
        $x = x - \eta \nabla f$
```

Gradient Descent Algorithm

To find $\mathrm{argmin}_x f(x)$

```
Initialize x somehow
Until converged:
        Computer gradient $\nabla f(x)$
        $x = x - \eta \nabla f$
```

η is *learning rate*

Gradient Descent Algorithm

To find $\text{argmin}_x f(x)$

```
Initialize x somehow (how?)
Until converged (when?):
        Computer gradient $\nabla f(x)$
        $x = x - \eta\nabla f$
```

η is *learning rate* (how do we pick?)

What does this have to do with ML?

Remember, we wanted to optimize our models to fit the data. First we need a measure of "goodness-of-fit":

Likelihood function - how likely our model thinks our data is

Loss function - how wrong is our model

Want to find parameters that maximize likelihood or minimize loss!

Case study: linear regression

Model: $f^*(x) = ax + b$

Loss function: Sum squared error

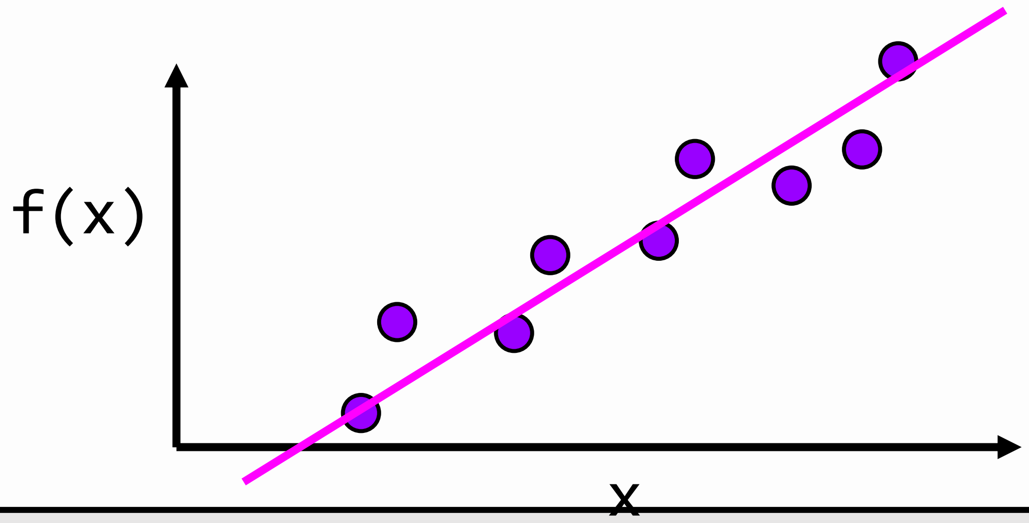
$L(f^*) = \Sigma_i ||f(x_i) - f^*(x_i)||^2$

Penalizes larger errors more than small errors

Optimization: want to find $\text{argmin}_{a,b}\ L(f^*)$

$d/da\ L(f^*) = \Sigma_i\ 2[f(x_i) - f^*(x_i)] * -x$

$d/db\ L(f^*) = \Sigma_i\ 2[f(x_i) - f^*(x_i)] * -1$



Case study: linear regression

Optimization: want to find $\text{argmin}_{a,b} L(f^*)$

$$\frac{d}{da} L(f^*) = \Sigma_i\, 2[f(x_i) - f^*(x_i)] \; * -x$$

$$\frac{d}{db} L(f^*) = \Sigma_i\, 2[f(x_i) - f^*(x_i)] \; * -1$$

How do we change b?

Case study: linear regression

Optimization: want to find $\mathrm{argmin}_{a,b} L(f^*)$

$$d/da\, L(f^*) \;=\; \Sigma_i\, 2[f(x_i) - \; f^*(x_i)]^* - x$$
$$d/db\, L(f^*) \;=\; \Sigma_i\, 2[f(x_i) - \; f^*(x_i)]^* - 1$$

How do we change b?

Sum over differences between truth and prediction, if it is positive, make b larger

Case study: linear regression

Optimization: want to find $\text{argmin}_{a,b} L(f^*)$

$$d/da\, L(f^*) \;=\; \Sigma_i\, 2[f(x_i) - \; f^*(x_i)]\;\; (-x)$$
$$d/db\, L(f^*) \;=\; \Sigma_i\, 2[f(x_i) - \; f^*(x_i)]\;\; (-1)$$

How do we change b?

Sum over differences between truth and prediction, if it is positive, make b larger

This makes sense, our model is predicting too small for most data, increase the bias!

Loss functions

Think of loss functions as functions that have the data as givens and take as input the parameters to the model (i.e. model weights, or a and b in linear regression)

Then we want to optimize these parameters to minimize the loss function, which we can do with gradient descent! (or other methods)

Stochastic gradient descent (SGD)

- Given a differentiable function, it's theoretically possible to find its minimum analytically: where derivative is 0.
  - Find the mimimum point where gradient is 0

- But, in neural networks there are thousands (often millions) are parameters
  - Impossible to solve analytically

- Instead, we can use SGD, an iterative algorithm
  - Change parameters little by little to minimize the *loss*

Gradient descent vs Stochastic GD

- In GD, in each iteration, computing the gradients requires a full pass over all the data (to compute the loss)
  - This is impossible for a huge dataset (too slow)

$$x_{k+1} = x_k - \eta_k \nabla f(x) = x_k - \eta_k \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x_k)$$

- Instead, we use a (dirty) random estimate
  - Compute the loss on just one training example (or a batch of examples)

What if, at iteration k, we randomly pick an integer $i(k) \in \{1, 2, \ldots, n\}$?

And instead just perform the update?

$$x_{k+1} = x_k - \eta_k \nabla f_{i(k)}(x_k)$$

Stochastic gradient descent (SGD)

Mini-batch stochastic gradient descent

1. Draw a batch of training samples **`x`** and corresponding targets **`y`**.
2. Run the network on **`x`** to obtain predictions **`y_pred`**.
3. Compute the *loss* of the network on the batch, a measure of the mismatch between **`y_pred`** and **`y`**.
4. Compute the gradient of the *loss* with regard to the network's parameters (a backward pass).
5. Move the parameters a little in the opposite direction from the gradient - for example **`W -= step * gradient`** - thus reducing the loss on the batch a bit.

Stochastic gradient descent (SGD)

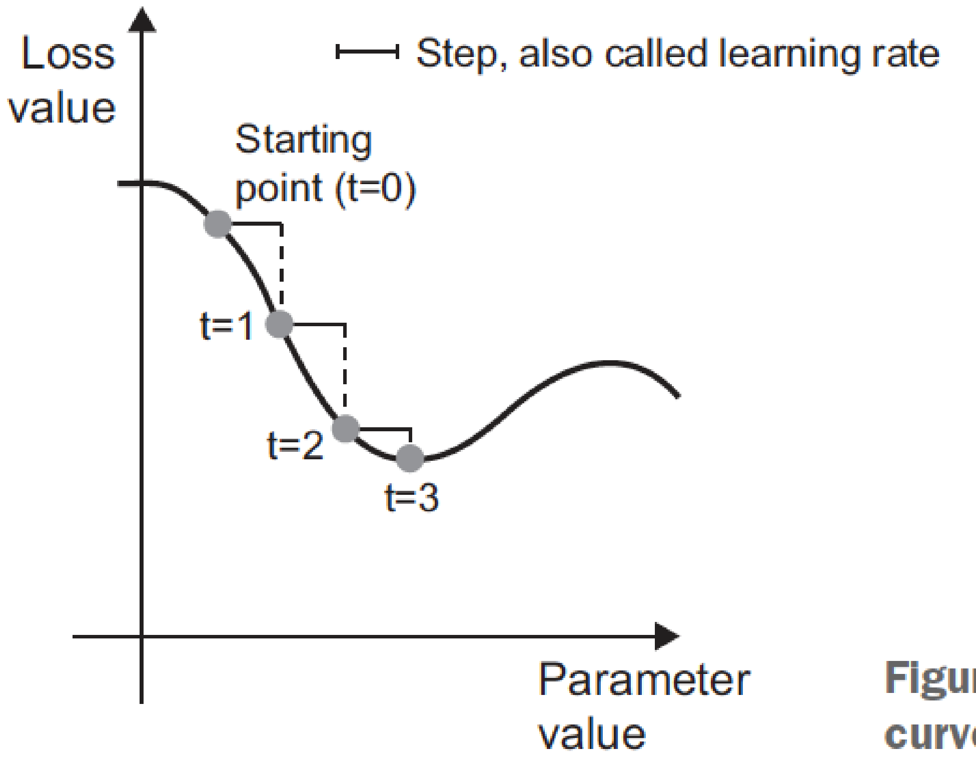


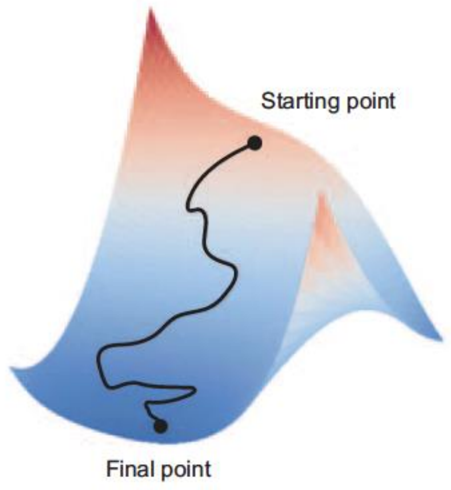
Figure 2.11 SGD down a 1D loss curve (one learnable parameter)

Stochastic gradient descent (SGD)

- Step (learning rate)
  - If too small, the descent down the curve will take many iterations, and it could get stuck in a local minimum.
  - If too large, your updates may end up taking you to completely random locations on the curve.

- True SGD: batch contains only one sample

- Batch SGD: batch contains the whole data



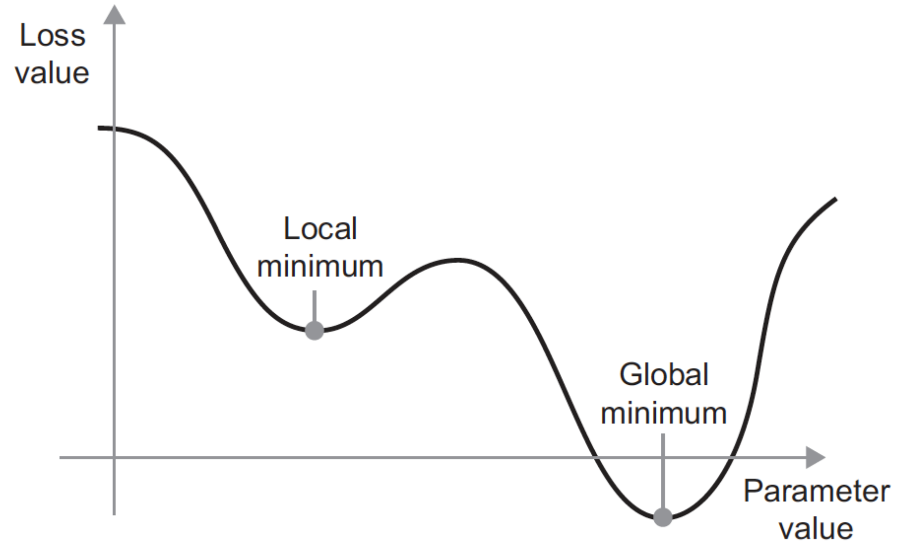
Other optimization methods

- Take into account previous weight updates when computing the next weight update, rather than just looking at the current value of the gradients.

- *Momentum*:

  Momentum addresses two issues with SGD: convergence speed and local minima.

Momentum

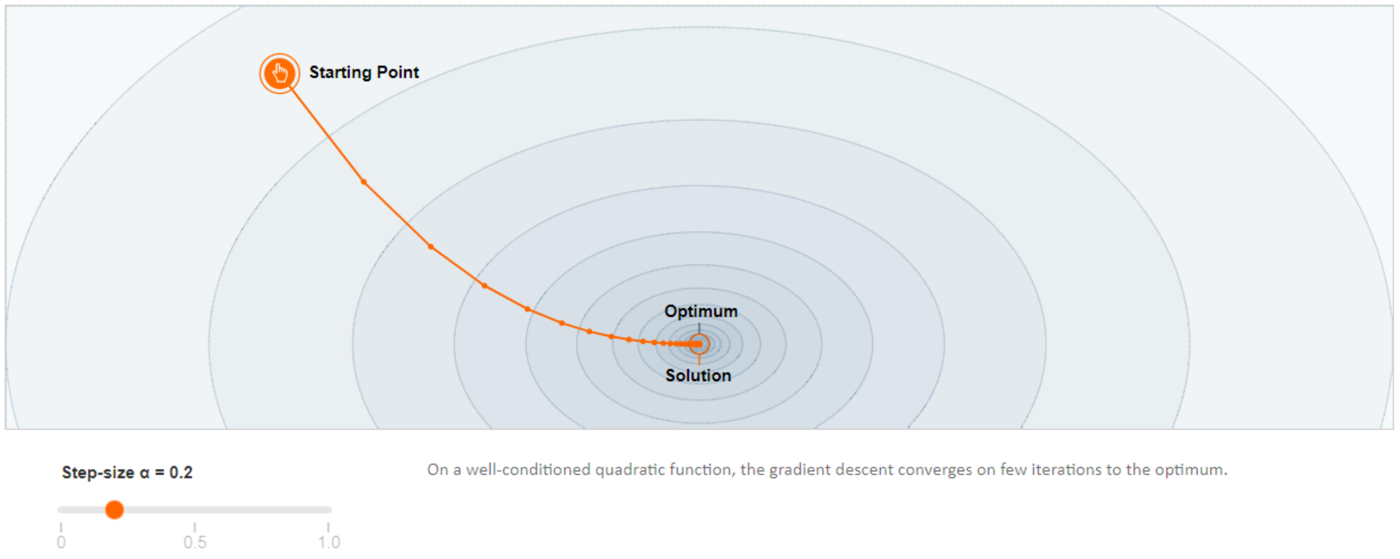


- Inspired from physics

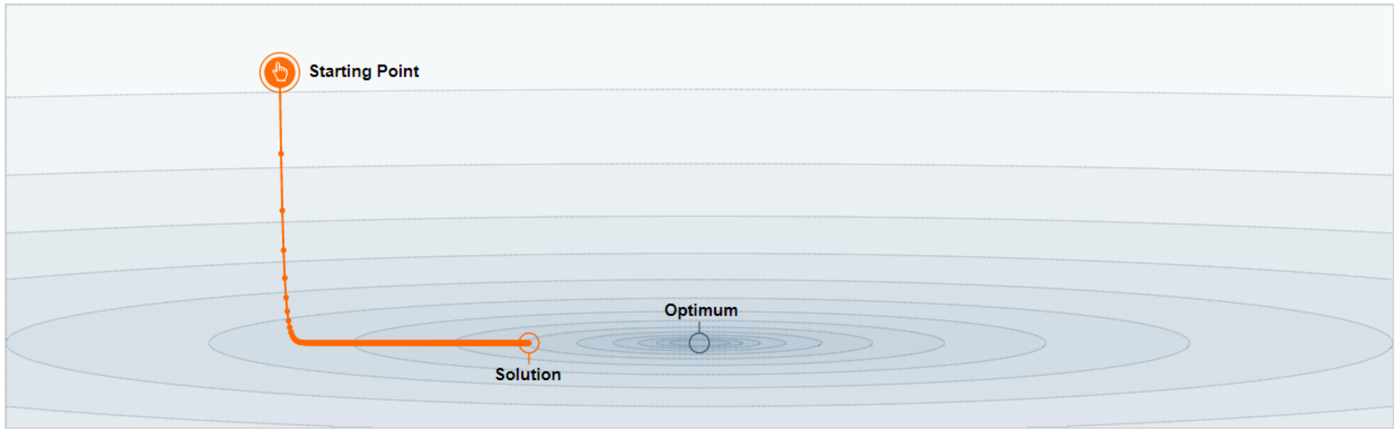
- Consider a small ball rolling down the loss curve
- If it has enough momentum, the ball won't get stuck in a ravine and will end up at the global minimum
- Current slope value (current acceleration) + current velocity (resulting from past acceleration)

Demo

- https://fa.bianp.net/teaching/2018/eecs227at/gradient_descent.html



On a well-conditioned quadratic function, the gradient descent converges on few iterations to the optimum.



Backpropagation

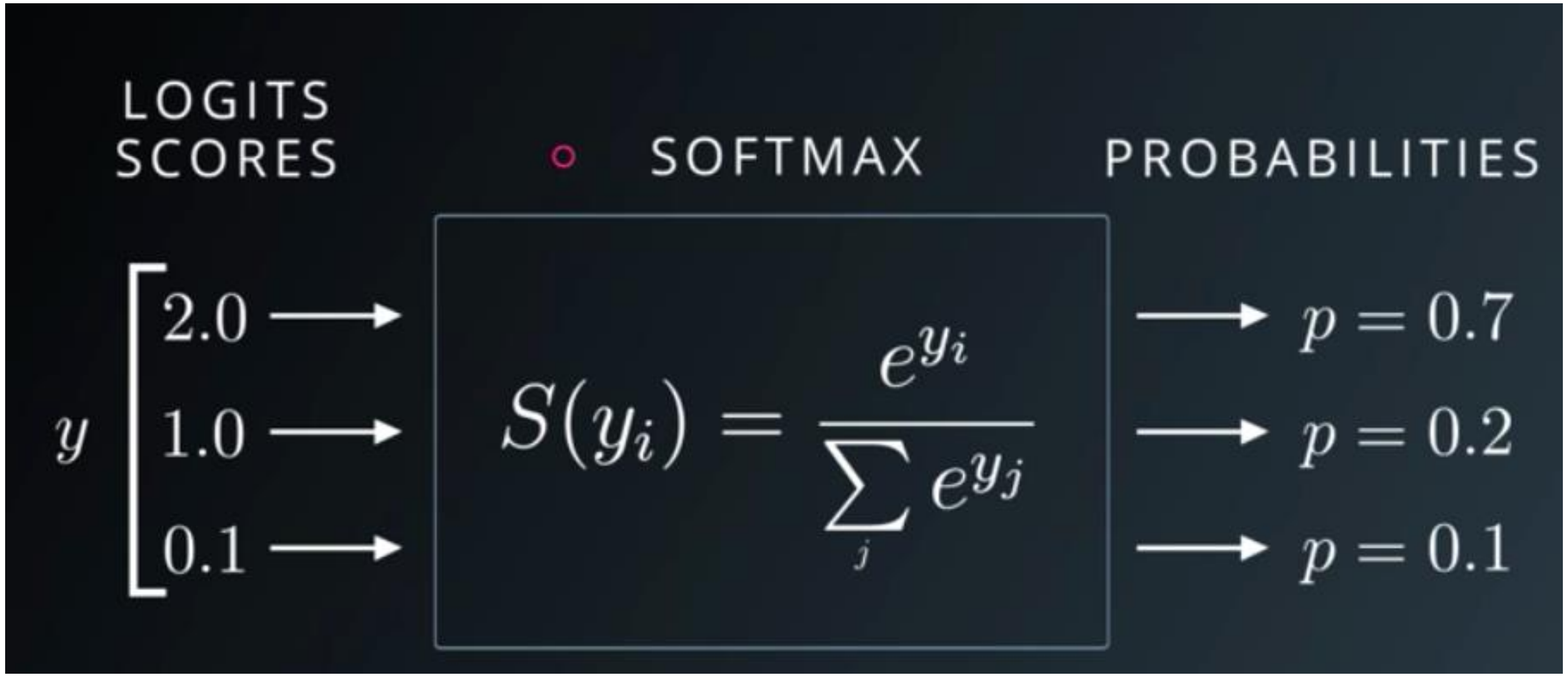
- A neural network function consists of many tensor operations chained together, each of which has a simple, known derivative
  - For instance, this is a network `f` composed of three tensor operations, `a`, `b`, and `c`, with weight matrices `W1`, `W2`, and `W3`:

```
f(W1, W2, W3) = a(W1, b(W2, c(W3)))
```

- Chain rule:
  - derivative of `f(g(x))` can be computed as `` f'(g(x)) * g`(x) ``

- Backpropagation starts with the final loss value and works backward from the top layers to the bottom layers, applying the chain rule to compute the contribution that each parameter had in the loss value.

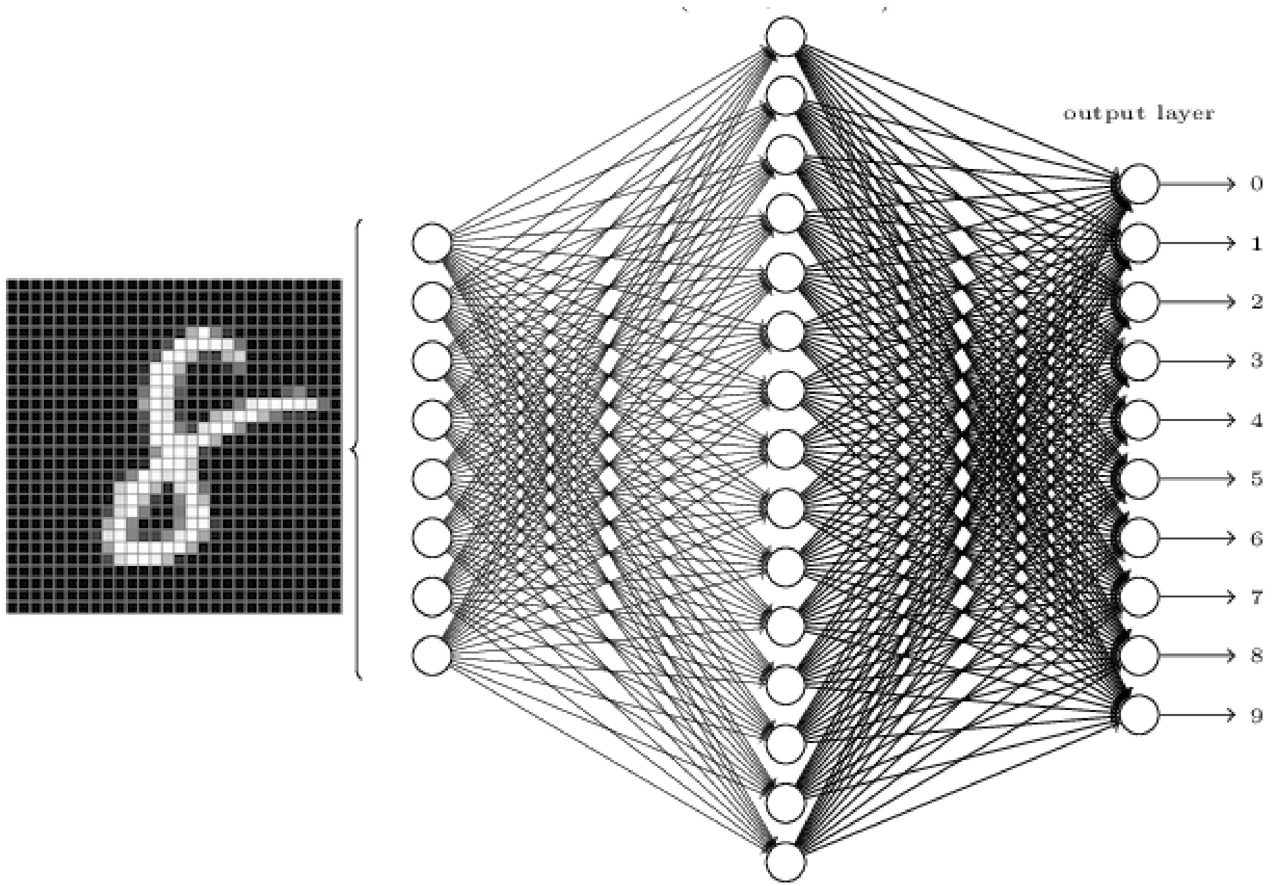
Softmax



$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Udacity Deep Learning Slide on Softmax

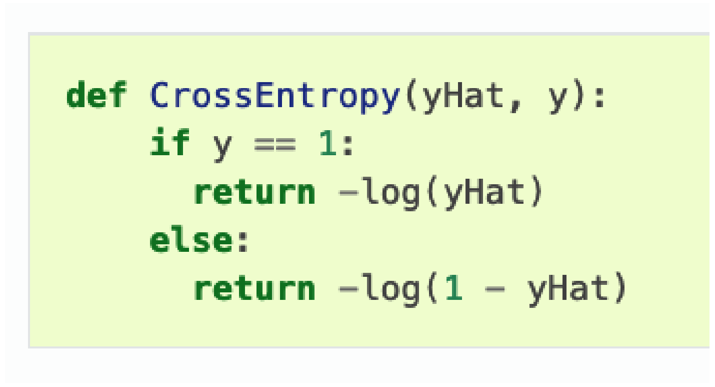
Loss function



System output	Gold label
0.010	0
0.001	0
0.010	0
0.015	0
0.001	0
0.035	0
0.030	0
0.020	0
0.840	1
0.035	0

Cross entropy (binary)

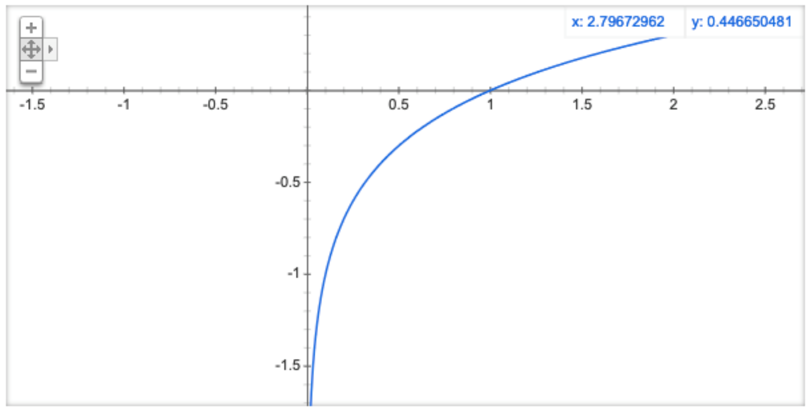
- Code



```
def CrossEntropy(yHat, y):
    if y == 1:
      return -log(yHat)
    else:
      return -log(1 - yHat)
```

- Math

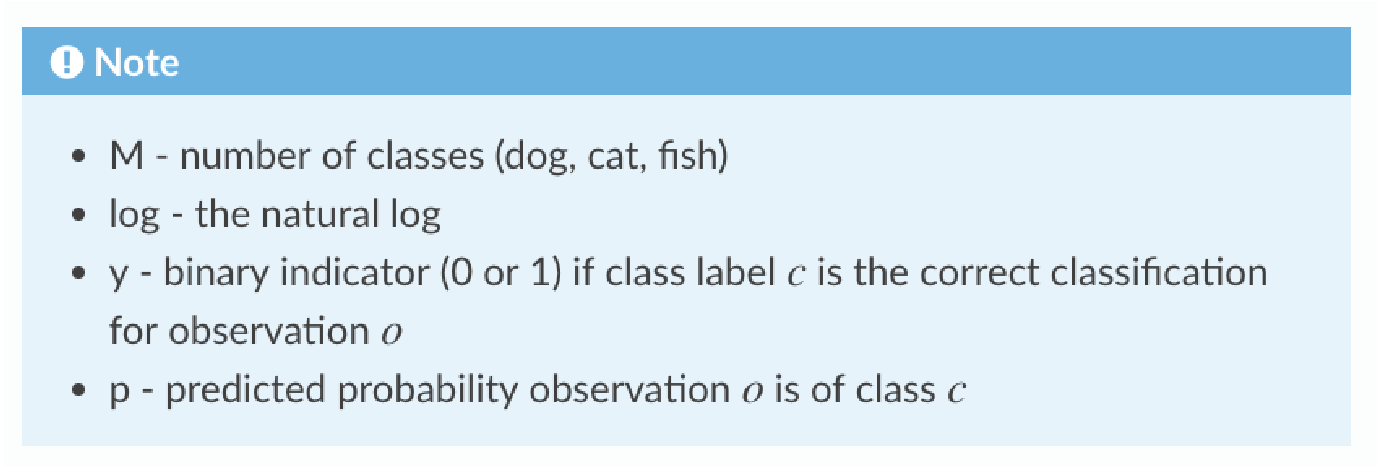
$$-(y\log(p) + (1-y)\log(1-p))$$



Log(x)

Categorical cross-entropy (multi-class)

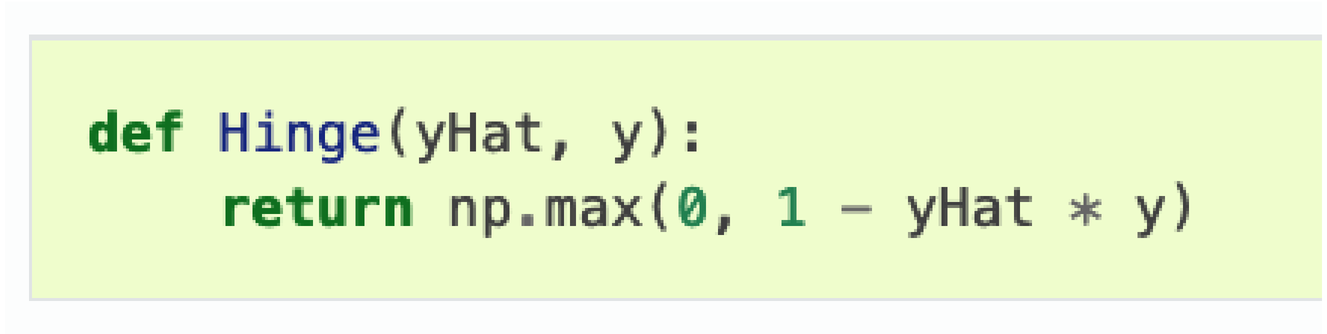
$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$



Note

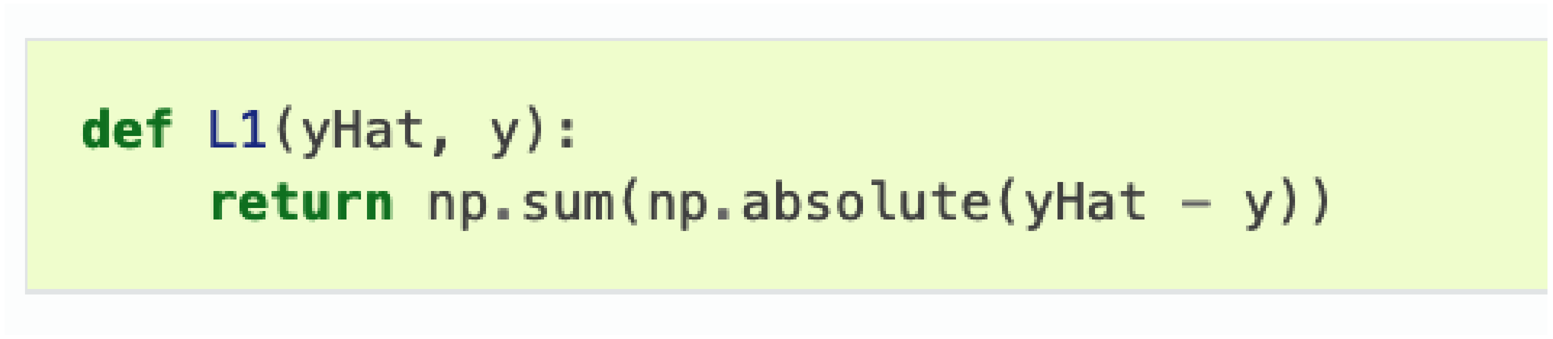
- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Hinge



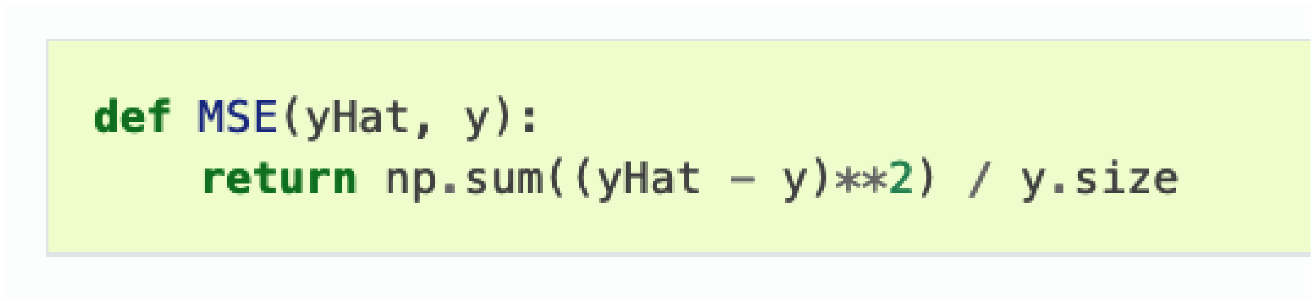
```
def Hinge(yHat, y):
    return np.max(0, 1 - yHat * y)
```

Mean Absolute Error (MAE, aka L1)



```
def L1(yHat, y):
    return np.sum(np.absolute(yHat - y))
```

Mean Squared Error (MSE, aka L2)



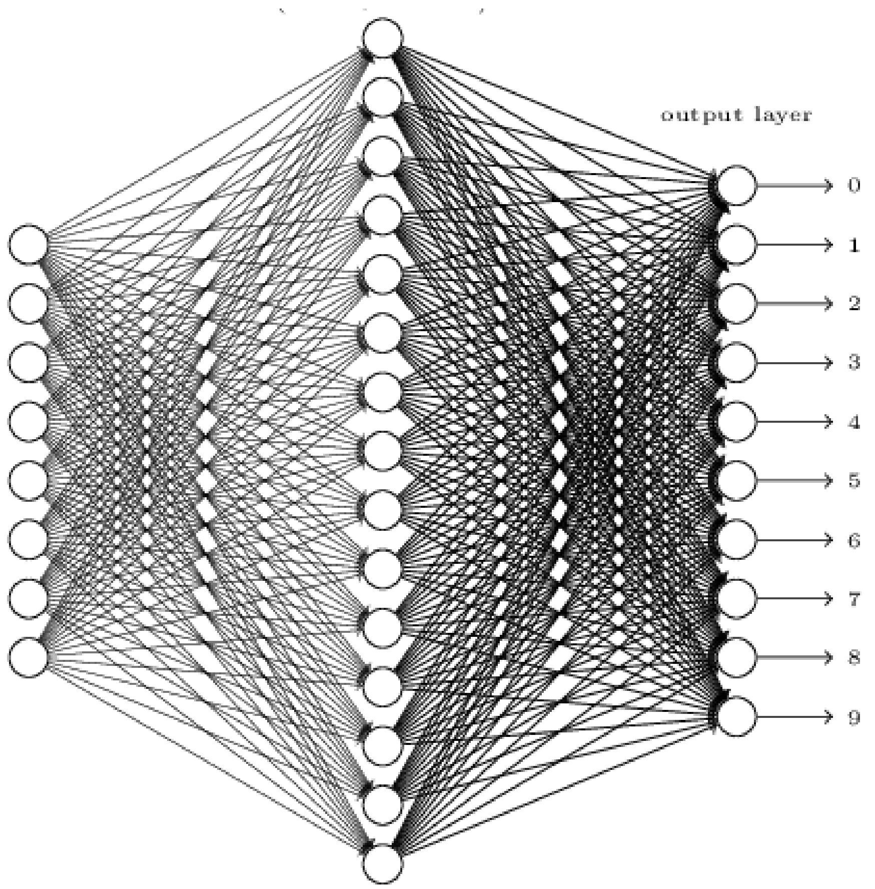
```
def MSE(yHat, y):
    return np.sum((yHat - y)**2) / y.size
```

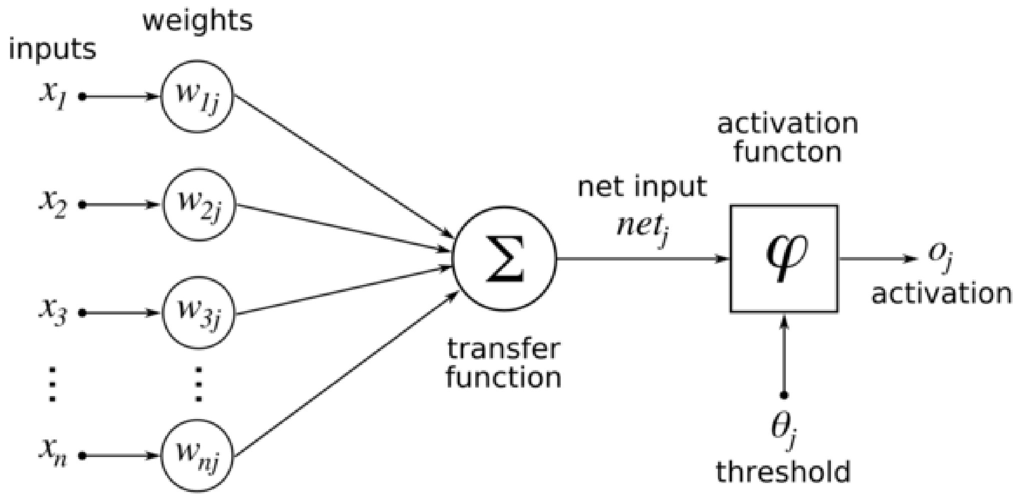
$$J = \frac{1}{N} \sum_{i=1}^{N} (y_i - h_\theta(x_i))^2$$

Loss functions in Keras

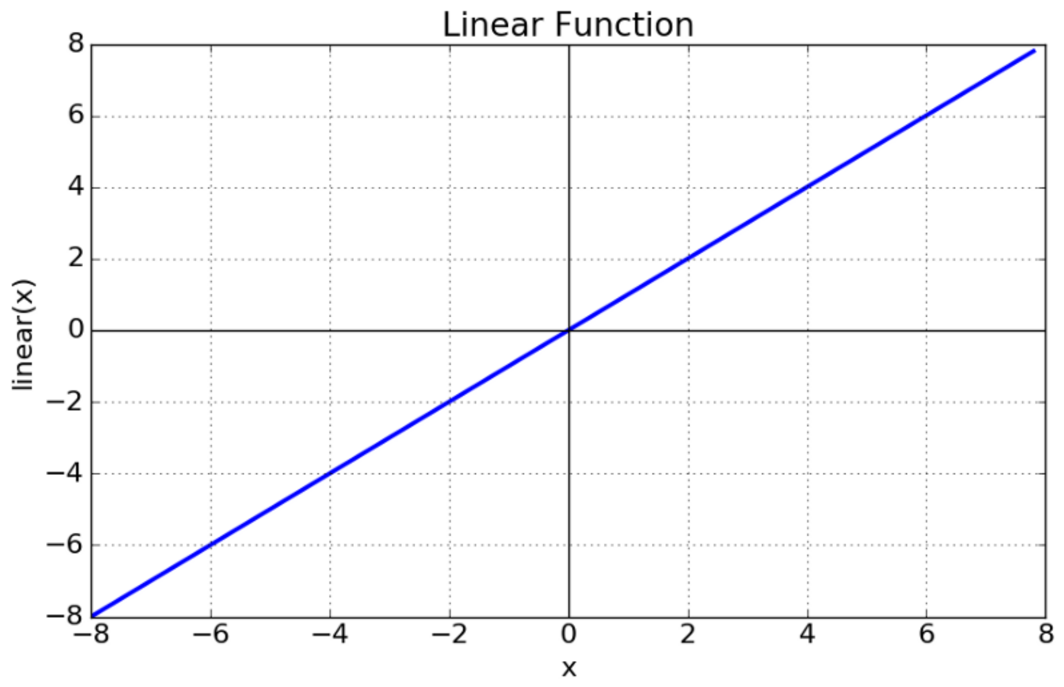
- **mean_squared_error**
- **mean_absolute_error**
- **mean_absolute_percentage_error**
- **mean_absolute_percentage_error**
- **squared_hinge**
- **hinge**
- **categorical_hinge**
- **logcosh**
- **categorical_crossentropy**
- **sparse_categorical_crossentropy**
- **binary_crossentropy**
- **kullback_leibler_divergence**
- **kullback_leibler_divergence**
- **poisson**
- **cosine_proximity**

Activation function

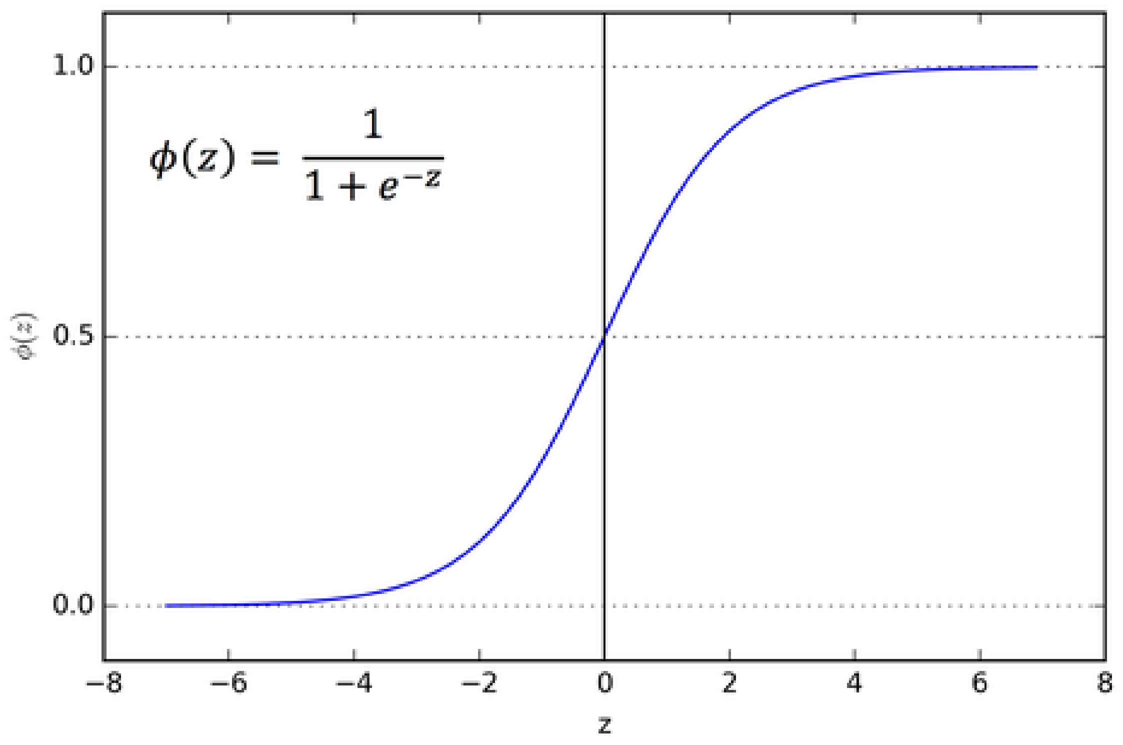




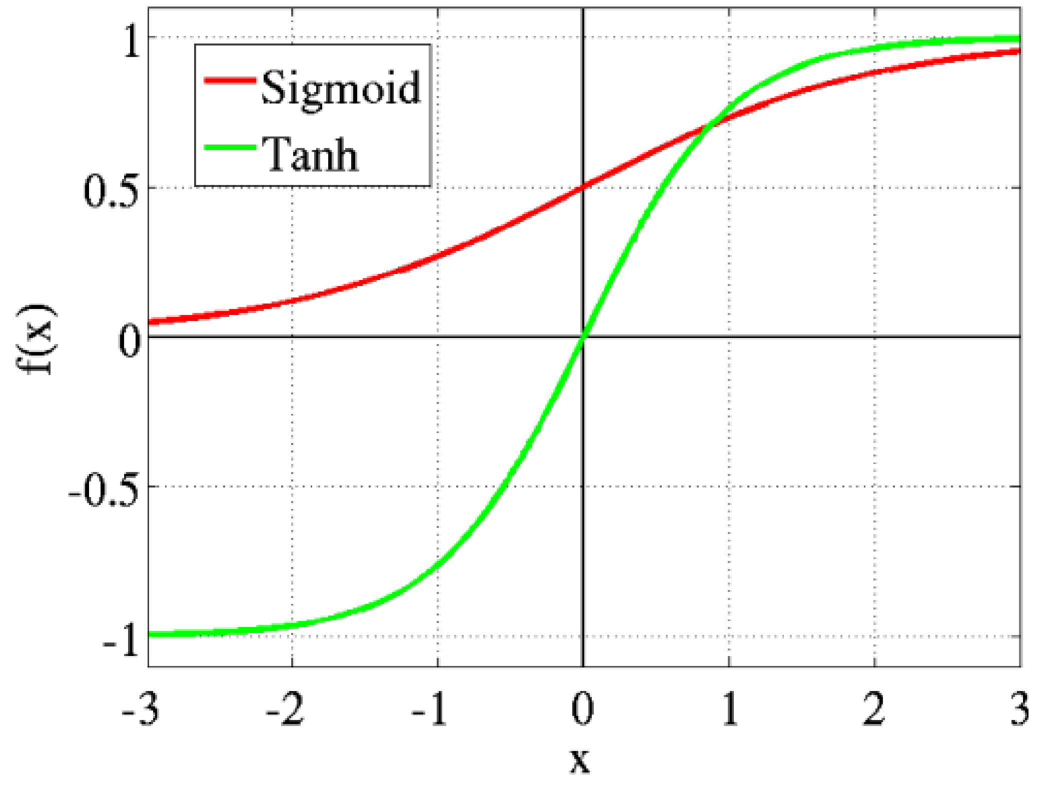
Linear



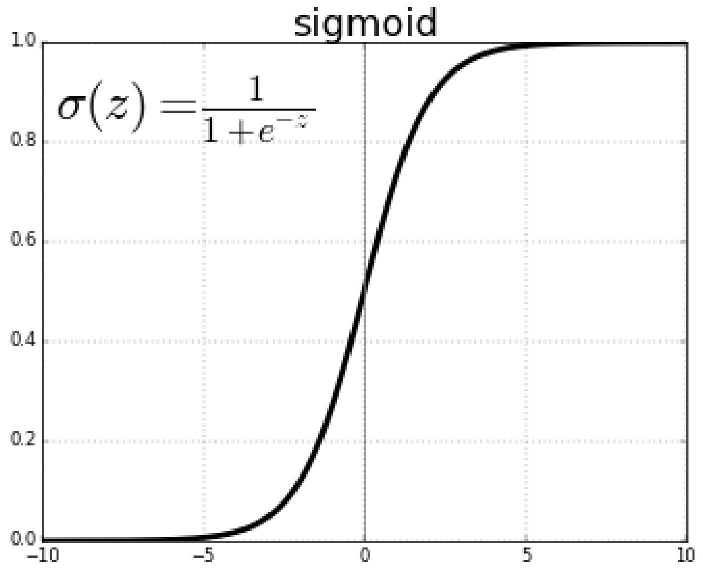
Sigmoid

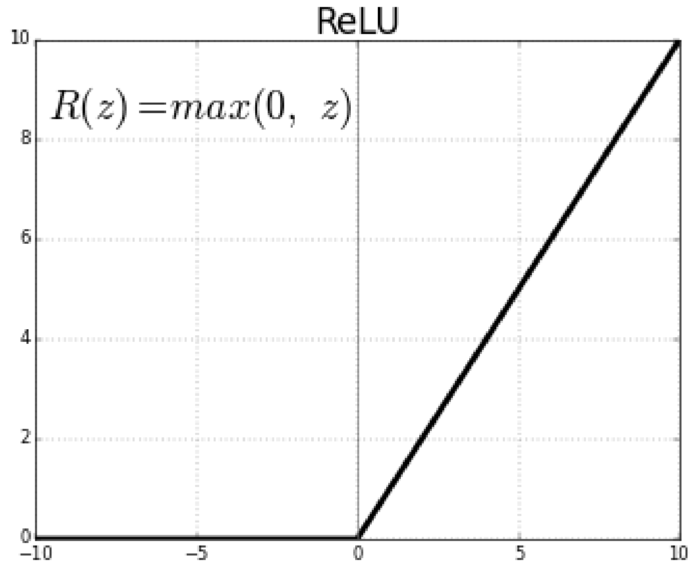


Tanh (Hyperbolic tangent)

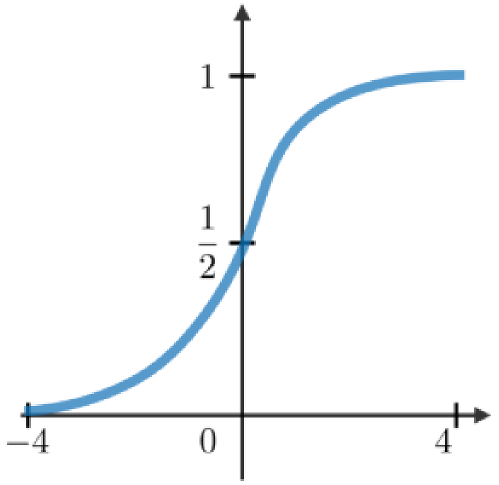
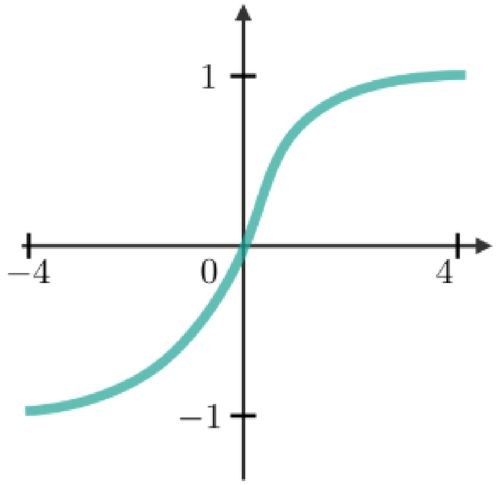
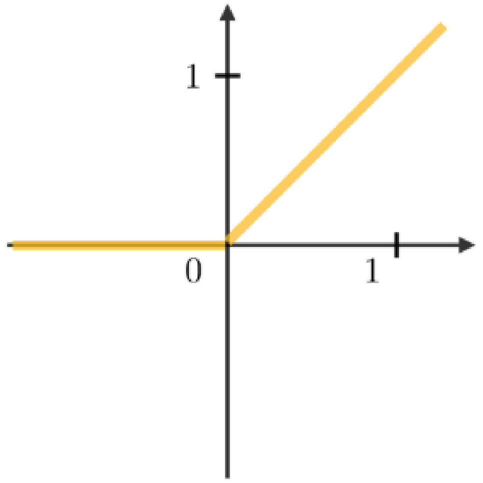
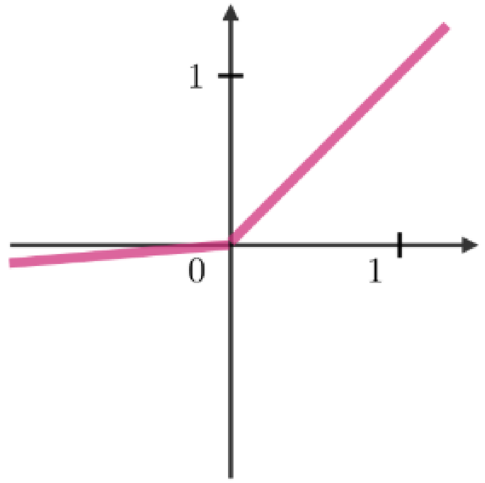


ReLU (Rectified Linear Unit)



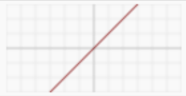
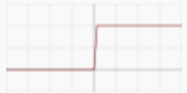
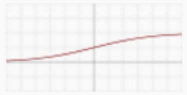
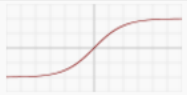
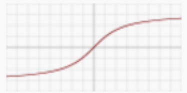
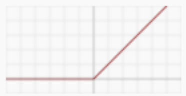
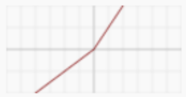
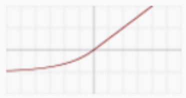
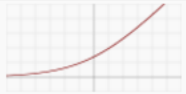


Activations

سیگموئید	تانژانت هذلولوی	یکسو ساز	یکسو ساز نشتی‌دار
$g(z)=\dfrac{1}{1+e^{-z}}$	$g(z)=\dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z)=\max(0,z)$	$g(z)=\max(\epsilon z,z)$ و $\epsilon \ll 1$
			

From https://stanford.edu/~shervine/l/fa/teaching/cs-229/cheatsheet-deep-learning

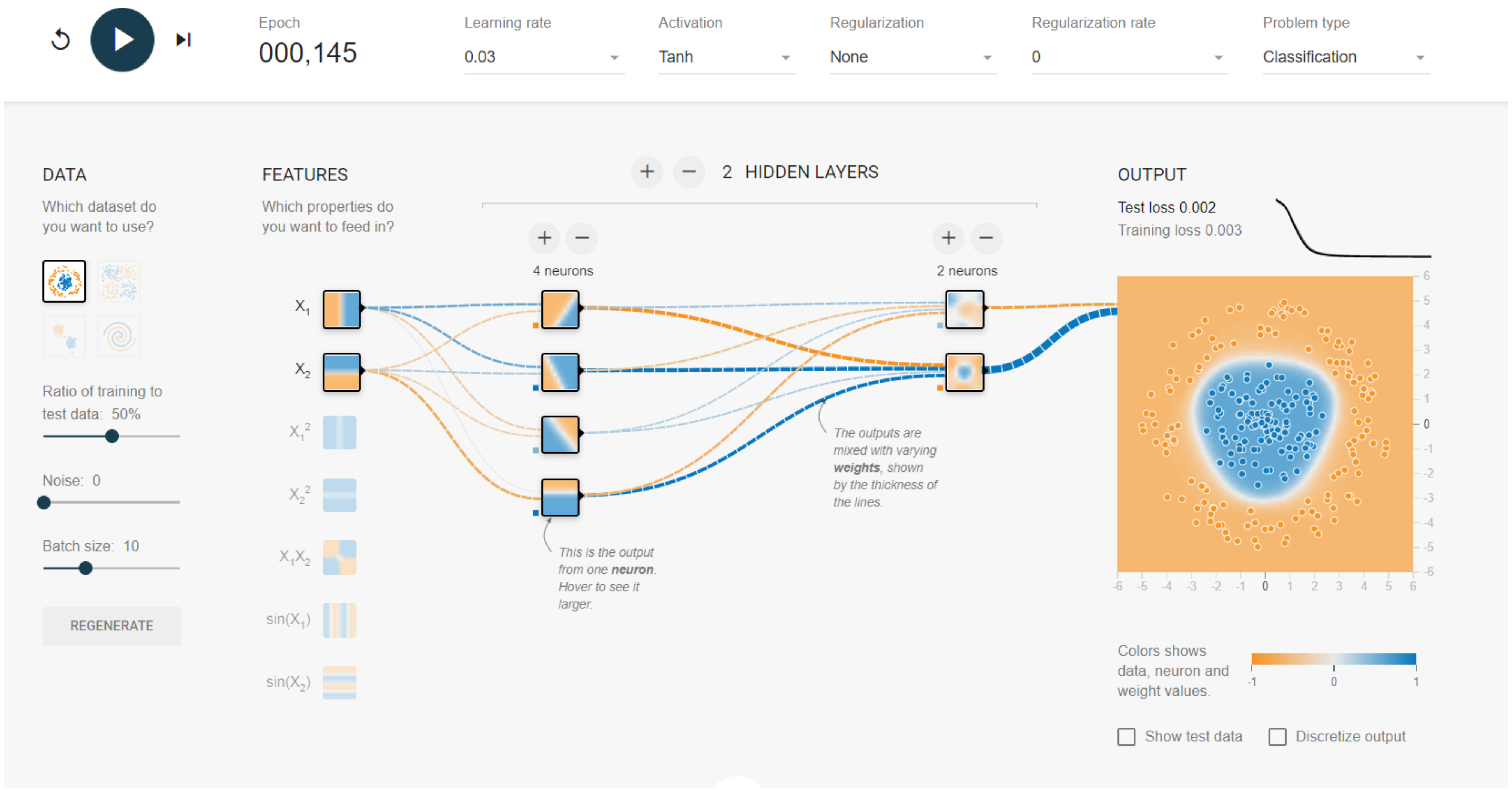
Activations and derivatives

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Gradient-based Optimization

Tensorflow playground



Epoch 000,145

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons

2 neurons

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

Test loss 0.002

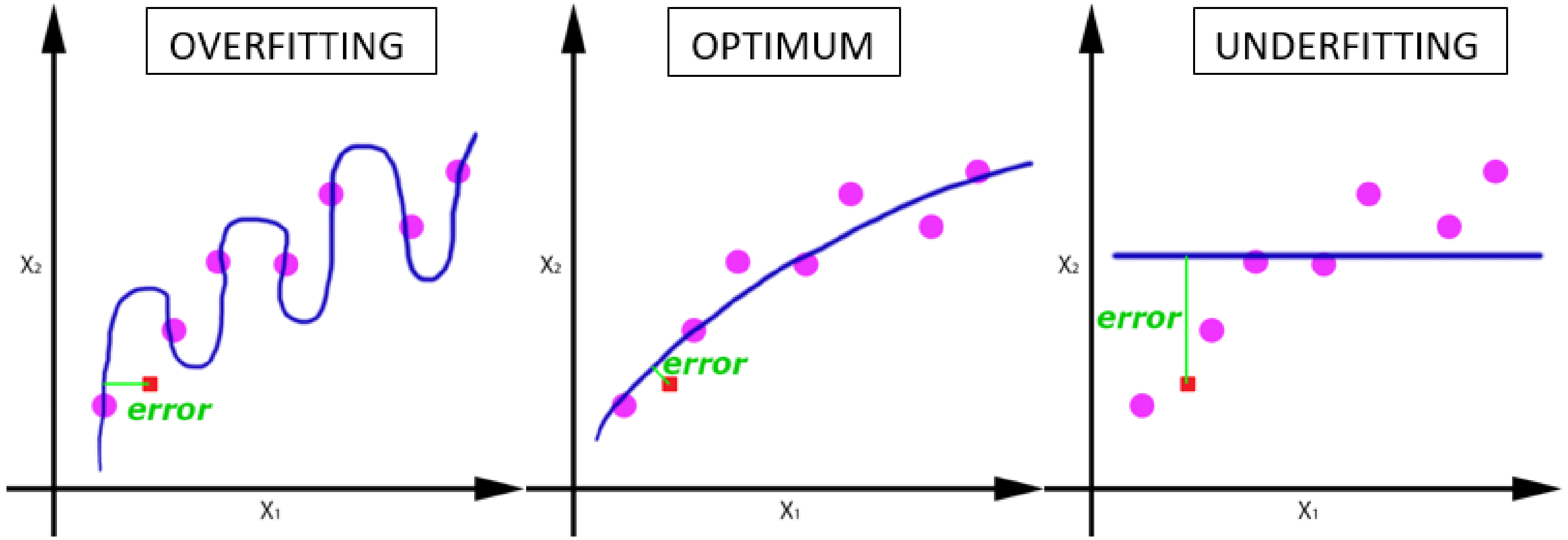
Training loss 0.003

Colors shows data, neuron and weight values.

Show test data

Discretize output

Questions?


OVERFITTING
OPTIMUM
UNDERFITTING
X₂
X₁
error