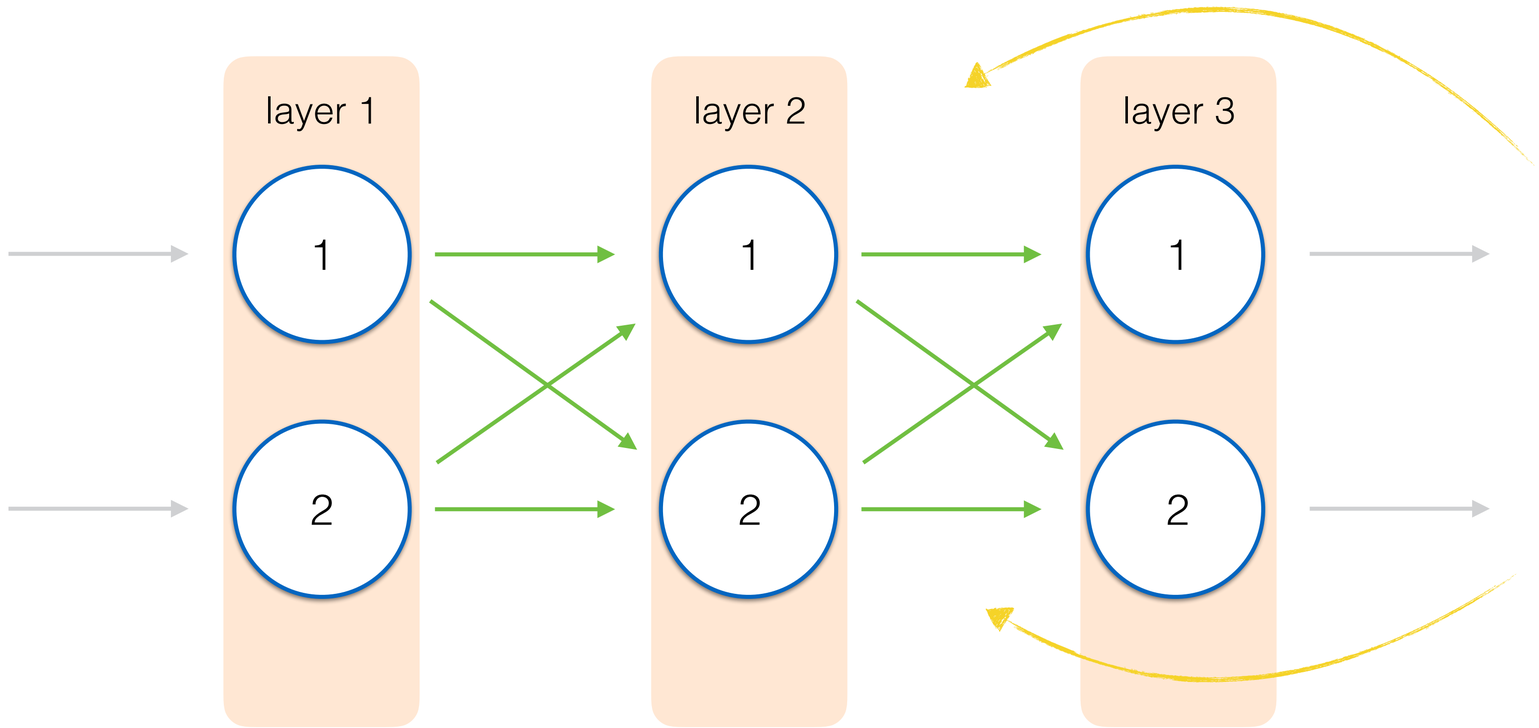
# Review




- Inputs adjusted by weights
- Activation function
- Calculate error
- Back propagate error through network
- Adjust weights with gradient descent

Forward pass:

$$X = W \bullet I$$
$$O = \text{sigmoid}(X)$$

Back propagation:

$$error_{hidden} = W^{T}_{hidden\_output} \bullet error_{output}$$

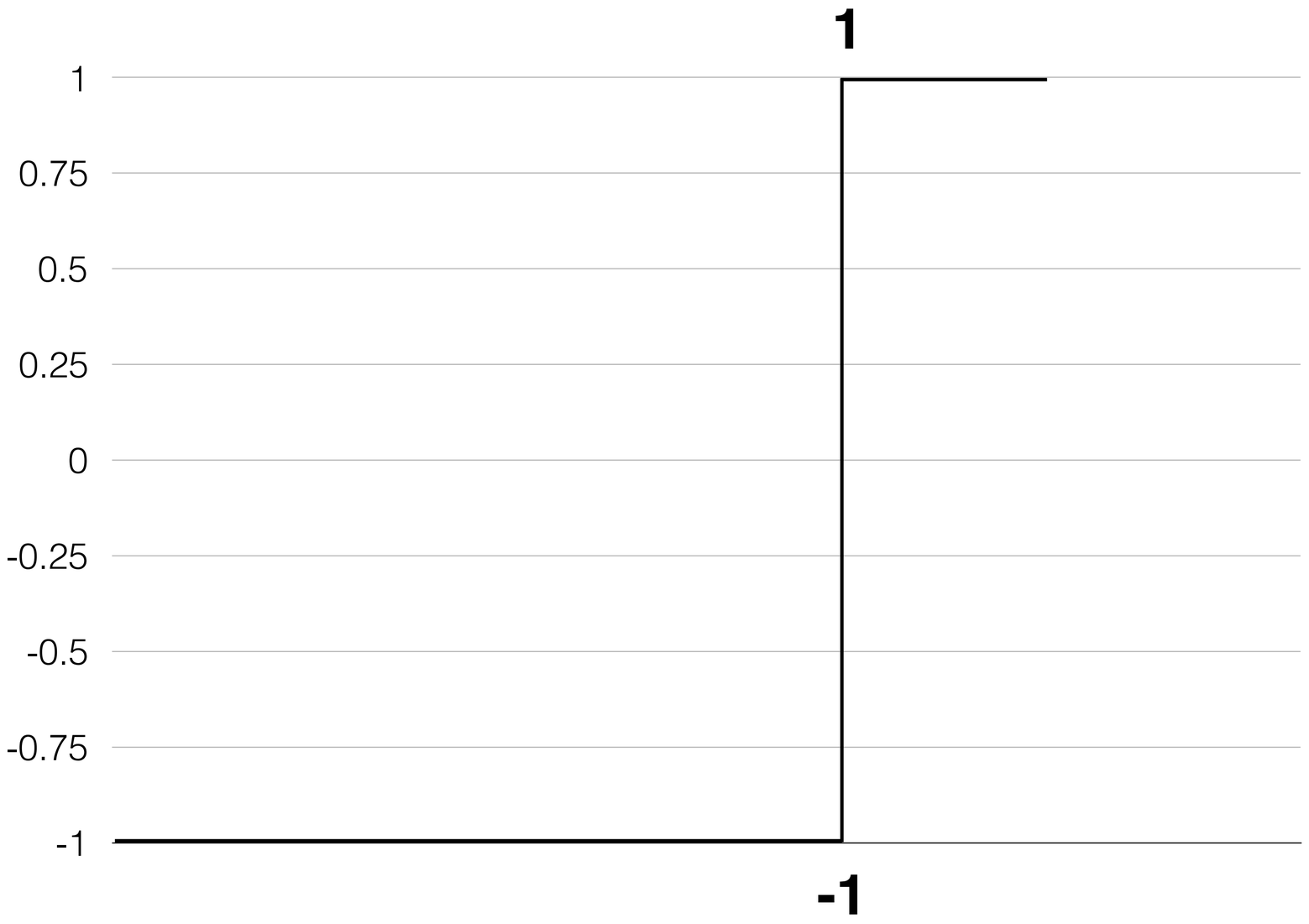
Update weights:

$$\Delta W_{j,k} = \alpha \bullet E_k \bullet O_k( 1 - O_k) \bullet O_j^{T}$$


WONDER TWIN POWERS, ACTIVATE!

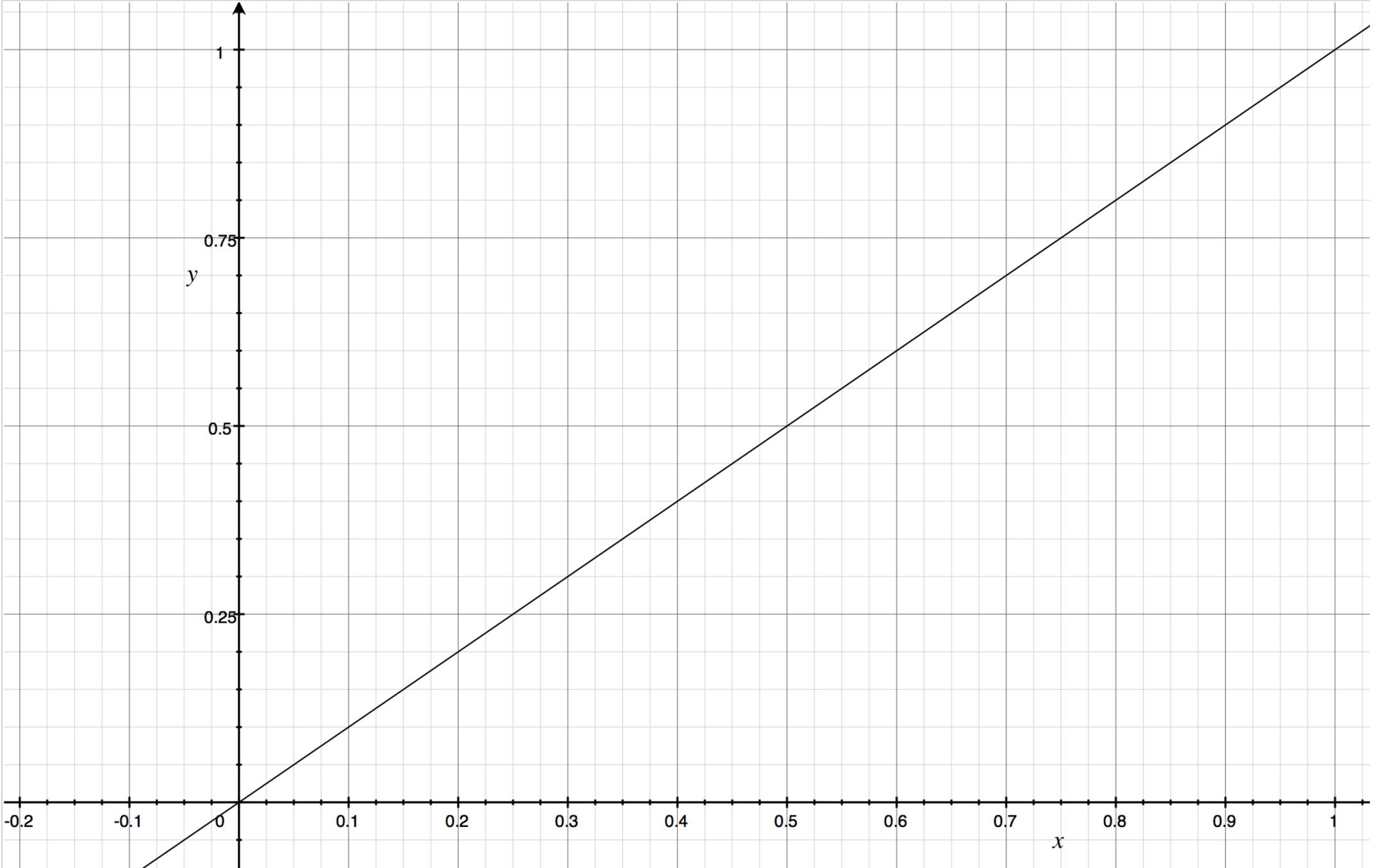
# Activation Functions

# Binary Step



$y=x$

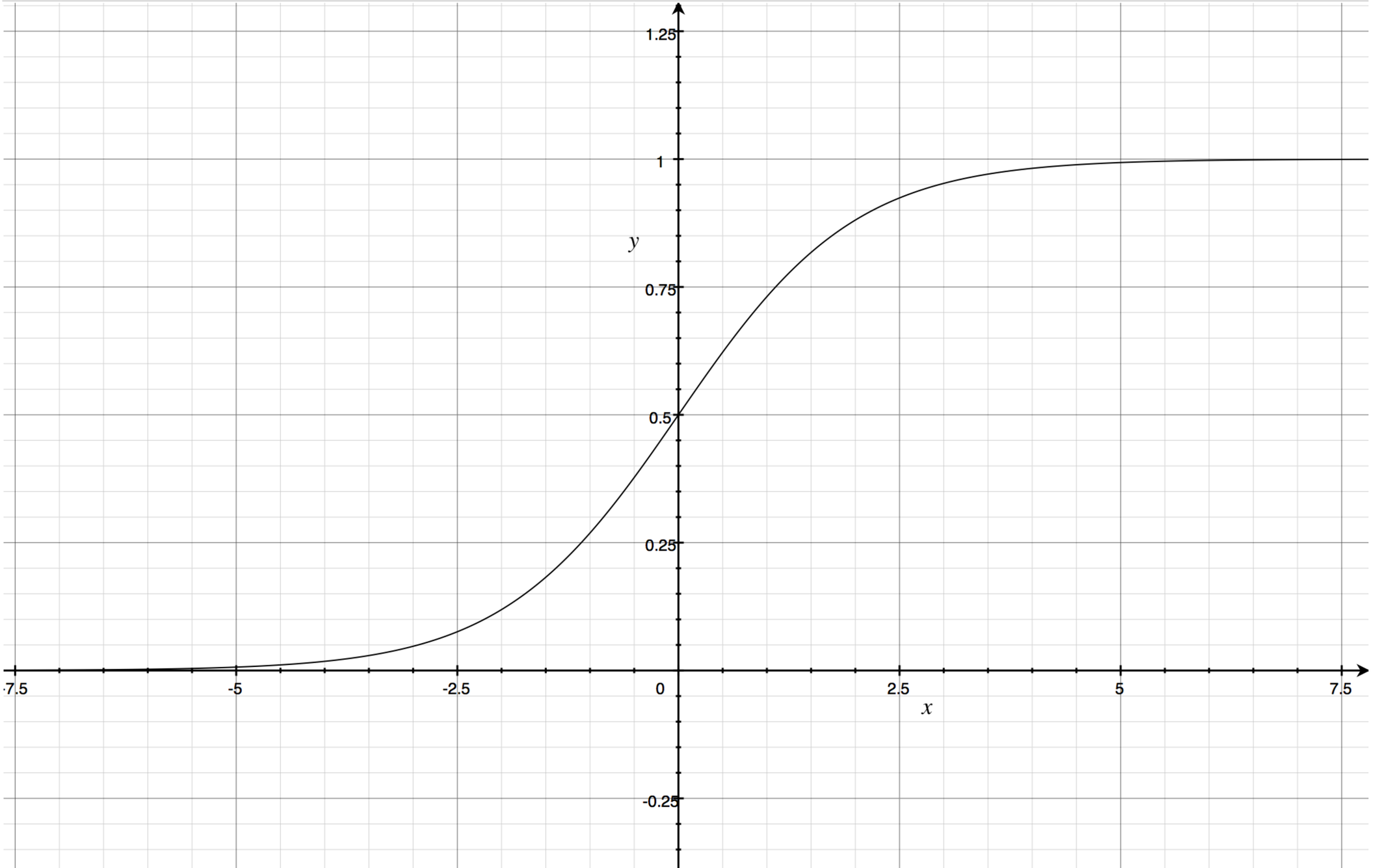
# Linear



# Introduce some non-linearity

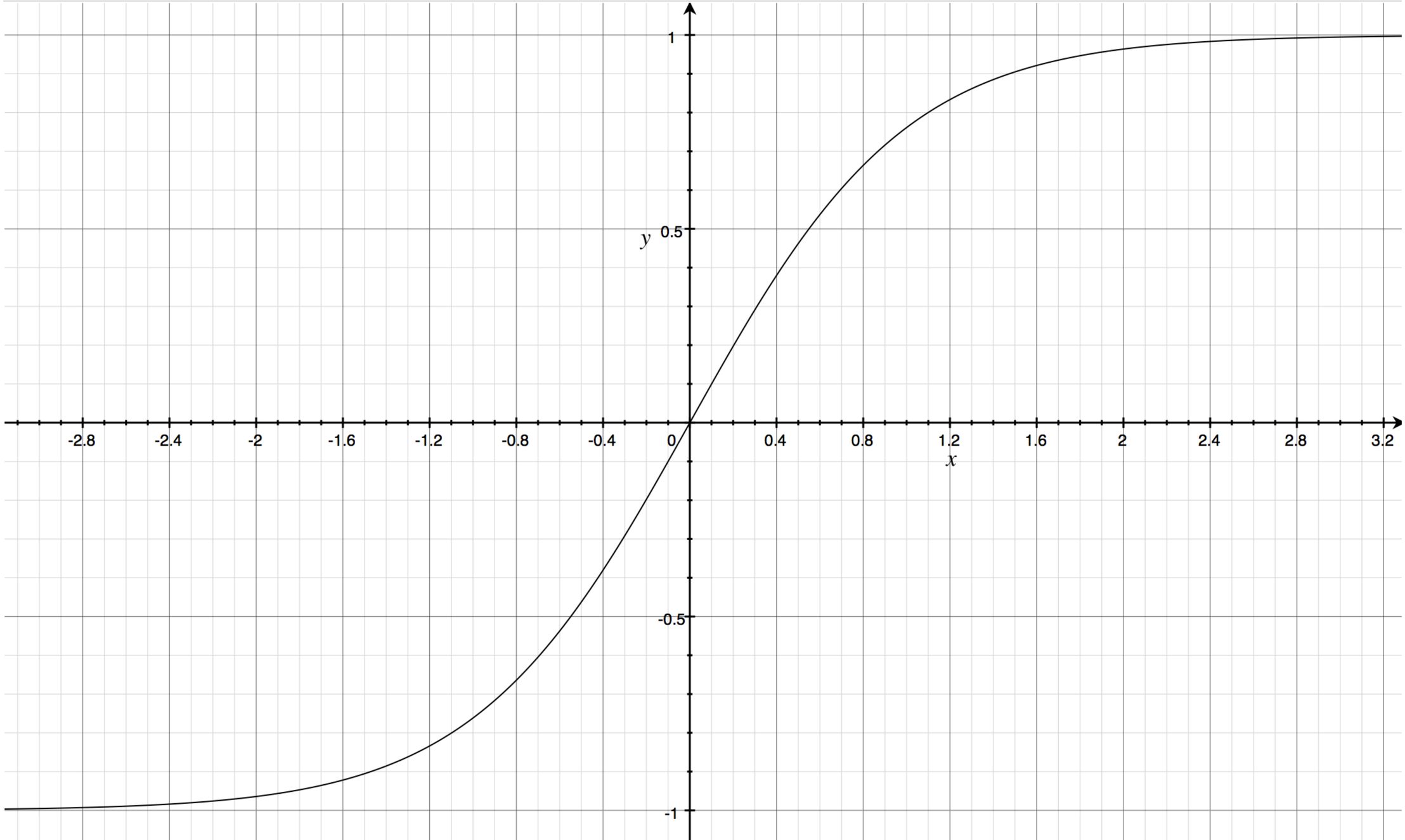
$y = \frac{1}{1+e^{-x}}$

# Sigmoid



$y=\frac{(e^{2x}-1)}{(e^{2x}+1)}$

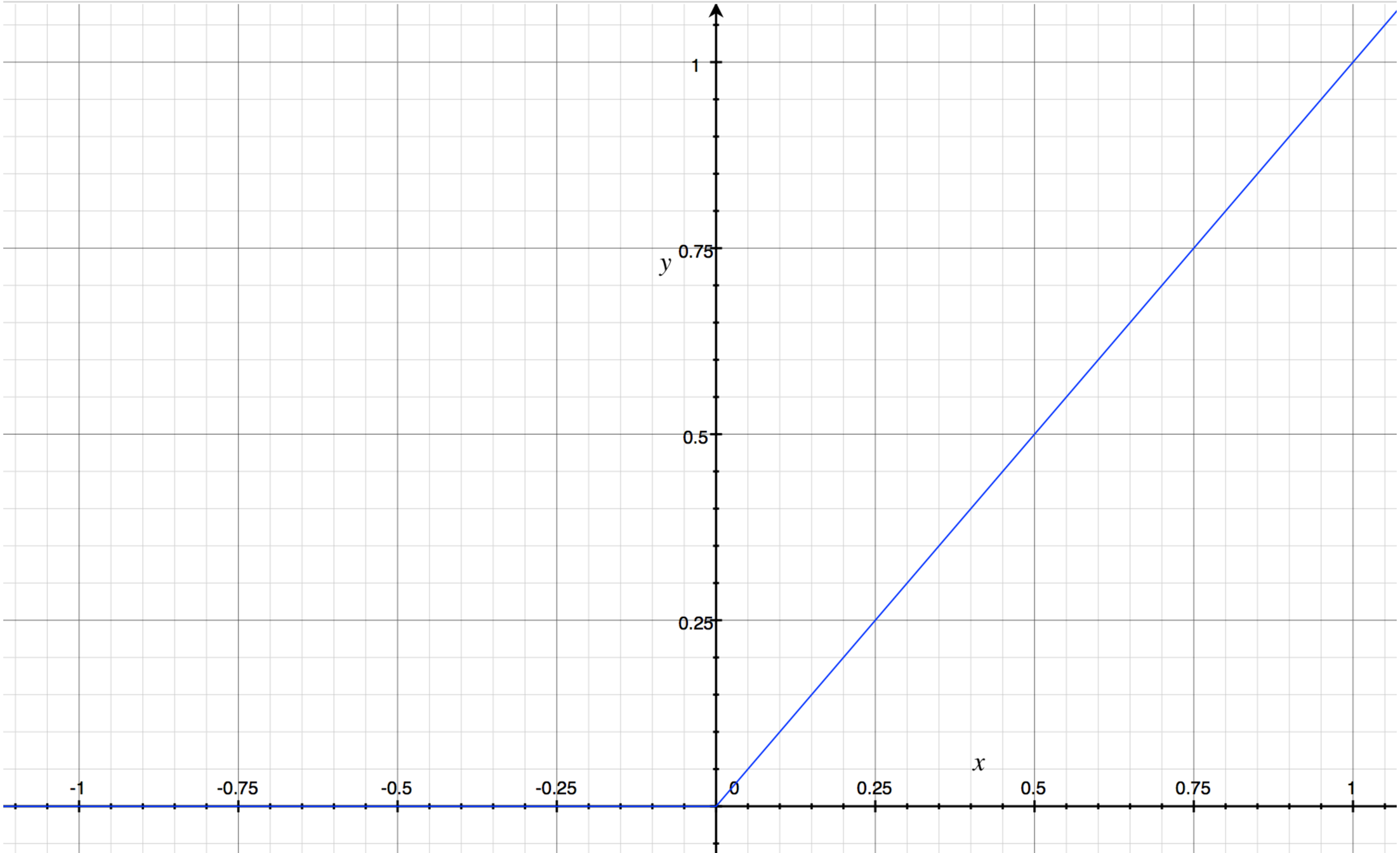
# Tanh (Hyperbolic Tangent)



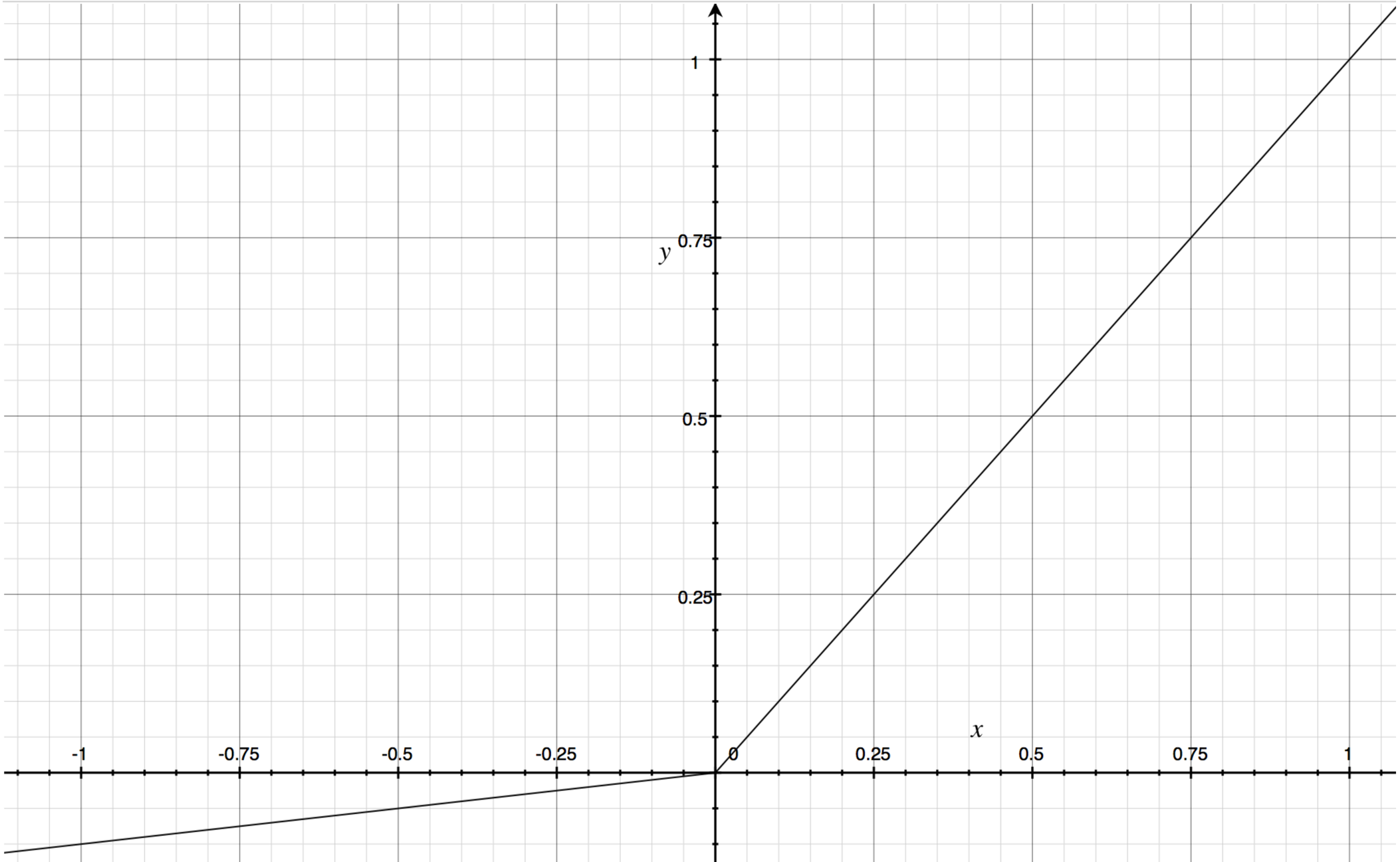
# Sigmoid vs Tanh

- Sigmoid lends itself to probabilities since it outputs values between 0 & 1
- Sigmoid can easily saturate
- Tanh will have stronger gradients and is less prone to saturation

$y=\max(0,x)$ ReLU (Rectified Linear Unit)



$$y=\begin{cases} x & x>0 \\ x\cdot 0.1 & x\leq 0 \end{cases}$$ Leaky ReLU can be used if you have ‘dying ReLU’ problem

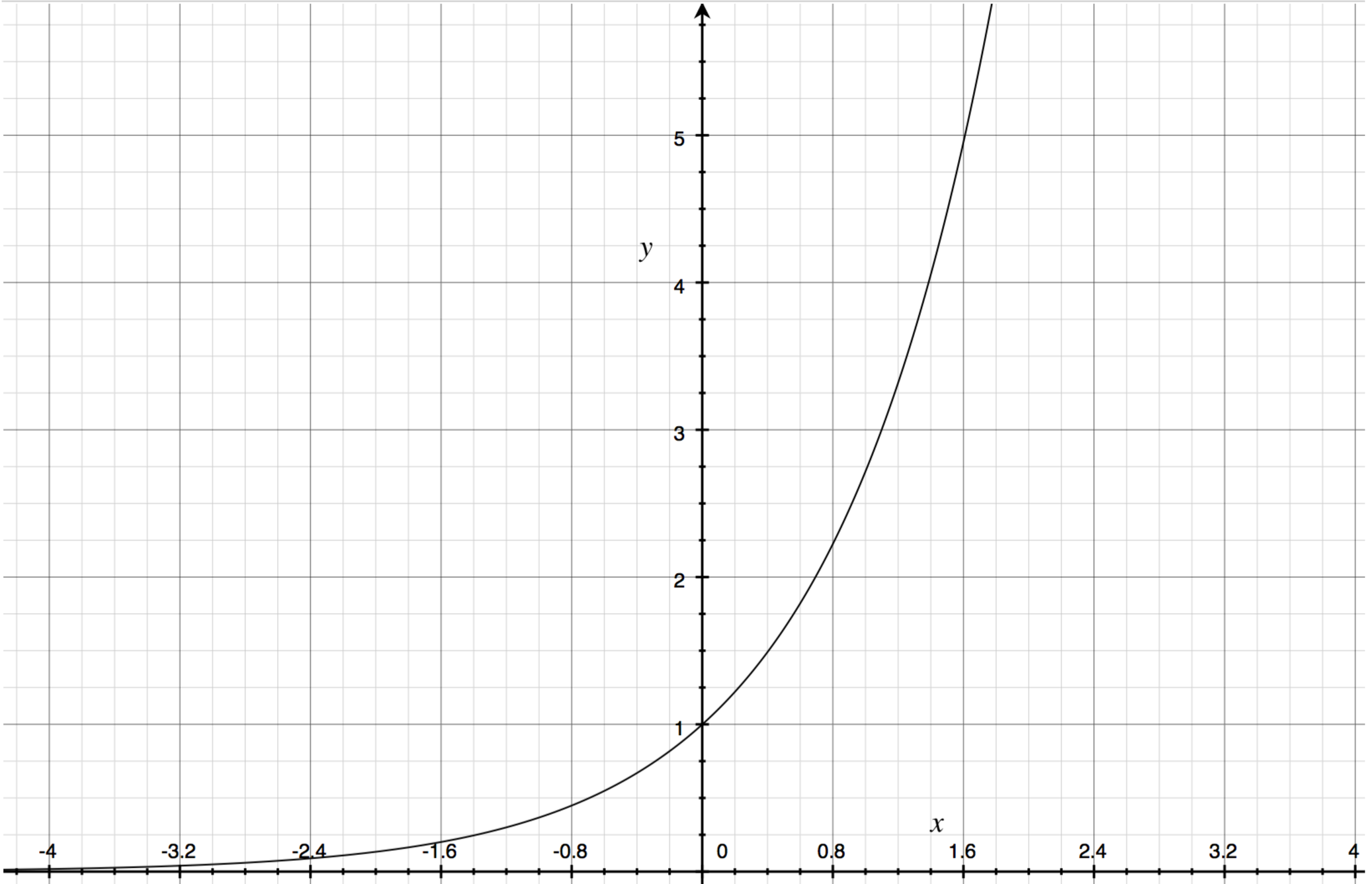


# Softmax

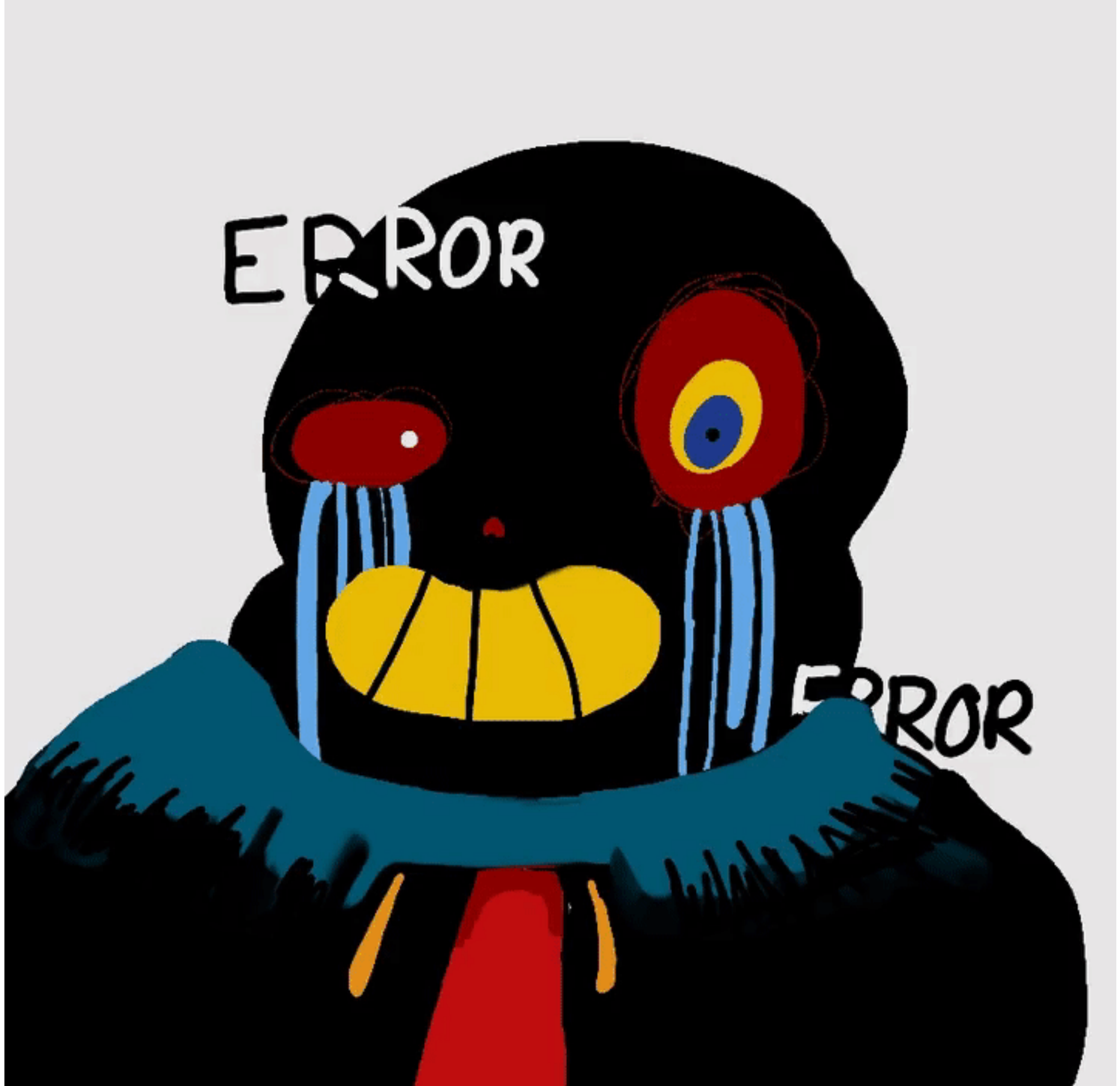
- Not really a traditional activation function, but many times used as the final output layer in a classifier
- Normalizes input array to between 0 & 1
- All outputs will sum to 1
- Outputs each class as percentage between 0 & 1
- Raises e to the power of each input $e^{input}$
- Then divides by the sum of all of those to normalize

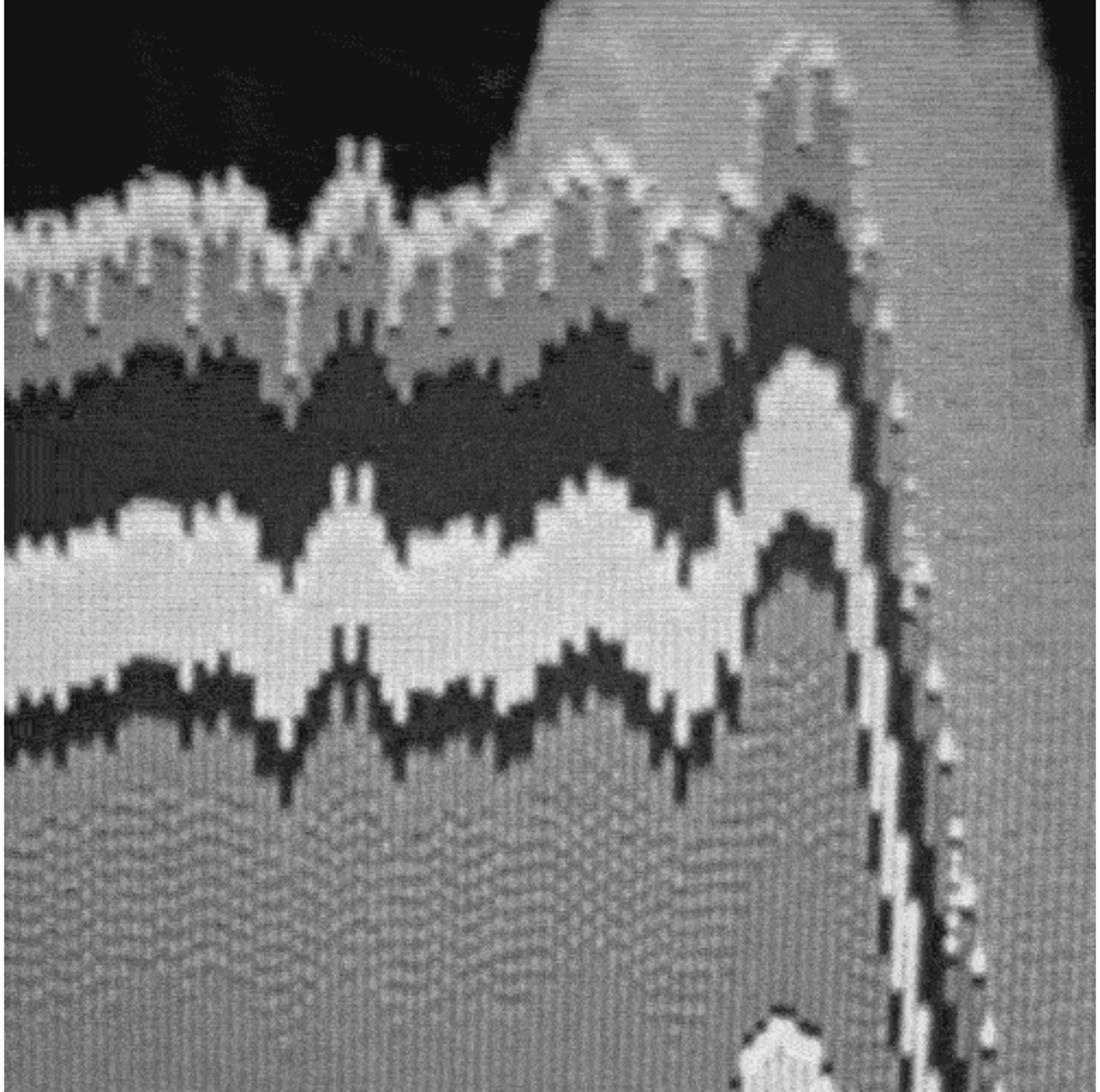
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

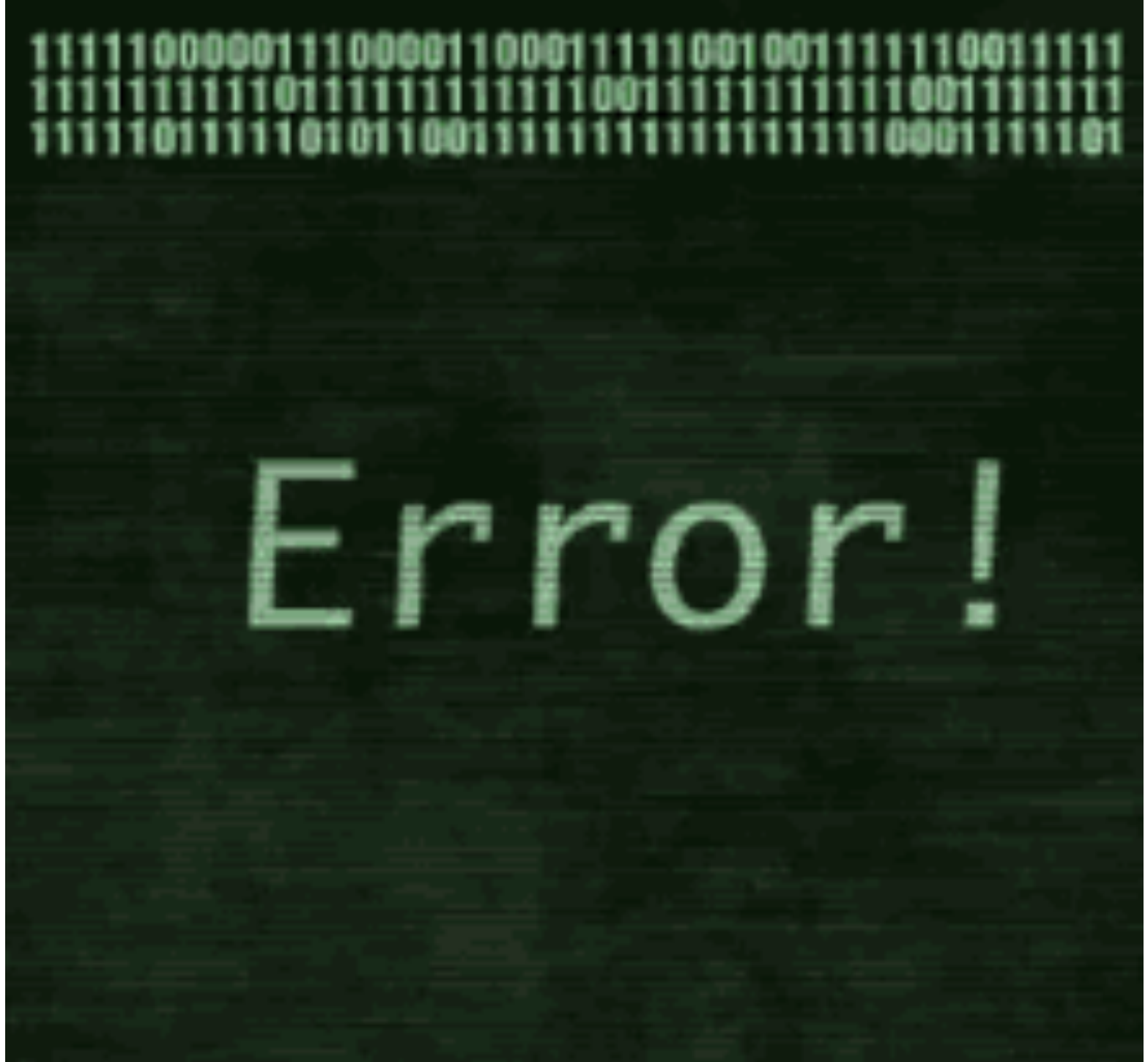
$y=e^x$



ERROR_


ERROR
ERROR




Error !

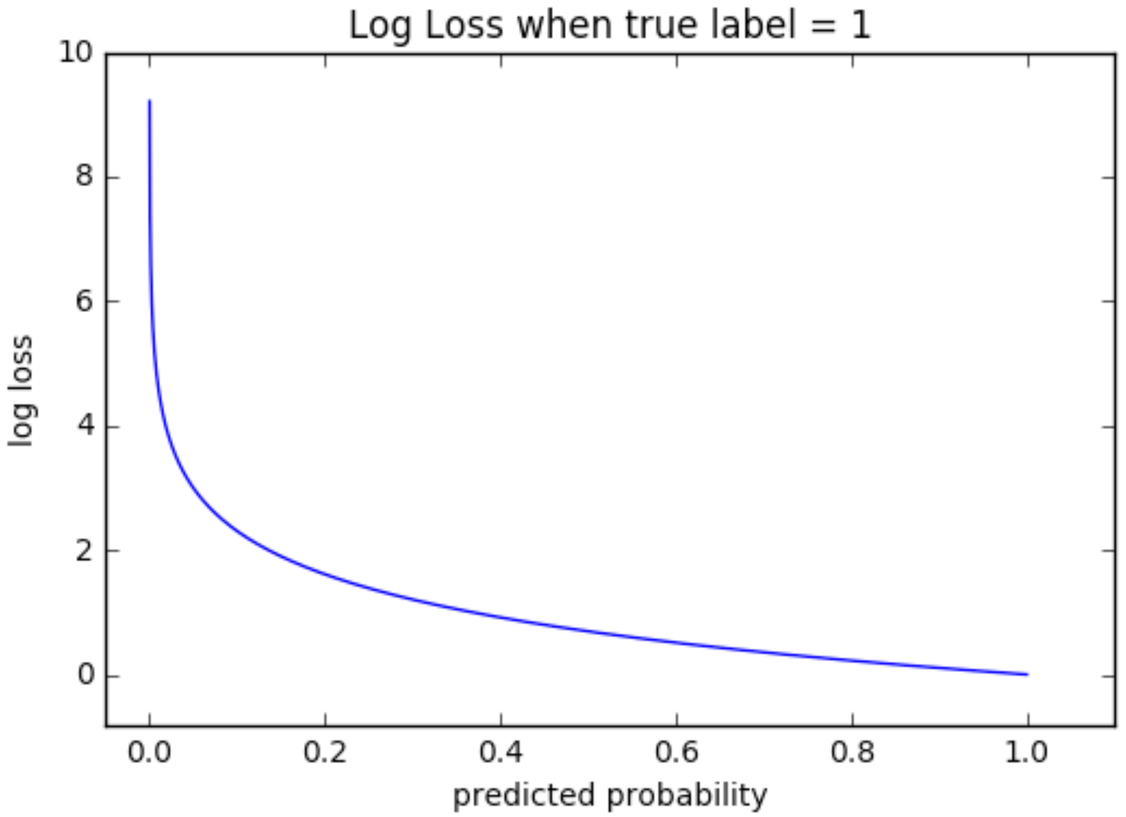
| Network Output | Target Ouput | Error (target-actual) | Error \|target-actual\| | Error $(target-actual)^2$ |
|---|---|---|---|---|
| 0.4 | 0.5 | 0.1 | 0.1 | 0.01 |
| 0.4 | 0.7 | -0.1 | 0.1 | 0.01 |
| 1.0 | 1.0 | 0 | 0 | 0 |
| Sum | | 0 | 0.2 | 0.02 |

# Mean Square Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

# Cross Entropy

- Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.
- This makes it good to use with a sigmoid or softmax.
- Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value.
- A perfect model would have a log loss of 0.



# Gradient Descent!

# ‘Vanilla’ Gradient Descent

Evaluate gradient descent on every sample in data set, then update weights.

# ~~‘Vanilla’~~ Batch Gradient Descent

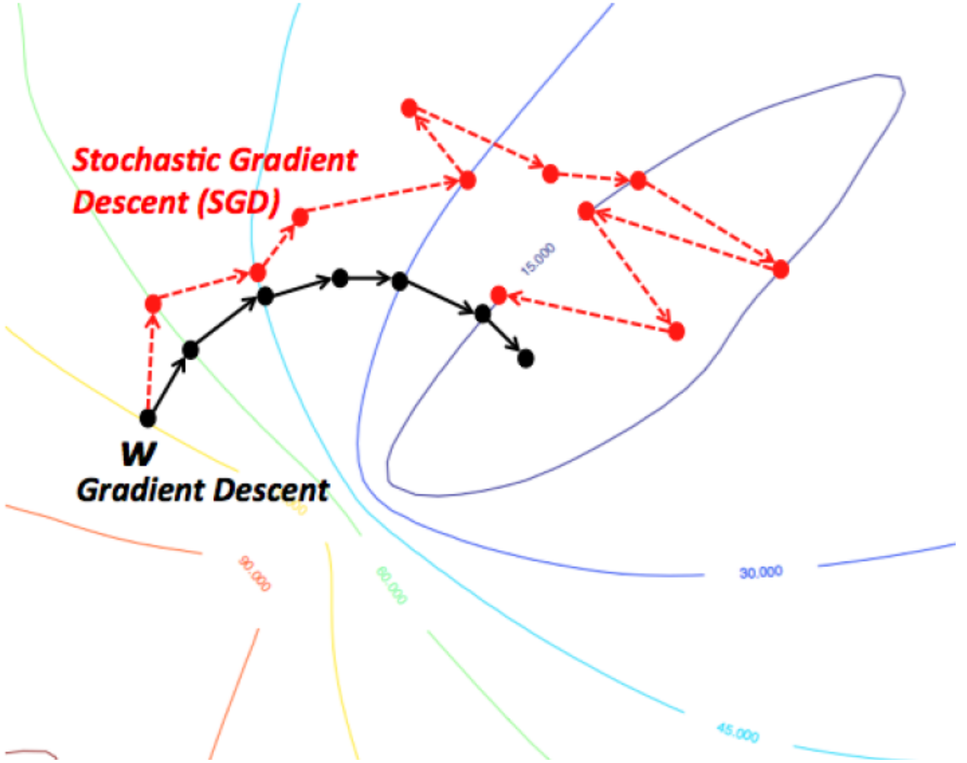
Evaluate gradient descent on every sample in data set, then update weights.

Known as **batch gradient descent**.

Is very slow, many operations, on every sample, and then only one update…

# Stochastic Gradient Descent (SGD)

- Go through each sample individually and calculate the gradient with based on just that particular point.
- Stochastic in the sense that the gradient based on a single training sample is an approximation of the true cost gradient.
- Path tends to zigzag
- Good at escaping local minima
- Typical to shuffle data samples


Stochastic Gradient
Descent (SGD)
W
Gradient Descent
15,000
30,000
45,000
60,000
90,000

# Computation/speed - Batch vs SGD Example

- MNIST digits has 60,000 samples
- Stochastic
    - Go through each sample once, for a total of 60,000
    - Would update weights 60,000 times
- Batch
    - To have the same number of weight updates as above (60,000) we would have to go through each sample 60,000*60,000 times, or 3,600,000,000 times

# Mini Batch Gradient Descent (MB-GD)

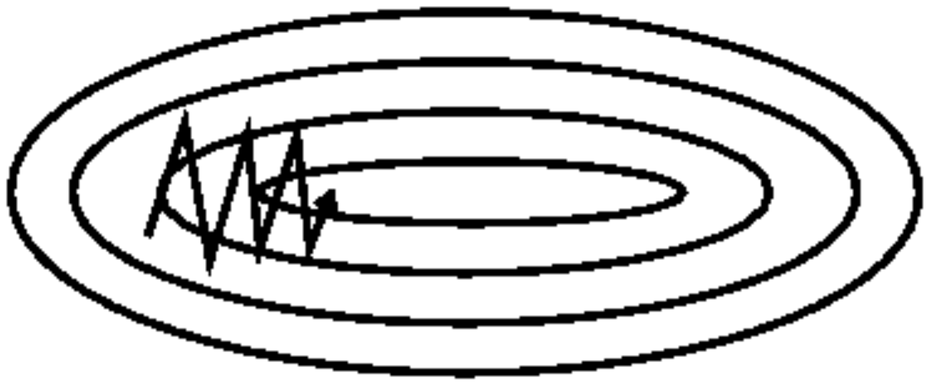
- Most commonly used
- Combination of Batch and SGD
- Divide data set into ***N*** equally sized batches of ***K*** samples
- If ***K*** is 1  you have SGD
- If ***K*** is the size of the data set you have Batch
- Smoother and more stable than SGD
- Faster than Batch

**Inertia Example #1:** Why you need to wear a seatbelt (especially if you are a giraffe)

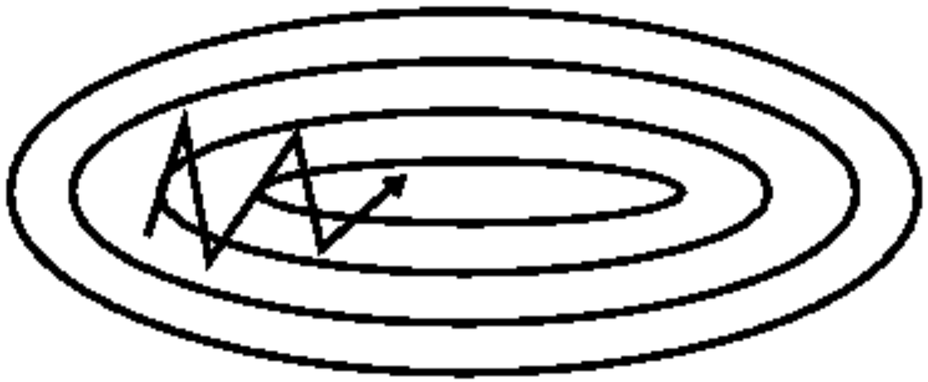
# Momentum

- Weight update has inertia
- Weights gradually adjusted from the rate of the previous update
- Helps escape saddle points and local minima by rolling out from them via speed built up from previous updates
- Helps counteract zig zagging
- Nesterov Accelerated Gradient Descent- momentum variation that tries to approximate where the momentum will be in the next step, helps to anticipate loss surface

SGD no momentum



SGD with momentum



# Adaptive Methods

- Momentum updates weights, but learning rate is still static
- One option is to decay the learning rate at a set rate
- Another option is to adapt the learning rate to each parameter individually

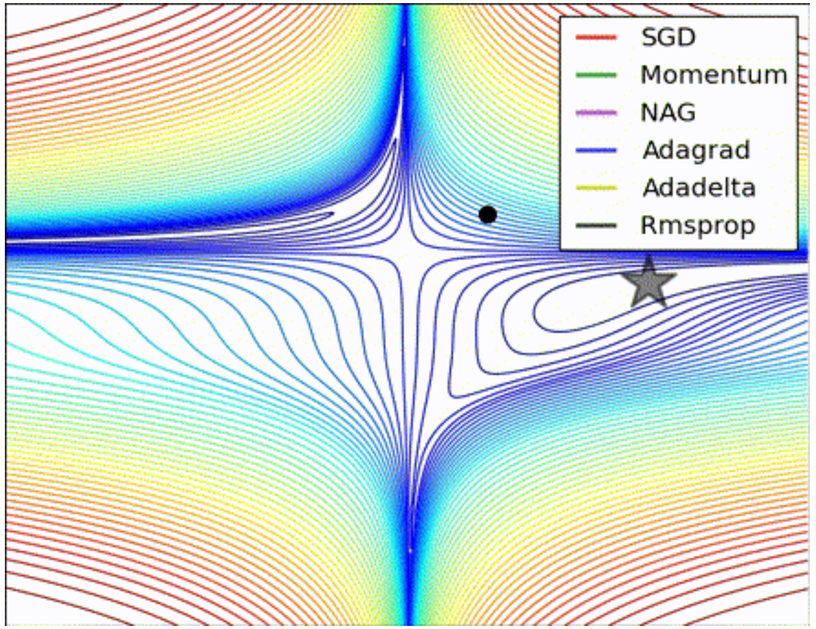
# Adaptive Methods

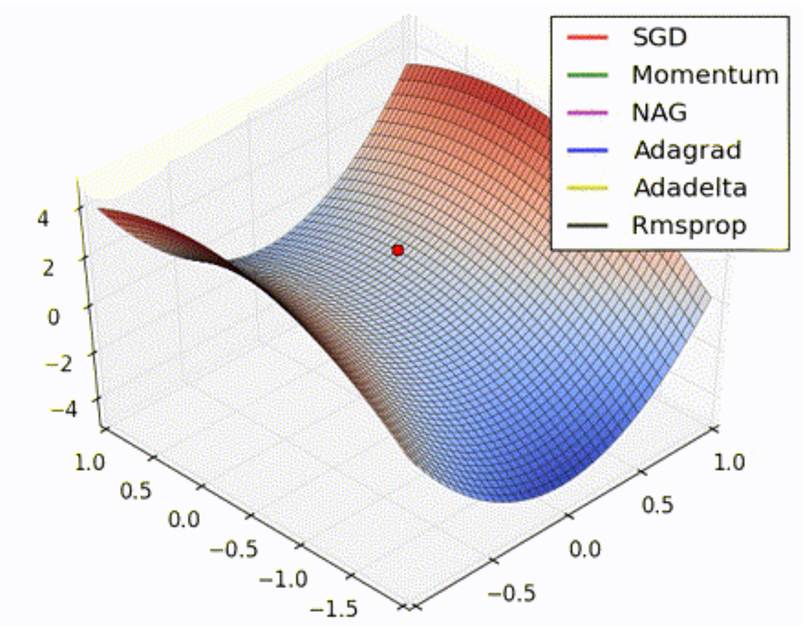
- AdaGrad (Adaptive subGradient): tries to equalize the learning rate between parameters with large gradients and small gradients, meaning slowing down the learning rate for parameters with large gradients and speeding up the learning rate for small gradient parameters.

- AdaDelta is similar to AdaGrad, but restricts the accumulation to recent updates.

- RMSProp is similar to AdaDelta. Both RMSProp and AdaDelta are adaptive in regards to both the parameters and time.

- Adaptive methods are good when data is sparse or unevenly distributed.

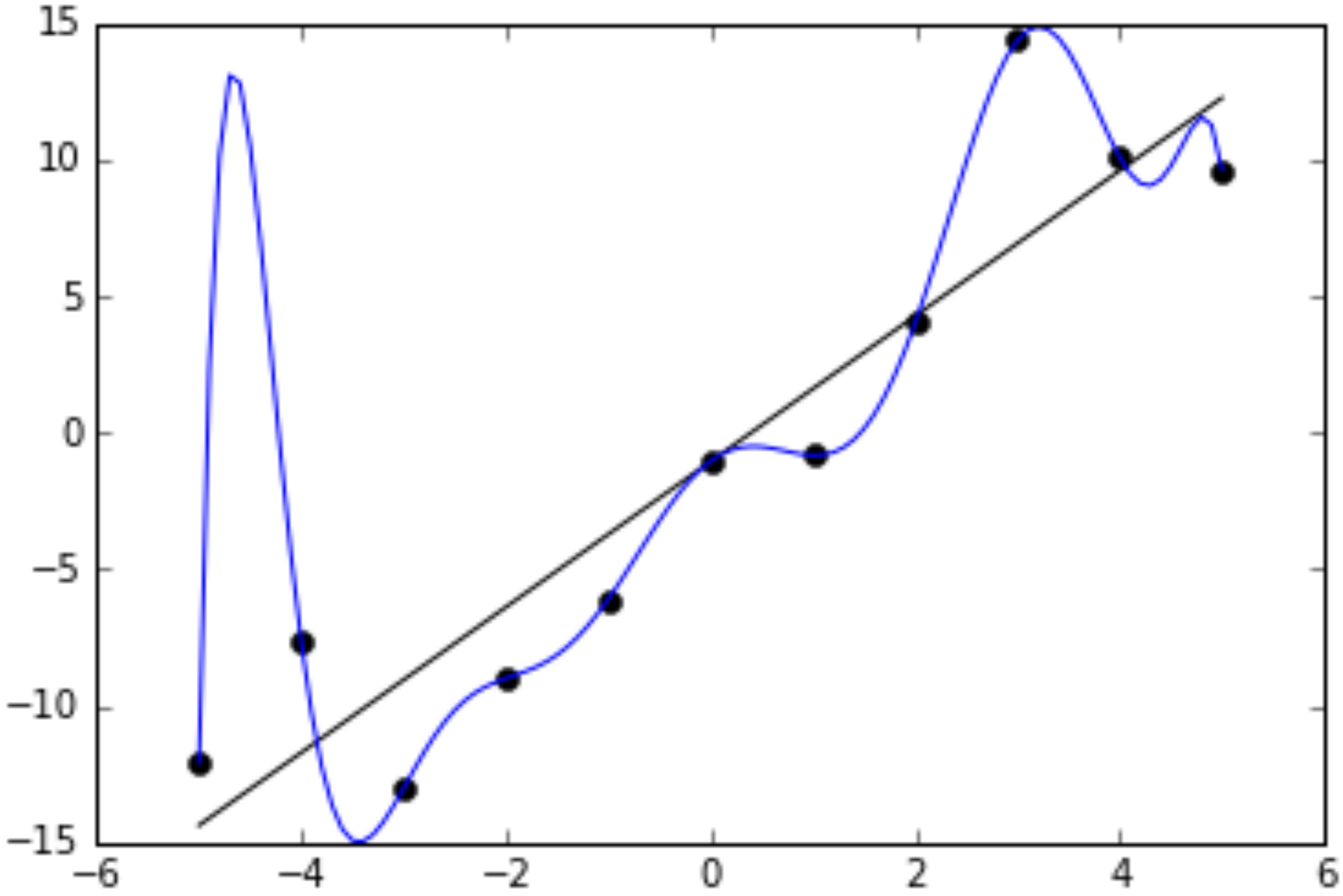

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

# Adam

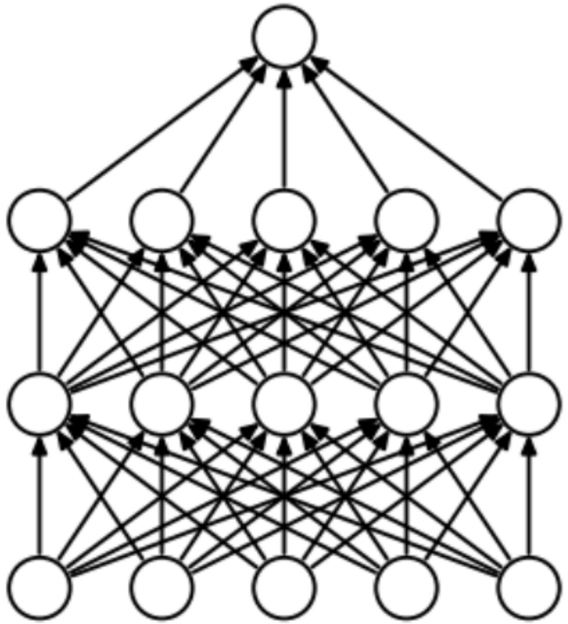
- Stands for: Adaptive Moment Estimation
- Combines both adaptive learning rates & momentum
- Like AdaDelta and RMSProp uses time to help determine new learning rate

# Overfitting

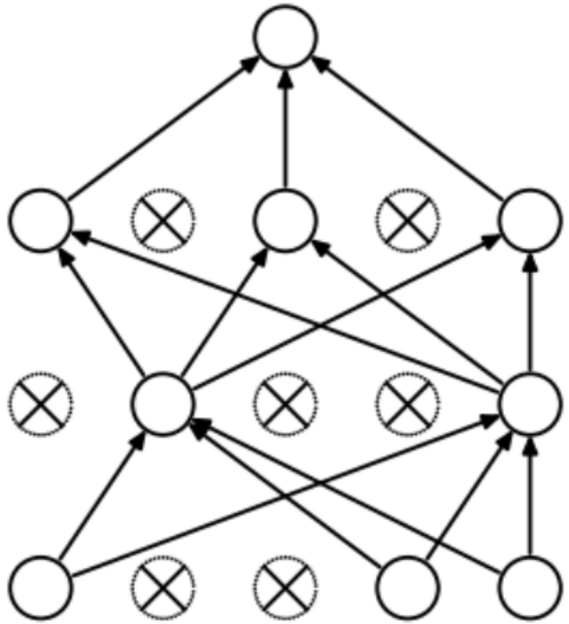



gifbox.ru

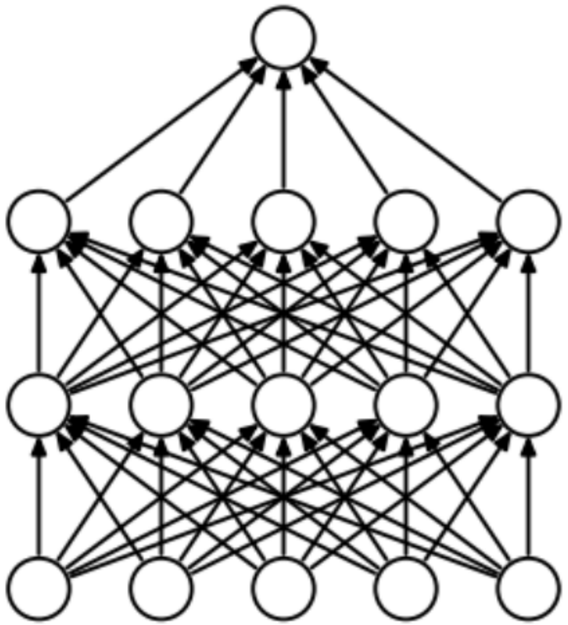
# Dropout



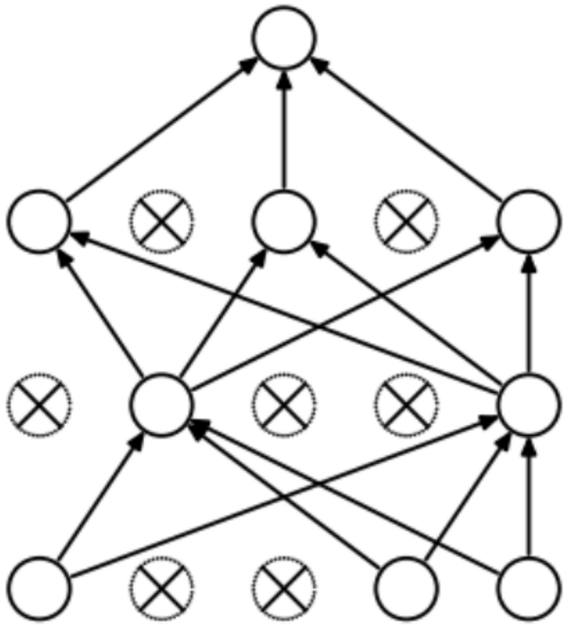
(a) Standard Neural Net



(b) After applying dropout.



(a) Standard Neural Net



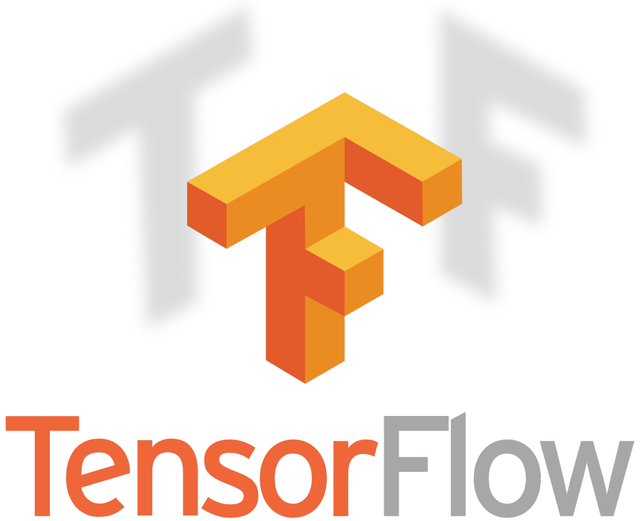
(b) After applying dropout.

- The dropped neurons are continuously shuffled during training.
- This help the network not to depend on some neurons.
- This creates a more balanced representation, and helps combat overfitting.

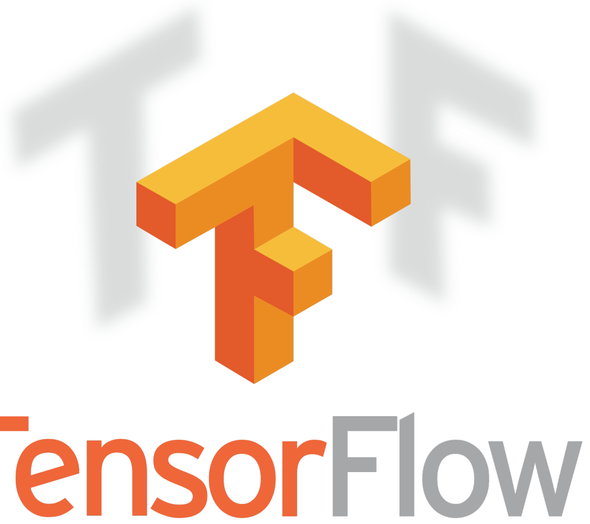
# Frameworks


theano
Caffe
?
TensorFlow
PYTORCH
K


PYTORCH


TensorFlow

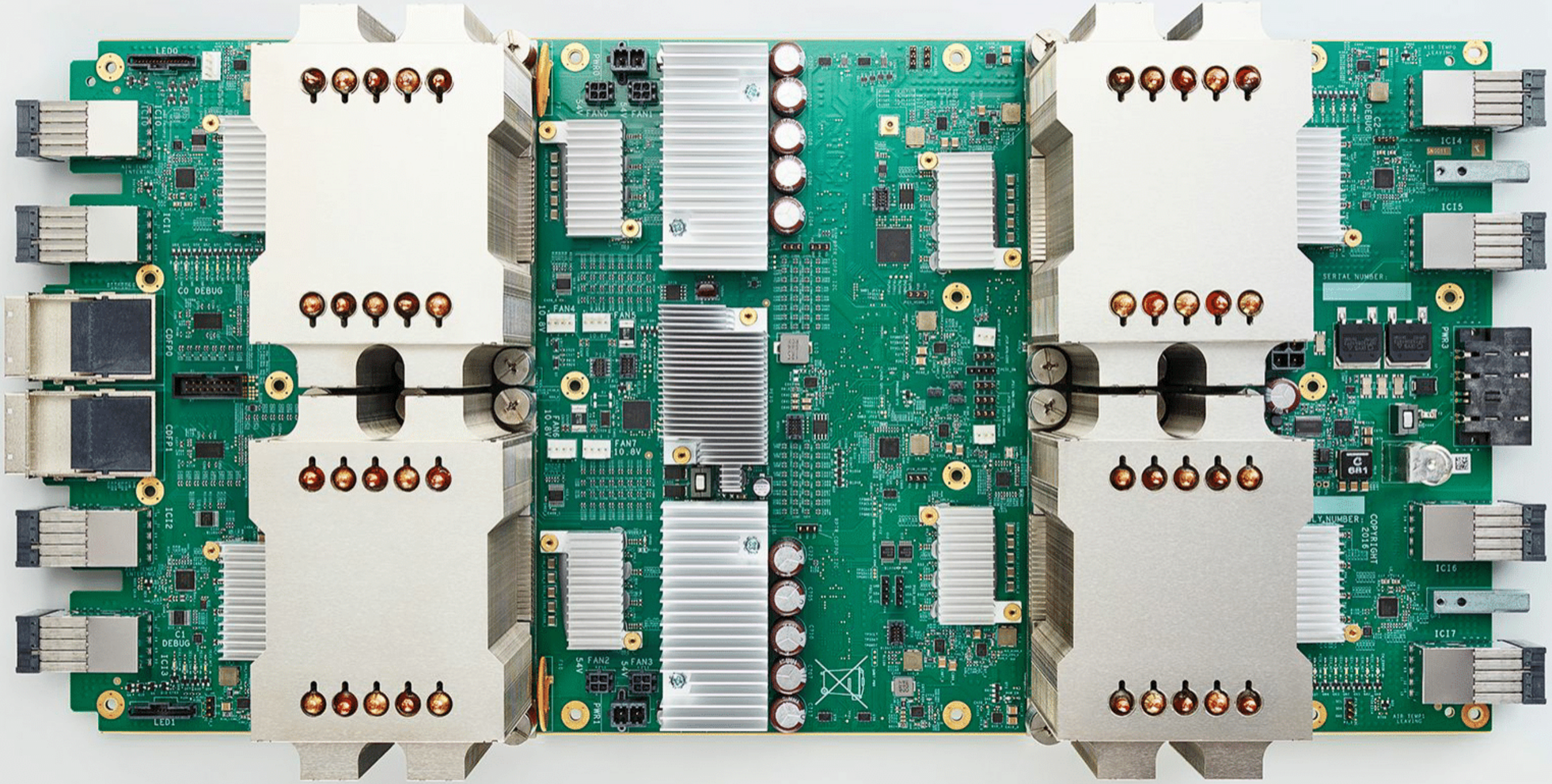

PYTORCH


TensorFlow


K

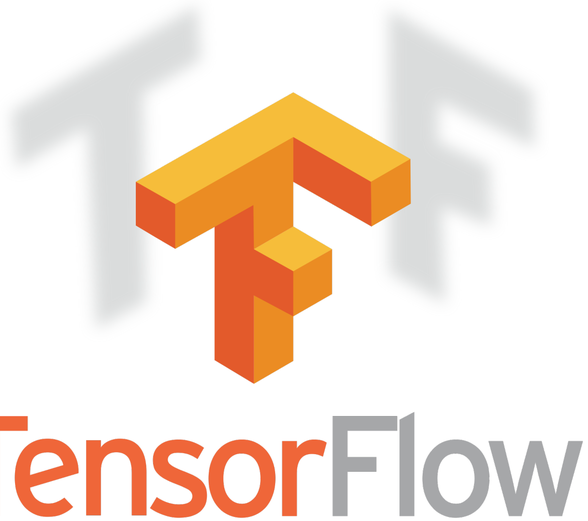
# Static vs Dynamic

# Tensor Processing Unit v2




Google-designed device for neural net **training** and **inference**

Deployable to web



TensorFlow

# Tensor

A tensor is a generalization of vectors and matrices to potentially higher dimensions.

| Rank | Math entity |
| --- | --- |
| 0 | Scalar (magnitude only) |
| 1 | Vector (magnitude and direction) |
| 2 | Matrix (table of numbers) |
| 3 | 3-Tensor (cube of numbers) |
| n | n-Tensor (you get the idea) |