Learning Systems 2018:
Lecture 5 – Activations, Loss

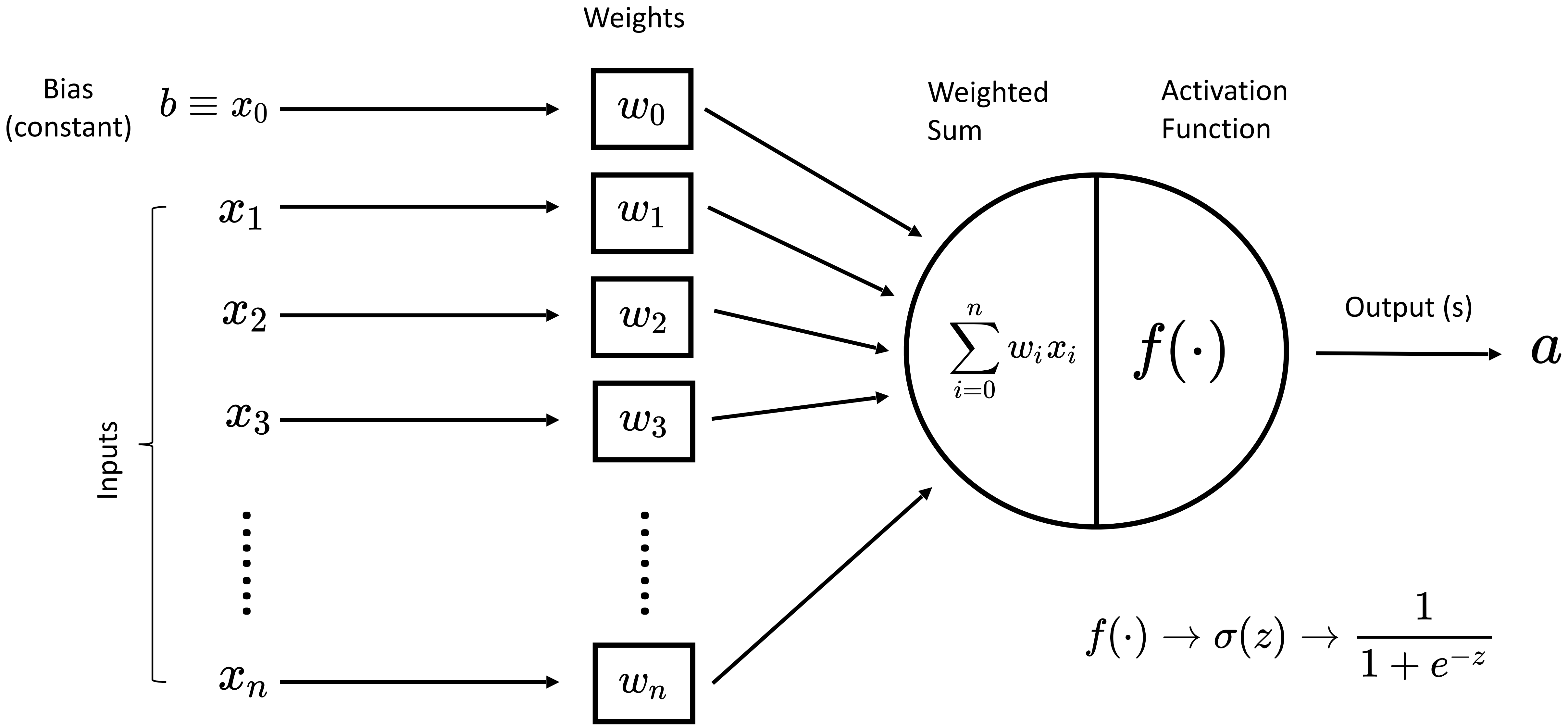
Ian Foster and Rick Stevens
Argonne National Laboratory
The University of Chicago

Crescat scientia; vita excolatur

Topics

- Basic Model
- Activations
- Loss Functions
- Math Preliminaries
- Review Gradient Descent
- Optimizers
- Back Propagation
- Implications of the Key Equations
- Implementation Issues

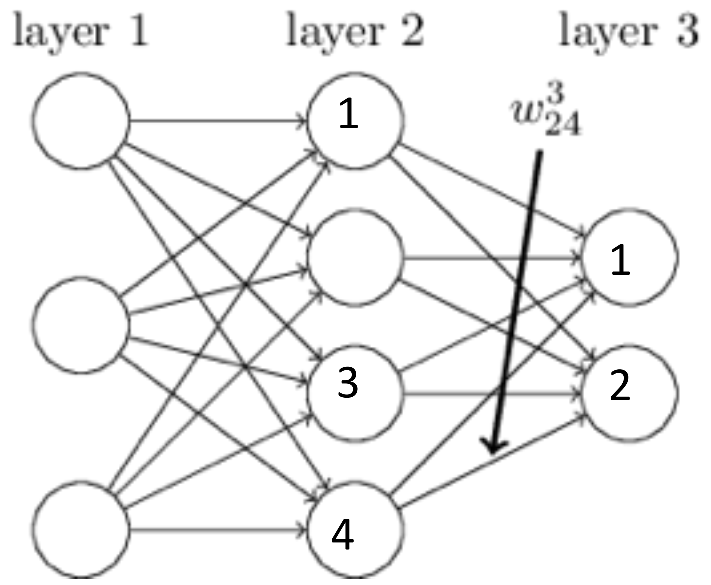
Some materials in this lecture are adapted from a talk by Javier Ruiz Hidalgo and from a blog by Michael Nielson



$$f(\cdot) \rightarrow \sigma(z) \rightarrow \frac{1}{1+e^{-z}}$$

Convention for naming things

$(l-2)^{th}$ $(l-1)^{th}$ $(l)^{th}$



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

a^l_j Activation of neuron j in layer l

w^l_{jk} Weight

l == to ($\longrightarrow$) layer

j == to ($\longrightarrow$) neuron

k == from ($\longleftarrow$) neuron

With these notations, the activation a^l_j of the j^{th} neuron in the l^{th} layer is related to the activations in the $(l-1)^{th}$ layer by the equation

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{(l-1)}_k + b^l_j\right)$$

To rewrite this expression in a matrix form we define a *weight matrix* w^l for each layer, l. The entries of the weight matrix w^l are just the weights connecting to the l^{th} layer of neurons, that is, the entry in the j^{th} row and k^{th} column is w^l_{jk} . We can simplify our vector notation a bit by dropping the explicit bias term.

$$a^l = \sigma\left(w^l a^{(l-1)} + b^l\right) \rightarrow a^l = \sigma\left(w^l a^{(l-1)}\right)$$

Note that *bias is also known as threshold*, which will now make more sense to you.

Say bank approves credit if $(\sum_{i=1}^{d} WiXi > threshold)$ and deny if it is lesser. So this linear formula can be written as a hypothesis which is:

$$h(x) = (\sum_{i=1}^{d} WiXi - threshold)$$

Suppose $threshold = -W0$, above equation can be rewritten as

$$h(x) = sign((\sum_{i=1}^{d} WiXi) + W0)$$

Introducing X0=1 in the equation.

$$h(x) = sign((\sum_{i=1}^{d} WiXi) + W0X0)$$

Now we can simply write the hypothesis equation as

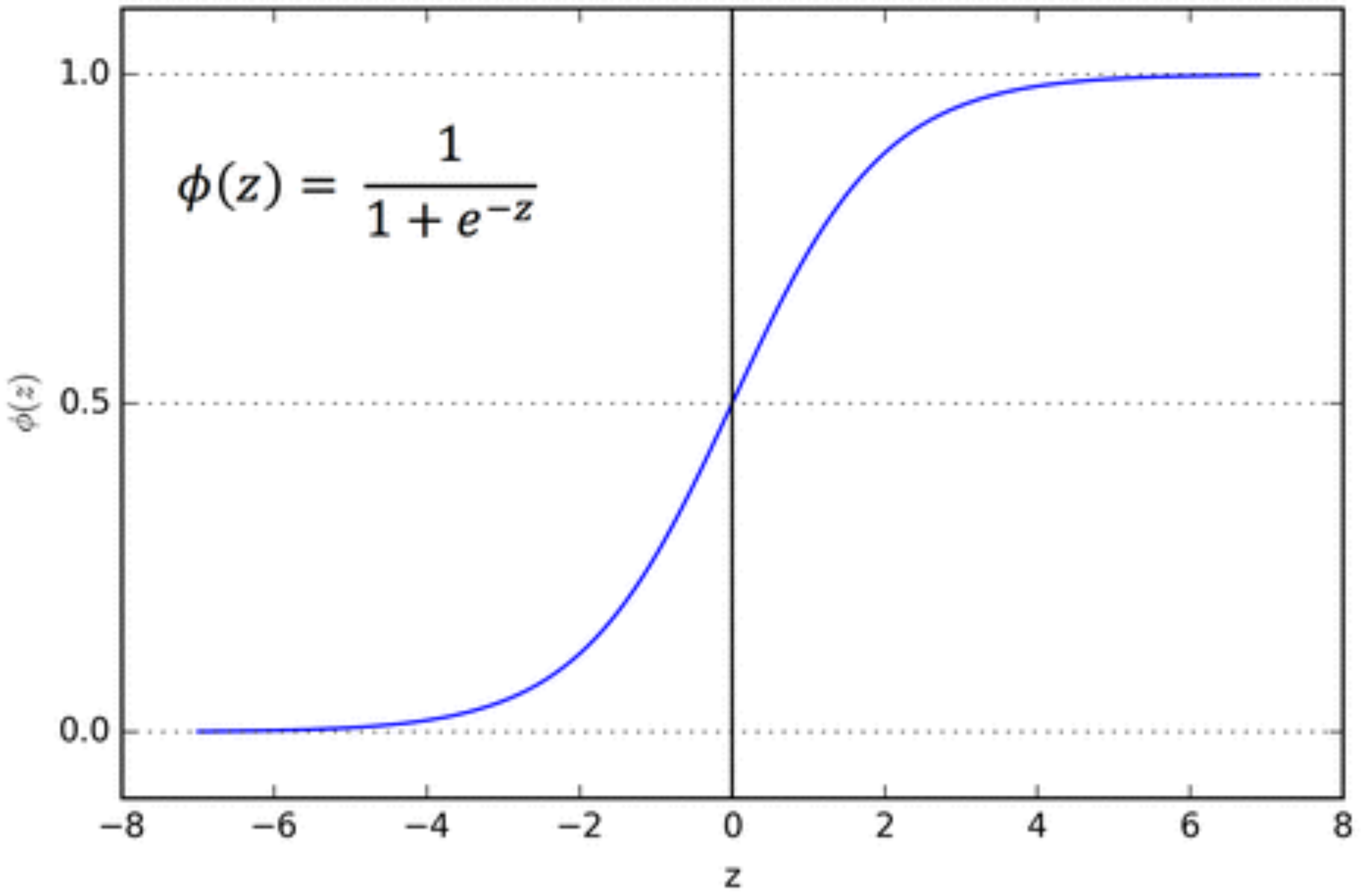
$$h(x) = sign(\sum_{i=0}^{d} WiXi).$$

This is the standard form. The vector form is $h(x) = sign(W'X)$.

Activations

- **Originally "Squash" functions** – compressing values into a given interval
- **Differentiable** – we can compute the derivative
- **Monotonic** – entirely increasing or decreasing
- **Simple** – fast to compute

Sigmoid

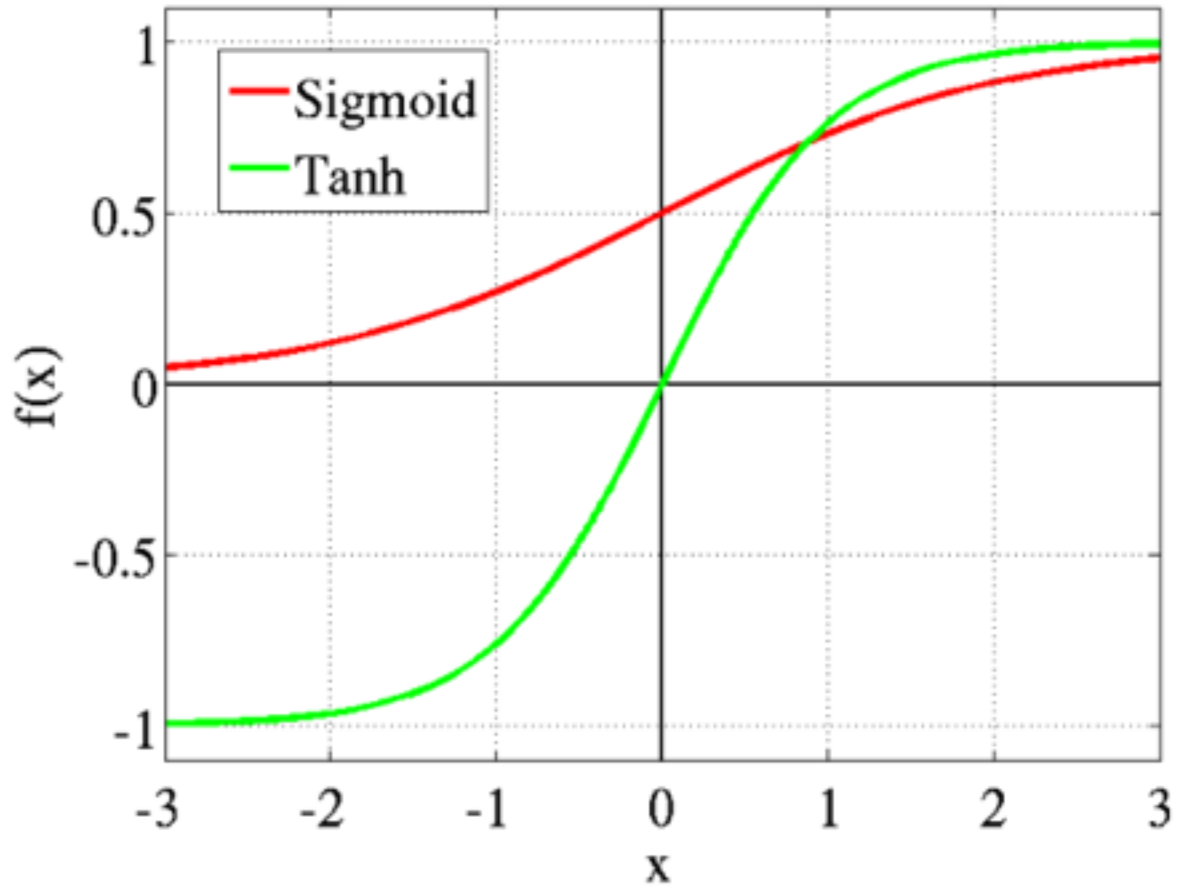


Sigmoid

- The main reason why we use sigmoid function is because it exists between **(0 to 1).** Therefore, it is especially used for models where we have to **predict the probability** as an output. Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice
- The function is **differentiable**. That means, we can find the slope of the sigmoid curve at any point.
- The function is **monotonic** but the function's derivative is not.
- The logistic sigmoid function can cause a neural network to get stuck during training time
- The **softmax function** is a more generalized logistic activation function which is used for multiclass classification

$$\boxed{\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}}$$

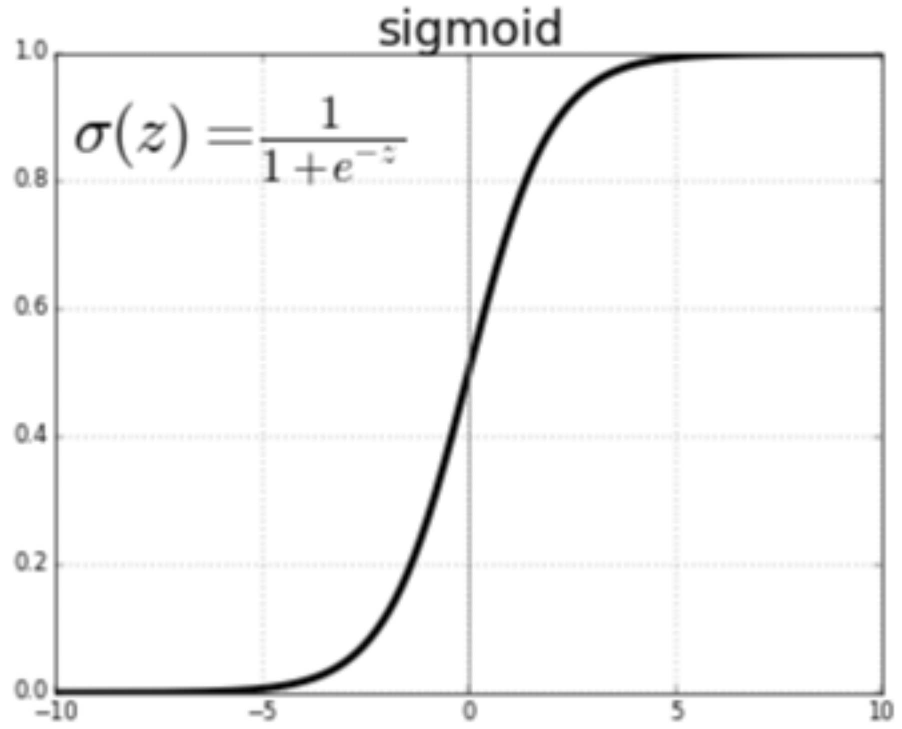
Tanh (hyperbolic tangent)

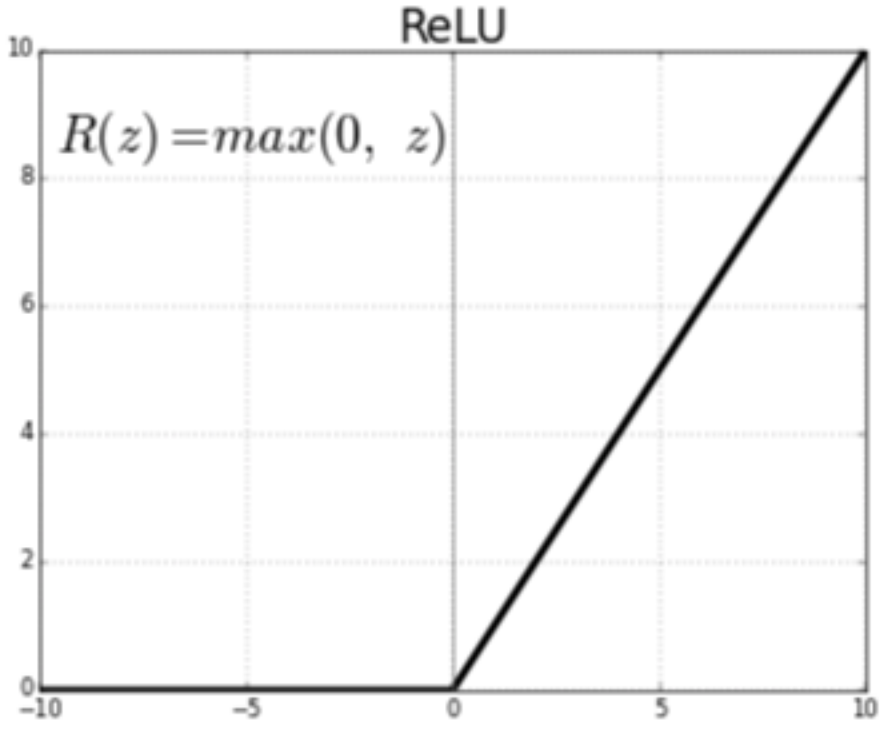


Tanh (hyperbolic tangent)

- Tanh is also like sigmoid but with -1 to 1 range
- Tanh is also sigmoidal (s - shaped)
- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph
- The function is **differentiable**
- The function is **monotonic** while its **derivative is not monotonic**
- The tanh function is mainly used in classification between two classes
- Both tanh and logistic sigmoid activation functions are used in feed-forward nets

ReLU (Rectified Linear Unit)

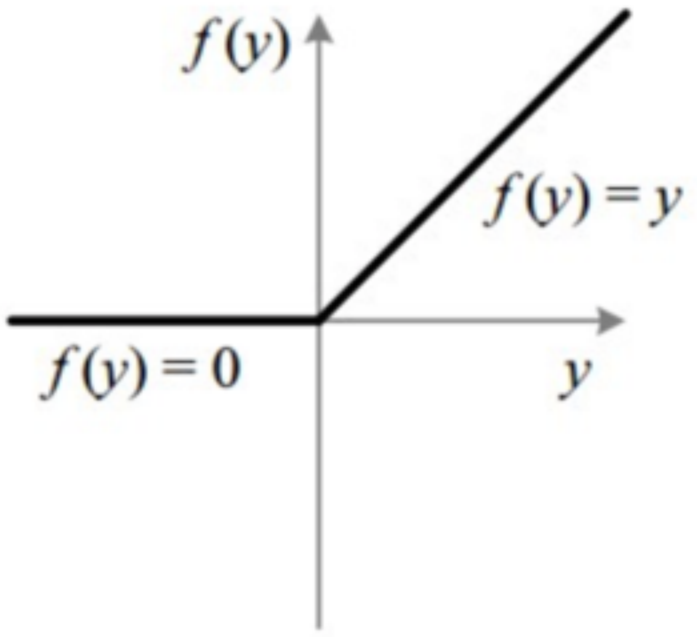


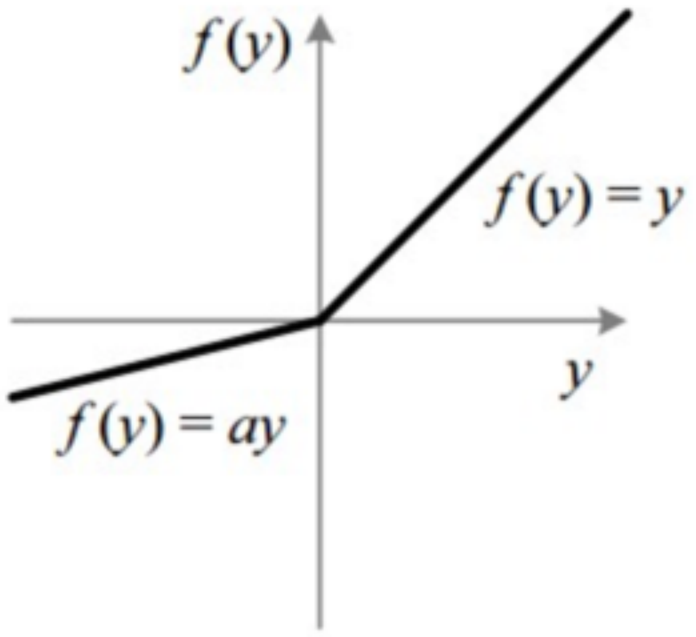


ReLU (Rectified Linear Unit)

- The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks in deep learning
- As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.
- **Range:** [0 to infinity)
- The function and its derivative **both are monotonic**.
- But the issue is that all the negative values become zero immediately which can limit the ability of the model to fit or train from the data
- Negative input values given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

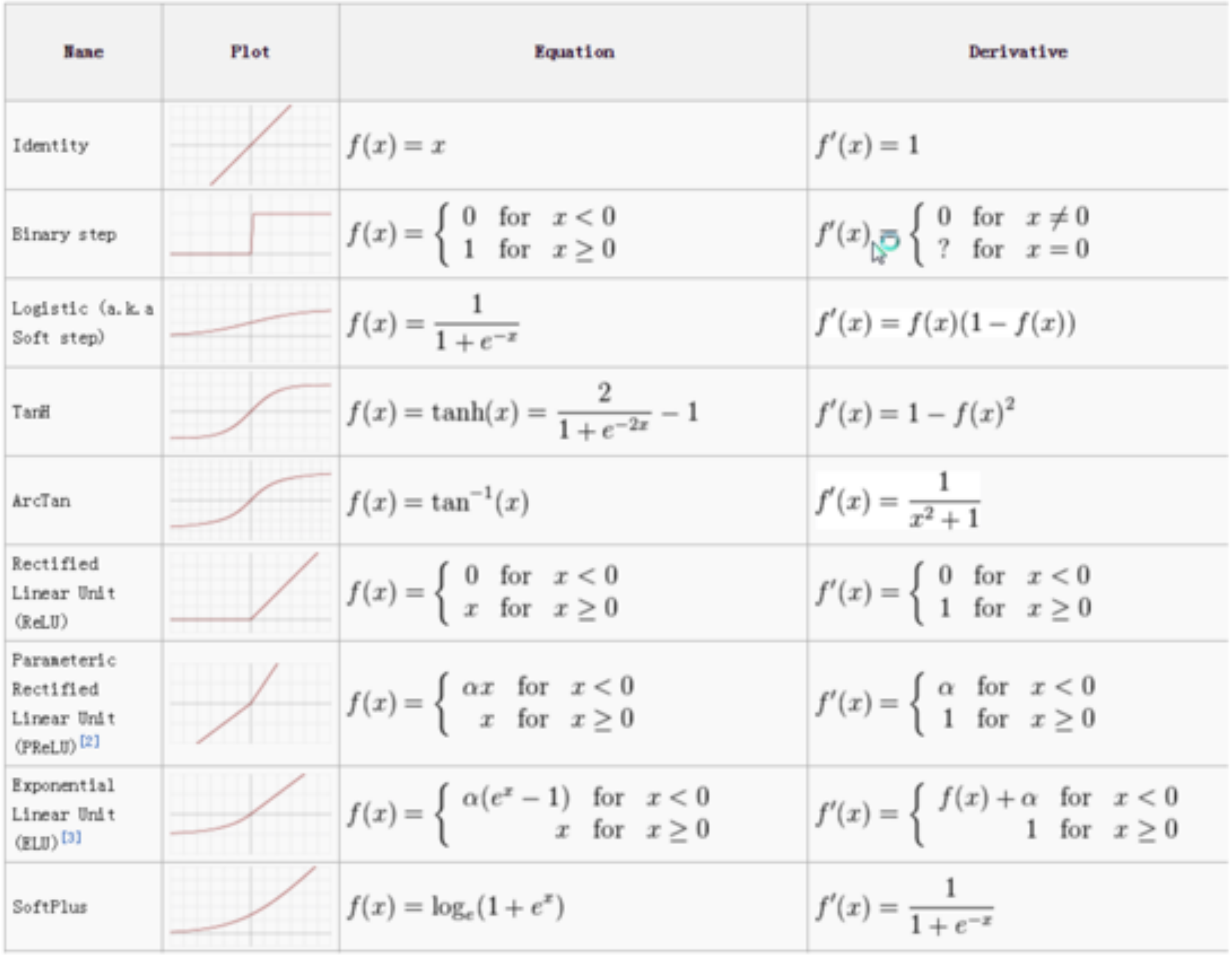
Leaky ReLU

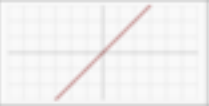
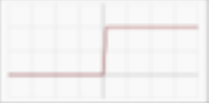
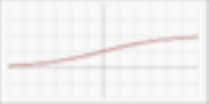
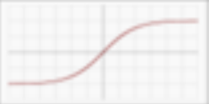
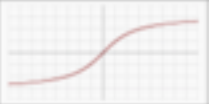
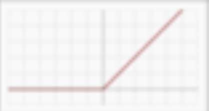
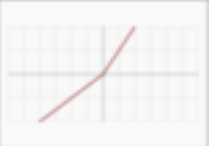
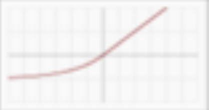
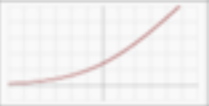


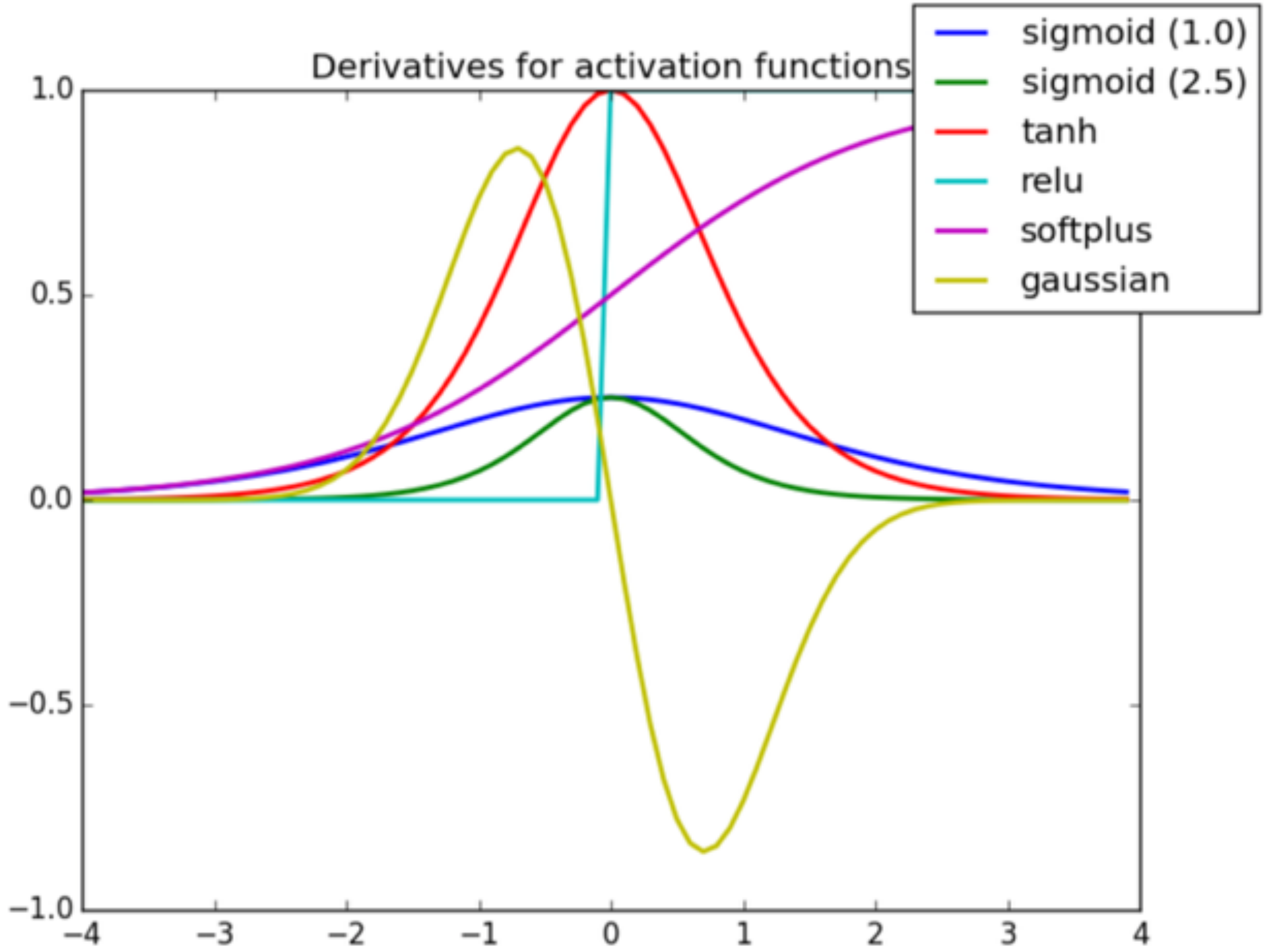


Leaky ReLU

- It is an attempt to solve the ReLU “zero” problem
- Can you see the Leak? 😆
- The leak helps to increase the range of the ReLU function. Usually, the value of **a** is 0.01 or so
- When **a is not 0.01** then it is called **Randomized ReLU**
- Therefore the **range** of the Leaky ReLU is (-infinity to infinity)
- Both Leaky and Randomized ReLU functions are monotonic, also, their derivatives also monotonic

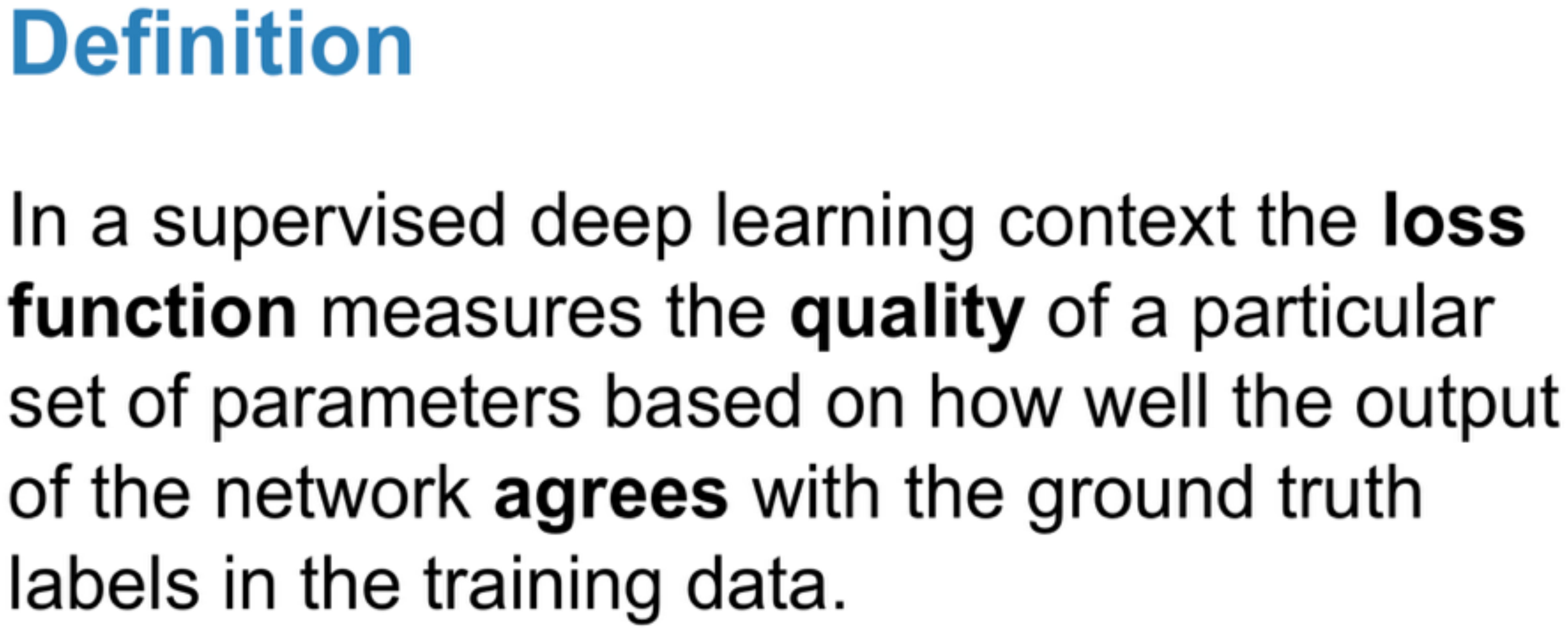


Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
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		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$



Derivatives for activation functions
sigmoid (1.0)
sigmoid (2.5)
tanh
relu
softplus
gaussian
1.0
0.5
0.0
−0.5
−1.0
−4
−3
−2
−1
0
1
2
3
4

Loss Functions



Definition

In a supervised deep learning context the **loss function** measures the **quality** of a particular set of parameters based on how well the output of the network **agrees** with the ground truth labels in the training data.

Nomenclature

loss function

=

cost function

=

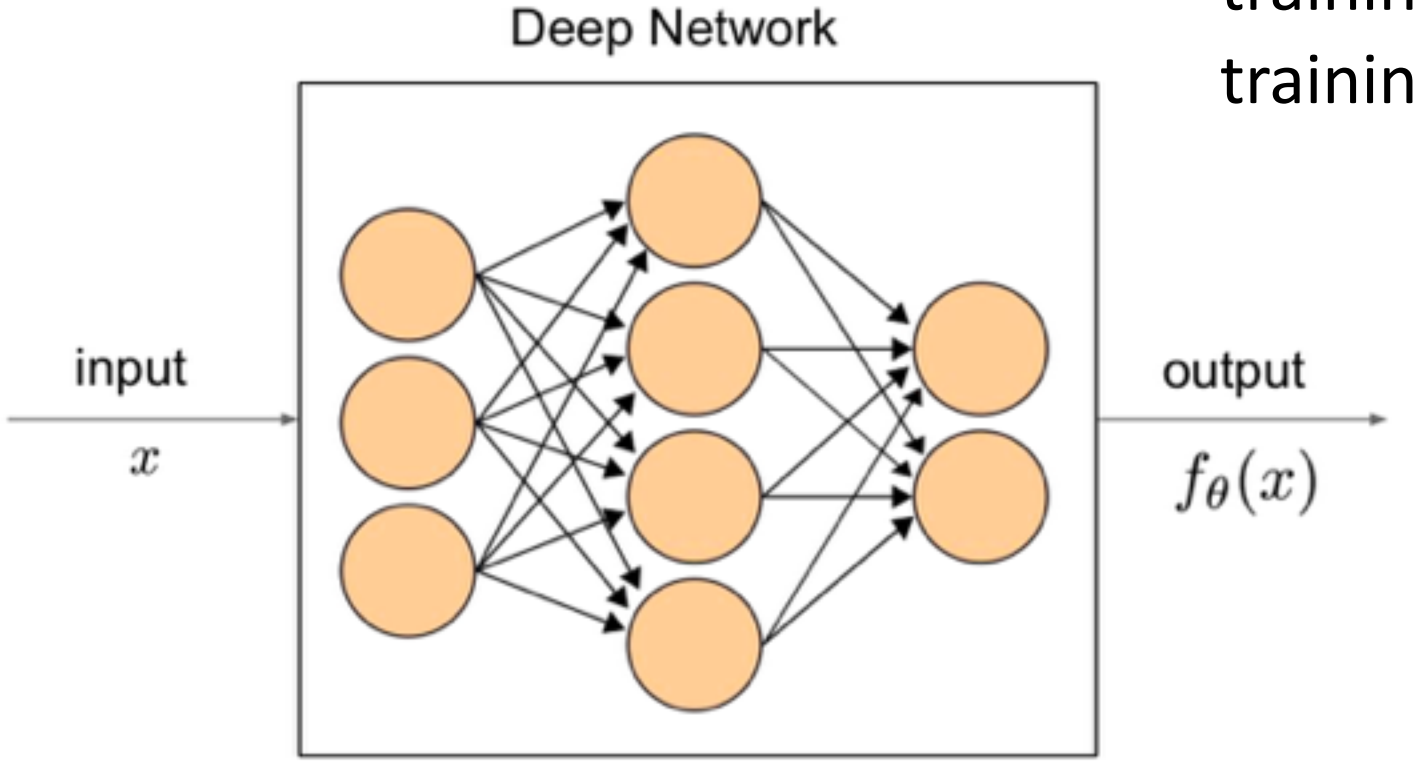
objective function

=

error function

Loss function

How close is the prediction of the model (trained from the training data) when compared to training data?



labels (ground truth)

input

$$\mathcal{L}(w) = distance(f_\theta(x), y)$$

error

parameters (weights, biases)

Training Loss vs Validation Loss (or Test Loss)

- The loss function is used to estimate the distance between the predictions the model makes and the labels in the training data
- This estimated distance (between what we are predicting and what the right answer is) or error is what will be used to improve the model

- The same function can be applied to compare the predictions of the model to labels in data not used to train the model (i.e. validation or testing data). This estimate is not used to improve the model directly and is referred to as "validation loss"
- Other metrics (perhaps more understandable to humans) might also be used to compare model predictions to validation data

Definition of Loss (MSE)

$$L(w,b) \equiv \frac{1}{2n} \sum_x \| y(x) - a \|^2$$

$y(x)$	The ground truth label for each sample x
a	The activation of the last layer for sample x

Loss Definition

$$L(w) \equiv \frac{1}{2n} \sum_{x} \| y - \hat{y} \|^2$$

y is the label for sample *x* and *y* (hat) is the prediction for sample *x*

we also collapse the bias into the weights for simplicity of presentation

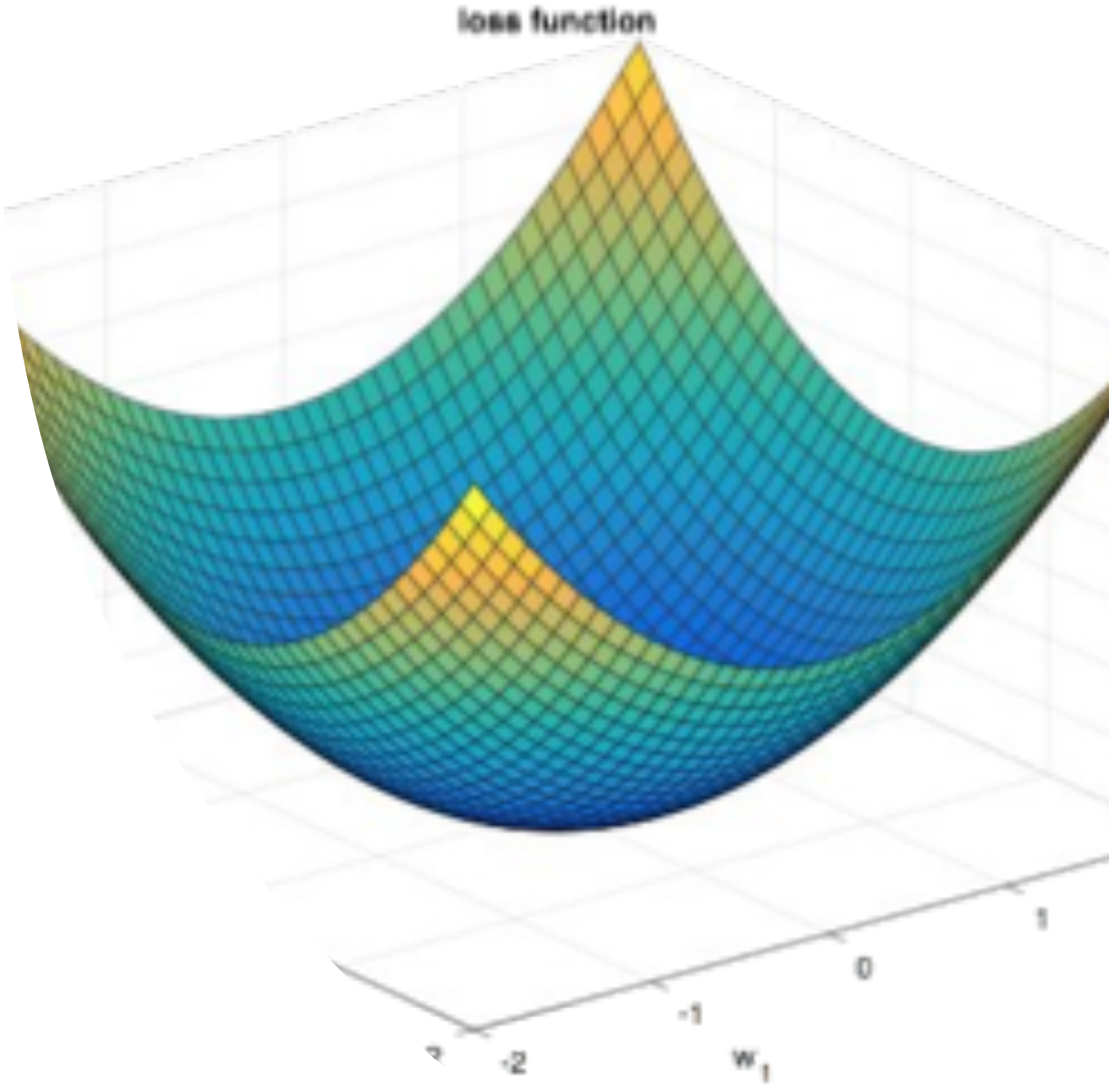
Overall Loss is Mean Loss over Samples

$$L = \frac{1}{n} \sum_{x} L_x$$

We simplify our notation to just refer to Loss and Loss for a specific sample

Properties we need in Loss Function

- Minimum (0 value) when the output of the network is equal to the ground truth data
- Increase value when output differs from ground truth
- Ideally $\Longrightarrow$ convex function
- In reality $\Longrightarrow$ (order millions of parameters and not convex)
- Varies smooth with changes in the output
    - Gradients for gradient descent
    - Easy to compute small changes in parameters to get an improvement in loss



Assumptions

- For **backpropagation** to work:
  - Loss function can be written as an average over loss functions for individual training examples:

    empirical risk $\longrightarrow$
    $$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_i$$

  - Loss functions can be written as a function of the output activations from the neural network.

Loss Functions depend on task type

- **Regression:** predict continuous numeric values
  - Absolute Error, Squared Error, etc.

- **Classification:** predict categorical values
  - Hinge, Cross-entropy, etc.

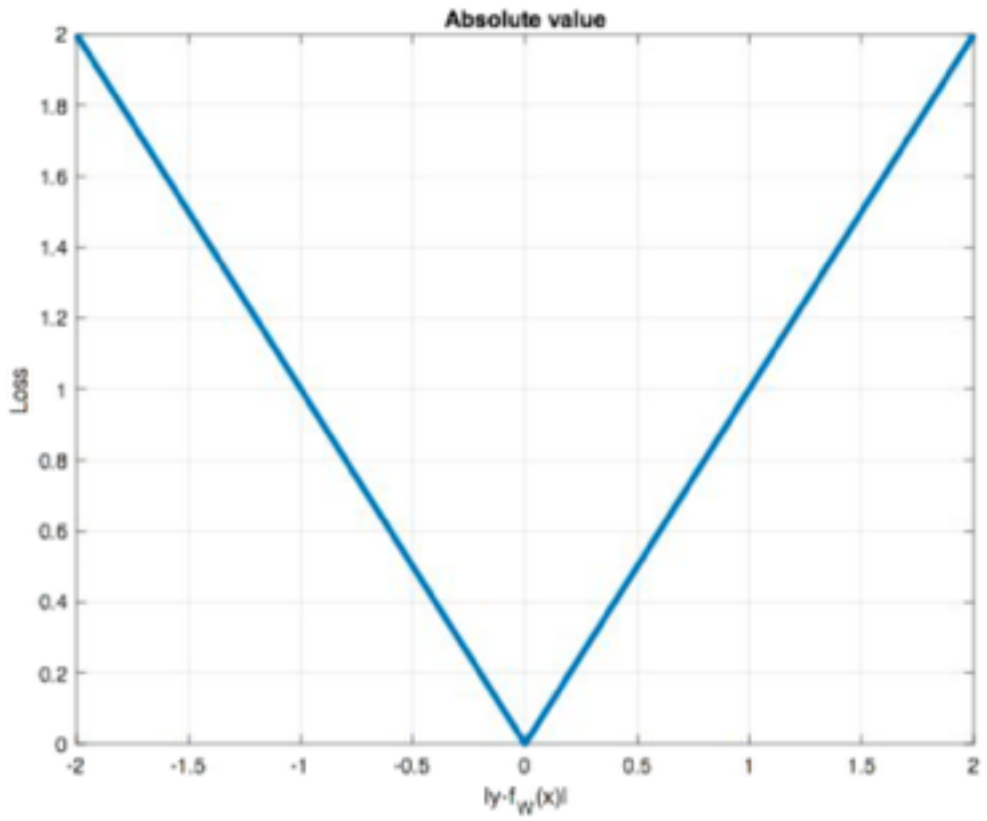
Absolute value, L1-norm

aka MAE

- Very intuitive loss function
  - produces sparser solutions
    - good in high dimensional spaces
    - prediction speed
  - less sensitive to outliers

Input Label Our Prediction

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} |y_i - f_\theta(x_i)|$$

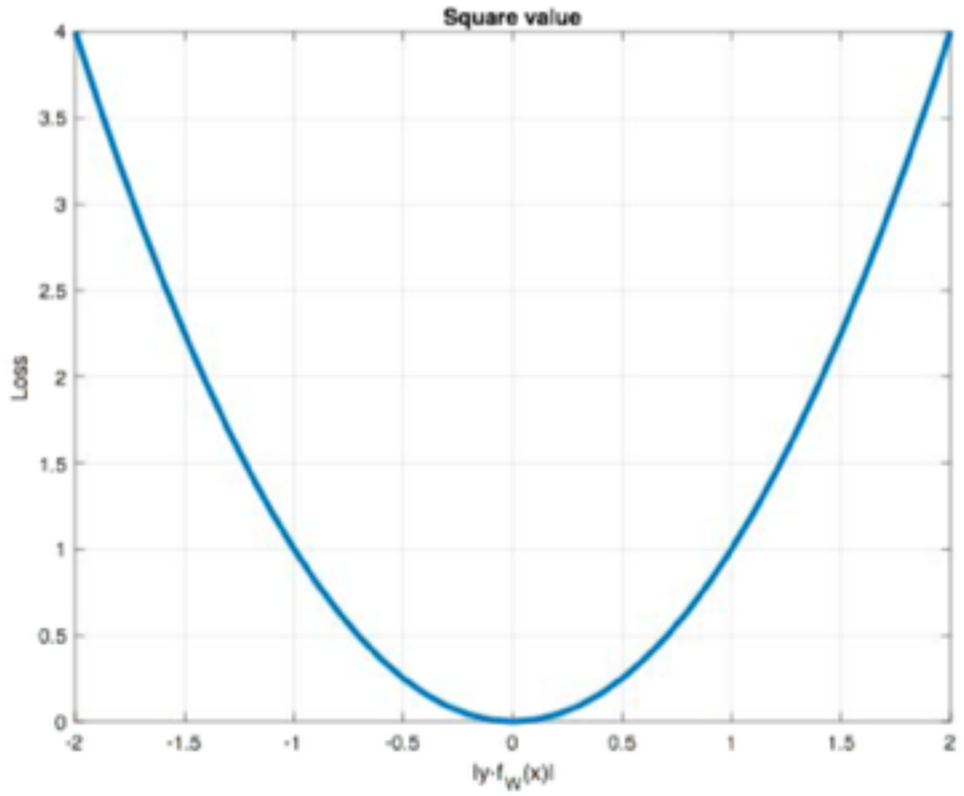


Square error, Euclidean loss, L2-norm

aka MSE

- Very common loss function
  - More precise and better than L1-norm
  - Penalizes large errors more strongly
  - Sensitive to outliers

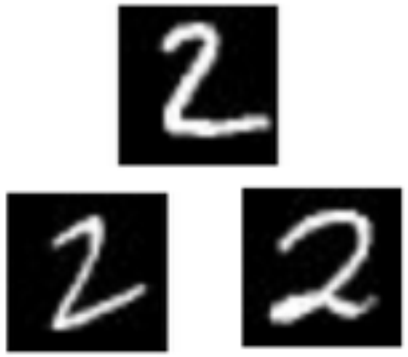
$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$



Classification (1)

We want the network to classify the input into a fixed number of classes

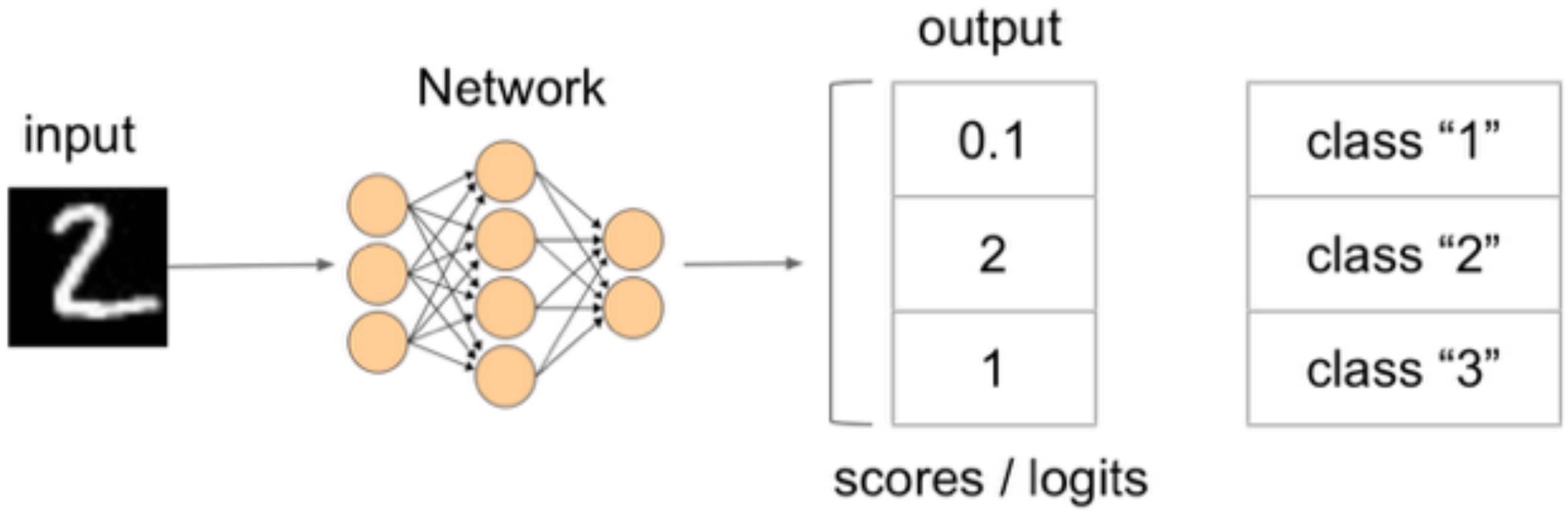
class “1”


class “2”

class “3”

Classification (2)

- Each input can have only one label
  - One prediction per output class
    - The network will have “k” outputs (number of classes)

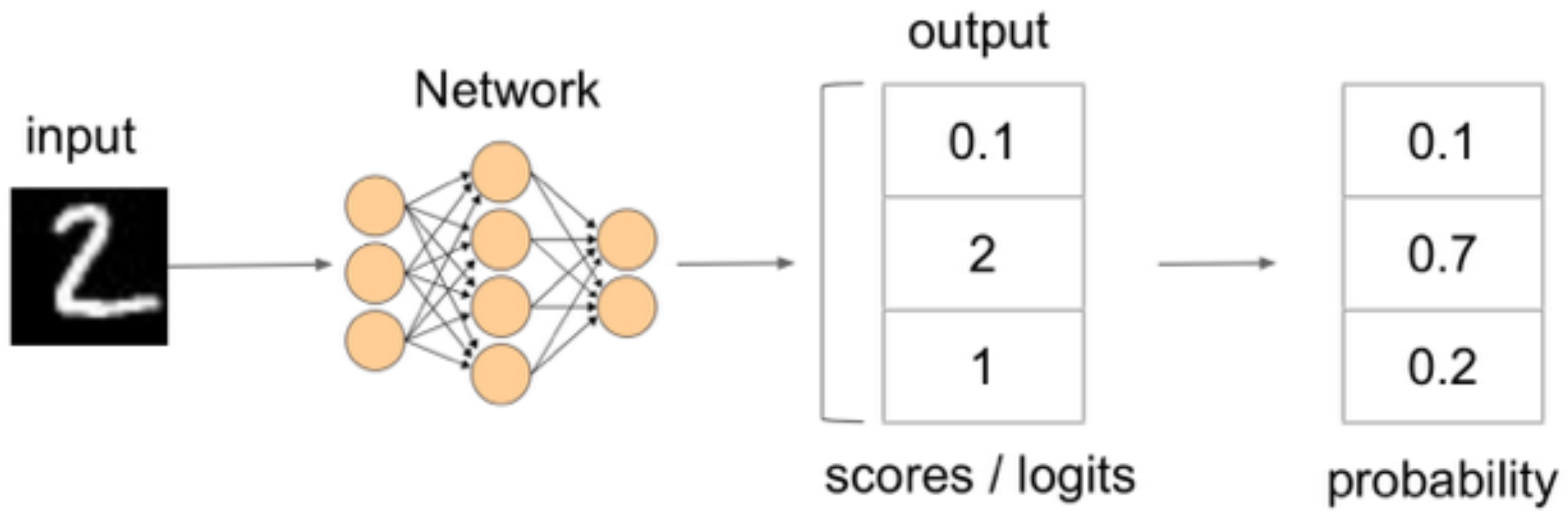


Classification (3)

- How can we create a loss function to improve the scores?
  - Somehow write the labels (ground truth of the data) into a vector → One-hot encoding
  - Non-probabilistic interpretation → **hinge loss**
  - Probabilistic interpretation: need to transform the scores into a probability function → Softmax

Softmax (1)

- Convert scores into probabilities
  - From 0.0 to 1.0
  - Probability for all classes adds to 1.0

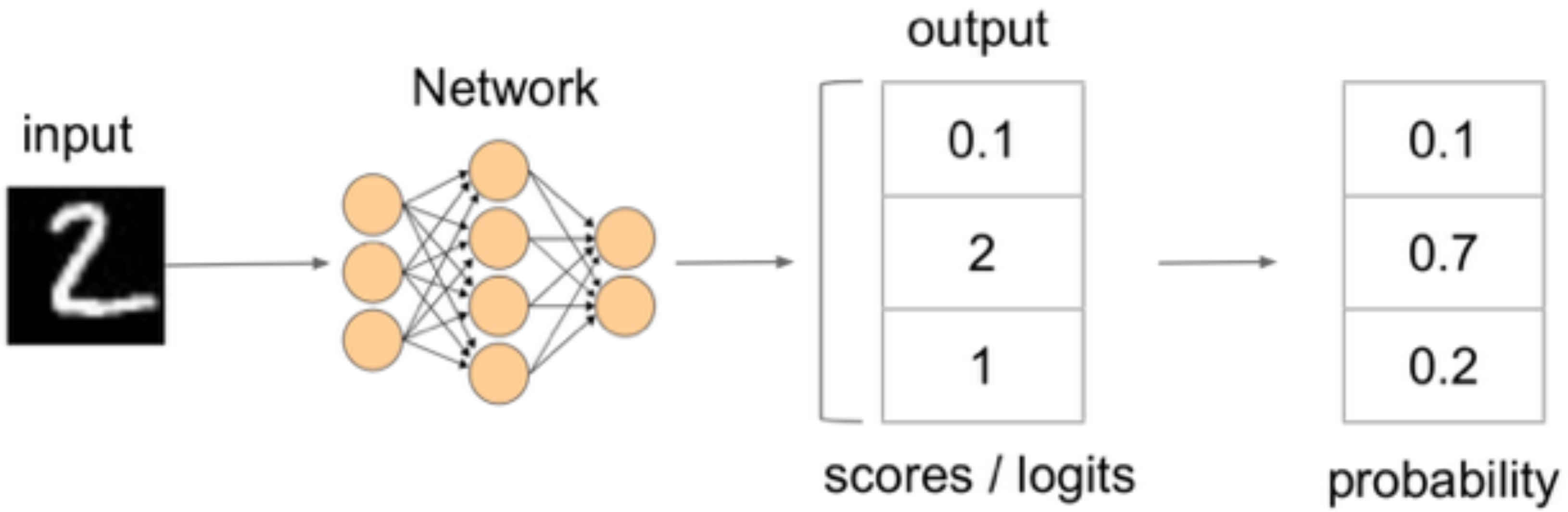


Softmax (2)

- Softmax function

scores (logits)

$$S(l_i) = \frac{e^{l_i}}{\sum_k e^{l_k}}$$

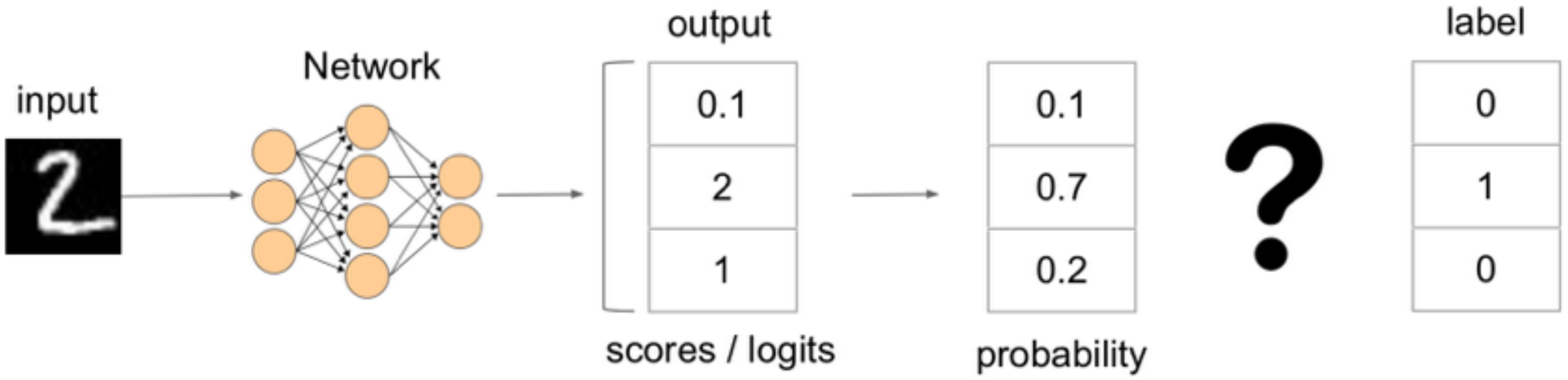


One-hot encoding

- Transform each label into a vector (with only 1 and 0)
  - Length equal to the total number of classes “k”
  - Value of 1 for the correct class and 0 elsewhere

class “1”	class “2”	class “3”
1	0	0
0	1	0
0	0	1

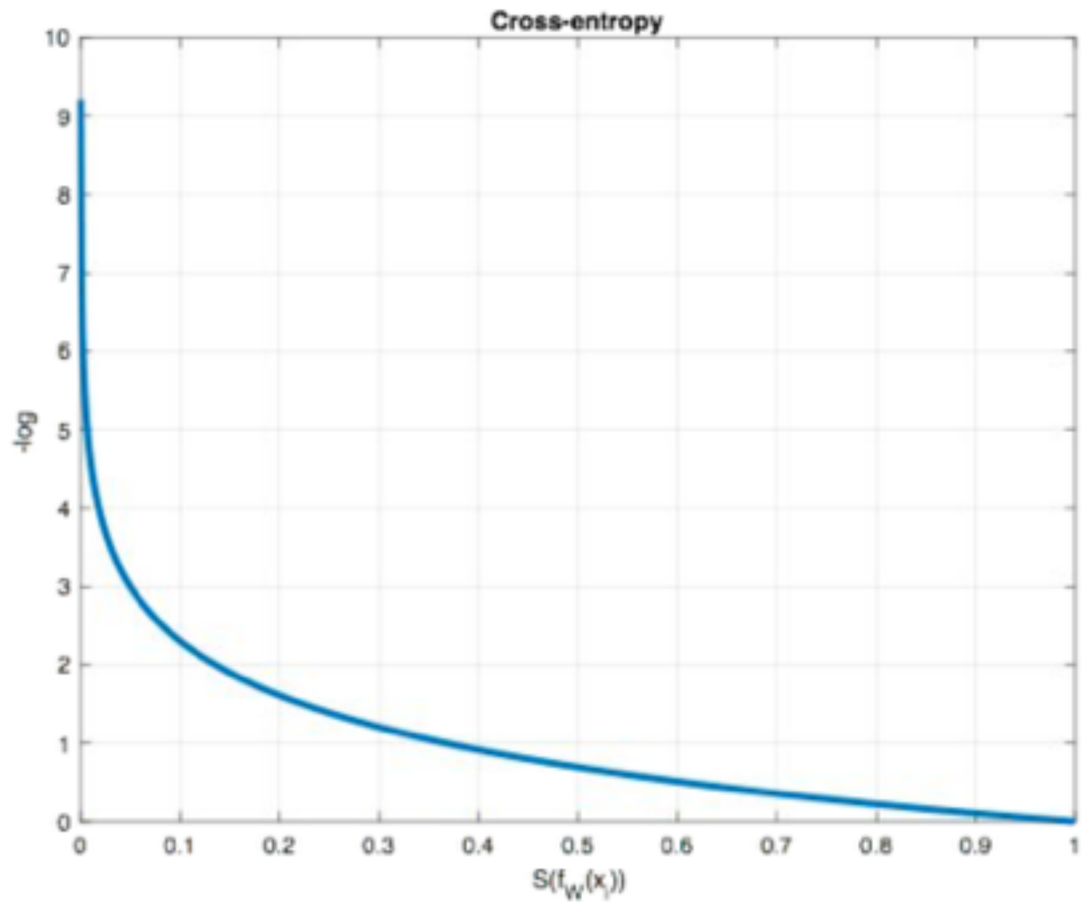
Cross-entropy loss (1)



$$\mathcal{L}_i = -\sum_k y_k \log(S(l_k)) = -\log(S(l))$$

Cross-entropy loss (2)

$$\mathcal{L}_i = -\sum_k y_k \log(S(l_k)) = -\log(S(l))$$



Cross-entropy loss (3)

- For a set of n inputs $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_i$

$$\mathcal{L} = -\sum_{i=1}^{n} \overset{\text{labels (one-hot)}}{\mathbf{y}_i} \log(\underset{\text{Softmax}}{S}(\overset{\text{Our predictions}}{f_\theta(\mathbf{x}_i)})))$$

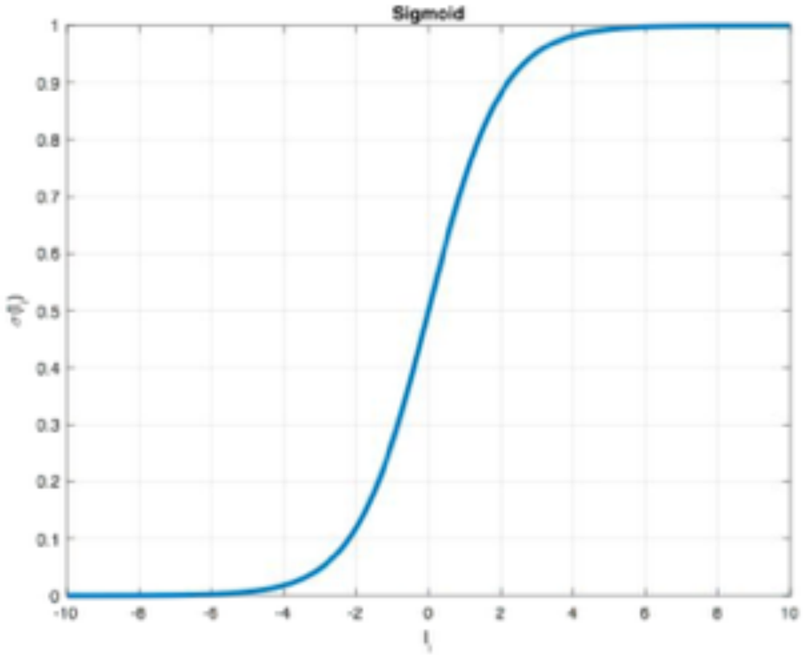
Cross-entropy loss (4)

- In general, cross-entropy loss works better than square error loss:
  - Square error loss usually gives too much emphasis to incorrect outputs.
  - In square error loss, as the output gets closer to either 0.0 or 1.0 the gradients get smaller, the change in weights gets smaller and training is slower.

Multi-label classification (1)

- Outputs can be matched to more than one label
    - “car”, “automobile”, “motor vehicle” can be applied to a same image of a car.
- Use sigmoid at each output independently instead of softmax

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



Multi-label classification (2)

- Cross-entropy loss for multi-label classification:

$$\mathcal{L}_i = -\sum_k y_k \log(\sigma(l_i)) + (1 - y_k) \log(1 - \sigma(l_i))$$

Regularization

Often added at the layer level

- Control the capacity of the network to **prevent overfitting**
  - L2-regularization (weight decay):

regularization parameter

$$\mathcal{L}_{new} = \mathcal{L} + \frac{\lambda}{2} W^2$$

  - L1-regularization:

$$\mathcal{L}_{new} = \mathcal{L} + \frac{\lambda}{2} |W|$$

Example

$$\mathcal{L} = -\sum_{i=1}^{n} \mathbf{y}_i \log(S(f_\theta(\mathbf{x}_i)))$$

0.1	0.3	0.3
0.2	0.4	0.3
0.7	0.3	0.4

“1”	**“2”**	**“3”**
1	0	0
0	1	0
0	0	1

Classification accuracy = 2/3
cross-entropy loss = 4.14

0.3	0.1	0.1
0.4	0.7	0.2
0.3	0.2	0.7

“1”	**“2”**	**“3”**
1	0	0
0	1	0
0	0	1

Classification accuracy = 2/3
cross-entropy loss = 1.92

Loss Functions for Classification

https://arxiv.org/pdf/1702.05659.pdf

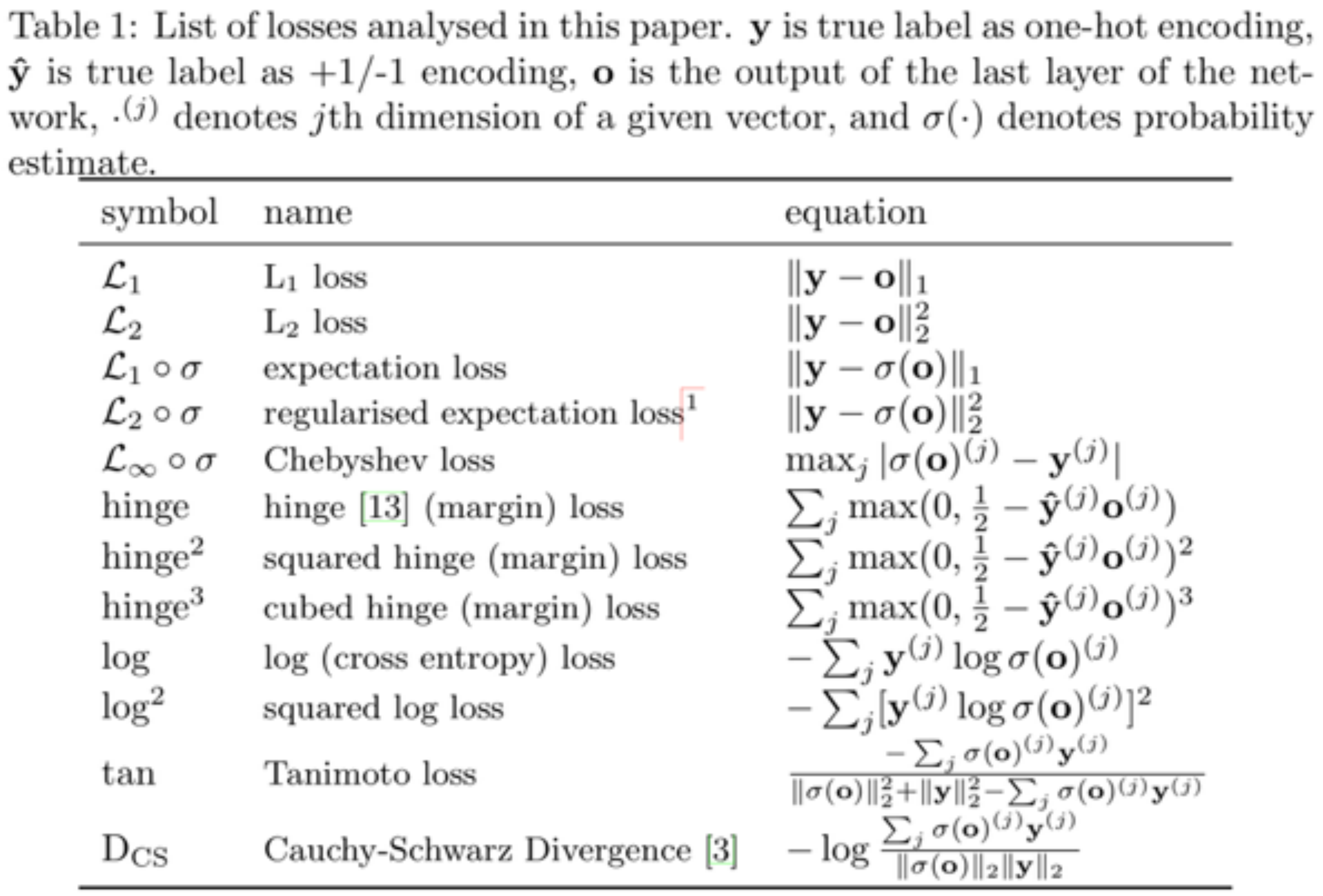


Table 1: List of losses analysed in this paper. $\mathbf{y}$ is true label as one-hot encoding, $\hat{\mathbf{y}}$ is true label as +1/-1 encoding, $\mathbf{o}$ is the output of the last layer of the network, $\cdot^{(j)}$ denotes jth dimension of a given vector, and $\sigma(\cdot)$ denotes probability estimate.

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y}-\mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y}-\mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y}-\sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y}-\sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert\sigma(\mathbf{o})^{(j)}-\mathbf{y}^{(j)}\rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)}\log\sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)}\log\sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2+\|\mathbf{y}\|_2^2-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log\frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2\|\mathbf{y}\|_2}$

$l(\alpha) = \frac{\sum_{k=1}^{r} p_k^\alpha \log_2 \frac{1}{p_k}}{\sum_{k=1}^{r} p_k^\alpha}$ $c_{ik}\sigma_k^2 = \lambda_i c_{ik}$ $\eta_1 = \sum_{k>1} a_k \xi_k$ $i^2 := -1; j^2 := -1; k^2 := -1$ $\lim_{n\to\infty} \frac{\binom{2n}{n+c}}{\binom{2n}{n}} = e^{-2z^2}$

$y = \phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} e^{-\frac{t^2}{2}} dt$ $S(\alpha, T) = \frac{2}{\pi}\int_0^{\pi} \frac{\sin \alpha t}{t} dt$ $P(\eta_\infty < x) = F(x)$

$W_k = \binom{n}{k} p^k (1-p)^{n-k}$ $P(\eta < y | \xi = x) = \sup_{y'<y, y'\in Q} P(\eta < y' | \xi = x)$

$f(t|y) = \frac{2e^{\frac{y^2}{2}}}{\sqrt{2\pi}} \int_{\frac{y}{\sqrt{t}}}^{+\infty} \frac{e^{-\frac{u^2}{2}} du}{\left(1-\frac{y^2}{u^2}\right)^{\frac{3}{2}}}$

$S_n = A_n \cup \pi A_n$

$|A_n| = \frac{n!}{2}$ $\left|\int_{|x|>A} f(x)\log_2\frac{1}{f(x)}dx\right| < \varepsilon$ $g^{-1}\cdot g = e$ $\zeta = \sqrt{\frac{\lambda_n}{\nu_n}}\left(\frac{\eta_{\lambda_n}}{\sqrt{\lambda_n}} + \frac{\eta_{\nu_n} - \eta_{\lambda_n}}{\sqrt{\lambda_n}}\right)$ $\Delta N = \frac{\sum_{n=1}^{N}\frac{\varepsilon_n}{n}}{\sum_{n=1}^{N}\frac{1}{n}}$

$H_r(x) = \frac{G_r(x)}{1+G_r(x)}$

$\int_{-\alpha_k}^{+\alpha_k} dG_k(x) \geq \frac{1}{2}$ $\sum_{k=-\infty}^{+\infty} e^{-\frac{k^2\pi^2}{x^2}} = H(k)$ $\prod_{k\le b}$; $\bigcup_{i=1}^{n-1} M_i$; $\bigcap_{n=0}^{\infty} X_n$ $f_n(t) = \frac{\lambda^n t^{n-1} e^{-\lambda t}}{(n-1)!}$ $U_n^+(c) = \binom{2n}{n} - \binom{2n}{n-c}$

$f_{n+1}(t) = \int_0^t f_n(u) f_1(t-u) du = \frac{\lambda^{n+1} t^n e^{-\lambda t}}{n!}$ $\lim_{t\to 0} (\varepsilon(t)) = 0$ $c_{iv} = \sum_{j=1}^{n} a_{ij} b_{jv}$ $\lim_{n\to+\infty} \mathrm{st} \frac{\zeta_n^{(k)}}{n} = P_k$ $R = \int_{-\infty}^{\infty} \varphi(t) dt$ $\left|\frac{\sin th}{th}[\varphi(t)e^{-itx} + \varphi(-t)e^{i}\right.$

$\log\varphi(t) = i\gamma t - c|t|^\alpha\left[1 + i\beta\frac{t}{|t|}\omega(t,\alpha)\right]$ $B(v) = \sum_{k=1}^{r} \Psi^*(b_k v)$ $\lim_{n\to\infty} P\left(\frac{\zeta_{n+1} - k(n) - \log\frac{1}{q}}{\sqrt{\frac{1-q}{q}}}\right)$ $C_n(\alpha) \geq \frac{n!}{\prod_{k=1}^{r} n_k(\alpha)!}$ $\frac{n}{m}\psi(t) = \psi\left(c\left(\frac{n}{m}\right)t\right)$

$\int_{-\infty}^{\infty} e^{-\frac{u^2}{2}} du = F(x)\left(\frac{1}{\sqrt{2\pi}}\right)^{-1}$ $|\Psi_\xi(t)| = \left|\int_{-\infty}^{\infty} e^{itx} dF(x)\right| \leq \int_{-\infty}^{\infty} e^{-vx} dF(x) = \varphi_\xi(iv)$ $q_k(\alpha) = \frac{p_k^\alpha}{\sum_{j=1}^{r} p_j^\alpha}$ $P(\Pi_2 =$

$\Pi_m = \Pi_r \Pi_{m-r}$ $g^{-1}Ng = \{g^{-1}ng | n \in N\}$ $Q = F^{-1}(q)$

$|X\cup\Psi| = |X| + |\Psi| - |X\cap\Psi|$ $\lim_{n\to\infty} \frac{1}{\sqrt{n}} k_n\left(\frac{x}{\sqrt{n}}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ $P_n(k) = p_{j_0 k}^{(n)}$ $P\left(\lim\sup_{n\to\infty} \frac{|h_n|}{\sqrt{2n\log\log n}} \le 1\right) = 1$ $\varphi(t) = 1 - \sqrt{1 - e^{2it}}$

$f: X \to X\cap W$

$fg(u_i) = f\left(\sum_{j=1}^{\dim V_2} a_{ji} v_j\right) = \sum_{j=1}^{\dim V_2} a_{ji}\left(\sum_{k=1}^{\dim V_3} b_{kj} w_k\right)$ $\frac{\binom{2l_k}{l_k}}{2^{2l_k}} \approx \frac{1}{\sqrt{\pi l_k}}$

$Q(A) = \int_A \chi(\omega) dP$ $l'(\alpha) = -\log 2\left(\frac{\sum_{k=1}^{r} p_k^\alpha \log_2^2 \frac{1}{p_k}}{\sum_{k=1}^{r} p_k^\alpha} - \left(\frac{\sum_{k=1}^{r} p_k^\alpha\log_2\frac{1}{p_k}}{\sum_{k=1}^{r} p_k^\alpha}\right)^2\right)$

$p_{jk}^{(m)} = \sum_{l=0}^{\infty} p_{jl}^{(r)} p_{lk}^{(m-r)}$ $\frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{Re}\left\{\varphi(t)\frac{e^{-ita} - e^{-itb}}{it}\right\} dt$

$q\left(c^{-x\sqrt{\frac{1-q}{nq}}} - 1\right) = -x\sqrt{\frac{q(1-q)}{n}} + o\left(\frac{1}{n}\right)$ $\prod_{k=1}^{r}\left[g_k\left(\frac{t}{\sqrt{N_0}}\right)\right]^{N_0\alpha_k} = e^{-\frac{t^2}{2}}$ $P(|\Delta N| > \varepsilon) \le \frac{C_q}{\log N}$

$\lim \int_{-A}^{+A} f_N(x)\log_2\frac{1}{f_N(x)} dx = \int_{-A}^{A} f(x)\log_2\frac{1}{f(x)} dx$

Math Preliminaries

- Derivatives and Differentiation

$$\frac{df}{dx}(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

- Partial Derivatives

$$\frac{\partial f}{\partial x}(x_0, y_0, \ldots) = \lim_{h \to 0} \frac{f(x_0 + h, y_0, \ldots) - f(x_0, y_0, \ldots)}{h}$$

- (Vector form) Gradients

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix}$$

$y = 1/x$

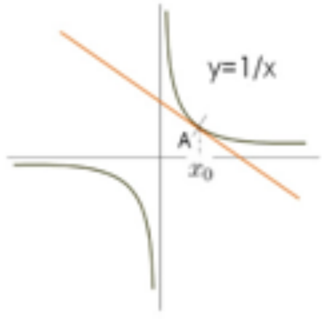
$$\frac{\Delta f}{\Delta x} = \frac{\frac{1}{x_0+\Delta x} - \frac{1}{x_0}}{\Delta x}$$

$$= \frac{1}{\Delta x}\left[\frac{x_0 - (x_0+\Delta x)}{(x_0+\Delta x)x_0}\right]$$

$$= \frac{1}{\Delta x}\left[\frac{-\Delta x}{(x_0+\Delta x)x_0}\right]$$

$$= \frac{-1}{(x_0+\Delta x)x_0}$$

$$\boxed{\lim_{\Delta x\to 0}\frac{-1}{(x_0+\Delta x)x_0} = \frac{-1}{x_0^2}}$$



$\boxed{dy/dx = -1/x^2}$

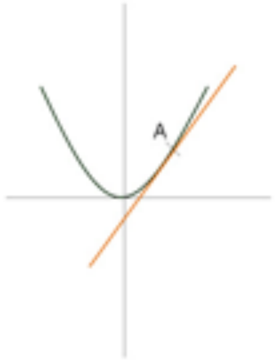
$y = x^2$

$$\lim_{\Delta x\to 0}\frac{\Delta f}{\Delta x} = \lim_{\Delta x\to 0}\frac{f(x_0+\Delta x) - f(x_0)}{\Delta x}$$

$$= \lim_{\Delta x\to 0}\frac{(x+\Delta x)^2 - x^2}{\Delta x}$$

$$= \lim_{\Delta x\to 0}\frac{(\Delta x)^2 + 2x\Delta x}{\Delta x}$$

$$= \lim_{\Delta x\to 0}\Delta x + 2x = \underline{\underline{2x}}$$



$\boxed{dy/dx = 2x}$

$y = x^n$

$$= \lim_{\Delta x\to 0}\frac{(x+\Delta x)^n - x^n}{\Delta x}$$

$$= \lim_{\Delta x\to 0}\frac{x^n + n(\Delta x)(x^{n-1}) + \Phi - x^n}{\Delta x}$$

$$= \lim_{\Delta x\to 0} nx^{n-1} + \phi(\Delta x)$$

$$\boxed{\lim_{\Delta x\to 0}\frac{\Delta y}{\Delta x} = nx^{n-1}}$$

$(x+\Delta x)^n = (x+\Delta x)(x+\Delta x)...(x+\Delta x)$ n times

$$\equiv x^n + n(\Delta x)x^{n-1} + \Phi$$

Where Φ consists of terms with coefficients $(\Delta x)^2$, $(\Delta x)^3$, and so on up to $(\Delta x)^n$

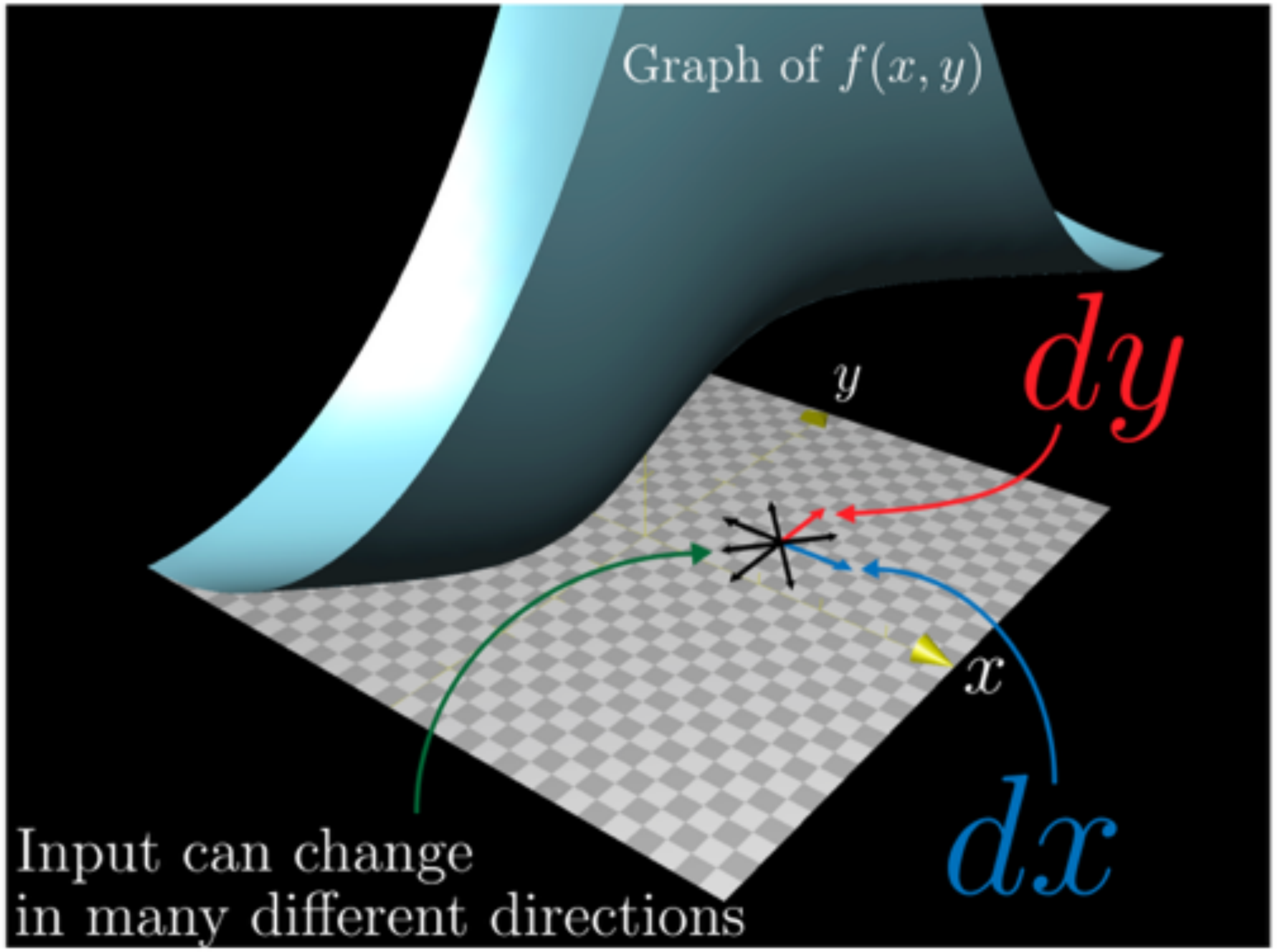
$$\boxed{\frac{d}{dx}x^n = nx^{n-1}}$$

Differentiation "dy dx"

Partial Derivatives “del”

$$\frac{\partial f}{\partial x} = \underbrace{\frac{\partial}{\partial x} x^2 y}_{\substack{\text{Treat } y \text{ as constant;} \\ \text{take derivative.}}} = 2xy$$

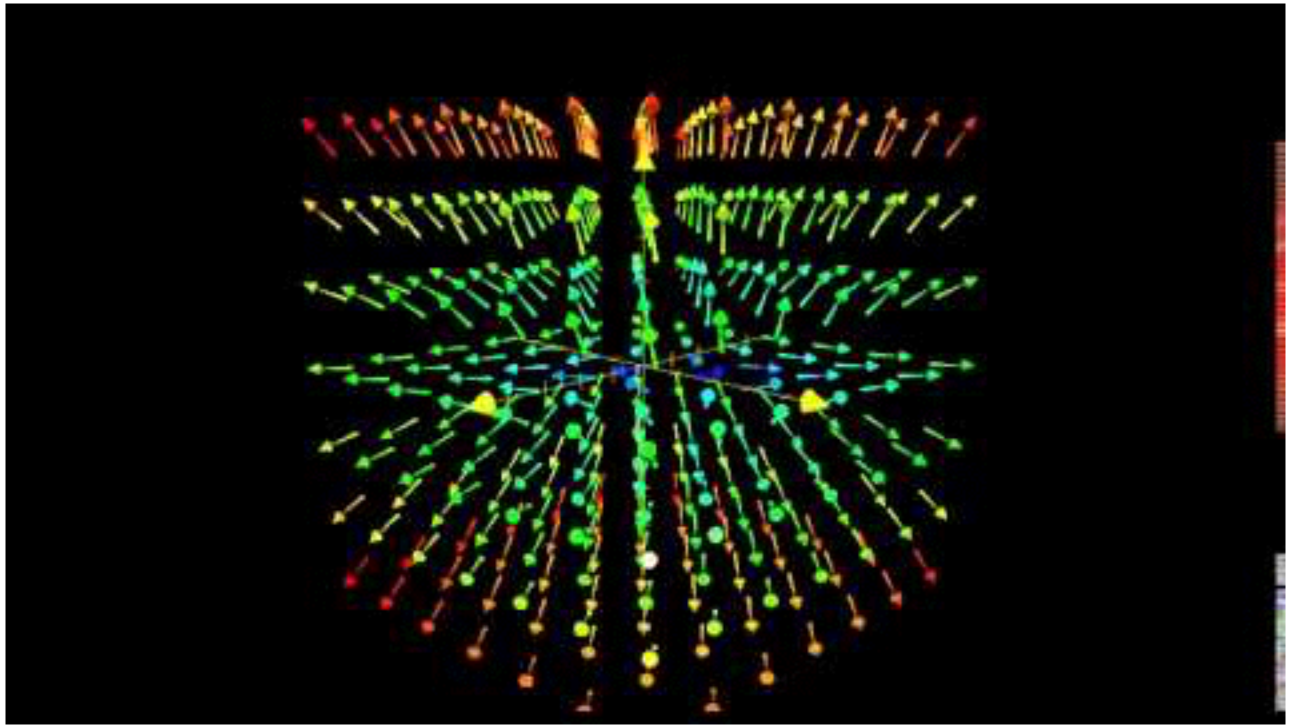
$$\frac{\partial f}{\partial y} = \underbrace{\frac{\partial}{\partial y} x^2 y}_{\substack{\text{Treat } x \text{ as constant;} \\ \text{take derivative.}}} = x^2 \cdot 1$$



Gradient “nabla”

What is the gradient of $f(x,y,z) = x - xy + z^2$?

$$\nabla f(x,y,z) = \begin{bmatrix} \frac{\partial}{\partial x}(x - xy + z^2) \\ \frac{\partial}{\partial y}(x - xy + z^2) \\ \frac{\partial}{\partial z}(x - xy + z^2) \end{bmatrix} = \begin{bmatrix} 1 - y \\ -x \\ 2z \end{bmatrix}$$



https://youtu.be/xchxDy0AFeI

Gradient of the Loss

The gradient of the loss is equal to the mean of the gradient over each sample loss

$$\nabla L = \frac{1}{n} \sum_{x} \nabla L_x$$

This fact will be useful when we are trying to estimate the overall loss gradient from subset of samples