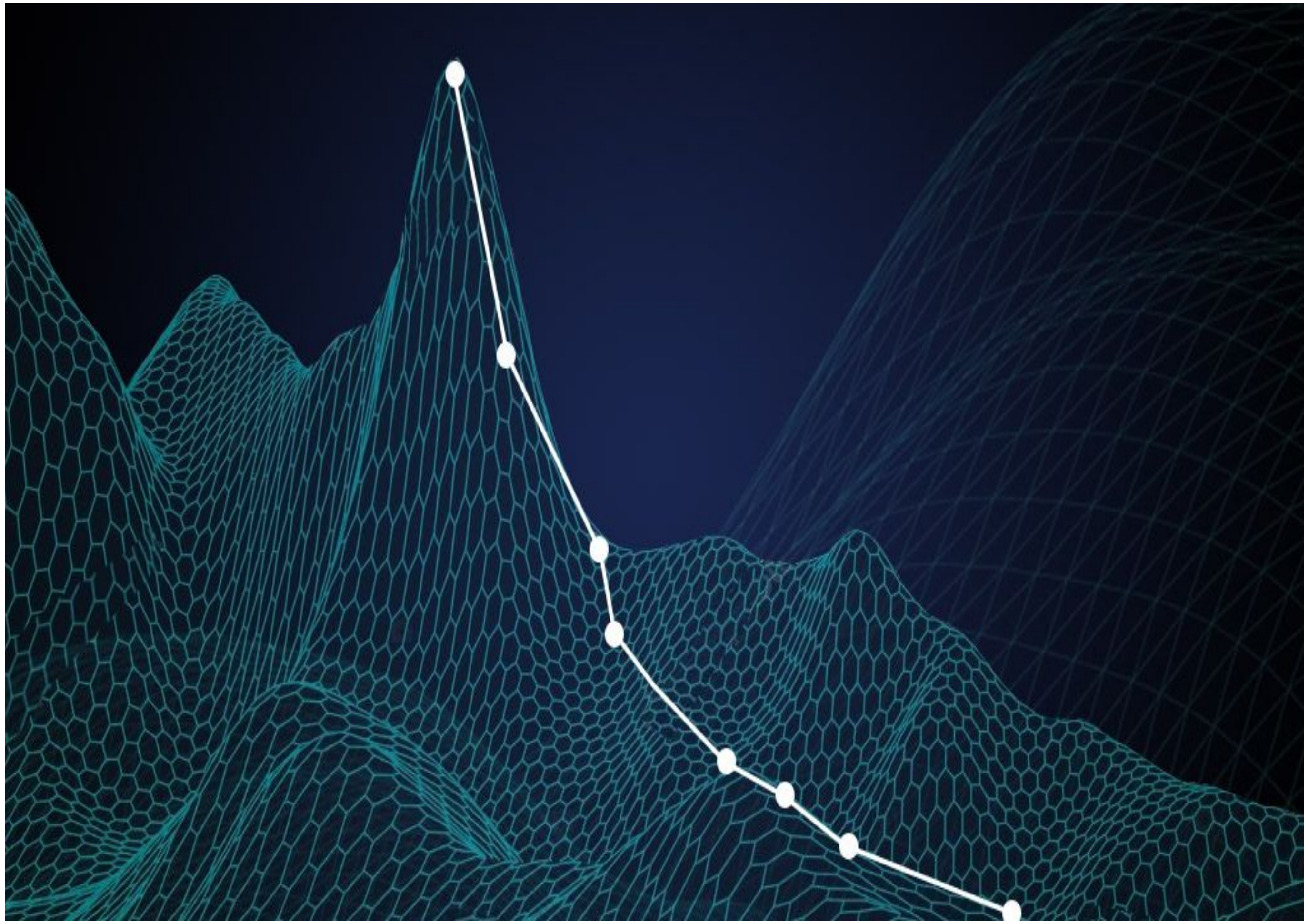


Activation and Loss Functions

PhD. Msc. David C. Baldears S.
PhD(s). Msc. Diego López Bernal

Output Activation and Loss Functions

Every neural net specifies

- an activation rule for the output unit(s)
- a loss defined in terms of the output activation

First a bit of review…

Cheat Sheet 1

Perceptron

- Activation function $z_j = \sum_i w_{ji} x_i$ $\quad y = \begin{cases} 1 & if\ z_j > 0 \\ 0 & \text{otherwise} \end{cases}$

- Weight update $\Delta w_{ji} = (t_j - y_j) x_i$ $\quad t_j \in \{0, 1\}$

assumes minimizing number of misclassifications

Linear associator (a.k.a. linear regression)

- Activation function $y_j = \sum_i w_{ji} x_i$

- Weight update $\Delta w_{ji} = \varepsilon (t_j - y_j) x_i$

assumes minimizing squared error loss

Cheat Sheet 2

Two layer net (a.k.a. logistic regression)

- activation function
- weight update

$$z_j = \sum_i w_{ji} x_i \qquad y_j = \frac{1}{1 + e^{-z_j}}$$

$$\Delta w_{ji} = \varepsilon (t_j - y_j) y_j (1 - y_j) x_i \qquad t_j \in [0, 1]$$

assumes minimizing squared error loss

Deep(er) net

assumes minimizing squared error loss

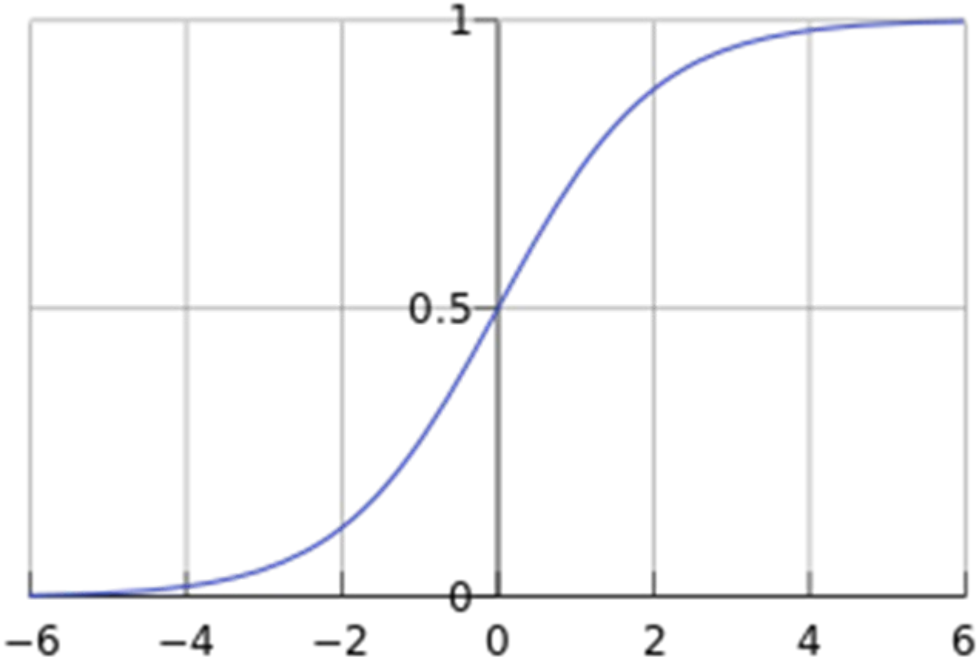
- activation function same as above
- weight update

$$\Delta w_{ji} = \varepsilon \delta_j x_i \qquad \delta_j = \begin{cases} (t_j - y_j) y_j (1 - y_j) & \text{for output unit} \\ \left(\sum_k w_{kj} \delta_j \right) y_j (1 - y_j) & \text{for hidden unit} \end{cases}$$

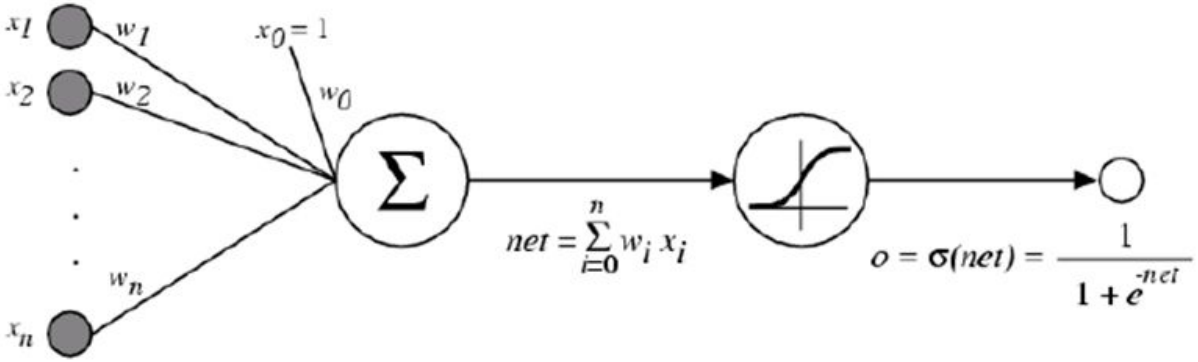
Activation Functions

Sigmoid / Logistic Function

$$logistic(u) = \frac{1}{1 + e^{-u}}$$



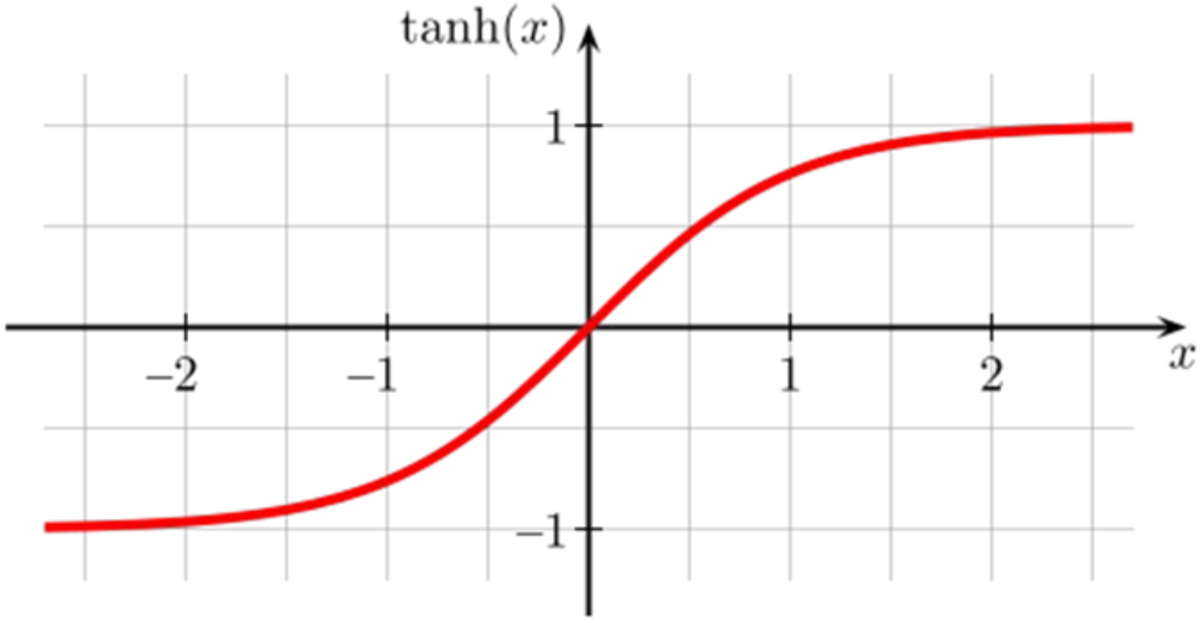
So far, the activation function (nonlinearity) is always the sigmoid function...



Activation Functions

A new change: modifying the nonlinearity

The logistic is not widely used in modern ANNs



Alternate 1: tanh

Like logistic function but shifted to range [-1, +1]

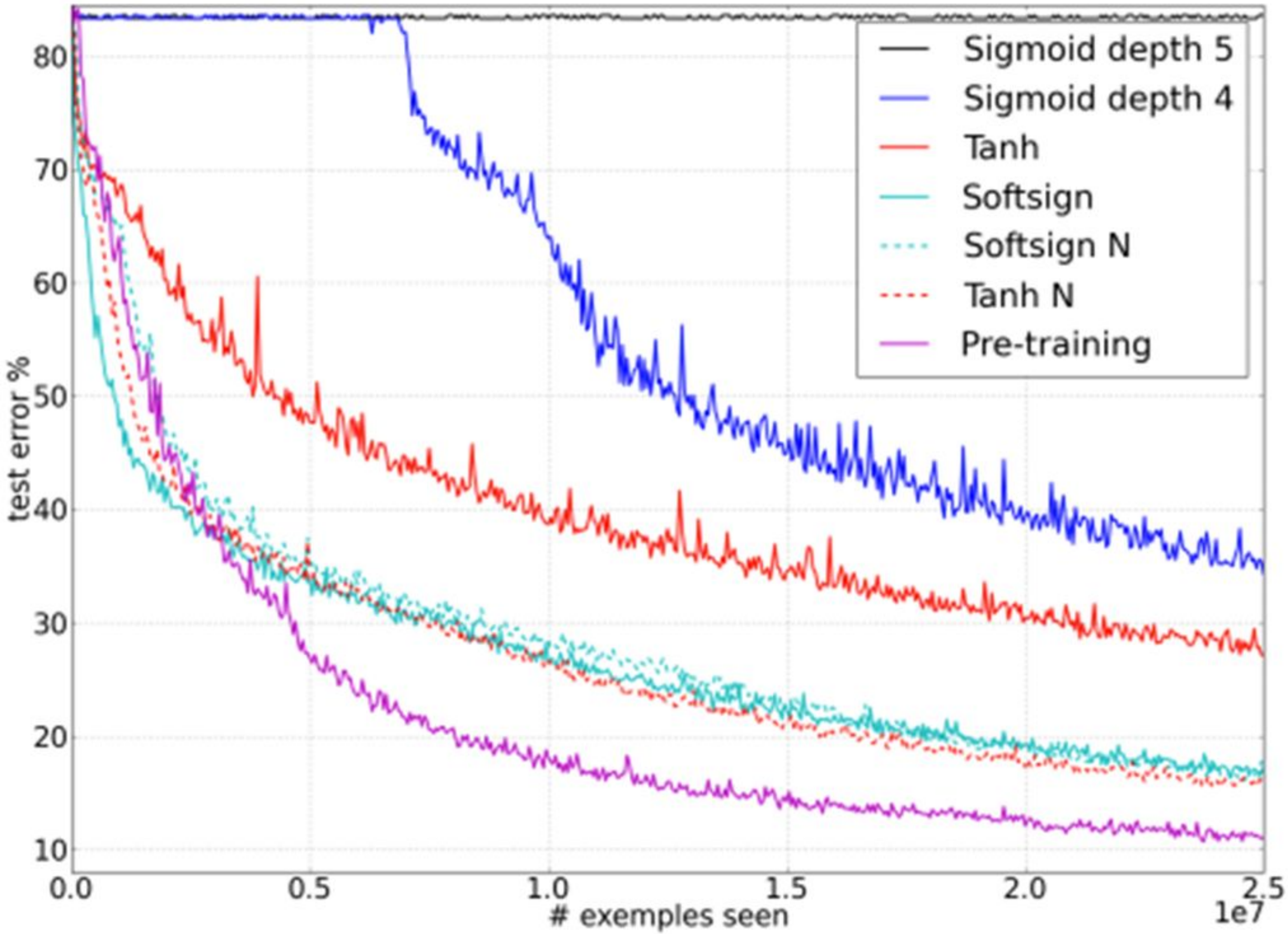
Logistic vs Tanh

Logistic

- Output = .5
  - when no input evidence, bias=0
- Will trigger activation in next layer
- Need large biases to neutralize
  - biases on different scale than other weights
- Does not satisfy weight initialization assumption of mean activation = 0

Tanh

- Output = 0
  - when no input evidence, bias=0
- Won’t trigger activation in next layer
- Don’t need large biases
- Satisfies weight initialization assumption



sigmoid
vs.
tanh

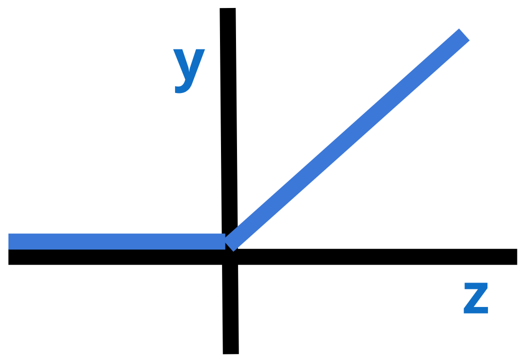
Rectified Linear Unit (ReLU)

Activation function

$$y = max(0, z)$$

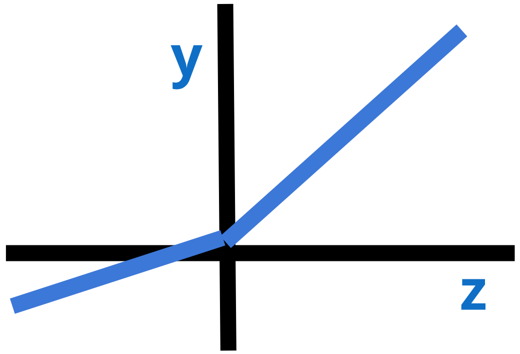
Derivative

$$\frac{\partial y}{\partial z} = \begin{cases} 0 & z \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



- Linear with a cutoff at zero
  - Implementation: clip the gradient when you pass zero)
  - Often used in vision tasks
- Advantages
  - fast to compute activation and derivatives
  - no squashing of back propagated error signal as long as unit is activated
  - discontinuity in derivative at z=0
  - sparsity ?
- Disadvantages
  - can potentially lead to exploding gradients and activations
  - may waste units: units that are never activated above threshold won't learn

Leaky ReLU



Activation function

$$y = \begin{cases} z & z > 0 \\ \alpha z & \text{otherwise} \end{cases}$$

Derivative

$$\frac{\partial y}{\partial z} = \begin{cases} 1 & z > 0 \\ \alpha & \text{otherwise} \end{cases}$$

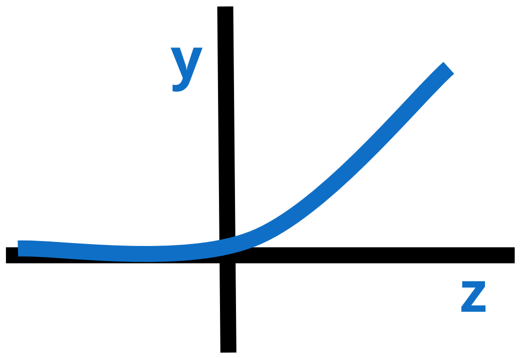
Reduces to standard ReLU if $\boldsymbol{\alpha}$**=0**

Trade off

$\boldsymbol{\alpha}$**=0** leads to inefficient use of resources (underutilized units)

$\boldsymbol{\alpha}$**=1** lose nonlinearity essential for interesting computation

Softplus



Activation function

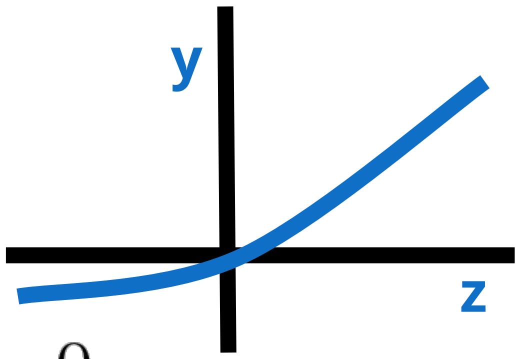
$$y = ln(1 + e^{z})$$

- Derivative defined everywhere
- zero only for $z \to -\infty$
- Doesn't saturate (at one end)
- Sparsifies outputs
- Helps with vanishing gradient

Derivative

$$\frac{\partial y}{\partial z} = \frac{1}{1 + e^{-z}} = logistic(z)$$

Exponential Linear Unit (ELU)



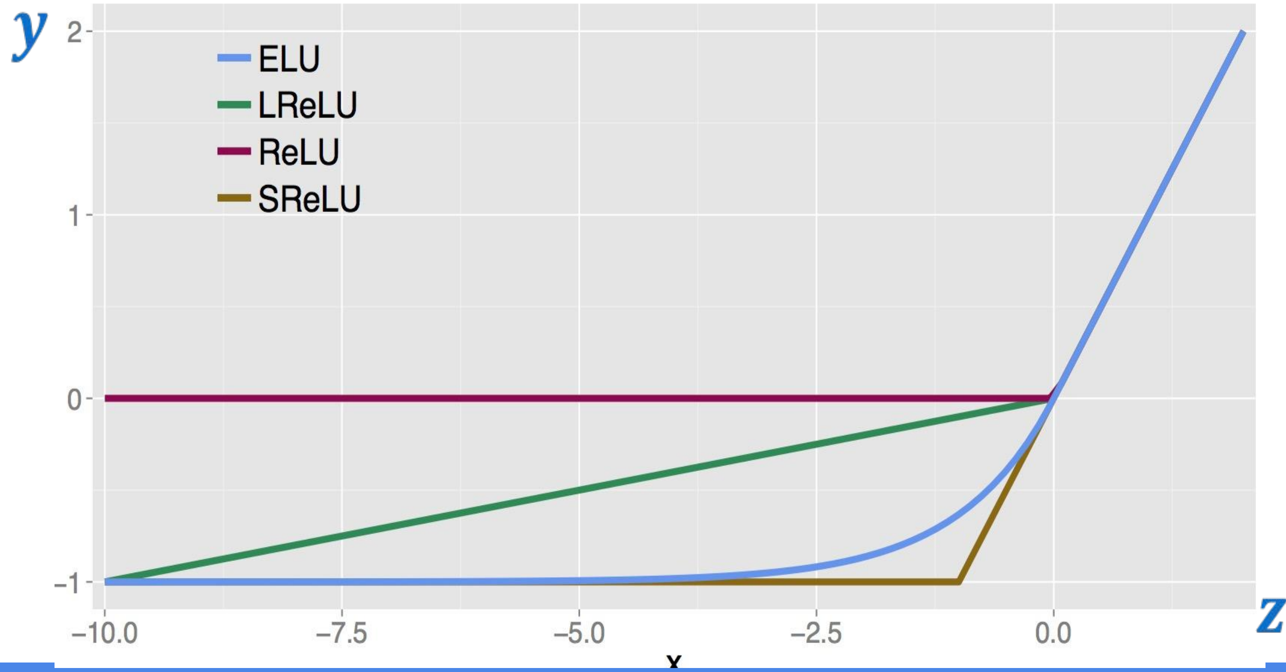
Activation function

$$y = \begin{cases} z & z > 0 \\ \alpha(e^{z} - 1) & z \leq 0 \end{cases}$$

Derivative

$$\frac{\partial y}{\partial z} == \begin{cases} 1 & z > 0 \\ y + \alpha & z \leq 0 \end{cases}$$

Reduces to standard ReLU if α=**0**



Radial Basis Functions

Activation function

$$y = e^{(-\|x-w\|^2)}$$

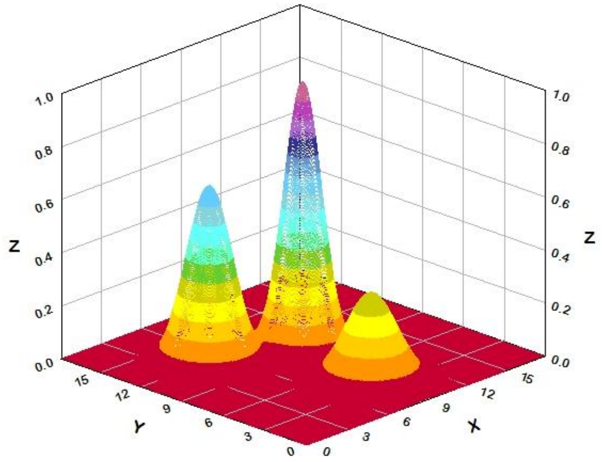
Sparse activation

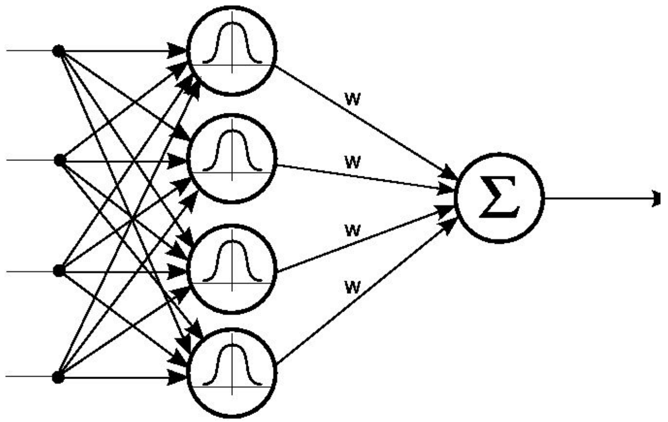
many units just don't learn

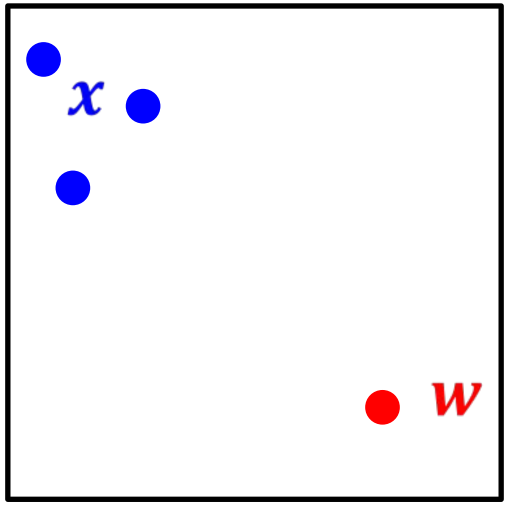
same issue as ReLUs

Clever schemes to initialize weights

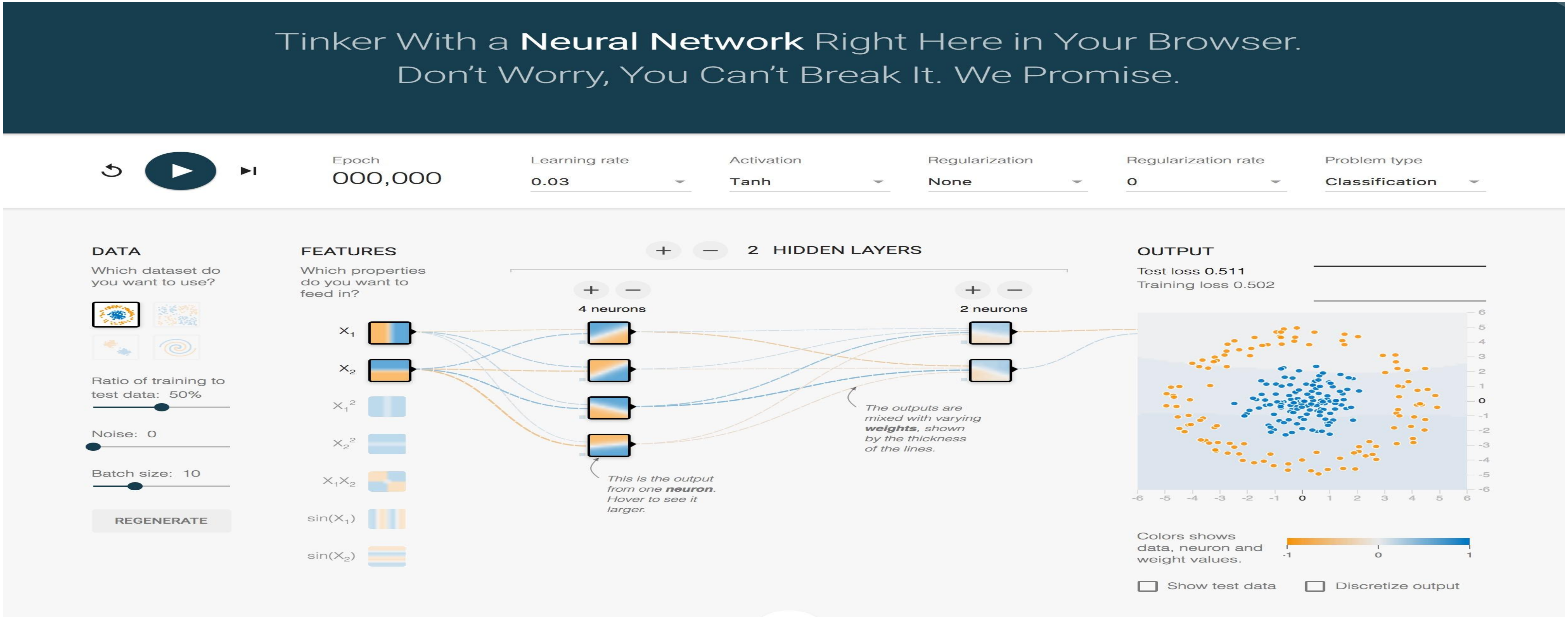
e.g., set $\boldsymbol{w}$ near cluster of $\boldsymbol{x}$'s







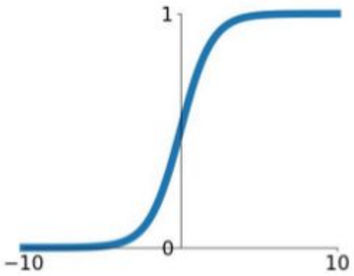
playground.tensorflow.org



Common Activation Functions

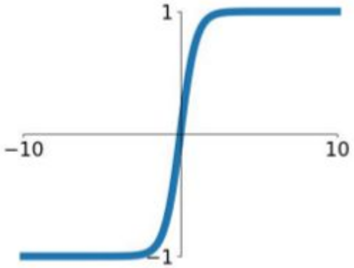
Sigmoid

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



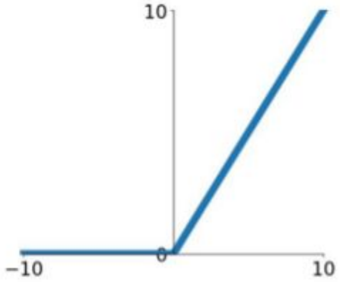
tanh

$\tanh(x)$



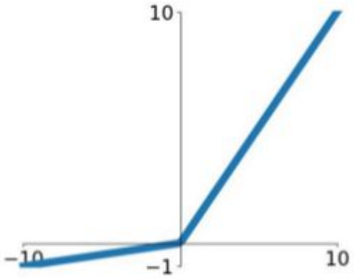
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

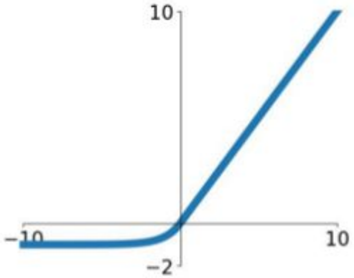


Maxout

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

ELU

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



Categorical Outputs

Considered the case where the output y denotes the probability of class membership

belonging to class A versus $\bar{A}$

Instead of two possible categories, suppose there are n

e.g., animal, vegetable, mineral

Categorical Outputs

Each input can belong to one category

- y_j denotes the probability that the input's category is j

To interpret y as a probability distribution over the alternatives

- $\sum_j y_j = 1$ and $0 \leq y_j \leq 1$

Activation function $y_j = \dfrac{e^{z_j}}{\sum_k e^{z_k}}$

Exponentiation ensures nonnegative values

Denominator ensures sum to 1

Known as softmax, and formerly, Luce choice rule

Derivatives For Categorical Outputs

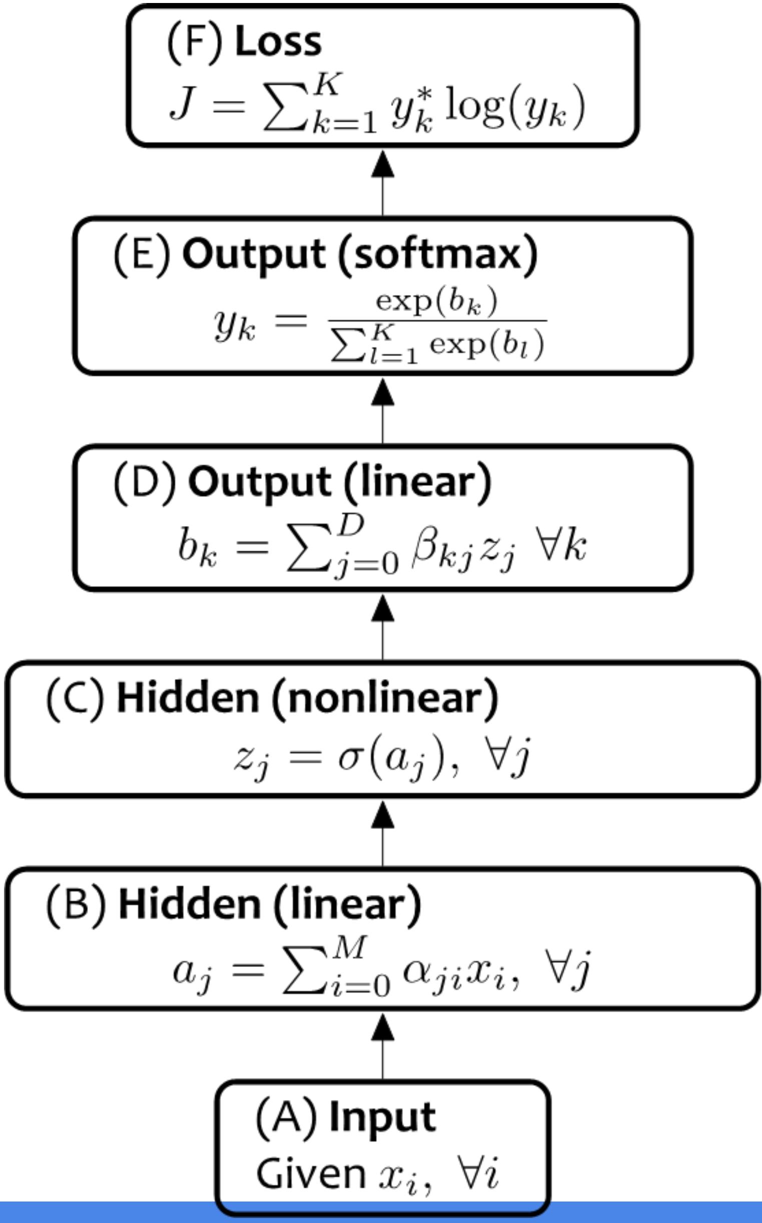
For softmax output function

$$y_j = \frac{e^{z_j}}{\sum_k e^{z_k}} \qquad z_j = \sum_{i=1}^{n} w_{ji} x_i$$

Weight update is the same as for two-category case!

$$\Delta w_{ji} = \varepsilon \delta_j x_i \qquad \delta_j = \begin{cases} \frac{\partial E}{\partial y_j} y_j(1-y_j) & \text{for output unit} \\ \left(\sum_k w_{kj} \delta_j\right) y_j(1-y_j) & \text{for hidden unit} \end{cases}$$

...when expressed in terms of ***y***

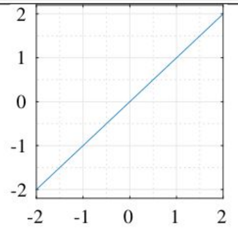
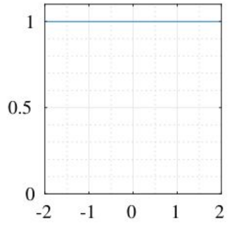
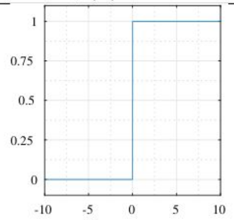
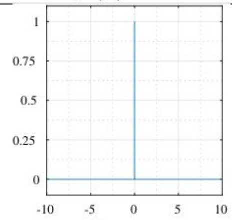
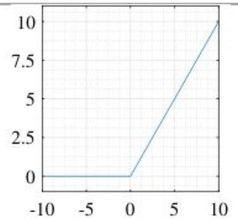
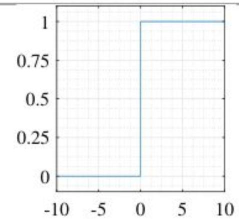
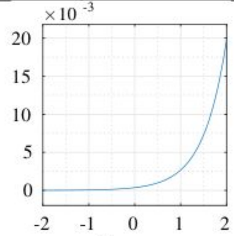
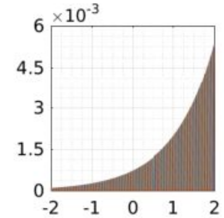
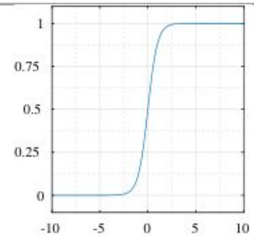
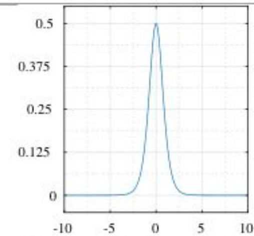
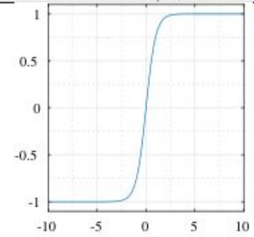
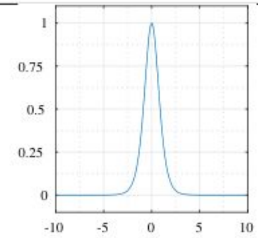
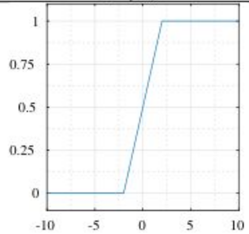
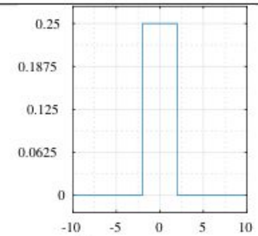
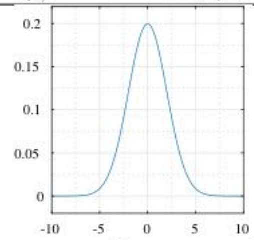
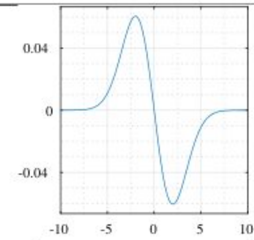


Objective Functions for NNs

- Regression:
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)
- Classification:
  - Use the same objective as Logistic Regression
  - Cross-entropy (i.e. negative log likelihood)
  - This requires probabilities, so add an additional “softmax” layer at the end of our network

	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$

Common Activation Functions

Name	Activation $y = f(x)$	Derivative $f'(x) = \partial y / \partial x$
Linear	 $f(x) = x$	 $f'(x) = 1$
Binary	 $f(x) = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x \geq 0 \end{cases}$	 $f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ 1 & \text{for } x = 0 \end{cases}$
ReLu	 $f(x) = \begin{cases} 0; & \text{for } x <= 0 \\ x; & \text{for } x > 0 \end{cases}$	 $f'(x) = \begin{cases} 0; & \text{for } x <= 0 \\ 1; & \text{for } x > 0 \end{cases}$
Softmax	 $f(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}, \ \forall i = 1 \dots N$	 $f'(x) = \begin{cases} f(x_i)(1 - f(x_j)); & \text{for } i == j \\ -f(x_i) f(x_j); & \text{for } i \neq j \end{cases}$
Logistic Sigmoid	 $f(x) = \frac{1}{1 + e^{-\beta x}}$	 $f'(x) = f(x)(1 - f(x))$
Tangent Sigmoid	 $f(x) = \frac{2}{1 + e^{-2x}} - 1$	 $f'(x) = 1 - f(x)^2$
Piecewise linear	 $f(x) = \begin{cases} 0; & x < xmin \\ \frac{x - xmin}{xmax - xmin}; & xmin \leq x \leq xmax \\ 1; & xmax < x \end{cases}$	 $f'(x) = \begin{cases} 1 & xmin \leq x \leq xmax \\ 0; & \text{otherwise} \end{cases}$
Gaussian	 $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	 $f'(x) = -x * f(x)/\sigma^2$

Squared Error Loss

Sensible regardless of output range and output activation function

$$E = \frac{1}{2}\sum_j (t_j - y_j)^2 \qquad \frac{\partial E}{\partial y_j} = y_j - t_j$$

Remember

$$\Delta w = \varepsilon \frac{\partial E}{\partial y}$$

- with logistic output unit

$$y_j = \frac{1}{1 + e^{-z_j}} \qquad \frac{\partial y_j}{\partial z_j} = y_j(1 - y_j)$$

- with tanh output unit

$$\begin{aligned} y_j &= tanh(z_j) \\ &= \frac{2}{1 + e^{-z_j}} - 1 \end{aligned} \qquad \frac{\partial y_j}{\partial z_j} = (1 + y_j)(1 - y_j)$$

Cross Entropy Loss

Used when the target output represents a probability distribution

- e.g., a single output unit that indicates the classification decision (yes, no) for an input
- Output $\boldsymbol{y} \in [\mathbf{0},\mathbf{1}]$ denotes Bernoulli likelihood of class membership
- Target $\boldsymbol{t}$ indicates true class probability (typically 0 or 1)

Note: single value represents probability distribution over 2 alternatives

Cross entropy, $\boldsymbol{H}$, measures distance in bits from predicted distribution to target distribution

$$E = H = -t\ln(y) - (1-t)\ln(1-y)$$

$$\frac{\partial E}{\partial y} = \frac{y-t}{y(1-y)}$$

Squared Error Versus Cross Entropy

$$\frac{\partial E_{\text{sqerr}}}{\partial y} = y - t$$

$$\frac{\partial y}{\partial z} = y(1 - y)$$

$$\frac{\partial E_{\text{sqerr}}}{\partial z} = (y - t)y(1 - y)$$

$$\frac{\partial E_{\text{xentropy}}}{\partial y} = \frac{y - t}{y(1 - y)}$$

$$\frac{\partial y}{\partial z} = y(1 - y)$$

$$\frac{\partial E_{\text{xentropy}}}{\partial z} = y - t$$

Essentially, cross entropy does not suppress learning when output is confident (near 0 or 1)

- net devotes its efforts to fitting target values exactly
- e.g., consider situation where $\boldsymbol{y}$**=.99** and $\boldsymbol{t}$**=1**

Maximum Likelihood Estimation

In statistics, many parameter estimation problems are formulated in terms of maximizing the likelihood of the data

- find model parameters that maximize the likelihood of the data under the model
- e.g., 10 coin flips producing 8 heads and 2 tails
- What is the coin's bias?

Likelihood formulation

$$\mathcal{L} = y^t(1-y)^{1-t} \quad for \quad t \in \{0,1\}$$

$$\ell = ln(\mathcal{L}) = tln(y) + (1-t)(1-y)$$

What's the relationship between ℓ and E_{xentropy}?

Probabilistic Interpretation of Squared-Error Loss

Consider a network output and target $\boldsymbol{y}, \boldsymbol{t} \in \mathbb{R}$

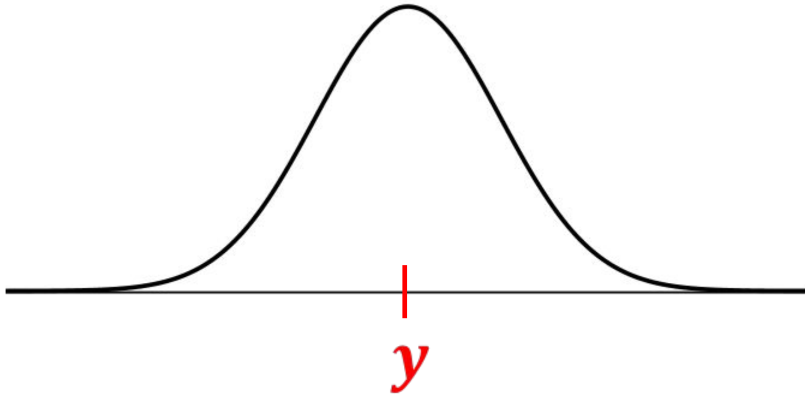
Suppose that the output is corrupted by Gaussian observation noise

$\boldsymbol{y} = \boldsymbol{t} + \boldsymbol{\eta}$

where $\boldsymbol{\eta}$ ~"Gaussian" $(\mathbf{0}, \mathbf{1})$

We can define the likelihood of the target under this noise model

$$P(t|y) = \frac{1}{\sqrt{2\pi}} e^{\left(-\frac{1}{2}(t-y)^2\right)}$$



Probabilistic Interpretation of Squared-Error Loss

For a set of training examples, $\alpha \in \{1,2,3, \ldots\}$, we can define the data set likelihood

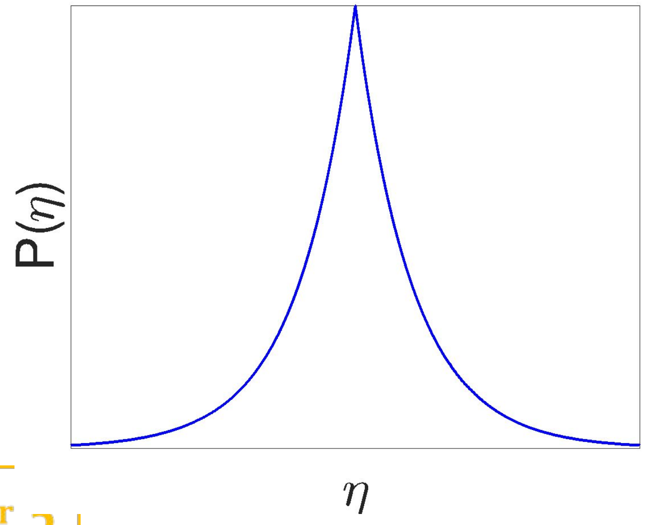
$$\mathcal{L} = \prod_{\alpha} P(t^\alpha | y^\alpha)$$

$$\ell = ln(\mathcal{L}) = \sum_{\alpha} ln(P(t^\alpha | y^\alpha))$$ where $$P(t|y) = \frac{1}{\sqrt{2\pi}} e^{(-\frac{1}{2}(t-y)^2)}$$

$$= -\frac{1}{2} \sum_{\alpha} (t^\alpha - y^\alpha)^2$$

Squared error can be viewed as likelihood under Gaussian observation noise $E_{\mathrm{sqerr}} = -c\ell$

Other noise distributions can motivate alternative losses.



e.g., Laplace distributed noise and $E_{\mathrm{abserr}} = |t - y|$

What is $\frac{\partial E_{\mathrm{abserr}}}{\partial y}$?