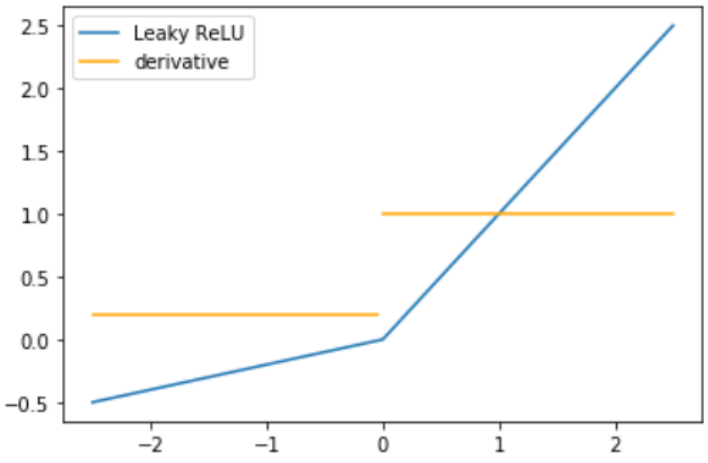
Deep Learning

Modern Activation Functions



Learning goals

-
-
-

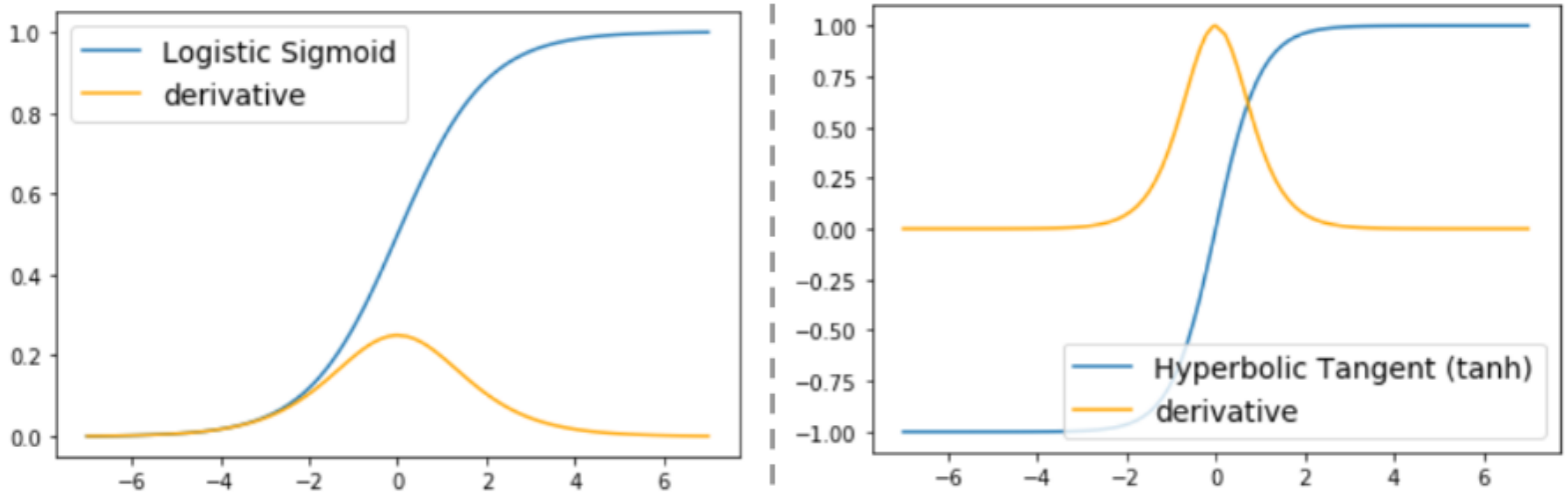
Hidden activations

HIDDEN ACTIVATIONS

- Recall, hidden-layer activation functions make it possible for deep neural nets to learn complex non-linear functions.
- The design of hidden units is an extremely active area of research. It is usually not possible to predict in advance which activation will work best. Therefore, the design process often consists of trial and error.
- In the following, we will limit ourselves to the most popular activations - Sigmoidal activation and ReLU.
- It is possible for many other functions to perform as well as these standard ones. An overview of further activations can be found ▸ here.

SIGMOIDAL ACTIVATIONS

- Sigmoidal functions such as *tanh* and the *logistic sigmoid* bound the outputs to a certain range by "squashing" their inputs.



- In each case, the function is only sensitive to its inputs in a small neighborhood around 0.
- Furthermore, the derivative is never greater than 1 and is close to zero across much of the domain.

SIGMOIDAL ACTIVATION FUNCTIONS

1. **Saturating Neurons:**
   - We know: $\sigma'(z_{in}) \to 0$ for $|z_{in}| \to \infty$.

   $\to$ Neurons with sigmoidal activations "saturate" easily, that is, they stop being responsive when $|z_{in}| \gg 0$.

SIGMOIDAL ACTIVATION FUNCTIONS

2. **Vanishing Gradients:** Consider the vector of error signals $\boldsymbol{\delta}^{(i)}$ in layer i

$$\boldsymbol{\delta}^{(i)} = \mathbf{W}^{(i+1)}\boldsymbol{\delta}^{(i+1)} \odot \sigma'\left(\boldsymbol{z}_{in}^{(i)}\right), i \in \{1, \ldots, O\}.$$

Each k-th component of the vector expresses how much the loss L changes when the input to the k-th neuron $z_{k,in}^{(i)}$ changes.

- We know: $\sigma'(z) < 1$ for all $z \in \mathbb{R}$.

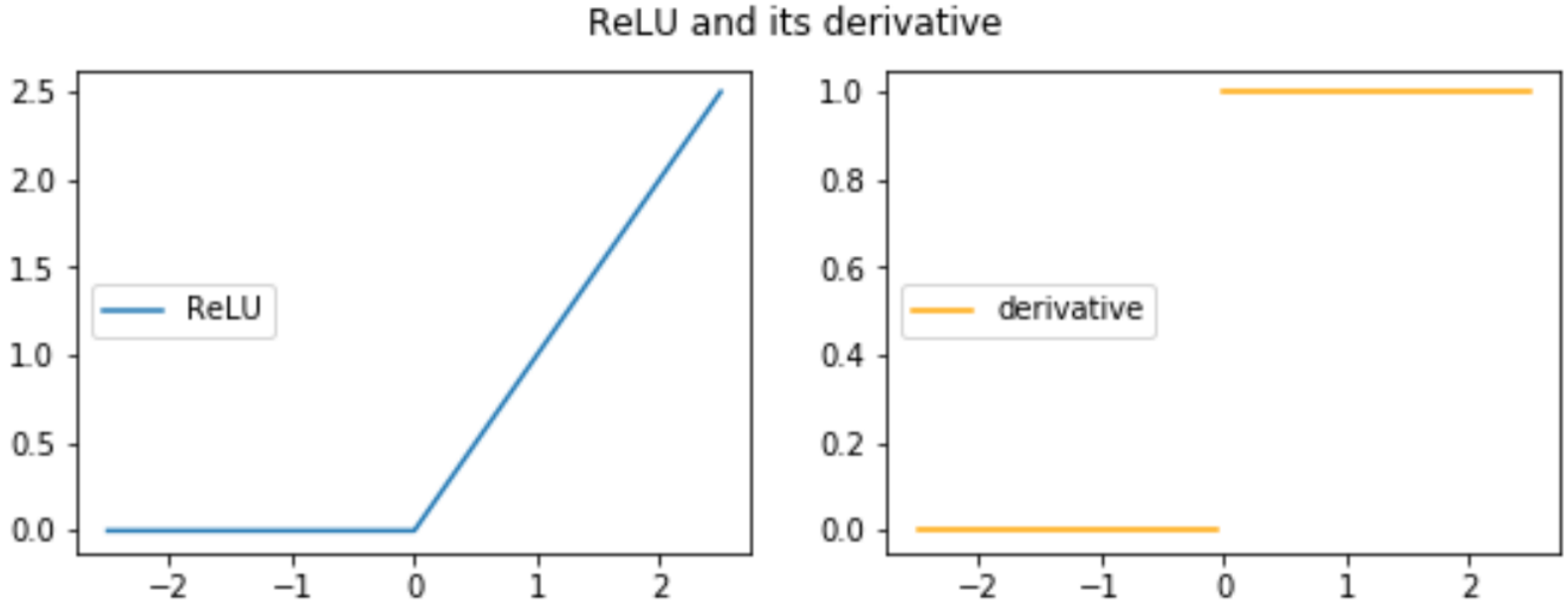
$\rightarrow$ In each step of the recursive formula above, the value will be multiplied by a value smaller than one

$$\begin{aligned}
\boldsymbol{\delta}^{(1)} &= \mathbf{W}^{(2)}\boldsymbol{\delta}^{(2)} \odot \sigma'\left(z_{in}^{(i)}\right) \\
&= \mathbf{W}^{(2)}\left(\mathbf{W}^{(3)}\boldsymbol{\delta}^{(3)} \odot \sigma'\left(z_{in}^{(i)}\right)\right) \odot \sigma'\left(z_{in}^{(i)}\right) \\
&= \ldots
\end{aligned}$$

- When this occurs, earlier layers train *very* slowly (or not at all).

RECTIFIED LINEAR UNITS (RELU)

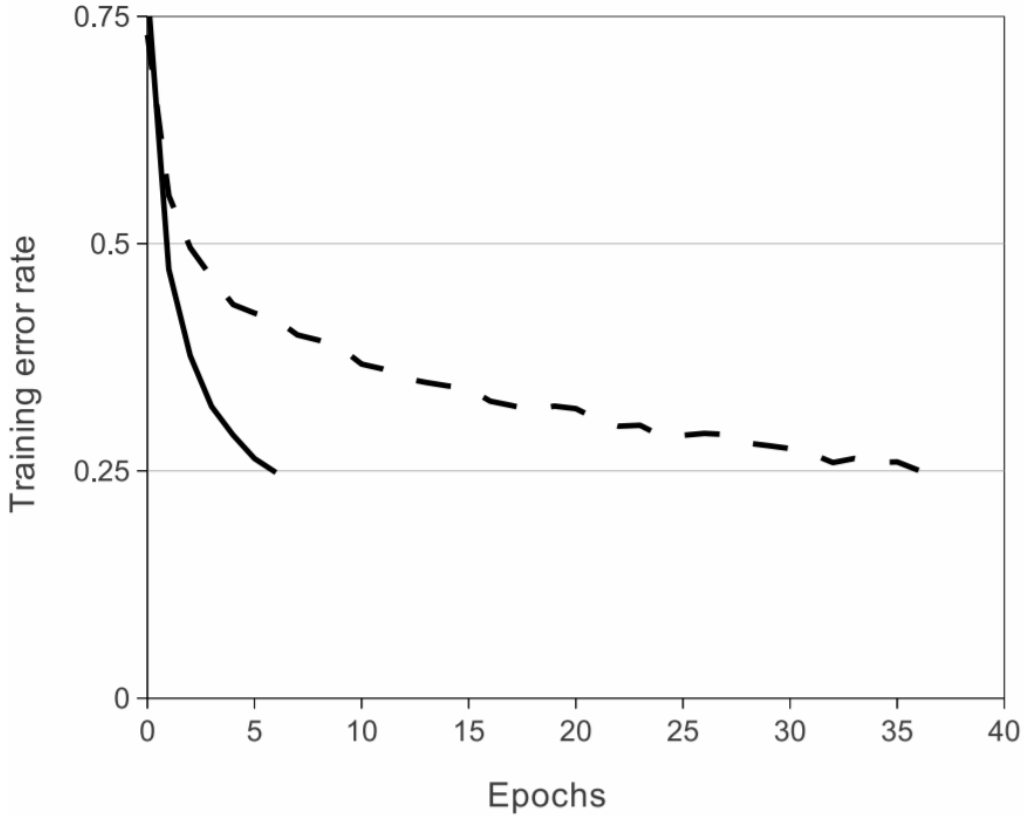
- The ReLU activation solves the vanishing gradient problem.



- In regions where the activation is positive, the derivative is 1.
- As a result, the derivatives do not vanish along paths that contain such "active" neurons even if the network is deep.
- Note that the ReLU is not differentiable at 0 (Software implementations return either 0 or 1 for the derivative at this point).

RECTIFIED LINEAR UNITS (RELU)

- ReLU units can significantly speed up training compared to units with saturating activations.

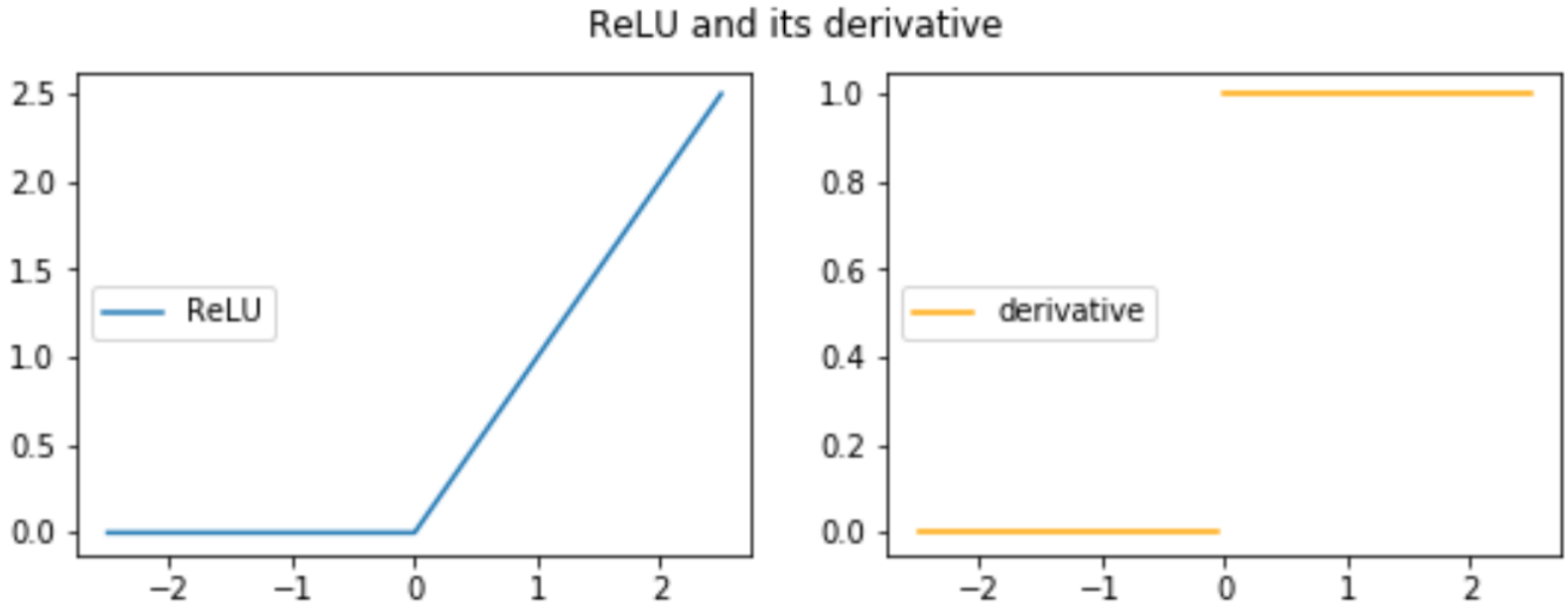


Source : Krizhevsky et al. (2012)

Figure: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on the CIFAR-10 dataset six times faster than an equivalent network with tanh neurons (dashed line).

RECTIFIED LINEAR UNITS (RELU)

- A downside of ReLU units is that when the input to the activation is negative, the derivative is zero. This is known as the "dying ReLU problem".

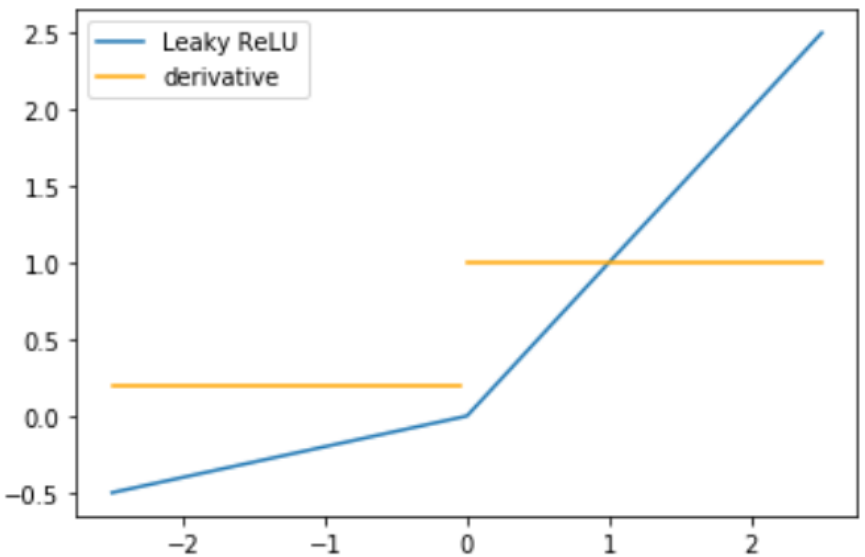


- When a ReLU unit "dies", that is, when its activation is 0 for all datapoints, it kills the gradient flowing through it during backpropogation.
- This means such units are never updated during training and the problem can be irreversible.

GENERALIZATIONS OF RELU

- There exist several generalizations of the ReLU activation that have non-zero derivatives throughout their domains.
- *Leaky ReLU*:

$$LReLU(v) = \begin{cases} v & v \geq 0 \\ \alpha v & v < 0 \end{cases}$$



- Unlike the ReLU, when the input to the Leaky ReLU activation is negative, the derivative is α which is a small positive value (such as 0.01).

GENERALIZATIONS OF RELU

- A variant of the Leaky ReLU is the *Parametric ReLU (PReLU)* which learns the α from the data through backpropagation.
- *Exponential Linear Unit (ELU)*:

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

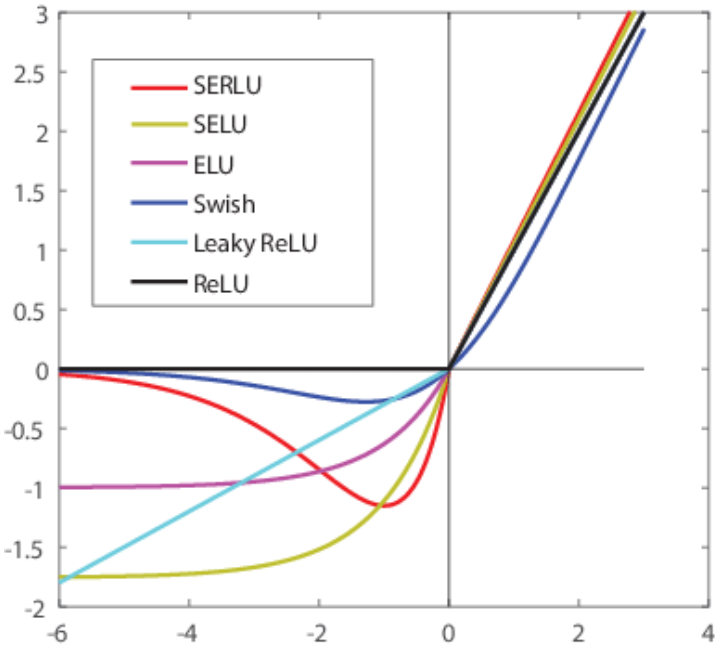
- *Scaled Exponential Linear Unit (SELU)*:

$$SELU(v) = \lambda \begin{cases} v & v \geq 0 \\ \alpha(e^v - 1) & v < 0 \end{cases}$$

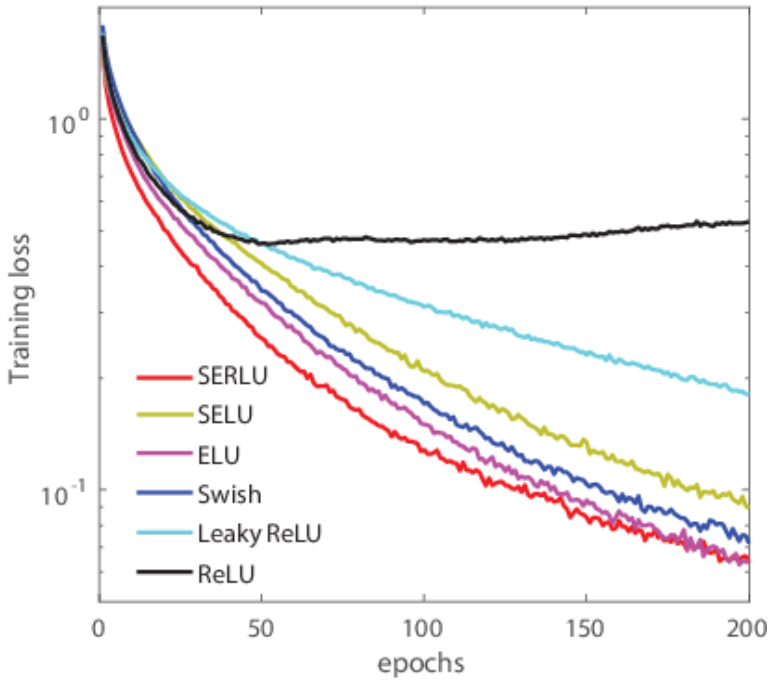
Note: In ELU and SELU, α and λ are hyperparameters that are set before training.

- These generalizations may perform as well as or better than the ReLU on some tasks.

GENERALIZATIONS OF RELU



(a): Different activation functions



(b): Performance on CIFAR10 without dropout

Source : Zhang et al. 2018

Output activations

OUTPUT ACTIVATIONS

- As we have seen previously, the role of the output activation is to get the final score on the same scale as the target.
- The output activations and the loss functions used to train neural networks can be viewed through the lens of maximum likelihood estimation (MLE).
- In general, the function $f(\mathbf{x} \mid \theta)$ represented by the neural network defines the conditional $p(y \mid \mathbf{x}, \theta)$ in a supervised learning task.
- Maximizing the likelihood is then equivalent to minimizing $-\log p(y \mid \mathbf{x}, \theta)$.
- An output unit with the identity function as the activation can be used to represent the mean of a Gaussian distribution.
- For such a unit, training with mean-squared error is equivalent to maximizing the log-likelihood (ignoring issues with non-convexity).

OUTPUT ACTIVATIONS

- Similarly, sigmoid and softmax units can output the parameter(s) of a Bernoulli distribution and Categorical distribution, respectively.
- It is straightforward to show that when the label is one-hot encoded, training with the cross-entropy loss is equivalent to maximizing log-likelihood. ▸ Click here
- Because these activations can saturate, an important advantage of maximizing log-likelihood is that the log undoes some of the exponentiation in the activation functions which is desirable when optimizing with gradient-based methods.
- For example, in the case of softmax, the loss is:

$$L(y, f(\mathbf{x})) = -f_{in,k} + \log \sum_{k'=1}^{g} \exp(f_{in,k'})$$

where k is the correct class. The first term, $-f_{in,k}$, does not saturate which means training can progress steadily even if the contribution of $f_{in,k}$ to the second term is negligible.

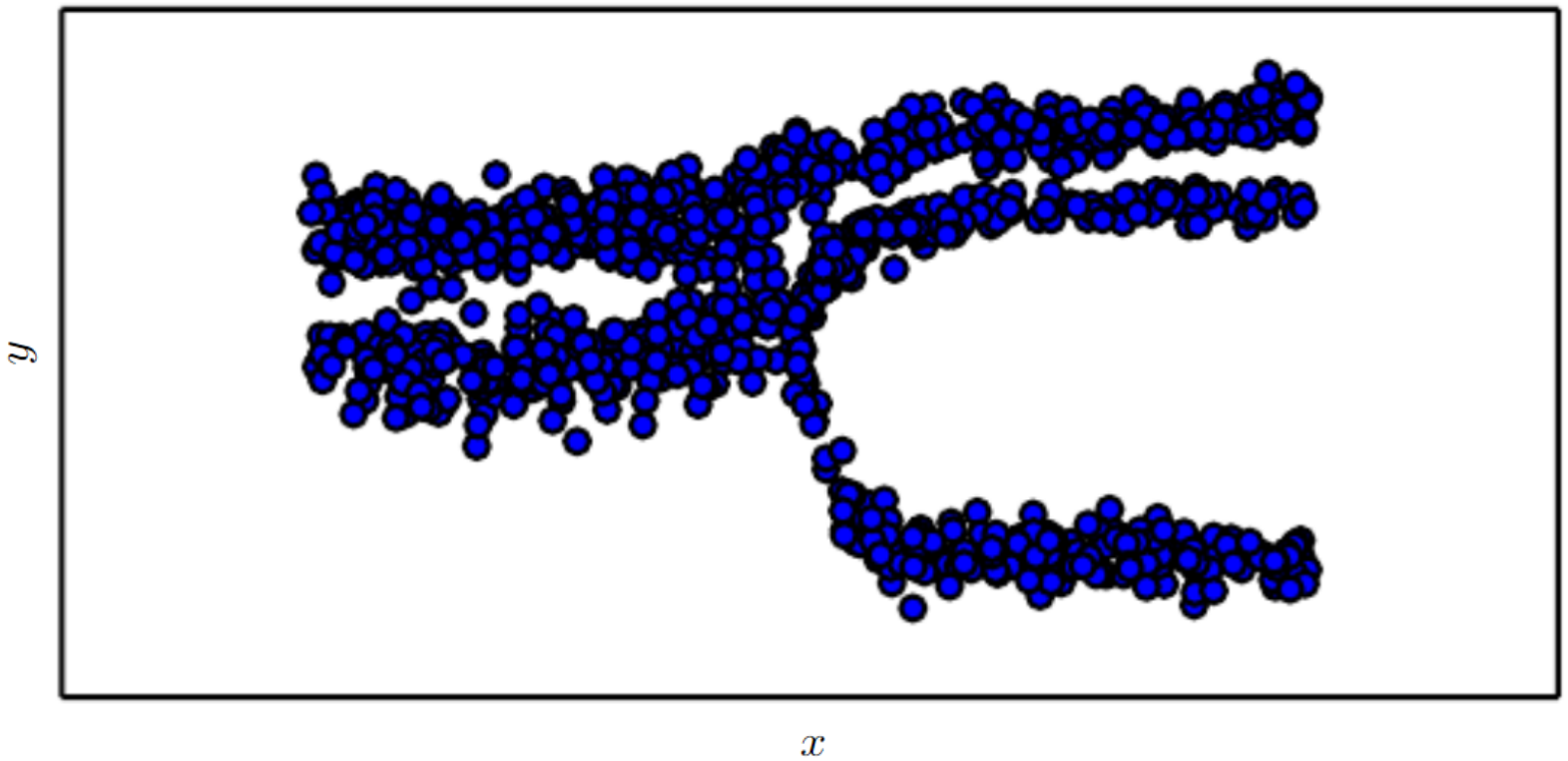
OUTPUT ACTIVATIONS

- A neural network can even be used to output the parameters of more complex distributions.
- A Mixture Density Network, for example, outputs the parameters of a Gaussian Mixture Model:

$$p(y|\mathbf{x}) = \sum_{c=1}^{m} \phi^{(c)}(\mathbf{x})\, \mathcal{N}\left(y; \boldsymbol{\mu}^{(c)}(\mathbf{x}), \boldsymbol{\Sigma}^{(c)}(\mathbf{x})\right)$$

  where m the number of components in the mixture.
- In such a network, the output units are divided into groups.
- One group of output neurons with softmax activation represents the weights ($\phi^{(c)}$) of the mixture.
- Another group with the identity activation represents the means ($\boldsymbol{\mu}^{(c)}$) and yet another group with a non-negative activation function (such as ReLU or the exponential function) can represent the variances of the (typically) diagonal covariance matrices $\boldsymbol{\Sigma}^{(c)}$.

OUTPUT ACTIVATIONS



Source : Goodfellow et al. (2016)

Figure: Samples drawn from a Mixture Density Network. The input **x** is sampled from a uniform distribution and y is sampled from $p(y \mid \mathbf{x}, \theta)$.

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/

Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton (2012)

ImageNet Classification with Deep Convolutional Neural Networks

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Guoqiang Zhang and Haopeng Li (2018)

Effectiveness of Scaled Exponentially-Regularized Linear Units

https://arxiv.org/abs/1807.10117