(Artificial) Neural Networks: Advanced

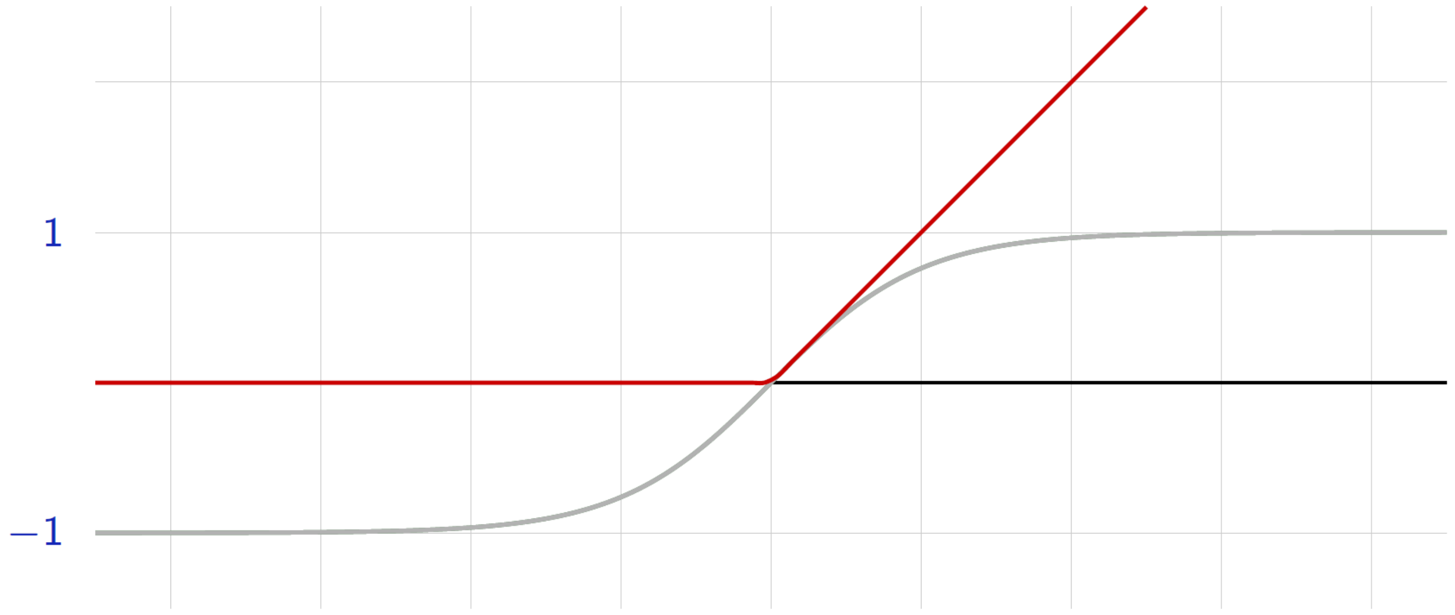
Industrial AI Lab.

Prof. Seungchul Lee

Nonlinear Activation Function

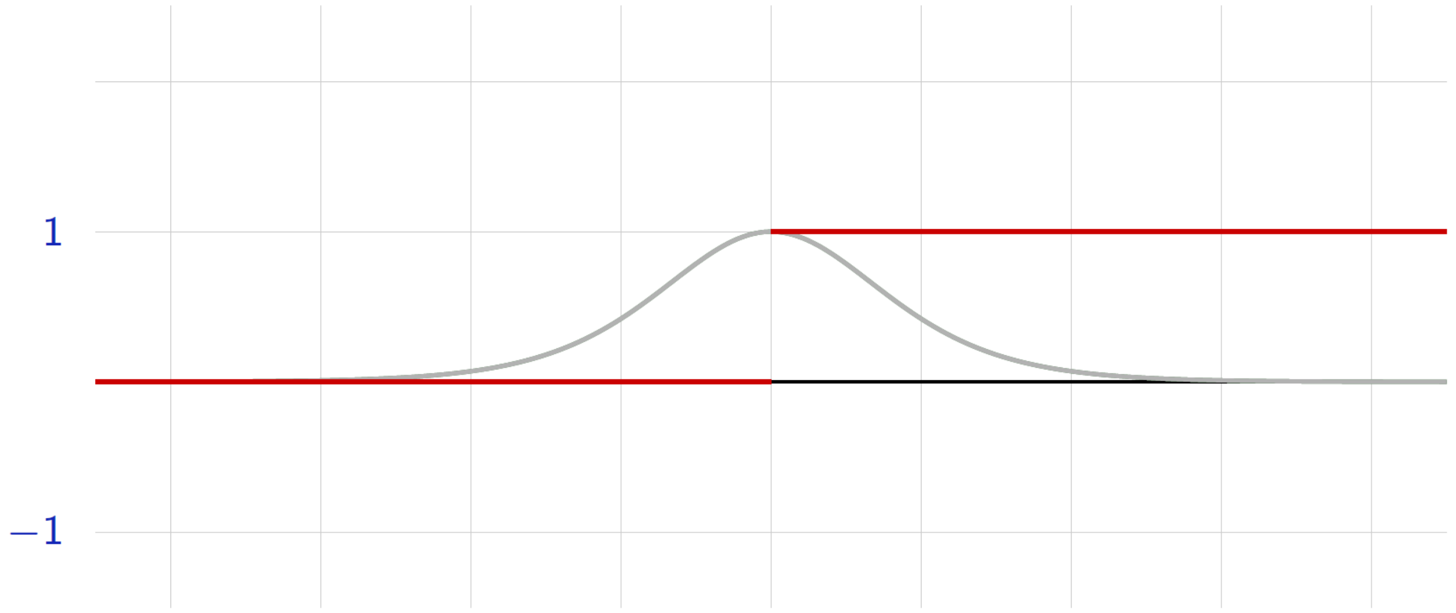
Rectifiers

- The use of the ReLU activation function was a great improvement compared to the historical tanh.



Rectifiers

- This can be explained by the derivative of ReLU itself not vanishing, and by the resulting coding being sparse (Glorot et al., 2011).



Rectifiers

- The steeper slope in the loss surface speeds up the training.

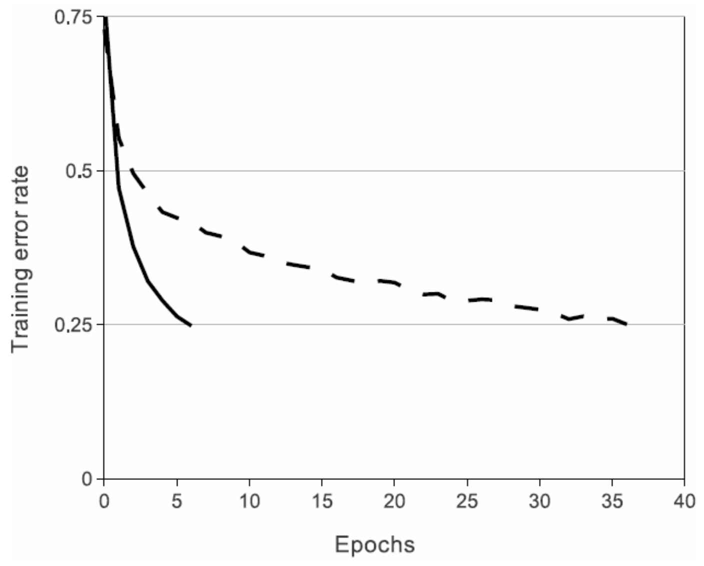


Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

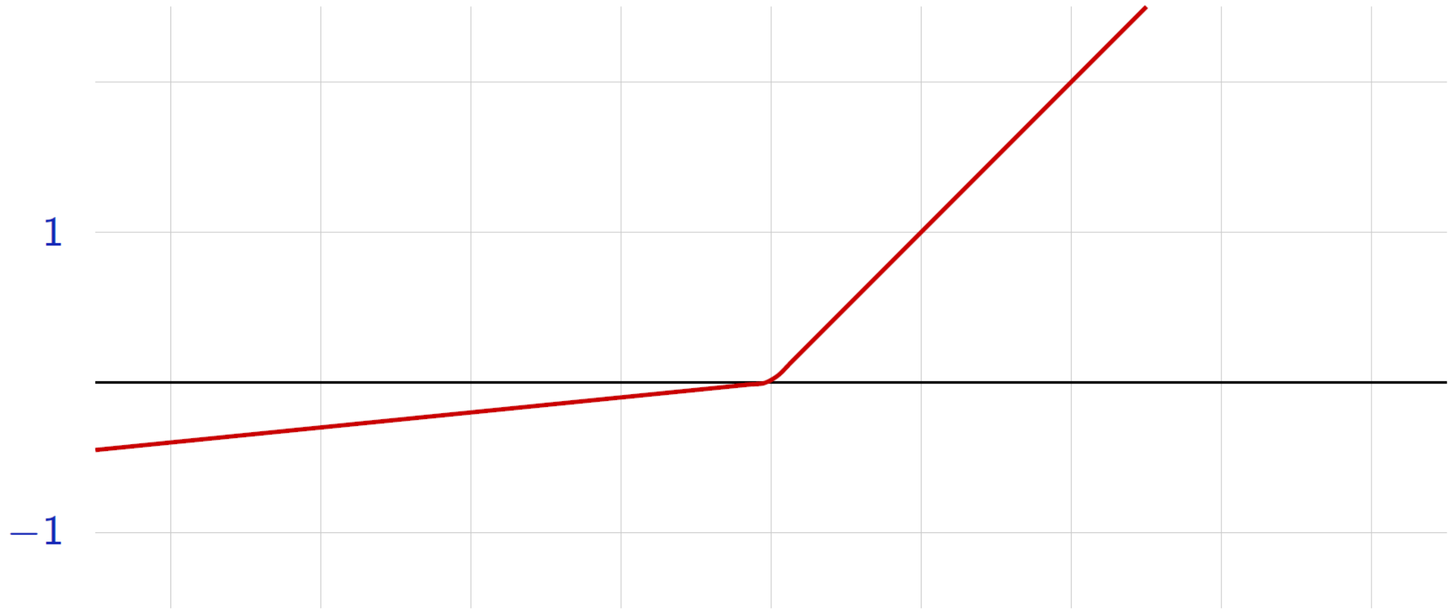
- A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classication with deep convolutional neural networks. In Neural Information Processing Systems (NIPS), 2012.

A Variant of ReLU

- A First variant of ReLU is Leaky-ReLU

$$\begin{aligned} \mathbb{R} &\to \mathbb{R} \\ x &\mapsto \max(ax, x) \end{aligned}$$

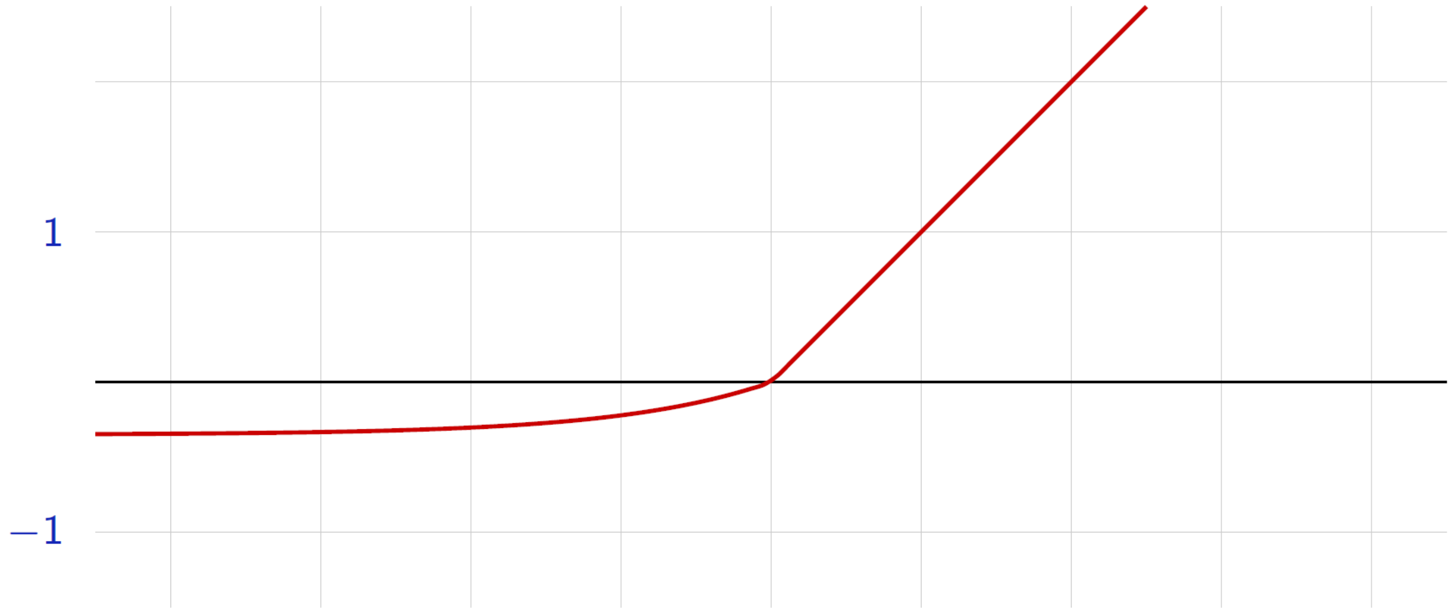
with $0 \leq a < 1$ usually small.



A Variant of ReLU

- Clevert et al. (2015) proposed the exponential linear unit (ELU), with an exponential saturation

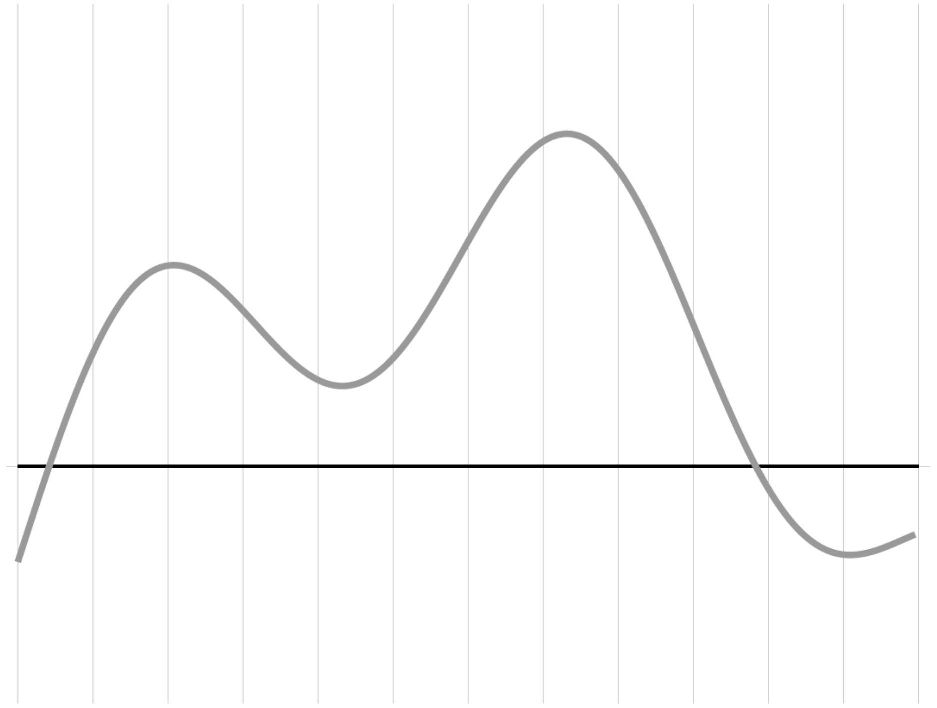
$$x \mapsto \begin{cases} x & \text{if } x \geq 0 \\ \alpha\left(e^x - 1\right) & \text{otherwise.} \end{cases}$$



D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). CoRR, abs/1511.07289, 2015

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

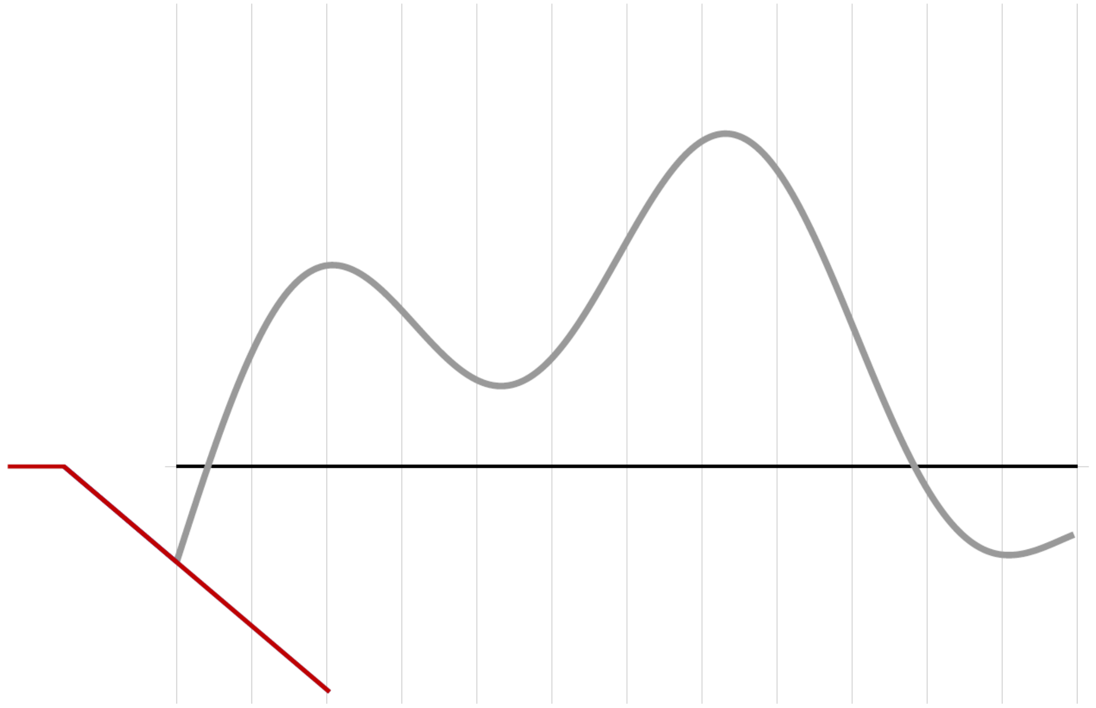


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1)$$

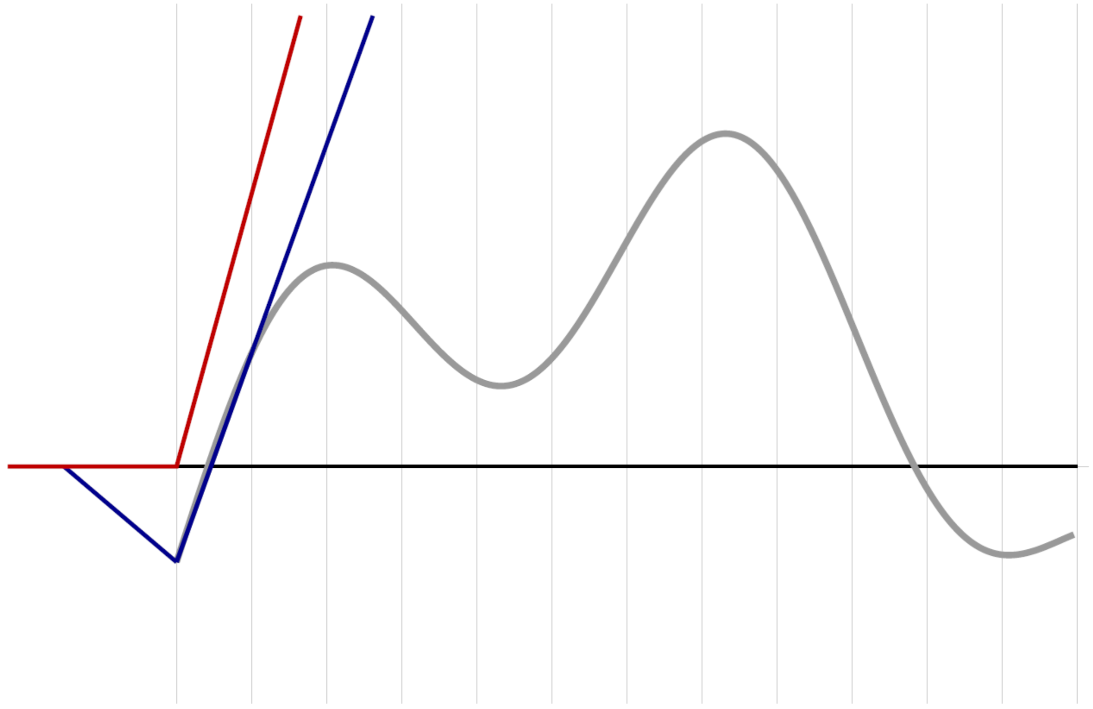


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2)$$

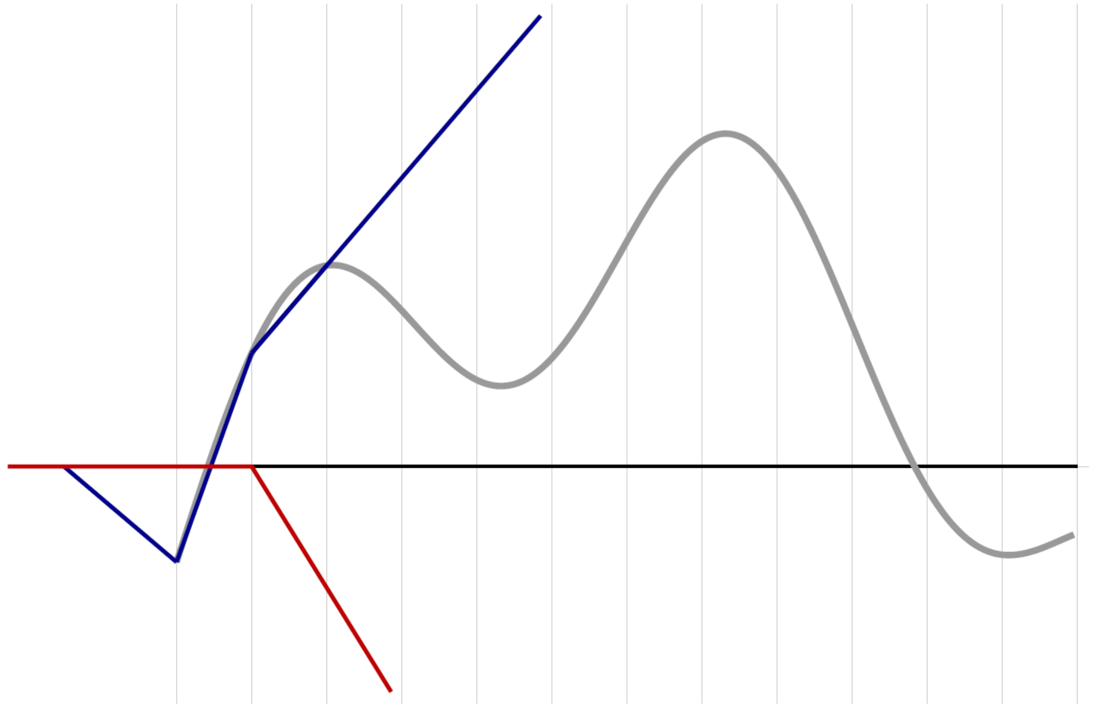


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3)$$



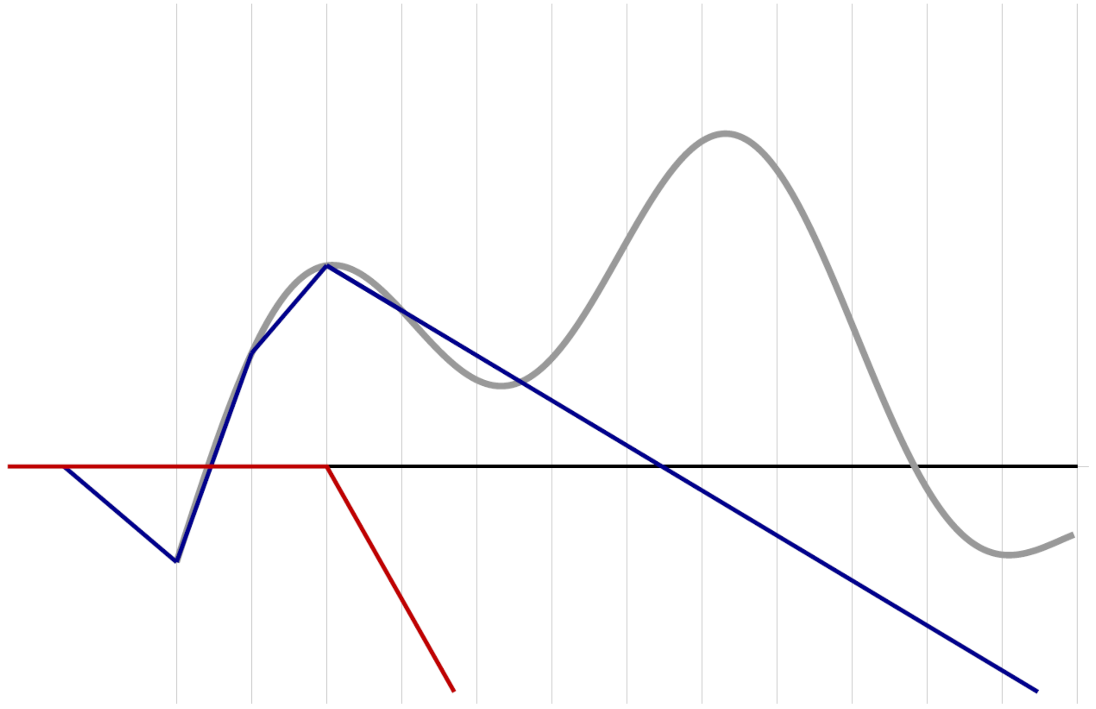
- Piece-wise linear function

POSTECH

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$

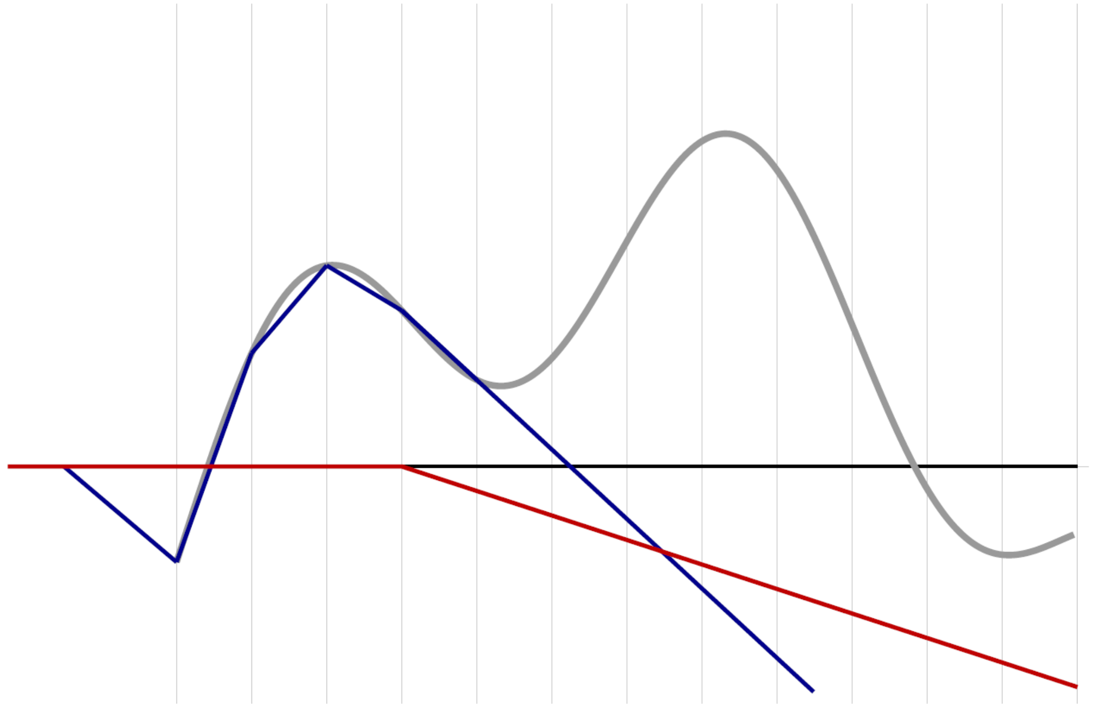


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$

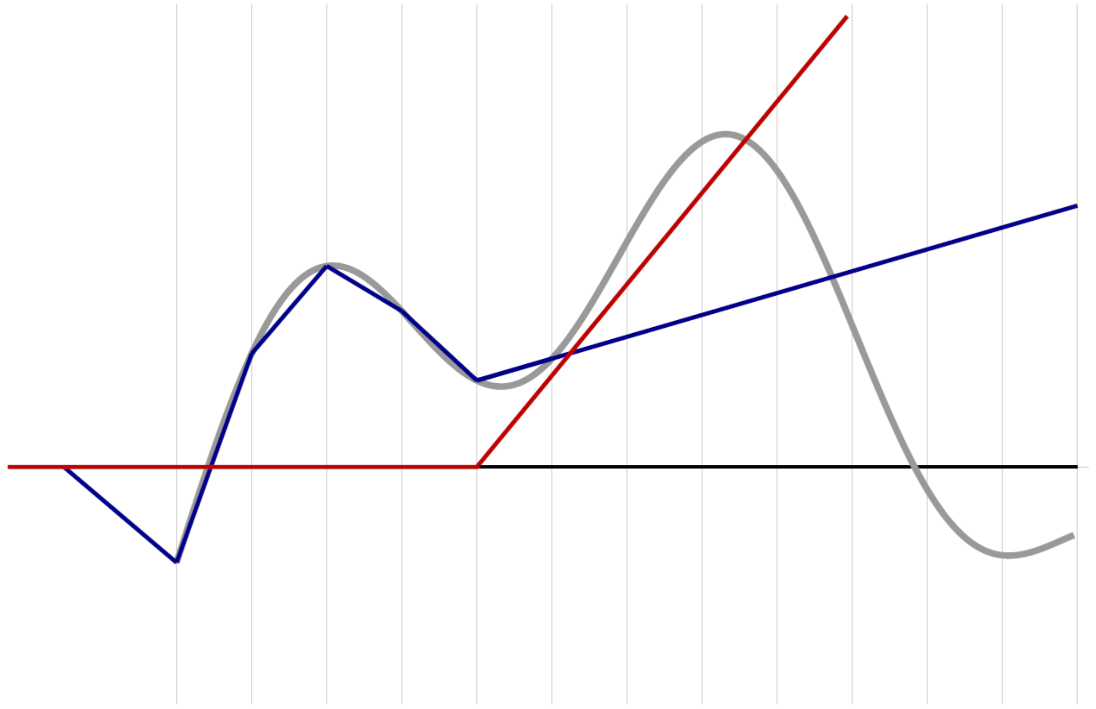


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$

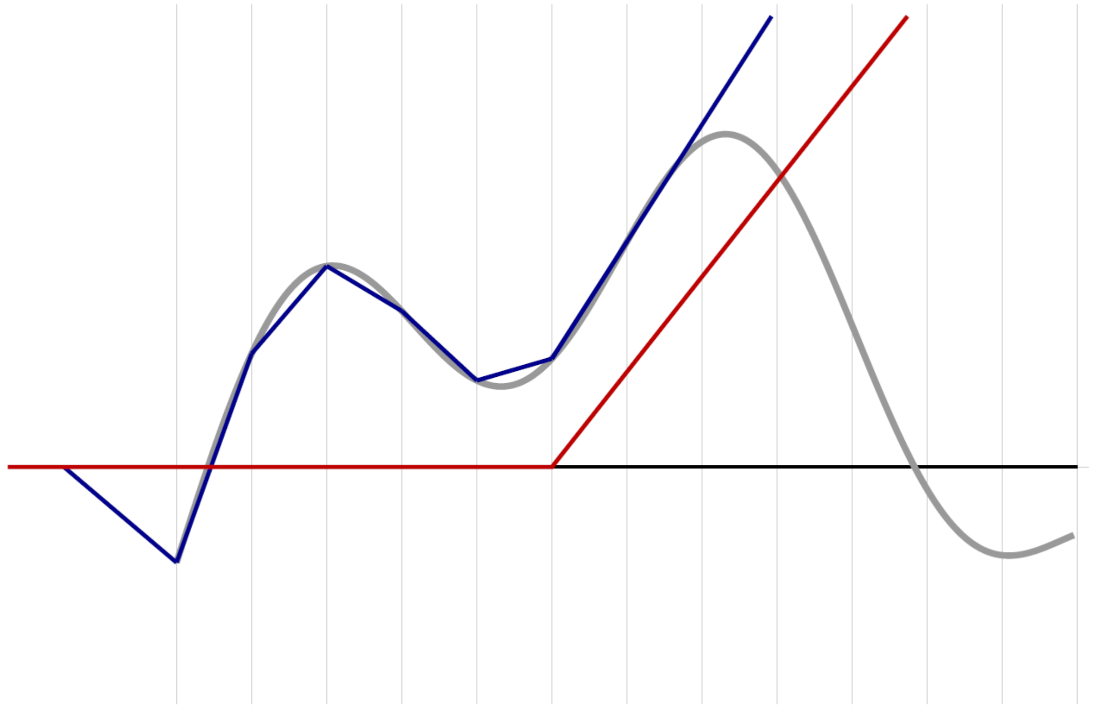


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$

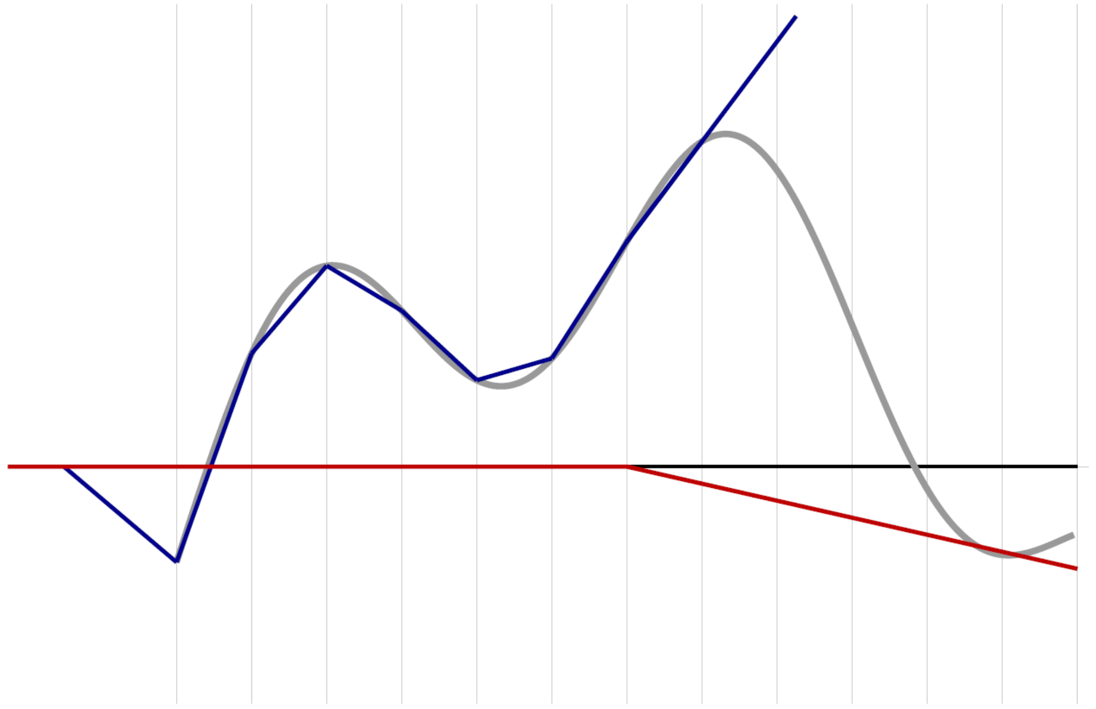


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$

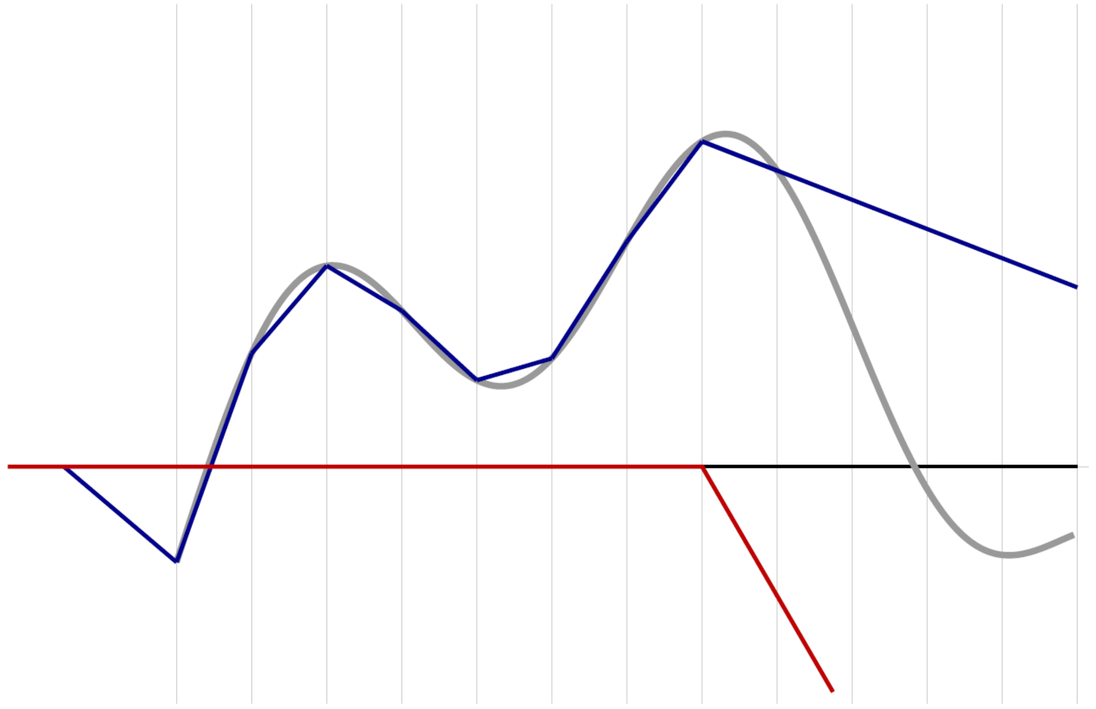


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$

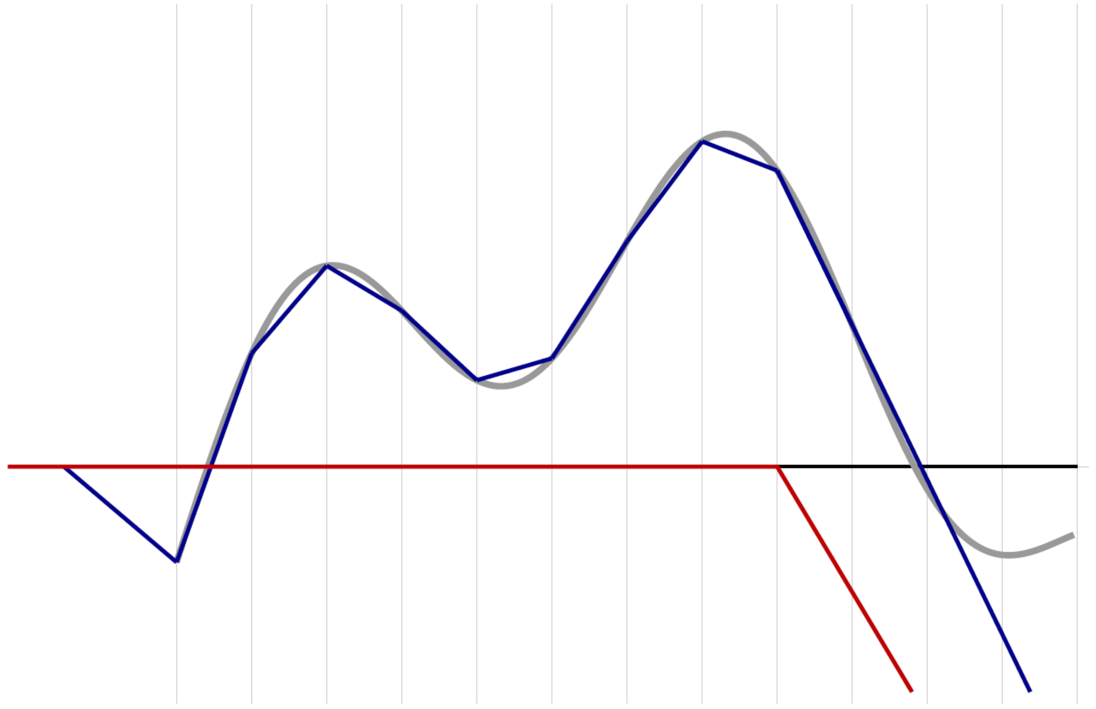


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$

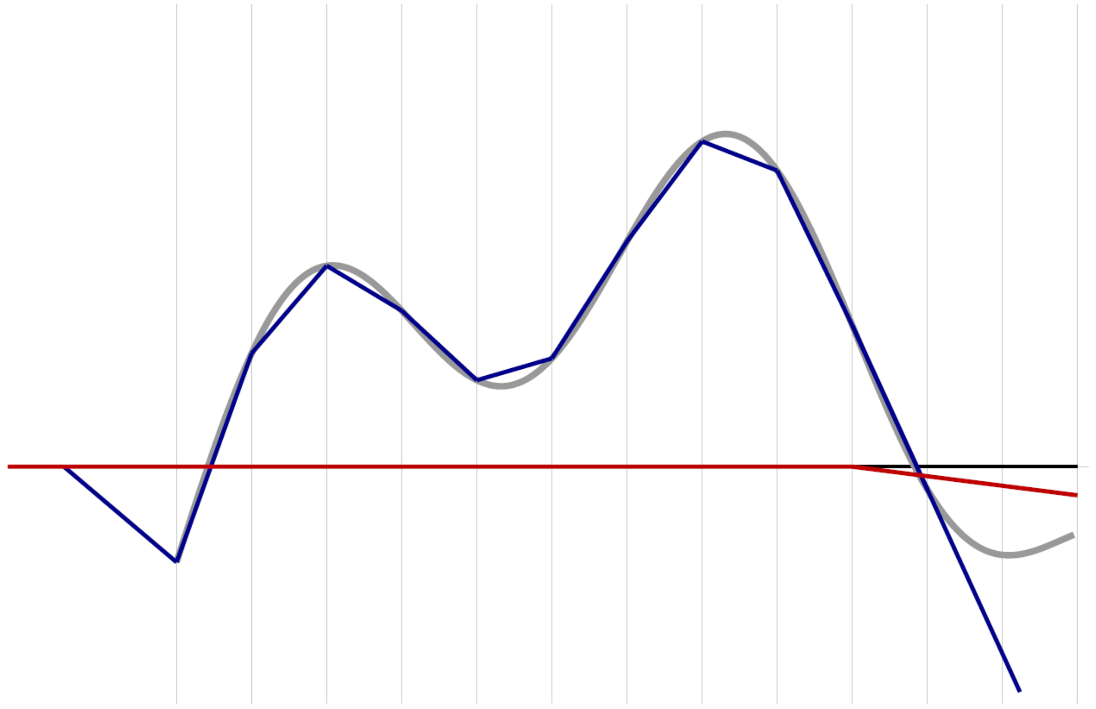


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$

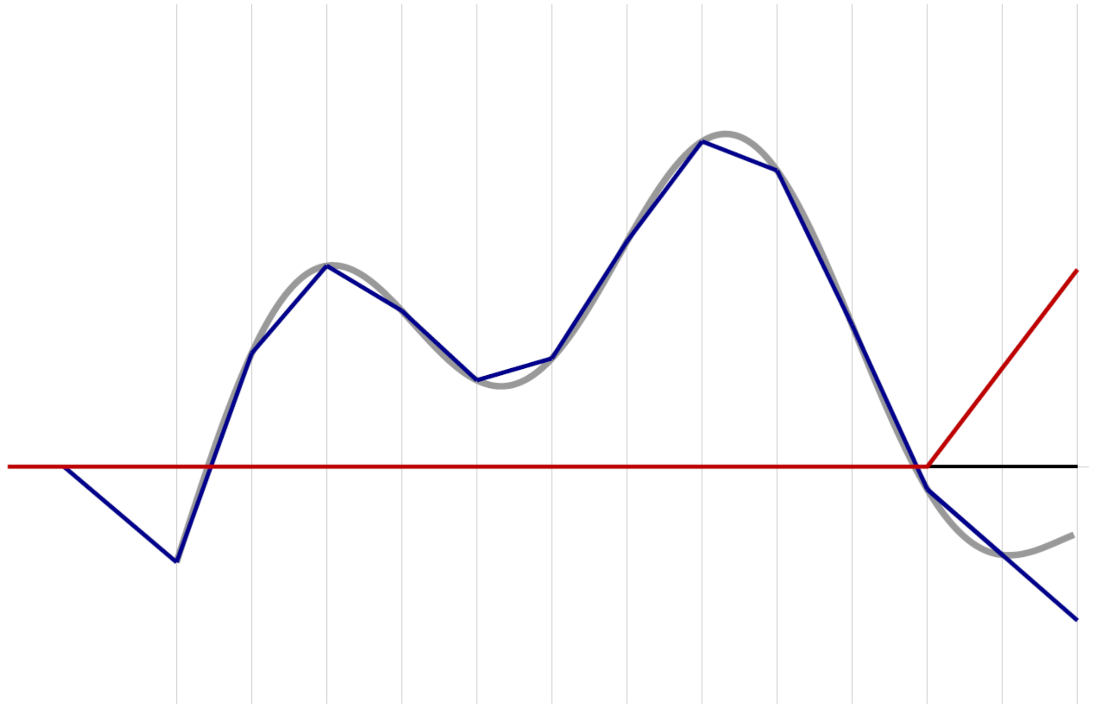


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$

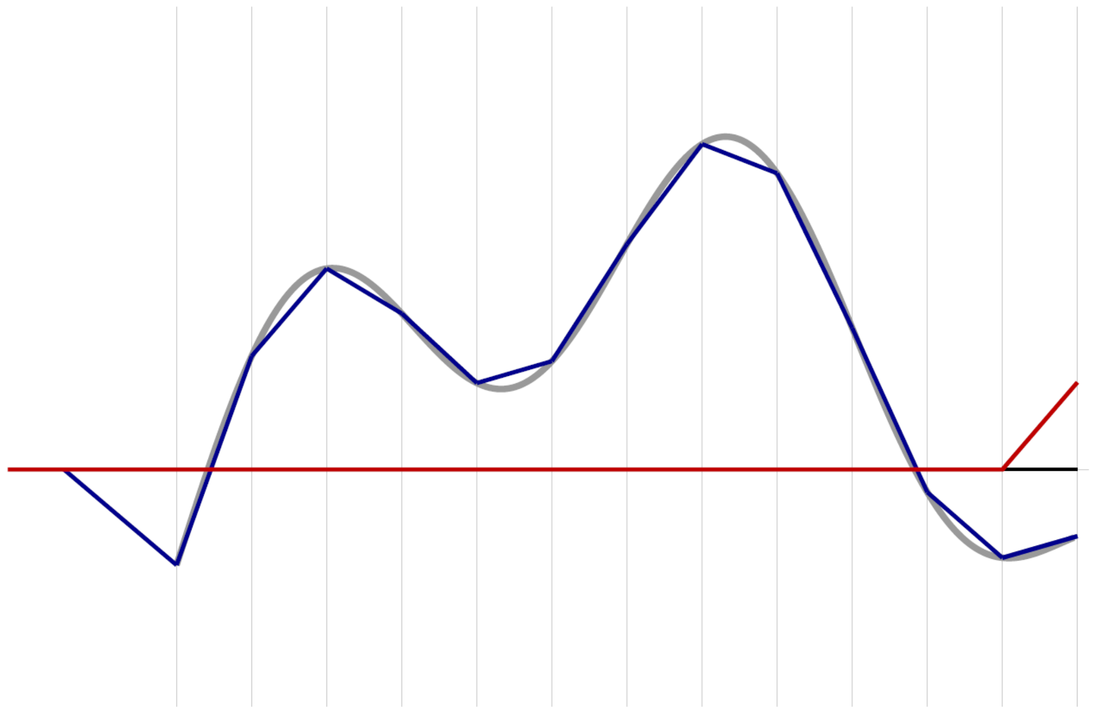


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$

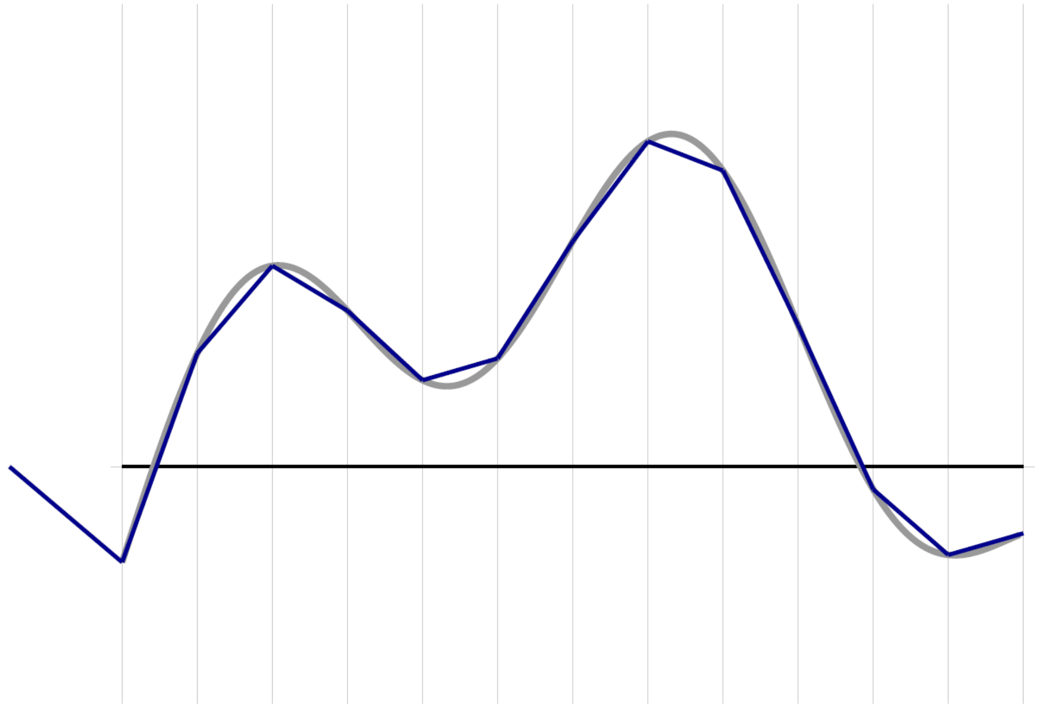


- Piece-wise linear function

Universal Approximation: Regression

- We can approximate any $\varphi \in \psi([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



- Piece-wise linear function

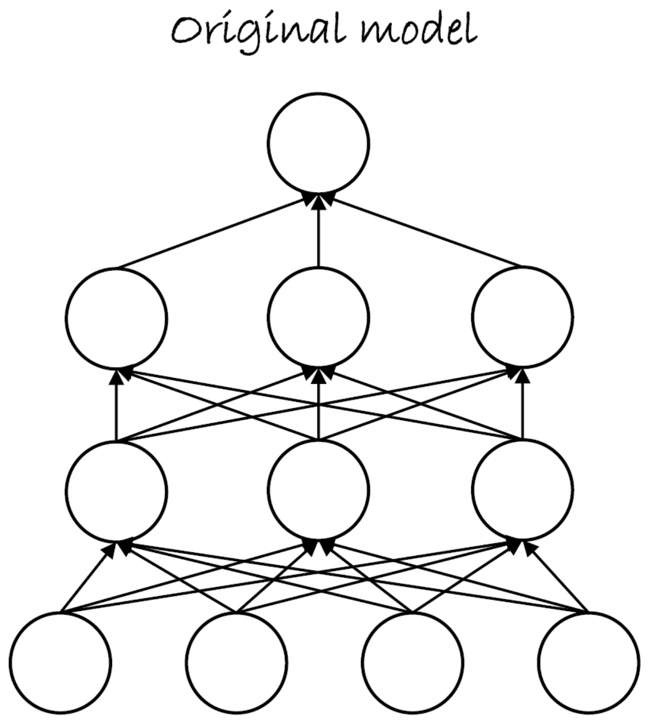
Dropout

Dropout (Srivastava, 2014)

- During training setting activations randomly to 0
  - Neurons sampled at random from a Bernoulli distribution with p = 0.5
- At test time all neurons are used
  - Neuron activations reweighted by p
- Benefits
  - Reduces complex co-adaptations or co-dependencies between neurons
  - No “free-rider” neurons that rely on others
  - Every neuron becomes more robust
  - Decreases significantly overfitting
  - Improves significantly training speed

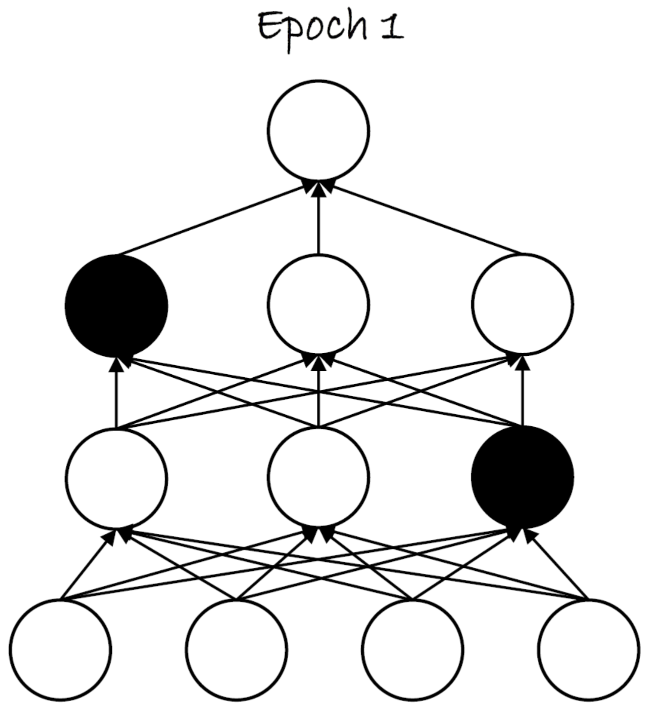
Dropout Illustration

- Effectively, a different architecture at every training epoch
- It can also be thought of as an ensemble technique in machine learning.



Dropout Illustration

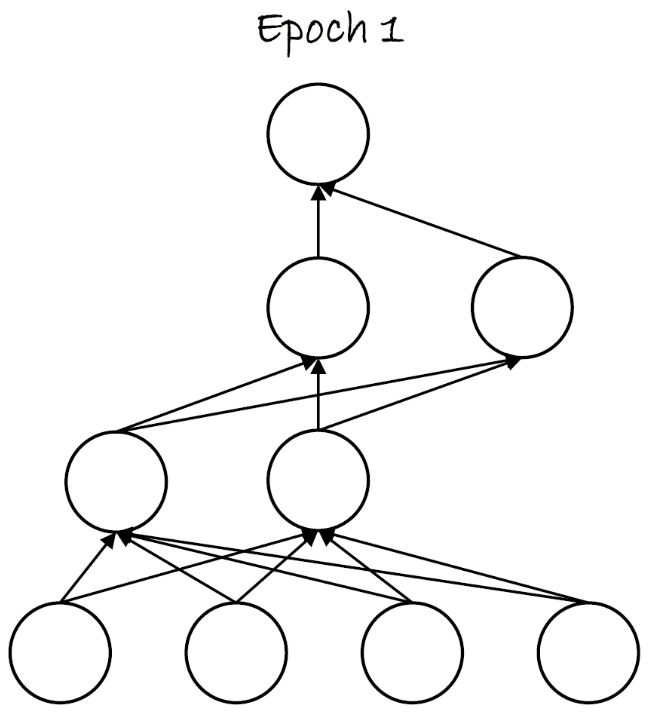
- Effectively, a different architecture at every training epoch
- It can also be thought of as an ensemble technique in machine learning.



tf.nn.dropout(hidden, p = 1/3)

Dropout Illustration

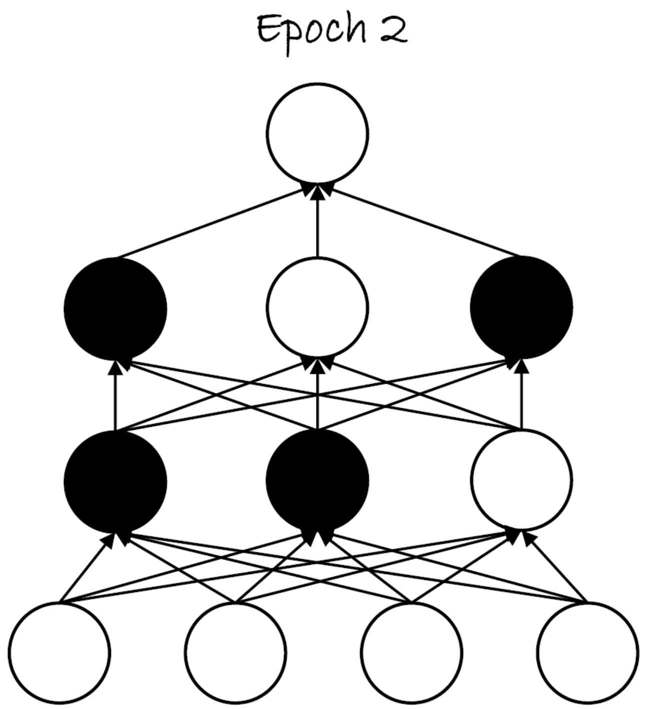
- Effectively, a different architecture at every training epoch
- It can also be thought of as an ensemble technique in machine learning.



tf.nn.dropout(hidden, p = 1/3)

Dropout Illustration

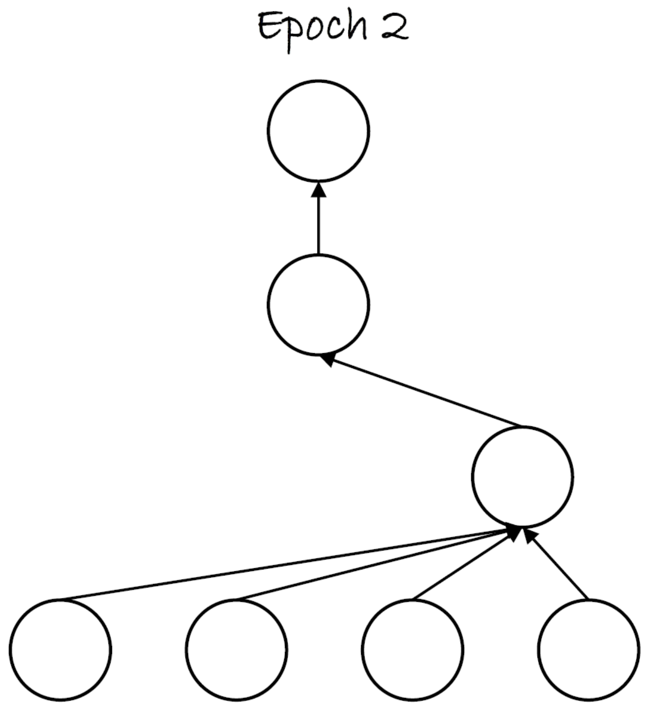
- Effectively, a different architecture at every training epoch
- It can also be thought of as an ensemble technique in machine learning.



tf.nn.dropout(hidden, p = 2/3)

Dropout Illustration

- Effectively, a different architecture at every training epoch
- It can also be thought of as an ensemble technique in machine learning.



tf.nn.dropout(hidden, p = 2/3)

Dropout as Regularization

- A first “deep” regularization technique is dropout (Srivastava et al., 2014). It consists of removing units at random during the forward pass on each sample, and putting them all back during test.
- This is a specific way of adding noise.
- It’s a recent discovery and it works very, very well.

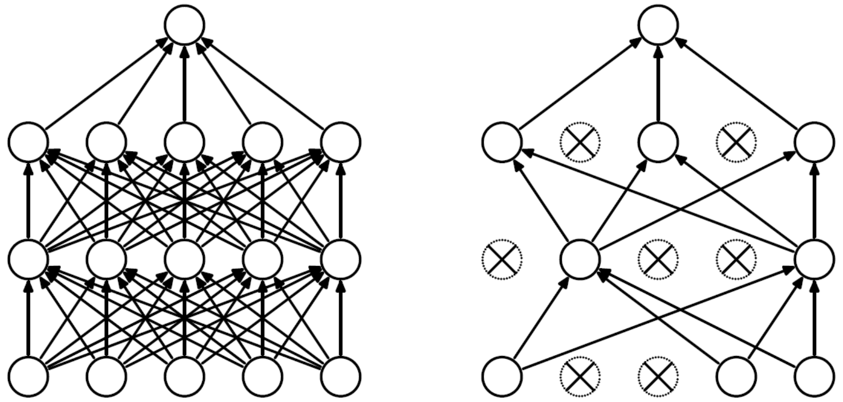


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

(Srivastava et al., 2014)

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research (JMLR), 15:1929-1958, 2014.

Dropout as Regularization

- Dropout: a simple way to prevent neural networks from overfitting
- This method increases independence between units, and distributes the representation. It generally improves performance.

"In a standard neural network, the derivative received by each parameter tells it how it should change so the final loss function is reduced, given what all other units are doing. Therefore, units may change in a way that they fix up the mistakes of the other units. This may lead to complex co-adaptations. This in turn leads to overfitting because these co-adaptations do not generalize to unseen data. **We hypothesize that for each hidden unit, dropout prevents co-adaptation by making the presence of other hidden units unreliable.** Therefore, a hidden unit cannot rely on other specific units to correct its mistakes. It must perform well in a wide variety of different contexts provided by the other hidden units."

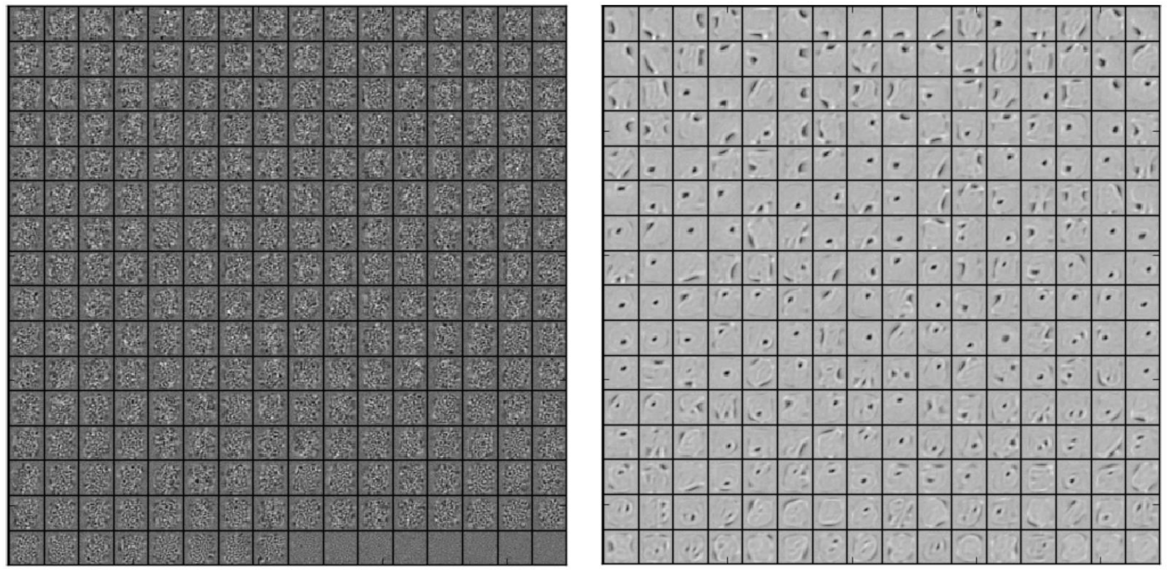
(Srivastava et al., 2014)

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research (JMLR), 15:1929-1958, 2014.

Dropout as Regularization

- One has to decide on which units/layers to use dropout, and with what probability p units are dropped.

- During training, for each sample, as many Bernoulli variables as units are sampled independently to select units to remove.



(a) Without dropout (b) Dropout with $p = 0.5$.

Figure 7: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

(Srivastava et al., 2014)

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research (JMLR), 15:1929-1958, 2014.

Batch Normalization

Batch Normalization

- Batch normalization is a technique for improving the performance and stability of artificial neural networks.
- It is used to normalize the input layer by adjusting and scaling the activations.

- “Training Deep Neural Networks is complicated by the fact that the distribution of each layer’s inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization.”

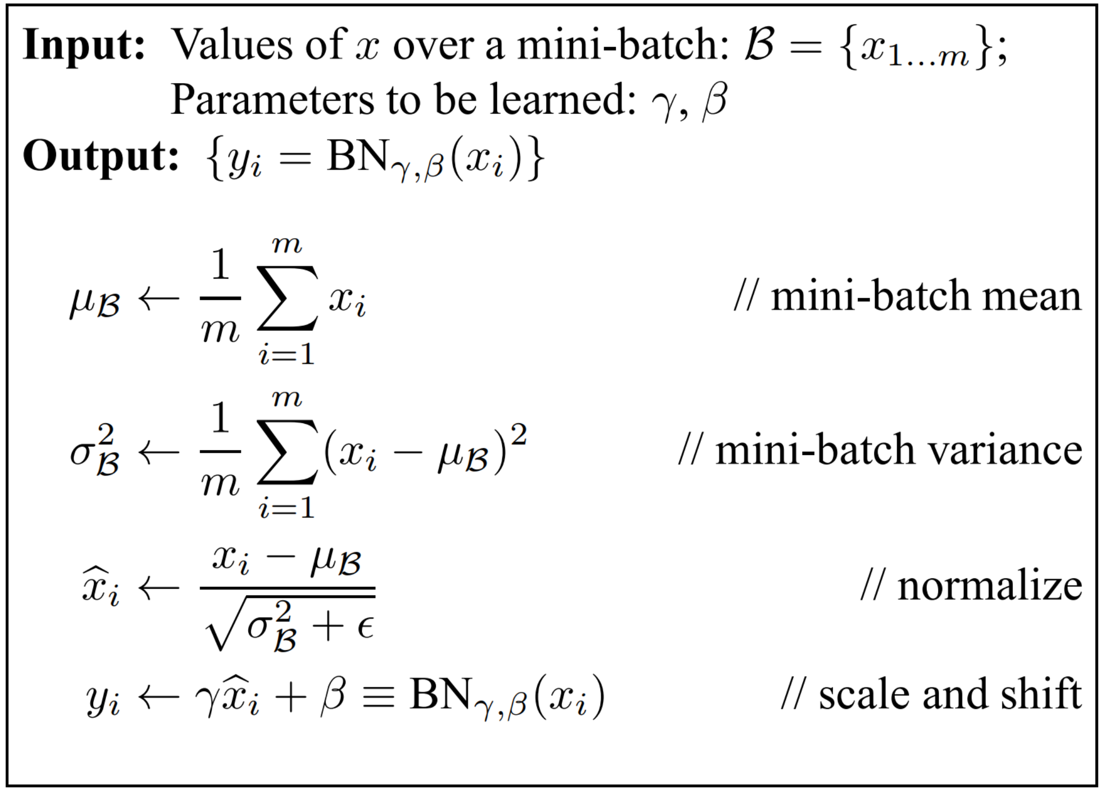
- Batch normalization can be done anywhere in a deep architecture, and forces the activations’ first and second order moments, so that the following layers do not need to adapt to their drift.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (ICML), 2015.

Batch Normalization

- During training batch normalization shifts and rescales according to the mean and variance estimated on the batch.

- During test, it simply shifts and rescales according to the empirical moments estimated during training.



```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma$, $\beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                                  // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$        // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$      // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$        // scale and shift
```

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (ICML), 2015.