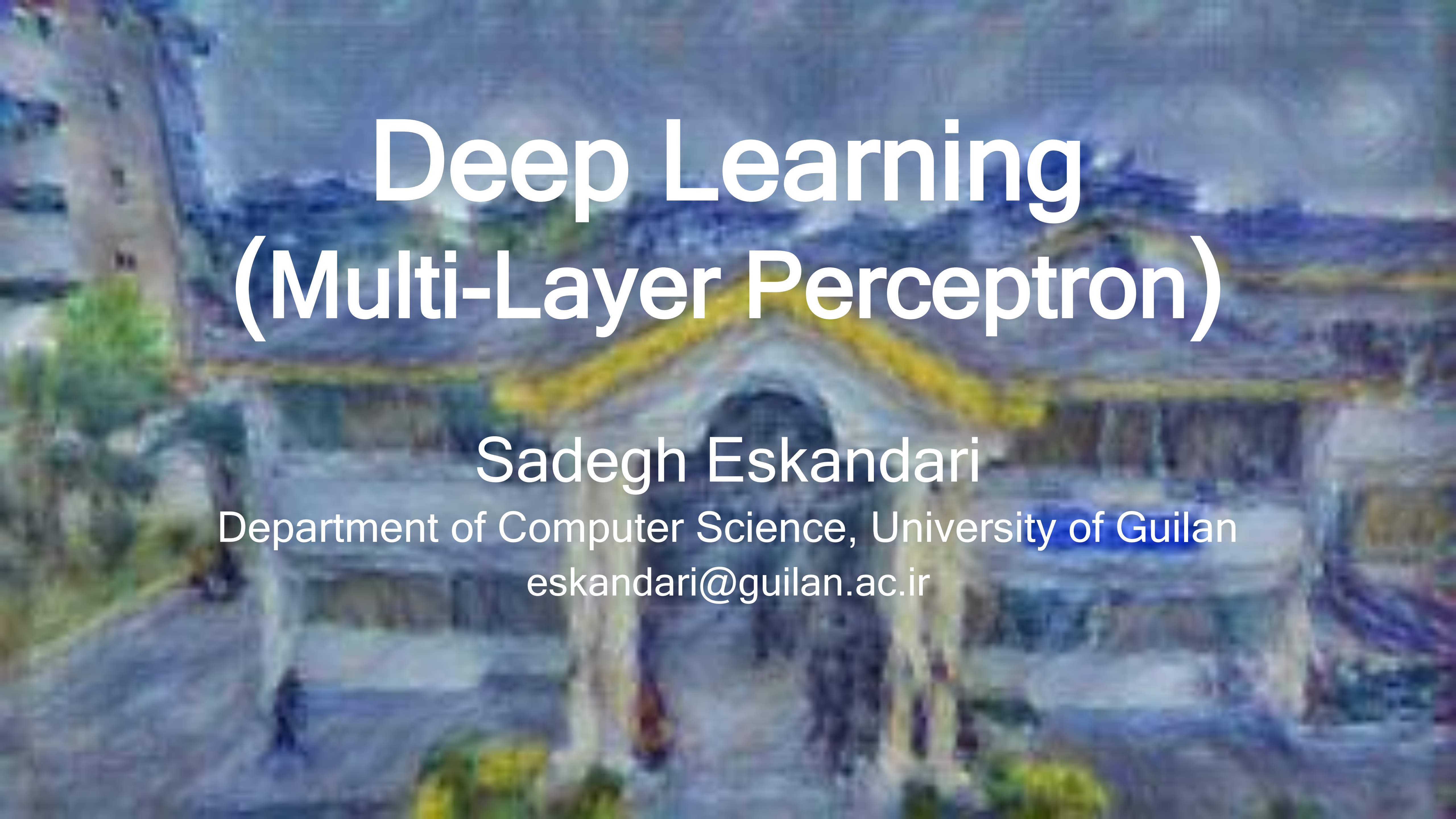


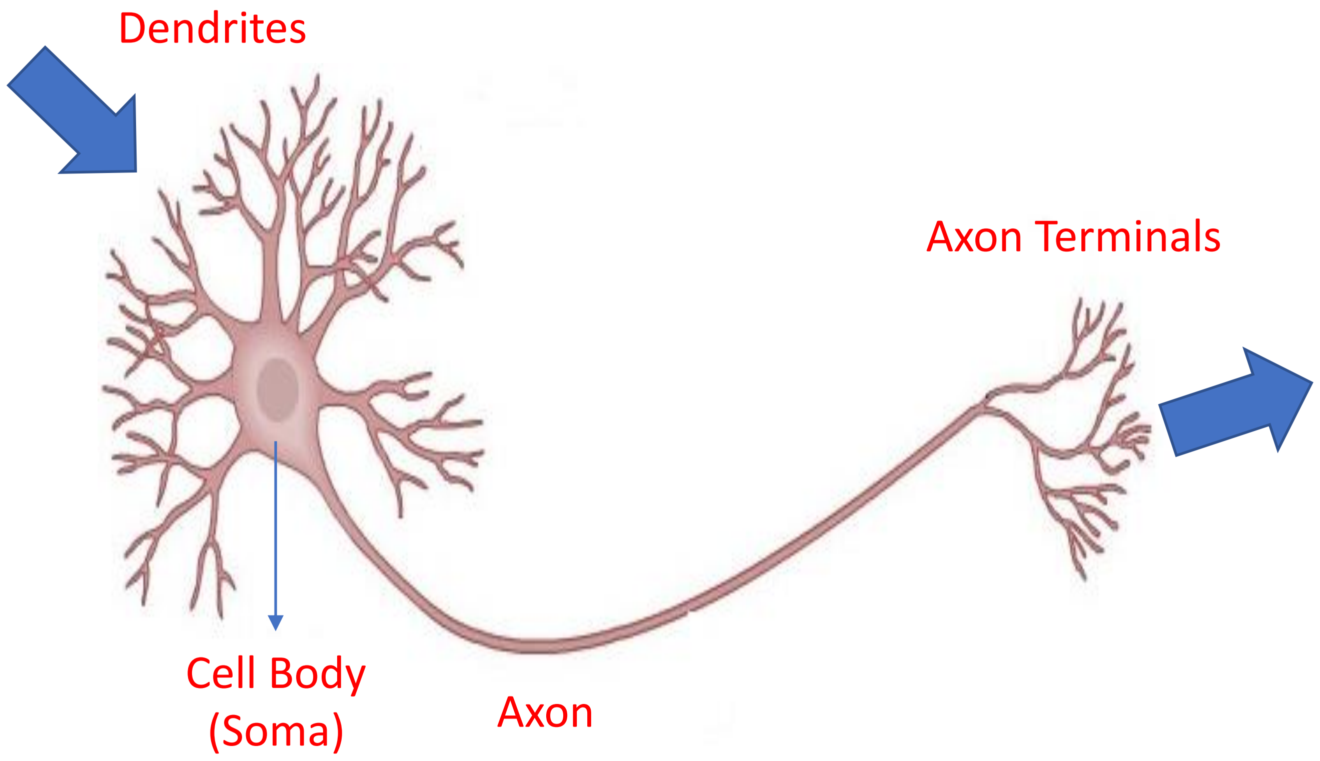
Deep Learning (Multi-Layer Perceptron)

Sadegh Eskandari
Department of Computer Science, University of Guilan
eskandari@guilan.ac.ir

Today …

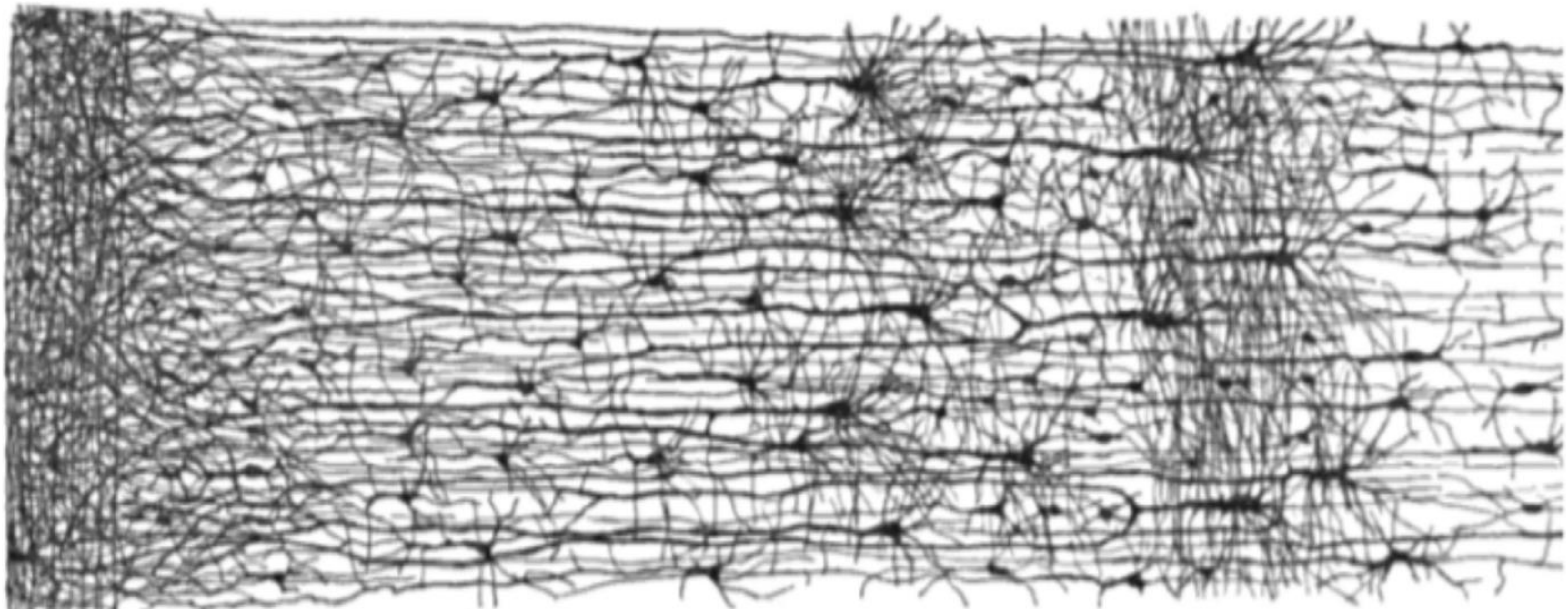
- Biological Neurons
- Artificial Neurons
- Multi-Layer Perceptron
- Training a Single Neuron
- Training an MLP (Backpropagation)
- Activation Functions
- Regularization
- Weight Initialization

Biological Neurons



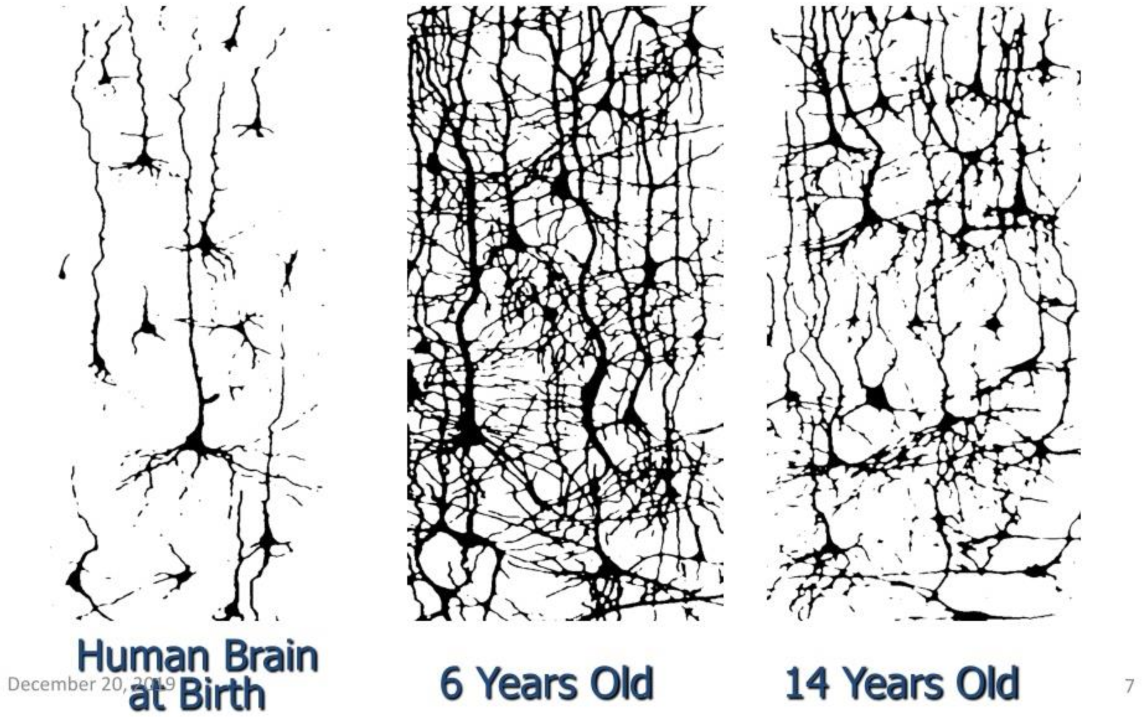
- Dendrites receive impulses (synapses) from other cells.
- The cell body fires (sends signal) if the received impulses are more than a threshold.
- Signal travels through the axon and is delivered to other cells via axon terminals

Biological Neurons


Multiple layers in a biological neural network of human cortex

Source: Montesinos López, O.A., Montesinos López, A. and Crossa, J., 2022. Fundamentals of Artificial Neural Networks and Deep Learning. In Multivariate Statistical Machine Learning Methods for Genomic Prediction (pp. 379-425). Springer, Cham.

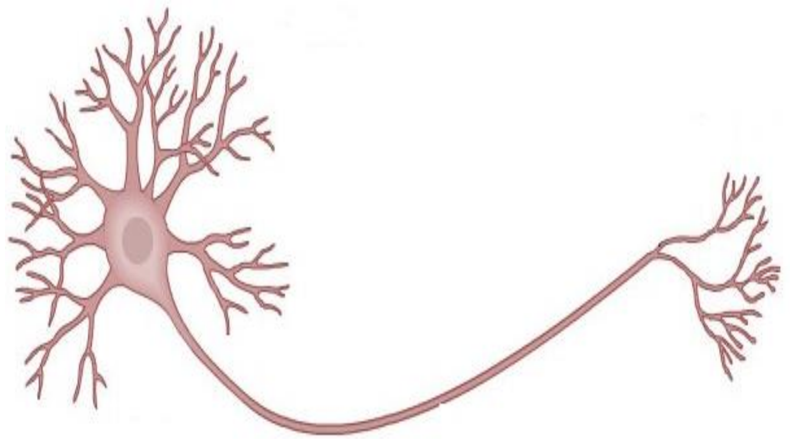
Biological Neurons

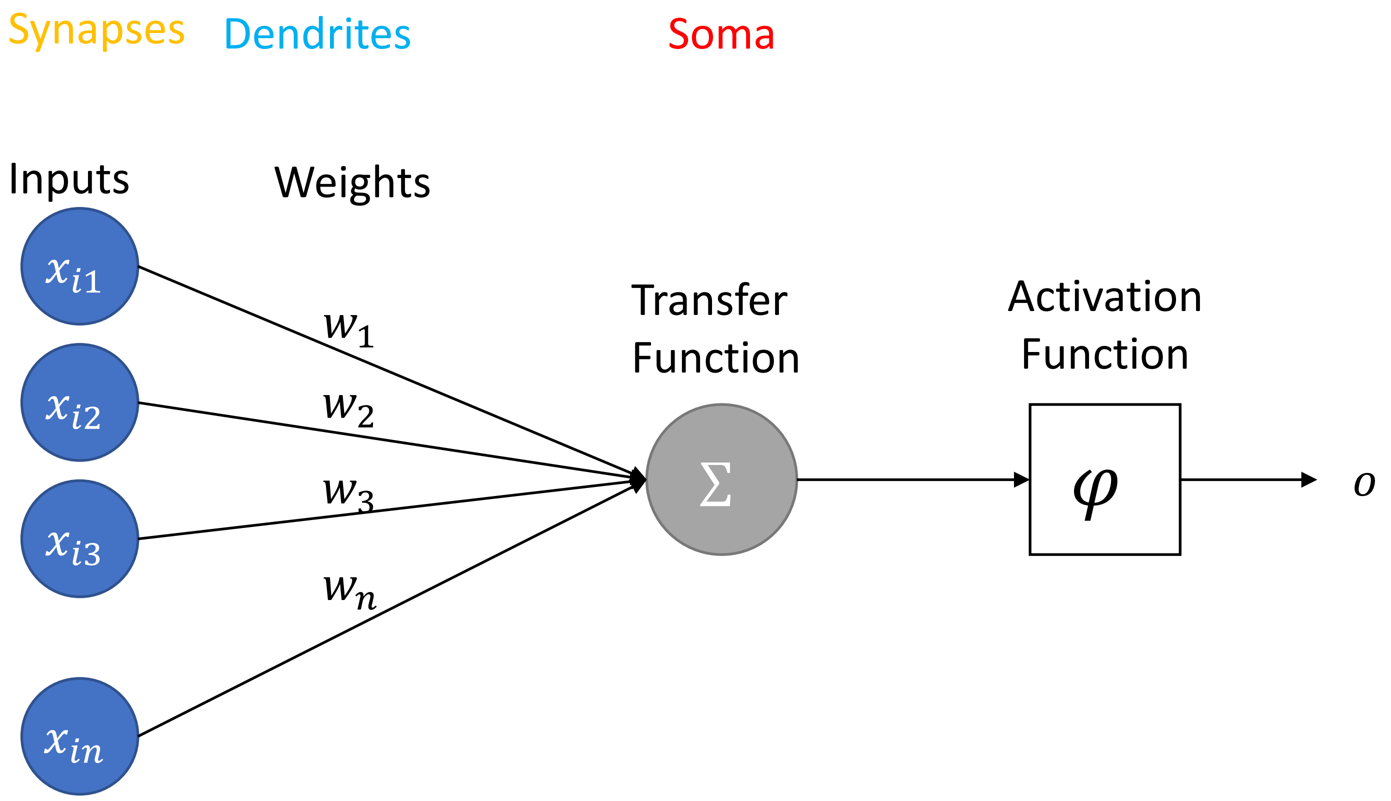


The postnatal development of the human cerebral cortex

Source: https://developingchild.harvard.edu/

Artificial Neurons

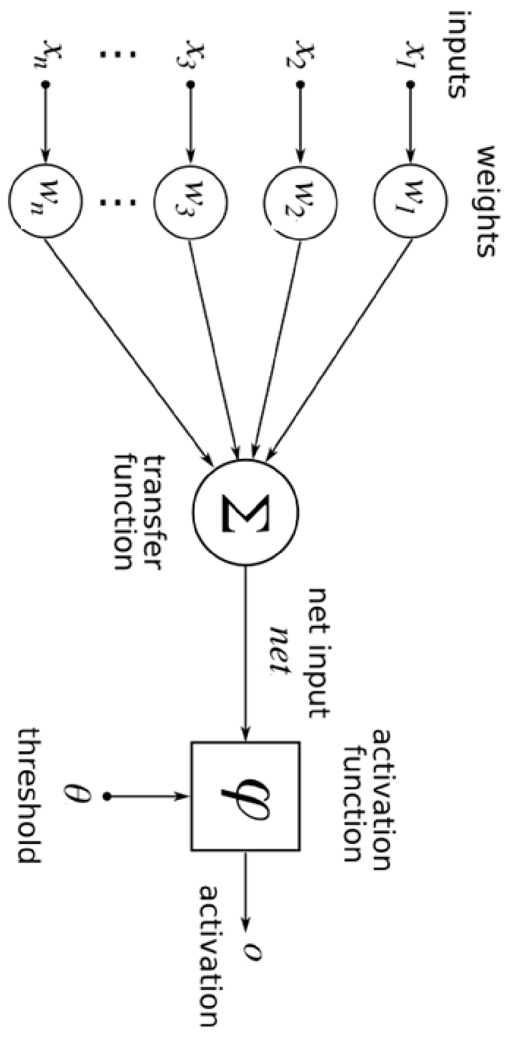




$$o = \varphi\left(\sum_{j=1}^{n} w_j x_j\right) = \varphi(\mathbf{w}^T \mathbf{x})$$

Where $\boldsymbol{w} = [w_1, w_2, \dots, w_n]^T$ and $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$

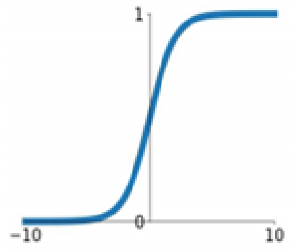
Artificial Neurons



Regular Activation Functions (φ)

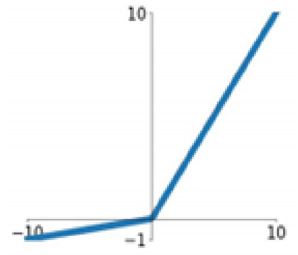
Sigmoid

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



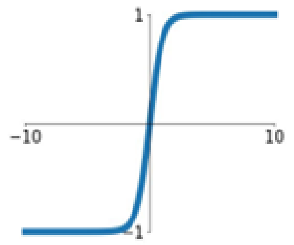
Leaky ReLU

$\max(0.1x, x)$



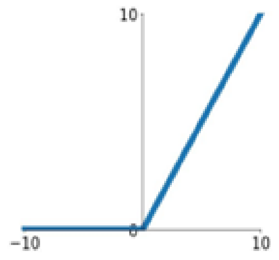
tanh

$\tanh(x)$



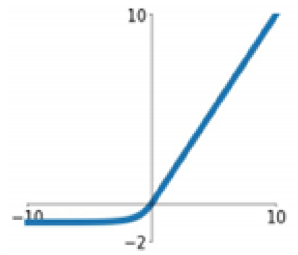
ReLU

$\max(0, x)$



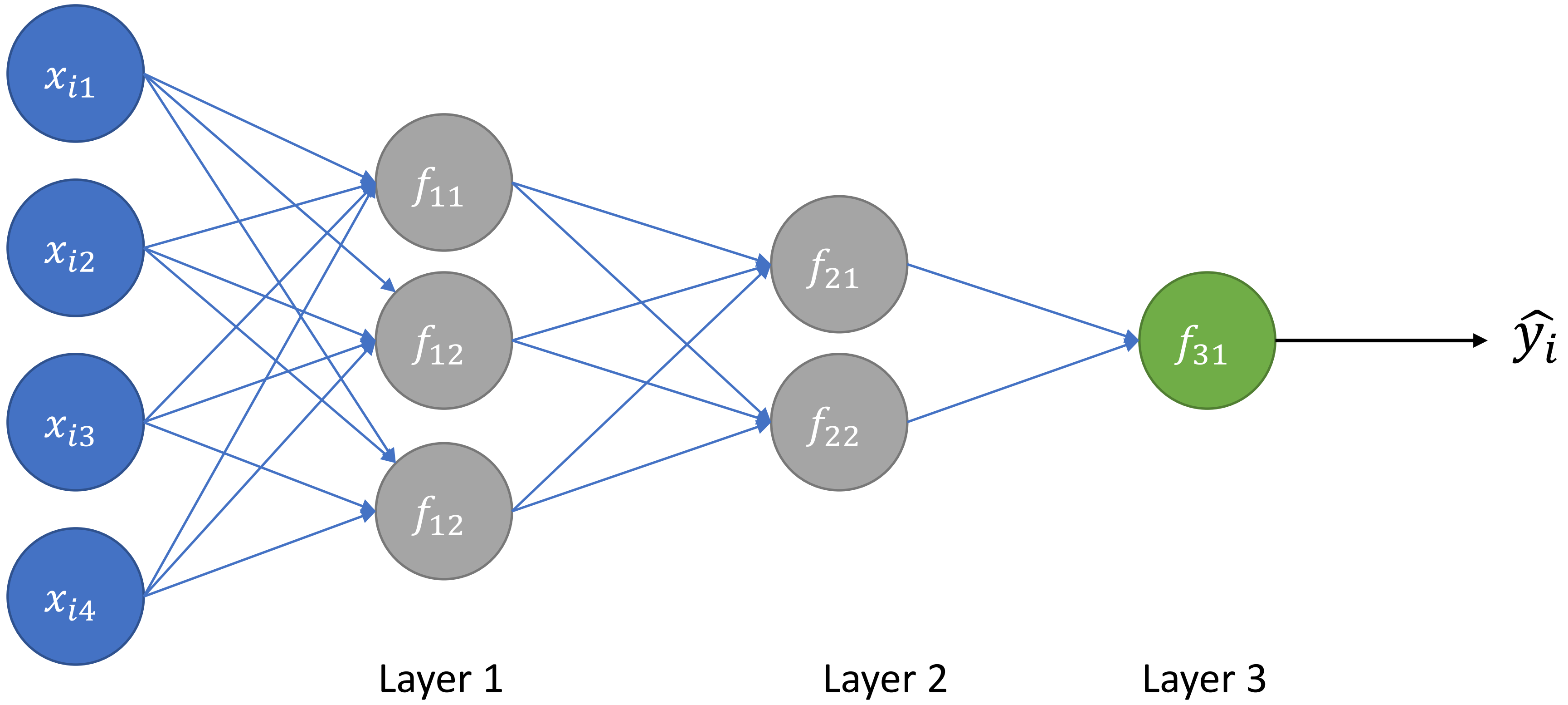
ELU

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



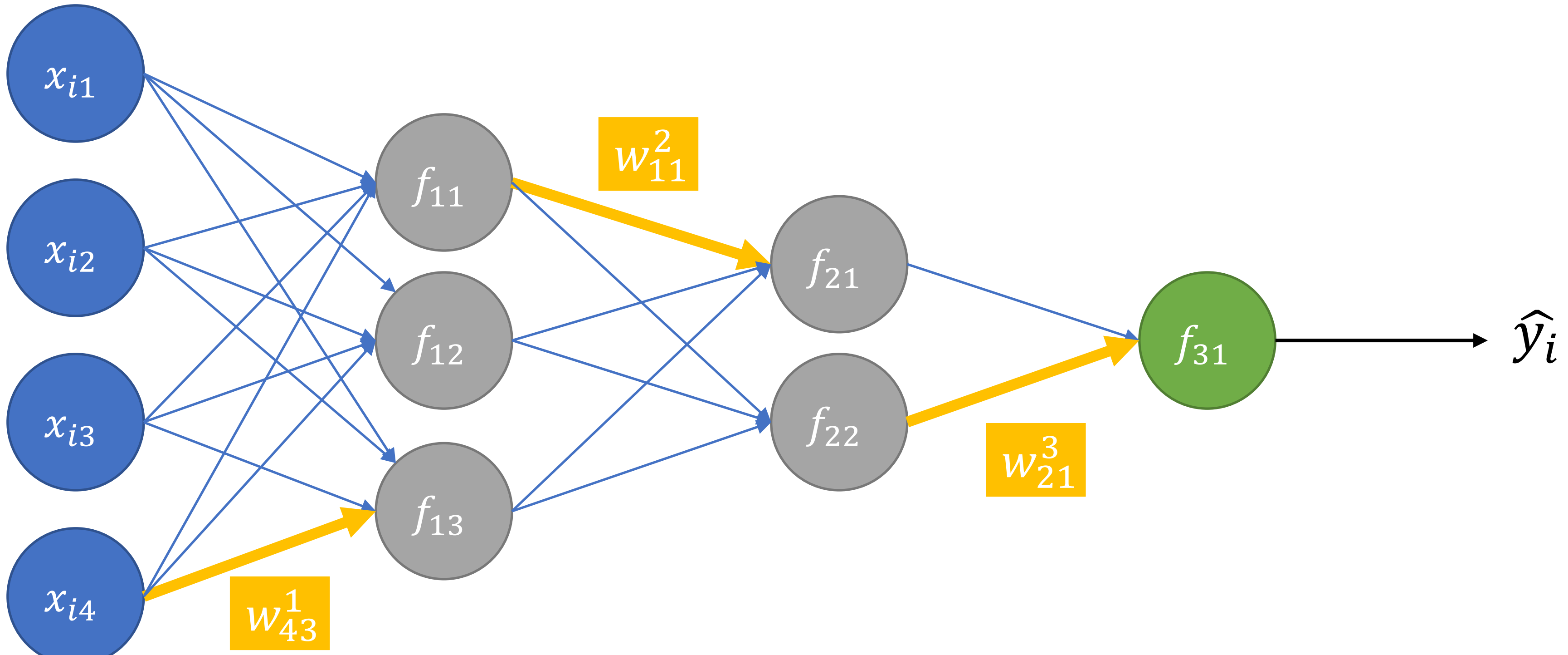
Multi-Layer Perceptron (MLP): Notations

3-Layer Fully-Connected Artificial Neural Network



Multi-Layer Perceptron (MLP): Notations

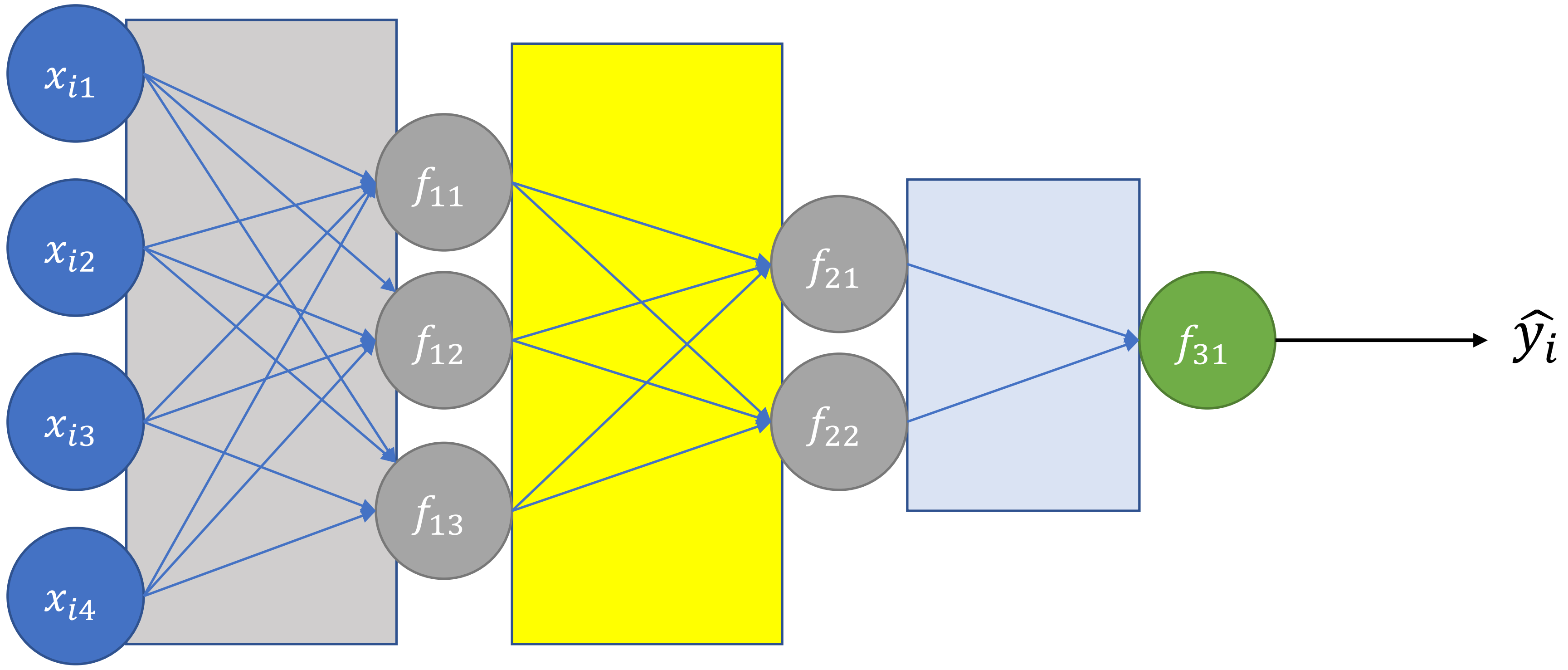
3-Layer Fully-Connected Artificial Neural Network



w_{ij}^l The weight of edge from neuron i of layer $l-1$ to neuron j of layer l

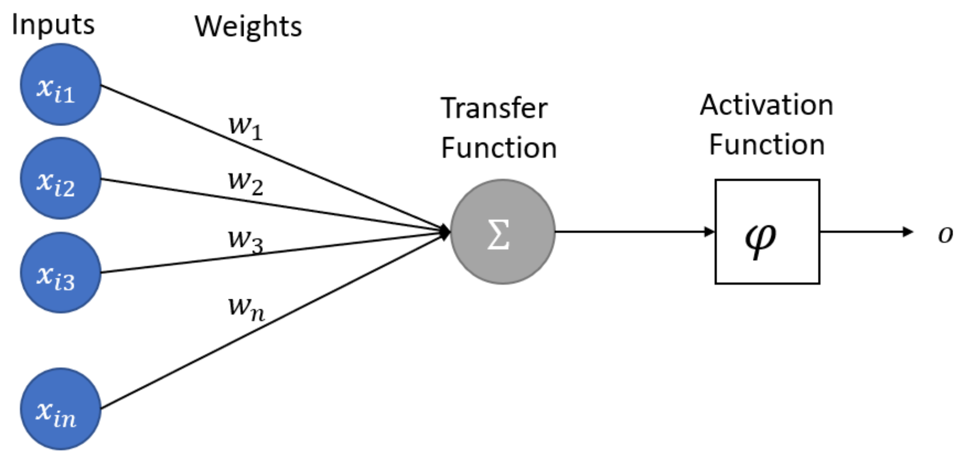
Multi-Layer Perceptron (MLP): Notations

3-Layer Fully-Connected Artificial Neural Network



$$w^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix} \quad w^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \\ w_{31}^2 & w_{32}^2 \end{bmatrix} \quad w^3 = \begin{bmatrix} w_{11}^3 \\ w_{21}^3 \end{bmatrix}$$

Training a Single Neuron



$$o = \varphi\left(\sum_{j=1}^{n} w_j x_j\right) = \varphi(\mathbf{w}^T \mathbf{x})$$

$$\mathbf{X} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_N})^{\boldsymbol{T}}$$

$$\mathbf{x_i} = (x_{i1}, x_{i2}, \dots, x_{in})^{\boldsymbol{T}}$$

$$\mathbf{y} = (y_1, y_2, \dots, y_N)^{\boldsymbol{T}}$$

Training data
N: num of data
n: dimension of each data point

Step 1: Loss function

$$l_i = (y_i - o_i)^2 = \left(y_i - \varphi(\boldsymbol{w}^T \boldsymbol{x}_i)\right)^2$$

$$L = \sum_{i=1}^{N} l_i$$

Step 2: The objective function

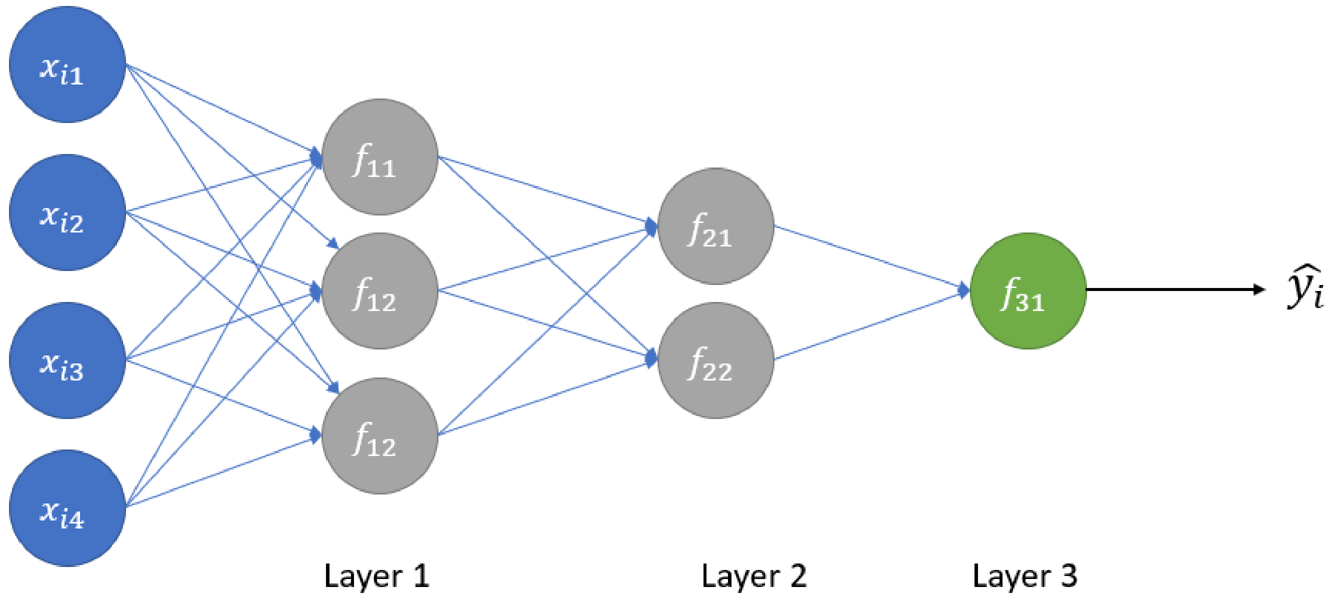
$$w^* = \arg\min_{\boldsymbol{w}} L$$

Step 3: Optimization

```
Initialize $\boldsymbol{w}$
for $iter = 1$ to $K$:
    $\nabla_{\boldsymbol{w}} L = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \dots, \frac{\partial L}{\partial w_n}\right]$
    $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \gamma \nabla_{\boldsymbol{w}} L$
```

Training an MLP (Backpropagation)

- As we know, $\nabla_W L$ plays a central role in finding the optimal parameters of model
- But calculating the gradients in multi-layer neural networks is not a trivial task!!

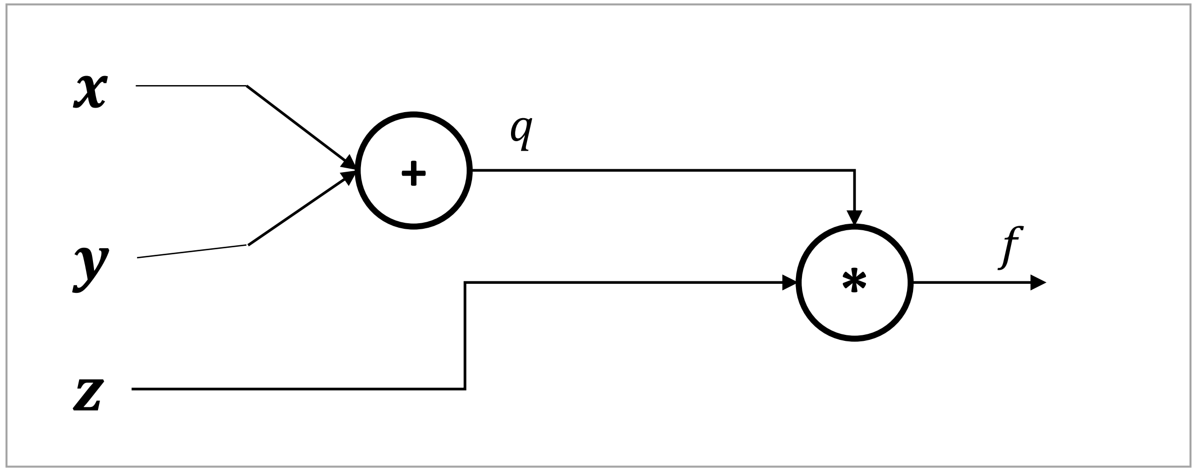


- Solution: Backpropagation (Rumelhart et al., 1986a)

Training an MLP (Backpropagation)

Computational Graphs: A simple example

$$f(x, y, z) = (x + y)z$$

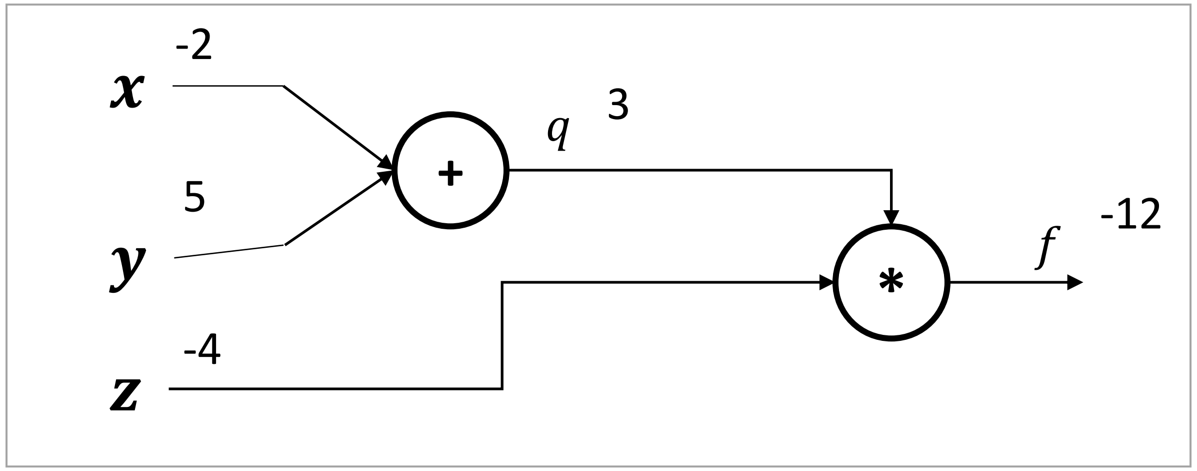


Training an MLP (Backpropagation)

Computational Graphs: A simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

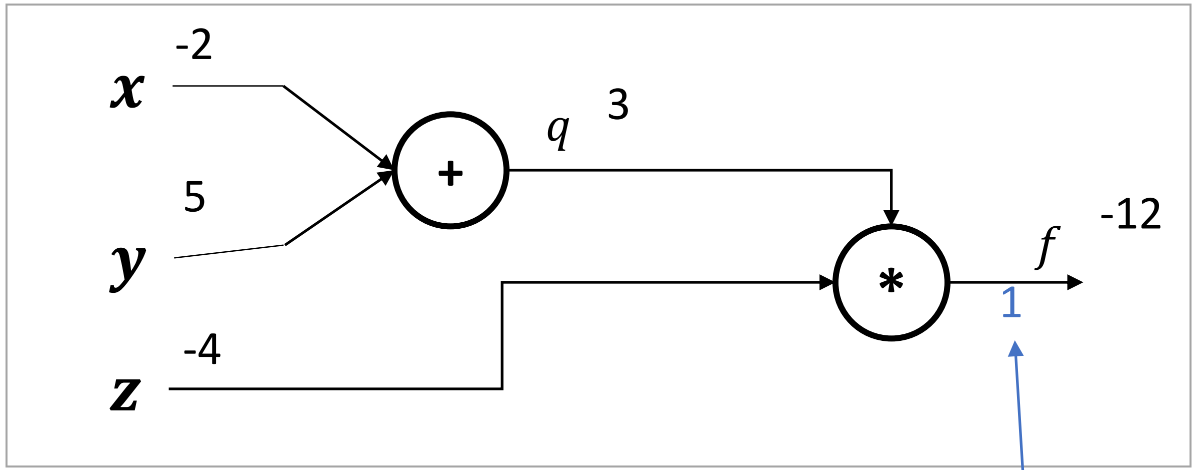
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Training an MLP (Backpropagation)

Computational Graphs: A simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

$$\boxed{\frac{\partial f}{\partial f}}$$

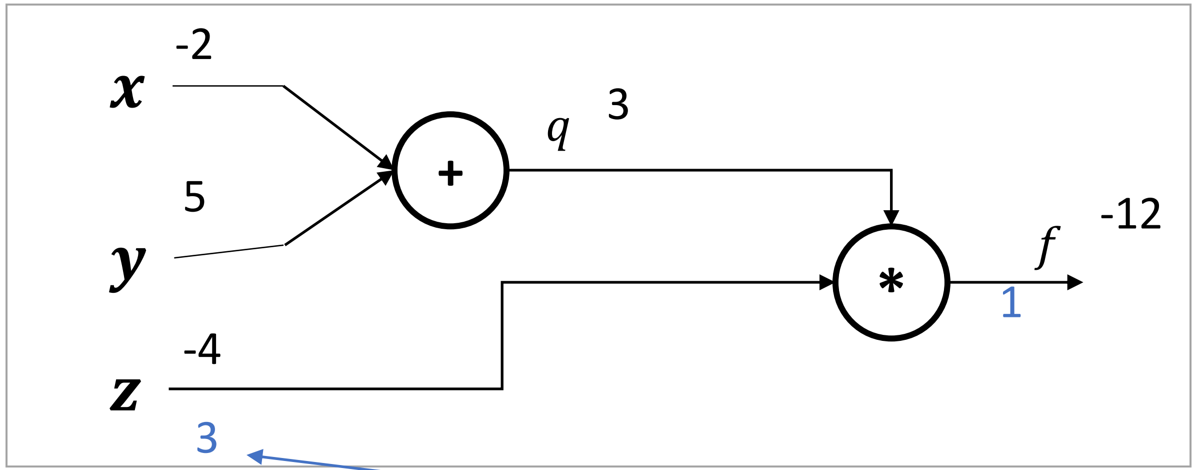
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Training an MLP (Backpropagation)

Computational Graphs: A simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$\boxed{q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

$$\boxed{\frac{\partial f}{\partial z}}$$

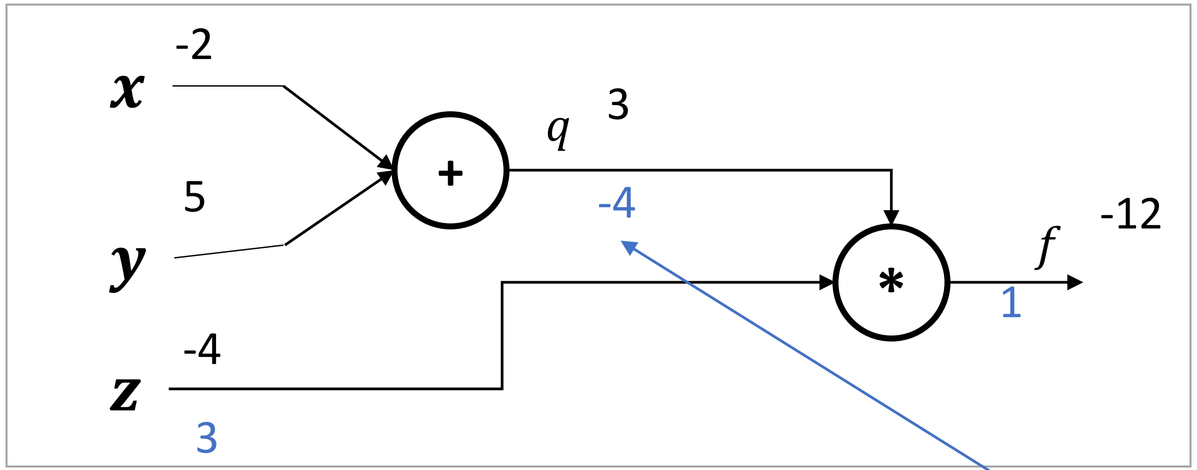
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Training an MLP (Backpropagation)

Computational Graphs: A simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial z}}$$

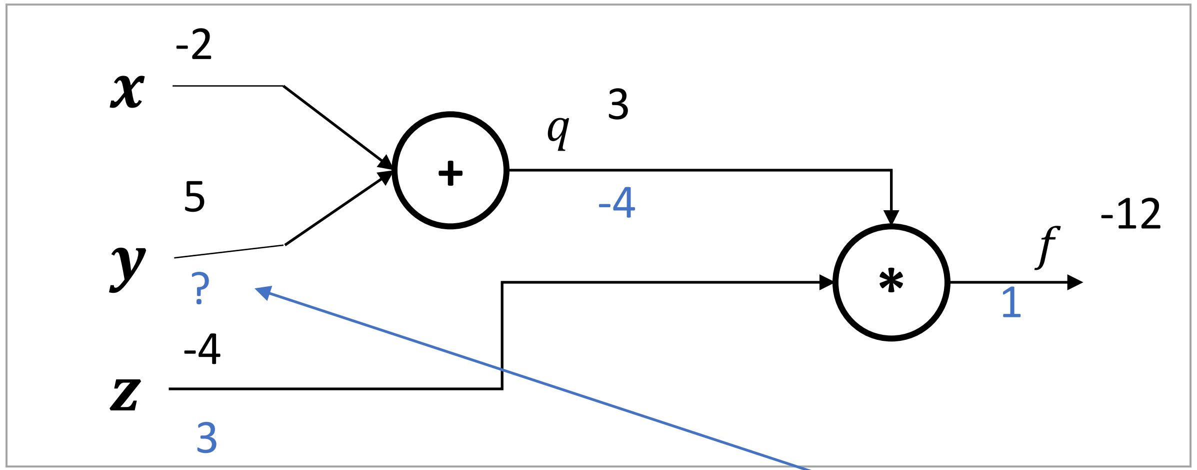
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Training an MLP (Backpropagation)

Computational Graphs: A simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial y}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain Rule:

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}}$$

Upstream gradient

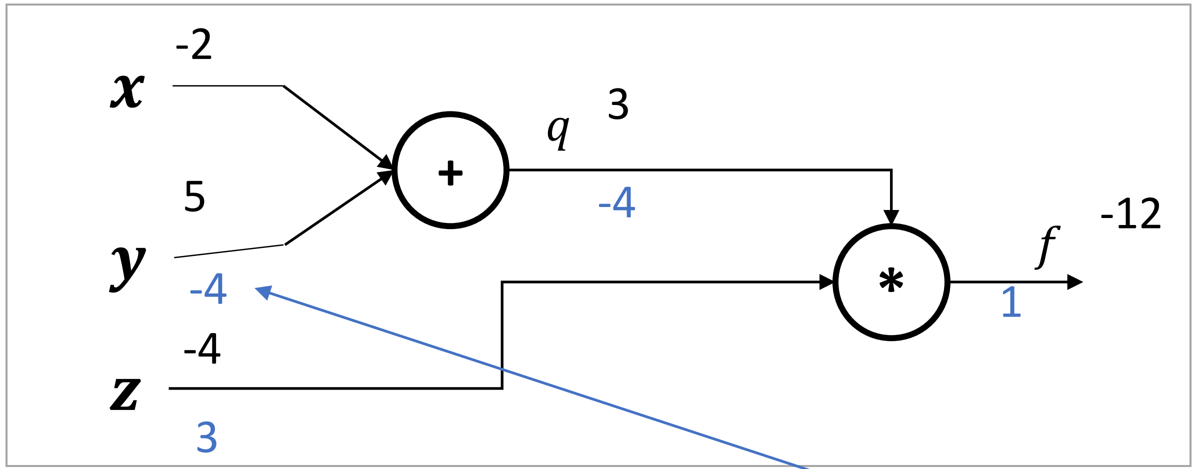
Local gradient

Training an MLP (Backpropagation)

Computational Graphs: A simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

$\boxed{\frac{\partial f}{\partial y}}$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain Rule:

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}}$$

Upstream gradient

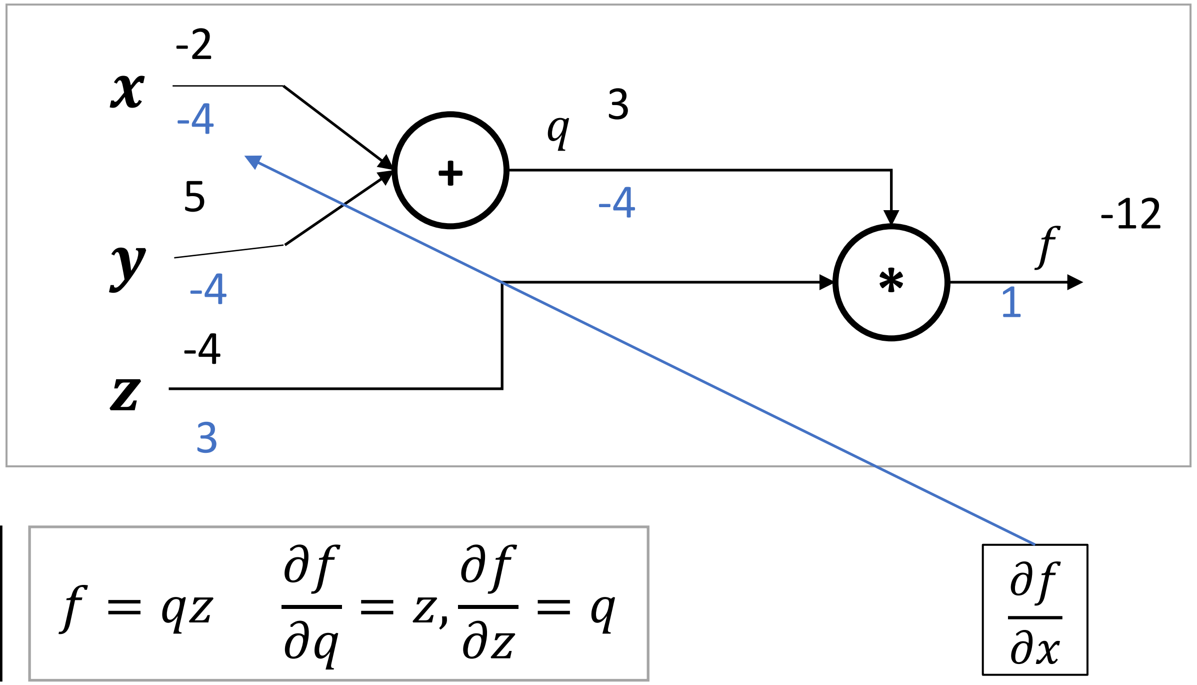
Local gradient

Training an MLP (Backpropagation)

Computational Graphs: A simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial x}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

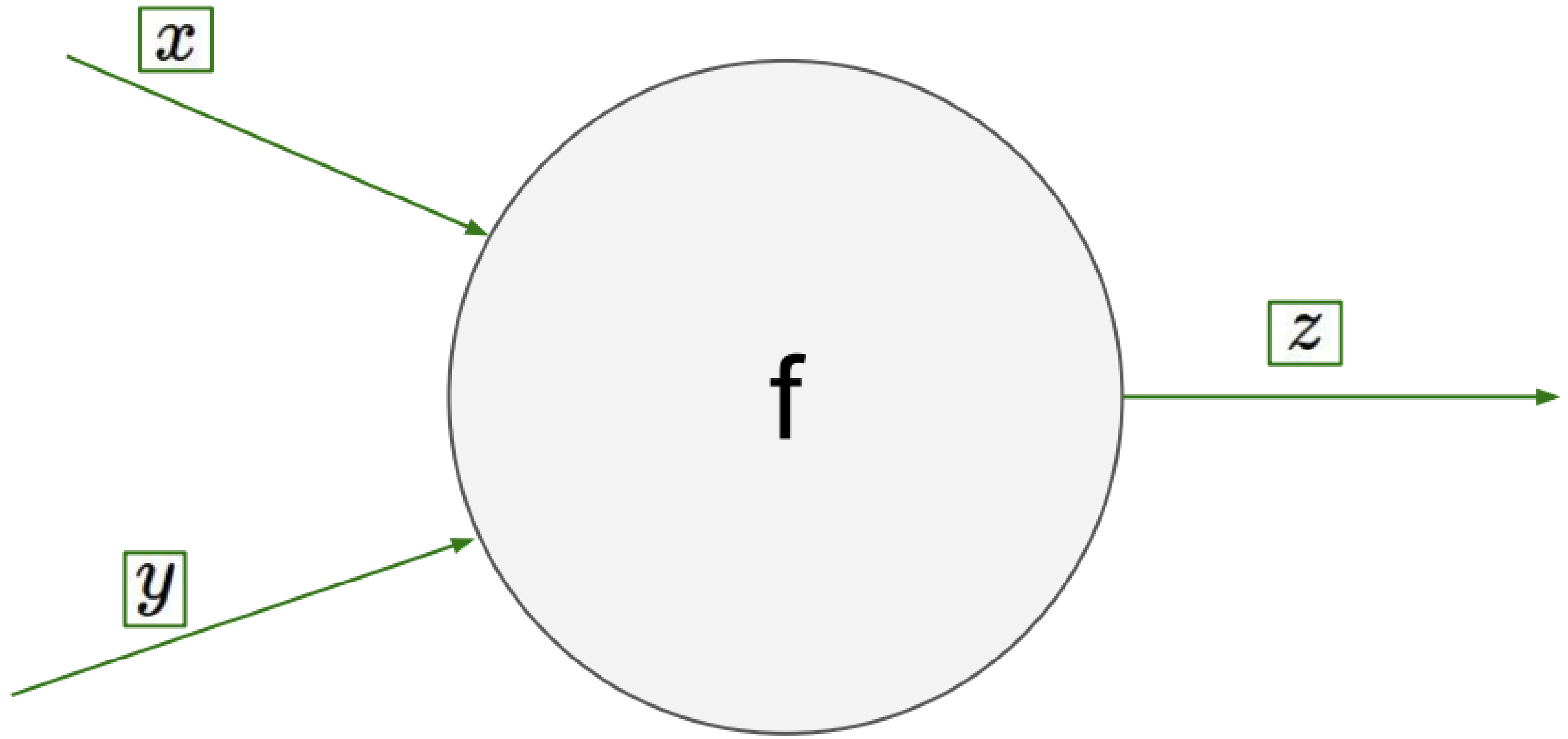
Chain Rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

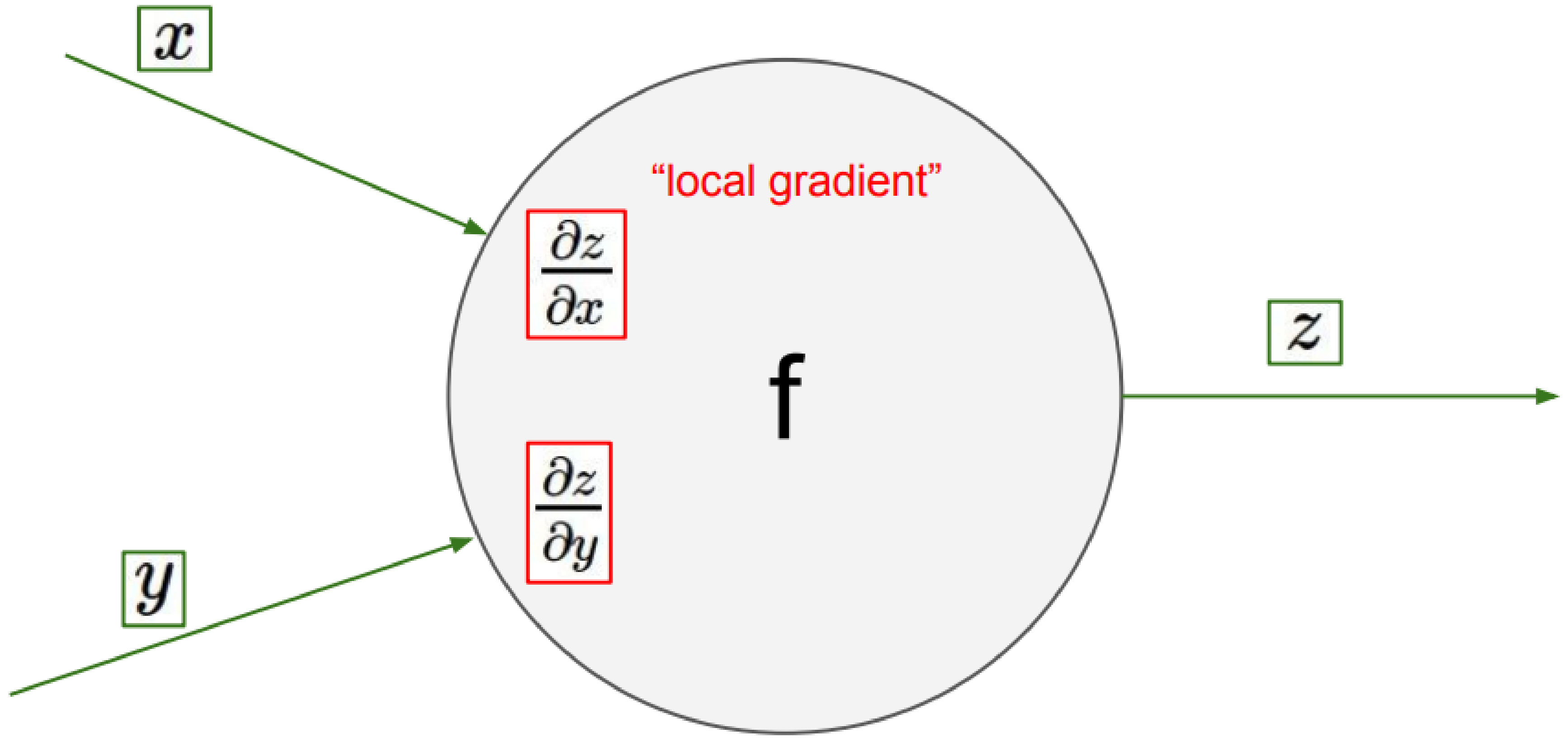
Upstream gradient

Local gradient

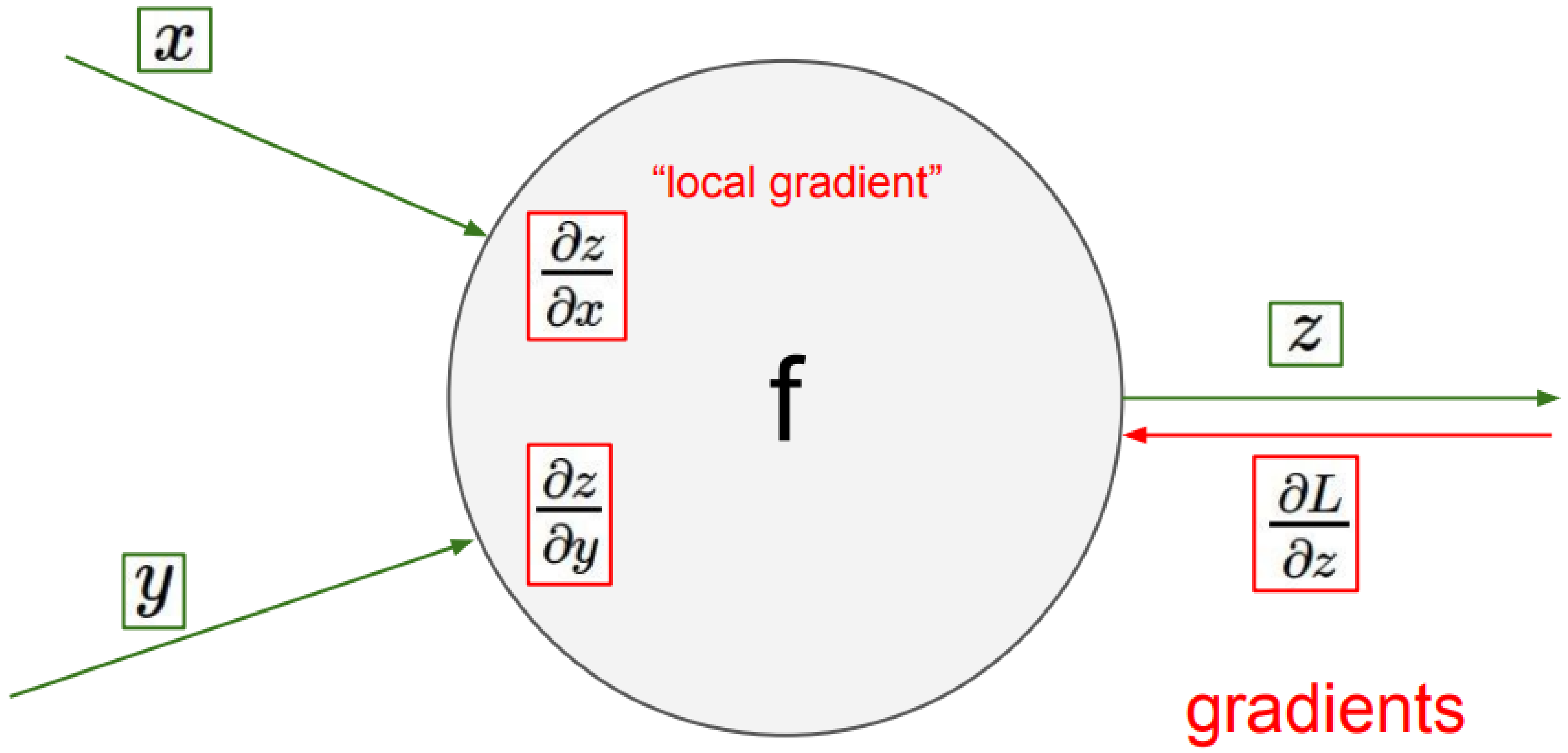
Training an MLP (Backpropagation)



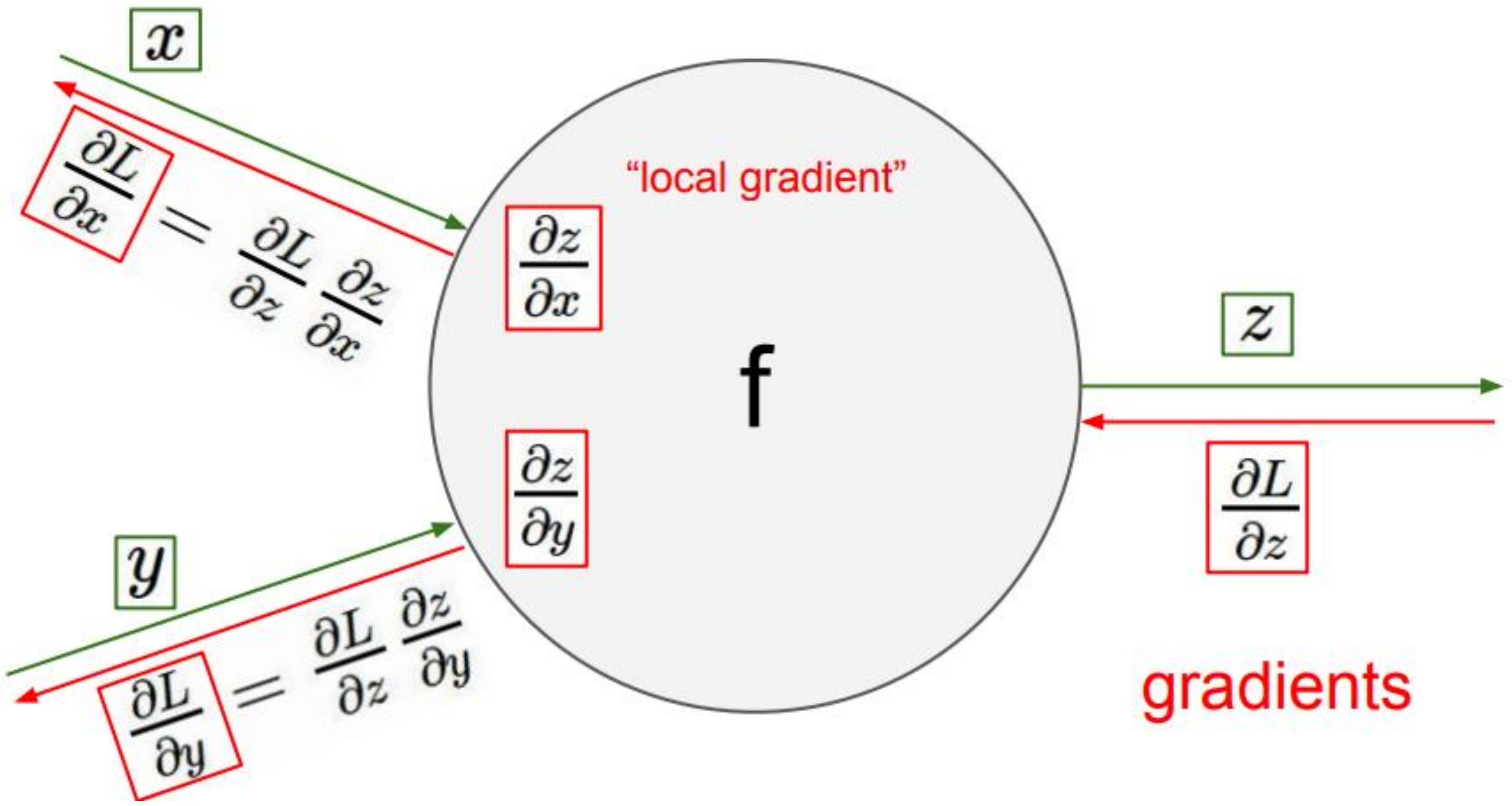
Training an MLP (Backpropagation)


x
y
z
f
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$

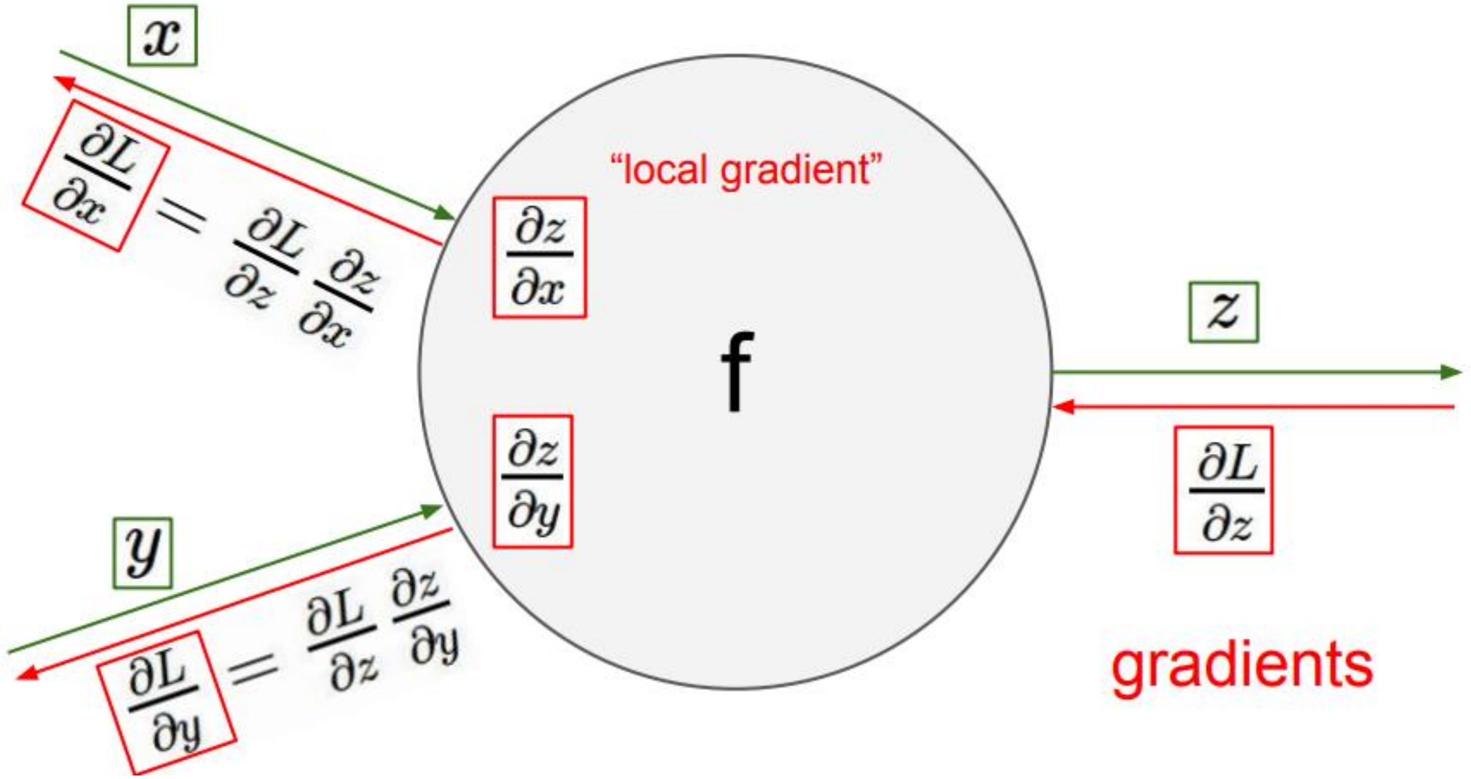
Training an MLP (Backpropagation)


x
y
z
f
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
gradients

Training an MLP (Backpropagation)

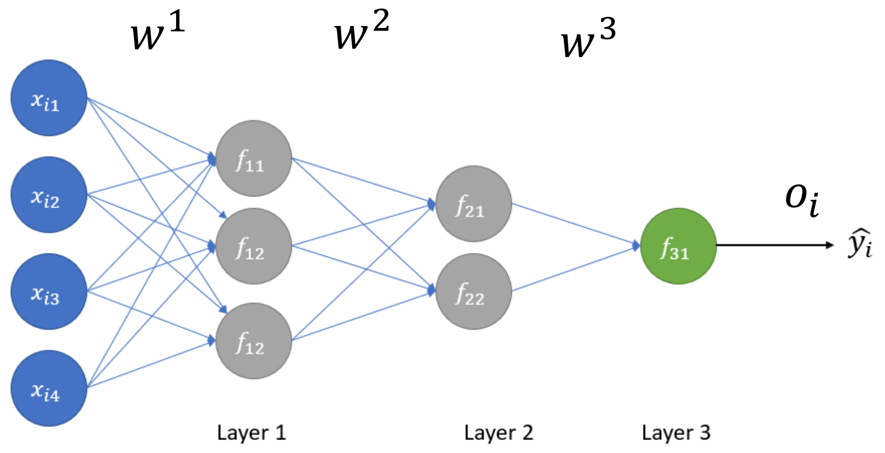

x
y
z
f
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Training an MLP (Backpropagation)



x
$\boxed{\frac{\partial L}{\partial x}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\boxed{\frac{\partial z}{\partial x}}$
f
$\boxed{\frac{\partial z}{\partial y}}$
z
$\boxed{\frac{\partial L}{\partial z}}$
y
$\boxed{\frac{\partial L}{\partial y}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Training an MLP (Backpropagation)



$$\mathbf{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N)^T$$

$$\mathbf{x_i} = (x_{i1}, x_{i2}, \dots, x_{in})^T$$

$$\mathbf{y} = (y_1, y_2, \dots, y_N)^T$$

Training data
N: num of data
n: dimension of each data point

Step 1: Loss function

$$l_i = (y_i - o_i)^2 = \left(y_i - \varphi(\boldsymbol{w}^T \boldsymbol{x}_i)\right)^2$$

$$L = \sum_{i=1}^{N} l_i$$

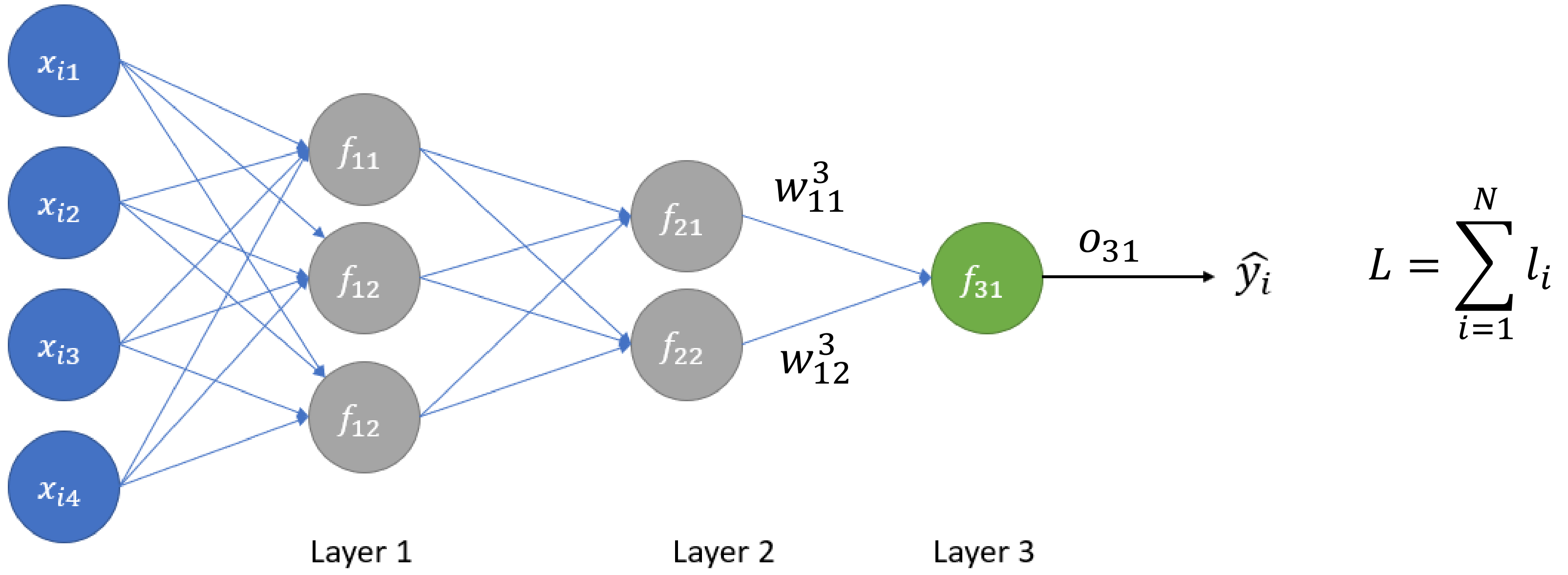
Step 2: The objective function

$$\arg \min_{w^1, w^2, w^3} L$$

Step 3: Optimization

```
Initialize $w_{ij}^k$
for $iter = 1$ to $K$:
    for each i, $j, k$:
        $(w_{ij}^k)_{new} = (w_{ij}^k)_{old} - \gamma \frac{\partial L}{\partial w_{ij}^k}$
```

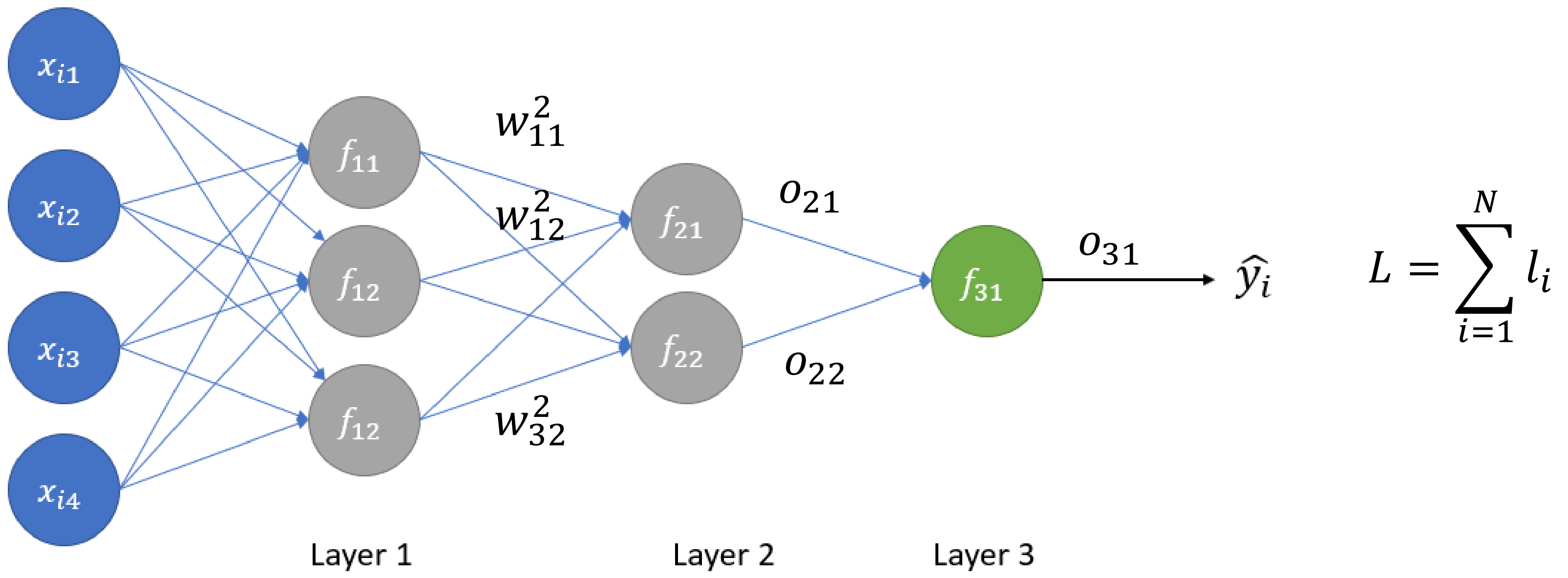
Training an MLP (Backpropagation)



$$\frac{\partial L}{\partial w_{11}^{3}} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{11}^{3}}$$

$$\frac{\partial L}{\partial w_{12}^{3}} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{12}^{3}}$$

Training an MLP (Backpropagation)



$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{11}^3}$$

$$\frac{\partial L}{\partial w_{12}^3} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{12}^3}$$

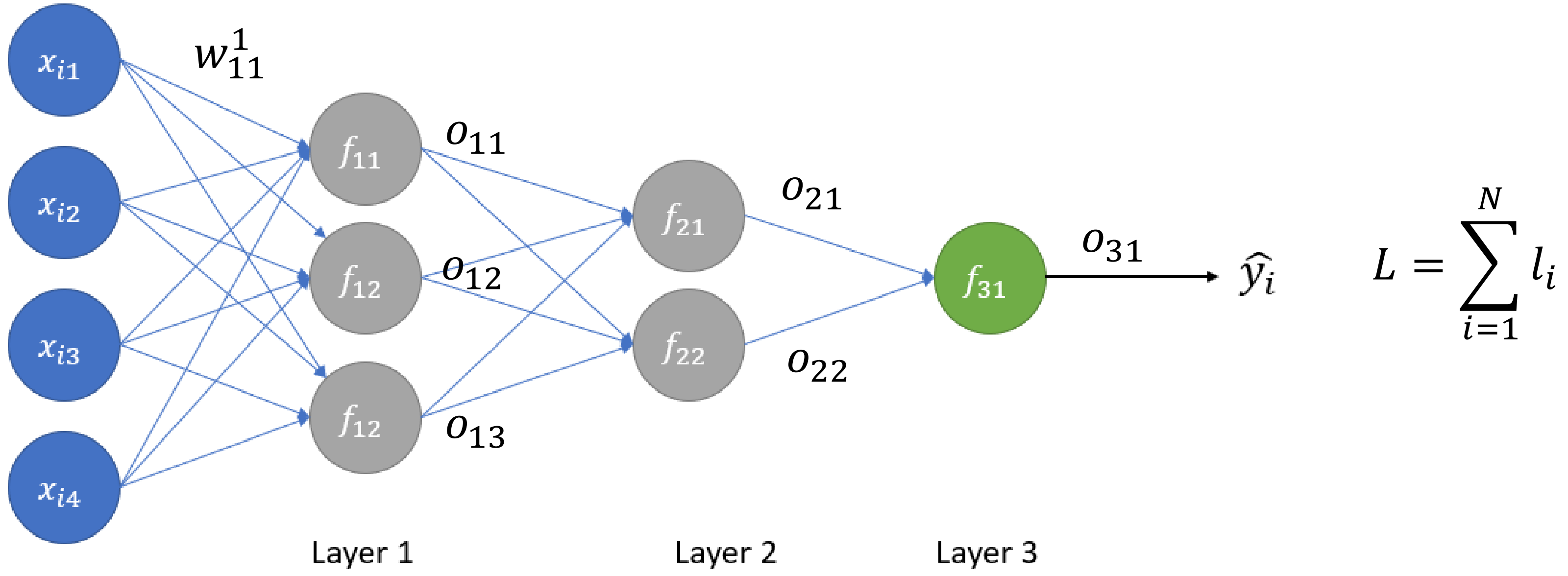
$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{11}^2}$$

$$\frac{\partial L}{\partial w_{12}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{12}^2}$$

$$\vdots$$

$$\frac{\partial L}{\partial w_{32}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial w_{32}^2}$$

Training an MLP (Backpropagation)



$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{11}^3}$$

$$\frac{\partial L}{\partial w_{12}^3} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{12}^3}$$

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{11}^2}$$

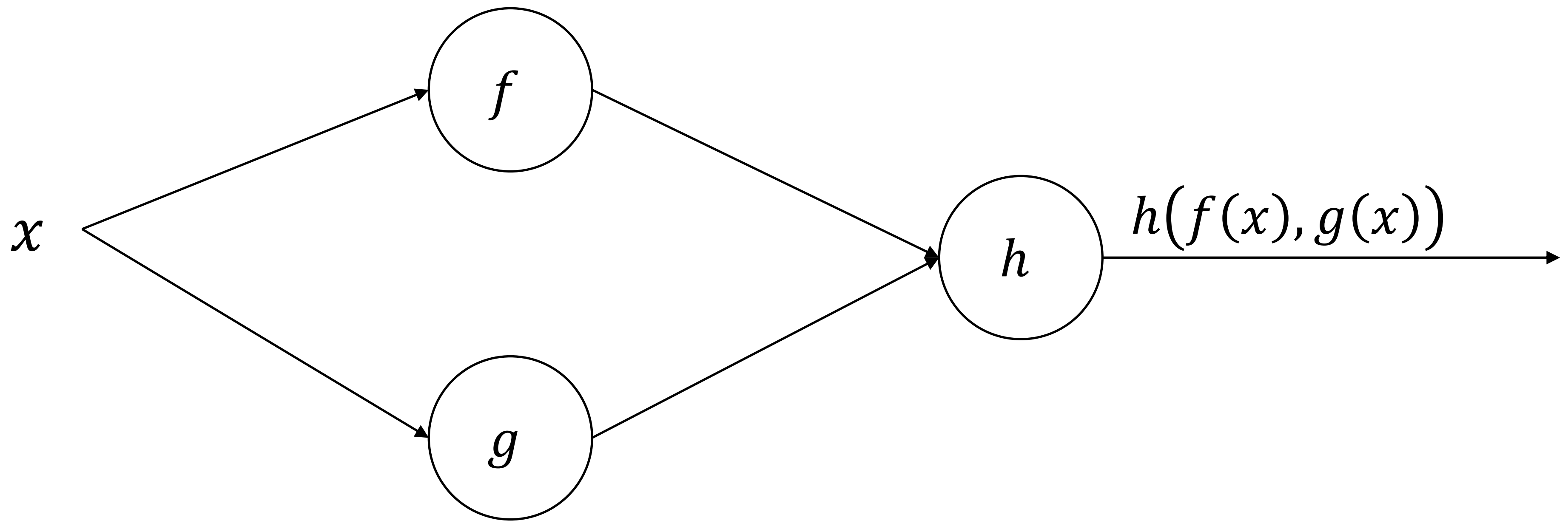
$$\frac{\partial L}{\partial w_{12}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{12}^2}$$

$$\vdots$$

$$\frac{\partial L}{\partial w_{32}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial w_{32}^2}$$

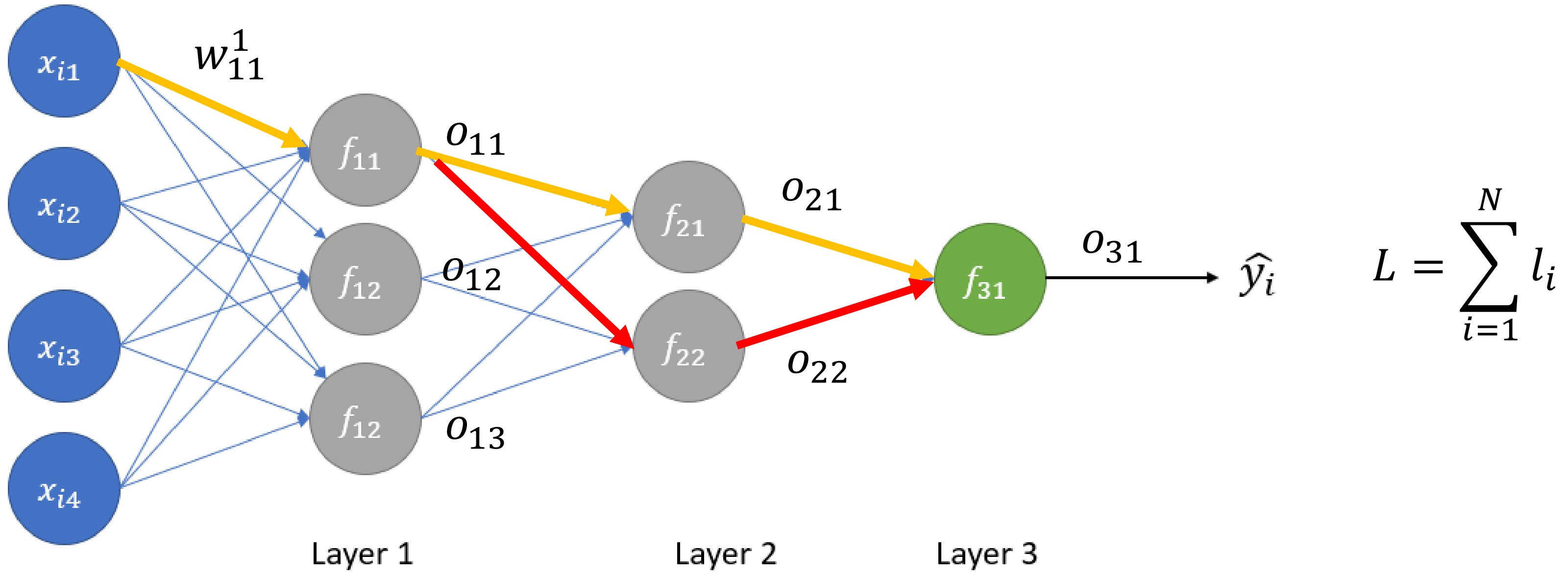
$$\frac{\partial L}{\partial w_{11}^1} = ?$$

Training an MLP (Backpropagation)



$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial f} \times \frac{\partial f}{\partial x} + \frac{\partial h}{\partial g} \times \frac{\partial g}{\partial x}$$

Training an MLP (Backpropagation)



$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{11}^3}$$

$$\frac{\partial L}{\partial w_{12}^3} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{12}^3}$$

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{11}^2}$$

$$\frac{\partial L}{\partial w_{12}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{12}^2}$$

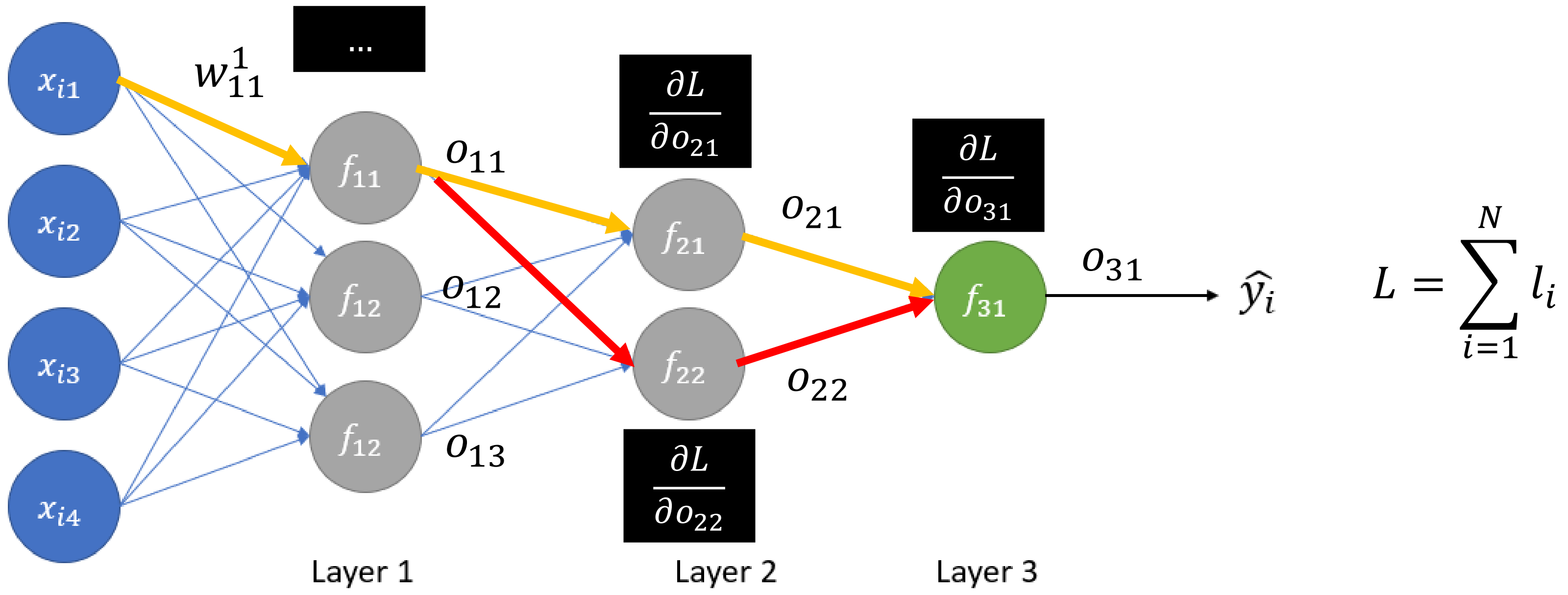
$$\vdots$$

$$\frac{\partial L}{\partial w_{32}^2} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial w_{32}^2}$$

$$\frac{\partial L}{\partial w_{11}^1} = \left(\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{11}^1}\right) + \left(\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{11}^1}\right)$$

$$\vdots$$

Training an MLP (Backpropagation)



$$\frac{\partial L}{\partial w_{11}^{3}} = \boxed{\frac{\partial L}{\partial o_{31}}} \times \frac{\partial o_{31}}{\partial w_{11}^{3}}$$

$$\frac{\partial L}{\partial w_{12}^{3}} = \boxed{\frac{\partial L}{\partial o_{31}}} \times \frac{\partial o_{31}}{\partial w_{12}^{3}}$$

$$\frac{\partial L}{\partial w_{11}^{2}} = \boxed{\frac{\partial L}{\partial o_{31}}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{11}^{2}}$$

$$\frac{\partial L}{\partial w_{12}^{2}} = \boxed{\frac{\partial L}{\partial o_{31}}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{12}^{2}}$$

$$\vdots$$

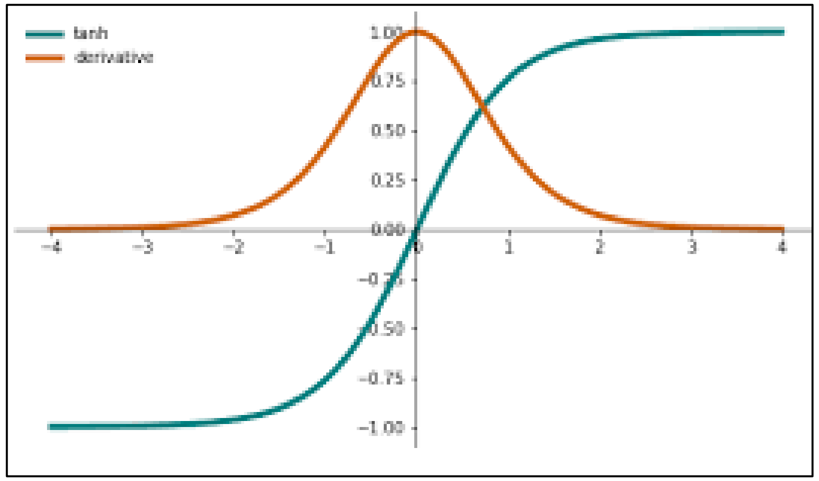
$$\frac{\partial L}{\partial w_{32}^{2}} = \boxed{\frac{\partial L}{\partial o_{31}}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial w_{32}^{2}}$$

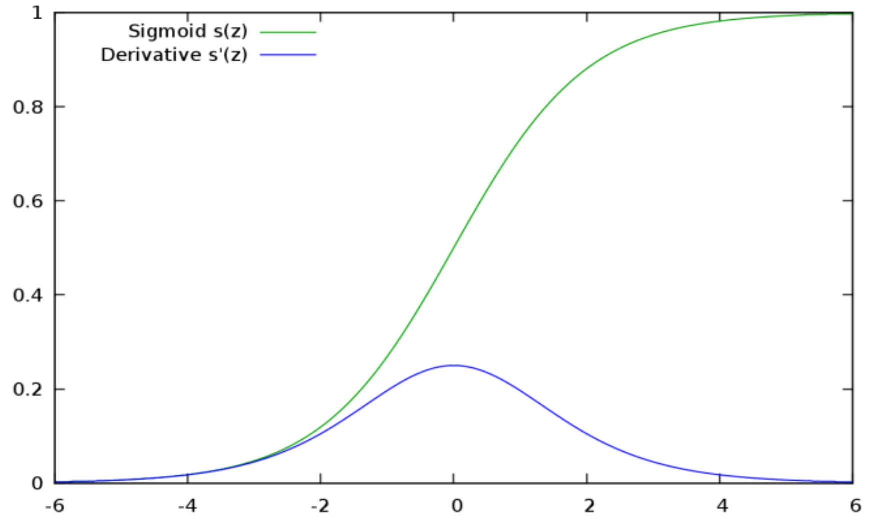
$$\frac{\partial L}{\partial w_{11}^{1}} = \left(\boxed{\frac{\partial L}{\partial o_{31}}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{11}^{1}}\right) + \left(\boxed{\frac{\partial L}{\partial o_{31}}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{11}^{1}}\right)$$

$$\vdots$$

Memoization: Avoid redundant computations

Activation Functions Revisited





- Sigmoid and Tanh are the two most used traditional activation functions in neural networks
- $\frac{df}{dx} \in [0,1]$ for both functions 😐
- Consider computing $\frac{\partial L}{\partial w_{11}^1}$ and $\frac{\partial L}{\partial w_{11}^3}$:

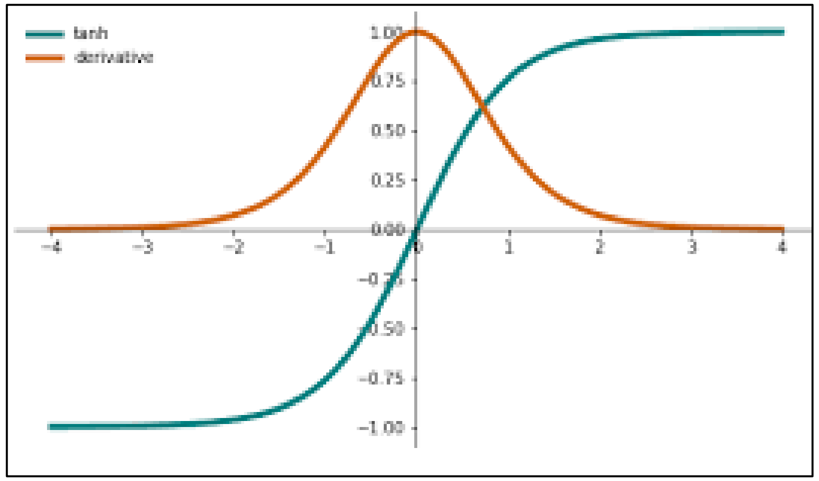
$$\frac{\partial L}{\partial w_{11}^1} = \left(\underbrace{\frac{\partial L}{\partial o_{31}}}_{0.4} \times \underbrace{\frac{\partial o_{31}}{\partial o_{21}}}_{0.2} \times \underbrace{\frac{\partial o_{21}}{\partial o_{11}}}_{0.08} \times \underbrace{\frac{\partial o_{11}}{\partial w_{11}^1}}_{0.9}\right) + \left(\underbrace{\frac{\partial L}{\partial o_{31}}}_{0.4} \times \underbrace{\frac{\partial o_{31}}{\partial o_{22}}}_{0.6} \times \underbrace{\frac{\partial o_{22}}{\partial o_{11}}}_{0.3} \times \underbrace{\frac{\partial o_{11}}{\partial w_{11}^1}}_{0.9}\right) \quad \boxed{0.07}$$

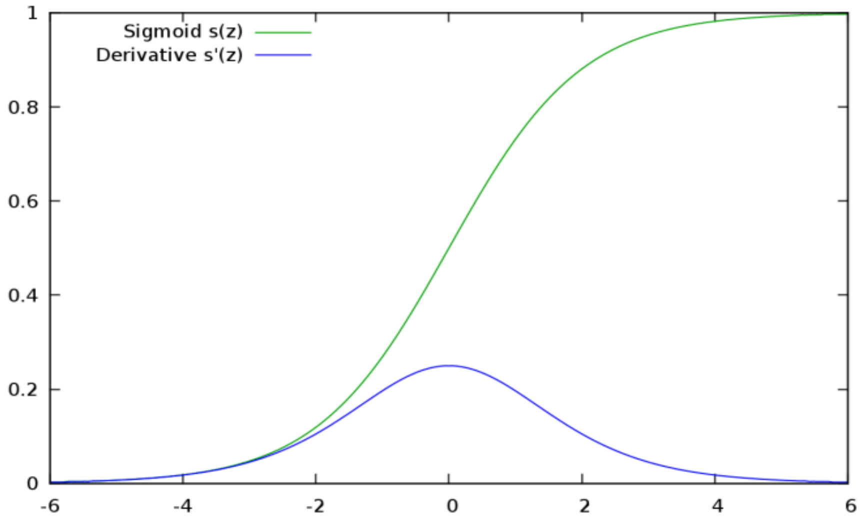
$$\frac{\partial L}{\partial w_{11}^3} = \underbrace{\frac{\partial L}{\partial o_{31}}}_{0.4} \times \underbrace{\frac{\partial o_{31}}{\partial w_{11}^3}}_{0.5} \quad \boxed{0.2}$$

$$\frac{\partial L}{\partial w_{11}^3} \gg \frac{\partial L}{\partial w_{11}^1}$$

- As we move backward the input, the values of the partial derivatives become smaller.
- This is called Vanishing Gradients

Activation Functions Revisited





- What's the problem with the vanishing gradients?

- Consider the optimization phase in GD method:

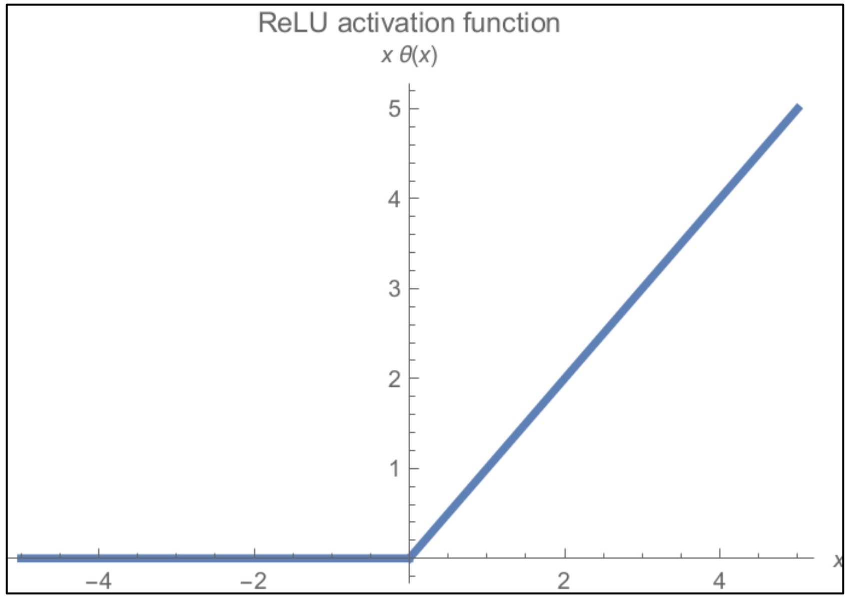
$$\left(w_{ij}^{k}\right)_{new} = \left(w_{ij}^{k}\right)_{old} - \gamma \frac{\partial L}{\partial w_{ij}^{k}}$$

- If the gradient is vanished ($\frac{\partial L}{\partial w_{ij}^{k}} \approx 0$),

$$\left(w_{ij}^{k}\right)_{new} \approx \left(w_{ij}^{k}\right)_{old}$$

- This happens in early layers of a large neural network

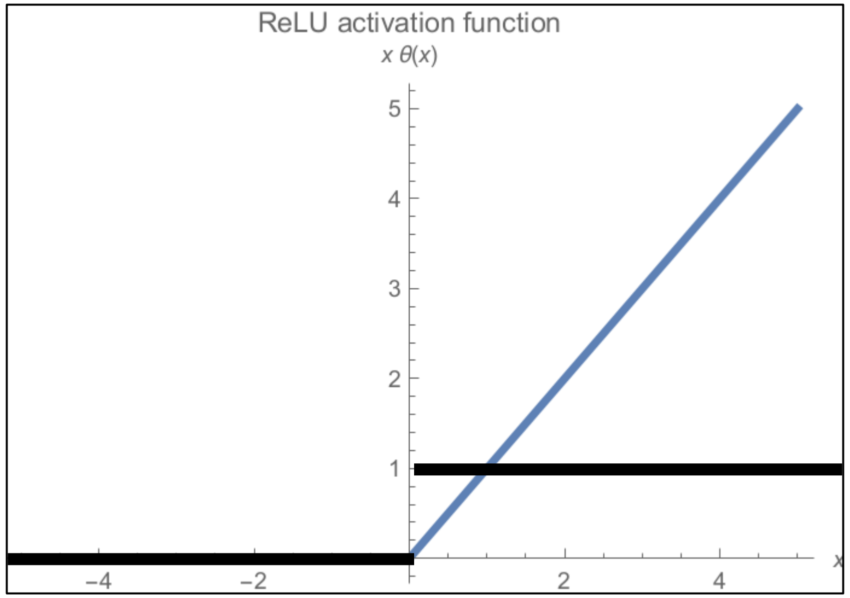
Activation Functions Revisited



$$f(z) = \max(0, z)$$

- ReLU: Rectified Linear Units

Activation Functions Revisited



$$f(x) = \max(0, x)$$

- **ReLU**: Rectified Linear Units
- $\frac{df}{dx} \in \{0,1\}$ for both functions 😐
- Consider computing $\frac{\partial L}{\partial w_{11}^{1}}$:

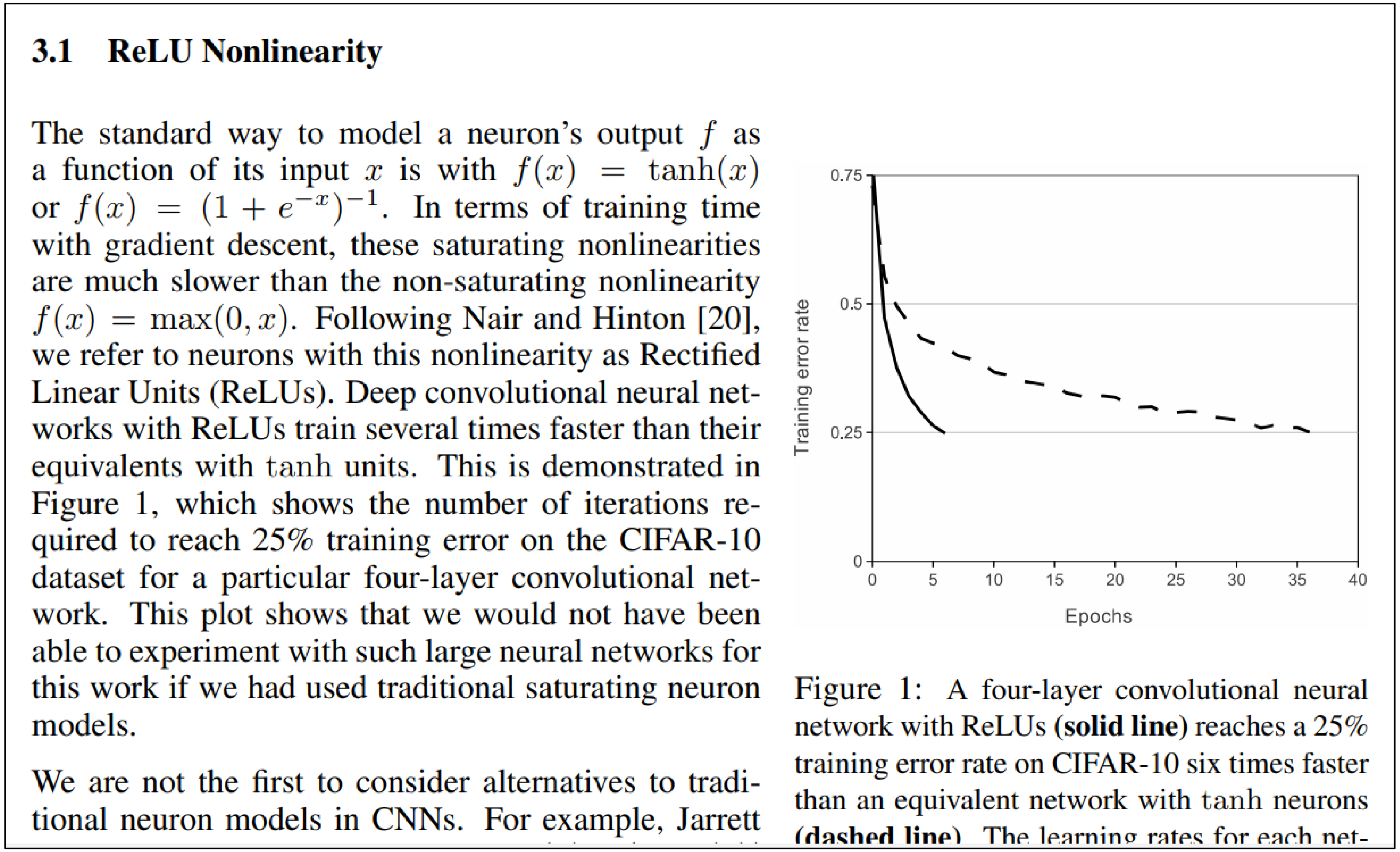
$$\frac{\partial L}{\partial w_{11}^{1}} = \left(\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{11}^{1}}\right) + \left(\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{11}^{1}}\right)$$

1 1 0 1 0 1 1 1 0

- ReLU neurons cannot learn on examples for which their activation is zero (Dead Activations).

Activation Functions Revisited

ReLU vs Tanh



3.1 ReLU Nonlinearity

The standard way to model a neuron's output f as a function of its input x is with $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = \max(0, x)$. Following Nair and Hinton [20], we refer to neurons with this nonlinearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units. This is demonstrated in Figure 1, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. This plot shows that we would not have been able to experiment with such large neural networks for this work if we had used traditional saturating neuron models.

We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett

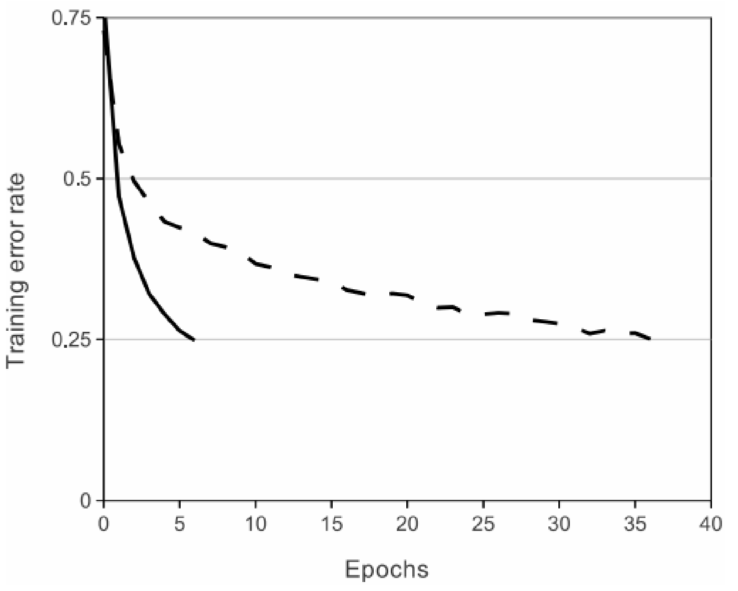
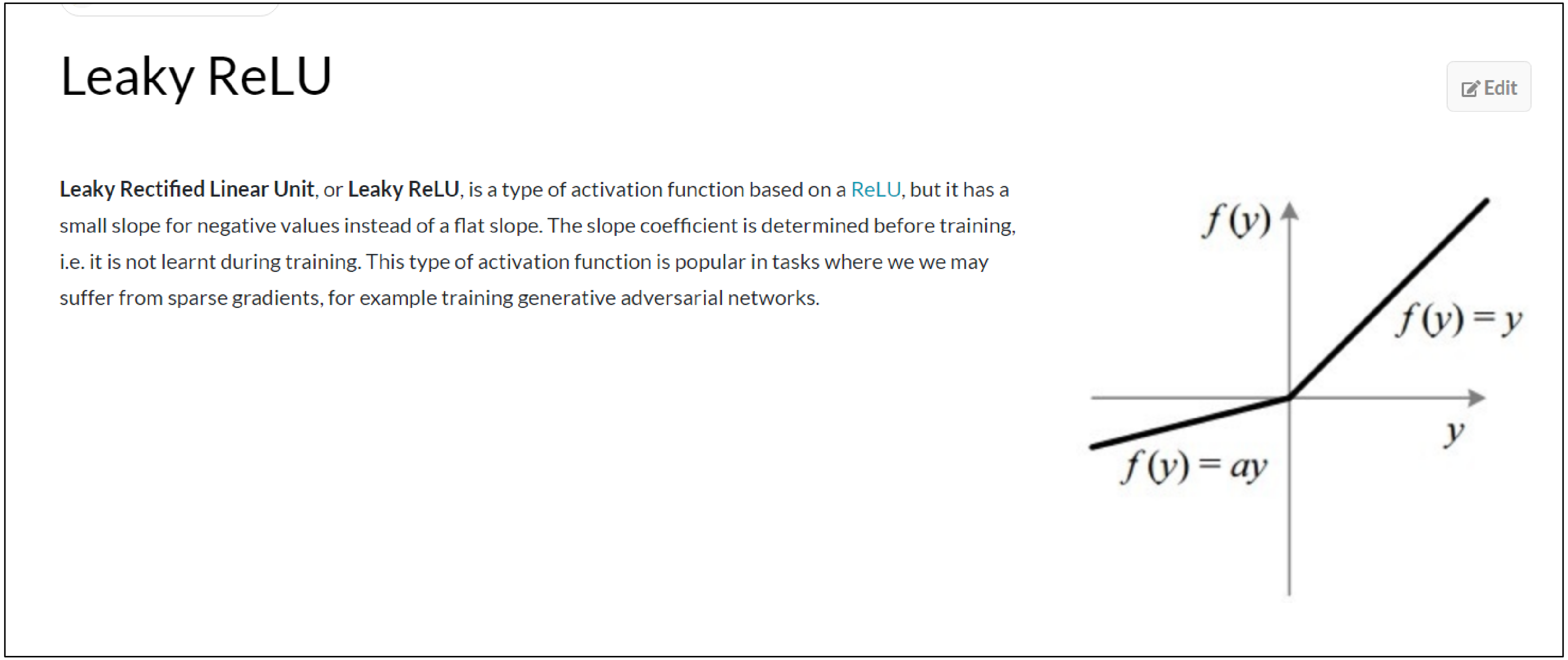


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each net-

Source: Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2017. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), pp.84-90.

Activation Functions Revisited



Leaky ReLU

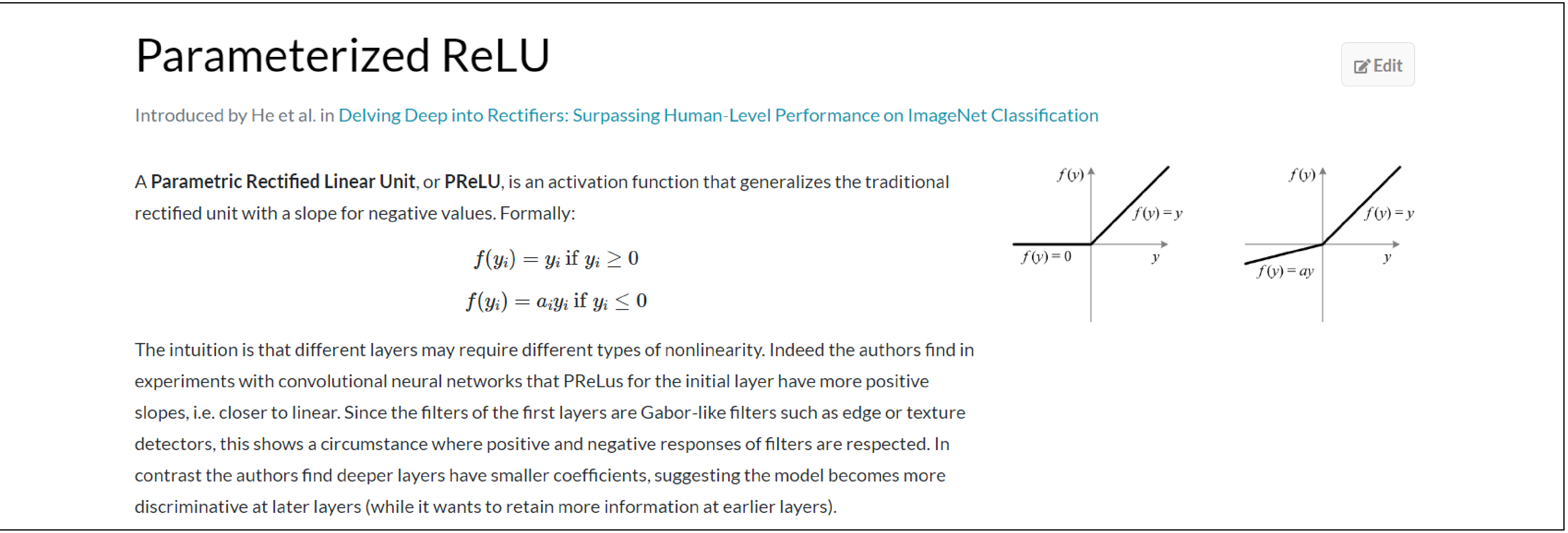
Edit

Leaky Rectified Linear Unit, or **Leaky ReLU**, is a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope. The slope coefficient is determined before training, i.e. it is not learnt during training. This type of activation function is popular in tasks where we we may suffer from sparse gradients, for example training generative adversarial networks.

Source: https://paperswithcode.com/method/leaky-relu

Paper: Maas, A.L., Hannun, A.Y. and Ng, A.Y., 2013, June. Rectifier nonlinearities improve neural network acoustic models. In Proc. icml (Vol. 30, No. 1, p. 3).

Activation Functions Revisited



Parameterized ReLU

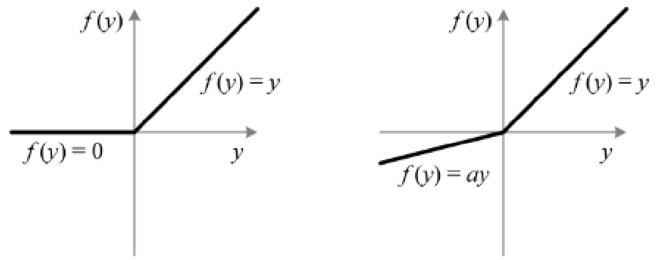
Edit

Introduced by He et al. in Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

A **Parametric Rectified Linear Unit**, or **PReLU**, is an activation function that generalizes the traditional rectified unit with a slope for negative values. Formally:

$$f(y_i) = y_i \text{ if } y_i \geq 0$$

$$f(y_i) = a_i y_i \text{ if } y_i \leq 0$$



The intuition is that different layers may require different types of nonlinearity. Indeed the authors find in experiments with convolutional neural networks that PReLus for the initial layer have more positive slopes, i.e. closer to linear. Since the filters of the first layers are Gabor-like filters such as edge or texture detectors, this shows a circumstance where positive and negative responses of filters are respected. In contrast the authors find deeper layers have smaller coefficients, suggesting the model becomes more discriminative at later layers (while it wants to retain more information at earlier layers).

Source:https://paperswithcode.com/method/prelu#:~:text=A%20Parametric%20Rectified%20Linear%20Unit,if%20y%20i%20%E2%89%A4%200

Paper: He, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

Regularization

- The aim of regularization is to avoid overfitting
- L1-Regularization

$$L = \sum_{i=1}^{N} l_i + \lambda \sum_{i,j,k} \left| w_{ij}^{k} \right|$$

- L2-Regularization

$$L = \sum_{i=1}^{N} l_i + \lambda \sum_{i,j,k} \left(w_{ij}^{k} \right)^2$$

Regularization

- **Dropout**: a very simple and elegant approach to apply regularization to neural networks

Source: Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), pp.1929-1958.

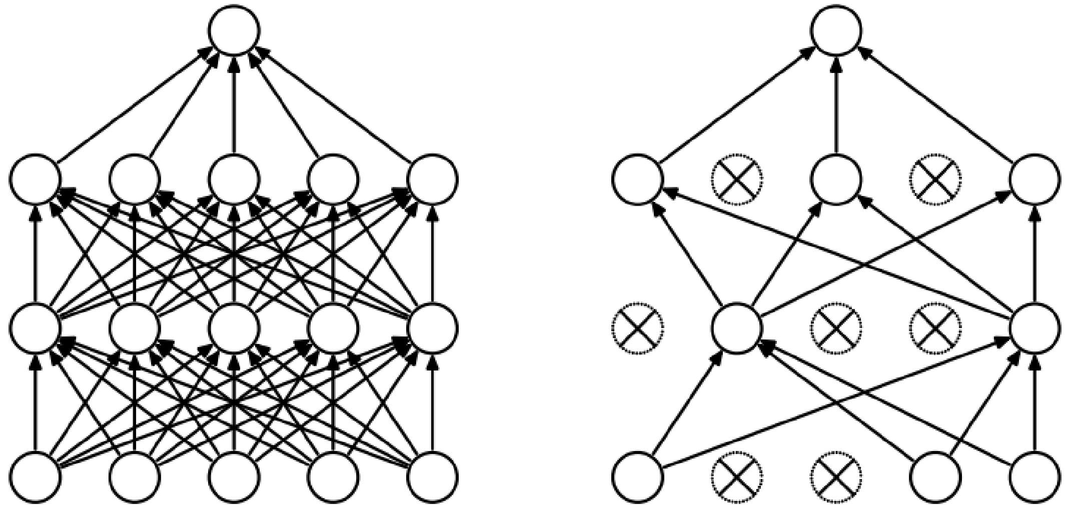


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably

Regularization

- **Dropout**: a very simple and elegant approach to apply regularization to neural networks

Source: Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), pp.1929-1958.

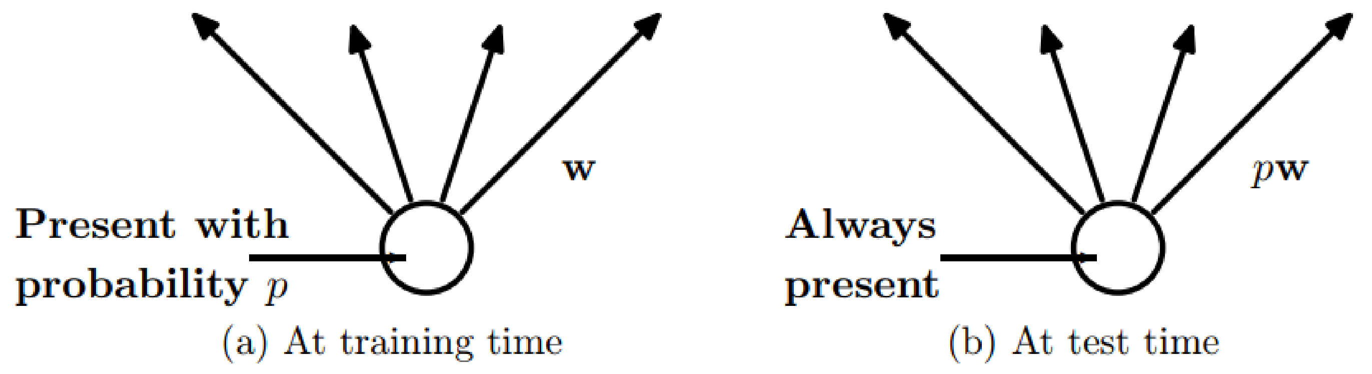
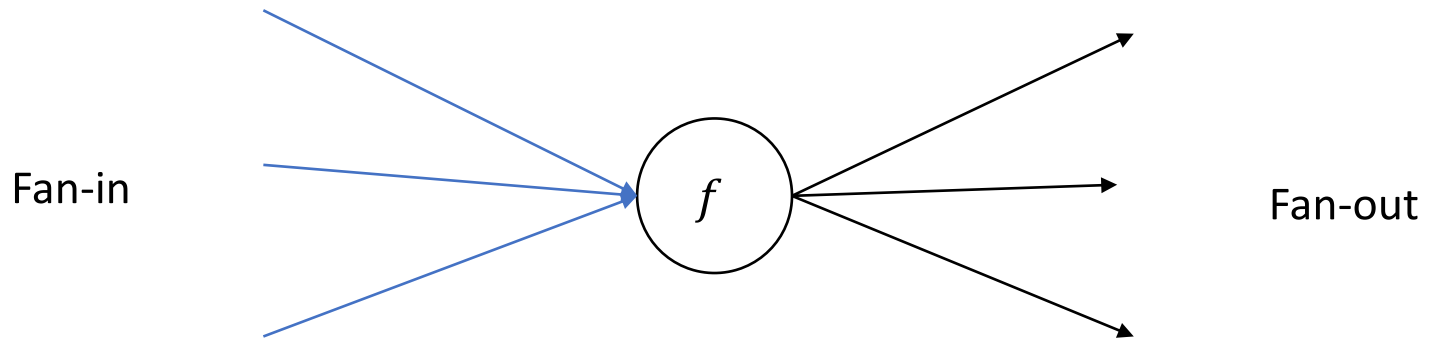


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Weight Initialization

- Designing initialization strategies is a difficult task because neural network optimization is not yet well understood.
- Our understanding of how the initial point affects generalization is especially primitive, offering little to no guidance for how to select the initial point.
- Perhaps the only property known with complete certainty is that the initial parameters need to "break symmetry" between different units.
  - If two hidden units with the same activation function are connected to the same inputs, then these units must have different initial parameters.
- General Rules:
  - Larger initial weights will yield a stronger symmetry-breaking effect. Therefore, the initial weights should be small (not very small)
  - The initial weights should not be zero to avoid dead activation phenomenon
  - Initial weights should have good variance to cover different viewpoints

Weight Initialization



- Uniform Initialization (traditionally for Sigmoid and Tanh units)

$$w_{ij}^{k} \sim U\left(\frac{-1}{\sqrt{fan_{in}}}, \frac{1}{\sqrt{fan_{in}}}\right)$$

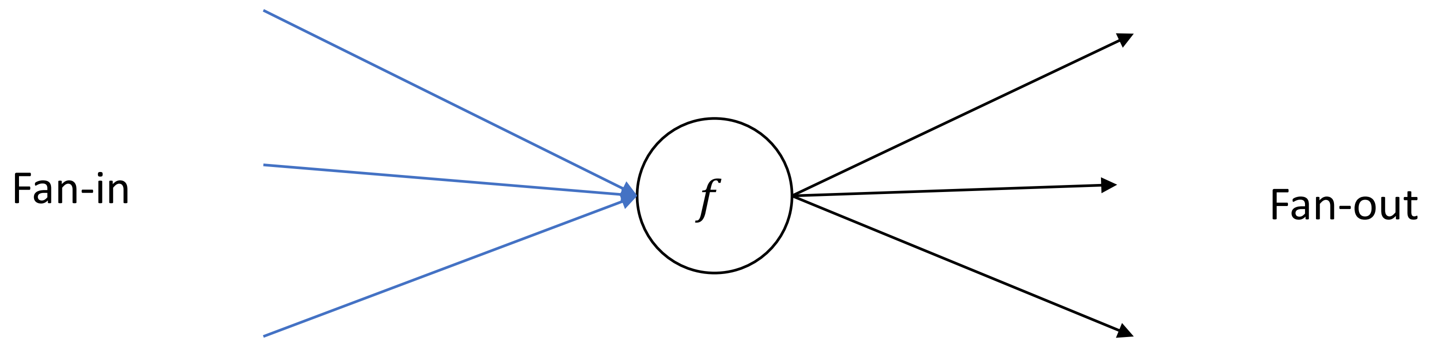
- Xavier / Glorot Initialization [1] (for Sigmoid)

$$w_{ij}^{k} \sim U\left(\frac{-\sqrt{6}}{\sqrt{fan_{in} + fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in} + fan_{out}}}\right)$$

$$w_{ij}^{k} \sim N\left(0, \frac{2}{fan_{in} + fan_{out}}\right)$$

[1]: Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

Weight Initialization



- He Initialization [1] (for ReLU)

$$w_{ij}^{k} \sim U\left(\frac{-\sqrt{6}}{\sqrt{fan_{in}}}, \frac{\sqrt{6}}{\sqrt{fan_{in}}}\right)$$

$$w_{ij}^{k} \sim N\left(0, \frac{2}{fan_{in}}\right)$$

[1]:, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034)..

Weight Initialization

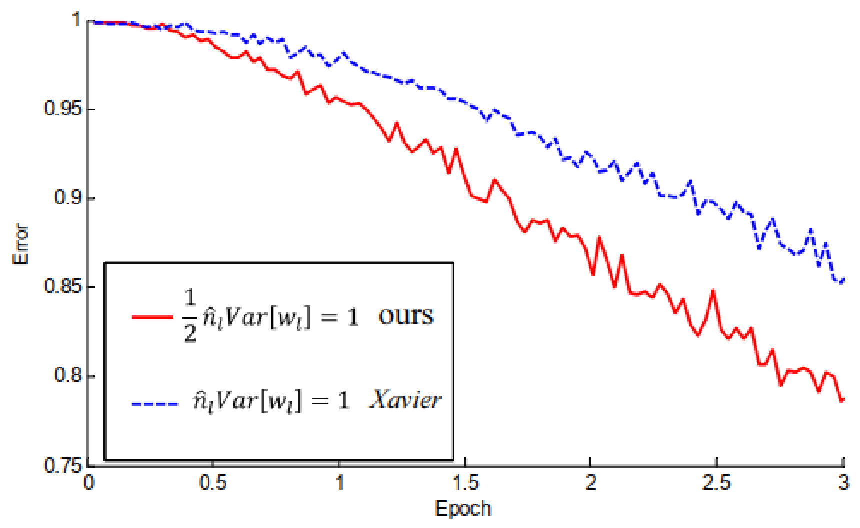


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

Source: K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034)..

Weight Initialization

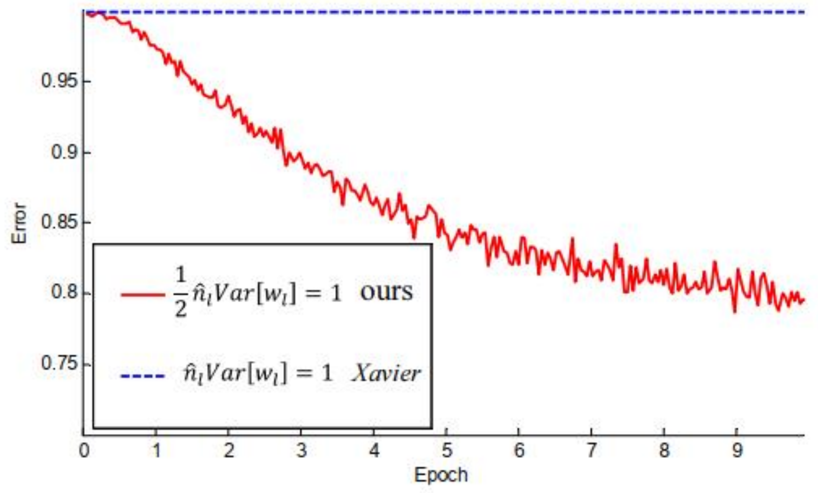


Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But "*Xavier*" (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.

Source: K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034)..