UNIT – II
INTRODUCTION TO DEEP LEARNING

Final Year BTECH

Subject : **Deep Learning (PE3)**

Unit II : Contents

Introduction to Deep Learning

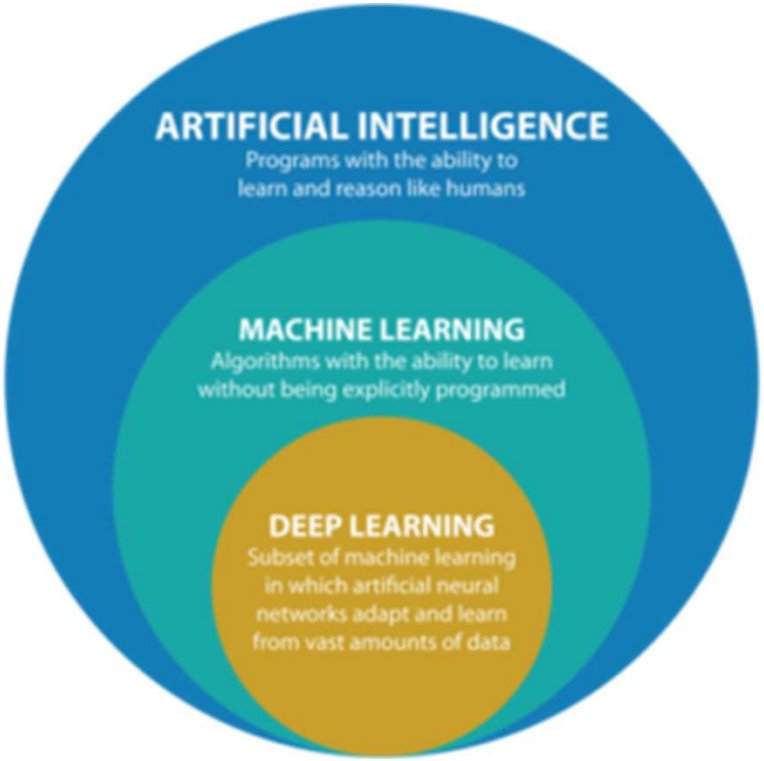
Introduction to Deep learning, Architecture, Multilayer Perceptron (MLP), Training MLP, Chain Rule, Back-propagation, Optimization Methods, Feedforward Neural Network, Examples of Deep Learning.

Introduction to Deep learning, Architecture, Multilayer Perceptron (MLP), Training MLP, Chain Rule

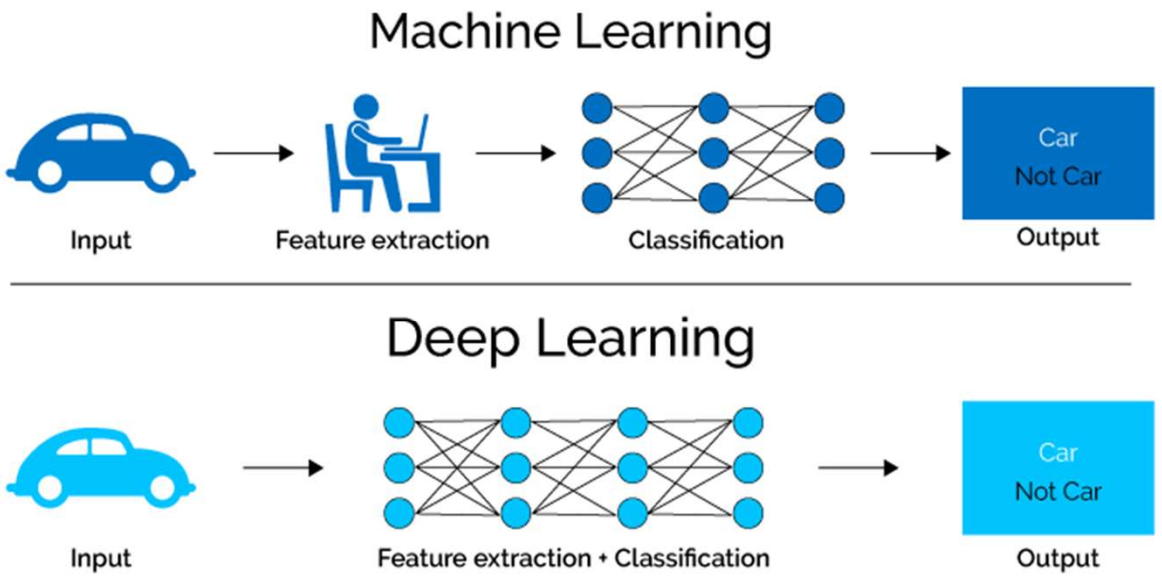
Sources:

- Machine learning with neural networks An introduction for scientists and engineers, Bernhard Mehlig

Introduction to Deep learning

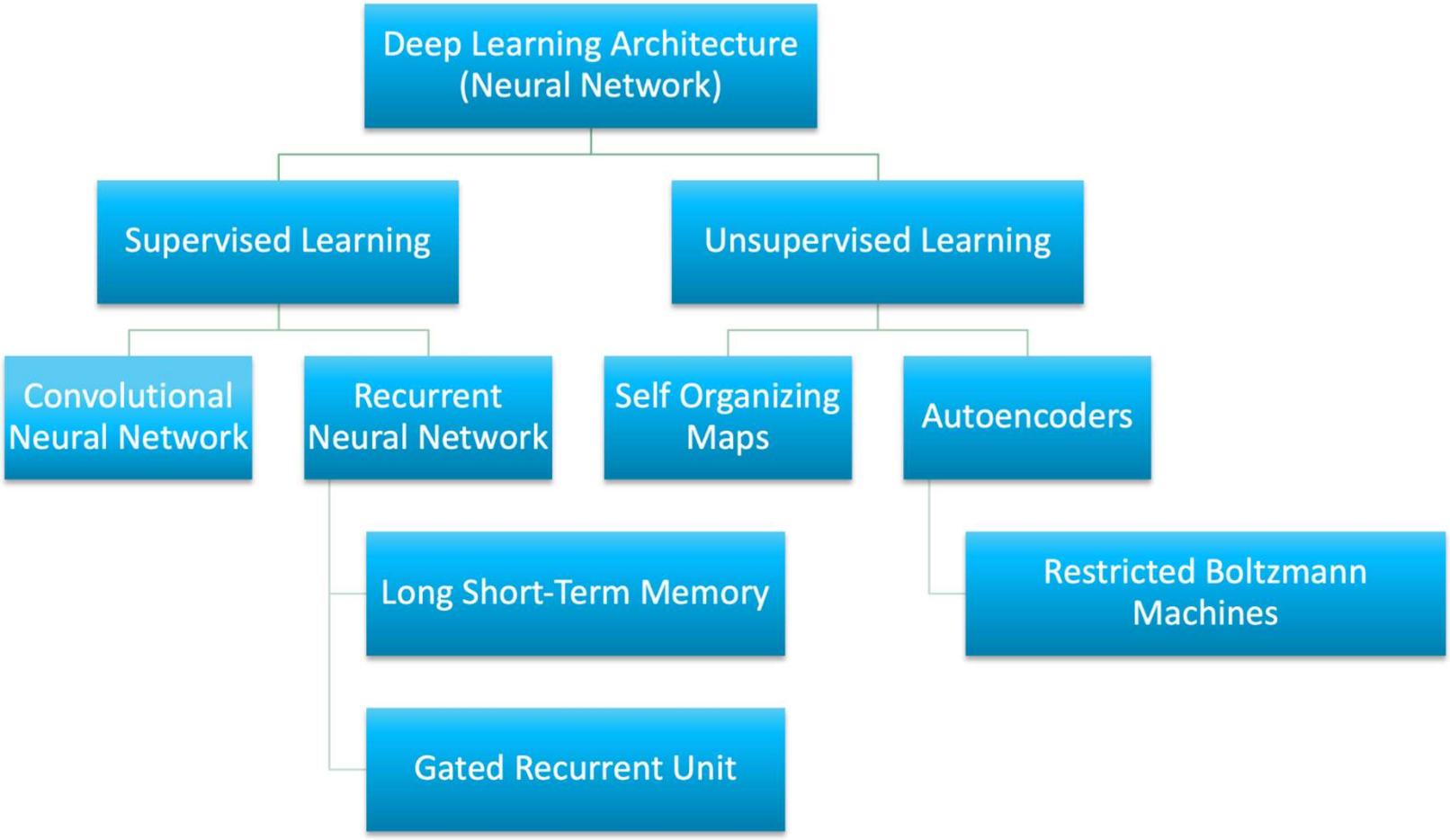


- Deep Learning comes under the artificial intelligence umbrella, either alongside machine learning or as a subset.
- The difference is that machine learning uses algorithms developed for specific tasks.
- Deep learning is more of a data representation based upon multiple layers of a matrix, where each layer uses output from the previous layer as input.



Source (Figure) : https://semiengineering.com/deep-learning-spreads/

Deep learning Architecture



Source (Figure) : https://developer.ibm.com/developer/default/articles/cc-machine-learning-deep-learning-architectures/images/DL-arch.png

Deep Neural Network

ANN is a **deep feed-forward neural** network as it processes inputs in the forward direction only. Artificial Neural Networks are capable of **learning non-linear functions**. The **activation function** of ANNs helps in learning **any complex relationship between input and output**.

RNN is designed to **overcome the looping constraint of ANN in hidden layers**. Deep recurrent networks are capable of **solving problems related to audio data, text data, and time-series data**. Recurrent neural networks capture sequential information available in the input data. **RNN works on parameter sharing**.

CNN based models in Deep Neural Networks are used in **video and image processing**. **Filters or kernels** are **the building blocks of CNN**. By using conventional operations kernels **extract relevant and correct features** from the input data.

https://www.jigsawacademy.com/blogs/ai-ml/deep-neural-network/

Deep Neural Network

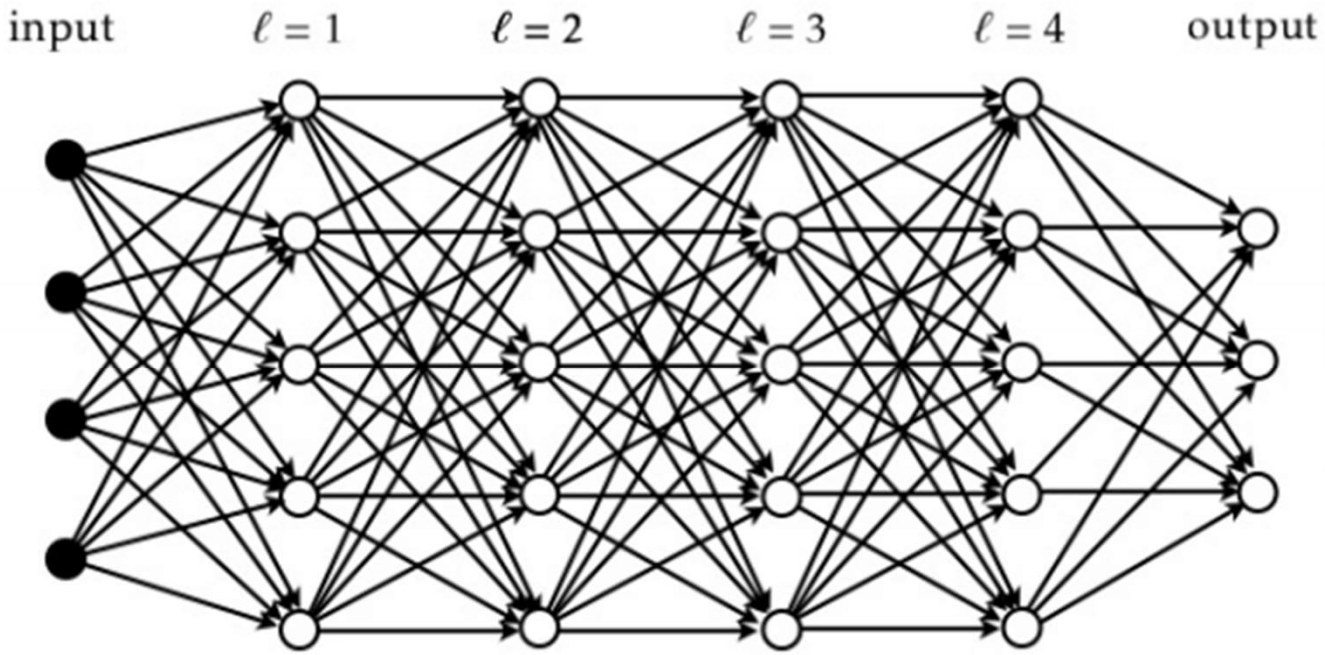


Figure 7.9: Fully connected deep network with four hidden layers.

Machine learning with neural networks An introduction for scientists and engineers, Bernhard Mehlig (page no.130)

Introduction to Deep learning

Why it is sometimes necessary to have a hidden layer?

In order to solve problems that are not linearly separable.

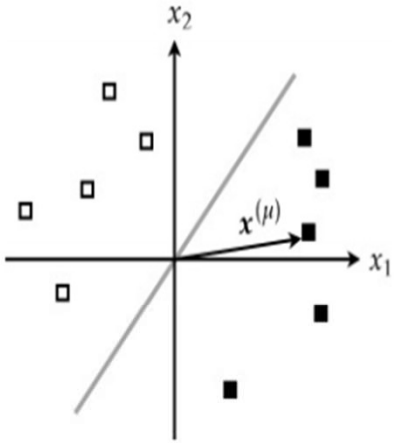


Figure 5.3: Classification problem with two-dimensional real-valued inputs and targets equal to ±1. The gray solid line is the decision boundary. Legend: ■ corresponds to $t^{(\mu)} = 1$, and □ to $t^{(\mu)} = -1$.

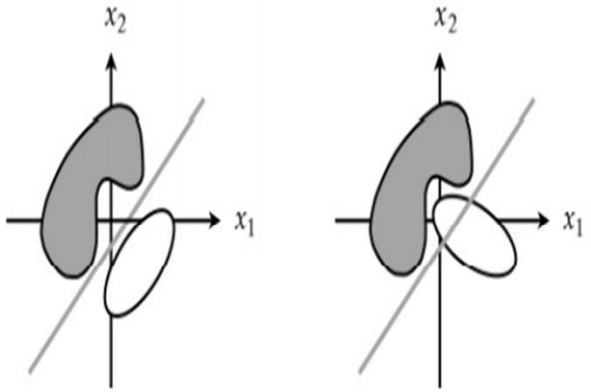


Figure 5.6: Linearly separable and non-separable data in two-dimensional input space.

Machine learning with neural networks An introduction for scientists and engineers, Bernhard Mehlig (page no.91-123)

Difference between Machine Learning and Deep Learning

Machine Learning	Deep Learning
Works on small amount of Dataset for accuracy.	Works on Large amount of Dataset.
Dependent on Low-end Machine.	Heavily dependent on High-end Machine.
Divides the tasks into sub-tasks, solves them individually and finally combine the results.	Solves problem end to end.
Takes less time to train.	Takes longer time to train.
Testing time may increase.	Less time to test the data.

https://www.geeksforgeeks.org/introduction-deep-learning/#:~:text=Deep%20Neural%20Network%20%E2%80%93%20It%20is,is%20multi%2Dlayer%20belief%20networks.

Difference Between Neural Network And Deep Neural Network

- The Deep Neural Network is more **creative and complicated** than the neural network. Deep Neural Network **algorithms can recognize sounds and voice commands, make predictions, think creatively, and do analysis**. They act like the human brain.

- Neural networks give **one result**. It can be an **action, a word, or a solution**. On the other hand, Deep Neural Networks provides solutions by globally solving problems based on the information given.

- **Specific data input and algorithm** is required for a neural network, whereas Deep Neural Networks are capable of solving problems **without a specific data amount**.

https://www.jigsawacademy.com/blogs/ai-ml/deep-neural-network/

Multilayer Perceptron (MLP)

- A **multilayer perceptron** (MLP) is a class of feedforward artificial neural network (ANN).
- The term MLP is used ambiguously, sometimes loosely to ***any*** **feedforward ANN**, sometimes strictly to refer to networks composed of **multiple layers of perceptrons** (with threshold activation);
- Multilayer perceptrons are sometimes colloquially referred to as "**vanilla**" neural networks, especially when they have a **single hidden layer**.
- An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function.
- MLP utilizes a supervised learning technique called **backpropagation for training**. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron.
- It can distinguish data that is not linearly separable

The multilayer perceptron (MLP) is used for a variety of tasks, such as stock analysis, image identification, spam detection, and election voting predictions.

https://en.wikipedia.org/wiki/Multilayer_perceptron

Multilayer Perceptron (MLP)

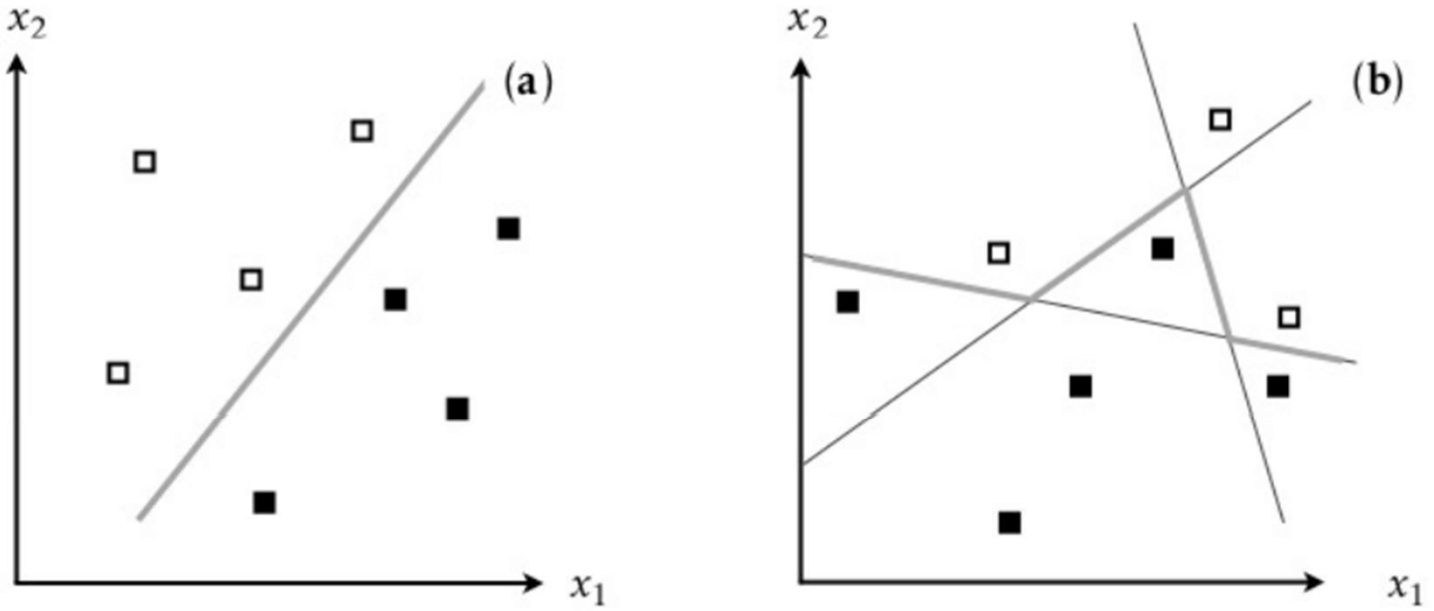
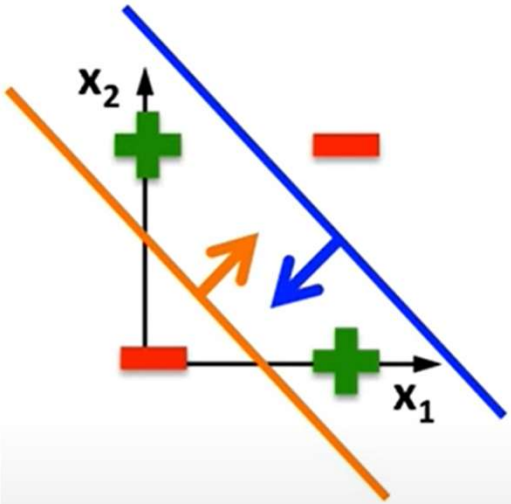


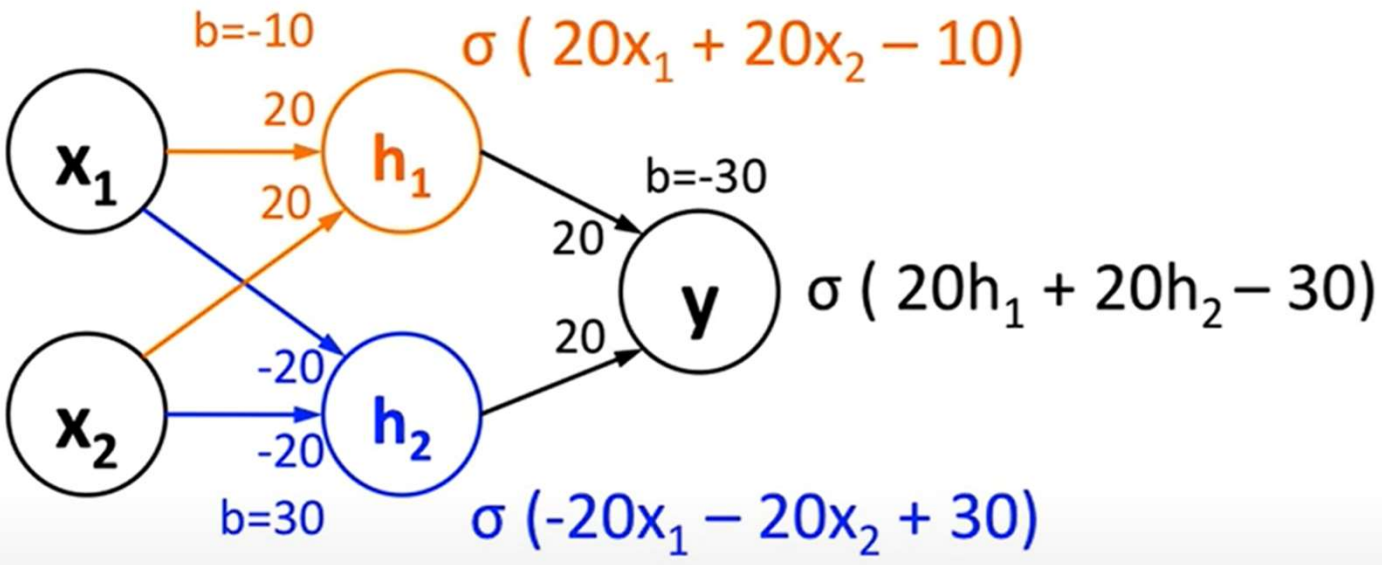
Figure 5.13: (**a**) Linearly separable problem. (**b**) Problems that are not linearly separable can be solved by a piecewise linear decision boundary. Legend: ■ corresponds to $t^{(\mu)} = 1$, and □ to $t^{(\mu)} = -1$.

Machine learning with neural networks An introduction for scientists and engineers, Bernhard Mehlig (page no.92)

Solving XOR using Multilayer Perceptron

Linear classifiers cannot solve this





$\sigma(20*0 + 20*0 - 10) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$
$\sigma(20*0 + 20*1 - 10) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$

$\sigma(-20*0 - 20*0 + 30) \approx 1$
$\sigma(-20*1 - 20*1 + 30) \approx 0$
$\sigma(-20*0 - 20*1 + 30) \approx 1$
$\sigma(-20*1 - 20*0 + 30) \approx 1$

$\sigma(20*0 + 20*1 - 30) \approx 0$
$\sigma(20*1 + 20*0 - 30) \approx 0$
$\sigma(20*1 + 20*1 - 30) \approx 1$
$\sigma(20*1 + 20*1 - 30) \approx 1$

Multilayer Perceptron (MLP)

x_1	x_2	t
0.1	0.95	0
0.2	0.85	0
0.2	0.9	0
0.3	0.75	1
0.4	0.65	1
0.4	0.75	1
0.6	0.45	0
0.8	0.25	0
0.1	0.65	1
0.2	0.75	1
0.7	0.2	1

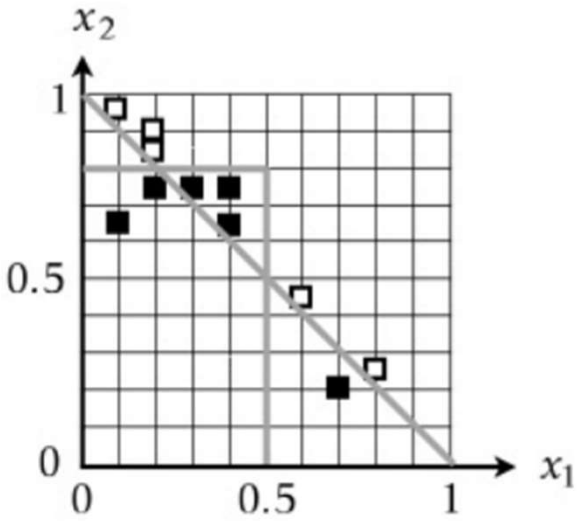
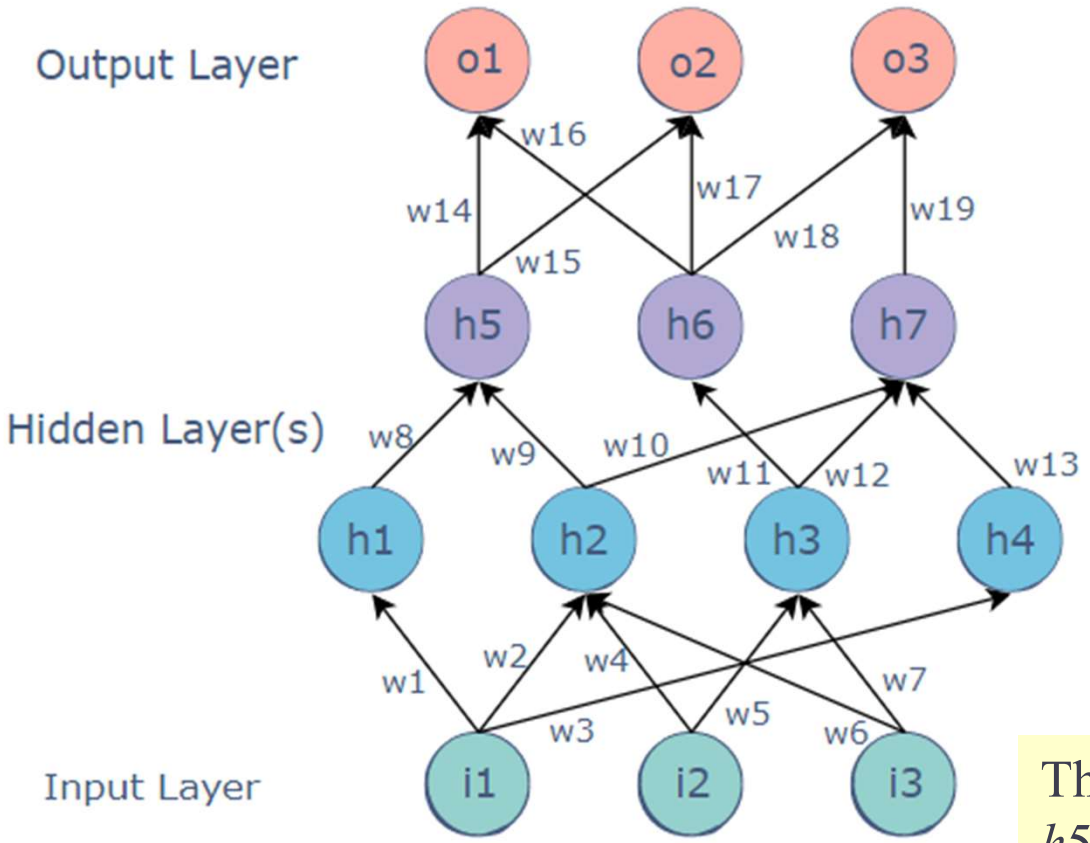


Figure 5.23: Inputs and targets for a classification problem. The targets are either $t = 0$ (□) or $t = 1$ (■). The three decision boundaries (gray lines) illustrate a solution to the problem using a multilayer perceptron.

Machine learning with neural networks An introduction for scientists and engineers, Bernhard Mehlig (page no.100)

Multilayer Perceptron (MLP)

The MLP is a **feedforward neural network**, which means that the data is transmitted from the input layer to the output layer in the forward direction.



All the connections have been assigned different weights

The value of the h5 node could be:
$h5=h1.w8+h2.w9$

https://www.educative.io/edpresso/what-is-a-multi-layered-perceptron

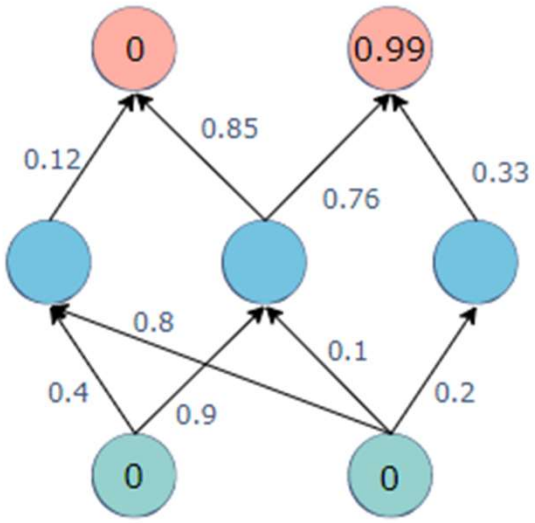
Backpropagation in Multilayer Perceptron (MLP)

- Backpropagation is a technique used to **optimize the weights** of an MLP using the outputs as inputs.
- In a conventional MLP, random weights are assigned to all the connections. These random weights propagate values through the network to produce the actual output.
- Naturally, this output would differ from the expected output. The difference between the two values is called the **error**.
- **Backpropagation refers to the process of sending this error back through the network, readjusting the weights automatically** so that eventually, the error between the actual and expected output is minimized.
- In this way, the output of the current iteration becomes the input and affects the next output. This is repeated until the correct output is produced. The weights at the end of the process would be the ones on which the neural network works correctly.

https://www.educative.io/edpresso/what-is-a-multi-layered-perceptron

Backpropagation in Multilayer Perceptron (MLP)

How Backpropagation works??



Repeat the process for all inputs to obtain the fully trained network!

Training Set

Input	Expected Output
0, 0	0, 1
0, 1	1, 0
1, 0	1, 1

6 of 6

https://www.educative.io/edpresso/what-is-a-multi-layered-perceptron

Chain Rule

- **Chain Rule is covered when you studied Differential Calculus.**
- **The derivatives of energy functions are evaluated the Chain rule.**

$$x = 5v - 2$$

$$y = 3x - 2$$

$$z = y^2$$

$$\frac{dz}{dv} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

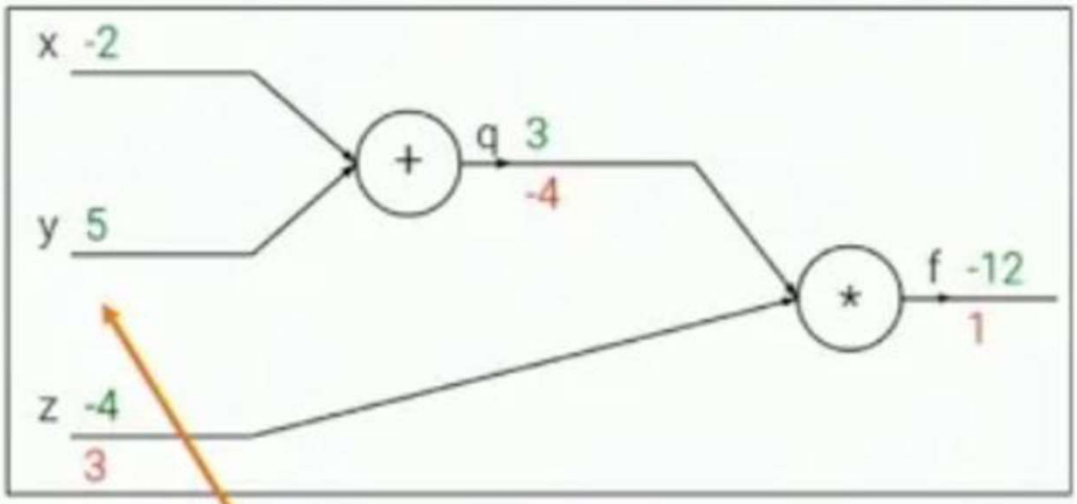
Machine learning with neural networks An introduction for scientists and engineers, Bernhard Mehlig.

Chain Rule

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

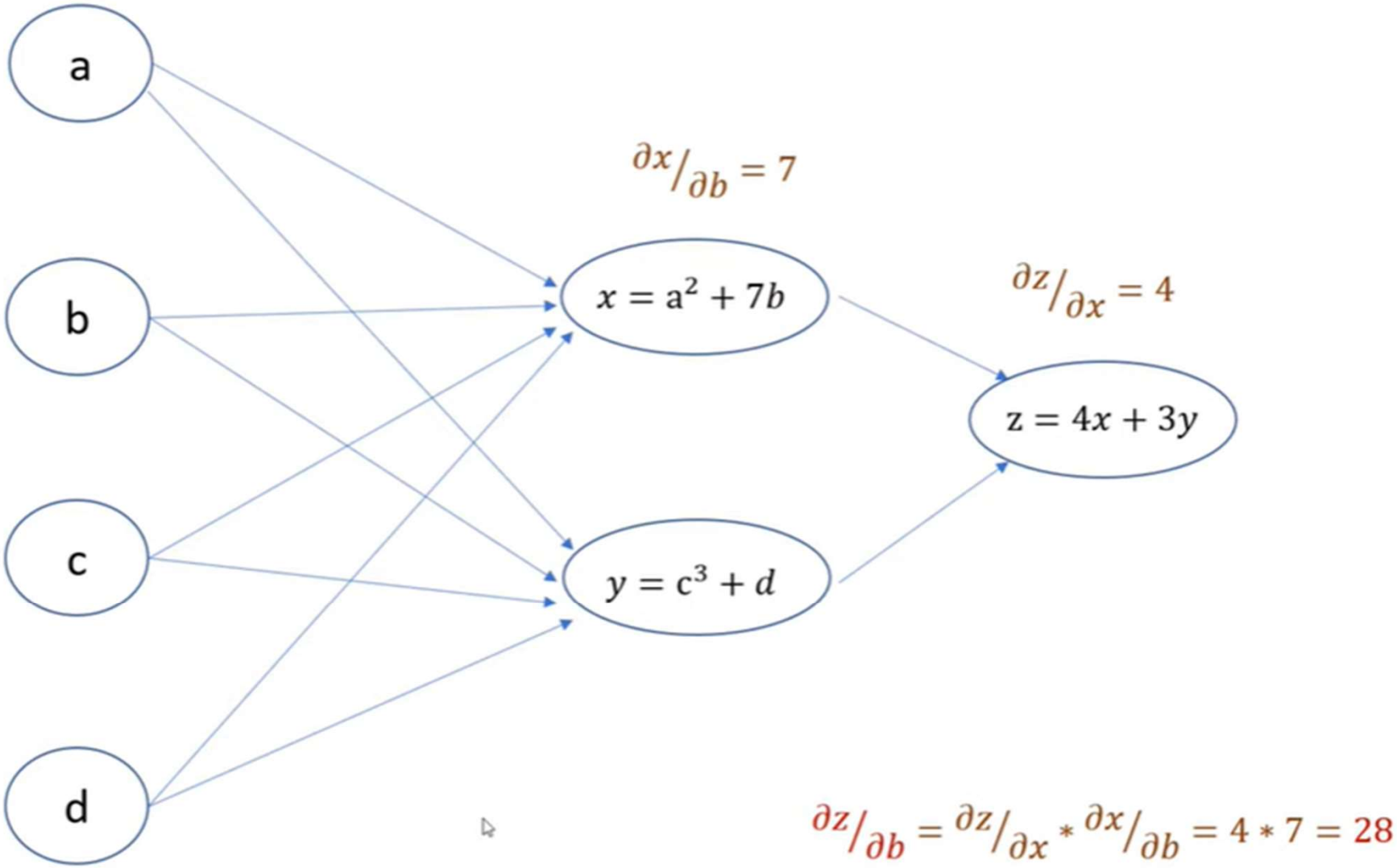
2. Backward pass: Compute derivatives

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

Chain Rule

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

Chain Rule



https://www.youtube.com/watch?v=5ogmEkujoqE

Training Multilayer Perceptron (MLP)

Neural network needs to be trained on dataset

1. Data Preparation
2. Weight Update
3. Prediction

Training MLP- Data Preparation

- Data must be numerical, real values.
- If **categorical data**, such as a sex attribute with the values "male" and "female", then you can **convert it to a real-valued representation** called a one hot encoding.
- one hot encoding can be used on the **output variable** in classification problems with more than one class.
- This would create a binary vector from a single column that would be easy to directly compare to the output of the neuron in the network's output layer, that as described above, would output one value for each class.
- Data Preprocessing

1. Normalization
2. Standardization
3. Scaling

Training MLP- Weight Update Algorithm

Stochastic Gradient Descent

- The **classical and still preferred training algorithm** for neural networks is called stochastic gradient descent.
- **One row of data** is exposed to the network at a time as input and it **processes the input upward activating neurons** as it goes to finally **produce an output value**.
- The output of the network is compared to the expected output and an **error is calculated**.
- This error is then **propagated back through the network**, one layer at a time, and the weights are updated according to the amount that they contributed to the error. This is called the backpropagation algorithm.
- The process is repeated for all of the examples in your training data.
- **One round of updating the network for the entire training dataset is called an epoch**.
- A network may be trained for tens, hundreds or many thousands of epochs.

Training MLP- Weight Update Algorithm

- The **weights in the network can be updated** from the **errors calculated** for each training example
- The **errors** can be saved up **across all of the training examples and the network can be updated at the end**.
- Datasets are so large and computational efficiencies, **the size of the batch**, the **number of examples the network is shown before an update** is often reduced to a small number, such as tens or hundreds of examples.
- **The amount that weights are updated is controlled by a configuration parameters called the learning rate**. It is also called the step size and controls the step or change made to network weight for a given error.
- Often small weight sizes are used such as 0.1 or 0.01 or smaller.

Training MLP- Weight Update Algorithm

- The update equation can be complemented with additional configuration terms that you can set.
- Momentum is a term that incorporates the properties from the previous weight update to allow the weights to continue to change in the same direction even when there is less error being calculated.
- Learning Rate Decay is used to decrease the learning rate over epochs to allow the network to make large changes to the weights at the beginning and smaller fine tuning changes later in the training schedule.

Training MLP- Prediction

Stochastic Gradient Descent

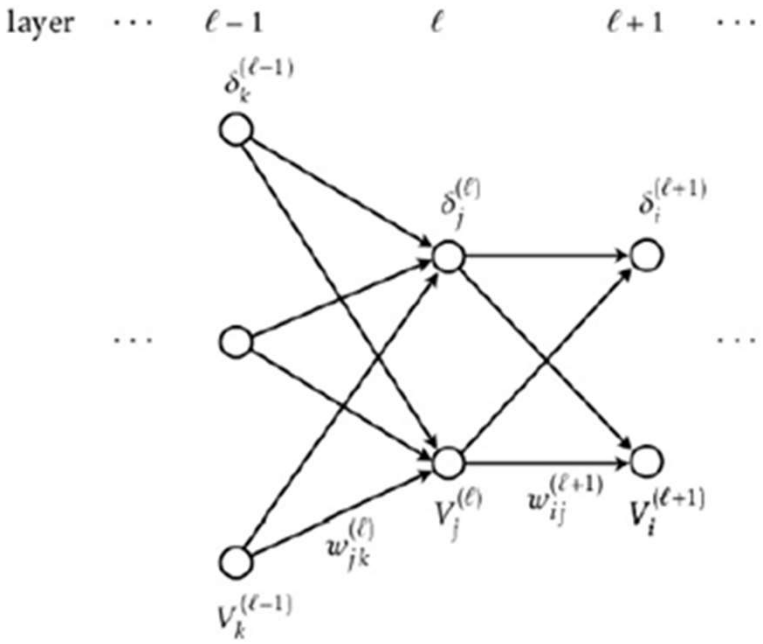


Figure 6.1: Illustrates the notation used in Algorithm 4.

Algorithm 4 stochastic gradient descent

initialise weights $w_{mn}^{(\ell)}$ to random numbers, thresholds to zero, $\theta_m^{(\ell)} = 0$;
for $\nu = 1, \ldots, \nu_{\max}$ **do**
  choose a value of μ and apply pattern $x^{(\mu)}$ to input layer, $V^{(0)} \leftarrow x^{(\mu)}$;
  for $\ell = 1, \ldots, L$ **do**
    propagate forward: $V_j^{(\ell)} \leftarrow g\left(\sum_k w_{jk}^{(\ell)} V_k^{(\ell-1)} - \theta_j^{(\ell)}\right)$;
  end for
  compute errors for output layer: $\delta_i^{(L)} \leftarrow g'(b_i^{(L)})(t_i - V_i^{(L)})$;
  for $\ell = L, \ldots, 2$ **do**
    propagate backward: $\delta_j^{(\ell-1)} \leftarrow \sum_i \delta_i^{(\ell)} w_{ij}^{(\ell)} g'(b_j^{(\ell-1)})$;
  end for
  for $\ell = 1, \ldots, L$ **do**
    change weights and thresholds: $w_{mn}^{(\ell)} \leftarrow w_{mn}^{(\ell)} + \eta \delta_m^{(\ell)} V_n^{(\ell-1)}$ and $\theta_m^{(\ell)} \leftarrow \theta_m^{(\ell)} - \eta \delta_m^{(\ell)}$;
  end for
end for

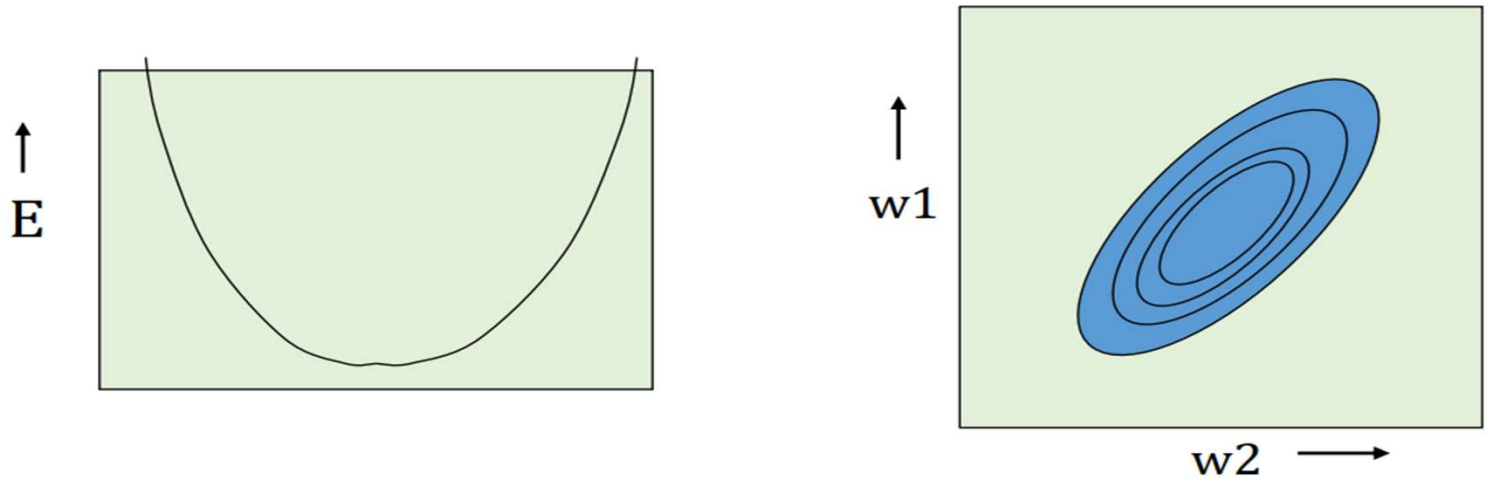
Machine learning with neural networks An introduction for scientists and engineers, Bernhard Mehlig, pg no-106

Gradient Descent

The Error Surface

The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.

- For a linear neuron, it is a quadratic bowl.
- Vertical cross-sections are parabolas.
- Horizontal cross-sections are ellipses.



Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence AlgorithmsNikhil Buduma, pg no-28

Gradient Descent

- How we might **minimize the squared error over all of the training examples** by simplifying the problem.
- E.g Linear neuron only has **two inputs** (and thus only **two weights**, w1 and w2).
- Imagine a three-dimensional space where the **horizontal dimensions** correspond to the **weights w1 and w2** , and the **vertical dimension** corresponds to the value of the **error function E**.
- In this space, **points in the horizontal plane correspond to different settings of the weights, and the height at those points corresponds to the incurred error**.
- If we consider the errors we make over all possible weights, we get a surface in this three-dimensional space, in particular, a quadratic bowl as shown in Figure.

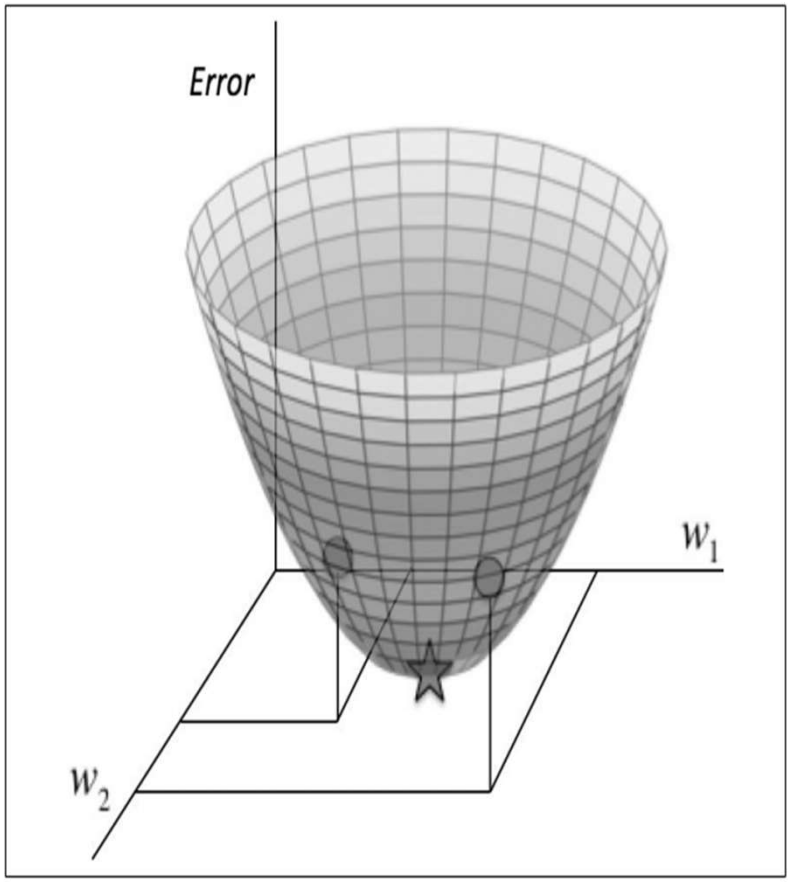


Figure 2-2. The quadratic error surface for a linear neuron

Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence AlgorithmsNikhil Buduma, pg no-28

Gradient Descent

- Conveniently visualize this surface as a set of elliptical contours, where the minimum error is at the center of the ellipses.
- Two-dimensional plane where the dimensions correspond to the two weights.
- Contours correspond to settings of w1 and w2 that evaluate to the same value of E.
- The closer the contours are to each other, the steeper the slope.
  In fact, it turns out that the direction of the steepest descent is always perpendicular to the contours.
- **This direction is expressed as a vector known as the gradient**.
- Develop a high-level strategy for how to find the values of the weights that minimizes the error function.
- Randomly initialize the weights of our network to find ourselves somewhere on the horizontal plane.

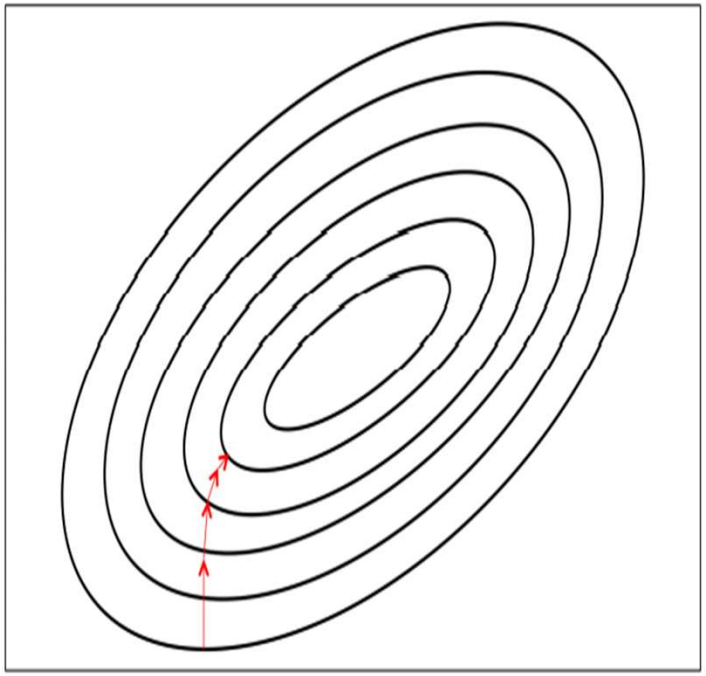


Figure 2-3. Visualizing the error surface as a set of contours

Gradient Descent

- By evaluating the gradient at current position, we can find the direction of steepest descent, and we can take a step in that direction.
- Find a new position that's closer to the minimum than before.
- Reevaluate the direction of steepest descent by taking the gradient at this new position and taking a step in this new direction.
- It's easy to see that, as shown in Figure, following this strategy will eventually get us to the point of minimum error.
- This algorithm is known as gradient descent, and it is used to tackle the problem of training individual neurons and the more general challenge of training entire networks.

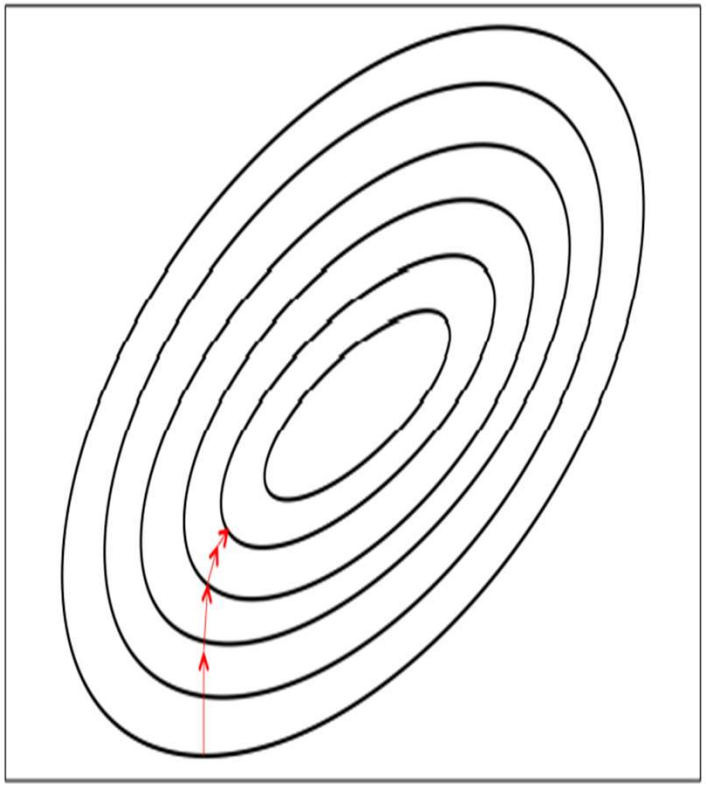
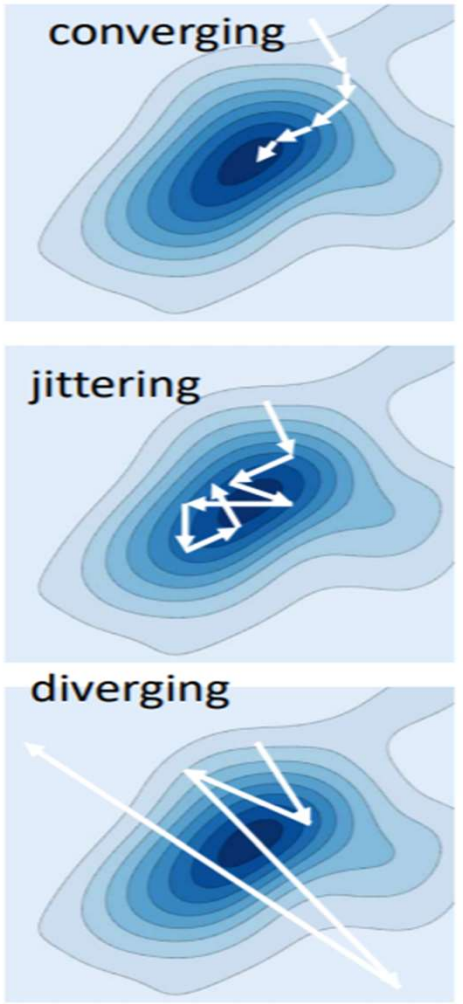


Figure 2-3. Visualizing the error surface as a set of contours

Convergence

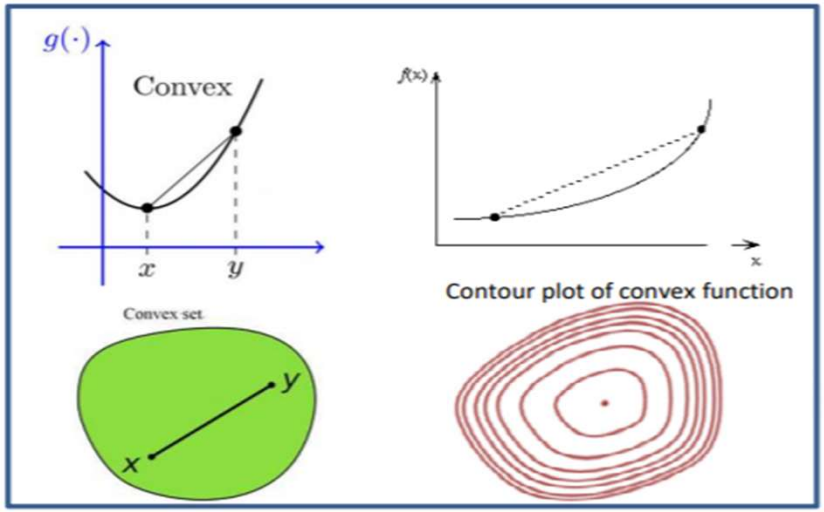
- An iterative algorithm is said to *converge* to a solution if the value updates arrive at a fixed point
  - Where the gradient is 0 and further updates do not change the estimate
- The algorithm may not actually converge
  - It may jitter around the local minimum
  - It may even diverge
- Conditions for convergence?



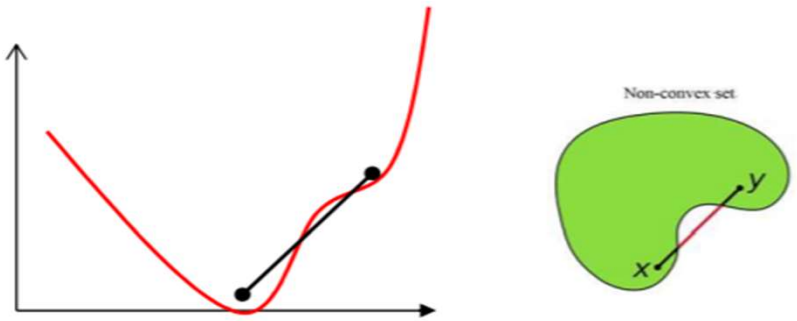
Convergence

Convex Loss Functions

- A surface is "convex" if it is continuously curving upward
  - We can connect any two points above the surface without intersecting it
  - Many mathematical definitions that are equivalent



- Caveat: Neural network error surface is generally not convex
  - Streetlight effect



Learning Rate (Hyper-parameter)

- In practice, at each step of moving perpendicular to the contour, it needs to determine how far we want to walk before recalculating our new direction.
- This distance needs to depend on the steepness of the surface.
- Picking the learning rate is a hard problem
- if we pick a learning rate that's too small, we risk taking too long during the training process.
- But if we pick a learning rate that's too big, we'll mostly likely start diverging away from the minimum.

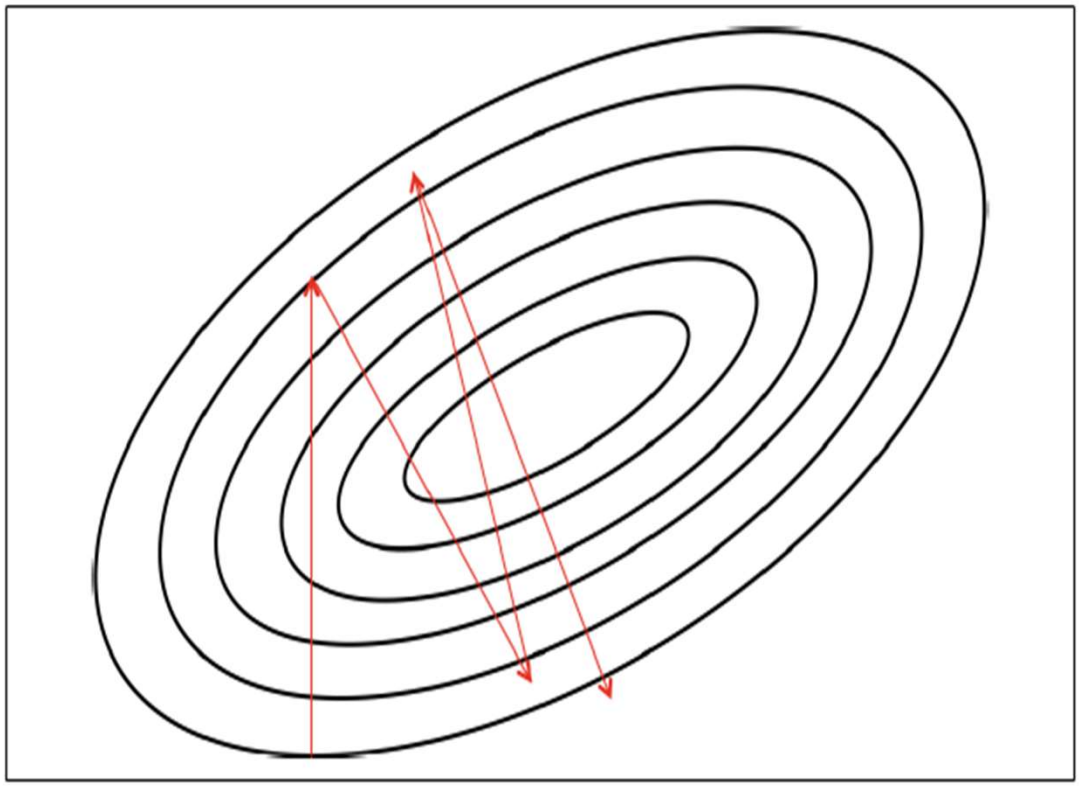
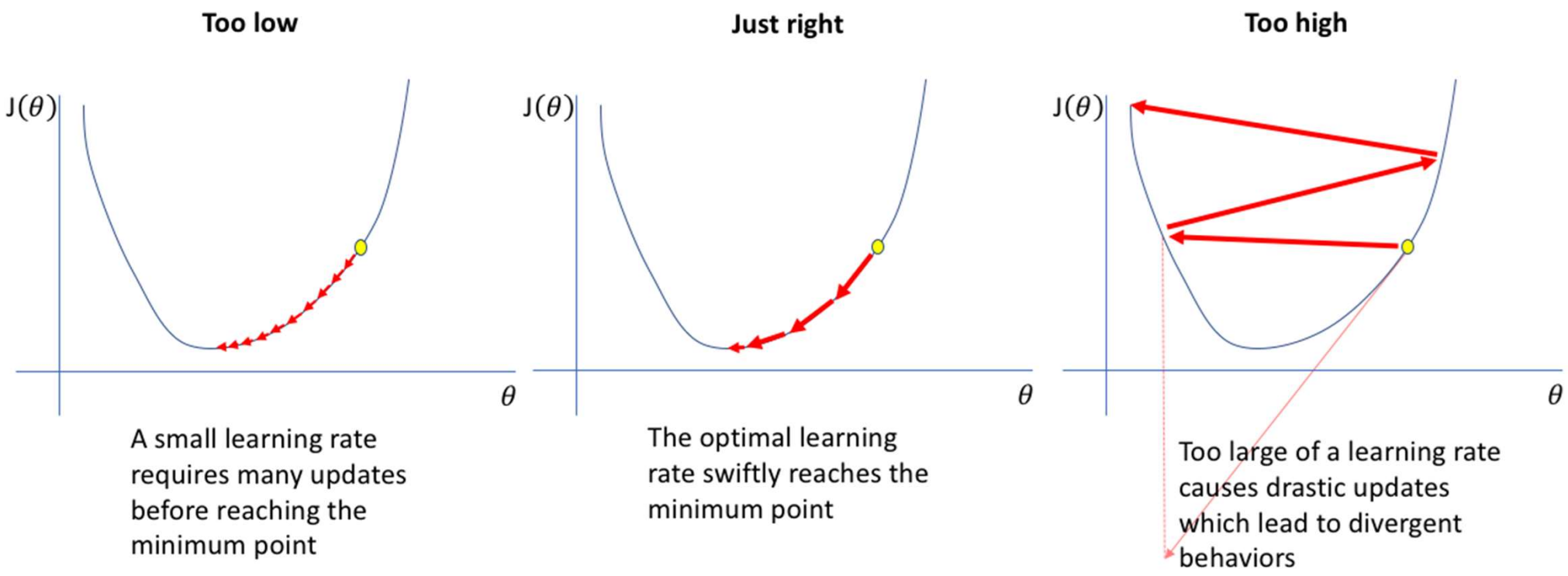


Figure 2-4. Convergence is difficult when our learning rate is too large

Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence AlgorithmsNikhil Buduma, pg no-30

Learning Rate



Learning rate selection. Source: https://www.jeremyjordan.me/nn-learning-rate/

Optimization

- Optimization is the most essential ingredient in the recipe of machine learning algorithms.
- It starts with defining some kind of loss function/cost function and ends with minimizing it using one or the other optimization routine.
- The choice of optimization algorithm can make a difference between getting a good accuracy in hours or days.

Optimization

- For a deep learning problem, we will usually define a *loss function* first. Once we have the loss function, we can use an optimization algorithm in attempt to minimize the loss.
- In optimization, a loss function is often referred to as the *objective function* of the optimization problem.
- By tradition and convention most optimization algorithms are concerned with *minimization*. If we ever need to maximize an objective there is a simple solution: just flip the sign on the objective.
- The objective function of the optimization algorithm is usually a loss function based on the training dataset, the goal of optimization is to reduce the training error means to reduce the generalization error.
- To accomplish the latter we need to pay attention to overfitting in addition to using the optimization algorithm to reduce the training error.
- In deep learning, generally, to approach the optimal value, gradient descent is applied to the weights, and optimization is achieved by running many epochs with large datasets.

https://machinelearningmastery.com/tour-of-optimization-algorithms/
https://d2l.ai/chapter_optimization/optimization-intro.html

How Learning Differs from Pure Optimization

- Goal : Finding the parameters Θ that reduce a cost function $J(\Theta)$.
- Typically, the cost function with respect to the training set can be written as:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

- where **L** is the per-example loss function,
  - **f(x;θ):**the predicted output when the input is **x**, $\mathbf{p'_{data}}$ is the empirical distribution.
  - In the supervised learning case, **y** is the target output.
- The objective is to minimize the corresponding objective function

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim p_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

https://medium.com/inveterate-learner/deep-learning-book-chapter-8-optimization-for-training-deep-models-part-i-20ae75984cb2

Gradient descent

Most optimization algorithms now are based on *Gradient Descent* (GD) which requires a derivative calculation and hence, may not work with certain loss functions like the 0–1 loss (as it is not differentiable).

Gradient descent is a way to minimize an objective function J(θ)

θ ∈ Rd : model parameters

η: learning rate

∇θ J(θ): gradient of the objective function with regard to the parameters

Update equation: $\theta = \theta - \eta \cdot \nabla\theta\, J(\theta)$

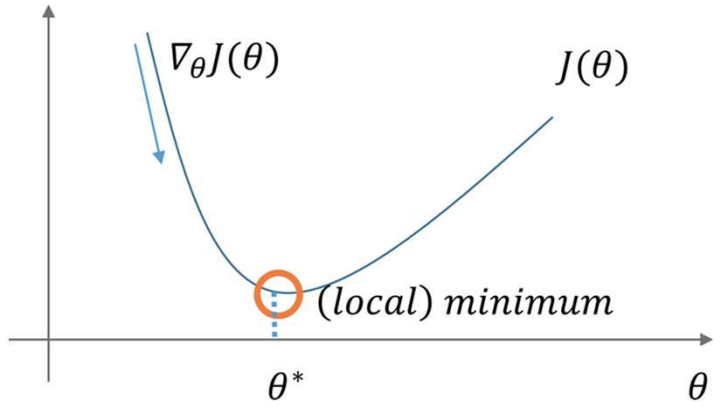


Figure: Optimization with gradient descent

https://medium.com/inveterate-learner/deep-learning-book-chapter-8-optimization-for-training-deep-models-part-i-20ae75984cb2

Gradient descent variants

Batch gradient descent

Stochastic gradient descent

Mini-batch gradient descent

Gradient Descent algorithm and its variants - GeeksforGeeks

Batch gradient descent

This is a type of gradient descent which processes all the training examples for each iteration of gradient descent.

But if the number of training examples is large, then batch gradient descent is computationally very expensive.

Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, prefer to use stochastic gradient descent or mini-batch gradient descent.

Pros:

Guaranteed to converge to **global** minimum for **convex** error surfaces and to a **local** minimum for **non-convex** surfaces.

Cons:

Very slow.

Intractable for datasets that **do not fit in memory**. **No online learning**.

Gradient Descent algorithm and its variants - GeeksforGeeks

Batch gradient descent

Variables used:

Let m be the number of training examples.

Let n be the number of features.

Note: if b == m, then mini batch gradient descent will behave similarly to batch gradient descent.

Algorithm for batch gradient descent :

Let $h_\theta(x)$ be the hypothesis for linear regression. Then, the cost function is given by:

Let Σ represents the sum of all training examples from i=1 to m.

Gradient Descent algorithm and its variants - GeeksforGeeks

Batch gradient descent Algorithm

$$J_{train}(\theta) = (1/2m)\ \Sigma(\ h_\theta(x^{(i)})\ -\ y^{(i)})^2$$

Repeat {

$\theta j = \theta j - (\text{learning rate}/m) * \Sigma(\ h_\theta(x^{(i)})\ -\ y^{(i)})x_j^{(i)}$

For every j =0 …n

}

Where $x_j^{(i)}$ Represents the j^{th} feature of the i^{th} training example. So if *m* is very large(e.g. 5 million training samples), then it takes hours or even days to converge to the global minimum.That's why for large datasets, it is not recommended to use batch gradient descent as it slows down the learning.

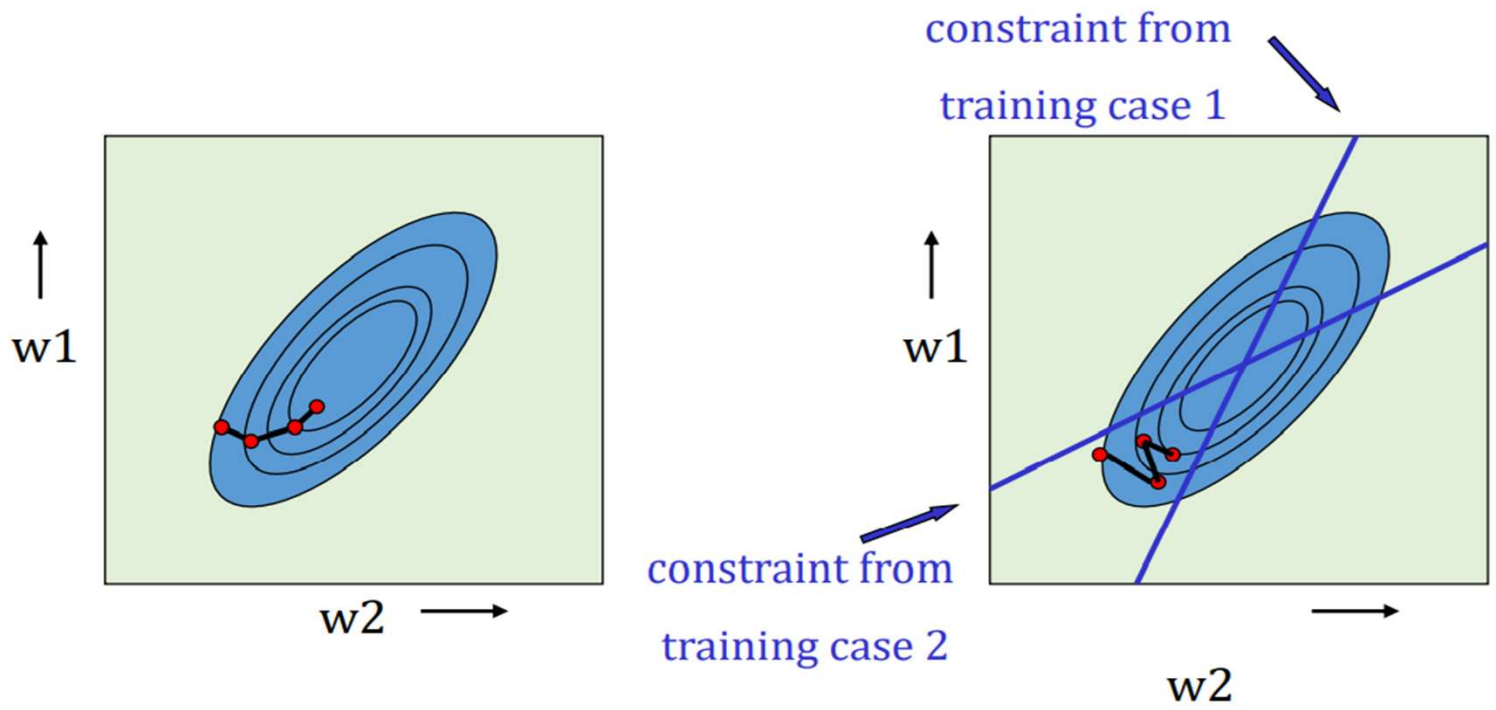
Gradient Descent algorithm and its variants - GeeksforGeeks

Batch gradient descent

Online versus Batch Learning

Batch learning does steepest descent on the error surface

Online learning zig-zags around the direction of steepest descent



[Gradient Descent algorithm and its variants - GeeksforGeeks]

Stochastic gradient descent

- This is a type of gradient descent which processes 1 training example per iteration.
- Hence, the parameters are being updated even after one iteration in which only a single example has been processed.
- Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.

Pros

Much faster than batch gradient descent.

Allows **online learning**.

Cons

High variance updates.

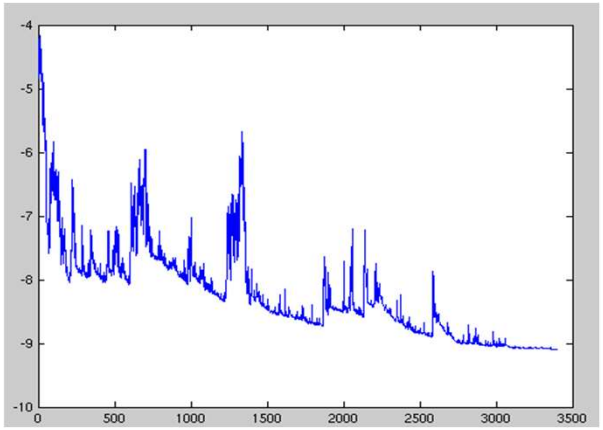


Figure: SGD fluctuation (Source: Wikipedia)

Gradient Descent algorithm and its variants - GeeksforGeeks

Stochastic gradient descent

Algorithm for stochastic gradient descent:

1) Randomly shuffle the data set so that the parameters can be trained evenly for each type of data.

2) As mentioned above, it takes into consideration one example per iteration.

Hence,

Let $(x^{(i)},y^{(i)})$ be the training example

$$Cost(\theta, (x^{(i)},y^{(i)})) = (1/2) \Sigma(h\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = (1/m) \Sigma Cost(\theta, (x^{(i)},y^{(i)}))$$

Repeat {

For i=1 to m{

$$\theta_j = \theta_j - (\text{learning rate}) * \Sigma(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

For every j =0 …n

}

}

Gradient Descent algorithm and its variants - GeeksforGeeks

Mini-batch gradient descent

- **Mini Batch gradient descent:** This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent.
- Here b examples where $b<m$ are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go.
- Thus, it works for larger training examples and that too with lesser number of iterations.

Pros

Reduces variance of updates.

Can exploit **matrix multiplication** primitives.

Cons

Mini-batch size is a hyperparameter. Common sizes are 50-256.

Usually referred to as SGD even when mini-batches are used.

Mini-batch gradient descent

Algorithm for mini batch gradient descent:

Say b be the no of examples in one batch, where b < m.

Assume b = 10, m = 100;

Note: However we can adjust the batch size. It is generally kept as power of 2. The reason behind it is because some hardware such as GPUs achieve better run time with common batch sizes such as power of 2.

```
Repeat {
 For i=1,11, 21,…..,91

    Let Σ be the summation from i to i+9 represented by k.

    θj = θj – (learning rate/size of (b) ) * Σ( hθ(x(k))  - y(k))xj(k)
        For every j =0 …n

}
```

[Gradient Descent algorithm and its variants - GeeksforGeeks]

Comparison of trade-offs of gradient descent variants

Method	Accuracy	Update Speed	Memory Usage	Online Learning
Batch gradient descent	Good	Slow	High	No
Stochastic gradient descent	Good (with annealing)	High	Low	Yes
Mini-batch gradient descent	Good	Medium	Medium	Yes

Table: Comparison of trade-offs of gradient descent variants

Gradient Descent algorithm and its variants – GeeksforGeeks

https://medium.com/inveterate-learner/deep-learning-book-chapter-8-optimization-for-training-deep-models-part-i-20ae75984cb2

Optimization Methods

Commonly used optimization methods

- Stochastic Gradient Descent with Momentum
- AdaGrad
- RMSProp
- Adam Optimizer

The choice of which algorithm to use, at this point, seems to depend largely on the user's familiarity with the algorithm (for ease of hyperparameter tuning)

https://medium.com/@minions.k/optimization-techniques-popularly-used-in-deep-learning-3c219ec8e0cc

https://towardsdatascience.com/optimization-algorithms-in-deep-learning-191bfc2737a4

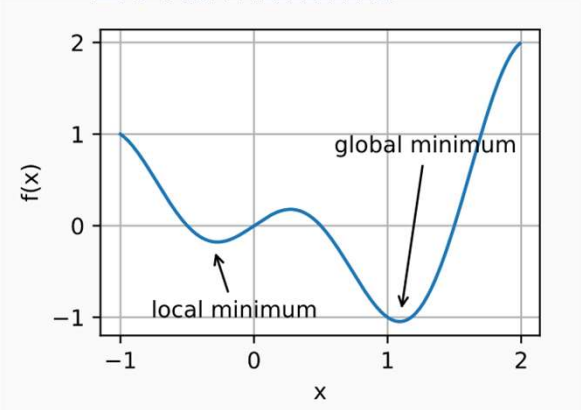
https://www.deeplearningbook.org/contents/optimization.html

Optimization Methods

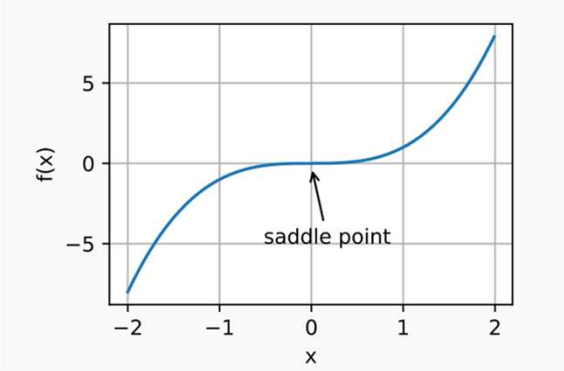
Optimization Challenges in Deep Learning

- Most vexing challenges are local minima, saddle points(a low part of a ridge between two higher points or peaks), and vanishing gradients

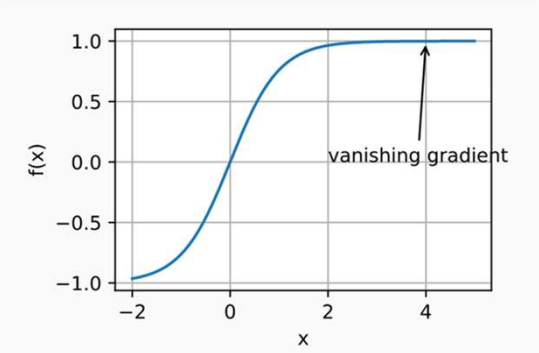
Local Minima



saddle points



vanishing gradients



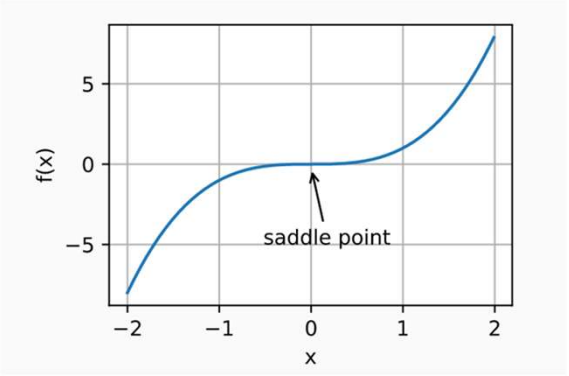
- The optimization problems may have many local minima.
- The problem may have even more saddle points, as generally the problems are not convex.
- Vanishing gradients can cause optimization to stall. Often a reparameterization of the problem helps. Good initialization of the parameters can be beneficial, too.

https://d2l.ai/chapter_optimization/optimization-intro.html

Optimization Methods

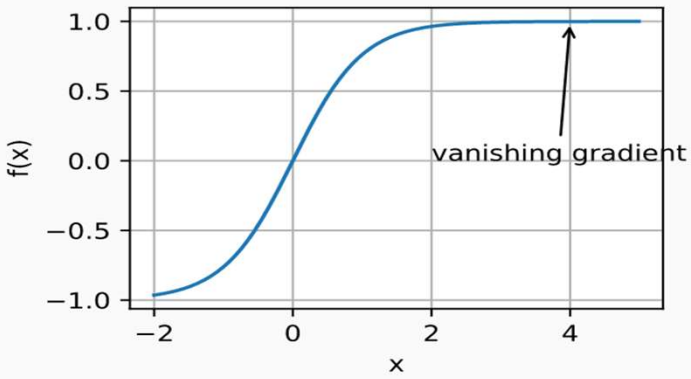
Optimization Challenges in Deep Learning

saddle points



- When we optimize neural networks or any high dimensional function, for most of the trajectory we optimize, the critical points(the points where the derivative is zero or close to zero) are saddle points. Saddle points, unlike local minima, are easily escapable

Optimization Methods



Optimization Challenges in Deep Learning

vanishing gradients

- When there are more layers in the network, the value of the product of derivative decreases until at some point the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes. this is vanishing gradient problem.
- With shallow networks, sigmoid function can be used as the small value of gradient does not become an issue.
- When it comes to deep networks, the vanishing gradient could have a significant impact on performance. The weights of the network remain unchanged as the derivative vanishes.
- During back propagation, a neural network learns by updating its weights and biases to reduce the loss function.
- In a network with vanishing gradient, the weights cannot be updated, so the network cannot learn. The performance of the network will decrease as a result.

Backpropogation

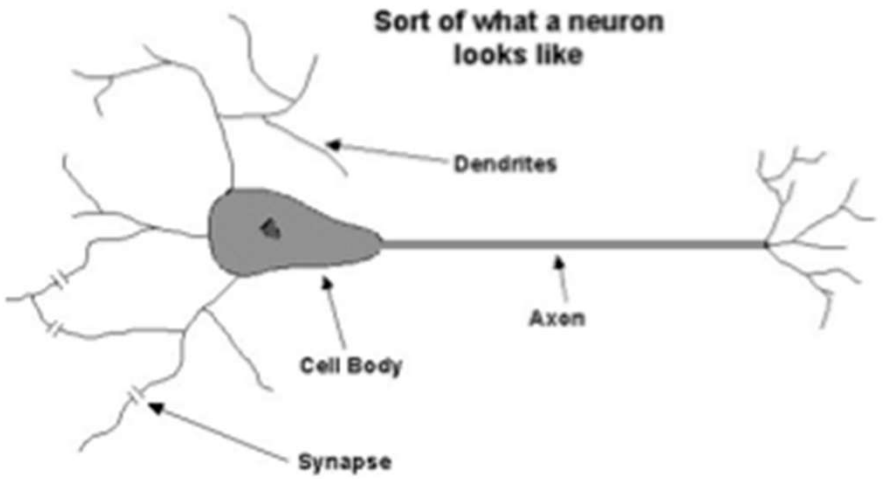
Feedforward Neural Network

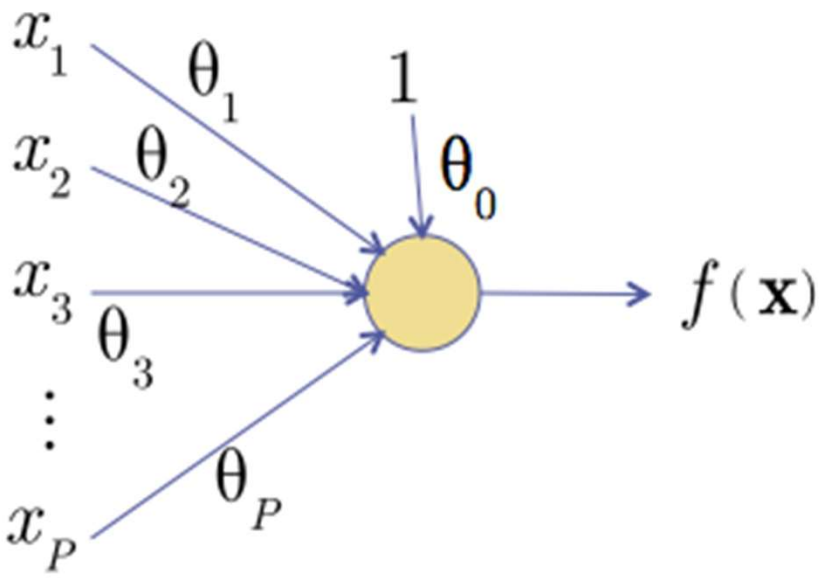
Sources:

1. https://home.cse.ust.hk/~dekai/4211/lectures/neural_nets/Lecture14
2. Lecture slides for Chapter 6 of *Deep Learning*, www.deeplearningbook.org, Ian Goodfellow

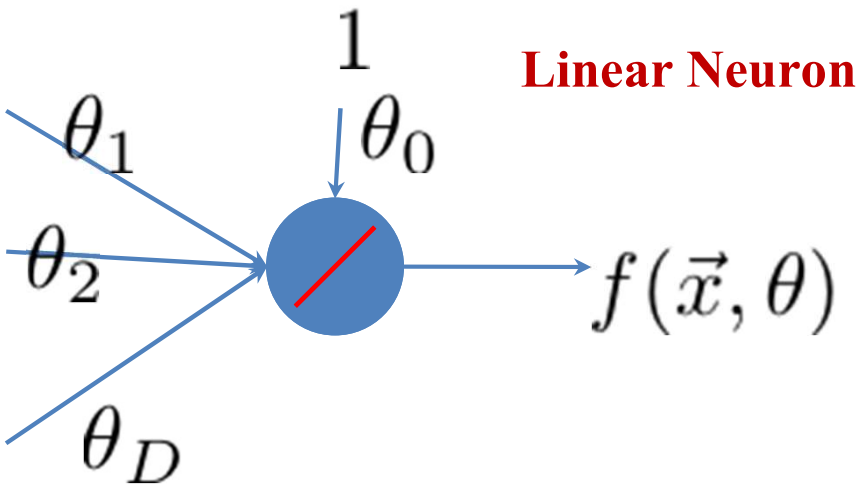
The Neuron Metaphor

- Neurons [1]
  - accept information from multiple inputs
  - transmit information to other neurons
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node

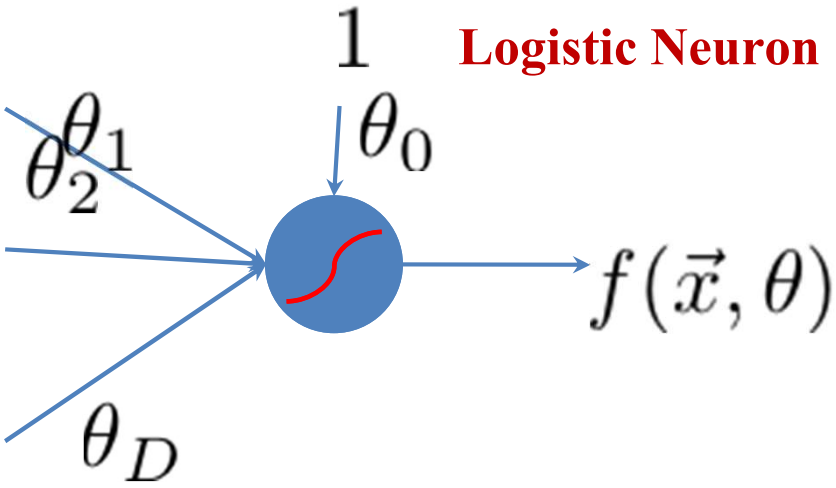




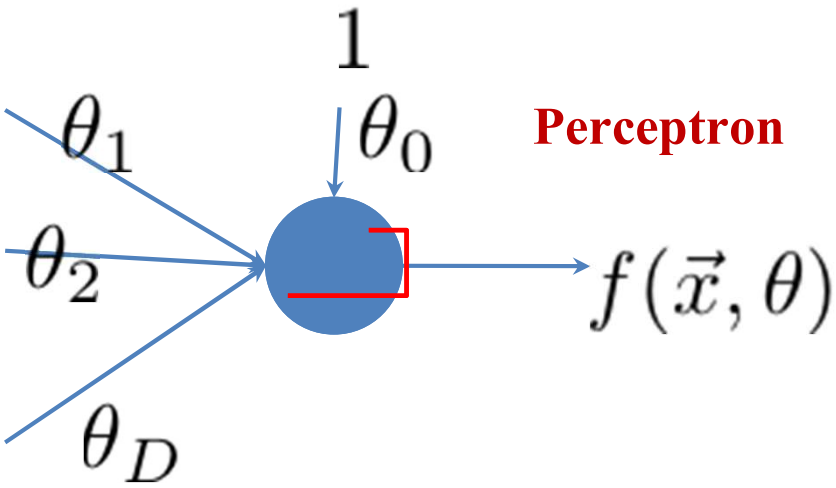
Types of Neurons



Linear Neuron



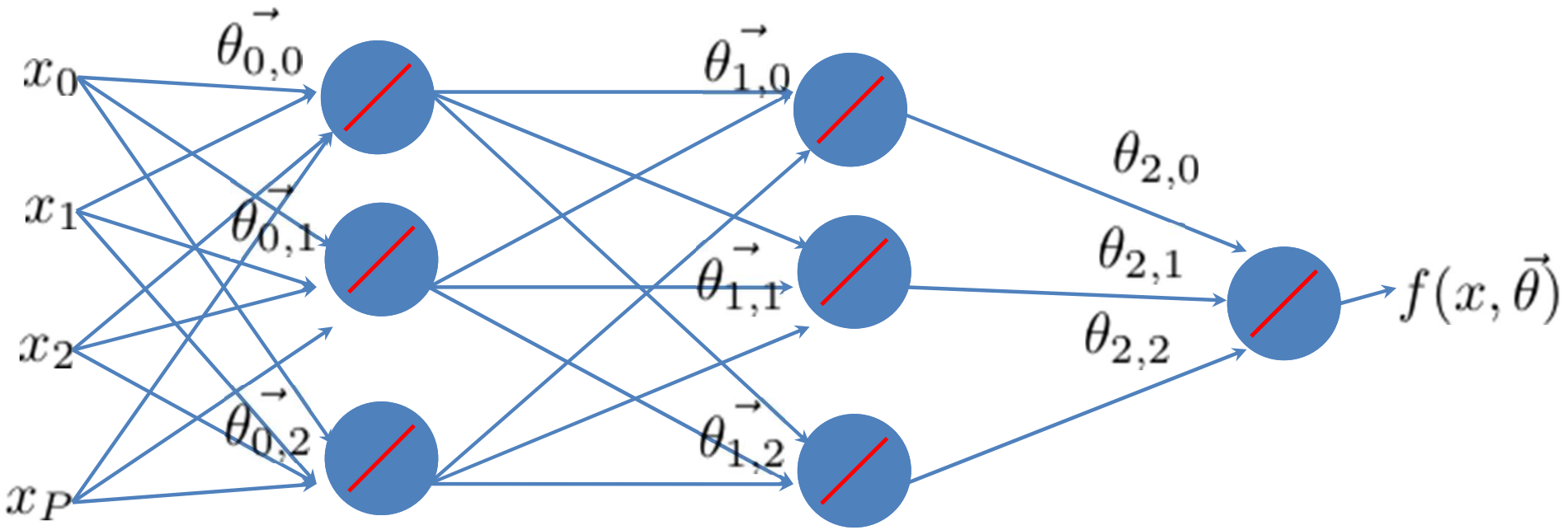
Logistic Neuron



Perceptron

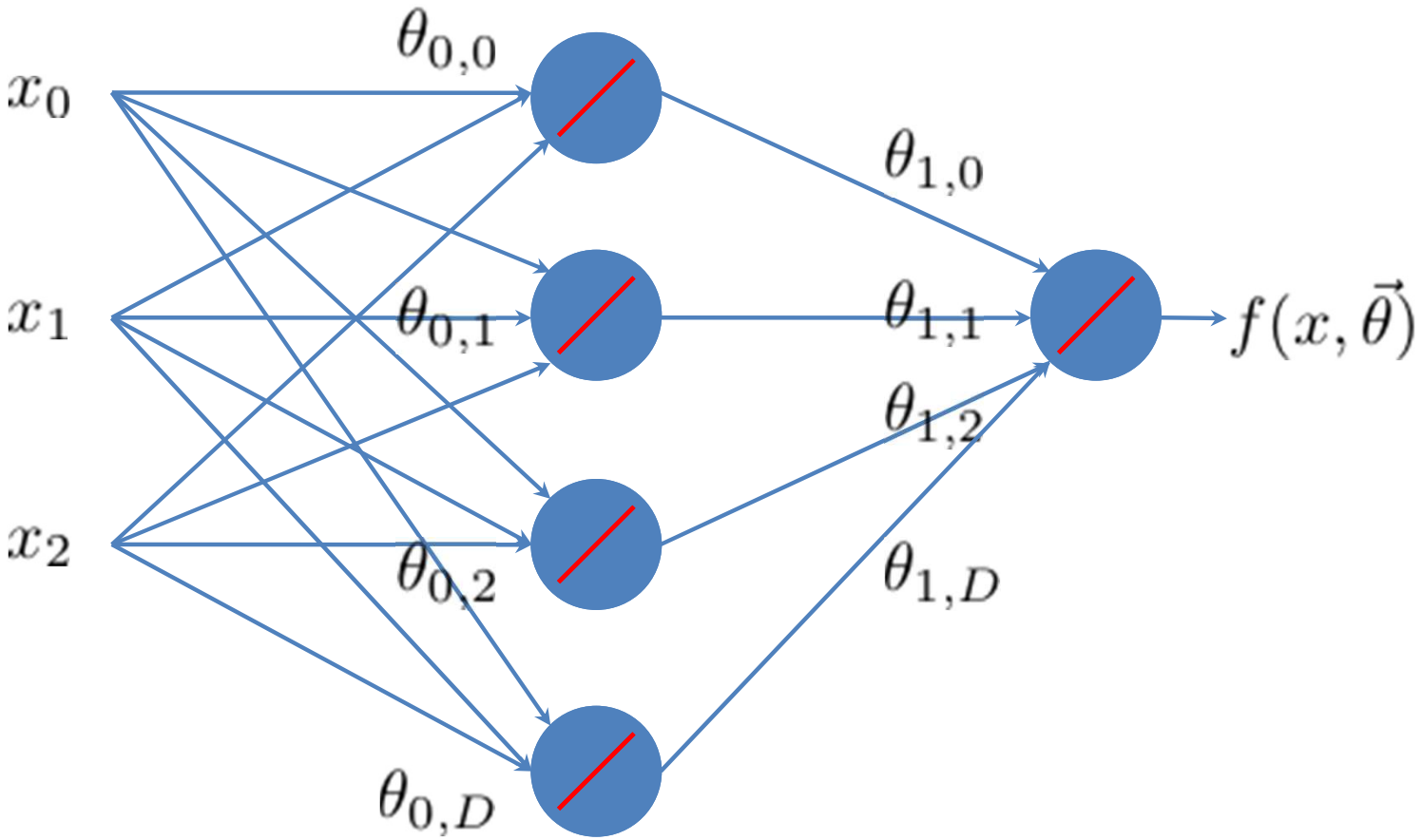
Multilayer Networks

- Cascade Neurons together
- The output from one layer is the input to the next
- Each Layer has its own sets of weights



Linear Neural Networks

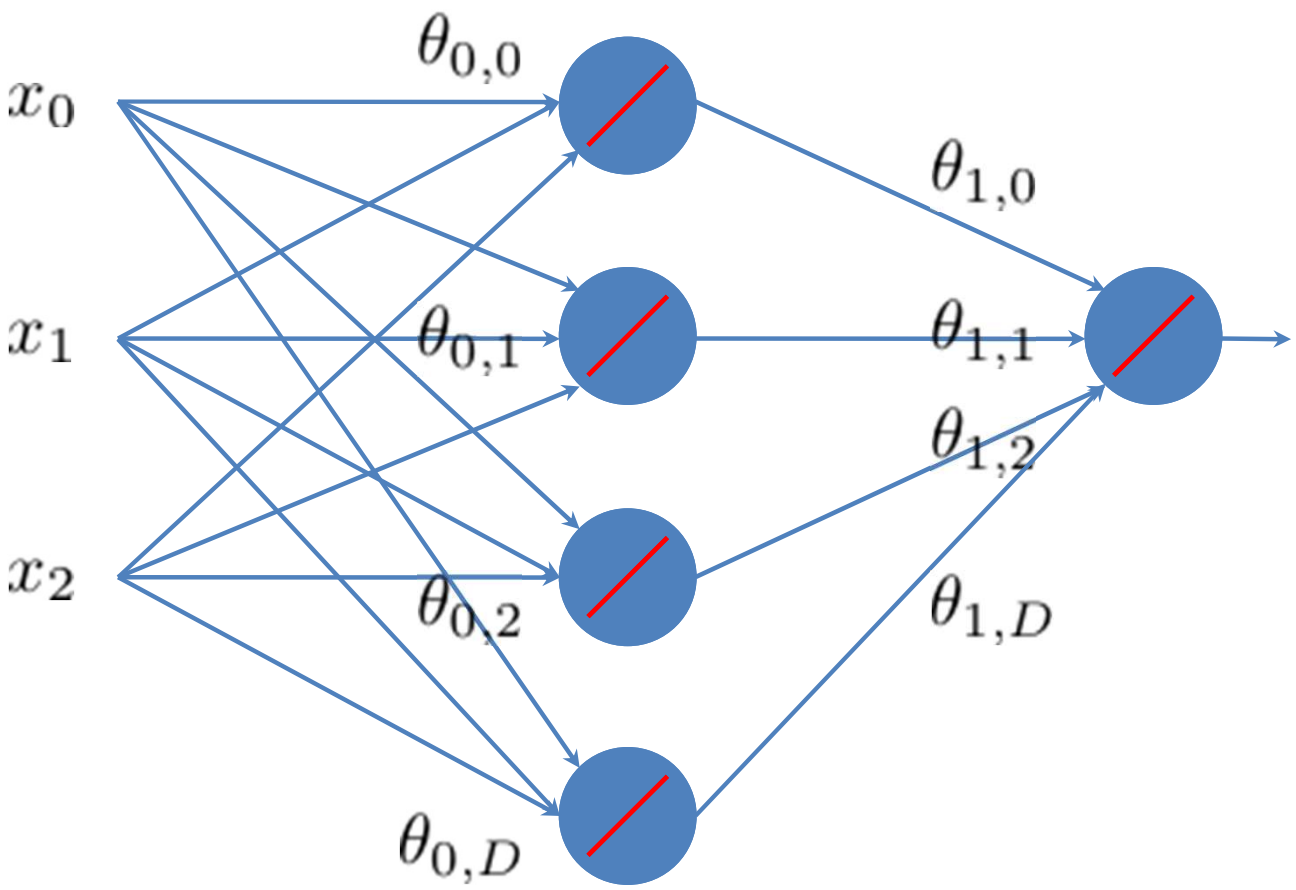
- Arrange **linear neurons** in a multilayer network



Linear Neural Networks

- The product of two linear transformations is itself a linear transformation

$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} \sum_{n=0}^{N-1} \theta_{0,i,n} x_n$$

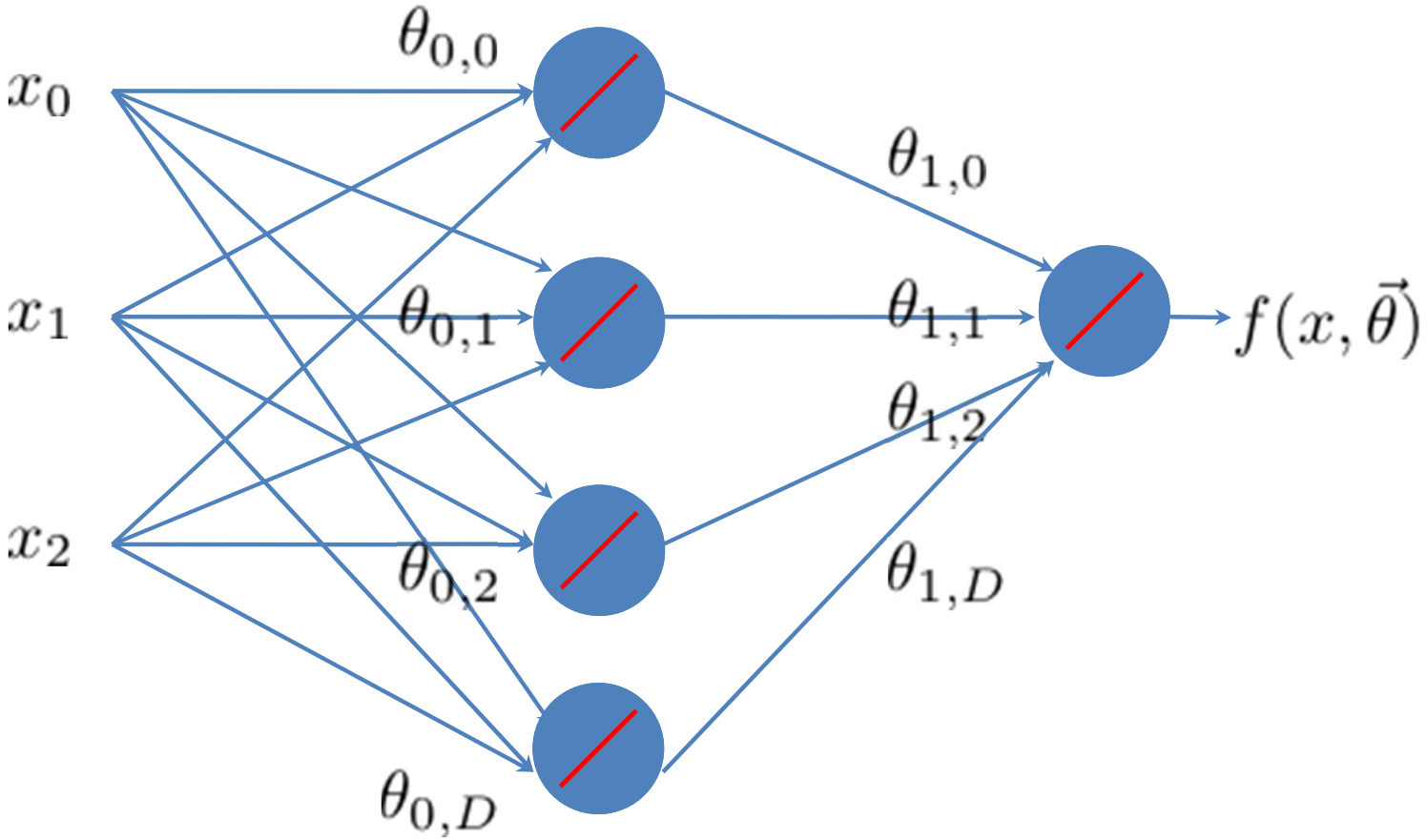


$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} [\theta_{0,i}^T \vec{x}]$$

$$f(x,\vec{\theta}) = \sum_{i=0}^{D} [\hat{\theta}_i^T \vec{x}]$$

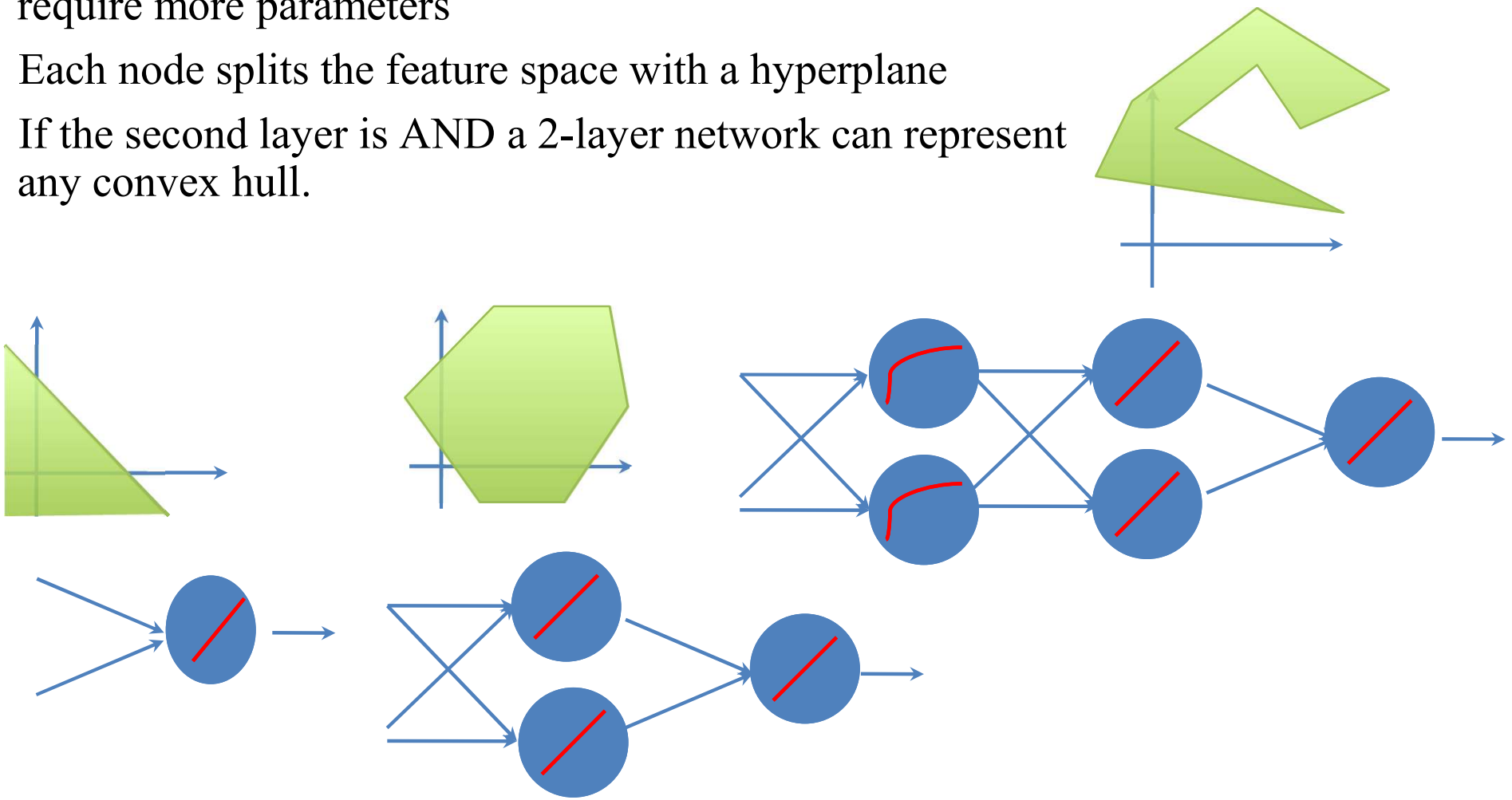
Neural Networks

- Introduce non-linearities to the network
  - Non-linearities allow a network to identify complex regions in space



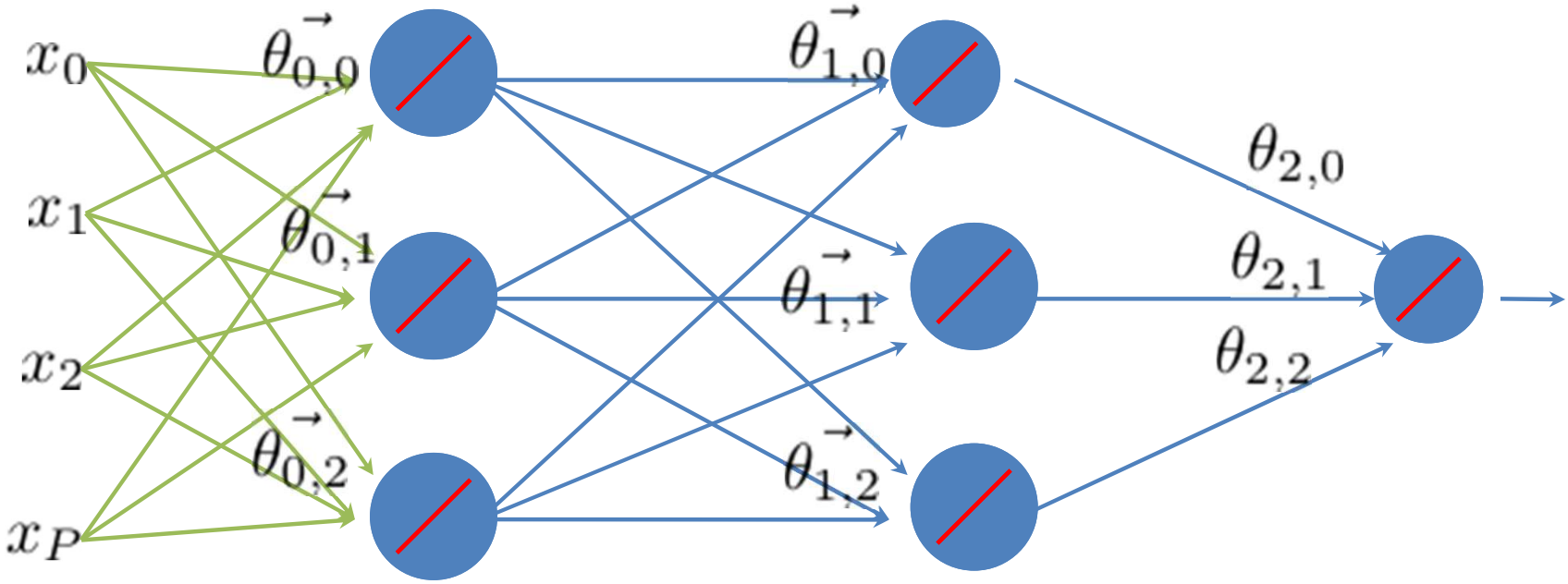
Linear Separability

- 1-layer cannot handle XOR
- More layers can handle more complicated spaces – but require more parameters
- Each node splits the feature space with a hyperplane
- If the second layer is AND a 2-layer network can represent any convex hull.



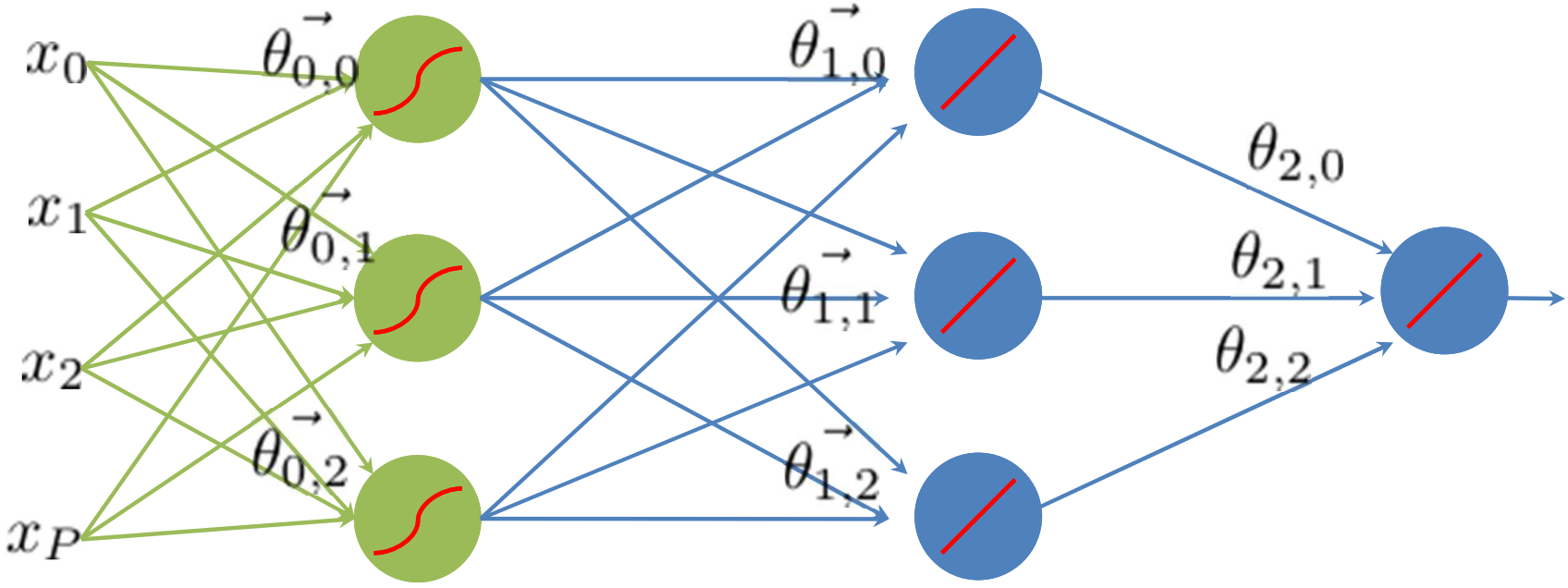
Feed-Forward Networks

- Predictions are fed forward through the network to classify



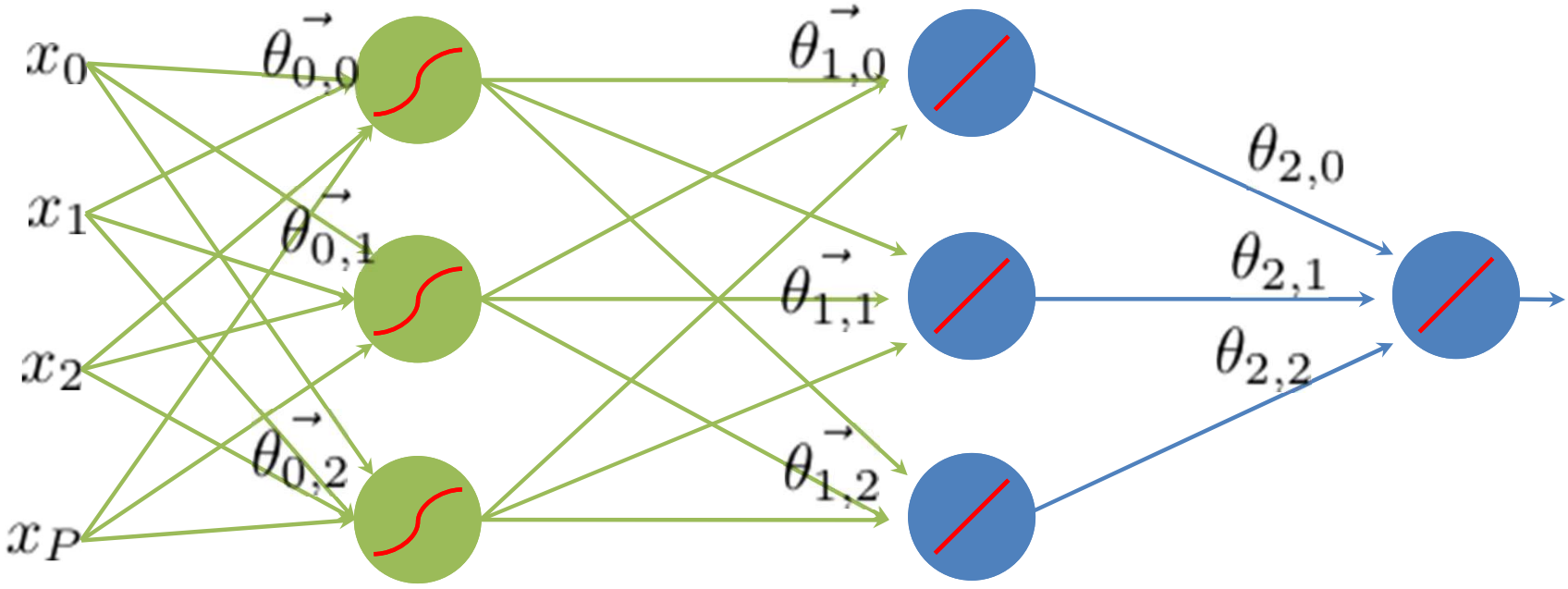
Feed-Forward Networks

- Predictions are fed forward through the network to classify



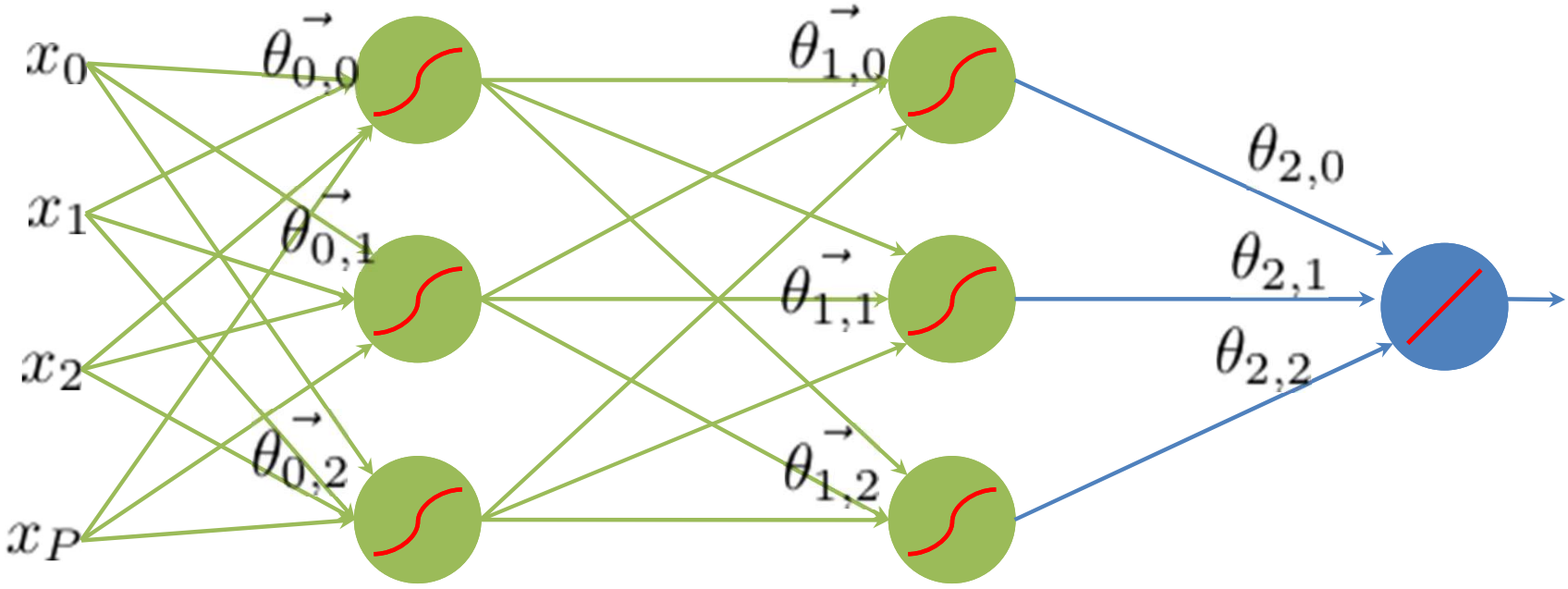
Feed-Forward Networks

- Predictions are fed forward through the network to classify



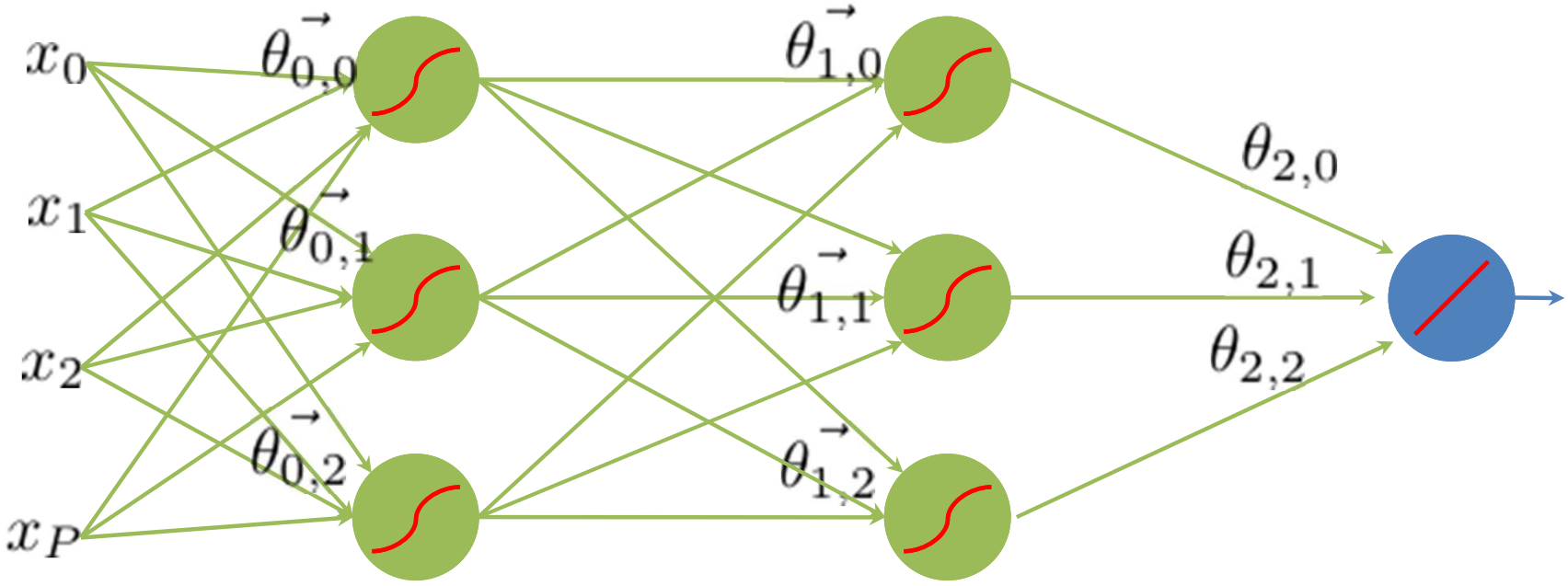
Feed-Forward Networks

- Predictions are fed forward through the network to classify



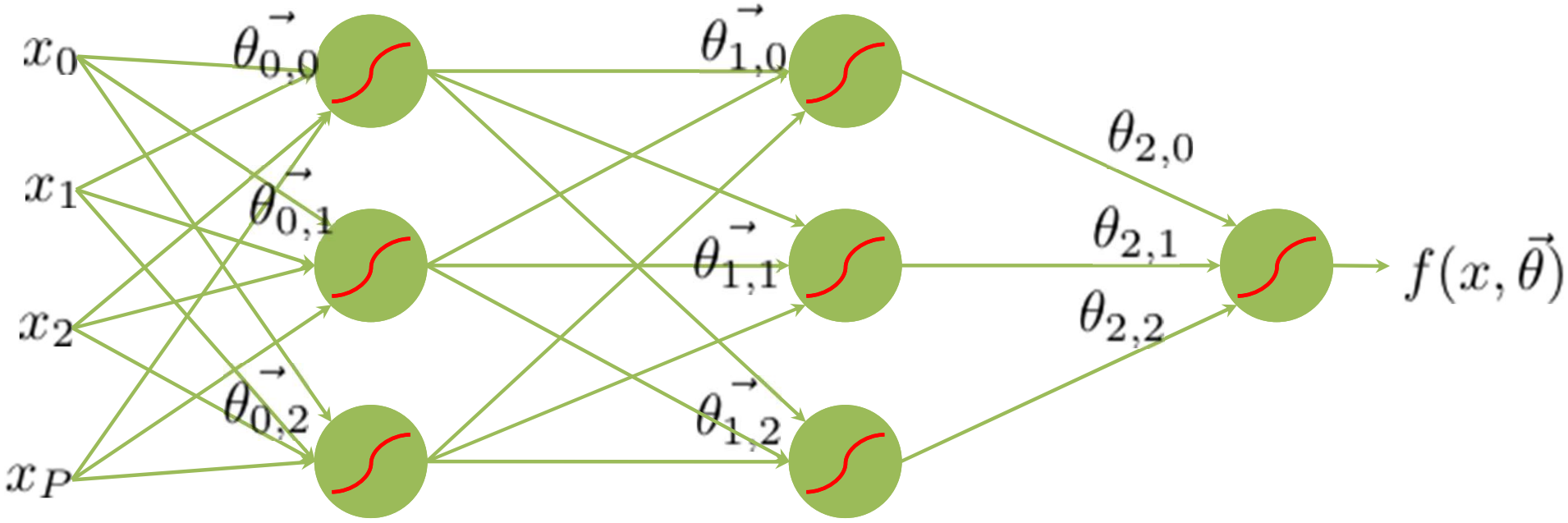
Feed-Forward Networks

- Predictions are fed forward through the network to classify

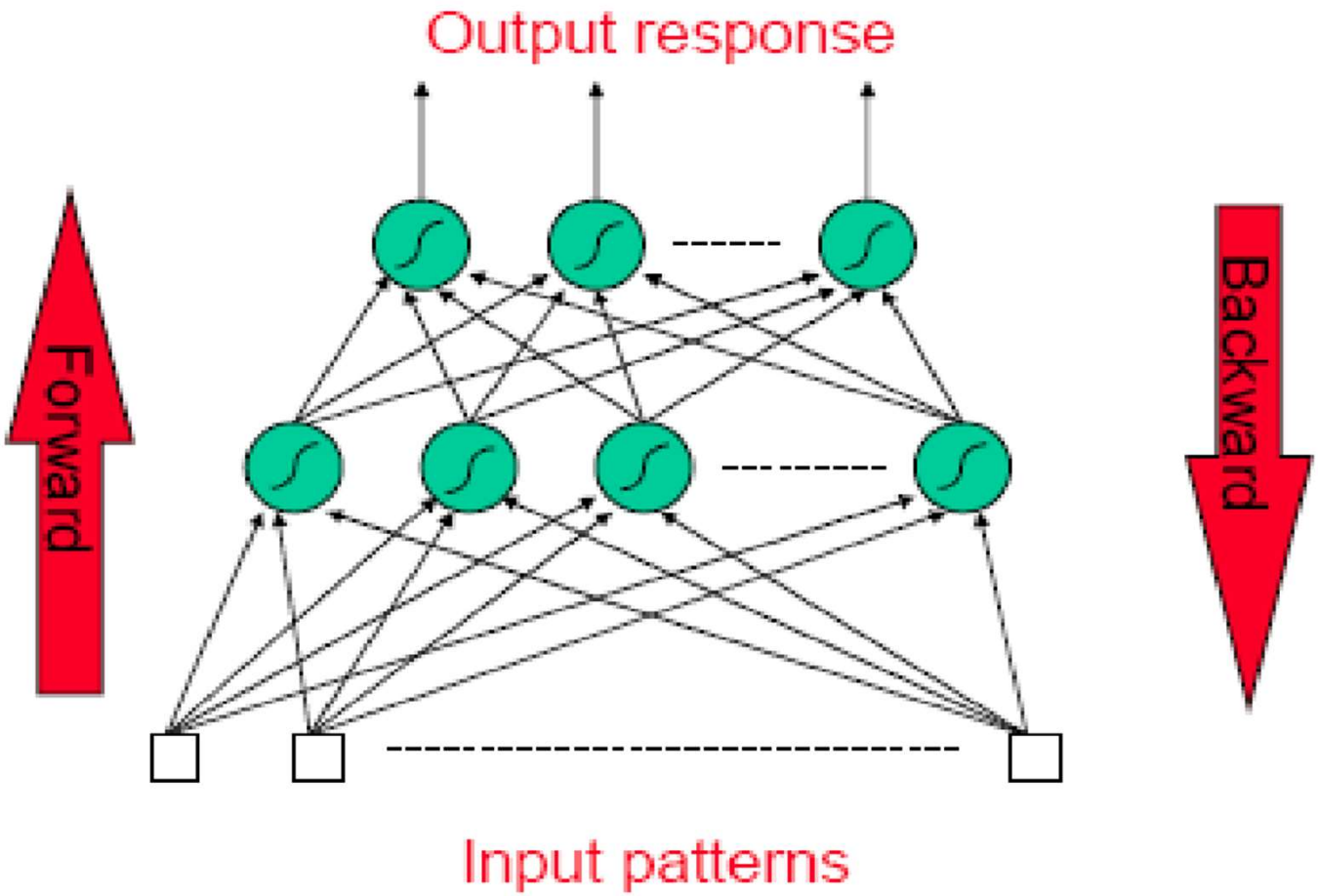


Feed-Forward Networks

- Predictions are fed forward through the network to classify



Forward Activity- Backward Error



Backward propagation

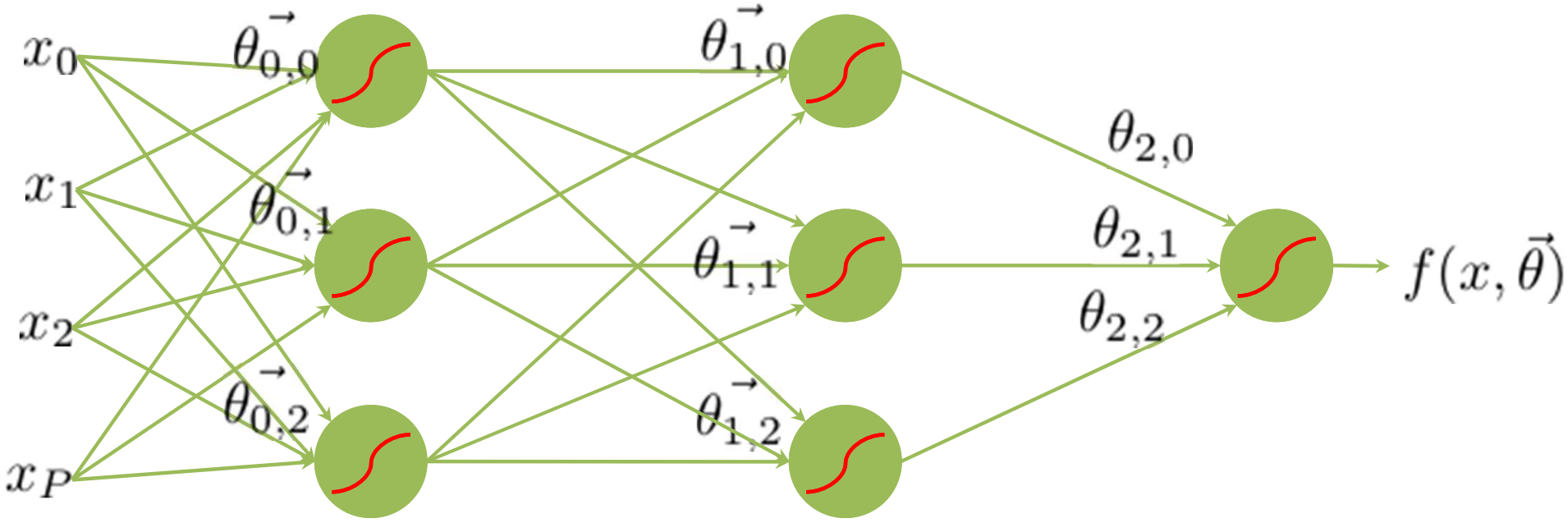
- **Backpropagation has two phases:**

Forward pass phase: computes 'functional signal', feed forward
Propagation of input pattern signals through network

Backward pass phase: computes 'error signal', *propagates* the error *backwards* through network starting at output units
(where the error is the difference between actual and desired output values)

Error Backpropagation

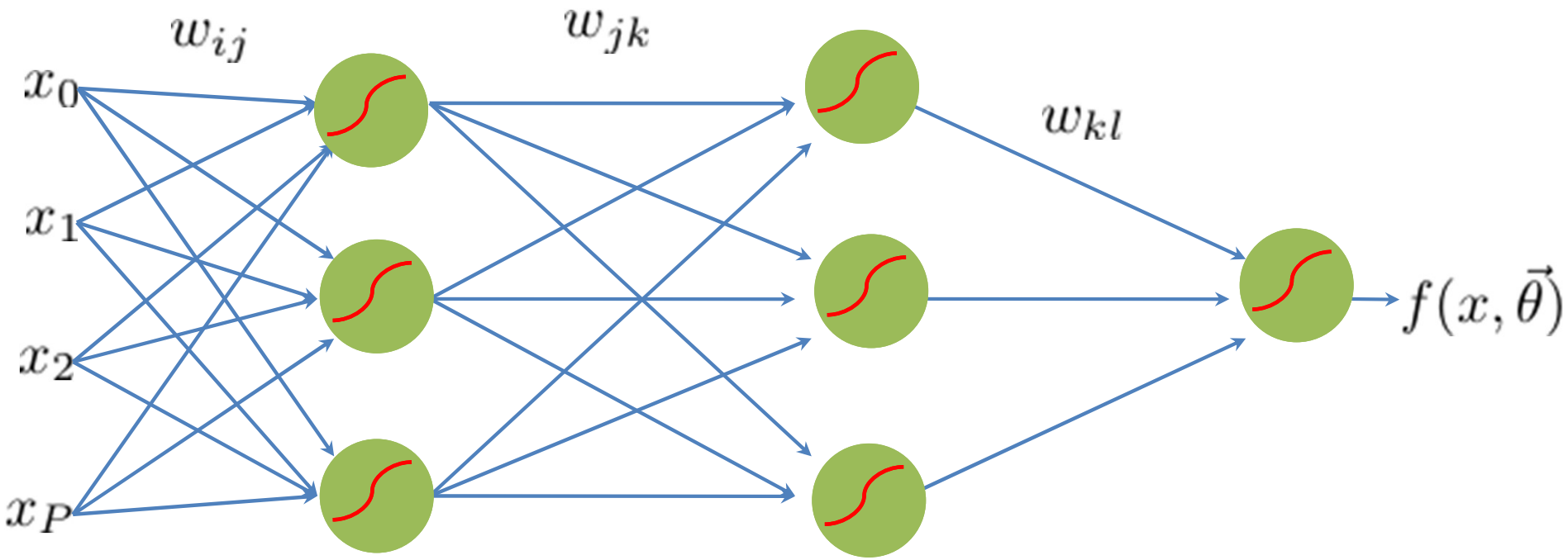
- Apply gradient descent on the whole network
- Training will proceed from the last layer to the first



Error Backpropagation

- Introduce variables over the neural network

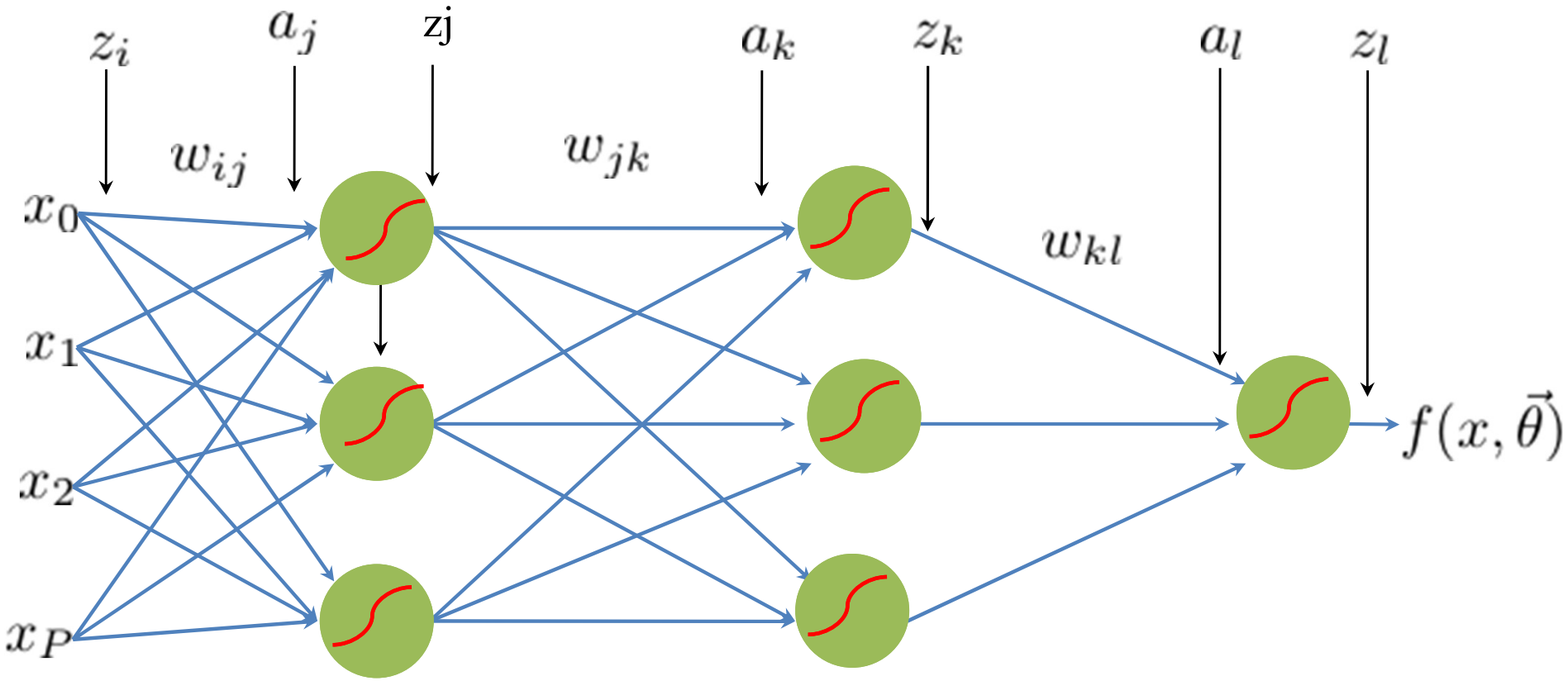
$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

- Introduce variables over the neural network
  - Distinguish the input and output of each node

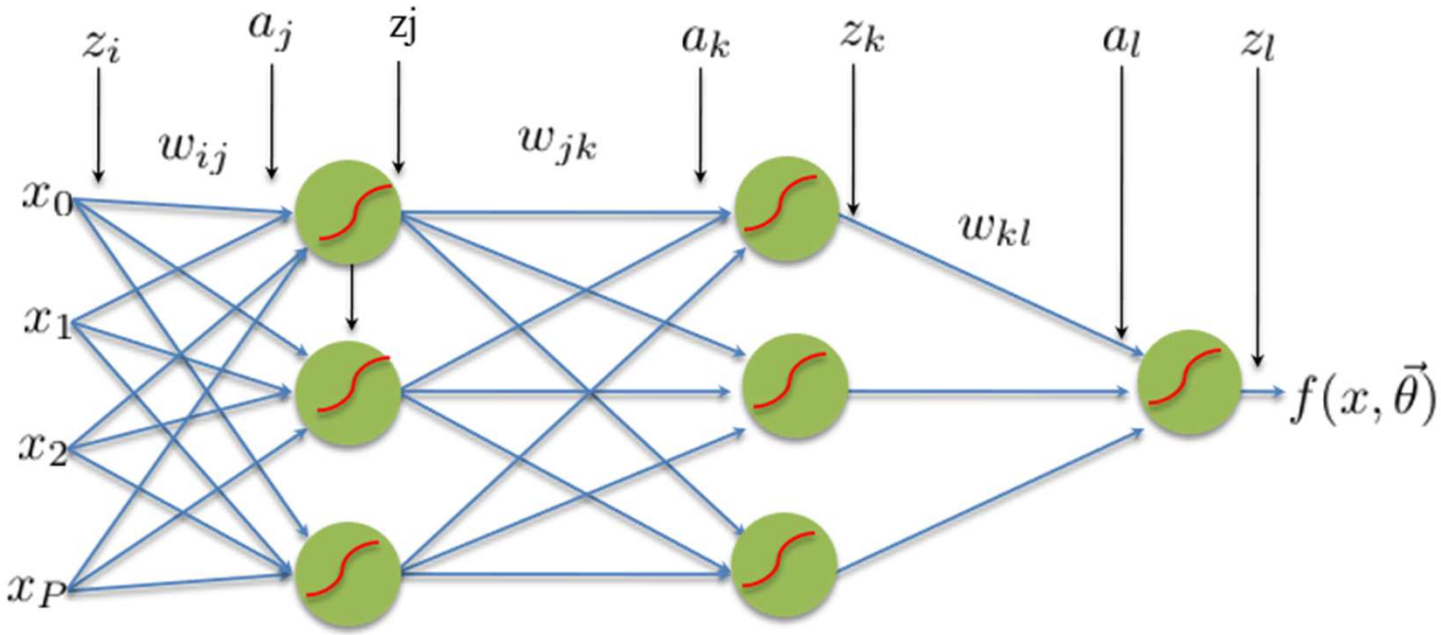


Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

$$a_j = \sum_i w_{ij} z_i \qquad a_k = \sum_j w_{jk} z_j \qquad a_l = \sum_k w_{kl} z_k$$

$$z_j = g(a_j) \qquad z_k = g(a_k) \qquad z_l = g(a_l)$$



Error Backpropagation

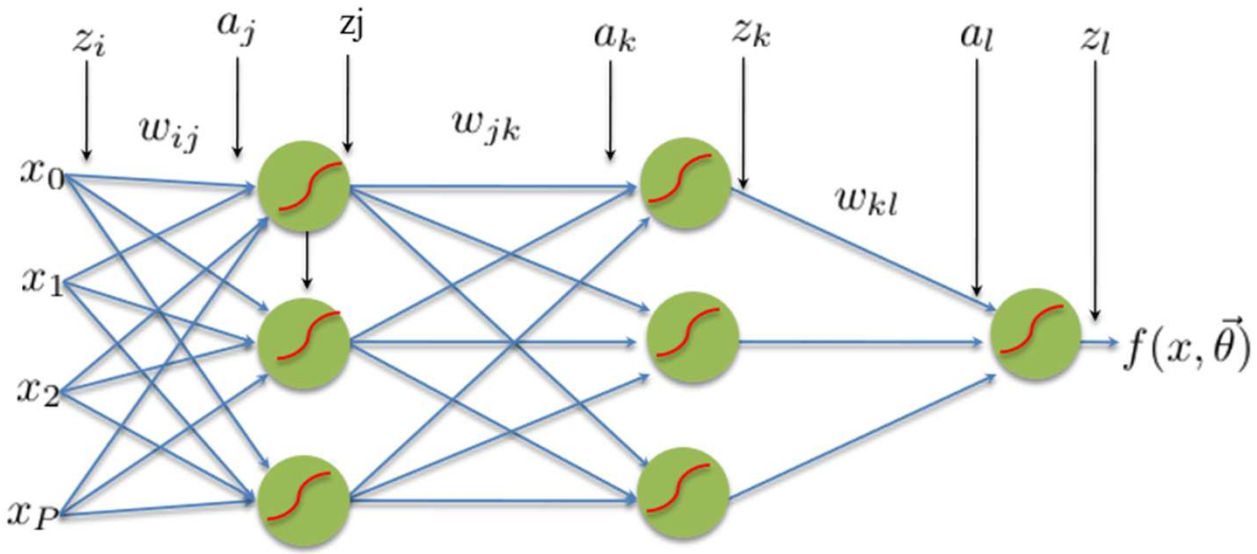
$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

Training: Take the gradient of the last component and iterate backwards

$$a_j = \sum_i w_{ij} z_i \qquad z_j = g(a_j)$$

$$a_k = \sum_j w_{jk} z_j \qquad z_k = g(a_k)$$

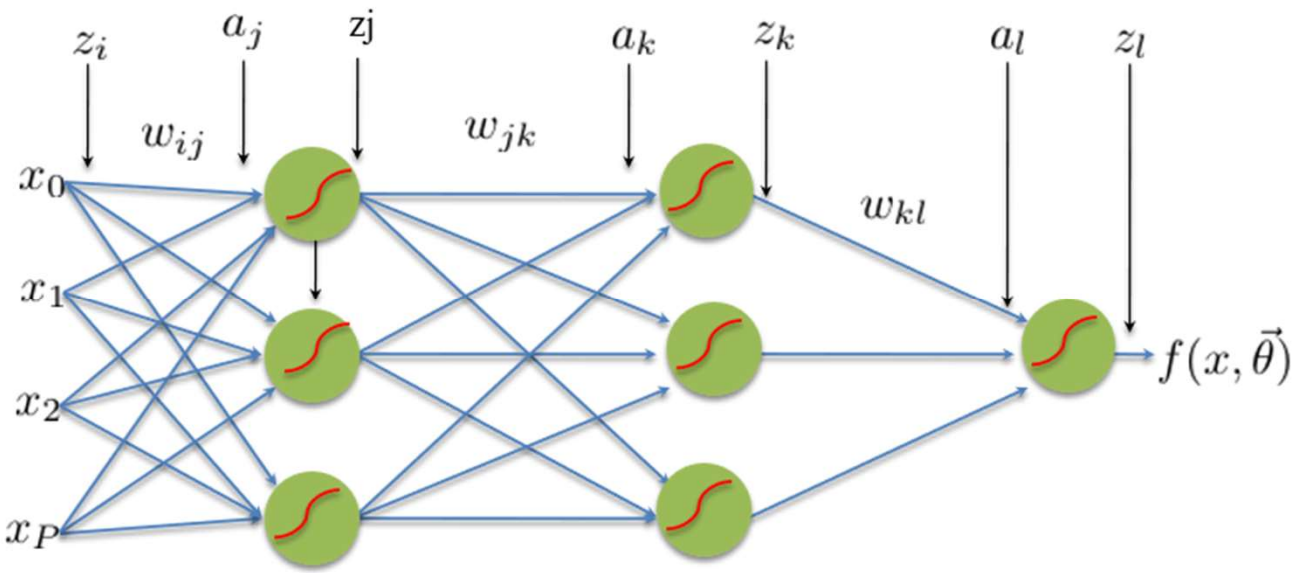
$$a_l = \sum_k w_{kl} z_k \qquad z_l = g(a_l)$$



Error Backpropagation

$$
\begin{aligned}
R(\theta) &= \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n)) \\
&= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - f(x_n)\right)^2 \\
&= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_{n,i}\right)\right)\right)\right)^2
\end{aligned}
$$

Empirical Risk Function



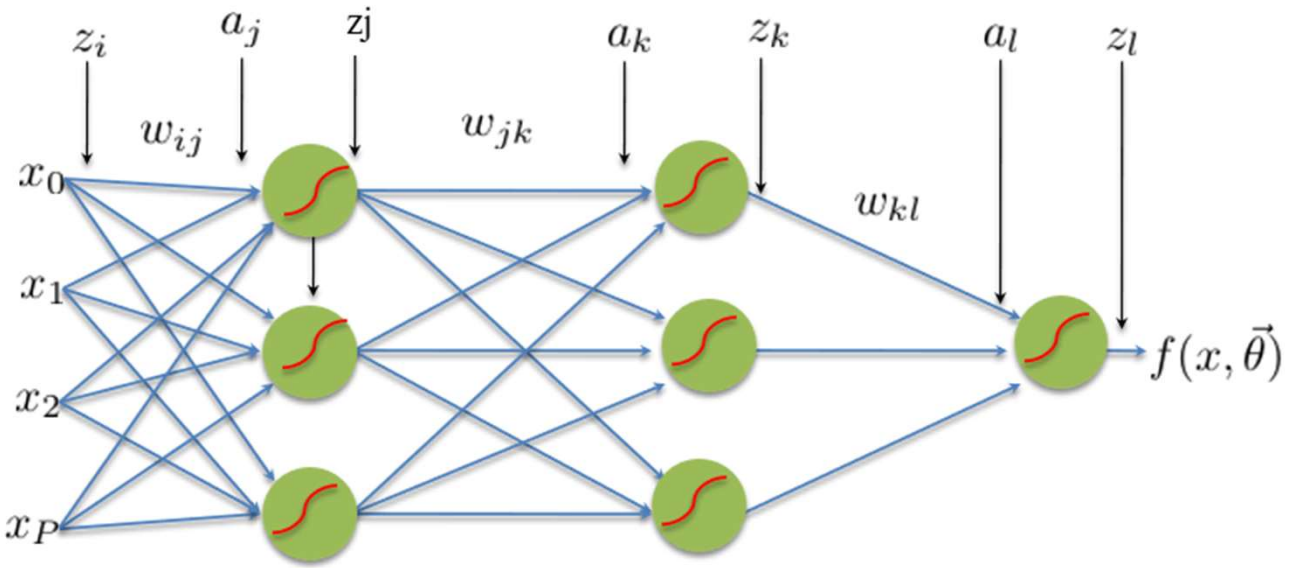
Error Backpropagation

Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule



Error Backpropagation

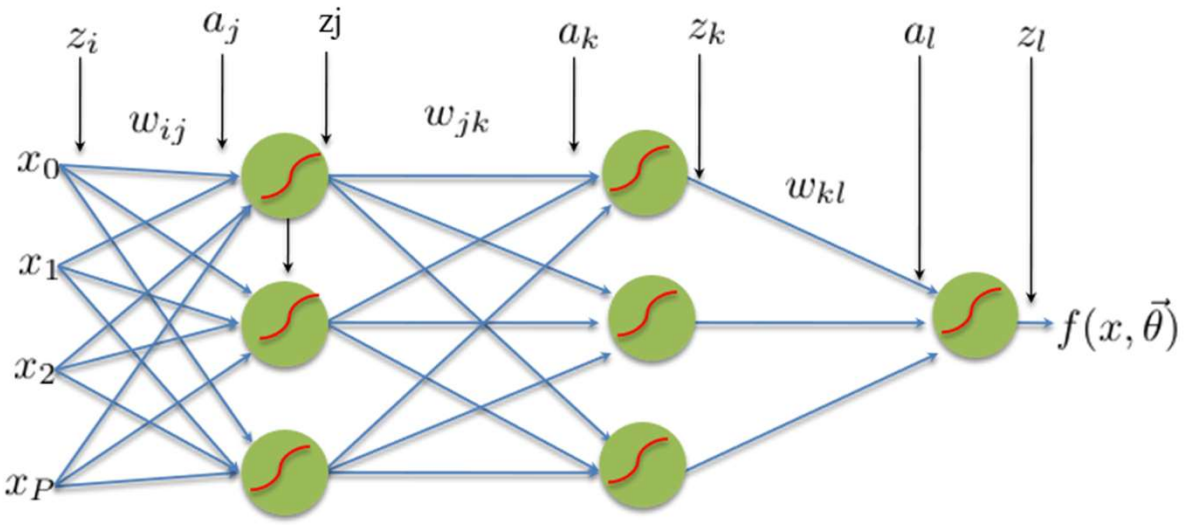
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$



Error Backpropagation

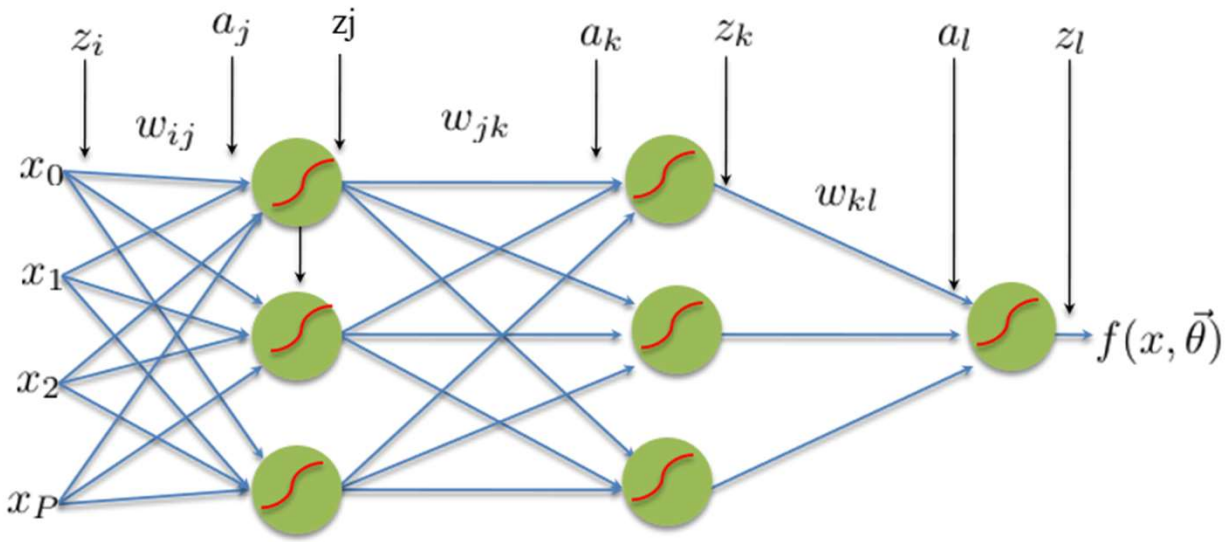
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right]$$



Error Backpropagation

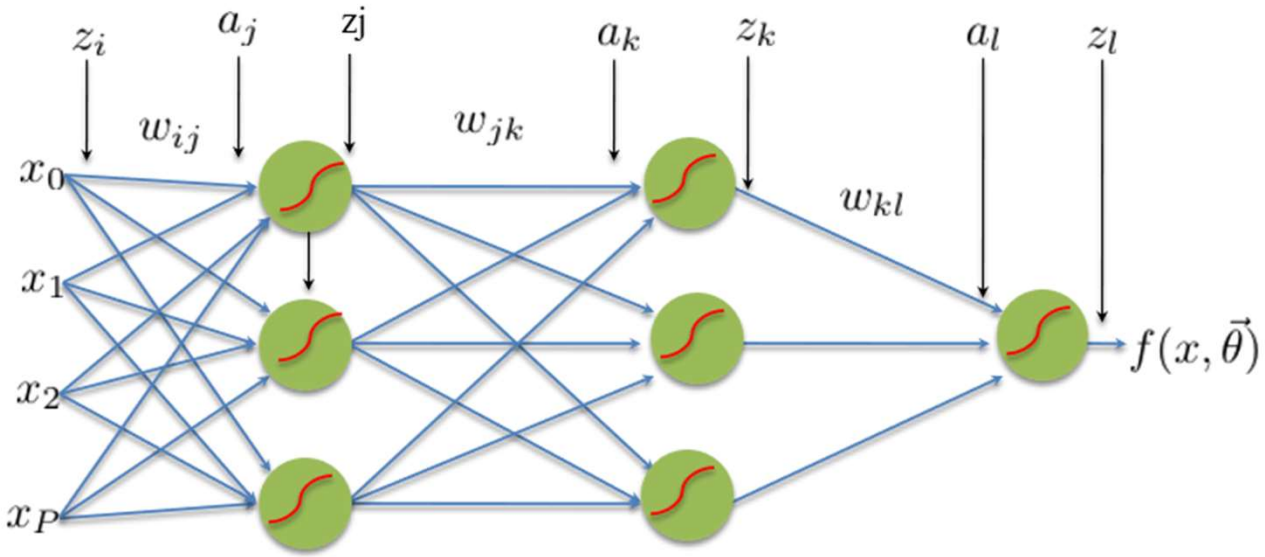
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$



Error Backpropagation

Optimize last layer weights w_{kl}

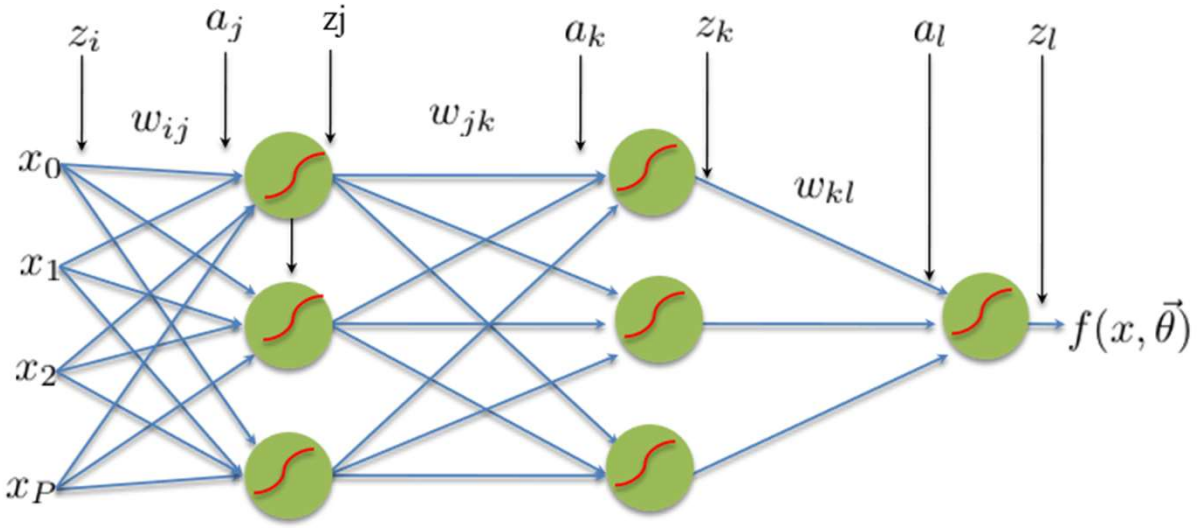
$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$

$$= \frac{1}{N}\sum_n \delta_{l,n} n z_{k,n}$$

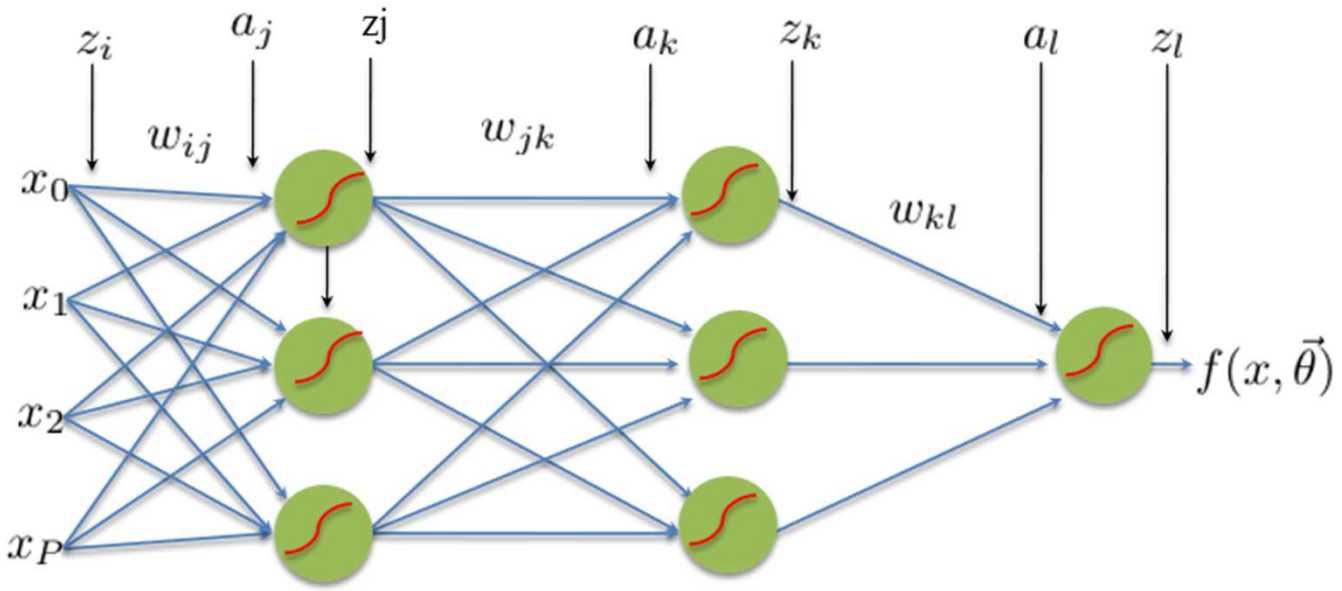


Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N} \sum_n \left[\frac{\partial L_n}{\partial a_{k,n}} \right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}} \right]$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \delta_{l,n} z_{k,n}$$



Error Backpropagation

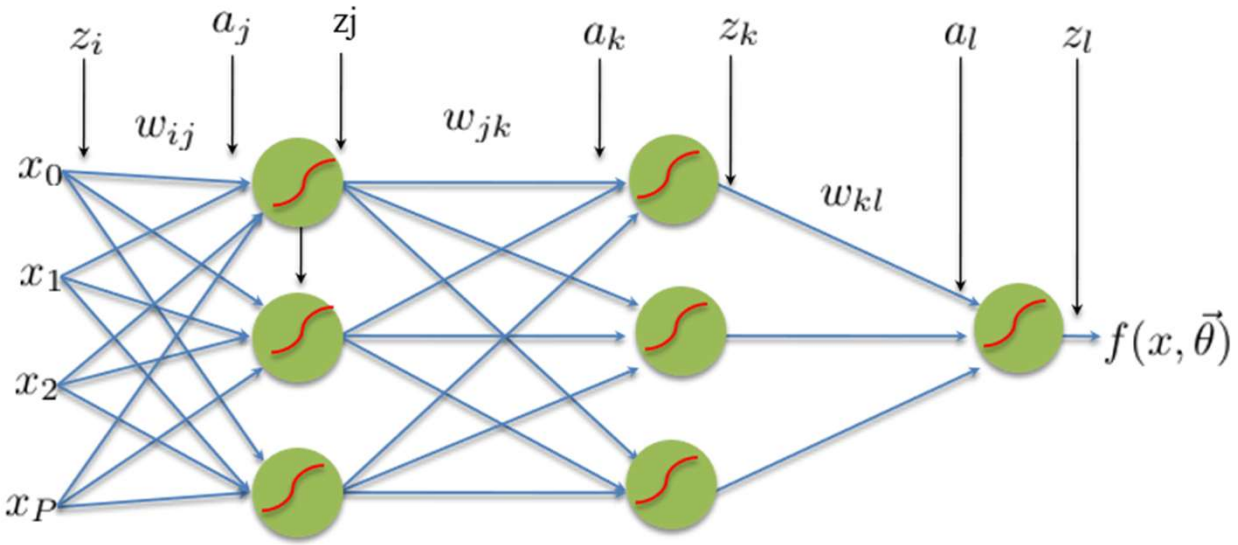
Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] [z_{j,n}]$$



Error Backpropagation

Optimize last hidden weights w_{jk}

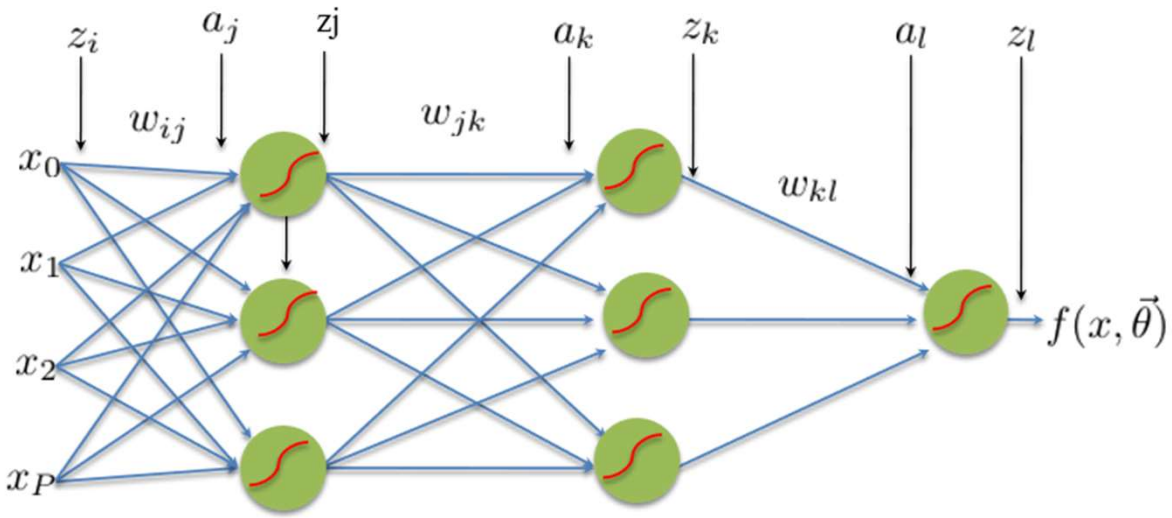
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] [z_{j,n}]$$

$$a_l = \sum_k w_{kl} g(a_k)$$



Error Backpropagation

Optimize last hidden weights w_{jk}

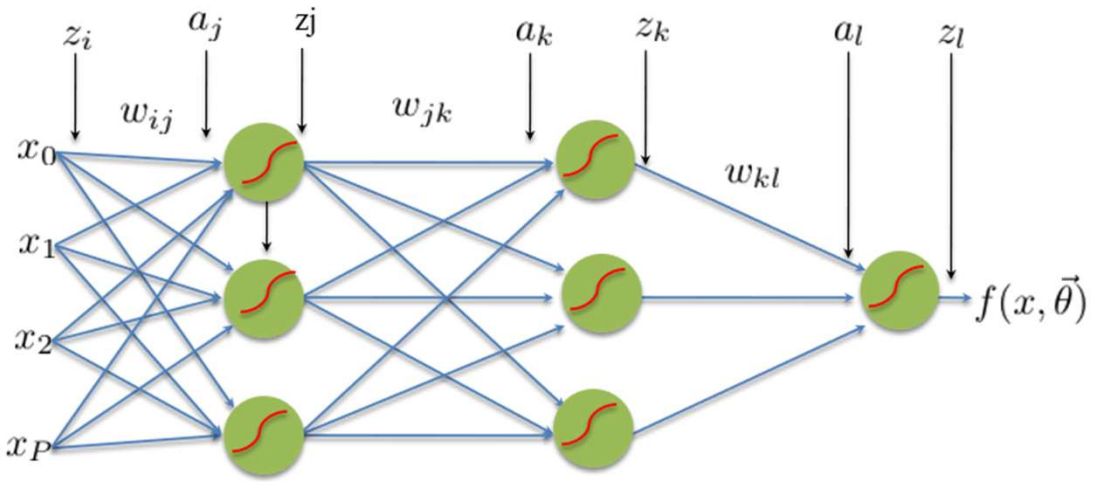
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}}\frac{\partial a_{l,n}}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l w_{kl} g'(a_{k,n})\right][z_{j,n}] = \frac{1}{N}\sum_n [\delta_{k,n}][z_{j,n}]$$

$$a_l = \sum_k w_{kl} g(a_k)$$



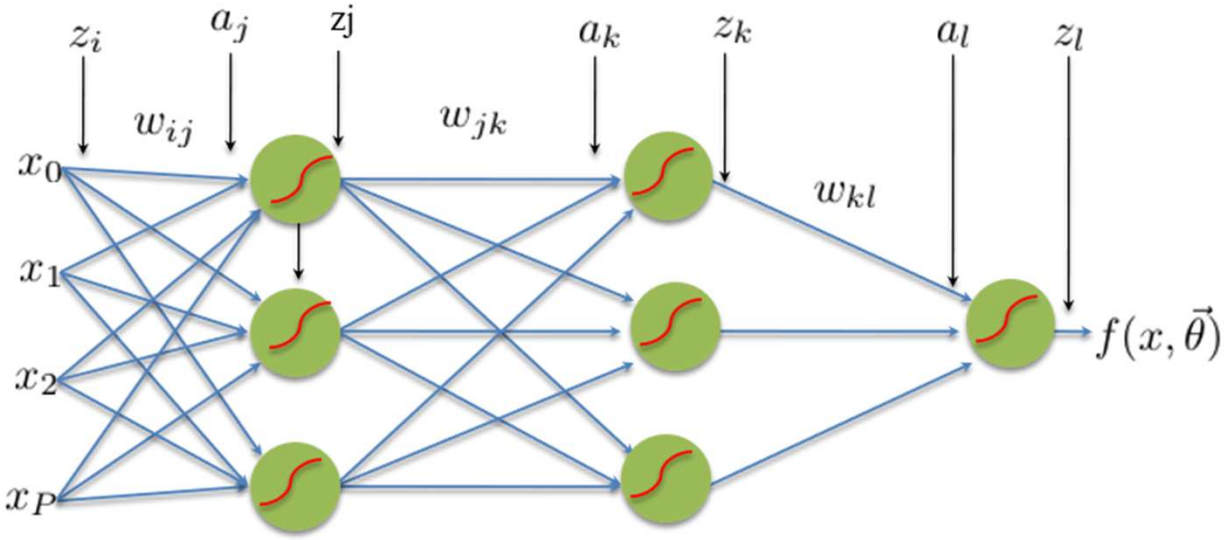
Error Backpropagation

Repeat for all previous layers

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n} w_{kl} g'(a_{k,n})\right] z_{j,n} = \frac{1}{N}\sum_n \delta_{k,n} z_{j,n}$$

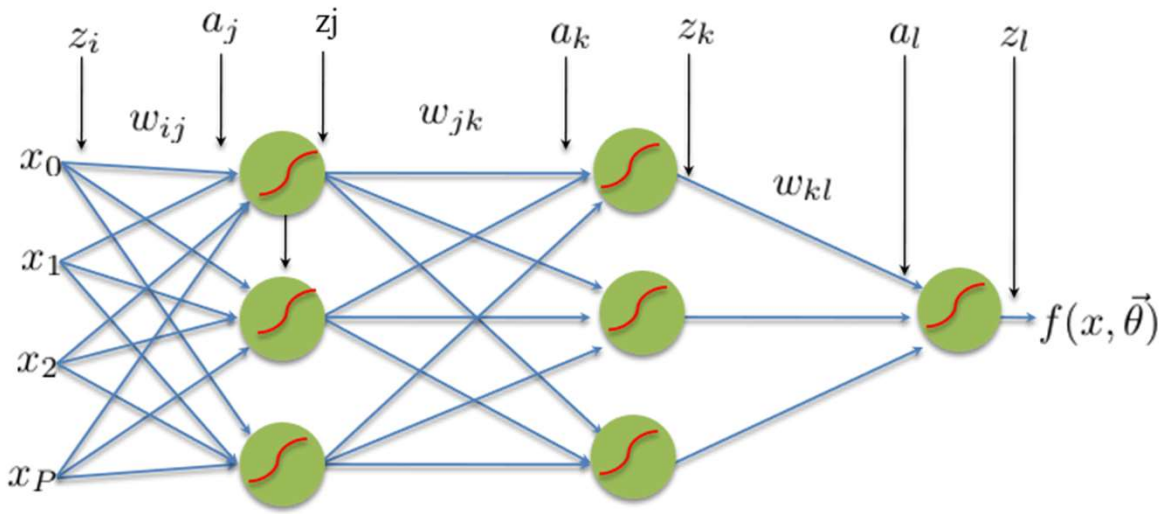
$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{j,n}}\right]\left[\frac{\partial a_{j,n}}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \delta_{k,n} w_{jk} g'(a_{j,n})\right] z_{i,n} = \frac{1}{N}\sum_n \delta_{j,n} z_{i,n}$$



Error Backpropagation

Now that we have well defined gradients for each parameter, update using Gradient Descent

$$
\begin{aligned}
w_{ij}^{t+1} &= w_{ij}^{t} - \eta \frac{\partial R}{w_{ij}} \\
w_{jk}^{t+1} &= w_{jk}^{t} - \eta \frac{\partial R}{w_{kl}} \\
w_{kl}^{t+1} &= w_{kl}^{t} - \eta \frac{\partial R}{w_{kl}}
\end{aligned}
$$



Error Back-propagation

- Error backpropagation unravels the multivariate chain rule and solves the gradient for each partial component separately.
- The target values for each layer come from the next layer.
- This feeds the errors back along the network.