



Deep Feedforwards Networks

Amir H. Payberah
payberah@kth.se
2021-11-18

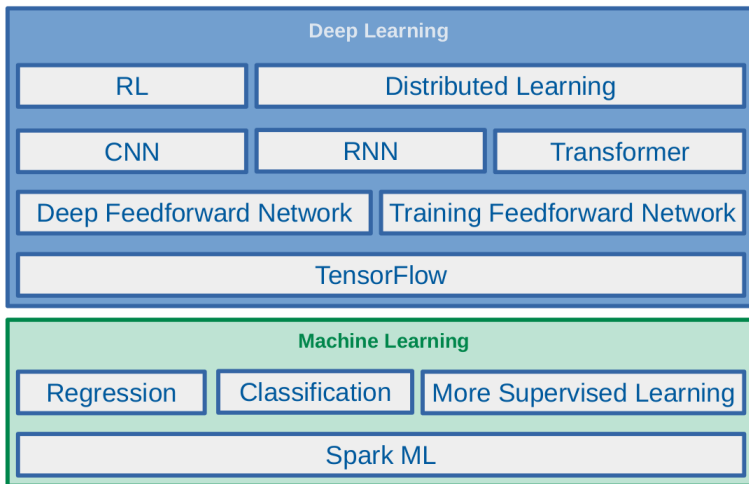




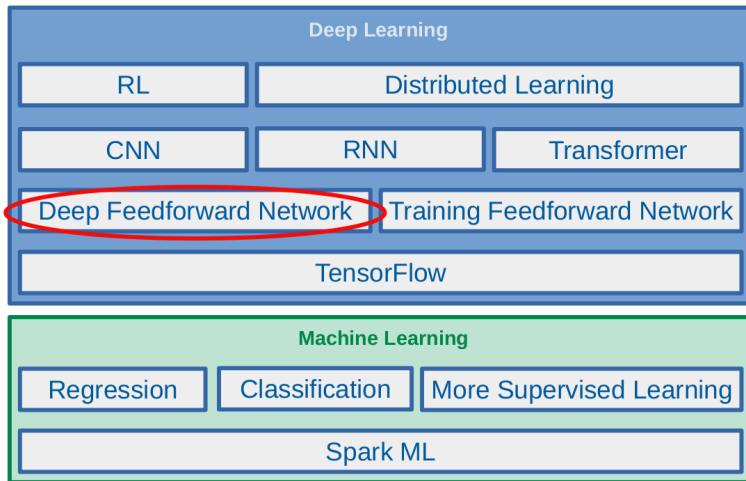
The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

Where Are We?

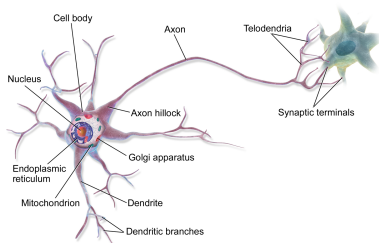


Where Are We?



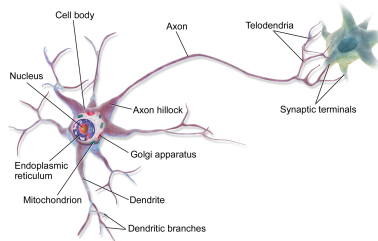
Biological Neurons (1/2)

- Brain architecture has inspired artificial neural networks.



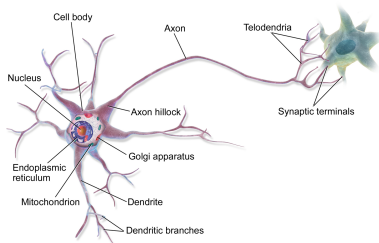
Biological Neurons (1/2)

- ▶ Brain architecture has inspired artificial neural networks.
- ▶ A biological neuron is composed of
 - Cell body, many dendrites (branching extensions), one axon (long extension), synapses



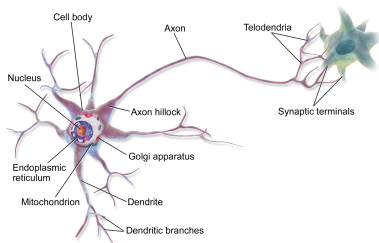
Biological Neurons (1/2)

- ▶ Brain architecture has inspired artificial neural networks.
- ▶ A biological neuron is composed of
 - Cell body, many dendrites (branching extensions), one axon (long extension), synapses
- ▶ Biological neurons receive signals from other neurons via these synapses.



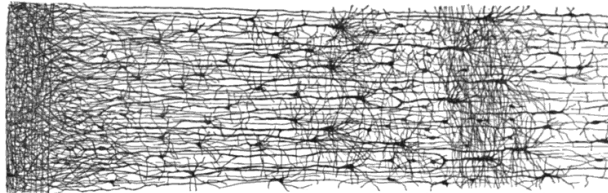
Biological Neurons (1/2)

- ▶ Brain architecture has inspired artificial neural networks.
- ▶ A biological neuron is composed of
 - Cell body, many dendrites (branching extensions), one axon (long extension), synapses
- ▶ Biological neurons receive signals from other neurons via these synapses.
- ▶ When a neuron receives a sufficient number of signals within a few milliseconds, it fires its own signals.



Biological Neurons (2/2)

- ▶ Biological neurons are organized in a vast **network of billions of neurons**.
- ▶ Each neuron typically is **connected** to **thousands of other neurons**.



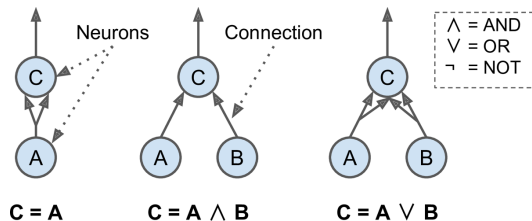


A Simple Artificial Neural Network

- ▶ One or more binary inputs and one binary output
- ▶ Activates its output when more than a certain number of its inputs are active.

A Simple Artificial Neural Network

- ▶ One or more binary inputs and one binary output
- ▶ Activates its output when more than a certain number of its inputs are active.



[A. Geron, O'Reilly Media, 2017]



The Linear Threshold Unit (LTU)

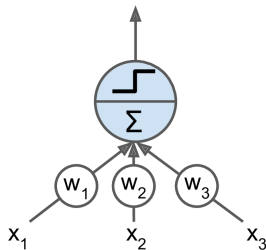
- ▶ Inputs of a LTU are numbers (not binary).

The Linear Threshold Unit (LTU)

- ▶ Inputs of a LTU are **numbers** (not **binary**).
- ▶ Each **input connection** is associated with a **weight**.
- ▶ Computes a **weighted sum of its inputs** and applies a **step function** to that **sum**.

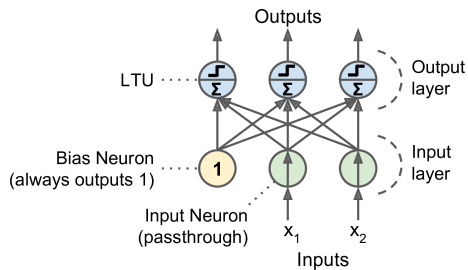
▶ $z = w_1x_1 + w_2x_2 + \dots + w_nx_n = w^T x$

▶ $\hat{y} = \text{step}(z) = \text{step}(w^T x)$



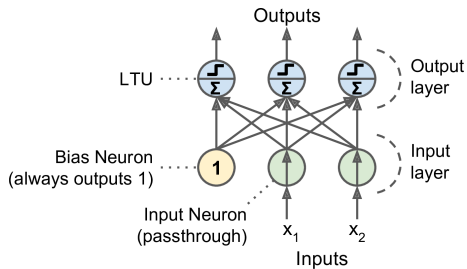
The Perceptron

- The **perceptron** is a **single layer** of LTUs.



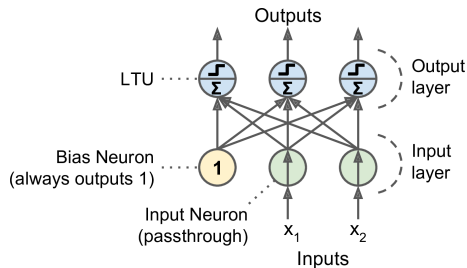
The Perceptron

- ▶ The **perceptron** is a **single layer** of LTUs.
- ▶ The **input neurons** output whatever **input they are fed**.



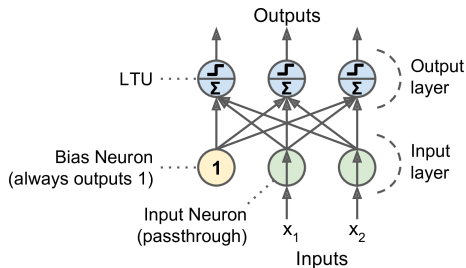
The Perceptron

- ▶ The **perceptron** is a **single layer** of LTUs.
- ▶ The **input neurons** output whatever **input they are fed**.
- ▶ A **bias neuron**, which just **outputs 1 all the time**.



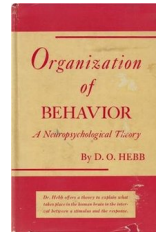
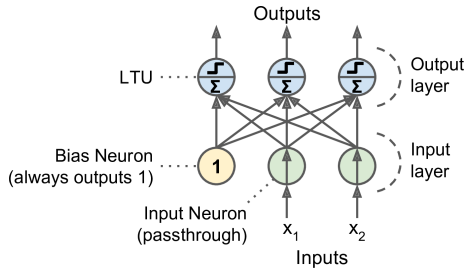
The Perceptron

- ▶ The **perceptron** is a **single layer** of LTUs.
- ▶ The **input neurons** output whatever **input they are fed**.
- ▶ A **bias neuron**, which just **outputs 1 all the time**.
- ▶ If we use **logistic function (sigmoid)** instead of a **step** function, it computes a **continuous** output.



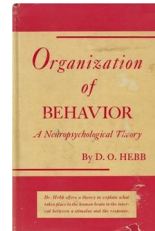
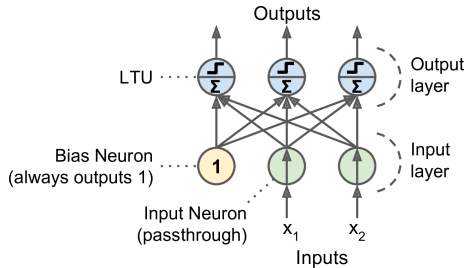
How is a Perceptron Trained? (1/2)

- The **Perceptron training algorithm** is inspired by **Hebb's rule**.



How is a Perceptron Trained? (1/2)

- ▶ The **Perceptron training algorithm** is inspired by **Hebb's rule**.
- ▶ When a **biological neuron** often **triggers another neuron**, the **connection** between these two neurons grows **stronger**.





How is a Perceptron Trained? (2/2)

- ▶ Feed one training instance x to each neuron j at a time and make its prediction \hat{y} .
- ▶ Update the connection weights.

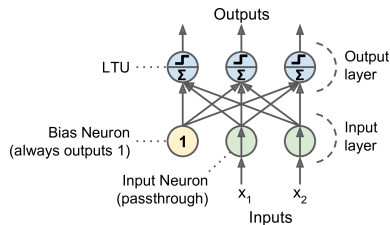
How is a Perceptron Trained? (2/2)

- ▶ Feed one training instance \mathbf{x} to each neuron j at a time and make its prediction \hat{y}_j .
- ▶ Update the connection weights.

$$\hat{y}_j = \sigma(\mathbf{w}_j^T \mathbf{x} + b)$$

$$J(\mathbf{w}_j) = \text{cross_entropy}(y_j, \hat{y}_j)$$

$$\mathbf{w}_{i,j}^{(\text{next})} = \mathbf{w}_{i,j} - \eta \frac{\partial J(\mathbf{w}_j)}{\partial w_i}$$



How is a Perceptron Trained? (2/2)

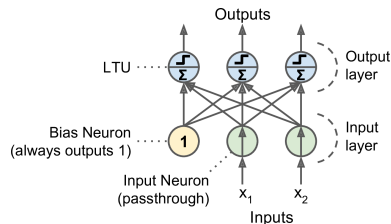
- ▶ Feed one training instance \mathbf{x} to each neuron j at a time and make its prediction \hat{y}_j .
- ▶ Update the connection weights.

$$\hat{y}_j = \sigma(\mathbf{w}_j^T \mathbf{x} + b)$$

$$J(\mathbf{w}_j) = \text{cross_entropy}(y_j, \hat{y}_j)$$

$$\mathbf{w}_{i,j}^{(\text{next})} = \mathbf{w}_{i,j} - \eta \frac{\partial J(\mathbf{w}_j)}{\partial \mathbf{w}_i}$$

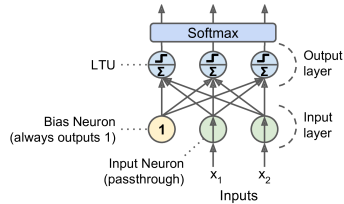
- ▶ $\mathbf{w}_{i,j}$: the weight between neurons i and j .
- ▶ \mathbf{x}_i : the i th input value.
- ▶ \hat{y}_j : the j th predicted output value.
- ▶ y_j : the j th true output value.
- ▶ η : the learning rate.



Perceptron in TensorFlow



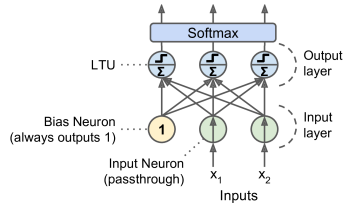
Perceptron in TensorFlow



```
n_neurons = 3
n_features = 2

model = keras.models.Sequential()
model.add(keras.layers.Dense(n_neurons, input_shape=(n_features,), activation="softmax"))
```


Perceptron in TensorFlow



```
n_neurons = 3
n_features = 2
```

```
model = keras.models.Sequential()
model.add(keras.layers.Dense(n_neurons, input_shape=(n_features,), activation="softmax"))
```

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=30)
```

Multi-Layer Perceptron (MLP)



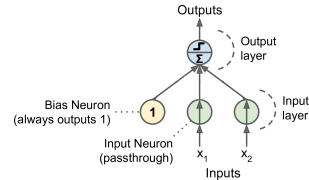
Perceptron Weakness (1/2)

- ▶ Incapable of solving some trivial problems, e.g., XOR classification problem. Why?

Perceptron Weakness (1/2)

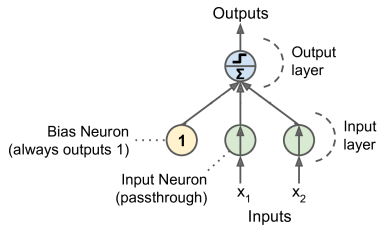
- Incapable of solving some trivial problems, e.g., XOR classification problem. Why?

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Perceptron Weakness (2/2)



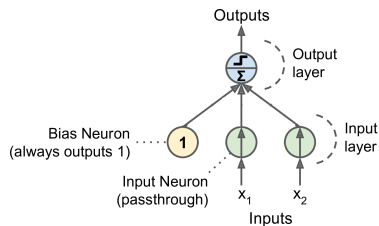
$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$\hat{y} = \text{step}(z), z = w_1 x_1 + w_2 x_2 + b$$

$$J(w) = \frac{1}{4} \sum_{x \in X} (\hat{y}(x) - y(x))^2$$

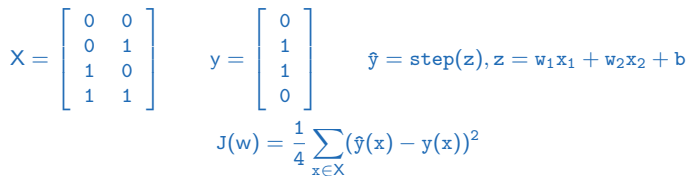
Perceptron Weakness (2/2)



$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \hat{y} = \text{step}(z), z = w_1 x_1 + w_2 x_2 + b$$

$$J(w) = \frac{1}{4} \sum_{x \in X} (\hat{y}(x) - y(x))^2$$

- If we minimize $J(w)$, we obtain $w_1 = 0$, $w_2 = 0$, and $b = \frac{1}{2}$.



- ▶ If we minimize $J(w)$, we obtain $w_1 = 0$, $w_2 = 0$, and $b = \frac{1}{2}$.
- ▶ But, the model outputs 0.5 everywhere.



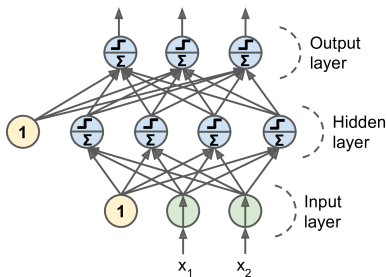
Multi-Layer Perceptron (MLP)

- ▶ The **limitations** of Perceptrons can be eliminated by **stacking multiple Perceptrons**.
- ▶ The resulting network is called a **Multi-Layer Perceptron (MLP)** or **deep feedforward neural network**.

Feedforward Neural Network Architecture

► A **feedforward neural network** is composed of:

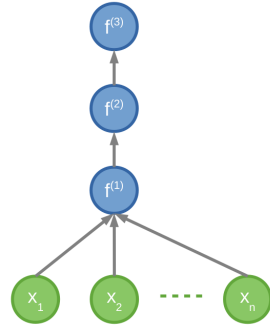
- One **input layer**
- One or more **hidden layers**
- One final **output layer**



-
- The diagram illustrates a fully connected neural network with three layers: Input, Hidden, and Output. The input layer consists of three nodes (one yellow, two green), the hidden layer consists of four nodes (all blue), and the output layer consists of three nodes (all blue). Each node contains a sigmoid function symbol. Arrows show connections between nodes in adjacent layers. Inputs x_1 and x_2 are shown at the bottom, and outputs are shown at the top.

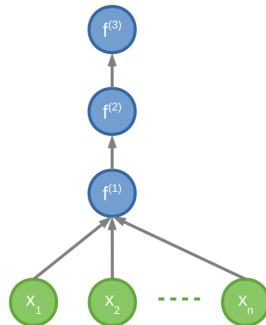
How Does it Work?

- ▶ The **model** is associated with a **directed acyclic graph** describing how the functions are **composed together**.



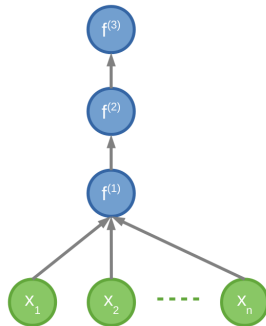
How Does it Work?

- ▶ The **model** is associated with a **directed acyclic graph** describing how the functions are **composed together**.
- ▶ E.g., assume a network with just a **single neuron** in **each layer**.
- ▶ Also assume we have **three functions** $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain: $\hat{y} = f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$



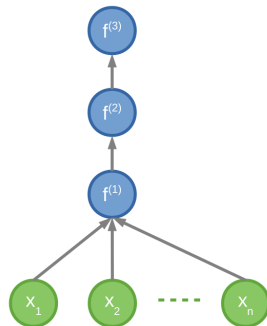
How Does it Work?

- ▶ The **model** is associated with a **directed acyclic graph** describing how the functions are **composed together**.
- ▶ E.g., assume a network with just a **single neuron** in **each layer**.
- ▶ Also assume we have **three functions** $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain: $\hat{y} = f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
- ▶ $f^{(1)}$ is called the **first layer** of the network.
- ▶ $f^{(2)}$ is called the **second layer**, and so on.

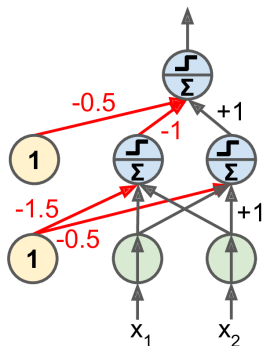


How Does it Work?

- ▶ The **model** is associated with a **directed acyclic graph** describing how the functions are **composed together**.
- ▶ E.g., assume a network with just a **single neuron** in **each layer**.
- ▶ Also assume we have **three functions** $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain: $\hat{y} = f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
- ▶ $f^{(1)}$ is called the **first layer** of the network.
- ▶ $f^{(2)}$ is called the **second layer**, and so on.
- ▶ The **length of the chain** gives the **depth of the model**.

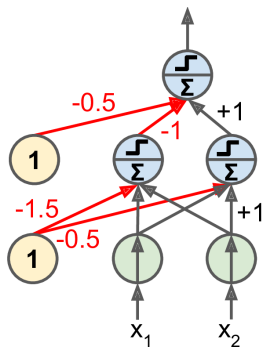


XOR with Feedforward Neural Network (1/3)



$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad W_x = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad b_x = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}$$

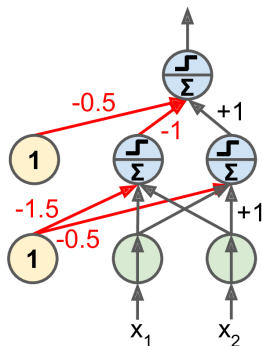
XOR with Feedforward Neural Network (2/3)



$$\text{out}_h = XW_x^T + b_x = \begin{bmatrix} -1.5 & -0.5 \\ -0.5 & 0.5 \\ -0.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix} \quad h = \text{step}(\text{out}_h) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$w_h = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad b_h = -0.5$$

XOR with Feedforward Neural Network (3/3)



$$\text{out} = w_h^T h + b_h = \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{bmatrix} \quad \text{step}(\text{out}) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

How to Learn Model Parameters W ?



Feedforward Neural Network - Cost Function

- We use the **cross-entropy** (minimizing the negative log-likelihood) between the training data \mathbf{y} and the model's predictions $\hat{\mathbf{y}}$ as the **cost function**.

$$\text{cost}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_j y_j \log(\hat{y}_j)$$

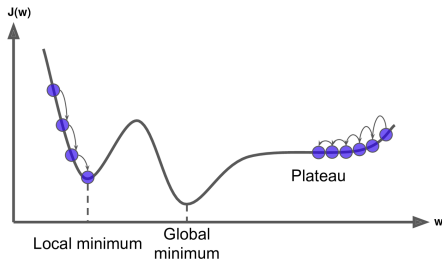


Gradient-Based Learning (1/2)

- ▶ The **most significant difference** between the **linear models** we have seen so far and feedforward neural network?

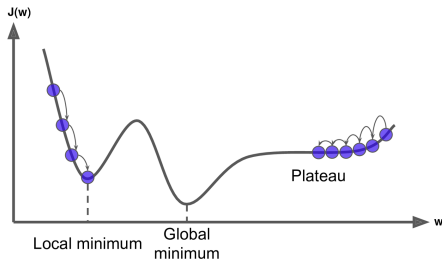
Gradient-Based Learning (1/2)

- ▶ The **most significant difference** between the **linear models** we have seen so far and **feedforward neural network**?
- ▶ The **non-linearity** of a neural network causes its **cost functions** to become **non-convex**.



Gradient-Based Learning (1/2)

- ▶ The **most significant difference** between the **linear models** we have seen so far and **feedforward neural network**?
- ▶ The **non-linearity** of a neural network causes its **cost functions** to become **non-convex**.
- ▶ Linear models, with **convex cost function**, **guarantee** to find **global minimum**.
 - Convex optimization converges starting from **any initial parameters**.





Gradient-Based Learning (2/2)

- ▶ Stochastic gradient descent applied to non-convex cost functions has no such convergence guarantee.



Gradient-Based Learning (2/2)

- ▶ Stochastic gradient descent applied to non-convex cost functions has no such convergence guarantee.
- ▶ It is sensitive to the values of the initial parameters.



Gradient-Based Learning (2/2)

- ▶ Stochastic gradient descent applied to non-convex cost functions has no such convergence guarantee.
- ▶ It is sensitive to the values of the initial parameters.
- ▶ For feedforward neural networks, it is important to initialize all weights to small random values.

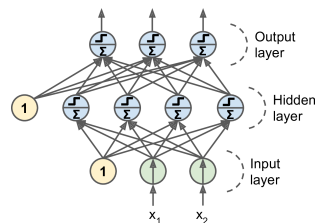


Gradient-Based Learning (2/2)

- ▶ Stochastic gradient descent applied to non-convex cost functions has no such convergence guarantee.
- ▶ It is sensitive to the values of the initial parameters.
- ▶ For feedforward neural networks, it is important to initialize all weights to small random values.
- ▶ The biases may be initialized to zero or to small positive values.

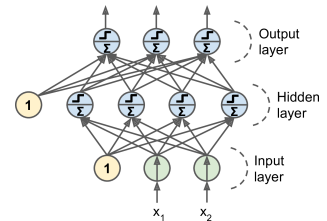
Training Feedforward Neural Networks

- How to **train** a **feedforward neural network**?



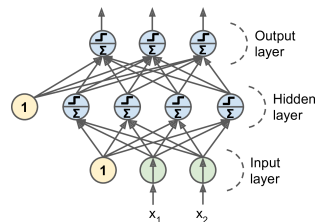
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:



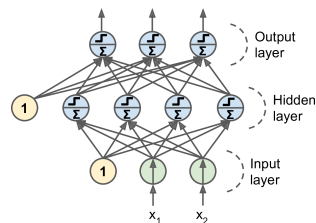
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass**: make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).



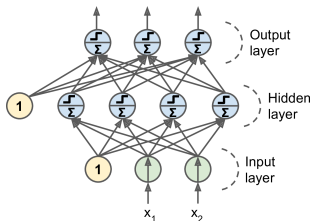
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass**: make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).
 2. Measure the **error** (compute $\text{cost}(\hat{y}^{(i)}, y^{(i)})$).



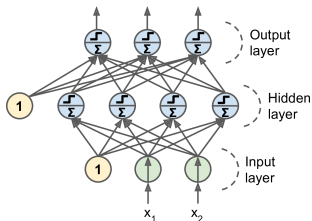
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass**: make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).
 2. Measure the **error** (compute $\text{cost}(\hat{y}^{(i)}, y^{(i)})$).
 3. **Backward pass**: go through each layer in **reverse** to measure the **error contribution** from **each connection**.

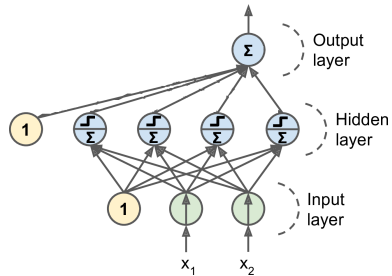


Training Feedforward Neural Networks

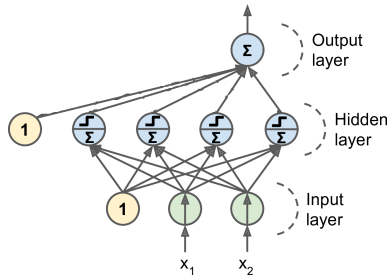
- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass**: make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).
 2. Measure the **error** (compute $\text{cost}(\hat{y}^{(i)}, y^{(i)})$).
 3. **Backward pass**: go through each layer in **reverse** to measure the **error contribution** from **each connection**.
 4. **Tweak the connection weights** to **reduce the error** (update W and b).



-
- The diagram illustrates a feedforward neural network with three layers: Input layer, Hidden layer, and Output layer. The input layer consists of three nodes (one yellow, two green), the hidden layer consists of four nodes (all blue), and the output layer consists of three nodes (all blue). Each node contains a sigmoid function symbol. The nodes are fully connected to each other. The input nodes are labeled x_1 and x_2 . The output nodes are labeled with σ symbols.



- **Output** function: $\hat{y}_j = w_j^T h + b_j$.



-
- The diagram illustrates a neural network with three layers: Input, Hidden, and Output. The Input layer consists of three nodes: a bias node labeled '1' (yellow) and two input nodes labeled x_1 and x_2 (green). The Hidden layer consists of four nodes, each containing a sigmoid function symbol σ and a summation symbol Σ (blue). The Output layer consists of one node containing a summation symbol Σ (blue). Arrows indicate the flow of information from the input nodes to the hidden nodes, and from the hidden nodes to the output node. Dashed lines group the nodes into their respective layers.

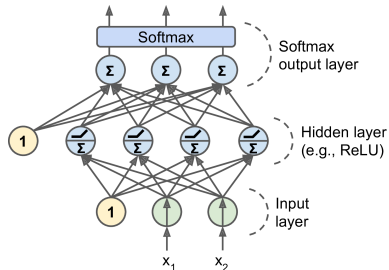
-
- The diagram illustrates a neural network structure with three layers: Input, Hidden, and Output. The Input layer consists of two green nodes; the first is a bias node labeled '1', and the second receives an input labeled x_1 . The Hidden layer contains four blue nodes, each featuring a sigmoid activation function symbol (σ) and a summation symbol (Σ). The Output layer has a single blue node, also with a sigmoid activation function symbol and a summation symbol. Arrows indicate the flow of information from the input nodes to the hidden nodes, and from the hidden nodes to the output node.

-
- The diagram illustrates a neural network with three layers: Input, Hidden, and Output. The input layer consists of two nodes, x_1 and x_2 , and a bias node labeled 1. The hidden layer consists of four nodes, each with a sigmoid activation function. The output layer consists of one node with a sigmoid activation function. All nodes are fully connected to the next layer.

-

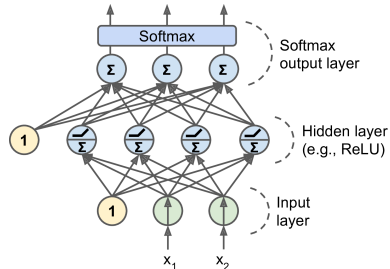
Output Unit (3/3)

- **Softmax units** in neurons of the **output layer** (**multinomial** classification).



Output Unit (3/3)

- **Softmax units** in neurons of the **output layer** (**multinomial** classification).
- **Output** function: $\hat{y}_j = \text{softmax}(w_j^T h + b_j)$.



-
- The diagram illustrates a neural network architecture with three layers:
- Input layer:** Consists of three nodes. The first node is a yellow circle containing the number '1'. The other two nodes are green circles. Two arrows, labeled x_1 and x_2 , point into the bottom of the green nodes.
 - Hidden layer (e.g., ReLU):** Consists of four nodes, each represented by a blue circle containing a summation symbol Σ and a diagonal slash. These nodes are fully connected to the input nodes.
 - Softmax output layer:** Consists of three nodes, each represented by a blue circle containing a summation symbol Σ . These nodes are fully connected to the hidden layer nodes. Above each node in this layer is an upward-pointing arrow. A dashed line groups these three nodes under the label 'Softmax output layer'.
- A dashed line also groups the hidden layer nodes under the label 'Hidden layer (e.g., ReLU)'.

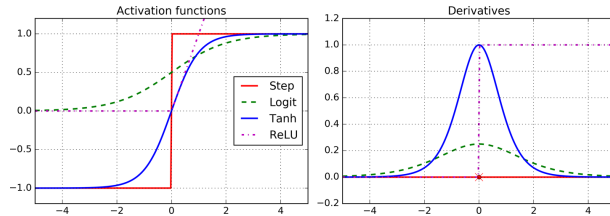


Hidden Units

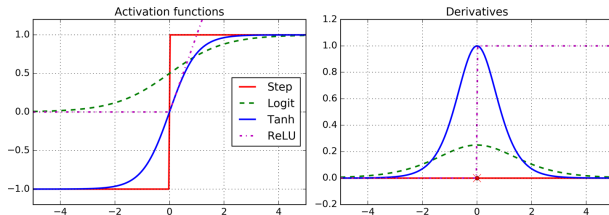
- ▶ In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**

Hidden Units

- In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**
- Alternative activation functions:

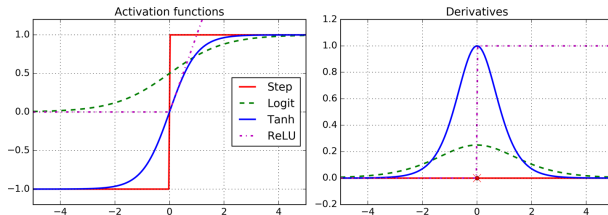


- In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**
- Alternative activation functions:
 1. **Logistic function (sigmoid)**: $\sigma(z) = \frac{1}{1+e^{-z}}$



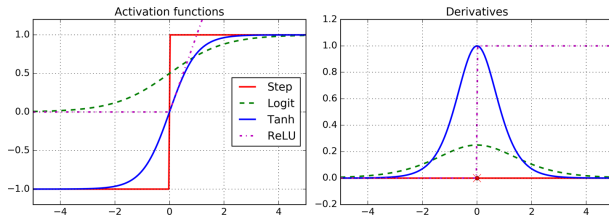
Hidden Units

- ▶ In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**
- ▶ Alternative activation functions:
 1. **Logistic function (sigmoid):** $\sigma(z) = \frac{1}{1+e^{-z}}$
 2. **Hyperbolic tangent function:** $\tanh(z) = 2\sigma(2z) - 1$



Hidden Units

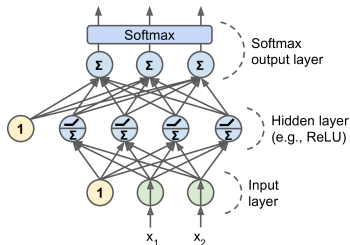
- ▶ In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**
- ▶ Alternative activation functions:
 1. **Logistic function (sigmoid)**: $\sigma(z) = \frac{1}{1+e^{-z}}$
 2. **Hyperbolic tangent function**: $\tanh(z) = 2\sigma(2z) - 1$
 3. **Rectified linear units (ReLU)**: $\text{ReLU}(z) = \max(0, z)$



Feedforward Network in TensorFlow



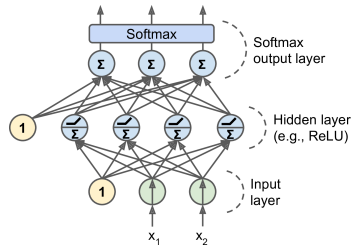
Feedforward Network in TensorFlow



```
n_output = 3
n_hidden = 4
n_features = 2
```

```
model = keras.models.Sequential()
model.add(keras.layers.Dense(n_hidden, input_shape=(n_features,), activation="relu"))
model.add(keras.layers.Dense(n_output, activation="softmax"))
```

Feedforward Network in TensorFlow

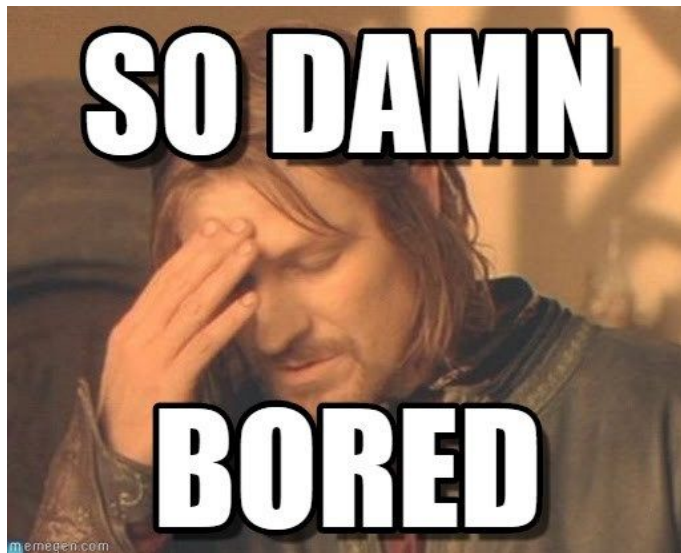


```
n_output = 3
n_hidden = 4
n_features = 2
```

```
model = keras.models.Sequential()
model.add(keras.layers.Dense(n_hidden, input_shape=(n_features,), activation="relu"))
model.add(keras.layers.Dense(n_output, activation="softmax"))
```

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=30)
```

Dive into Backpropagation Algorithm



[<https://i.pining.com/originals/82/d9/2c/82d92c2c15c580c2b2fce65a83fe0b3f.jpg>]



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- ▶ Example:

$$z = f(y) = 5y^4 \text{ and } y = g(x) = x^3 + 7$$

Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- ▶ Example:

$$z = f(y) = 5y^4 \text{ and } y = g(x) = x^3 + 7$$

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

Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- ▶ Example:

$$z = f(y) = 5y^4 \text{ and } y = g(x) = x^3 + 7$$

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

$$\frac{dz}{dy} = 20y^3 \text{ and } \frac{dy}{dx} = 3x^2$$

Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- ▶ Example:

$$z = f(y) = 5y^4 \text{ and } y = g(x) = x^3 + 7$$

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

$$\frac{dz}{dy} = 20y^3 \text{ and } \frac{dy}{dx} = 3x^2$$

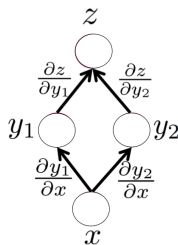
$$\frac{dz}{dx} = 20y^3 \times 3x^2 = 20(x^3 + 7) \times 3x^2$$

Chain Rule of Calculus (2/2)

- Two paths chain rule.

$$z = f(y_1, y_2) \text{ where } y_1 = g(x) \text{ and } y_2 = h(x)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$



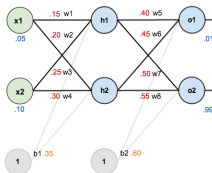


Backpropagation

- ▶ Backpropagation training algorithm for MLPs
- ▶ The algorithm repeats the following steps:
 1. Forward pass
 2. Backward pass

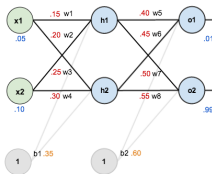
Backpropagation - Forward Pass

- Calculates outputs given input patterns.



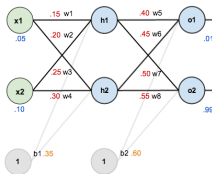
Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.
- ▶ For each training instance



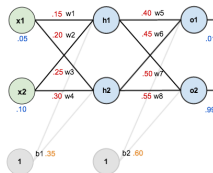
Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.
- ▶ For each training instance
 - Feeds it to the network and computes the output of every neuron in each consecutive layer.



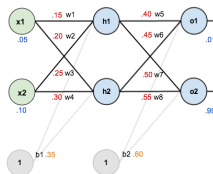
Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.
- ▶ For each training instance
 - Feeds it to the network and computes the output of every neuron in each consecutive layer.
 - Measures the network's output error (i.e., the difference between the true and the predicted output of the network)



Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.
- ▶ For each training instance
 - Feeds it to the network and computes the output of every neuron in each consecutive layer.
 - Measures the network's output error (i.e., the difference between the true and the predicted output of the network)
 - Computes how much each neuron in the last hidden layer contributed to each output neuron's error.

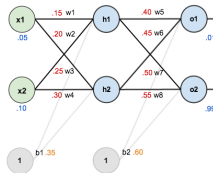




-
- The diagram illustrates a structural equation model (SEM) with the following components and paths:
- Latent Variables (Nodes):**
 - x_1 (green circle, top left)
 - x_2 (green circle, middle left)
 - b_1 (grey circle, bottom left)
 - b_2 (grey circle, bottom right)
 - Observed Variables (Nodes):**
 - h_1 (blue circle, top middle)
 - h_2 (blue circle, middle right)
 - o_1 (blue circle, top right)
 - o_2 (blue circle, middle right)
 - Measurement Model (Paths from Latent to Observed):**
 - $x_1 \rightarrow h_1$ (loading: .15, error: .05)
 - $x_1 \rightarrow h_2$ (loading: .20, error: .10)
 - $x_2 \rightarrow h_1$ (loading: .25, error: .10)
 - $x_2 \rightarrow h_2$ (loading: .30, error: .10)
 - $b_1 \rightarrow h_1$ (loading: .35, error: .05)
 - $b_1 \rightarrow h_2$ (loading: .45, error: .05)
 - $b_2 \rightarrow h_1$ (loading: .40, error: .05)
 - $b_2 \rightarrow h_2$ (loading: .50, error: .05)
 - $b_1 \rightarrow o_1$ (loading: .40, error: .05)
 - $b_1 \rightarrow o_2$ (loading: .50, error: .05)
 - $b_2 \rightarrow o_1$ (loading: .45, error: .05)
 - $b_2 \rightarrow o_2$ (loading: .55, error: .05)
 - Error Terms (Nodes):**
 - w_1 (green circle, top left)
 - w_2 (green circle, middle left)
 - w_3 (green circle, bottom left)
 - w_4 (green circle, bottom left)
 - w_5 (green circle, top middle)
 - w_6 (green circle, middle right)
 - w_7 (green circle, middle right)
 - w_8 (green circle, bottom right)
 - Structural Model (Paths between Observed Variables):**
 - $h_1 \rightarrow o_1$ (path coefficient: .40)
 - $h_1 \rightarrow o_2$ (path coefficient: .45)
 - $h_2 \rightarrow o_1$ (path coefficient: .50)
 - $h_2 \rightarrow o_2$ (path coefficient: .55)

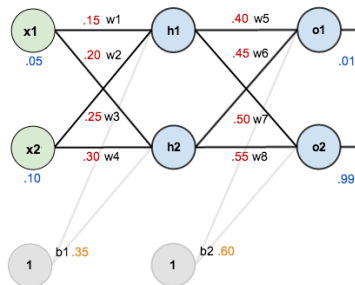
Backpropagation - Backward Pass

- ▶ Updates weights by calculating gradients.
- ▶ Measures how much of these error contributions came from each neuron in the previous hidden layer
 - Proceeds until the algorithm reaches the input layer.
- ▶ The last step is the gradient descent step on all the connection weights in the network, using the error gradients measured earlier.



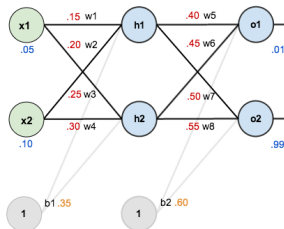
Backpropagation Example

- ▶ Two **inputs**, two **hidden**, and two **output** neurons.
- ▶ Bias in **hidden** and **output** neurons.
- ▶ Logistic activation in all the neurons.
- ▶ Squared error function as the cost function.



Backpropagation - Forward Pass (1/3)

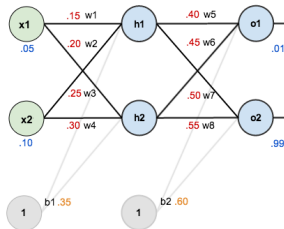
- Compute the **output** of the **hidden layer**



$$\text{net}_{h1} = w_1x_1 + w_2x_2 + b_1 = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

Backpropagation - Forward Pass (1/3)

- Compute the **output** of the **hidden layer**



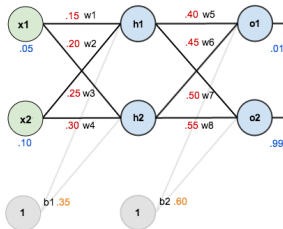
$$\text{net}_{h1} = w_1x_1 + w_2x_2 + b_1 = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

$$\text{out}_{h1} = \frac{1}{1 + e^{\text{net}_{h1}}} = \frac{1}{1 + e^{0.3775}} = 0.59327$$

$$\text{out}_{h2} = 0.59688$$

Backpropagation - Forward Pass (2/3)

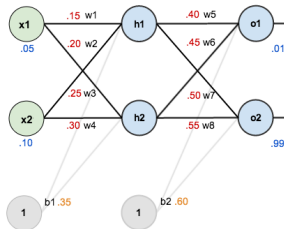
- Compute the **output** of the **output layer**



$$\text{net}_{o1} = w_5 \text{out}_{h1} + w_6 \text{out}_{h2} + b_2 = 0.4 \times 0.59327 + 0.45 \times 0.59688 + 0.6 = 1.1059$$

Backpropagation - Forward Pass (2/3)

- Compute the **output** of the **output layer**



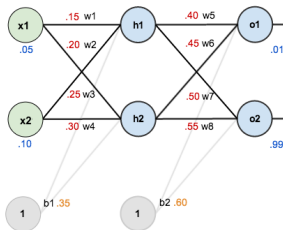
$$\text{net}_{o1} = w_5 \text{out}_{h1} + w_6 \text{out}_{h2} + b_2 = 0.4 \times 0.59327 + 0.45 \times 0.59688 + 0.6 = 1.1059$$

$$\text{out}_{o1} = \frac{1}{1 + e^{\text{net}_{o1}}} = \frac{1}{1 + e^{1.1059}} = 0.75136$$

$$\text{out}_{o2} = 0.77292$$

Backpropagation - Forward Pass (3/3)

- Calculate the **error** for each output



$$E_{o1} = \frac{1}{2}(\text{target}_{o1} - \text{output}_{o1})^2 = \frac{1}{2}(0.01 - 0.75136)^2 = 0.27481$$

$$E_{o2} = 0.02356$$

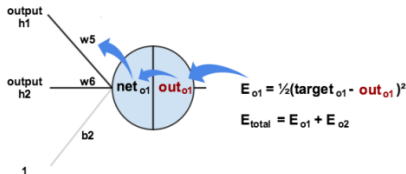
$$E_{\text{total}} = \sum \frac{1}{2}(\text{target} - \text{output})^2 = E_{o1} + E_{o2} = 0.27481 + 0.02356 = 0.29837$$



[<http://marimancusi.blogspot.com/2015/09/are-you-book-dragon.html>]

Backpropagation - Backward Pass - Output Layer (1/6)

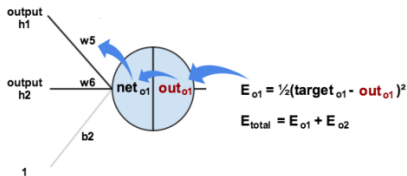
- ▶ Consider w_5
- ▶ We want to know how much a **change** in w_5 affects the **total error** ($\frac{\partial E_{\text{total}}}{\partial w_5}$)
- ▶ Applying the **chain rule**



$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

Backpropagation - Backward Pass - Output Layer (2/6)

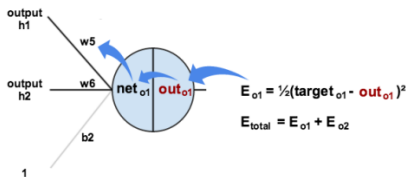
- First, how much does the **total error** change with **respect to the output**? ($\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}}$)



$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

Backpropagation - Backward Pass - Output Layer (2/6)

- First, how much does the **total error** change with **respect to the output**? ($\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}}$)



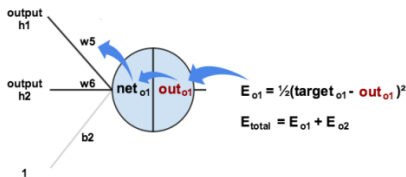
$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

$$E_{\text{total}} = \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1})^2 + \frac{1}{2}(\text{target}_{o2} - \text{out}_{o2})^2$$

$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} = -2 \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1}) = -(0.01 - 0.75136) = 0.74136$$

Backpropagation - Backward Pass - Output Layer (3/6)

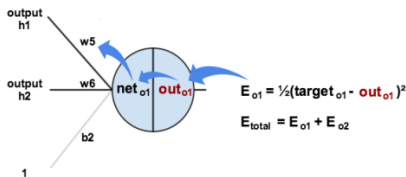
- Next, how much does the out_{o1} change with respect to its total input net_{o1} ?
 $\left(\frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \right)$



$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

Backpropagation - Backward Pass - Output Layer (3/6)

- Next, how much does the out_{o1} change with respect to its total input net_{o1} ?
 $\left(\frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \right)$



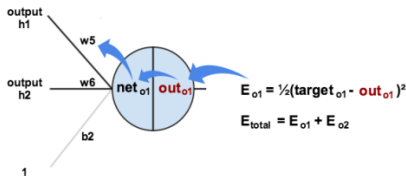
$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

$$\text{out}_{o1} = \frac{1}{1 + e^{-\text{net}_{o1}}}$$

$$\frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} = \text{out}_{o1}(1 - \text{out}_{o1}) = 0.75136(1 - 0.75136) = 0.18681$$

Backpropagation - Backward Pass - Output Layer (4/6)

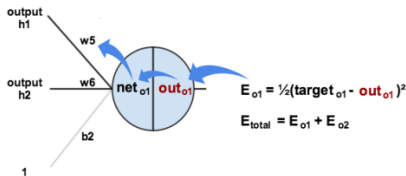
- Finally, how much does the total net_{o1} change with respect to w_5 ? ($\frac{\partial \text{net}_{o1}}{\partial w_5}$)



$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

Backpropagation - Backward Pass - Output Layer (4/6)

- Finally, how much does the total net_{o1} change with respect to w_5 ? ($\frac{\partial \text{net}_{o1}}{\partial w_5}$)



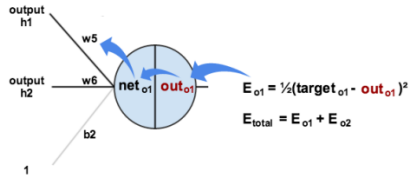
$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

$$\text{net}_{o1} = w_5 \times \text{out}_{h1} + w_6 \times \text{out}_{h2} + b_2$$

$$\frac{\partial \text{net}_{o1}}{\partial w_5} = \text{out}_{h1} = 0.59327$$

Backpropagation - Backward Pass - Output Layer (5/6)

- Putting it all together:

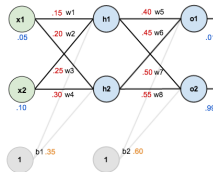


$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{\text{total}}}{\partial w_5} = 0.74136 \times 0.18681 \times 0.59327 = 0.08216$$

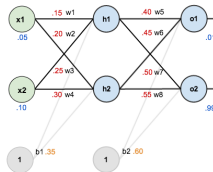
Backpropagation - Backward Pass - Output Layer (6/6)

- To **decrease the error**, we subtract this value from the **current weight**.



Backpropagation - Backward Pass - Output Layer (6/6)

- ▶ To **decrease the error**, we subtract this value from the **current weight**.
- ▶ We assume that the **learning rate** is $\eta = 0.5$.



$$w_5^{(next)} = w_5 - \eta \times \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 \times 0.08216 = 0.35891$$

$$w_6^{(next)} = 0.40866$$

$$w_7^{(next)} = 0.5113$$

$$w_8^{(next)} = 0.56137$$

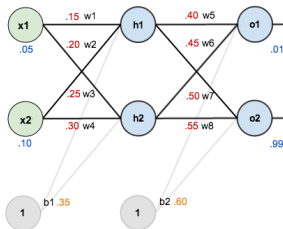


[<https://makeameme.org/meme/oooh-this>]

Backpropagation - Backward Pass - Hidden Layer (1/8)

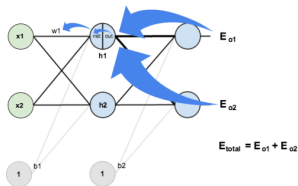
- ▶ Continue the **backwards pass** by calculating new values for w_1 , w_2 , w_3 , and w_4 .
- ▶ For w_1 we have:

$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \times \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \times \frac{\partial \text{net}_{h1}}{\partial w_1}$$



Backpropagation - Backward Pass - Hidden Layer (2/8)

- ▶ Here, the **output of each hidden layer neuron** contributes to the **output of multiple output neurons**.
- ▶ E.g., out_{h1} affects both out_{o1} and out_{o2} , so $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons.

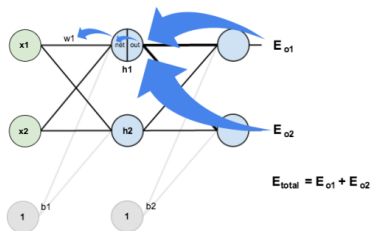


$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Backpropagation - Backward Pass - Hidden Layer (3/8)

► Starting with $\frac{\partial E_{o1}}{\partial \text{out}_{h1}}$



$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{h1}} + \frac{\partial E_{o2}}{\partial \text{out}_{h1}}$$

$$\frac{\partial E_{o1}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial \text{out}_{h1}}$$

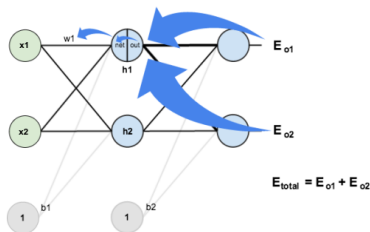
$$\frac{\partial E_{o1}}{\partial \text{out}_{o1}} = 0.74136, \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} = 0.18681$$

$$\text{net}_{o1} = w_5 \times \text{out}_{h1} + w_6 \times \text{out}_{h2} + b_2$$

$$\frac{\partial \text{net}_{o1}}{\partial \text{out}_{h1}} = w_5 = 0.40$$

Backpropagation - Backward Pass - Hidden Layer (4/8)

- Plugging them together.

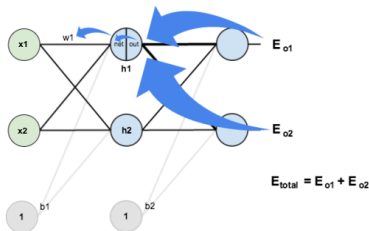


$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial out_{h1}} = 0.74136 \times 0.18681 \times 0.40 = 0.0554$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.01905$$

Backpropagation - Backward Pass - Hidden Layer (4/8)

- Plugging them together.



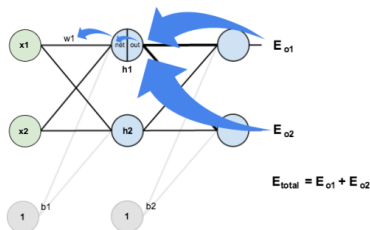
$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial out_{h1}} = 0.74136 \times 0.18681 \times 0.40 = 0.0554$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.01905$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.0554 + -0.01905 = 0.03635$$

Backpropagation - Backward Pass - Hidden Layer (5/8)

- Now we need to figure out $\frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}}$.



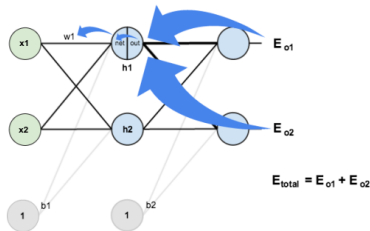
$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \times \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \times \frac{\partial \text{net}_{h1}}{\partial w_1}$$

$$\text{out}_{h1} = \frac{1}{1 + e^{-\text{net}_{h1}}}$$

$$\frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} = \text{out}_{h1}(1 - \text{out}_{h1}) = 0.59327(1 - 0.59327) = 0.2413$$

Backpropagation - Backward Pass - Hidden Layer (6/8)

► And then $\frac{\partial \text{net}_{h1}}{\partial w_1}$.



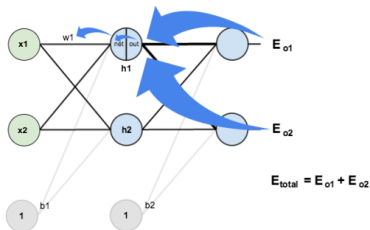
$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \times \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \times \frac{\partial \text{net}_{h1}}{\partial w_1}$$

$$\text{net}_{h1} = w_1 x_1 + w_2 x_2 + b_1$$

$$\frac{\partial \text{net}_{h1}}{\partial w_1} = x_1 = 0.05$$

Backpropagation - Backward Pass - Hidden Layer (7/8)

- ▶ Putting it all together.

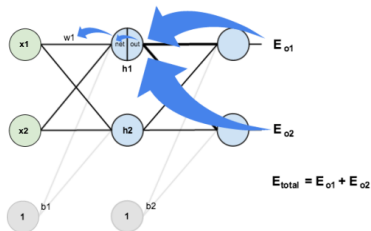


$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.03635 \times 0.2413 \times 0.05 = 0.00043$$

Backpropagation - Backward Pass - Hidden Layer (8/8)

- ▶ We can now update w_1 .
- ▶ Repeating this for w_2 , w_3 , and w_4 .



$$w_1^{(next)} = w_1 - \eta \times \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 \times 0.00043 = 0.14978$$

$$w_2^{(next)} = 0.19956$$

$$w_3^{(next)} = 0.24975$$

$$w_4^{(next)} = 0.2995$$

Summary



Summary

- ▶ LTU
- ▶ Perceptron
- ▶ Perceptron weakness
- ▶ MLP and feedforward neural network
- ▶ Gradient-based learning
- ▶ Backpropagation: forward pass and backward pass
- ▶ Output unit: linear, sigmoid, softmax
- ▶ Hidden units: sigmoid, tanh, relu



Reference

- ▶ Ian Goodfellow et al., Deep Learning (Ch. 6)
- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 10)

Questions?