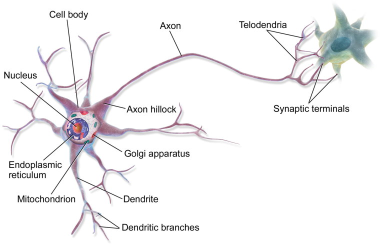
Deep Feedforwards Networks

Amir H. Payberah
payberah@kth.se
2020-10-22

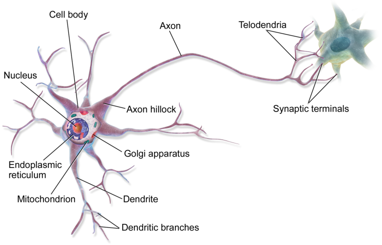
Biological Neurons (1/2)

- Brain architecture has inspired artificial neural networks.



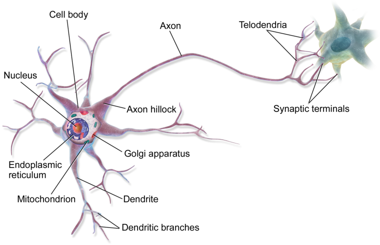
Biological Neurons (1/2)

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



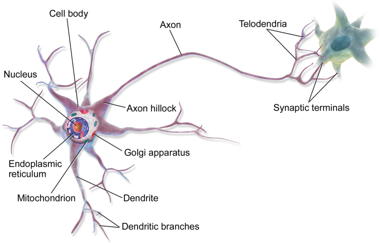
Biological Neurons (1/2)

- 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.



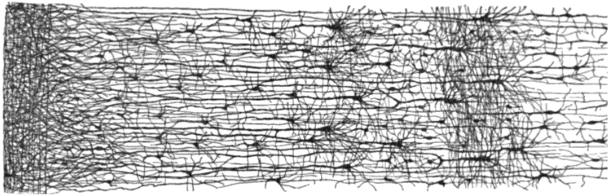
Biological Neurons (1/2)

- 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.

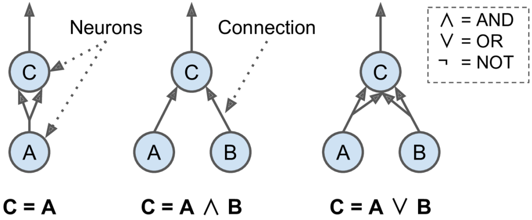


A Simple Artificial Neural Network

- 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]

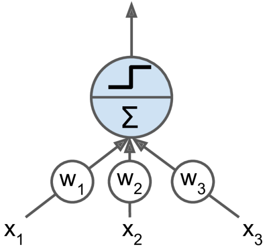
The Linear Threshold Unit (LTU)

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

The Linear Threshold Unit (LTU)

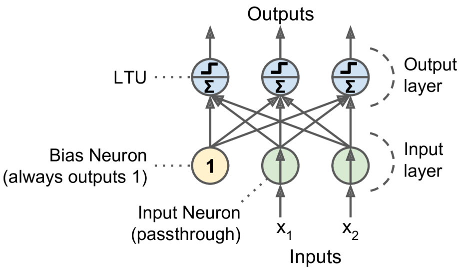
- Inputs of a LTU are numbers (not binary).
- Each input connection is associated with a weight.
- Computes a weighted sum of its inputs and applies a step function to that sum.

- $z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{w}^\intercal \mathbf{x}$
- $\hat{y} = \mathrm{step}(z) = \mathrm{step}(\mathbf{w}^\intercal \mathbf{x})$



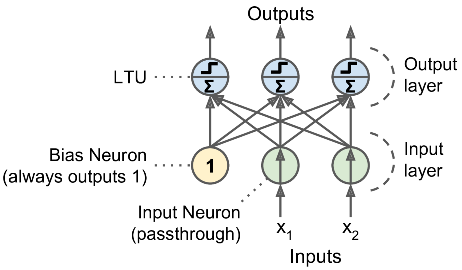
The Perceptron

- The perceptron is a single layer of LTUs.



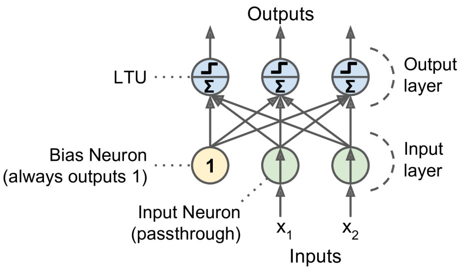
The Perceptron

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



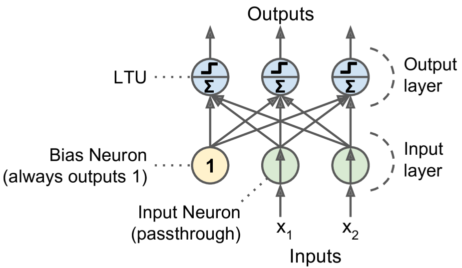
The Perceptron

- 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.



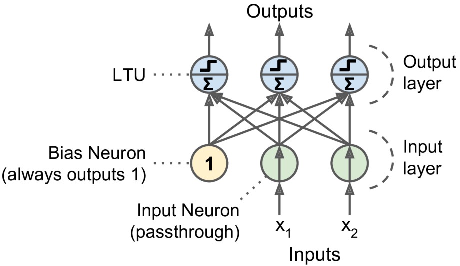
The Perceptron

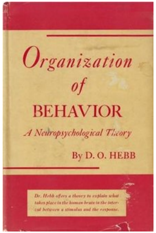
- The perceptron is a single layer of LTUs.
- The input neurons output whatever input they are fed.
- A bias neuron, which just outputs 1 all the time.
- If we use logistic function (sigmoid) instead of a step function, it computes a continuous output.



How is a Perceptron Trained? (1/2)

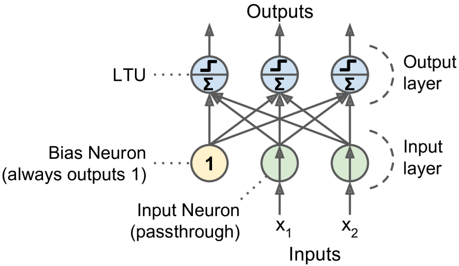
- The Perceptron training algorithm is inspired by Hebb's rule.

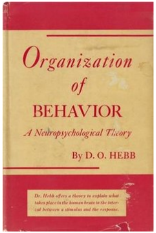




How is a Perceptron Trained? (1/2)

- 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.





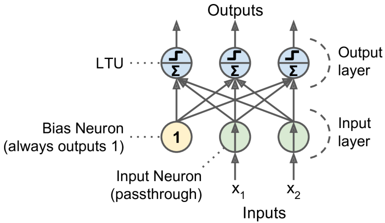
How is a Perceptron Trained? (2/2)

- Feed one training instance $\mathbf{x}$ 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 $\hat{y}$.
- Update the connection weights.

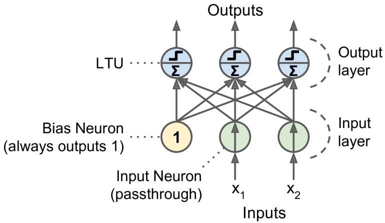
$$\hat{y}_j = \sigma(\mathbf{w}_j^\intercal \mathbf{x} + b)$$
$$J(\mathbf{w}_j) = \mathtt{cross\_entropy}(y_j, \hat{y}_j)$$
$$w_{i,j}^{(\mathtt{next})} = w_{i,j} - \eta \frac{\partial J(\mathbf{w}_j)}{w_i}$$



How is a Perceptron Trained? (2/2)

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

$$\hat{\mathtt{y}}_{\mathtt{j}} = \sigma(\mathbf{w}_{\mathtt{j}}^{\mathsf{T}}\mathbf{x} + \mathtt{b})$$
$$\mathtt{J}(\mathbf{w}_{\mathtt{j}}) = \mathtt{cross\_entropy}(\mathtt{y}_{\mathtt{j}}, \hat{\mathtt{y}}_{\mathtt{j}})$$
$$\mathtt{w}_{\mathtt{i,j}}^{(\mathtt{next})} = \mathtt{w}_{\mathtt{i,j}} - \eta \frac{\partial \mathtt{J}(\mathbf{w}_{\mathtt{j}})}{\mathtt{w}_{\mathtt{i}}}$$

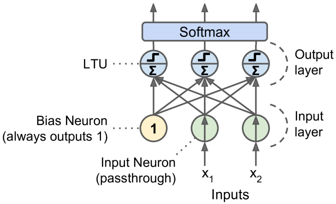


- $\mathtt{w}_{\mathtt{i,j}}$: the weight between neurons $\mathtt{i}$ and $\mathtt{j}$.
- $\mathtt{x}_{\mathtt{i}}$: the $\mathtt{i}$th input value.
- $\hat{\mathtt{y}}_{\mathtt{j}}$: the $\mathtt{j}$th predicted output value.
- $\mathtt{y}_{\mathtt{j}}$: the $\mathtt{j}$th true output value.
- η: the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

Perceptron in TensorFlow

TensorFlow

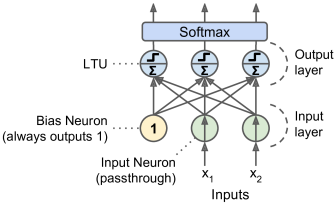
Perceptron in TensorFlow




```python
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)

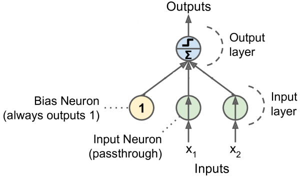
Perceptron Weakness (1/2)

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

Perceptron Weakness (1/2)

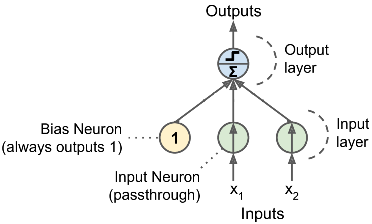
- Incapable of solving some trivial problems, e.g., XOR classification problem. Why?

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



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

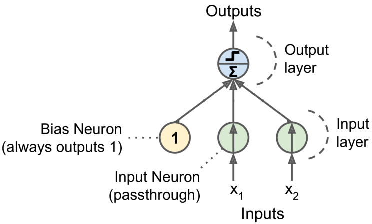
Perceptron Weakness (2/2)



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

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

Perceptron Weakness (2/2)

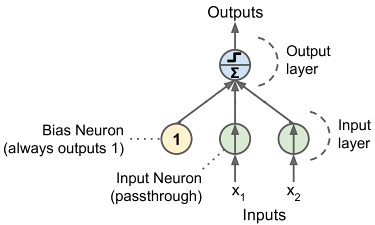


$$\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad \hat{y} = \mathtt{step}(z), z = w_1 x_1 + w_2 x_2 + b$$

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

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

Perceptron Weakness (2/2)



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

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

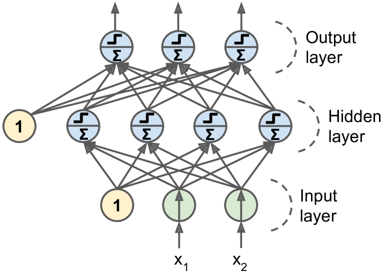
- If we minimize $J(\mathbf{w})$, we obtain $w_1 = 0$, $w_2 = 0$, and $b = \frac{1}{2}$.
- But, the model outputs 0.5 everywhere.

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.

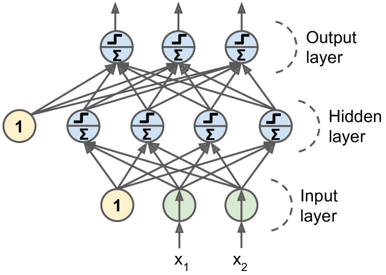
Feedforward Neural Network Architecture

- A feedforward neural network is composed of:
  - One input layer
  - One or more hidden layers
  - One final output layer



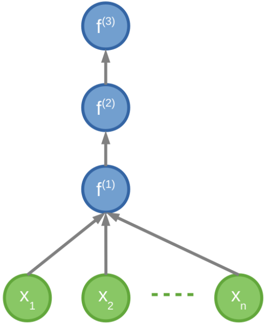
Feedforward Neural Network Architecture

- A feedforward neural network is composed of:
  - One input layer
  - One or more hidden layers
  - One final output layer
- Every layer except the output layer includes a bias neuron and is fully connected to the next layer.



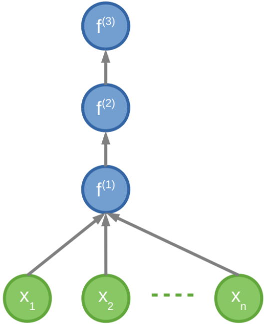
How Does it Work?

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



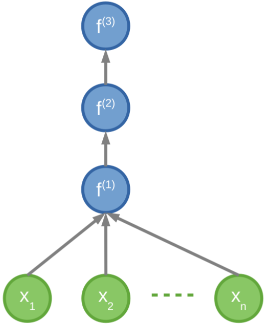
How Does it Work?

- 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 $\mathtt{f}^{(1)}$, $\mathtt{f}^{(2)}$, and $\mathtt{f}^{(3)}$ connected in a chain: $\hat{\mathtt{y}} = \mathtt{f}(\mathbf{x}) = \mathtt{f}^{(3)}(\mathtt{f}^{(2)}(\mathtt{f}^{(1)}(\mathbf{x})))$



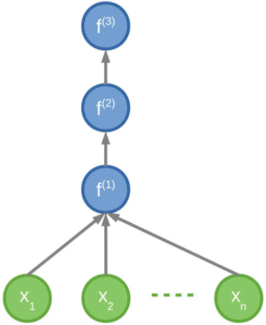
How Does it Work?

- 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 $\mathtt{f}^{(1)}$, $\mathtt{f}^{(2)}$, and $\mathtt{f}^{(3)}$ connected in a chain: $\hat{y} = \mathtt{f}(\mathbf{x}) = \mathtt{f}^{(3)}(\mathtt{f}^{(2)}(\mathtt{f}^{(1)}(\mathbf{x})))$
- $\mathtt{f}^{(1)}$ is called the first layer of the network.
- $\mathtt{f}^{(2)}$ is called the second layer, and so on.

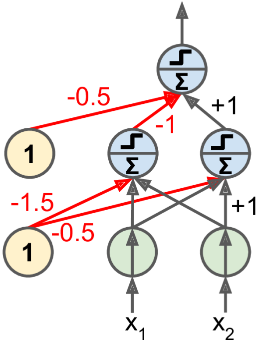


How Does it Work?

- 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 $\mathtt{f}^{(1)}$, $\mathtt{f}^{(2)}$, and $\mathtt{f}^{(3)}$ connected in a chain: $\hat{y} = \mathtt{f}(\mathbf{x}) = \mathtt{f}^{(3)}(\mathtt{f}^{(2)}(\mathtt{f}^{(1)}(\mathbf{x})))$
- $\mathtt{f}^{(1)}$ is called the first layer of the network.
- $\mathtt{f}^{(2)}$ is called the second layer, and so on.
- The length of the chain gives the depth of the model.

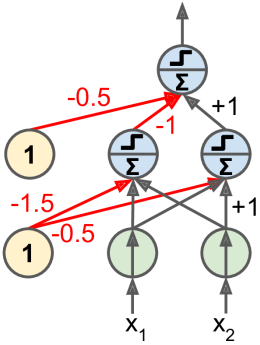


XOR with Feedforward Neural Network (1/3)



$$\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{W}_x = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{b}_x = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}$$

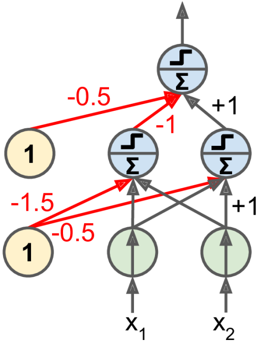
XOR with Feedforward Neural Network (2/3)



$$\mathbf{out_h} = \mathbf{X}\mathbf{W}_x^T + \mathbf{b}_x = \begin{bmatrix} -1.5 & -0.5 \\ -0.5 & 0.5 \\ -0.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix} \qquad \mathbf{h} = \mathtt{step}(\mathbf{out_h}) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{w}_h = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad \mathrm{b}_h = -0.5$$

XOR with Feedforward Neural Network (3/3)



$$\mathbf{out} = \mathbf{w}_h^{\intercal}\mathbf{h} + b_h = \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{bmatrix} \qquad \mathtt{step}(\mathbf{out}) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

How to Learn Model Parameters $\mathbf{W}$?

Feedforward Neural Network - Cost Function

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

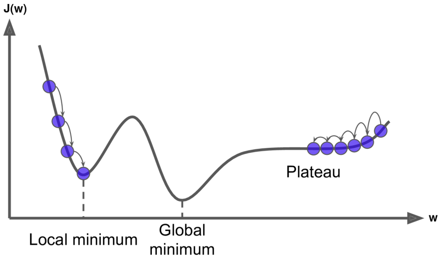
$$\mathtt{cost}(\mathtt{y}, \hat{\mathtt{y}}) = -\sum_{\mathtt{j}} \mathtt{y}_{\mathtt{j}} \mathtt{log}(\hat{\mathtt{y}}_{\mathtt{j}})$$

Gradient-Based Learning (1/2)

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

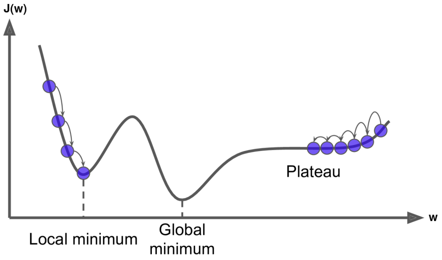
Gradient-Based Learning (1/2)

- 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)

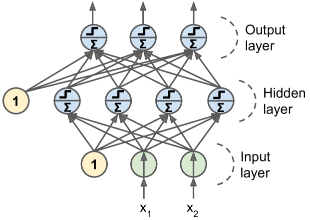
- Stochastic gradient descent applied to non-convex cost functions has no such convergence guarantee.
- It is sensitive to the values of the initial parameters.
- For feedforward neural networks, it is important to initialize all weights to small random values.

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.

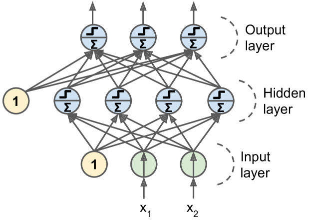
Training Feedforward Neural Networks

- How to train a feedforward neural network?



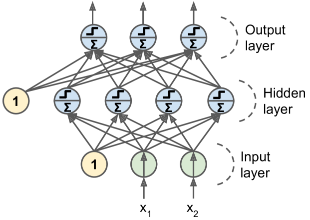
Training Feedforward Neural Networks

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



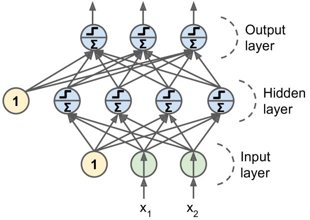
Training Feedforward Neural Networks

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



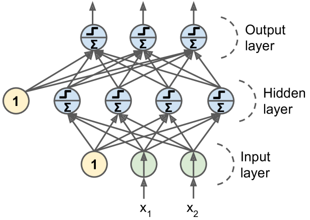
Training Feedforward Neural Networks

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



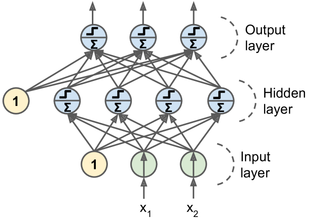
Training Feedforward Neural Networks

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



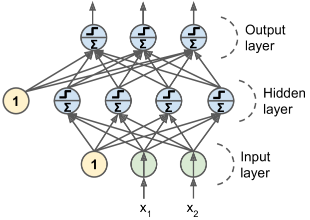
Training Feedforward Neural Networks

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



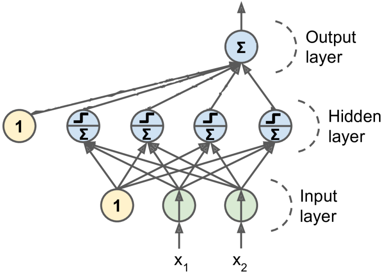
Training Feedforward Neural Networks

- How to train a feedforward neural network?
- For each training instance $\mathbf{x}^{(i)}$ the algorithm does the following steps:
  1. Forward pass: make a prediction (compute $\hat{y}^{(i)} = f(\mathbf{x}^{(i)})$).
  2. Measure the error (compute $\mathtt{cost}(\hat{y}^{(i)}, y^{(i)})$).
  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 $\mathbf{W}$ and $\mathbf{b}$).
- It's called the backpropagation training algorithm



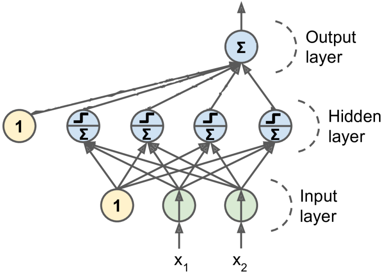
Output Unit (1/3)

- Linear units in neurons of the output layer.



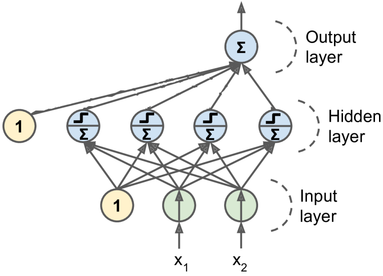
Output Unit (1/3)

- Linear units in neurons of the output layer.
- Output function: $\hat{y}_j = \mathbf{w}_j^\intercal \mathbf{h} + b_j$.



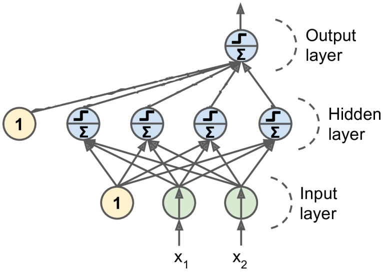
Output Unit (1/3)

- Linear units in neurons of the output layer.
- Output function: $\hat{y}_j = \mathbf{w}_j^\top \mathbf{h} + b_j$.
- Cost function: minimizing the mean squared error.

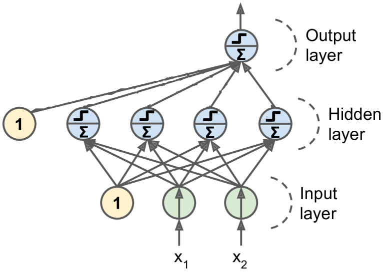


Output Unit (2/3)

- Sigmoid units in neurons of the output layer (binomial classification).

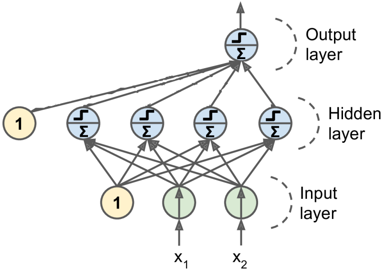


- Sigmoid units in neurons of the output layer (binomial classification).
- Output function: $\hat{y}_j = \sigma(\mathbf{w}_j^\mathsf{T}\mathbf{h} + b_j)$.



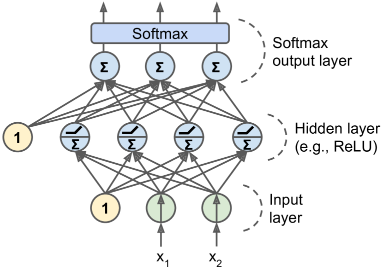
Output Unit (2/3)

- Sigmoid units in neurons of the output layer (binomial classification).
- Output function: $\hat{y}_j = \sigma(\mathbf{w}_j^\intercal \mathbf{h} + b_j)$.
- Cost function: minimizing the cross-entropy.



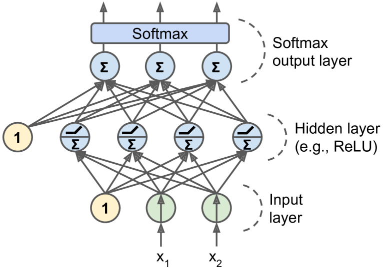
Output Unit (3/3)

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



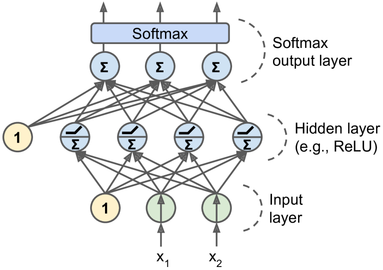
Output Unit (3/3)

- Softmax units in neurons of the output layer (multinomial classification).
- Output function: $\hat{y}_j = \mathtt{softmax}(\mathbf{w}_j^\intercal \mathbf{h} + b_j)$.



Output Unit (3/3)

- Softmax units in neurons of the output layer (multinomial classification).
- Output function: $\hat{\mathtt{y}}_\mathtt{j} = \mathtt{softmax}(\mathbf{w}_\mathtt{j}^\intercal \mathbf{h} + \mathtt{b}_\mathtt{j})$.
- Cost function: minimizing the cross-entropy.

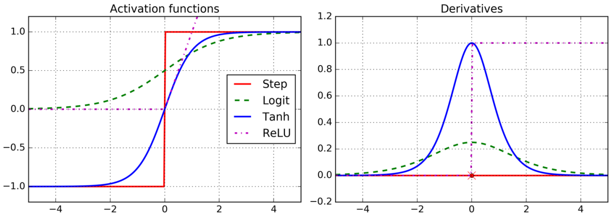


Hidden Units

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

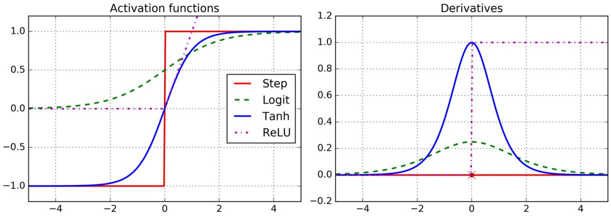
Hidden Units

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



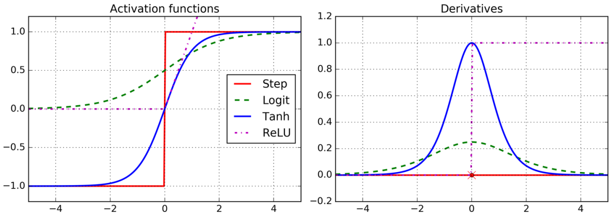
Hidden Units

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



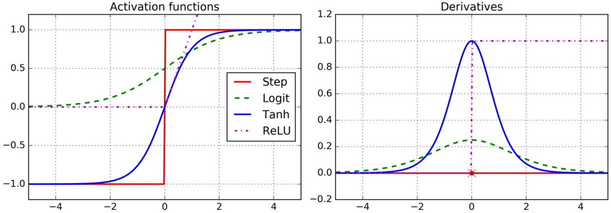
Hidden Units

- 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: $\tanh(z) = 2\sigma(2z) - 1$



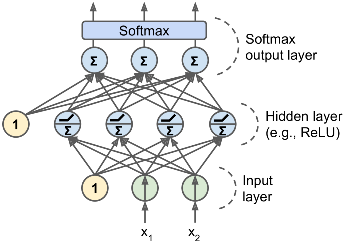
Hidden Units

- 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 (ReLUs): $\mathrm{ReLU}(z) = \max(0, z)$



Feedforward Network in TensorFlow

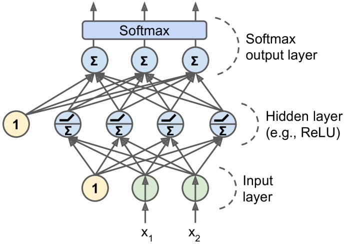
Feedforward Network in TensorFlow




```
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

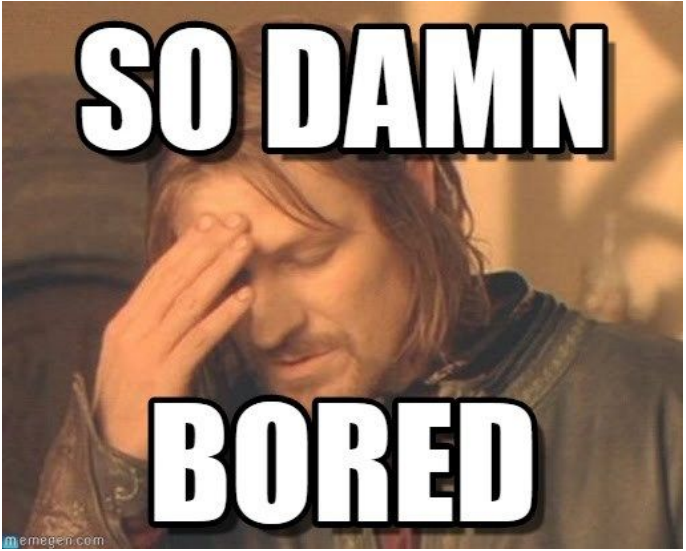



```python
n_output = 3
n_hidden = 4
n_features = 2

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"))
```

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

Dive into Backpropagation Algorithm



[https://i.pinimg.com/originals/82/d9/2c/82d92c2c15c580c2b2fce65a83fe0b3f.jpg]

Chain Rule of Calculus (1/2)

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

Chain Rule of Calculus (1/2)

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

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:

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

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

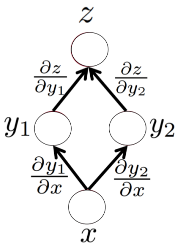
$$\frac{dz}{dy} = 20y^3 \text{ and } \frac{dy}{dx} = 3x^2$$

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

Chain Rule of Calculus (2/2)

- Two paths chain rule.

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

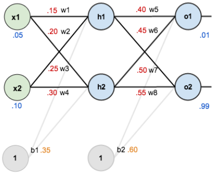


Backpropagation

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

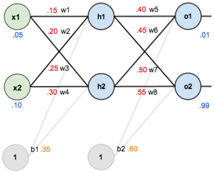
Backpropagation - Forward Pass

- Calculates outputs given input patterns.



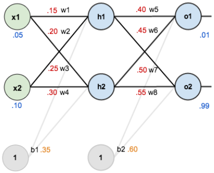
Backpropagation - Forward Pass

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



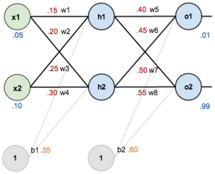
Backpropagation - Forward Pass

- 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.



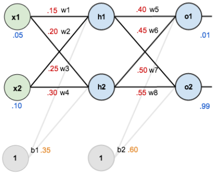
Backpropagation - Forward Pass

- 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)



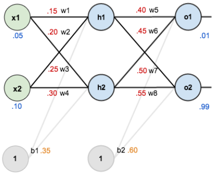
Backpropagation - Forward Pass

- 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.



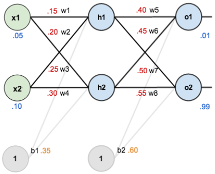
Backpropagation - Backward Pass

- Updates weights by calculating gradients.



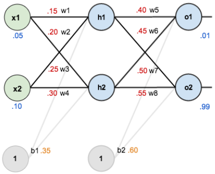
Backpropagation - Backward Pass

- 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.



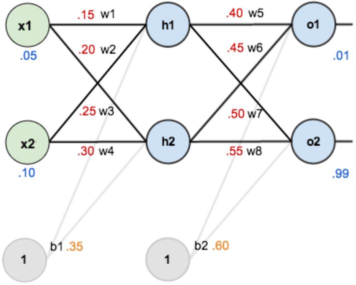
Backpropagation - Backward Pass

- 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.



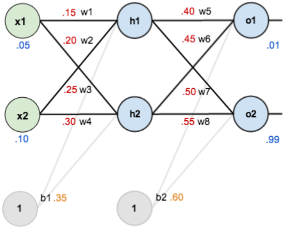
Backpropagation Example

- 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)

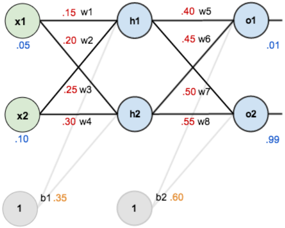
- Compute the output of the hidden layer



$$\mathtt{net}_{h1} = \mathtt{w}_1\mathtt{x}_1 + \mathtt{w}_2\mathtt{x}_2 + \mathtt{b}_1 = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

Backpropagation - Forward Pass (1/3)

- Compute the output of the hidden layer



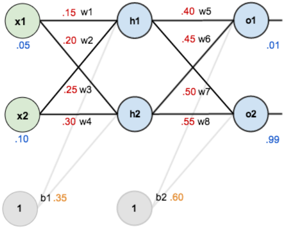
$$\mathtt{net}_{h1} = \mathtt{w}_1\mathtt{x}_1 + \mathtt{w}_2\mathtt{x}_2 + \mathtt{b}_1 = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

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

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

Backpropagation - Forward Pass (2/3)

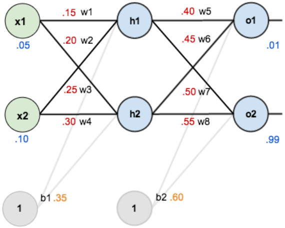
- Compute the output of the output layer



$$\mathtt{net}_{o1} = \mathtt{w}_5\mathtt{out}_{h1} + \mathtt{w}_6\mathtt{out}_{h2} + \mathtt{b}_2 = 0.4 \times 0.59327 + 0.45 \times 0.59688 + 0.6 = 1.1059$$

Backpropagation - Forward Pass (2/3)

- Compute the output of the output layer



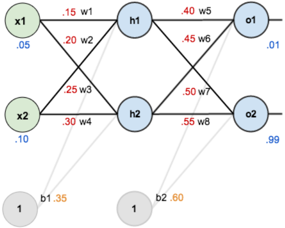
$$\mathtt{net}_{o1} = \mathtt{w}_5\mathtt{out}_{h1} + \mathtt{w}_6\mathtt{out}_{h2} + \mathtt{b}_2 = 0.4 \times 0.59327 + 0.45 \times 0.59688 + 0.6 = 1.1059$$

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

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

Backpropagation - Forward Pass (3/3)

- Calculate the error for each output



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

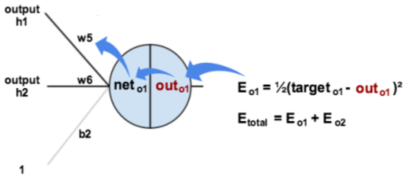
$$E_{o2} = 0.02356$$

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

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

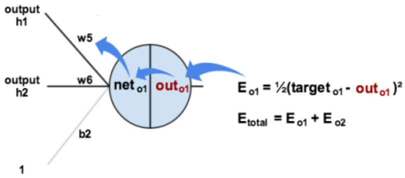
Backpropagation - Backward Pass - Output Layer (1/6)

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



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

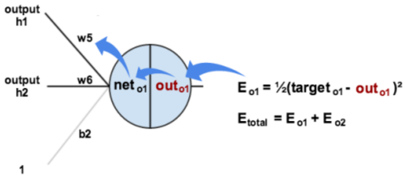
- First, how much does the total error change with respect to the output? ($\frac{\partial E_{total}}{\partial out_{o1}}$)



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

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

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



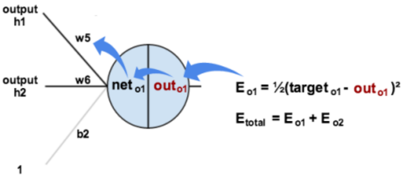
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -2\frac{1}{2}(target_{o1} - out_{o1}) = -(0.01 - 0.75136) = 0.74136$$

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

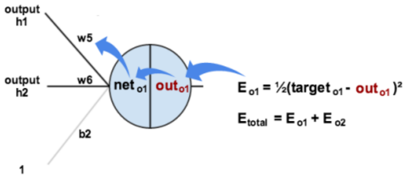
- Next, how much does the $\mathtt{out}_{o1}$ change with respect to its total input $\mathtt{net}_{o1}$? $\left(\frac{\partial \mathtt{out}_{o1}}{\partial \mathtt{net}_{o1}}\right)$



$$\frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{w}_5} = \frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{out}_{o1}} \times \frac{\partial \mathtt{out}_{o1}}{\partial \mathtt{net}_{o1}} \times \frac{\partial \mathtt{net}_{o1}}{\partial \mathtt{w}_5}$$

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

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



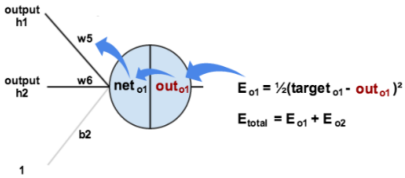
$$\frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{w}_5} = \frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{out}_{o1}} \times \frac{\partial \mathtt{out}_{o1}}{\partial \mathtt{net}_{o1}} \times \frac{\partial \mathtt{net}_{o1}}{\partial \mathtt{w}_5}$$

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

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

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

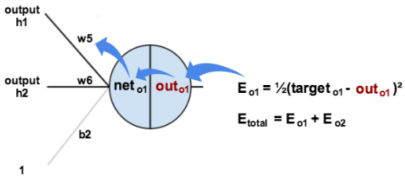
- Finally, how much does the total $\mathtt{net}_{o1}$ change with respect to $\mathtt{w}_5$? ($\frac{\partial \mathtt{net}_{o1}}{\partial \mathtt{w}_5}$)



$$\frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{w}_5} = \frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{out}_{o1}} \times \frac{\partial \mathtt{out}_{o1}}{\partial \mathtt{net}_{o1}} \times \frac{\partial \mathtt{net}_{o1}}{\partial \mathtt{w}_5}$$

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

- Finally, how much does the total $\mathtt{net}_{o1}$ change with respect to $\mathtt{w}_5$? ($\frac{\partial \mathtt{net}_{o1}}{\partial \mathtt{w}_5}$)



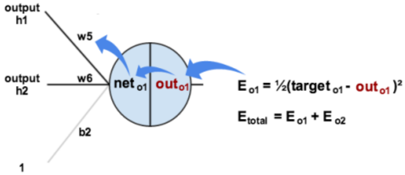
$$\frac{\partial E_{\mathtt{total}}}{\partial \mathtt{w}_5} = \frac{\partial E_{\mathtt{total}}}{\partial \mathtt{out}_{o1}} \times \frac{\partial \mathtt{out}_{o1}}{\partial \mathtt{net}_{o1}} \times \frac{\partial \mathtt{net}_{o1}}{\partial \mathtt{w}_5}$$

$$\mathtt{net}_{o1} = \mathtt{w}_5 \times \mathtt{out}_{h1} + \mathtt{w}_6 \times \mathtt{out}_{h2} + \mathtt{b}_2$$

$$\frac{\partial \mathtt{net}_{o1}}{\partial \mathtt{w}_5} = \mathtt{out}_{h1} = 0.59327$$

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

- Putting it all together:

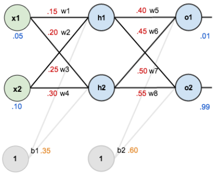


$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

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

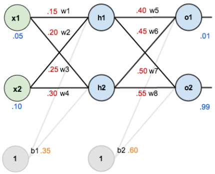
Backpropagation - Backward Pass - Output Layer (6/6)

- 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 $\eta = 0.5$.

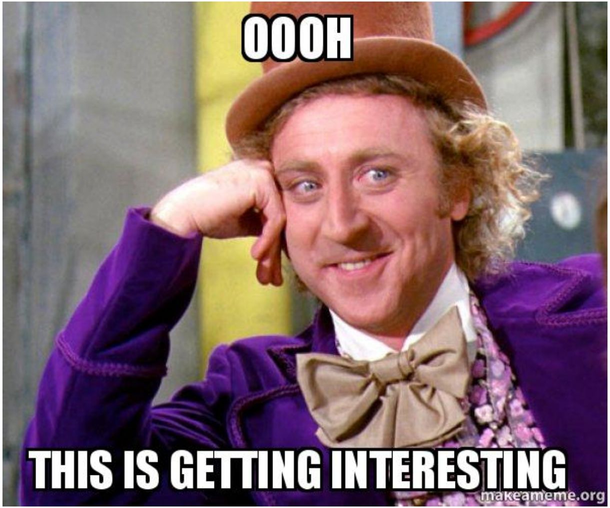


$$\mathtt{w}_5^{(\mathtt{next})} = \mathtt{w}_5 - \eta \times \frac{\partial \mathrm{E}_{\mathtt{total}}}{\partial \mathtt{w}_5} = 0.4 - 0.5 \times 0.08216 = 0.35891$$

$$\mathtt{w}_6^{(\mathtt{next})} = 0.40866$$

$$\mathtt{w}_7^{(\mathtt{next})} = 0.5113$$

$$\mathtt{w}_8^{(\mathtt{next})} = 0.56137$$

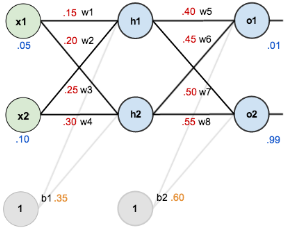


[https://makeameme.org/meme/oooh-this]

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

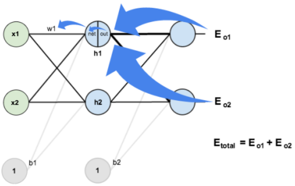
- Continue the backwards pass by calculating new values for $\mathtt{w}_1$, $\mathtt{w}_2$, $\mathtt{w}_3$, and $\mathtt{w}_4$.
- For $\mathtt{w}_1$ we have:

$$\frac{\partial E_{\mathtt{total}}}{\partial \mathtt{w}_1} = \frac{\partial E_{\mathtt{total}}}{\partial \mathtt{out}_{\mathtt{h1}}} \times \frac{\partial \mathtt{out}_{\mathtt{h1}}}{\partial \mathtt{net}_{\mathtt{h1}}} \times \frac{\partial \mathtt{net}_{\mathtt{h1}}}{\partial \mathtt{w}_1}$$



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

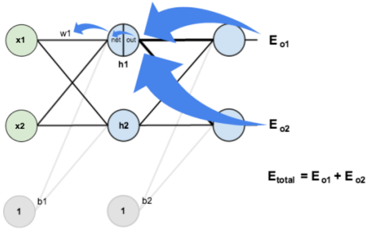
- Here, the output of each hidden layer neuron contributes to the output of multiple output neurons.
- E.g., $\mathtt{out_{h1}}$ affects both $\mathtt{out_{o1}}$ and $\mathtt{out_{o2}}$, so $\frac{\partial \mathtt{E_{total}}}{\partial \mathtt{out_{h1}}}$ needs to take into consideration its effect on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



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

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

- Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$



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

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial out_{h1}}$$

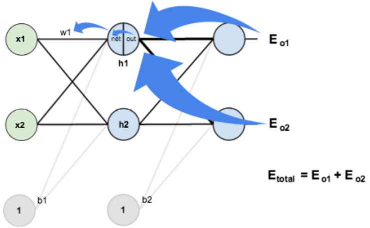
$$\frac{\partial E_{o1}}{\partial out_{o1}} = 0.74136, \frac{\partial out_{o1}}{\partial net_{o1}} = 0.18681$$

$$net_{o1} = w_5 \times out_{h1} + w_6 \times out_{h2} + b_2$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

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

- Plugging them together.

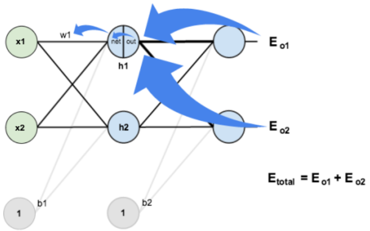


$$\frac{\partial \mathtt{E}_{\mathtt{o1}}}{\partial \mathtt{out}_{\mathtt{h1}}} = \frac{\partial \mathtt{E}_{\mathtt{o1}}}{\partial \mathtt{out}_{\mathtt{o1}}} \times \frac{\partial \mathtt{out}_{\mathtt{o1}}}{\partial \mathtt{net}_{\mathtt{o1}}} \times \frac{\partial \mathtt{net}_{\mathtt{o1}}}{\partial \mathtt{out}_{\mathtt{h1}}} = 0.74136 \times 0.18681 \times 0.40 = 0.0554$$

$$\frac{\partial \mathtt{E}_{\mathtt{o2}}}{\partial \mathtt{out}_{\mathtt{h1}}} = -0.01905$$

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

- Plugging them together.



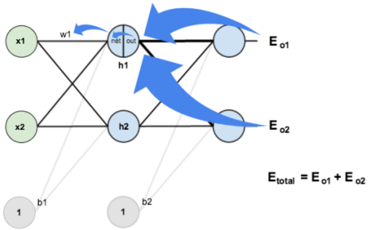
$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial out_{h1}} = 0.74136 \times 0.18681 \times 0.40 = 0.0554$$

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

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

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

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



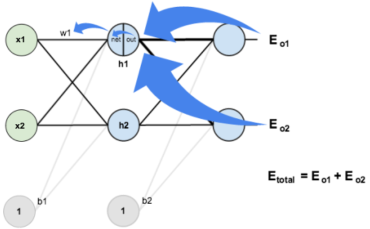
$$\frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{w}_1} = \frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{out}_{h1}} \times \frac{\partial \mathtt{out}_{h1}}{\partial \mathtt{net}_{h1}} \times \frac{\partial \mathtt{net}_{h1}}{\partial \mathtt{w}_1}$$

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

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

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

- And then $\frac{\partial \mathtt{net}_{\mathtt{h1}}}{\partial \mathtt{w}_1}$.



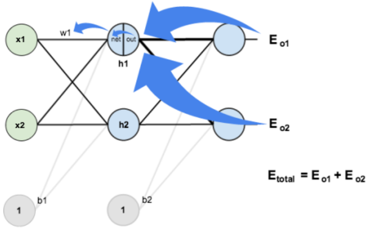
$$\frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{w}_1} = \frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{out}_{\mathtt{h1}}} \times \frac{\partial \mathtt{out}_{\mathtt{h1}}}{\partial \mathtt{net}_{\mathtt{h1}}} \times \frac{\partial \mathtt{net}_{\mathtt{h1}}}{\partial \mathtt{w}_1}$$

$$\mathtt{net}_{\mathtt{h1}} = \mathtt{w}_1\mathtt{x}_1 + \mathtt{w}_2\mathtt{x}_2 + \mathtt{b}_1$$

$$\frac{\partial \mathtt{net}_{\mathtt{h1}}}{\partial \mathtt{w}_1} = \mathtt{x}_1 = 0.05$$

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

- Putting it all together.

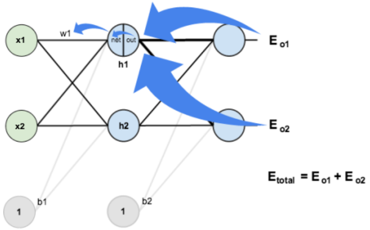


$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1}$$

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

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

- We can now update $\mathtt{w}_1$.
- Repeating this for $\mathtt{w}_2$, $\mathtt{w}_3$, and $\mathtt{w}_4$.



$$\mathtt{w}_1^{(\mathtt{next})} = \mathtt{w}_1 - \eta \times \frac{\partial \mathtt{E}_{\mathtt{total}}}{\partial \mathtt{w}_1} = 0.15 - 0.5 \times 0.00043 = 0.14978$$

$$\mathtt{w}_2^{(\mathtt{next})} = 0.19956$$

$$\mathtt{w}_3^{(\mathtt{next})} = 0.24975$$

$$\mathtt{w}_4^{(\mathtt{next})} = 0.2995$$

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?