CSCI E-89C Deep Reinforcement Learning

Harvard Extension School

Dmitry Kurochkin

Spring 2020
Lecture 10

Contents

Contents

Quiz 9

Question 1

4 / 4 pts

Deep Q-Network refers to Reinforcement Learning with approximation of

(A) state-value function via Linear Regression

(B) action-value function via Deep Neural Network

(C) action-value function via Random Forest

(D) none of the above

Correct!

- B
- D
- C
- A

Quiz 9

Question 2

4 / 4 pts

Suppose we have a MDP with $\mathcal{S} = \{s_A, s_B, s_C\}$.

The state-value function is being approximated by

$$v_\pi(s) \approx \hat{v}(s, \mathbf{w}) \doteq w_1 \cdot \mathbf{1}_{(s=s_A)} + w_2 \cdot \mathbf{1}_{(s=s_B)} + w_3 \cdot \mathbf{1}_{(s=s_C)},$$

where $\mathbf{w} = (w_1, w_2, w_3)^T$ are *weights*.

Then in TD(λ) with $\lambda \in (0, 1]$,

$$\begin{aligned} \mathbf{z}_{-1} &\doteq (0,0,0)^T, \\ \mathbf{z}_t &\doteq \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{v}(S_t, \mathbf{w}_t) \quad \text{for } t \geq 0, \\ \mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\left[R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)\right]\mathbf{z}_t \quad \text{for } t \geq 0, \end{aligned}$$

only one of the weights is updated on each time step.

- True
- Correct! False

Quiz 9

Question 3

4 / 4 pts

One of the advantages of the Stochastic Gradient Descent (i.e. mini-batch size is 1) is that the calculations can be efficiently parallelized.

- True
- False

Correct!

Quiz 9

Question 4

4 / 4 pts

Let $G_1, G_2, \ldots, G_T$ denote returns (cumulative discounted rewards) that are generated under policy π.

Suppose each state $s \in \mathcal{S}$ of the Markov Decision Process can be represented by a vector of d features: $\mathbf{x}(s) = (x_1(s), x_2(s), \ldots, x_d(s))^T$.

If we build a Linear Model on the following training set:

$\langle \mathbf{x}(S_1), G_1 \rangle, \langle \mathbf{x}(S_2), G_2 \rangle, \langle \mathbf{x}(S_3), G_3 \rangle, \ldots, \langle \mathbf{x}(S_T), G_T \rangle$

with $\mathbf{x}(S_t)$ being an input and return G_t being an output (one can also include observations from multiple episodes into the training set in order to increase accuracy),

then the output of the Linear Model for input $\mathbf{x}(s)$ will be an estimate of

(A) state-value function $v_\pi(s)$

(B) action-value function $q_\pi(s, a)$

(C) state-value $v_*(s)$ that corresponds to the optimal policy π_*

(D) action-value $q_*(s, a)$ that corresponds to the optimal policy π_*

(E) none of the above

Please select:

Correct!

A

Quiz 9

Question 5

4 / 4 pts

Let $G_1, G_2, \dots, G_T$ denote returns (cumulative discounted rewards) that are generated under policy π.

Suppose each state $s \in \mathcal{S}$ of the Markov Decision Process can be represented by a vector of d features: $\mathbf{x}(s) = (x_1(s), x_2(s), \dots, x_d(s))^T$.

The set of admissible actions in each state $s \in \mathcal{S}$ is $\mathcal{A}(s) = \{1, 2, 3, 4\}$.

If we build a Linear Model on the following training set:

$\langle(\mathbf{x}(S_1), A_1), G_1\rangle, \langle(\mathbf{x}(S_2), A_2), G_2\rangle, \langle(\mathbf{x}(S_3), A_3), G_3\rangle, \dots$

with $(\mathbf{x}(S_t), A_t)$ being an input and return G_t being an output (one can also include observations from multiple episodes into the training set in order to increase accuracy),

then the output of the Linear Model for input $(\mathbf{x}(s), a)$ will be an estimate of

(A) state-value function $v_\pi(s)$

(B) action-value function $q_\pi(s, a)$

(C) state-value $v_*(s)$ that corresponds to the optimal policy π_*

(D) action-value $q_*(s, a)$ that corresponds to the optimal policy π_*

(E) none of the above

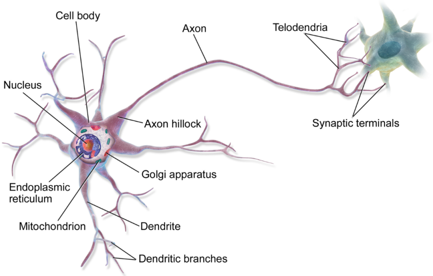
Please select:

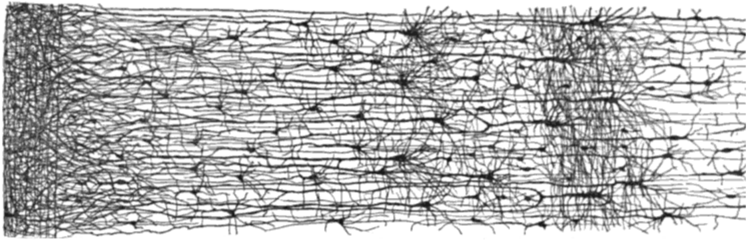
Correct!

B

Contents

Biological Neurons





Contents

Example: Artificial NN with 2 inputs and 1 output

Let's consider a *Deep Neural Network* which has two real-valued inputs (denoted by x_1 and x_2), one hidden layer that consists of two neurons (u_1 and u_2) with ReLU activation functions, and one output $\hat{y}$ with the ReLU activation function.

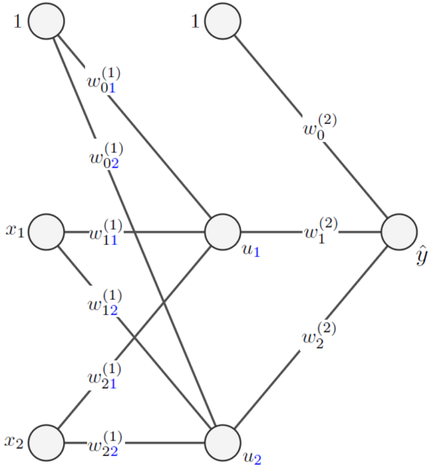
The explicit representation of this network is

input layer	hidden layer	output layer
x_1 x_2	$u_1 = f(w_{01}^{(1)} + w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2)$ $u_2 = f(w_{02}^{(1)} + w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2)$	$\hat{y} = f(w_0^{(2)} + w_1^{(2)} u_1 + w_2^{(2)} u_2)$

Here, $f(x)$ denotes the rectified linear unit (ReLU) defined as follows:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases}$$

Example: Artificial NN with 2 inputs and 1 output (cont.)



Artificial Neural Network

Neural Network (NN) with

n inputs,
M outputs and
1 hidden layer with H neurons is defined as:

$$\hat{\boldsymbol{y}} = f^{(2)}(f^{(1)}(\boldsymbol{x})),$$

where
$\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ are inputs,
$\hat{\boldsymbol{y}} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_M)^T \in \mathbb{R}^M$ are outputs,
$f^{(1)} : \mathbb{R}^n \mapsto \mathbb{R}^H$ and $f^{(2)} : \mathbb{R}^H \mapsto \mathbb{R}^M$,
where H is the number of *neurons* in the hidden layer.

Regression with NNs

NN

$$\hat{\boldsymbol{y}} = f^{(2)}(\underbrace{f^{(1)}(\boldsymbol{x})}_{\doteq \boldsymbol{u}})$$

with

- $f^{(2)} : \mathbb{R}^H \mapsto \mathbb{R}$, i.e. $M = 1$,

  is used for *regression*.

Regression with NNs

Keras:

```
import keras
from keras import models
from keras import layers

# number of inputs
n = 900

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(n,)))
model.add(layers.Dense(1, activation='relu'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 16)                14416
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 14,433
Trainable params: 14,433
Non-trainable params: 0
_________________________________________________________________
```

Classification with NNs

NN

$$\hat{\boldsymbol{y}} = f^{(2)}(\underbrace{f^{(1)}(\boldsymbol{x})}_{\doteq \boldsymbol{u}})$$

with

- $f_m^{(2)}(\boldsymbol{u}) \geq 0$ for all $m \in \{1, 2, \dots, M\}$ and $\boldsymbol{u} \in \mathbb{R}^H$ and $\sum_{m=1}^{M} f_m^{(2)}(\boldsymbol{u}) = 1$ for all $\boldsymbol{u} \in \mathbb{R}^H$

  is used for *classification*.

Classification with NNs

Keras:

```
import keras
from keras import models
from keras import layers

# number of inputs
n = 900

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(n,)))
model.add(layers.Dense(2, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 16)                14416
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 34
=================================================================
Total params: 14,450
Trainable params: 14,450
Non-trainable params: 0
_________________________________________________________________
```

Activation Functions

NN

$$\hat{\boldsymbol{y}} = f^{(2)}(\underbrace{f^{(1)}(\boldsymbol{x})}_{\doteq \boldsymbol{u}}).$$

What $f^{(1)}$ and $f^{(2)}$ should use?

Activation Functions

NN

$$\hat{\boldsymbol{y}} = f^{(2)}(\underbrace{f^{(1)}(\boldsymbol{x})}_{\doteq \boldsymbol{u}}).$$

What $f^{(1)}$ and $f^{(2)}$ should use?

Let

- $u_h \doteq f_h^{(1)}(\boldsymbol{x}) = \sigma_h^{(1)}\left(\sum_{j=0}^{n} w_{jh}^{(1)} x_j\right)$, where we define $x_0 \doteq 1$.
- $\hat{y}_m \doteq f_m^{(2)}(\boldsymbol{u}) = \sigma_m^{(2)}\left(\sum_{h=0}^{H} w_{hm}^{(2)} u_h\right)$, where we define $u_0 \doteq 1$.

Activation Functions

The NN

$$\hat{\boldsymbol{y}} = f^{(2)}(\underbrace{f^{(1)}(\boldsymbol{x})}_{\doteq \boldsymbol{u}})$$

becomes

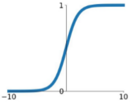
$$\hat{y}_m = \sigma_m^{(2)}\left(\sum_{h=0}^{H} w_{hm}^{(2)} \underbrace{\sigma_h^{(1)}\left(\sum_{j=0}^{n} w_{jh}^{(1)} x_j\right)}_{\doteq u_h}\right).$$

Activation Functions

What $\sigma_h^{(1)}$ and $\sigma_m^{(2)}$ should use?

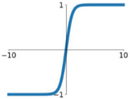
Sigmoid

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



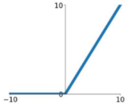
tanh

$\tanh(x)$



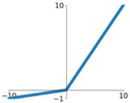
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

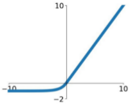


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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



Contents

Loss Function

Assume we observe

$$(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}),$$

where
$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ are n inputs and
$\boldsymbol{y} = (y_1, y_2, \ldots, y_M)^T$ are M outputs.

The prediction obtained with a supervised model (e.g., Neural Network) is denoted by

$$\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w}) = \hat{\boldsymbol{y}}(\boldsymbol{x}^{(i)}; \boldsymbol{w}),$$

where $\boldsymbol{w} = (w_1, w_2, \ldots, w_k)^T$ are parameters of the model.

Loss Function

Assume we observe

$$(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}),$$

where
$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ are n inputs and
$\boldsymbol{y} = (y_1, y_2, \ldots, y_M)^T$ are M outputs.

The prediction obtained with a supervised model (e.g., Neural Network) is denoted by

$$\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w}) = \hat{\boldsymbol{y}}(\boldsymbol{x}^{(i)}; \boldsymbol{w}),$$

where $\boldsymbol{w} = (w_1, w_2, \ldots, w_k)^T$ are parameters of the model.

The loss incurred from incorrect prediction of $\boldsymbol{y}^{(i)}$ is

$$L^{(i)}(\boldsymbol{w}) = L(\underbrace{\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w})}_{\text{prediction}}, \underbrace{\boldsymbol{y}^{(i)}}_{\text{observed}}).$$

Loss Function: Prediction

Squared Error

What Loss can we choose in case of prediction? One of the most common loss functions is Squared Error.

If the output is scalar (i.e. $M = 1$), the Squared Error Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = (\hat{y}^{(i)} - y^{(i)})^2.$$

Loss Function: Prediction

Squared Error

What Loss can we choose in case of prediction? One of the most common loss functions is Squared Error.

If the output is scalar (i.e. $M = 1$), the Squared Error Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = (\hat{y}^{(i)} - y^{(i)})^2.$$

If the output is vector-valued (i.e. $M > 1$), the Squared Error Loss is then similarly defined as:

$$L^{(i)}(\boldsymbol{w}) = |\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}|^2 = \sum_{j=1}^{M} (\hat{y}_j^{(i)} - y_j^{(i)})^2.$$

Loss Function: Prediction

Absolute Error

An alternative to the Squared Error is Absolute Error.

If the output is scalar (i.e. $M = 1$), the Absolute Error Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = |\hat{y}^{(i)} - y^{(i)}|.$$

Loss Function: Prediction

Absolute Error

An alternative to the Squared Error is Absolute Error.

If the output is scalar (i.e. $M = 1$), the Absolute Error Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = |\hat{y}^{(i)} - y^{(i)}|.$$

If the output is vector-valued (i.e. $M > 1$), the Absolute Error Loss is then defined as:

$$L^{(i)}(\boldsymbol{w}) = |\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}| = \left(\sum_{j=1}^{M} (\hat{y}_j^{(i)} - y_j^{(i)})^2 \right)^{\frac{1}{2}}.$$

Loss Function: Classification

Cross-Entropy

What Loss can we choose in case of classification? One of the most common loss functions is Cross-Entropy.

If there are two classes (i.e. $M = 2$), the (Binary) Cross-Entropy Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = -\left(y_1^{(i)} \ln \hat{y}_1^{(i)} + y_2^{(i)} \ln \hat{y}_2^{(i)}\right).$$

Loss Function: Classification

Cross-Entropy

What Loss can we choose in case of classification? One of the most common loss functions is Cross-Entropy.

If there are two classes (i.e. $M = 2$), the (Binary) Cross-Entropy Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = -\left(y_1^{(i)} \ln \hat{y}_1^{(i)} + y_2^{(i)} \ln \hat{y}_2^{(i)}\right).$$

If there are multiple classes (i.e. $M > 2$), the (Multi-Class) Cross-Entropy Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = -\sum_{j=1}^{M} y_j^{(i)} \ln \hat{y}_j^{(i)}.$$

Objective (Cost) Function

Suppose we want to train a supervised model (e.g., Neural Network) using a set of observations:

$$(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), (\boldsymbol{x}_3, \boldsymbol{y}_3), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)$$

then we define the objective (or cost) function as mean loss:

$$J(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} L^{(i)}(\boldsymbol{w}),$$

where

$$L^{(i)}(\boldsymbol{w}) = L(\underbrace{\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w})}_{\text{prediction}}, \underbrace{\boldsymbol{y}^{(i)}}_{\text{observed}})$$

is the loss associated with a single observation i as defined earlier.

Objective (Cost) Function

The list of the most common cost functions:

- Mean Squared Error:

$$J(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{M}(\hat{y}_j^{(i)} - y_j^{(i)})^2$$

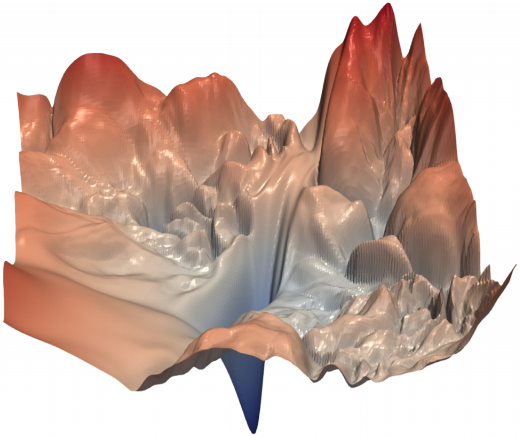
- Mean Absolute Error:

$$J(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{M}(\hat{y}_j^{(i)} - y_j^{(i)})^2\right)^{\frac{1}{2}}$$

- Cross-Entropy:

$$J(\boldsymbol{w}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{M} y_j^{(i)} \ln \hat{y}_j^{(i)}$$

Cost Function Landscape



Source: *Visualizing the Loss Landscape of Neural Nets* by Hao Li et al., 2017

Contents

Keras: Classification Example

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(layers.Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_36 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_37 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

```
nepochs = 35
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

history = model.fit(X_train, y_train,
          batch_size=128, epochs=nepochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

Keras: Loss Functions

More loss functions available in Keras:

mean_squared_error
mean_absolute_error
mean_absolute_percentage_error
mean_squared_logarithmic_error
squared_hinge
hinge
categorical_hinge
logcosh
categorical_crossentropy
sparse_categorical_crossentropy
binary_crossentropy
kullback_leibler_divergence
poisson
cosine_proximity

The complete list can be found at `https://keras.io/losses/`

Contents

Gradient Descent

The Gradient Descent (GD) update of the weights using learning rate α:

$$\mathbf{w} := \mathbf{w} - \alpha \nabla J(\boldsymbol{w}),$$

where

$$\begin{aligned} J(\boldsymbol{w}) =& \frac{1}{m} \sum_{i=1}^{m} L^{(i)}(\boldsymbol{w}) \\ =& \frac{1}{m} \sum_{i=1}^{m} L(\underbrace{\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w})}_{\text{prediction}}, \underbrace{\boldsymbol{y}^{(i)}}_{\text{observed}}). \end{aligned}$$

Contents

Stochastic Gradient Descent

The Stochastic Gradient Descent (SGD) update of the weights using learning rate α:

$$\mathbf{w} := \mathbf{w} - \alpha \nabla L^{(i)}(\boldsymbol{w}),$$

i.e. we assume

$$\begin{aligned} J(\boldsymbol{w}) \approx & L^{(i)}(\boldsymbol{w}) \\ = & L(\underbrace{\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w})}_{\text{prediction}}, \underbrace{\boldsymbol{y}^{(i)}}_{\text{observed}}), \end{aligned}$$

where $L^{(i)}(\boldsymbol{w})$ is based on a single observation.

Contents

Mini-batch Gradient Descent

The Mini-batch Gradient Descent update of the weights using learning rate α:

$$\boldsymbol{w} := \boldsymbol{w} - \alpha \nabla \frac{1}{s} \sum_{i=1}^{s} L^{(i)}(\boldsymbol{w}),$$

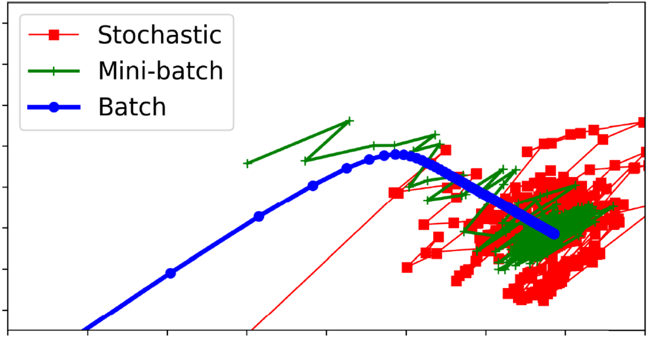
where $s < m$ is mini-batch size, i.e. we assume

$$\begin{aligned} J(\boldsymbol{w}) \approx& \frac{1}{s} \sum_{i=1}^{s} L^{(i)}(\boldsymbol{w}) \\ =& \frac{1}{s} \sum_{i=1}^{s} L(\underbrace{\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w})}_{\text{prediction}}, \underbrace{\boldsymbol{y}^{(i)}}_{\text{observed}}) \end{aligned}$$

where $L^{(i)}(\boldsymbol{w})$ is based on one observation.

Mini-batch Gradient Descent

Example: Path in (w_1, w_2) plane:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Contents

Forward Propagation

Let's again consider a Neural Network (NN) with

n inputs,
M outputs and
1 hidden layer with H neurons.

The Forward Propagation is then

$$\hat{y}_m = \sigma_m^{(2)}\left(\sum_{h=0}^{H} w_{hm}^{(2)} \underbrace{\sigma_h^{(1)}\left(\sum_{j=0}^{n} w_{jh}^{(1)} x_j\right)}_{\doteq u_h}\right).$$

Contents

Backpropagation

Given an observation $(\boldsymbol{x}, \boldsymbol{y})$, assume we want to minimize the Mean Squared Error Loss

$$L(\boldsymbol{w}) = \sum_{m=1}^{M} \left(\hat{y}_m - y_m\right)^2 ,$$

where

$$\hat{y}_m = \sigma_m^{(2)} \left(\sum_{h=0}^{H} w_{hm}^{(2)} \underbrace{\sigma_h^{(1)} \left(\sum_{j=0}^{n} w_{jh}^{(1)} x_j \right)}_{\doteq u_h} \right) .$$

Then need to compute $\frac{L(\boldsymbol{w})}{\partial w_{hm}^{(2)}}$ and $\frac{L(\boldsymbol{w})}{\partial w_{jh}^{(1)}}$.

But we know derivatives of $\sigma_m^{(2)}$ and $\sigma_h^{(1)}$ exactly!

Also, we have $\sum_{j=0}^{n} w_{jh}^{(1)} x_j$, u_h, $\sum_{h=0}^{H} w_{hm}^{(2)} u_h$, and $\hat{y}_m$ computed during forward propagation!

Contents

Example of Forward Propagation/Backpropagation

Let's consider the follwoing Neural Network:

input layer	hidden layer	output layer
x_1	$u_1 = f(\underbrace{w_{01}^{(1)} + w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2}_{z_1^{(1)}})$	$\hat{y} = f(\underbrace{w_0^{(2)} + w_1^{(2)} u_1 + w_2^{(2)} u_2}_{z^{(2)}})$
x_2	$u_2 = f(\underbrace{w_{02}^{(1)} + w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2}_{z_2^{(1)}})$	

Here, $f(x)$ denotes the activation function, for example, ReLU.

Example of Forward Propagation/Backpropagation

Let's consider the follwoing Neural Network:

input layer	hidden layer	output layer
x_1	$u_1 = f(\underbrace{w_{01}^{(1)} + w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2}_{z_1^{(1)}})$	$\hat{y} = f(\underbrace{w_0^{(2)} + w_1^{(2)} u_1 + w_2^{(2)} u_2}_{z^{(2)}})$
x_2	$u_2 = f(\underbrace{w_{02}^{(1)} + w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2}_{z_2^{(1)}})$	

Here, $f(x)$ denotes the activation function, for example, ReLU.

Forward Propagation: Given weights $\boldsymbol{w}$ and inputs x_1, x_2, compute

- $z_1^{(1)}$ and $z_2^{(1)}$
- u_1 and u_2
- $\hat{y}$

Example of Forward Propagation/Backpropagation

Backpropagation:

Given weights $\boldsymbol{w}$, inputs x_1, x_2, and $z_1^{(1)}$, $z_2^{(1)}$, u_1, u_2, $\hat{y}$, compute

- Error associated with the output layer:

$$\varepsilon^{(2)} \doteq \frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left[(\hat{y} - y)^2 \right] = 2(\hat{y} - y)$$

- Errors associated with the hidden layer:

$$\varepsilon_h^{(1)} \doteq \frac{\partial L}{\partial u_h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u_h} = \varepsilon^{(2)} f'(z^{(2)}) w_h^{(2)}, \quad h = 1, 2.$$

Example of Forward Propagation/Backpropagation

Computation of $\nabla L(\boldsymbol{w})$:

- Partial derivatives of the loss function with respect to weights in the output layer:

$$\frac{\partial L}{\partial w_h^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_h^{(2)}} = \varepsilon^{(2)} \frac{\partial}{\partial w_h^{(2)}} \left[f(\underbrace{w_0^{(2)} + w_1^{(2)} u_1 + w_2^{(2)} u_2}_{z^{(2)}}) \right] = \varepsilon^{(2)} f'(z^{(2)}) u_h,$$

where $h = 0, 1, 2$.

- Partial derivatives of the loss function with respect to weights in the hidden layer:

$$\frac{\partial L}{\partial w_{jh}^{(1)}} = \frac{\partial L}{\partial u_h} \frac{\partial u_h}{\partial w_{jh}^{(1)}} = \varepsilon_h^{(1)} \frac{\partial}{\partial w_{jh}^{(1)}} \left[f(\underbrace{w_{0h}^{(1)} + w_{1h}^{(1)} x_1 + w_{2h}^{(1)} x_2}_{z_h{}^{(1)}}) \right] = \varepsilon_h^{(1)} f'(z_h{}^{(1)}) x_j,$$

for each $j = 0, 1, 2$ and $h = 1, 2$. Here, we define $x_0 \doteq 1$.

Example of Forward Propagation/Backpropagation

The Stochastic Gradient Descent (SGD) update of the weights using learning rate α:

$$\mathbf{w} := \mathbf{w} - \alpha \nabla L,$$

where $\nabla L \doteq \Big(\underbrace{\frac{\partial L}{\partial w_{01}^{(1)}}, \frac{\partial L}{\partial w_{11}^{(1)}}, \frac{\partial L}{\partial w_{21}^{(1)}}, \frac{\partial L}{\partial w_{02}^{(1)}}, \frac{\partial L}{\partial w_{12}^{(1)}}, \frac{\partial L}{\partial w_{22}^{(1)}}}_{\text{hidden layer}}, \underbrace{\frac{\partial L}{\partial w_{0}^{(2)}}, \frac{\partial L}{\partial w_{1}^{(2)}}, \frac{\partial L}{\partial w_{2}^{(2)}}}_{\text{output layer}} \Big)^T.$

Therefore,

$$\begin{aligned}
\mathbf{w} &:= \mathbf{w} - \alpha \nabla L \\
&= \Big(\underbrace{w_{01}^{(1)}, w_{11}^{(1)}, w_{21}^{(1)}, w_{02}^{(1)}, w_{12}^{(1)}, w_{22}^{(1)}}_{\text{hidden layer}}, \underbrace{w_{0}^{(2)}, w_{1}^{(2)}, w_{2}^{(2)}}_{\text{output layer}} \Big)^T \\
&- \alpha \Big(\underbrace{\frac{\partial L}{\partial w_{01}^{(1)}}, \frac{\partial L}{\partial w_{11}^{(1)}}, \frac{\partial L}{\partial w_{21}^{(1)}}, \frac{\partial L}{\partial w_{02}^{(1)}}, \frac{\partial L}{\partial w_{12}^{(1)}}, \frac{\partial L}{\partial w_{22}^{(1)}}}_{\text{hidden layer}}, \underbrace{\frac{\partial L}{\partial w_{0}^{(2)}}, \frac{\partial L}{\partial w_{1}^{(2)}}, \frac{\partial L}{\partial w_{2}^{(2)}}}_{\text{output layer}} \Big)^T
\end{aligned}$$