バックプロパゲーション【逆誤差伝播法】(1)

前田利之（まえだ　としゆき）
aipr@e-chan.jp

阪南大学　経営情報学部

(2024.11.15)

バックプロパゲーションとは

- 前週では、重みパラメータの勾配を数値微分で求めた
- しかしこれは計算に
- →
  - 本講義では、数式ベースではなく による説明をおこなう

バックプロパゲーションとは

- 前週では、重みパラメータの勾配を数値微分で求めた
- しかしこれは計算に **時間がかかる**
- →
  - 本講義では、数式ベースではなく　　による説明をおこなう

バックプロパゲーションとは

- 前週では、重みパラメータの勾配を数値微分で求めた
- しかしこれは計算に **時間がかかる**
- → **バックプロパゲーション**

  - 本講義では、数式ベースではなく [ ] による説明をおこなう

バックプロパゲーションとは

- 前週では、重みパラメータの勾配を数値微分で求めた
- しかしこれは計算に **時間がかかる**
- → **バックプロパゲーション（逆誤差伝播法）**
  - 本講義では、数式ベースではなく [　　] による説明をおこなう

バックプロパゲーションとは

- 前週では、重みパラメータの勾配を数値微分で求めた
- しかしこれは計算に **時間がかかる**
- → **バックプロパゲーション（逆誤差伝播法）**
  - 本講義では、数式ベースではなく **計算グラフ** による説明をおこなう

計算グラフ

- 計算の [　　] をグラフで表わしたもの
- データ構造としてのグラフ＝複数の [　　] と [　　] で表現
  - ノード：処理やデータそのもの
  - エッジ：ノードの間を結ぶもの（直線）

計算グラフ

- 計算の 過程 をグラフで表わしたもの
- データ構造としてのグラフ＝複数の [　　] と [　　] で表現
  - ノード：処理やデータそのもの
  - エッジ：ノードの間を結ぶもの（直線）

計算グラフ

- 計算の **過程** をグラフで表わしたもの
- データ構造としてのグラフ＝複数の **ノード** と
  [　　] で表現
  - ノード：処理やデータそのもの
  - エッジ：ノードの間を結ぶもの（直線）

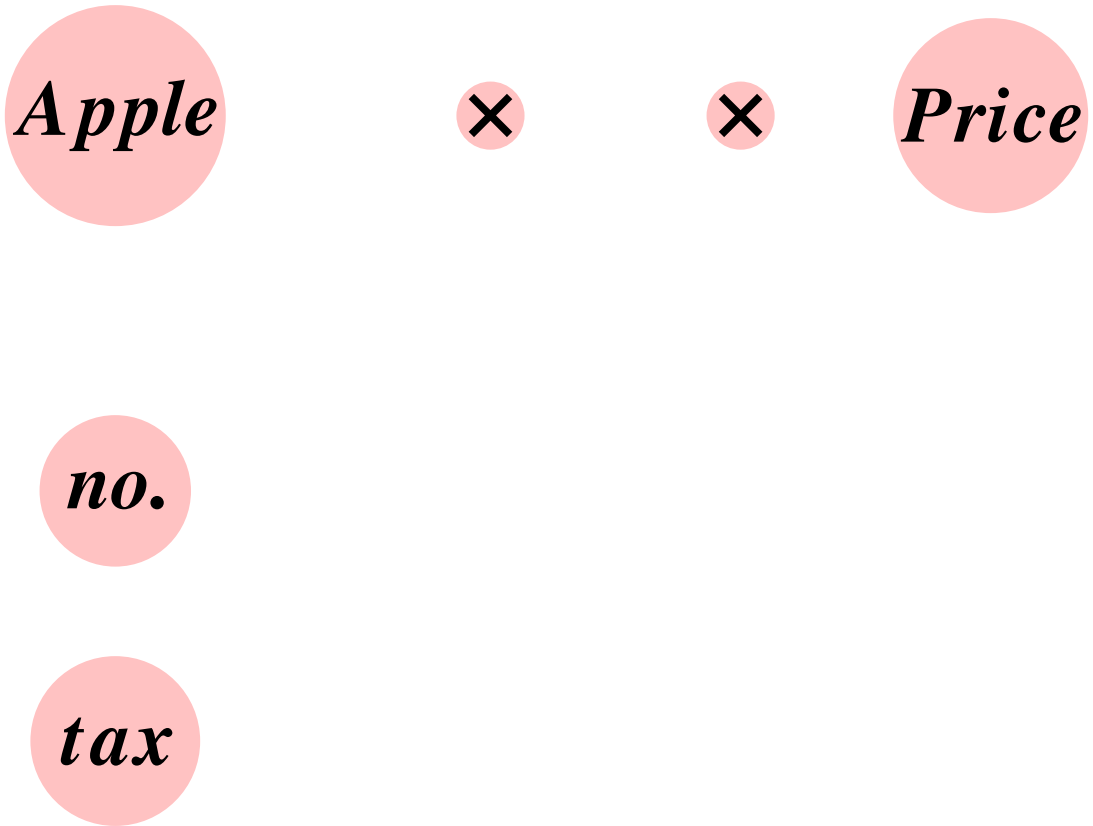
計算グラフ

- 計算の **過程** をグラフで表わしたもの
- データ構造としてのグラフ＝複数の **ノード** と **エッジ** で表現
  - ノード：処理やデータそのもの
  - エッジ：ノードの間を結ぶもの（直線）

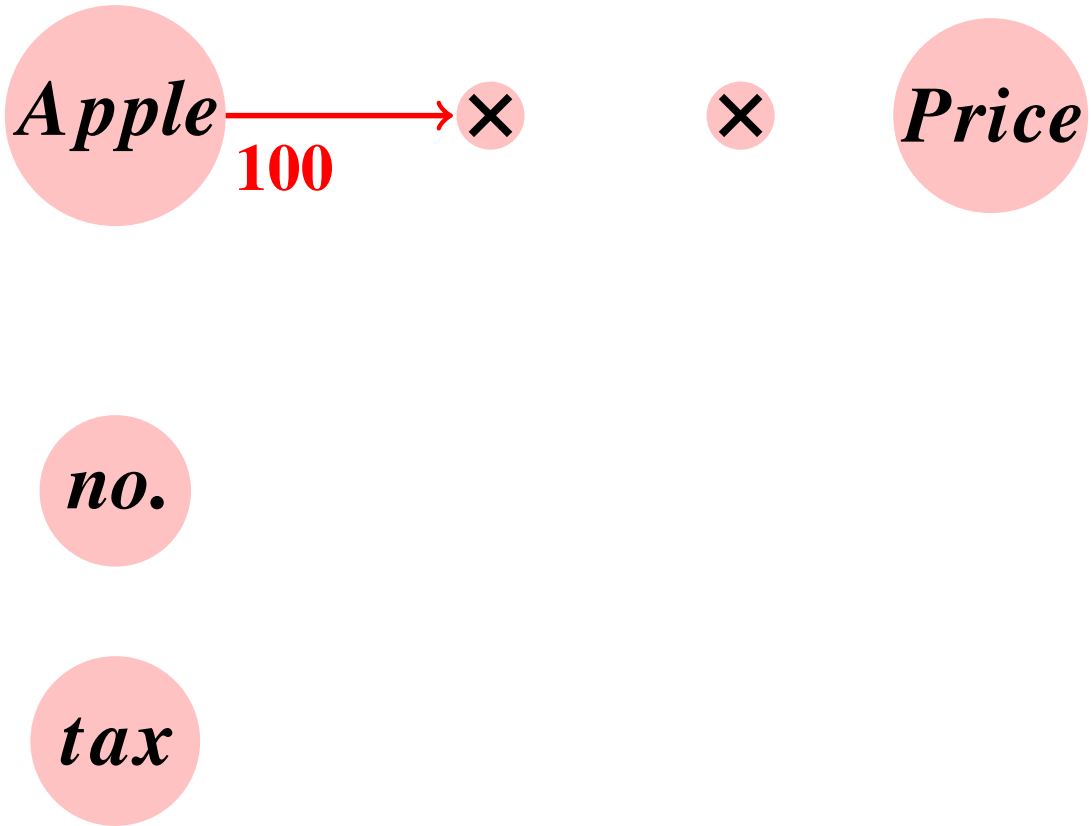
計算グラフで解く

- 問１：「１個100円のリンゴを2つ買いました。消費税が10%として支払い金額はいくらでしょう？」をグラフで解くことを考える

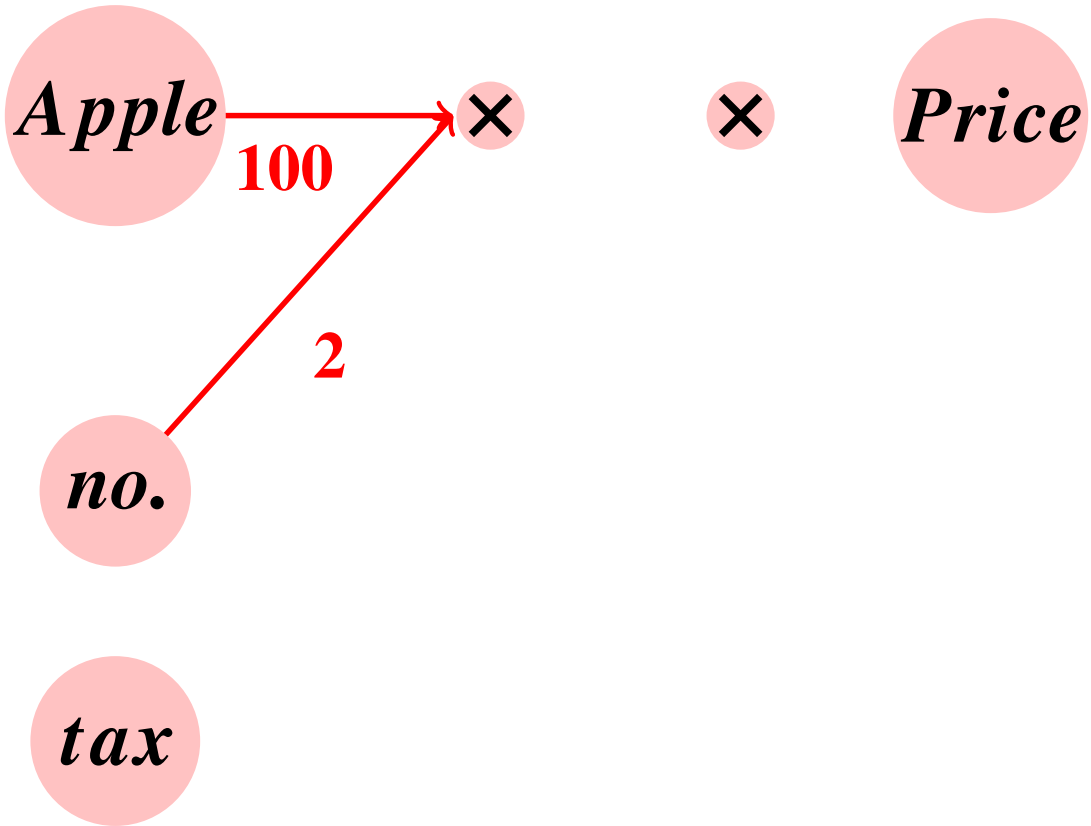
計算グラフで解く



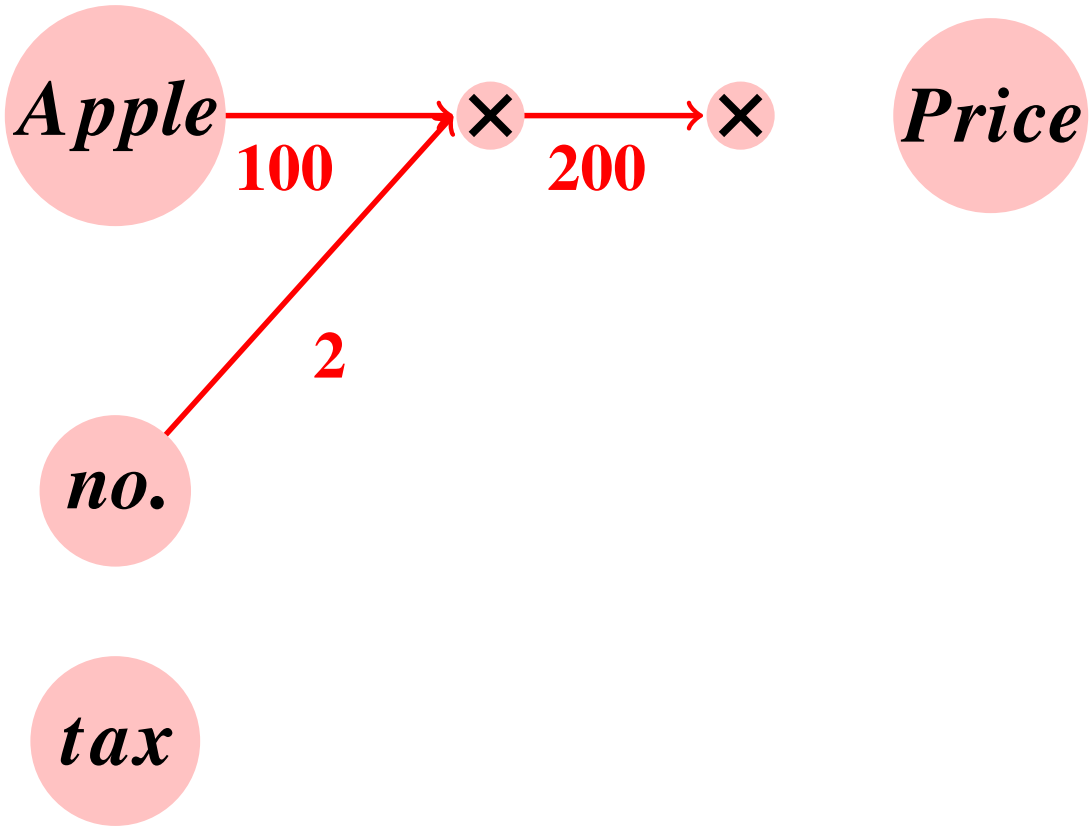
計算グラフで解く



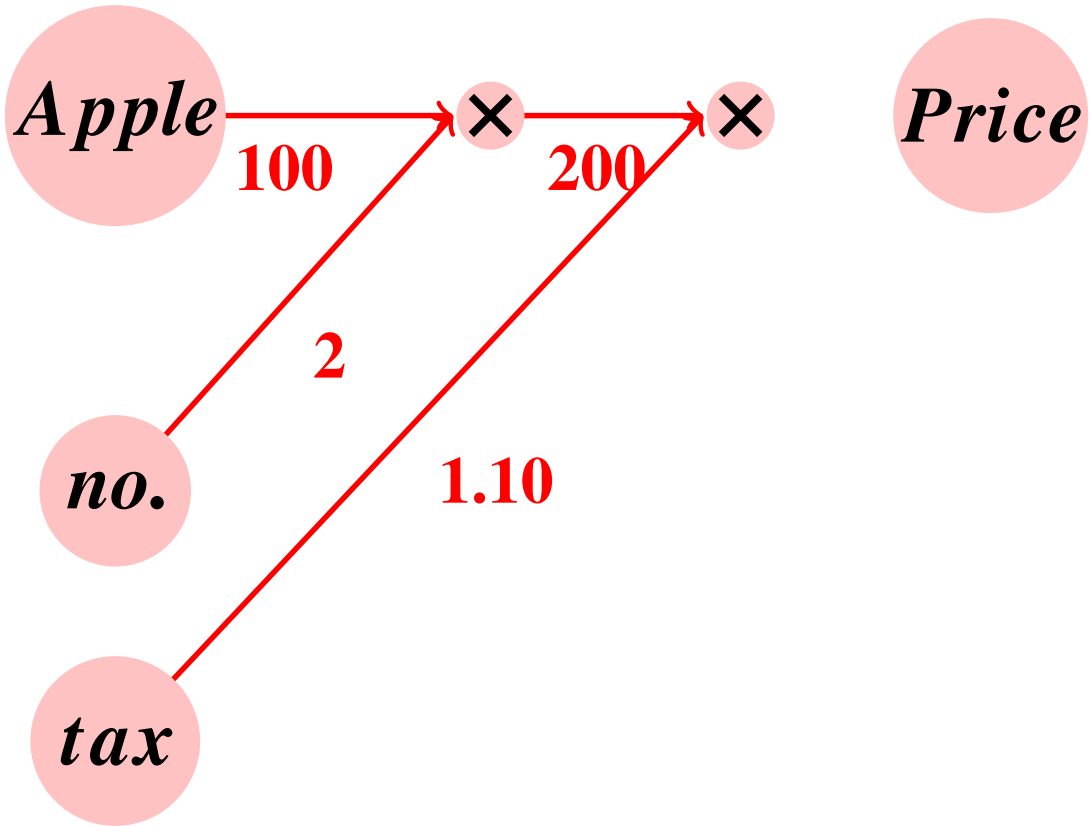
計算グラフで解く



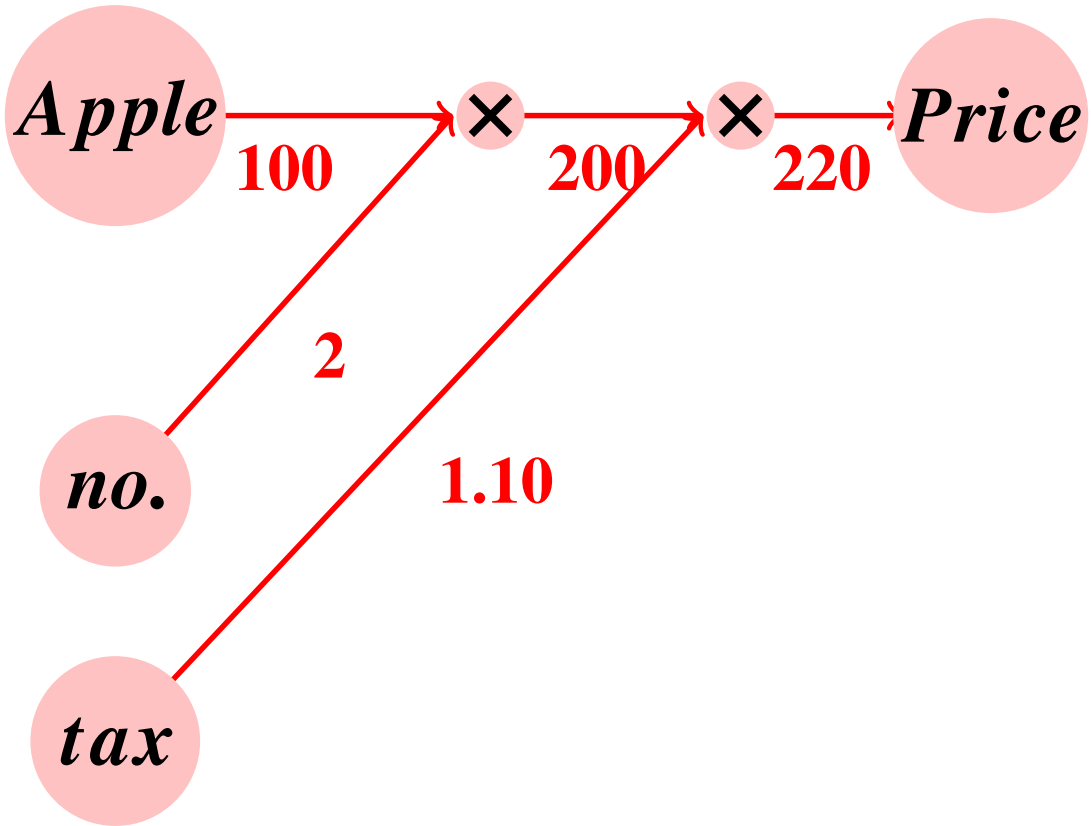
計算グラフで解く



計算グラフで解く



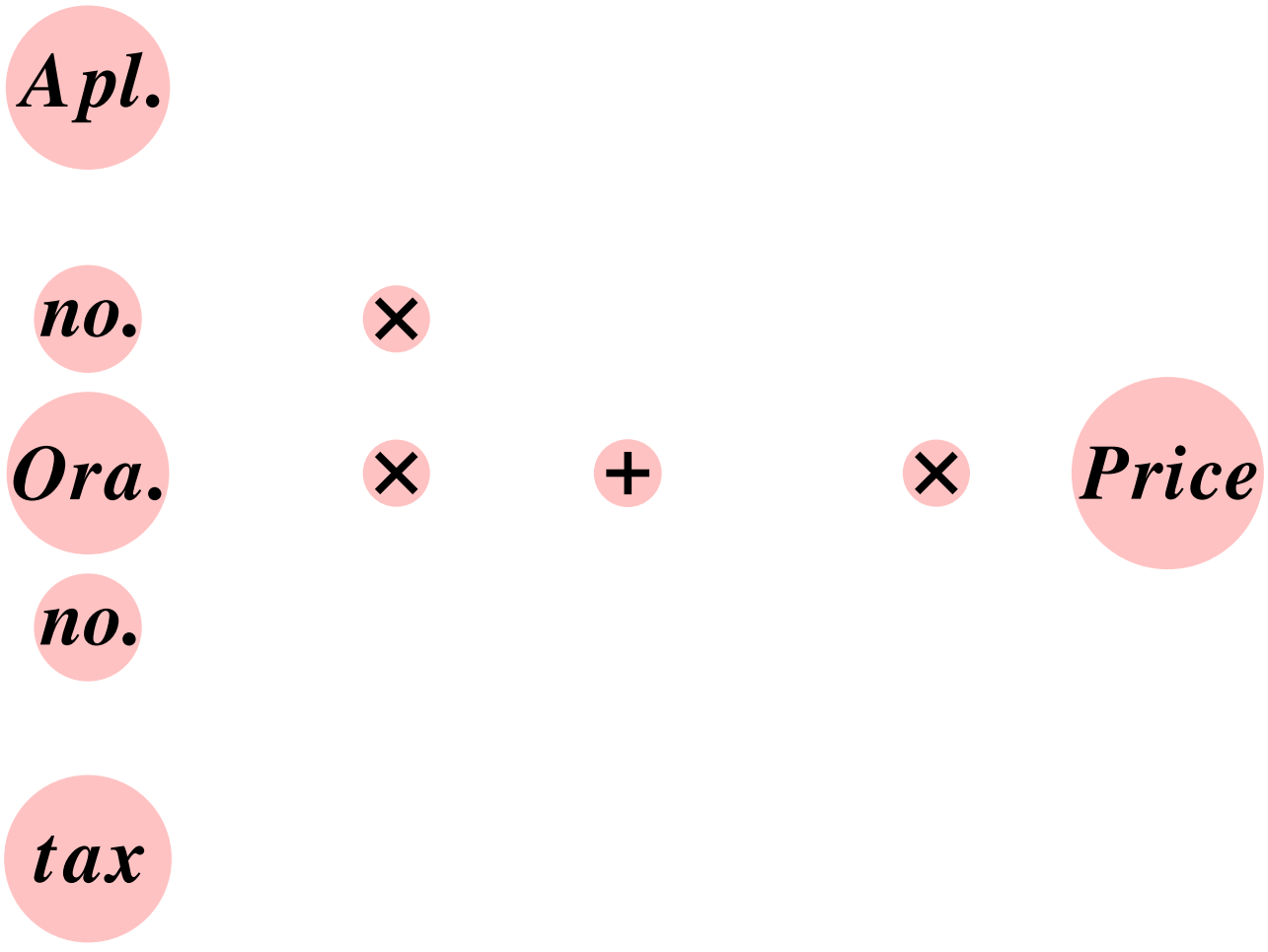
計算グラフで解く



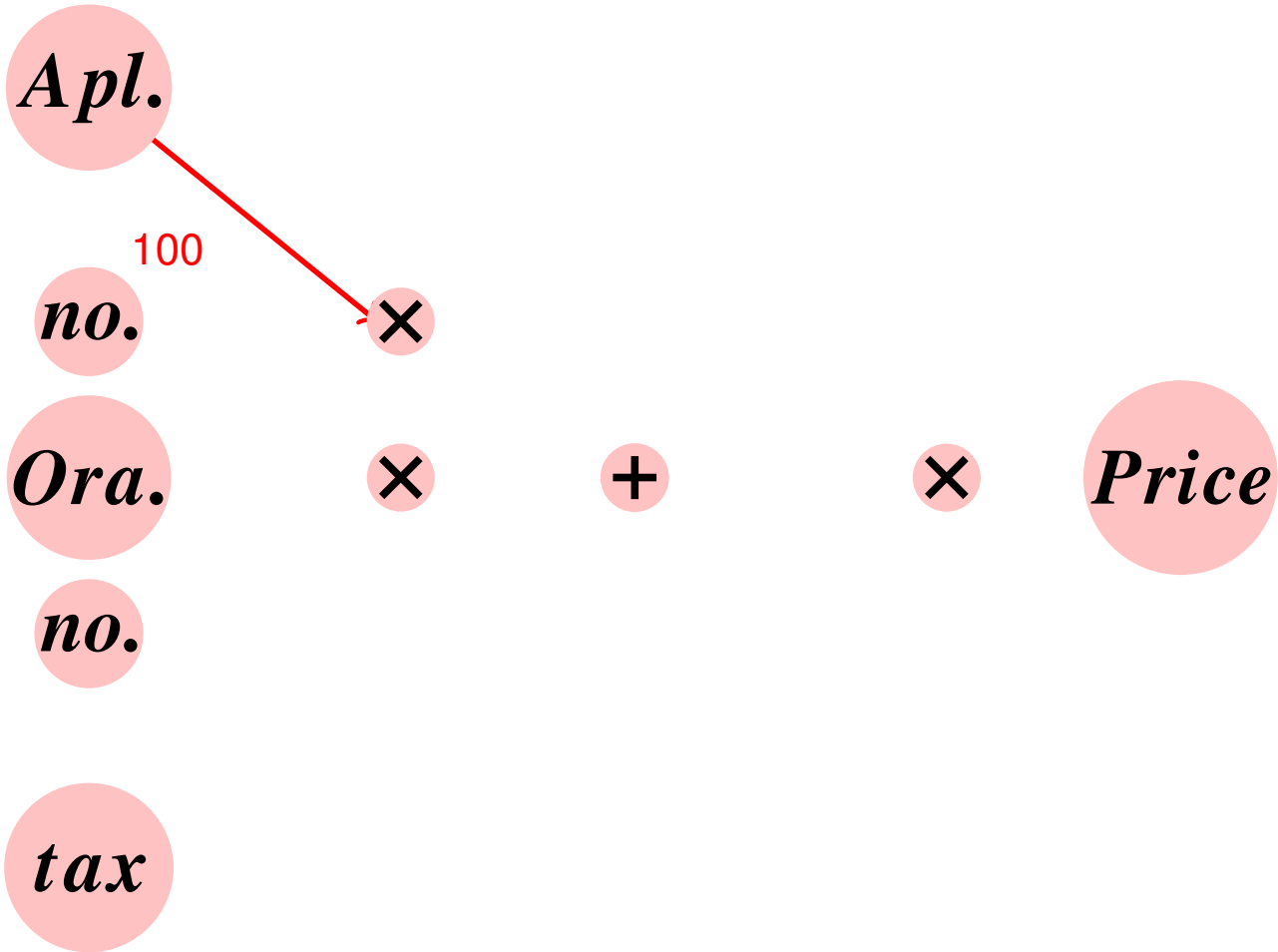
計算グラフで解く

- 問２：「１個100円のリンゴを２つ、１個150円のミカンを３個買いました。消費税が10%として支払い金額はいくらでしょう？」をグラフで解くことを考える

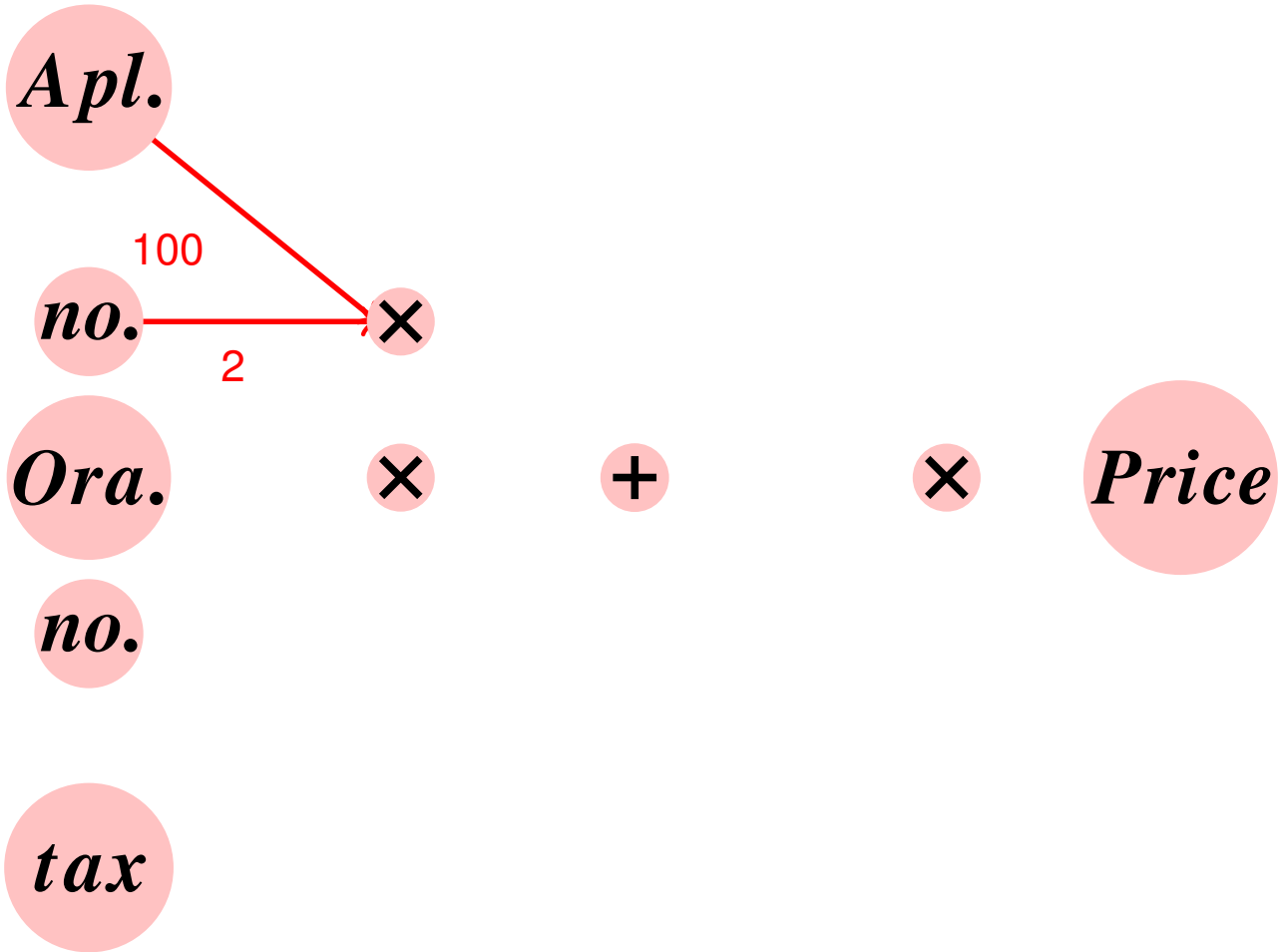
計算グラフで解く



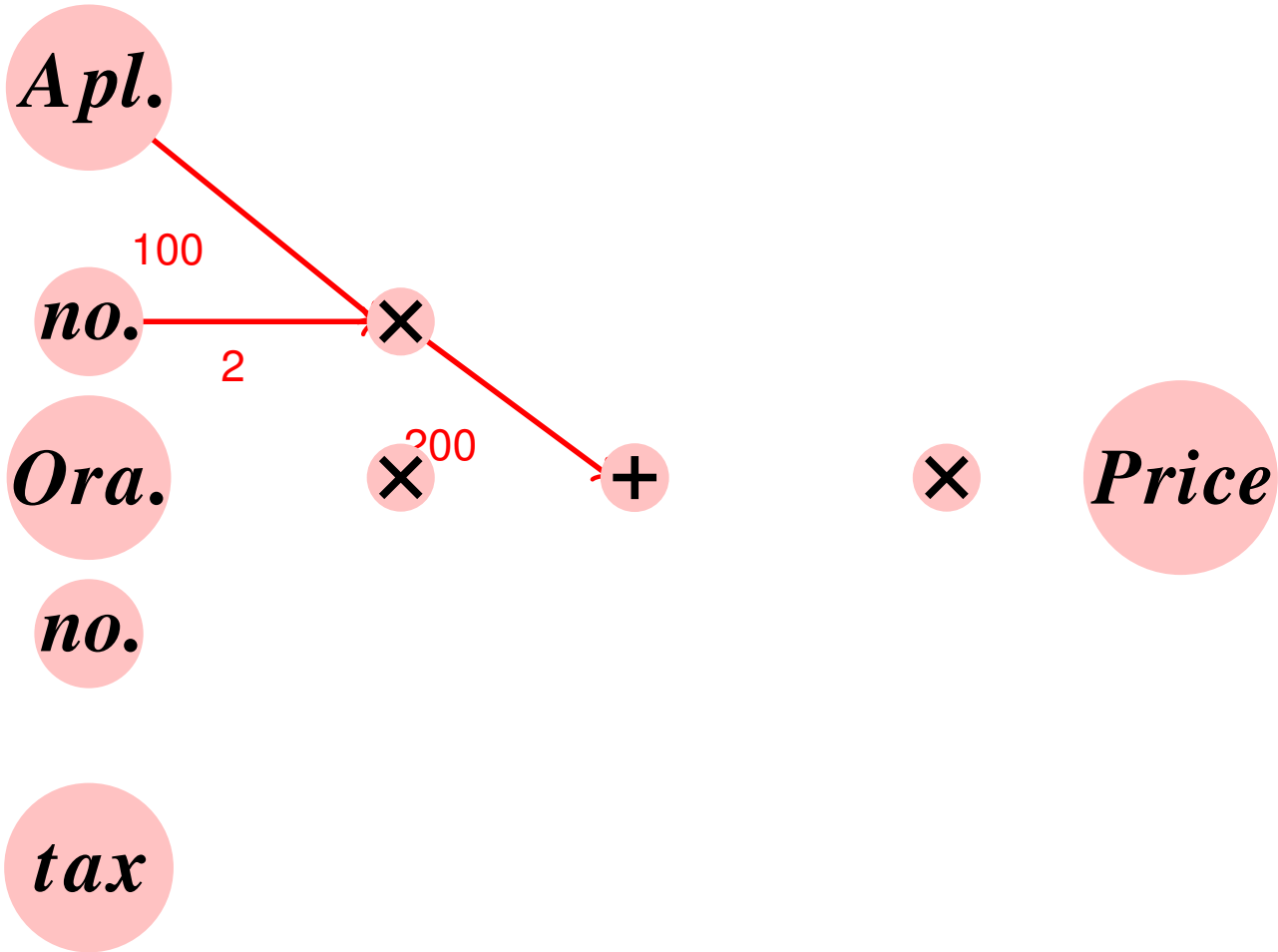
計算グラフで解く



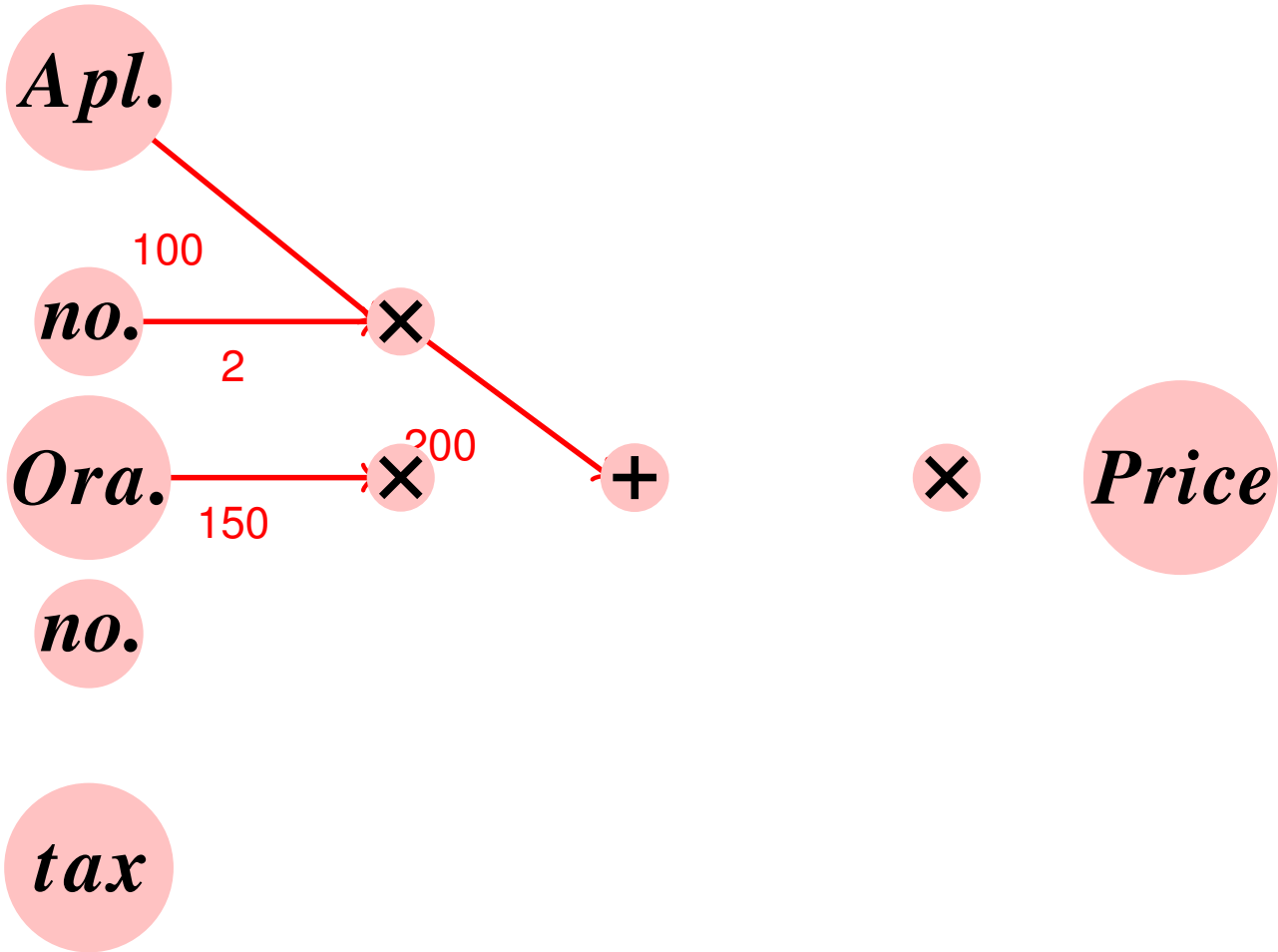
計算グラフで解く



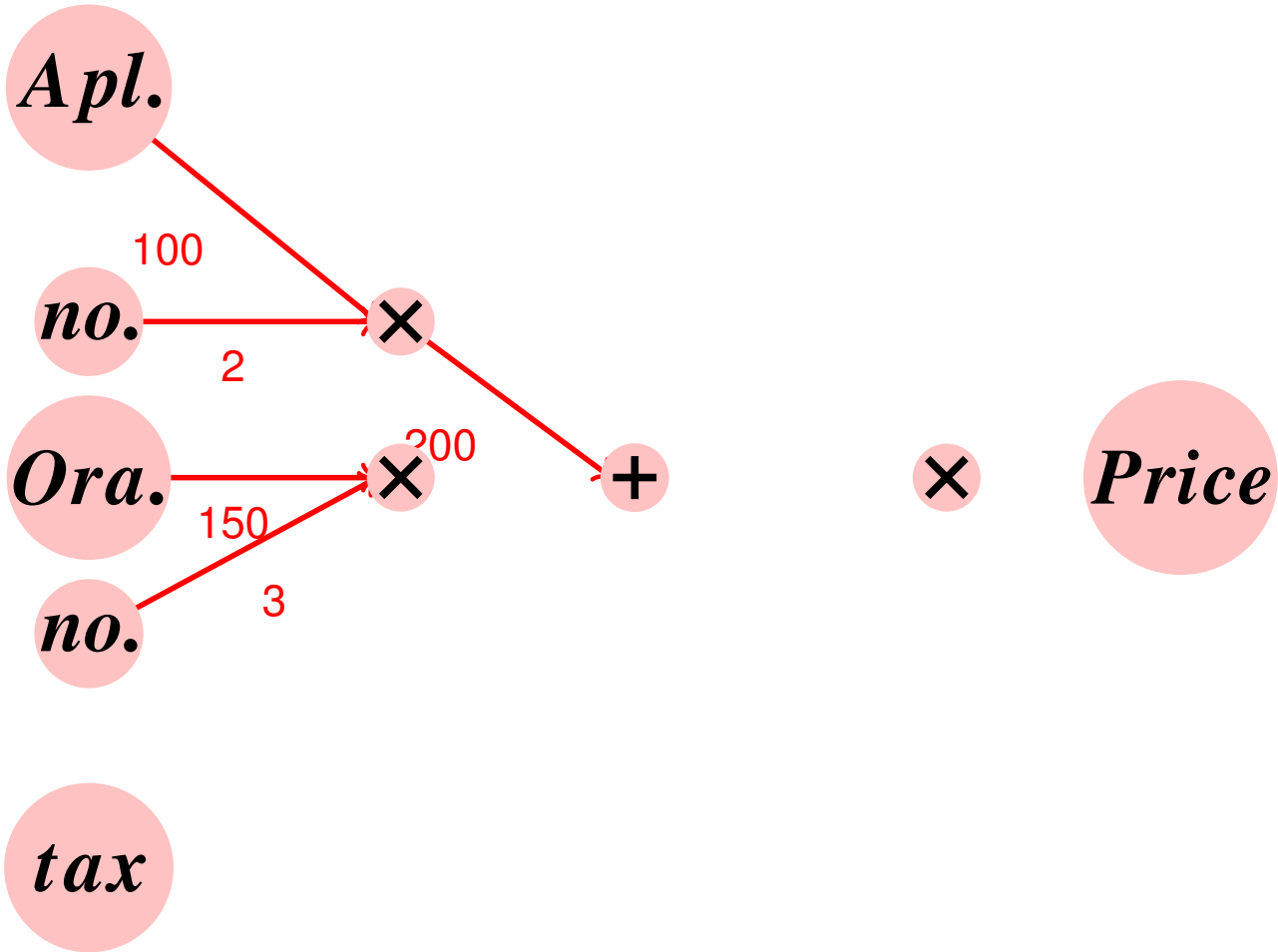
計算グラフで解く



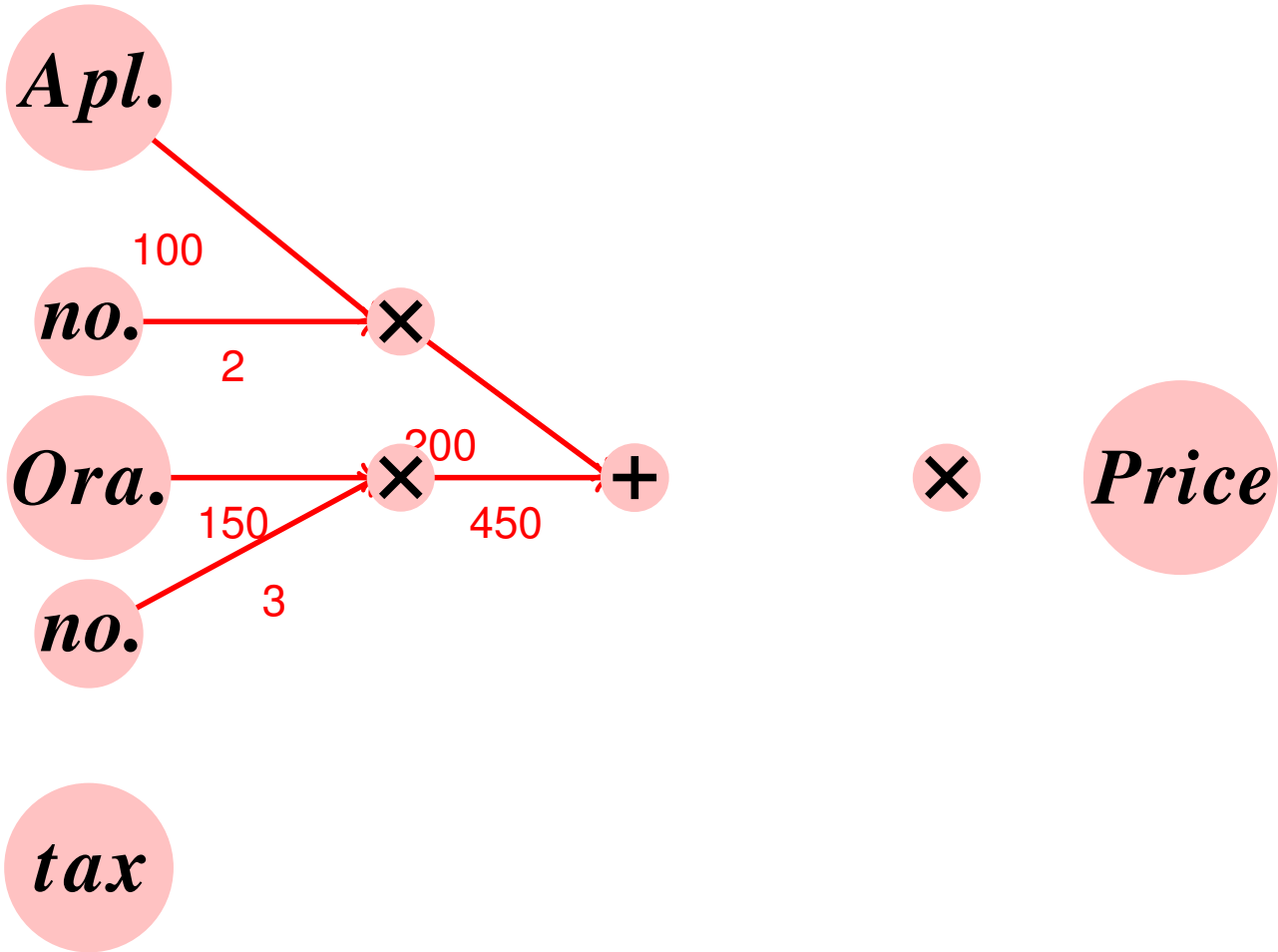
計算グラフで解く



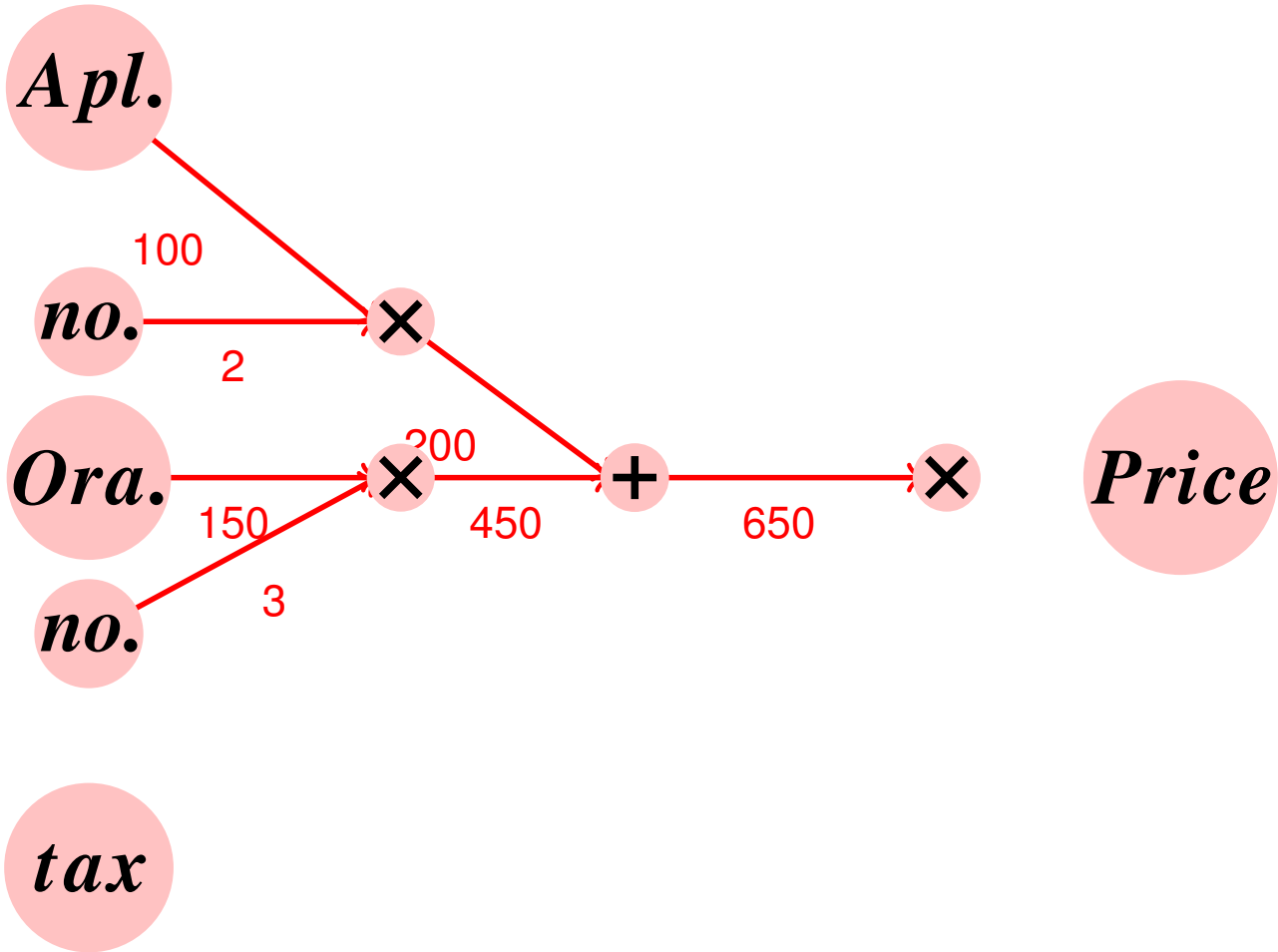
計算グラフで解く



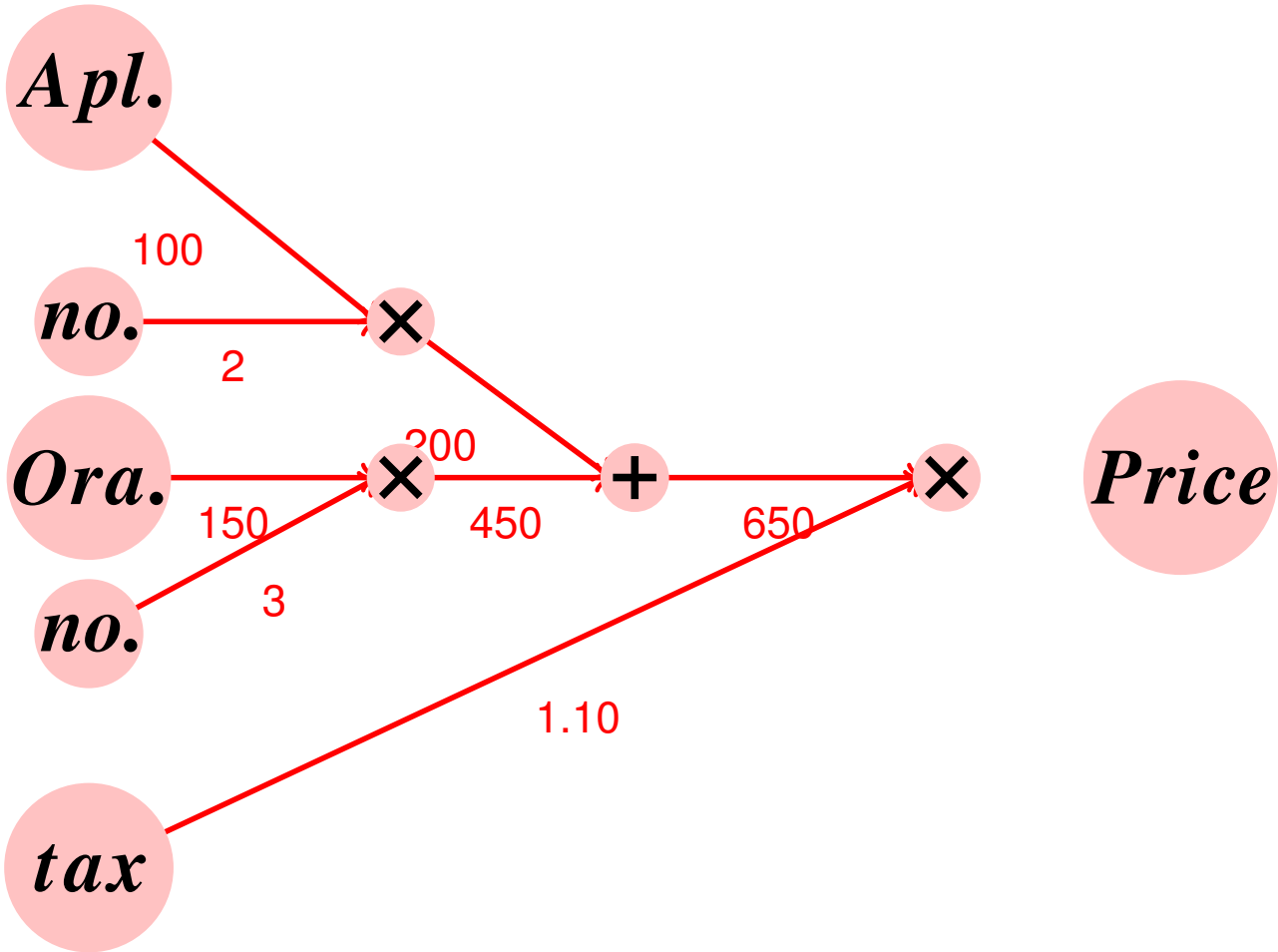
計算グラフで解く



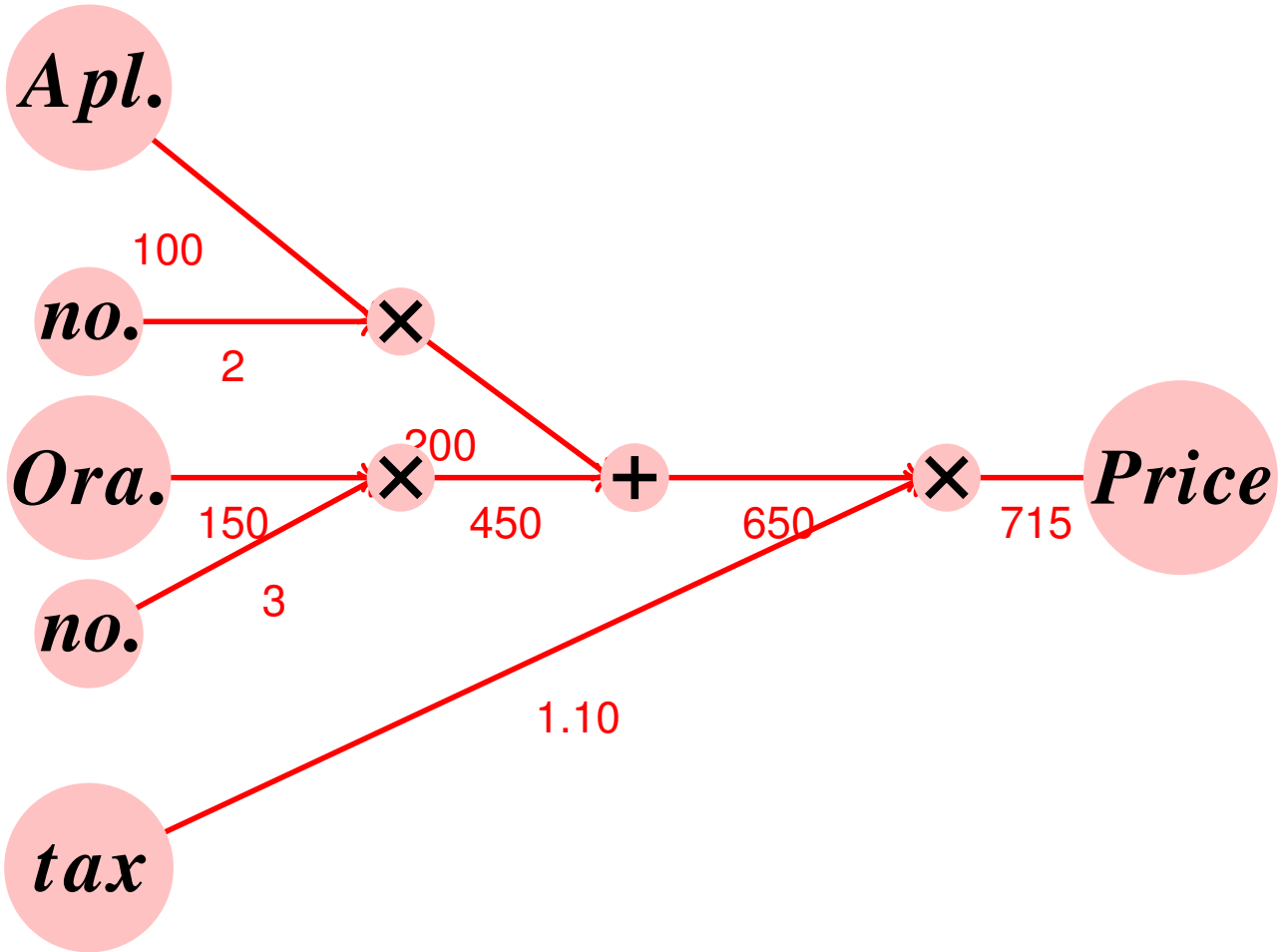
計算グラフで解く



計算グラフで解く



計算グラフで解く



計算グラフ

- 計算グラフで問題を解く、とは
  1. 計算グラフを作る
  2. 計算グラフ上で計算を左から右へ進める＝ [　　]
     (forward propagation)
- [　　]があれば [　　] もある

【今回の授業の主題】

計算グラフ

- 計算グラフで問題を解く、とは
  1. 計算グラフを作る
  2. 計算グラフ上で計算を左から右へ進める＝ 順伝搬
     (forward propagation)
- [　　]があれば [　　] もある
  【今回の授業の主題】

計算グラフ

- 計算グラフで問題を解く、とは
  1. 計算グラフを作る
  2. 計算グラフ上で計算を左から右へ進める＝ 順伝搬 (forward propagation)
- 順伝搬があれば [ ] もある
  【今回の授業の主題】

計算グラフ

- 計算グラフで問題を解く、とは
  1. 計算グラフを作る
  2. 計算グラフ上で計算を左から右へ進める＝ **順伝搬** (forward propagation)
- **順伝搬** があれば **逆伝搬** もある
  【今回の授業の主題】

局所的な計算

- 計算グラフの特徴：[　　]を伝搬し最終結果を得る
- [　　]：全体を意識せず、[　　]情報だけから次の結果を出す
  - 前の例の場合だと、リンゴの合計価格はミカンとは関係ない、というイメージ

局所的な計算

- 計算グラフの特徴：「局所的な計算」を伝搬し最終結果を得る
- ：全体を意識せず、情報だけから次の結果を出す
  - 前の例の場合だと、リンゴの合計価格はミカンとは関係ない、というイメージ

局所的な計算

- 計算グラフの特徴：「局所的な計算」を伝搬し最終結果を得る
- 局所的な計算：全体を意識せず、

  情報だけから次の結果を出す
  - 前の例の場合だと、リンゴの合計価格はミカンとは関係ない、というイメージ

局所的な計算

- 計算グラフの特徴：「局所的な計算」を伝搬し最終結果を得る
- 局所的な計算：全体を意識せず、自分に関係する情報だけから次の結果を出す
  - 前の例の場合だと、リンゴの合計価格はミカンとは関係ない、というイメージ

なぜ計算グラフ？

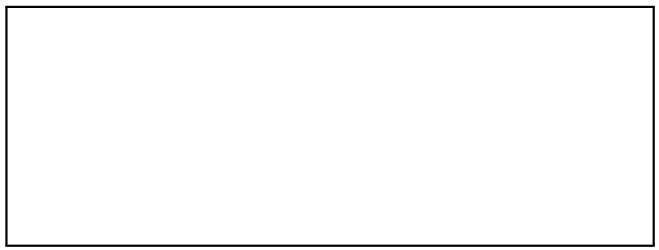
- 局所的な計算→問題を [　　] できる
- 逆伝搬によって [　　] を [　　] 良く計算できる
  - 前の問１でリンゴが値上がりした場合の支払い金額への影響を考える
  - ⇒ リンゴの値段に関する支払い金額の [　　]

なぜ計算グラフ？

- 局所的な計算→問題を 単純化 できる
- 逆伝搬によって [　　] を [　　] 良く計算できる
  - 前の問１でリンゴが値上がりした場合の支払い金額への影響を考える
  - ⇒ リンゴの値段に関する支払い金額の [　　]

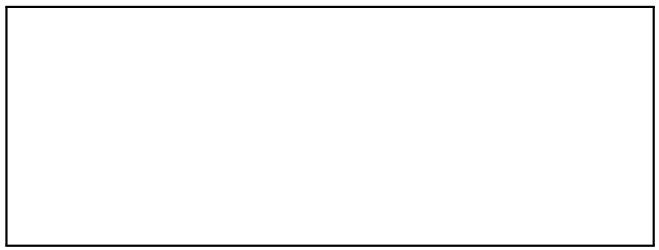
なぜ計算グラフ？

- 局所的な計算→問題を 単純化 できる
- 逆伝搬によって 微分 を [　　] 良く計算できる
  - 前の問１でリンゴが値上がりした場合の支払い金額への影響を考える
  - ⇒ リンゴの値段に関する支払い金額の



なぜ計算グラフ？

- 局所的な計算→問題を **単純化** できる
- 逆伝搬によって **微分** を **効率** 良く計算できる
  - 前の問１でリンゴが値上がりした場合の支払い金額への影響を考える
  - ⇒ リンゴの値段に関する支払い金額の

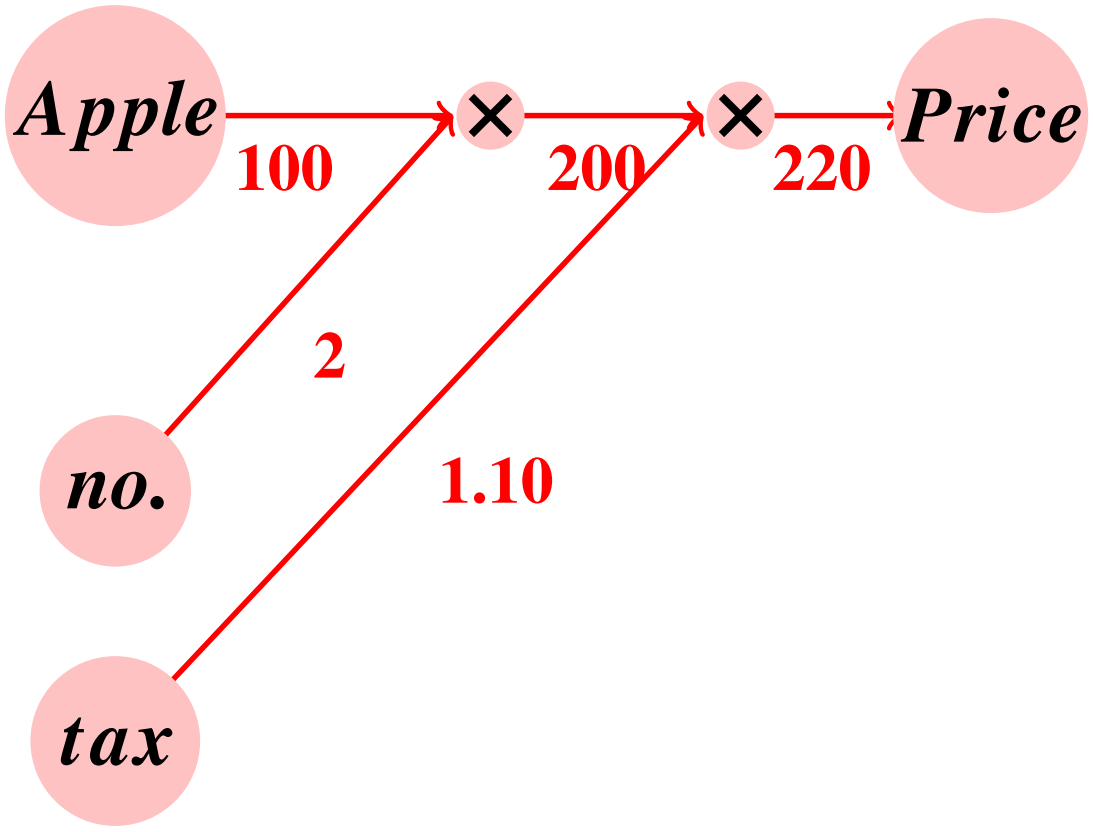


なぜ計算グラフ？

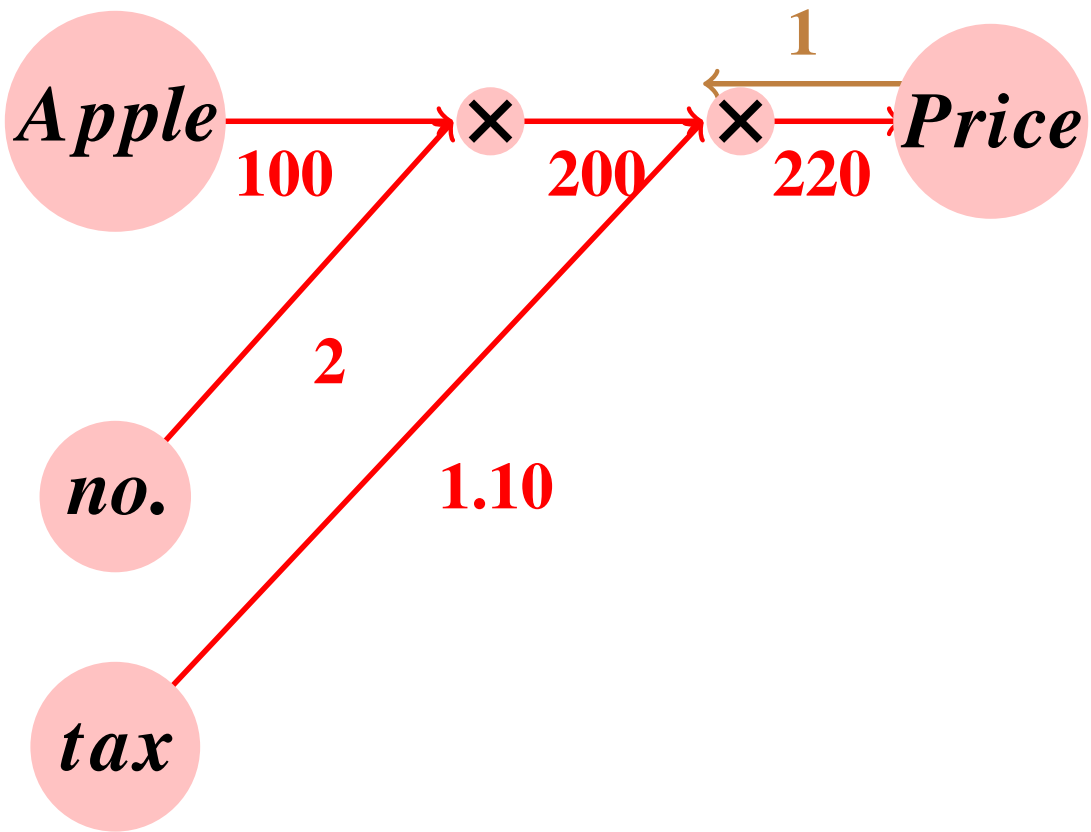
- 局所的な計算→問題を **単純化** できる
- 逆伝搬によって **微分** を **効率** 良く計算できる
  - 前の問１でリンゴが値上がりした場合の支払い金額への影響を考える
  - ⇒ リンゴの値段に関する支払い金額の

（偏）微分 $\dfrac{\partial L}{\partial x}$

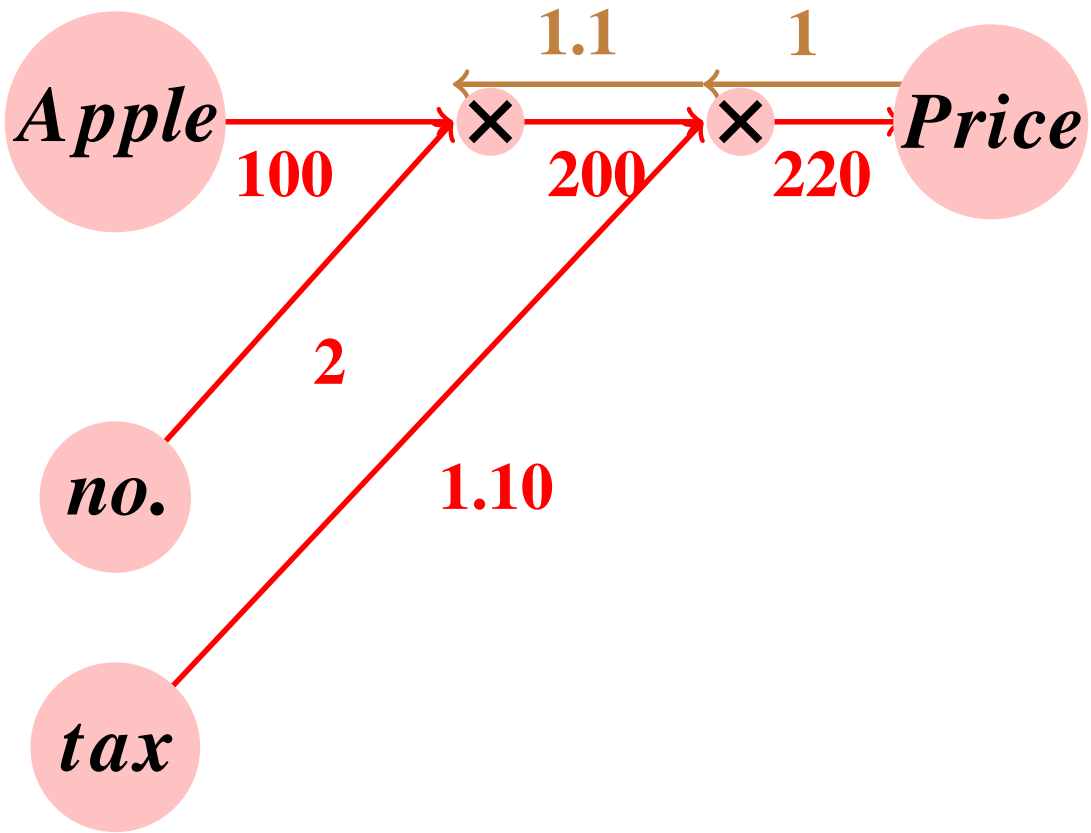
逆伝搬による微分値の伝達



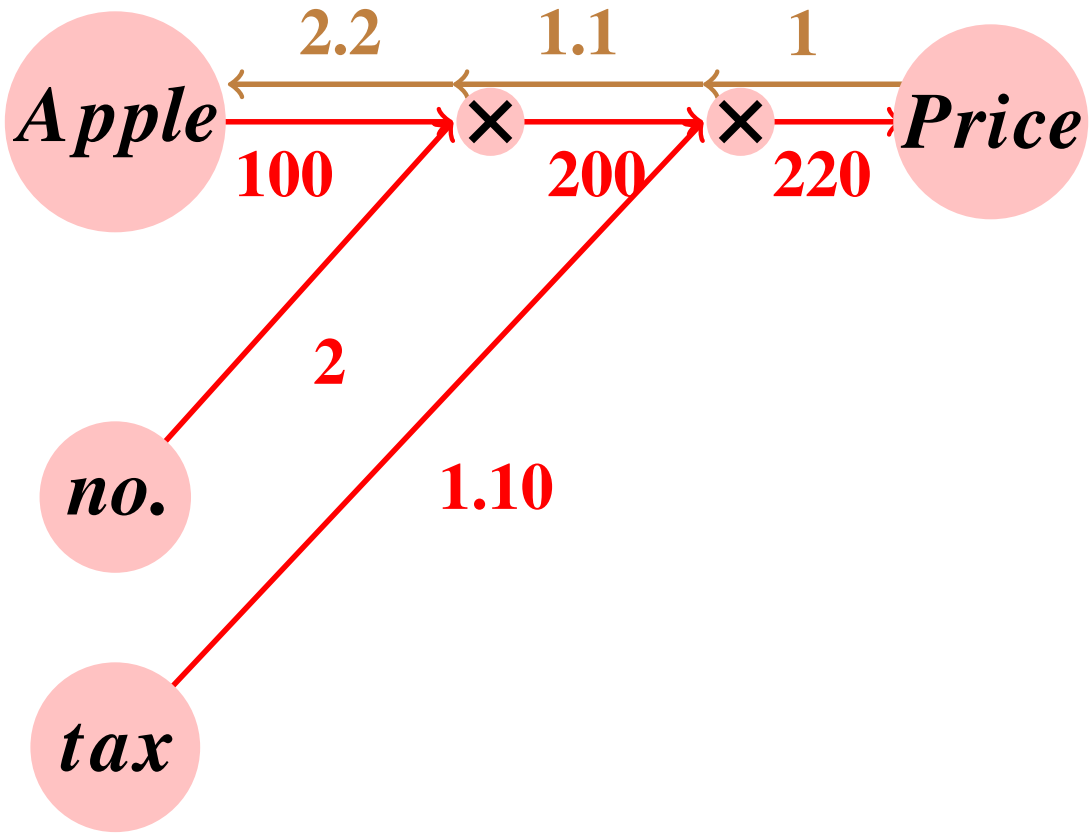
逆伝搬による微分値の伝達



逆伝搬による微分値の伝達



逆伝搬による微分値の伝達



逆伝搬による微分値の伝達

- 局所的な微分を右から左へ [　　] 伝達している
- 前頁の例だと「$1 \to 1.1 \to 2.2$」
- ⇒ リンゴが１円値上りすると最終的な支払いが [　　]、という意味
- （当然、消費税についても出来るが略）

逆伝搬による微分値の伝達

- 局所的な微分を右から左へ (逆向きに) 伝達している
- 前頁の例だと「$1 \to 1.1 \to 2.2$」
- $\Rightarrow$ リンゴが１円値上りすると最終的な支払いが [　　] 、という意味
- （当然、消費税についても出来るが略）

逆伝搬による微分値の伝達

- 局所的な微分を右から左へ (逆向きに) 伝達している
- 前頁の例だと「$1 \to 1.1 \to 2.2$」
- $\Rightarrow$ リンゴが１円値上りすると最終的な支払いが 2.2 円増える、という意味
- （当然、消費税についても出来るが略）

連鎖律

- 逆方向の伝搬では「局所的な微分」を逆向きに伝達する
- この「局所的な微分」を伝達する原理＝
- 合成関数の微分の性質（後述）

連鎖律

- 逆方向の伝搬では「局所的な微分」を逆向きに伝達する
- この「局所的な微分」を伝達する原理＝**連鎖律**
- 合成関数の微分の性質（後述）

連鎖律

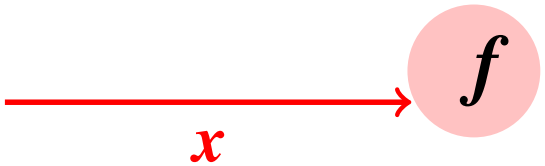
☆計算グラフの逆伝搬

連鎖律

☆計算グラフの逆伝搬

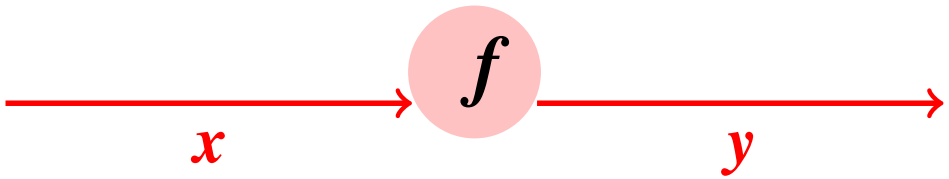
連鎖律

☆計算グラフの逆伝搬



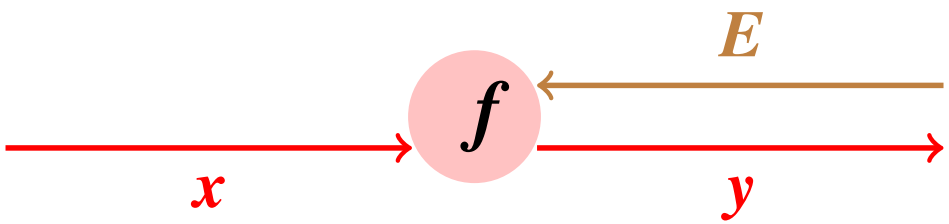
連鎖律

☆計算グラフの逆伝搬



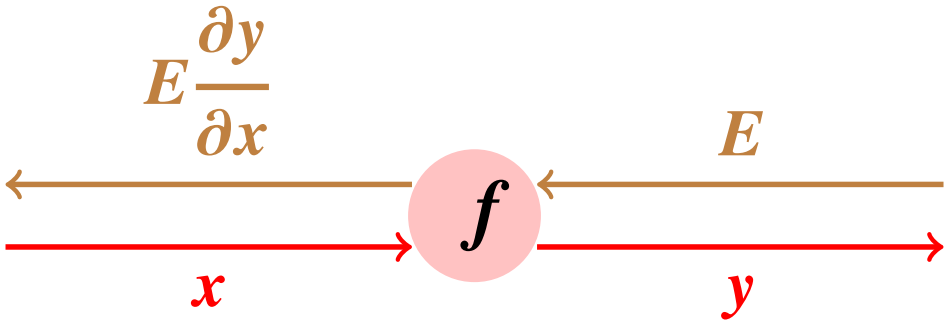
連鎖律

☆計算グラフの逆伝搬



連鎖律

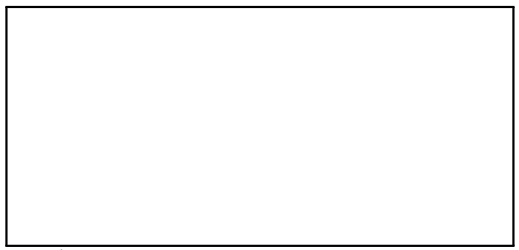
☆計算グラフの逆伝搬



合成関数の微分

- 連鎖律とは合成関数の微分の性質
  - ある関数が合成関数で表される場合、その微分は合成関数を構成するそれぞれの関数の微分の [　　] によって表すことが出来る
  - [　　]
  - （分数の通分のイメージ（厳密には違うもの））
- たとえば $z = (x + y)^2$ は $z = t^2, t = x + y$ であるので、$\dfrac{\partial z}{\partial x} = \dfrac{\partial z}{\partial t}\dfrac{\partial t}{\partial x} = 2t \cdot 1 =$ [　　]

合成関数の微分

- 連鎖律とは合成関数の微分の性質
  - ある関数が合成関数で表される場合、その微分は合成関数を構成するそれぞれの関数の微分の **積** によって表すことが出来る
  - 
  - （分数の通分のイメージ（厳密には違うもの））
- たとえば $z = (x + y)^2$ は $z = t^2, t = x + y$ であるので、$\dfrac{\partial z}{\partial x} = \dfrac{\partial z}{\partial t}\dfrac{\partial t}{\partial x} = 2t \cdot 1 =$ 

合成関数の微分

- 連鎖律とは合成関数の微分の性質
  - ある関数が合成関数で表される場合、その微分は合成関数を構成するそれぞれの関数の微分の **積** によって表すことが出来る
  - $$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$
  - （分数の通分のイメージ（厳密には違うもの））
- たとえば $z = (x+y)^2$ は $z = t^2, t = x + y$ であるので、$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 =$ □

合成関数の微分

- 連鎖律とは合成関数の微分の性質
  - ある関数が合成関数で表される場合、その微分は合成関数を構成するそれぞれの関数の微分の **積** によって表すことが出来る
  - $$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$
  - （分数の通分のイメージ（厳密には違うもの））
- たとえば $z = (x + y)^2$ は $z = t^2, t = x + y$ であるので、$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 = \boxed{2(x + y)}$$

連鎖律と計算グラフ

☆逆向きに局所的な微分を乗算して渡す

連鎖律と計算グラフ

☆逆向きに局所的な微分を乗算して渡す

連鎖律と計算グラフ

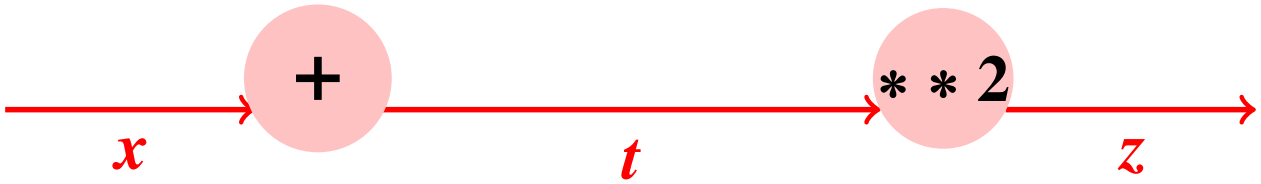
☆逆向きに局所的な微分を乗算して渡す

連鎖律と計算グラフ

☆逆向きに局所的な微分を乗算して渡す

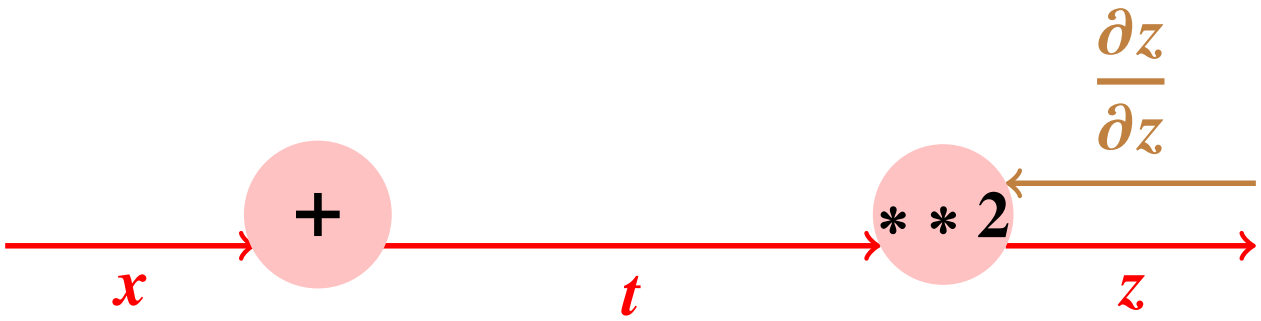
連鎖律と計算グラフ

☆逆向きに局所的な微分を乗算して渡す



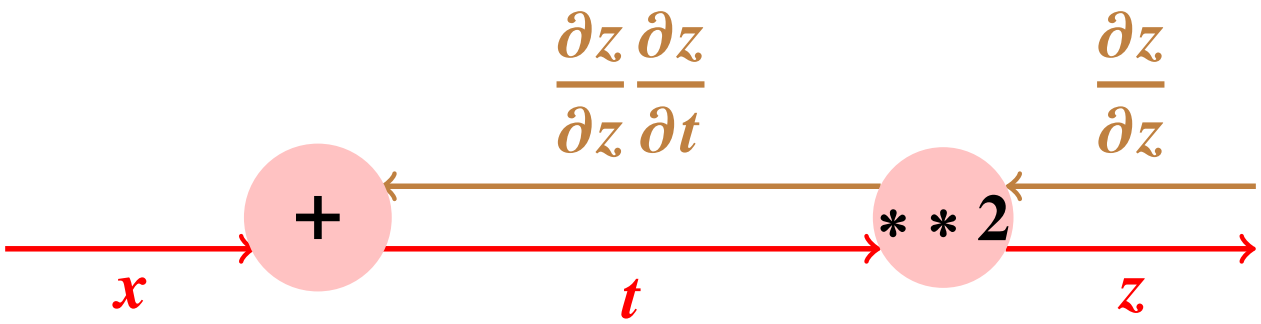
連鎖律と計算グラフ

☆逆向きに局所的な微分を乗算して渡す



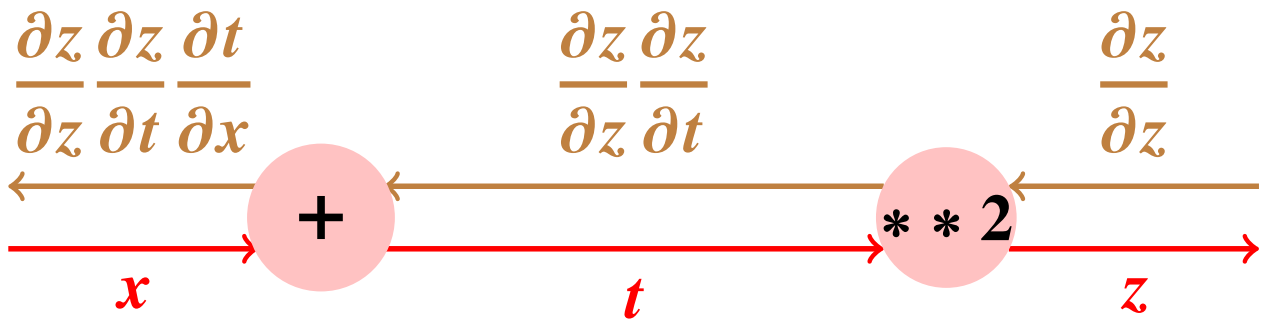
連鎖律と計算グラフ

☆逆向きに局所的な微分を乗算して渡す



連鎖律と計算グラフ

☆逆向きに局所的な微分を乗算して渡す



連鎖律と計算グラフ

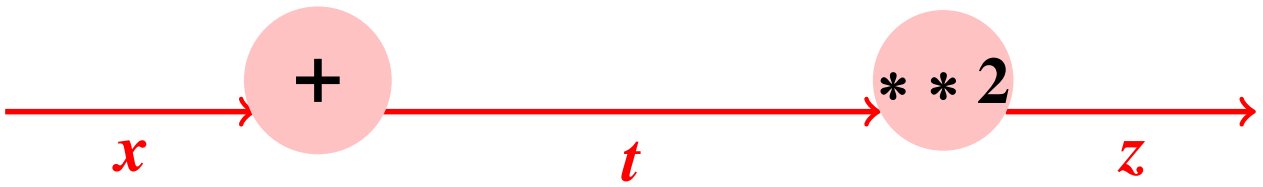
- 前頁の一番左は $\frac{\partial z}{\partial z}\frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = \frac{\partial z}{\partial x}$ が成り立っている
- →逆伝搬が行なっていることは [　　] から導かれていることである (重要!)

連鎖律と計算グラフ

- 前頁の一番左は $\frac{\partial z}{\partial z}\frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = \frac{\partial z}{\partial x}$ が成り立っている
- →逆伝搬が行なっていることは **連鎖律の原理** から導かれていることである (重要!)

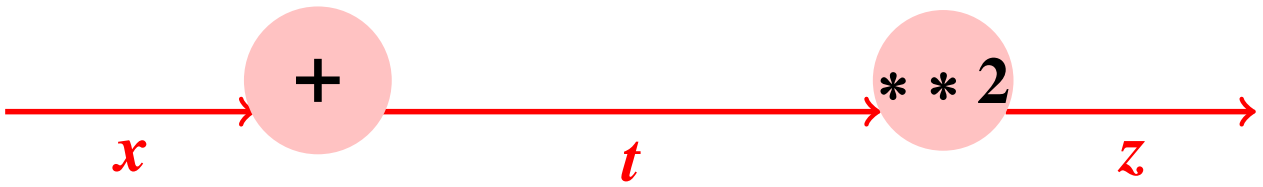
連鎖律と計算グラフ

☆ 具体的に $z = (x + y)^2$ の場合



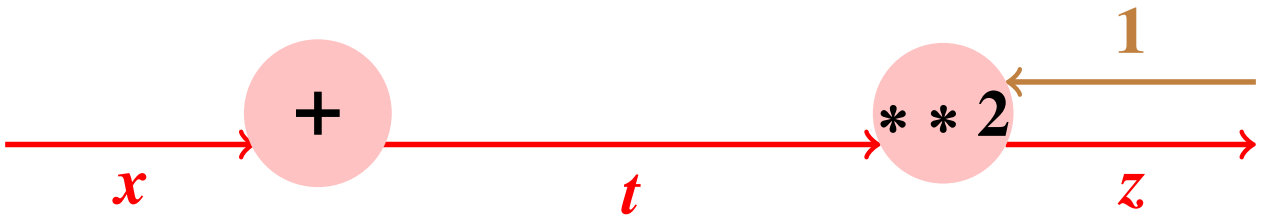
連鎖律と計算グラフ

☆ 具体的に $z = (x + y)^2$ の場合



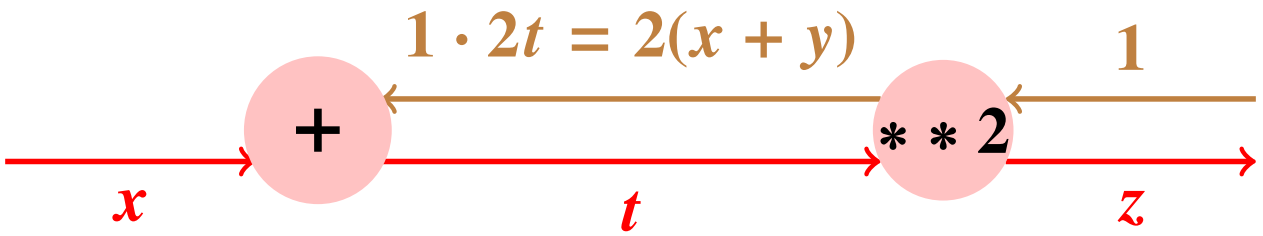
連鎖律と計算グラフ

☆ 具体的に $z = (x + y)^2$ の場合



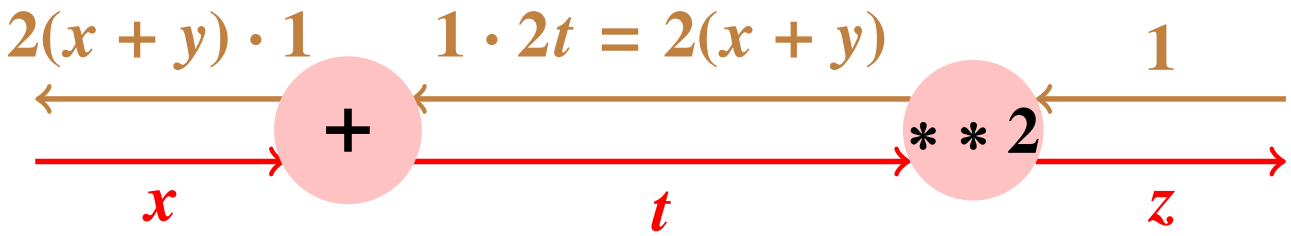
連鎖律と計算グラフ

☆ 具体的に $z = (x + y)^2$ の場合



連鎖律と計算グラフ

☆ 具体的に $z = (x + y)^2$ の場合



加算ノードへの逆伝搬

- $z = x + y$のとき $\dfrac{\partial z}{\partial x} = 1, \dfrac{\partial z}{\partial y} = 1$
- つまり、加算ノードへの逆伝搬は１を掛けるだけなので、入力値を　　だけ

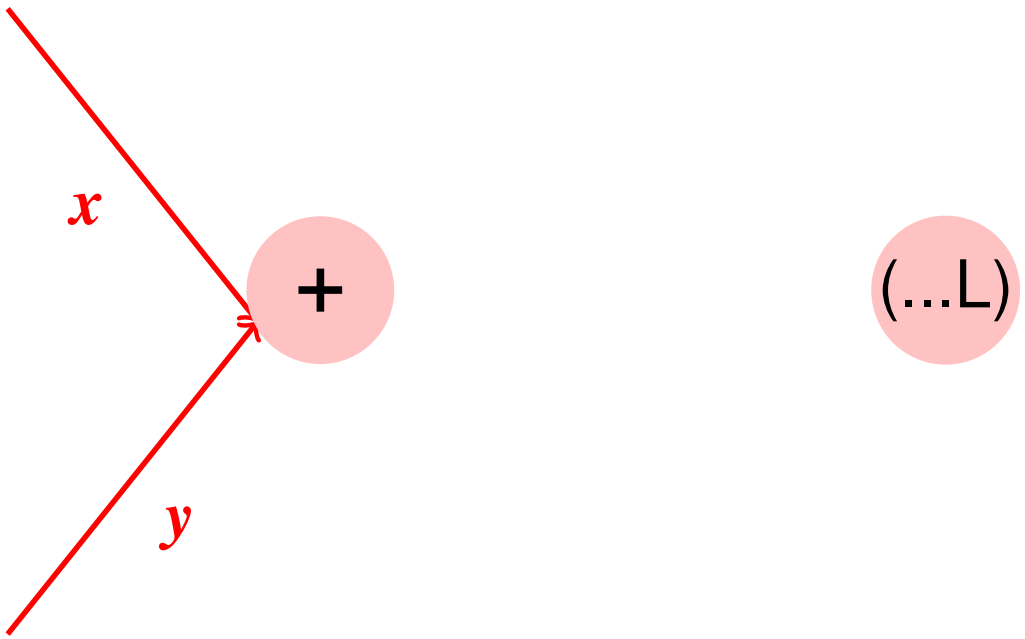
加算ノードへの逆伝搬

- $z = x + y$のとき $\dfrac{\partial z}{\partial x} = 1, \dfrac{\partial z}{\partial y} = 1$
- つまり、加算ノードへの逆伝搬は１を掛けるだけなので、入力値を そのまま流す だけ

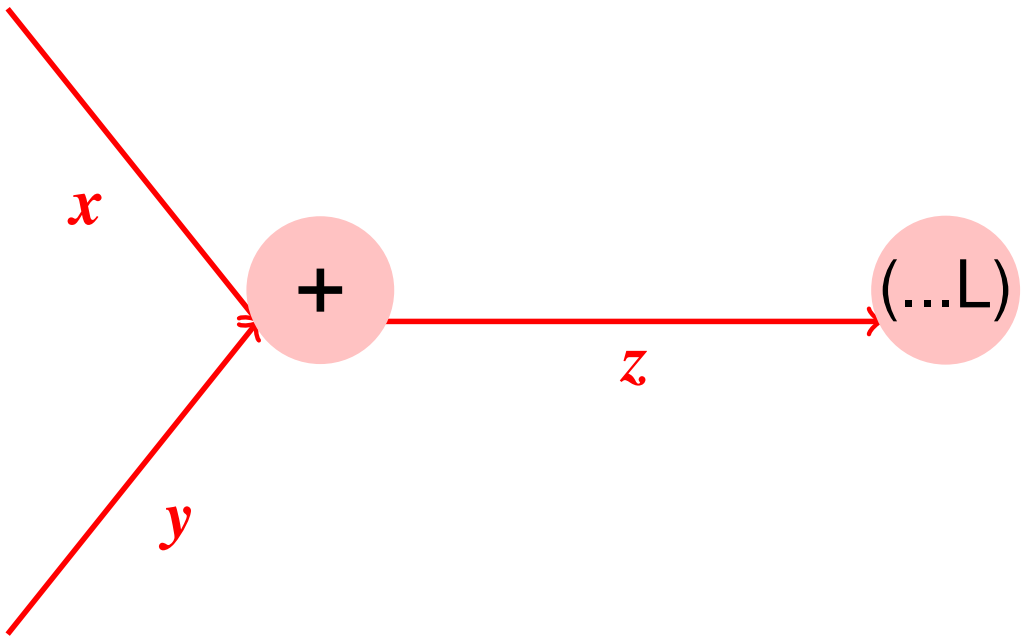
加算ノードへの逆伝搬

加算ノードへの逆伝搬

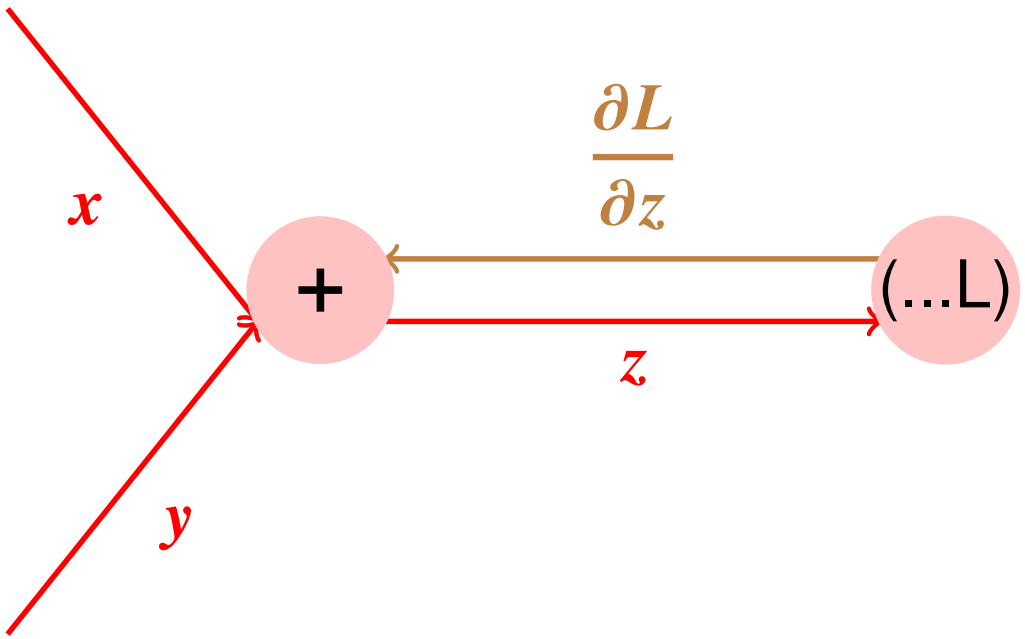
加算ノードへの逆伝搬



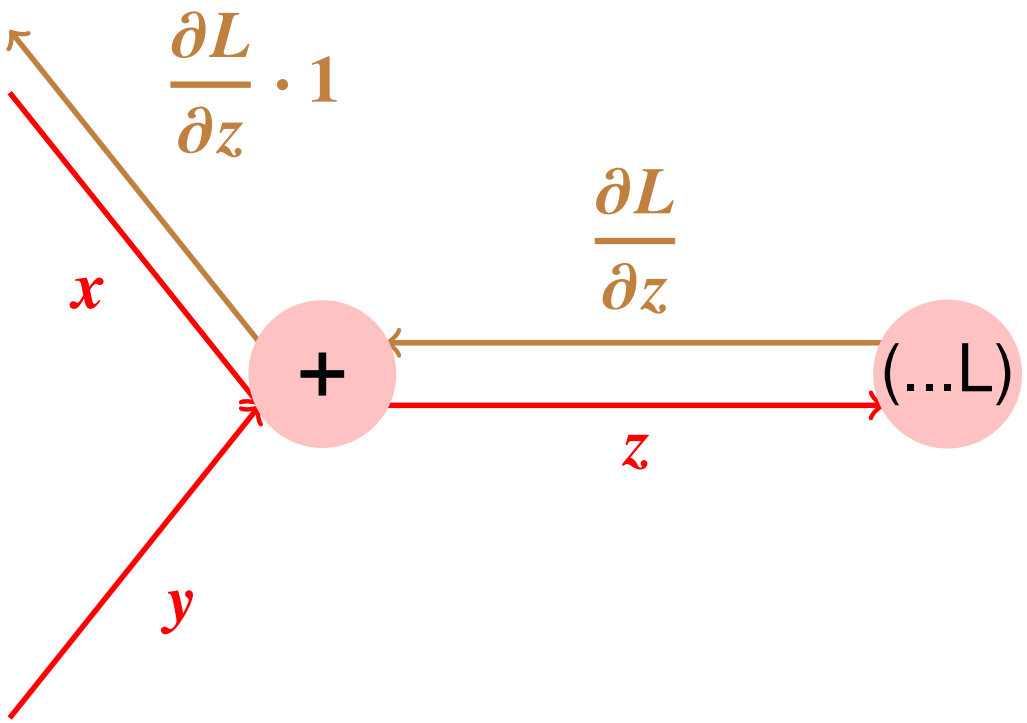
加算ノードへの逆伝搬



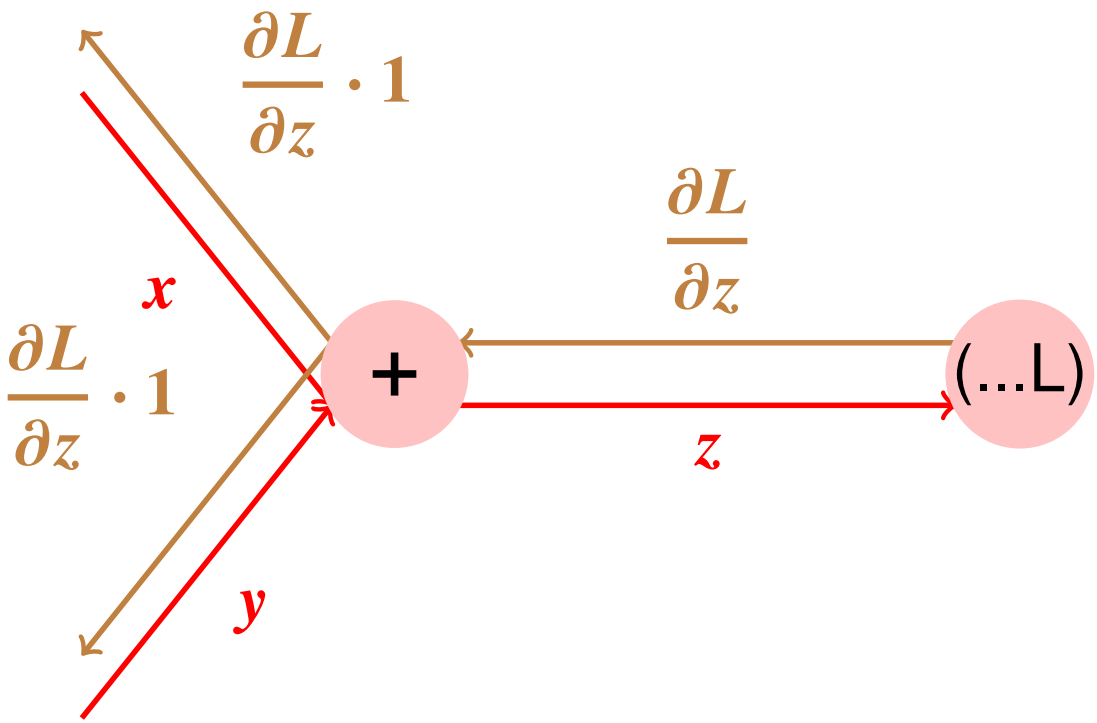
加算ノードへの逆伝搬



加算ノードへの逆伝搬



加算ノードへの逆伝搬


$\frac{\partial L}{\partial z} \cdot 1$
$\frac{\partial L}{\partial z}$
x
$\frac{\partial L}{\partial z} \cdot 1$
+
(...L)
z
y

乗算ノードへの逆伝搬

- $z = xy$ のとき $\dfrac{\partial z}{\partial x} = y, \dfrac{\partial z}{\partial y} = x$
- つまり、乗算ノードへの逆伝搬は入力値を [　　] て [　　] して流す

乗算ノードへの逆伝搬

- $z = xy$ のとき $\dfrac{\partial z}{\partial x} = y, \dfrac{\partial z}{\partial y} = x$
- つまり、乗算ノードへの逆伝搬は入力値を ひっくり返し て [　　] して流す

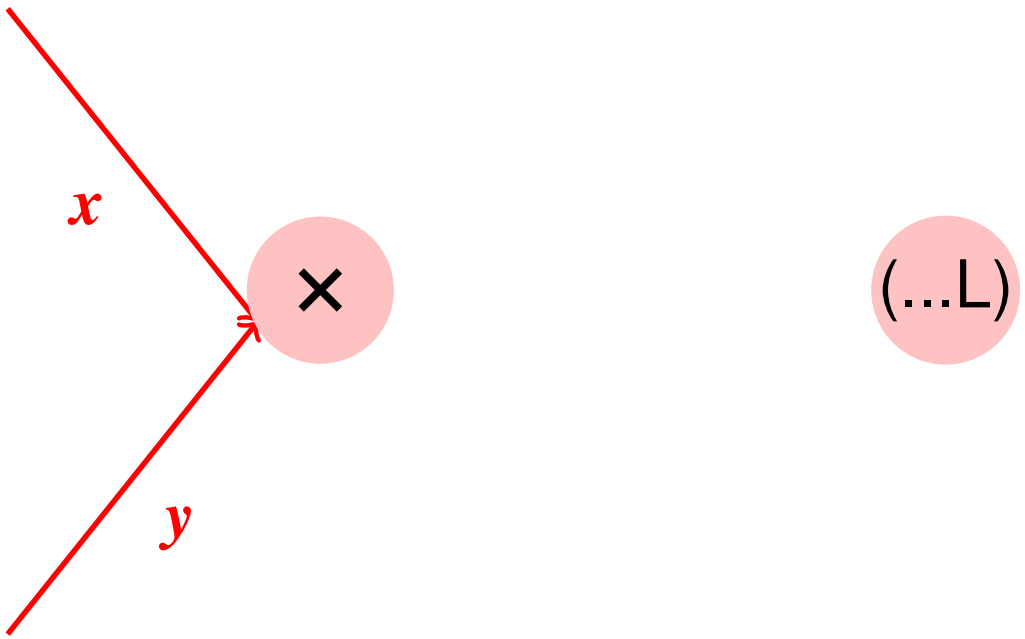
乗算ノードへの逆伝搬

- $z = xy$ のとき $\dfrac{\partial z}{\partial x} = y, \dfrac{\partial z}{\partial y} = x$
- つまり、乗算ノードへの逆伝搬は入力値を ひっくり返し て 乗算 して流す

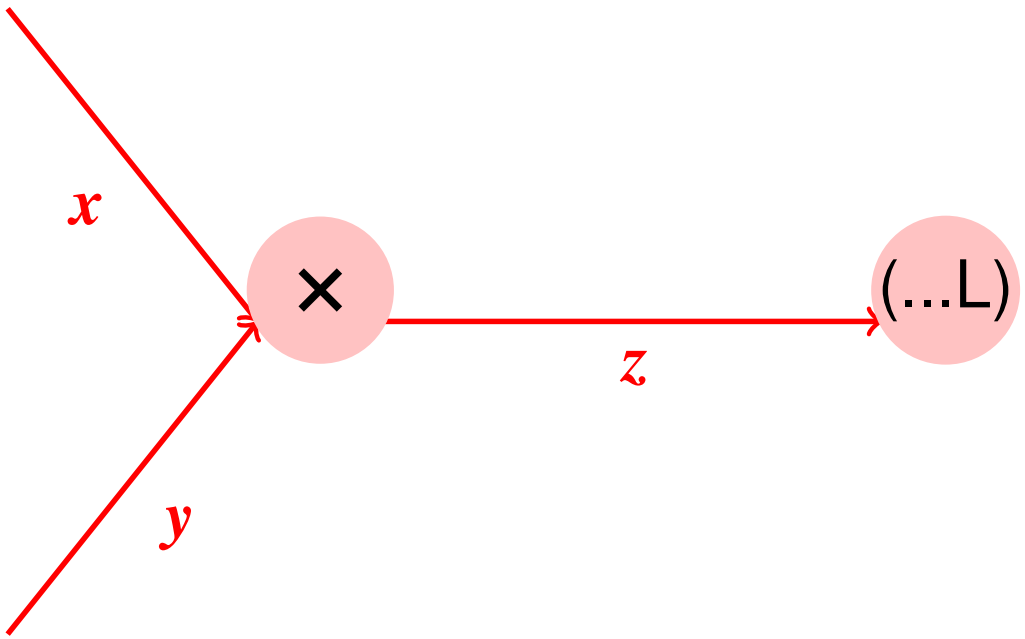
乗算ノードへの逆伝搬

乗算ノードへの逆伝搬

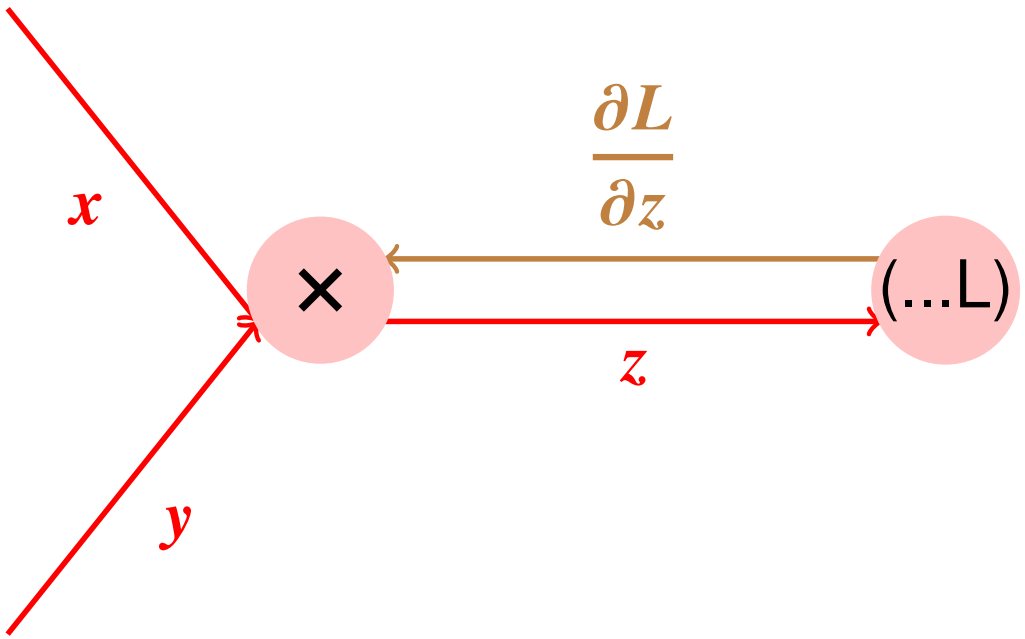
乗算ノードへの逆伝搬



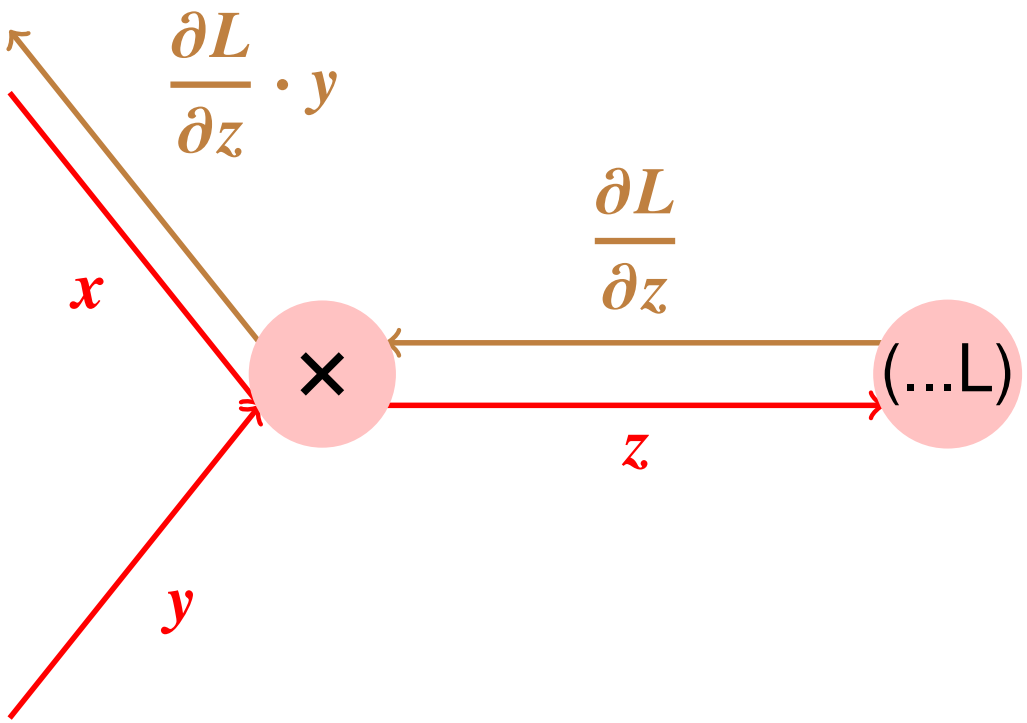
乗算ノードへの逆伝搬



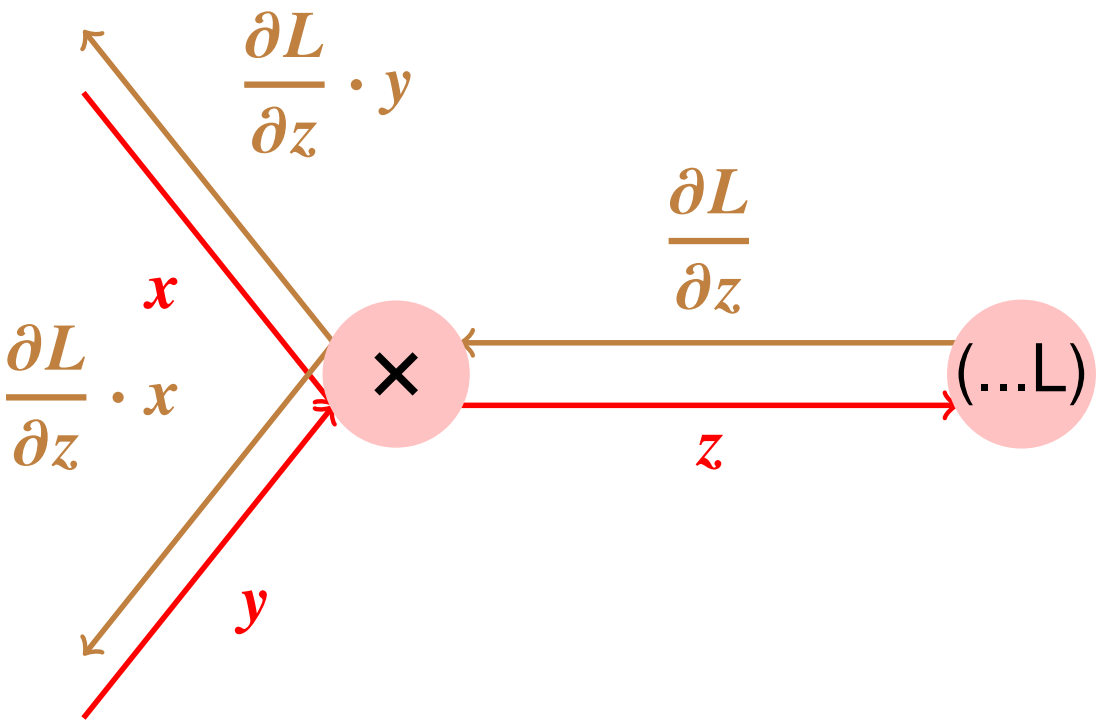
乗算ノードへの逆伝搬



乗算ノードへの逆伝搬



乗算ノードへの逆伝搬

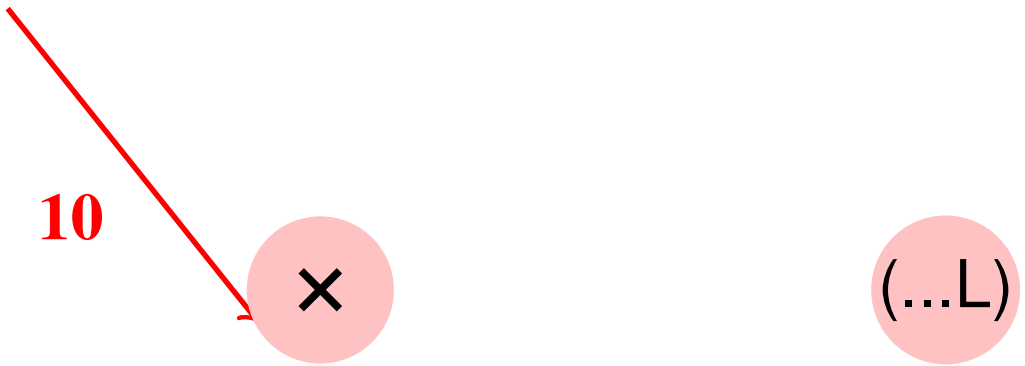

$\frac{\partial L}{\partial z} \cdot y$
$\frac{\partial L}{\partial z}$
x
$\frac{\partial L}{\partial z} \cdot x$
×
(...L)
z
y

乗算ノードへの逆伝搬

☆ 具体例：**10 × 5 = 50** で、 L （上位の計算）から 1.2 が流れてきたとすると；

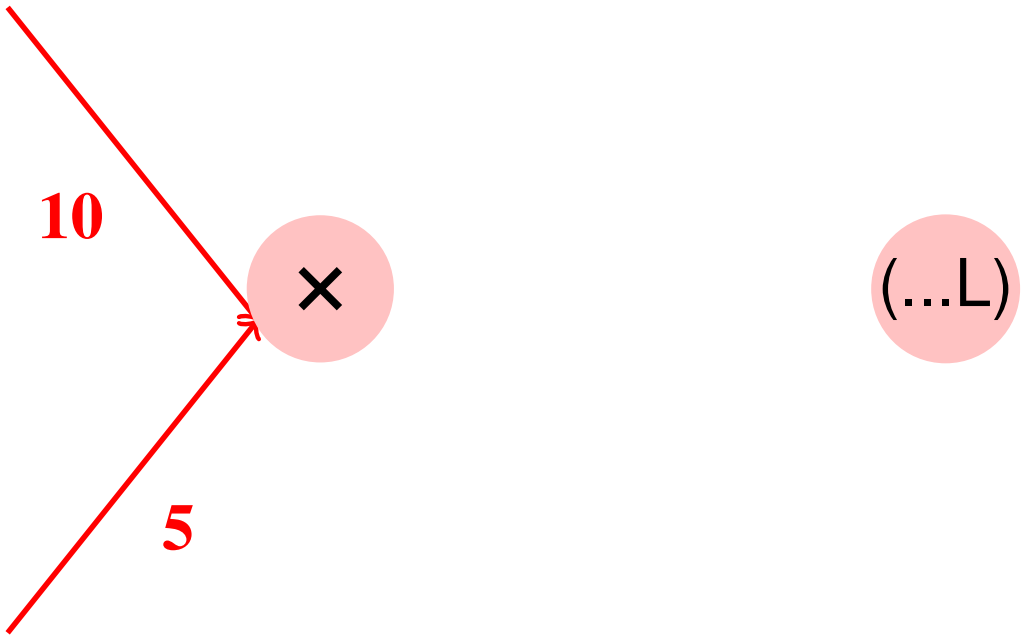
乗算ノードへの逆伝搬

☆ 具体例：**10 × 5 = 50** で、 L （上位の計算）から 1.2 が流れてきたとすると；



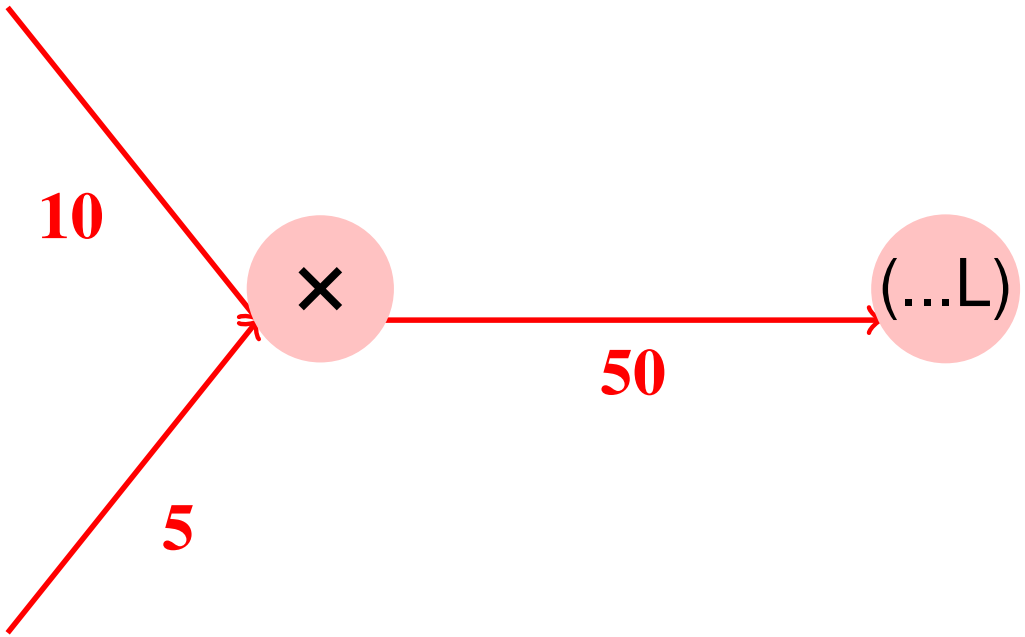
乗算ノードへの逆伝搬

☆ 具体例：$10 \times 5 = 50$ で、 L （上位の計算）から 1.2 が流れてきたとすると；



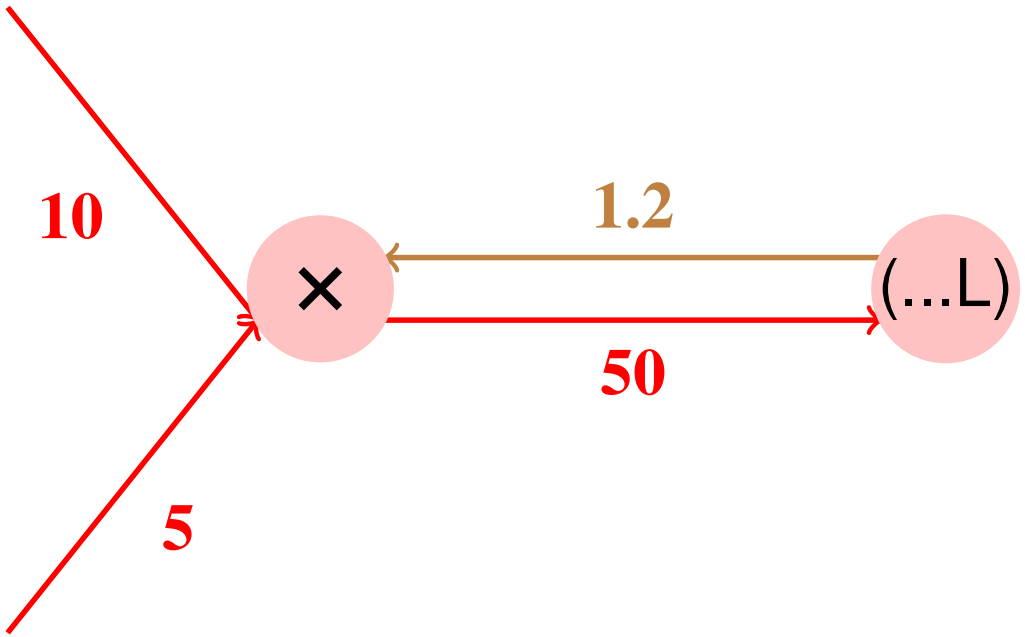
乗算ノードへの逆伝搬

☆ 具体例：$10 \times 5 = 50$ で、 L （上位の計算）から 1.2 が流れてきたとすると；



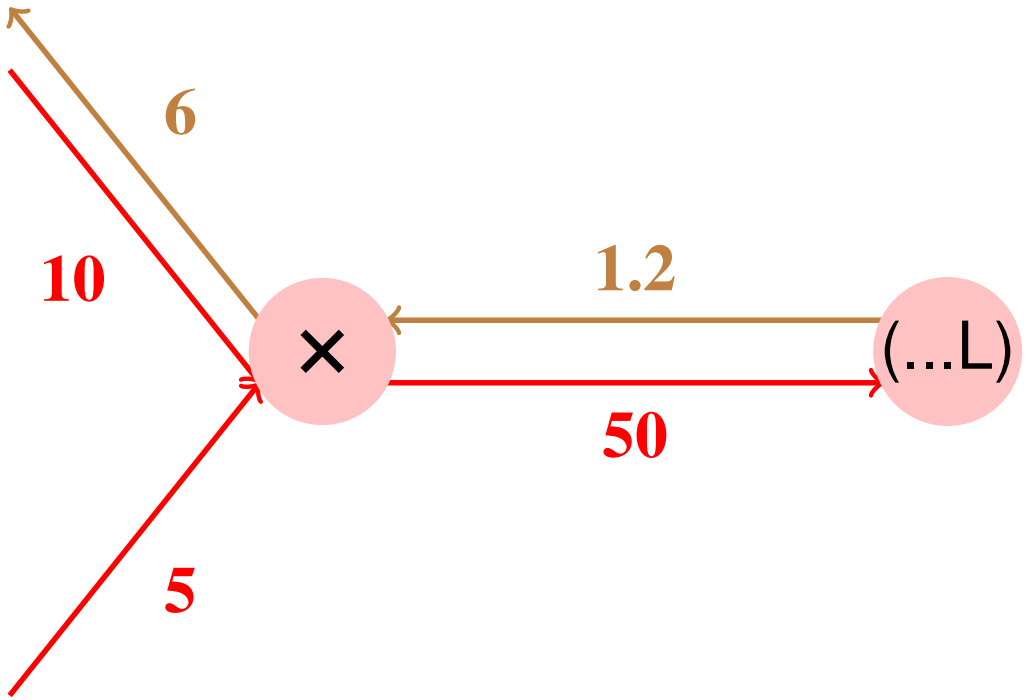
乗算ノードへの逆伝搬

☆ 具体例：$10 \times 5 = 50$ で、 L （上位の計算）から 1.2 が流れてきたとすると；



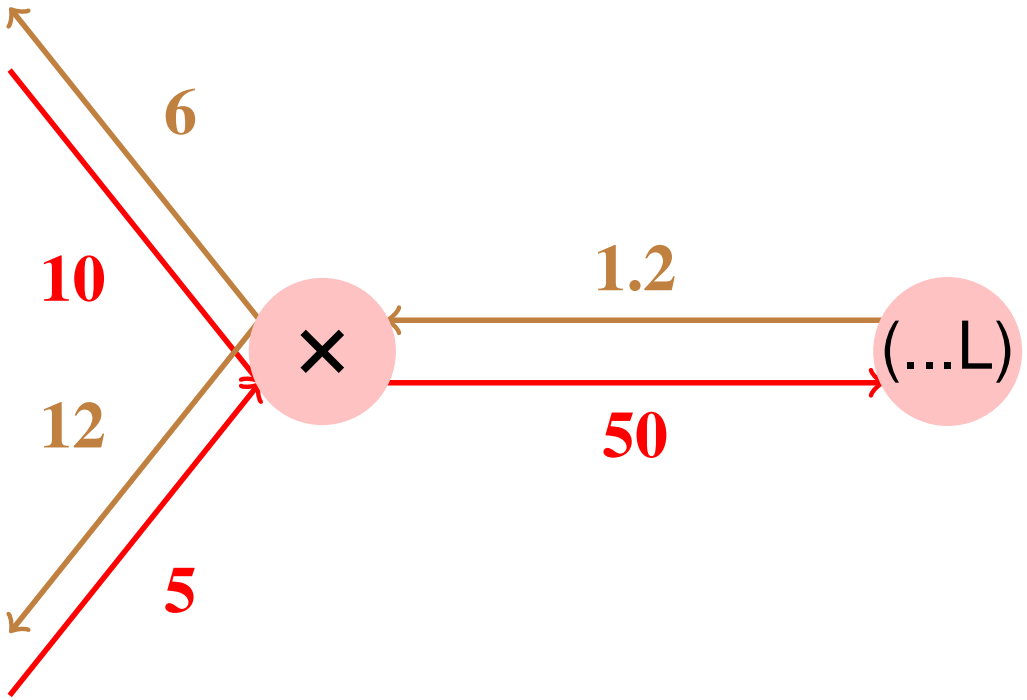
乗算ノードへの逆伝搬

☆ 具体例：$10 \times 5 = 50$ で、 L （上位の計算）から 1.2 が流れてきたとすると；

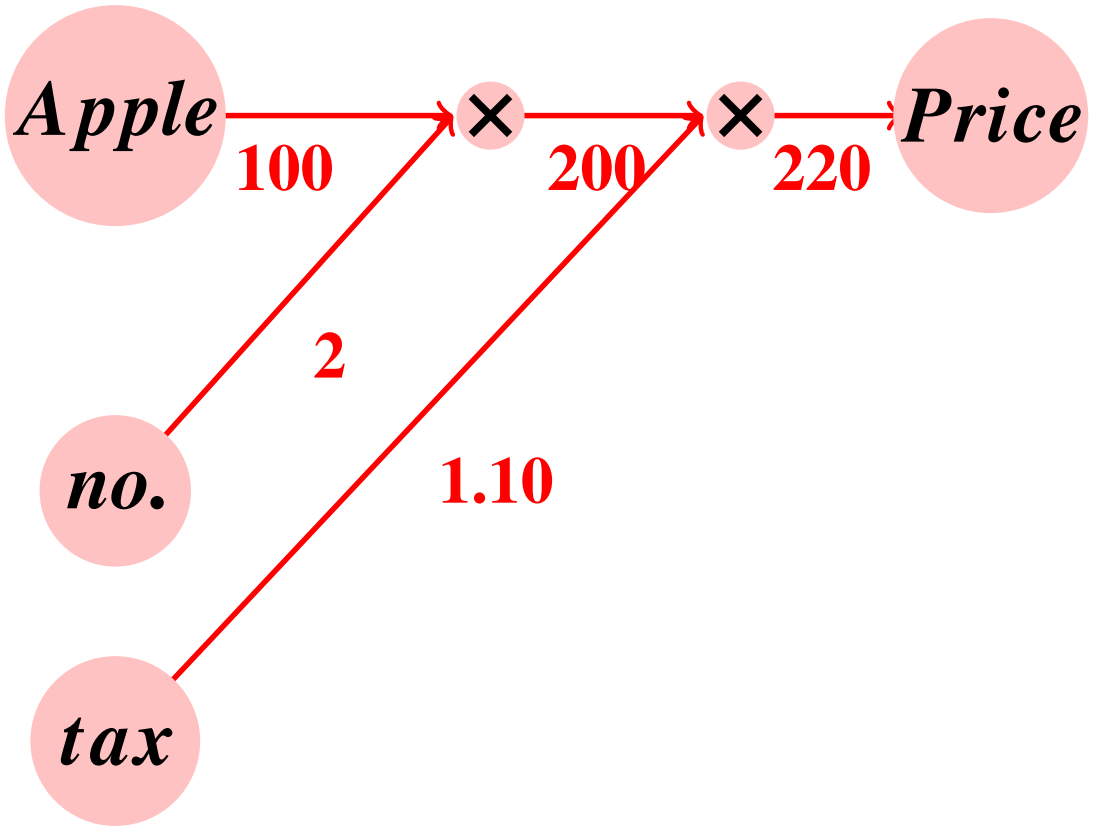


乗算ノードへの逆伝搬

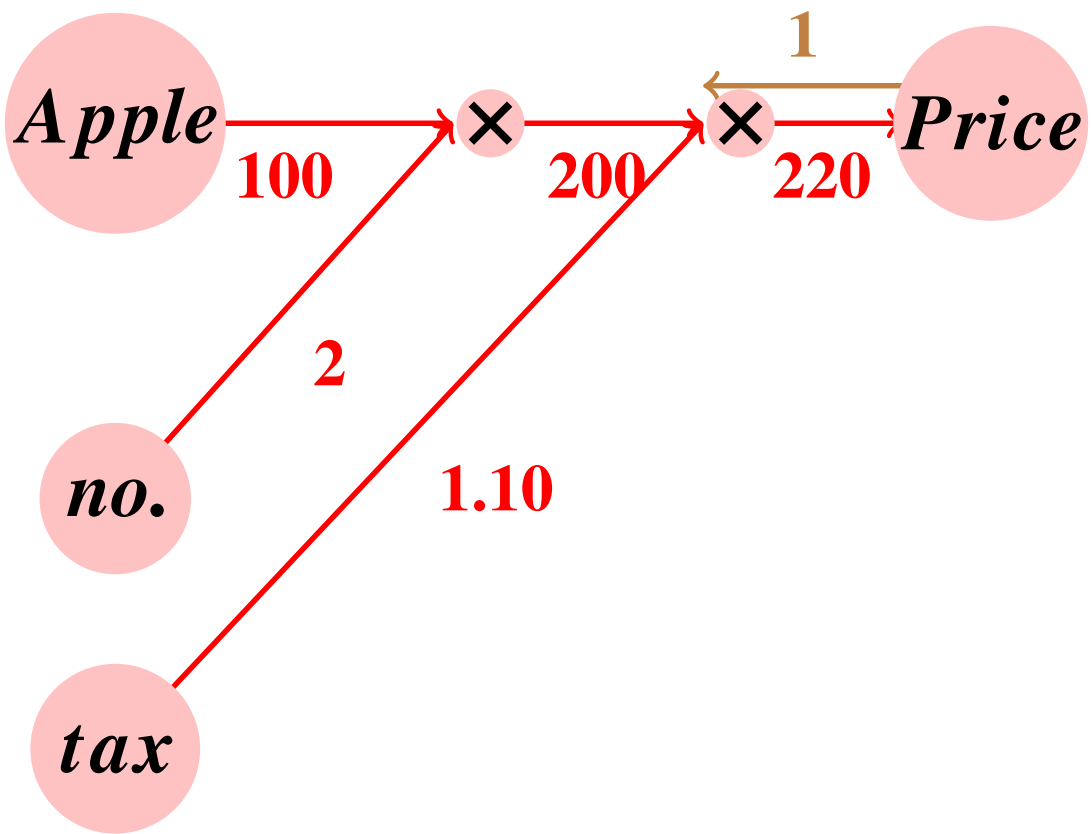
☆ 具体例：$10 \times 5 = 50$ で、 L （上位の計算）から 1.2 が流れてきたとすると；



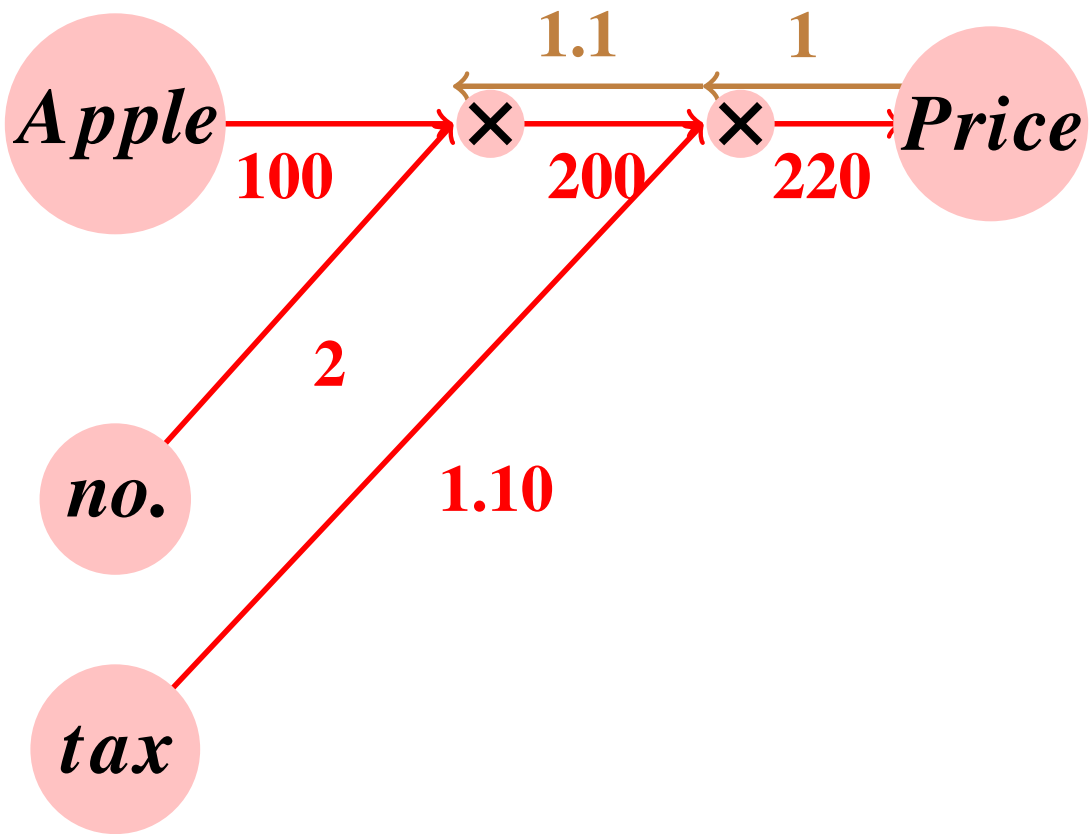
問１のリンゴの計算の逆伝搬の例



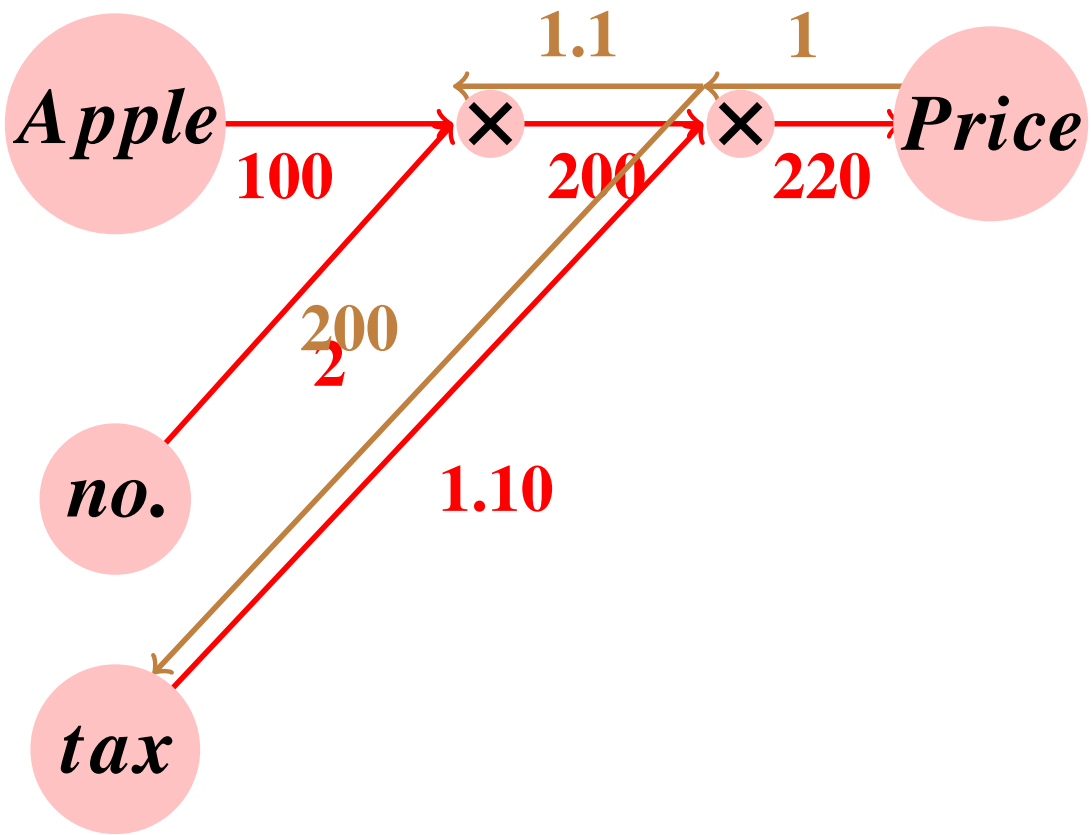
問１のリンゴの計算の逆伝搬の例



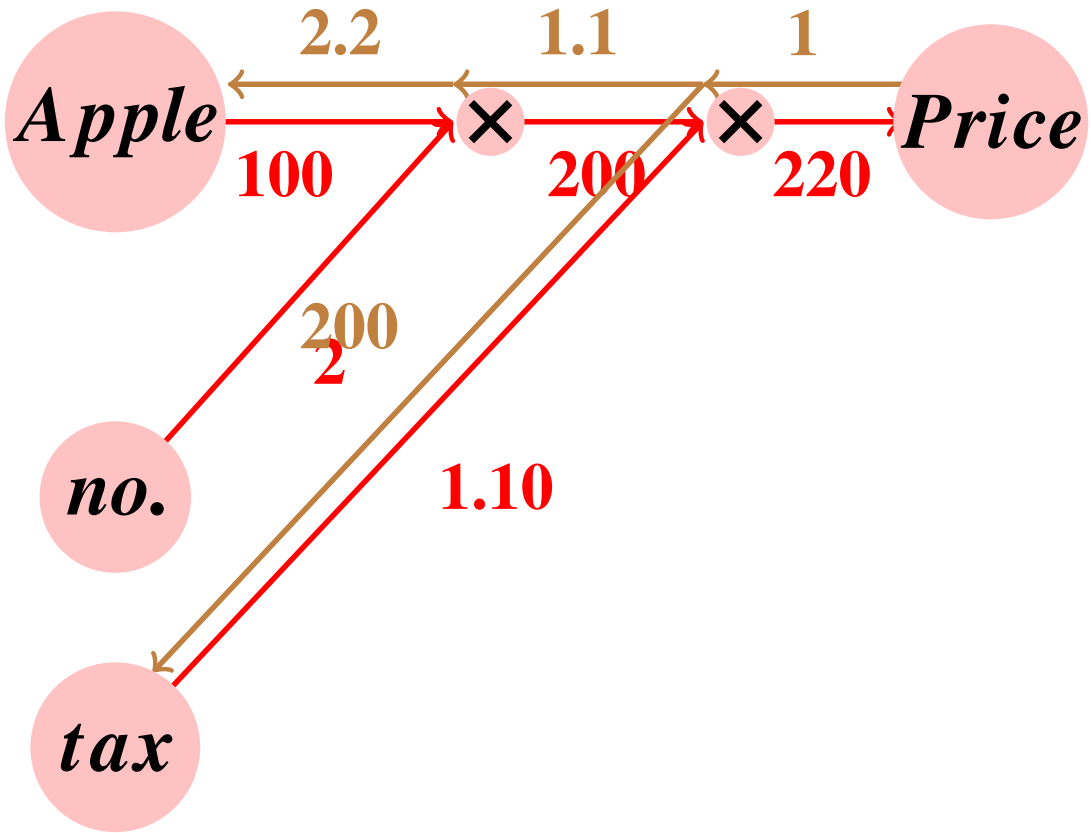
問１のリンゴの計算の逆伝搬の例



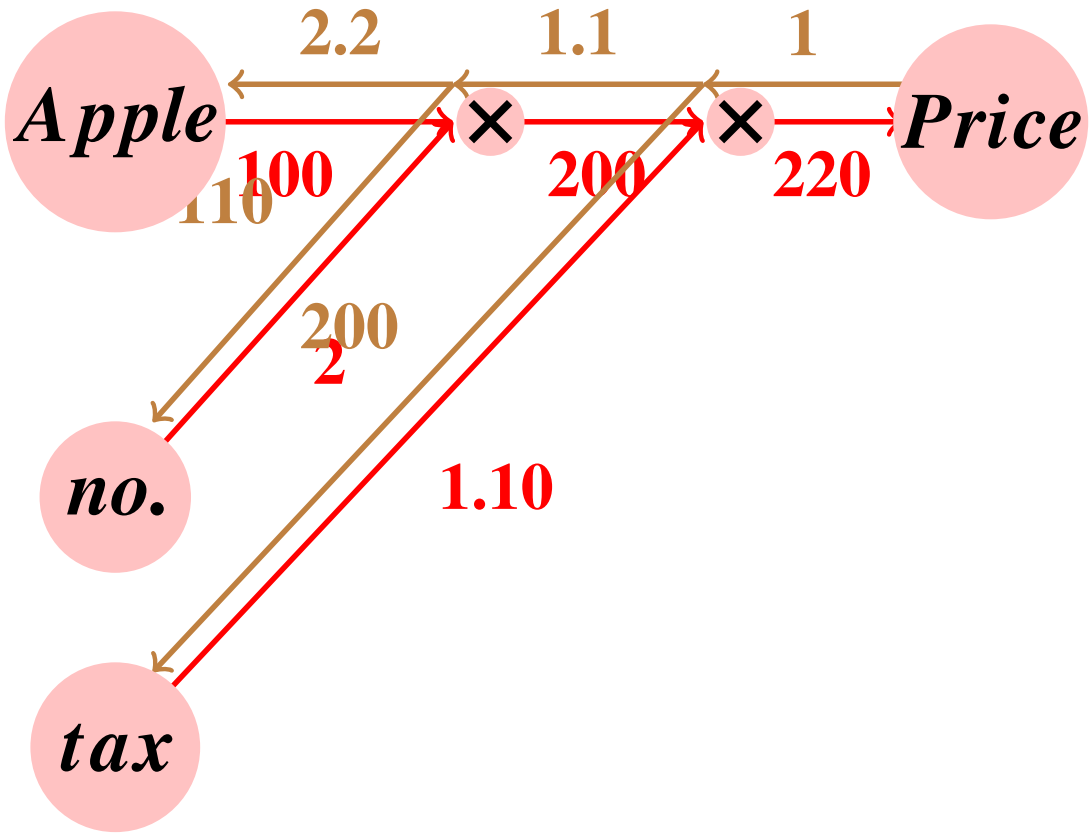
問１のリンゴの計算の逆伝搬の例



問１のリンゴの計算の逆伝搬の例



問１のリンゴの計算の逆伝搬の例



加算レイヤの実装

☆以下を Colab. の mymodules の中に layer_naive.py という名前で保存する

```
class AddLayer:
    def __init__(self):
        pass
    def forward(self, x, y):
        out = x + y
        return out
    def backward(self, dout):
        dx = dout * 1
        dy = dout * 1
        return dx, dy
```

加算レイヤの実装（解説）

- __init__ は 何もしない（初期化は不要）
- forward で x, y を受けとり、それらを加算
- backward では上流から伝わってきた微分 dout をそのまま下流にながすだけ

乗算レイヤの実装

☆以下を Colab. の mymodules の中に先程作った layer_naive.py というファイルに追加する

```
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None
    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y
        return out
    def backward(self, dout):
        dx = dout * self.y
        dy = dout * self.x
        return dx, dy
```

乗算レイヤの実装（解説）

- __init__ でインスタンス変数 x, y を初期化
  - [ ] とは、オブジェクトそれぞれが個別に持てる変数で、クラス共通で使えるクラス変数と区別される（詳細略）
- forward で x, y を受けとり、それらを乗算
- backward では上流から伝わってきた微分 dout に対して順伝搬の「ひっくり返した」値を乗算して下流にながす

乗算レイヤの実装（解説）

- __init__ でインスタンス変数 x, y を初期化
  - **インスタンス変数**とは、オブジェクトそれぞれが個別に持てる変数で、クラス共通で使えるクラス変数と区別される（詳細略）
- forward で x, y を受けとり、それらを乗算
- backward では上流から伝わってきた微分 dout に対して順伝搬の「ひっくり返した」値を乗算して下流にながす

乗算レイヤの実装

☆以下によりリンゴの計算の逆伝搬を実行する
（ buy_apple.py のリンク先を Colab. にコピー＆ペーストし実行する）

```
# coding: utf-8
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
bd = 'drive/My Drive/Colab Notebooks/mymodules'
import sys, os
import numpy as np
sys.path.append(bd)
from layer_naive import *
apple = 100
apple_num = 2
tax = 1.1
mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()
```

（…次頁に続く）

乗算レイヤの実装

（前頁からの続き）

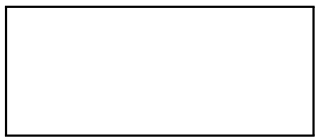
```
# forward
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)
# backward
dprice = 1
dapple_price, dtax = mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)
print("price:", int(price))
print("dApple:", dapple)
print("dApple_num:", int(dapple_num))
print("dTax:", dtax)
```

乗算レイヤの実装

☆ 実行結果

```
Mounted at /content/drive
price: 220
dApple: 2.2
dApple_num: 110
dTax: 200
```

次週の予定

- 活性化関数レイヤの実装 (ReLU, Sigmoid)
- 行列計算 (Affine) レイヤの実装
- 損失関数と出力 (Softmax) レイヤの実装
- →全体の実装 

次週の予定

- 活性化関数レイヤの実装 (ReLU, Sigmoid)
- 行列計算 (Affine) レイヤの実装
- 損失関数と出力 (Softmax) レイヤの実装
- →全体の実装 (完成!)

おしまい

質問・コメント等あれば、何でもお気軽に
aipr@e-chan.jp に
メールください！