Homework 2
COSE212, Fall 2018

Hakjoo Oh

Due: 10/14, 24:00

Academic Integrity / Assignment Policy

- All assignments must be your own work.
- Discussion with fellow students is encouraged including how to approach the problem. However, your code must be your own.
  - Discussion must be limited to general discussion and must not involve details of how to write code.
  - You must write your code by yourself and must not look at someone else's code (including ones on the web).
  - Do not allow other students to copy your code.
  - Do not post your code on the public web.
- **Violating above rules gets you 0 points for the entire HW score.**

Problem 1 (10pts) In class, we defined the function `reverse` as follows:

```
let rec reverse l =
  match l with
  | [] -> []
  | hd::tl -> (reverse tl) @ [hd]
```

This function is too slow in practice as its time complexity is $O(n^2)$. For instance, `reverse (range 1 100000)` may not terminate quickly on typical machines. However, list reversal can be implemented efficiently with time complexity $O(n)$. Write a function

`fastrev : 'a list -> 'a list`

that reverses a given list with in $O(n)$. For instance, `fastrev (range 1 100000)` should produce `[100000; 99999; ...; 1]` immediately.

Problem 2 (10pts) Write a function

`app: 'a list ->'a list -> 'a list`

which appends the first list to the second list while removing duplicated elements. For instance, given two lists [4;5;6;7] and [1;2;3;4], the function should output [1;2;3;4;5;6;7]:

```
app [4;5;6;7] [1;2;3;4] = [1;2;3;4;5;6;7].
```

Problem 3 (10pts) Write a function

```
uniq: 'a list -> 'a list
```

which removes duplicated elements from a given list so that the list contains unique elements. For instance,

```
uniq [5;6;5;4] = [5;6;4]
```

Problem 4 (10pts) Write a function `reduce` of the type:

```
reduce : ('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> 'c -> 'c
```

Given a function `f` of type `'a -> 'b -> 'c -> 'c`, the expression

```
reduce f [x1;x2;...;xn] [y1;y2;...;yn] c1
```

evaluates to `f xn yn (... (f x2 y2 (f x1 y1 c1))...)`. For example,

```
reduce (fun x y z -> x * y + z) [1;2;3] [0;1;2] 0
```

evaluates to 8.

Problem 5 (15pts) Write a function

```
diff : aexp * string -> aexp
```

that differentiates the given algebraic expression with respect to the variable given as the second argument. The algebraic expression `aexp` is defined as follows:

```
type aexp =
 | Const of int
 | Var of string
 | Power of string * int
 | Times of aexp list
 | Sum of aexp list
```

For example, $x^2 + 2x + 1$ is represented by

```
Sum [Power ("x", 2); Times [Const 2; Var "x"]; Const 1]
```

and differentiating it (w.r.t. "x") gives $2x + 2$, which can be represented by

```
Sum [Times [Const 2; Var "x"]; Const 2]
```

Note that the representation of $2x + 2$ in `aexp` is not unique. For instance, the following also represents $2x + 2$:

```
Sum
 [Times [Const 2; Power ("x", 1)];
  Sum
   [Times [Const 0; Var "x"];
    Times [Const 2; Sum [Times [Const 1]; Times [Var "x"; Const 0]]]];
  Const 0]
```

Problem 6 (15pts) Consider the following expressions:

```
type exp = X
         | INT of int
         | ADD of exp * exp
         | SUB of exp * exp
         | MUL of exp * exp
         | DIV of exp * exp
         | SIGMA of exp * exp * exp
```

Implement a calculator for the expressions:

```
calculator : exp -> int
```

For instance,

$$\sum_{x=1}^{10}(x * x - 1)$$

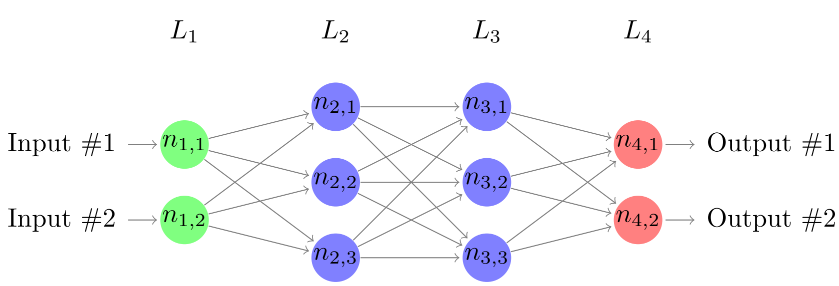
is represented by

```
SIGMA(INT 1, INT 10, SUB(MUL(X, X), INT 1))
```

and evaluating it should give 375.

Problem 7 (30pts) It is fun to implement neural networks with higher-order functions. Neural networks are essentially a programming language.[1] The goal of this problem is to implement an evaluator (i.e., interpreter) for simple feed-forward neural networks.

Let us first define neural networks. A neural network $N = \{L_1, L_2, \dots, L_K\}$ ($K \geq 2$) is a collection of layers ($1 \leq L_i \leq K$), where K is the number of layers. For example, the following network is comprised of four layers (i.e., $K = 4$):



[1]See `https://medium.com/@karpathy/software-2-0-a64152b37c35`

We call L_1 the input layer, L_K the output layer, and others (i.e., $L_2, \ldots, L_{K-1}$) the hidden layers. Each layer consists of nodes (called neurons). Let s_i be the number of nodes in layer L_i. Let $n_{i,j}$ $(1 \leq i \leq K, 1 \leq j \leq s_i)$ be the j-th node of layer L_i. Each node computes a value of real number ($\mathbb{R}$). Let $v_{i,j} \in \mathbb{R}$ be the value of $n_{i,j}$. Except for the input layer, nodes in layer L_i is fully connected to nodes in layer L_{i-1}, where each connection from $n_{i-1,j}$ to $n_{i,k}$ is associated with a *weight* $w_{i,j,k} \in \mathbb{R}$. Also each node $n_{i,j}$ $(i \geq 2)$ is associated with a *bias* $b_{i,j} \in \mathbb{R}$. The weights and biases for nodes of layer L_i $(2 \leq i \leq K)$ can be represented by a matrix $W_i \in \mathbb{R}^{s_{i-1} \times s_i}$ and a vector $B_i \in \mathbb{R}^{s_i}$ as follows:

$$W_i = \begin{bmatrix} w_{i,1,1} & w_{i,1,2} & \cdots & w_{i,1,s_i} \\ w_{i,2,1} & w_{i,2,2} & \cdots & w_{i,2,s_i} \\ \cdots & & & \\ w_{i,s_{i-1},1} & w_{i,s_{i-1},2} & \cdots & w_{i,s_{i-1},s_i} \end{bmatrix} \qquad B_i = \begin{bmatrix} b_{i,1} \\ b_{i,2} \\ \vdots \\ b_{i,s_i} \end{bmatrix}$$

Let $I = \left[v_{1,1}, v_{1,2}, \ldots, v_{1,s_1}\right]^T \in \mathbb{R}^{s_1}$ be an input vector.[2] Evaluating a network N for I is to propagate I through the network via the following equation:

$$\begin{array}{rcl} V_1 & = & I \\ V_i & = & f_i(W_i^T V_{i-1} + B_i) \quad (2 \leq i < K) \\ V_K & = & W_K^T V_{K-1} + B_K \end{array}$$

where $f_i \in \mathbb{R}^{s_i} \to \mathbb{R}^{s_i}$ is the *activation function* of layer L_i, which is defined as follows:

$$f_i([x_1, x_2, \ldots, x_{s_i}]^T) = [\max(x_1, 0), \max(x_2, 0), \ldots, \max(x_{s_i}, 0)]^T.$$

where $\max(a, b)$ is a if $a > b$ and otherwise b. Note that we apply the activation function only for the hidden layers. Finally, using the output $V_K = [v_{K,1}, v_{K,2}, \ldots, v_{K,s_K}]^T$, the neural network assigns a *label* l, i.e., the index of the node of the output layer L_K with the largest value. That is, the output of the neural network N for input I, which is denoted by $N(I)$, is defined as follows:

$$N(I) = \text{argmax}_{1 \leq l \leq s_K} v_{K,l}$$

where $\text{argmax}_{1 \leq l \leq s_K} v_{K,l}$ denotes the index l at which the output $(v_{K,l})$ of the neural network is maximized (when there are multiple such indices, it returns anything of them).

Let us implement a neural-network evaluator. In OCaml, a neural network can be defined as follows:

```
type vector = float list
type matrix = float list list
type layer =
  | Input
  | Hidden of (matrix * vector)
  | Output of (matrix * vector)
type network = layer list
```

[2]For matrix or vector A, A^T denotes the transpose of A.

A vector is a list of real numbers and a matrix is a list of vectors. A layer is input, hidden, or output, where hidden and output layers are associated with a weight matrix and a bias vector.

1. (5pts) Write a function

   `addvec: vector -> vector -> vector`

   which adds two vectors. For example, `addvec [1.0; 2.0] [3.0; 4.0]` evaluates to `[4.0; 6.0]`.

2. (5pts) Write a function

   `mulmat: matrix -> vector -> vector`

   which multiplies a matrix and a vector. For example `mulmat [[1.0; 2.0]; [3.0; 4.0]] [5.0; 6.0]` evaluates to `[17.0; 39.0]`.

3. (5pts) Write a function

   `transpose: matrix -> matrix`

   which performs the matrix transpose. For example `tranpose [[1.0; 2.0]; [3.0; 4.0]] [5.0; 6.0]` evaluates to `[[1.0; 3.0]; [2.0; 4.0]]`.

4. (5pts) Write a function

   `argmax: float list -> int`

   which takes a list of floats and returns the index of the maximal element. For example, `argmax [-0.46; 0.53; 0.64; 0.12]` evaluates to 2.

5. (10pts) Write a function

   `nneval: network -> vector -> int`

   which takes a neural network (N) and an input vector (I), and computes $N(I)$.

   Consider the following network (`net`):

```
let layer2 = ([[-0.46; 0.53; 0.64; 0.12];
               [ 0.78; 0.62; -0.89; -0.34]], [ 0.03; -0.03; 0.03; -0.02])
let layer3 = ([[-0.13; -0.96];
               [-0.48; 0.13];
               [0.78; -0.49];
               [-0.04; -0.05]], [0.03; -0.03])
let net = [Input; Hidden layer2; Output layer3]
```

   which has an input layer (with 2 neurons), a hidden layer (with 4 neurons), and an output layer (with 2 neurons). For example,

- `nneval net [0.0; 0.0] = 0`
- `nneval net [0.0; 1.0] = 0`
- `nneval net [1.0; 0.0] = 0`
- `nneval net [1.0; 1.0] = 1`

Note that this neural network can be seen as a program that implements the “AND” function.