Pseudo-Random Number Generators

Functional Programming and Intelligent Algorithms

Prof Hans Georg Schaathun

Høgskolen i Ålesund

11th February 2015

Outline

HØGSKOLEN I ÅLESUND
Aalesund University College

Review of yesterday

- + What did you learn yesterday?
- △ What is your greatest challenge?

The implementation of the perceptron

1. How does your neural network perform?
2. What needs improvement

Outline

Randomness

What is randomness?

Probabilistic programs

How do we create probabilistic computer programs?
I.e. how do we make the computer act at random?

Two options

True randomness uses physical sources of entropy

1. `/dev/random` on many systems
2. `random-fu` in Haskell

Pseudo-random number generators (PRNG) are deterministic but random-*looking*

- `random`, standard package in Haskell
- `random-tf`, more recent Haskell package

Linear Congruential Generators

A classic PRNG

- Recurrence $x_i = a + cx_{i-1} \mod m$
- Seed (initial state) x_0
- Pseudo-random sequence $[x_0, x_1, x_2, \ldots]$
- Known as *Lehmer's algorithm*

Will this pseudo-random sequence look random?

Ciphers in counter mode

PRNG through cryptography

- Cipher $e_k(x) = y$
- Pseudo-random sequence $[x_0, x_1, x_2, \ldots]$ where
  - $x_i = e_k(i)$

Why does this give good randomness?

The PRNG is a state machine

1. The state decides what the PRNG will output
   - *Lehmer's algorithm* x_{i-1} is the state
   - *Counter mode* i is the state
2. State transition
   - *Lehmer's algorithm* $x \mapsto a + cx \mod m$
   - *Counter mode* $i \mapsto i + 1$
3. Output function
   - *Lehmer's algorithm* $x \mapsto a + cx \mod m$
   - *Counter mode* $i \mapsto e_k(i + 1)$

State machines in functional programming

How do we handle state machines in functional programming?

1. What is special about functional programming?
2. What is difficult?
3. How can we do it?

Outline

Random sequence

1. `next :: TFGen -> (TFGen,Word32)`

Exercise

Given a TFGen object, how do you generate an random, infinite list of Word32 objects?

Splitting a PRNG

Problem. After you have generated the infinite list, how do you get the updated state?

1. `split :: TFGen -> (TFGen,TFGen)`
2. `(g',newstate) = split g`
3. Use `g'` to generate the list
4. `newstate` is your new state

Exercise

Given a TFGen object, how do you generate an random, infinite list of Word32 objects?

Splitting a PRNG

Problem. After you have generated the infinite list, how do you get the updated state?

1. `split :: TFGen -> (TFGen,TFGen)`
2. `(g',newstate) = split g`
3. Use `g'` to generate the list
4. `newstate` is your new state

Exercise

Given a TFGen object, how do you generate an random, infinite list of Word32 objects?

Question

Where do you get the initial state?

Different options

1. Hardcode an arbitrary seed
2. Use initialisation functions in the library
    1. `initTFGen`
3. Use a library which provides true random values
    - random-fu

Outline

Tuning parameters

1. Distribution of random initial weights?
2. β in the sigmoid function?
3. Number of iterations?

Artificial intelligence is not an exact science.

- Trial and error.
- Test different choices.

Tuning parameters

1. Distribution of random initial weights?
2. β in the sigmoid function?
3. Number of iterations?

Artificial intelligence is not an exact science.

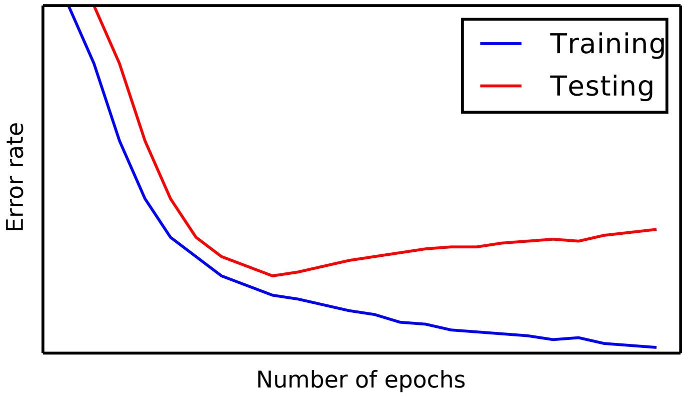
- Trial and error.
- Test different choices.

Some guidelines

- Weights: $-1/\sqrt{n} \leq w \leq 1/\sqrt{n}$
  - where n is the number of inputs to the layer
- The weights should have similar magnitude
- Small β — $\beta \leq 3$
  1. $\beta = 1$ is a good starting point

HØGSKOLEN I ÅLESUND
Aalesund University College

Number of epochs



Exercise

- Update the `initNeuron` and `initNetwork` functions with randomised weights
- You need to import a library, either
  - `System.Random`; or
  - `System.Random.TF`
- Initialise the PRNG
- Generate and use the weights for the network