# Implementasi *Q-Learning* dan *Backpropagation* pada Agen yang Memainkan Permainan *Flappy Bird*

Ardiansyah[1], Ednawati Rainarli[2]

***Abstract*—This paper shows how to implement a combination of Q-learning and backpropagation on the case of agent learning to play Flappy Bird game. Q-learning and backpropagation are combined to predict the value-function of each action, or called value-function approximation. The value-function approximation is used to reduce learning time and to reduce weights stored in memory. Previous studies using only regular reinforcement learning took longer time and more amount of weights stored in memory. The artificial neural network architecture (ANN) used in this study is an ANN for each action. The results show that combining Q-learning and backpropagation can reduce agent's learning time to play Flappy Bird up to 92% and reduce the weights stored in memory up to 94%, compared to regular Q-learning only. Although the learning time and the weights stored are reduced, Q-learning combined with backpropagation have the same ability as regular Q-learning to play Flappy Bird game.**

***Intisari*—Makalah ini mengimplementasikan *Q-learning* yang dikombinasikan dengan *backpropagation* pada kasus agen (*agent*) yang belajar memainkan permainan *Flappy Bird*. Mengombinasikan *Q-learning* dengan *backpropagation* dimaksudkan untuk memprediksi nilai *value-function* tiap aksi (*action*) atau biasa disebut *value-function approximation*. Penggunaan *value-function approximation* diharapkan bisa mempercepat waktu pembelajaran dan mengurangi bobot yang disimpan, karena dari hasil penelitian sebelumnya dibutuhkan waktu yang lama dan banyaknya bobot yang disimpan di memori ketika hanya digunakan *reinforcement learning* saja. Arsitektur *artificial neural network* (ANN) yang digunakan adalah satu ANN pada masing-masing kemungkinan aksi. Berdasarkan hasil pengujian, diperoleh kesimpulan bahwa implementasi *Q-learning* yang dikombinasikan dengan *backpropagation* dapat membuat waktu pembelajaran agen untuk memainkan *Flappy Bird* lebih cepat hingga mencapai 92% dan dapat mengurangi bobot yang disimpan di memori hingga 94% jika dibandingkan dengan penggunaan *Q-learning* saja. Walaupun waktu pembelajaran lebih cepat dan bobot yang disimpan berkurang, *Q-learning* yang dikombinasikan dengan *backpropagation* memiliki kemampuan yang sama dengan penggunaan *Q-learning* saja untuk memainkan permainan *Flappy Bird*.**

***Kata Kunci— Flappy Bird, Q-Learning, Value-Function Approximation, Artificial Neural Netowrk, Backpropagation***

[1]*Alumni, Universitas Komputer Indonesia, Jl. Dipati Ukur no. 112 – 116 Bandung 40132 INDONESIA (tlp: 022-2533754; fax: 022-2504119; e-mail: ard333.ardiansyah@gmail.com)*

[2]*Dosen, Universitas Komputer Indonesia, Jl. Dipati Ukur no. 112 – 116 Bandung 40132 INDONESIA (tlp: 022-2533754; fax: 022-2504119; e-mail: ednawati.rainarli@email.unikom.ac.id)*

## I. PENDAHULUAN

*Flappy Bird* adalah permainan yang mengharuskan pemainnya mengontrol seekor burung untuk melewati celah antara dua pipa yang datang dan tidak terbatas dengan ketinggian yang berbeda-beda tanpa menabraknya atau jatuh ke tanah. Beberapa penelitian yang berkaitan dengan *reinforcement learning* dan *Flappy Bird* sudah pernah dilakukan sebelumnya. Sebuah penelitian membandingkan beberapa metode *machine learning* untuk membuat agen yang dapat memainkan permainan *Flappy Bird* [1]. Dari hasil penelitian ini diperoleh kesimpulan bahwa *reinforcement learning* memberikan hasil yang lebih baik daripada metode *machine learning* lainnya. Akan tetapi, penggunaan metode *reinforcement learning* membutuhkan proses pembelajaran yang lama dan komputasi yang besar dikarenakan banyaknya bobot yang disimpan. Penelitian lain menggunakan metode *Q-learning* sebagai salah satu metode dalam *reinforcement learning* yang menghasilkan rancangan keadaan (*state*) yang lebih efisien. Namun, pembelajaran dengan *Q-learning* masih membutuhkan waktu pembelajaran yang lama dan jumlah memori yang besar untuk menyimpan bobot [2].

Lebih lanjut, beberapa penelitian yang berkaitan dengan *Q-learning* memanfaatkan penggunaan *artificial neural networks* (ANN) *backpropagation* sebagai *function approximator*. ANN digunakan sebagai *function approximator* pada simulasi robot untuk mencari jalan di lingkungan statis [3] - [5]. Simulasi robot ini berhasil menemukan solusi pada proses pembelajarannya. Pada penelitian lain, penggunaan ANN sebagai *function approximator* pada simulasi robot untuk mencari cahaya memberikan hasil yang lebih buruk dari *table representation* [6]. Namun, penelitian tersebut menyampaikan banyak aspek yang dapat ditingkatkan untuk membuat implementasi ANN menjadi lebih baik, seperti pemilihan perancangan arsitektur ataupun pemilihan *activation function*.

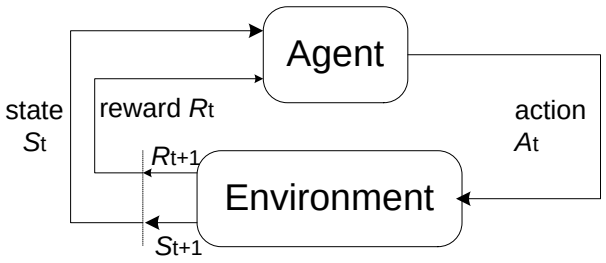
Pembelajaran agen dalam permainan *Flappy Bird* juga memiliki ukuran keadaan yang besar, terlihat dari posisi burung dan posisi pipa yang selalu berubah setiap waktu. Perubahan beberapa *pixel* saja merupakan keadaan yang berbeda. Oleh karena itu, dalam makalah ini akan dikaji seberapa besar pengaruh penggunaan *backpropagation* sebagai *value function approximation* pada proses pembelajaran agen *flappy bird* yang belajar untuk melewati pipa dengan menggunakan algoritme *Q-learning*. Adapun parameter yang digunakan untuk mengukur keberhasilan penggunaan *backpropagation* yang dikombinasikan dengan algoritme *Q-learning* dilihat dari waktu pembelajaran dan jumlah memori yang digunakan. Dalam makalah ini ditunjukkan bahwa penggunaan *backpropagation* dan algoritme *Q-learning* akan membuat waktu pembelajaran

menjadi lebih cepat dan dapat mengurangi jumlah bobot yang disimpan di memori, tetapi tidak mengurangi kemampuan agen dalam memainkan permainan *Flappy Bird* apabila dibandingkan dengan penggunaan *Q-learning* saja.

Pada bagian awal dari penulisan ini akan disampaikan tentang algoritme *Q-learning* dan *backpropagation* serta penggunaan *backpropagation* sebagai *value function approximation.* Selanjutnya, dijelaskan pula alur pembelajaran agen *Flappy Bird* yang akan diuji. Pada bagian hasil ditunjukkan perbandingan pengujian dari penggunaan *Q-learning* saja dan *Q-learning* dengan *backpropagation*, dengan mengukur waktu pembelajaran dan jumlah memori yang digunakan.

## II. *REINFORCEMENT LEARNING*

*Reinforcement learning* adalah metode pembelajaran untuk memetakan setiap keadaan terhadap aksi yang dipilih untuk memaksimalkan *reward* yang diterima [7]. Setiap keadaan dan aksi yang dipetakan diberi nilai yang biasanya disebut *value function* yang direpresentasikan sebagai sebuah tabel. Agen yang melakukan pembelajaran tidak diberitahu aksi apa yang harus dipilih, tetapi harus menemukan sendiri aksi apa yang menghasilkan *reward* paling besar dengan mencobanya. Karakteristik lain dari *reinforcement learning* adalah mempertimbangkan masalah dan mengarahkan pada tujuan saat berinteraksi dengan lingkungan yang tidak pasti. Gbr. 1 menunjukkan interaksi agen dengan lingkungan (*environment*).




Gbr. 1 Interaksi antara agen dan lingkungan.

Secara spesifik agen dan lingkungan berinteraksi di setiap waktu $t$ = 0,1,2,3,..., dan di setiap waktu $t$, agen mendapat keadaan $S_t \in \hat{S}$, dengan $\hat{S}$ adalah himpunan dari keadaan yang mungkin terjadi. Selanjutnya agen akan memilih aksi $A_t \in \hat{A}(S_t)$, dengan $\hat{A}(S_t)$ adalah himpunan dari aksi pada keadaan $S_t$. Selanjutnya agen mendapatkan *reward* $R_t$ lalu mendapat keadaan baru $S_{t+1}$. Beberapa algoritme dalam *reinforcement learning* antara lain *Temporal-Difference Learning*, *Q-learning*, dan SARSA [7].

Dalam *reinforcement learning*, pemilihan aksi untuk setiap keadaan yang didapat saat pembelajaran sangat mempengaruhi keberhasilan pembelajaran. Pemilihan aksi secara acak biasa disebut *exploration*, sedangkan pemilihan aksi berdasarkan *value function* terbesar biasa disebut *exploitation*. Salah satu cara untuk pemilihan aksi tersebut adalah ε-*greedy*. Dalam ε-*greedy*, agen akan memilih aksi secara acak dengan probabilitas ε dan akan memilih aksi dengan nilai terbesar dengan probabilitas (1-ε).

### *A. Q-learning*

*Q-learning* adalah pengembangan dari *Temporal-Difference* yang juga biasa dikenal *off-policy TD control*. *Q-learning* merupakan salah satu terobosan paling penting dalam *reinforcement learning* [7]. *Q-learning* melakukan *update* terhadap *action-value function* seperti pada (1).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)) \quad (1)$$

$Q(S_t, A_t)$ adalah *action-value function* pada keadaan ke-$t$ ($S_t$) dan aksi ke -$t$ ($A_t$). $R_{t+1}$ adalah *reward* yang diperoleh pada waktu ke-$t$+1, sedangkan $\max_a Q(S_{t+1}, a)$ adalah nilai maksimum dari *action-value function* pada keadaan ke-$t$+1 untuk suatu aksi $a$. Parameter α adalah *learning rate*, $0 < \alpha \leq 1$, menyatakan ukuran laju perubahan nilai lama yang akan digantikan dengan nilai baru. Nilai γ, $0 \leq \gamma \leq 1$, menentukan nilai dari *reward* di masa depan. Semakin kecil nilai γ, maka agen akan semakin mementingkan *reward* dekat, bukan *reward* di masa depan.

Tidak seperti *TD-learning* yang memperbaharui *value function* berdasarkan keadaan selanjutnya, *Q-learning* memperbaharui *value function* berdasarkan nilai *action-value function* terbesar di keadaan selanjutnya.

### *B. Algoritme Q-learning*

Pada Gbr. 2 ditunjukkan algoritme *Q-learning* secara lengkap dalam bentuk prosedural. Proses pembelajaran *Q-learning* diawali dengan menginisialisasi nilai *action-value function* $Q(S,A)$ dan proses perulangan pemilihan aksi $A$ serta nilai *action-value function* diperbaharui sampai kondisi pembelajaran yang digunakan terpenuhi.

***Q-Learning***

Menginisialisasi $Q(S,A)$ dengan nilai sembarang
Untuk setiap peristiwa dilakukan perulangan:
  Menginisialisasi $S$ (dari *environment*)
  Untuk setiap langkah dilakukan perulangan:
    Dipilih *action* $A$ pada *state* $S$ (contohnya dengan *ε-greedy*)
    Dilakukan *action* $A$, lalu didapatkan *reward* $R$ dan *next state* $S'$
    *Action-value function* diperbaharui pada *state* $S$ dan *action* $A$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

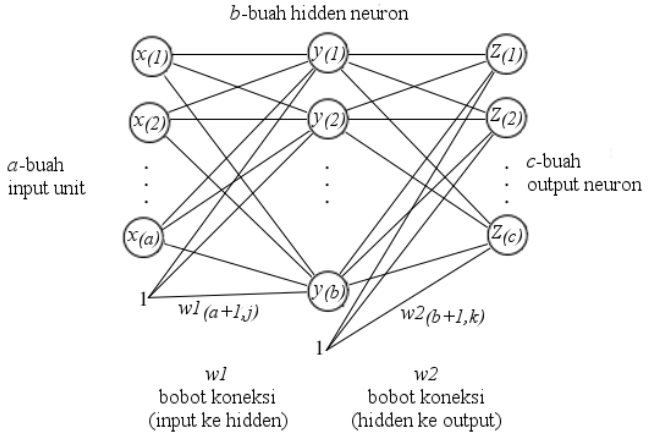
    Diperbaharui *next state* menjadi *current state* $S \leftarrow S'$
  Sampai $S$ adalah akhir dari peristiwa
Sampai proses pembelajaran selesai

Gbr. 2 Algoritme *Q-Learning.*

## III. *ARTIFICIAL NEURAL NETWORK (ANN)*

ANN adalah model dari kemampuan jaringan saraf biologis untuk memproses informasi [8]. Jaringan saraf biologis adalah sistem yang mengatur dirinya sendiri dan masing-masing neuron juga mengorganisasi struktur dirinya dalam kemampuan memproses informasi dalam berbagai cara. ANN terdiri atas beberapa neuron yang saling terhubung. Pada umumnya, struktur ANN dibagi menjadi tiga, yaitu *input layer*, *hidden layer*, dan *output layer*. Jumlah neuron atau unit di setiap lapisan (*layer*) disesuaikan dengan informasi yang akan diproses. Jumlah *input layer* dan *output layer* adalah satu, sedangkan jumlah *hidden layer* bisa satu atau lebih. Dalam

kebanyakan kasus pembelajaran dapat dinyatakan lebih mudah dengan menambahkan *bias*.




Gbr. 3 Ilustrasi ANN dengan *bias*.

Gbr. 3 menunjukkan ilustrasi ANN dengan *a*-buah masukan unit *x* di *input layer*, *b*-buah neuron *y* di *hidden layer*, *c*-buah neuron *z* di *output layer*, bobot $w1_{(i,j)}$ yang menghubungkan unit di *input layer* $x_{(i)}$ ke neuron di *hidden layer* $y_{(j)}$, bobot $w2_{(j,k)}$ yang menghubungkan neuron di *hidden layer* $y_{(j)}$ ke neuron di *output layer* $z_{(k)}$, dengan $i = 1,2,\dots,a$ , $j = 1,2,\dots,b$ dan $k = 1,2,\dots,c$. Jadi, ada sebanyak *c* buah nilai keluaran yang diharapkan.

### A. Backpropagation

*Backpropagation* adalah salah satu algoritme *supervised learning* yang digunakan dalam ANN. *Backpropagation* mencari kombinasi bobot untuk meminimalkan kesalahan keluaran untuk dianggap menjadi solusi yang benar [8]. Secara sederhana, *backpropagation* dilakukan dalam dua tahap sebagai berikut.

- *Feed-forward*, yaitu proses pelatihan pola yang akan diset ke setiap unit di *input layer*, lalu keluaran yang dihasilkan ditransmisikan ke lapisan selanjutnya, terus sampai *output layer*.
- *Backpropagation*, yaitu proses penyesuaian setiap bobot berdasarkan keluaran yang diharapkan, agar dihasilkan galat (*error*) yang minimal, mulai dari bobot yang terhubung ke neuron keluaran, lalu terus mundur sampai ke *input layer*.

### B. Algoritme Backpropagation

Detail dari algoritme *backpropagation* ditunjukkan pada Gbr. 4 yang didasarkan pada ilustrasi ANN pada Gbr. 3. Alur *backpropagation* diawali dengan inisialisasi bobot *w*1 dan *w*2 untuk setiap pola dan dilakukan proses perulangan *feed-forward* dan *backpropagation* sampai kondisi henti dipenuhi. Nilai $w1_{(i,j)}$ adalah nilai bobot *w*1 yang menghubungkan neuron $x_{(i)}$ menuju neuron $y_{(j)}$. Begitu pula untuk $w2_{(j,k)}$ adalah bobot *w*2 yang menghubungkan neuron $y_{(j)}$ menuju neuron $z_{(k)}$. Parameter $t_{(k)}$ adalah target nilai dari neuron *k* untuk suatu pola. Parameter $\alpha$ adalah nilai *learning rate*, sedangkan $\Delta w1_{(i,j)}$ adalah besarnya perubahan bobot *w*1 dari neuron $x_{(i)}$ menuju neuron $y_{(j)}$ dan $\Delta w2_{(j,k)}$ adalah besarnya perubahan bobot *w*2 yang menghubungkan neuron $y_{(j)}$ menuju neuron $z_{(k)}$.

*Backpropagation*

Dilakukan proses inisialisasi semua bobot $w_1$ dan $w_2$ dengan nilai kecil secara berulang:

Diulangi untuk setiap pola:

Atur pola ke *input* unit *x*

*Feed-forward*

Dihitung *output y* dari setiap neuron di *hidden layer*

$$y\_net_{(j)} = \sum_{i=1}^{a+1} w1_{(i,j)} x_{(i)}$$

$$y_{(j)} = f(y\_net_{(j)})$$

Dihitung *output* dari setiap neuron di *output layer*

$$z\_net_{(k)} = \sum_{j=1}^{b+1} w2_{(j,k)} y_{(j)}$$

$$z_{(k)} = f(z\_net_{(k)})$$

*Backpropagation*

Dihitung perhitungan semua bobot *w*2

$$\delta\_o_{(k)} = (t_{(k)} - z_{(k)}) f'(z\_net_{(k)})$$

$$\Delta w2_{(j,k)} = \alpha.\delta\_o_{(k)}.y_{(j)}$$

Dihitung perubahan semua bobot *w*1

$$\delta\_h\_net_{(j)} = \sum_{k=1}^{c} w2_{(j,k)} \delta\_o_{(k)}$$

$$\delta\_h_{(j)} = \delta\_h\_net_{(j)} f'(y\_net_{(j)})$$

$$\Delta w1_{(i,j)} = \alpha\delta\_h_{(j)} x_{(i)}$$

Dilakukan perubahan nilai bobot *w*1 dan *w*2

$$w2_{(j,k)} = w2_{(j,k)} + \Delta w2_{(j,k)}$$

$$w1_{(i,j)} = w1_{(i,j)} + \Delta w1_{(i,j)}$$

Sampai kondisi henti terpenuhi

Gbr. 4 Algoritme *backpropagation.*

Dari Gbr. 4 terlihat bahwa nilai $y_{(j)}$ dan $z_{(k)}$ dihasilkan dari sebuah fungsi, yaitu *activation function f(x)*. Salah satu *activation function* yang populer untuk jaringan *backpropagation* adalah *bipolar sigmoid* dengan rentang nilai (-1,1), yang didefinisikan sebagai (2). Untuk tahap *backpropagation* digunakan fungsi turunannya *f'(x)*, yaitu (3).

$$f(x) = (2/(1+e^{-x})) - 1 \quad (2)$$

$$f'(x) = \tfrac{1}{2}.(1+f(x))(1-f(x)) \quad (3)$$

## IV. VALUE FUNCTION APPROXIMATION

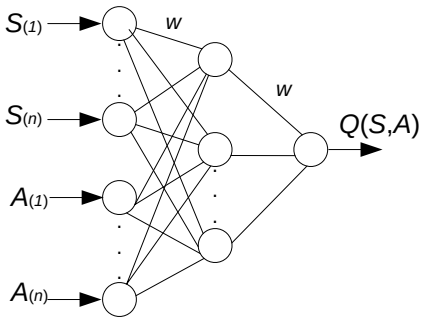
Dalam *reinforcement learning*, setiap nilai dari *value function* direpresentasikan sebagai sebuah tabel. Masalah yang terjadi pada beberapa kasus ketika pasangan keadaan dan aksi menjadi sangat besar adalah kebutuhan memori yang besar dan waktu pembelajaran yang lama. Dari masalah tersebut, dibutuhkan generalisasi cara mendapatkan pengalaman yang baik dari bagian keadaan yang terbatas untuk mendapatkan perkiraan nilai yang baik. Jenis generalisasi yang dibutuhkan biasa disebut *function approximation*. *Function approximation*

merupakan bagian dari *supervised learning*, topik utama yang banyak diteliti, salah satunya adalah ANN [7].

Nilai dari perkiraan *value function* tidak direpresentasikan lagi sebagai tabel, tapi sebagai fungsi berparameter dengan tambahan vektor bobot *w*. Sebagai contoh, *value function* diperkirakan dengan ANN, maka *w* adalah semua bobot yang menghubungkan masing-masing unit atau neuron.

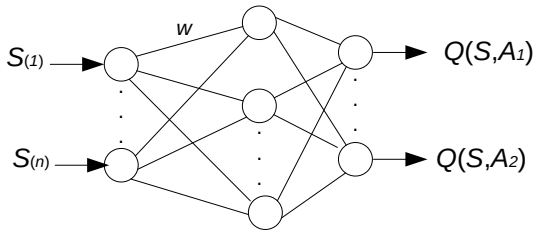
### *A. Backpropagation sebagai Value Function Approximation*

ANN dapat digunakan sebagai *value function approximation*. Nilai dari *value function* diperoleh dengan proses *feed-forward* dan mengambil nilai pada neuron di *output layer*, sedangkan proses *update value function* dilakukan dengan *backpropagation* [3] - [6]. Gbr. 5, Gbr. 6 dan Gbr. 7 menunjukkan beberapa alternatif arsitektur ANN yang digunakan sebagai *value function approximation*. $S_{(1)}, \dots, S_{(n)}$ menunjukkan nilai keadaan ke-1 sampai ke-*n*, sedangkan $A_{(1)}, \dots, A_{(n)}$ menunjukkan nilai aksi ke-1 sampai ke-*n* dan *w* adalah bobot antar neuron. Keluaran dari proses ini adalah nilai *action-value function* $Q(S, A)$.



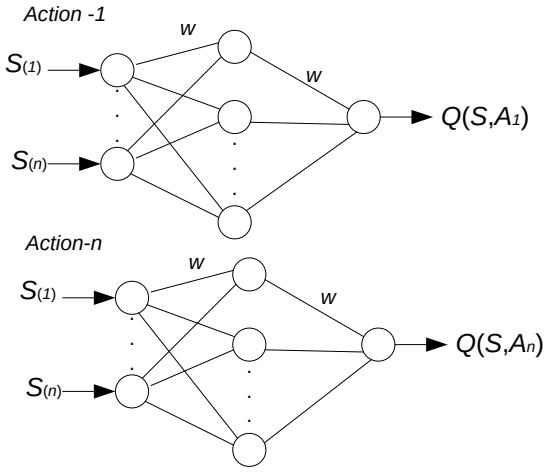

Gbr. 5 Arsitektur pertama.

Masukan arsitektur pada Gbr. 5 didapat dari keadaan dan aksi yang dipilih. Arsitektur ini digunakan pada beberapa penelitian, tetapi pada salah satu penelitian arsitektur ini memberikan hasil yang tidak begitu baik [3], [6]. Alternatif arsitektur lain ditampilkan pada Gbr. 6, berbeda dengan arsitektur pada Gbr. 5 yang mendefinisikan masukan dari keadaan yang didapat dan aksi yang dipilih.




Gbr. 6 Arsitektur kedua.

Masukan pada Gbr. 6 didapat dari keadaan yang didapat saja, hanya jumlah dari neuron pada *output layer* berjumlah *n*, dengan *n* adalah jumlah kemungkinan aksi. Arsitektur ini telah digunakan pada beberapa penelitian [4], [5]. Alternatif arsitektur lain ditampilkan pada Gbr. 7.




Gbr. 7 Arsitektur ketiga.

Masukan pada Gbr. 7 didapat dari keadaan saja, hanya jumlah dari ANN berjumlah *n*, dengan *n* adalah jumlah kemungkinan aksi. Arsitektur ini telah digunakan dan disimpulkan bahwa arsitektur ini lebih mudah digunakan untuk melakukan kontrol dibandingkan arsitektur pada Gbr 6 [4]. Arsitektur ini juga direkomendasikan dalam penelitian lain [6]. Oleh karena itu, dalam makalah ini digunakan pula arsitektur ketiga yang ditunjukkan pada Gbr. 7.

## V. *Q-LEARNING* YANG DIKOMBINASIKAN DENGAN *BACKPROPAGATION*

*Flappy Bird* mempunyai ukuran keadaan yang besar, mulai dari posisi burung dan posisi pipa yang selalu berubah setiap waktu, perubahan beberapa *pixel* saja merupakan keadaan yang berbeda, sehingga penggunaan *function approximation* diharapkan akan membuat proses pembelajaran lebih efisien dalam waktu dan dapat mengurangi jumlah bobot yang disimpan di memori tetapi tidak mengurangi kemampuan agen dalam memainkan permainan *Flappy Bird* jika dibandingkan dengan penggunaan *Q-learning* saja.

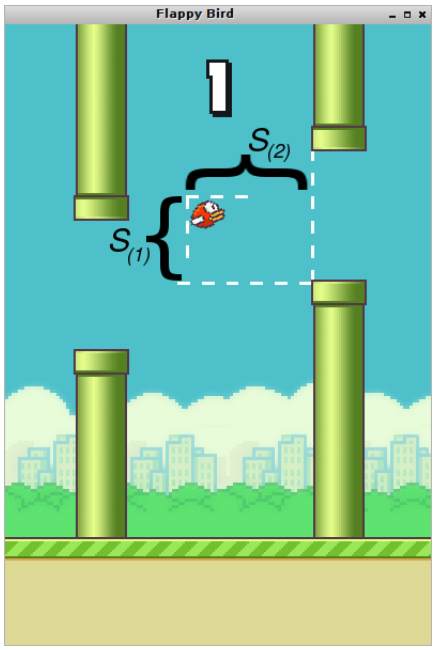
Beberapa masukan yang akan digunakan pada makalah ini akan dijelaskan sebagai berikut.

*1) Keadaan:* Keadaan yang digunakan dalam sistem ini adalah

- selisih antara burung dan pipa bagian bawah di depannya, $S_{(1)}$, dan
- jarak antara burung dan pipa di depannya, $S_{(2)}$.

Gbr. 8 menunjukan ilustrasi nilai yang digunakan untuk keadaan. Selisih antara burung dan pipa bagian bawah di depannya didapat dari kordinat *y* pipa bagian bawah yang ada di depannya dikurang kordinat *y* burung. Sedangkan jarak antara burung dan pipa di depannya didapatkan dari nilai kordinat *x* pipa di depannya dikurangi kordinat *x* burung. Dua nilai tersebut dibagi dengan 100 sebelum dijadikan masukan ke ANN, sehingga perubahan bobotnya tidak terlalu jauh.

Nilai 100 dipilih karena ukuran *frame* bernilai ratusan *pixel* sehingga nantinya akan menjadi nilai satuan.




Gbr. 8 Ilustrasi nilai untuk keadaan.

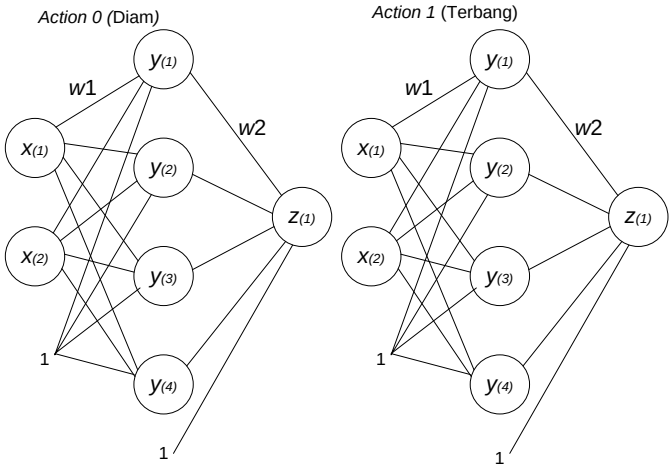
*2) Aksi*: Aksi yang digunakan adalah terbang dan diam. Nilai aksi didefinisikan sebagai nilai *integer*, terbang = 1 dan diam = 0.

*3) Reward: Reward* yang digunakan adalah

- -1 ketika burung mati,
- +0.1 ketika burung hidup, dan
- +1 ketika *score* bertambah.

Penentuan *reward* dipengaruhi *activation function* yang digunakan. Di sini digunakan fungsi *sigmoid bipolar* seperti pada (2) dan turunannya pada (3).

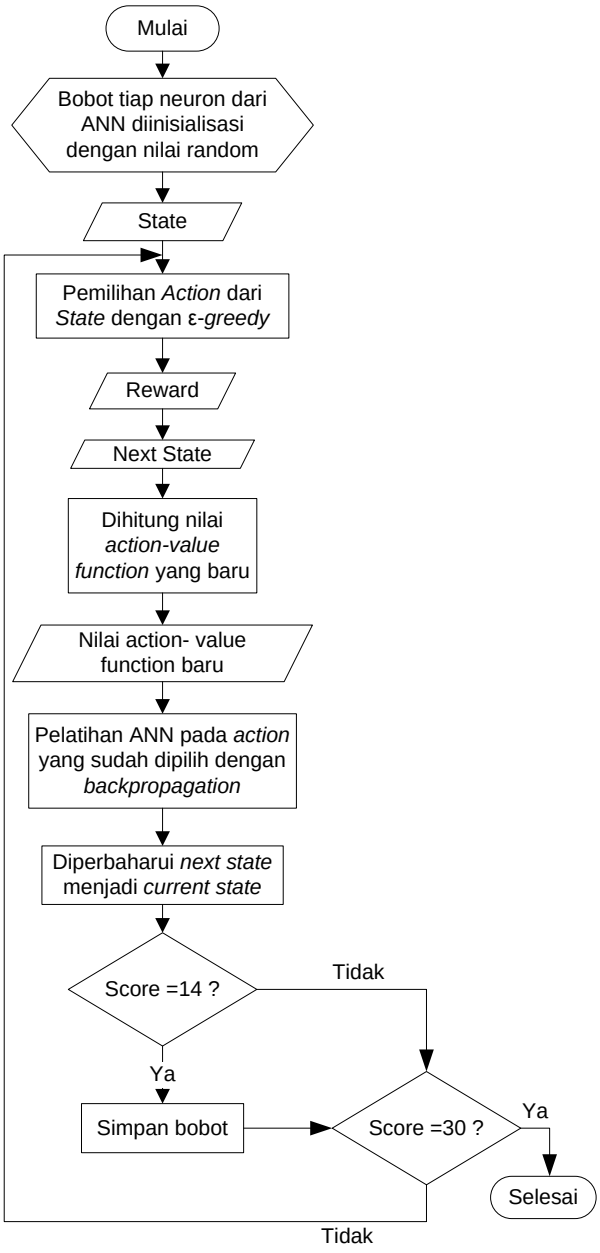
Arsitektur ANN yang digunakan ditunjukkan pada Gbr. 9.




Gbr. 9 Arsitektur ANN yang digunakan.

Gbr. 9 menunjukkan arsitektur untuk satu ANN tiap kemungkinan aksi. Masing-masing ANN memiliki dua masukan unit yang didapat dari keadaan, empat neuron tersembunyi, dan satu neuron keluaran sebagai nilai *action-value function*.

Adapun proses pembelajaran *Q-learning* yang dikombinasikan dengan *backpropagation* pada agen yang memainkan permainan *Flappy Bird* bisa dilihat pada Gbr. 10.




Kondisi Tambahan: Setiap 2 menit akan dilakukan pengecekan apakah dalam 1 menit tersebut tidak pernah mendapat score? Jika iya maka akan dilakukan inisialisasi ulang

Gbr. 10 Diagram alir pembelajaran.

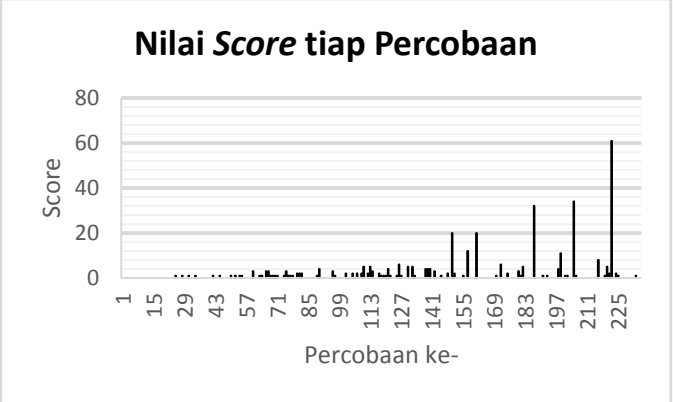
Detail dari proses pembelajaran *Q-learning* yang dikombinasikan dengan *backpropagation* adalah sebagai berikut.

- Bobot masing-masing ANN diinisialisasi dengan nilai acak. Pertama, agen melakukan inisialisasi bobot masing-masing ANN dengan nilai acak antara 0 sampai 1.

- Setelah didapatkan nilai acak, selanjutnya ditentukan keadaan dari agen. Keadaan yang didapat menunjukkan posisi vertikal dan horizontal dari agen.
- Langkah berikutnya adalah agen memilih aksi yang akan dilakukan dengan ε-*greedy*. Nilai ε yang digunakan adalah 0.0, yang berarti bahwa agen akan selalu memilih aksi dengan nilai *action-value function* terbesar. Pemilihan nilai tersebut digunakan karena pada kasus ini aksi akan diminta sangat cepat tiap *frame*-nya. Jika ada pemilihan aksi acak (*random action*) dan aksi yang diambil salah, maka akan sangat mengganggu proses pembelajaran. Agen akan sulit mencari solusi yang optimal.
- Agen akan mendapat *reward* berupa poin untuk setiap kondisi pipa yang dilewati. Setelah poin bertambah, dilanjutkan dengan keadaan berikutnya (*next state*) dari aksi yang sudah dilakukan.
- Selanjutnya, agen menghitung nilai *update action-value function* seperti pada (1) yang nantinya digunakan sebagai keluaran yang diharapkan pada proses *backpropagation*. *Learning rate* yang digunakan adalah 0.9. Nilai ini dipilih karena diasumsikan dibutuhkan perubahan yang besar pada setiap nilai untuk mencapai solusi optimal. Nilai *discount rate* yang digunakan adalah 0.9, bertujuan supaya agen lebih mementingkan *reward* di masa depan daripada *reward* yang dekat.
- Digunakan ANN *backpropagation* untuk memilih aksi yang akan digunakan yaitu terbang atau diam. Adapun keluaran yang diharapkan diperoleh dari nilai *update action-value function*. Untuk nilai *learning rate* masih digunakan 0.9.
- Setelah dipilih aksi dari agen, selanjutnya agen akan melakukan *update next state*. Adapun *update next state* yang didapat akan dijadikan sebagai *current state*.
- Terakhir adalah pengecekan kondisi penyimpanan bobot dan kondisi henti. Kondisi henti diambil berdasarkan *score* yang didapat. Batas *score* yang dijadikan acuan adalah 30. Ketika sudah mencapai batas *score*, maka proses pembelajaran akan dihentikan. Namun, sebelumnnya bobot saat *score* 14 disimpan terlebih dahulu dan nantinya akan digunakan setelah proses pembelajaran selesai. Jika *score* tidak mencapai 30, maka akan kembali ke langkah 3.

Kondisi henti *score* = 30 ditentukan berdasarkan hasil percobaan yang dilakukan sebelumnya. Gbr. 11 menunjukkan salah satu hasil dari beberapa kali percobaan agen dalam pembelajaran. Grafik tersebut menunjukkan terjadi beberapa kali agen mendapat *score* yang relatif besar (*score* 20, 32, 34, 61), tetapi setelah itu *score* dari percobaan-percobaan selanjutnya kecil beberapa kali. Hal ini disebabkan ANN mengalami pelatihan berlebih di beberapa keadaan, sehingga mempengaruhi pola keadaan lain yang tidak dilatih. Setelah beberapa kali mendapat *score* kecil agen kembali mendapat *score* yang relatif besar. Bobot 30 dipilih sebagai nilai tengah *score* yang relatif besar yaitu 61.

Pemilihan *score* 14 sebagai kondisi penyimpanan bobot dan nilai 30 sebagai batas *score* pada kondisi henti adalah sebagai hipotesis awal, karena nilai 30 dianggap sudah mencukupi dan tidak terlalu awal untuk diambil sebagai bobot. *Score* 14 diambil karena saat mendapat *score* 14 itu merupakan bobot ditengah saat nilai 30, karena sesaat sebelum mendapat *score* 15 masih dalam dalam kondisi *score* 14.




Gbr. 11 Percobaan sebelum penentuan kondisi henti.

Adapun proses tambahan adalah setiap 2 menit akan dilakukan pengecekan, jika dalam dua menit tersebut agen tidak mendapat *score* sama sekali, maka akan dilakukan inisialisasi ulang bobot ANN. Inisialisasi ulang dibutuhkan karena ada kondisi ketika pembelajaran tidak menemukan solusi hingga lebih dari 30 menit berdasarkan percoban sebelumnya. Waktu 2 menit dipilih karena dianggap tidak terlalu cepat dan tidak terlalu lama untuk memperkirakan bahwa proses pembelajaran tidak akan menemukan solusi.

Jika proses pembelajaran selesai, agen akan menggunakan bobot yang telah disimpan untuk proses pengujian. Agen tidak akan lagi memilih aksi dengan ε-*greedy*, tetapi akan selalu memilih aksi dengan nilai *action-value function* terbesar. Agen juga tidak lagi menghitung nilai *update* untuk *action-value function* dan tidak lagi melatih ANN.

## VI. PENGUJIAN

### *A. Pengujian Waktu Pembelajaran*

Pengujian dilakukan sampai agen berhasil mencapai *score* 1000. Adapun waktu pembelajaran dari *Q-learning* yang dikombinasikan dengan *backpropagation* disajikan pada Tabel I.

Dari sepuluh kali percobaan, rata-rata waktu dari *Q-learning* yang dikombinasikan dengan *backpropagation* adalah 541 detik atau 9 menit 1 detik. Untuk penggunaan *Q-learning* saja, hasil pengujian adalah semakin lama pelatihan akan semakin baik. Adapun hasil pengujian waktu pembelajaran *Q-learning* ditunjukkan pada Tabel II.

Jika dibandingkan dengan *Q-learning* yang dikombinasikan dengan *backpropagation*, maka perbedaan penggunaan waktu pembelajarannya adalah:

- 541 detik / 7200 detik * 100 = 7,514%
- 100% - 7,514% = 92,486%

Waktu pembelajaran *Q-learning* yang dikombinasikan dengan *backpropagation* bisa berkurang hingga lebih dari 92% jika dibandingkan dengan penggunaan *Q-learning* saja.

TABEL I
WAKTU PEMBELAJARAN *Q-LEARNING* YANG DIKOMBINASIKAN DENGAN *BACKPROPAGATION*

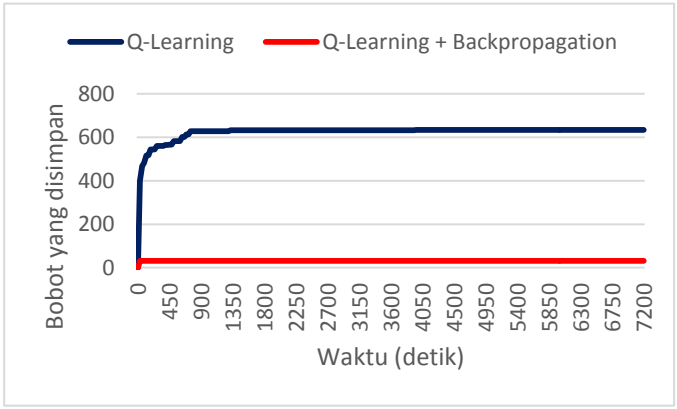
| Percobaan ke | Waktu (detik) | *Score* |
|---|---|---|
| 1 | 482 | > 1000 |
| 2 | 662 | > 1000 |
| 3 | 366 | > 1000 |
| 4 | 649 | > 1000 |
| 5 | 637 | > 1000 |
| 6 | 441 | > 1000 |
| 7 | 342 | > 1000 |
| 8 | 444 | > 1000 |
| 9 | 585 | > 1000 |
| 10 | 802 | > 1000 |
| Rata-rata | 541 | |

TABEL II
WAKTU PEMBELAJARAN *Q-LEARNING*

| Waktu Pembelajaran | *Score* |
|---|---|
| 15 menit | 94.9 |
| 30 menit | 131.9 |
| 60 menit | 638.6 |
| 90 menit | 571 |
| 105 menit | 574.7 |
| 120 menit | >1000 |

### *B. Pengujian Bobot yang Disimpan*

Jumlah bobot yang disimpan dalam memori ditunjukkan pada Gbr. 12.




Gbr. 12 Grafik jumlah bobot yang disimpan.

Gbr. 12 menunjukkan grafik jumlah bobot yang disimpan di memori dari 0 s hingga 7200 s (120 menit). *Q-learning* saja menghasilkan jumlah bobot sebanyak 634 bobot di detik ke 7200, sedangkan *Q-learning* yang dikombinasikan dengan *backpropagation* hanya menghasilkan 32 bobot dari awal hingga akhir. Jika dibandingkan dengan *Q-learning* yang dikombinasikan dengan *backpropagation*, maka perbedaan penggunaan bobot yang disimpan di memori adalah

- 32 bobot / 634 bobot * 100 = 5,047%
- 100% - 5,047% = 94,953%

Bobot yang disimpan *Q-learning* yang dikombinasikan dengan *backpropagation* bisa berkurang hingga lebih dari 94% jika dibandingkan dengan penggunaan *Q-learning* saja.

## VII. KESIMPULAN

Berdasarkan hasil yang diperoleh, dapat diambil kesimpulan bahwa implementasi *Q-learning* yang dikombinasikan dengan *backpropagation* dapat membuat waktu pembelajaran agen untuk memainkan *Flappy Bird* lebih cepat hingga 92% dan dapat mengurangi bobot yang disimpan di memori hingga 94%, jika dibandingkan dengan penggunaan *Q-learning* saja. Walaupun waktu pembelajaran lebih cepat dan bobot yang disimpan berkurang, tetapi *Q-learning* yang dikombinasikan dengan *backpropagation* memiliki kemampuan yang sama dengan *Q-learning* saja untuk memainkan permainan *Flappy Bird*. Dalam implementasinya, dibutuhkan proses inisialisasi ulang bobot ANN ketika proses pembelajaran tidak berjalan baik. *Source code* sistem ini bisa diakses di <https://github.com/ard333/FlappyBird/>.

## REFERENSI

[1] Y. Shu *et al*, "Obstacles Avoidance with Machine Learning Control Methods in Flappy Birds Setting," Univ. of Stanford, CS229 Machine Learning Final Projects Stanford University, 2014.

[2] S. Vaish. Flappy Bird RL by SarvagyaVaish. [Online], http://sarvagyavaish.github.io/FlappyBirdRL/, tanggal akses 8 Maret 2016.

[3] M. Hatem and F. Abdessemed, "Simulation of the Navigation of a Mobile Robot by the Q-Learning using Artificial Neuron Networks," *CEUR Workshop Proceeding Conférence Internationale sur l'Informatique et ses Applications*, vol. 547, paper 81, 2009.

[4] R. Jaksa *et al*. "Backpropagation in Supervised and Reinforcement Learning for Mobile Robot Control," *Proceedings of Computational Intelligence for Modelling, Control and Automation*, 1999.

[5] B. Huang *et al*. "Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance," *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, hal. 85-89, 2005.

[6] S. Dini and M. Serrano, "Combining Q-Learning with Artificial Neural Networks in an Adaptive Light Seeking Robot," Swarthmore College, CS81 Adaptive Robotics Final Projects Swarthmore College, 2012.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, London, England: MIT press, 1998.

[8] R. Rojas, *Neural Networks A Systematic Introduction*, Berlin, Germany: Springer-Verlag, 1996.