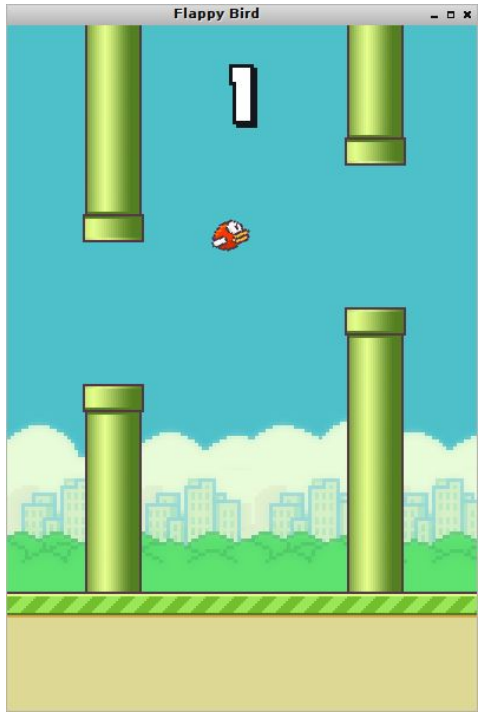
BAB 2

LANDASAN TEORI

2.1. *Flappy Bird*

Flappy Bird adalah *game* 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. *Flappy Bird* dikembangkan oleh Dong Nguyen pada tahun 2013 pada *platform mobile* [8].



Gambar 2.1 *Flappy Bird*

Pada *release* pertamanya *Flappy Bird* menjadi peringkat pertama di *Apple App Store* selama berbulan-bulan sampai akhirnya dihapus oleh Dong Nguyen sendiri karena menurutnya itu produk adiktif. Setelah itu terjadi, banyak *clone* bermunculan untuk menggebrak pasar. Gambar 2.1 menampilkan salah satu *clone* dari *Flappy Bird* yang juga digunakan pada penelitian ini.

2.2. *Artificial Intelligence*

Artificial intelligence adalah bidang yang mempelajari sintesis dan analisis dari komputasi *agent* yang bertindak cerdas [9]. *Agent* adalah sesuatu yang bertindak didalam linkungan(*environment*) untuk melakukan sesuatu. *Agent* bertindak dalam linkungannya dan hanya memiliki akses ke pengetahuan sebelumnya(*prior knowledge*), riwayat pengalaman(*history of observation*), dan tujuan(*goal*). *Agent* yang cerdas adalah sistem simbol fisik yang melakukan manipulasi untuk menentukan apa yang harus dilakukan.

Seorang desainer dari *agent* yang cerdas harus bisa menggambarkan *environment*, seberapa jauh perencanaan, struktur dari tujuan, *agent* lain, bagaimana belajar dari pengalaman, dan harus mengerti bahwa agent memiliki keterbatasan sumberdaya untuk melakukan proses komputasinya. Selain itu desainer juga harus merumuskan dan menguji hipotesis tentang apa yang diperlukan untuk membangun *agent* yang bertindak cerdas dan merancang, membangun, dan bereksperimen dengan sistem yang melakukan tugas-tugas yang memerlukan kecerdasan.

2.3. *Machine Learning*

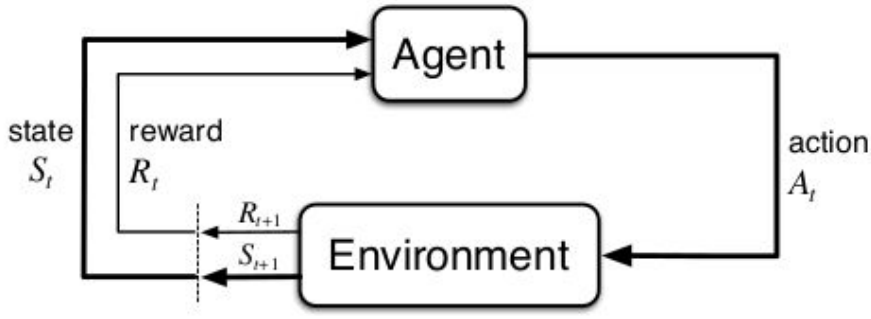
Machine learning adalah salah satu disiplin ilmu di bidang *artificial intelligence* untuk membangun mesin yang cerdas yang bisa belajar dan beradaptasi dengan lingkungannya [10]. Untuk membangun mesin yang cerdas desainer harus menyadari bahwa mesin ini harus belajar dan beradaptasi dengan lingkungan mereka. Bidang *machine learning* berkembang sangat cepat dan sangat beragam.

Ada beberapa alasan mengapa bidang *machine learning* bekembang dengan cepat. Alasan pertama *machine Learning* muncul dari bidang *artificial intelligence* tapi dengan cepat menggabungkan ide-ide dari bidang yang beragam seperti statistik, teori informasi, teori kontrol, *theoretical neuroscience* dan masih banyak lagi. Alasan yang kedua adalah pertumbuhan yang besar dari data yang

tersedia dan kemampuan komputer yang semakin meningkat. *Machine learning* dapat dibagi menjadi beberapa jenis, yaitu *supervised learning*, *unsupervised learning*, *semi-supervised learning*, dan *reinfocement learning*.

2.4. *Reinforcement Learning*

Reinforcement learning adalah metode pembelajaran untuk memetakan setiap *state* terhadap *action* yang dipilih untuk memaksimalkan *reward* yang diterima [3]. Setiap *state* dan *action* yang dipetakan diberi nilai yang representasikan sebagai sebuah tabel. *Agent* yang melakukan pembelajaran tidak diberitahu *action* apa yang harus dipilih, melainkan harus menemukan sendiri *action* apa yang menghasilkan *reward* paling besar dengan mencobanya. Pada beberapa kasus, *action* dapat mempengaruhi tidak hanya pada *reward* yang secara langsung didapat, tetapi juga *reward* yang tertunda. Karakteristik lain dari *reinforcement learning* adalah mempertimbangkan masalah dan mengarahkan pada tujuan saat berinteraksi dengan lingkungan yang tidak pasti. *Agent* menerima *state*(representasi dari *enviroment*) dan memilih *action*. Selanjutnya *agent* akan menerima nilai *reward* dari *action* yang dipilih. *Agent* akan berusaha memaksimalkan reward yang diterima dari waktu ke waktu.



Gambar 2.2 Interaksi antara *Agent* dan *Environment* [3]

Gambar 2.2 mengilustrasikan interaksi antara *agent* dan *environment*. Secara spesifik *agent* dan *environment* berinteraksi di setiap waktu t = 0,1,2,3,... dan disetiap waktu t, *agent* mendapat *state* $S_t \in \hat{S}$, dimana $\hat{S}$ adalah set dari *state* yang

mungkin terjadi. Selanjutnya *agent* akan memilih *action* $A_t \in \hat{A}(S_t)$, dimana $\hat{A}(S_t)$ adalah set dari *action* di *state* S_t. Selanjutnya *agent* mendapatkan *reward* R_t lalu mendapat *state* baru S_{t+1}. Selain *agent* dan *environment* terdapat unsur-unsur lain yang terdapat di *reinforcement learning*, yaitu: *policy*, *reward function*, *value function*, dan *model* dari *environment*(opsional).

1) *Policy*

   Policy adalah pemetaan dari *state* yang didapatkan terhadap *action* yang dipilih. *Policy* biasanya direpresentasikan sebagai tabel. Pada intinya *policy* mendefinisikan cara *agent* berprilaku setiap waktunya.

2) *Reward Function*

   Reward function mendefinisikan tujuan dari masalah yang akan diselesaikan. *Reward function* menghasilkan nilai *reward* dari *state* yang didapat dan *action* yang dipilih. Pada intinya *reward function* menentukan suatu kejadian adalah baik atau buruk bagi *agent*.

3) *Value Function*

   Value function mendefinisikan apa yang baik atau buruk untuk *agent* dalam jangka panjang. Hampir semua algoritma di *reinforcement learning* didasarkan pada *value function* dari *state* $V(s)$ biasa disebut *state-value function* atau dari pasangan *state* dan *action* $Q(s, a)$ atau biasa disebut *action-value function*, untuk memperkirakan seberapa baik bagi *agent*. Seberapa baik disini didefinisikan *reward* yang diterima dalam jangka panjang. *Value function* menghasilkan nilai dari *state* berdasarkan *reward* dan nilai dari *value function* pada *state* selanjutnya.

4) *Model*

   Model adalah sesuatu yang meniru prilaku lingkungan. Misalnya, dari *state* yang didapat dan *action* yang dipilih, *model* mungkin memprediksi *state* dan *reward* apa yang akan didapat selanjutnya.

Dalam *reinforcement learning* pemilihan *action* untuk setiap *state* yang didapat saat pembelajaran sangat mempengaruhi keberhasilan pembelajaran. Pemilihan *action* secara acak biasa disebut *exploration*, sedangkan pemilihan

action berdasarkan *value function* terbesar biasa disebut *exploitation*. Salah satu cara untuk pemilihan *action* tersebut adalah ε-*greedy*. Dalam ε-*greedy*, *agent* akan memilih *action* secara *random* dengan probabilitas ε, dan akan memilih *action* dengan *value* terbesar dengan probabilitas (1-ε). Pertama tentukan nilai ε antara 0 sampai 1, lalu bangkitkan nilai *random* antara 0 sampai 1. Jika nilai *random* lebih besar dari ε, maka *agent* akan memilih *action* dengan *value function* terbesar, jika tidak maka *agent* akan memilih *action* secara *random*. Semakin besar nilai ε, maka *agent* akan semakin banyak melakukan *exploration*.

2.4.1. *Temporal-Difference Learning*

Temporal-Difference Learning atau biasa disingkat *TD-learning* adalah salah satu metode dari *reinforcement learning*. *TD-learning* melakukan evaluasi *policy* atau memprediksi untuk mengestimasi nilai dari *value function* agar mendapatkan *policy* yang optimal [3]. Metode *TD-learning* yang paling sederhana dikenal sebagai TD(0), dengan melakukan *update* terhadap *value function* seperti pada (2.1).

$$V(S_t) \leftarrow V(S_t)+\alpha(R_{t+1}+\gamma V(S_{t+1})-V(S_t)) \qquad (2.1)$$

Daftar Notasi

- α = *Learning rate*, $0 < \alpha \leq 1$, menentukan ukuran laju dimana nilai yang lama akan digantikan dengan nilai baru.
- γ = *Discount rate*, $0 \leq \gamma \leq 1$, menentukan nilai dari *reward* di masa depan, semakin kecil nilai γ, maka *agent* akan semakin mementingkan *reward* dekat, bukan *reward* di masa depan.

2.4.2. *Q-Learning*

Q-learning adalah pengembangan dari *Temporal-Difference* yang juga biasa dikenal *off-policy TD control* [3]. *Q-learning* merupakan salah satu terobosan paling penting dalam *reinfrocement learning*. *Q-learning* melakukan update

terhadap *action-value function* seperti pada (2.2).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)) \quad (2.2)$$

Daftar Notasi

α = *Learning rate*, $0 < \alpha \leq 1$, menentukan ukuran laju dimana nilai yang lama akan digantikan dengan nilai baru.

γ = *Discount rate*, $0 \leq \gamma \leq 1$, menentukan nilai dari *reward* di masa depan, semakin kecil nilai γ, maka *agent* akan semakin mementingkan *reward* dekat, bukan *reward* di masa depan.

Tidak seperti *TD-learning* yang memperbarui *value function* berdasarkan *state* selanjutnya, *Q-learning* memperbarui *value function* berdasarkan nilai *action-value function* terbesar di state selanjutnya. Gambar 2.3 menunjukkan algoritma *Q-learning* secara lengkap dalam bentuk prosedural.

Q-learning

Inisialisasi Q(*S*, *A*) dengan nilai sembarang
ulangi (untuk setiap peristiwa):
    Inisialisasi *S* (dari *environment*)
    ulangi (untuk setiap langkah):
        Pilih *action A* pada *state S* (contohnya dengan ε-*greedy*)
        Lakukan *action A*, lalu dapatkan *reward R* dan *next state S'*
        Update action-value function pada *state S* dan *action A*
        $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S, A))$
        Update next state menjadi *current state*
        $S \leftarrow S'$
    sampai *S* adalah akhir dari peristiwa
sampai proses pembelajaran selesai

Gambar 2.3 Algoritma *Q-Learning*

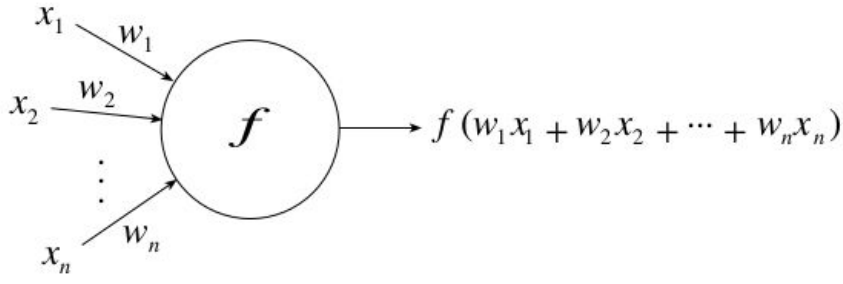
2.5. *Supervised Learning*

Supervised learning adalah metode pembelajaran yang melibatkan prediksi *output* dari *input* baru yang diberikan [9]. *Agent* menerima contoh pelatihan dan membangun representasi yang dapat digunakan untuk prediksi baru. Tujuannya adalah untuk memprediksi nilai-nilai dari fitur target untuk contoh uji dan contoh yang belum diberikan. *Agent* dapat memilih hipotesis contoh pelatihan terbaik, menggambarkan semua hipotesis yang konsisten dengan data, atau menghitung probabilitas dari hipotesis contoh pelatihan.

Metode *supervised learning* mengambil fitur *input* dan fitur target atau biasa dikenal sebagai data pelatihan yang digunakan untuk memprediksi nilai di masa depan. Banyak metode *supervised learning* yang bisa digunakan, salah satu metode yang bisa digunakan adalah *artificial neural networks*.

2.6. *Artificial Neural Networks*

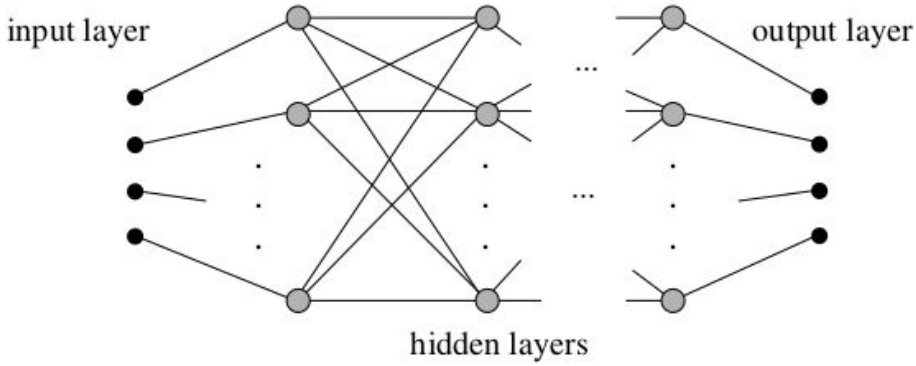
Artificial neural networks atau biasa disingkat ANN adalah model dari kemampuan jaringan saraf biologis untuk memproses informasi [11]. Jaringan saraf biologis adalah sistem yang mengatur dirinya sendiri dan masing-masing *neuron* juga mengorganisir struktur dirinya dalam kemampuan memproses informasi dalam berbagai cara. Perbedaan utama antara jaringan saraf dan sistem komputer konvensional adalah paralelisme besar dan redundansi yang mereka eksploitasi untuk menangani unit komputasi individual yang tidak dapat diandalkan.



Gambar 2.4 Ilustrasi Sebuah *Neuron* pada ANN [11]

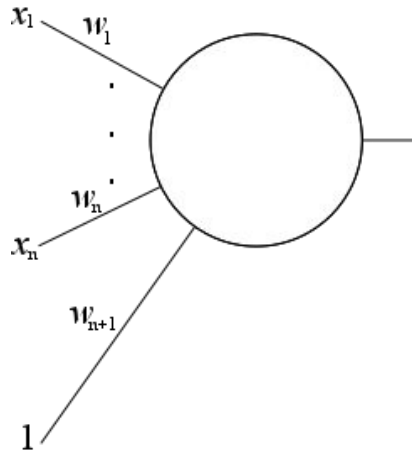
Gambar 2.4 menampilkan ilustrasi dari sebuat *neuron* pada *artificial neural networks* dengan *n* buah *input*. Masing masing *input i* mengirimkan nilai x_i. Biasanya *input* memiliki bobot yang berhubungan, yang berarti bahwa informasi yang masuk x_i dikalikan sesuai dengan bobotnya w_i.

Artificial neural networks terdiri dari beberapa *neuron* yang saling terhubung. Pada umumnya struktur *artificial neural networks* dibagi menjadi tiga, yaitu *input layer*, *hidden layer*, dan *output layer*. Jumlah *neuron* atau unit di setiap *layer* disesuaikan dengan informasi yang akan diproses. Jumlah *input layer* dan *output layer* adalah satu, sedangkan jumlah *hidden layer* bisa satu atau lebih. Gambar 2.5 menampilkan ilustrasi dari arsitektur *artificial neural networks*.



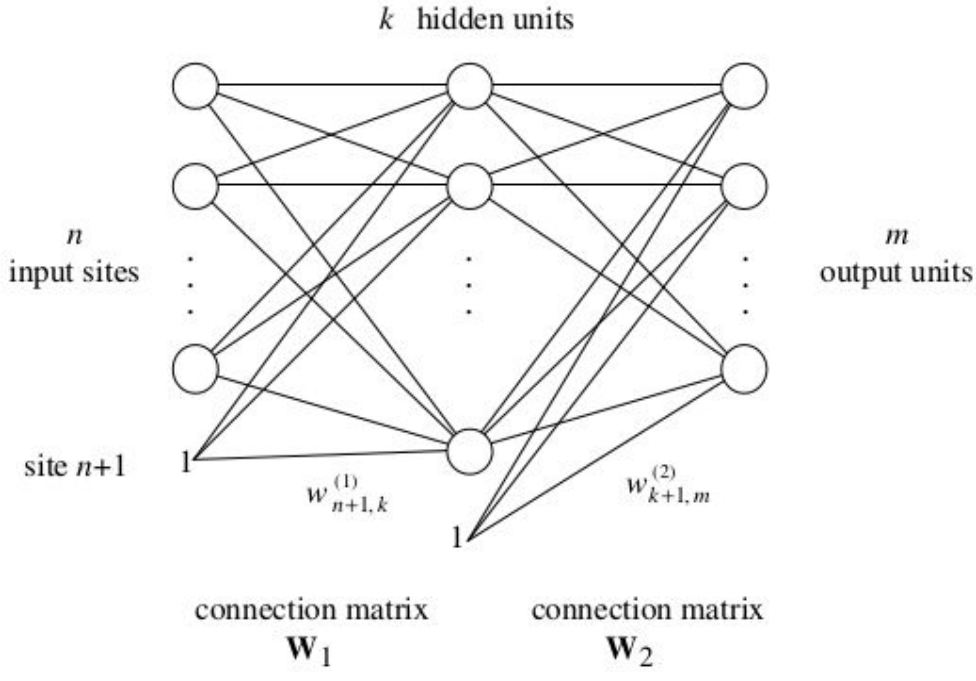
Gambar 2.5 Arsitektur *Artificial Neural Networks* [11]

Setiap unit atau *neuron* di setiap *layer* tidak terhubung satu sama lain. Setiap unit atau *neuron* mentransmisikan nilai ke setiap *neuron* di *layer* setelahnya, kecuali *layer* terakhir(*output layer*). Dalam kebanyakan kasus pembelajaran dapat dinyatakan lebih mudah dengan menambahkan bias. Gambar 2.6 menunjukan ilustrasi *neuron* dengan bias.



Gambar 2.6 *Neuron* Dengan Bias

Vektor input (x_1, x_2,..., x_n) harus ditambah satu dan dihasilkan (n_{+1})-dimensi vektor (x_1, x_2,..., x_n, 1). Bobot yang menghubungkan neuron juga ditambahkan (w_1, ..., w_n, w_{n+1}).



Gambar 2.7 *Artificial Neural Networks* Dengan Bias[11]

Gambar 2.7 menunjukkan arsitektur *artificial neural networks* dengan *n* buah unit di *input layer*, satu *hidden layer* dengan *k* buah *neuron*, dan *m* buah *neuron* di

output layer, dan ditambah bias.

2.6.1. *Backpropagation*

Backpropagation adalah salah satu algoritma *supervised learning* yang digunakan dalam *artificial neural networks*. *Backpropagation* mencari kombinasi bobot untuk meminimalkan kesalahan *output* untuk dianggap menjadi solusi yang benar [11]. Secara sederhana *backpropagation* dilakuakn dalam dua tahap, yaitu:

1) *Feed-forward*

   Pola yang akan dilatih diset ke setiap unit di *input layer*, lalu *output* yang dihasilkan ditransmisikan ke *layer* selanjutnya terus sampai *output layer*.

2) *Backpropagation*

   Berdasarkan *output* yang diharapkan, setiap bobot disesuaikan agar menghasilkan *error* yang minimal mulai dari bobot yang terhubung ke *output neuron*, lalu terus mundur sampai ke *input layer*.

Pada Gambar 2.3 terlihat bahwa nilai *output* dari *neuron* dihasilkan dari sebuah fungsi, fungsi tersebut biasa disebut *activation function*. Salah satu *activation function* yang populer untuk jaringan *backpropagation* adalah *binary sigmoid* dengan rentang nilai (0,1) yang didefinisikan sebagai (2.3). Untuk tahap *backpropagation* menggunakan fungsi turunannya yaitu (2.4).

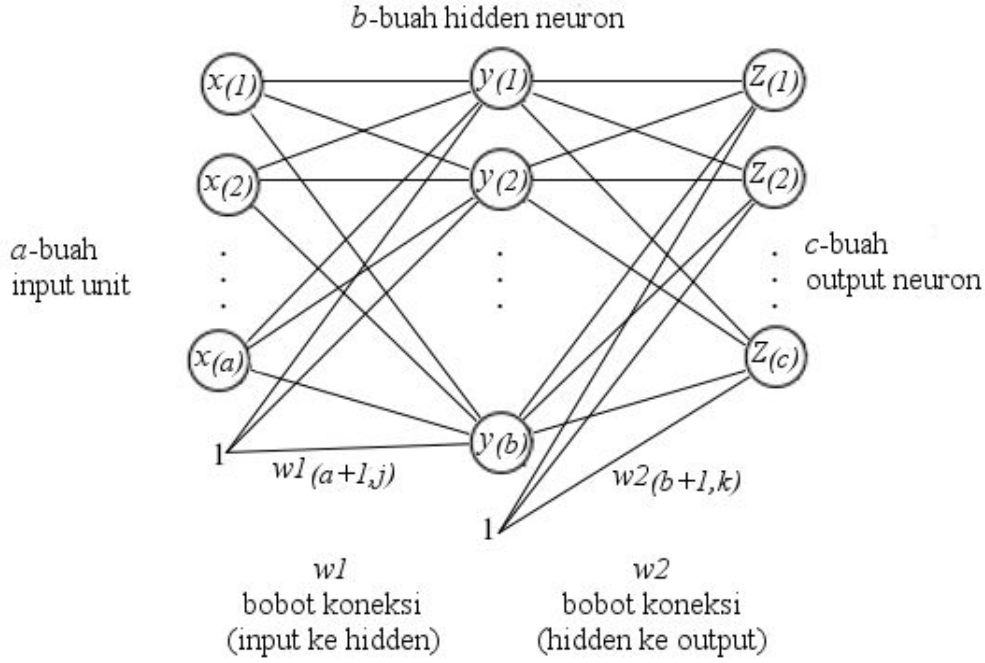
$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

$$f'(x) = f(x)(1 - f(x)) \tag{2.4}$$

Alternatif selain fungsi *binary sigmoid* adalah *bipolar sigmoid* dengan rentang nilai (-1,1) yang didefinisikan sebagai (2.5). Untuk tahap *backpropagation* menggunakan fungsi turunannya yaitu (2.6).

$$f(x) = \frac{2}{1 + e^{-x}} - 1 \tag{2.5}$$

$$f'(x) = \frac{1}{2}(1 + f(x))(1 - f(x)) \tag{2.6}$$



Gambar 2.8 Ilustrasi *Artificial Neural Networks*

Gambar 2.8 menunjukkan ilustrasi *artificial neural network* dengan *a*-buah *input* 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* ke *neuron* di *hidden layer*, bobot $w2_{(j,k)}$ yang menghubungkan *neuron* di *hidden layer* ke *neuron* di *output layer*, dan *t* dengan *c*-buah nilai adalah *output* yang diharapkan. Ilustrasi tersebut digunakan pada Gambar 2.9 Algoritma *Backpropagation*.

Backpropagation

Inisialisasi semua bobot *w1* dan *w2* dengan nilai kecil

ulangi:

ulagi untuk setiap pola:

Set pola ke *input* unit *x*

Feed-forward

Hitung *output* dari setiap *neuron* di *hidden layer*

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

$$y_{(j)} = f(y_net_{(j)})$$

Hitung *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

Hitung perubahan semua bobot *w2*

$$\delta_o_{(k)} = (t_{(k)} - z_{(k)}) f'(z_net_{(k)})$$

$$\Delta w2_{(j,k)} = \alpha \ \delta_o_{(k)} \ y_{(j)}$$

Hitung perubahan semua bobot *w1*

$$\delta_h_net_{(j)} = \sum_{k=1}^{c} w2_{(j,k)} \delta_o_{(k)}$$

$$\delta_h_{(j)} = \delta_h_net_{(j)} f'(y_net_{(j)})$$

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

Update semua bobot *w1* dan *w2*

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

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

sampai kondisi henti terpenuhi

Gambar 2.9 Algoritma *Backpropagation*

Daftar Notasi

α = *Learning rate*, $0 < \alpha \leq 1$, menentukan ukuran laju dimana nilai yang lama akan digantikan dengan nilai baru.

Update bobot bisa dilakukan secara *off-line* ataupun *on-line*. Ketika berbicara tentang *off-line training* maka semua nilai perubahan dijumlahkan terlebih dahulu dan dilakukan update setelah semua pola dilatih. Biasanya *update* bobot dibuat secara sekuensial pada setiap masing masing pola (ini disebut *on-line training*) seperti tahapan pada Tabel 2.2. Dalam kasus ini pola pelatihan dipilih secara acak. Ketika training set menjadi sangat banyak, maka penggunaan *on-line training* menjadi lebih efisien.

2.7. *Value Function Approximation*

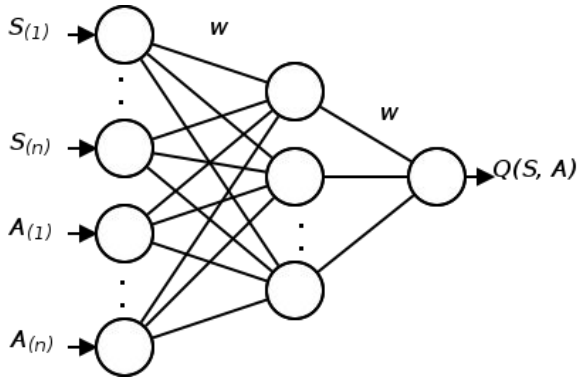
Dalam *reinforcement learning* setiap nilai dari *value funtion* direpresentasikan sebagai sebuah tabel. Masalah yang terjadi dibeberapa kasus ketika pasangan *state* dan *action* menjadi sangat besar adalah kebutuhan *memory* yang besar, dan waktu pembelajaran yang lama. Dari masalah tersebut maka dibutuhkan generalisasi. Bagaimana mendapatkan pengalaman yang baik dari bagian *state* yang terbatas untuk mendapatkan perkiraan nilai yang baik. Jenis dari generalisasi yang dibutuhkan biasa disebut *function approximation*. *Function approximation* merupakan bagian dari *supervised learning*, topik utama yang banyak diteliti salah satunya adalah *artificial neural networks* [3].

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 *artificial neural networks*, maka *w* adalah semua bobot yang menghubungkan masing masing unit atau neuron.

2.7.1. *Backpropagation* sebagai *Value Function Approximation*

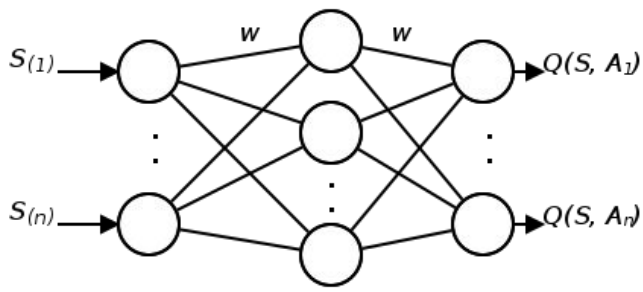
Seperti yang sudah dijelaskan dibagian sebelumnya, *artificial neural*

networks dapat digunakan sebagai *value function approximation*. Nilai dari *value function* didapat dengan proses *feed-forward* dan mengambil nilai pada *neuron* di *output layer* sedangkan proses *update value function* dilakukan dengan *backpropagation* [4] [5] [6] [7]. Gambar 2.10, Gambar 2.11 dan Gambar 2.12 menunjukkan beberapa alternatif arsitektur dari *artificial neural networks* yang digunakan sebagai *value function approximation*.



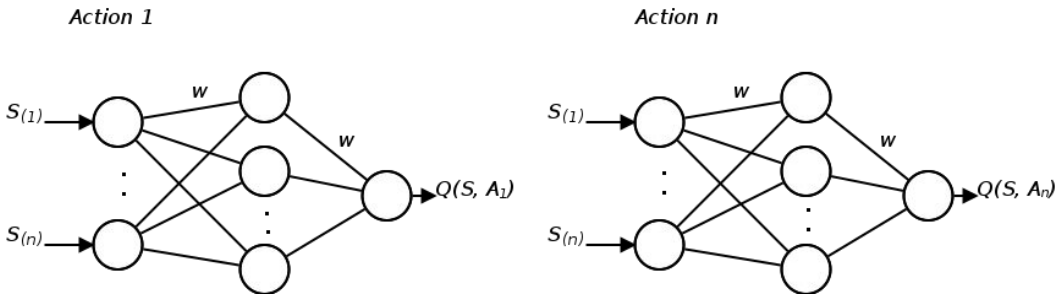
Gambar 2.10 Arsitektur Satu ANN untuk Semua Pasangan *State-Action* (*State* dan *Action* Menjadi *Input*)

Input dari arsitektur pada Gambar 2.10 didapat dari *state* yang didapat dan *action* yang dipilih. Arsitektur ini digunakan pada penelitian Mezaache Hatem dan Abdessemed Foudil [4] dan penelitian Steve Dini dan Mark Serrano [7] namun pada penelitian Steve Dini dan Mark Serrano [7] arsitektur ini memberikan hasil yang tidak begitu baik. Alternatif arsitektur lain ditampilkan pada Gambar 2.11. Berbeda dengan arsitektur pada Gambar 2.10 yang mendefinisikan input dari *state* yang didapat dan *action* yang dipilih.



Gambar 2.11 Arsitektur Satu ANN untuk Semua Pasangan *State-Action* (Hanya *State* yang Menjadi *Input*)

Input pada Gambar 2.11 didapat dari *state* yang didapat, hanya jumlah dari *neuron* pada *output layer* berjumlah *n*, dimana *n* adalah jumlah kemungkinan action. Arsitektur ini digunakan pada penelitian Rudolf Jaksa dan kawan-kawan [5] dan penelitian Bing-Qiang Huang dan kawan-kawan [6]. Alternatif arsitektur lain ditampilkan pada Gambar 2.12.



Gambar 2.12 Arsitektur untuk Satu ANN Tiap Kemungkinan *Action*

Input pada Gambar 2.12 didapat dari *state* yang didapat saja, hanya jumlah dari *artificial neural networks* berjumlah *n*, dimana *n* adalah jumlah kemungkinan action. Arsitektur ini digunakan pada penelitian Rudolf Jaksa dan kawan-kawan [5] dan menyimpulkan arsitektur ini lebih mudah digunakan untuk melakukan kontrol dibandingkan arsitektur pada Gambar 2.11. Arsitektur ini juga direkomendasikan penelitian Steve Dini dan Mark Serrano [7].

2.8. UML (*Unified Modeling Language*)

UML adalah bahasa yang dimaksudkan untuk memodelkan aplikasi komputer [13]. Salah satu tujuan dari UML adalah untuk menyediakan lingkungan *develompment* dengan bahasa desain yang stabil dan umum yang dapat digunakan untuk mengembangkan dan membangun aplikasi komputer. Dengan menggunakan UML, *developer* bisa membaca dan mendistribusikan struktur sistem dan rencana desain aplikasi dengan lebih mudah. UML menyediakan beberapa jenis diagram yang meningkatkan kemudahan memahami aplikasi dalam proses *development*. Beberapa diagram yang digunakan pada

penelitian ini antara lain:

1) *Use-case diagram*

   Use-case diagram menggambarkan unit fungsionalitas yang disediakan oleh sistem. Tujuan utama dari *use-case diagram* adalah untuk membantu memvisualisasikan kebutuhan fungsional dari sistem, termasuk hubungan "aktor" dengan *use-case*, serta hubungan antara *use-case* yang berbeda [12][13].

2) *Activity diagram*

   Activity diagram menunjukkan aliran kontrol antara dua atau lebih abjek saat memproses suatu kegiata. *Activity diagram* dapat digunakan untuk memodelkan tindakan objek pada *higher-level processes*, karena *activity diagram* yang "kurang teknis" jika dibandingkan dengan sequence diagram [12][13].

3) *Class diagram*

   Class diagram menunjukkan bagaimana entitas yang berbeda berhubungan satu sama lain. *Class diagram* menampilkan struktur statis dari sistem. *Class diagram* juga menunjukan implementasi dari *class* yang ada di sistem [12][13].

4) *Sequence diagram*

   Sequence diagram menunjukkan aliran rinci untuk *use-case* tertentu atau bahkan hanya bagian dari *use-case* tertentu. *Sequence diagram* menunjukkan hubungan antara objek yang berbeda dalam urutan, dan dapat menunjukkan secara rinci urutan pemanggilan kepada objek yang berbeda [12][13].

2.9. *Java Programming Language*

Java programming language adalah bahasa pemprograman berorientasi objek dan berbasis *class* untuk keperluan umun dan dirancang cukup sederhana untuk *programmer* [14]. *Java programming language* tergolong bahasa pemrograman tingkat tinggi dimana representasi mesin tidak tersedia melalui

bahasa. *Java programming language* memiliki manajemen penyimpanan otomatis untuk menghindari masalah kemanan dealokasi secara eksplisit. *Java programming language* mendukung pembatasan akses eksternal ke member dari *packages*, *classes*, dan *interfaces*. Hal ini membantu dalam menulis program dengan skala besar.

Java programming language disusun dengan instruksi *bytecode* dan *binary format* yang didefinisikan dalam *Java Virtual Machine* (JVM). Program yang dibuat dengan *Java programming language* berjalan di *Java Runtime EnvironmentI* (JRE), sehingga kita tidak perlu membuat program ulang atau melakukan kompilasi ulang untuk sistem operasi yang berbeda. Sistem operasi yang didukung antara lain *Linux*, *BSD*, *Mac OS X*, *Solaris* dan *Windows*.