BAB II

STUDI LITERATUR

2.1 Kajian Pustaka

Penelitian terdahulu yang terkait dengan prediksi pasang surut air laut dengan beragam metode akan digunakan sebagai acuan dalam penelitian ini. Penelitian yang dilakukan oleh Nikentari, dkk. [9] mengoptimasi nilai *hyperparameter* PSO dan JST seperti, jumlah *neuron input, learning rate, swarm*, c1, c2 *inertia min, inertia max*. Data yang digunakan sebanyak 1000 yang terbagi menjadi 700 data *training* dan 300 data *testing*. Hasil penelitian mendapatkan akurasi prediksi 91.56 % dengan menggunakan 90 *swarm*, *learning rate* 0,9 dan iterasi sebanyak 20 kali. Selanjutnya penelitian yang dilakukan oleh Sonya [10] merancang sistem prediksi tinggi gelombang yang mengintegrasikan metode *thiessen polygon* dan JST. Hasil validasi terhadap tinggi gelombang ramalan metode *thiessen polygon* dan JST masing-masing diperoleh nilai MAPE 12.59% dan 18,86%. Hasil ketepatan peramalan prediksi tinggi gelombang tahun 2017 dari sistem prediktor jaringan saraf tiruan diperoleh MAPE sebesar 15,95%.

Pada penelitian yang dilakukan Setiawan [11], melakukan analisa hasil ketinggian muka air khususnya di daerah Marabahan Kabupaten Barito Kuala Kalimantan Selatan. Metode yang diusulkan SVM dengan PSO yang menggunakan data dari instansi terkait khususnya di daerah Marabahan. Masing-masing algoritma akan implementasikan dengan menggunakan RapidMiner 5.1. Pengukuran kinerja dilakukan dengan menghitung rata-rata *error* yang terjadi melalui besaran *Root Mean Square Error* (RMSE). Semakin kecil nilai dari masing-masing parameter kinerja ini menyatakan semakin dekat nilai prediksi dengan nilai sebenarnya. Dengan demikian dapat diketahui algoritma yang lebih akurat. Hasil RMSE *Support Vector Machines* Berbasis PSO adalah 37.685. Selanjutnya penelitian yang dilakukan oleh Salim, dkk. [12] melakukan prediksi tingkat pasang surut per jam di Mangalore, Karnataka, menggunakan tingkat pasang surut per jam seminggu sebagai *input*. Dalam memprediksi tingkat pasang menggunakan *Feed Forward Back Propagation* (FFBP) dan *Non-linear Auto Regressive with eXogenous input*

(NARX). Jaringan FFBP menghasilkan nilai koefisien korelasi 0,564 dan jaringan NARX menghasilkan koefisien korelasi sangat tinggi 0,915 untuk prediksi tingkat pasang per jam selama setahun. Studi ini membuktikan bahwa teknik JST dapat berhasil digunakan untuk prediksi pasang di Mangalore.

Pada penelitian yang dilakukan oleh Hermanto [13], membuat model prediksi menggunakan konsep penambangan data, *association rule,* klasifikasi, serta *Random Forest*. Penelitian ini menggunakan data dari stasiun pengamatan maritim Cilacap mulai Agustus 2012 sampai dengan Agustus 2016. Data tersebut terdiri atas tanggal, waktu, kecepatan angin, arah angin, arah arus, kecepatan arus, arah gelombang, dan kecepatan gelombang. Data pengujian adalah sebagian data yang diambil secara acak dari keseluruhan data yang digunakan. Dari pengujian model, didapatkan bahwa *Association Rule* menghasilkan akurasi 79%, sedangkan *Classification Tree* menghasilkan akurasi 88%.

Berdasarkan penelitian prediksi pasang surut air laut yang telah dilakukan dengan berbagai macam metode, pada penelitian ini dilakukan eksperimen menggunakan metode Gated Recurrent Unit dalam masalah prediksi ketinggian pasang surut air laut. Tabel 2.1 merupakan rangkuman penelitian terdahulu yang terkait dengan prediksi pasang surut air laut dengan beragam metode.

Tabel 2.1 Penelitian terkait

Peneliti	Tahun	Judul	Hasil
Nikentari, dkk. [9].	2018	Optimasi Jaringan Saraf Tiruan *Backpropagation* dengan *Particle Swarm Optimization* untuk Prediksi Pasang Surut Air Laut	akurasi prediksi adalah 91.56 % dengan menggunakan 90 *swarm, learning rate* 0,9 dan iterasi sebanyak 20 kali.

Peneliti	Tahun	Judul	Hasil
Sonya [10].	2017	Perancangan Sistem Prediktor Ketinggian Gelombang Berbasis Thiessen Polygon Dan Jaringan Saraf Tiruan Di Perairan Dangkal Jawa Timur	Hasil ketepatan peramalan prediksi tinggi gelombang tahun 2017 dari sistem prediktor jaringan saraf tiruan diperoleh MAPE sebesar 15,95%.
Setiawan [11].	2016	Prediksi Tinggi Muka Air Menggunakan Support Vector Machine Berbasis Particle Swarm Optimization	Hasil RMSE Support Vector Machines Berbasis PSO adalah 37.685.
Salim, dkk. [12].	2015	*Weekly prediction of tides using neural networks*	nilai koefisien korelasi 0,564 dan jaringan NARX menghasilkan koefisien korelasi sangat tinggi 0,915 untuk prediksi tingkat pasang per jam selama setahun.
Hermanto [13].	2018	Prakiraan Tinggi Gelombang Air Laut Menggunakan Data Mining	Association Rule menghasilkan akurasi 79%, sedangkan Classification Tree menghasilkan akurasi 88%.

2.2 Teknik Peramalan

Peramalan atau prediksi merupakan suatu cara memperkirakan dengan sistematis untuk mengetahui sesuatu yang akan terjadi di masa depan berdasarkan pengalaman informasi masa lalu dan sekarang. Prediksi tidak harus memberikan jawaban secara pasti kejadian yang akan terjadi, melainkan berusaha untuk mencari jawaban sedekat mungkin yang akan terjadi [14].

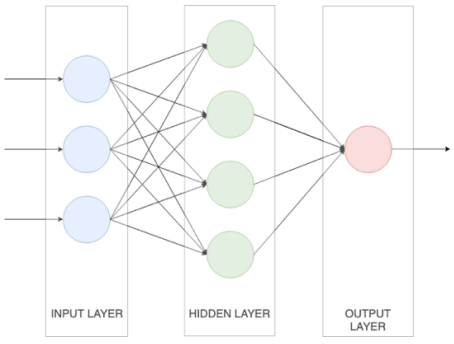
Pada saat melakukan teknik peramalan terdapat dua metode yaitu kualitatif dan kuantitatif. Metode peramalan kualitatif adalah memperkirakan secara subjektif (intuitif), biasanya digunakan ketika data historis tidak tersedia. Metode peramalan kuantitatif dapat dibagi menjadi dua tipe, *causal* dan *time series*. Metode peramalan *causal* meliputi faktor-faktor yang berhubungan dengan variabel yang diprediksi seperti analisis regresi. Peramalan *time series* merupakan metode kuantitatif untuk menganalisis data masa lampau yang telah dikumpulkan secara teratur menggunakan teknik yang tepat. Hasilnya dapat dijadikan acuan untuk peramalan nilai di masa yang akan datang [15].

2.3 Jaringan Saraf Tiruan

Jaringan saraf tiruan atau *neural network* merupakan sebuah model yang mengadopsi dari sistem saraf otak manusia. Model jaringan saraf ditunjukkan dengan kemampuannya dalam emulasi, analisis, prediksi dan asosiasi. Kemampuan yang dimiliki jaringan saraf tiruan dapat digunakan untuk belajar dan menghasilkan aturan atau operasi dari beberapa contoh atau *input* yang dimasukkan dan membuat prediksi tentang kemungkinan *output* yang akan muncul atau menyimpan karakteristik input yang diberikan kepada jaringan saraf tiruan [16].

2.3.1 Arsitektur Jaringan Saraf Tiruan

Arsitektur jaringan saraf tiruan terdapat tiga lapisan seperti Gambar 2.1 yaitu lapisan masukan (*input layer)*, lapisan tersembunyi (*hidden layer*) dan lapisan keluaran (*output layer*).



Gambar 2.1 Lapisan pada jaringan saraf tiruan.

Lapisan masukan merupakan lapisan yang terdiri dari beberapa neuron yang akan menerima sinyal dari luar dan kemudian meneruskan ke neuron-neuron lain

dalam jaringan. Lapisan ini berdasarkan ciri-ciri dan cara kerja sel-sel saraf sensorik pada jaringan saraf biologi. Lapisan tersembunyi merupakan tiruan dari sel-sel saraf konektor pada jaringan saraf biologi. Lapisan tersembunyi berfungsi meningkatkan kemampuan jaringan dalam memecahkan masalah. Konsekuensi dari adanya lapisan ini adalah pelatihan menjadi makin sulit atau lama. Lapisan keluaran berfungsi menyalurkan sinyal-sinyal keluaran hasil pemrosesan jaringan. Lapisan ini juga terdiri dari sejumlah neuron. Lapisan keluaran merupakan tiruan dari sel saraf motor pada jaringan saraf biologis [16].

2.3.2 Bobot

Bobot adalah nilai yang mendefinisikan tingkat atau kepentingan hubungan antara suatu neuron dengan neuron yang lain. Bobot-bobot tersebut bisa ditentukan untuk berada di dalam interval tertentu. Bobot dapat disesuaikan dengan pola-pola selama proses pelatihan. Semakin besar bobot suatu hubungan menandakan semakin pentingnya hubungan kedua *node* tersebut. Pemilihan bobot dan bias akan mempengaruhi apakah jaringan mencapai *error* minimum [10].

2.3.3 Fungsi Aktivasi

Fungsi aktivasi merupakan fungsi yang digunakan pada jaringan saraf untuk mengaktifkan atau tidak mengaktifkan neuron. Karakteristik yang harus dimiliki oleh fungsi aktivasi jaringan perambatan balik antara lain harus kontinu, terdiferensialkan, dan tidak menurun secara monotonis (*monotonically non-decreasing*) [17]. Terdapat beberapa fungsi aktivasi sebagai berikut:

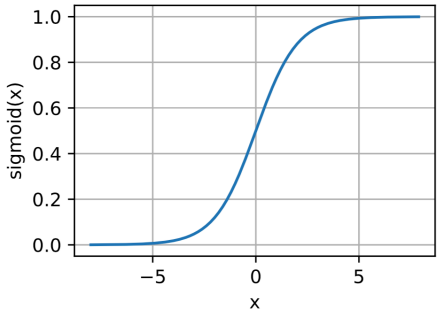
1. Fungsi sigmoid

Fungsi sigmoid mengubah *input*-nya yang mengambil nilai dalam $\mathbb{R}$ ke interval (0, 1). Karena alasan itu, sigmoid ini sering disebut fungsi *squashing*: ia menekan *input* apa pun dalam rentang (-inf, inf) ke beberapa nilai dalam kisaran (0,1) [18]. Persamaan II-1 merupakan persamaan fungsi sigmoid dan Gambar 2.2 merupakan kurva dari fungsi sigmoid.

$$sigmoid(x) = \frac{1}{1+\exp(-x)} \quad \text{(II-1)}$$

Keterangan:

x = Data
exp = Fungsi exponensial



Gambar 2.2 Fungsi aktivasi sigmoid [18].

Berdasarkan kurva fungsi aktivasi sigmoid, ketika *input* mendekati 0, fungsi sigmoid mendekati transformasi linier. Adapun fungsi aktivasi hasil modifikasi sigmoid yaitu Hard sigmoid untuk memprediksi nilai sigmoid supaya lebih cepat. Hard sigmoid merupakan fungsi yang dibagi menjadi *three-segment piecewise linear* [19]. Hard sigmoid didefinisikan dengan Persamaan II-2.

$$\text{Hard sigmoid}(x) = \max(0, \min(1, 0.25 \cdot x + 0.5)) \quad \text{(II-2)}$$

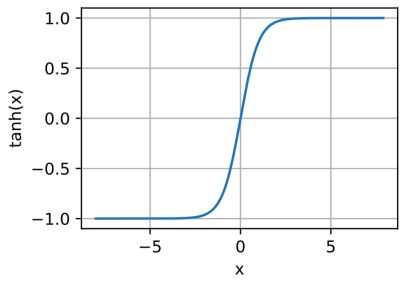
2. Fungsi tanh

Seperti fungsi sigmoid, fungsi tanh (*Hiperbolik Tangen*) juga menekan *input*-nya, mengubahnya menjadi elemen pada interval antara -1 dan 1 [18]. Persamaan II-3 merupakan persamaan fungsi tanh dan Gambar 2.3 merupakan kurva dari fungsi tanh.

$$\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)} \quad \text{(II-3)}$$

Keterangan:

x = Data
exp = Fungsi exponensial



Gambar 2.3 Fungsi aktivasi tanh [18].

Berdasarkan kurva fungsi aktivasi tanh, ketika input mendekati 0, fungsi tanh mendekati transformasi linier. Meskipun bentuk fungsi mirip dengan fungsi sigmoid, fungsi tanh menunjukkan titik simetri tentang asal-usul sistem koordinat.

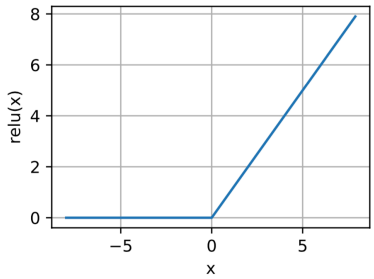
3. Fungsi RELU

Fungsi RELU menyediakan transformasi nonlinear yang sangat sederhana. Dengan elemen z, fungsi didefinisikan sebagai maksimum dari elemen itu dan 0 [18]. Persamaan II-4 merupakan persamaan fungsi RELU dan Gambar 2.4 merupakan kurva dari fungsi RELU.

$$\mathrm{RELU}(z) = \max(z, 0) \tag{II-4}$$

Keterangan:

z = Data
max = Fungsi maksimum



Gambar 2.4 Fungsi aktivasi RELU [18].

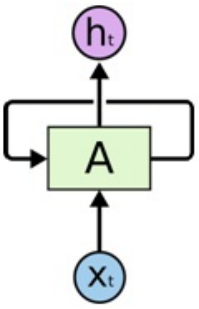
Berdasarkan kurva fungsi aktivasi RELU, ketika mendapatkan *input* negatif, turunan dari fungsi RELU adalah 0 dan ketika mendapatkan *input* positif, turunan

dari fungsi RELU adalah 1. Perhatikan bahwa fungsi RELU tidak dapat dibedakan ketika *input* mengambil nilai sama dengan 0.

2.3.4 Recurrent Neural Network

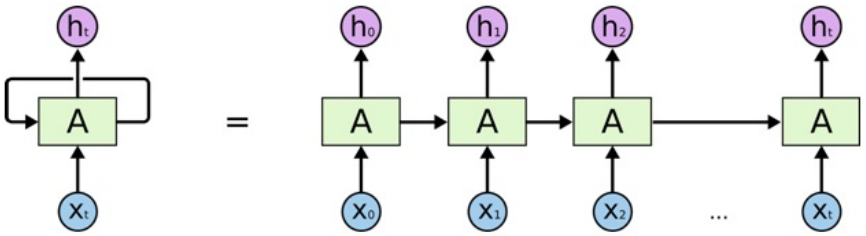
Pada umumnya, manusia tidak membuat keputusan secara tunggal setiap saat. Kita akan selalu memperhitungkan masa lalu dalam membuat sebuah keputusan. Begitu juga dengan RNN, informasi dari masa lalu digunakan dalam proses pembelajarannya. Hal inilah yang membedakan RNN dari Artificial Neural Network biasa [20].

Secara singkat, RNN adalah salah satu bagian dari keluarga *Neural Network* untuk memproses data yang bersambung (*sequential data*). Cara yang dilakukan RNN untuk dapat menyimpan informasi dari masa lalu adalah dengan melakukan *looping* di dalam arsitekturnya, yang secara otomatis membuat informasi dari masa lalu tetap tersimpan [20].



Gambar 2.5 Arsitektur Recurrent Neural Network [20].

Gambar 2.5 adalah visualisasi contoh potongan dari sebuah RNN. Terlihat X_t merupakan *input*, h_t merupakan *output* dan A merupakan *hidden layer*. Dan alur perulangan tersebut memungkinkan informasi untuk dapat dilempar dari satu langkah menuju langkah selanjutnya seperti pada Gambar 2.6 [20].



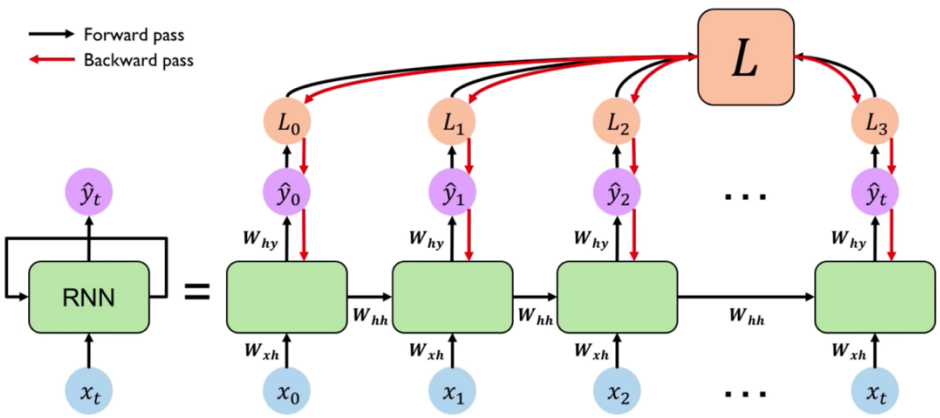
Gambar 2.6 Arsitektur Recurrent Neural Network secara spesifik [20].

Perulangan dari RNN sebenarnya akan memproses *input* dari skala waktu 0 sampai t. RNN akan memproses data *input* satu per satu secara sekuensial, *hidden layer* pun akan melempar data menuju ke *hidden layer* pada skala waktu selanjutnya. Begitu seterusnya secara sekuensial [20].

2.3.5 Backpropagation Through Time

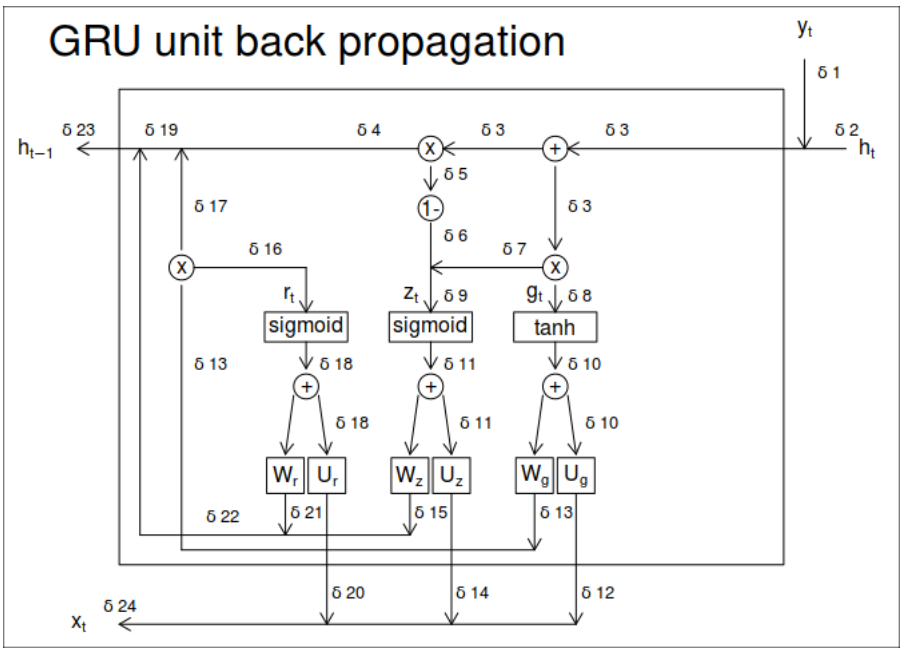
Backpropagation Through Time (BPTT) adalah sebuah algoritma yang digunakan untuk mengubah nilai bobot dalam metode RNN. Algoritma pelatihan BPTT biasanya digunakan untuk data yang memiliki ketergantungan urutan seperti data *time series* [21].

Konsep utama dari BPTT adalah membentangkan jaringan ke dalam waktu dengan meletakkan salinan yang sama dari RNN dan mengatur kembali koneksi jaringan untuk mendapatkan koneksi antara salinan selanjutnya. Untuk menghasilkan peramalan yang akurat, parameter yang ada dalam RNN akan diuji seperti *learning rate*, jumlah *neuron* dan banyaknya data.



Gambar 2.7 Backpropagation through time pada RNN.

Gambar 2.7 merupakan visualisasi BPTT pada RNN. BPTT akan melakukan akumulasi *error* di setiap *timesteps* pada RNN dan memperbarui nilai W yang merupakan nilai bobot.



Gambar 2.8 Backpropagation pada GRU unit.

Pada Gambar 2.8 merupakan simulasi BPTT pada GRU. Untuk melakukan BPTT dengan unit GRU, harus memiliki *eror* yang berasal dari *top layer* (δ1), dan *future hidden state* (δ2). Selain itu, telah menyimpan *error* selama *feed forward* pada setiap langkah *feeding*. Dalam kasus *future hidden state*, *error* ini diatur menjadi nol jika belum dihitung. Untuk konvensi, · sesuai dengan perkalian *point wise*, sedangkan ∗ sesuai dengan perkalian matriks. Berikut adalah langkah-langkah perhitungan backpropagation di unit GRU [22].

Langkah pertama untuk menghitung *error* GRU yang berasal dari *hidden state* adalah menghitung δ3, dengan menjumlahkan *error* dari *top layer* (δ1), dan *future hidden state* (δ2). Selanjutnya hasil δ3 akan digunakan untuk menghitung δ4, δ5, δ7 dan δ8, sedangkan δ6 didapat dari hasil pengurang 1 dikurang δ5 dan nilai δ9 merupakan hasil dari penjumlahan δ7 dan δ8. Berikut adalah persamaan untuk mencari nilai δ3, δ4, δ5, δ6, δ7, δ8 dan δ9.

$\delta 3 = \delta 1 + \delta 2$

$\delta 4 = (1 - z_t) \cdot \delta 3$

$\delta 5 = \delta 3 \cdot h_{t-1}$

$\delta 6 = 1 - \delta 5$

$\delta 7 = \delta 3 \cdot g_t$

$\delta 8 = \delta 3 \cdot z_t$

$\delta 9 = \delta 7 + \delta 8$

Langkah kedua adalah menghitung error pada *candidate hidden state* dan *update gate.* Pada candidate hidden state akan menghitung dan mendapatkan hasil *error* δ10, δ12 dan δ13. Sedangkan *untuk update gate* akan menghitung dan mendapatkan hasil *error* δ11, δ14 dan δ15. Berikut adalah persamaan untuk mencari nilai δ10, δ11, δ12, δ13, δ14 dan δ15.

$\delta 10 = \delta 8 \cdot \tanh'(g_t)$

$\delta 11 = \delta 9 \cdot \text{sigmoid}'(z_t)$

$\delta 12 = \delta 10 * W_g^T$

$\delta 13 = \delta 10 * U_g^T$

$\delta 14 = \delta 11 * W_z^T$

$\delta 15 = \delta 11 * U_z^T$

Langkah ketiga adalah menghitung *error* untuk *reset gate, input gate* dan *hidden state* T-1. Pada *reset gate* akan menghitung dan mendapatkan hasil error δ16, δ18, δ20 dan δ21. Selanjutnya untuk mendapatkan *error input gate* (δ24) dengan menjumlahkan nilai error δ12, δ14 dan δ20. Pada hidden state T-1 (δ23) akan menghitung nilai error yang digunakan yaitu δ4, δ17, δ19 dan δ22. Berikut adalah persamaan untuk mencari nilai error δ16, δ17, δ18, δ19, δ20, δ21, δ22, δ23 dan δ24.

$\delta 16 = \delta 13 \cdot h_{t-1}$

$\delta 17 = \delta 13 \cdot r_t$

$\delta 18 = \delta 17 \cdot \text{sigmoid}'(r_t)$

$\delta 19 = \delta 17 + \delta 4$

$\delta 20 = \delta 18 * W_r^T$

$\delta 21 = \delta 18 * U_r^T$

$\delta 22 = \delta 21 + \delta 15$

$\delta 23 = \delta 19 + \delta 22$

$\delta 24 = \delta 12 + \delta 14 + \delta 20$

Error δ23 dan δ24 digunakan untuk lapisan berikutnya. Setelah semua *error* itu tersedia, dimungkinkan untuk menghitung pembaruan bobot. Untuk menghitung nilai error δW_r, δU_r, δW_z, δU_z, δW_g, dan δU_g dengan persamaan berikut:

$\delta W_r = \delta W_f + h_{t-1}^T * \delta 10$

$\delta U_r = \delta U_f + x_t^T * \delta 10$

$\delta W_z = \delta W_i + h_{t-1}^T * \delta 11$

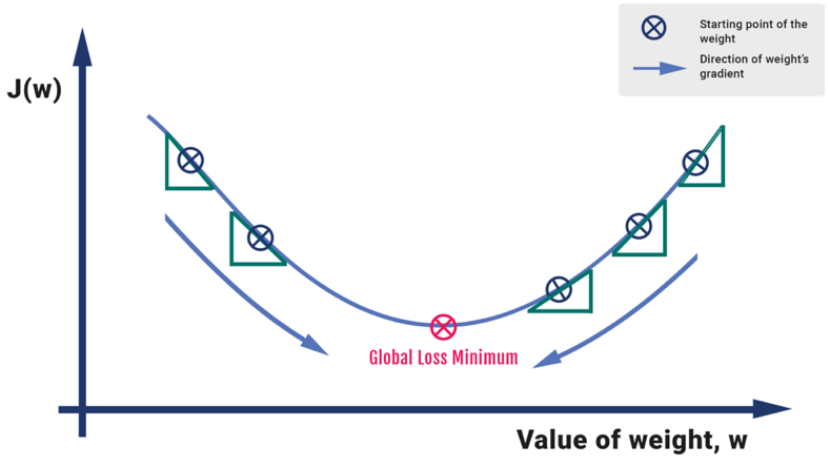
$\delta U_z = \delta U_i + x_t^T * \delta 11$

$\delta W_g = \delta W_g + (h_{t-1}^T \cdot r_t) * \delta 18$

$\delta U_g = \delta U_g + x_t^T * \delta 18$

2.3.6 Optimizer

Optimizer merupakan algoritma optimasi yang digunakan untuk meminimasi sebuah fungsi *error*, yang merupakan sebuah fungsi matematika yang bergantung pada parameter internal yang dapat dipelajari oleh model *learning*. Parameter internal tersebut digunakan dalam menghitung target *value* (y) dari sekumpulan *predictor* (x) yang digunakan dalam model [23]. Gradient Descent adalah proses yang terjadi dalam fase propagasi mundur di mana tujuannya adalah mengubah bobot berdasarkan gradien negatif dan memperbarui secara konsisten sehingga dicapai *error* minimum global J (w) [24].



Gambar 2.9 Ilustrasi proses gradient descent [23].

Pada Gambar 2.9 merupakan proses pembaruan bobot dalam proses balik dari Gradient pada bobot. Kurva berbentuk U merupakan Gradient. Seperti dalam ilustrasi, jika bobot terlalu besar ataupun terlalu kecil maka model akan memiliki nilai *error* yang besar. Jadi, untuk mendapatkan kesalahan minimum maka harus terus dihitung *update* parameter sampai mendapatkan *error* yang minimum. Persamaan II-5 adalah formula dari parameter menggunakan gradient descent [25].

$$\theta = \theta - \eta \,.\nabla_i \,.(\theta) \qquad \text{(II-5)}$$

Keterangan:

θ = nilai bobot *neural network*

η = nilai *learning rate*

∇_i = nilai gradient descent terhadap bobot

Adapun beberapa jenis dari optimizer sebagai berikut:

1. Stochastic Gradient Descent (SGD)

Modifikasi dibuat untuk jumlah data yang digunakan untuk memperbarui parameter internal (θ). Saat menyertakan seluruh data pelatihan, SGD menggunakan *subset* dari data pelatihan ($x^{(i)}$; $y^{(i)}$) secara bertahap. Ini menghitung satu parameter *update*

berdasarkan satu *subset* kecil dari data pelatihan. Ini akan menghitung ulang gradien untuk *subset* lain dan melakukan pembaruan parameter berikutnya. Setiap *subset* dipilih secara acak. Oleh karena itu SGD biasanya berjalan lebih cepat dan bisa melakukan pembelajaran. Parameter *update* pada waktu t+1 dihitung menggunakan Persamaan II-6, η adalah *learning rate* dan *J* adalah fungsi *error*. Pembaruan yang sering terjadi di SGD dapat memberikan kemungkinan untuk menemukan lokal minimum baru dan lebih baik. Namun, dampak dari pembaruan adalah kesulitan dalam memperoleh konvergensi [26].

$$\theta_{t+1} = \theta_t - \eta . \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \quad \text{(II-6)}$$

2. Adaptive Gradient (Adagrad)

Stochastic Gradient Descent menggunakan *learning rate* (η) yang sama untuk semua pembaruan parameter. Adagrad memodifikasi proses ini dengan memungkinkan *learning rate* dapat diubah setiap langkah mengenai pentingnya parameter yang diperbarui. Pembaruan signifikan dilakukan untuk parameter jarang, sedangkan pembaruan kecil dilakukan untuk parameter yang sering. Mekanisme ini akan memungkinkan setiap parameter untuk menemukan nilai optimal secara independen dan mendapatkan peningkatan akurasi secara keseluruhan. Parameter *update* dihitung menggunakan Persamaan II-7. g_t adalah gradien parameter pada fungsi error pada waktu t dan G_t adalah elemen diagonal dari matriks $\sum_{\tau=1}^{t} g_\tau . g_\tau^T$. Setiap elemen diagonal mewakili jumlah squares setiap parameter gradien hingga *time step* [26].

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t \quad \text{(II-7)}$$

3. Adaptive Delta (Adadelta)

Adadelta memodifikasi Adagrad untuk mengurangi penurunan monotonnya tingkat pembelajaran. Di Adagrad, gradien kuadrat dari setiap *epoch* terakumulasi, oleh karena itu nilai berkembang dengan proses pelatihan sehingga *learning rate* pada

setiap dimensi menyusut. Gagasan Adadelta untuk mengatasi masalah tersebut hanya mengumpulkan gradien kuadrat *w* terakhir. Untuk alasan efisiensi, akumulasi tersebut diimplementasikan sebagai rata-rata berjalan yang merusak gradien kuadrat sebagai Persamaan II-8 dengan β adalah peluruhan konstan. Pembaruan parameter pada saat langkah t+1 dihitung menggunakan Persamaan II-9 dan II-10 [26].

$$RMS[g]_t = E[g^2]_t = \beta . E[g^2]_{t+1} + (1 - \beta) . g_t^2 \quad \text{(II-8)}$$

$$\Delta\theta_t = -\frac{RMS|\Delta\theta|_{t+1}}{RMS|g|_t} \cdot g_t \quad \text{(II-9)}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \quad \text{(II-10)}$$

4. Root Mean Square Propagation (RMSProp)

RMSProp dikembangkan untuk menyelesaikan masalah Adagrad, seperti yang dilakukan adadelta. Ini membagi *learning rate* dengan rata-rata gradien kuadrat ($E[g^2]_t$) yang merusak eksponensial. Pembaruan parameter pada saat langkah t+1 dihitung menggunakan Persamaan II-11 dan II-12 [26].

$$E[g^2]_t = 0.9E[g^2]_{t+1} + 0.1g_t^2 \quad \text{(II-11)}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \quad \text{(II-12)}$$

5. Adaptive Momentum (Adam)

Adam menghitung tingkat pembelajaran adaptif untuk setiap parameter. Sama seperti Adadelta dan RMSProp, Adam menyimpan gradien kuadrat dari *epoch* masa lalu yang dikoreksi bias ($\hat{v}_t$). Sebagai tambahan, Adam juga menyimpan gradien rata-rata *epoch* yang terkoreksi bias ($\hat{m}_t$). Kedua nilai (v_t dan m_t) memperkirakan saat pertama merusak dan saat kedua gradien seperti yang dihitung oleh Persamaan II-13 sampai II-17, β adalah konstanta *decay*. Pembaruan parameter pada saat langkah t+1 dihitung menggunakan persamaan [26].

$$m_t = \beta_1 . m_{t-1} + (1 - \beta_1) . g_t \quad \text{(II-13)}$$

$$\widehat{m}_t = m_t / (1 - \beta_1^t) \quad \text{(II-14)}$$

$$v_t = \beta_2 . v_{t-1} + (1 - \beta_2) . g_t \quad \text{(II-15)}$$

$$\hat{v}_t = v_t / (1 - \beta_2^t) \quad \text{(II-16)}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_{tt} + \epsilon}} . \widehat{m}_t \quad \text{(II-17)}$$

6. Adaptive Max Pooling (Adamax)

Adamax adalah pengembangan dari Adam. Modifikasi adalah dalam penggunaan *infinity norm* (u_t). Itu menunjukkan bahwa nilai v_t di Adam dengan ℓ_∞ akan bertemu dengan nilai yang lebih stabil. parameter *update* pada *time step* t + 1 kemudian dihitung menggunakan Persamaan II-18 dan II-19 [26].

$$\theta_{t+1} = \theta_t - \frac{\eta}{u^t} . \widehat{m}_t \quad \text{(II-18)}$$

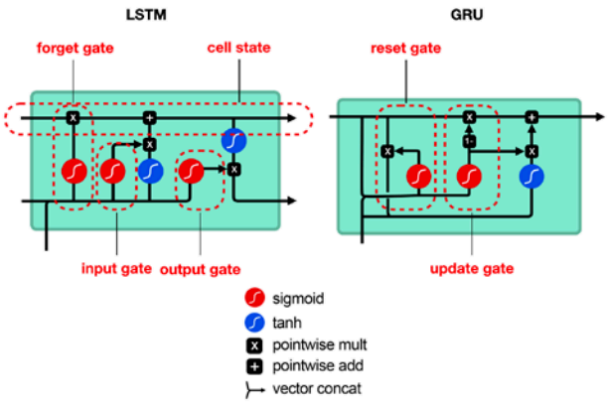
$$u_t = \beta_2^\infty . v_{t-1} + (1 - \beta_2^\infty) . |g_t|^\infty = \max (\beta_2 . v_{t-1}, |g_t|) \quad \text{(II-19)}$$

2.3.7 Gated Recurrent Unit

Gated Recurrent Unit merupakan salah satu arsitektur RNN yang diciptakan pada tahun 2014 oleh Kyunghun Cho [8]. GRU merupakan salah satu solusi dari masalah *vanishing gradient* atau hilangnya gradien dan mengatasi masalah memori jangka panjang yang terjadi pada RNN.

Arsitektur RNN selain GRU adalah Long Term Short Memory. Perbadaan antara kedua arsitektur tersebut adalah pada sistem gerbang, LSTM memiliki *input gate, forget gate,* dan *output gate*, seperti yang ditunjukkan pada Gambar 2.8 Sedangkan GRU tidak menggunakan *cell state*, tetapi memanfaatkan *hidden state* untuk menyimpan informasi. *Reset gate* dalam GRU menentukan informasi baru harus dilupakan atau tidak, sedangkan *update gate* untuk mengingat. Dari Gambar 2.10 dapat dilihat bahwa LSTM memiliki tiga gerbang sigmoid dan dua gerbang tanh, sedangkan GRU hanya memerlukan dua sigmoid dan sebuah tanh. Karena

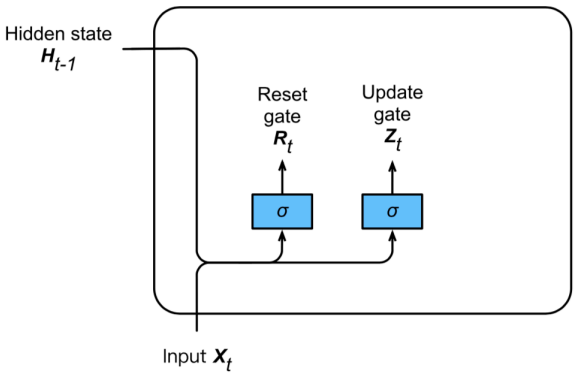
GRU melibatkan perhitungan yang lebih sedikit daripada LSTM, secara teori GRU dapat dilatih lebih cepat daripada LSTM [7].



Gambar 2.10 Perbandingan arsitektur LSTM dan GRU [7].

Di dalam GRU, komponen pengatur alur informasi disebut sebagai *gate*. GRU mempunyai 2 *gate*, yaitu *reset gate* dan *update gate*. *Reset gate* pada GRU akan menentukan bagaimana untuk menggabungkan *input* baru dengan informasi masa lalu, dan *update gate*, akan menentukan berapa banyak informasi masa lalu yang harus tetap disimpan [20]. Berikut adalah tahapan metode GRU:

Pada **tahap pertama**, menentukan informasi *reset gate* dan *update gate*. Gambar 2.11 mengilustrasikan *untuk reset gate* dan *update gate* dalam GRU.



Gambar 2.11 Proses *reset gate* dan *update gate* [18].

Terdapat *input* waktu saat ini X_t dan *hidden stat*e dari waktu sebelumnya H_{t-1} yang terhubung dengan sigmoid sebagai fungsi aktivasi. Persamaan *reset gate* diuraikan pada Persamaan II-20.

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \quad \text{(II-20)}$$

Keterangan:

R_t = *Reset gate*
σ = Fungsi aktivasi sigmoid
X_t = Nilai *input* pada waktu ke t
H_{t-1} = Nilai *hidden state* sebelum waktu ke t
W_{xr} = Nilai bobot untuk *reset gat*e
W_{hr} = Nilai bobot untuk *reset gate*
b_r = Nilai bias untuk *reset gate*

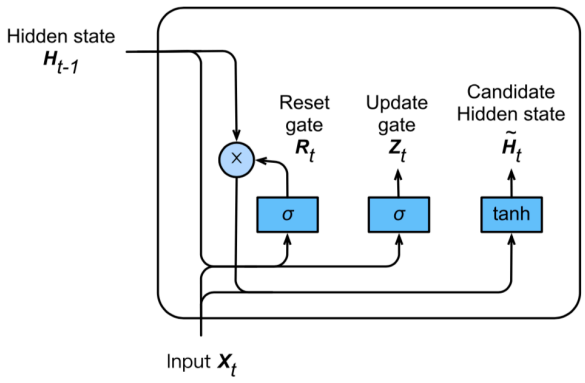
Persamaan *update gate* diuraikan pada Persamaan II-21.

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z) \quad \text{(II-21)}$$

Keterangan:

Z_t = *Update gate*
σ = Fungsi aktivasi sigmoid
X_t = Nilai *input* pada waktu ke t
H_{t-1} = Nilai *hidden state* sebelum waktu ke t
W_{xz} = Nilai bobot untuk *update gate*
W_{hz} = Nilai bobot untuk *update gate*
b_z = Nilai bias untuk *update gate*

Pada **tahap kedua**, akan menentukan informasi *candidate hidden state* untuk waktu saat ini. Hasil *reset gate* akan digunakan pada tahap ini, perkalian *element-wise* (*Hadamard Product*) antara *reset gate* dengan *hidden stat*e dari waktu sebelumnya. Pada tahap ini menggunakan tanh sebagai fungsi aktivasi.



Gambar 2.12 Proses *candidate hidden state* [18].

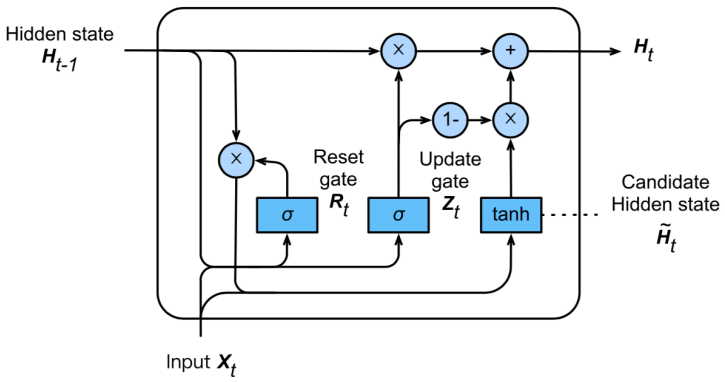
Pada Gambar 2.12 terdapat nilai *reset gate* R_t dan *hidden state* dari waktu sebelumnya H_{t-1} yang terhubung dengan perhitungan *element-wise* $\odot$. Nilai *input* waktu saat ini X_t dan hasil dari operasi tersebut selanjutnya akan digunakan pada *candidate hidden state*. Persamaan *candidate hidden state* diuraikan pada Persamaan II-22.

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h) \qquad \text{(II-22)}$$

Keterangan:

$\tilde{H}_t$ = *Candidate hidden state*
$tanh$ = Fungsi aktivasi tanh
X_t = Nilai *input* pada waktu ke t
H_{t-1} = Nilai *hidden state* sebelum waktu ke t
W_{xh} = Nilai bobot untuk *candidate hidden state*
W_{hh} = Nilai bobot untuk *candidate hidden state*
b_h = Nilai bias untuk *candidate hidden state*

Pada **tahap ketiga**, merupakan tahap terakhir untuk menentukan *hidden state* waktu saat ini. Informasi yang dibutuhkan untuk menentukan *hidden state* adalah informasi *update gate* dan *candidate hidden state*.



Gambar 2.13 Proses *hidden state* [18].

Pada Gambar 2.13 terdapat 4 operasi dalam tahap ini untuk mendapatkan *hidden state* H_t. *Update gate* Z_t digunakan dalam 2 operasi, yang pertama adalah operasi pengurangan dimana 1 dikurang Z_t dan operasi kedua adalah perhitungan *element-wise* antara Z_t dengan $\mathrm{H_{t-1}}$. Operasi ketiga adalah perhitungan *element-wise* antara *candidate hidden state* $\tilde{H}_t$ dengan hasil operasi pertama, selanjutnya operasi keempat adalah menjumlahkan antara hasil operasi kedua dengan hasil operasi ketiga untuk mendapatkan H_t. Persamaan *hidden state* untuk waktu saat ini diuraikan pada persamaan II-23.

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t \qquad \text{(II-23)}$$

Keterangan:

H_t = *Hidden state* untuk waktu saat ini
$\tilde{H}_t$ = *Candidate hidden state*
$\mathrm{H_{t-1}}$ = Nilai *hidden state* sebelum waktu ke t
Z_t = *Update gate*

2.4 Interpolasi

Interpolasi adalah proses menemukan nilai-nilai tertentu yang didasarkan pada nilai-nilai yang ada pada periode sebelum dan sesudahnya. Persamaan II-24 merupakan persamaan untuk mencari interpolasi [27].

$$X = \frac{(A+(B-A))}{C} \quad \text{(II-24)}$$

Keterangan:

X	=	Data yang diprediksi
A	=	Data pada *t* periode sebelumnya (diketahui)
B	=	Data pada *t* periode terakhir (diketahui)
C	=	Interval data A-B ke *t*

2.5 Min-max Scaler

Min-Max yang menyediakan transformasi linier pada rentang data asli. Skala umum yang digunakan pada penelitian ini yaitu nilai dari rentang 0 sampai 1. Persamaan II-25 merupakan persamaan *Min-Max.*

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad \text{(II-25)}$$

Keterangan:

x'	=	Data hasil normalisasi
x	=	Data
x_{max}	=	Nilai maksimum
x_{min}	=	Nilai minimum

2.6 Nilai Ketepatan Prediksi

Nilai ketepatan prediksi dilakukan untuk mengukur kesesuaian antara data yang sudah ada dengan data peramalan [28]. Untuk menentukan besar kesalahan yang dihasilkan prediksi dapat dilakukan dengan beberapa cara yang popular yaitu *Mean Absolute Percentage Error* (MAPE), *Mean Square Error* (MSE) dan *Root Mean Square Error* (RMSE).

1. *Mean Absolute Percentage Error* (MAPE)

Mean Absolute Percentage Error (MAPE) merupakan nilai yang dapat digunakan untuk mengukur ketepatan nilai dugaan model yang dinyatakan dalam bentuk rata-

rata persentase absolut kesalahan [29]. Persamaan MAPE dapat dilihat pada persamaan II-27.

$$PE = \frac{x_t - F_t}{x_t} \quad \text{(II-26)}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}|PE| \times 100 \quad \text{(II-27)}$$

keterangan:

PE	=	*Percentage error*
n	=	Jumlah data
x_t	=	Data target pada waktu ke t
F_t	=	Data prediksi pada waktu ke t

2. *Mean Square Error* (MSE)

Mean Square Error (MSE) secara historis telah menjadi ukuran utama yang digunakan untuk membandingkan kinerja metode peramalan, sebagian besar karena kemudahan komputasi dan relevansi teoretisnya dengan statistik [30]. Persamaan MSE dapat dilihat pada persamaan II-28.

$$MSE = \frac{1}{n}\sum_{t=1}^{n}((Y_t - \acute{Y}_t)^2 \quad \text{(II-28)}$$

keterangan:

n	=	Jumlah data
Y_t	=	Data target pada waktu ke t
$\acute{Y}_t$	=	Data prediksi pada waktu ke t

3. *Root Mean Square Error* (RMSE)

Root Mean Square Error (RMSE) adalah ukuran yang baik dari akurasi prediksi dan sering digunakan untuk mengukur perbedaan antara nilai yang diprediksi oleh

model atau estimator dan nilai yang sebenarnya diamati dari hal yang dimodelkan atau diperkirakan [30]. Persamaan RMSE dapat dilihat pada persamaan II-29.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}((Y_t - \acute{Y}_t)^2} \quad \text{(II-29)}$$

Keterangan:

n = Jumlah data

Y_t = Data target pada waktu ke t

$\acute{Y}_t$ = Data prediksi pada waktu ke t