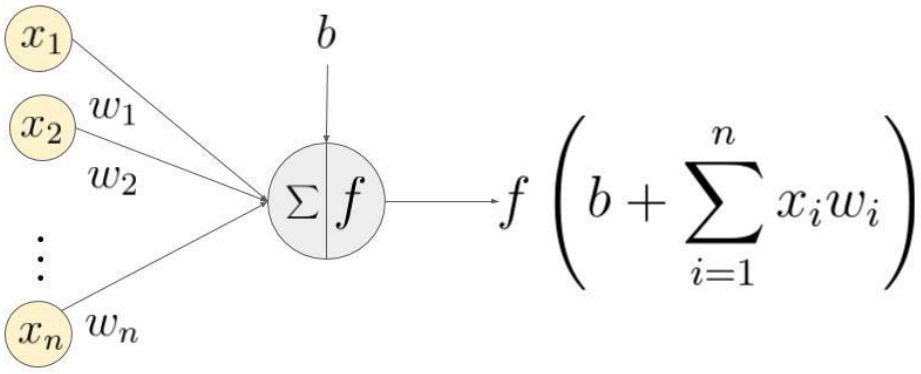
Richard Rivaldo
13519185 – K-04

Artificial Neural Network

1. Jelaskan bagaimana forward propagation pada ANN!

   Forward Propagation adalah proses mempropagasikan input data yang ada dari *input layer* sampai menuju ke *output layer*. Proses ini akan memerlukan inisialisasi *weight* atau bobot dan *bias* untuk setiap neuron yang ada pada ANN. *Weight* mengindikasikan seberapa penting suatu fitur tertentu terhadap penentuan *output*, sedangkan *bias* merupakan cara untuk membantu model dalam melakukan *fitting* terhadap dataset, mirip seperti *intercept* pada regresi biasa.
   Untuk mendapatkan suatu *output*, maka data input harus diberikan hanya pada arah depan (menuju ke *output layer*). Propagasi yang demikian dimaksudkan untuk menghindari terjadinya siklus pada graf neuron yang ada. Konfigurasi model yang dimaksud disebut dengan *feed-forward network*.
   Secara umum, pada proses ini, model hanya melakukan dua buah tahap pemrosesan input saja. Tahap pertama merupakan tahap *pre-activation*. Pada tahap ini, model akan melakukan pemrosesan input yang didapat dari neuron pada *layer* sebelumnya. Model akan melakukan transformasi linear untuk mengkalkulasi *weighted sum* atau agregasi bobot terhadap masukan yang diberikan. Ilustrasi kalkulasi *weighted input sum* ini untuk sebuah *layer* dapat terlihat pada gambar berikut.



An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

   Agregasi bobot yang telah didapat dari hasil tersebut kemudian akan ditentukan untuk diteruskan atau tidak informasinya ke neuron pada *layer* selanjutnya. Penentuan ini dilakukan pada tahap kedua, yaitu *activation*. Pada tahap ini, nilai yang didapat pada tahap sebelumnya akan dilewatkan ke dalam *activation function*. *Activation Function* merupakan fungsi matematis yang digunakan untuk menambah non-linearitas kepada *network* yang dibangun.
   Kedua tahap ini dilakukan untuk semua lapisan neuron hingga mencapai *output layer*. Hasil yang didapat pada *output layer* dapat dikatakan sebagai sebuah prediksi oleh model pada saat itu dan dengan demikian dilabeli sebagai $\hat{y}$. $\hat{y}$ ini bisa digunakan untuk dibandingkan dengan nilai yang sebenarnya pada data. Perbedaan keduanya dapat dihitung dengan menggunakan *cost* atau *loss function* sebagai bentuk evaluasi model.

Richard Rivaldo
13519185 – K-04

2. Jelaskan bagaimana back propagation pada ANN!

Back Propagation adalah proses dalam *Neural Network* yang berlawanan dengan *Forward Propagation*. *Back Propagation* merupakan propagasi informasi mengenai *error* dari *layer terakhir* (*output layer*) sampai *layer* pertama (*input layer*). Informasi tersebut diperoleh dari kalkulasi *error function* terhadap $\hat{y}$ yang telah didapat tadi. Tujuan dari propagasi ini ialah untuk memperbaiki *weight* dan *bias* yang telah diinisialisasi sebelumnya.
Teknik yang biasanya digunakan dalam *back propagation* adalah *Gradient Descent*. *Back propagation* akan mencari gradien dari *error* hasil *Forward Propagation* terhadap *weight* dan *bias* yang dimilikinya. Gradien ini nantinya akan digunakan untuk memperbaiki kedua hiperparameter bersamaan dengan parameter lain yang disebut dengan *learning rate*. Tujuan dari *gradient descent* sendiri adalah untuk mencari titik global minimum dari *error function* yang dimiliki. Dengan demikian, model yang dibangun memiliki kemampuan untuk melakukan *fitting* dengan baik terhadap data. Kalkulasi yang bisa dilakukan untuk mendapatkan gradien ini secara sederhana adalah sebagai berikut.

$$dl\_wrt\_Yhat = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$dSigmoid = sigmoid(x) \times (1.0 - sigmoid(x))$$

$$dl\_wrt\_Z2 = dl\_wrt\_Yhat \times dSigmoid$$

$$dl\_wrt\_A1 = W2 * dl\_wrt\_Z2$$

$$dl\_wrt\_W2 = A1 * dl\_wrt\_A1$$

$$dl\_wrt\_b2 = dl\_wrt\_Z2$$

$$dl\_wrt\_Relu = dRelu$$

$$dl\_wrt\_Z1 = dl\_wrt\_A1 \times dRelu$$

$$dl\_wrt\_W1 = X \times dl\_wrt\_Z1$$

$$dl\_wrt\_b1 = dl\_wrt\_Z1$$

atau secara lebih umum:

$$dZ^{[L]} = A^{[L]} - Y$$
$$dW^{[L]} = \frac{1}{m} dZ^{[L]} A^{[L]^T}$$
$$db^{[L]} = \frac{1}{m} np.\text{sum}(\text{d}Z^{[L]}, axis = 1, keepdims = True)$$
$$dZ^{[L-1]} = W^{[L]^T} dZ^{[L]} g'^{[L]}(Z^{[L-1]})$$
$$\vdots$$
$$dZ^{[1]} = W^{[L]^T} dZ^{[2]} g'^{[1]}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[1]^T}$$
$$db^{[1]} = \frac{1}{m} np.\text{sum}(\text{d}Z^{[1]}, axis = 1, keepdims = True)$$

Perubahan *weight* dan *bias* dapat dilakukan dengan mengurangi *weights* dan *bias* pada saat itu dengan hasil perkalian *learning rate* terhadap gradien hiperparameter yang berkaitan. Kedua (atau bahkan ketiga jika memperhitungkan perubahan *weight* dan *bias*) dilakukan secara iteratif sebanyak jumlah *epochs* yang ditentukan pada pemakaian algoritma.

$$W_1 = W_1 - lr \times \frac{\partial L}{\partial W_1}$$

$$W_2 = W_2 - lr \times \frac{\partial L}{\partial W_2}$$

Gradient Descent merupakan salah satu algoritma optimisasi yang bisa digunakan. Dalam implementasi tugas ini, digunakan teknik lain yang disebut dengan *Stochastic Gradient Descent*. Cara kerja teknik ini mirip dengan algoritma *Gradient Descent* yang juga biasa disebut dengan *Batch Gradient Descent*. Bedanya, pada *Batch Gradient Descent* gradien dihitung untuk semua data yang ada, sedangkan pada SGD gradien dihitung dengan menggunakan metode stokastik dan memilih data dari dataset dalam jumlah tertentu secara acak.

3. Apa itu optimizer, loss function, dan activation function?

Optimizer adalah algoritma atau fungsi yang digunakan untuk meminimalkan *error function. Optimizer* bekerja dengan memperbarui hiperparameter model, yaitu *weights* dan *biases*, sebagai respon terhadap *error function* yang didapat pada *forward propagation*. Secara umum, optimisasi dilakukan dengan mencari gradien dari *error function* terhadap hiperparameter tersebut. Teknik yang paling umum dan sederhana untuk digunakan adalah *Gradient Descent* yang juga menjadi cikal bakal teknik lainnya.

Loss Function merupakan salah satu jenis *error function* yang sering digunakan secara *interchangeably* dengan *Cost Function*. Secara sederhana, *loss function* merupakan sebuah cara untuk mengevaluasi apakah kondisi model pada saat tertentu sudah cukup baik dalam mempelajari data masukan yang diberikan. Dengan demikian, maka model diinginkan mempunyai *loss function* yang nilainya semakin rendah sehingga mengindikasikan bahwa model berhasil melakukan *fitting* terhadap data yang ada. *Loss function* berbeda dengan *Cost Function* yang bersifat lebih umum dan biasanya digunakan untuk menyatakan akumulasi *loss function* pada suatu *batch* data tertentu.
Activation Function merupakan fungsi matematis yang mendefinisikan bagaimana *weighted sum* yang didapat dari kalkulasi neuron ditransformasi menjadi keluaran untuk neuron pada *layer* tersebut. *Activation Function* terkadang disebut dengan *transfer function*, atau bahkan *squashing function* jika *output range* nya terbatas. Kebanyakan *activation function* bersifat non-linear dan dengan demikian, keberadaannya pada *Neural Network* memberikan non-linearitas pada kalkulasi dan desain model tersebut.

4. Sebutkan beberapa optimizer, loss function, dan activation function lainnya selain yang tercantum pada spesifikasi!

   Beberapa *optimizer* lain yang ada pada saat ini adalah *Gradient Descent* dan *Mini-Batch Gradient Descent* yang menggunakan *batches of data* dalam perhitungan gradien dan merupakan kombinasi SGD dan *Batch Gradient Descent* biasa. Selain itu, SGD juga memiliki versi yang menggunakan momentum untuk mensimulasikan adanya inersia pada pergerakan gradiennya.
   Beberapa *optimizer* lain yang sering digunakan pada saat ini adalah *Nesterove Accelerated Gradient* (NAG), *Adaptive Gradient Descent* (AdaGrad), *Root Mean Square Propagation* (RMS-Prop) dan AdaDelta yang merupakan perkembangan dari AdaGrad, serta Adam (*Adaptive Moment Estimation*) yang mengombinasikan RMS-Prop dengan *Momentum Based Gradient* dan dapat dikatakan merupakan *optimizer* paling popular untuk *Deep Learning* pada saat ini.
   Beberapa contoh dari *loss function* adalah *Mean Absolute Error* (MAE atau L1 *Loss*), *Mean Squared Error* (MSE atau L2 *Loss*, sering juga disebut dengan *Quadratic Loss*), *Root Mean Square Error* (RMSE), *Mean Bias Error*, dan *Mean Squared Logarithmic Error* yang digunakan untuk permasalahan regresi. Adapun permasalahan klasifikasi biner biasanya menggunakan *Binary Cross-Entropy Loss*, *Hinge Loss* (atau *Multiclass* SVM *Loss*), atau *Squared Hinge Loss*, sedangkan permasalahan klasifikasi multikelas dapat menggunakan *Cross-Entropy Loss* (*Negative Log Likelihood*), *Sparse-Multiclass Entropy Loss*, dan *Kullback Leibler Divergence Loss*.
   Selain *Rectified Linear Unit* (ReLU) dan *Sigmoid* atau *Logistic Function*, beberapa fungsi lain yang sering digunakan adalah *Hyperbolic Tangent Activation Function* (Tanh) yang memperluas jangkauan *output* dari -1 sampai 1, *Leaky ReLU* yang merupakan perkembangan dari ReLU untuk mengatasi *diminishing* ReLU untuk nilai masukan negatif, dan juga *Softmax Activation Function* yang sering digunakan utnuk permasalahan klasifikasi multikelas.

5. Pada deep learning, terdapat beberapa arsitektur seperti CNN atau RNN. Coba tuliskan hal yang kamu ketahui tentang beberapa arsitektur lain selain ANN!

   Layaknya *Machine Learning* tradisional, algoritma *Deep Learning* juga dapat dibedakan menjadi dua, yaitu *Supervised Deep Learning* dan *Unsupervised Deep Learning*. *Supervised Deep Learning* merupakan algoritma yang memerlukan label pada *training*

dataset nya, dan sebaliknya untuk *Unsupervised Deep Learning*. Adapun *Perceptron* dapat dikatakan sebagai arsitektur pertama dari *Neural Network* yang berkembang hingga saat ini.

Contoh arsitektur yang bersifat *Supervised* adalah *Convolutional Neural Network* atau CNN. CNN merupakan *multilayer neural network* yang mirip seperti ANN, tetapi lebih terinspirasi dengan korteks dari penglihatan binatang. Dengan demikian, arsitektur ini secara luas digunakan utamanya untuk pengolahan dan pemrosesan citra. Data masukan yang biasanya diberikan kepada CNN berupa gambar yang nantinya akan diproses lebih lanjut, biasanya dengan memecah gambar tersebut ke dalam *pixel* yang dimasukkan ke dalam matriks atau vektor dengan setiap elemennya merupakan nilai warna RGBA atau hitam putih dari *pixel* tersebut pada gambar yang diolah.

Setiap *layer* pada arsitektur CNN memiliki tugasnya masing-masing. Ketika gambar dimasukan ke dalam model melalui *input layer*, *Convolution layer* kemudian menerima masukan tersebut dan mengekstrak fitur penting yang ada pada gambar. Hasilnya kemudian diteruskan pada *Pooling layer* yang ditujukan untuk melakukan *dimensionality reduction* melalui *down-sampling* untuk mengurangi beban pemrosesan pada tahap selanjutnya. Kedua proses ini merupakan proses utama yang ada pada CNN, dan bisa dilakukan berulang-ulang sampai menuju *output layer* yang bisa digunakan untuk mengklasifikasikan gambar. Adapun model tetap akan menggunakan *backward propagation* dalam fase *training* nya.

Selain CNN, ada juga yang dikenal dengan RNN atau *Recurrent Neural Network*. RNN merupakan arsitektur yang utamanya berurusan dengan *Sequence Modelling*. Tidak seperti ANN dan CNN yang menerapkan *fully feed-forward neural network*, RNN dalam hal ini bisa mempropagasikan kalkulasi input ke neuron pada *layer* sebelumnya atau bahkan ke neuron pada *layer* yang sama. Dengan arsitektur yang demikian, maka hal ini memungkinkan RNN untuk menjaga memori dari input yang diterima sebelumnya serta tetap berhasil memodelkan permasalahan yang ingin dipecahkan. Kebanyakan RNN diutilisasi untuk data dengan tipe sekuens-rekurens, seperti suara dan teks, sehingga RNN sering digunakan untuk *Speech Recognition* dan pada bidang *Natural Language* Processing. Dalam fase *training*, model RNN bisa dilatih dengan menggunakan *back propagation* biasa atau dengan varian *back propagation in time* (BPPT).

RNN juga merupakan salah satu arsitektur fundamental yang memunculkan banyak model lainnya. Salah satu model tersebut adalah LSTM atau *Long Short-Term Memory*. LSTM secara konsep berbeda dengan neuron biasa, di mana pada LSTM diperkenalkan konsep sel memori baru. Sel memori ini bisa mempertahankan suatu nilai yang dianggapnya penting baik dalam jangka waktu pendek ataupun lama, sehingga sel memori bisa mengingat nilai-nilai yang penting dan tidak hanya nilai pada saat tertentu.

Kontrol alur informasi pada LSTM dilakukan melalui 3 buah gerbang, yaitu gerbang input, output, dan *forget gate*. *Input gate* mengatur masuknya informasi ke dalam memori, sedangkan *forget gate* mengatur sel ketika suatu informasi dilupakan, sehingga sel bisa mengingat informasi baru.

Output gate mengatur penggunaan informasi yang ada pada sel untuk memberikan keluaran tertentu. Biasanya, model ini dilatih dengan menggunakan BPPT. LSTM sendiri sering digunakan untuk *image captioning* ataupun *poem generator* pada contoh spesifikasi bonus. LSTM juga memiliki simplifikasi arsitektur yang diberi nama GRU atau *Gated Recurrent Unit* yang hanya terdiri dari dua gerbang, yaitu *Reset Gate* dan *Update Gate*. *Update gate* mengatur proses penyimpanan informasi lama, sedangkan *reset gate* mengatur bagaimana menggabungkan informasi yang baru diterima dengan informasi lama yang ada pada sel memori.

Adapun contoh dari arsitektur yang bersifat *unsupervised* salah satunya adalah *Self-Organizing Maps* atau SOM yang juga dikenal dengan *Kohonen Map*. SOM merupakan arsitektur klasterisasi yang menciptakan kelompok dari data masukan dengan *dimensionality reduction*. Berbeda dengan *neural network* biasa, *weight* atau bobot pada SOM berperan sebagai karakteristik dari sebuah *node*. Awalnya, data masukan dipilih secara acak dan kemudian *weight* diinisialisasi dengan nilai yang dekat dengan 0 berdasarkan data masukan tersebut.

Kombinasi dari kumpulan *random weight* tadi kemudian merepresentasikan *node* masukan untuk model tersebut. Antara *node* tersebut dengan *output node* kemudian dicari jaraknya dengan metriks tertentu, misalnya *Euclidean Distance*. *Node* dengan jarak terkecil terhadap *output node* akan menjadi pusat klaster yang disebut dengan *Best Matching Unit* (BMU). Setiap data kemudian dikelompokkan berdasarkan kemiripannya dengan data yang menjadi BMU.

Radius dari BMU ini akan diperbarui seiring berjalannya proses *training*. Pada SOM juga tidak terdapat *activation function*, karena tidak terdapat label sehingga tidak ada kalkulasi *error* dan *back propagation*. SOM sering digunakan untuk permasalahan klasterisasi biasa, *dimensionality reduction*, atau pelacakan *fraud*.

Selain itu, ANN juga mempunyai varian arsitektur yang dikenal dengan nama *Autoencoders*. Model ini memiliki tiga buah *layer*, yaitu *input layer*, *hidden layer*, dan *output layer*. *Input layer* pada model ini di-*encode* menjadi *hidden layer* berdasarkan fungsi *encoding* tertentu. Biasanya jumlah *node* yang ada pada *hidden layer* jauh lebih sedikit dibandingkan jumlah *node* pada *input layer*. *Hidden layer* ini mengandung representasi yang telah terkompres dari masukan data awal. *Output layer* kemudian bertugas untuk mengkonstruksi kembali *input layer* dengan menggunakan fungsi *decoder*.

Pada saat fase *training*, perbedaan dari *input* dan *output layer* dihitung dengan menggunakan *error function*. Bobot dari setiap *node* kemudian diperbaiki untuk meminimalisir nilai kesalahan tersebut. Tidak seperti kebanyakan *unsupervised learning* yang menggunakan data tidak berlabel, *autoencoders* mempelajari data masukan secara kontinu dengan *back propagation*. Dengan demikian, *autoencoders* sering disebut sebagai *Self-Supervising Algorithm* dan digunakan untuk interpolasi dan kompresi data.

Ada juga *Boltzmann Machine Neural Network* yang merupakan salah satu RNN yang bersifat stokastik. Model ini merupakan salah satu model yang bisa mempelajari representasi internal suatu data serta juga mampu menyelesaikan permasalahan kombinatorial yang sulit untuk dipecahkan. Tujuan utama dari *Boltzmann Machine* adalah untuk memaksimalkan hasil produk probabilitas yang ditetapkan oleh model untuk setiap vektor biner pada data masukan.

Pada *Boltzmann Machine* struktur setiap *node* nya bisa saja tidak beraturan dan tidak tersusun seperti representasi *layer* pada *neural network* biasa. Dalam hal ini, *Boltzmann Machine* biasanya memiliki *visible layer* dan *hidden layer* saja, dan semua *node* yang ada pada kedua *layer* akan bersifat *densely connected*. Karena itu, *Boltzmann Machine* memiliki kompleksitas waktu yang sangat tinggi.

Untuk mengatasi hal tersebut, terdapat beberapa pengembangan *Boltzmann Machine*, seperti *Deep Boltzmann Machine* yang dibuat dengan mengurangi jumlah koneksi antar-*node* dan memungkinkan pemrosesan *mini-batches* secara paralel. Selain itu, juga ada *Restricted Boltzmann Machine* yang melarang adanya koneksi antara *node* yang terdapat pada *layer* yang sama. Model ini merupakan model yang bersifat generatif karena menghasilkan keluaran yang berbeda dengan masukan awal. *Boltzmann Machine* secara luas digunakan utamanya untuk sistem rekomendasi film dan lagu seperti yang terdapat pada Netflix dan Spotify.

Selain model-model di atas, juga terdapat beberapa perkembangan dari model-model tersebut seperti *Deep Belief Networks*. DBN merupakan algoritma yang memperkenalkan *novel training*, dan setiap pasang *layer* yang terhubung pada arsitekturnya merepresentasikan kumpulan *Restricted Boltzmann Machine*. DBN sering digunakan untuk *image recognition*, *natural language understanding*, *information retrieval*, dan *failure prediction*. Ada juga *Deep Stacking Networks* yang merupakan kumpulan dari banyak arsitektur lengkap *neural networks* biasa (*input*, *hidden*, dan *output layer*). DSN cenderung bekerja lebih baik dan efisien dibandingkan dengan DBN, serta biasanya digunakan untuk *speech recognition* dan *information retrieval*.

Referensi Tambahan:

Activation Functions in Neural Networks.
Backpropagation.
Common Loss functions in machine learning.
Cost, Activation, Loss Function|| Neural Network|| Deep Learning. What are these?
Deep learning architectures.
Forward Propagation and Errors in a Neural Network.
Forward Propagation in Neural Networks — Simplified Math and Code Version.
How Does Back-Propagation in Artificial Neural Networks Work?
How to Choose an Activation Function for Deep Learning.
Introduction to loss functions.
Introduction to optimizers.
Keras optimizers.
Optimizers in Deep Learning.
Parameter optimization in neural networks.
The 8 Neural Network Architectures Machine Learning Researchers Need to Learn.
Types of Optimizers in Deep Learning Every AI Engineer Should Know.