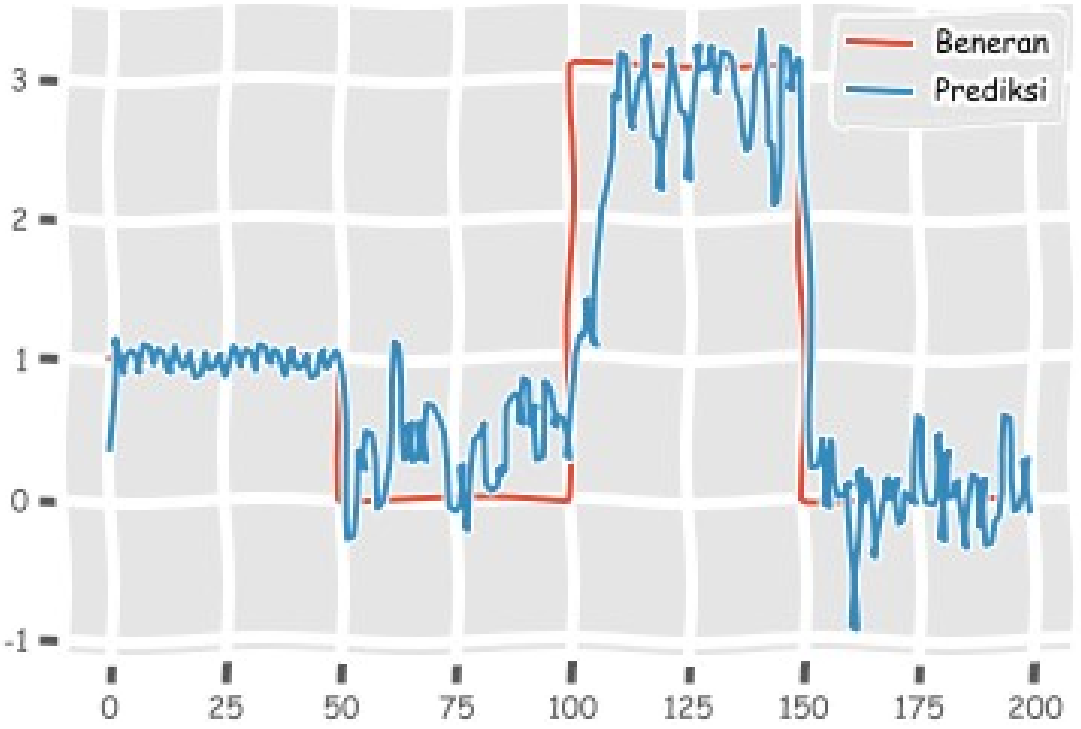
Seri Komputasi



Implementasi algoritma jaringan saraf tiruan menggunakan NeuroLab

Sandy Hardian Susanto Herho

WEATHER AND CLIMATE PREDICTION LABORATORY

GFDL

Pendahuluan

Pada tutorial ini saya akan membahas tentang aspek praktis dari algoritma jaringan saraf tiruan menggunakan pustaka NeuroLab yang berbasis pada bahasa pemrograman Python. Jaringan saraf tiruan saat ini sedang menjadi buah bibir dalam perkembangan teknologi dunia semenjak Alex Krizhevsky (pendiri laboratorium Dessa), Ilya Sutskever (ilmuwan kepala di laboratorium OpenAI), dan Geoffrey E. Hinton (guru besar di Departemen Ilmu Komputer, Universitas Toronto) menerbitkan makalah tentang penerapan pemelajaran mendalam (*deep learning*) untuk mengklasifikasikan 1,2 juta citra menggunakan arsitektur yang nantinya dikenal sebagai AlexNet (Krizhevsky dkk., 2012) pada ajang bergengsi *Conference and Workshop on Neural Information Processing Systems* (NeurIPS) yang diselenggarakan di Stateline, Nevada, Amerika Serikat tahun 2012 silam. Semenjak itu, hampir setiap perusahaan teknologi raksasa, seperti Google; Facebook; Baidu; dll melakukan ekspansi besar - besaran untuk mendanai tim riset algoritma ini.

Algoritma jaringan saraf tiruan saat ini merupakan bagian penting dari riset dan pengembangan teknologi buatan yang menjadi sendi utama dalam revolusi industri ke-4, seperti yang diterangkan oleh ekonom dunia Karl Schwab (Schwab, 2016). Presiden kita, Ir. H. Joko Widodo (beserta staf - staf kepresidenan-nya), yang menurut hemat saya juga tidak mengerti apa - apa tentang algoritma jaringan saraf tiruan, sempat mewacanakan perampingan struktur jabatan aparatur sipil negara (ASN) dengan memanfaatkan dengan menggunakan aplikasi kecerdasan buatan (Tempo, 2019), yang saya anggap sebagai suatu utopia kosong yang umum digaungkan orang - orang bodoh yang belum belajar sulitnya penerapan algoritma jaringan saraf tiruan di dunia nyata.

Daripada membicarakan politik yang bukan level-nya saya, maka baiknya kita kembali ke isi buku singkat ini, kita tidak akan membahas hal - hal yang tinggi - tinggi juga muluk - muluk seperti penerapan *Generative Adversarial Network* (GAN) (Goodfellow dkk., 2014), untuk membuat foto *bohongan* seperti yang saat ini menjadi kontroversi. Kita akan membahas hal - hal sederhana, seperti:

- konsep - konsep dasar dalam algoritma jaringan saraf tiruan,
- penerapan klasifikasi menggunakan perseptron tunggal,
- penerapan algoritma jaringan saraf buatan berlapis tunggal (*single-layer neural network*),
- penerapan algoritma jaringan saraf buatan berlapis banyak (*multi-layer neural networks*), dan
- pengantar analisis data sikuensial menggunakan algoritma *Recurrent Neural Networks* (RNNs)

Seluruhnya akan diterapkan pada pustaka NeuroLab versi 0.3.5 (alih - alih pustaka *deep learning* Python lainnya yang lebih populer seperti: TensorFlow (Abadi dkk., 2016), Keras (Chollet dkk., 2015), dan PyTorch (Paszke dkk., 2019)), yang dikembangkan oleh tiga orang pengembang Rusia dengan nama samaran: maksimovVva; zueve; dan solarjoe,untuk mempermudah para akademisi Indonesia di luar bidang ilmu komputer yang lebih terbiasa dengan *Neural Network Toolbox* (NNT) yang dijalankan pada piranti lunak berbayar: MATLAB®. Saya juga pada dasarnya akan mengikuti struktur buku Prateek Joshi (Joshi, 2017) dalam alur pembagian materi di tutorial ini. *Oh, ya* satu hal lagi yang perlu saya ingatkan karena tutorial ini bersifat praktis, saya tidak akan membahas tentang konsep matematis algoritma ini. Disarankan pembaca menunggu buku yang saya dan Dr. Dasapta Erwin Irawan (Teknik Geologi ITB) akan siapkan pada pertengahan tahun depan (jika saya masih hidup dan sehat) untuk pembahasan lebih mendasar tentang konsep *back-propagation*, fungsi aktifasi, dll. Atau kalau sudah tidak sabar, *monggo* untuk melihat buku - buku seperti: Goodfellow, dkk. (2016), Burkov (2019), dan (mungkin yang lebih *asik* dan praktis bagi saya pribadi yang *tolol* dalam matematika) Géron (2019). Tutorial ini juga tidak membahas implementasi *deep learning* menggunakan GPU seperti NVIDIA CUDA dan OpenCL, oleh karena itu pembaca diharapkan mencari tahu hal tersebut secara mandiri di mesin pencarinya masing - masing. Sebagai tambahan lagi, di sini saya juga mengharapkan pembaca punya sedikit pengalaman dengan bahasa pemrograman Python sampai tingkatan pemrograman ber-*gaya* fungsional dan mahir dalam penggunaan

pustaka NumPy (Oliphant, 2006) serta matplotlib (Hunter, 2007) karena saya tidak akan membahas lagi apa itu vektorisasi *array*, apa itu `if __name__ == '__main__':`, dan tata cara visualisasi sederhana menggunakan modul pyplot di pustaka matplotlib.

Untuk melakukan instalasi ikuti prosedur sebagai berikut:

1. Kunjungi https://docs.conda.io/projects/conda/en/latest/user-guide/install/ untuk instalasi Miniconda versi 3.
2. Ikuti prosedur instalasi (jangan lupa atur PATH -nya).
3. Ikuti prosedur instalasi git pada situs: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git.
4. Buku *Command Line Interface* (CLI), jalankan perintah:

```
git clone git@github.com:sandyherho/NeuroLab-tutor.git
```

5. Lakukan instalasi lingkungan virtual conda dengan menjalankan perintah di folder hasil *cloning* tersebut:

```
conda env create -f environment.yml
```

6. Aktifkan lingkungan virtual dengan perintah:

```
conda activate neuroLabs
```

7. Untuk mengaktifkan Jupyter Notebook, jalankan perintah:

```
jupyter notebook
```

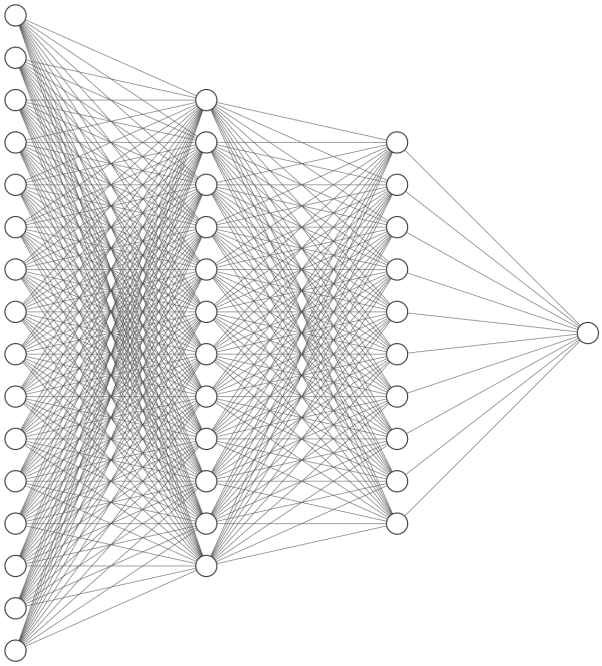
8. Jika sudah terbuka di *browser* masing - masing sesi Python interaktif dapat dimulai.
9. Untuk mengakhiri sesi tekan tombol `<CTRL> + C` di CLI, dan jalankan perintah sebagai berikut untuk menonaktifkan lingkungan virtual conda:

```
conda deactivate
```

Konsep dasar jaringan saraf tiruan

Konsep tentang algoritma jaringan saraf tiruan (ANN) pada dasarnya bukanlah konsep yang baru. Konsep ini pertama kali dicetuskan oleh McCulloch dan Pitts (1943) di dalam telaah *Bulletin of Mathematical Biophysics*. Namun pengembangan praktis untuk implementasi algoritma ini sangatlah lambat akibat kekurangan sumber data, sehingga perkembangan penelitian untuk penerapan algoritma ANN untuk kecerdasan buatan menjadi stagnan pada periode 1960an hingga awal 2000an, suatu periode yang dikenal sebagai AI Winter. Baru kemudian ketika perkembangan internet makin pesat dan data raksasa tersebar di mana - mana, terjadi perkembangan cukup signifikan pada penelitian ANN semenjak tahun 2011 yang lalu. Saat ini, algoritma ANN digunakan hampir di seluruh bidang teknologi informasi modern, seperti pengenalan suara; pengelihatan komputer; sistem rekomendasi; diagnosa medis; dll.

ANN pada dasarnya merupakan tiruan dari cara kerja sistem otak manusia di dalam melakukan pemecahan masalah. Oleh karena itu, satuan terkecil dari ANN yang dikenal sebagai perseptron juga merupakan tiruan dari neuron pada otak. Perseptron menangkap *input* dengan menggunakan fungsi transfer tertentu, seperti yang paling terkenal adalah fungsi linear dan Sigmoid (tidak akan dibahas pada tutorial singkat ini) untuk memproses data hingga menghasilkan Perseptron ini *output*. Jika jaringan perseptron ini mempunyai lebih dari dua buah lapisan, maka dikenal sebagai *deep neural networks*.



Algoritma ANN dapat digunakan untuk melakukan pemelajaran mesin tersuprevisi (*supervised learning*) maupun tidak tersupervisi (*unsupervised learning*). Seperti pada model pemelajaran mesin lainnya, model ANN juga memerlukan proses *training* untuk meminimalkan galat. Secara umum proses kerja dari algoritma ANN adalah sebagai berikut:

$$\text{Dataset} \rightarrow \text{Data numerik} \rightarrow \text{Training untuk meminimalkan galat}$$

Yang perlu diingat dari aspek praktisnya, di sini adalah suatu proses *training* pada suatu dataset hanya dapat dilakukan jika data telah diubah ke dalam bentuk numerik, tidak peduli apakah data tersebut awalnya berupa teks maupun citra.

Sebetulnya masih banyak aspek - aspek menarik, utamanya dari segi matematis dan sejarah algoritma ANN yang tidak dibahas di sini, namun karena tutorial singkat ini bersifat praktis, maka sengaja saya abaikan. Pembaca diharapkan dapat mencari sendiri di internet untuk mendalami hal - hal tersebut. Pada bagian - bagian selanjutnya, saya akan fokus sepenuhnya pada penerapan ANN secara praktis dengan menggunakan pustaka NeuroLab.

Perseptron

Seperti yang telah saya jabarkan pada bagian sebelumnya, pada dasarnya perseptron merupakan sel saraf tunggal tiruan yang merupakan bangunan dasar dari algoritma ANN. Proses kerja perseptron cukup sederhana, yakni menerima *input*, kemudian melakukan komputasi terhadap *input* numerik tersebut dengan menggunakan fungsi aktifasi (linier, *symetric-saturating linear*, log sigmoid, tangen sigmoid, atau *radial basis* (Araghinejad, 2014)), kemudian menghasilkan satu *output* yang dapat diteruskan ke perseptron pada lapisan selanjutnya sebagai *input*.

Pada bagian ini kita hanya akan membahas cara kerja perseptron tunggal menggunakan NeuroLab. Untuk mengawalinya kita harus mengimpor beberapa pustaka sebagai berikut:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
plt.xkcd() # biar lucu
plt.style.use('ggplot') # karena saya pengguna R juga biar bagus, Bro plotnya!
%matplotlib inline
```

Kemudian kita mendefinisikan dataset sederhana dalam bentuk `ndarray` dengan 2 fitur dan 1 label:

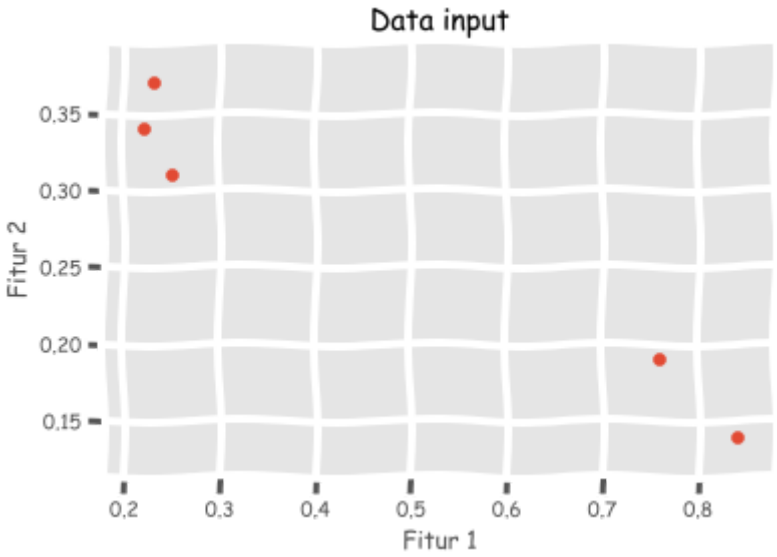
```
dataset = np.array([[.22,.34,0],
                    [.23,.37,0],
                    [.25,.31,0],
                    [.76,.19,1],
                    [.84,.14,1]])
```

Lakukan pemisahan fitur dan label dengan menggunakan perintah sebagai berikut:

```
fitur = dataset[:,:2]
label = dataset[:,2].reshape(dataset.shape[0],1) # konversi ke vektor kolom
```

Untuk melihat *input* dari dataset sederhana ini, baiknya kita tampilkan secara visual:

```
plt.scatter(fitur[:,0], fitur[:,1]);
plt.xlabel('Fitur 1');
plt.ylabel('Fitur 2');
plt.title('Data input');
```



Kemudian kita mulai kegiatan pra-pemrosesan data dengan mengatur nilai fitur minimum dan maksimum:

```
fit1_min, fit1_maks, fit2_min, fit2_maks = 0, 1, 0, 1
fit1 = [fit1_min, fit1_maks]
fit2 = [fit2_min, fit2_maks]
```

Lalu kita definisikan *output* yang kita inginkan, yakni hanya berupa perseptron tunggal:

```
output = label.shape[1]
output
```

```
1
```

Kemudian kita definisikan model perseptron tunggal dengan menggunakan fungsi newp:

```
perseptron = nl.net.newp([fit1,fit2], output)
```

Untuk meminimalkan galat, kita perlu melakukan *training* pada model ini untuk mencapai *learning rate* 0,01:

```
progres_galat = perseptron.train(fitur, label, epochs=100, show=20, lr=.01)
```

```
The goal of learning is reached
```

Kemudian kita visualisasikan hasil *training*-nya:

```
plt.plot(progres_galat);
plt.xlabel('Jumlah epochs');
plt.ylabel('Galat training');
plt.title('Progres training galat');
```

Proses *training* hanya berlangsung dalam lima *epochs* karena jumlah data yang sedikit. Karena model ini sudah *mantap*, maka saatnya kita membuat prediksi:

```
# fitur 1 = 0,8; fitur 2 = 0,2
perseptron.sim([[.8,.2]])
```

```
array([[1.]])
```

```
# fitur 1 = 0,3; fitur 2 = 0,4
perseptron.sim([[.2,.4]])
```

```
array([[0.]])
```

Patut kita ingat di sini saya tidak melakukan penilaian performa seperti yang umum dilakukan ketika kita hendak menerapkan model pemelajaran mesin seperti: *train-test splitting*, *scaling* fitur, *cross-validation*, dll karena jumlah data yang sangat kecil.

Jaringan berlapis tunggal

Jaringan berlapis tunggal (*single-layer neural network*) merupakan arsitektur ANN sederhana yang hanya mempunyai sebuah lapisan tersembunyi (*hidden layer*). Pada bagian ini kita akan menggunakan jaringan berlapis tunggal sebagai sistem pengklasifikasi sederhana. Untuk mengawalinya seperti biasa, kita harus lakukan pengimporan beberapa pustaka terlebih dahulu:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
plt.xkcd() # biar lucu
plt.style.use('ggplot') # karena saya pengguna R juga biar bagus, Bro plotnya!
%matplotlib inline
```

Kemudian kita akan menggunakan fungsi `loadtxt` untuk membuka file yang telah saya sediakan:

```
data = np.loadtxt('../data/data_nn_sederhana.txt')
data
```

```
array([[1. , 4. , 0. , 0. ],
       [1.1, 3.9, 0. , 0. ],
       [1.2, 4.1, 0. , 0. ],
       [0.9, 3.7, 0. , 0. ],
       [7. , 4. , 0. , 1. ],
       [7.2, 4.1, 0. , 1. ],
       [6.9, 3.9, 0. , 1. ],
       [7.1, 4.2, 0. , 1. ],
       [4. , 1. , 1. , 0. ],
       [4.1, 0.9, 1. , 0. ],
       [4.2, 1.1, 1. , 0. ],
       [3.9, 0.8, 1. , 0. ],
       [4. , 7. , 1. , 1. ],
       [4.2, 7.2, 1. , 1. ],
       [3.9, 7.1, 1. , 1. ],
       [4.1, 6.8, 1. , 1. ]])
```

Kemudian kita melakukan pemisahan fitur (dua kolom pertama) dan label (dua kolom terakhir):

```
fitur = data[:, :2]
fitur
```

```
array([[1. , 4. ],
       [1.1, 3.9],
```

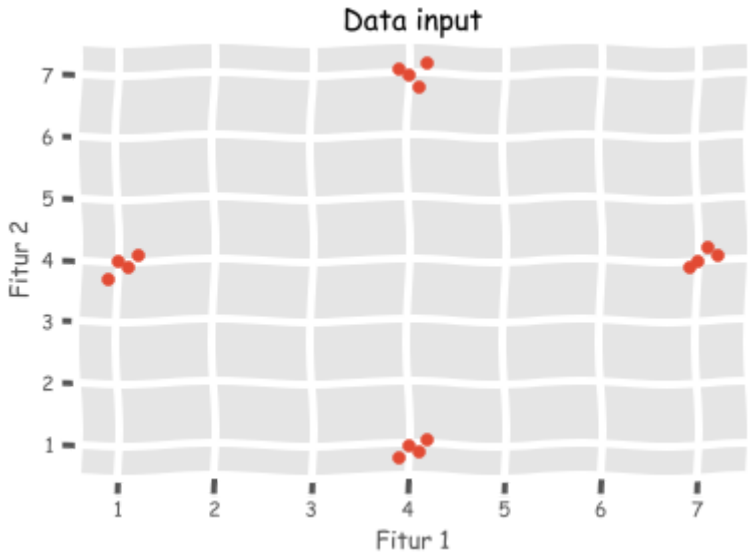
```
       [1.2, 4.1],
       [0.9, 3.7],
       [7. , 4. ],
       [7.2, 4.1],
       [6.9, 3.9],
       [7.1, 4.2],
       [4. , 1. ],
       [4.1, 0.9],
       [4.2, 1.1],
       [3.9, 0.8],
       [4. , 7. ],
       [4.2, 7.2],
       [3.9, 7.1],
       [4.1, 6.8]])
```

```
label = data[:,2:]
label
```

```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.]])
```

Supaya terbayang, mari kita visualisasikan dulu fitur - fitur pada data ini:

```
plt.scatter(fitur[:,0], fitur[:,1]);
plt.xlabel('Fitur 1');
plt.ylabel('Fitur 2');
plt.title('Data input');
```



Seperti biasa kita akan melakukan pra-pemrosesan data sebelum dijadikan *input*-an model:

```
fit1_min, fit1_maks = fitur[:,0].min(), fitur[:,0].max()
fit2_min, fit2_maks = fitur[:,1].min(), fitur[:,1].max()
fit1 = [fit1_min, fit1_maks]
fit2 = [fit2_min, fit2_maks]

output = label.shape[1]
```

Masukkan ke dalam model ANN:

```
snn = nl.net.newp([fit1,fit2], output)
```

Lakukan *training* untuk mencapai *learning rate* 0,01 dan kemudian kita visualisasikan hasilnya:

```
progres_galat = snn.train(fitur, label, epochs=100, show=20, lr=.01)

# visualisasi
plt.plot(progres_galat);
plt.xlabel('Jumlah epochs');
plt.ylabel('Galat training');
plt.title('Progres galat training');
```

```
Epoch: 20; Error: 4.0;
Epoch: 40; Error: 4.0;
Epoch: 60; Error: 4.0;
Epoch: 80; Error: 4.0;
Epoch: 100; Error: 4.0;
The maximum number of train epochs is reached
```

Ternyata kita *kebanyakan* di dalam mengatur jumlah *epochs*-nya, tapi ya sudah *lah ya*...

Sekarang saatnya kita melakukan prediksi dengan menggunakan tiga pasang nilai:

```
print('Prediksi:\n')
tes = [[0.3, 4.2], [4.3, 0.5], [4.6, 8]]
for i in tes:
    print(i, '==>', snn.sim([i])[0])
```

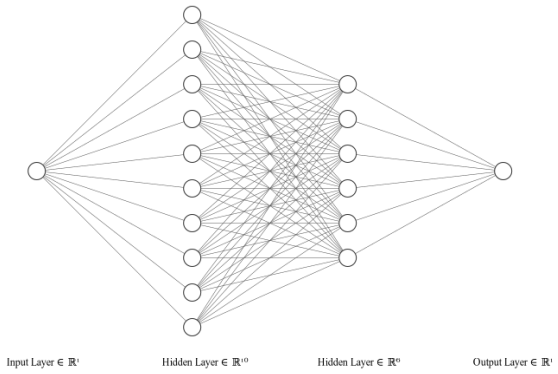
```
Prediksi:

[0.3, 4.2] ==> [0. 0.]
[4.3, 0.5] ==> [1. 0.]
[4.6, 8] ==> [1. 1.]
```

Jaringan berlapis banyak

Pada bagian ini saya akan mengajarkan pada pembaca untuk membangun arsitektur *feed-forward multi-layer neural networks* (Svozil dkk., 1997) sederhana dengan 10 perseptron di *hidden layer* pertama, 6 perseptron di *hidden layer* kedua, dan 1 perseptron *output*.

Ilustrasinya saya tampilkan pada gambar berikut ini yang saya produksi menggunakan perangkat NN-SVG yang disediakan secara gratis oleh Alex Lenail (mantan peneliti di Google Brain, saat ini mahasiswa doktoral sistem komputasi biologi di MIT):



Ok, daripada seperti kebanyakan politis yang NATO (*No Action Talks Only*), baiknya kita awali perjalanan komputasi kita dengan mengimpor beberapa pustaka seperti biasa:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
plt.xkcd() # biar lucu
plt.style.use('ggplot') # karena saya pengguna R juga biar bagus, Bro plotnya!
%matplotlib inline
```

Berbeda dengan pada pembahasan jaringan berlapis tunggal, di mana saya memperlakukan ANN sebagai sistem pengklasifikasi, kini kita akan menggunakan ANN sebagai *regressor* dari persamaan sebagai berikut:

$y = 3x^2 + 5$

Untuk itu, kita perlu mendefinisikan beberapa parameter terlebih dahulu:

```
nilai_min = -20
nilai_maks = 20
jumlah_titik = 150

x = np.linspace(nilai_min, nilai_maks, jumlah_titik)
y = 3*np.square(x) + 5
```

Dengan mempertimbangkan efisiensi sumberdaya komputasi saya yang sangat terbatas, maka ada baiknya kita lakukan normalisasi data terlebih dahulu:

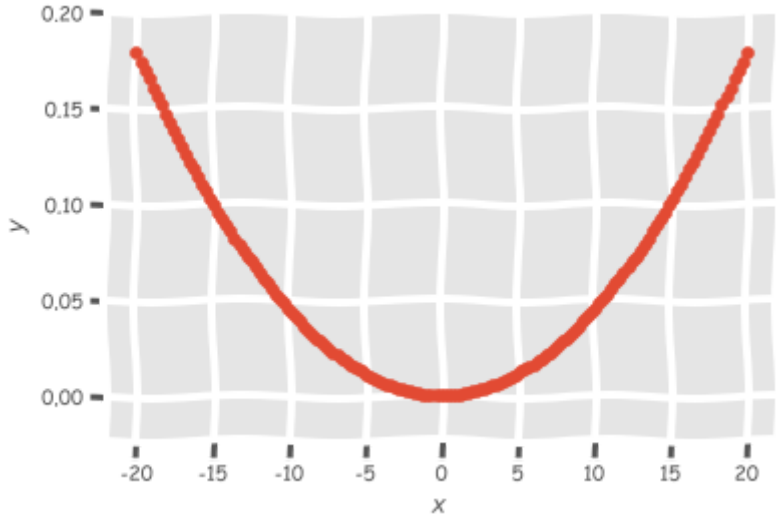
```
y /= np.linalg.norm(y)
y
```

```
array([0.17967734, 0.17490604, 0.17019922, 0.16555687, 0.160979  ,
       0.15646561, 0.1520167 , 0.14763226, 0.1433123 , 0.13905682,
       0.13486581, 0.13073928, 0.12667723, 0.12267965, 0.11874655,
       0.11487793, 0.11107379, 0.10733412, 0.10365893, 0.10004822,
       0.09650198, 0.09302022, 0.08960294, 0.08625014, 0.08296181,
       0.07973796, 0.07657858, 0.07348369, 0.07045327, 0.06748732,
       0.06458586, 0.06174887, 0.05897636, 0.05626832, 0.05362476,
       0.05104568, 0.04853108, 0.04608095, 0.0436953 , 0.04137413,
       0.03911744, 0.03692522, 0.03479748, 0.03273421, 0.03073542,
       0.02880111, 0.02693128, 0.02512592, 0.02338504, 0.02170864,
       0.02009671, 0.01854927, 0.01706629, 0.0156478 , 0.01429378,
       0.01300424, 0.01177918, 0.01061859, 0.00952248, 0.00849085,
       0.0075237 , 0.00662102, 0.00578282, 0.00500909, 0.00429984,
       0.00365507, 0.00307478, 0.00255897, 0.00210763, 0.00172076,
       0.00139838, 0.00114047, 0.00094704, 0.00081809, 0.00075361,
       0.00075361, 0.00081809, 0.00094704, 0.00114047, 0.00139838,
       0.00172076, 0.00210763, 0.00255897, 0.00307478, 0.00365507,
       0.00429984, 0.00500909, 0.00578282, 0.00662102, 0.0075237 ,
       0.00849085, 0.00952248, 0.01061859, 0.01177918, 0.01300424,
       0.01429378, 0.0156478 , 0.01706629, 0.01854927, 0.02009671,
       0.02170864, 0.02338504, 0.02512592, 0.02693128, 0.02880111,
       0.03073542, 0.03273421, 0.03479748, 0.03692522, 0.03911744,
       0.04137413, 0.0436953 , 0.04608095, 0.04853108, 0.05104568,
       0.05362476, 0.05626832, 0.05897636, 0.06174887, 0.06458586,
       0.06748732, 0.07045327, 0.07348369, 0.07657858, 0.07973796,
       0.08296181, 0.08625014, 0.08960294, 0.09302022, 0.09650198,
       0.10004822, 0.10365893, 0.10733412, 0.11107379, 0.11487793,
       0.11874655, 0.12267965, 0.12667723, 0.13073928, 0.13486581,
       0.13905682, 0.1433123 , 0.14763226, 0.1520167 , 0.15646561,
       0.160979  , 0.16555687, 0.17019922, 0.17490604, 0.17967734])
```

Kemudian kita lakukan pra-pemrosesan data dan hasilnya kita visualisasikan agar terbayang seperti apa data yang hendak kita tangani ini:

```
# pra-pemrosesan
fitur = x.reshape(jumlah_titik,1)
label = y.reshape(jumlah_titik,1)

# visualisasi
plt.scatter(fitur, label);
plt.xlabel('$x$');
plt.ylabel('$y$');
```



Lalu kita definisikan model *feed-forward multi-layer neural networks* dengan menggunakan fungsi `newff`

pada pustaka NeuroLab:

```
# 10 perseptron di hidden layer 1, 6 perseptron di hidden layer 2, 1 perseptron output
mnn = nl.net.newff([[nilai_min, nilai_maks]], [10,6,1])
```

Untuk men-*training* model yang telah kita definisikan, saya memilih menggunakan algoritma optimasi syang paling sederhana, yakni *gradient descent* (Ruder, 2017):

```
mnn.trainf = nl.train.train_gd
```

Sesudah itu, tinggal kita latih model-nya seperti biasa dengan hasil akhir galatnya hanya 0,01 dan kita visualisasikan prosesnya dalam bentuk diagram garis sederhana:

```
# training
progres_galat = mnn.train(fitur, label, epochs=2000, show=100, goal=0.01)

# visualisasi
plt.plot(progres_galat);
plt.xlabel('Jumlah epochs');
plt.ylabel('Galat training');
plt.title('Progres galat training');
```

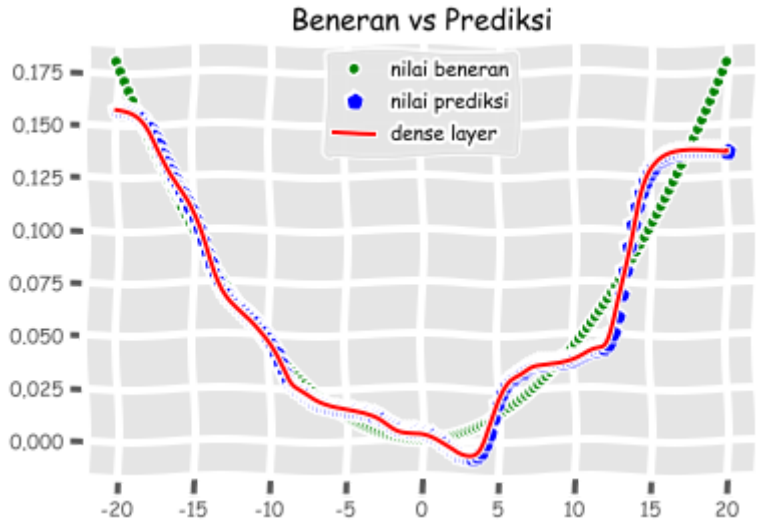
```
Epoch: 100; Error: 0.08215027262587492;
Epoch: 200; Error: 0.02098621723962554;
Epoch: 300; Error: 0.014578715088583605;
Epoch: 400; Error: 0.048040222178511095;
Epoch: 500; Error: 0.011805378213184553;
Epoch: 600; Error: 0.010520525822746438;
The goal of learning is reached
```

Kemudian kita melakukan prediksi dan memvisualisasikan hasilnya:

```
# prediksi pakai datanya sendiri
output = mnn.sim(fitur)
y_pred = output.reshape(jumlah_titik)

# dense layer
x_dense = np.linspace(nilai_min, nilai_maks, jumlah_titik*2)
y_dense_pred = mnn.sim(x_dense.reshape(x_dense.size,1)).reshape(x_dense.size)

plt.plot(x, y, '.', label='nilai beneran', color='green');
plt.plot(x, y_pred, 'p', label='nilai prediksi', color='blue');
plt.plot(x_dense, y_dense_pred, '-',label ='dense layer', color='red');
plt.title('Beneran vs Prediksi');
plt.legend();
```



Sangat *gampang*, kan?

Tentu saja tidak, *Sis-Bro*! Di proyek *deep learning* sungguhan, kita harus melakukan pra-pemrosesan data yang jauh lebih rumit dan juga harus menerapkan konsep - konsep seperti *cross-validation* dan *confusion matrix* untuk pengujian akurasi model supaya tidak *ngaco* dan menimbulkan kerugian saat menghasilkan prediksi. Hal - hal seperti ini lebih mudah diterapkan dengan menggunakan pustaka pemelajaran mendalam Python yang lebih populer seperti Keras (*bikinan* idola saya, F. Chollet), TensorFlow, PyTorch, (atau bahkan) scikit-learn. Nanti tutorialnya akan saya buat kalau saya sudah punya *laptop* yang mendukung komputasi menggunakan NVIDIA CUDA (bagi pembaca yang tertarik buat membelikan atau punya kenalan anak orang kaya yang *laptop*-nya

hanya dipakai untuk buka MS Office, boleh disumbangkan ke saya untuk mempercepat tugas ke-*nabi*-an saya di dunia ini).