Tugas 2

MA4072 - Pembelajaran Mendalam

Oleh: Michella Chandra (10118011) || Thirafi Najwan Kurniatama (13520157)

Instruksi

Buatlah 2 (dua) buah paket eksperimen klasifikasi dataset menggunakan berbagai kombinasi fungsi aktivasi dan berbagai ketentuan. Dataset yang akan diuji adalah sebagai berikut.

1. Dataset pertama (selanjutnya disebut **E1)** berupa masalah klasikasi angka, dengan data training sebanyak 1.797 observasi dan 64 *features* (variabel). Buat model ANN dengan dua buah hidden layers, setiap hidden layer terdiri dari setidaknya 128 neurons. Target yang ingin dicapai berupa hasil training dengan prediksi yang mempunyai akurasi minimal 95%.

2. Dataset kedua (selanjutnya disebut **E2)** berupa klasifikasi jenis pakaian, dengan data training sebanyak 60.000 observasi dan 784 *features* (variables). Buat model ANN dengan dua buah hidden layer, setiap hidden layer terdiri dari setidaknya 784 neurons (bila perlu neurons boleh ditambah). Gunakan data test sebanyak 10.000 observasi untuk melakukan validasi hasil prediksi. Target yang ingin dicapai berupa hasil training dengan prediksi yang mempunyai akurasi minimal 85%.

Kombinasi fungsi aktivasi yang perlu dilakukan adalah sebagai berikut.

(a) Hidden layers memakai fungsi aktivasi **Sigmoid**, output layer memakai fungsi aktivasi **Softmax** dengan fungsi kerugian (*loss function*) berupa **Cross-Entropy (log-loss)**,

(b) hidden layers memakai fungsi aktivasi **Tanh**, output layer memakai fungsi aktivasi **Softmax** dengan fungsi kerugian (*loss function*) berupa **Cross-Entropy (log-loss)**,

(c) hidden layers memakai fungsi aktivasi **Sigmoid**, output layer memakai fungsi aktivasi **Sigmoid** dengan fungsi biaya (*cost function*) berupa **Mean-Squared Errors (MSE)**, dan

(d) hidden layers memakai fungsi aktivasi **Tanh**, output layer memakai fungsi aktivasi **Sigmoid** dengan fungsi biaya (*cost function*) berupa **Mean-Squared Errors (MSE)**.

Ketentuan lain yang perlu diuji dan dilaporkan adalah sebagai berikut.

(a) Penggunaan **epoch** (jumlah iterasi) dengan angka kecil terlebih dahulu untuk menyesuaikan nilai **alpha** (dimulai dari satu), kemudian membesarkan epoch ketika sudah menemukan nilai alpha yang cocok, dan

(b) pengujian *feature* (variabel) tanpa standardisasi terlebih dahulu, dilanjutkan pengujian ke *feature* yang sudah dinormalisasi dan distandarisasi.

1 Langkah Eskperimen

1.1 Penurunan Forward dan Backward Propagation

Sebelum membuat kode, kita perlu menenurunkan terlebih dahulu variabel-variabel yang akan ditentukan pada tahap Forward Propagation dan Backward Propagation untuk setiap kombinasi fungsi aktivasi. Sebagai catatan khusus, tidak semua kombinasi fungsi aktivasi yang ditinjau akan memperhitungkan nilai bias, karena dalam percobaan yang telah dilakukan akurasi justru akan berkurang saat bias digunakan untuk kombinasi tertentu. Selain itu, untuk mempercepat konvergensi, akan digunakan metode optimisasi gradient descent **NADAM (Nesterov-Accelerated Adaptive Moment Estimation** dalam penghitungan Backward Propagation. Berikut akan dijabarkan secara singkat penurunan ini untuk setiap kombinasi fungsi aktivasi dengan input berupa data $\mathbf{X}$, output $\mathbf{y}$, dan nilai-nilai awal $\mathbf{W}, \mathbf{M}, \mathbf{V}, \beta_1, \beta_2$.

1.1.1 Sigmoid untuk hidden layers, Softmax untuk output layer, fungsi kerugian Cross-Entropy

Misalkan fungsi Sigmoid terhadap z dinyatakan sebagai $\sigma(z) = \frac{1}{1+e^{-z}}$, dan fungsi Softmax dinotasikan $S(z_{tj}) = \frac{e^{z_{tj}}}{\Sigma_{j=1}^{No} e^{(z_{tj})}}$, dengan No menyatakan banyak kelas output. Dalam tahap Forward Propagation, kita akan mendapatkan data *neuro transmitter* $\mathbf{Z}^{(l)}$ dan output $\mathbf{A}^{(l)}$ untuk layer $l = 1, 2$ dengan perhitungan sebagai berikut.

$$\begin{aligned}
\mathbf{Z}^{(1)} &= \mathbf{X}\mathbf{W}^{(0)}, \\
\mathbf{A}^{(1)} &= \sigma(\mathbf{Z}^{(1)}), \\
\mathbf{Z}^{(2)} &= \mathbf{A}^{(1)}\mathbf{W}^{(1)}, \\
\mathbf{A}^{(2)} &= \sigma(\mathbf{Z}^{(2)}).
\end{aligned}$$

Sedangkan, pada layer ketiga yaitu layer output, data *neuro transmitter* $\mathbf{Z}^{(3)}$ dan output $\mathbf{A}^{(3)}$ adalah sebagai berikut.

$$\begin{aligned}
\mathbf{Z}^{(3)} &= \mathbf{A}^{(2)}\mathbf{W}^{(2)}, \\
\mathbf{A}^{(3)} &= S(\mathbf{Z}^{(3)}).
\end{aligned}$$

Dalam tahap Backward Propagation, kita punya fungsi kerugian Cross Entropy C_j untuk kelas output j yaitu

$$C_j = -\frac{1}{T}\sum_{t=1}^{T} y_{tj} log(\hat{y}_{tj} + (1 - y_{tj}) log(1 - \hat{y}_{tj}),\ j = 1, 2, \cdots, No.$$

dan fungsi biayanya adalah

$$C = \sum_{j=1}^{No} C_j.$$

Akan dicari nilai dari $\frac{\partial \mathbf{E}}{\partial \mathbf{W}^{(l)}}$ untuk setiap layer l sebagai berikut.

$$\begin{aligned}
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(2)}} &= \mathbf{A}^{(2)\prime}\boldsymbol{\delta}^{(3)} \\
\boldsymbol{\delta}^{(3)} &= -\frac{1}{T}\mathbf{e} = -\frac{1}{T}(\mathbf{y} - \mathbf{A}^{(3)}) \\
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(1)}} &= \mathbf{A}^{(1)\prime}\boldsymbol{\delta}^{(2)} \\
\boldsymbol{\delta}^{(2)} &= \boldsymbol{\delta}^{(3)}\mathbf{W}^{(2)\prime} \circ \mathbf{A}^{(2)} \circ (1 - \mathbf{A}^{(2)}) \\
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(0)}} &= \mathbf{A}^{(0)\prime}\boldsymbol{\delta}^{(1)} = \mathbf{X}'\boldsymbol{\delta}^{(1)} \\
\boldsymbol{\delta}^{(1)} &= \boldsymbol{\delta}^{(2)}\mathbf{W}^{(1)\prime} \circ \mathbf{A}^{(1)} \circ (1 - \mathbf{A}^{(1)})
\end{aligned}$$

dimana nilai-nilai ini akan digunakan untuk menghitung $W^{(2)}, W^{(1)}, W^{(0)}$ baru di akhir dengan gradient descent. Penjabarannya sendiri ada di akhir bagian ini agar tidak redundan karena sama untuk keempat kombinasi fungsi aktivasi.

1.1.2 Tanh untuk hidden layers, Softmax untuk output layer, fungsi kerugian Cross-Entropy

Misalkan fungsi Tanh terhadap z dinyatakan sebagai $T(z) = \tanh(z)$, dan fungsi Softmax dinotasikan $S(z_{tj}) = \frac{e^{z_{tj}}}{\Sigma_{j=1}^{No} e^{(z_{tj})}}$, dengan No menyatakan banyak kelas output. Dalam tahap Forward Propagation, kita akan mendapatkan data *neuro transmitter* $\mathbf{Z}^{(l)}$ dan output $\mathbf{A}^{(l)}$ untuk layer $l = 1, 2$ dengan perhitungan sebagai berikut.

$$\begin{aligned}
\mathbf{Z}^{(1)} &= \mathbf{X}\mathbf{W}^{(0)}, \\
\mathbf{A}^{(1)} &= T(\mathbf{Z}^{(1)}), \\
\mathbf{Z}^{(2)} &= \mathbf{A}^{(1)}\mathbf{W}^{(1)}, \\
\mathbf{A}^{(2)} &= T(\mathbf{Z}^{(2)}).
\end{aligned}$$

Sedangkan, pada layer ketiga yaitu layer output, data *neuro transmitter* $\mathbf{Z}^{(3)}$ dan output $\mathbf{A}^{(3)}$ adalah sebagai berikut.

$$\begin{aligned}
\mathbf{Z}^{(3)} &= \mathbf{A}^{(2)}\mathbf{W}^{(2)}, \\
\mathbf{A}^{(3)} &= S(\mathbf{Z}^{(3)}).
\end{aligned}$$

Dalam tahap Backward Propagation, kita punya fungsi kerugian Cross Entropy C_j untuk kelas

output j yaitu

$$C_j = -\frac{1}{T}\sum_{t=1}^{T} y_{tj} log(\hat{y}_{tj} + (1 - y_{tj}) log(1 - \hat{y}_{tj}),\ j = 1, 2, \cdots, No.$$

dan fungsi biayanya adalah

$$C = \sum_{j=1}^{No} C_j.$$

Akan dicari terlebih dahulu nilai dari $\frac{\partial \mathbf{E}}{\partial \mathbf{W}^{(l)}}$ untuk setiap layer l sebagai berikut.

$$\begin{aligned}
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(2)}} &= \mathbf{A}^{(2)\prime}\boldsymbol{\delta}^{(3)} \\
\boldsymbol{\delta}^{(3)} &= -\frac{1}{T}\mathbf{e} = -\frac{1}{T}(\mathbf{y} - \mathbf{A}^{(3)}) \\
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(1)}} &= \mathbf{A}^{(1)\prime}\boldsymbol{\delta}^{(2)} \\
\boldsymbol{\delta}^{(2)} &= \boldsymbol{\delta}^{(3)}\mathbf{W}^{(2)\prime} \circ (1 - (T(\mathbf{Z}^{(2)}))^2) \\
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(0)}} &= \mathbf{A}^{(0)\prime}\boldsymbol{\delta}^{(1)} = \mathbf{X}'\boldsymbol{\delta}^{(1)} \\
\boldsymbol{\delta}^{(1)} &= \boldsymbol{\delta}^{(2)}\mathbf{W}^{(1)\prime} \circ (1 - (T(\mathbf{Z}^{(1)}))^2)
\end{aligned}$$

dimana nilai-nilai ini akan digunakan untuk menghitung $W^{(2)}, W^{(1)}, W^{(0)}$ baru di akhir dengan gradient descent.

1.1.3 Sigmoid untuk hidden layers, Sigmoid untuk output layer, fungsi kerugian MSE

Misalkan fungsi Sigmoid terhadap z dinyatakan sebagai $\sigma(z) = \frac{1}{1+e^{-z}}$. Dalam tahap Forward Propagation, kita akan mendapatkan data *neuro transmitter* $\mathbf{Z}^{(l)}$ dan output $\mathbf{A}^{(l)}$ untuk layer $l = 1, 2, 3$ dengan perhitungan sebagai berikut.

$$\begin{aligned}
\mathbf{Z}^{(1)} &= \mathbf{X}\mathbf{W}^{(0)} + \mathbf{b}^{(0)}, \\
\mathbf{A}^{(1)} &= \sigma(\mathbf{Z}^{(1)}), \\
\mathbf{Z}^{(2)} &= \mathbf{A}^{(1)}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}, \\
\mathbf{A}^{(2)} &= \sigma(\mathbf{Z}^{(2)}), \\
\mathbf{Z}^{(3)} &= \mathbf{A}^{(2)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \\
\mathbf{A}^{(3)} &= \sigma(\mathbf{Z}^{(3)}).
\end{aligned}$$

Perhatikan bahwa dalam pada kombinasi ini, kita akan memanfaatkan nilai $\mathbf{b}$ atau bias karena bisa membantu dalam menambah akurasi model.

Dalam tahap Backward Propagation, kita punya fungsi biaya MSE C untuk kelas output j yaitu

$$C = \frac{1}{2}\sum_{t=1}^{T}(\mathbf{y} - \hat{\mathbf{y}}_t)^2.$$

Akan dicari nilai dari $\frac{\partial \mathbf{E}}{\partial \mathbf{W}^{(l)}}$ untuk setiap layer l sebagai berikut.

$$\begin{aligned}
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(2)}} &= \mathbf{A}^{(2)\prime}\boldsymbol{\delta}^{(3)} \\
\boldsymbol{\delta}^{(3)} &= -\frac{1}{T}(\mathbf{y} - \mathbf{A}^{(3)}) \circ \mathbf{A}^{(3)} \circ (1 - \mathbf{A}^{(3)}) \\
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(1)}} &= \mathbf{A}^{(1)\prime}\boldsymbol{\delta}^{(2)} \\
\boldsymbol{\delta}^{(2)} &= \boldsymbol{\delta}^{(3)}\mathbf{W}^{(2)\prime} \circ \mathbf{A}^{(2)} \circ (1 - \mathbf{A}^{(2)}) \\
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(0)}} &= \mathbf{A}^{(0)\prime}\boldsymbol{\delta}^{(1)} = \mathbf{X}'\boldsymbol{\delta}^{(1)} \\
\boldsymbol{\delta}^{(1)} &= \boldsymbol{\delta}^{(2)}\mathbf{W}^{(1)\prime} \circ \mathbf{A}^{(1)} \circ (1 - \mathbf{A}^{(1)})
\end{aligned}$$

dimana nilai-nilai ini akan digunakan untuk menghitung $W^{(2)}, W^{(1)}, W^{(0)}$ baru di akhir dengan gradient descent.

1.1.4 Tanh untuk hidden layers, Sigmoid untuk output layer, fungsi kerugian MSE

Misalkan fungsi Tanh terhadap z dinyatakan sebagai $T(z) = \tanh(z)$, dan fungsi Sigmoid dinotasikan $\sigma(z) = \frac{1}{1+e^{-z}}$. Dalam tahap Forward Propagation, kita akan mendapatkan data *neuro transmitter* $\mathbf{Z}^{(l)}$ dan output $\mathbf{A}^{(l)}$ untuk layer $l = 1, 2$ dengan perhitungan sebagai berikut.

$$\begin{aligned}
\mathbf{Z}^{(1)} &= \mathbf{X}\mathbf{W}^{(0)}, \\
\mathbf{A}^{(1)} &= T(\mathbf{Z}^{(1)}), \\
\mathbf{Z}^{(2)} &= \mathbf{A}^{(1)}\mathbf{W}^{(1)}, \\
\mathbf{A}^{(2)} &= T(\mathbf{Z}^{(2)}).
\end{aligned}$$

Sedangkan, pada layer ketiga yaitu layer output, data *neuro transmitter* $\mathbf{Z}^{(3)}$ dan output $\mathbf{A}^{(3)}$ adalah sebagai berikut.

$$\begin{aligned}
\mathbf{Z}^{(3)} &= \mathbf{A}^{(2)}\mathbf{W}^{(2)}, \\
\mathbf{A}^{(3)} &= \sigma(\mathbf{Z}^{(3)}).
\end{aligned}$$

Dalam tahap Backward Propagation, kita punya fungsi biaya MSE C untuk kelas output j yaitu

$$C = \frac{1}{2}\sum_{t=1}^{T}(\mathbf{y} - \hat{\mathbf{y}}_t)^2.$$

Akan dicari nilai dari $\frac{\partial \mathbf{E}}{\partial \mathbf{W}^{(l)}}$ untuk setiap layer l sebagai berikut.

$$\begin{aligned}
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(2)}} &= \mathbf{A}^{(2)\prime}\boldsymbol{\delta}^{(3)} \\
\boldsymbol{\delta}^{(3)} &= -\frac{1}{T}(\mathbf{y} - \mathbf{A}^{(3)}) \circ \mathbf{A}^{(3)} \circ (1 - \mathbf{A}^{(3)}) \\
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(1)}} &= \mathbf{A}^{(1)\prime}\boldsymbol{\delta}^{(2)} \\
\boldsymbol{\delta}^{(2)} &= \boldsymbol{\delta}^{(3)}\mathbf{W}^{(2)\prime} \circ (1 - (T(\mathbf{Z}^{(2)}))^2) \\
\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(0)}} &= \mathbf{A}^{(0)\prime}\boldsymbol{\delta}^{(1)} = \mathbf{X}'\boldsymbol{\delta}^{(1)} \\
\boldsymbol{\delta}^{(1)} &= \boldsymbol{\delta}^{(2)}\mathbf{W}^{(1)\prime} \circ (1 - (T(\mathbf{Z}^{(1)}))^2)
\end{aligned}$$

dimana nilai-nilai ini akan digunakan untuk menghitung $W^{(2)}, W^{(1)}, W^{(0)}$ baru di akhir dengan gradient descent.

Seperti yang dikatakan sebelumnya, dalam proses *gradient descent* untuk memperbaharui nilai W, akan digunakan metode optimisasi NADAM. Dengan demikian, algoritma gradient descent untuk model ANN yang sedang dibahas adalah sebagai berikut:

$$\begin{aligned}
\mathbf{M}^{(2)} &= \beta_1\mathbf{M}^{(2)} + (1-\beta_1)\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(2)}} \\
\mathbf{V}^{(2)} &= \beta_2\mathbf{V}^{(2)} + (1-\beta_2)\left(\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(2)}}\right)^2 \\
\mathbf{W}^{(2)} &= \mathbf{W}^{(2)} - \alpha\frac{\hat{\mathbf{M}}^{(2)}}{\sqrt{\hat{\mathbf{V}}^{(2)}} + \epsilon} \\
&= \mathbf{W}^{(2)} - \alpha\frac{\mathbf{M}^{(2)}/(1-\beta_1^n)}{\sqrt{\mathbf{V}^{(2)}/(1-\beta_2^n)} + \epsilon} \\
\mathbf{M}^{(1)} &= \beta_1\mathbf{M}^{(1)} + (1-\beta_1)\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(1)}} \\
\mathbf{V}^{(1)} &= \beta_2\mathbf{V}^{(1)} + (1-\beta_2)\left(\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(1)}}\right)^2 \\
\mathbf{W}^{(1)} &= \mathbf{W}^{(1)} - \alpha\frac{\hat{\mathbf{M}}^{(1)}}{\sqrt{\hat{\mathbf{V}}^{(1)}} + \epsilon} \\
&= \mathbf{W}^{(1)} - \alpha\frac{\mathbf{M}^{(1)}/(1-\beta_1^n)}{\sqrt{\mathbf{V}^{(1)}/(1-\beta_2^n)} + \epsilon}
\end{aligned}$$

$$
\begin{aligned}
\mathbf{M}^{(0)} &= \beta_1 \mathbf{M}^{(0)} + (1-\beta_1)\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(0)}} \\
\mathbf{V}^{(0)} &= \beta_2 \mathbf{V}^{(0)} + (1-\beta_2)\left(\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{(0)}}\right)^2 \\
\mathbf{W}^{(0)} &= \mathbf{W}^{(0)} - \alpha \frac{\hat{\mathbf{M}}^{(0)}}{\sqrt{\hat{\mathbf{V}}^{(0)}} + \epsilon} \\
&= \mathbf{W}^{(0)} - \alpha \frac{\mathbf{M}^{(0)}/(1-\beta_1^n)}{\sqrt{\mathbf{V}^{(0)}/(1-\beta_2^n)} + \epsilon}
\end{aligned}
$$

dengan n menyatakan iterasi keberapa yang sedang dijalankan. Untuk kombinasi yang memakai bias, pembaharuan nilainya adalah sebagai berikut.

$$
\begin{aligned}
\mathbf{b}^{(2)} &= \mathbf{b}^{(2)} - \alpha \boldsymbol{\gamma}^{(3)} \boldsymbol{\delta}^{(3)} \\
\mathbf{b}^{(1)} &= \mathbf{b}^{(1)} - \alpha \boldsymbol{\gamma}^{(2)} \boldsymbol{\delta}^{(2)} \\
\mathbf{b}^{(0)} &= \mathbf{b}^{(0)} - \alpha \boldsymbol{\gamma}^{(1)} \boldsymbol{\delta}^{(1)}
\end{aligned}
$$

dengan $\boldsymbol{\gamma}^{(3)}$ menandakan vektor satuan dengan dimensi $T \times No$ (jumlah kelas output) $\boldsymbol{\gamma}^{(3)}$ berdimensi $T \times H_2$ (jumlah neuron di hidden layer 2), dan $\boldsymbol{\gamma}^{(3)}$ berdimensi $T \times H_1$ (jumlah neuron di hidden layer 1).

1.2 Kode Eksperimen

Kode eksperimen yang telah dibuat dapat dilihat pada notebook yang dilampirkan bersamaan dengan laporan ini, atau lewat link Google Colab berikut.
`https://colab.research.google.com/drive/15Dj4gxiNdDTN9pPo5ol1tc6kGF0Xc6Gx?usp=sharing`

Hal yang perlu diperhatikan dalam model yang kami buat adalah bahwa seed yang dipakai **bukanlah gabungan dari NIM kami**, karena seed yang digunakan haruslah bernilai dibawah $2^{32}-1$. Dengan demikian, kami memakai penjumlahannya, yaitu $10118011+13520157 = 23638168$.

Kemudian, selain menggunakan **NADAM**, terdapat beberapa metode lain yang digunakan untuk mempercepat konvergensi dari model. Yang pertama adalah **Mini-batch Gradient Descent**, yaitu varian dari Gradient Descent yang melakukan penghitungan gradien berdasarkan bagian bagian kecil dari seluruh dataset, tetapi bukan satuan. Dengan Mini-batch Gradient Descent, variansi dari pembaharuan parameter bisa diperkecil jika dibandingkan dengan Batch Gradient Descent yang biasa digunakan sehingga konvergensi bisa lebih cepat dan stabil. Aplikasi dari Mini-batch Gradient Descent ini dapat dilihat pada bagian `train` dari setiap model sebagai berikut..

```
...
def train(self, epochs, batch_size=None):
        if batch_size is None:
            batch_size = self.T
        else:
            tmp = list(zip(self.X, self.output))
            np.random.shuffle(tmp)
            self.X, self.output = zip(*tmp)
            self.X = np.array(self.X)
            self.output = np.array(self.output)
        for i in range(epochs):
            for j in range(0, self.T, batch_size):
                data = self.X[j:min(j+batch_size, self.T)]
                output = self.output[j:min(j+batch_size, self.T)]
                self.forward(data)
                self.backward(data, output, i)
...
```

Selain itu, kami menggunakan **Xavier Weight Initialization**, yaitu metode pemilihan nilai awal secara tidak acak, tepatnya dari distribusi uniform yang acak dari batas atas dan batas bawah tertentu. Aplikasi dari metode ini dapat dilihat dalam penentuan variabel inisialisasi awal setiap kelas.

```
...
        # INITIALIZE FIRST PARAM
        limw0 = np.sqrt(6)/np.sqrt(self.K+self.H1)
        self.W0 = np.random.uniform(-limw0, limw0, (self.K, self.H1))
        self.b0 = np.random.uniform(-limw0, limw0, (1, self.H1))
        # INITIALIZE SECOND PARAM
        limw1 = np.sqrt(6)/np.sqrt(self.H1+self.H2)
        self.W1 = np.random.uniform(-limw1, limw1, (self.H1, self.H2))
        self.b1 = np.random.uniform(-limw1, limw1, (1, self.H2))
        # INITIALIZE THIRD PARAM
        limw2 = np.sqrt(6)/np.sqrt(self.H2+self.N0)
        self.W2 = np.random.uniform(-limw2, limw2, (self.H2, self.N0))
        self.b2 = np.random.uniform(-limw2, limw2, (1, self.N0))
...
```

2 Hasil Eksperimen dan Pengamatan

Pembahasan hasil eksperimen akan dibagi berdasarkan dataset yang diberikan, yaitu E1 (digit) dan E2 (pakaian). Setiap dataset akan diuji dengan keempat kombinasi fungsi aktivasi, dimana akan ada tiga jenis dataset untuk setiap kombinasi tersebut: yang variabelnya dinormalisasi, distandarisasi, dan tidak diberikan perlakuan. Adapun kedua perlakuan terhadap variabel ini adalah sebagai berikut:

a) **standarisasi**

$$x_{\text{stand}} = \frac{x - \bar{x}}{\text{std}(x)} = \frac{x - \bar{x}}{\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}}$$

b) **normalisasi**

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Selain itu, akan ditunjukkan juga hasil dari model dengan nilai $\alpha = 1$, yang akan dibandingkan dengan α optimal. Dengan demikian, untuk setiap dataset, akan ada $4 \cdot 3 \cdot 2 = 24$ hasil eksperimen.

2.1 Dataset Pertama - E1

2.1.1 Hasil Eksperimen dengan *Learning Rate* 1

Akan diamati hasil eksperimen dengan $\alpha = 1$ untuk epoch berjumlah 1 sampai 10. Jumlah neuron untuk setiap layer adalah 128, dan training data dibagi ke dalam mini-batch yang masing-masing berisi 128. Berikut akan dilampirkan hasil training dari data pada Eksperimen 1 dengan variabel yang sudah dinormalisasi, distandarisasi, dan tidak diberikan perlakuan. Perhatikan bahwa dengan $\alpha = 1$, kombinasi fungsi aktivasi kedua (Tanh, Softmax, Log-Loss) tidak bisa berjalan, dengan demikian akan digunakan $\alpha = 0.1$ untuk kombinasi ini.

Hidden Layer Sigmoid, Output Softmax, Log-Loss

```
Epoch:  1 Cost: 31.05991243971936 Accuracy: 10.07 %
Epoch:  2 Cost: 25.774008222468357 Accuracy: 10.02 %
Epoch:  3 Cost: 14.529371093505308 Accuracy: 10.02 %
Epoch:  4 Cost: 14.22581711820688 Accuracy: 9.85 %
Epoch:  5 Cost: 8.655555022439543 Accuracy: 9.91 %
Epoch:  6 Cost: 7.519175200534449 Accuracy: 10.13 %
Epoch:  7 Cost: 5.271348294968337 Accuracy: 10.07 %
Epoch:  8 Cost: 4.023622206459888 Accuracy: 10.02 %
Epoch:  9 Cost: 4.6922243562523684 Accuracy: 10.13 %
Epoch:  10 Cost: 4.480879191607103 Accuracy: 9.91 %
```

Hidden Layer Tanh, Output Softmax, Log-Loss

```
Epoch:  1 Cost: 23.36363327573508 Accuracy: 10.13 %
Epoch:  2 Cost: 15.370250230110804 Accuracy: 9.96 %
Epoch:  3 Cost: 6.484552619643895 Accuracy: 10.13 %
Epoch:  4 Cost: 4.621572703276246 Accuracy: 10.02 %
Epoch:  5 Cost: 3.2015072567376213 Accuracy: 9.96 %
Epoch:  6 Cost: 2.552841939402927 Accuracy: 9.68 %
Epoch:  7 Cost: 2.4306967383690203 Accuracy: 9.96 %
Epoch:  8 Cost: 2.4703511489341654 Accuracy: 9.68 %
Epoch:  9 Cost: 2.4570875027191574 Accuracy: 9.96 %
Epoch:  10 Cost: 2.4659861808413623 Accuracy: 9.96 %
```

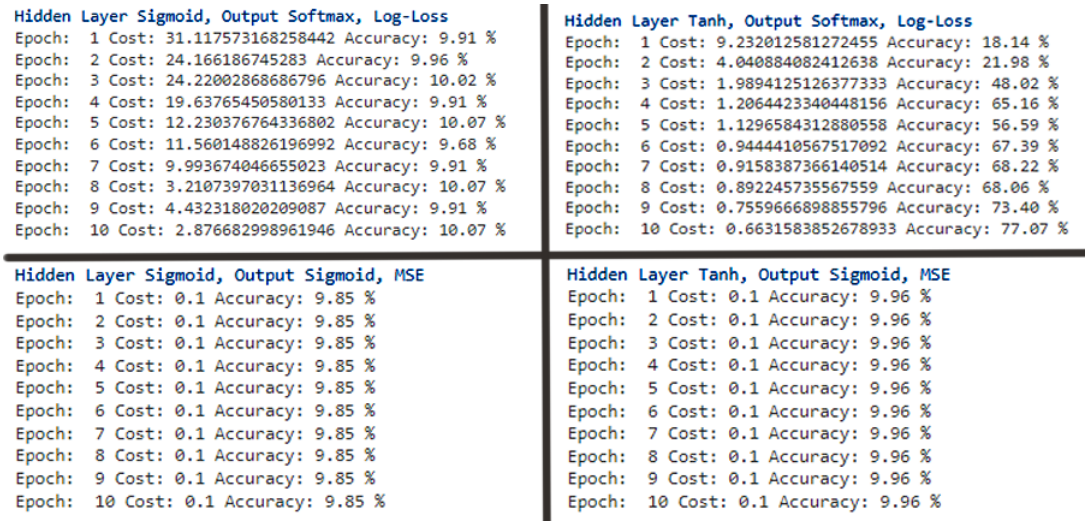
Hidden Layer Sigmoid, Output Sigmoid, MSE

```
Epoch:  1 Cost: 0.1 Accuracy: 9.85 %
Epoch:  2 Cost: 0.1 Accuracy: 9.85 %
Epoch:  3 Cost: 0.1 Accuracy: 9.85 %
Epoch:  4 Cost: 0.1 Accuracy: 9.85 %
Epoch:  5 Cost: 0.1 Accuracy: 9.85 %
Epoch:  6 Cost: 0.1 Accuracy: 9.85 %
Epoch:  7 Cost: 0.1 Accuracy: 9.85 %
Epoch:  8 Cost: 0.1 Accuracy: 9.85 %
Epoch:  9 Cost: 0.1 Accuracy: 9.85 %
Epoch:  10 Cost: 0.1 Accuracy: 9.85 %
```

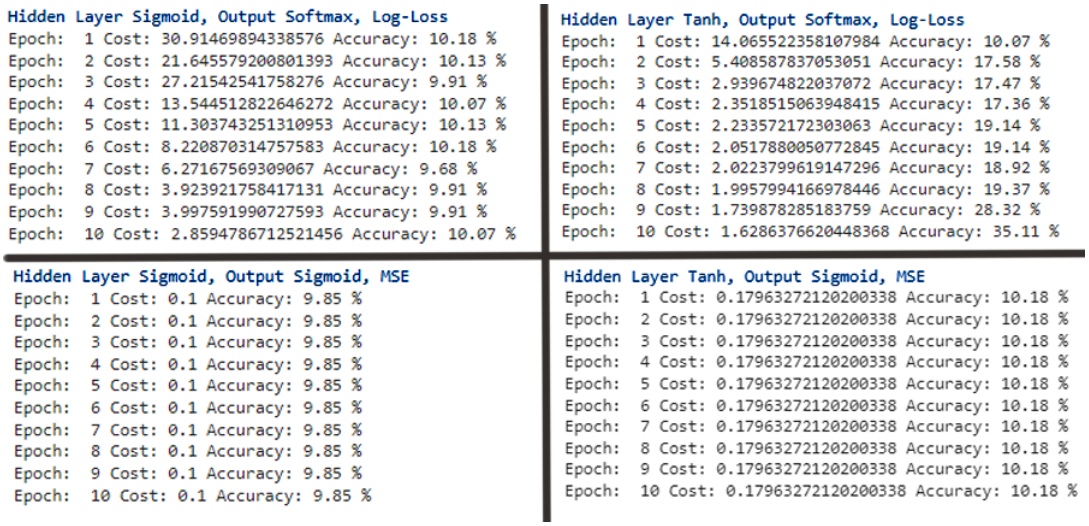
Hidden Layer Tanh, Output Sigmoid, MSE

```
Epoch:  1 Cost: 0.1 Accuracy: 9.91 %
Epoch:  2 Cost: 0.1 Accuracy: 9.91 %
Epoch:  3 Cost: 0.1 Accuracy: 9.91 %
Epoch:  4 Cost: 0.1 Accuracy: 9.91 %
Epoch:  5 Cost: 0.1 Accuracy: 9.91 %
Epoch:  6 Cost: 0.1 Accuracy: 9.91 %
Epoch:  7 Cost: 0.1 Accuracy: 9.91 %
Epoch:  8 Cost: 0.1 Accuracy: 9.91 %
Epoch:  9 Cost: 0.1 Accuracy: 9.91 %
Epoch:  10 Cost: 0.1 Accuracy: 9.91 %
```

Gambar 1: Hasil Training Data E1 **tanpa perlakuan** dengan alpha = 1 (0.1 untuk kombinasi kanan atas)



Gambar 2: Hasil Training Data E1 **dinormalisasi** dengan alpha = 1 (0.1 untuk kombinasi kanan atas)



Gambar 3: Hasil Training Data E1 **distandarisasi** dengan alpha = 1 (0.1 untuk kombinasi kanan atas)

Data nilai **cost function** || **akurasi** yang kita dapatkan bisa dibentuk ke dalam tabel berikut.

Perlakuan Variabel	**Epoch**	**Kombinasi Fungsi Aktivasi**			
		Sigmoid-Softmax-LogLoss	**Tanh-Softmax-LogLoss**	**Sigmoid-Sigmoid-MSE**	**Tanh-Sigmoid-MSE**
Tanpa Perlakuan	1	31.06 || 10.07%	23.364 || 10.13%	0.1 || 9.85%	0.1 || 9.91%
	3	14.529 || 10.02%	6.485 || 10.13%	0.1 || 9.85%	0.1 || 9.91%
	7	5.271 || 10.07%	2.43 || 9.96%	0.1 || 9.85%	0.1 || 9.91%
	10	4.481 || 9.91%	2.466 || 9.96%	0.1 || 9.85%	0.1 || 9.91%
Normalisasi	1	31.117 || 9.91%	9.232 || 18.14%	0.1 || 9.85%	0.1 || 9.96%
	3	24.22|| 10.02%	1.989 || 48.02%	0.1 || 9.85%	0.1 || 9.96%
	7	9.994 || 9.91%	0.916 || 68.22%	0.1 || 9.85%	0.1 || 9.96%
	10	2.877 || 10.07%	0.663 || 77.07%	0.1 || 9.85%	0.1 || 9.96%
Standarisasi	1	30.915 || 10.18%	14.066 || 10.07%	0.1 || 9.85%	0.1797 || 10.18%
	3	27.215 || 9.91%	2.9397 || 17.47%	0.1 || 9.85%	0.1797 || 10.18%
	7	6.272 || 9.68%	2.022 || 18.92%	0.1 || 9.85%	0.1797 || 10.18%
	10	2.859 || 10.07%	1.629 || 35.11%	0.1 || 9.85%	0.1797 || 10.18%

Tabel 1: Perbandingan Cost Function dan Akurasi E1 dengan $\alpha = 1$ (0.1 untuk kombinasi kedua)

2.1.2 Hasil Eksperimen dengan *Learning Rate* Optimal

Setelah melakukan pengulangan percobaan untuk beberapa kali, ditemukan nilai α yang optimal untuk semua fungsi aktivasi adalah sebesar **0.001**. Akan ditinjau hasil eksperimen dengan $\alpha = 0.001$ untuk epoch berjumlah 1 sampai 10. Jumlah neuron untuk setiap layer adalah 128, dan data akan dibagi ke dalam batch yang masing-masing berisi 128 training data. Berikut dilampirkan data yang bersangkutan.

Hidden Layer Sigmoid, Output Softmax, Log-Loss

```
Epoch:  1 Cost: 1.8542754488438742 Accuracy: 56.48 %
Epoch:  2 Cost: 1.4482946889595327 Accuracy: 84.81 %
Epoch:  3 Cost: 1.175784040040996 Accuracy: 88.31 %
Epoch:  4 Cost: 0.9936444334279915 Accuracy: 89.93 %
Epoch:  5 Cost: 0.8629863786210981 Accuracy: 91.15 %
Epoch:  6 Cost: 0.7649860213070719 Accuracy: 91.93 %
Epoch:  7 Cost: 0.6862840107714709 Accuracy: 93.10 %
Epoch:  8 Cost: 0.6220518960978149 Accuracy: 93.54 %
Epoch:  9 Cost: 0.5692806720317707 Accuracy: 93.93 %
Epoch:  10 Cost: 0.5246426018144641 Accuracy: 94.49 %
```

Hidden Layer Tanh, Output Softmax, Log-Loss

```
Epoch:  1 Cost: 0.5745352696834305 Accuracy: 86.20 %
Epoch:  2 Cost: 0.3015463079962684 Accuracy: 92.65 %
Epoch:  3 Cost: 0.21104172022031029 Accuracy: 94.99 %
Epoch:  4 Cost: 0.16891191364598523 Accuracy: 96.55 %
Epoch:  5 Cost: 0.1435358713746371 Accuracy: 97.72 %
Epoch:  6 Cost: 0.1257124105221612 Accuracy: 98.16 %
Epoch:  7 Cost: 0.11224953169626664 Accuracy: 98.44 %
Epoch:  8 Cost: 0.10142405010050494 Accuracy: 98.78 %
Epoch:  9 Cost: 0.09247092985172102 Accuracy: 98.78 %
Epoch:  10 Cost: 0.0848671885883983 Accuracy: 98.89 %
```

Hidden Layer Sigmoid, Output Sigmoid, MSE

```
Epoch:  1 Cost: 0.08896645338068868 Accuracy: 28.38 %
Epoch:  2 Cost: 0.08859319237079732 Accuracy: 17.20 %
Epoch:  3 Cost: 0.08550220403278172 Accuracy: 26.21 %
Epoch:  4 Cost: 0.08169853246635943 Accuracy: 39.18 %
Epoch:  5 Cost: 0.07757064334710241 Accuracy: 58.43 %
Epoch:  6 Cost: 0.0733244078537239 Accuracy: 70.23 %
Epoch:  7 Cost: 0.06904679615483453 Accuracy: 76.96 %
Epoch:  8 Cost: 0.06487813413655173 Accuracy: 80.91 %
Epoch:  9 Cost: 0.0609400721953412 Accuracy: 82.47 %
Epoch:  10 Cost: 0.05734682155531985 Accuracy: 83.36 %
```

Hidden Layer Tanh, Output Sigmoid, MSE

```
Epoch:  1 Cost: 0.06785613471694339 Accuracy: 64.39 %
Epoch:  2 Cost: 0.046534552654141206 Accuracy: 75.01 %
Epoch:  3 Cost: 0.036652652817885725 Accuracy: 80.80 %
Epoch:  4 Cost: 0.029843319639472028 Accuracy: 87.59 %
Epoch:  5 Cost: 0.02521310582415535 Accuracy: 91.71 %
Epoch:  6 Cost: 0.022070485610371626 Accuracy: 92.60 %
Epoch:  7 Cost: 0.019761368385419222 Accuracy: 93.49 %
Epoch:  8 Cost: 0.01796234074446923 Accuracy: 94.32 %
Epoch:  9 Cost: 0.01649619576794578 Accuracy: 94.82 %
Epoch:  10 Cost: 0.015263174794093105 Accuracy: 95.49 %
```

Gambar 4: Hasil Training Data E1 **tanpa perlakuan** dengan alpha = 0.001

Hidden Layer Sigmoid, Output Softmax, Log-Loss

```
Epoch:  1 Cost: 2.1305124497785104 Accuracy: 29.05 %
Epoch:  2 Cost: 1.909816275286345 Accuracy: 68.17 %
Epoch:  3 Cost: 1.7229025678528007 Accuracy: 80.80 %
Epoch:  4 Cost: 1.5638994361121 Accuracy: 84.03 %
Epoch:  5 Cost: 1.4305972927571677 Accuracy: 85.87 %
Epoch:  6 Cost: 1.3198315409322947 Accuracy: 86.64 %
Epoch:  7 Cost: 1.2268794462493702 Accuracy: 87.09 %
Epoch:  8 Cost: 1.1475051212583105 Accuracy: 87.65 %
Epoch:  9 Cost: 1.078567725933378 Accuracy: 88.43 %
Epoch:  10 Cost: 1.0178312677927452 Accuracy: 88.98 %
```

Hidden Layer Tanh, Output Softmax, Log-Loss

```
Epoch:  1 Cost: 0.49079986430592626 Accuracy: 88.87 %
Epoch:  2 Cost: 0.28244264952244647 Accuracy: 93.16 %
Epoch:  3 Cost: 0.21303907388867538 Accuracy: 95.33 %
Epoch:  4 Cost: 0.18205948940263605 Accuracy: 95.77 %
Epoch:  5 Cost: 0.1623333809629589 Accuracy: 96.38 %
Epoch:  6 Cost: 0.1482400810376816 Accuracy: 96.94 %
Epoch:  7 Cost: 0.13721159039628766 Accuracy: 97.11 %
Epoch:  8 Cost: 0.12815848675769217 Accuracy: 97.27 %
Epoch:  9 Cost: 0.12047653852333932 Accuracy: 97.61 %
Epoch:  10 Cost: 0.11381716056967908 Accuracy: 97.83 %
```

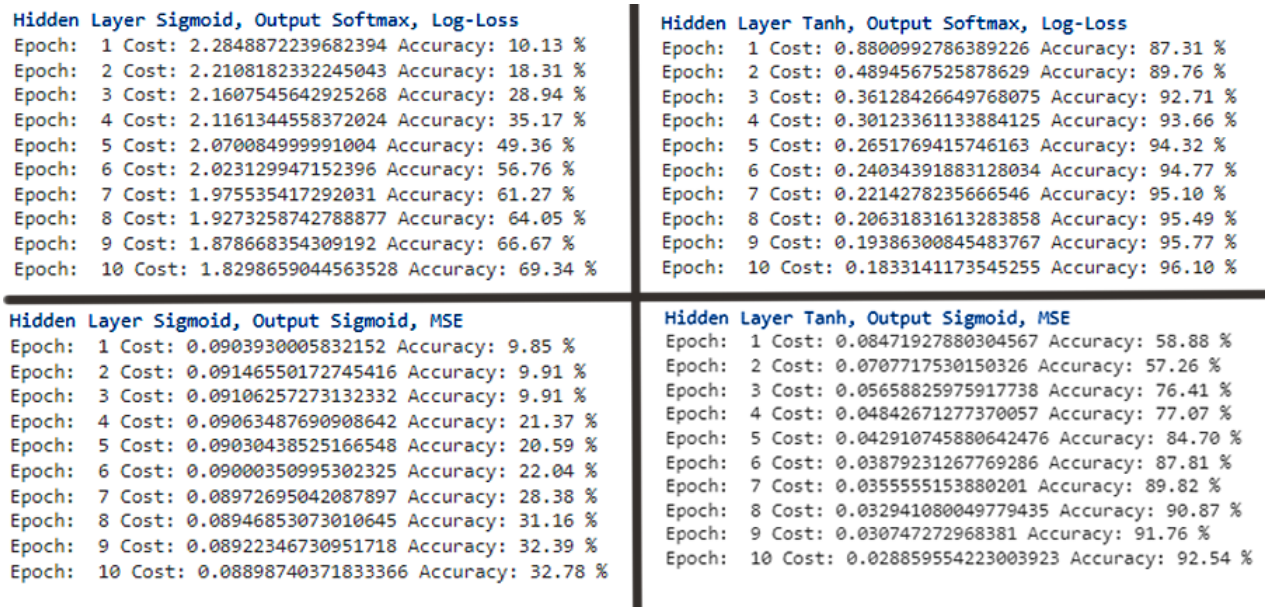
Hidden Layer Sigmoid, Output Sigmoid, MSE

```
Epoch:  1 Cost: 0.08985951923783572 Accuracy: 15.36 %
Epoch:  2 Cost: 0.0906230061980482 Accuracy: 10.52 %
Epoch:  3 Cost: 0.08965883860008955 Accuracy: 17.31 %
Epoch:  4 Cost: 0.08856574419127763 Accuracy: 29.66 %
Epoch:  5 Cost: 0.087443023005814 Accuracy: 29.72 %
Epoch:  6 Cost: 0.086219568083402 Accuracy: 31.61 %
Epoch:  7 Cost: 0.0848813516146429 Accuracy: 43.41 %
Epoch:  8 Cost: 0.08339583834484844 Accuracy: 53.42 %
Epoch:  9 Cost: 0.0817412523273732 Accuracy: 58.26 %
Epoch:  10 Cost: 0.07991886666289497 Accuracy: 60.71 %
```

Hidden Layer Tanh, Output Sigmoid, MSE

```
Epoch:  1 Cost: 0.06291348056841581 Accuracy: 69.17 %
Epoch:  2 Cost: 0.042995783074561034 Accuracy: 78.91 %
Epoch:  3 Cost: 0.03228832256421678 Accuracy: 90.98 %
Epoch:  4 Cost: 0.027004130428393787 Accuracy: 92.82 %
Epoch:  5 Cost: 0.02356268641527965 Accuracy: 94.16 %
Epoch:  6 Cost: 0.021224157117355064 Accuracy: 94.32 %
Epoch:  7 Cost: 0.019459735749271047 Accuracy: 94.71 %
Epoch:  8 Cost: 0.018066733167134356 Accuracy: 95.10 %
Epoch:  9 Cost: 0.01692551271279646 Accuracy: 95.27 %
Epoch:  10 Cost: 0.015960999691776036 Accuracy: 95.49 %
```

Gambar 5: Hasil Training Data E1 **dinormalisasi** dengan alpha = 0.001



Gambar 6: Hasil Training Data E1 **distandarisasi** dengan alpha = 0.001

Data nilai **cost function** || **akurasi** yang kita dapatkan bisa dibentuk ke dalam tabel berikut.

Perlakuan Variabel	Epoch	Kombinasi Fungsi Aktivasi			
		Sigmoid-Softmax-LogLoss	Tanh-Softmax-LogLoss	Sigmoid-Sigmoid-MSE	Tanh-Sigmoid-MSE
Tanpa Perlakuan	1	1.854 || 56.48%	0.575 || 86.2%	0.089|| 28.38%	0.068 || 64.39%
	3	1.176 || 88.31%	0.211 || 94.99%	0.086 || 26.21%	0.037 || 80.80%
	7	0.686 || 93.10%	0.112 || 98.44%	0.069 || 76.96%	0.0198 || 93.49%
	10	0.525 || 94.49%	0.085 || 98.89%	0.0573 || 83.36%	0.0153 || 95.49%
Normalisasi	1	2.13 || 29.05%	0.49 || 88.87%	0.0899 || 15.36%	0.063|| 69.17%
	3	1.723 || 80.80%	0.213 || 95.33%	0.0897 || 17.31%	0.0323 || 90.98%
	7	1.227 || 87.09%	0.137 || 97.11%	0.085 || 43.41%	0.0195 || 94.71%
	10	1.018 || 88.98%	0.114 || 97.83%	0.0799 || 60.71%	0.016 || 95.49%
Standarisasi	1	2.285 || 10.13%	0.88 || 87.31%	0.0904 || 9.85%	0.085 || 58.88%
	3	2.161 || 28.94%	0.361 || 92.71%	0.091 || 9.91%	0.057|| 76.41%
	7	1.976 || 61.27%	0.221 || 95.1%	0.0897 || 28.38%	0.036 || 89.82%
	10	1.83 || 69.34%	0.183 || 96.1%	0.089 || 32.78%	0.0289 || 92.54%

Tabel 2: Perbandingan Cost Function dan Akurasi E1 dengan $\alpha = 0.001$

2.1.3 Pembahasan

Seperti apa yang diduga sebelumnya, pengujian dengan nilai $\alpha = 1$ menghasilkan akurasi yang sangat buruk meskipun jumlah epoch sudah diperbesar. Nilai Cost Function tetap berkurang seiring epoch membesar, namun perubahannya tidaklah seberapa sehingga tingkat akurasi hanya meningkat sedikit, stagnan, ataupun berkurang. Stagnansi paling besar terlihat untuk kombinasi fungsi aktivasi ketiga dan keempat, tepatnya yang memakai cost function MSE. Karena untuk setiap model akurasi bisa bertambah maupun berkurang di pengulangan berapapun sewaktu-waktu, tidak hanya α, nilai epoch pun perlu dipilih dengan tepat.

Fungsi aktivasi kedua yaitu **Tanh-Softmax-LogLoss** juga mengalami masalah yang sama pada nilai $\alpha = 0.1$. Akan tetapi, pada kombinasi fungsi aktivasi dan nilai α yang sama, akurasi model meningkat drastis ketika variabel input dinormalisasi atau distandarisasi. Hal ini membuktikan bahwa perlakuan standarisasi atau normalisasi terhadap data input dapat mengakibatkan perubahan performa model.

Sekarang, mari kita tinjau hasil eksperimen dengan *learning rate* yang optimal. Nilai α atau *learning rate* yang diambil adalah **0.01** untuk semua kombinasi fungsi aktivasi, karena dapat menghasilkan akurasi yang sangat baik dengan cost function yang relatif kecil. Mayoritas hasil training membuahkan hasil yang baik, dengan akurasi rata-rata diatas 90% untuk kombinasi pertama, kedua, dan keempat. Akurasi dari kombinasi Sigmoid-MSE membuahkan hasil yang kurang baik karena memang lebih cocok digunakan untuk model prediksi dibandingkan klasifikasi.

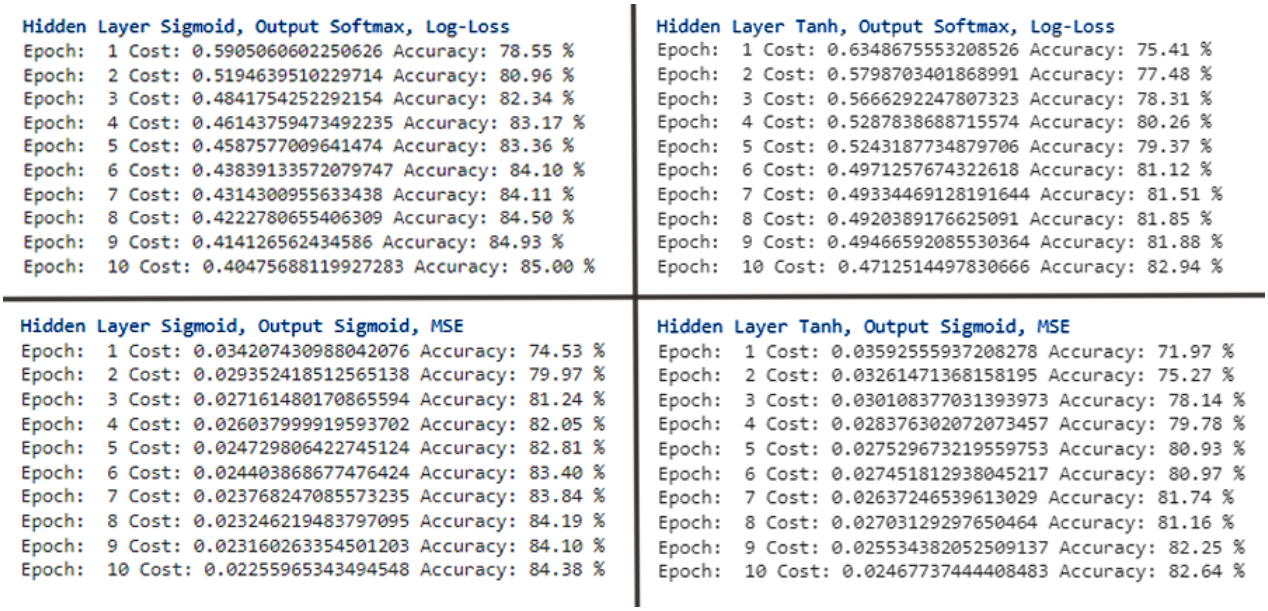
Ketika membandingkan hasil training data Eksperimen 1 antar jenis perlakuan variabel dan fungsi aktivasi yang berbeda, ditemukan bahwa secara umum hasil training model dengan data yang variabelnya tidak diberikan perlakuan apapun akan memiliki performa lebih baik dibandingkan yang dinormalisasi atau distandarisasi. Hal ini tentunya tidak berlaku secara umum dan bergantung dengan jenis dan jumlah data training. Sebagai perbandingan, model training Eksperimen 2 yang akan dibahas setelah ini memiliki performa terbaik jika datanya dinormalisasi terlebih dahulu.

Dengan akurasi 98.89%, kita dapat mengambil kesimpulan bahwa dengan epoch 10 dan neuron sebanyak 128 untuk kedua hidden layer model Deep Learning yang paling akurat untuk menguji data pada Eksperimen 1 adalah kombinasi fungsi aktivasi **Tanh** pada hidden layer, **Softmax** pada layer output, dan loss function **Log Loss**, dengan variabel yang tidak diberi perlakuan. Selain itu, untuk mencapai akurasi 95% pada epoch ke-10 untuk jumlah neuron 128, kita dapat menggunakan kombinasi fungsi aktivasi Tanh-Softmax-LogLoss untuk semua perlakuan atau Tanh-Sigmoid-MSE dengan data yang tidak diberi perlakuan atau dinormalisasi.

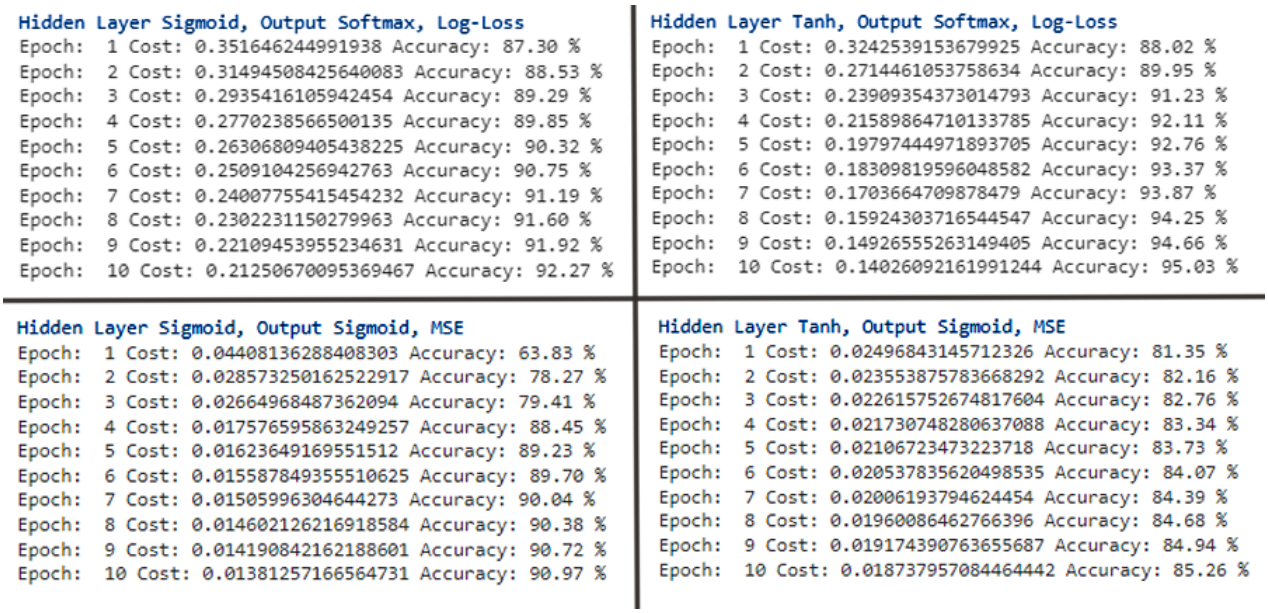
2.2 Dataset Kedua - E2

2.2.1 Hasil Eksperimen

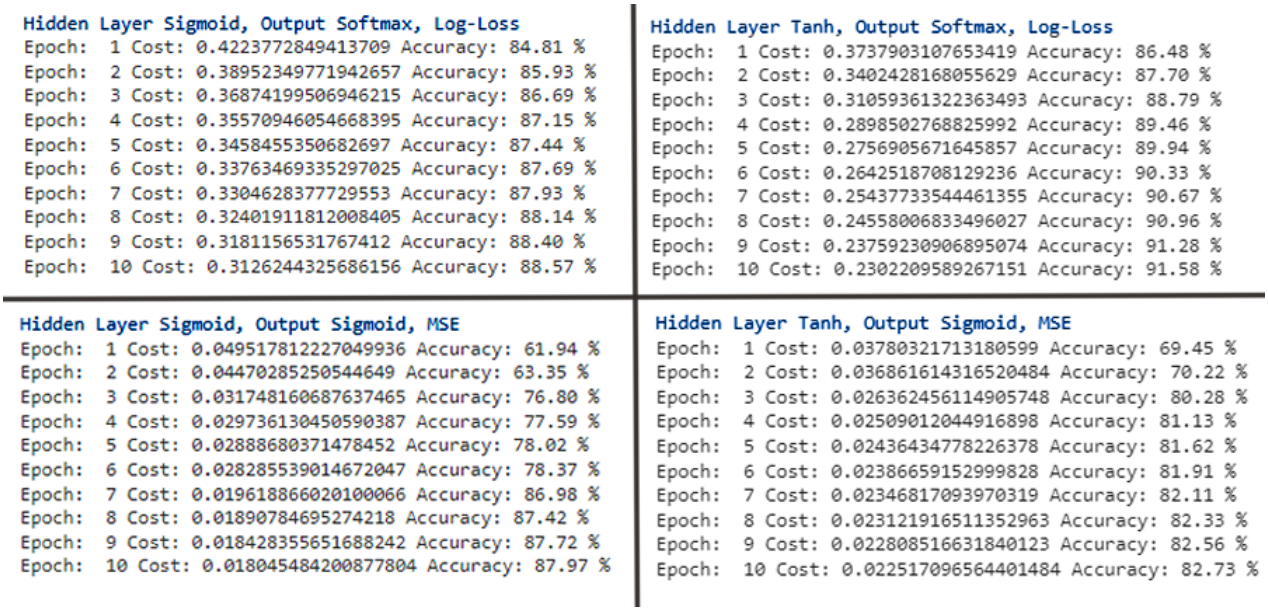
Karena keterbatasan waktu dan kemampuan mesin *hardware* yang digunakan, hanya akan ditampilkan Hasil Eksperimen untuk Dataset E2 dengan Learning Rate yang optimal, yaitu **0.001**. Seperti sebelumnya, epoch berjumlah 1 sampai 10 dan training data dibagi ke dalam mini-batch berukuran 128. Jumlah neuron di setiap hidden layer adalah 784. Berikut adalah hasil training dari data Eksperimen 2 dengan variabel yang tidak dikenakan perlakuan, dinormalisasi, dan distandarisasi.



Gambar 7: Hasil Training Data E2 **tanpa perlakuan** dengan alpha = 0.001



Gambar 8: Hasil Training Data E2 **dinormalisasi** dengan alpha = 0.001



Gambar 9: Hasil Training Data E2 **distandarisasi** dengan alpha = 0.001

Data **cost function** || **akurasi** yang kita dapatkan bisa dibentuk ke dalam tabel sebagai berikut.

Perlakuan Variabel	Epoch	Kombinasi Fungsi Aktivasi			
		Sigmoid-Softmax-LogLoss	Tanh-Softmax-LogLoss	Sigmoid-Sigmoid-MSE	Tanh-Sigmoid-MSE
Tanpa Perlakuan	1	0.5905 || 78.55%	0.635 || 75.41%	0.034 || 74.53%	0.0359 || 71.97%
	3	0.484 || 82.34%	0.567 || 78.31%	0.027 || 81.24%	0.0301 || 78.14%
	7	0.431 || 84.11%	0.493 || 81.51%	0.0237 || 83.84%	0.0264 || 81.74%
	10	0.405 || 85%	0.471|| 82.94%	0.0226|| 84.38%	0.0247 || 82.64%
Normalisasi	1	0.352|| 87.3%	0.324|| 88.02%	0.044 || 63.83%	0.025 || 81.35%
	3	0.294 || 89.29%	0.239 || 91.23%	0.0266 || 79.41%	0.0226 || 82.76%
	7	0.24|| 91.19%	0.17 || 93.37%	0.015|| 90.04%	0.02|| 84.39%
	10	0.213|| 92.27%	0.14 || 95.03%	0.014 || 90.97%	0.019|| 85.26%
Standarisasi	1	0.422 || 84.81%	0.374 || 86.48%	0.0495 || 61.94%	0.0378 || 69.45%
	3	0.369 || 86.69%	0.311 || 88.79%	0.032 || 76.80%	0.0264|| 80.28%
	7	0.33 || 87.93%	0.254 || 90.67%	0.0196 || 86.98%	0.0235 || 82.11%
	10	0.313 || 88.57%	0.23 || 91.58%	0.018 || 87.97%	0.0225 || 82.73%

Tabel 3: Perbandingan Cost Function dan Akurasi E2 dengan $\alpha = 0.001$

2.2.2 Pembahasan

Amati Tabel 3. Berbeda dengan Eksperimen 1, hasil eksperimen dengan variabel yang sudah dinormalisasi memiliki tingkat akurasi terbesar dan nilai loss function terkecil untuk semua kombinasi fungsi aktivasi jika dibandingkan dengan yang tidak memiliki perlakuan ataupun distandarisasi. Tanpa perlakuan terhadap variabel, kombinasi fungsi aktivasi yang mencapai batas minimal akurasi hanyalah Sigmoid-Softmax-LogLoss, sedangkan untuk perlakuan variabel yang dinormalisasi dan standarisasi, hanya kombinasi Tanh-Sigmoid-MSE dengan data yang distandarisasi sajalah yang tidak mencapai 85% akurasi.

Perbedaan lain dari Eksperimen 1 adalah bahwa perlakuan dataset yang berbeda bisa menghasilkan perbedaan kombinasi fungsi aktivasi paling optimal untuk memodelkan data. Dalam hal ini, kombinasi Sigmoid-Softmax-LogLoss memberikan performa terbaik untuk data Eksperimen 2 yang tidak diberikan perlakuan. Namun, jika data dinormalisasi atau distandarisasi, kombinasi fungsi yang dipakai untuk menghasilkan tingkat akurasi paling tinggi adalah Tanh-Softmax-LogLoss.

Dengan melihat keseluruhan hasil pengujian, ditemukan bahwa kombinasi fungsi aktivasi yang memiliki akurasi paling tinggi adalah **Tanh-Softmax-LogLoss** dengan variabel yang sudah **dinormalisasi**, yang memiliki akurasi 95.03% di epoch ke-10 dengan neuron sejumlah 784 di kedua hidden layer. Secara keseluruhan, terlihat juga bahwa kombinasi fungsi aktivasi dengan cost function Log Loss (pertama dan kedua) lebih baik dibandingkan dengan yang memakai MSE (ketiga keempat). Untuk mencapai minimum akurasi 85% pada epoch ke-10 untuk jumlah neuron 784, kita dapat menggunakan kombinasi fungsi aktivasi Sigmoid-Softmax-LogLoss untuk semua perlakuan data, kombinasi Tanh-Softmax-LogLoss dan Sigmoid-MSE untuk data yang dinormalisasi dan distandarisasi, serta Tanh-Sigmoid-MSE untuk data yang sudah dinormalisasi.

3 Kesimpulan

Dengan menggunakan epoch sebesar 10 dan $\alpha = 0.001$ untuk semua uji, ditemukan kesimpulan hasil sebagai berikut.

1. Dengan jumlah neuron sebanyak 128 untuk kedua hidden layer, model Deep Learning yang paling akurat untuk menguji data Eksperimen 1 adalah kombinasi fungsi aktivasi **Tanh** pada hidden layer, **Softmax** pada layer output, dan loss function **Log Loss** tanpa perlakuan terhadap variabel. Tingkat akurasi model ini adalah **98.89%** pada epoch ke-10.

2. Cara termudah untuk mencapai tingkat akurasi minimal 95% di epoch 10 dan neuron 128 pada Eksperimen 1 adalah dengan menggunakan kombinasi fungsi aktivasi **Tanh-Softmax-LogLoss**.

3. Dengan jumlah neuron sebanyak 784 untuk kedua hidden layer, model Deep Learning yang paling akurat untuk menguji data Eksperimen 2 adalah kombinasi fungsi aktivasi **Tanh** pada hidden layer, **Softmax pada layer output**, dan loss function **Log Loss** disertai dengan **normalisasi variabel**. Tingkat akurasi model ini adalah **95.03%** pada epoch ke-10.

4. Cara termudah untuk mencapai tingkat akurasi minimal 85% di epoch 10 dan neuron 784 pada Eksperimen 2 adalah dengan menggunakan kombinasi fungsi aktivasi **Sigmoid-Softmax-LogLoss** atau dengan melakukan **normalisasi** data.