

USING NEURAL NETWORK FOR THE DAMPING FUNCTIONS IN THE $k - \omega$ AKN TURBULENCE MODEL

Lars Davidson, Div. of Fluid Dynamics
Dept. of Mechanical Engineering (ME)
Chalmers University of Technology
Gothenburg, Sweden

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
 - **Photographs** where the machine learning algorithm should recognize, e.g., traffic lights [7].

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
 - **Photographs** where the machine learning algorithm should recognize, e.g., traffic lights [7].
 - **ECG signals** where the machine learning algorithm should recognize certain unhealthy conditions of the heart [5].

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
 - **Photographs** where the machine learning algorithm should recognize, e.g., traffic lights [7].
 - **ECG signals** where the machine learning algorithm should recognize certain unhealthy conditions of the heart [5].
 - **Detecting fraud** for credit card payments [6].

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
 - **Photographs** where the machine learning algorithm should recognize, e.g., traffic lights [7].
 - **ECG signals** where the machine learning algorithm should recognize certain unhealthy conditions of the heart [5].
 - **Detecting fraud** for credit card payments [6].
- In my case, input and output are **numerical** values.

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
 - **Photographs** where the machine learning algorithm should recognize, e.g., traffic lights [7].
 - **ECG signals** where the machine learning algorithm should recognize certain unhealthy conditions of the heart [5].
 - **Detecting fraud** for credit card payments [6].
- In my case, input and output are **numerical** values.
- The ML will then be some form of **regression method**.

You will use NN to model damping functions in the AKN turbulence model which reads [1]

$$\frac{\partial \bar{\nu}_j k}{\partial x_j} = \frac{\partial}{\partial x_j} \left[\left(\nu + \frac{\nu_t}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] + P_k - \varepsilon, \quad P_k = \nu_t \left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i} \right) \frac{\partial \bar{v}_i}{\partial x_j}$$

$$\frac{\partial \bar{\nu}_j \varepsilon}{\partial x_j} = \frac{\partial}{\partial x_j} \left[\left(\nu + \frac{\nu_t}{\sigma_\varepsilon} \right) \frac{\partial \varepsilon}{\partial x_j} \right] + C_{\varepsilon 1} P_k \frac{\varepsilon}{k} - C_{\varepsilon 2} f_2 \frac{\varepsilon^2}{k}$$

$$\nu_t = C_\mu f_\mu \frac{k^2}{\varepsilon}, \quad C_{\varepsilon 1} = 1.5, \quad C_{\varepsilon 2} = 1.9, \quad C_\mu = 0.09, \quad \sigma_k = 1.4, \quad \sigma_\varepsilon = 1.4$$

The damping functions are defined as

$$f_2 = \left[1 - \exp \left(-\frac{y^*}{3.1} \right) \right]^2 \left\{ 1 - 0.3 \exp \left[-\left(\frac{R_t}{6.5} \right)^2 \right] \right\}, \quad y^* = \frac{u_\varepsilon y}{\nu}$$

$$f_\mu = \left[1 - \exp \left(-\frac{y^*}{14} \right) \right]^2 \left\{ 1 + \frac{5}{R_t^{3/4}} \exp \left[-\left(\frac{R_t}{200} \right)^2 \right] \right\}, \quad u_\varepsilon = (\varepsilon \nu)^{1/4}, \quad R_t = \frac{k^2}{\nu \varepsilon}$$

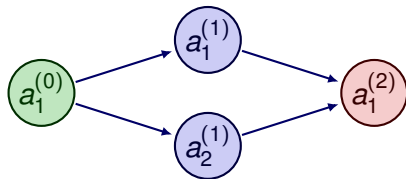
Wall b.c. is implemented by setting ε at the wall-adjacent cells as $\varepsilon = 2\nu k/y^2$

CRASH COURSE ON NEURAL NETWORK (NN)

Below I give you some useful links for NN and PyTorch.

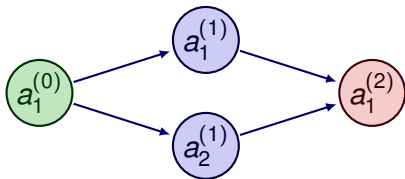
- For in-depth understanding of autograd and NN, the reader is referred to [this report \[3\]](#) and the references therein.
- Good YouTube lectures
 - 3Blue1Brown: But what is a neural network
 - 3Blue1Brown: gradient descent, how neural network learn
 - 3Blue1Brown: backpropagation, intuitively
 - 3Blue1Brown: backpropagation, calculus
 - Sebastian Lague: how to create a neural network
- Celsius to Fahrenheit with PyTorch NN
- Building a Regression Model in PyTorch
- Multi-Target Predictions with Multilinear Regression in PyTorch

NEURAL NETWORK (NN). PYTHON'S `PYTORCH`. CRASH COURSE



- I create a NN that finds a damping function, $Y \equiv f_{DNS}$, as a function of input $X \equiv y^+$
- 1 input ($X = a_1^{(0)}$), 1 hidden layer with 2 neurons ($a_1^{(1)}, a_2^{(1)}$) and 1 output ($Y = a_1^{(2)}$)

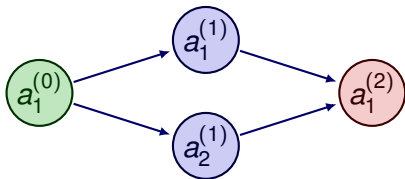
NEURAL NETWORK (NN). PYTHON'S `PYTORCH`. CRASH COURSE



- I create a NN that finds a damping function, $Y \equiv f_{DNS}$, as a function of input $X \equiv y^+$
- 1 input ($X = a_1^{(0)}$), 1 hidden layer with 2 neurons ($a_1^{(1)}, a_2^{(1)}$) and 1 output ($Y = a_1^{(2)}$)

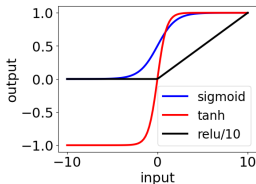
```
class NN(nn.Module):  
    def super__init__(self):  
        self.layer_1=nn.Linear(1, 2) # Connection 0-1  
        self.layer_2=nn.Linear(2, 1) # Connection 1-2  
    def forward(self, x):  
        y = torch.nn.functional.sigmoid(self.layer_1(x)) #  $a_1^{\{1\}}, a_2^{\{1\}}$ , hidden-layer  
        output = torch.nn.functional.sigmoid(self.layer_2(y)) #  $a_1^{\{2\}}$ , output-layer
```

NEURAL NETWORK (NN). PYTHON'S `PYTORCH`. CRASH COURSE

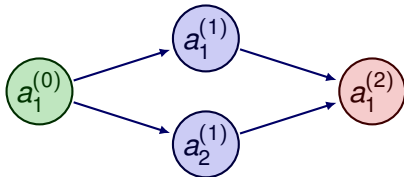


- I create a NN that finds a damping function, $Y \equiv f_{DNS}$, as a function of input $X \equiv y^+$
- 1 input ($X = a_1^{(0)}$), 1 hidden layer with 2 neurons ($a_1^{(1)}, a_2^{(1)}$) and 1 output ($Y = a_1^{(2)}$)

```
class NN(nn.Module):  
    def super__init__(self):  
        self.layer_1=nn.Linear(1, 2) # Connection 0-1  
        self.layer_2=nn.Linear(2, 1) # Connection 1-2  
    def forward(self, x):  
        y = torch.nn.functional.sigmoid(self.layer_1(x)) #  $a_1^{(1)}, a_2^{(1)}$ , hidden-layer  
        output = torch.nn.functional.sigmoid(self.layer_2(y)) #  $a_1^{(2)}$ , output-layer
```



NEURAL NETWORK (NN). FORWARD



Activation 1:
$$a_1^{(1)} = s \left\{ w_1^{(0)} a_1^{(0)} + b_1^{(0)} \right\}$$

Activation 2:
$$a_2^{(1)} = s \left\{ w_2^{(0)} a_1^{(0)} + b_2^{(0)} \right\}$$

Output:
$$a_1^{(2)} = s \left\{ w_1^{(1)} a_1^{(1)} + b_1^{(1)} + w_2^{(1)} a_2^{(1)} + b_2^{(1)} \right\} \equiv Y$$

- s is an activation function (linear, sigmoid, tanh, ...)

NEURAL NETWORK (NN). BACKWARD

The Python code for the simple NN model is given in the listing below

```
# initiate the NN model
model = NN()
# define input, X
X=np.zeros(nj,1))
X[:,0] = scaler_yplus.fit_transform(yplus)[: ,0]
# define output, Y (f is known)
Y = f
# Training loop
for epoch in range(max_no_epoch):
# Compute prediction and loss, L
    o = model(X) #prediction
    L = loss_fn(o, Y) #  $L=|o-Y|_2$ 
    L.backward()
```

- `loss.backward()` computes $dL/dw_1, dL/db_1, dL/ds_1, \dots$
- They are used to get new improved $w_1, b_1, \dots$

THE ASSIGNMENT. APPENDIX Q4 IN THE **eBOOK**

- Download the data Python scripts [here](#).
- The damping functions f_μ and f_2 are used in the AKN model, see Eq. 1. They are functions of y^* and Re_t .
- In this assignment you will create new f_μ and f_2 using NN.
- You will first make them as functions of y^* and y^+ .
- The created NN model will be saved to disk, and then you will load the NN model into the Python CFD code, `RANS0.py` (download it [here](#))
- The Python script `NN-f2-5200-no-batch.py` uses PyTorch to train a f_2 damping function in the NN model on 80% of the DNS data (randomly chosen) and then test (predict) on the remaining 20%.

RANDOM INITIAL WEIGHTS AND BIASES

- Study `NN-f2-5200-no-batch.py` carefully.
- Note that the initial weights and biases are randomized in Pytorch.
- If you want to make sure that you get the same results every time you run the script you can save the initial weights and biases, i.e.

```
        if epoch == 0 and batch == 0:
# Define checkpoint
    checkpoint = {
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': loss,
    }
    torch.save(checkpoint, 'checkpoint-f2.ct')
```

from one run that converged well.

- Then you load the initial data in a subsequent run (see next slide)

RANDOM INITIAL WEIGHTS AND BIASES

```
for epoch in range(N_epochs):
    if epoch == 0:
        print('checkpoint loaded')
        checkpoint = torch.load('checkpoint-f2.ct', weights_only=True)

    # Apply the state_dict to model and optimizer
    NN = MyNet()
    NN.load_state_dict(checkpoint['model_state_dict'])

    optimizer = torch.optim.SGD(NN.parameters(), lr=learning_rate)
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])

    # Retrieve the training epoch
    #     epoch = checkpoint['epoch']
    #     loss = checkpoint['loss']

    NN.train()  # For training mode (resuming training)
```

INPUT DATA

- The input data, y^* and y^+ , are taken from DNS and they are scaled and then assigned to X . i.e.

```
# we choose two inputs: yplus_DNS and ystar_DNS
# re-shape
yplus= y_plus_DNS.reshape(-1,1)
ystar= y_star_DNS.reshape(-1,1)
# use scaling, One for each input
scaler_yplus = MinMaxScaler()
scaler_ystar = MinMaxScaler()
# X = input matrix
X=np.zeros((len(yplus_DNS),2))
X[:,0] = scaler_yplus.fit_transform(yplus)[:,0]
X[:,1] = scaler_ystar.fit_transform(ystar)[:,0]
```

- The NN model will try to find suitable weights and biases to fit the input parameters to the output (the **target**), which is f_2 (also taken from DNS). It is assigned to Y (next slides)

THE MODEL

```
# current model of f_2
f_2=((1.-np.exp(-y_star_DNS/3.1))**2)*(1.-0.3*np.exp(-(re_t/6.5)**2))

# output is f_2_NN (see below) predicted by the NN. Our target is f_2
# transpose the target vector to make it a column vector
Y = f_2.transpose()
Y= Y.reshape(-1,1)  # makes an array of size [len(Y), 1]
```

The NN model is using 10 neurons and three hidden layers (instead of two and one, respectively, in the example above) and the ReLu actuator is used (instead of Sigmoid above)

```
class MyNet(nn.Module):

    def __init__(self):
        super().__init__()
        self.input      = nn.Linear(2, 10)  #axis 0: number of inputs
        self.hidden1     = nn.Linear(10, 10)
        self.hidden2     = nn.Linear(10, 1)  #axis 1: number of outputs

    def forward(self, x):
        x = nn.functional.relu(self.input(x))
        x = nn.functional.relu(self.hidden1(x))
        x = self.hidden2(x)
```

At the end of `NN-f2-5200-no-batch.py` the NN model is saved to disk

```
# save NN model to disk
filename = 'model-neural-k-omega-f_2.pth'
torch.save(NN, filename)
dump(scaler_yplus, 'scaler-yplus-k-omega-f_2.bin')
dump(scaler_ystar, 'scaler-ystar-k-omega-f_2.bin')

yplus_min = np.min(yplus_train)
ystar_min = np.min(ystar_train)
f_2_min = np.min(f_2_train)

yplus_max = np.max(yplus_train)
ystar_max = np.max(ystar_train)
f_2_max = np.max(f_2_train)

np.savetxt('min-max-model-f_2.txt', [yplus_min, yplus_max, ystar_min, ystar_max, f_2_min, f_2_max])
```

- The NN model is used `NN_bool = True`; if not, the standard AKN $k - \varepsilon$ model is used
- I define a 1D grid. `yc`, from the wall ($y = 0$) to the centerline ($y = 1$). The location of the cell centers are stored in `yp`.
- `if NN_bool = True`, The NN model `NN-f2-5200-no-batch.py` is loaded.
- The friction Reynolds number, $Re_\tau \equiv u_\tau \delta / \nu = 5\,200$, where δ is half-channel width. u_τ and δ are set to one and hence $\nu = 1/Re_\tau$.
- We're computing fully-developed channel flow (i.e. $\partial \bar{v}_1 / \partial x_1 = 0$) and the driving pressure gradient, $d\bar{p}/dx_1 = 1$ is prescribed. Since the half-channel width is one and $\rho = 1$, we get $\tau_w \equiv \rho u_\tau^2 = 1$ (see Eq. 3.30 in the [eBook](#)). Hence $u_\tau = 1$.
- First, the $\bar{v}_1$ equation is solved.

- Next, the k and ε equations are solved.
- You may note that the negative source terms in both equations are included in S_P where the total source is $S = S_U + S_P\Phi$, see p. 16 in my [CFD lecture notes](#). With this procedure, there is no way that k and ε can go negative since all terms in the discretized equation (see Eq. 13 on p, 17) then are positive.
- Recall that $a_P = a_W + a_E - S_P$.
- Note that there are no computational nodes at the boundaries. The b.c. are S_P and S_U , see Slide 10 in [The Finite Volume Method: A Crash Course](#).

GETTING STARTED

- Run the `RANS0.py` script and create the NN model. Then run the CFD code, both using the standard AKN $k - \varepsilon$ model, i.e. `NN_bool = False` and using the NN model, i.e. `NN_bool = True`
- In `NN-f2-5200-no-batch.py` I use y^+ and y^* as input parameters. For example, use only one of them. Or maybe y^* and R_t , as in the original model. Or P^k , or You want your NN model to work also in other flows and/or at other Reynolds numbers. Hence, your input parameters must be non-dimensional. The DNS data are non-dimensionalized with u_τ and ν (i.e. you can regard $u_\tau = \nu = 1$)
- Do one of many or the subtasks below
 - Use your NN model at other Reynolds numbers. You find DNS data for $Re_\tau = 550$, $Re_\tau = 2000$ and $Re_\tau = 10\,000$, [here](#). Note that when you increase the Reynolds number in `RANS0.py` you may need to increase the number of cells (i.e. increasing `nj`) so that $y^+ < 1$ for the wall-adjacent cells.
 - Include another NN model for f_μ
 - Train your NN model(s) at $Re_\tau = 10\,000$ and use it in the CFD code for lower Reynolds numbers.

REFERENCES

- [1] K. Abe, T. Kondoh, and Y. Nagano. A new turbulence model for predicting fluid flow and heat transfer in separating and reattaching flows - 1. Flow field calculations. *Int. J. Heat Mass Transfer*, 37(1):139–151, 1994.
- [2] L. Davidson. Using Physical Informed Neural Network (PINN) and Neural Network (NN) to improve a $k - \omega$ turbulence model: Python CFD code and PINN script. Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2025. URL <https://www.cfd-sweden.se/lada/Using-Physical-Informed-Neural-Network-PINN-and-NN-improve-a-k-omega-turbulence-model-Python-CFD-code-and-PINN-script-PDF.pdf>.
- [3] L. Davidson. Understanding autograd and neural network in pytorch. Division of Fluid Dynamics, Dept. of Mechanic Engineering, Chalmers University of Technology, Gothenburg, 2026. URL https://www.cfd-sweden.se/lada/postscript_files/understanding-autograd-and-neural-network-in-pytorch.pdf.

REFERENCES

- [4] Lars Davidson. Using physics informed neural network (PINN) and neural network (NN) to improve a k-omega turbulence model. *Journal of Turbulence*, pages 1–22, 2026. doi: <https://doi.org/10.1080/14685248.2026.2665148>. URL <https://doi.org/10.1080/14685248.2026.2665148>.
- [5] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas Schön. *Machine Learning: A First Course for Engineers and Scientists*. Cambridge University Press, 2022. ISBN 9781108843607. doi: 10.1017/9781108919371. URL <http://smlbook.org/>.
- [6] Menneni Rachana, Jegadeesan Ramalingam, Gajula Ramana, Adigoppula Tejaswi, Sagar Mamidala, and G Srikanth. Fraud detection of credit card using machine learning. *GIS-Zeitschrift für Geoinformatik*, 8:1421–1436, 10 2021.

REFERENCES

- [7] Sudarshana S Rao and Santosh R Desai. Machine learning based traffic light detection and ir sensor based proximity sensing for autonomous cars. In *Proceedings of the International Conference on IoT Based Control Networks & Intelligent Systems – ICICNIS*, 2021. URL <http://dx.doi.org/10.2139/ssrn.3883931>.