Multitask learning: teach your AI more to make it better

Alexandr Honchar Nov 30, 2018 · 17 min read ★

https://bitesizebio.com/27766/multitasking-lab-not-multitasking/

Hi everyone! Today I want to tell you about the topic in machine learning that is, on one hand, very research oriented and supposed to bring machine learning algorithms to more human-like reasoning and, on the other hand, is very well known to us from basics of machine learning, but very rarely interpreted as I want to show it today. It's called **multitask learning** and it has (almost) nothing to do with multitasking as on the picture above. In this post, I will show what is multitask learning for humans and algorithms, how researchers today already apply this concept, how you can use it for

from the photos, movement identification with accelerometer data, siamese network boosting and solving Numerai challenge: all using multitask learning!), that you can use as a template (and, I hope, inspiration) for your own projects!

This post is highly inspired by Sebastian Ruder and Rich Caruana materials.

If you want to jump directly to the code in Keras, here is the corresponding GitHub repo. I was also giving a talk on this topic at Data Science UA conference, slides are about to be uploaded too.

Multitasking for us

The term "multitasking" itself usually means something negative — a person who tries to do a lot of things and doesn't succeed in anything of that is the best example. But here I want to talk about a bit different "multitasking". I would like to imagine, that all the functions that our biological brain does are some kind of deep neural network that is also trained with backpropagation and gives some labels as output. If I think about my **vision system** like that, what it outputs when I look at something? I can not just identify objects, segment them and understand the scene. I can also estimate the moods of people on the street, how they're dressed, the weather based on their dresses, I can estimate what country or even city it is and a lot of other significant details that make my understanding from the visual channel really very rich.

Okay, what about **listening**? If I hear anyone, I immediately understand the sex of a person, tonality, I understand words, whole sentence and meaning, I know the language, I understand all the entities and even am able to answer to what I heard.

I can do the same exercise with any of my sensory systems — I always feel a lot of things from a single "testing" example, whatever it is — a picture, a sound, a text or even a feeling.

Multitasking for machines

On the other hand, modern neural networks (and other machine learning algorithms) usually solve a single problem from a single example — it can be classification, regression, structured prediction, anomaly detection or even object generation, but usually, we expect a single thing at the end. In this blog, I want to talk and show how

Well-known L2 regularization for a loss function

You're right, we all know this famous L2 regularization formula. Why do I say it's an example of multitask learning? Because we solve two optimization problems at the same time: minimization of loss function and making norm of our parameters *w* smaller. We usually don't think about it very deeply, but there is a lot of theory behind choosing this particular second optimization function:

Different explanations of a regularization effect

For example, **Andrey Tikhonov** was mainly interested in solving inverse ill-posed problems and adding regularization by the norm of parameters helped him to achieve smoothness and good conditioning of the problem. Polish mathematician **Zaremba** introduced it for statistical learning theory as the complexity of the model which is very desirable to have small — hence, we minimize it. From **Bayes'** point of view this way we condition our weights to have some certain distribution — these are our priors on the parameters. And, of course, most of us studied it on some course where we didn't really worry about it, but we knew that it prevents overfitting of our models :)

brain) I'd like to interpret it more specifically from a **reasoning** point of view. Can I add some "regularization" that takes into account the "real", "human-like" context of a problem? If I want to recognize emotions from a photo, should I restrict weights to "know" about the movements of the nose and lips? Most probably yes. If I want to translate a text, can be helpful for my parameters to know also parts of speech of given words? I think it will be. If I am working on signal processing problem, do I want my model to be able to have some intuition about the past and the future of the given time series? It might be very helpful!

http://ruder.io/multi-task/

All this "knowing", "conditioning" I can incorporate into the machine learning algorithm as an additional input, but since I started to talk about multitask learning here… you understand now where I am leading. If such additional loss as regularization of the algorithm parameters helps better performance, why shouldn't we try **additional losses** that supposed to help performance from the human logic point of view? Then our algorithms will look something like the picture above: a single input, several layers (in case of neural networks) and numerous outputs, that can play a role of regularizer with some hyperparameter *lambda*, or an independent objective that also wants to be optimized and solved.

separate neural networks is simply much cooler from a plain engineering point of view. Better performance, regularization, and generalization comes as a bonus.

Motivational use cases

Before starting to dive into the details I'd like to show some motivational examples. First one is amazing research in computer vision, where they did not just show how a single neural network can solve more than 20 tasks simultaneously, but also automatically had built a hierarchy (that's why it's called Taskonomy) over different tasks in 2D and 3D space and showed a boost in performance compared to transfer learning of one task to another one. This work is an amazing inspiration on how to bring a complex understanding of a single image into a single algorithm.

Taskonomy | Stanford

Taskonomy: Disentangling Task Transfer Learning, CVPR 2018 (Best Paper). Stanford, UC Berkeley. We propose a fully...

taskonomy.stanford.edu

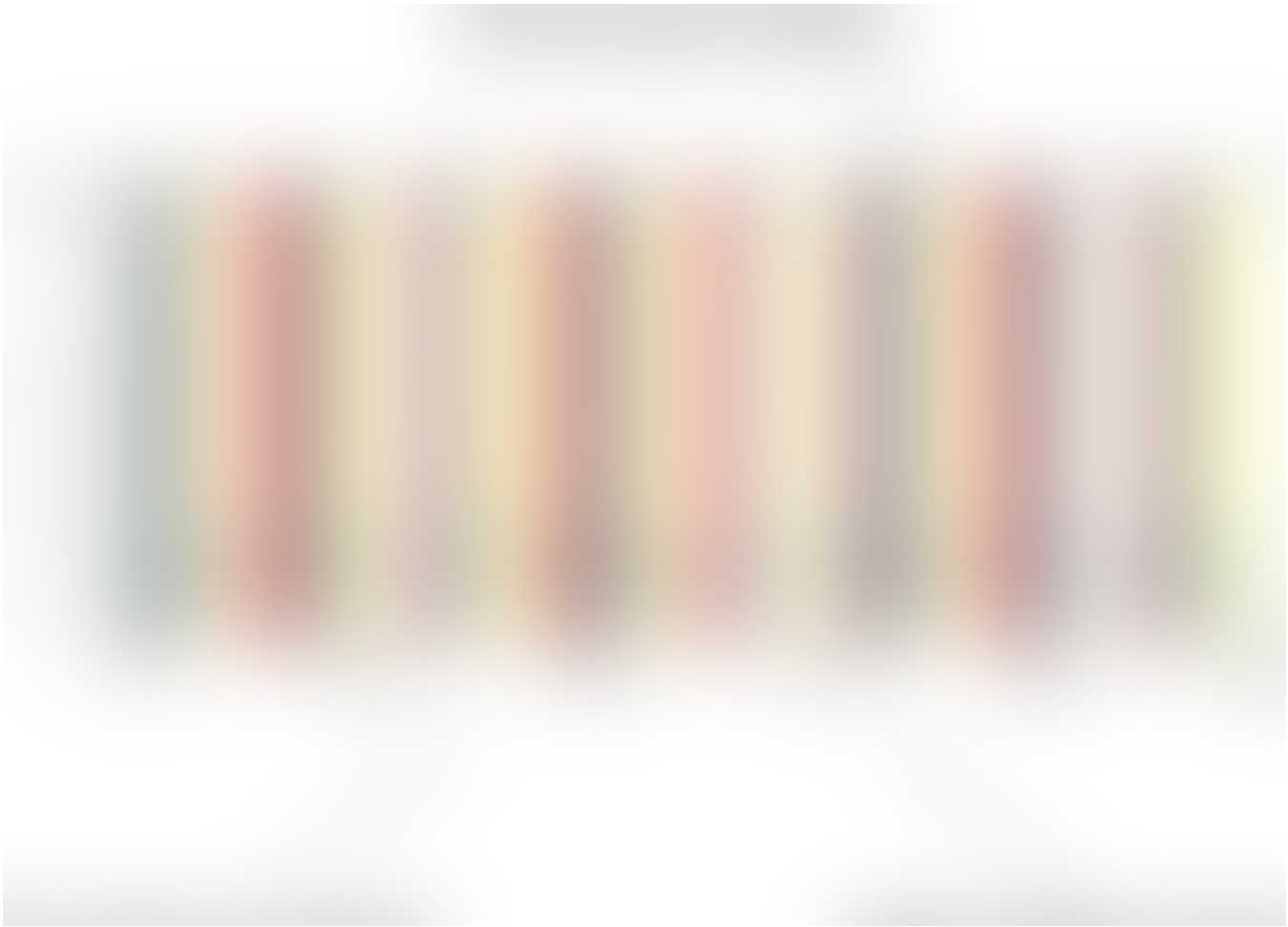
A second great example comes in Natural Language Processing: why don't we solve sentiment analysis, semantic parsing, translation, answering a question, translation from a single sentence using a single neural network? Researchers from Salesforce thought so too and developed their own creative solution.

decaNLP

Deep learning has significantly improved state-of-the-art performance for natural language processing (NLP) tasks, but...

decanlp.com

In Mawi Solutions where I am responsible for machine learning, we also apply a concept of multitask learning for live cardiogram analysis. During measuring an ECG we need to tag its morphological structure for further analysis, estimate the quality of the signal and solve anomaly detection task same time every second. We could run three different algorithms (not necessarily machine learning ones) same time on a mobile phone, but we succeed to solve them all with a single neural network that is



Multitask learning for live ECG analysis in Mawi Solutions

Multitask learning is also very important for reinforcement learning as a crucial concept of “human-like” behaviors. One of the good examples can be DeepMind’s work Distral: Instead of sharing parameters between the different losses, we propose to share a “distilled” policy that captures common behavior across tasks.

Distral: Robust Multitask Reinforcement Learning

different, even maybe close tasks can help to learn the main task or why they can learn jointly together?

1. **Regularization:** additional loss is in principle does literally the same what different researchers wanted from regularization: it smoothes the loss function, minimizes the complexity of your model and gives very informative priors to your model.

2. **Representation bias:** when you train your model over an intersection of several tasks you push your learning algorithm to find a solution on a smaller area of representations on the intersection rather on a large area of a single task. It leads to faster and better convergence.

3. **Feature selection double check:** if one feature is important for more than just one task, then most probably this feature is indeed very important and representative for your data

4. **Transfer learning:** machine learning already benefits from learning several tasks, but sequentially using transfer learning: pretraining a model on a large dataset on one task and after using the obtained model to train another task over it. Empirical results show that it's a bit more beneficial and definitely faster to learn these tasks jointly.

5. **Outputs that can't be inputs:** this moment is the most interesting personally for me. There are particular types of data that by their sequential nature are being conditioned by the past observations and usually, we need to deal with them: words in a sentence (that have to predict next word, frames and a video, that have to predict next frame and so on). We would like a representation from these past moments to be able to model future ones — and here where an auxiliary task arises so naturally — past frames have to predict next frame(s), that we *can't use as inputs*! Other nice examples are features, that are too difficult or long to calculate in the testing/production phase, but we can use them as auxiliary loss functions to regularize our model for "knowing" about them.

Predicting next frame in a video cant be a very informative prior: https://coxlab.github.io/prednet/

Auxiliary tasks for everyone

Alright, maybe now you're more or less convinced that multitask learning is cool, works for researchers and has some theoretical and empirical motivation behind it. But you might answer me, that you don't really need it in your projects, because you don't need to solve several tasks at the same time, or you simply don't have some additional labels for your dataset or there is no possibility to get them. In this section, I want to show some examples of auxiliary losses that *every one* of you can use in *any* project as an additional regularizer that will boost your performance.

1. **Natural language processing**
   - language modeling: a core of NLP is n-gram modeling: the probability of a word given last *n* words. You can predict next word in your sentence as a prior knowledge you want your model to have.
   - NLTK: part of speech, sentence tree, NER — you sure your deep learning won't benefit from them? Try basic NLP features from NLTK or spacy as priors.
   - DecaNLP: clear inspiration :)

2. **Computer vision**
   - OpenCV: why don't give your object tracking model some priors like HOG feature or some "basic" thresholding segmentation for semantic segmentation? You have them all in OpenCV, scikit-image or dlib.
   - Taskonomy: clear inspiration :)

3. **Signal processing**
   - Statistics/feature modeling: good idea is to use statistical or other features from your given time series, for example, it can be anything from standard deviation, skewness, and kurtosis to sample entropy, AR or Fourier coefficients

4. **General recommendations**
   - Autoencoder / generative modeling as models push any representation to be as most useful and descriptive as possible
   - Hints (predicting important features, like HOG in computer vision, NER in NLP and entropy in DSP)

A note on reproducible research

Before showing how to implement multitask lerning, I want to show what I had to do to make my code in Keras at least to some point reproducible on another computers (but still it wasn't enough as showed my workshop in Kiev). First, I had to fix all random seeds:

```
random.seed(42)
np.random.seed(42)
tf.set_random_seed(42)
```

fix initializers for neural network layers and run a session:

```
my_init = initializers.glorot_uniform(seed=42)

session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
                              inter_op_parallelism_threads=1)
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

and fix random seed while creating dataset:

```
train_test_split(X, Y1, Y2, Y3, random_state = 42)
```

and even set "shuffle = False" while training a model. Moreover, I fixed all hyperparameters of the models and training conditions as the number of epochs, optimizer, and batch size. **But it wasn't enough.** Still on some computers (I ran my code on MacBook Air) results very slightly different from mine. I can explain it (thanks to hints from the auditory) only like this: different hardware led to different float

random seeds before every training process) and will be very thankful for any hint.

Practice 1: emotions recognition

As I promised, this article will have several code examples, so you can try to do multitask learning by yourself (mainly concentrated on using additional tasks as regularizers). The first example is related to computer vision and, in particular, to the dataset (which is not freely distributed, but you can ask access for research purposes):

The Affect Analysis Group at Pittsburgh

Edit description

www.pitt.edu

And my full code is here:

Rachnog/education

Materials for my lectures and for self-study in machine learning - Rachnog/education

github.com

This dataset consists of images of different people and corresponding labels:

- Emotions their faces express
- Facial action units (later FAU, see more info here)
- Facial key points

We will use emotion recognition with a pretty simple neural network (comparing to the state of the art ones) as the main task and will try to adapt FAU and key points detection as regularizing ones. This is a neural network we will try to augment with multitask learning. As you can see, in 2018 deep learning era we can easily call it “shallow”:

```
kernel_initializer=my_init)(visible)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(20, kernel_size=3, activation='elu', padding='same',
kernel_initializer=my_init)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(30, kernel_size=3, activation='elu', padding='same',
kernel_initializer=my_init)(x)
x = GlobalMaxPooling2D()(x)
output = Dense(Y_train.shape[1], activation='softmax',
kernel_initializer=my_init)(x)
model = Model(inputs=visible, outputs=output)
```

It shows the following result (which is not satisfying of course):

	precision	recall	f1-score	support
0	0.39	0.56	0.46	149
1	0.00	0.00	0.00	10
2	0.00	0.00	0.00	71
3	0.32	0.11	0.16	74
4	0.47	0.64	0.54	163
5	0.44	0.11	0.17	64
6	0.38	0.54	0.44	130
avg / total	0.36	0.41	0.36	661

Next, I added a second output for a model that had to predict **facial action units** alongside emotions:

```
output2 = Dense(Y_train2.shape[1], activation='sigmoid', name =
'facs', kernel_initializer=my_init)(x)
```

and the results became slightly better!

	precision	recall	f1-score	support
0	0.58	0.36	0.44	149
1	0.00	0.00	0.00	10
2	0.47	0.27	0.34	71
3	0.36	0.65	0.46	74
4	0.56	0.83	0.67	163
5	0.56	0.08	0.14	64

Okay, if this auxiliary loss helps, maybe predicting **key points** can be good too? At least it performs better than a baseline as well:

```
precision    recall  f1-score   support

          0       0.69      0.34      0.45       149
          1       0.00      0.00      0.00        10
          2       0.71      0.07      0.13        71
          3       0.38      0.65      0.48        74
          4       0.52      0.88      0.65       163
          5       0.50      0.05      0.09        64
          6       0.53      0.72      0.61       130

avg / total       0.56      0.52      0.46       661
```

Okay, seems like now I have two auxiliary losses that both regularize my baseline solution pretty good, so I would like to combine them and I expect to have the even better result. I will do this in a “smart” way. I know, that when I (as a human) decompose my own emotion recognition, I first see the key points, after I combine them into action units and only, in the end, I understand emotions from this hierarchy of features. In terms of neural network, it can mean, that I should attach an FAU loss to the first convolutional layer, a loss for key points to the second and leave the main loss in the end:

```
visible = Input(shape=input_shape)
x1 = Conv2D(10, kernel_size=3, activation='elu', padding='same',
kernel_initializer=my_init)(visible)
x1 = MaxPooling2D(pool_size=(2, 2))(x1)
x2 = Conv2D(20, kernel_size=3, activation='elu', padding='same',
kernel_initializer=my_init)(x1)
x2 = MaxPooling2D(pool_size=(2, 2))(x2)
x3 = Conv2D(30, kernel_size=3, activation='elu', padding='same',
kernel_initializer=my_init)(x2)
x = GlobalMaxPooling2D()(x3)

output = Dense(Y_train.shape[1], activation='softmax', name =
'emotion', kernel_initializer=my_init)(x)
output2 = Dense(Y_train2.shape[1], activation='sigmoid',
kernel_initializer=my_init)(GlobalMaxPooling2D()(x2))
output3 = Dense(Y_train3.shape[1], activation='linear',
kernel_initializer=my_init)(GlobalMaxPooling2D()(x1))
```

To avoid “killing” the gradients from the main loss function on lower layers I add *lambdas* for my losses 0.1 for the second and first layer:

```
model.compile(loss=['categorical_crossentropy',
'binary_crossentropy', 'mse'],
              optimizer=Adam(clipnorm = 1.), metrics = {'emotion':
'accuracy'}, loss_weights = [1, 1e-1, 1e-1])
```

We train for the same 10 epochs… and… voila!

```
precision    recall  f1-score   support

          0       0.75      0.56      0.64       149
          1       0.00      0.00      0.00        10
          2       0.79      0.27      0.40        71
          3       0.45      0.80      0.58        74
          4       0.67      0.88      0.76       163
          5       0.64      0.14      0.23        64
          6       0.61      0.78      0.68       130

avg / total       0.65      0.63      0.60       661
```

We could improve our result **from 0.36 F1-score to 0.60 F1-score** just with using additional loss functions in a smart way. Same simple convnet, same initializers, same data, same training process. Looks like magic? Let’s check the next example then.

Practice 2: accelerometer movement analysis

You can tell me, that usually, you don’t have such nice additional features as facial action units to use alongside main loss. It’s alright, now I will show you an example of a task, where you can create these losses by yourself. This dataset is about classifying activity from a wrist-worn accelerometer:

https://archive.ics.uci.edu/ml/datasets/Dataset+for+ADL+Recognition+with+Wrist-worn+Accelerometer

This is my full code for this problem:

Rachnog/education

github.com

My input is multivariate time series of three axes of an accelerometer and an output is one of 21 activity classes. No additional losses attached. What should I do now? Of course, create some! I've chosen the losses that look logical to me:

- the standard deviation of given time series
- the sum of absolute changes of time series
- Fourier coefficients.

I have a hypothesis that if my neural network will know about one of these features, it will perform better than without. I check the baseline solution with a simple 1D convolutional neural network first:

```
inn = Input(shape = (X_train.shape[1], X_train.shape[2], ))

x = Conv1D(16, 5, activation='relu', kernel_initializer=my_init)
(inn)
x = MaxPooling1D(5)(x)
x = Conv1D(32, 5, activation='relu', kernel_initializer=my_init)(x)
x = MaxPooling1D(5)(x)
x = Conv1D(64, 5, activation='relu', kernel_initializer=my_init)(x)
x = GlobalMaxPooling1D()(x)
out = Dense(Y_test.shape[1], activation = 'softmax',
kernel_initializer=my_init)(x)

model = Model(inputs=[inn], outputs=[out])

model.compile(loss='categorical_crossentropy',
              metrics = ['accuracy'], optimizer = Adam(clipnorm =
1.))
```

and get following precision-recall-F1:

```
avg / total       0.61      0.57      0.54       245
```

What about adding **standard deviation loss**?

and with a **sum of changes**?

```
avg / total       0.57      0.60      0.57       245
```

okay, they both don't help that much, but **Fourier** must be a powerful shot…

```
avg / total       0.60      0.62      0.59       245
```

Seems like that we could improve the baseline **just by 5%**, but this is a good result as well! We often fight for exactly these five percent using different tricky approaches, so now you about one more :)

Practice 3: boosting siamese networks

Siamese neural networks are a special class of models that aim to compare objects in a new embedding space: similar objects have to be close to each other (have small distance) and different — far away (have large distance). It's not really smart idea to calculate a Euclidean distance for two 64x64 images (in case of MNIST dataset), but we can embed these images into some vector space, where it will make sense. The main specialty of this family of models is having two inputs and layers' weights that are being shared between these inputs to allow commutativity (as we basically calculate a distance between two objects). In this example, I will show how adding an additional loss to each of these "sleeves" of layers will improve performance on MNIST dataset. Full code is here :

Rachnog/education

Materials for my lectures and for self-study in machine learning - Rachnog/education

github.com

First, we have an encoder for each input looking like the following:

```
x = Dense(64, activation='relu')(x)
x = Dropout(0.1)(x)
m = Model(input, x)
```

as you can see, we embed input images into some 64-dimensional space, where we can compare them and the loss is designed to minimize the distance between images of the same number and maximize of different ones:

```
def contrastive_loss(y_true, y_pred):
    margin = 1
    sqaure_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(margin - y_pred, 0))
    return K.mean(y_true * sqaure_pred + (1 - y_true) *
margin_square)
```

I have a hypothesis, that in this 64-dimensional embedding space I can not only compare images but also classify them. **So my second loss will be simple classification into 10 digits classes.** Training a baseline siamese network and testing it as a classifier shows the following result:

```
* Accuracy on training set: 85.47%
* Accuracy on test set: 85.61%
```

and with an auxiliary loss:

```
* Accuracy on training set: 88.49%
* Accuracy on test set: 88.54%
```

That's it. :)

Practice 4: financial forecasting at Numerai

I think this blog is already too long, but it's too tempting not to add the last example related to financial forecasting. Numerai is a data science challenge on encrypted financial data, where you have everything prepared and wrapped for you, and all that's left — to predict 0 or 1 for **five separate variables** and not overfit to the train data.

the whole code here:

In this case, my additional losses are not auxiliary, but they all are important. So, I build my neural network like the following:

```
inputs = Input(shape=(50,))
c = Dense(100, activation='relu')(inputs)
c = Dropout(0.1)(c)

predictions_target_bernie = Dense(1, activation='sigmoid', name =
'target_bernie')(c)
predictions_target_charles = Dense(1, activation='sigmoid', name =
'target_charles')(c)
predictions_target_elizabeth = Dense(1, activation='sigmoid', name =
'target_elizabeth')(c)
predictions_target_jordan = Dense(1, activation='sigmoid', name =
'target_jordan')(c)
predictions_target_ken = Dense(1, activation='sigmoid', name =
'target_ken')(c)
```

with equal *lambdas*:

```
model.compile(loss='binary_crossentropy',
optimizer=Adam(lr = 0.0001),
loss_weights = [0.2, 0.2, 0.2, 0.2, 0.2])
```

This model allowed to beat the benchmark **on live financial data 9 times out 15 and still counting**. I think it's not a bad result and I am happy to share it with you guys.

9/15 on Numerai with multitask learning

Conclusions

I understand that all this material might be a bit overwhelming, but don't worry about it. All I tried to show can be compactly squeezed into three main conclusions:

1. Multitask learning is **normal for humans** (but maybe not multitasking haha)
2. Multitask learning is **normal for machine learning**, moreover, we already do it for decades
3. Next time you see you're overfitting or not getting the most from your machine learning model, it's time to think — maybe you don't need more data, but **you need more losses?**

Moreover, we reviewed some practical use cases concerning multitask learning and learned how to code it in Keras, how to use auxiliary losses and select weights for them (at least in some cases). I hope this blog post was useful for you and you'll use the concept of multitask learning to get better results in your current project!

P.S.
Follow me also on Facebook for AI articles that are too short for Medium, Instagram for personal stuff and Linkedin!