ML for physical and natural scientists 2023 10

Generative AI: autoencoders

dr.federica bianco | *fbb.space* | *fedhere* | *fedhere*

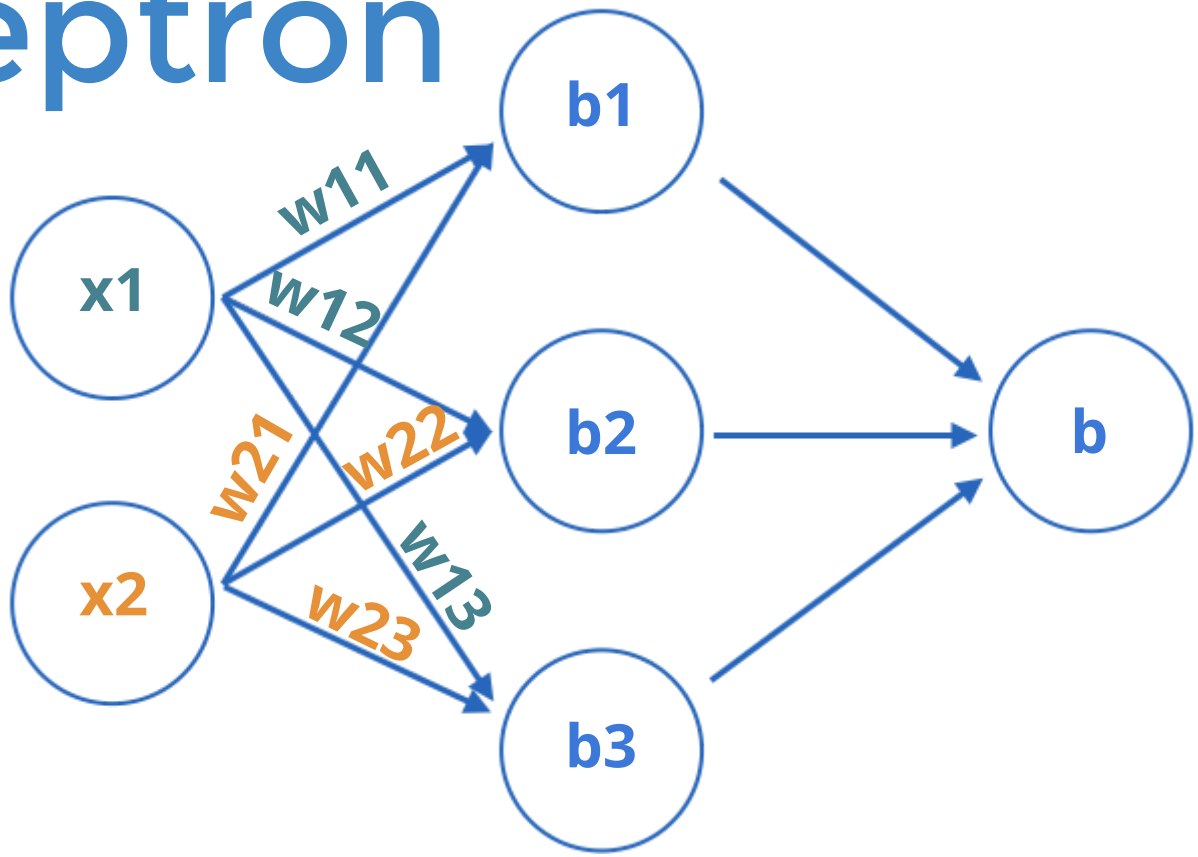
this slide deck:

https://slides.com/federicabianco/mlpns23_10

recap

Deep Learning

multilayer perceptron



w: weight

sets the sensitivity of a neuron

b: bias:

up-down weights a neuron

$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1...W_N + b_N)))$$

multilayer perceptron

f: activation function:

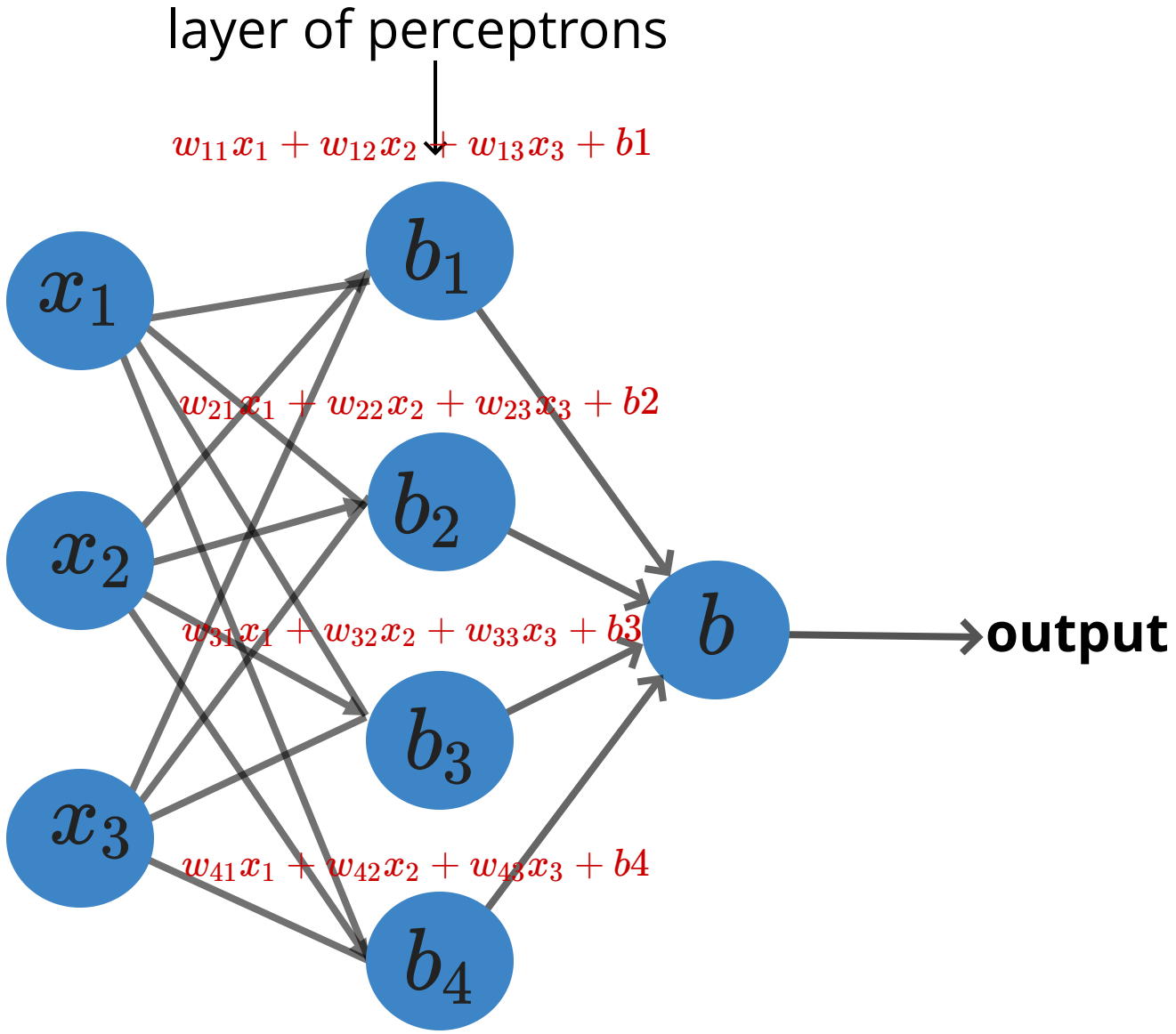
turns neurons on-off

w: weight

sets the sensitivity of a neuron

b: bias:

up-down weights a neuron



multilayer perceptron

f: activation function:

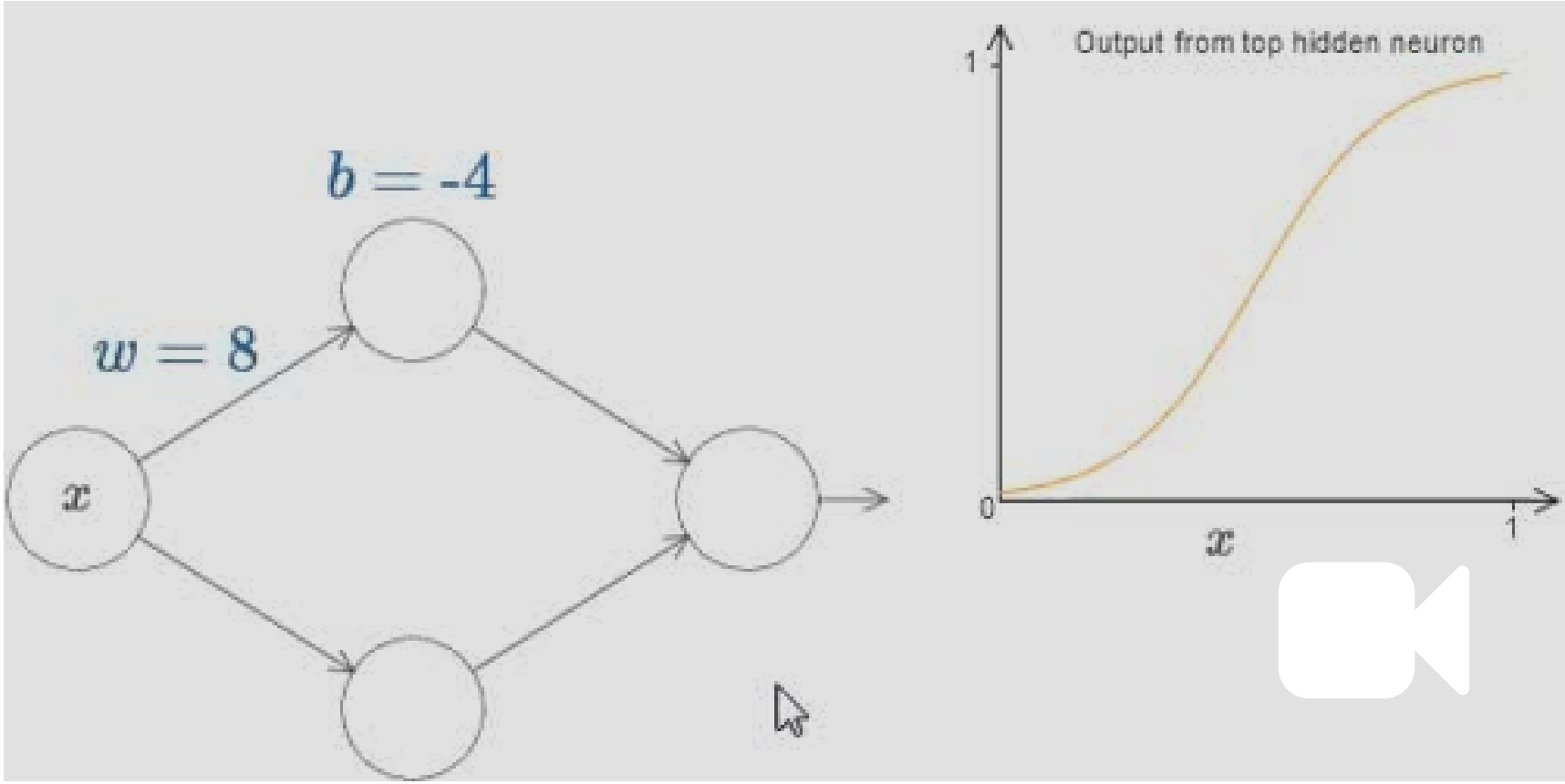
turns neurons on-off

w: weight

sets the sensitivity of a neuron

b: bias:

up-down weights a neuron

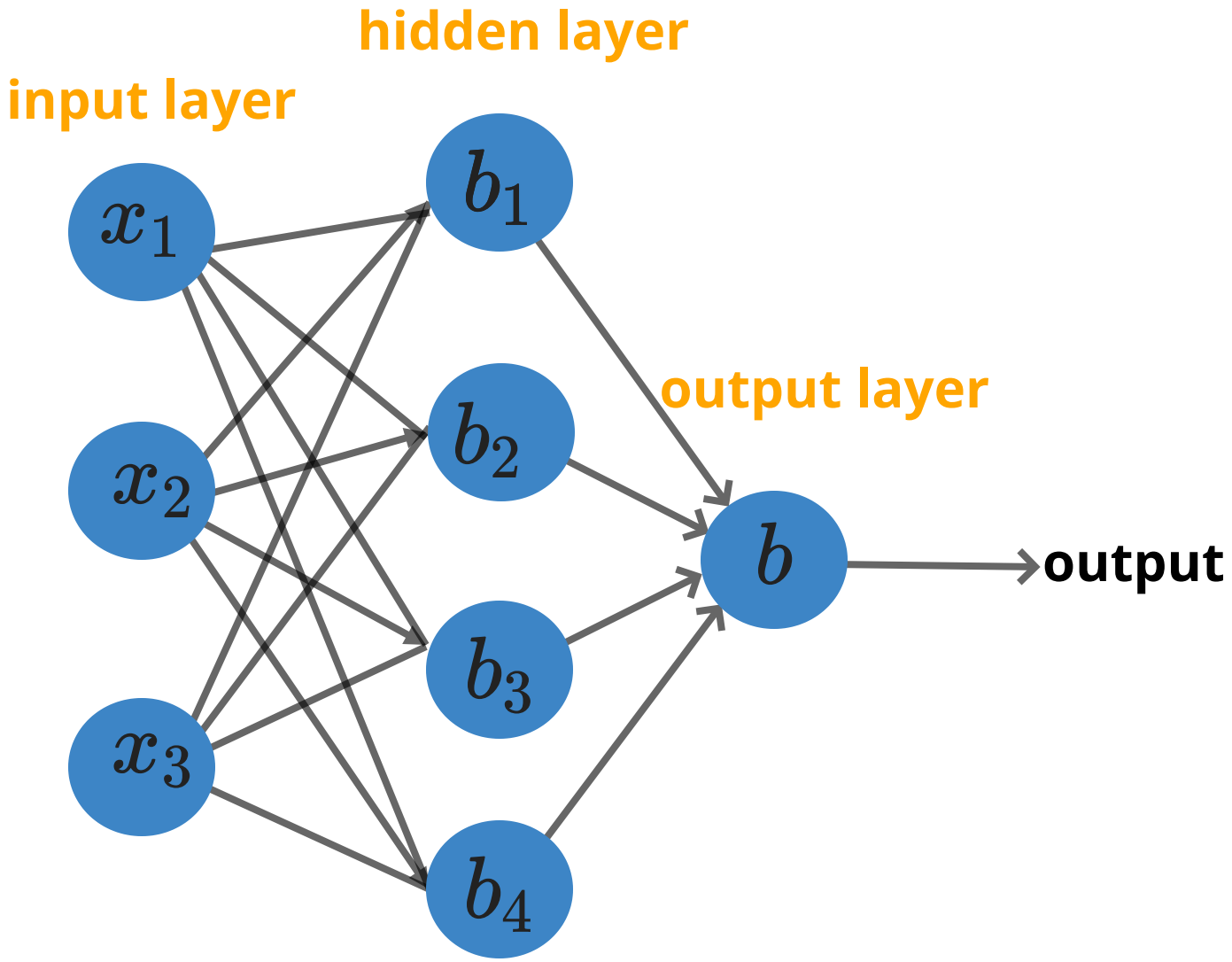


http://neuralnetworksanddeeplearning.com/chap4.html

$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1...W_N + b_N)))$$

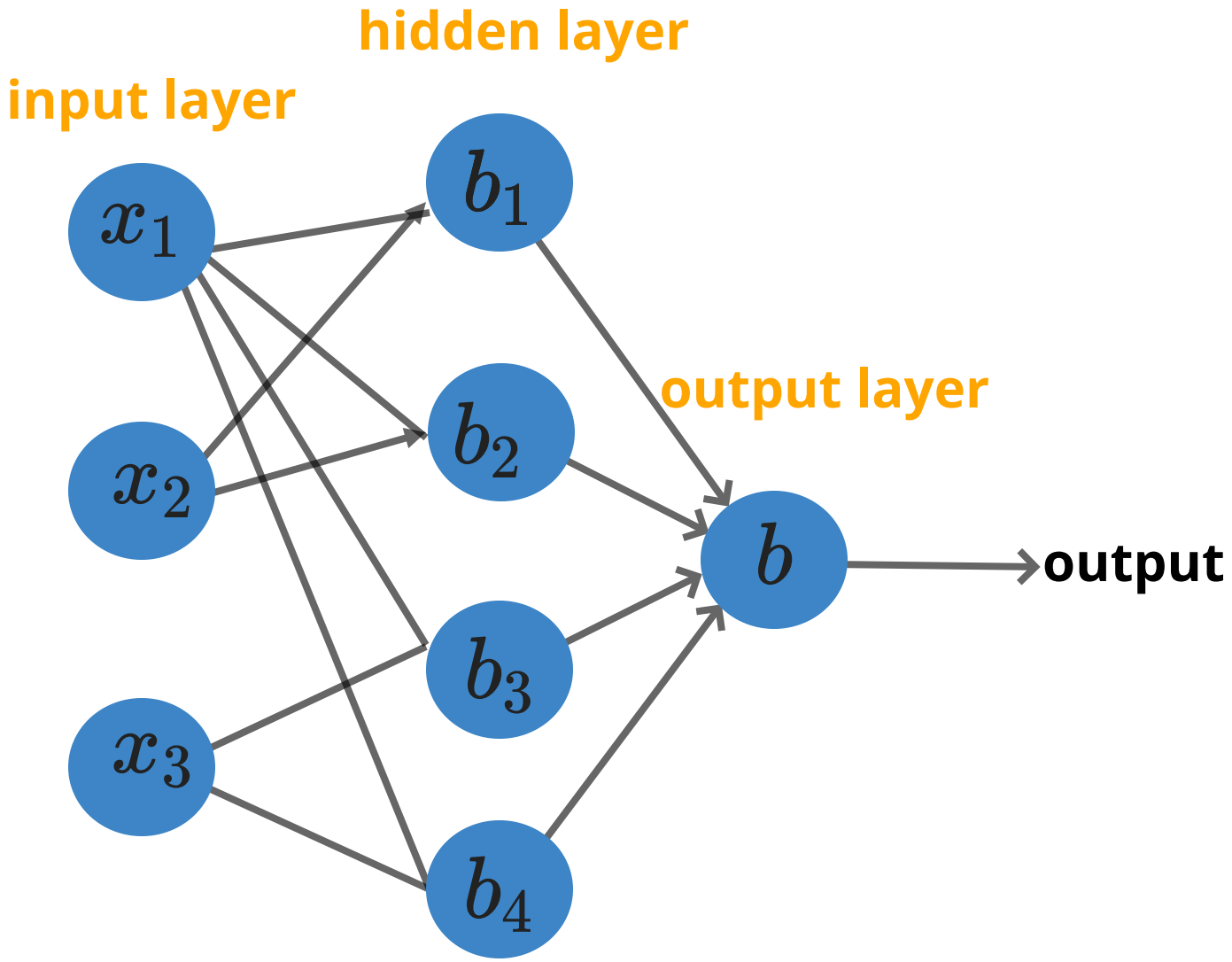
layer connectivity

Fully connected: all nodes go to all nodes of the next layer.



layer connectivity

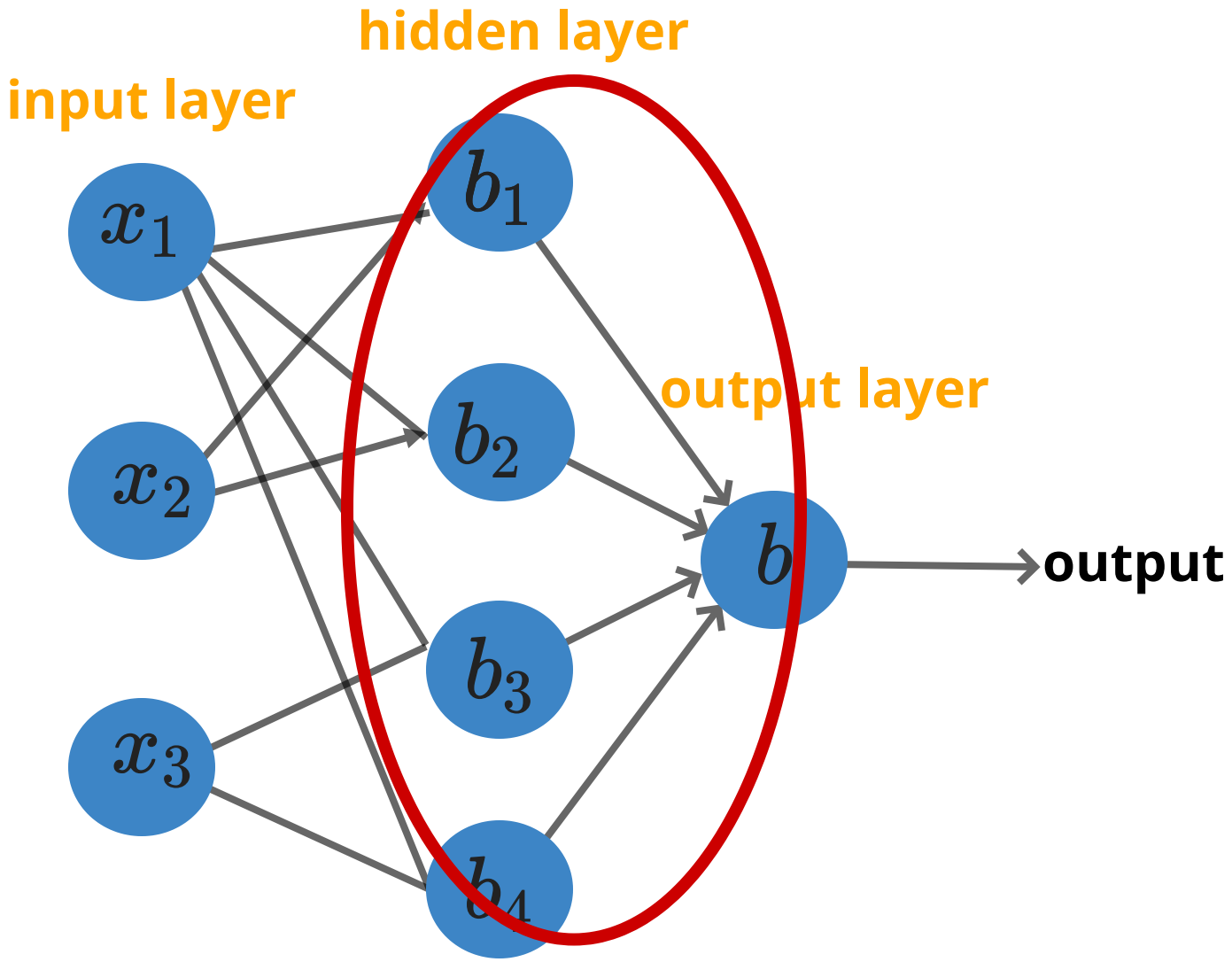
Sparcely connected: all nodes go to all nodes of the next layer.



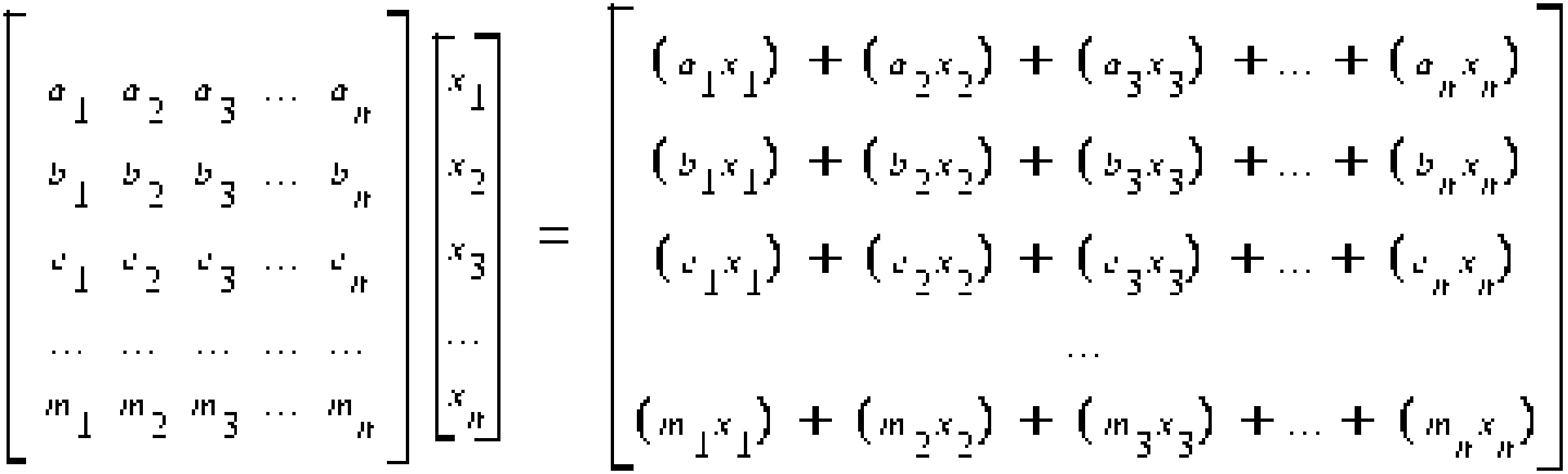
layer connectivity

Sparcely connected: all nodes go to all nodes of the next layer.

The last layer is always connected

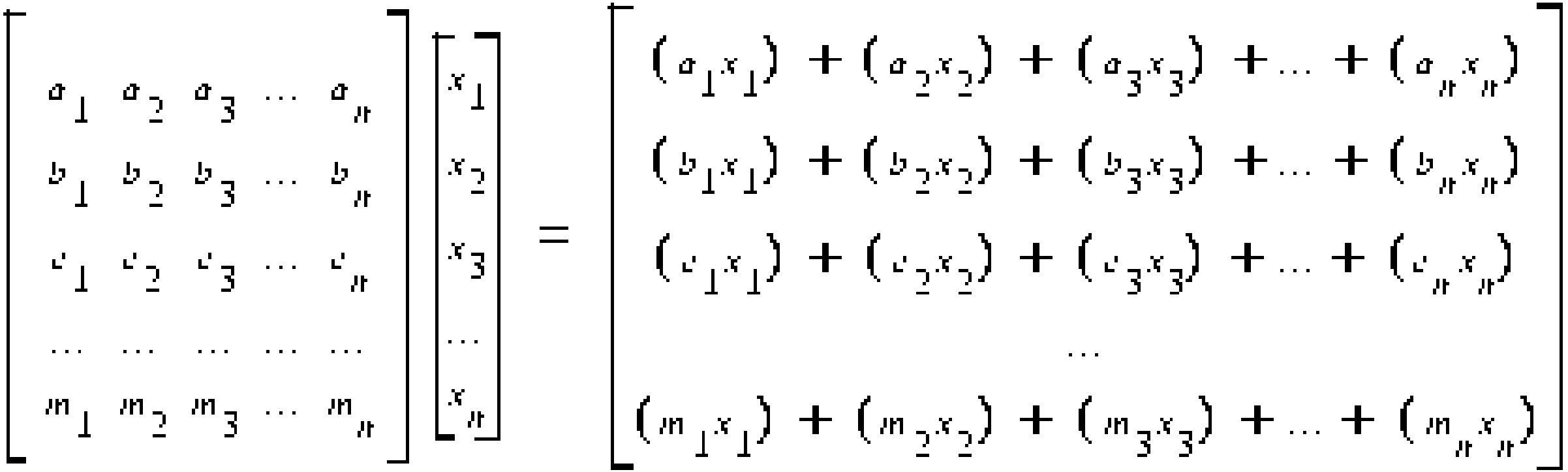


how does it relate to matrix multiplication



each layer is a matrix

Except this is a very misleading representation

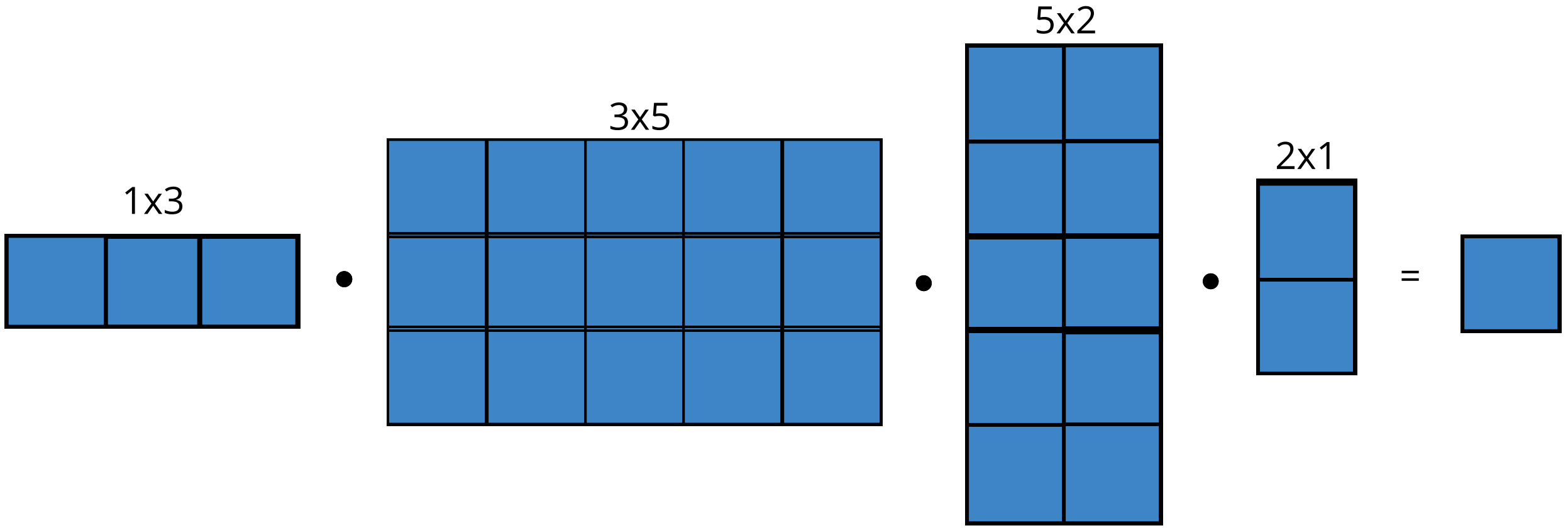


there are no biases or activation functions

each layer should be a different shape

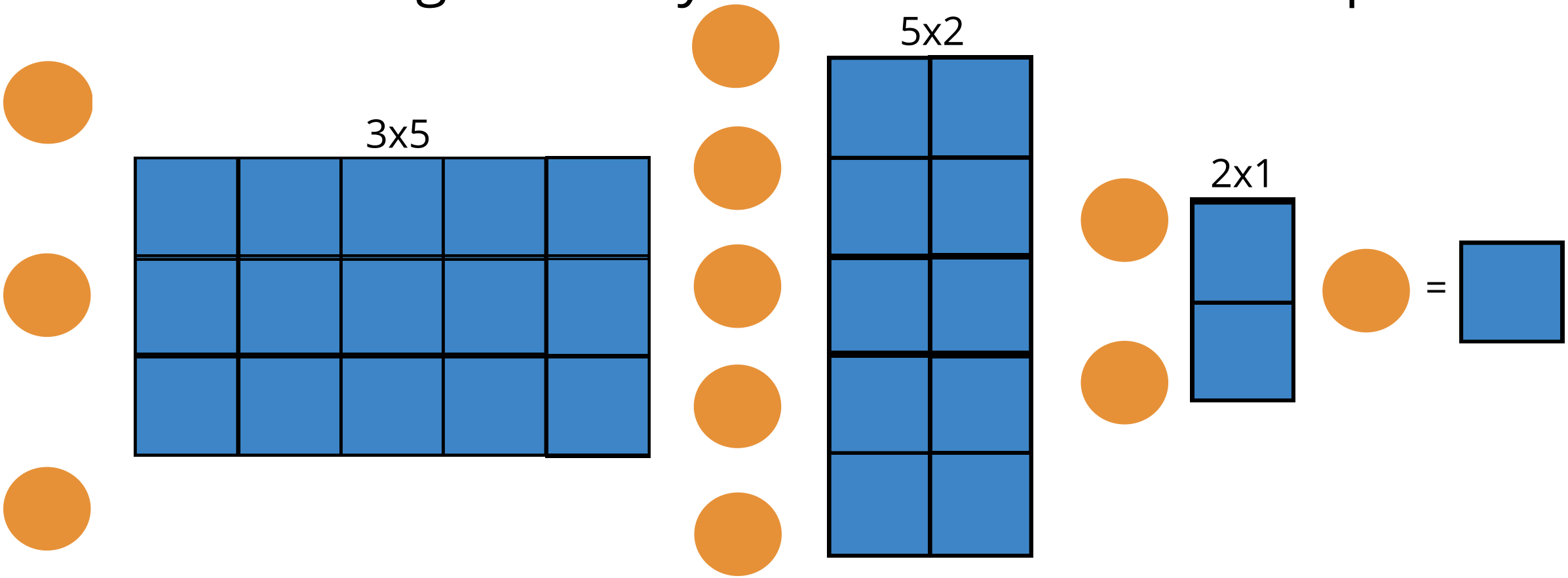
DeepNeuralNetwork

what we are doing is just a series of matrix multiplictions.



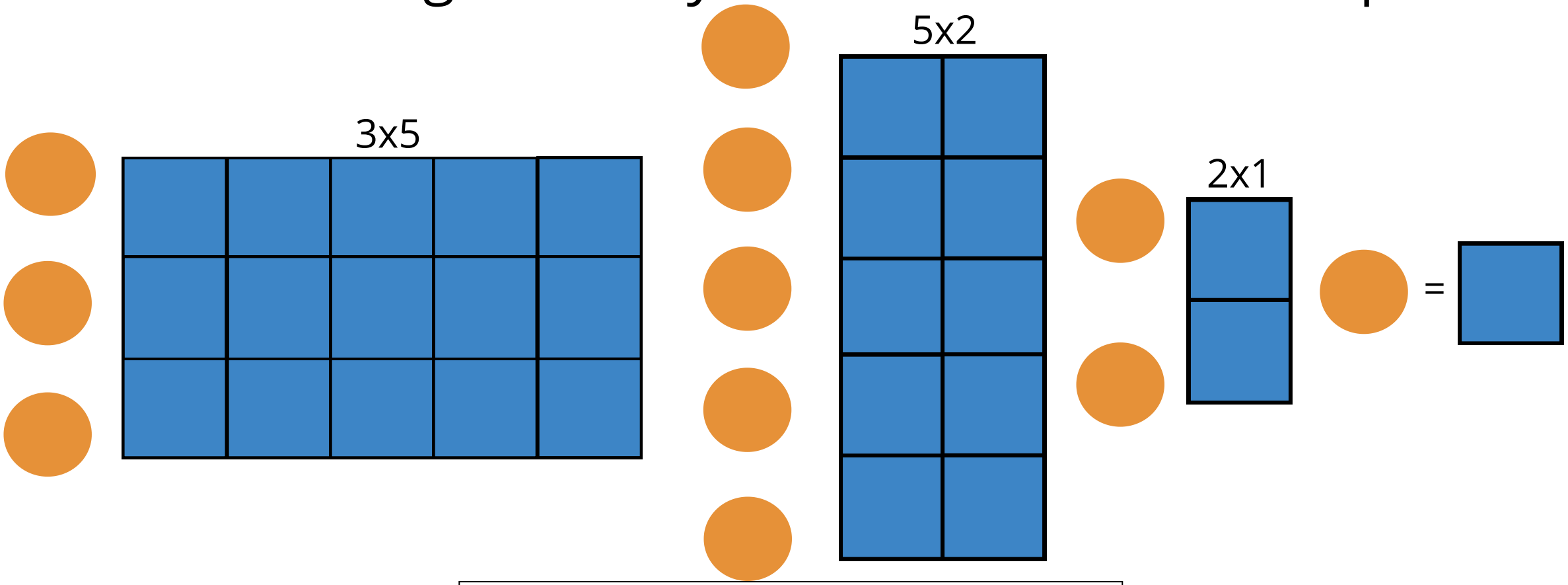
DeepNeuralNetwork

what we are doing is exactly a series of matrix multiplictions.



DeepNeuralNetwork

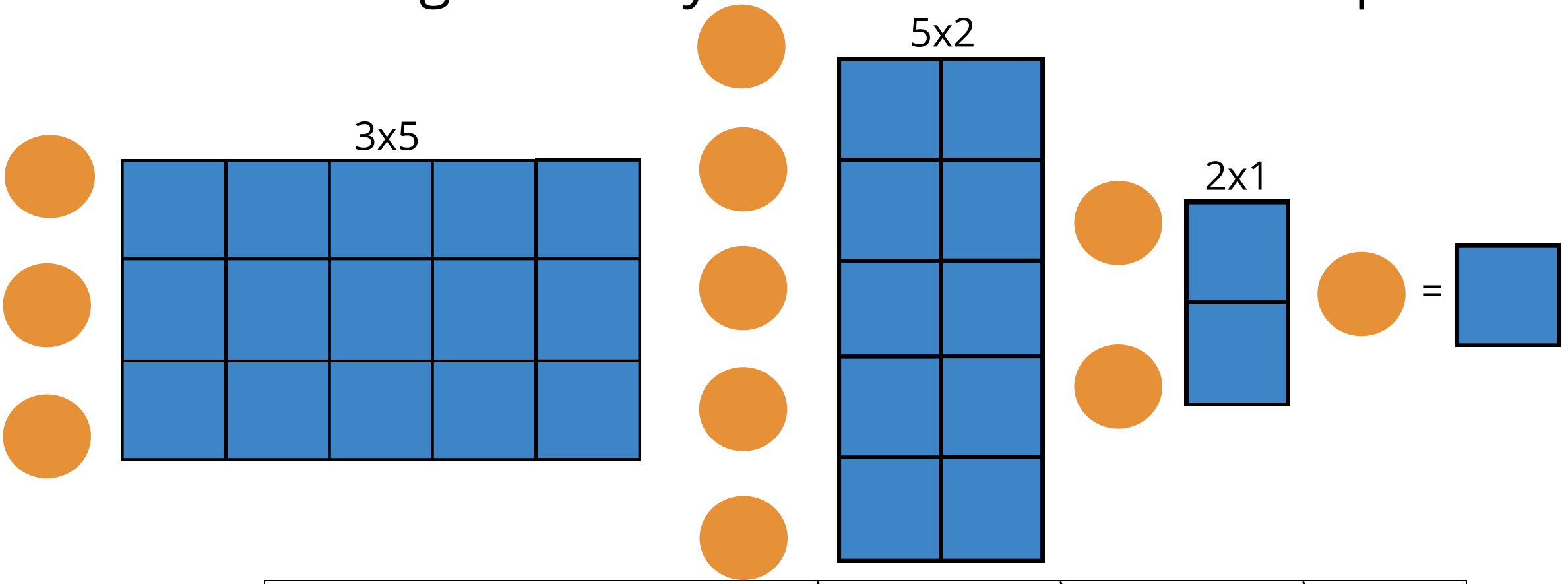
what we are doing is exactly a series of matrix multiplictions.



$$(((\vec{x} \cdot W_1) \cdot W_2) \cdot W_3) = y$$

DeepNeuralNetwork

what we are doing is exactly a series of matrix multiplictions.



$$f^{(3)}(f^{(2)}(f^{(1)}(\vec{x} \cdot W_1 + \vec{b_1}) \cdot W_2 + \vec{b_2}) \cdot W_3 + \vec{b_3}) = y$$

DeepNeuralNetwork

what we are doing is exactly a series of matrix multiplictions.

The purpose is to
approximate a function φ

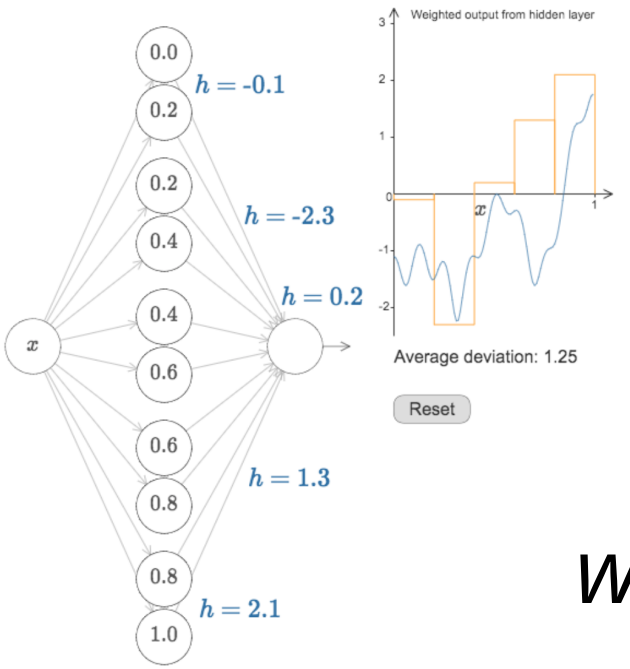
y = *φ(x)*

which (in general) is not linear
with linear operations

$$\phi(\vec{x}) \sim f^{(3)}(f^{(2)}(f^{(1)}(\vec{x} \cdot W_1 + \vec{b_1}) \cdot W_2 + \vec{b_2}) \cdot W_3 + \vec{b_3}) = y$$

DeepNeuralNetwork

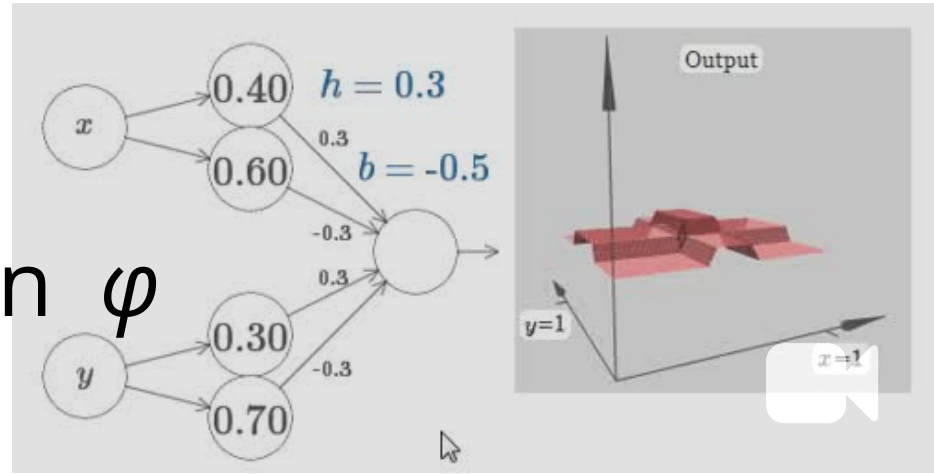
http://neuralnetworksanddeeplearning.com/chap4.html



The purpose is to approximate a function φ

y = φ(x)

which (in general) is not linear with linear operations



$$\phi(\vec{x}) \sim f^{(3)}(f^{(2)}(f^{(1)}(\vec{x} \cdot W_1 + \vec{b_1}) \cdot W_2 + \vec{b_2}) \cdot W_3 + \vec{b_3}) = y$$

Parameters and hyperparameters

how many hyperparameters?

32 parameters and

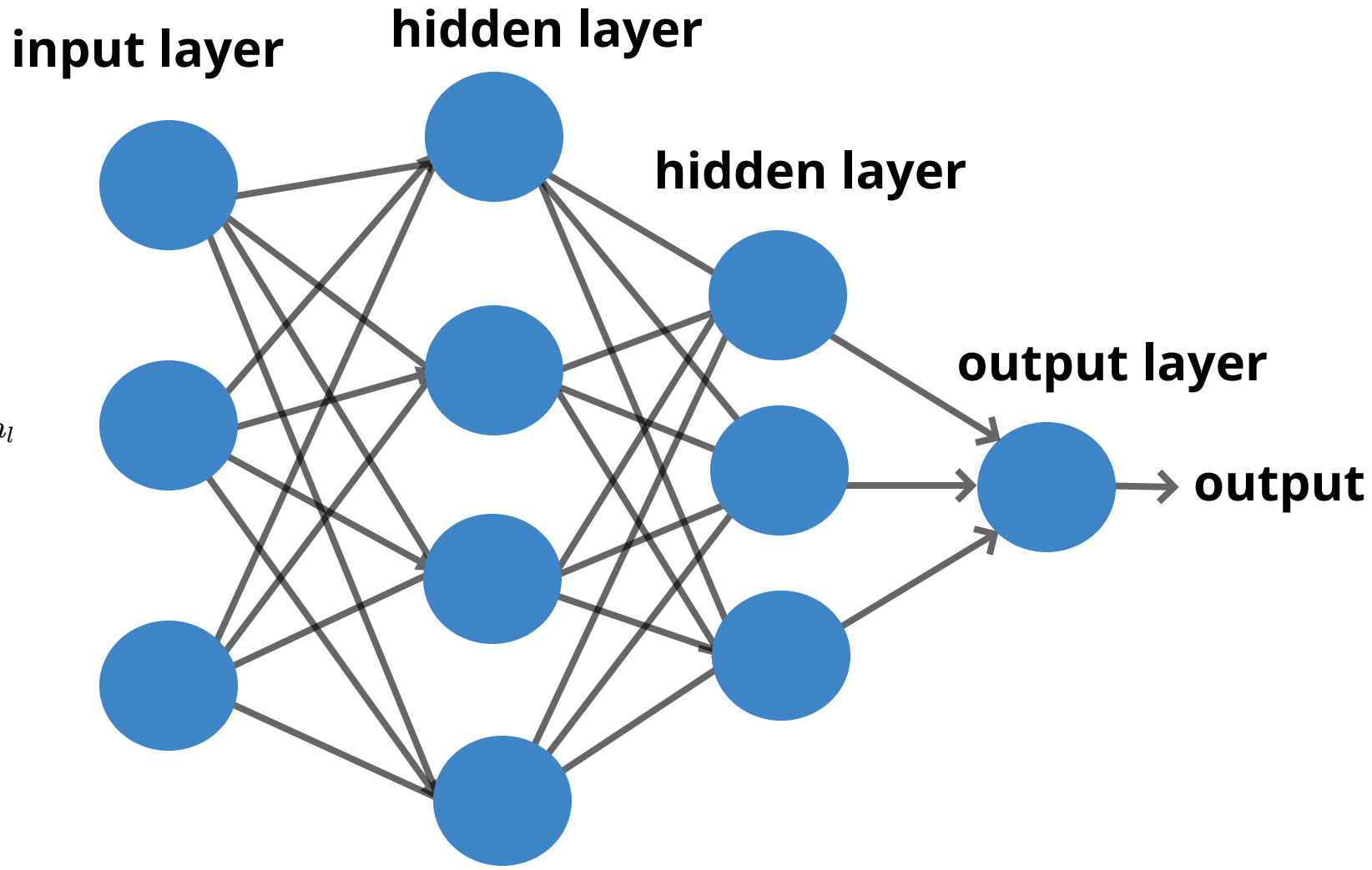
?? hyperparameters

activation functions - $\sum_{l=1}^{N} N_{n_l}$

loss function - *1*

optimization method - *1*

architecture - *M*



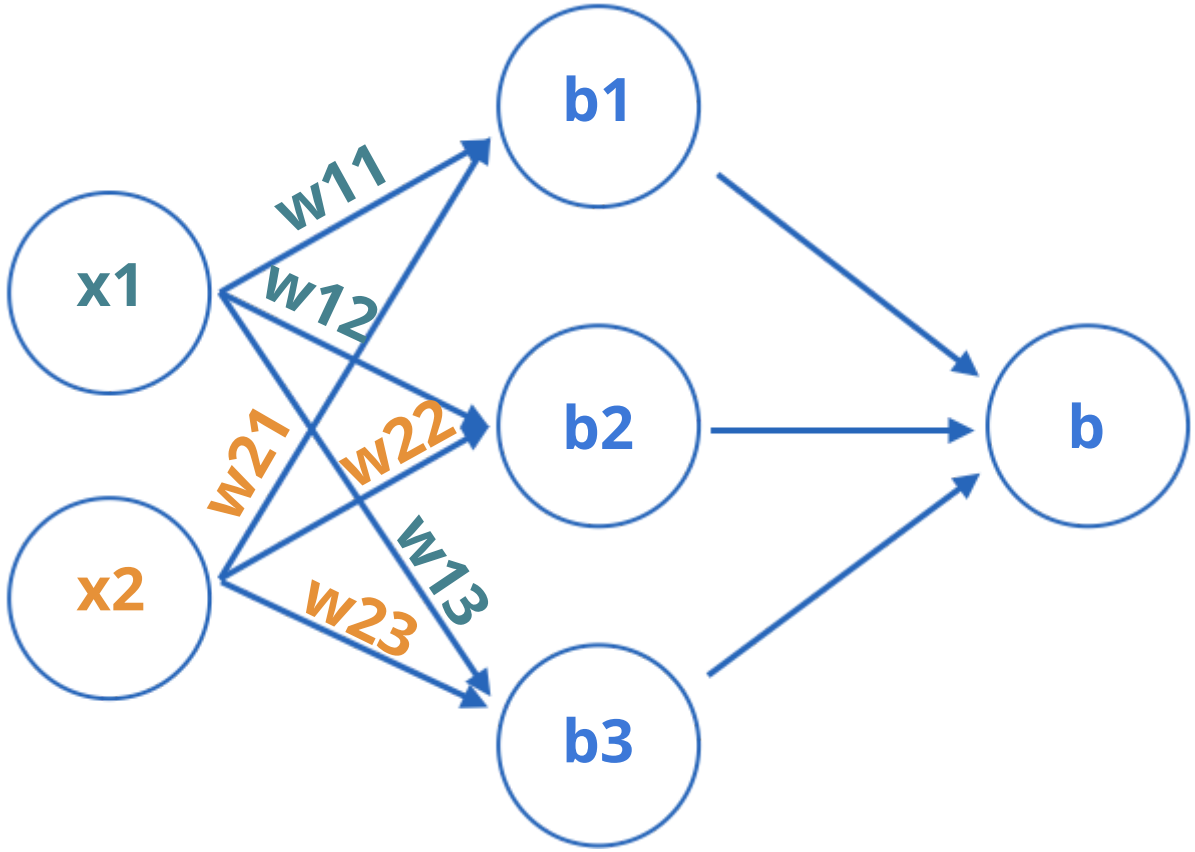
Training models with this many parameters requires a lot of care:

. defining the metric

. optimization schemes

. training/validation/testing sets

But just like our simple linear regression case, the fact that small changes in the parameters leads to small changes in the output for the right activation functions.



define a cost function, e.g.

$$C = \frac{1}{2}|y - a^L|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

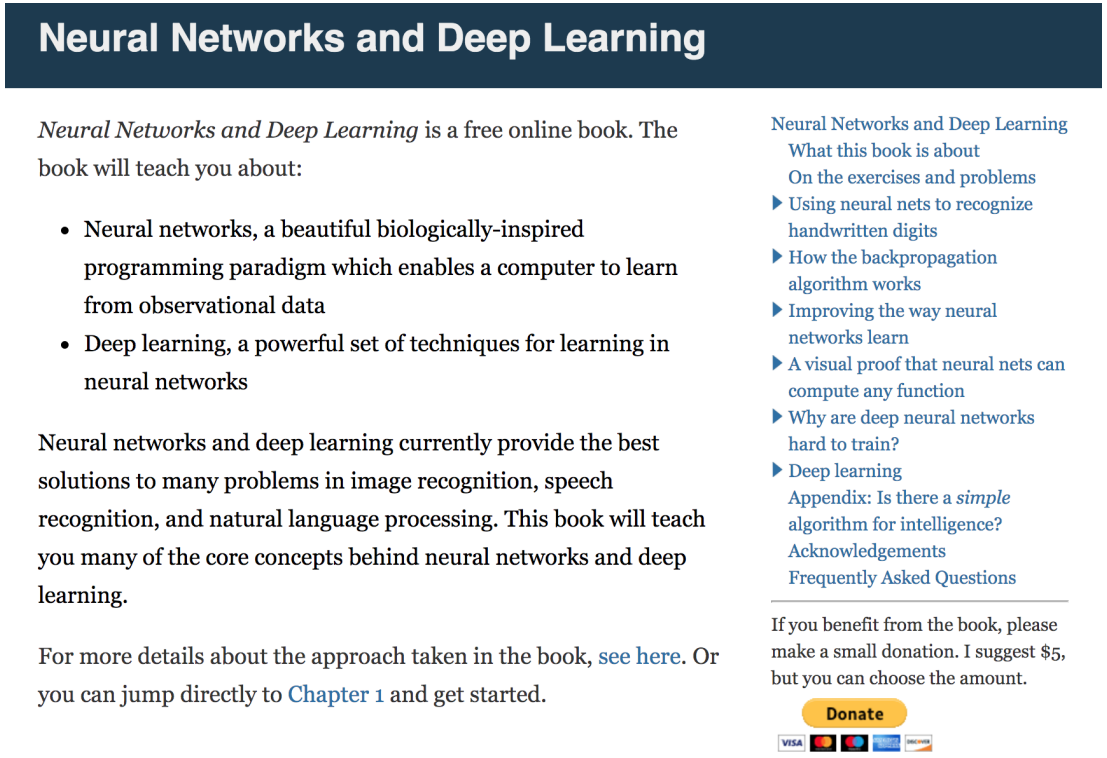
$$\text{output} = \frac{1}{1+e^{-\frac{w_7}{1+e^{-w_1x_1-w_4x_2-b_1}}-\frac{w_8}{1+e^{-w_2x_1-w_5x_2-b_2}}-\frac{w_9}{1+e^{-w_3x_1-w_6x_2-b_3}}-b_4}}$$

$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1 ... W_N + b_N)))$$


0
proper care of your DNN 0

NN are a vast topics and we only have 2 weeks!

Some FREE references!



Neural Networks and Deep Learning

Neural Networks and Deep Learning is a free online book. The book will teach you about:

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data
- Deep learning, a powerful set of techniques for learning in neural networks

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. This book will teach you many of the core concepts behind neural networks and deep learning.

For more details about the approach taken in the book, see here. Or you can jump directly to Chapter 1 and get started.

Neural Networks and Deep Learning
- What this book is about
- On the exercises and problems
- Using neural nets to recognize handwritten digits
- How the backpropagation algorithm works
- Improving the way neural networks learn
- A visual proof that neural nets can compute any function
- Why are deep neural networks hard to train?
- Deep learning

Appendix: Is there a *simple* algorithm for intelligence?
Acknowledgements
Frequently Asked Questions

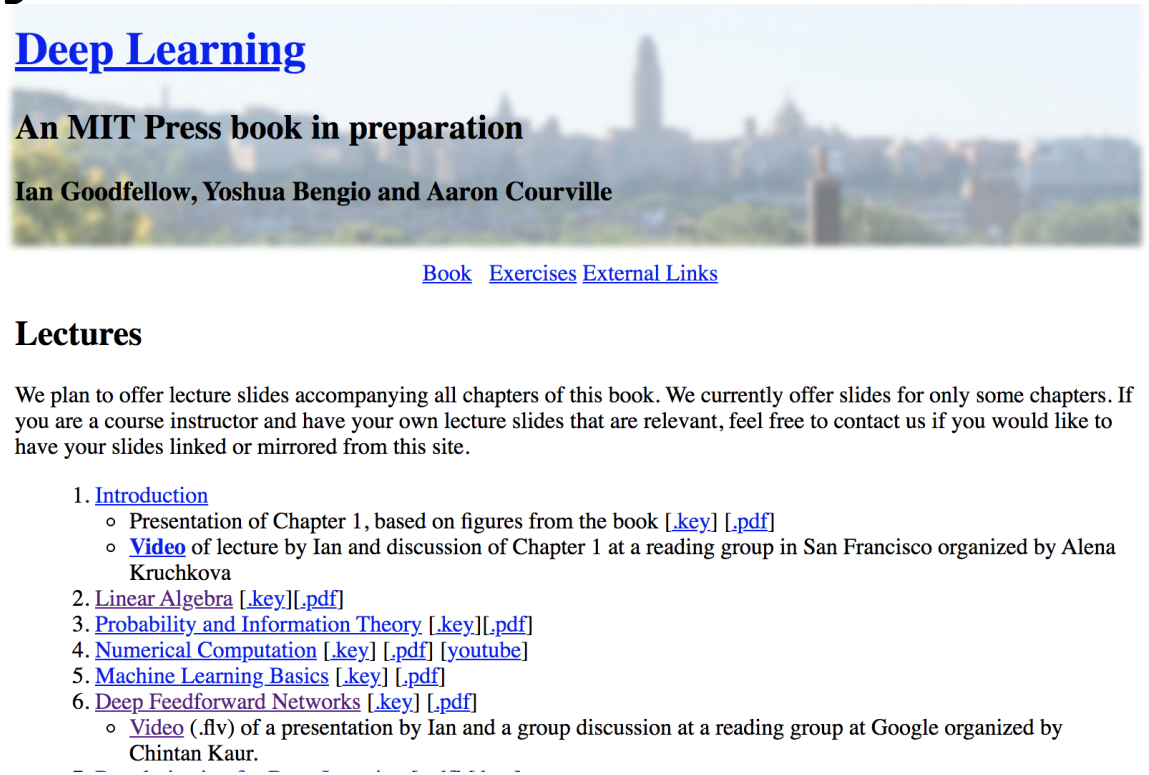
If you benefit from the book, please make a small donation. I suggest $5, but you can choose the amount.

Donate

http://neuralnetworksanddeeplearning.com/index.html

michael nielsen

better pedagogical approach, more basic, more clear



Deep Learning

An MIT Press book in preparation

Ian Goodfellow, Yoshua Bengio and Aaron Courville

Book Exercises External Links

Lectures

We plan to offer lecture slides accompanying all chapters of this book. We currently offer slides for only some chapters. If you are a course instructor and have your own lecture slides that are relevant, feel free to contact us if you would like to have your slides linked or mirrored from this site.

1. Introduction
   - Presentation of Chapter 1, based on figures from the book [.key] [.pdf]
   - **Video** of lecture by Ian and discussion of Chapter 1 at a reading group in San Francisco organized by Alena Kruchkova
2. Linear Algebra [.key][.pdf]
3. Probability and Information Theory [.key][.pdf]
4. Numerical Computation [.key] [.pdf] [youtube]
5. Machine Learning Basics [.key] [.pdf]
6. Deep Feedforward Networks [.key] [.pdf]
   - Video (.flv) of a presentation by Ian and a group discussion at a reading group at Google organized by Chintan Kaur.

https://www.deeplearningbook.org/

ian goodfellow

mathematical approach, more advanced, unfinished

Lots of parameters and lots of hyperparameters! What to choose? cheatsheet

1. **architecture - wide networks tend to overfit, deep networks are hard to train**

2. number of epochs - the sweet spot is when learning slows down, but before you start overfitting... it may take DAYS! jumps may indicate bad initial choices (like in all gradient descent)

3. loss function - needs to be appropriate to the task, e.g. classification vs regression

4. activation functions - needs to be consistent with the loss function

5. optimization scheme - needs to be appropriate to the task and data

6. learning rate in optimization - balance speed and accuracy

7. batch size - smaller batch size is faster but leads to overtraining

An article that compars various DNNs

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

Alfredo Canziani & Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
{canziani,euge}@purdue.edu

Adam Paszke
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
a.paszke@students.mimuw.edu.pl

ABSTRACT

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.

https://arxiv.org/pdf/1605.07678.pdf

An article that compars various DNNs

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

Alfredo Canziani & Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
{canziani,euge}@purdue.edu

Adam Paszke
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
a.paszke@students.mimuw.edu.pl

ABSTRACT

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.

https://arxiv.org/pdf/1605.07678.pdf

accuracy comparison

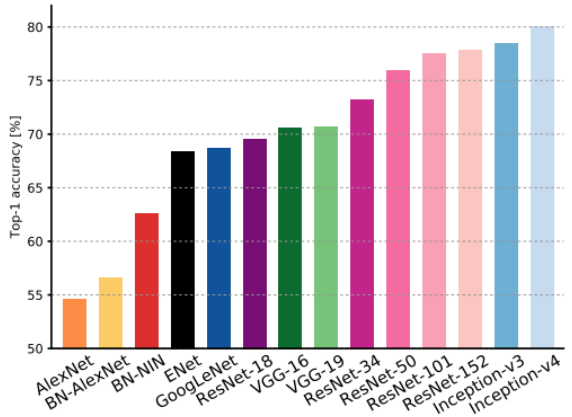


Figure 1: **Top1 *vs.* network.** Single-crop top-1 validation accuracies for top scoring single-model architectures. We introduce with this chart our choice of

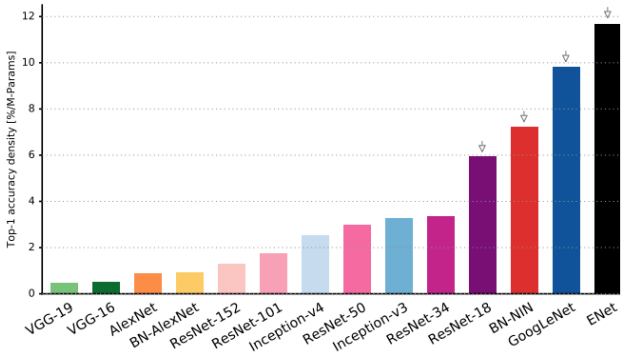


Figure 10: **Accuracy per parameter *vs.* network.** Information density (accuracy per parameters) is an efficiency metric that highlight that capacity of a specific architecture to better utilise its parametric space. Models like VGG and AlexNet are clearly oversized, and do not take fully advantage of their potential learning ability. On the far right, ResNet-18, BN-NIN, GoogLeNet and ENet (marked by grey arrows) do a better job at "squeezing" all their neurons to learn the given task, and are the winners of this section.

An article that compars various DNNs

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

Alfredo Canziani & Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
{canziani,euge}@purdue.edu

Adam Paszke
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
a.paszke@students.mimuw.edu.pl

ABSTRACT

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.

https://arxiv.org/pdf/1605.07678.pdf

accuracy comparison

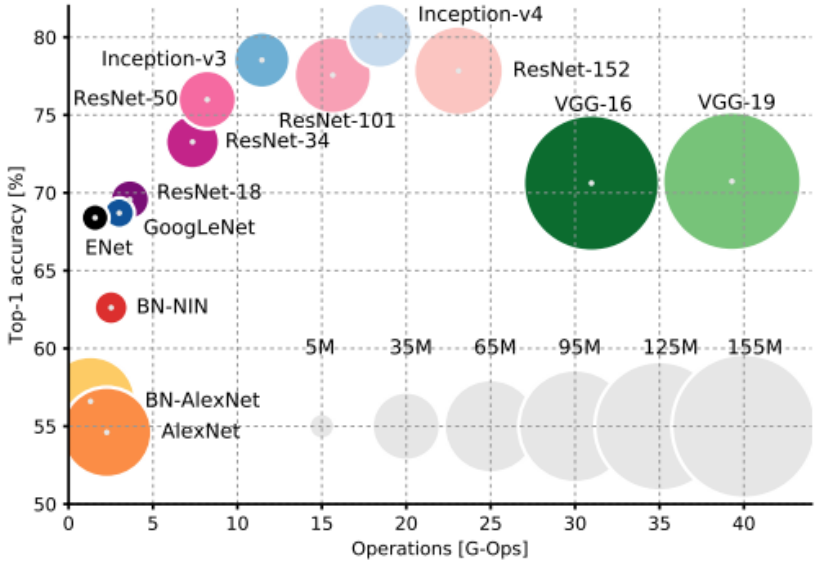


Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.** Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from 5×10^6 to 155×10^6 params. Both these figures share the same *y*-axis, and the grey dots highlight the centre of the blobs.

An article that compars various DNNs

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

Alfredo Canziani & Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
{canziani,euge}@purdue.edu

Adam Paszke
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
a.paszke@students.mimuw.edu.pl

ABSTRACT

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.

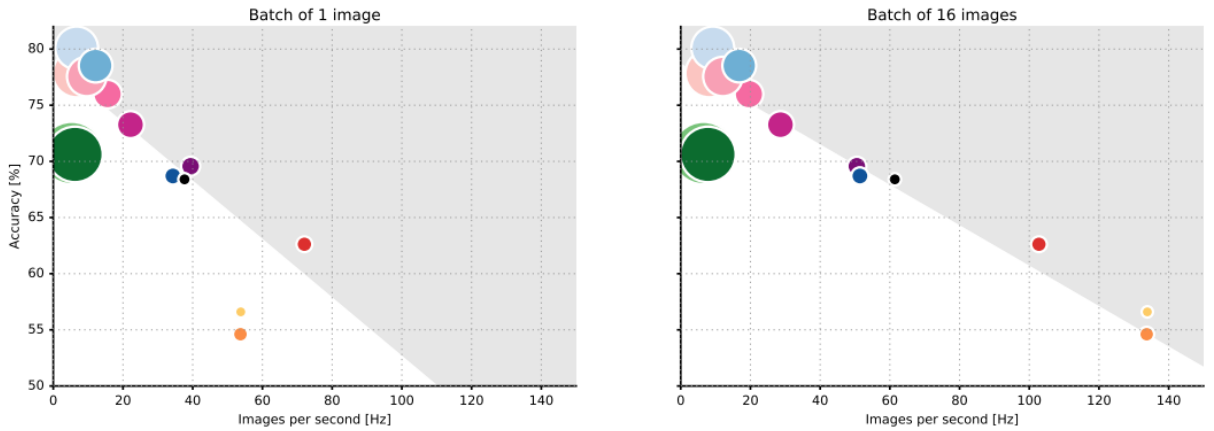


Figure 9: **Accuracy *vs.* inferences per second, size $\propto$ operations.** Non trivial linear upper bound is shown in these scatter plots, illustrating the relationship between prediction accuracy and throughput of all examined architectures. These are the first charts in which the area of the blobs is proportional to the amount of operations, instead of the parameters count. We can notice that larger blobs are concentrated on the left side of the charts, in correspondence of low throughput, *i.e.* longer inference times. Most of the architectures lay on the linear interface between the grey and white areas. If a network falls in the shaded area, it means it achieves exceptional accuracy or inference speed. The white area indicates a suboptimal region. *E.g.* both AlexNet architectures improve processing speed as larger batches are adopted, gaining 80 Hz.

https://arxiv.org/pdf/1605.07678.pdf

batch size

Lots of parameters and lots of hyperparameters! What to choose? cheatsheet

1. architecture - wide networks tend to overfit, deep networks are hard to train

2. **number of epochs - the sweet spot is when learning slows down, but before you start overfitting... it may take DAYS!** jumps may indicate bad initial choices

3. **loss function - needs to be appropriate to the task, e.g. classification vs regression**

4. activation functions - needs to be consistent with the loss function

5. optimization scheme - needs to be appropriate to the task and data

6. learning rate in optimization - balance speed and accuracy

7. batch size - smaller batch size is faster but leads to overtraining

Lots of parameters and lots of hyperparameters! What to choose?

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

3.0.1 regression

- loss='mean_squared_error' L2: default loss to use for regression problems. => linear activation funct in output layer, one node out

alternatives: loss='mean_squared_logarithmic_error', 'mean_absolute_error' (which is L1 instead of L2)

3.0.2 binary classification

- loss='binary_crossentropy' => sigmoid activation function in output layer, one node out

alternatives: 'hinge'

3.0.3 multiclass classification

categorical encoded as numerical

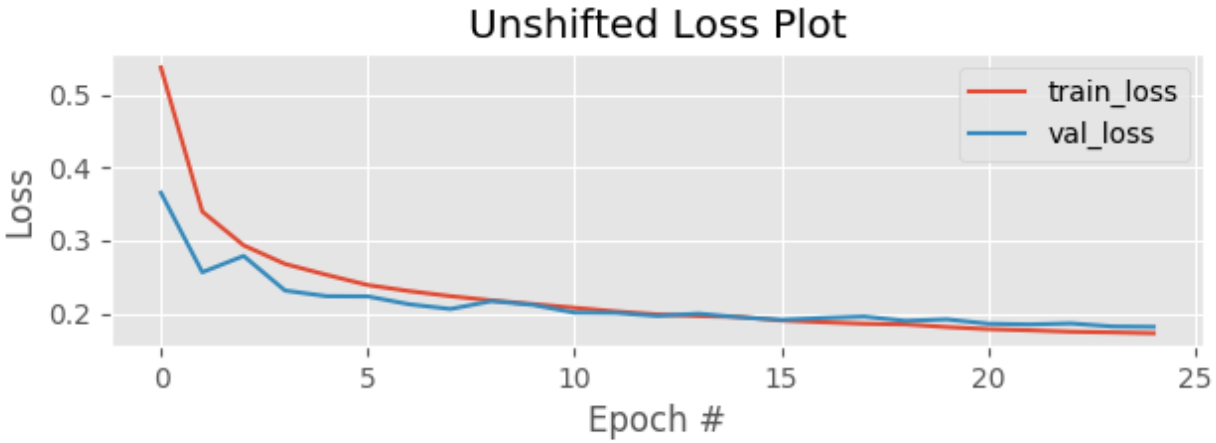
- loss='categorical_crossentropy' => softmax n nodes out

onehot encoded categoridal

- 'parse_categorical_crossentropy' => softmax n nodes out
- 'kullback Leibler Divergence Loss' => probabilistic categorical classification; log(P/Q)

What should I choose for the loss function and how does that relate to the activation functiom and optimization?

Lots of parameters and lots of hyperparameters! What to choose? cheatsheet



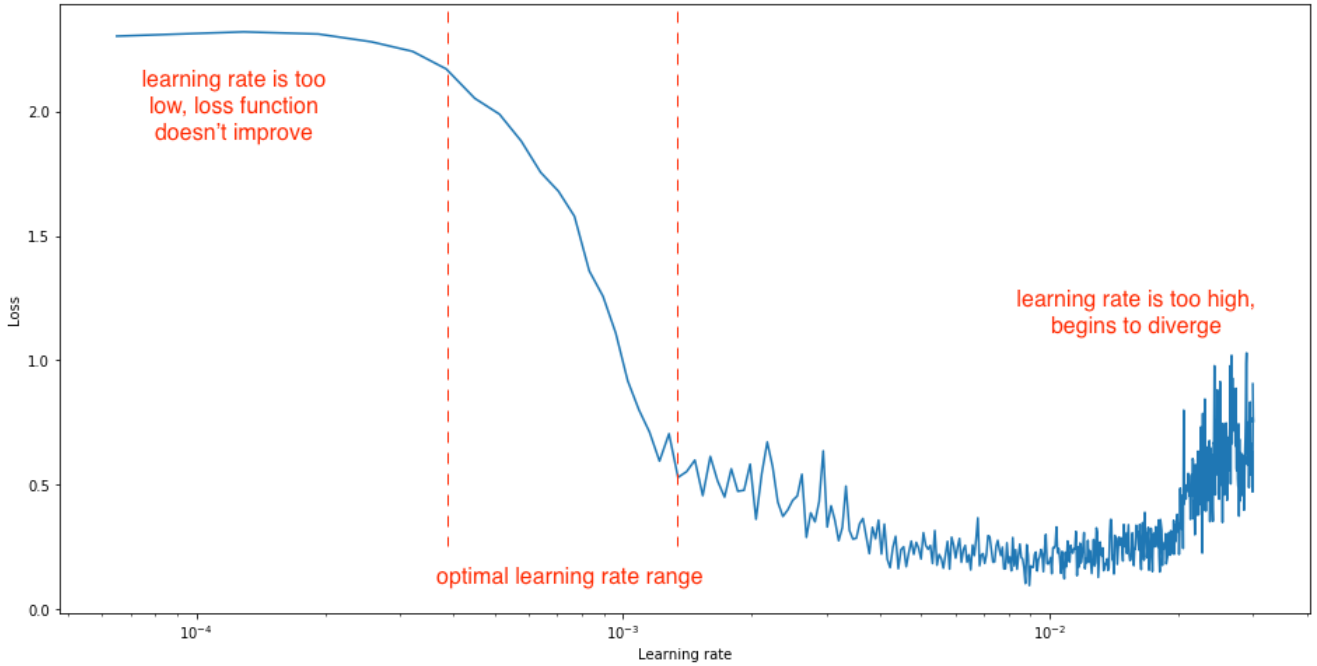
always check your loss function! it should go down smoothly and flatten out at the end of the training.

not flat? you are still learning!

too flat? you are overfitting...

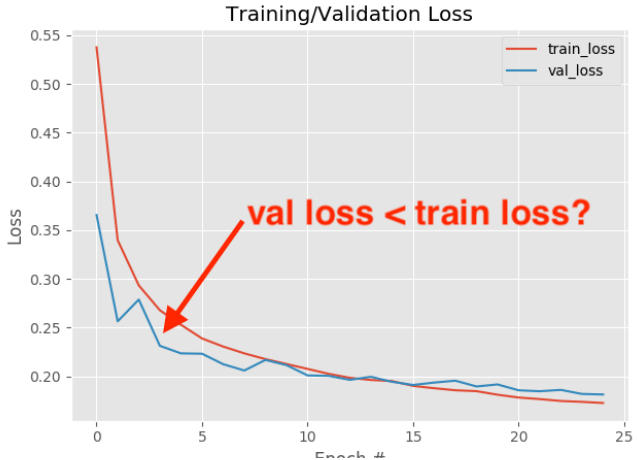
loss (gallery of horrors)

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb



loss and learning rate (not that the appropriate learning rate depends on the chosen optimization scheme!)

jumps are not unlikely (and not necessarily a problem) if your activations are discontinuous (e.g. relu)



when you use validation you are introducing regularizations (e.g. dropout) so the loss can be smaller than for the training set

Building a DNN

with keras and tensorflow

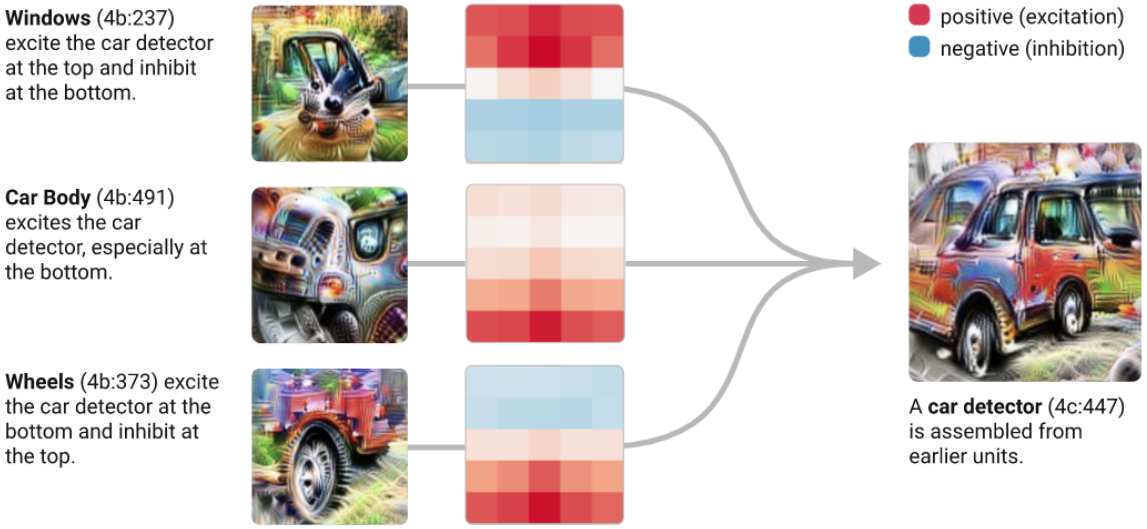
autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

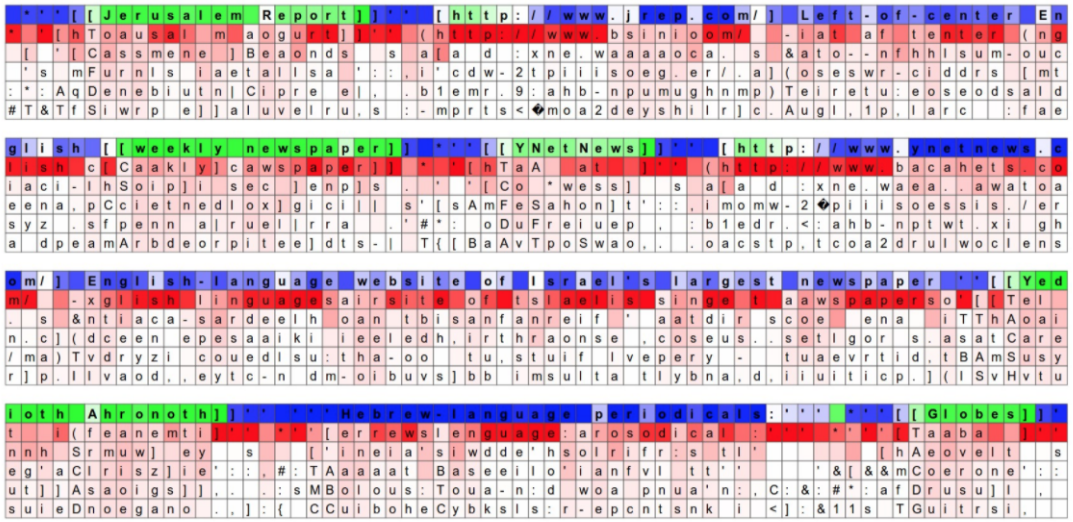
What should I choose for the loss function and how does that relate to the activation functiom and optimization?

loss	**good for**	**activation last layer**	**size last layer**
mean_squared_error	regression	linear	one node
mean_absolute_error	regression	linear	one node
mean_squared_logarithmit_error	regression	linear	one node
binary_crossentropy	binary classification	sigmoid	one node
categorical_crossentropy	multiclass classification	sigmoid	N nodes
Kullback_Divergence	multiclass classification, probabilistic inerpretation	sigmoid	N nodes

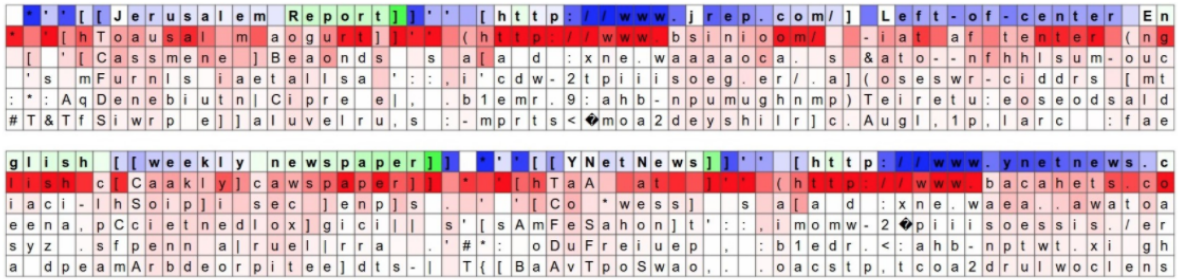
On the interpretability of DNNs



https://distill.pub/2020/circuits/zoom-in/



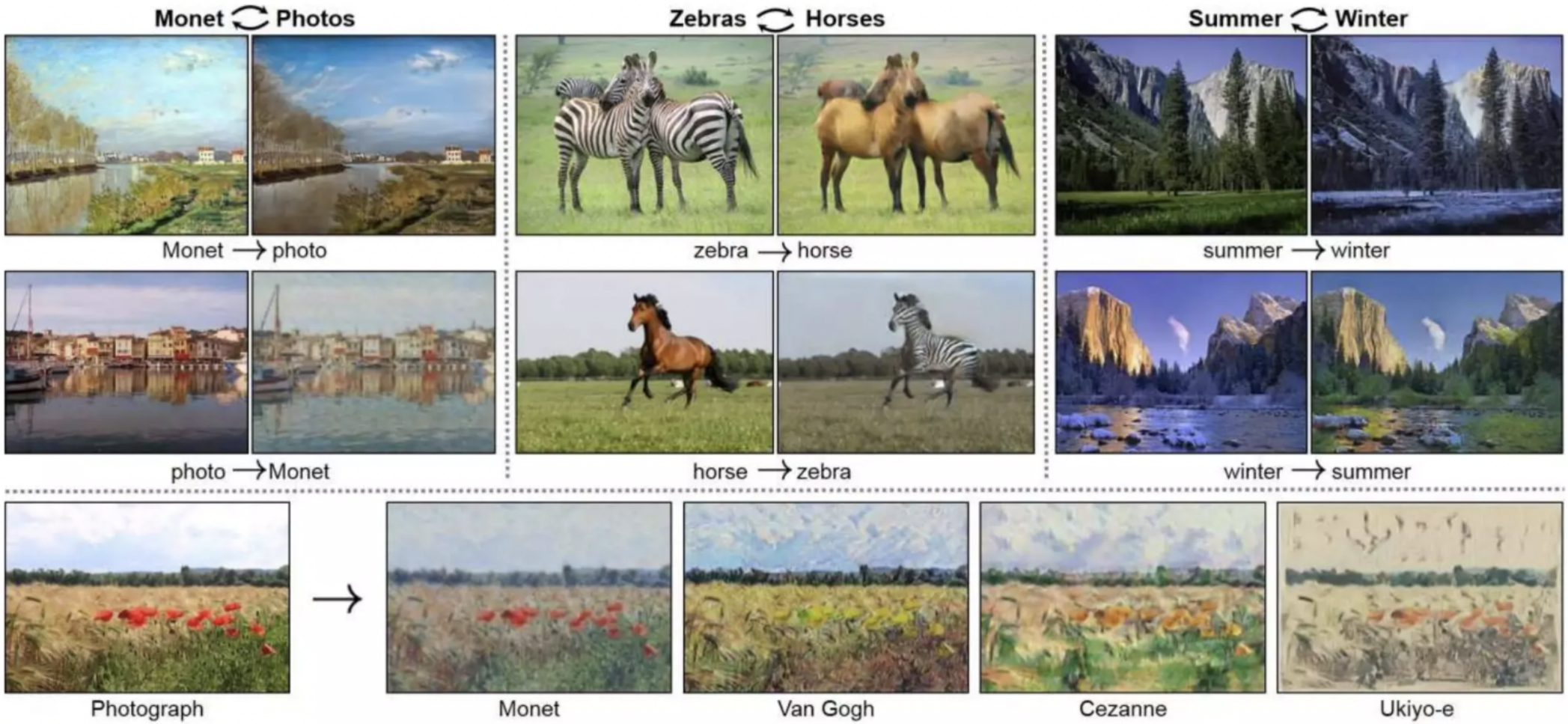
The highlighted neuron here gets very excited when the RNN is inside the [[ ]] markdown environment and turns off outside of it. Interestingly, the neuron can't turn on right after it sees the character "[", it must wait for the second "[" and then activate. This task of counting whether the model has seen one or two "[" is likely done with a different neuron.



Here we see a neuron that varies seemingly linearly across the [[ ]] environment. In other words its activation is giving the RNN a time-aligned coordinate system across the [[ ]] scope. The RNN can use this information to make different characters more or less likely depending on how early/late it is in the [[ ]] scope (perhaps?).

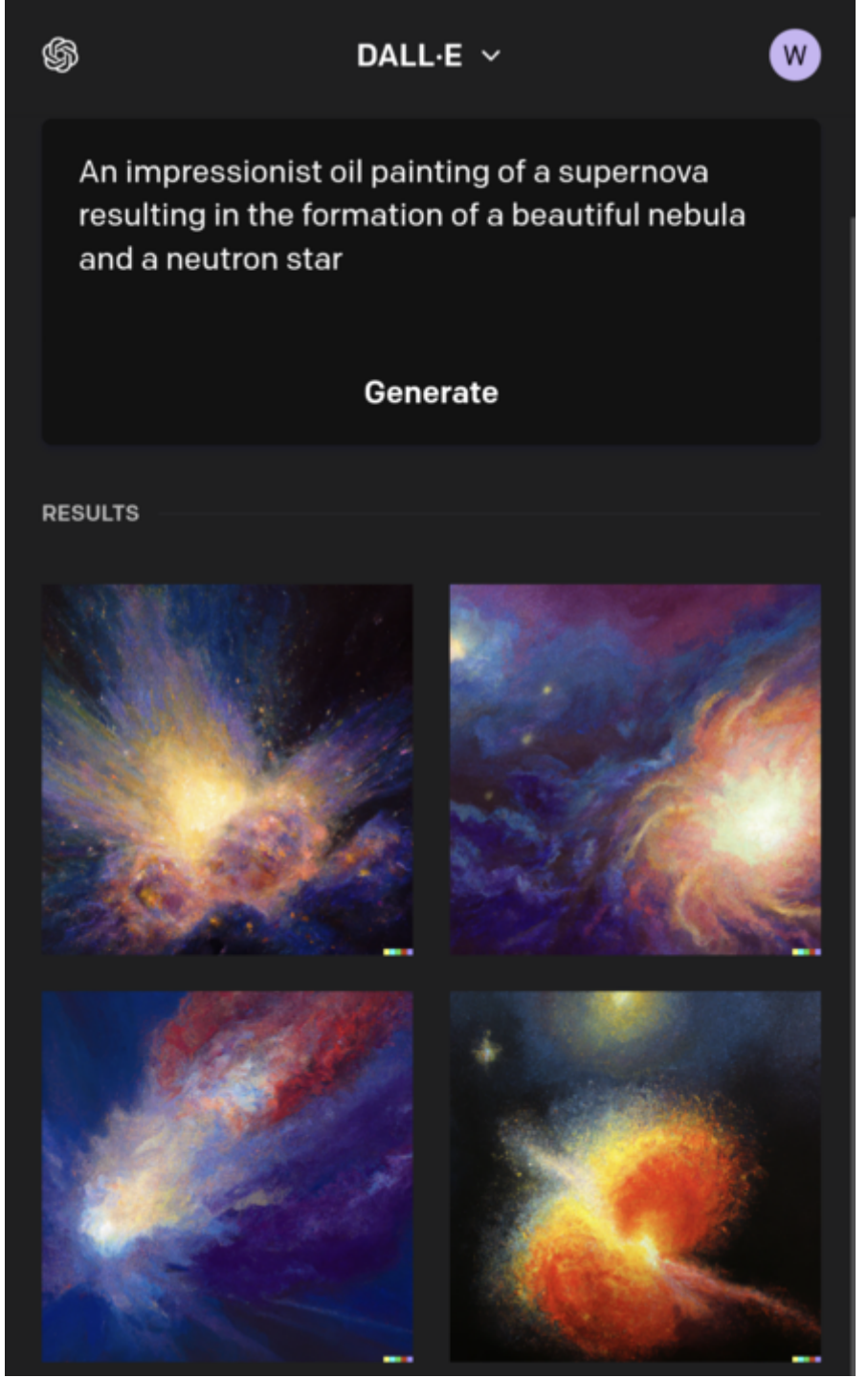
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

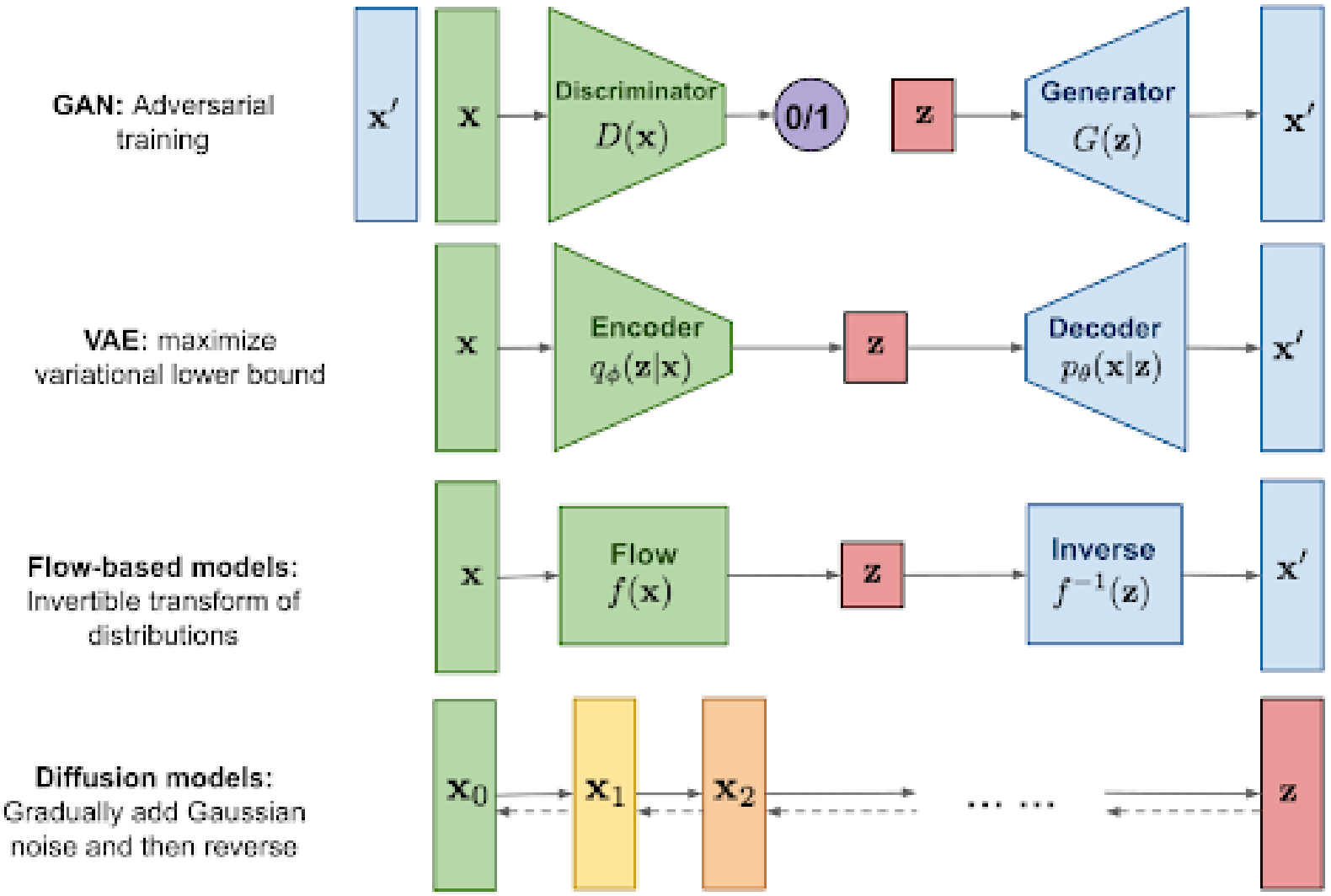
1 generative AI

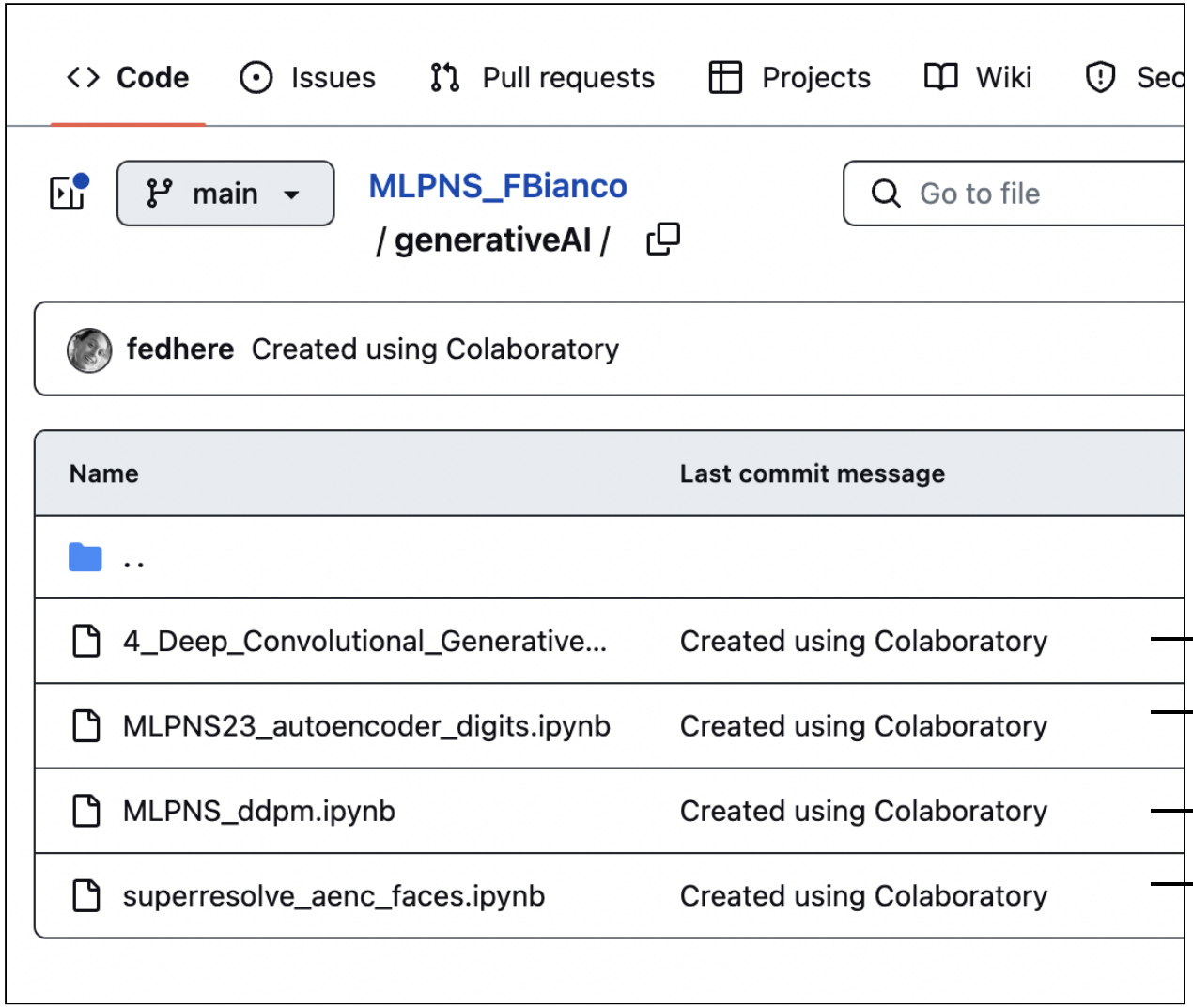

Monet ⇄ Photos
Zebras ⇄ Horses
Summer ⇄ Winter
Monet → photo
zebra → horse
summer → winter
photo → Monet
horse → zebra
winter → summer
Photograph
Monet
Van Gogh
Cezanne
Ukiyo-e

Applications

1. **Image Generation (and 3D Shape Generation)**
2. **Semantic Image-to-Photo Translation**
3. **Image Resolution Increase**
4. **Text-to-Speech Generator**
5. **Speech-to-Speech Conversion**
6. **Text Generation (Chat GP3)**
7. **Music Generation**
8. **Image-to-Image Conversion**




GAN: Adversarial training
$\mathbf{x}'$
$\mathbf{x}$
Discriminator
$D(\mathbf{x})$
0/1
$\mathbf{z}$
Generator
$G(\mathbf{z})$
$\mathbf{x}'$
VAE: maximize variational lower bound
$\mathbf{x}$
Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{z}$
Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
Flow-based models: Invertible transform of distributions
$\mathbf{x}$
Flow
$f(\mathbf{x})$
$\mathbf{z}$
Inverse
$f^{-1}(\mathbf{z})$
$\mathbf{x}'$
Diffusion models: Gradually add Gaussian noise and then reverse
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
... ...
$\mathbf{z}$



GANs

VAE

Diffusion models

VAE

https://github.com/fedhere/MLPNS_FBianco/tree/main/generativeAI

2 Autoencoders

Unsupervised learning with Neural Networks

What do NN do? approximate complex functions with series of linear functions

…. so if my layers are smaller what I have is a compact representation of the data

Unsupervised learning with Neural Networks

What do NN do? approximate complex functions with series of linear functions

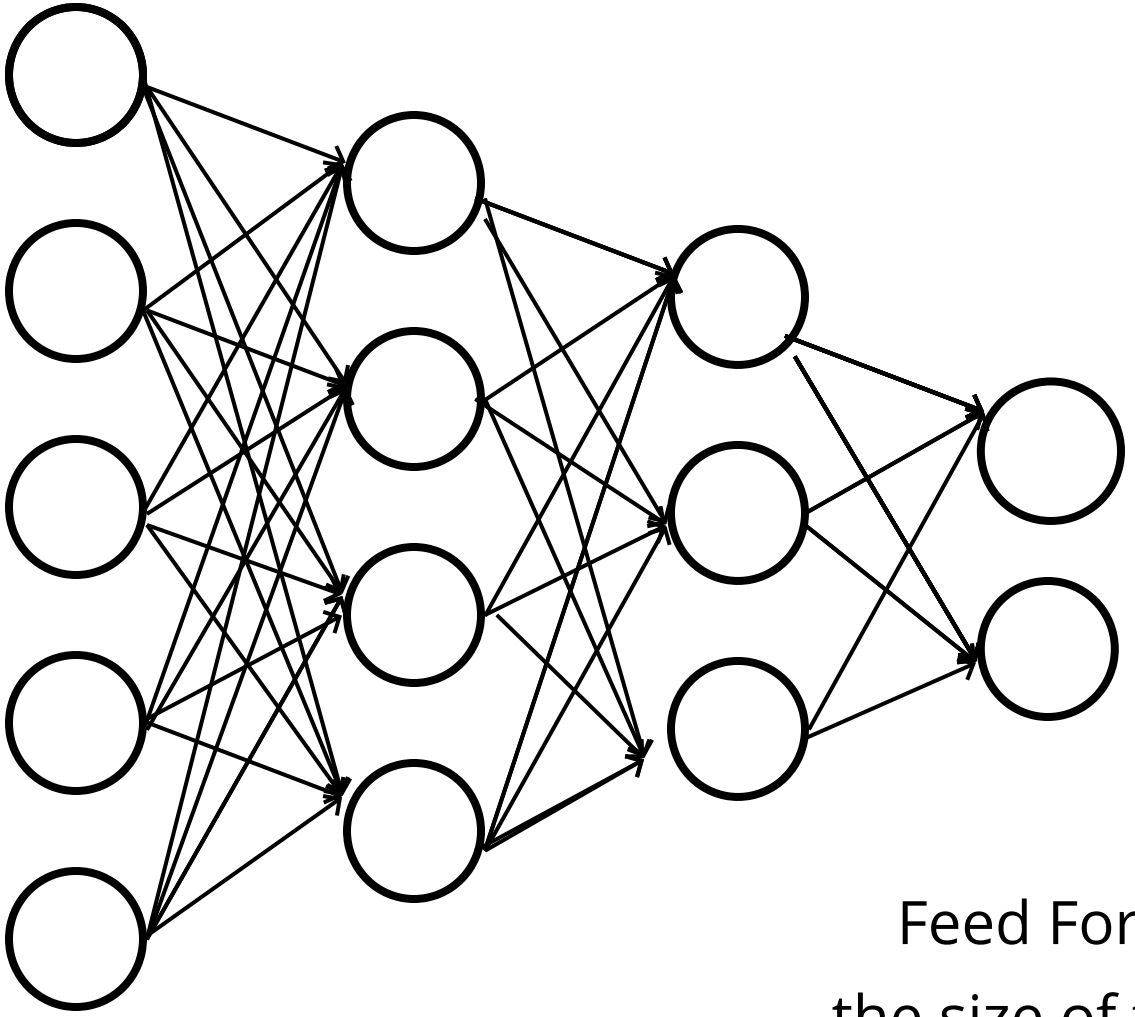
To do that they extract information from the data

Each layer of the DNN produces a representation of the data a "latent representation" .

The dimensionality of that latent representation is determined by the size of the layer (and its connectivity, but we will ignore this bit for now)

…. so if my layers are smaller what I have is a compact representation of the data

Autoencoder Architecture

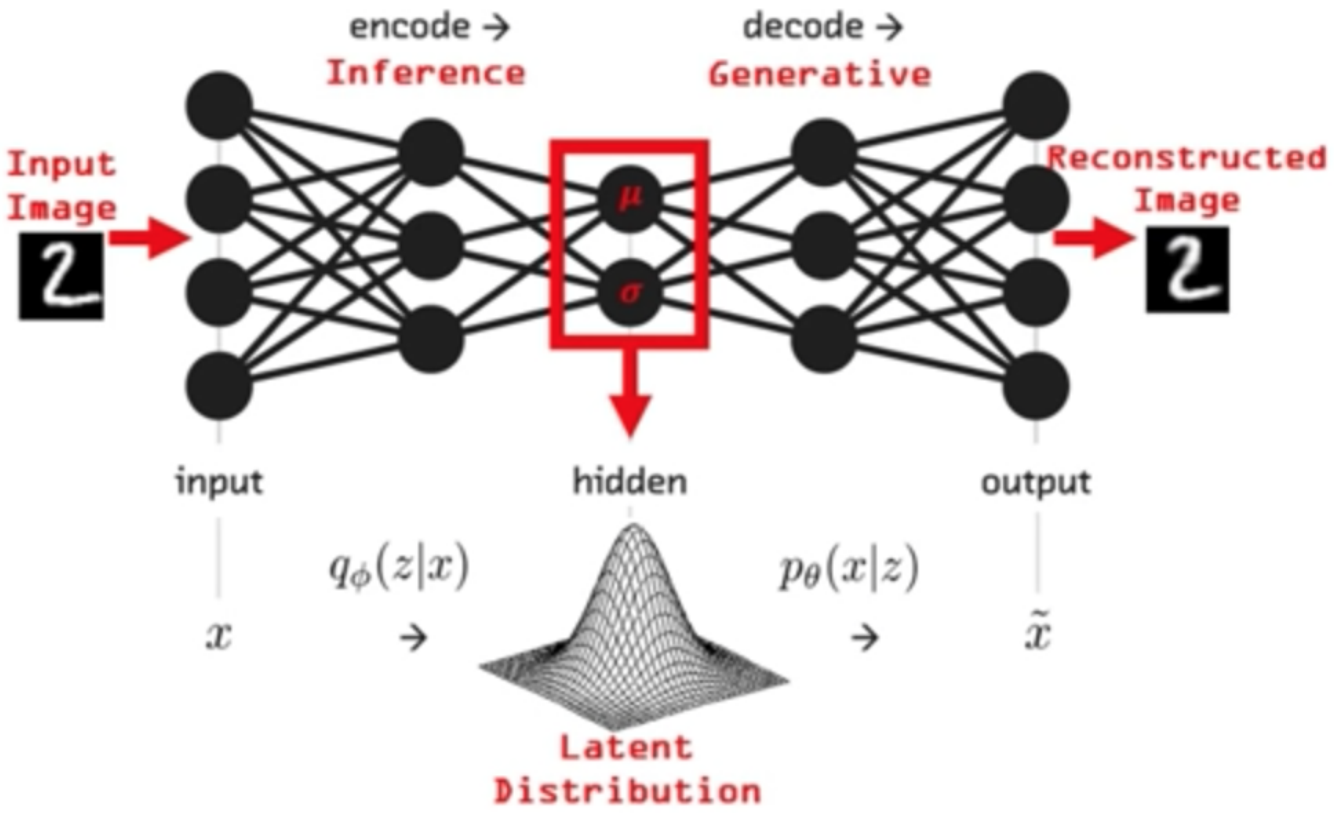


Feed Forward DNN:

the size of the input is 5,

the size of the last layer is 2

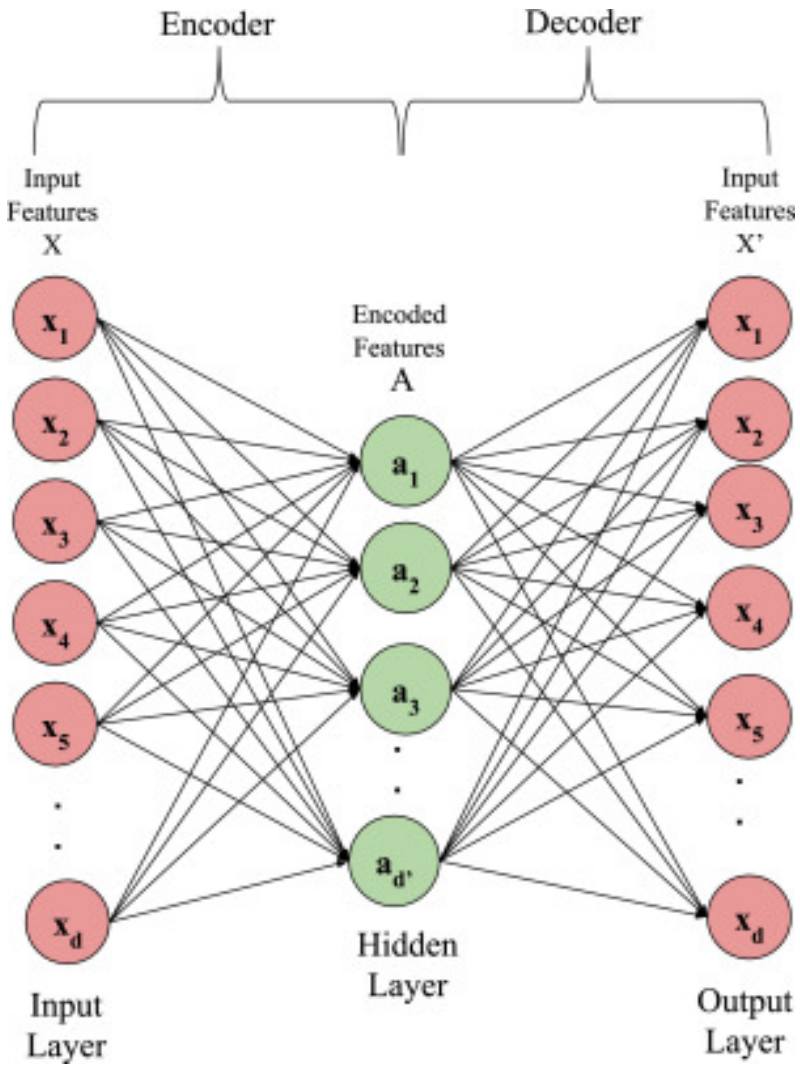
Autoencoder Architecture



- **Encoder**: outputs a lower dimensional representation ***z*** of the data ***x*** (similar to PCA, tSNE...)
- **Decoder**: Learns how to reconstruct ***x*** given ***z:*** learns *p(x|**z**)*

Autoencoder Architecture

https://link.springer.com/chapter/10.1007/978-981-13-6661-1_3



Building a DNN
with keras and tensorflow

Trivial to build, but the devil is in the details!

Building a DNN with keras and tensorflow

Trivial to build, but the devil is in the details!

```
from keras.models import Sequential
#can upload pretrained models from keras.models
from keras.layers import Dense,  Conv2D, MaxPooling2D
#create model
model = Sequential()


#create the model architecture by adding model layers
model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1))

#need to choose the loss function, metric, optimization scheme
model.compile(optimizer='adam', loss='mean_squared_error')

#need to learn what to look for - always plot the loss function!
model.fit(x_train, y_train, validation_data=(x_test, y_test),
                    epochs=20, batch_size=100, verbose=1)
#note that the model allows to give a validation test,
#this is for a 3fold cross valiation: train-validate-test
#predict
```

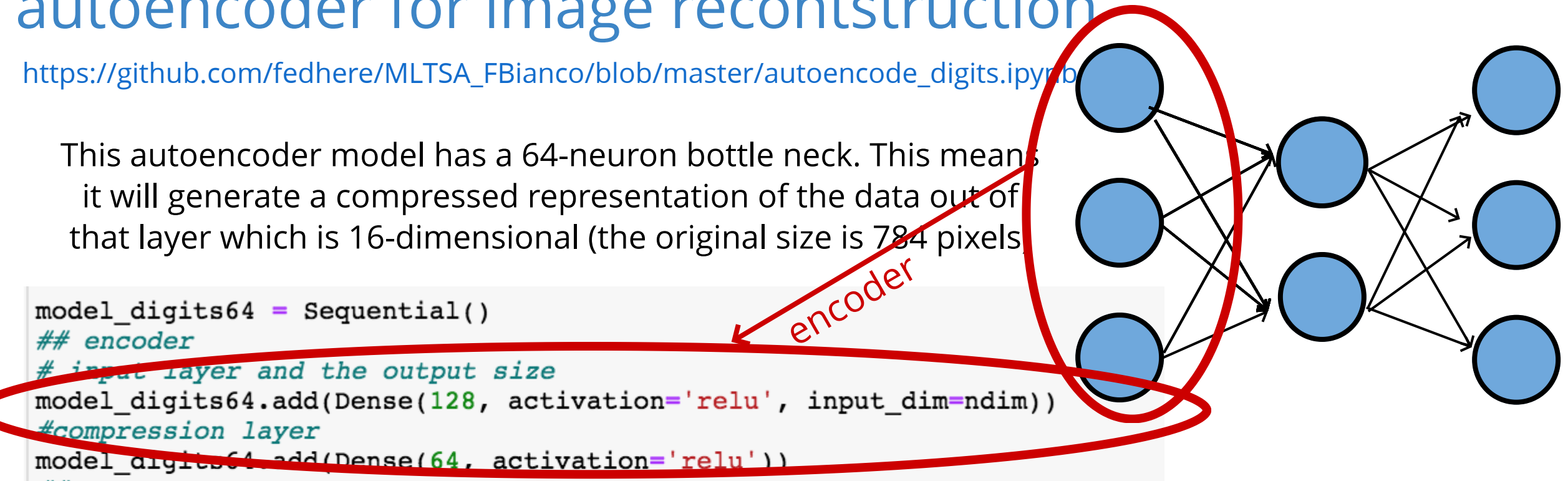
Building a DNN with keras and tensorflow

autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

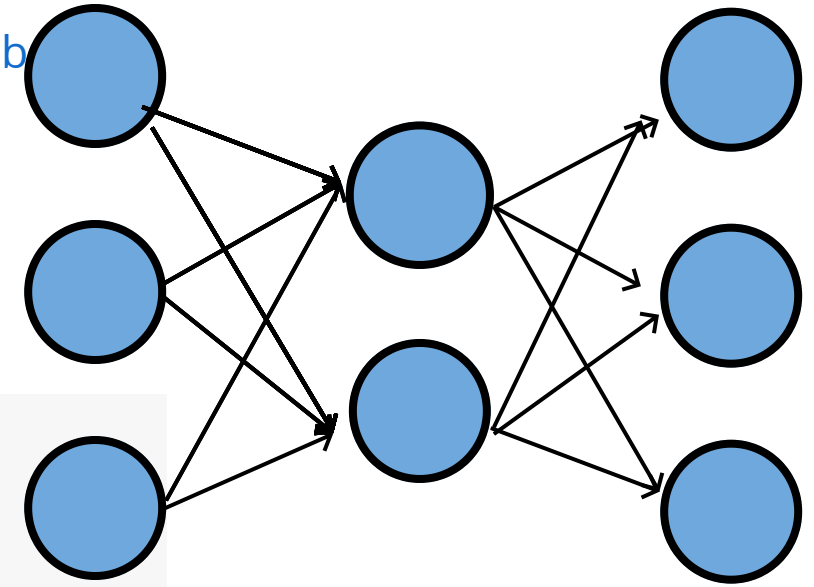
This autoencoder model has a 64-neuron bottle neck. This means it will generate a compressed representation of the data out of that layer which is 16-dimensional (the original size is 784 pixels)

```
model_digits64 = Sequential()
## encoder
# input layer and the output size
model_digits64.add(Dense(128, activation='relu', input_dim=ndim))
#compression layer
model_digits64.add(Dense(64, activation='relu'))
## deencoder
#decompression layer, same size as in the encoder
model_digits64.add(Dense(128, activation='relu'))
#output layer, same size as input
model_digits64.add(Dense(ndim, activation='linear'))
```



Building a DNN with keras and tensorflow autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb



This autoencoder model has a 64-neuron bottle neck. This means it will generate a compressed representation of the data out of that layer which is 16-dimensional (the original size is 784 pixels)

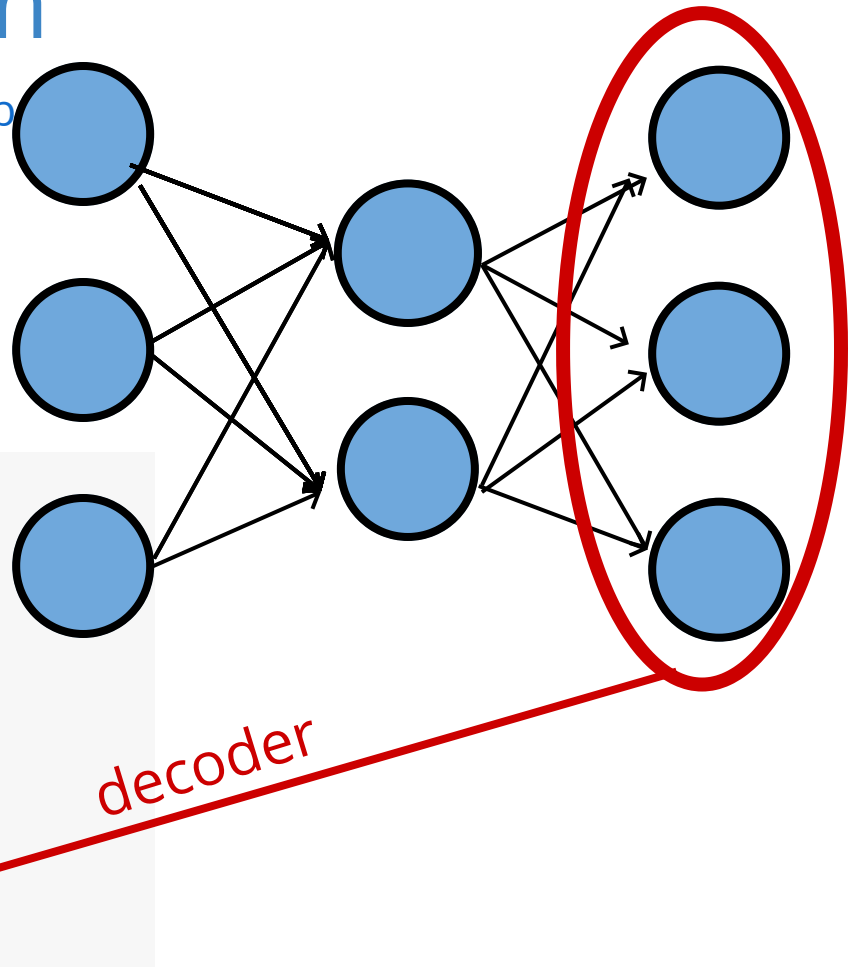
```
model_digits64 = Sequential()
## encoder
# input layer and the output size
model_digits64.add(Dense(128, activation='relu', input_dim=ndim))
#compression layer
model_digits64.add(Dense(64, activation='relu'))
## deencoder
#decompression layer, same size as in the encoder
model_digits64.add(Dense(128, activation='relu'))
#output layer, same size as input
model_digits64.add(Dense(ndim, activation='linear'))
```

Building a DNN with keras and tensorflow autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

This autoencoder model has a 64-neuron bottle neck. This means it will generate a compressed representation of the data out of that layer which is 16-dimensional (the original size is 784 pixels)

```
model_digits64 = Sequential()
## encoder
# input layer and the output size
model_digits64.add(Dense(128, activation='relu', input_dim=ndim))
#compression layer
model_digits64.add(Dense(64, activation='relu'))
## deencoder
#decompression layer, same size as in the encoder
model_digits64.add(Dense(128, activation='relu'))
#output layer, same size as input
model_digits64.add(Dense(ndim, activation='linear'))
```

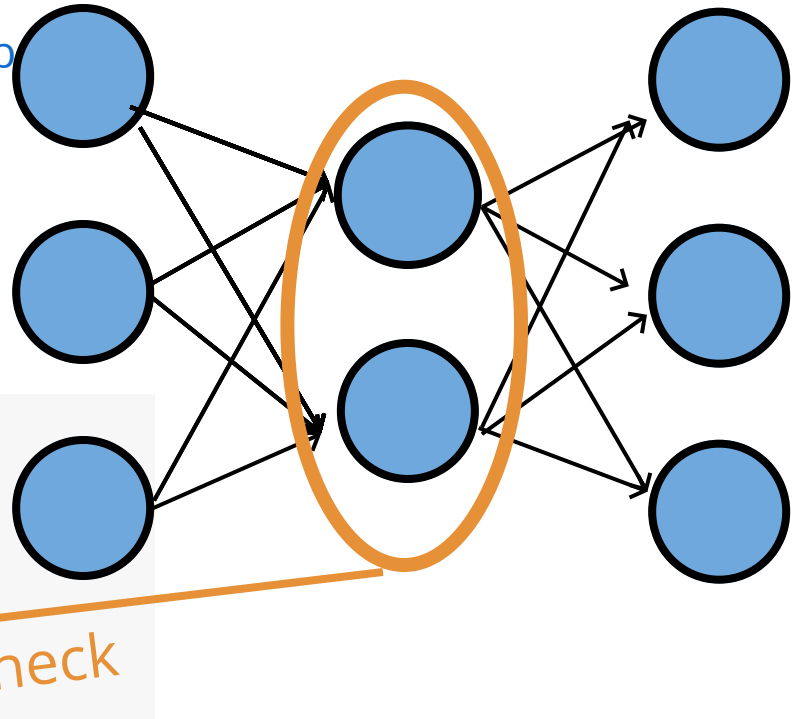


Building a DNN with keras and tensorflow autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

This autoencoder model has a 64-neuron bottle neck. This means it will generate a compressed representation of the data out of that layer which is 16-dimensional (the original size is 784 pixels)

```
model_digits64 = Sequential()
## encoder
# input layer and the output size
model_digits64.add(Dense(128, activation='relu', input_dim=ndim))
#compression layer
model_digits64.add(Dense(64, activation='relu'))
## deencoder
#decompression layer, same size as in the encoder
model_digits64.add(Dense(128, activation='relu'))
#output layer, same size as input
model_digits64.add(Dense(ndim, activation='linear'))
```



Building a DNN with keras and tensorflow autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

```
# choose the optimizer and loss appropriately!
model_digits64.compile(optimizer="adadelta", loss="mean_squared_error")
```

```
print(model_digits64.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_4 (Dense)              (None, 784)               101136
=================================================================
Total params: 218,192
Trainable params: 218,192
Non-trainable params: 0
_________________________________________________________________
None
```

This simple model has 200K parameters!

My original choice is to train it with "adadelta" with a mean squared loss function, all activation functions are relu, appropriate for a linear regression

Building a DNN

with keras and tensorflow

autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

3.0.1 regression

- loss='mean_squared_error' L2: default loss to use for regression problems. => linear activation funct in output layer, one node out

alternatives: loss='mean_squared_logarithmic_error', 'mean_absolute_error' (which is L1 instead of L2)

3.0.2 binary classification

- loss='binary_crossentropy' => sigmoid activation function in output layer, one node out

alternatives: 'hinge'

3.0.3 multiclass classification

categorical encoded as numerical

- loss='categorical_crossentropy' => softmax n nodes out

onehot encoded categoridal

- 'parse_categorical_crossentropy' => softmax n nodes out
- 'kullback Leibler Divergence Loss' => probabilistic categorical classification; log(P/Q)

What should I choose for the loss function and how does that relate to the activation functiom and optimization?

Building a DNN

with keras and tensorflow

autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

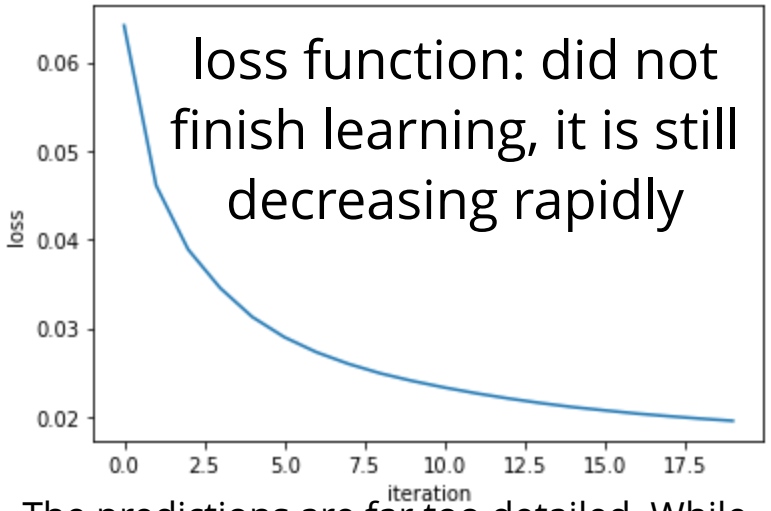
What should I choose for the loss function and how does that relate to the activation functiom and optimization?

loss	**good for**	**activation last layer**	**size last layer**
mean_squared_error	regression	linear	one node
mean_absolute_error	regression	linear	one node
mean_squared_logarithmit_error	regression	linear	one node
binary_crossentropy	binary classification	sigmoid	one node
categorical_crossentropy	multiclass classification	sigmoid	N nodes
Kullback_Divergence	multiclass classification, probabilistic inerpretation	sigmoid	N nodes

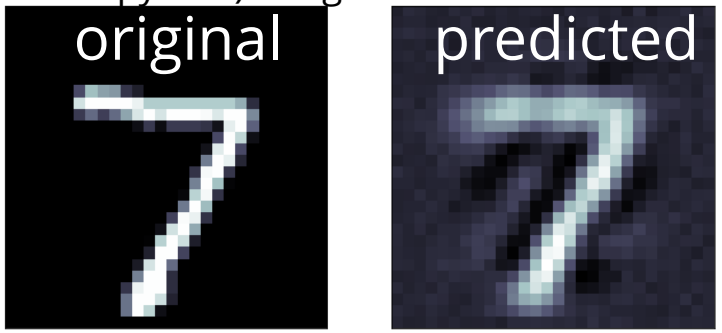
autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

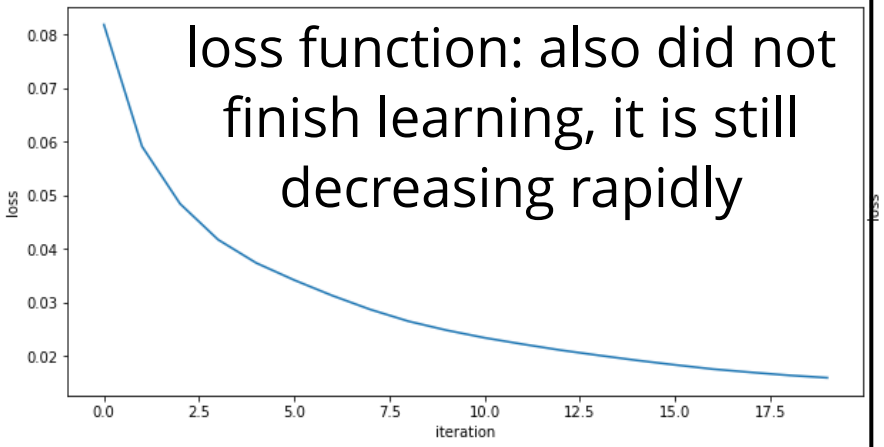
```
model_digits64.add(Dense(ndim,
                         activation='linear'))
model_digits64_sig.compile(optimizer="adadelta",
                    loss="mean_squared_error")
```



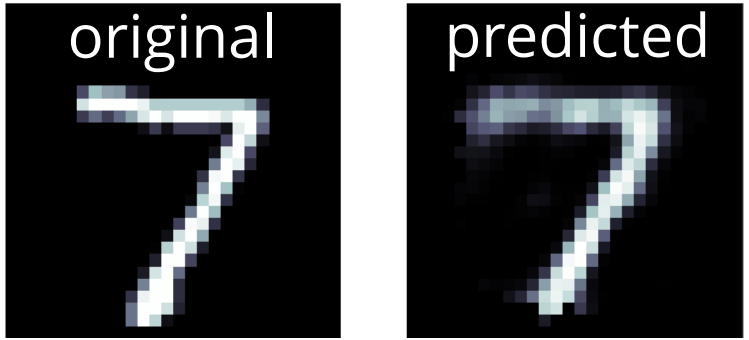
The predictions are far too detailed. While the input is not binary, it does not have a lot of details. Maybe approaching it as a binary problem (with a sigmoid and a binary cross entropy loss) will give better results



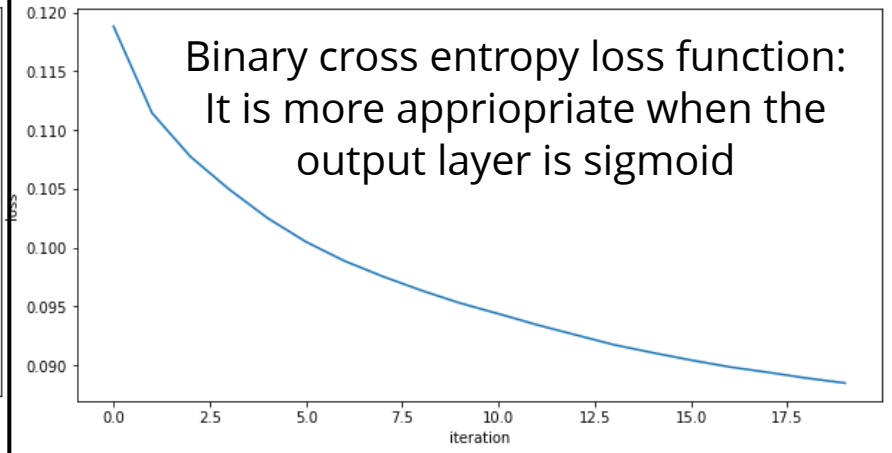

```
model_digits64_sig.add(Dense(ndim,
                             activation='sigmoid'))
model_digits64_sig.compile(optimizer="adadelta",
                           loss="mean_squared_error")
```



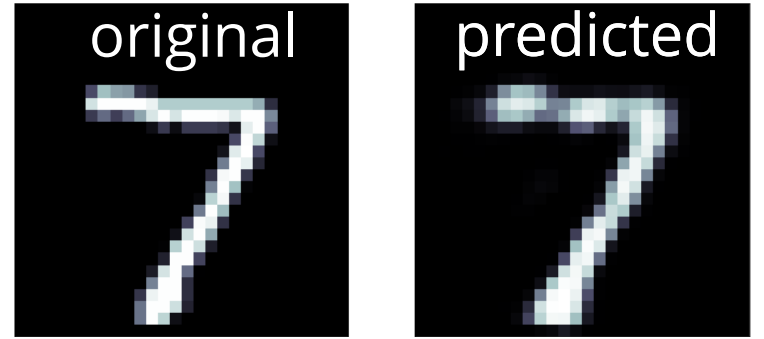
A sigmoid gives activation gives a much better result!




```
model_digits64_sig.add(Dense(ndim,
                             activation='sigmoid'))
model_digits64_bce.compile(optimizer="adadelta",
                           loss="binary_crossentropy")
```



Even better results!



autoencoder for image recontstruction

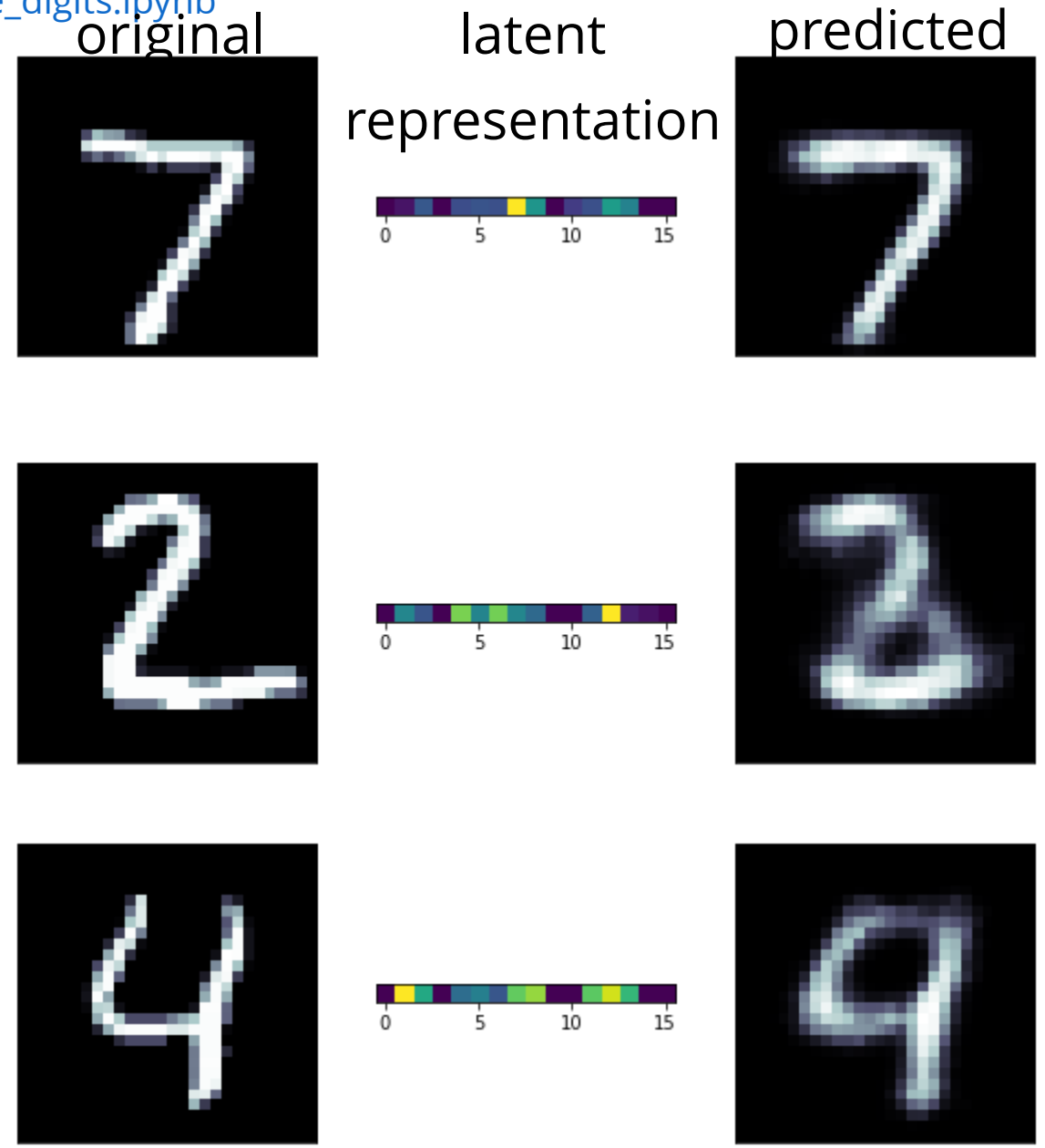
https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

A more ambitious model has a 16 neurons bottle neck: we are trying to extract 16 numbers to reconstruct the entire image! its pretty remarcable! those 16 number are ***extracted features*** from the data

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 128)	100480
dense_10 (Dense)	(None, 64)	8256
dense_11 (Dense)	(None, 32)	2080
dense_12 (Dense)	(None, 16)	528
dense_13 (Dense)	(None, 32)	544
dense_14 (Dense)	(None, 64)	2112
dense_15 (Dense)	(None, 128)	8320
dense_16 (Dense)	(None, 784)	101136

Total params: 223,456
Trainable params: 223,456
Non-trainable params: 0



models are neutral, the bias is in the data (or is it?)

Why does this AI model whitens Obama face?

Simple answer: the data is biased. The algorithm is fed more images of white people

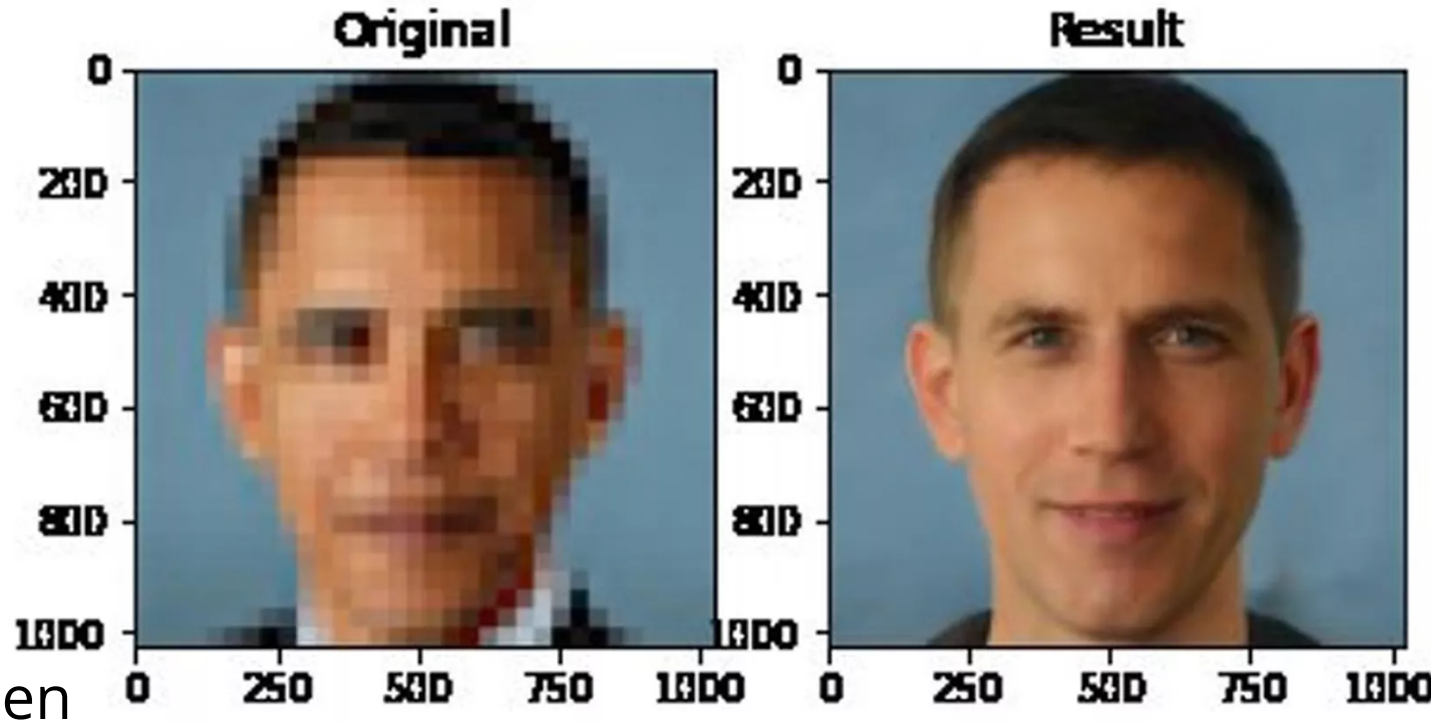
https://www.theverge.com/21298762/face-depixelizer-ai-machine-learning-tool-pulse-stylegan-obama-bias

models are neutral, the bias is in the data (or is it?)

Why does this AI model whitens Obama face?

Simple answer: the data is biased. The algorithm is fed more images of white people

But really, would the opposite have been acceptable? *The bias is in society*



https://www.theverge.com/21298762/face-depixelizer-ai-machine-learning-tool-pulse-stylegan-obama-bias

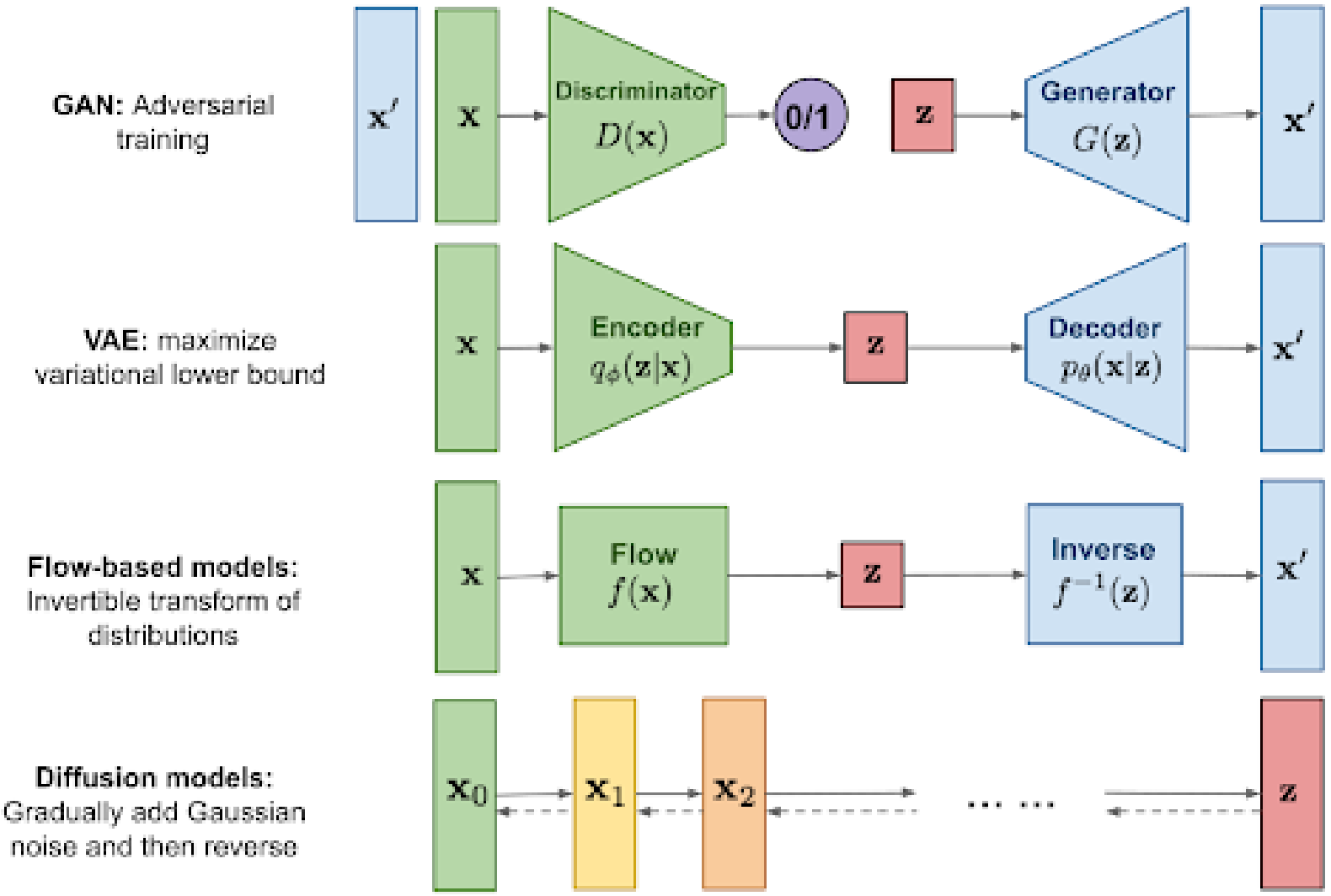
models are neutral, the bias is in the data (or is it?)

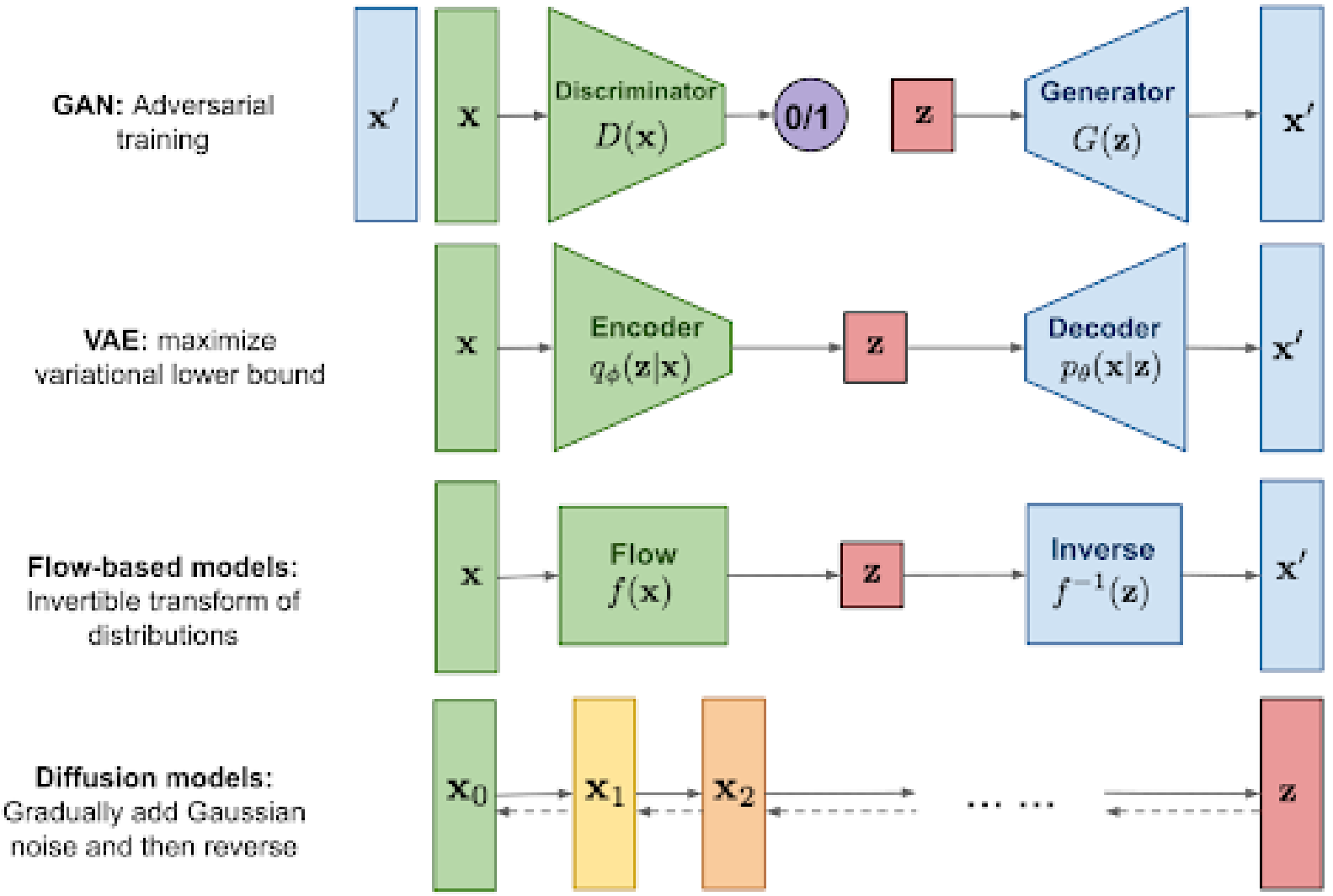
Joy Boulamwini

3

comparing generative AI models

see also
https://arxiv.org/pdf/2103.04922.pdf





The latent space is assumed to be Gaussian distributed - this causes inaccuracy (blurry) generation

Normalizing Flows

BOND-TAYLOR ET AL.: DEEP GENERATIVE MODELLING: A COMPARATIVE REVIEW

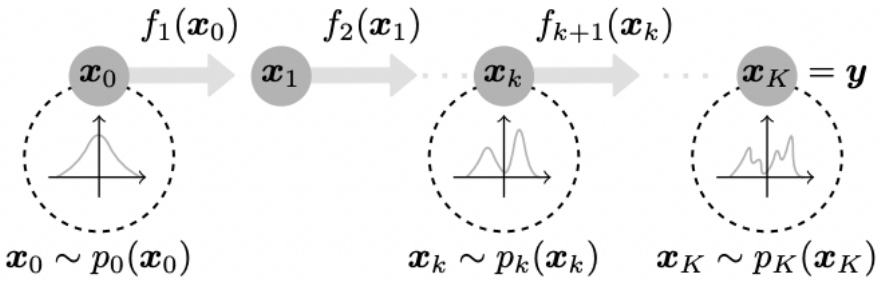
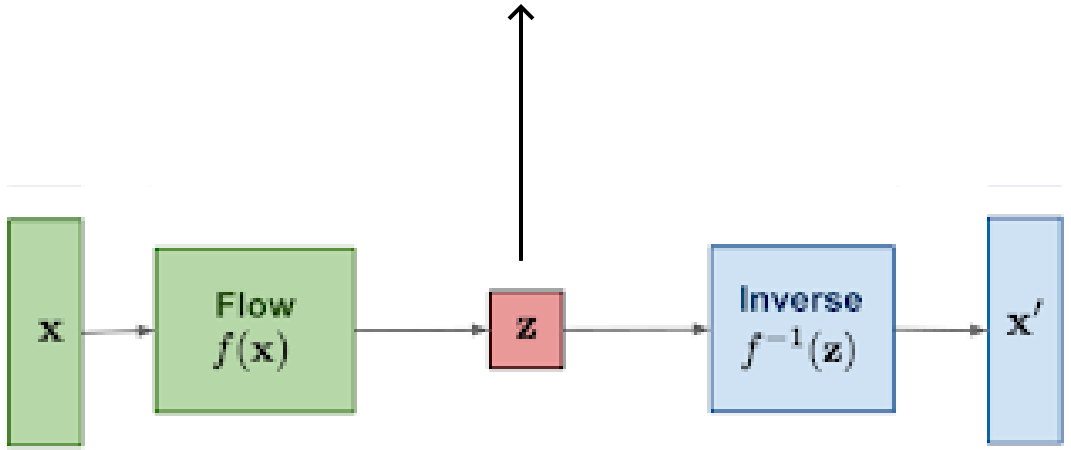


Fig. 7: Normalizing flows build complex distributions by mapping a simple distribution through invertible functions.

Flow-based models:
Invertible transform of distributions



The latent space is assumed to be Gaussian distributed - this causes inaccuracy (blurry) generation

similar to a VAE but with a NN in the middle that approximates the true distribution of the latent space

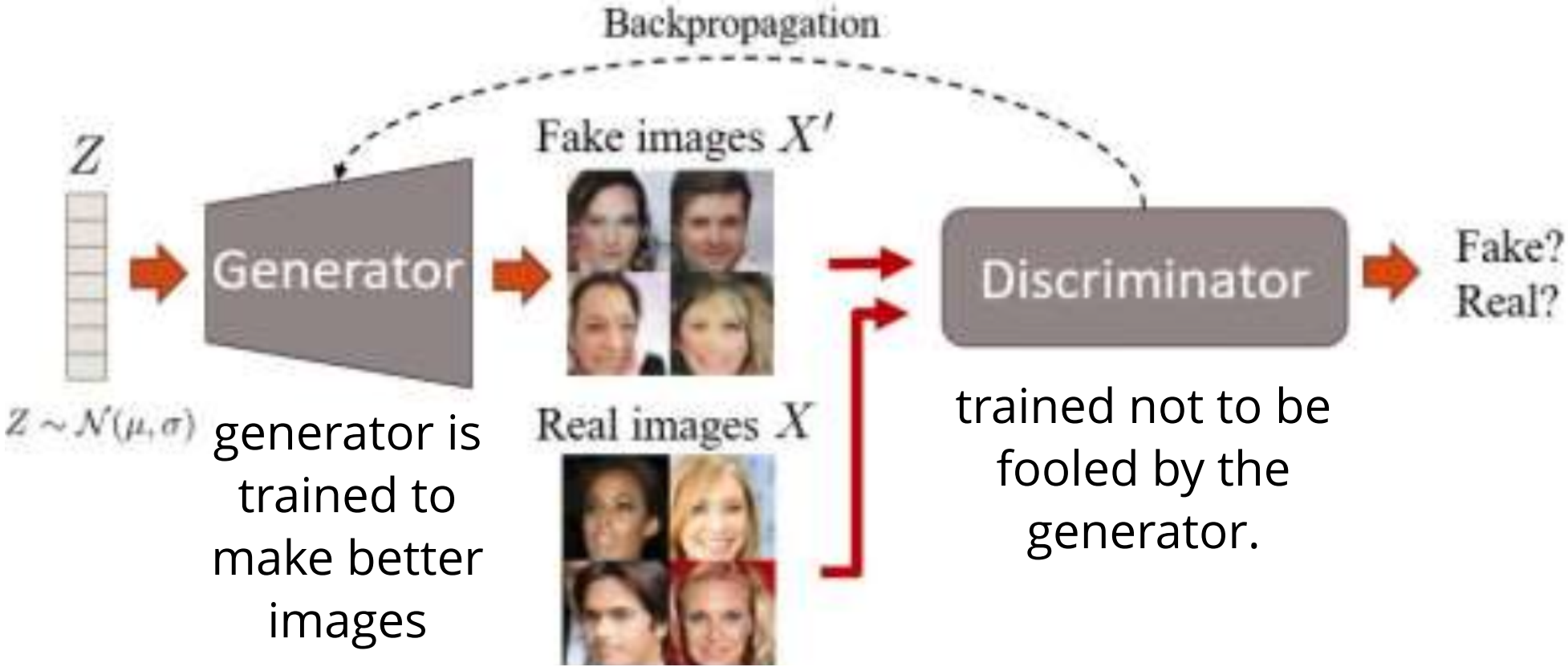
GANs: Generative Adversarial NN

Ian Goodfellow et al., 2014
Generative Adversarial Networks

have two networks trained at the same time that compete again each other in a minimax game.

The **generator** generates images, starting with pure noise.

The **discriminator** classifies the image from the generator as Real/Fake



GANs: Generative Adversarial NN

Minmax Loss Function: $E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$

https://danieltakeshi.github.io/2017/03/05/understanding-generative-adversarial-networks/

minimize

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

maximize

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

generator is trained to make better images

trained not to be fooled by the generator.

GANs: Generative Adversarial NN

change introduced to minimize geneerator saturation

Minmax Loss Function:

https://danieltakeshi.github.io/2017/03/05/understanding-generative-adversarial-networks/

log(D(G(z)))

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

minimize

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

maximize

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

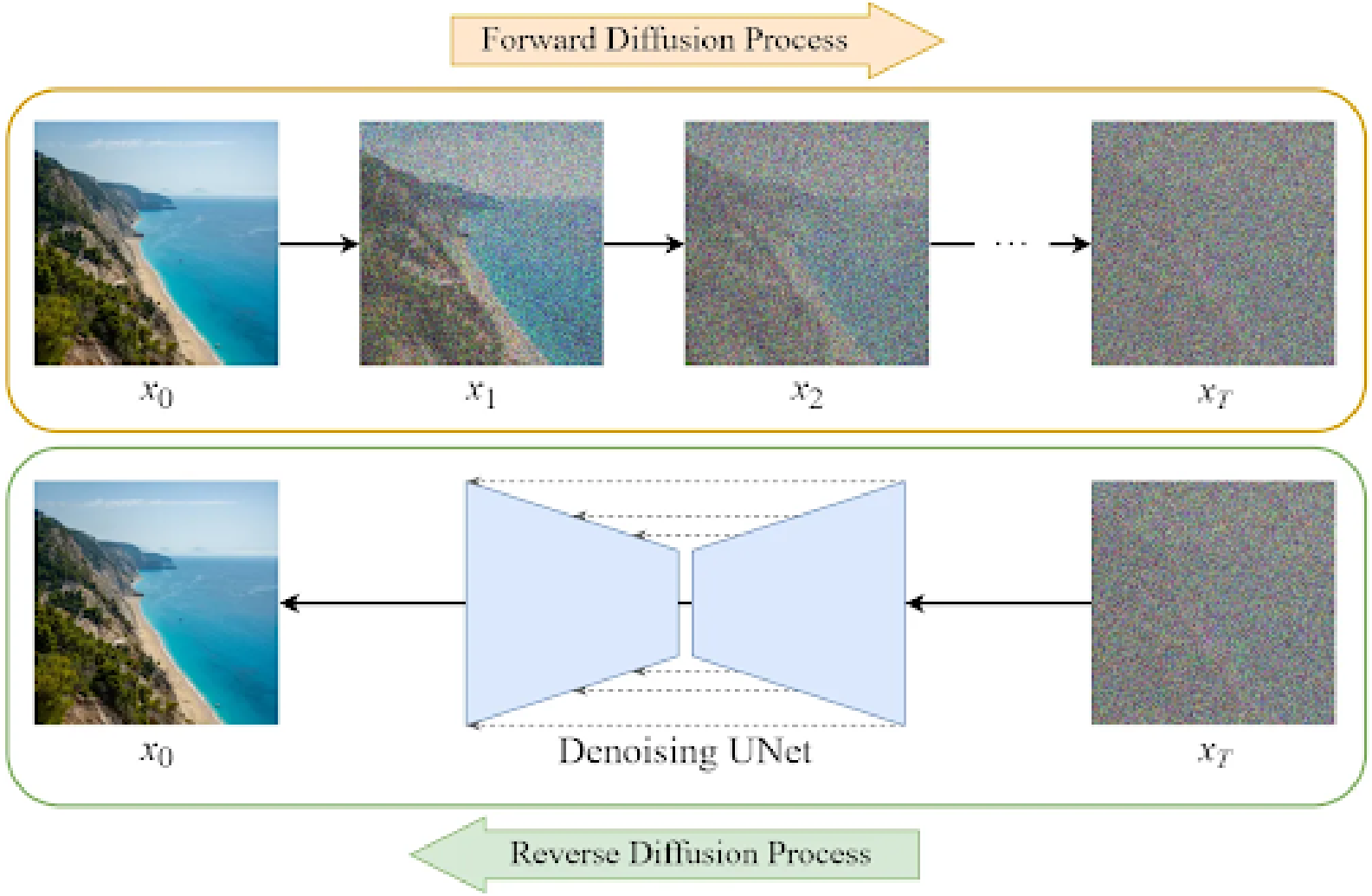
generator is trained to make better images

trained not to be fooled by the generator.

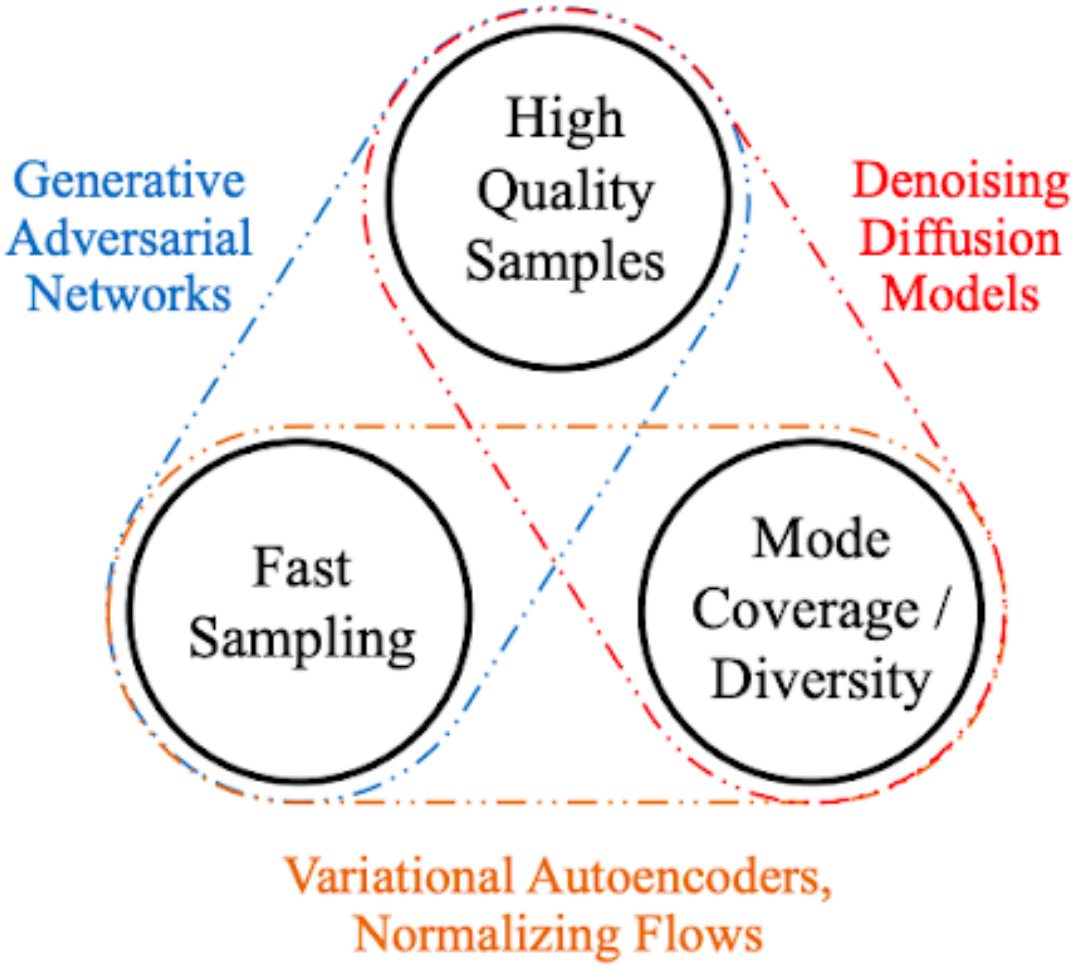
DDPM:Denoising Diffusion Probabilistic Model

Ho Jain Abbel 2006

https://arxiv.org/abs/2006.11239



Which generative AI is right for you??



Neural Network and Deep Learning

an excellent and free book on NN and DL

http://neuralnetworksanddeeplearning.com/index.html

Deep Learning An MIT Press book in preparation

Ian Goodfellow, Yoshua Bengio and Aaron Courville

https://www.deeplearningbook.org/lecture_slides.html

History of NN

https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html