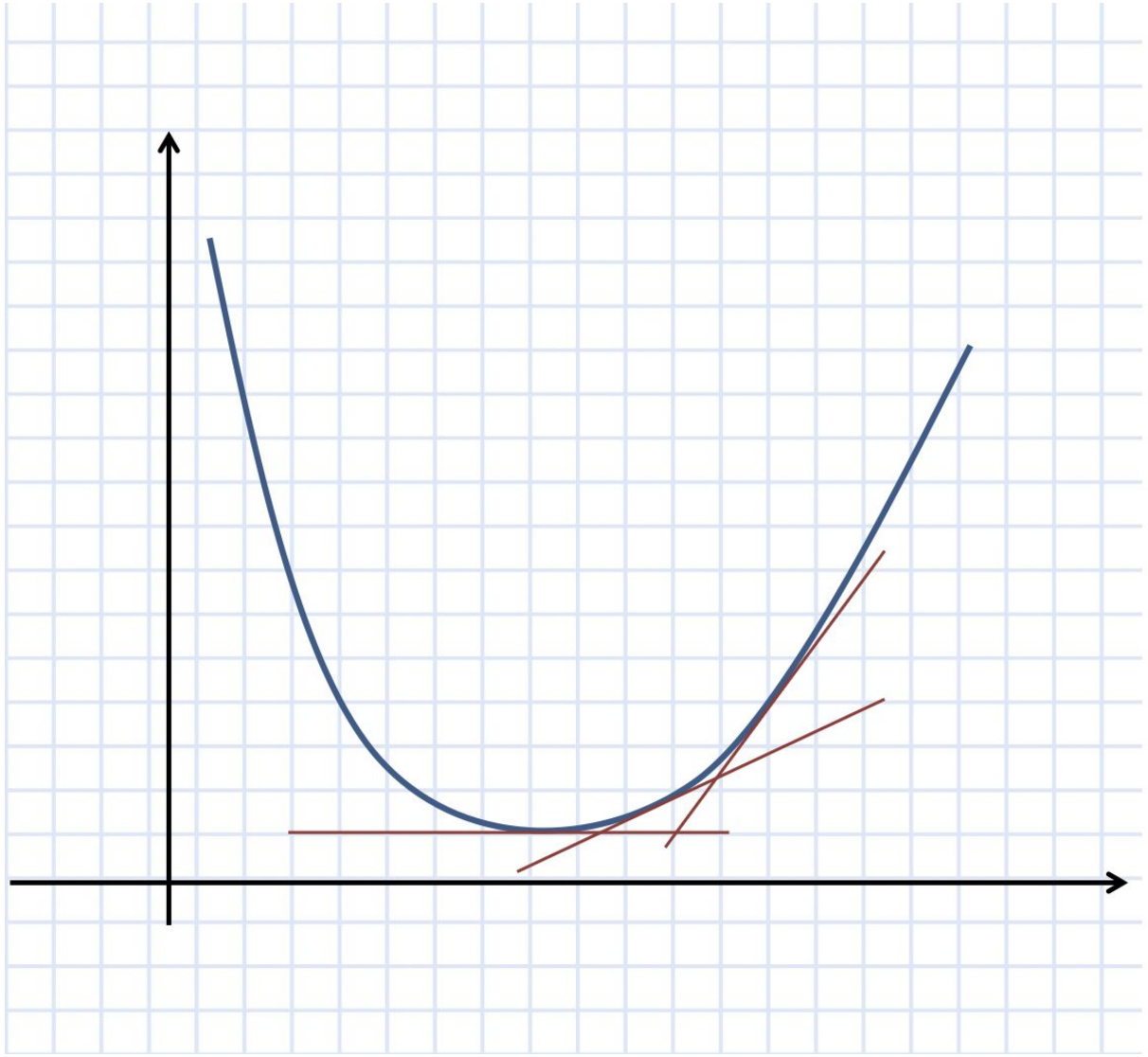
Deep Learning: Session 6

Deep Neural Nets

Outline

1. Recap
2. Local Minima
3. Optimization
4. Hyperparams Tuning

In previous chapters...



1. Pick a random w (w^0)
2. Repeat until convergence {

$$w^{i+1} = w^i - \alpha dL(w)/d(w)_{[w^i]}$$

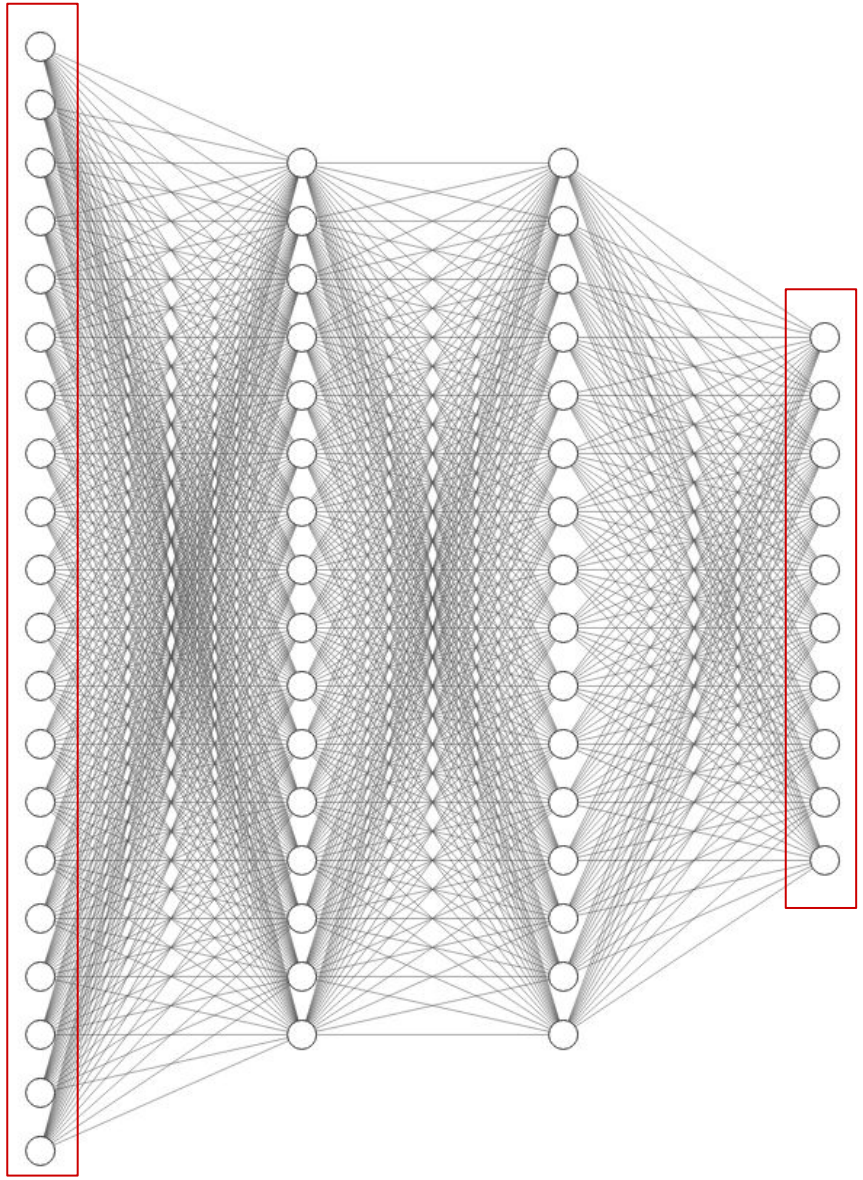
}

In previous chapters...

- Backpropagation (BP) is the smart and elegant way to apply gradient descent in multilayer neural nets.
- It is working by ‘propagating’ the error backwards (from the output layer to the input one) layer by layer.
- All the trainable params, weights and bias are contributing to the error. By using BP, we can assign to all of them their specific contribution to the final error.
- Mathematically, BP exploits the chain rule of the derivatives.

In previous chapters...

784 pixels



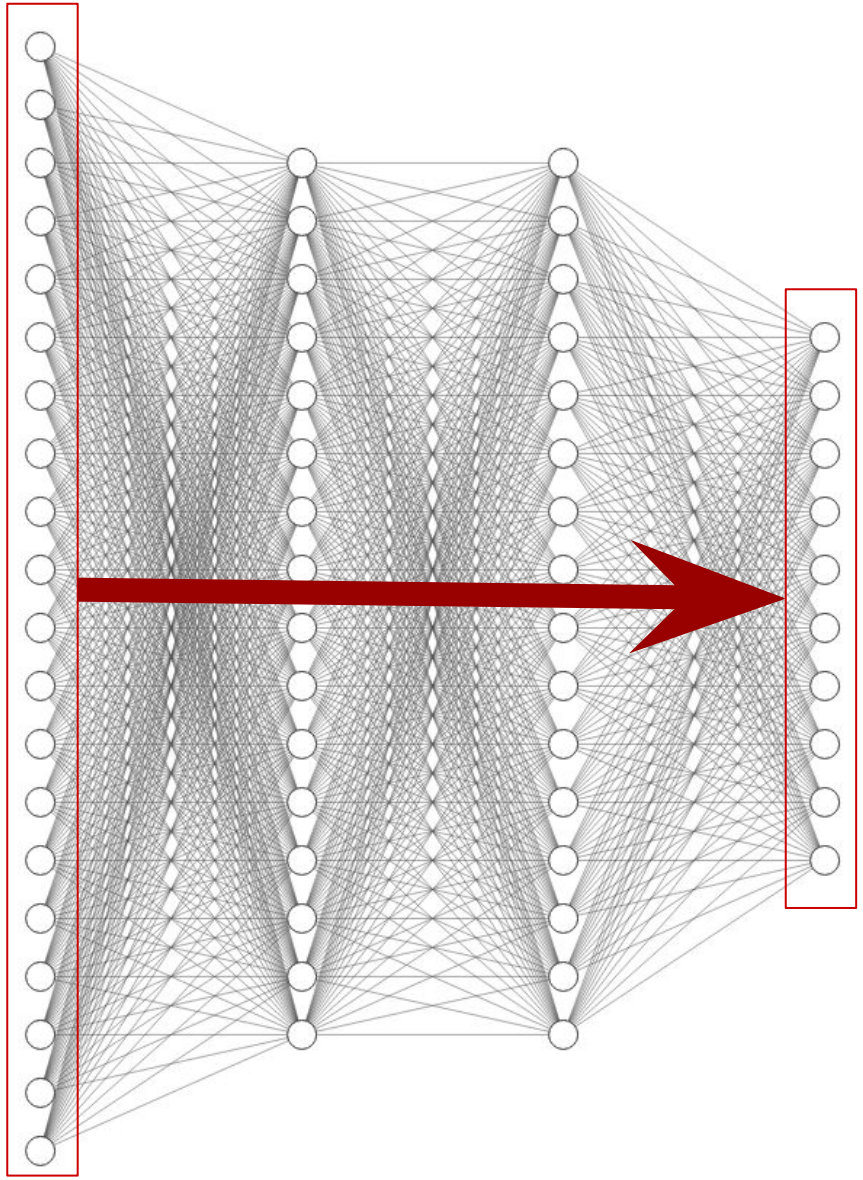
Input Layer: 784 neurons

Output Layer: 10 neurons.

How can we apply Gradient Descent?

In previous chapters...

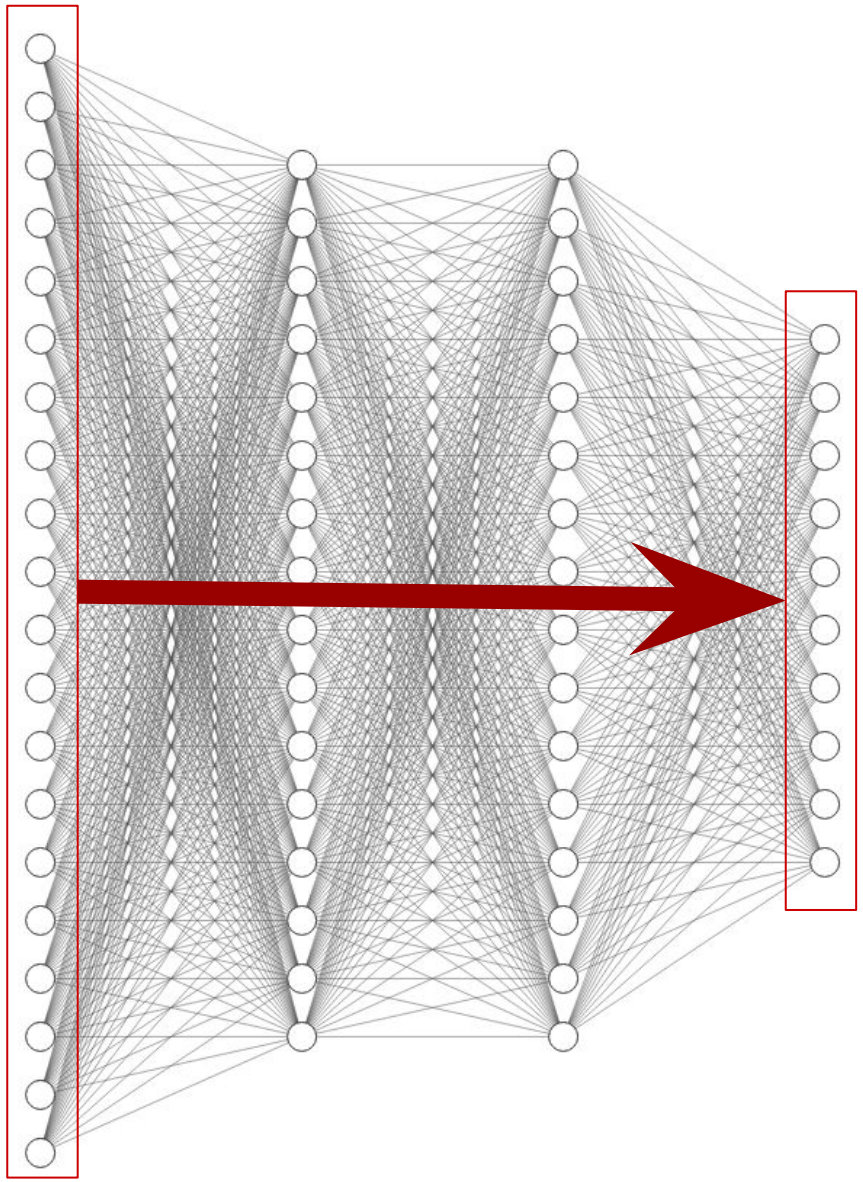
784 pixels



0.1	0
0.12	0
0	0
0.21	1
0.1	0
0.14	0
..	..
..	..
..	..
..	..
0.11	0

In previous chapters...

784 pixels

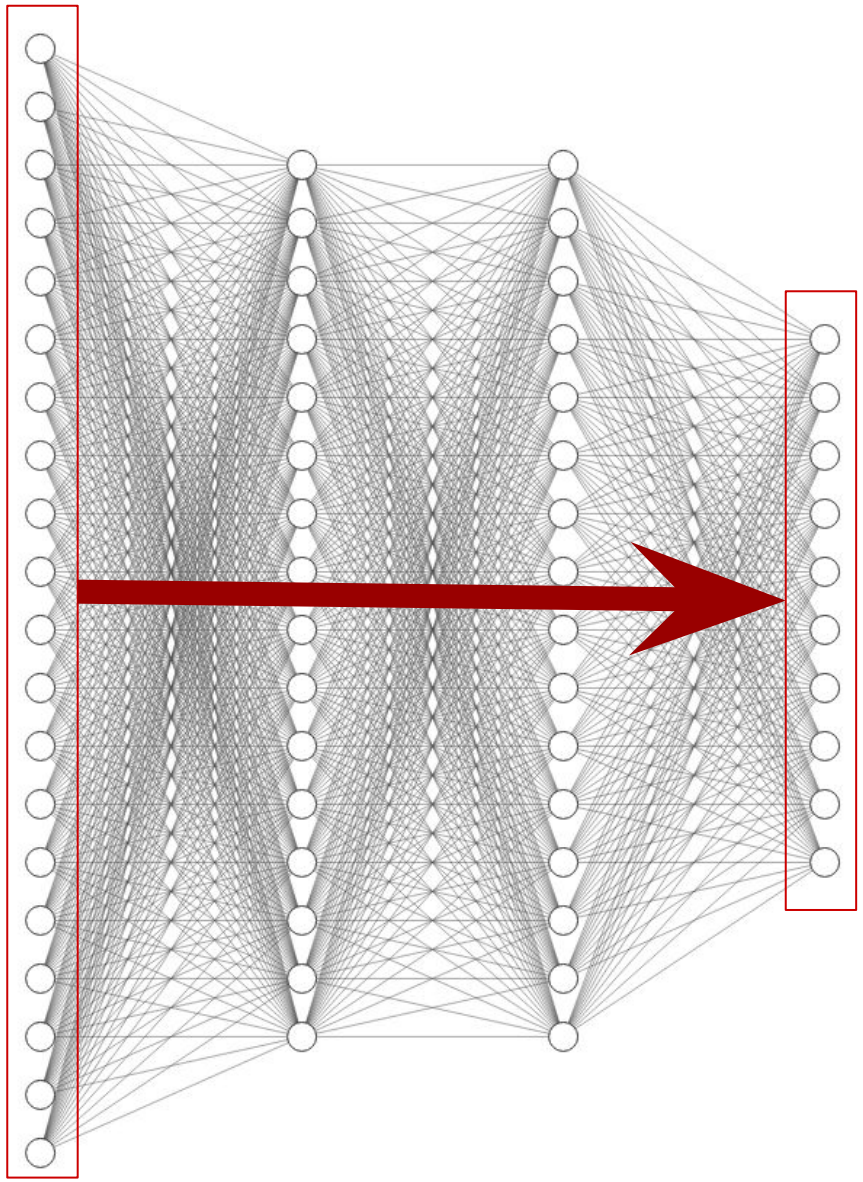


At a high level:

- Compute Error: Cost Function
- Compute the negative of the gradient for every weight of the net on every input-output pair
- Modify Weights
- Repeat

In previous chapters...

784 pixels

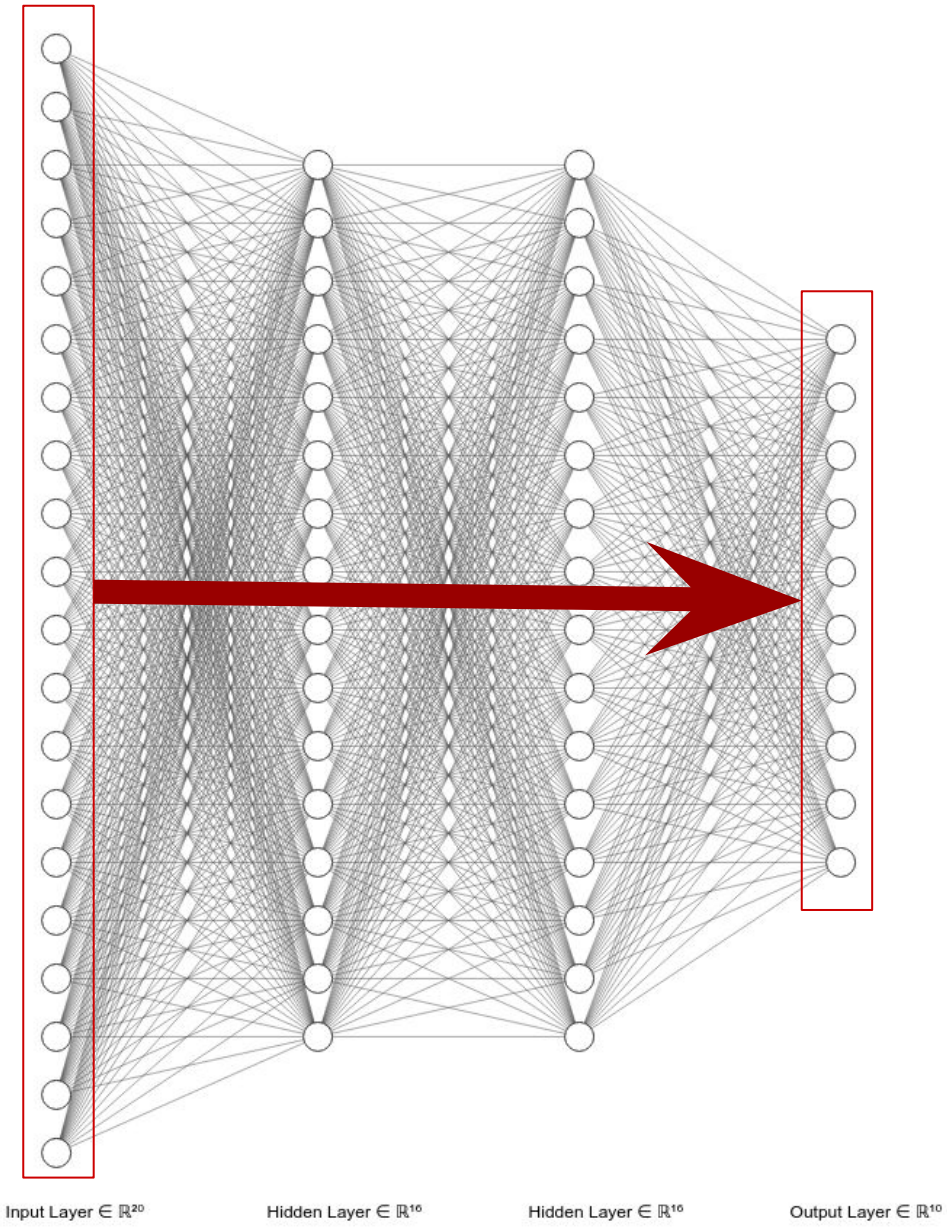


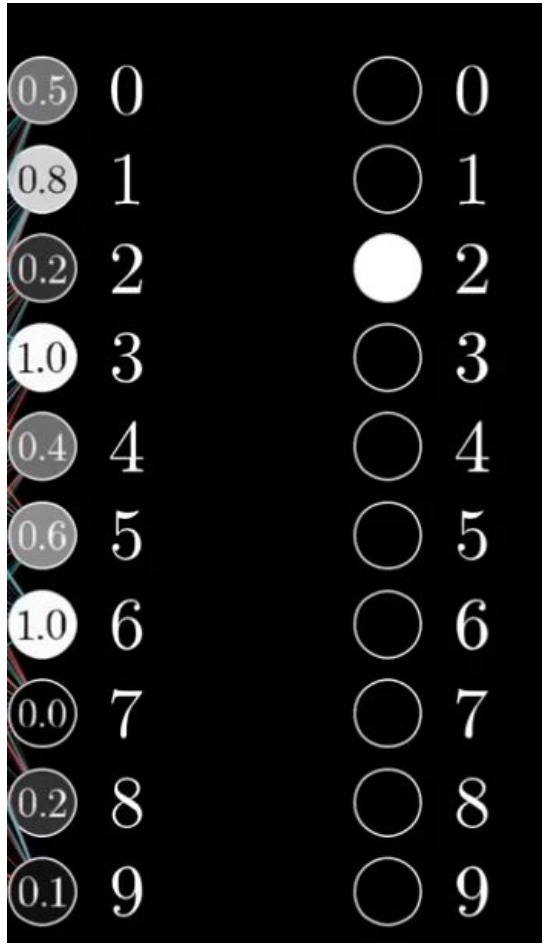
At a high level:

- How can we modify a neuron activation?
  - Weights
  - Previous Activations
- Can we modify Previous Activations?

In previous chapters...

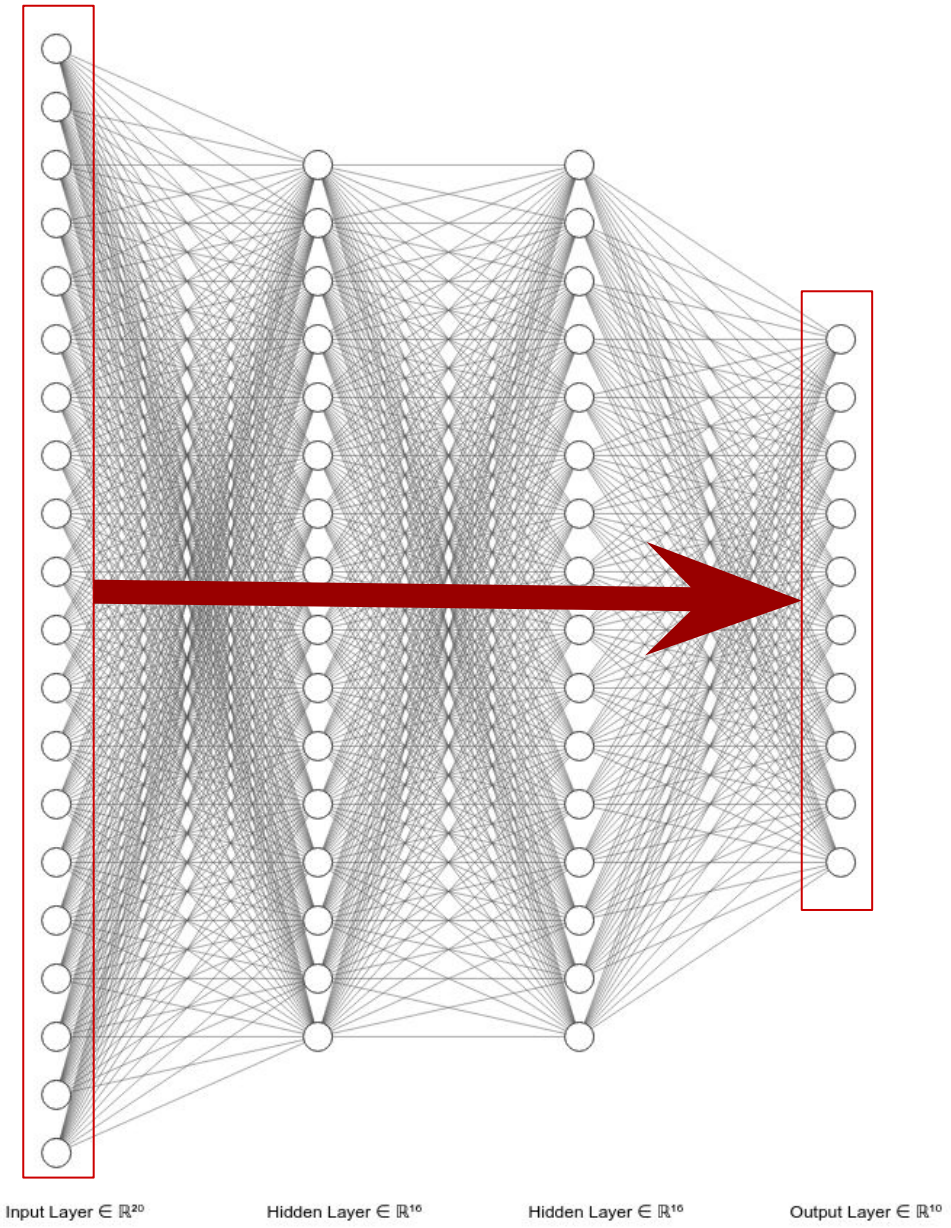
784 pixels

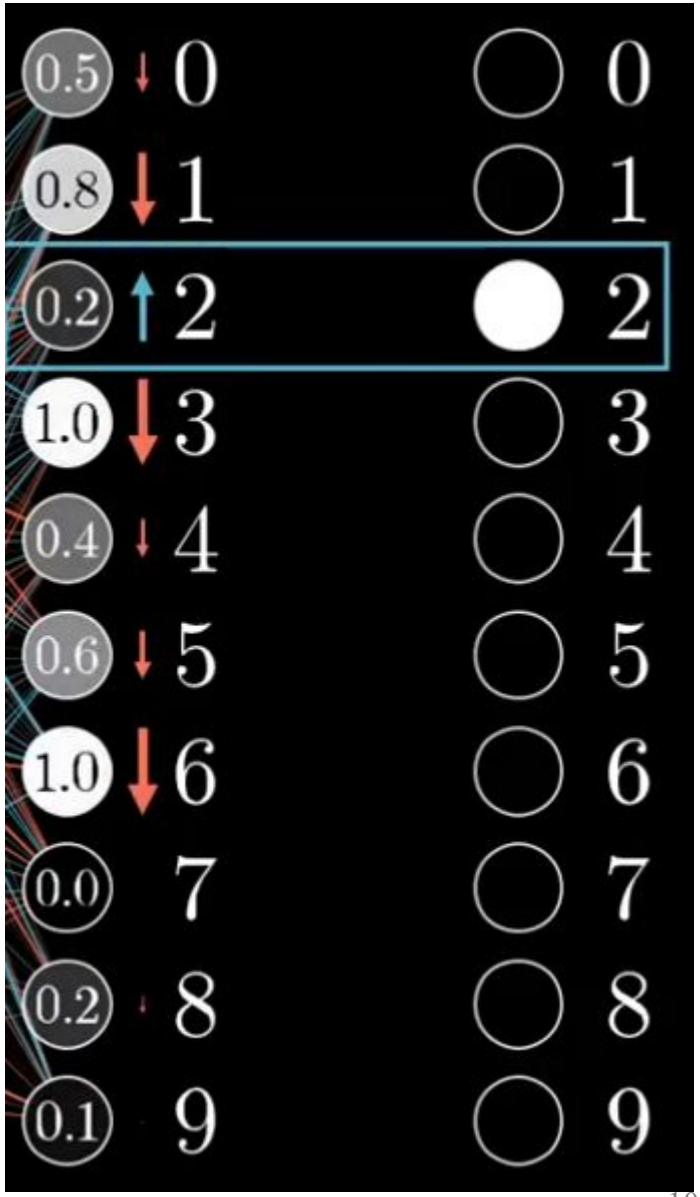




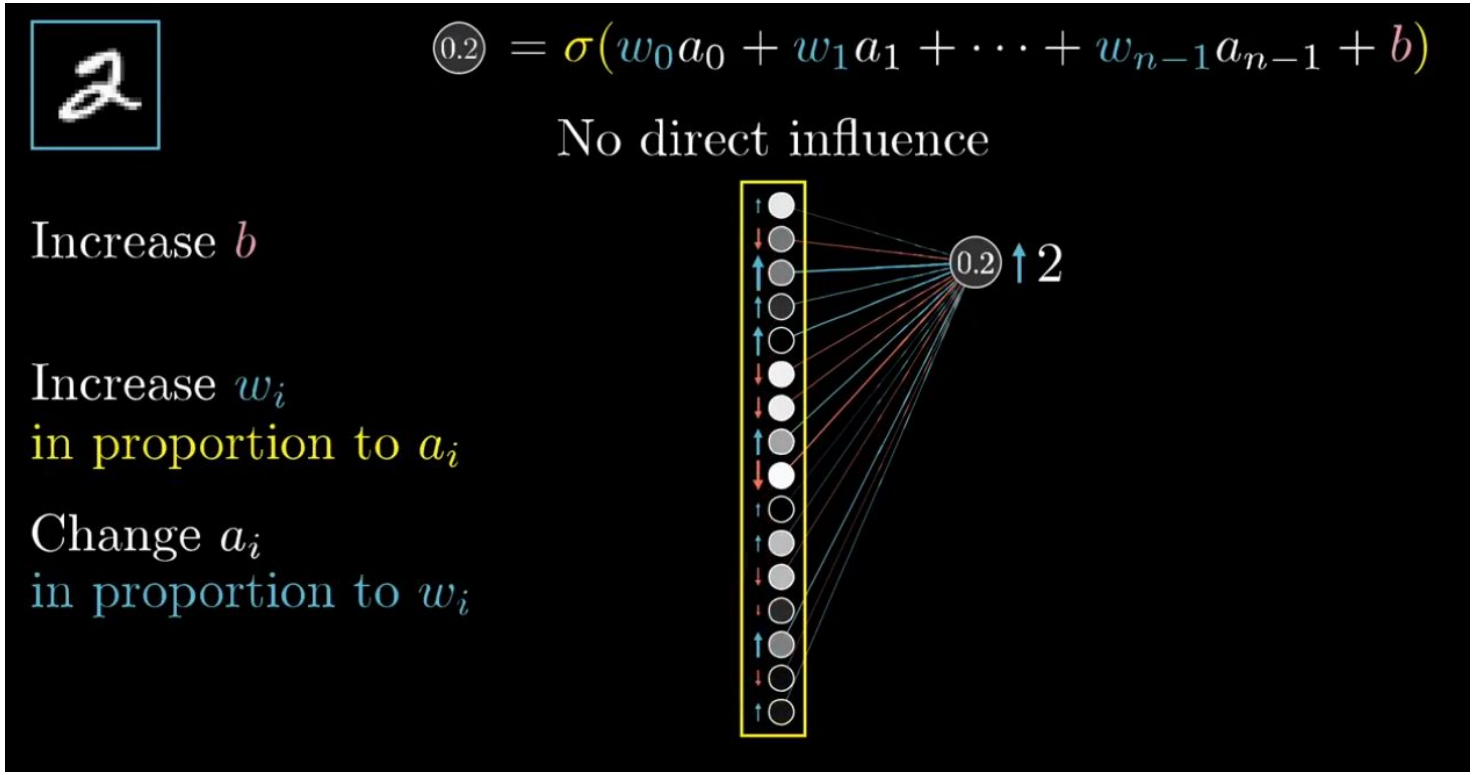
In previous chapters...

784 pixels





In previous chapters...



Changing weights:

- Weights connecting neurons with higher values have more impact

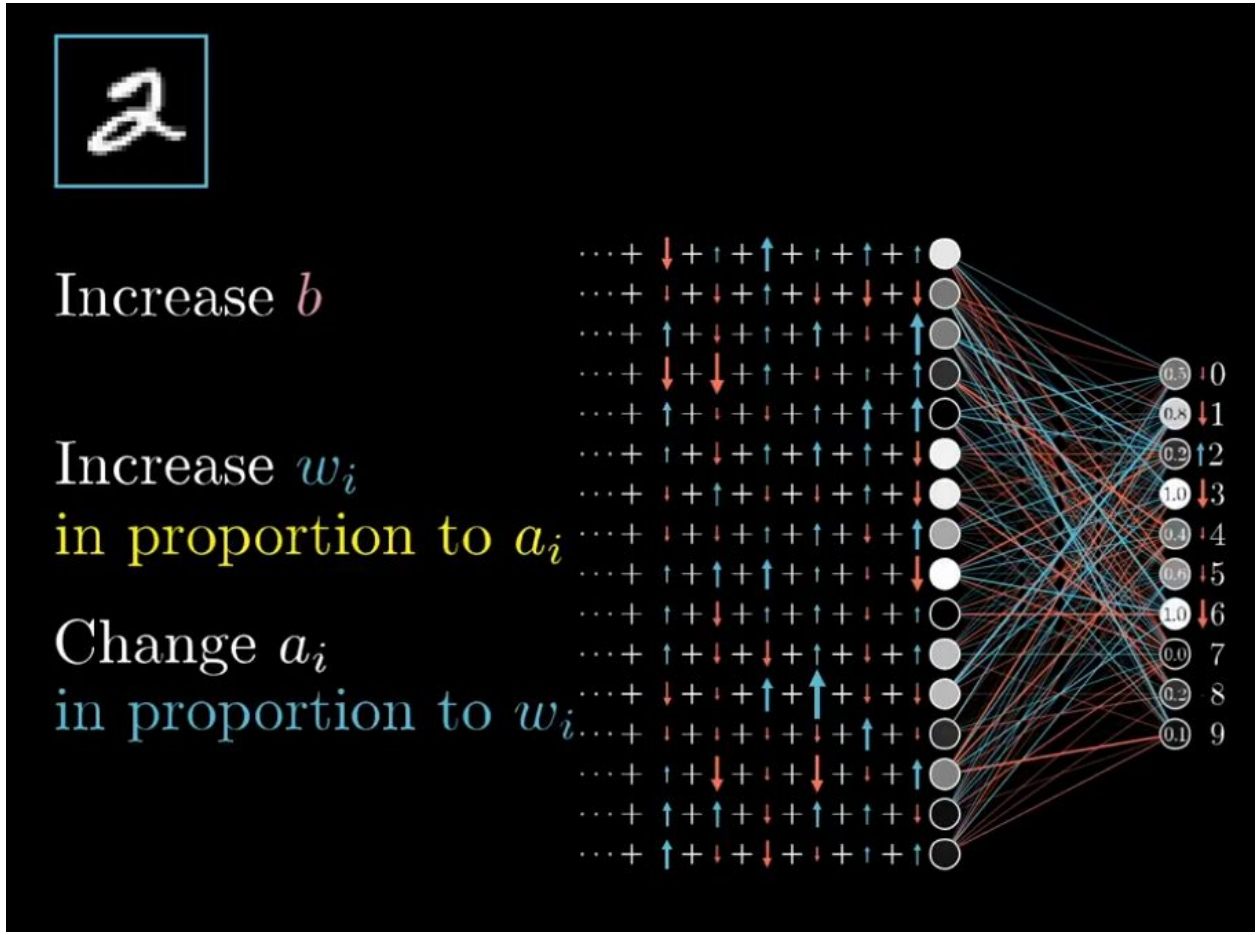
Changing bias:

- Affects all data, cannot be adjusted for a single input

Changing Activations:

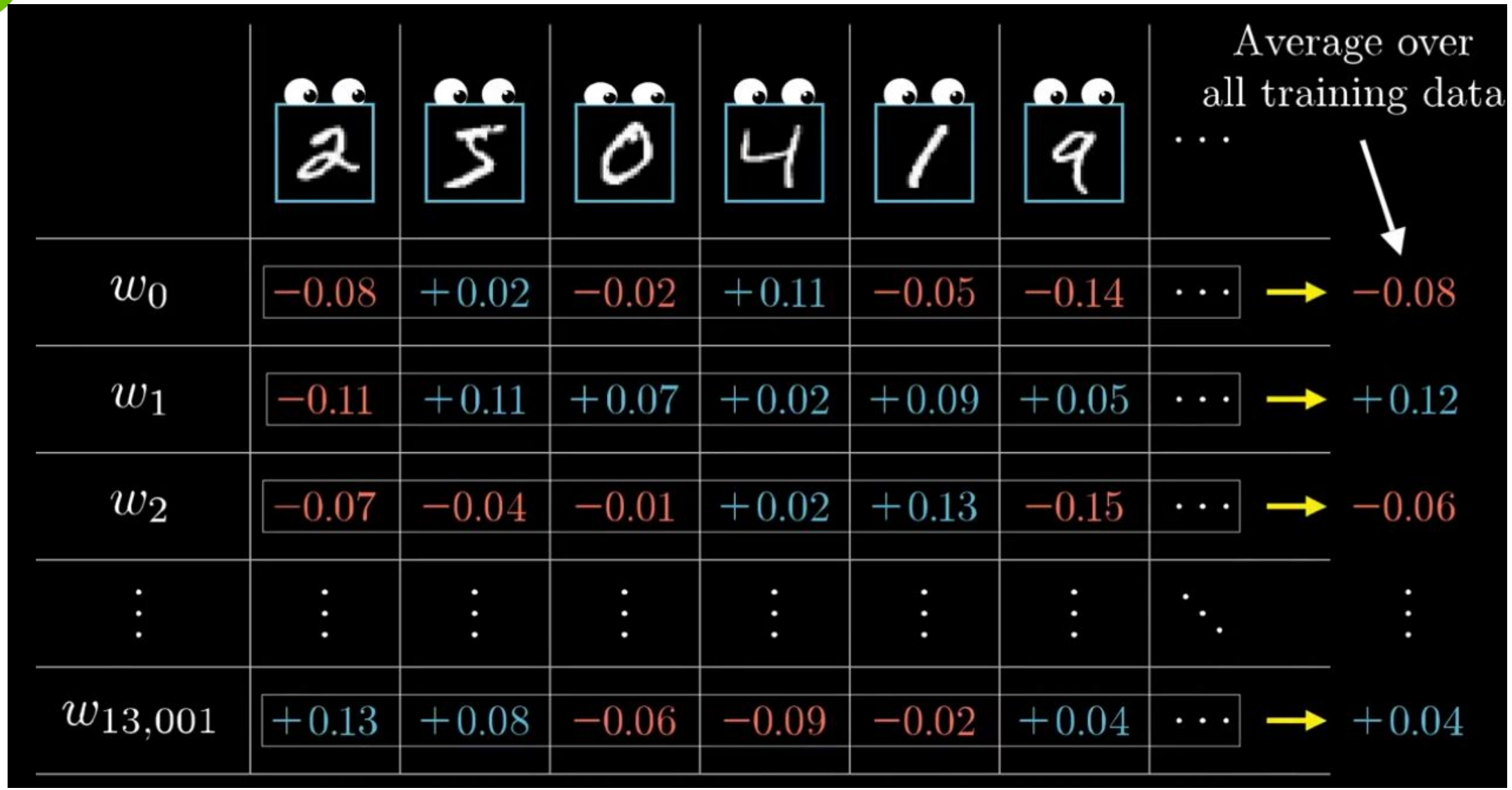
- We do not have direct influence but we note down how we would like to change them

In previous chapters...

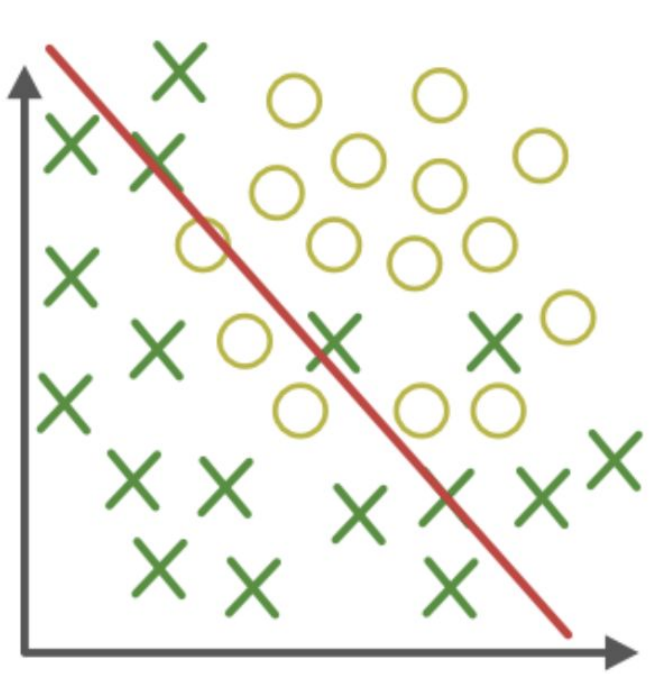


Watch out! Every neuron has its own idea of how the rest of the neurons in previous layer should be changed… Backward Pass!
But… of course… this is only one input-output… we do not want all numbers to be twos!

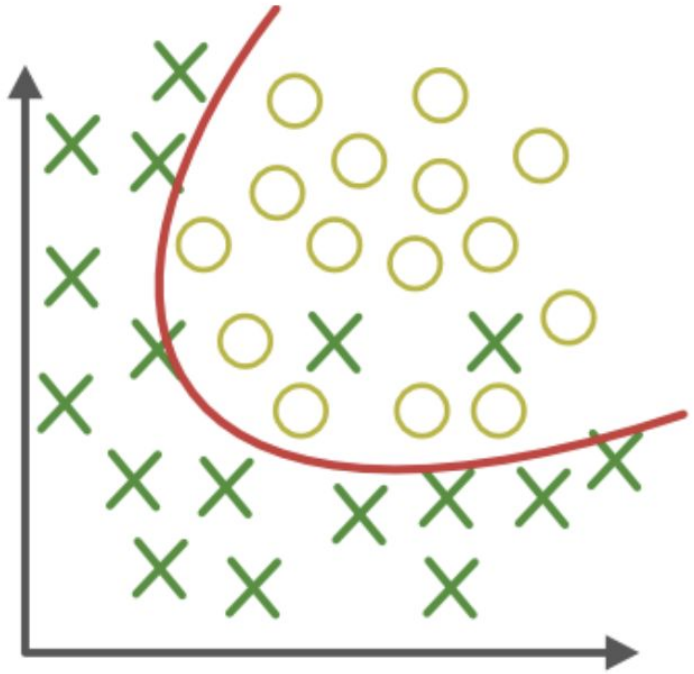
In previous chapters...



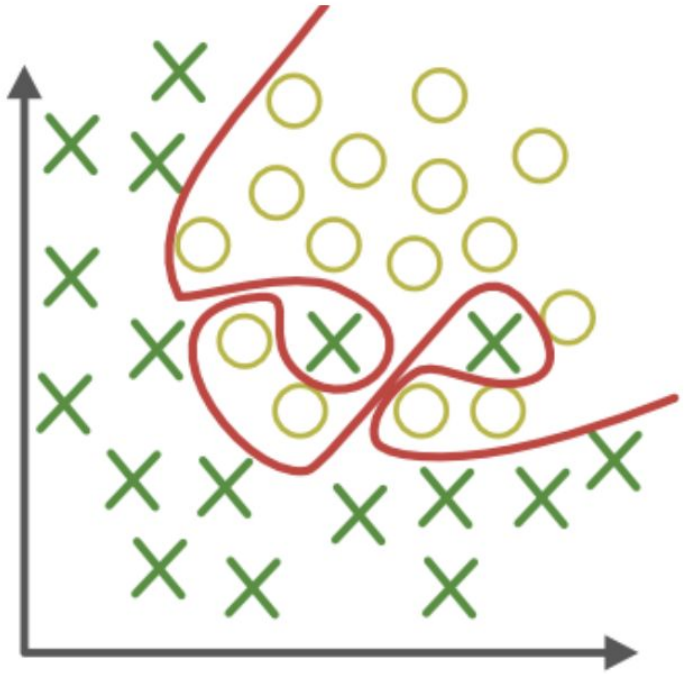
In previous chapters...



High Bias



OK



High Variance

In previous chapters...

Training error	Dev/Test error	Scenario
High	High	High Bias
High	Low	What did you do? ☺
Low	High	High Variance
Low	Low	Ok

- **High Bias (looking at the train set performance you have a high error)**
  - Is your model too simple? Try a more complex model. Increase hidden units and hidden layers.
  - Train longer or try some advance optimization algorithm.
  - Are you using the appropriate DNN architecture? Are DNN your best bet?

In previous chapters...

Training error	Dev/Test error	Scenario
High	High	High Bias
High	Low	What did you do? ☺
Low	High	High Variance
Low	Low	Ok

- **High Variance (looking at the dev set performance you have a high error)**
  - Do you need more data? Try more data
  - Reduce the complexity of your model
  - Regularization

In previous chapters...

- **L2 and L1 regularization**

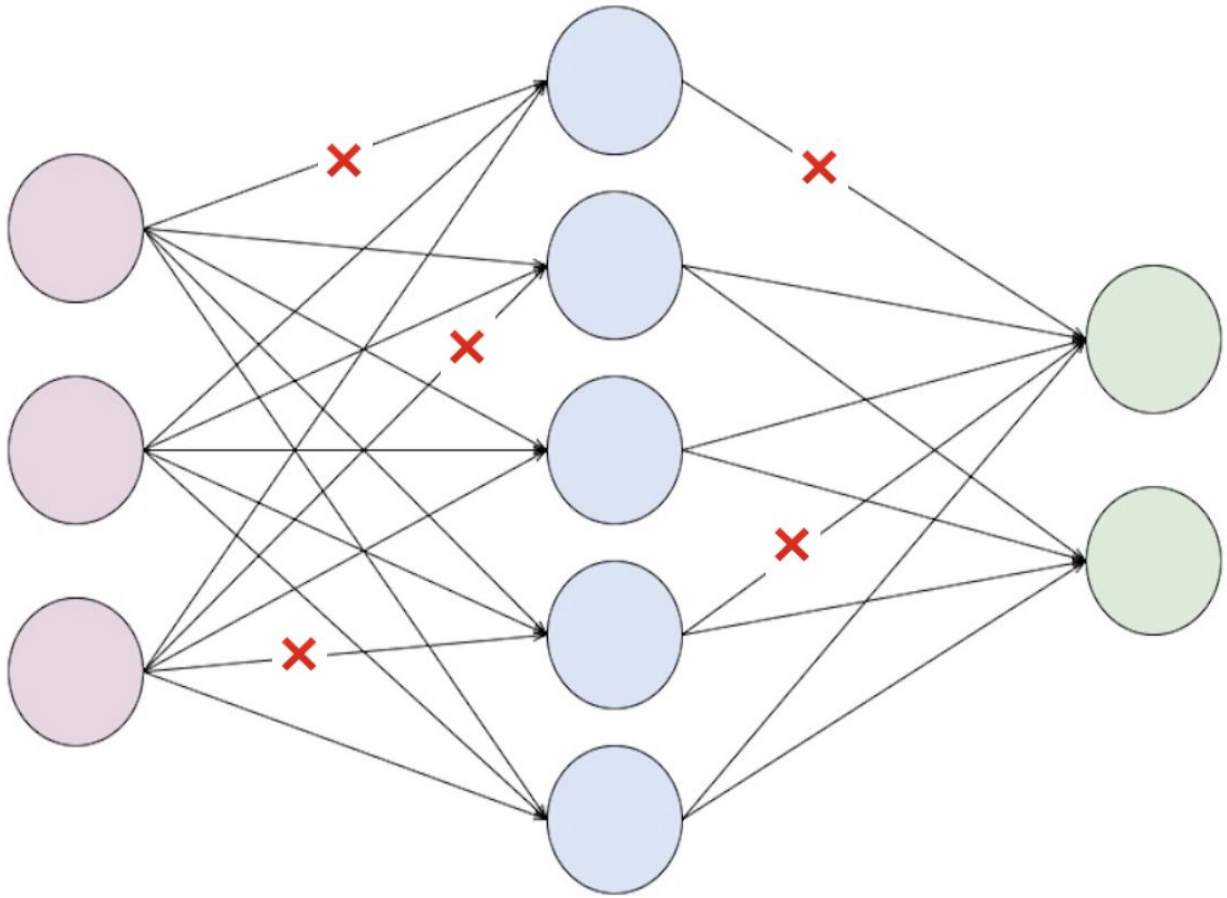
$$L2 \rightarrow \quad L(w) = 1/2M \sum (h_w(x)^{(i)} - y^{(i)})^2 + \lambda/2M \sum w^2$$

$$L1 \rightarrow \quad L(w) = 1/2M \sum (h_w(x)^{(i)} - y^{(i)})^2 + \lambda/2M \sum |w|$$

  - The main effect of both is to reduce the value of the weights, as the larger the weights the larger the loss.
  - You can also regularize the bias, but it is much more significant to regularize the weights as we have much more weights than biases.

In previous chapters...

- **Why L1/L2 works?**



In previous chapters...

- **L2 and L1 regularization in Keras/Tensorflow**

https://keras.io/api/layers/regularizers/

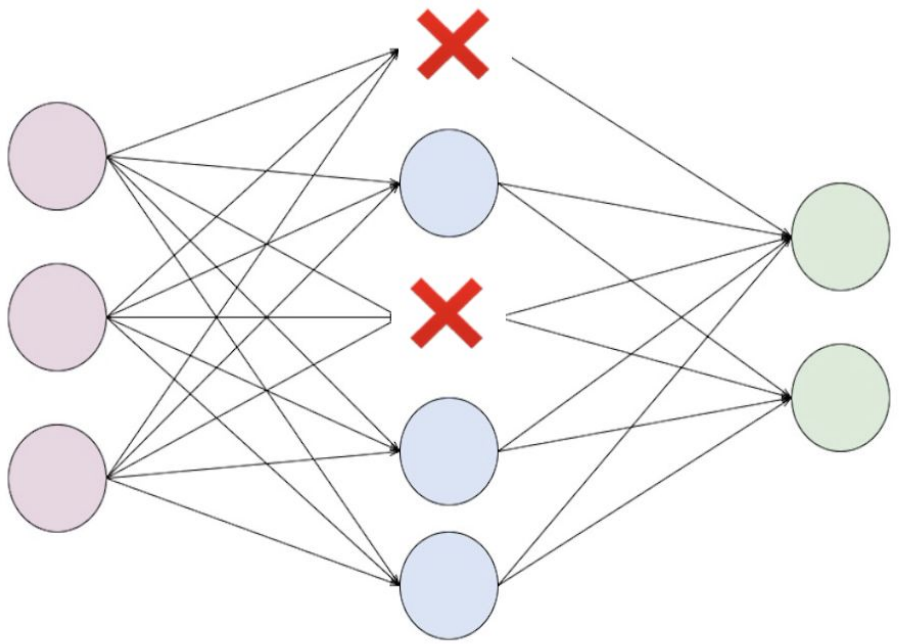
```
from tensorflow.keras import layers
from tensorflow.keras import regularizers

layer = layers.Dense(
    units=64,
    kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
    bias_regularizer=regularizers.L2(1e-4),
    activity_regularizer=regularizers.L2(1e-5)
)
```

In previous chapters...

- **Dropout**

In each iteration of gradient descent, we randomly remove some neurons with a given probability.



Prob to dropout = 0.5

In previous chapters...

- **Dropout in Keras/Tensorflow**

https://keras.io/api/layers/regularization_layers/dropout/

```
tf.keras.layers.Dropout(rate, noise_shape=None, seed=None, **kwargs)
```

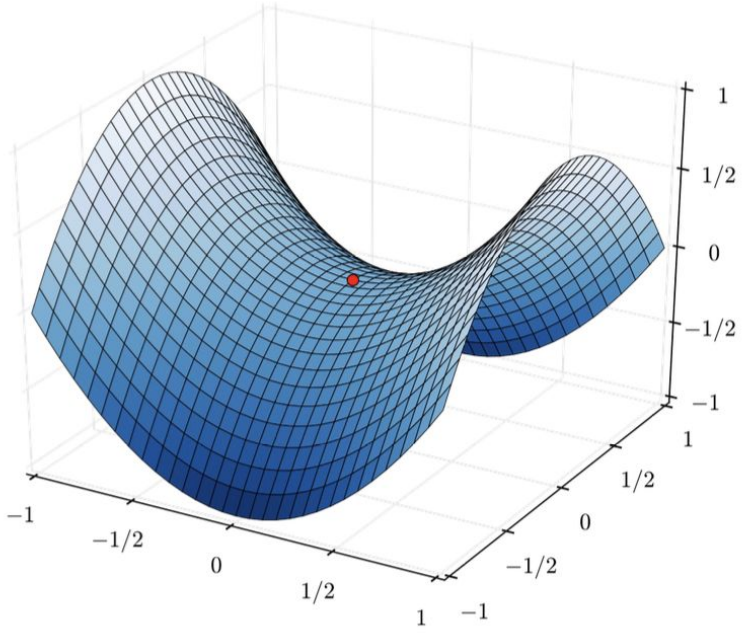
Outline

1. Recap
2. Local Minima
3. Optimization
4. Hyperparams Tuning

Local Minima

- The intuitions in 2 or 3 dimensions are not valid for high dimensional spaces, where we don't know what is exactly happens…

- But where we know that local minima tends to be Saddle Points rather than actual local minima



Optimization

- There are several ways to make faster Gradient Descent.
- Some of them are based on the amount of data used to compute the gradients (e.g., SGD using minibatches).
- Others are based on optimize the Gradient Descent algorithm itself
  - Learning Rate Decay
  - Momentum
  - RMSProp
  - Adam

SGD with Mini-Batches

- Shuffle training data (M)
- Pick up a mini-batch (X << M)
- Compute gradient for those X training points (step)
- Upgrade gradient.
- Epoch is an iteration over all the training data. So #steps = M / X

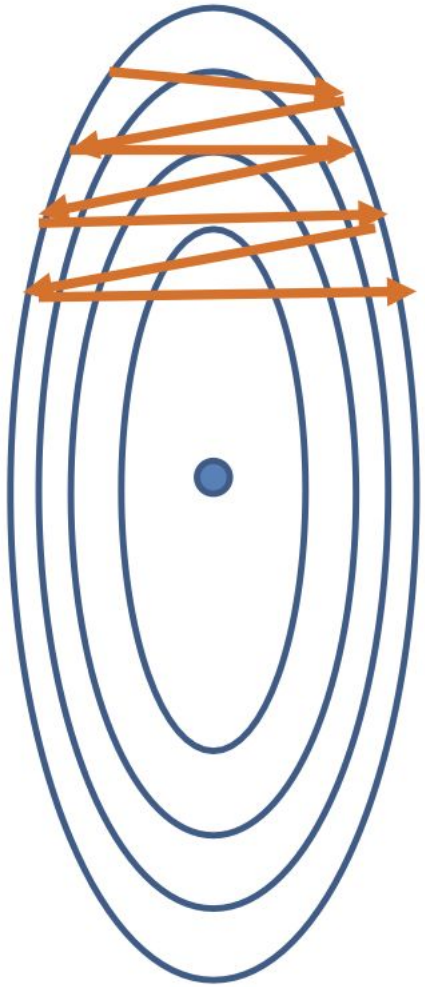
SGD with Mini-Batches in tf

`fit` method [source]

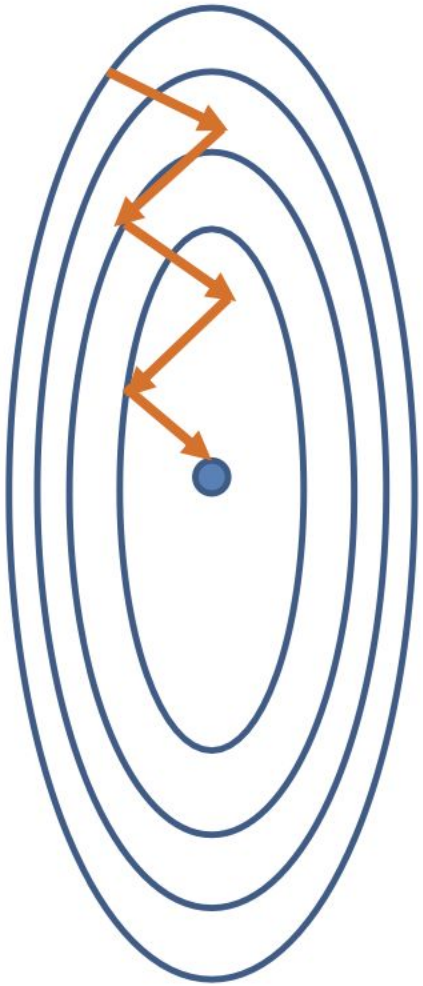
```
Model.fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose="auto",
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
)
```

- **batch_size**: Integer or `None`. Number of samples per gradient update. If unspecified, `batch_size` will default to 32. Do not specify the `batch_size` if your data is in the form of datasets, generators, or `keras.utils.Sequence` instances (since they generate batches).

Momentum RMSprop and Adam. Intuition



Momentum RMSprop and Adam. Intuition



Momentum RMSprop and Adam. Intuition

- Momentum and RMSProp are techniques to drive gradient descent, avoiding large steps in some directions that are cancel out in the following iteration.

- Adam combines momentum and RMSProp.

Momentum RMSprop and Adam. Intuition

- **Optimizers in Keras/Tensorflow**

https://keras.io/api/optimizers/

Optimizers

Usage with `compile()` & `fit()`

An optimizer is one of the two arguments required for compiling a Keras model:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential()
model.add(layers.Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(layers.Activation('softmax'))

opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt)
```

Callbacks

- **Callbacks in Keras/Tensorflow**

https://keras.io/api/callbacks/

You can pass a list of callbacks (as the keyword argument `callbacks`) to the `.fit()` method of a model:

```
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]
model.fit(dataset, epochs=10, callbacks=my_callbacks)
```

The relevant methods of the callbacks will then be called at each stage of the training.

Callbacks

- **Callbacks in Keras/Tensorflow**

https://keras.io/api/callbacks/early_stopping/

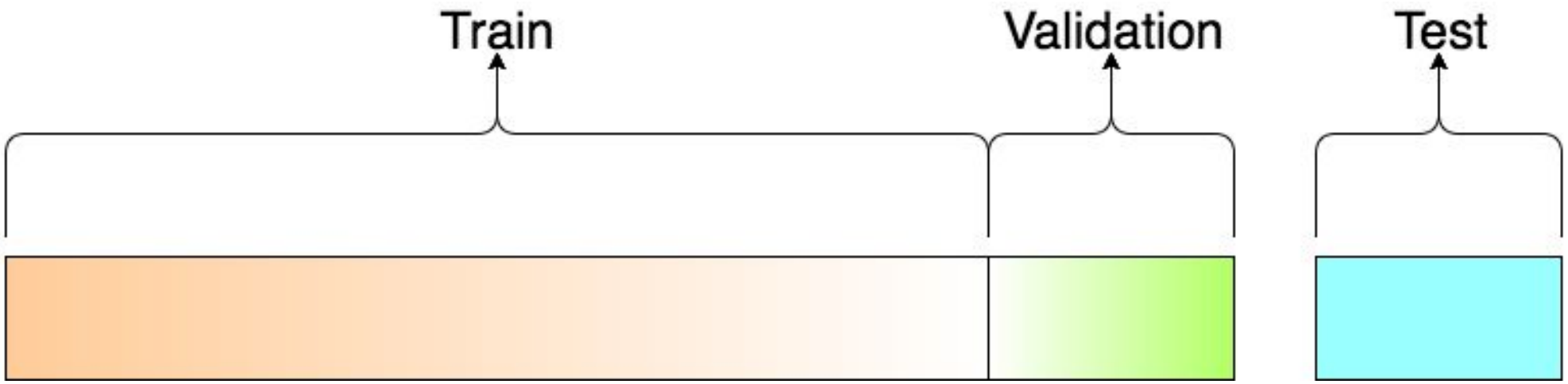
EarlyStopping class [source]

```
tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=0,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
    start_from_epoch=0,
)
```

Hyperparams

- Deep nets have many hyperparams:
  - Learning rate
  - Number of hidden layers
  - Number of hidden units
  - Size of minibatches
  - Hyperparams related with the optimization algorithm
  - ...

Hyperparams



Train
- Adjust parameters

Validation
- Adjust hyper parameters

Test
- Kept as if it was gold!

Hyperparams

- Two different approaches:

  – One model where you carefully change hyperparams while training by looking at val_loss.

  – Different models where you just look at val_loss

- When running different models don't use exhaustive grids of every pair of hyperparams.

- There are packages that search for you the best configuration (e.g. Keras Tuner)

Hyperparams

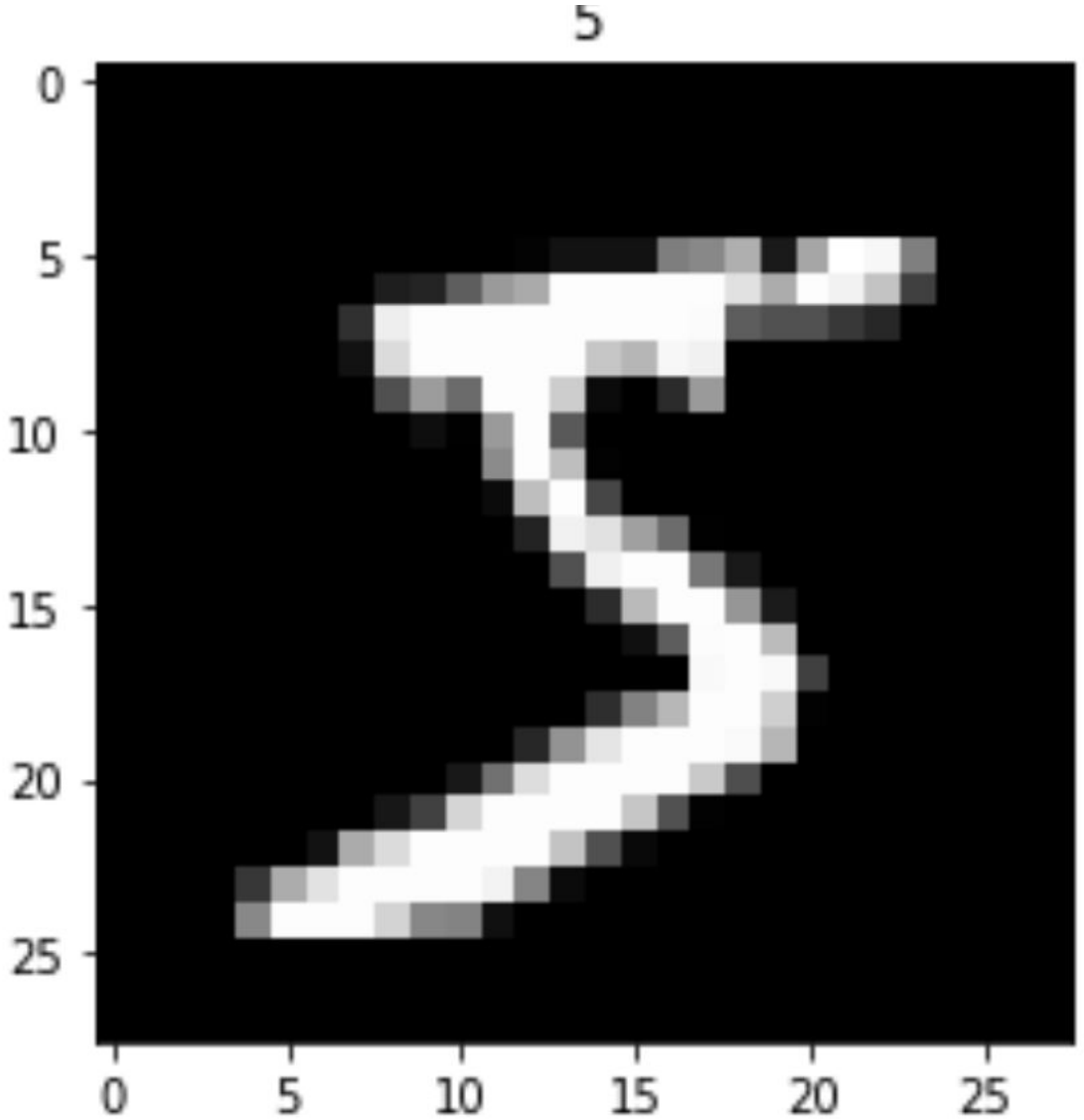
Keras Tuner

https://keras.io/keras_tuner/

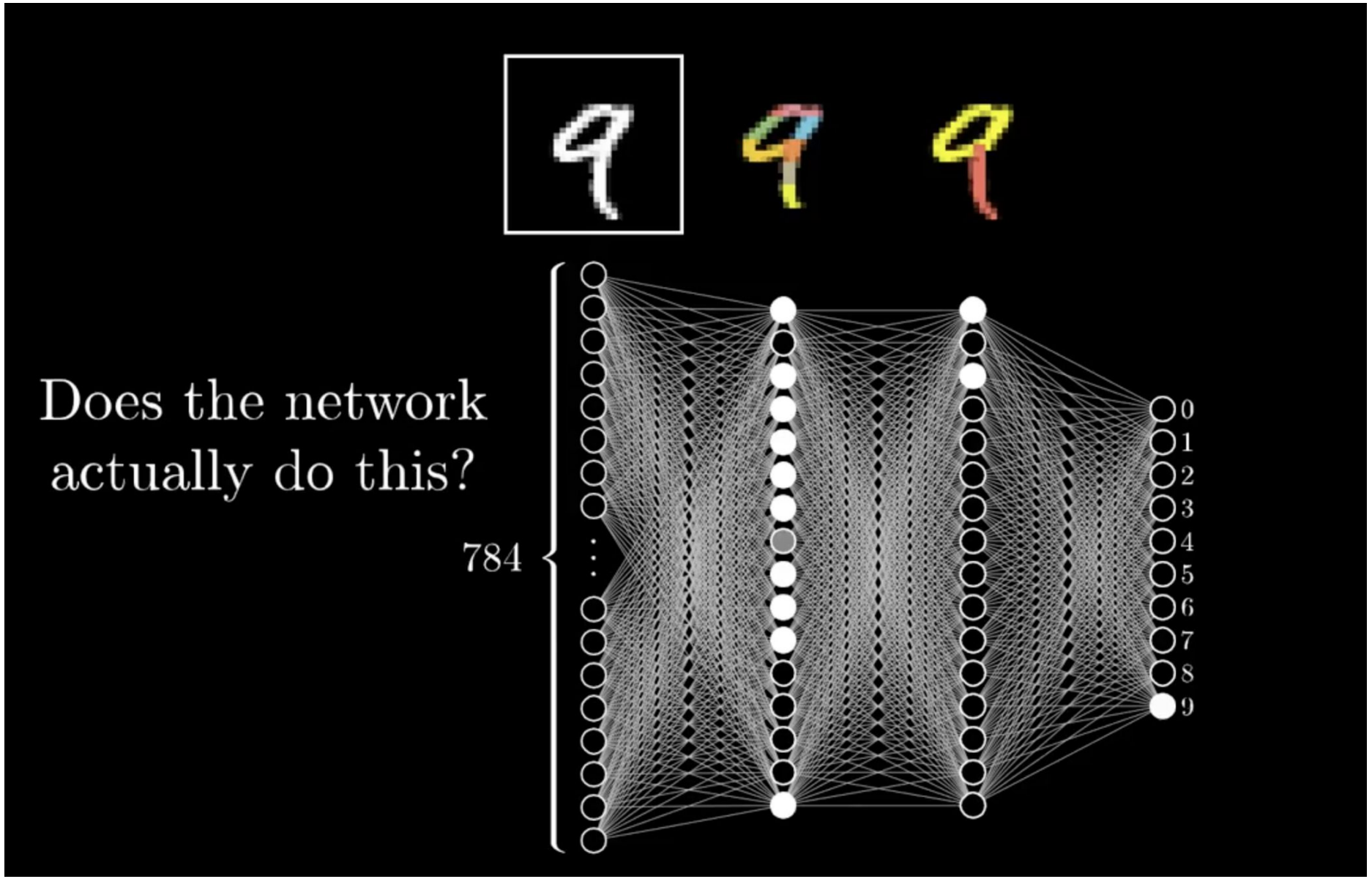
KerasTuner

KerasTuner is an easy-to-use, scalable hyperparameter optimization framework that solves the pain points of hyperparameter search. Easily configure your search space with a define-by-run syntax, then leverage one of the available search algorithms to find the best hyperparameter values for your models. KerasTuner comes with Bayesian Optimization, Hyperband, and Random Search algorithms built-in, and is also designed to be easy for researchers to extend in order to experiment with new search algorithms.

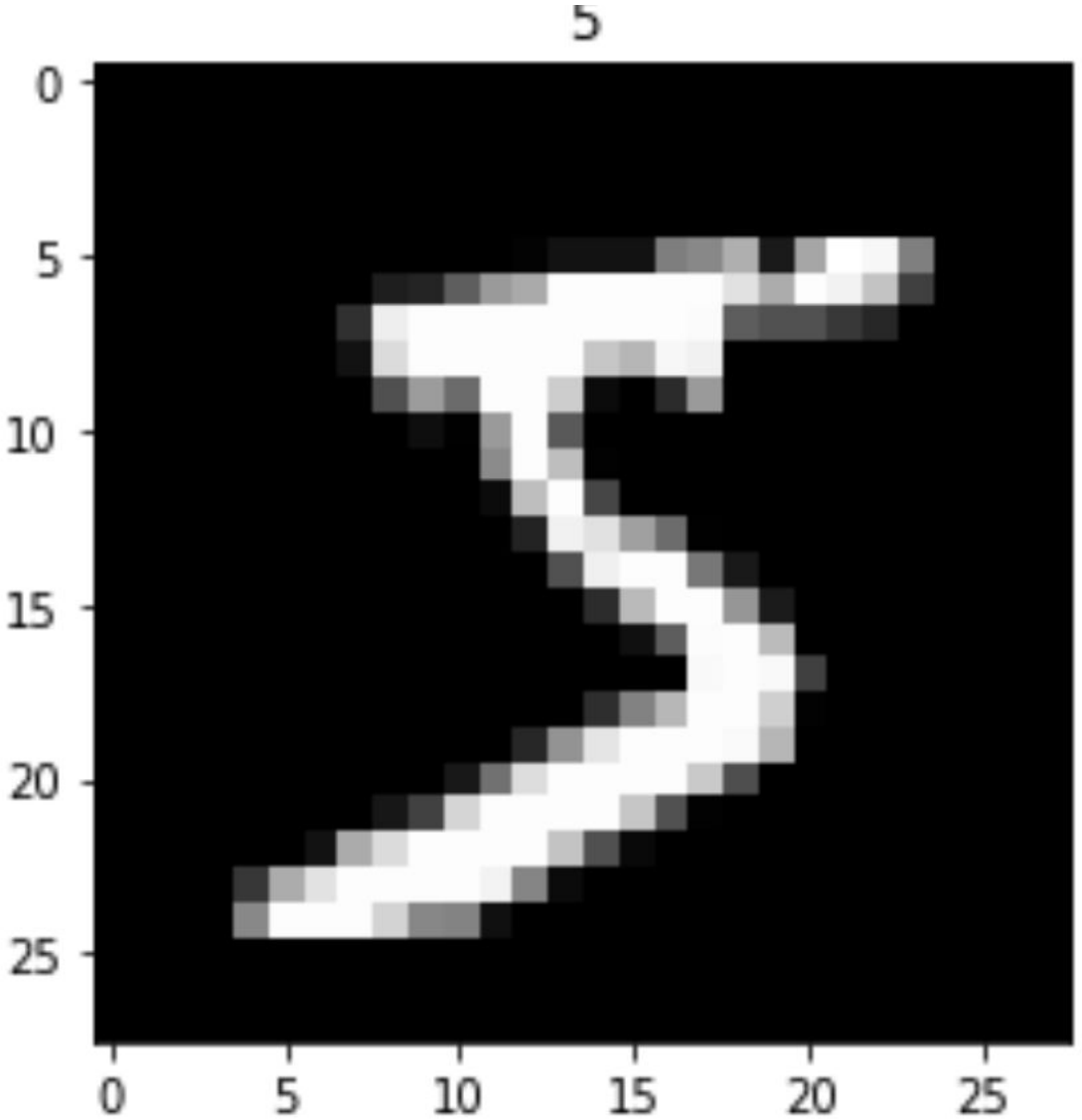
Is MLP Smart?



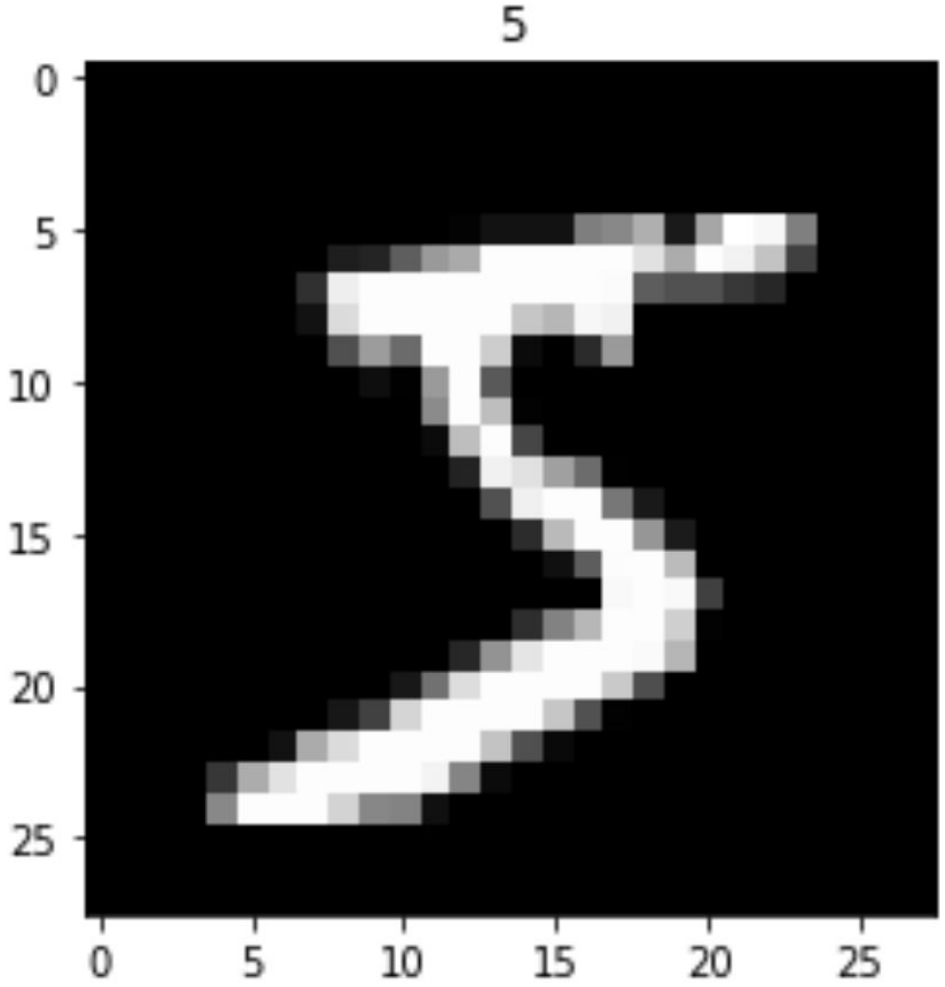
Video



Is MLP the best we can do for images?



Is MLP the best we can do for images?



- 28 x 28 inputs = 784
- 128 units first hidden layer
  - 784 * 128 + 128 ~= 100k
- 128 units second hidden layer
  - 128 * 128 + 128 ~= 16k
- 10 units output layer
  - 128 * 10 + 10 ~= 1k

Total number o params ~ 117k

Is MLP the best we can do for images?