Data Science: Principles and Practice

Lecture 6: Deep Learning, Part II

Ekaterina Kochmar[1]

UNIVERSITY OF CAMBRIDGE

[1] Based on slides by Marek Rei

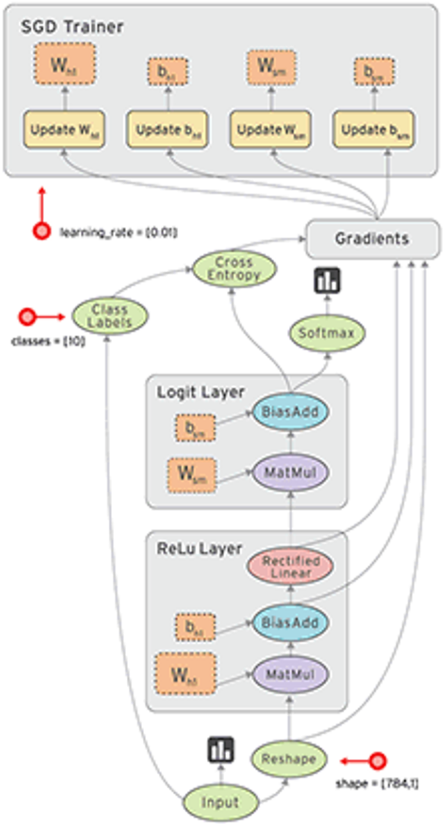
Today:

Focusing on **TensorFlow**

Giving you all the **basics** you need in order to use TensorFlow for building neural networks.

Can't cover everything (not even close).
There is a lot of **material online** if you're looking for how to do something specific in TensoFlow.

Looking at some **practical tips** for training neural networks.



Data Science: Principles and Practice

01 Introduction to `TensorFlow`

02 First steps with `TensorFlow`

03 Training a network

04 Useful things to know about Deep Learning

05 Practical 4

TensorFlow

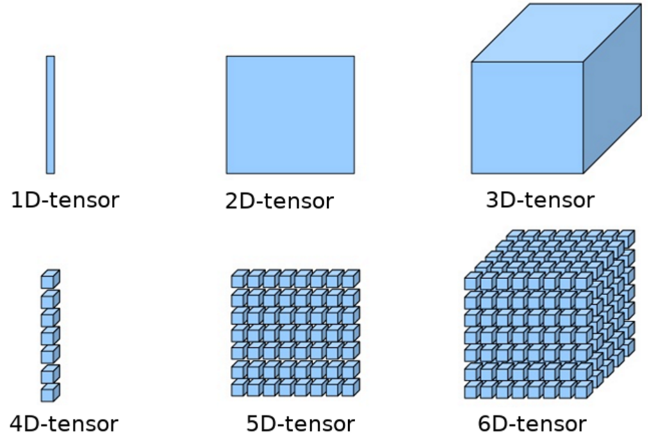
Open source library for implementing **neural networks**.

Developed by **Google**, for both production code and research.

Performs **automatic differentiation**.

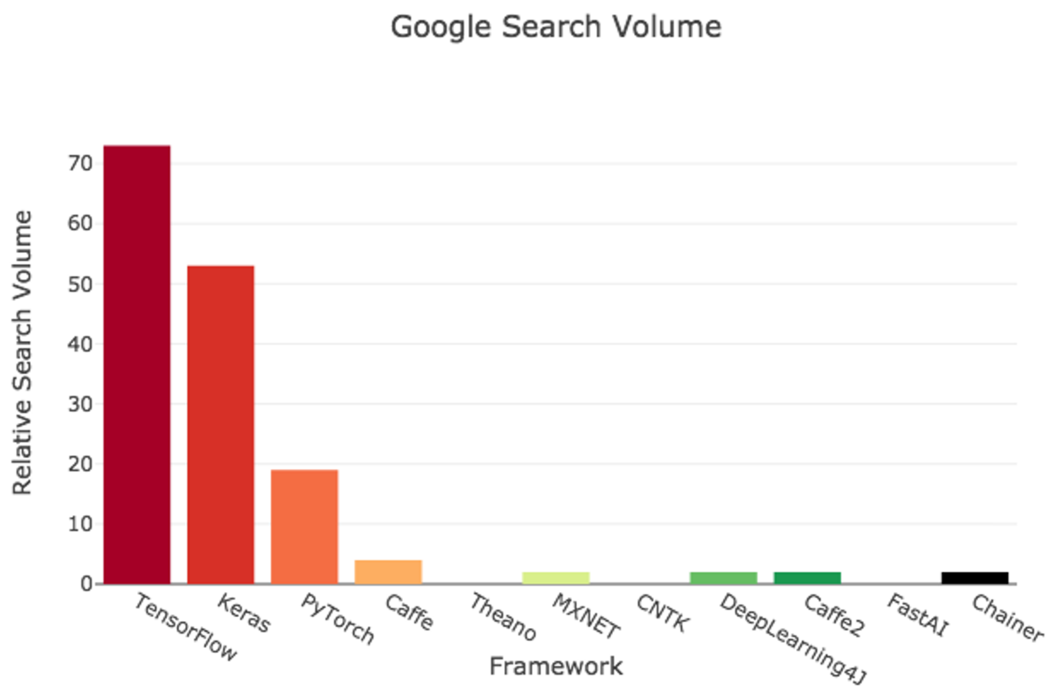
Comes with many neural network **modules** implemented.

Tensor – an n-dimensional vector.



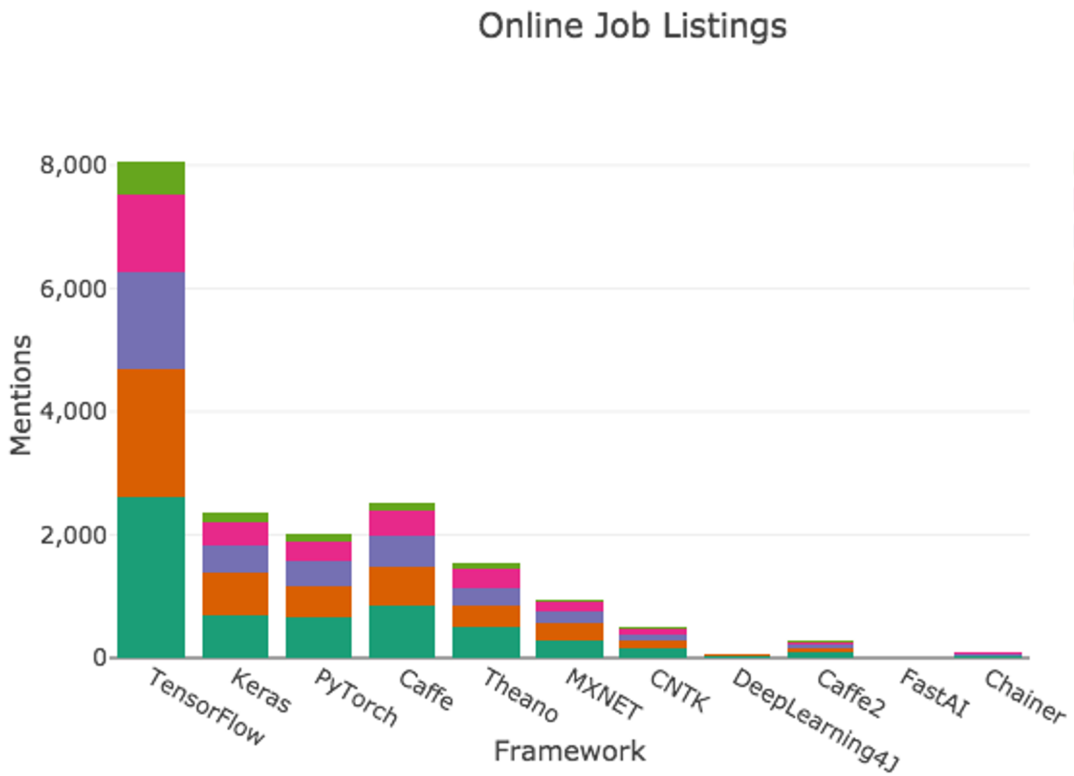
https://www.cc.gatech.edu/~san37/post/dlhc-start/

Why TensorFlow?



https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Why TensorFlow?

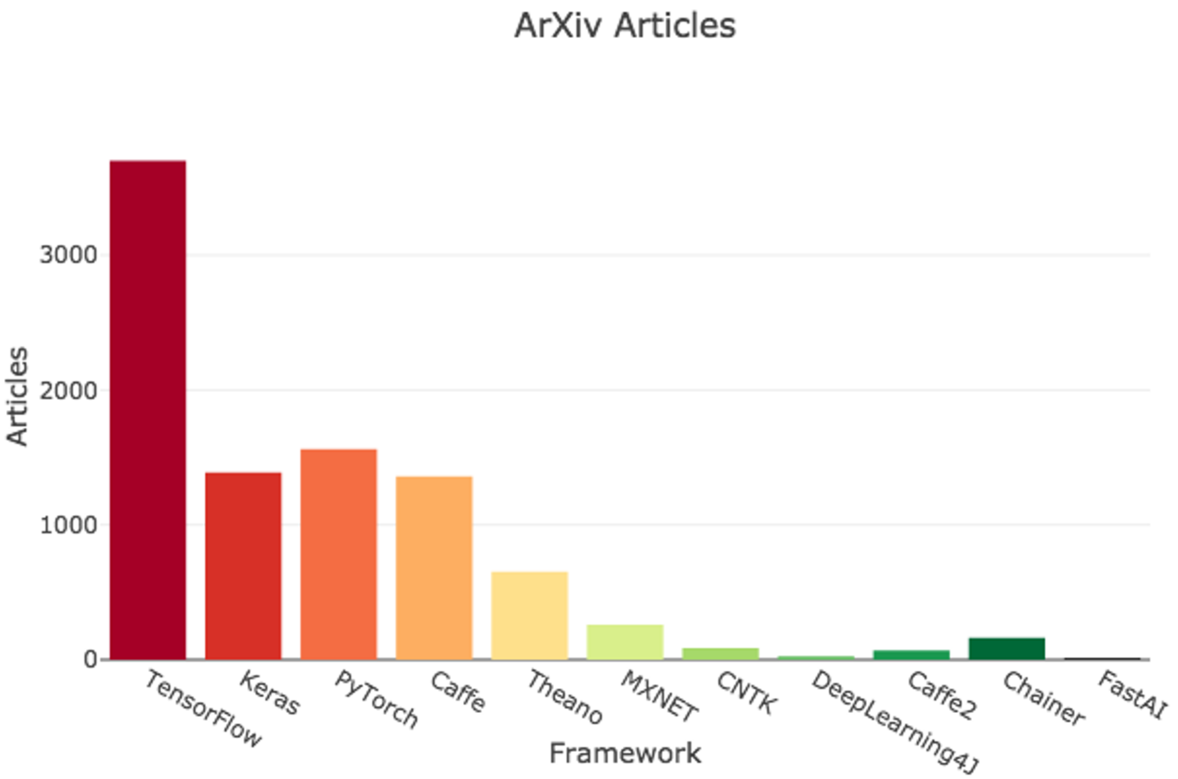


https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Companies Using TensorFlow

Google
NVIDIA
Dropbox
intel
DeepMind
UBER
ebay
twitter
LinkedIn

Why TensorFlow?

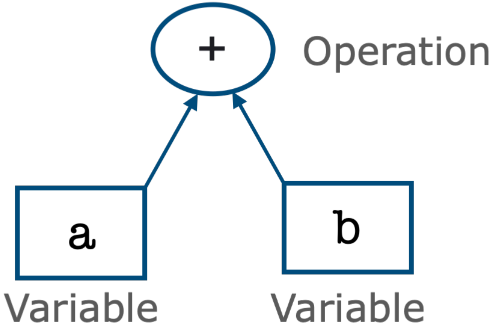


https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

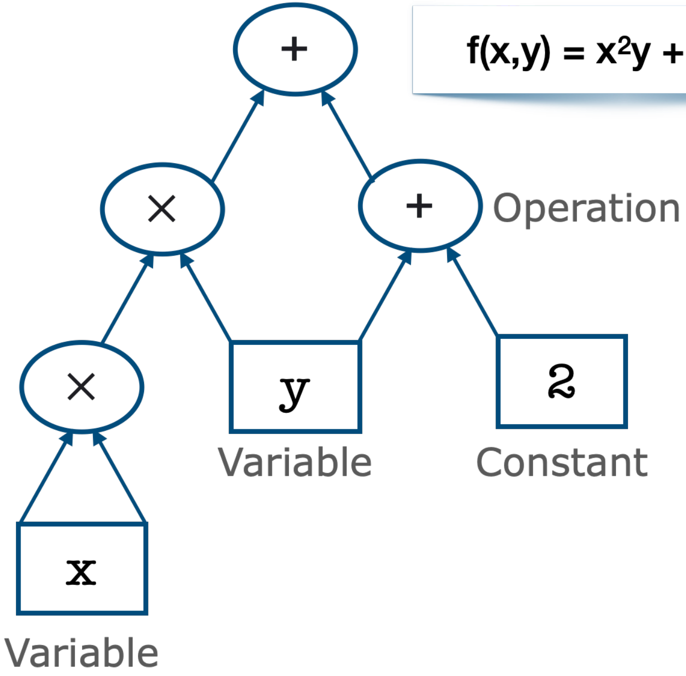
TensorFlow: The First Steps

TensorFlow 1 static graph

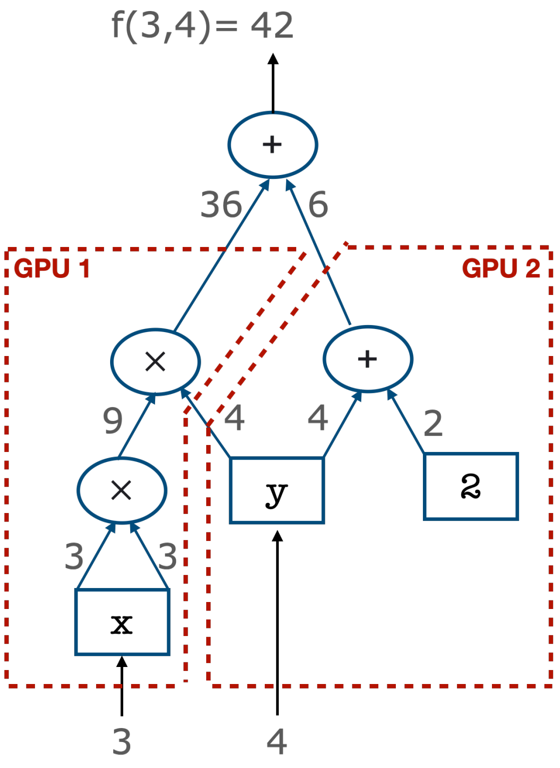
$f(a,b) = a + b$



$f(x,y) = x^2y + y + 2$



Distributed computing with TensorFlow



Geron (2017). "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*"

Minimal Example with TensorFlow 1

```
import tensorflow as tf

a = tf.placeholder(tf.float32, name="a")
b = tf.placeholder(tf.float32, name="b")
y = a + b

with tf.Session() as sess:
    result = sess.run(y,
                      feed_dict={a:4, b:5})
    print("Result: ", result)
```

```
Result:  9.0
```

One of the **smallest examples** of running TensorFlow, while actually looking like a normal TensorFlow code.

Creates a **computation graph** that takes two inputs and sums them together.

We then **execute this graph** with values 4 and 5, and print the result.

Minimal Example with TensorFlow 1

```
import tensorflow as tf

a = tf.placeholder(tf.float32, name="a")
b = tf.placeholder(tf.float32, name="b")
y = a + b

with tf.Session() as sess:
    result = sess.run(y,
                      feed_dict={a:4, b:5})
    print("Result: ", result)
```

```
Result:  9.0
```

One of the **smallest examples** of running TensorFlow, while actually looking like a normal TensorFlow code.

Creates a **computation graph** that takes two inputs and sums them together.

We then **execute this graph** with values 4 and 5, and print the result.

Let's go though this in more detail!

Minimal Example with TensorFlow 1

```python
import tensorflow as tf

a = tf.placeholder(tf.float32, name="a")
b = tf.placeholder(tf.float32, name="b")
y = a + b

with tf.Session() as sess:
    result = sess.run(y,
                      feed_dict={a:4, b:5})
    print("Result: ", result)
```

```
Result:  9.0
```

import tensorflow as tf

Install tensorflow for **CPU**:

```
pip install tensorflow
```

Install tensorflow for **GPU**:

```
pip install tensorflow-gpu
```

Construction Phase with TensorFlow 1

```
import tensorflow as tf

a = tf.placeholder(tf.float32, name="a")
b = tf.placeholder(tf.float32, name="b")
y = a + b

with tf.Session() as sess:
    result = sess.run(y,
                      feed_dict={a:4, b:5})
    print("Result: ", result)
```

```
Result:  9.0
```

tf.placeholder()

Define an **input argument** for our network.

Can have different **types**
(float32, float64, int32, …)

and **shapes**
(scalar, vector, matrix, …)

Right now, we defined two single **scalar placeholders**: a and b.

Construction Phase with TensorFlow 1

```
import tensorflow as tf

a = tf.placeholder(tf.float32, name="a")
b = tf.placeholder(tf.float32, name="b")
y = a + b

with tf.Session() as sess:
    result = sess.run(y,
                      feed_dict={a:4, b:5})
    print("Result: ", result)
```

```
Result:  9.0
```

$y = a + b$

Probably the most **important** thing to understand about classic TensorFlow!

Symbolic Graphs

We first construct a **symbolic graph** and then apply it later with suitable data.

For example, what happens when this TensorFlow 1 line is **executed** in our code?

```
y = a + b
```

The system takes **a** and **b**, adds them together and stores the value in **y**. Right?

Symbolic Graphs

We first construct a **symbolic graph** and then apply it later with suitable data.

For example, what happens when this TensorFlow line is **executed** in our code?

```
y = a + b
```

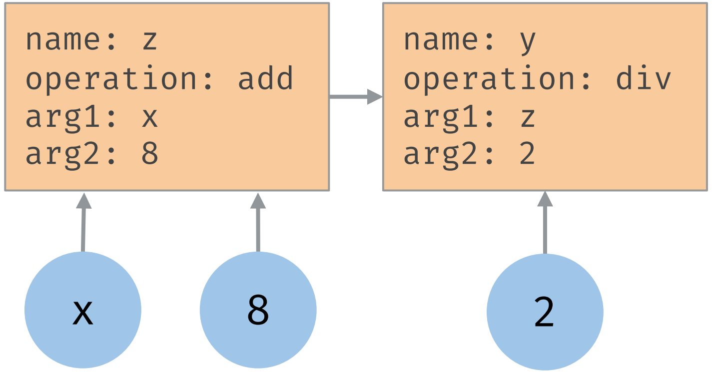
The system takes **a** and **b**, adds them together and stores the value in **y**. Right?

Not really!

Instead, we create a TensorFlow-specific object **y** that knows its value can be calculated by summing together **a** and **b**. But the addition itself is not performed here!

Symbolic Graphs

Can construct a whole network structure by intuitively **combining operations**.




```
z = x + 8

y = z / 2
```

We can only use **TensorFlow-specific*** operations to construct a TensorFlow graph - they return TensorFlow objects, as opposed to trying to execute the operation.

* Most of numpy and standard operations are compatible with TensorFlow

Execution Phase with TensorFlow 1

```python
import tensorflow as tf

a = tf.placeholder(tf.float32, name="a")
b = tf.placeholder(tf.float32, name="b")
y = a + b

with tf.Session() as sess:
    result = sess.run(y,
                      feed_dict={a:4, b:5})
    print("Result: ", result)
```

```
Result:  9.0
```

tf.Session()

Constructs the **environment** in which the operations are performed and evaluated.

Allocates the **memory** to store current value of variables.

When starting a new session, all the values will be **reset**.

Execution Phase with TensorFlow 1

sess.run()

```
import tensorflow as tf

a = tf.placeholder(tf.float32, name="a")
b = tf.placeholder(tf.float32, name="b")
y = a + b

with tf.Session() as sess:
    result = sess.run(y,
                      feed_dict={a:4, b:5})
    print("Result: ", result)
```

```
Result:  9.0
```

Execute the network – actually perform the calculations in the symbolic graph.

Specify which values you want calculated and **returned** from the graph.

feed_dict specifies the values that you give to placeholders for this execution.

result contains the executed value of y.

The keys in feed_dict are the tensors!

From TensorFlow 1 to TensorFlow 2

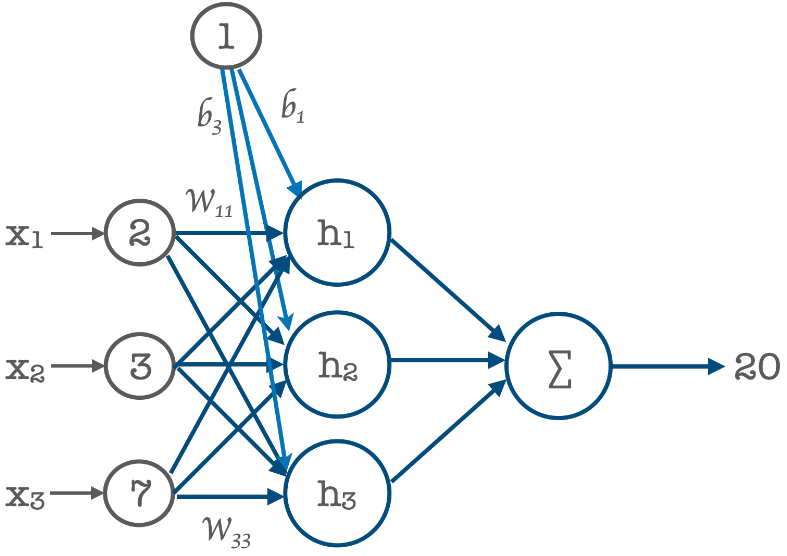
- **TensorFlow 1** relies on symbolic graphs (“Define-and-Run” scheme): the network architecture is statically defined and fixed before computation; the graph cannot be modified after compilation
- **TensorFlow 2** – eager execution (“Define-by-Run” scheme): the network is defined dynamically via the forward computation and can be modified during runtime
- This makes implementation less challenging and more intuitively clear
- **Keras** provides interpretable user-friendly interface on top of TensorFlow
- TensorFlow 2 has a compatibility mode for version 1 – see notebooks on github

https://docs.chainer.org/en/stable/guides/define_by_run.html
https://blog.udacity.com/2020/05/pytorch-vs-tensorflow-what-you-need-to-know.html

Training a Network

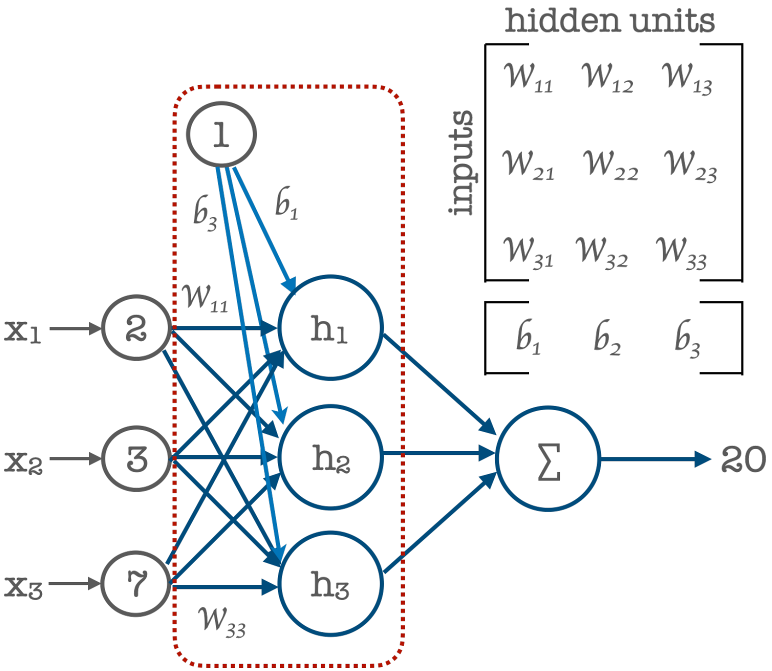
Training a Model in TensorFlow

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, input_shape=(3,)),
    tf.keras.layers.Lambda(lambda x: tf.math.reduce_sum(x, axis=1))
])

def loss_fn(predicted, gold):
    return tf.square(predicted - gold)

input = tf.constant([[2.,3.,7.]])
gold_output = 20

def loss():
    return loss_fn(model(input), gold_output)

opt = tf.keras.optimizers.SGD(learning_rate=1e-3)

for epoch in range(10):
    opt.minimize(loss, var_list=model.trainable_variables)
    print(model(input))
```



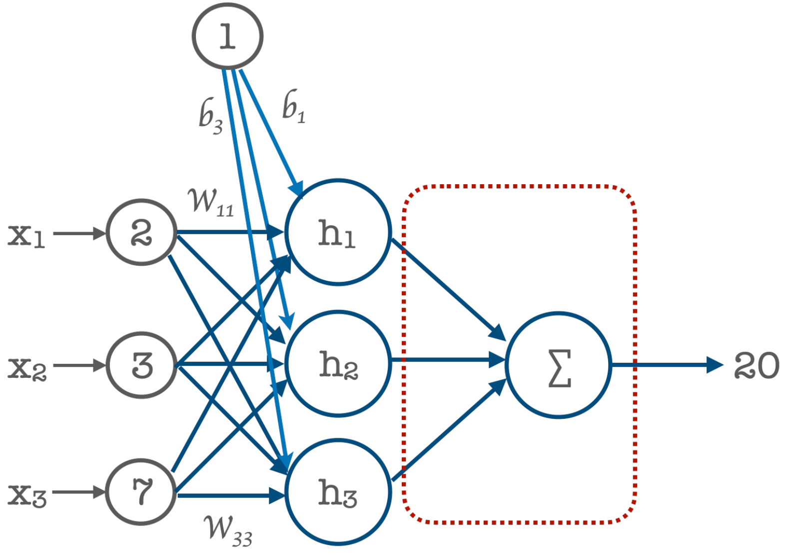
Training a Model in TensorFlow

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, input_shape=(3,)),
    tf.keras.layers.Lambda(lambda x: tf.math.reduce_sum(x, axis=1))
])

def loss_fn(predicted, gold):
    return tf.square(predicted - gold)

input = tf.constant([[2.,3.,7.]])
gold_output = 20

def loss():
    return loss_fn(model(input), gold_output)

opt = tf.keras.optimizers.SGD(learning_rate=1e-3)

for epoch in range(10):
    opt.minimize(loss, var_list=model.trainable_variables)
    print(model(input))
```



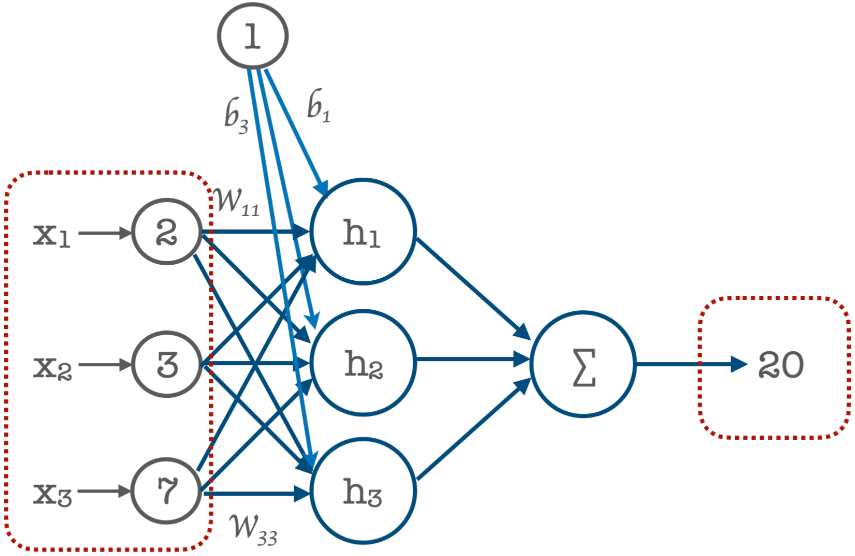
Training a Model in TensorFlow

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, input_shape=(3,)),
    tf.keras.layers.Lambda(lambda x: tf.math.reduce_sum(x, axis=1))
])

def loss_fn(predicted, gold):
    return tf.square(predicted - gold)

input = tf.constant([[2.,3.,7.]])
gold_output = 20

def loss():
    return loss_fn(model(input), gold_output)

opt = tf.keras.optimizers.SGD(learning_rate=1e-3)

for epoch in range(10):
    opt.minimize(loss, var_list=model.trainable_variables)
    print(model(input))
```



Training a Model in TensorFlow

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, input_shape=(3,)),
    tf.keras.layers.Lambda(lambda x: tf.math.reduce_sum(x, axis=1))
])

def loss_fn(predicted, gold):
    return tf.square(predicted - gold)

input = tf.constant([[2.,3.,7.]])
gold_output = 20

def loss():
    return loss_fn(model(input), gold_output)

opt = tf.keras.optimizers.SGD(learning_rate=1e-3)

for epoch in range(10):
    opt.minimize(loss, var_list=model.trainable_variables)
    print(model(input))
```



Training a Model in TensorFlow

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, input_shape=(3,)),
    tf.keras.layers.Lambda(lambda x: tf.math.reduce_sum(x, axis=1))
])

def loss_fn(predicted, gold):
    return tf.square(predicted - gold)

input = tf.constant([[2.,3.,7.]])
gold_output = 20

def loss():
    return loss_fn(model(input), gold_output)

opt = tf.keras.optimizers.SGD(learning_rate=1e-3)

for epoch in range(10):
    opt.minimize(loss, var_list=model.trainable_variables)
    print(model(input))
```

This is where we define the **strategy** for our model training.

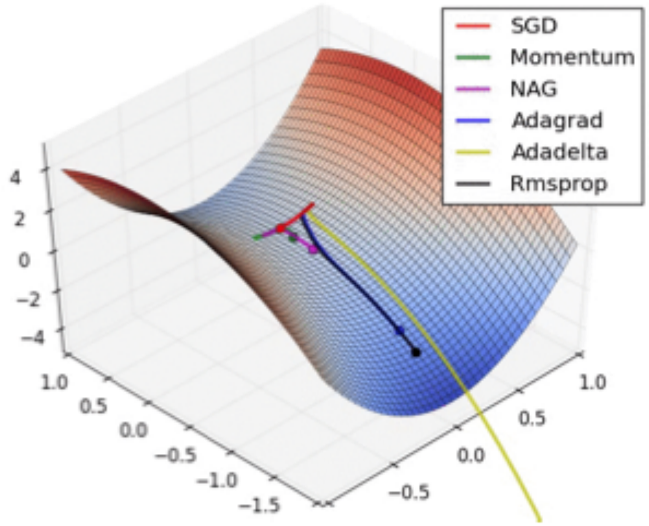
Other strategies are available:

```
tf.keras.optimizers.SGD
tf.keras.optimizers.Adadelta
tf.keras.optimizers.Adam
tf.keras.optimizers.RMSprop
```

Overview: Ruder (2016). *An overview of gradient descent optimization algorithms*. https://arxiv.org/pdf/1609.04747.pdf
TensorFlow documentation: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/

Training a Model in TensorFlow

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, input_shape=(3,)),
    tf.keras.layers.Lambda(lambda x: tf.math.reduce_sum(x, axis=1))
])

def loss_fn(predicted, gold):
    return tf.square(predicted - gold)

input = tf.constant([[2.,3.,7.]])
gold_output = 20

def loss():
    return loss_fn(model(input), gold_output)

opt = tf.keras.optimizers.SGD(learning_rate=1e-3)

for epoch in range(10):
    opt.minimize(loss, var_list=model.trainable_variables)
    print(model(input))
```



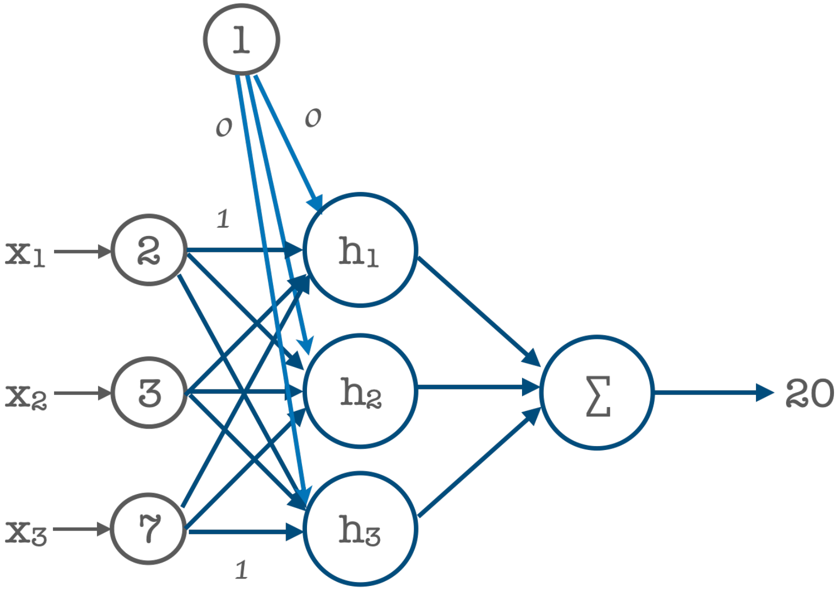
Overview: Ruder (2016). *An overview of gradient descent optimization algorithms*. https://arxiv.org/pdf/1609.04747.pdf
TensorFlow documentation: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/

Training a Model in TensorFlow

```
weight_matrix = tf.Variable(tf.ones(shape=(3,3)))
weight_vector = tf.Variable(tf.zeros(shape=(3,)))
```

With tf.keras.layers.Dense these parameters are initialised randomly

```
weights, biases = model.layers[0].get_weights()
print(weights)
print(biases)
```



Training a Model in TensorFlow

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, input_shape=(3,)),
    tf.keras.layers.Lambda(lambda x: tf.math.reduce_sum(x, axis=1))
])

def loss_fn(predicted, gold):
    return tf.square(predicted - gold)

input = tf.constant([[2.,3.,7.]])
gold_output = 20

def loss():
    return loss_fn(model(input), gold_output)

opt = tf.keras.optimizers.SGD(learning_rate=1e-3)

for epoch in range(10):
    opt.minimize(loss, var_list=model.trainable_variables)
    print(model(input))
```

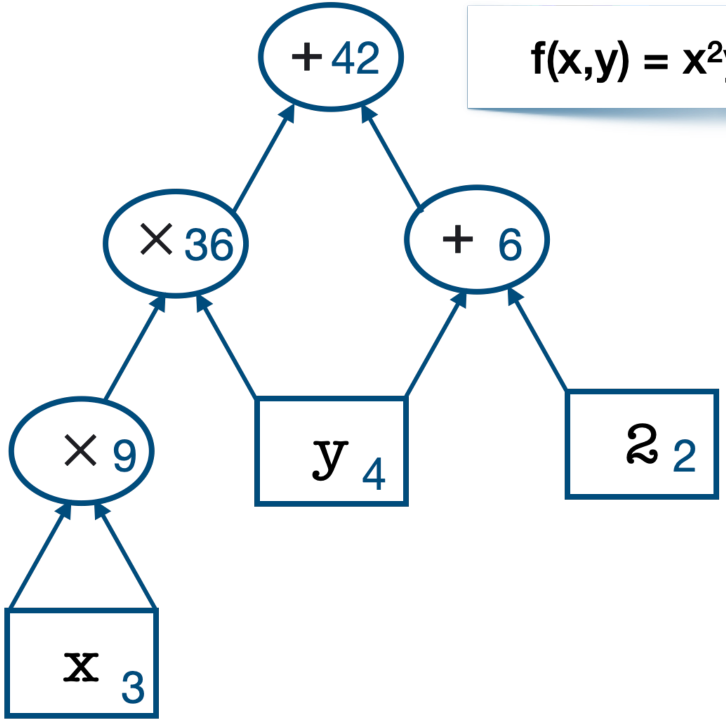
Result:

7.6623473
12.32598
15.226759
17.031046
18.153309
18.851358
19.285545
19.555609
19.72359
19.828072

Backpropagation in a nutshell

- For every training instance, the algorithm feeds it into the network and computes the outputs in each consecutive layer (**forward pass**)
- The algorithm measures the network's **output error** (difference between predicted output and actual / desired output)
- It then computes **how much each neuron** in the last hidden layer **contributed** to each output neuron's error. It proceeds to measure how much of these error contributions came from each neuron in the previous hidden layer – repeat for each layer (**reverse pass**)
- It efficiently measures the error gradient across all the connection weights in the network by **propagating the error gradient backward** in the network

Reverse-mode Autodiff: Forward pass

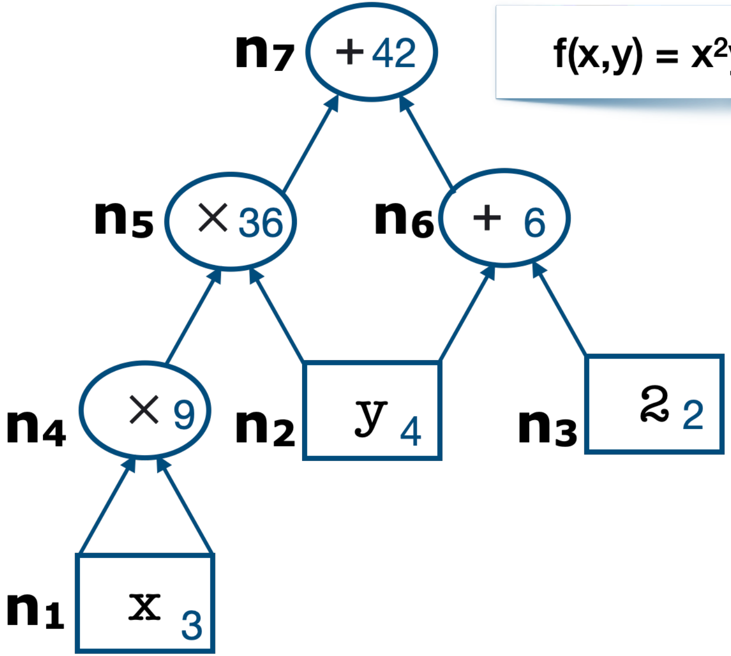


$f(x,y) = x^2y + y + 2$

$\partial f / \partial x$ – ?
$\partial f / \partial y$ – ?

Geron (2017). "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*"

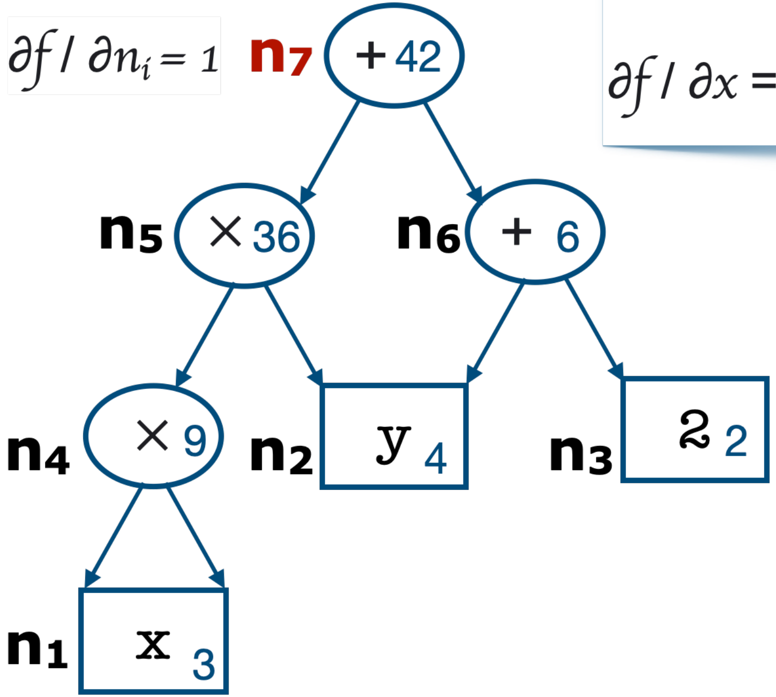
Reverse-mode Autodiff: Forward pass



$f(x,y) = x^2y + y + 2$

Geron (2017). "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*"

Reverse-mode Autodiff: Reverse pass



Chain rule:

$$\partial f / \partial x = \partial f / \partial n_i \times \partial n_i / \partial x$$

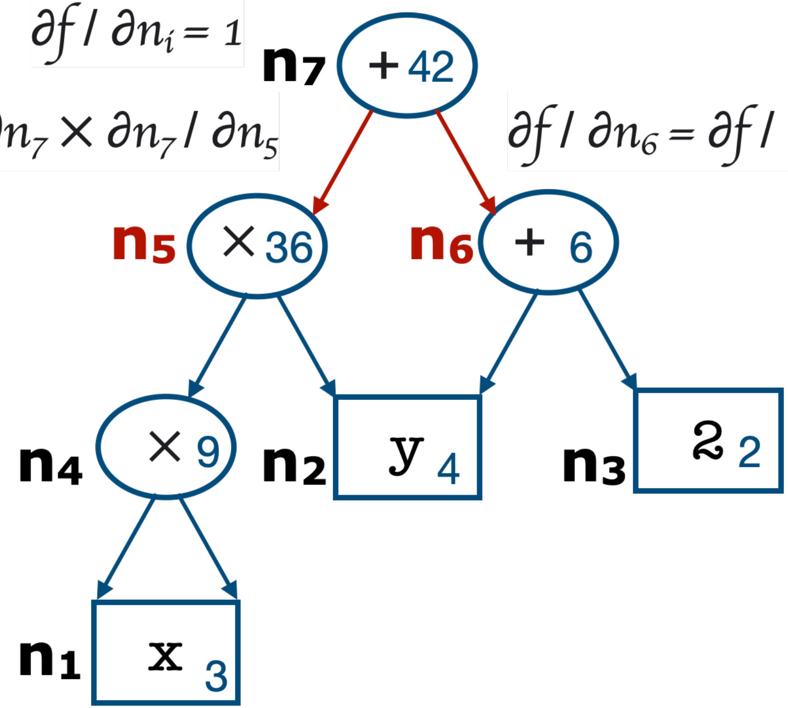
Geron (2017). "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*"

Reverse-mode Autodiff: Reverse pass

$\partial f / \partial n_i = 1$

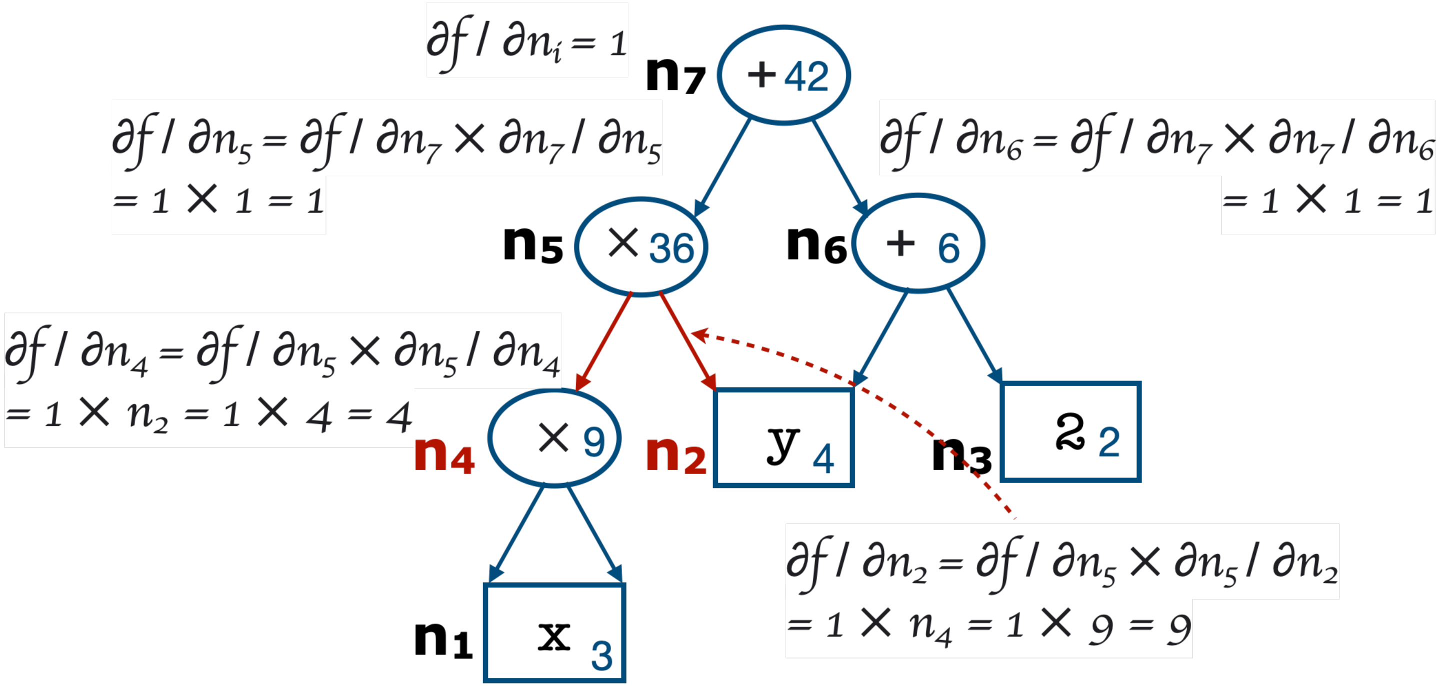
$\partial f / \partial n_5 = \partial f / \partial n_7 \times \partial n_7 / \partial n_5$
$= 1 \times 1 = 1$

$\partial f / \partial n_6 = \partial f / \partial n_7 \times \partial n_7 / \partial n_6$
$= 1 \times 1 = 1$



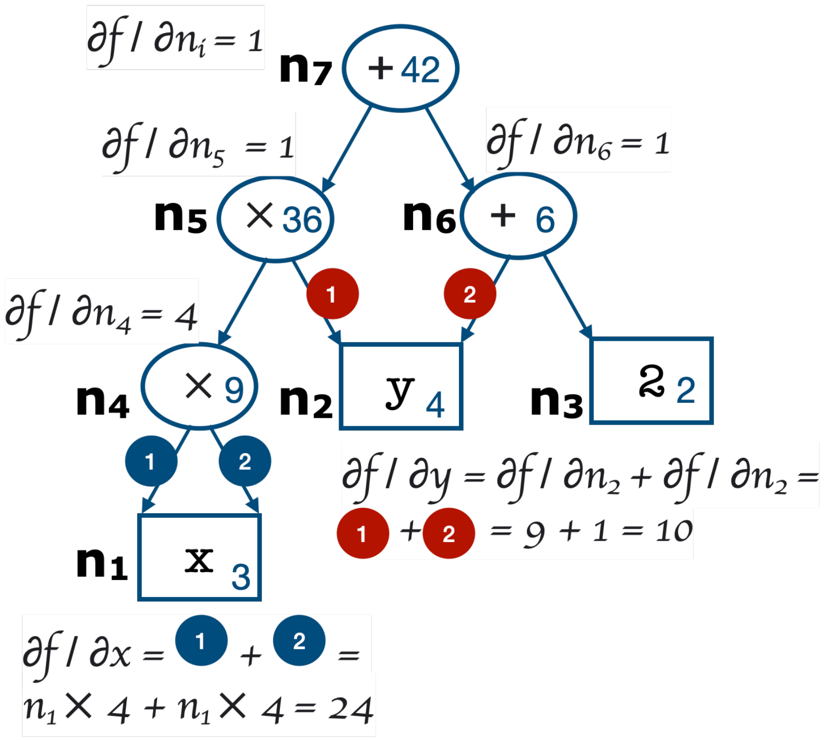
Geron (2017). "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*"

Reverse-mode Autodiff: Reverse pass



Geron (2017). "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*"

Reverse-mode Autodiff: Reverse pass



Geron (2017). "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*"

Recap: Activation Functions

- **Logistic function**:

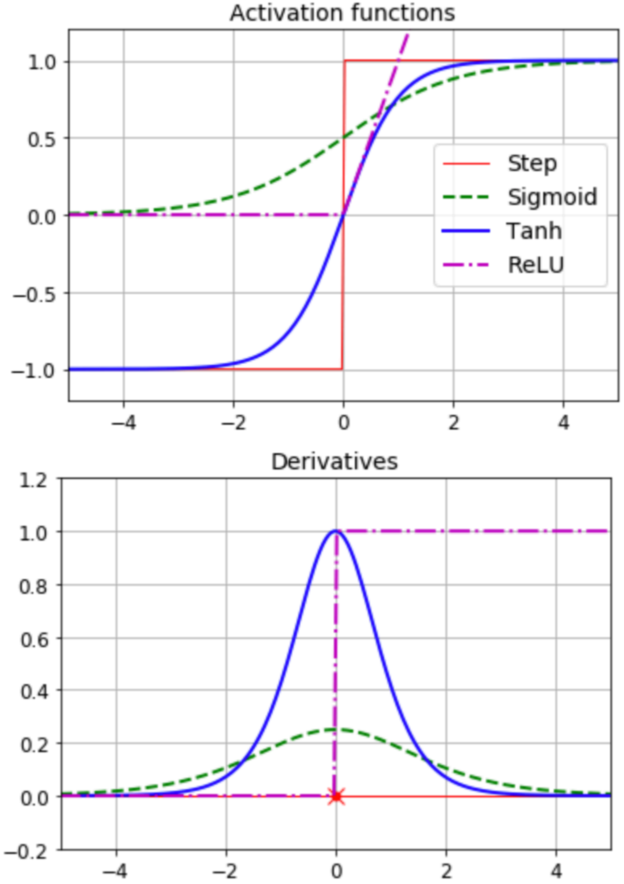
$$\sigma(z) = \frac{1}{1+exp(-z)}$$

- **Hyperbolic tangent function**:

$$\tanh(z) = 2\sigma(2z) - 1$$

- **Rectified linear unit (ReLU) function**:

$$ReLU(z) = max(0, z)$$



Softmax Activation Function

```
tf.keras.backend.clear_session

nonlinear_model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, input_shape=(2,), activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
])

nonlinear_model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=1),
                        loss='sparse_categorical_crossentropy')
```

$$\sigma(s(x))_k = \frac{exp(s_k(x))}{\sum_{j=i}^{K} exp(s_j(x))}$$

- K – number of classes
- $s(x)$ – vector of scores of each class for instance x
- $\sigma(s(x))_k$ – estimated probability that x belongs to class k

- An important activation function for classification problems (used not only in neural networks but also in multiclass classification methods in general)
- In neural network-based classifiers – commonly used in the final layer
- Normalises the output of a network to a probability distribution over output classes

Cross-Entropy Loss Function

```
tf.keras.backend.clear_session

nonlinear_model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, input_shape=(2,), activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
])

nonlinear_model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=1),
                        loss='sparse_categorical_crossentropy')
```

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(\hat{p}_k^{(i)})$$

- $y_k^{(i)}$ is equal to 1 if the target class for the i-th instance is k; otherwise 0
- $\hat{p}_k^{(i)}$ – estimated probability that x belongs to class k

- **Objective**: build a model that estimates a high probability for the target class (and a low probability for all other classes)
- Cross-entropy loss function penalises the model when it estimates a low probability for the target class

Useful Things to Know about Deep Learning

PyTorch

PyTorch was designed for **eager execution** from the very beginning – no symbolic graphs, operations are performed where they appear in the code.

Advantages of Symbolic Graphs

- Can be internally optimized
- Faster (in theory)
- Easily deployable, even across languages

Advantages of Eager Execution

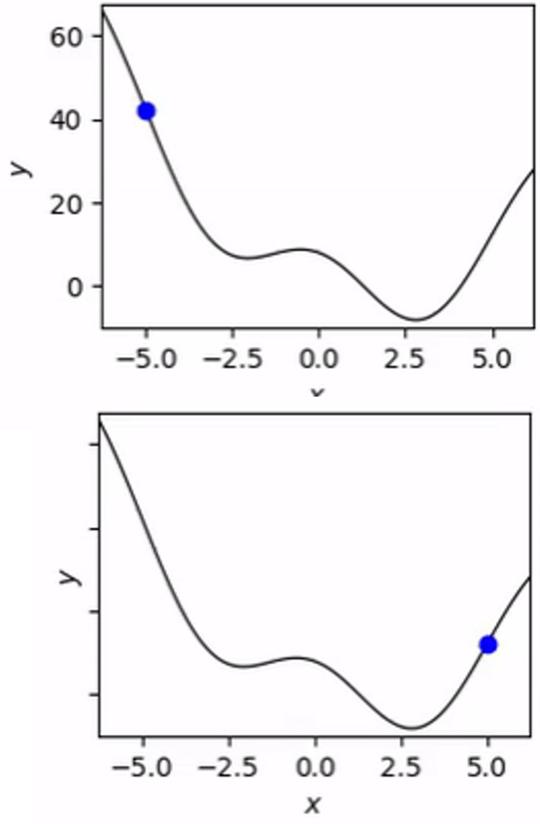
- Easier to understand
- Easier to debug
- Supports dynamic graphs

TensorFlow 2 also has **eager execution support**

Randomness in the Network

Different **random initializations** lead to different results.

Solution: Explicitly set the random seed.
All the random seeds!



Randomness in the Network

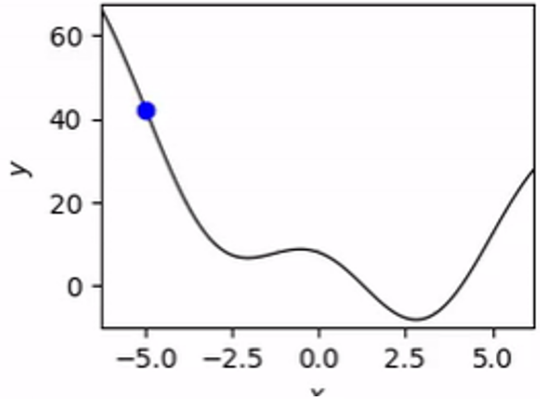
Different **random initializations** lead to different results.

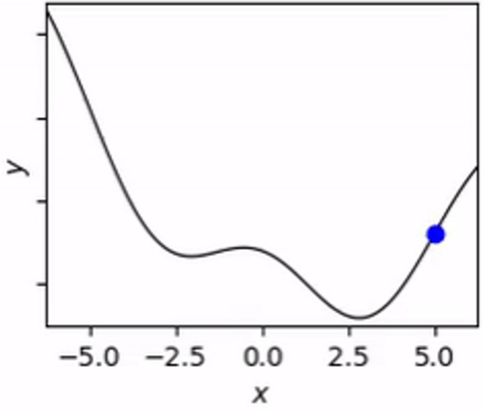
Solution: Explicitly set the random seed.
All the random seeds!

BUT!

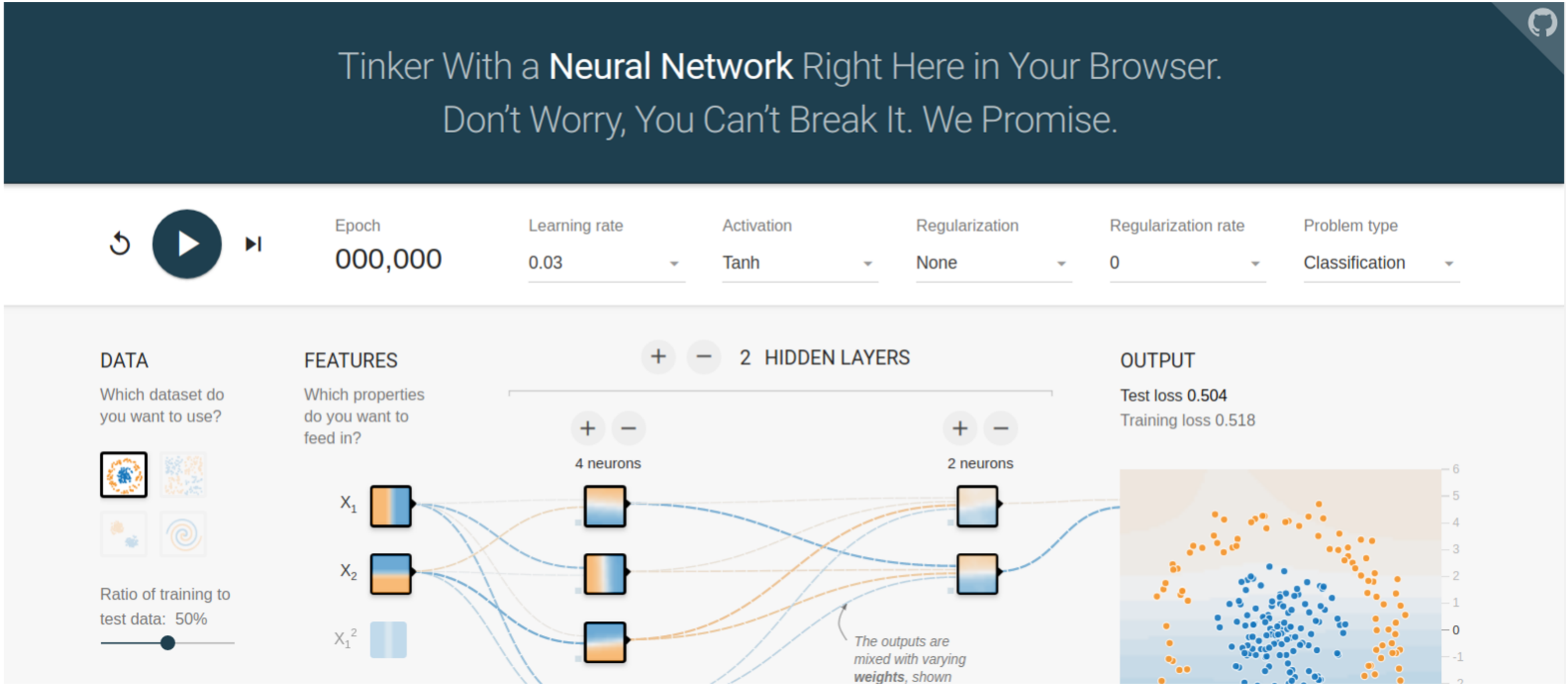
GPU threads finish in a random order, also leading to randomness!
Small rounding errors really add up!
Doesn't affect all operations.

Solution: Embrace randomness, run with different random seeds and report the average.





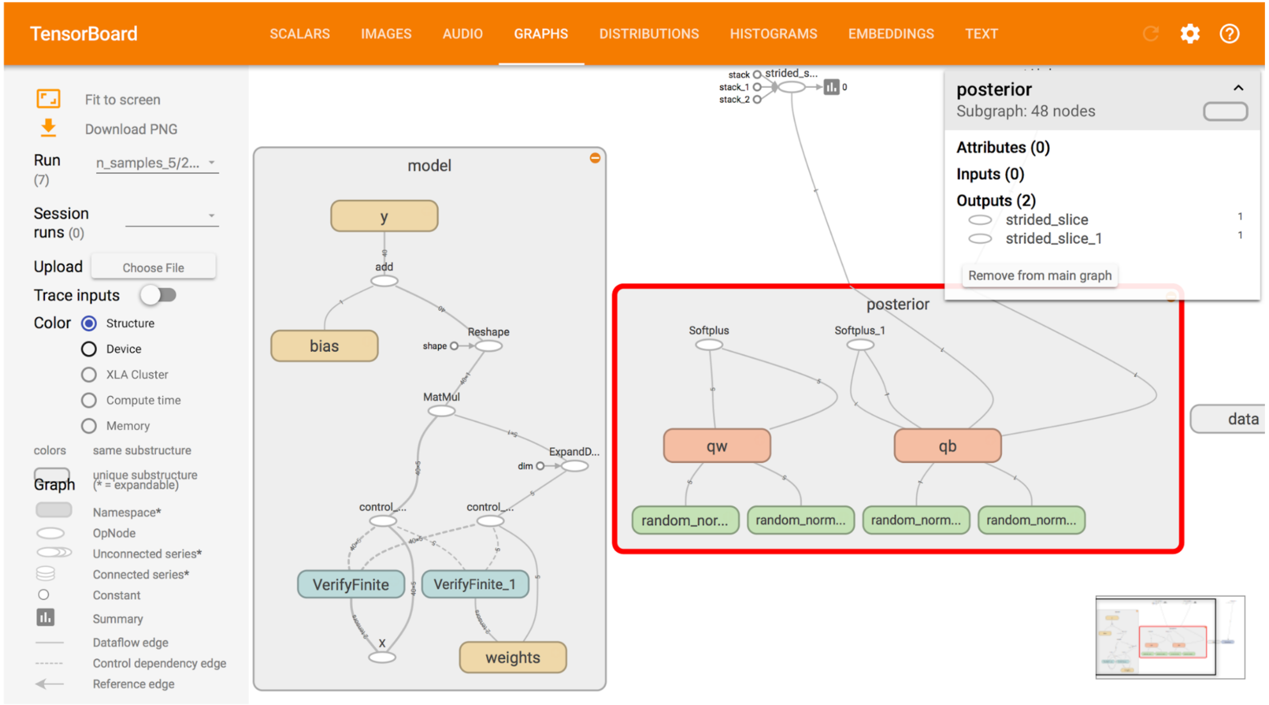
Tensorflow Playground



playground.tensorflow.org

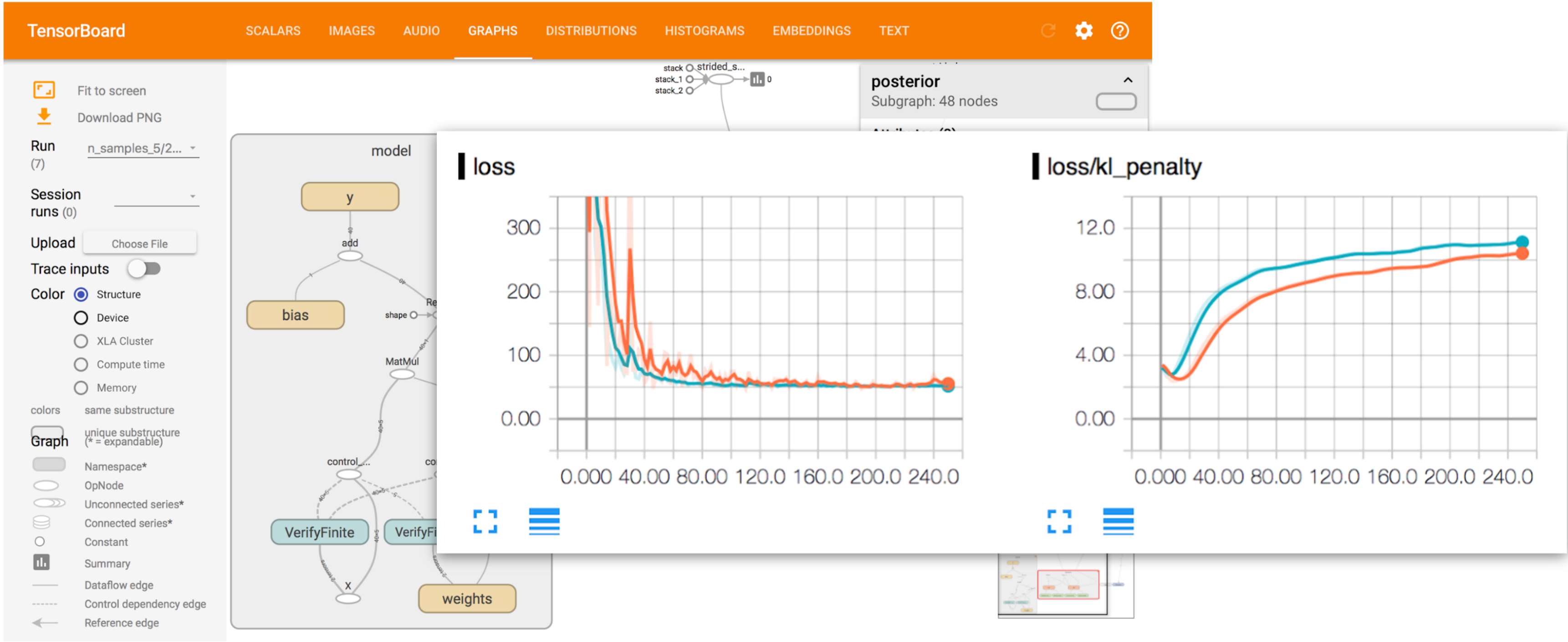
TensorBoard

A tool for **visualizing** your own Tensorflow networks.



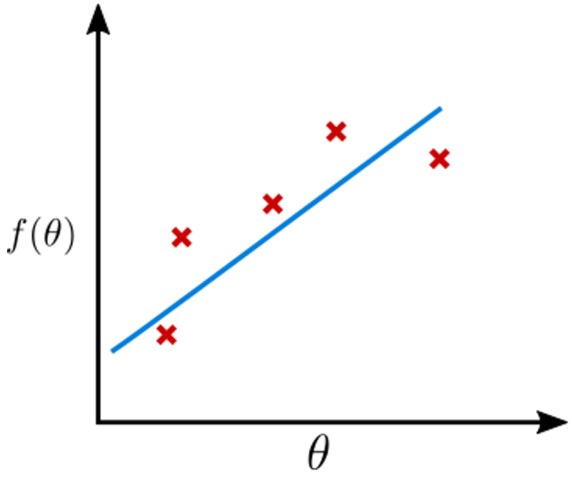
TensorBoard

A tool for **visualizing** your own Tensorflow networks.

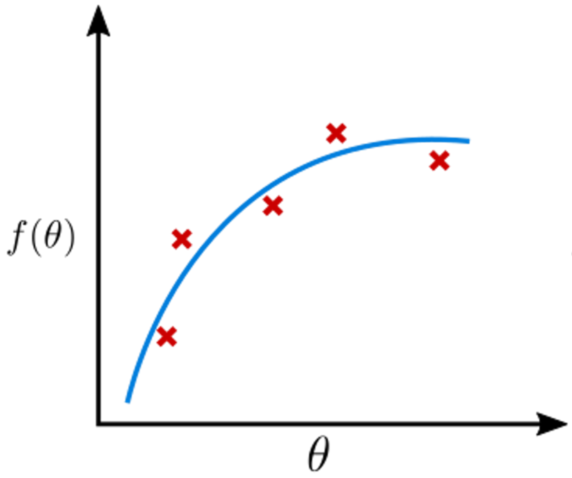


Fitting to the Data

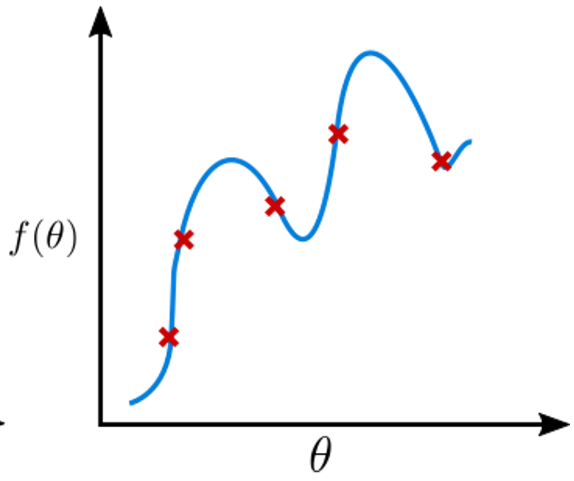
Underfitting
The model does not have the capacity to properly model the data.



Ideal fit



Overfitting
Too complex, the model memorizes the data, does not generalize.



Splitting the Dataset

In order to get realistic results for our experiments, we need to evaluate on a **held-out test set**.

Using a separate development set for choosing hyperparameters is even better.

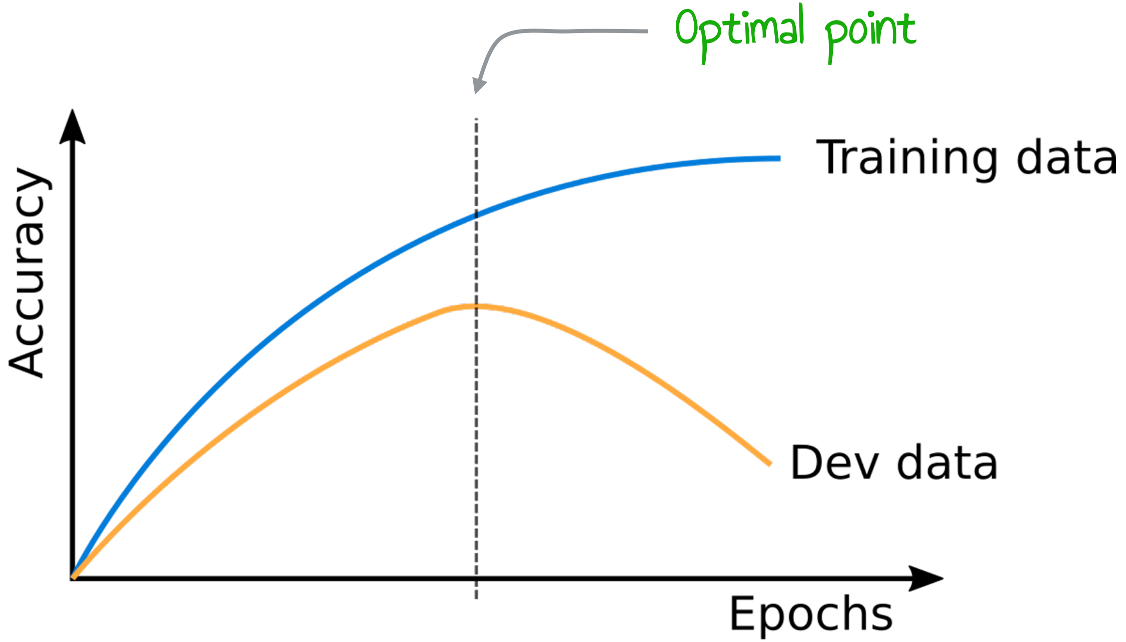
For training your models, fitting the parameters

For continuous evaluation and hyperparameter selection

For realistic evaluation once the training and tuning is done

Early Stopping

A sufficiently powerful model will keep improving on the training data until it **overfits**.
We can use the **development** data to choose when to stop.

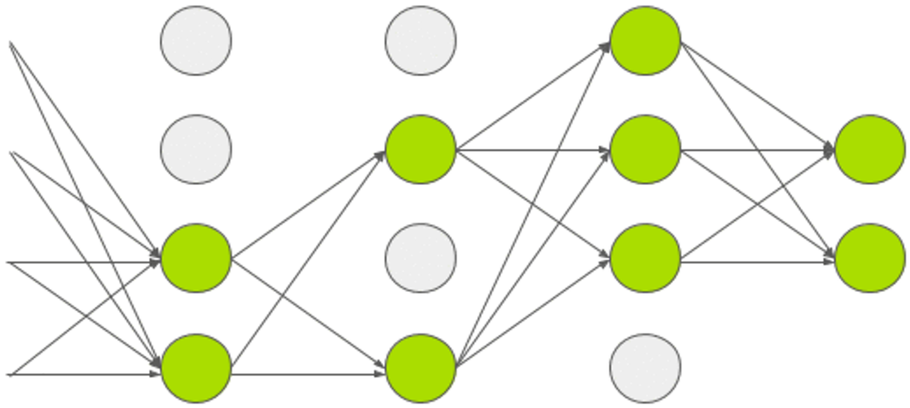


Dropout

During training, randomly set some activations to **zero**.

Typically **drop 50%** of activations in a layer

Form of regularization - prevents the network from **relying** on any one node.

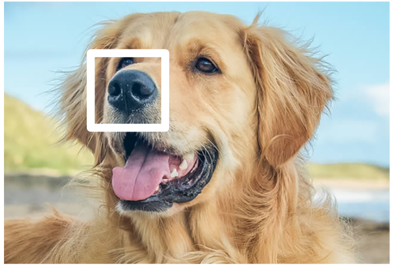


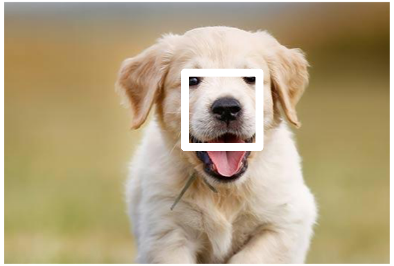
https://www.learnopencv.com/understanding-alexnet/

Next time: Convolutional Neural Networks

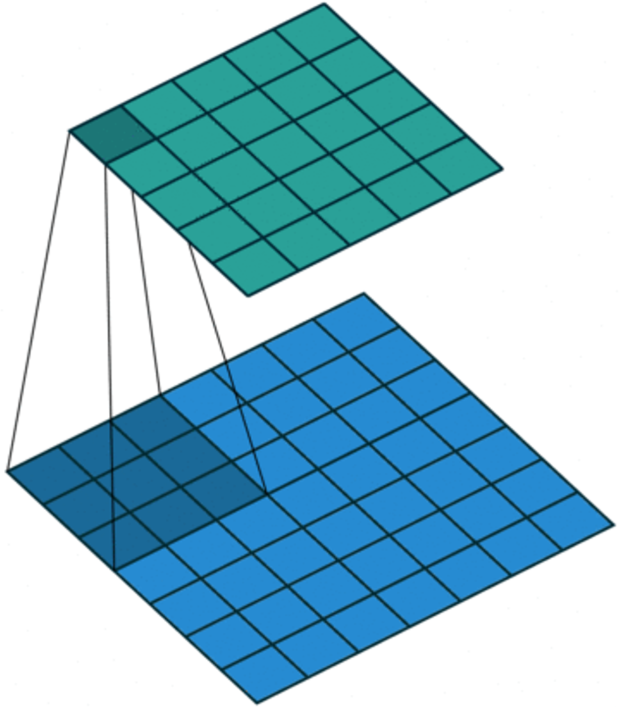
Neural modules operating **repeatedly** over different subsections of the input space.

Great when **searching** for feature patterns, without knowing where they might be located in the input.





The main driver in **image recognition**.
Can also be used for text.



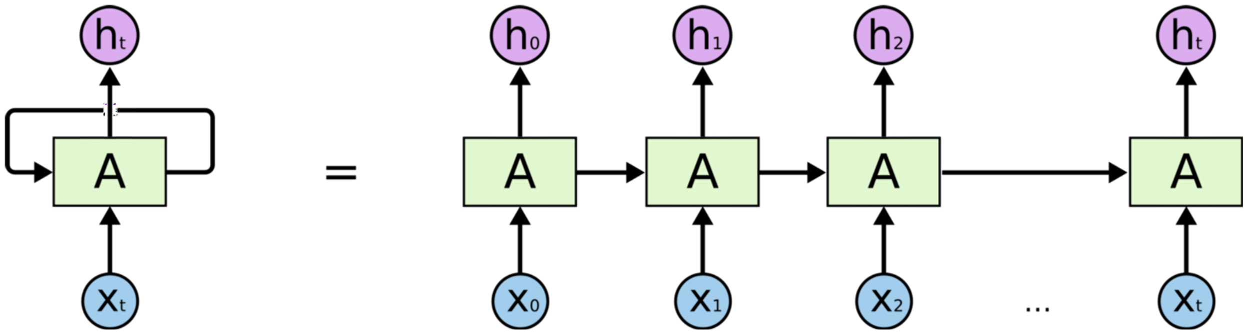
https://github.com/vdumoulin/conv_arithmetic

Next time: Recurrent Neural Networks

Designed to process **input sequences** of arbitrary length.

Each hidden state A is calculated based on the **current input** and the **previous hidden state**.

Main neural architecture for **processing text**, with each input being a word representation.



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Practical 4

Your task: Learning objectives

- The basics of running `TensorFlow`
- How to implement a feedforward neural network in Python
- How to visualise your network architecture using `TensorBoard` and track changes
- How to apply deep learning to both classification and regression tasks.

- **Assignment**: Build a neural classification model to predict “ocean proximity“ of a house (California House Prices Dataset)

- **Optional**: Visualise your network architecture, changes in loss and metrics, explore the results (e.g., print out and visualise confusion matrices), compare to more “traditional“ ML models from previous practicals

Practical 4 Logistics

- Data and code for Practical 4 can be found on: Github (https://github.com/ekochmar/cl-datasci-pnp-2021/tree/main/DSPNP_practical4)

- Practical (‘ticking’) session over Zoom at the time allocated by your demonstrator

- At the practical, be prepared to discuss the task and answer the questions about the code to get a ‘pass’

- Upload your solutions (Jupyter notebook or Python code) to Moodle by the deadline (Tuesday 24 November, 4pm)