DaST
Data • (Systems+Theory)

University of Zurich[UZH]

FDS22 Practical 3

DaST Team

Foundations of Data Science

Practical 3 Tasks

The deadline for Practical 3 has been moved to **04/01/2023**, end of day

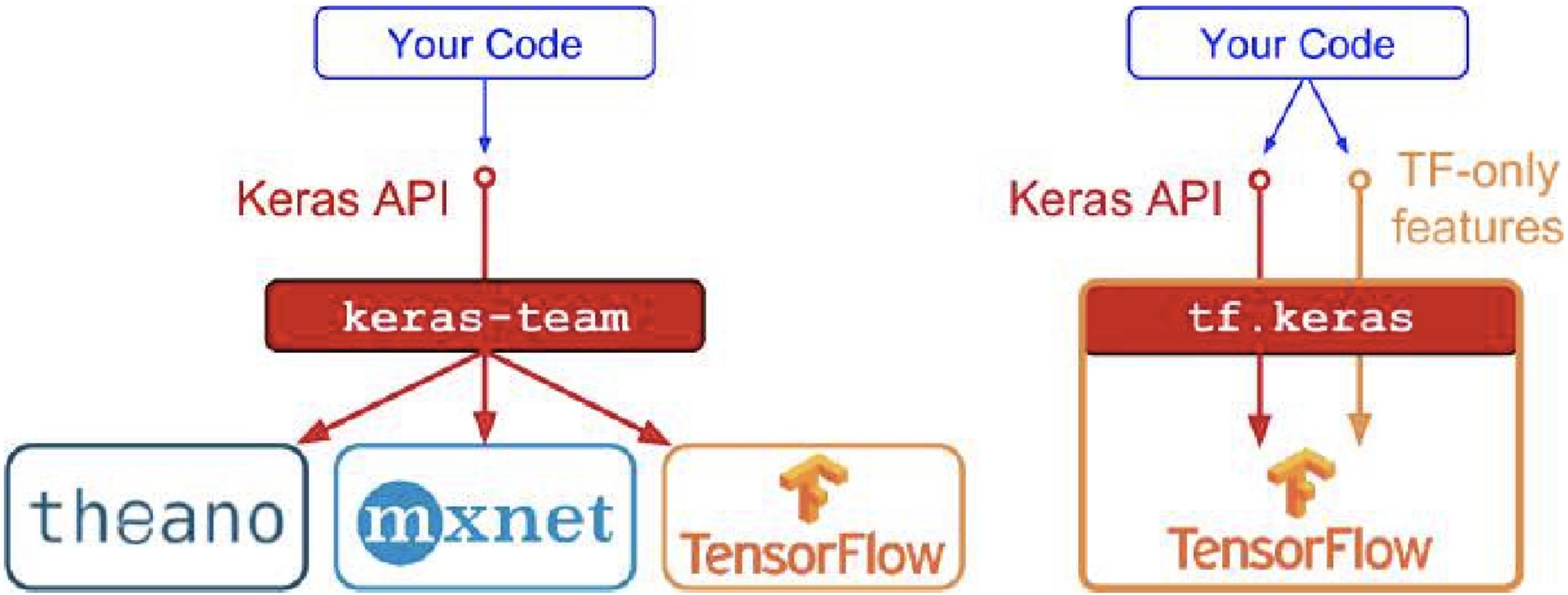
Task 1: Build a Convolutional Neural Network (CNN) for **MNIST** dataset

- Classify handwritten digits
- Achieve a min accuracy of 97% to pass

Task 2: Use Transfer Learning to build a CNN for **CIFAR-10** dataset

- Classify images
- Achieve a min accuracy of 75% to pass
- Achieve an accuracy of above 85% to get **5 bonus points**

TensorFlow and Keras



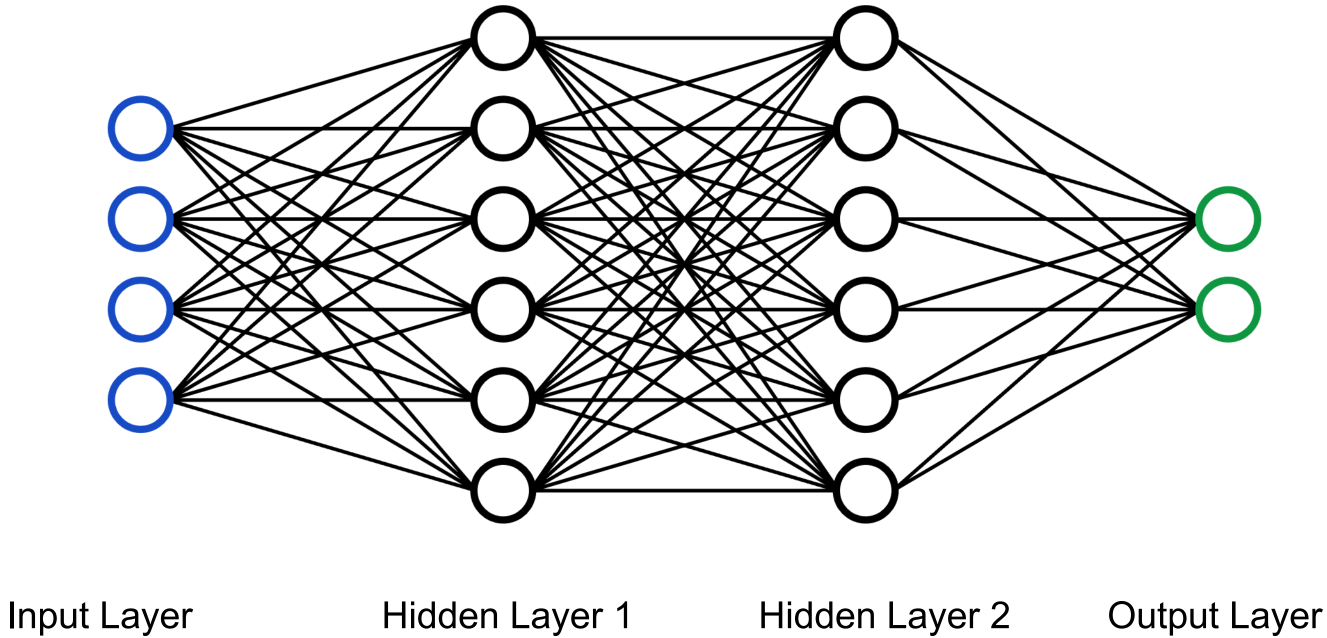
Setup

Google Colab is recommended for this practical

- Google Colab has TensorFlow installed
- Enable GPU/TPU for the Colab Notebook
    - Navigate to Edit -> Notebook Settings
    - Select TPU from the Hardware Accelerator drop-down
- https://colab.research.google.com/notebooks/tpu.ipynb#scrollTo=_pQCOmISAQBu

Notebook: Build a Neural Network using Keras

Neural Network Structure



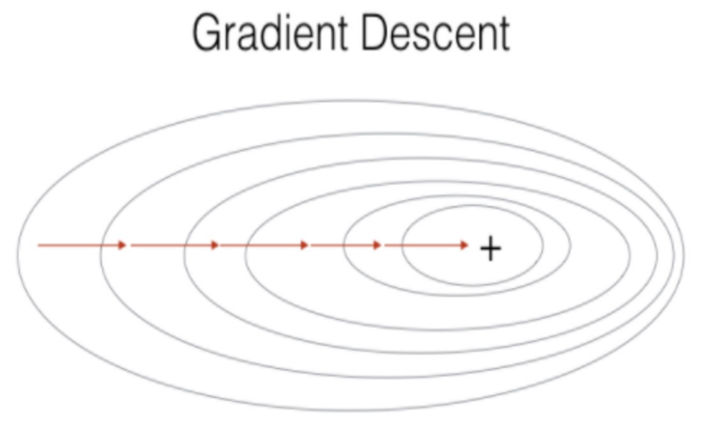
Batch Size and Epochs

Gradient Descent

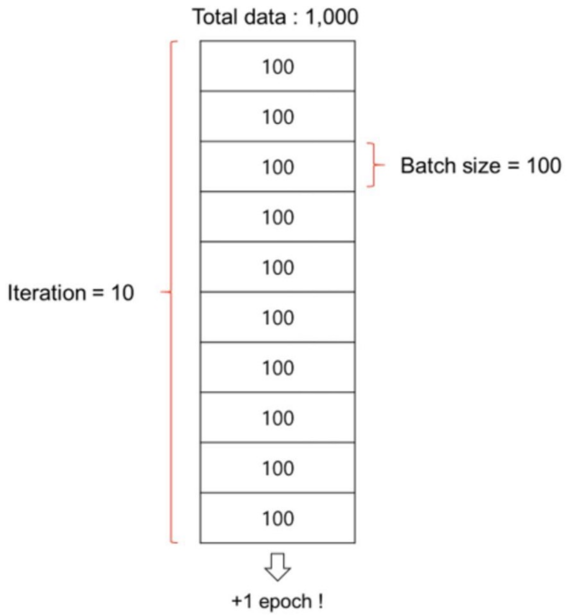
- $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{g}_t$
- Computed over the whole dataset

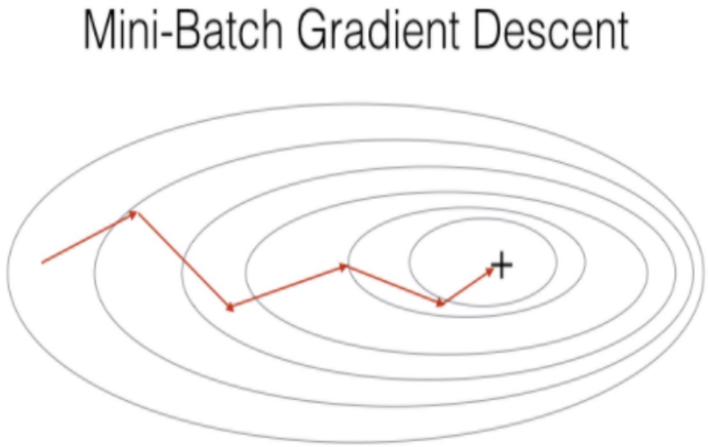
Hard to fit in the memory for large inputs

=> mini-batch gradient descent



Batch Size and Epochs





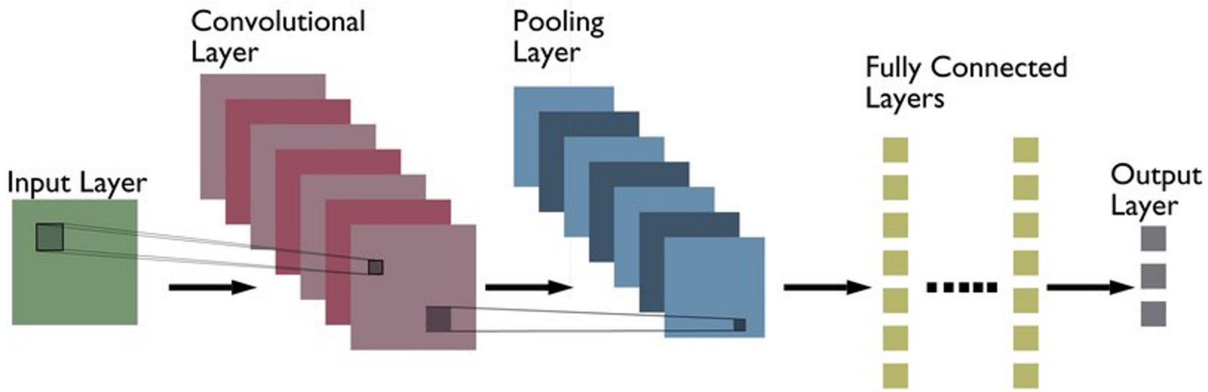
Convolutional Neural Networks

Two important building blocks of CNN

- Convolutional layers
- Pooling layers

Typical CNN architecture

- Input layer
- A few convolutional layers (+ReLU)
- A pooling layer
- A few convolutional layers (+ReLU)
- A pooling layer
- ...
- Fully connected layers (+ReLU)
- Output layer



Source: Yazdani Abyaneh, Amir Hossein & Hossein Gharari, Ali & Pourahmadi, Vahid. (2018). Deep Neural Networks Meet CSI-Based Authentication.

Hyperparameters

- Number of hidden layers
- Number of neurons in each layer
- Activation function
- Weight initialisation strategies
- Learning rate
- Batch size
- Number of epochs
- Optimiser
- More…

What combination of the hyperparameters is the best for your task?

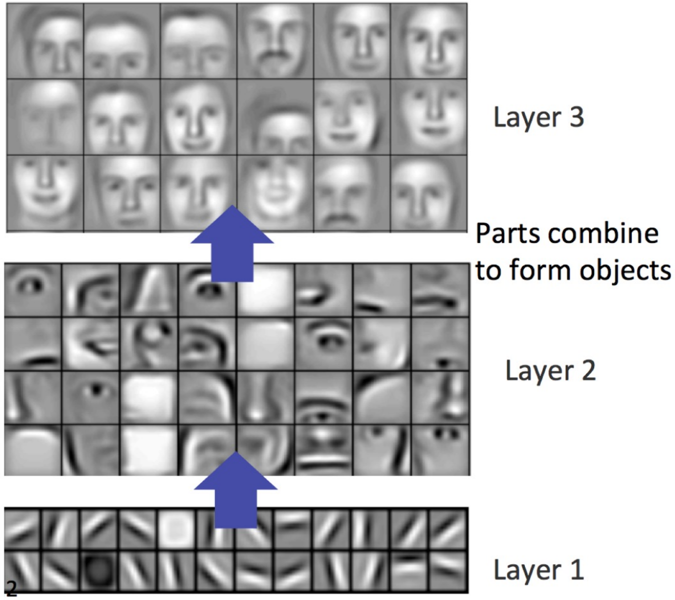
Try different combinations on the validation set

- GridSearchCV
  - Try all combinations of hyperparameters in the search space
- RandomizedSearchCV
  - A fixed number of combinations of hyperparameters in the search space are sampled

- Optimisation techniques to explore a search space more efficiently rather randomly
  - When a region of the space turns out to be good, it should be explored more
  - Hyperopt: Distributed hyperparameter optimisation
    - https://github.com/hyperopt/hyperopt
    - Hyperopt is a Python library for serial and parallel optimization over awkward search spaces, which may include real-valued, discrete, and conditional dimensions.
  - Hyperas: Hyperopt + Keras
    - https://github.com/maxpumperla/hyperas

Number of Hidden Layers

- A single hidden layer can theoretically model any complex functions
- But it is not very **parameter efficient**: Much more neurons will be required in the single layer
- Deep networks have higher parameter efficiency
- Deep neural networks automatically structure the features in an hierarchical way
  - Lower hidden layers model low-level structures
  - Intermediate layers combine these low-level structures to model intermediate-level structures
  - Highest layers combine intermediate structures to model high-level structures
  - Lower-level structures are shared => higher parameter efficiency

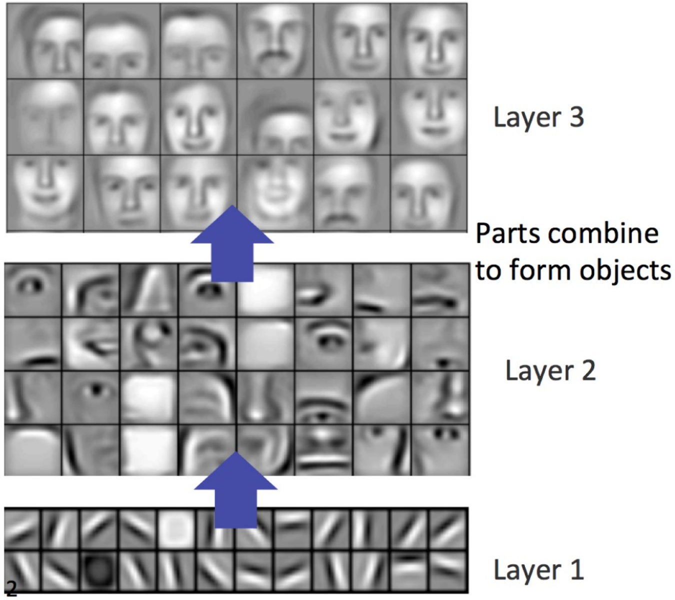


Source: Albawi, Saad & Abed Mohammed, Tareq & ALZAWI, Saad. (2017). Understanding of a Convolutional Neural Network. 10.1109/ICEngTechnol.2017.8308186.

Number of Hidden Layers

Practical Strategy

- For simple problems, start with one or two hidden layers
- For complex problems, increase the number of hidden layers until overfitting
- For even more complex problems, use “transfer learning”



Source: Albawi, Saad & Abed Mohammed, Tareq & ALZAWI, Saad. (2017). Understanding of a Convolutional Neural Network. 10.1109/ICEngTechnol.2017.8308186.

Number of Neurons per Hidden Layer

Past common practice:

- Form a pyramid
- E.g., 300 neurons in the first layer, 200 neurons in the second layer, and 100 neurons in the third layer

New trend: same number of neurons in all hidden layers (except the first layer, which has more neurons than the others)

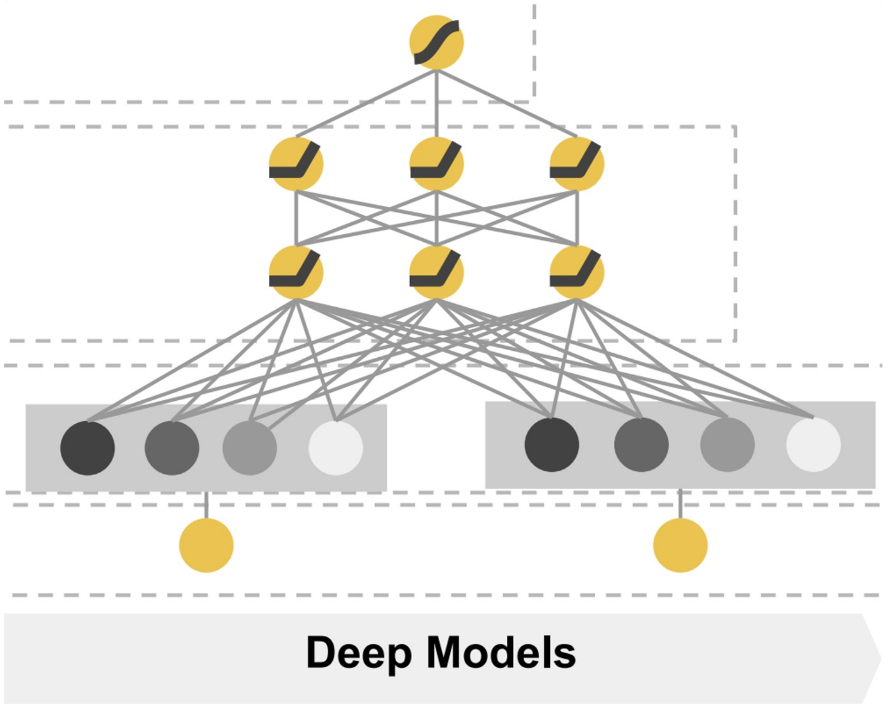
In practice:

- Increase the number of neurons until overfitting
- “Stretching pants” strategy: Pick a model with more layers and neurons than you actually need, then use regularisation techniques to prevent it from overfitting

Non-sequential models: Wide and Deep Models

Source: https://arxiv.org/abs/1606.07792

- Performs well for generic large-scale regression and classification problems: recommender systems, search, and ranking problems
- **Deep neural networks** can learn complex structures through the combinations of lower-level structures

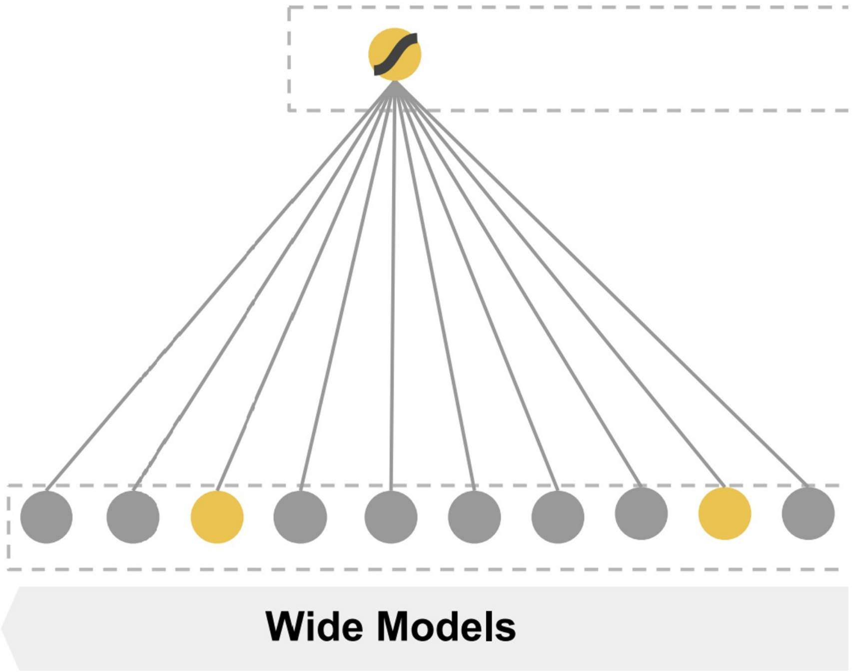


Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." *Proceedings of the 1st workshop on deep learning for recommender systems*. 2016.

Non-sequential models: Wide and Deep Models

Source: https://arxiv.org/abs/1606.07792

- Performs well for generic large-scale regression and classification problems: recommender systems, search, and ranking problems
- **Deep neural networks** can learn complex structures through the combinations of lower-level structures
- **Wide linear models** can effectively memorise sparse feature interactions using cross-product feature transformation

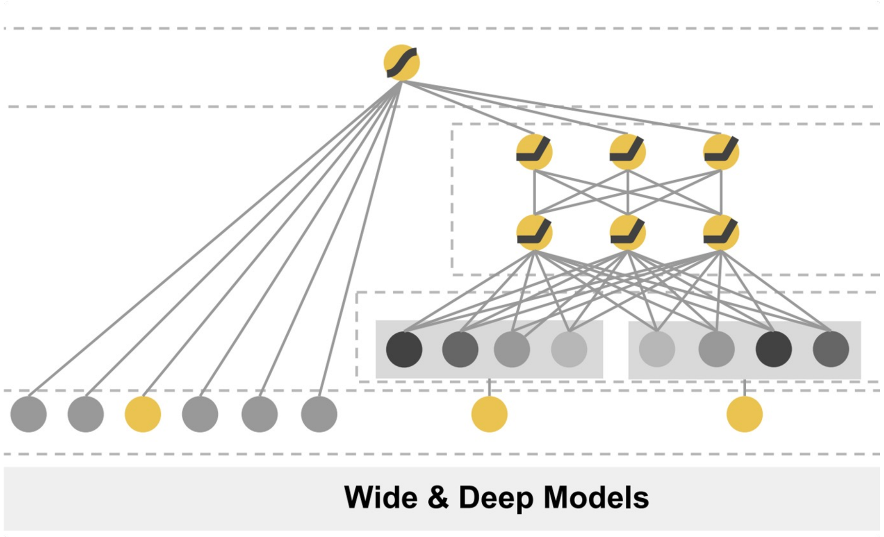


Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." *Proceedings of the 1st workshop on deep learning for recommender systems*. 2016.

Non-sequential models: Wide and Deep Models

Source: https://arxiv.org/abs/1606.07792

- Performs well for generic large-scale regression and classification problems: recommender systems, search, and ranking problems
- **Deep neural networks** can learn complex structures through the combinations of lower-level structures
- **Wide linear models** can effectively memorise sparse feature interactions using cross-product feature transformation
- Combine the strengths of both types of models



Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." *Proceedings of the 1st workshop on deep learning for recommender systems*. 2016.

Notebook: Build a Non-Sequential Model

Difficulties in Training Deep Neural Networks

- Vanishing Gradients
  - Weight Initialisation Strategies
  - Nonsaturating Activation Functions
  - Batch Normalisation
- Lack of training data for a large network
  - Transfer Learning
- Training may be extremely slow
  - Faster Optimisers
- Overfitting
  - Early Stopping
  - Data Augmentation
  - Dropout

Vanishing Gradients

- Gradients often get smaller and smaller as the algorithm progresses to the deeper layers
- Never converges to a good solution
- One of the reasons deep neural networks were mostly abandoned in early 2000s

Understanding the difficulty of training deep feedforward neural networks
Xavier Glorot and Yoshua Bengio

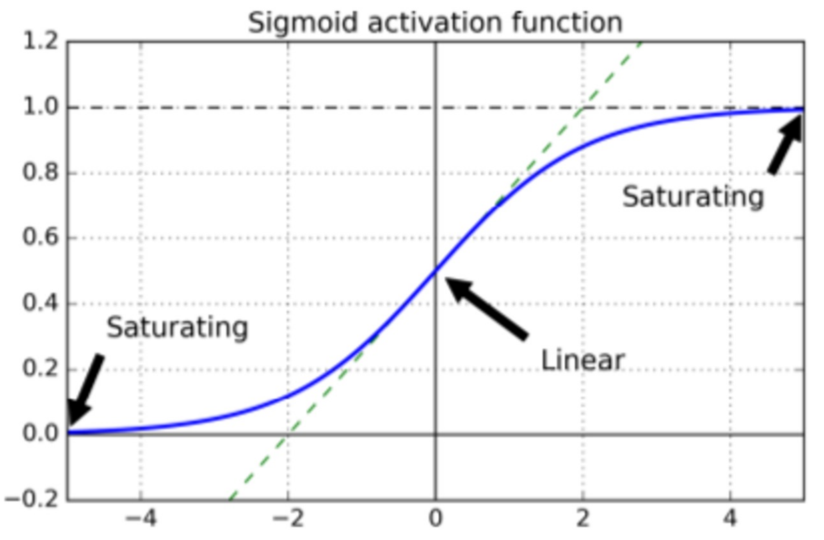
- Published in 2010
- Cited more than 10,000 times

Vanishing Gradients

At the time, the popular activation function and the weight initialisation strategy are

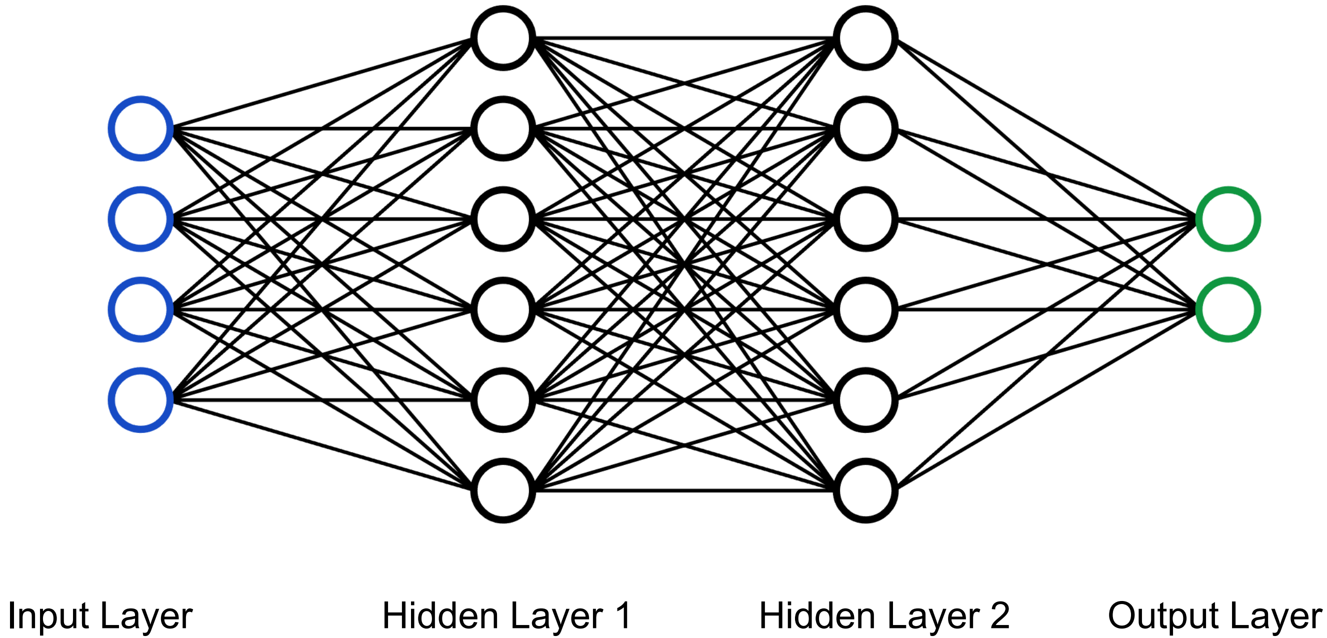
- Sigmoid function
- Weights are initialised to have mean = 0 and standard deviation = 1

- The variance of the outputs of each layer is much greater than the variance of its inputs
- Going deeper, the variance keeps increasing until saturation



Source: Aurlien Gron. 2017. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (1st. ed.). O'Reilly Media, Inc.

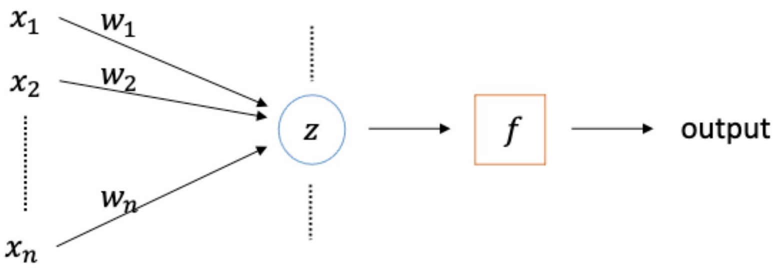
Vanishing Gradients



Glorot and He Initialisation

Glorot (Xavier) initialisation

- The variance of the output of a neuron is determined by the number of weights connected from the previous layer
- We should reduce the variance
- Initialise the weights to have mean of 0 and variance of 1 / n, where n is the number of weights connected to this node from previous layer
- By default, Keras uses Glorot initialisation



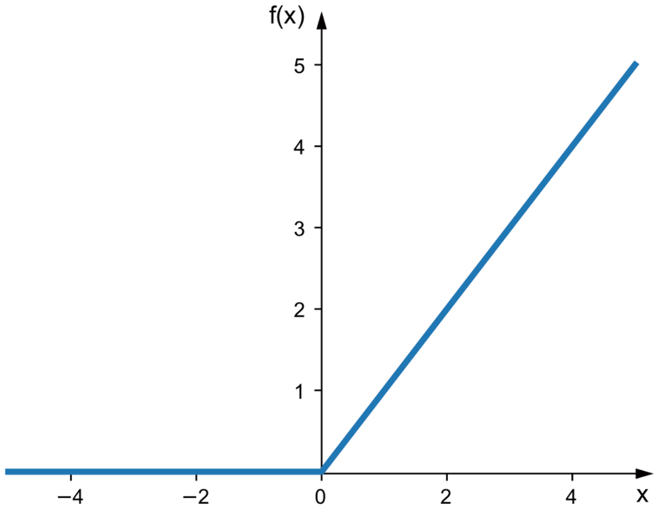
$$z = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

He initialisation is a variant of Glorot initialisation designed for the ReLU activation function and its variants

Nonsaturating Activation Functions

ReLU

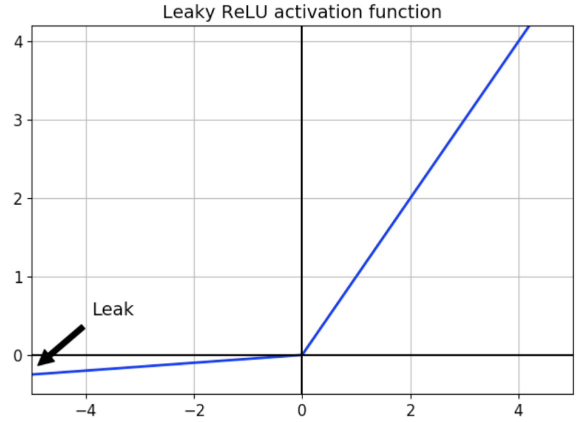
- Does not staurate for positive values
- Fast to compute
- Dying ReLUs
  - Neurons keep outputting zeros
  - Sometimes half of the neurons in the network are died
  - The gradient of the ReLU function is zero when its input is negative

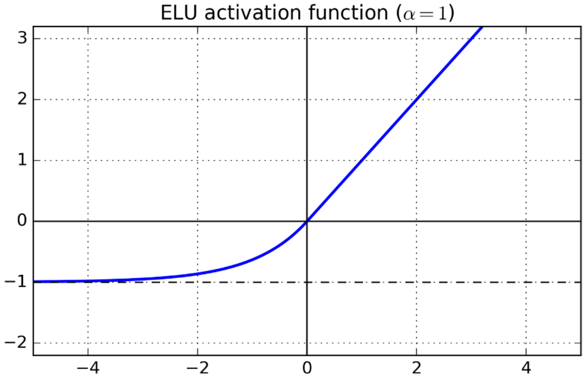


Nonsaturating Activation Functions

Variants of ReLU were proposed

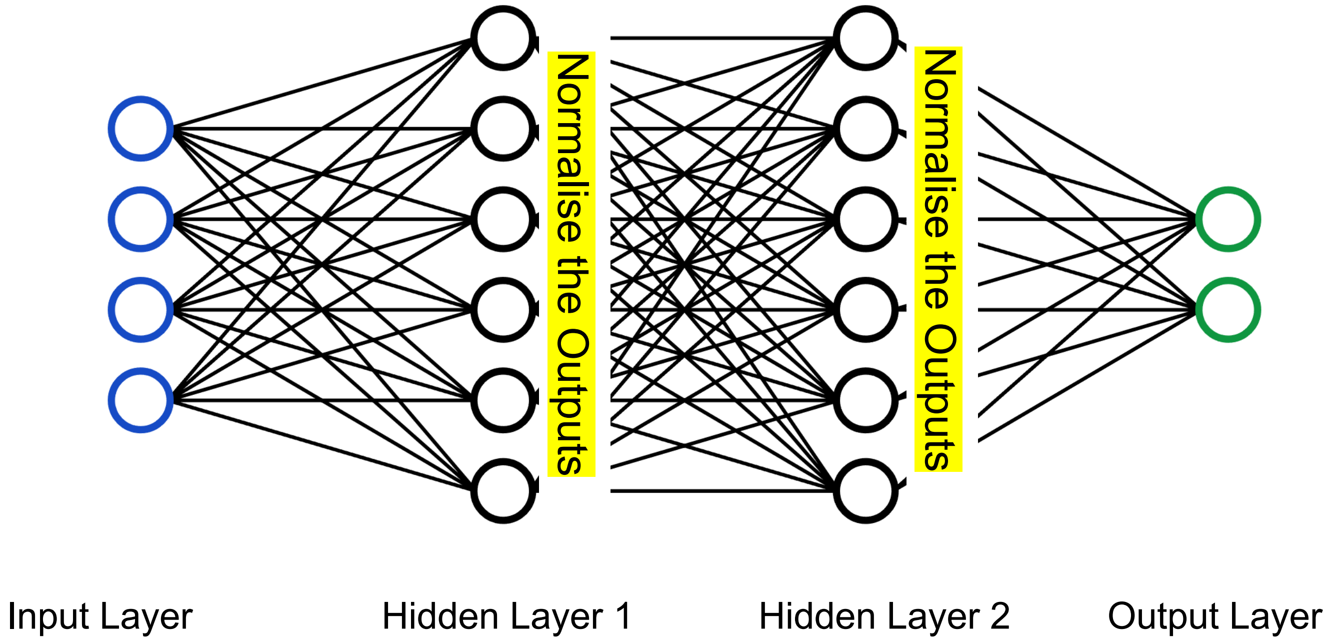
- Leaky ReLU
  - Use the hyperparameter to adjust the slope of the leak
- Randomised leaky ReLU
  - The slope hyperparameter is picked randomly in a given range
- Exponential linear unit (ELU)
  - Outperformed all ReLU variants in the authors’ experiments
  - Computation of ELU is slower than that of ReLU and its variances





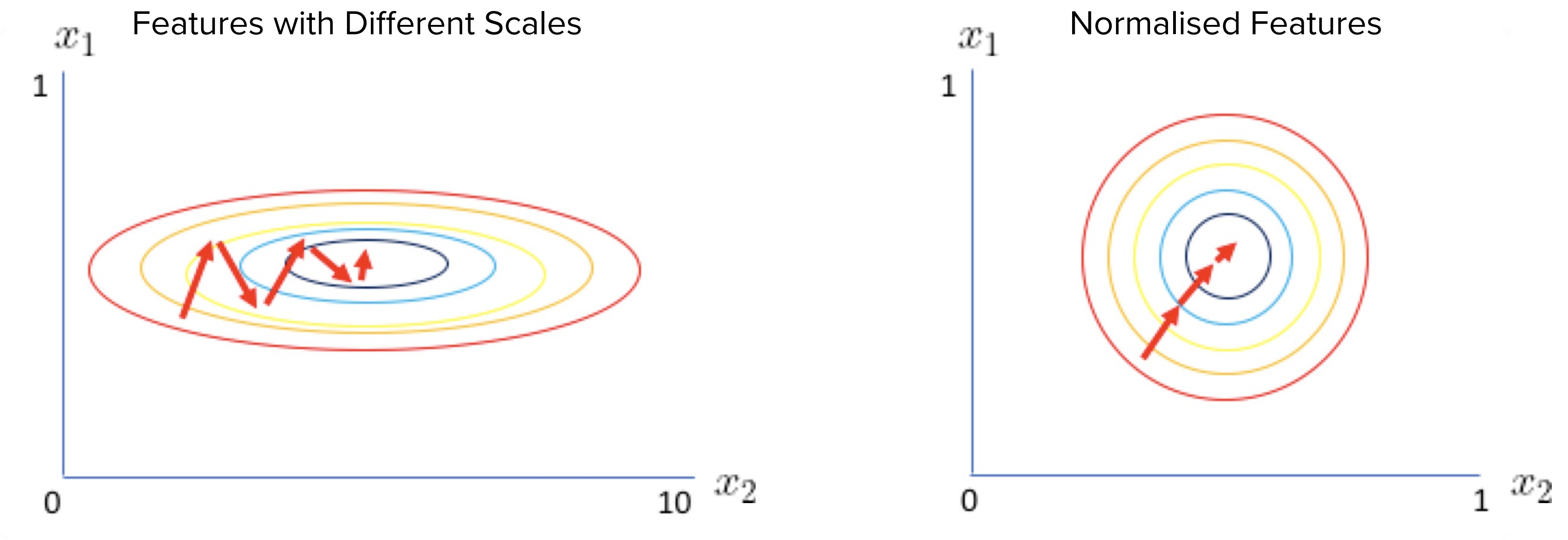
Notebook: Weight Initialisation and Activation Functions

Batch Normalisation



Batch Normalisation

- Normalise the outputs of every hidden layer



Batch Normalisation

- Normalise the outputs of every hidden layer
- Why normalisation?
  - Reduces vanishing gradients problem
  - Speeds up the learning

- Increases the complexity of the model
  - Slower training and predictions

Notebook: BatchNormalisation Layers

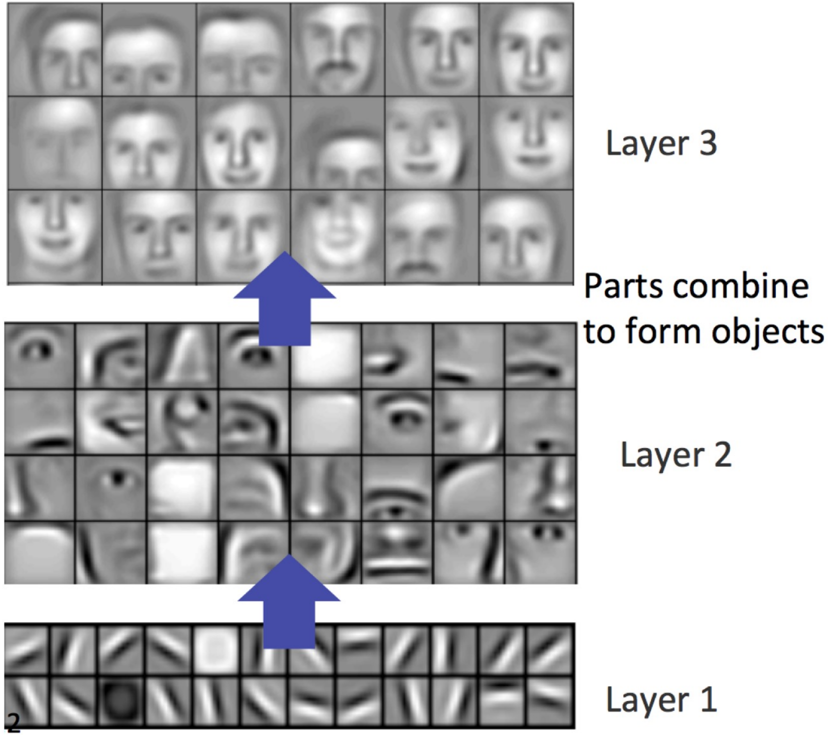
Difficulties in Training Deep Neural Networks

- Vanishing / Exploding Gradients
  - Weight Initialisation Strategies
  - Nonsaturating Activation Functions
  - Batch Normalisation
- **Lack of training data for a large network**
  - Transfer Learning
- Training may be extremely slow
  - Faster Optimisers
- Overfitting
  - Early Stopping
  - Data Augmentation
  - Dropout

Transfer Learning

Instead of training a very large and deep neural network

- Try to find an existing neural network that accomplishes a similar task
- Reuse the **lower layers** of the network



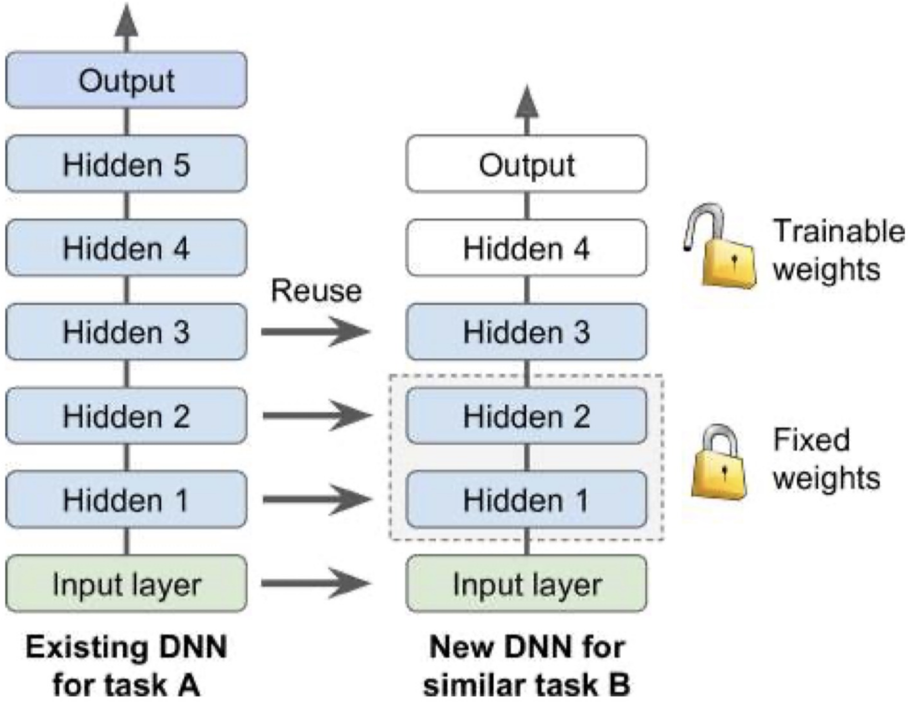
Transfer Learning

Instead of training a very large and deep neural network

- Try to find an existing neural network that accomplishes a similar task
- Reuse the **lower layers** of the network

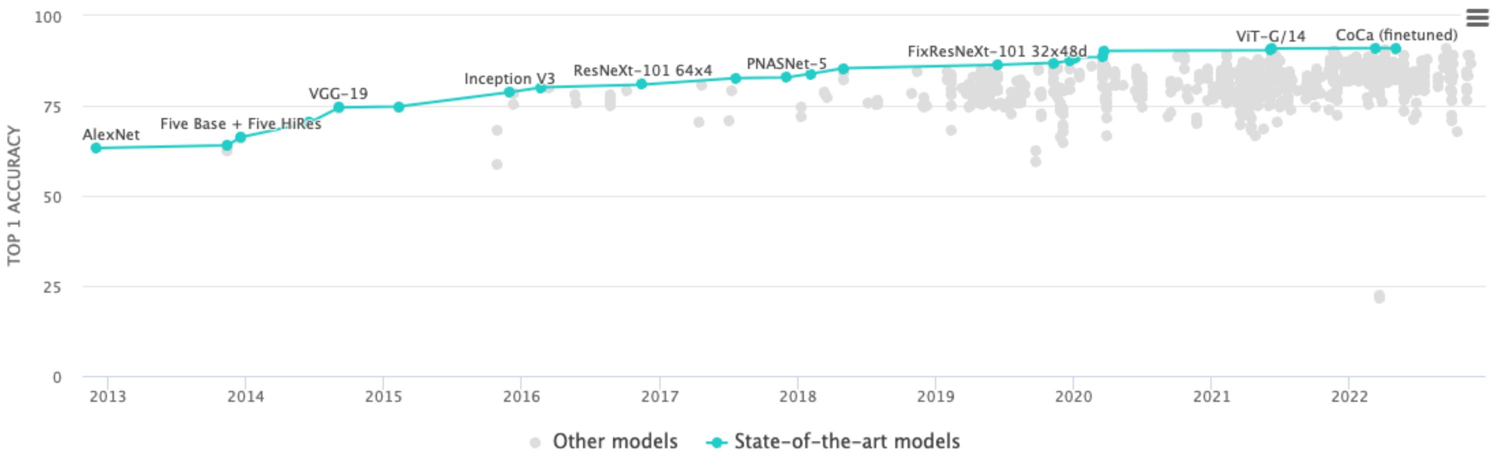
Transfer Learning

- Output layer of the original model should usually be replaced by the output layer for the new task
- Lower hidden layers are more useful
    - They learn lower-level features
- Find the right number of layers to reuse



CNN architectures

- ILSVRC ImageNet challenge
  - http://image-net.org/
  - https://www.kaggle.com/c/imagenet-object-localization-challenge
  - Classify images into 1000 classes



Source: https://paperswithcode.com/sota/image-classification-on-imagenet

CNN architectures

- ILSVRC ImageNet challenge
  - http://image-net.org/
  - https://www.kaggle.com/c/imagenet-object-localization-challenge
  - Classify images into 1000 classes

- Keras Applications
  - Deep learning models that are made available alongside pre-trained weights
  - https://keras.io/api/applications/

Notebook: TransferLearning

Difficulties in Training Deep Neural Networks

- Vanishing / Exploding Gradients
  - Weight Initialisation Strategies
  - Nonsaturating Activation Functions
  - Batch Normalisation
- Lack of training data for a large network
  - Transfer Learning
- **Training may be extremely slow**
  - Faster Optimisers
- Overfitting
  - Early Stopping
  - Data Augmentation
  - Dropout

Training may be extremely slow

Fast optimisers: Less steps to converge to the optimal

- Momentum
- Adaptive learning rates: AdGrad, RMSProp
- Momentum + Adaptive learning rates: Adam

Momentum

Gradient Descent

- Uses gradient for speed

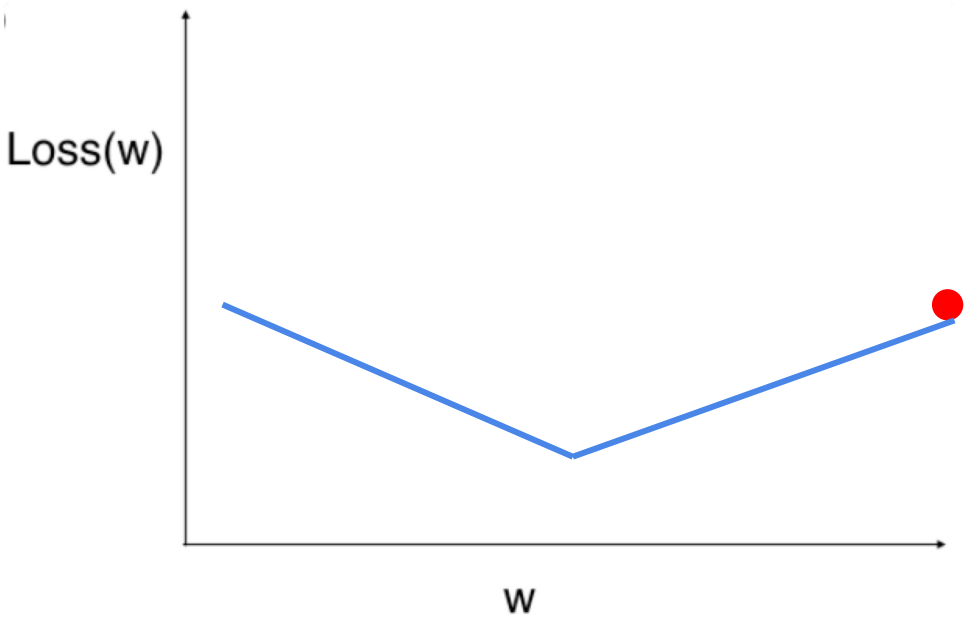
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Momentum

- Cares what previous gradients were
- Uses gradient for **acceleration**

$$\mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$$



Momentum

Gradient Descent

- Uses gradient for speed

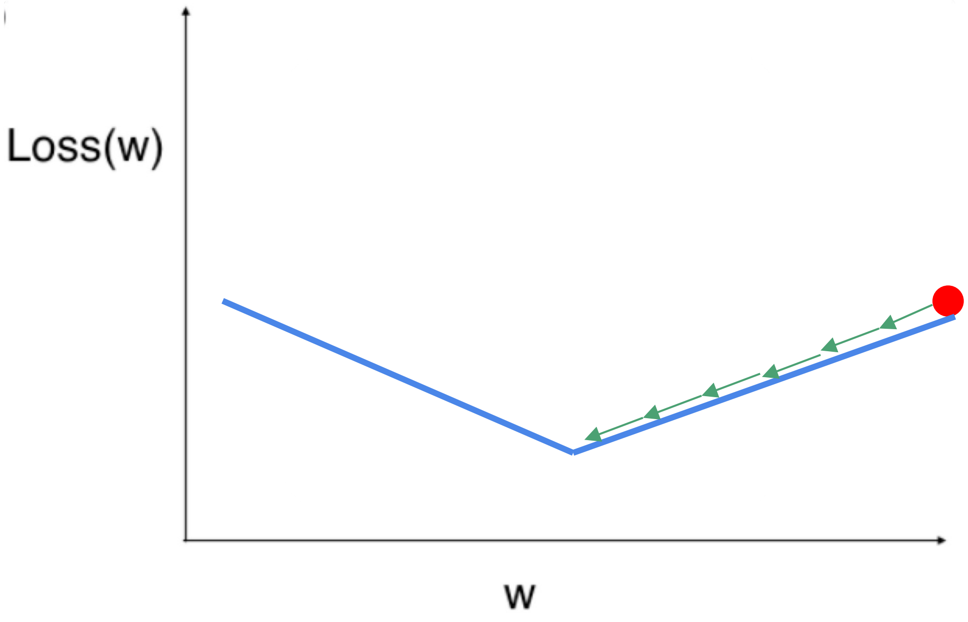
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Momentum

- Cares what previous gradients were
- Uses gradient for **acceleration**

$$\mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$$



Momentum

Gradient Descent

- Uses gradient for speed

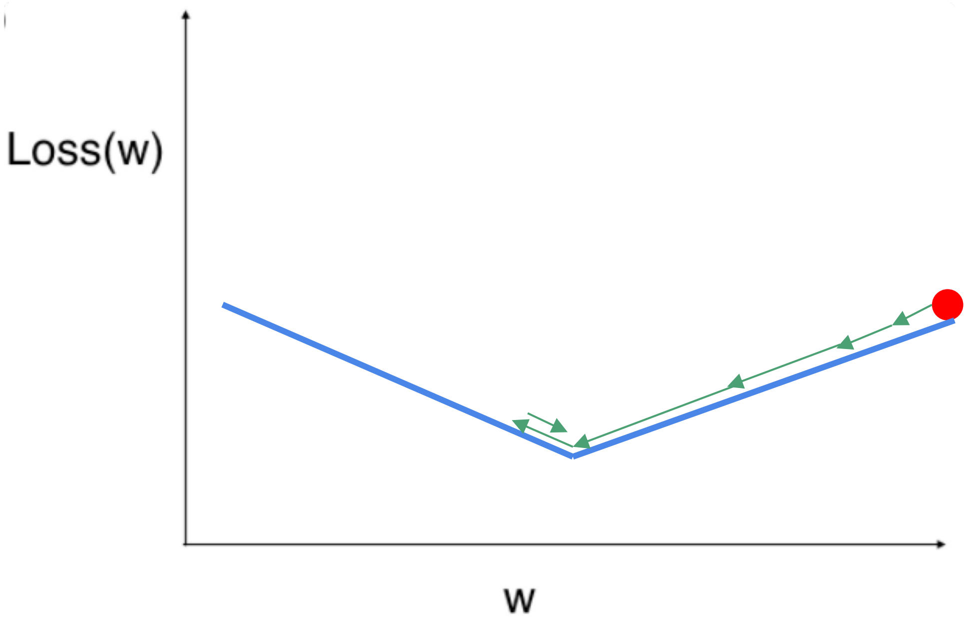
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Momentum

- Cares what previous gradients were
- Uses gradient for **acceleration**

$$\mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$$



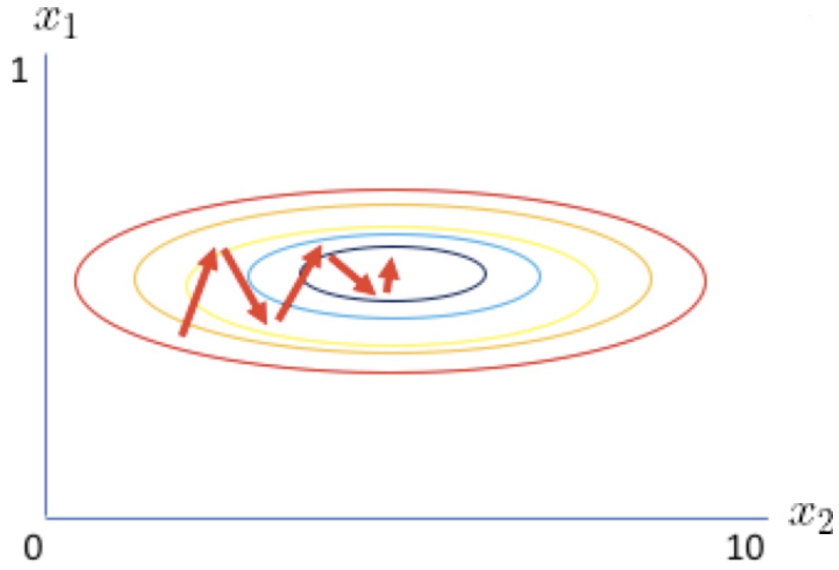
Adaptive Learning Rates

Consider a 2d optimization problem

- One feature has a large scale while the other one has a small scale

We need a learning rate adapts to each dimension

- Smaller learning rates for features with small scales while larger learning rates for features with larger scales
- AdaGrad
- RMSProp



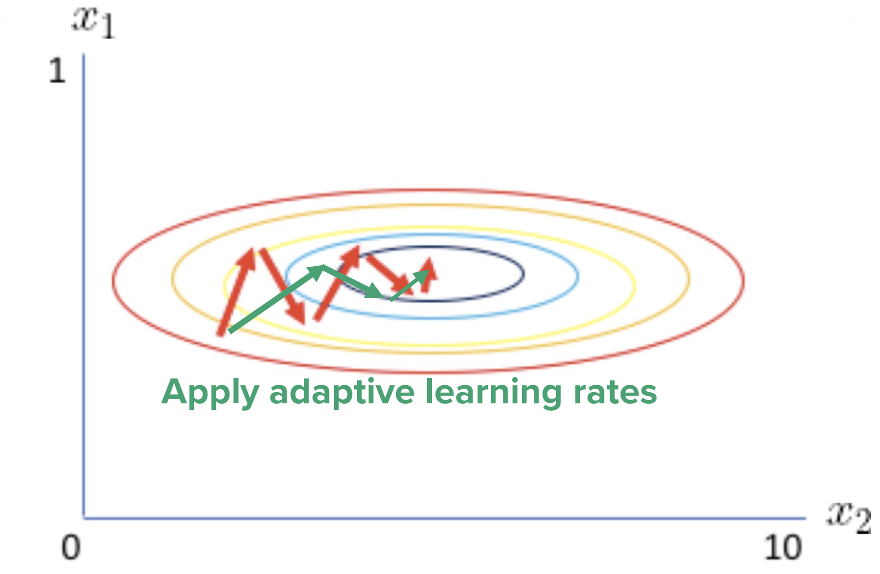
Adaptive Learning Rates

Consider a 2d optimization problem

- One feature has a large scale while the other one has a small scale

We need a learning rate adapts to each dimension

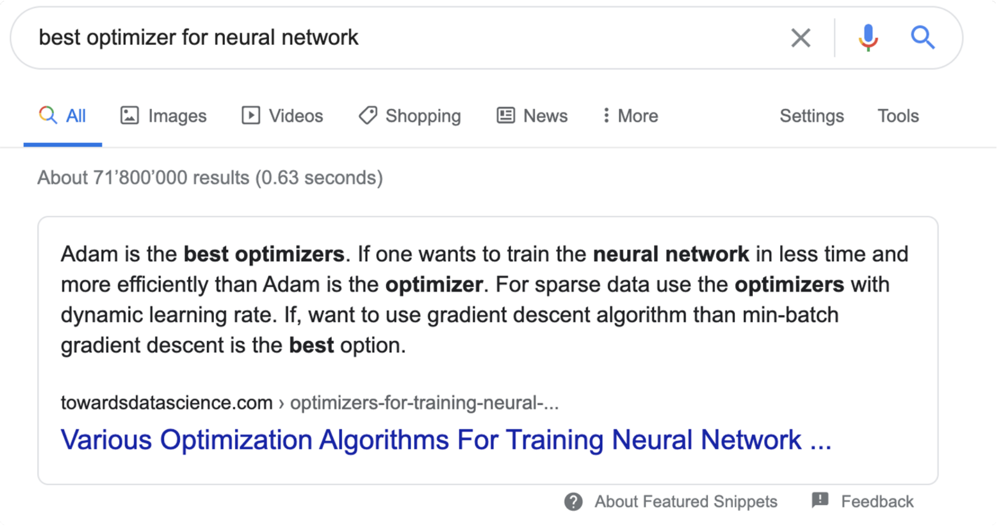
- Smaller learning rates for features with small scales while larger learning rates for features with larger scales
- AdaGrad
- RMSProp



Momentum + Adaptive Learning Rates

Adam: Adaptive moment estimation

- Combines the ideas of momentum optimisation and adaptive learning rates

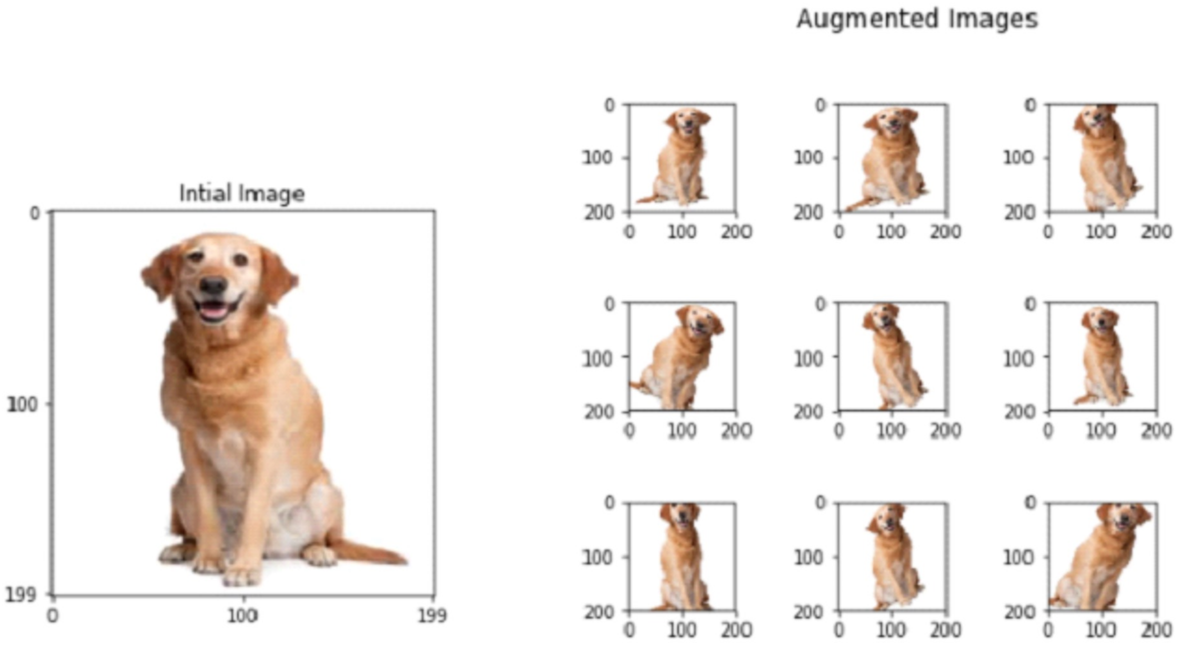


Difficulties in Training Deep Neural Networks

- Vanishing / Exploding Gradients
  - Weight Initialisation Strategies
  - Nonsaturating Activation Functions
  - Batch Normalisation
- Lack of training data for a large network
  - Transfer Learning
- Training may be extremely slow
  - Faster Optimisers
- **Overfitting**
  - Early Stopping
  - Data Augmentation
  - Dropout

Overfitting

- Early Stopping
- Dropout
- Data Augmentation



Notebook: Data Augmentation