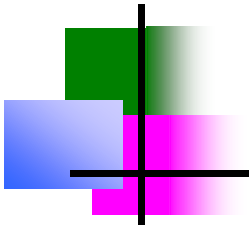
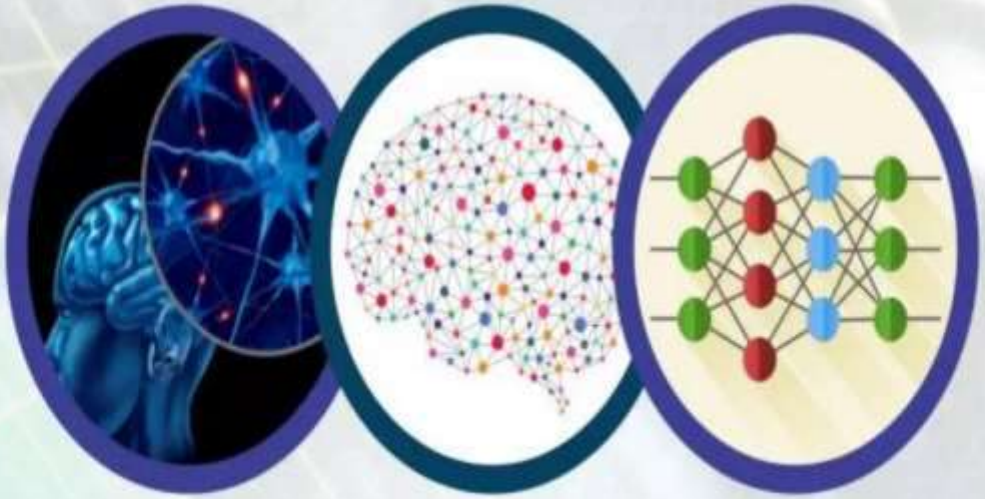


Machine Learning and Computational Intelligence
Lecture 14

Sanjeeb Prasad Panday, PhD
Associate Professor
Dept. of Electronics and Computer Engineering
Director (ICTC)
IOE, TU

DEEP LEARNING: LEARNING BASED ON DEEP NEURAL NETWORK



Seminar Presentation by:

AJOSE - ISMAIL, BABATUNDE M.

(16PCG01368)

DR. (MRS) OLADIPUPO OLUFUNKE

Supervisor

OUTLINE

- Introduction
- Aim and Objectives
- Review of Deep Learning Concept
- Review of Related Works & Specific Application Areas
- Practical Application of Deep Learning (Problem Scenario)
- Trends & Research Issues
- Conclusion
- References
- Acknowledgment

INTRODUCTION

(Deep Learning: A transitive subset of Artificial Intelligence Via Machine Learning

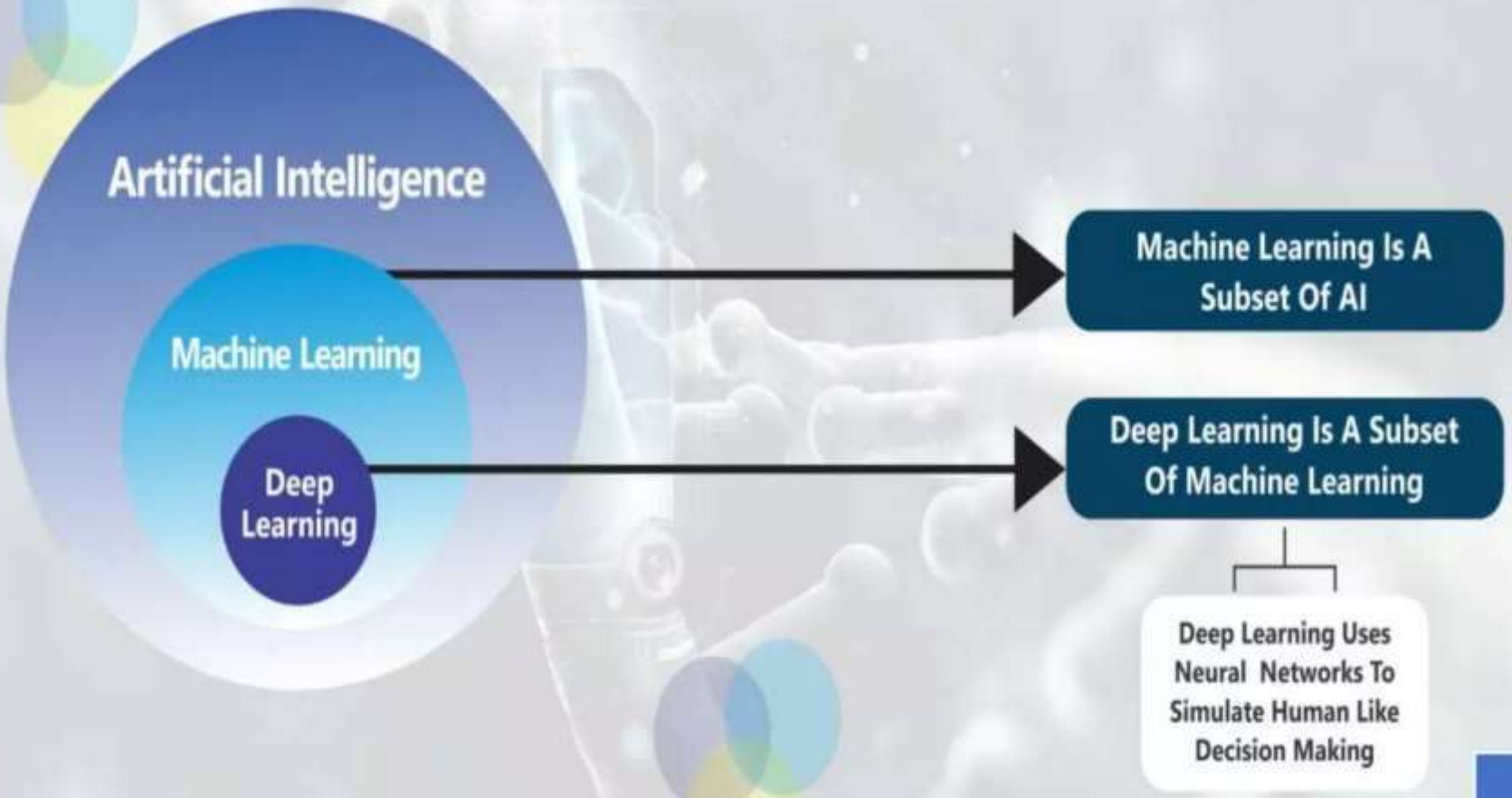


Figure 1 : Subsets of AI(Adapted from: www.edureka.co)

What is Artificial Intelligence (AI)?

- The theory and development of computer systems that are able to perform task normally requiring human intelligence such as visual perception, speech recognition, decision-making and translation between languages

AI INCLUDES FOLLOWING AREAS OF SPECIALIZATION

- Game playing
- Expert System
- Robotics
- Neural networks
- Natural Language Processing

CHESS PLAYING GAME

NUMBER PLATE RECOGNITION

SIRI

Figure 2: Some Applications of Artificial Intelligence

AI Technologies Timeline

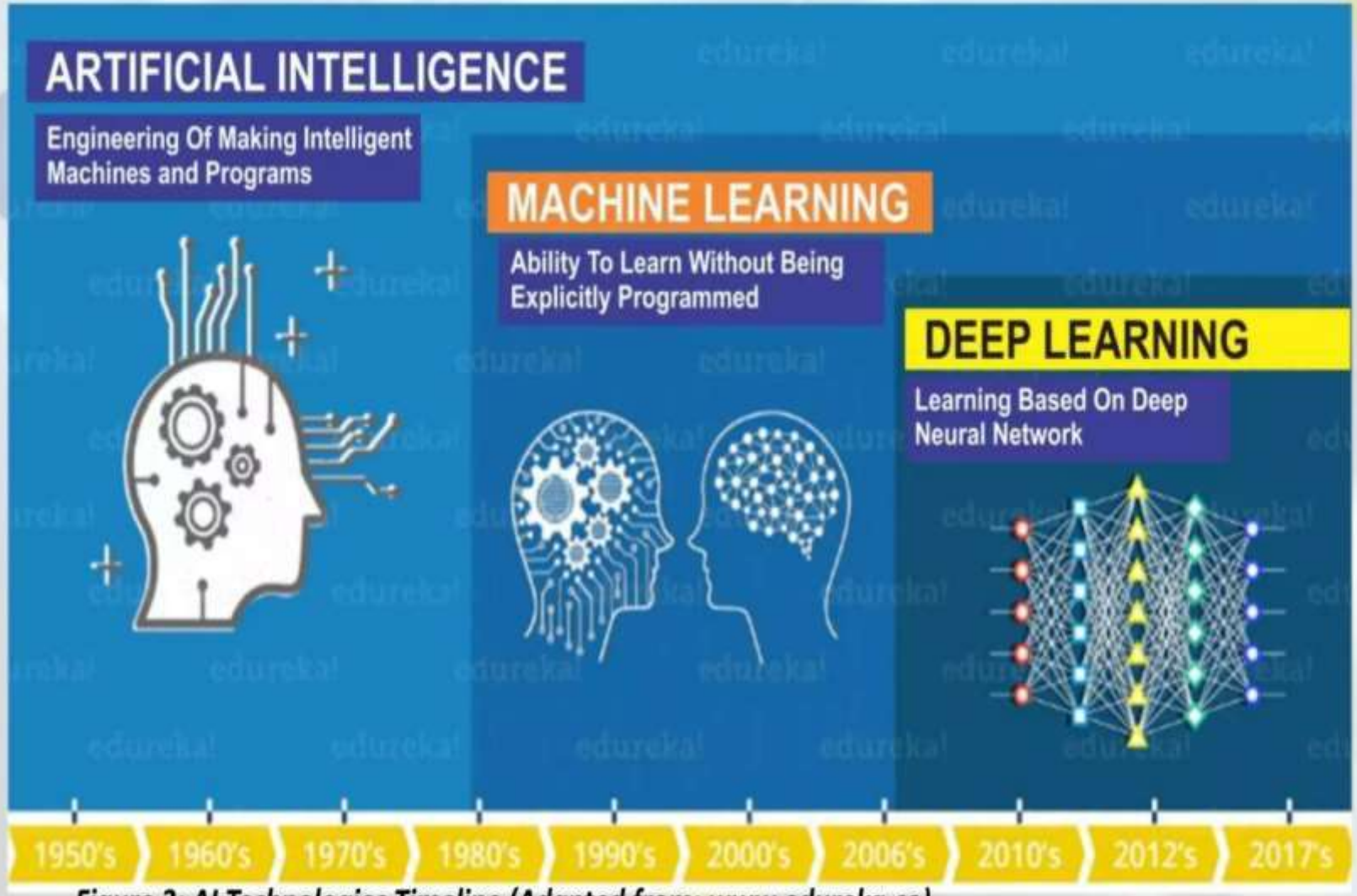


Figure 3: AI Technologies Timeline (Adapted from: www.edureka.co)

What Is Machine Learning?

- Machine learning is a type of artificial intelligence that provide computers with the ability to learn without being explicitly programmed

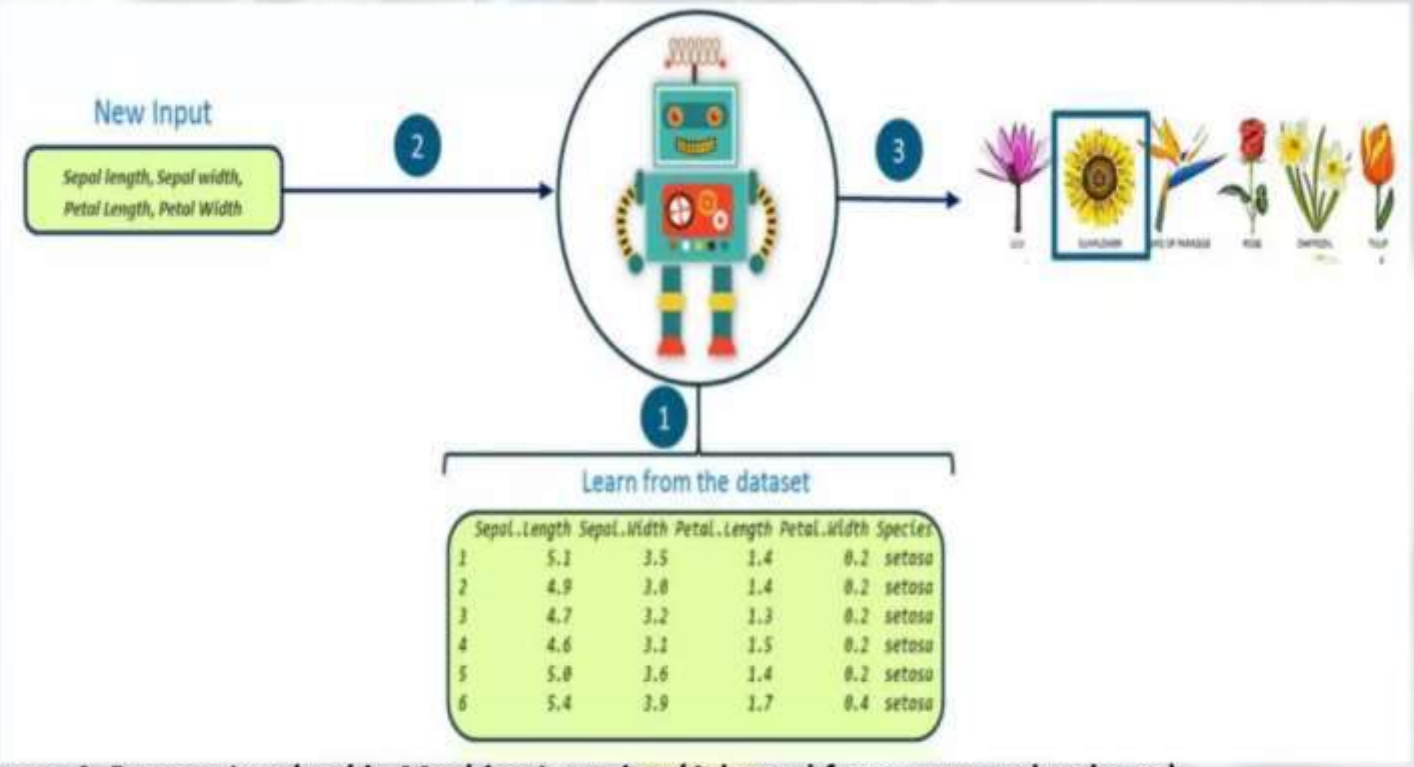


Figure 4: Process Involved in Machine Learning (Adapted from: www.edureka.co)

Limitations of Machine Learning

One Of The Big Challenges With Traditional Machine Learning Models Is A Process Called Feature Extraction. For Complex Problems Such As Object Recognition Of Handwriting Recognition, This Is A Huge Challenge.

Deep Learning To The Rescue

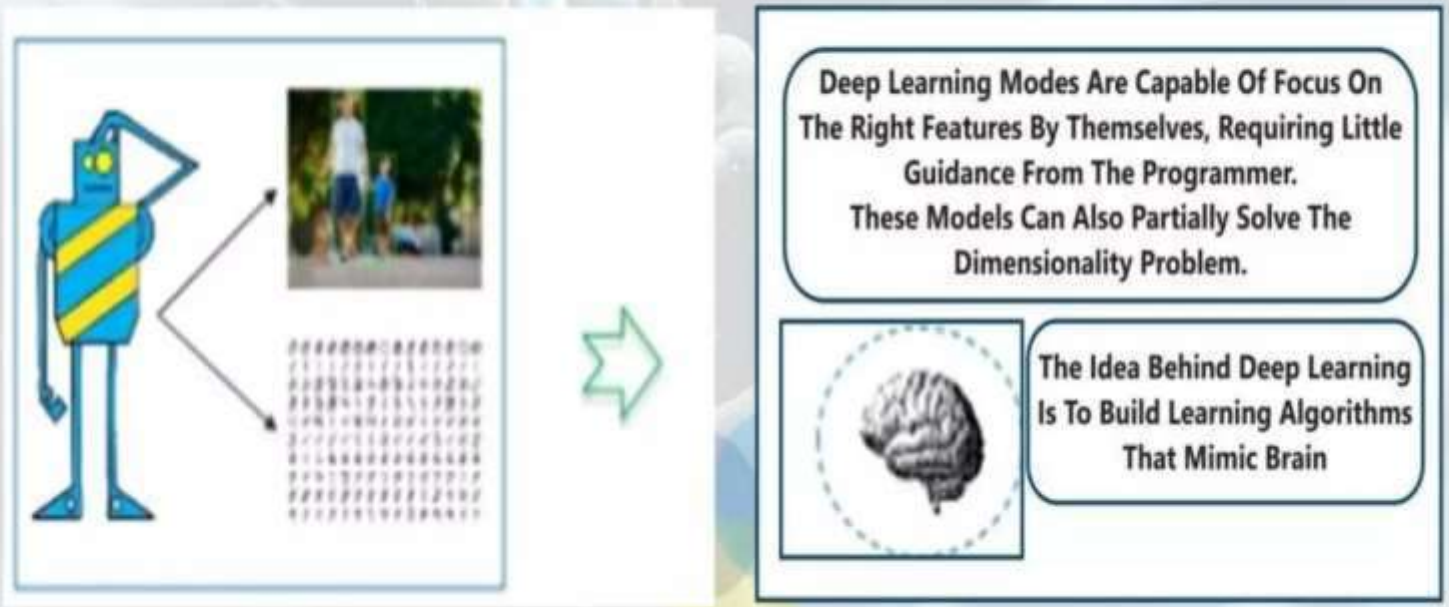
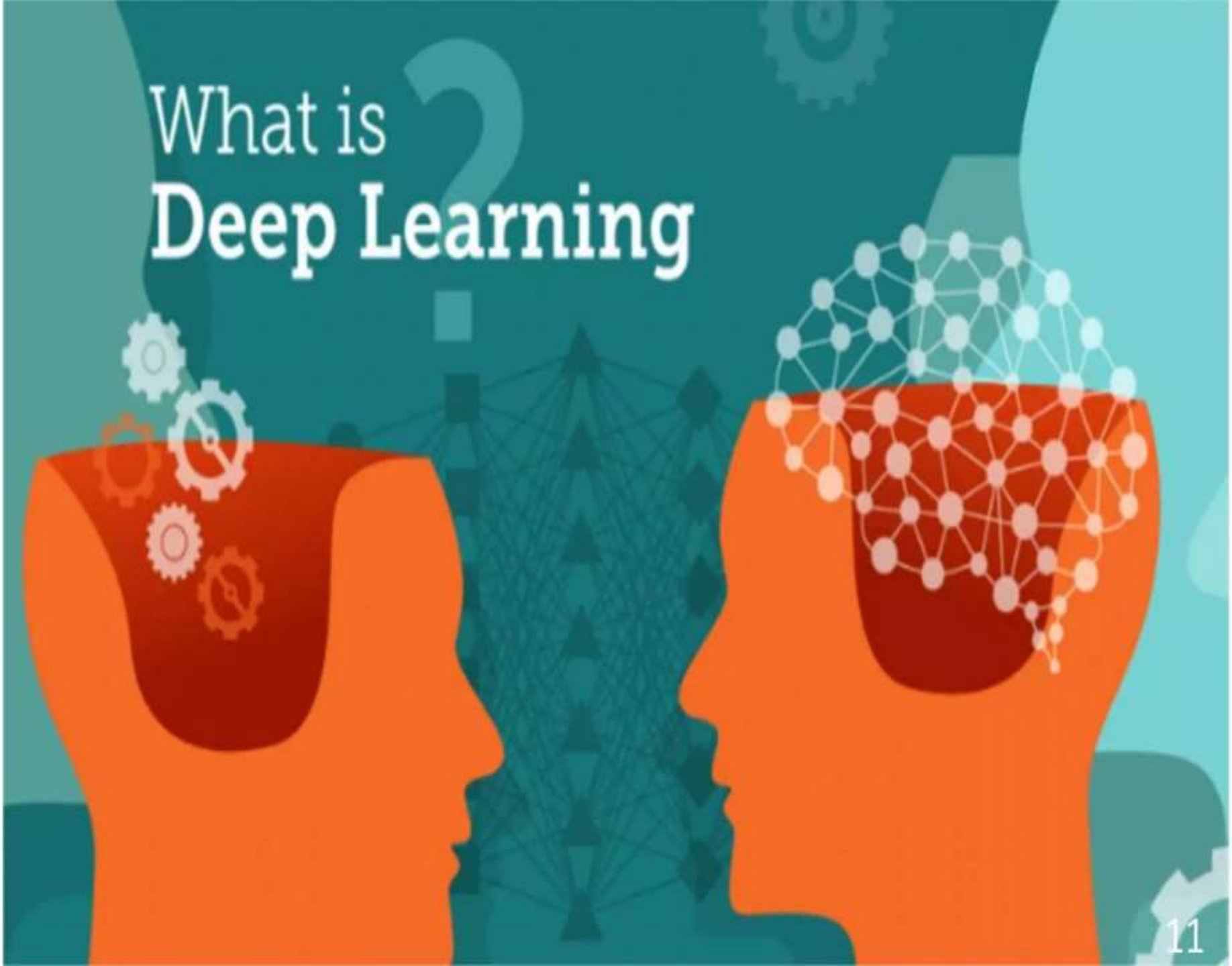


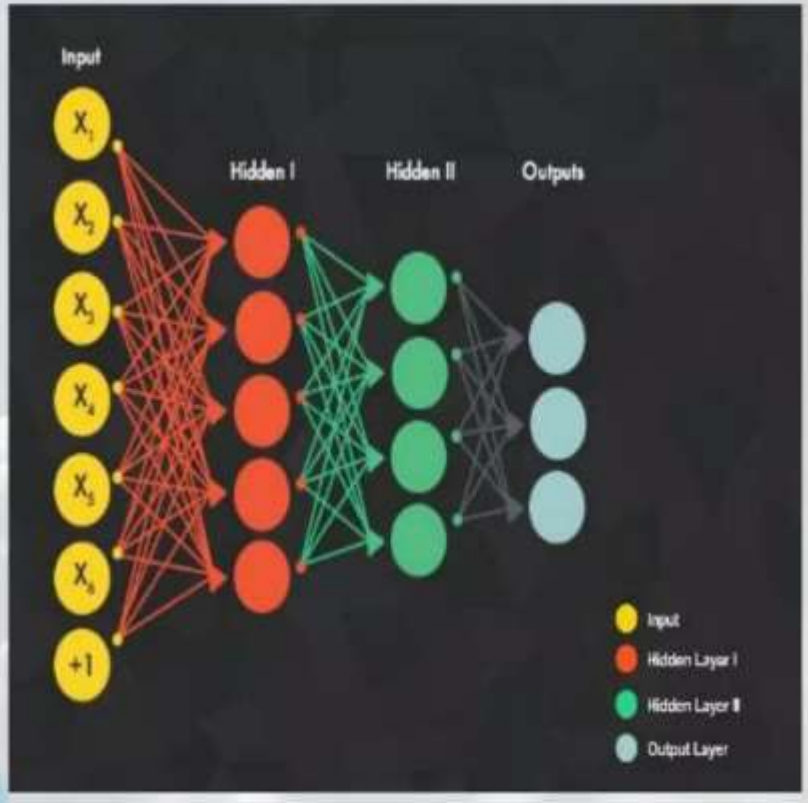
Figure 5: Limitation of ML (Source: www.edureka.co)


What is
Deep Learning

What Is Deep Learning?

- A class of machine learning techniques that exploit many layer of non linear information processing for supervised or unsupervised feature extraction and trasformation and for pattern analysis and classification (Deng & Yu, 2014)

- It typically uses deep artificial neural networks.



What is deep Learning?

- A sub-field of machine learning that is based on learning several levels of representations corresponding to a hierarchy of features or factors or concepts, where higher-level concepts are defined from lower-level ones, and the same lower level concepts can help to define many higher-level concepts.

(LeCun, Bengio, & Hinton, 2015).

Deep Learning = Machine Learning + Representation Learning

- *Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification.*(Deng & Yu, 2014).

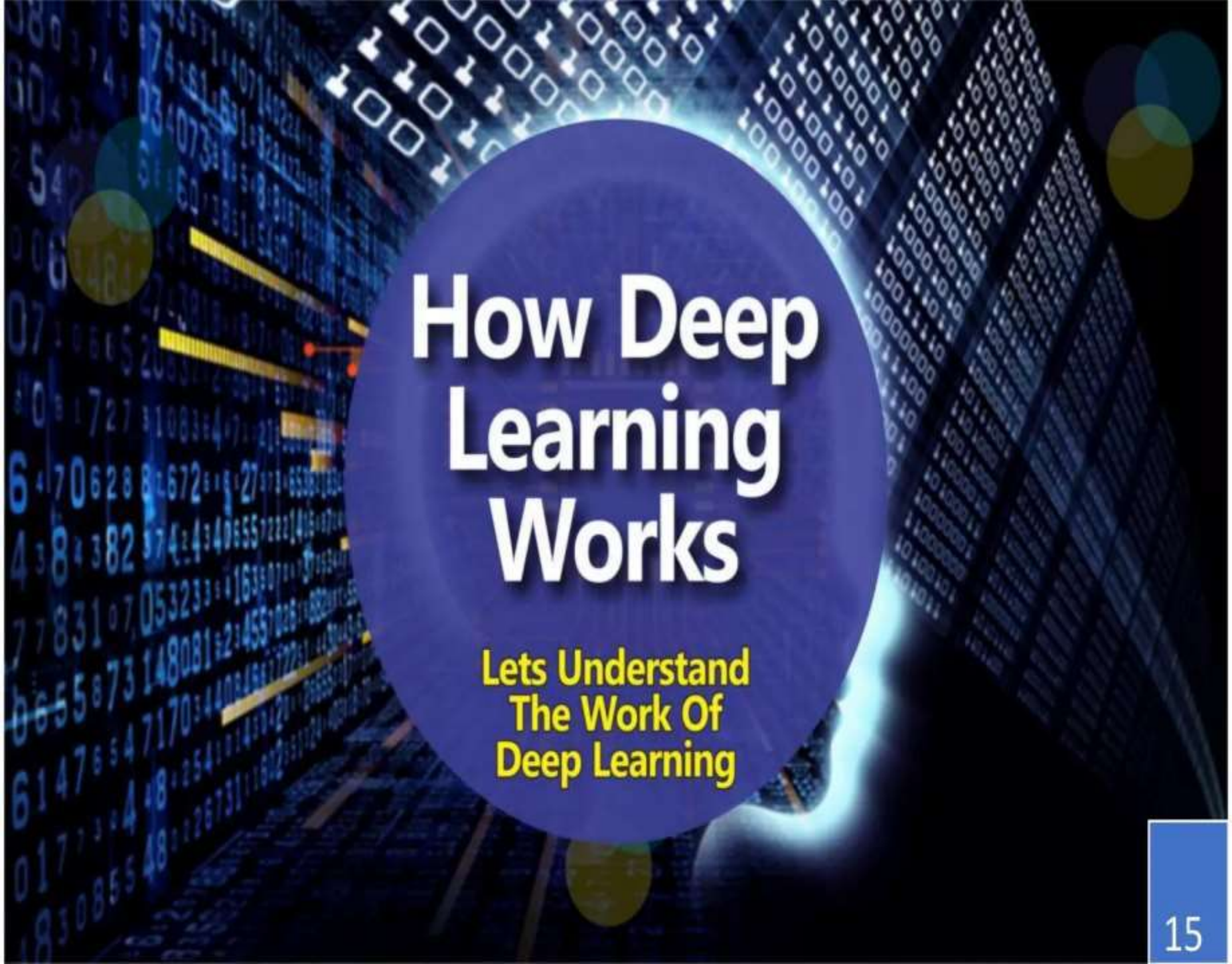
Why Is Deep Learning Happening Now?

- Actually it is not - first papers published in '970s
- Third resurgence of neural networks
  - Research breakthroughs

	Problem	Solution
Vanishing Gradient	Sigmoid type activation functions easily saturate. Gradients are small. in deeper layers updates become almost zero.	Earlier: layer-by-layer pretraining Recently: non-saturating Activation functions
Gradient Descent	First order methods (e.g. SGD) are easily stuck. Second order methods are feasible on larger data.	Adaptive training: adagrad, adam, adadelta, RMSProp Nestrov momentum
Regularisation	Networks easily overfit (even with l2 regularization	Dropout

- Increase in computational power
- GP GPUs

Adapted from: www.edureka.co


How Deep Learning Works
Lets Understand The Work Of Deep Learning

How Deep Learning Works?

- Deep Learning Is Implemented Through Neural Networks
- Motivation Behind Neural Networks Is The Biological Neuron.

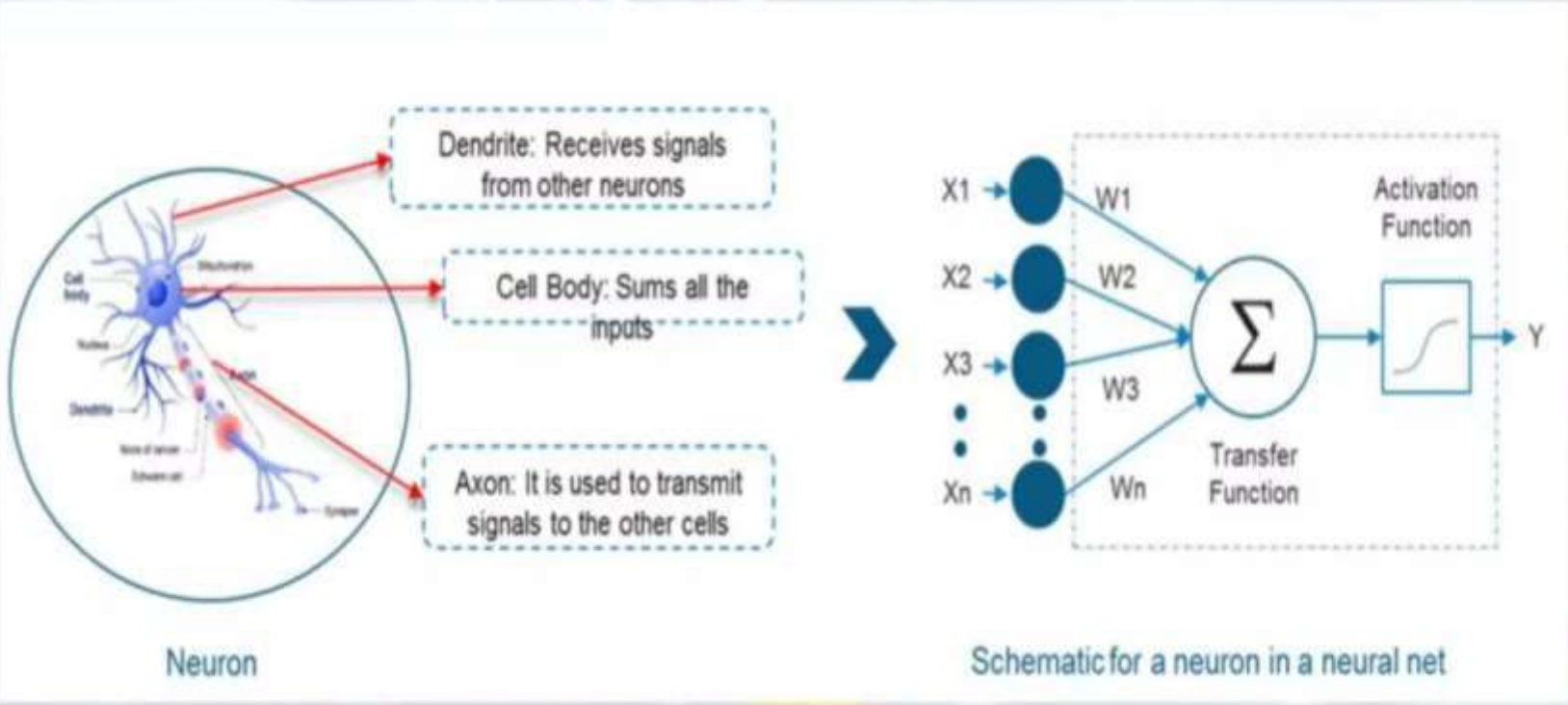


Figure 7: Biological and Artificial Neuron (Adapted from: www.edureka.co)

Unique Characteristics Of Deep Learning

- Automatic feature extraction
- Makes use of High-dimensional data for training
- Defines higher level features from lower level features obtained from the previous layer
- It is a representation learning method
- Can solve real-world or unconstrained problems such as NLP, image recognition etc

Some Deep Learning Software

The Following Are The List Of Some Deep Learning Software's But Not Limited To:

- Tensorflow
- Theano
- Torch
- Keras
- Blocks
- Caffe
- Caffe2
- MxNet

- MatConvNet
- Microsoft Cognitive Toolkit
- Deep learning 4j
- Dlib

...but not limited to these.

Categories of Deep Architecture with Algorithms

1. Supervised or Discriminative Deep Learning
2. Unsupervised or Generative Deep Learning
3. Hybrid Deep Architecture

1 Supervised or Discriminative Deep Learning

- **Convolutional Neural Networks (CNNs)**

A subtype of the discriminative architecture is the CNN.

- The Convolutional Neural Networks (CNN) is one of the most notable deep learning approaches where multiple layers are trained in a robust manner (LeCun, Bottou, Bengio, & Haffner, 1998).

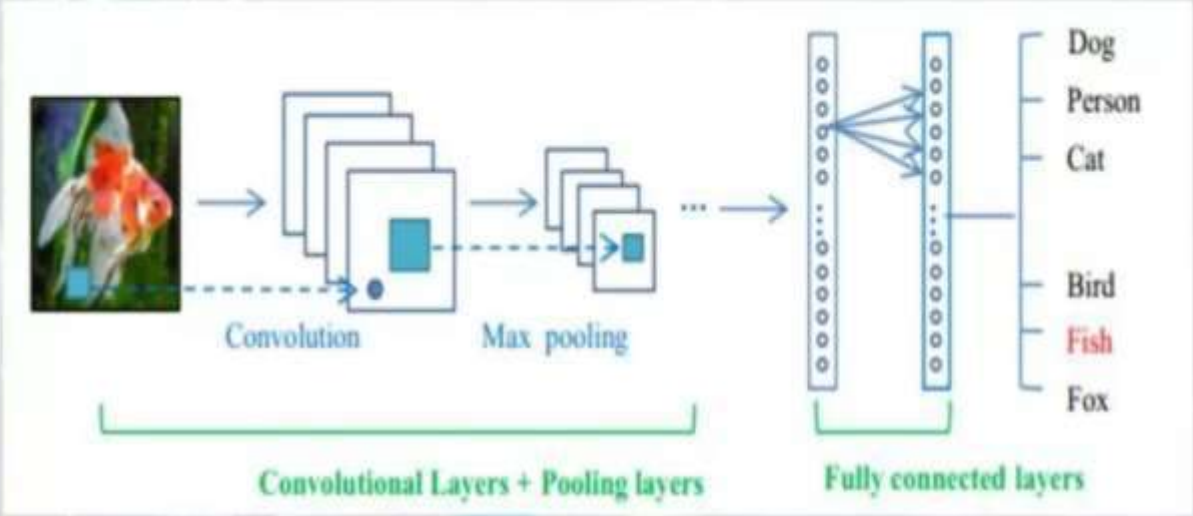


Figure 8: Pipeline of the general CNN Architecture (Source: Guo et al., 2016)

CNN in Action: Facial Recognition

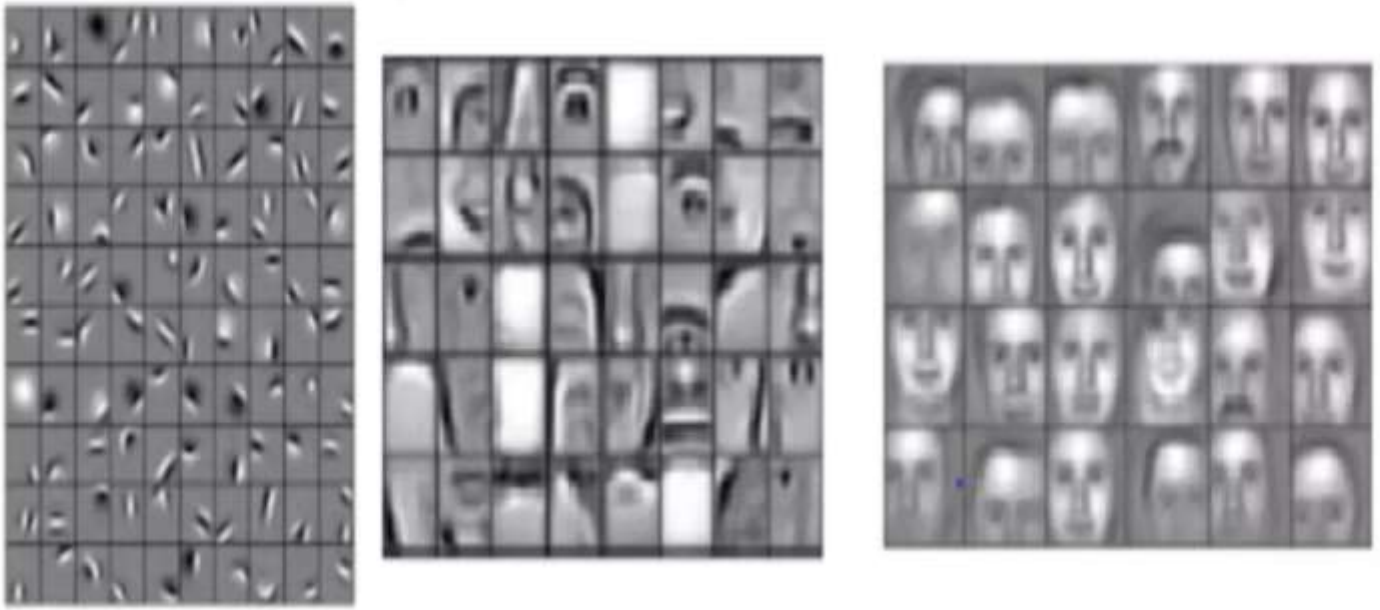


Figure 9: Learned hierarchical features from a deep learning algorithm, CNN
(Source: lee et al., 2011)

2 Unsupervised or Generative Deep Learning

Restricted Boltzmann Machines (RBM)

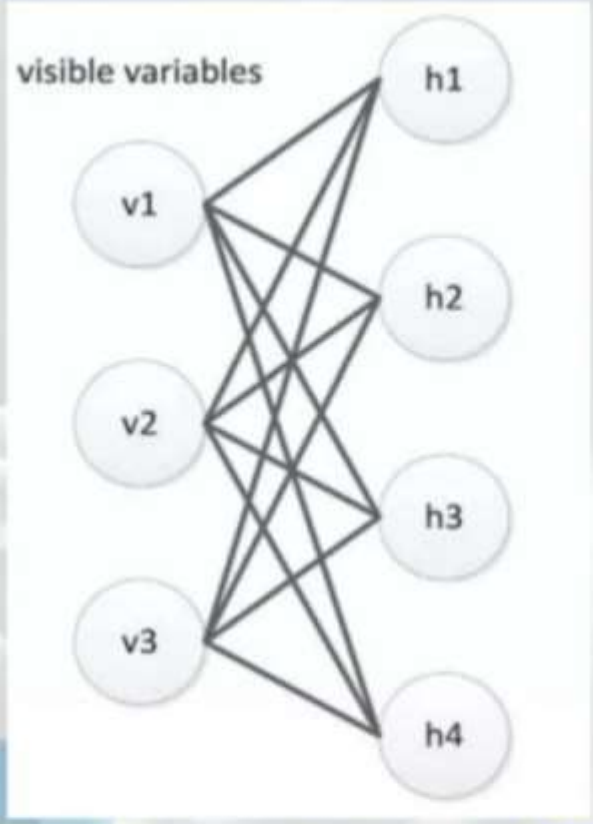


Figure 10 : Schematic structure of RBM
(Source: Guo et al., 2016)

Variants of Restricted Boltzmann Machines (RBM)

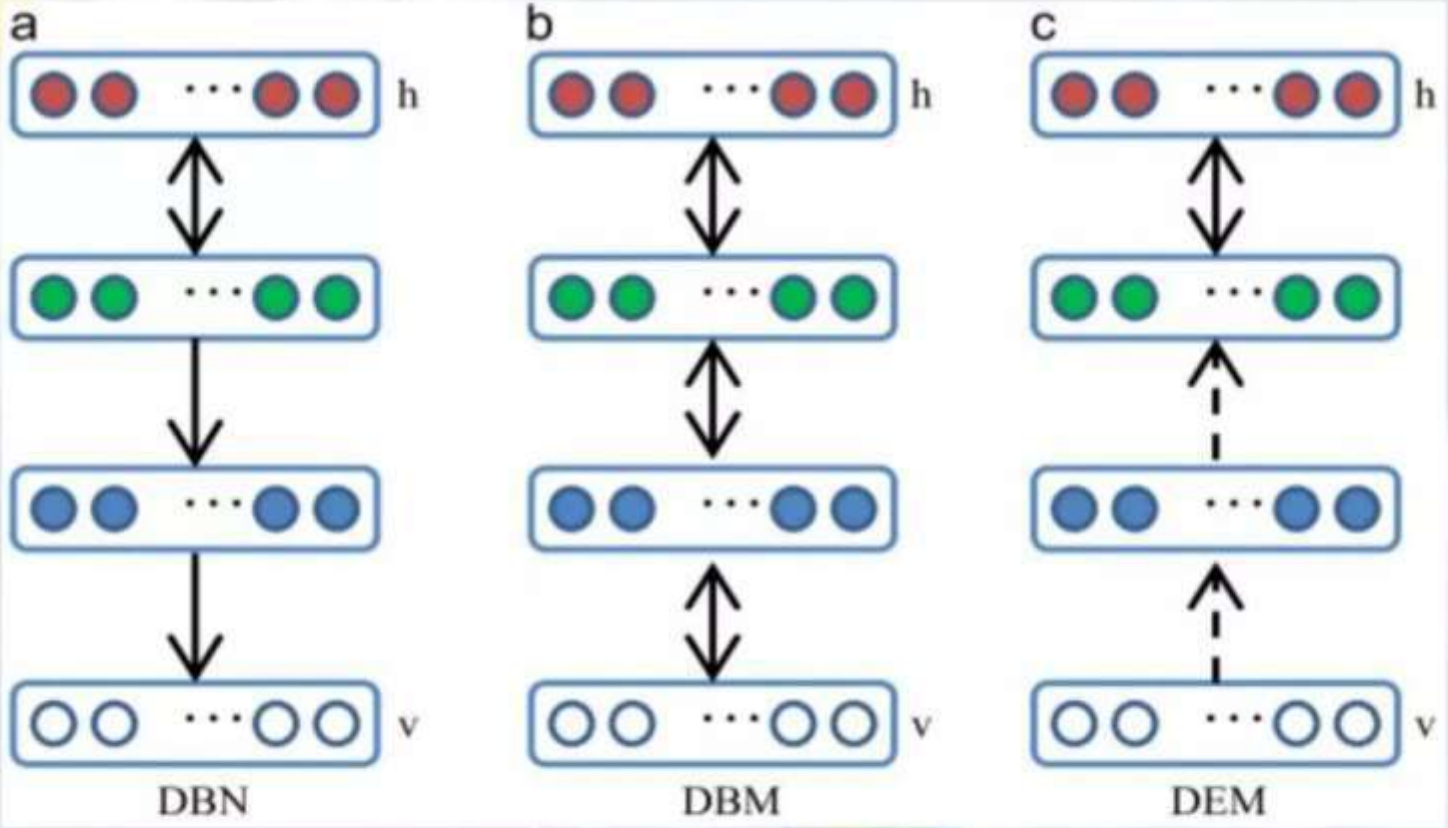


Figure 11: DBN, DBN and DEM (Source: Guo et al., 2016).

Auto-Encoder (AE)

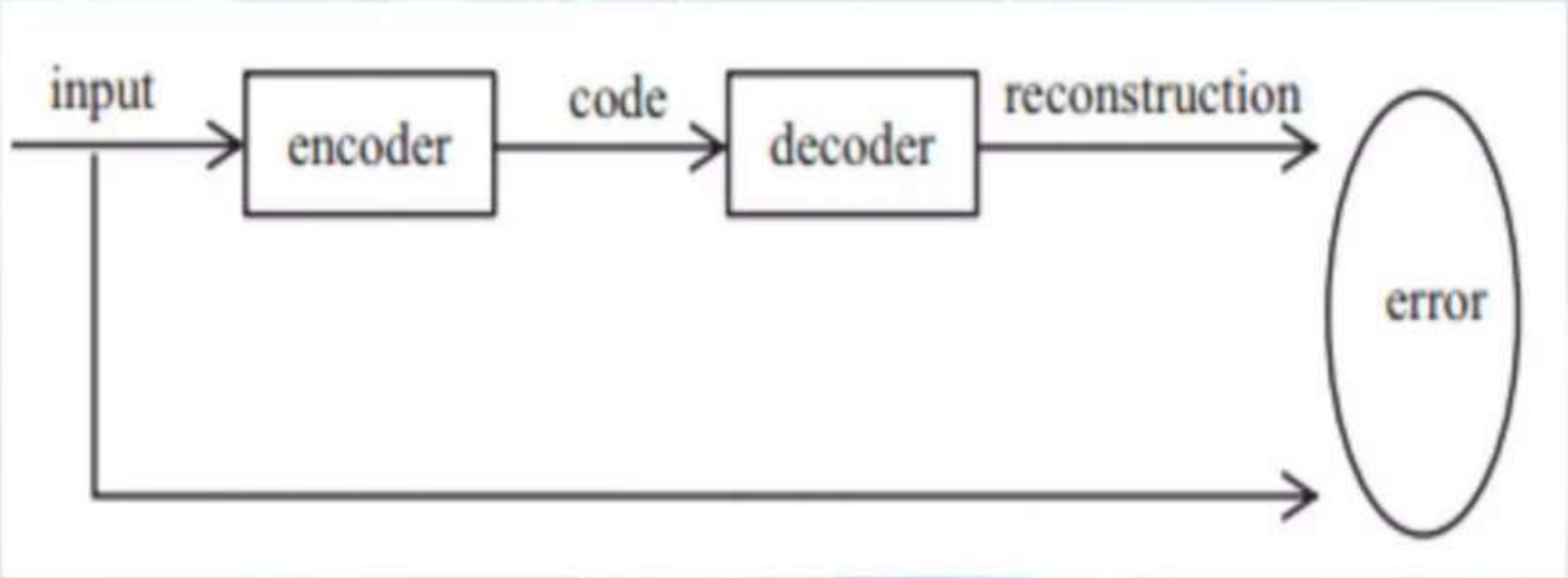
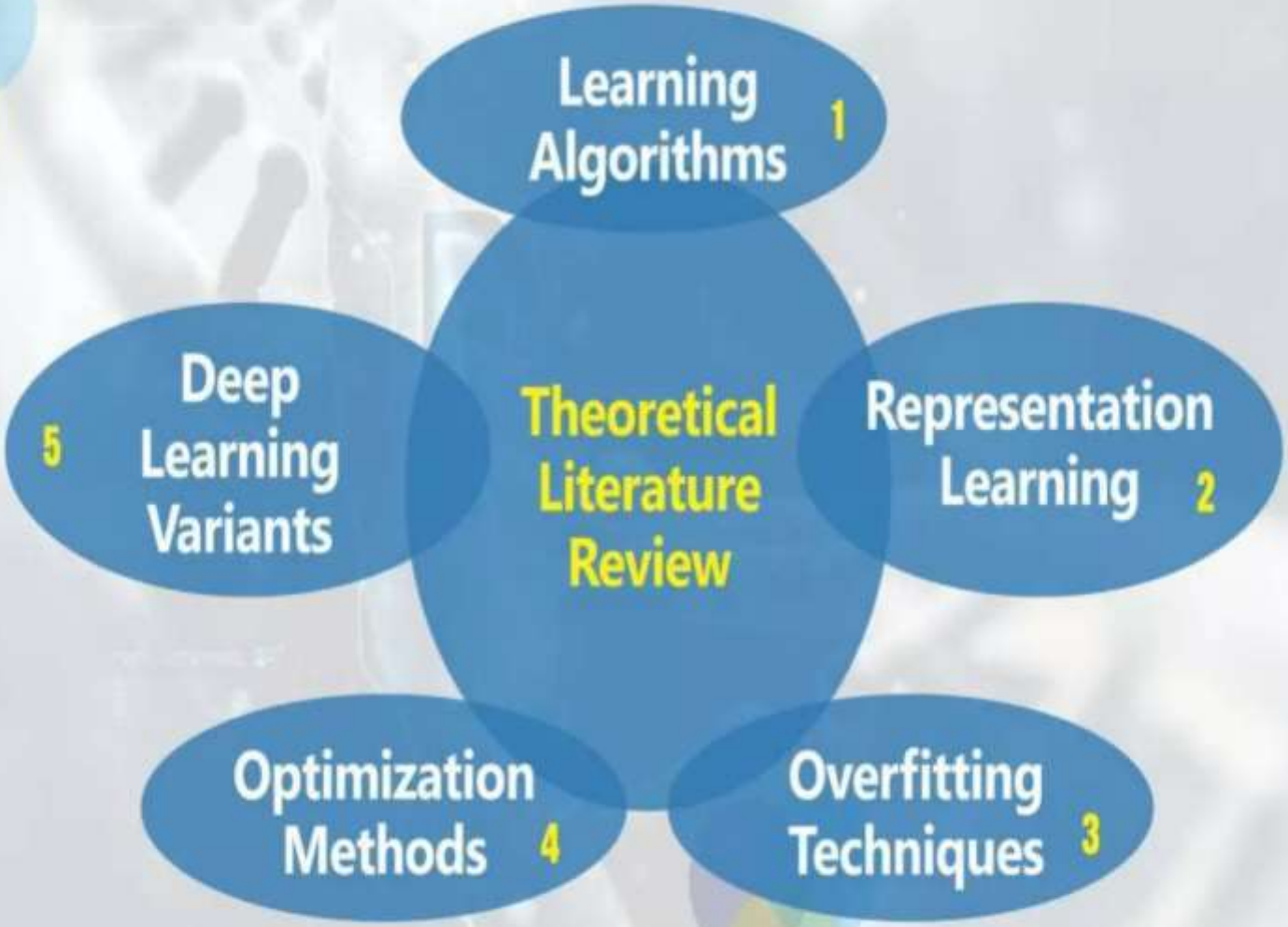


Figure 12: The pipeline of an autoencoder (Source: Guo et al., 2016).


Learning Algorithms 1
Representation Learning 2
Overfitting Techniques 3
Optimization Methods 4
5 Deep Learning Variants
Theoretical Literature Review

(1) Learning Algorithms

Table 1: A categorization of the basic deep NN learning algorithms and related approaches.(Source: Guo et al., 2016).

CNN	RBM	AUTOENCODER	SPARSE CODING
AlexNet (Krizhevsky et al, 2012)	Deep Belief Net (Hinton, et al, 2006)	Sparse Autoencoder (Poultney et al 2006)	Sparse Coding (Yang et al, 2009)
Clarifai (Zeiler, et al 2014)	Deep Boltzmann Machine (Salakhutdinov et al., 2009)	Denoising Autoencoder (Vincent, et al. 2008)	Laplacian Sparse coding (Gao et al, 2010)
SPP (He et al, 2014)	Deep Energy Models (Ngiam et al., 2011)	Contractive Autoencoder (Rifai, et al.2011)	Local Co-ordinate coding (Yu et al, 2009)
VGG (Simonyan et al., 2014)			Super-Vector coding (Zhou et al, 2010)
GoogLeNet (Szegedy et al., 2015)			

2. Representation Learning

S/N	Research Focus	Contribution
1	To discover a fast and more efficient way of initializing weights for effective learning of low-dimensional codes from high-dimensional data in multi-layer neural networks (**Hinton et al., 2006; Salakhutdinov et al., 2009; Vincent et al., 2010; Cho et al., 2011).**	Implementation of a novel learning algorithm for initializing weights that allows deep AE networks and deep boltmzann machines to learn useful higher representations
2	To explore the possibility of allowing hashing function learning(learning of efficient binary codes that preserve neighborhood structure in the original data space) and feature learning occur simultaneously (**Salakhutdinov et al., 2009; Erin et al., 2015; Zhong et al., 2016).**	Introduction of a state-of-the-art deep hashing, supervised deep hashing and semantic hashing methods for large scale visual search, image retrieval and text mining
3	To bridge the gap between the success of CNNSs for supervised learning and unsupervised learning (**Springenberg et al., 2014; Radford et al., 2015**).	Introduction of a class of CNNs called deep convolutional generative adversarial networks (DCGANs) for unsupervised learning.

3. Overfitting Techniques

S/N	Research Focus	Contribution
1	To mitigate the problem of overfitting in large neural networks with sparse datasets (**Zeiler et al., 2013; Srivastava et al., 2014; Pasupa et al., 2016;**).	Implementation of several regularization techniques such as "dropout", stochastic pooling, weight decay, flipped image augmentation amongst others for ensuring stability in DNN
2	To investigate how to automatically rank source CNNs for transfer learning and use transfer learning to improve a Sum-Product Network for probabilistic inference when using sparse datasets (**Afridi et al., 2017; Zhao et al., 2017**).	Design of a reliable theoretical framework that perform zeroshot ranking of CNNs for transfer learning for a given target task in Sum-Product networks

4. Optimization Methods

S/N	Research Focus	Contribution
1	To develop a computationally efficient algorithm for gradient based optimization in deep neural networks **(Hinton et al., 2006; Duchi et al., 2011; Ngiam et al., 2011; Tieleman et al., 2012; Sutskever et al., 2013; Kingma et al., 2014; Patel, 2016).**	Introduction of several first-order and second-order stochastic gradient based optimization methods for minimizing large objective functions in deep networks such as Complementary priors, Adam, Adagrad, RMSprop, Momentum, L-BFGS and Kalman- based SGD
2	To develop an accelerator that can deliver state-of-the-art accuracy with minimum energy consumption when using large CNNs(**Chen et al., 2017).**	Implementation of an Energy-Efficient Reconfigurable Accelerator for Deep CNN using efficient dataflow to minimize energy through zeros skipping/gating.

5. Deep Learning Variants

S/N	Research Focus	Contribution
1	To demonstrate the advantage of combining deep neural networks with support vector machines (**Zhong et al., 2000; Nagi et al., 2012; Tang et al., 2013; Li et al., 2017).**	Introduction of a novel classifier architecture that combines **two heterogeneous supervised classification techniques**, CNN and SVM for feature extraction and for classification
2	To exploit the power of deep neural networks in optimizing the performance of nearest neighbor classifiers in kNN Classification tasks (**Min et al., 2009; Ren et al., 2014).**	They presented a framework for learning convolutional nonlinear features for K nearest neighbor (kNN) classification.

Application Literature Review

1
Methodology
Application of DL

2
Domain
Application of DL

Deep learning has been applied in so many ways to solve real life problems among which are:

(1) Methodology Application of DL

Author and Title	Objective	Methodology	Contribution
Araque et al.(2017). Enhancing deep learning sentiment analysis with ensemble techniques in social applications.	To improve the performance of sentiment analysis in social applications by integrating deep learning techniques with traditional feature based approaches based on hand-crafted or manually extracted features	The utilization of a word embedding's model and a linear machine learning algorithm to develop a deep learning based sentiment classifier(baseline), the use of two ensemble techniques namely ensemble of classifiers (CEM) and ensemble of features (M_{SG} and M_{GA})	Development of ensemble models for sentiment analysis which surpass that of the original baseline classifier

Methodology Application of DL (Contd)

Author and Title	Objective	Methodology	Contribution
Betru et al. (2017). Deep Learning Methods on Recommender System. A Survey of State-of-the-art:	To distinguish between the various traditional recommendation techniques and introducing deep learning collaborative and content based approaches	As pointed out by the authors, the methodology adopted in (Wang, Wang, & Yeung, 2015) integrated a Bayesian Stack De-noising Auto Encoder (SDAE) and Collaborative Topic Regression to perform collaborative deep learning.	The implementation of a novel collaborative deep learning approach, the first of its kind to learn from review texts and ratings.

Methodology Application of DL (Contd)

Author and Title	Objective	Methodology	Contribution
Luo et al.(2016). A deep learning approach for credit scoring using credit default swaps.	To implement a novel method which leverages a DBN model for carrying out credit scoring in credit default swaps (CDS) markets	The methodology adopted by the researchers in their experiments was to compare the results of MLR, MLP, and SVM with the Deep Belief Networks (DBN) with the Restricted Boltzmann Machine by applying 10-fold cross-validation on a dataset	The contribution made by the researchers to this literature is investigating the performance of DBN in corporate credit scoring. The results demonstrate that the deep learning algorithm significantly outperforms the baselines.

Methodology Application of DL (Contd)

Author and Title	Objective	Methodology	Contribution
Grinblat et al.(2016). Deep learning for plant identification using vein morphological patterns.	The authors aimed to eliminate the use of handcrafted features extractors by proposing the use of deep convolutional network for the problem of plant identification from leaf vein patterns.	The methodology adopted to classify three plant species: white bean, red bean and soybean was the use of dataset containing leaf images, a CNN of 6 layers trained with the SGD method, a training set using 20 samples as mini batches with a 50% dropout for regularization.	The relevance of deep learning to agriculture using CNN as a model for plant identification based on vein morphological pattern.

Methodology Application of DL (Contd)

Author and Title	Objective	Methodology	Contribution
Evermann et al.(2017). Predicting process behaviour using deep learning.	To come up with a novel method of carrying out process prediction without the use of explicit models using deep learning.	The approach used to implement this novel idea included the use of a framework called Tensorflow as it provides (RNN) functionality embedded with LSTM cells which can be run on high performance parallel, cluster and GPU platforms.	Improvement in state-of-the-art in process prediction, the needless use of explicit model and the inherent advantages of using an artificial intelligence approach.

Methodology Application of DL (Contd)

Author and Title	Objective	Methodology	Contribution
Kang et al. (2016). A deep-learning-based emergency alert system.	proposed a deep learning emergency alert system to overcome the limitations of the traditional emergency alert systems	A heuristic based machine learning technology was used to generate descriptors starts for labels in the problem domain, an API analyzer that utilized convolutional neural network for object detection and parsing to generate compositional models was also used.	Contribution of this research shows that the EAS can be adapted to other monitoring devices asides from CCTV

(2) Domain Applications Of Deep Learning

Domain	Deep learning is Applied to perform	Topic &Reference
Recommender System	Sentiment Analysis/Opinion mining)	Collaborative Deep Learning for Recommender Systems(Wang, Wang, & Yeung, 2015)
Social Applications	(Sentiment Analysis/Opinion mining/Facial Recognition)	Enhancing deep learning sentiment analysis with ensemble techniques in social applications (Araque et al., 2017).
Medicine	(Medical Diagnosis)	A survey on deep learning in medical image analysis(Litjens et al., 2017)
Finance	(Credit Scoring, stock market prediction)	A deep learning approach for credit scoring using credit default swaps (Luo et al., 2016).

Domain Applications Of Deep Learning (Contd)

Domain	Deep learning is Applied to perform	Topic &Reference
Transportation	Traffic flow prediction	Deep learning for short-term traffic flow prediction(Polson et al.,2017).
Business	Process prediction	Predicting Process Behaviour Using Deep Learning (Evermann et al., 2017).
Emergency	Emergency Alert	A Deep-Learning-Based Emergency Alert System (Kang et al., 2016)
Agriculture	(Plant Identification)	Deep Learning for Plant Identification Using Vein Morphological Patterns (Grinblat et al.,2016).

Applications of Deep Learning

Some Amazing & Recent Application Of Deep Learning Are:

- Automatic Machine Translation
- Object Classification In Photographs
- Character Text Generation
- Colorization Of Black And White Images
- Automatic Game Playing

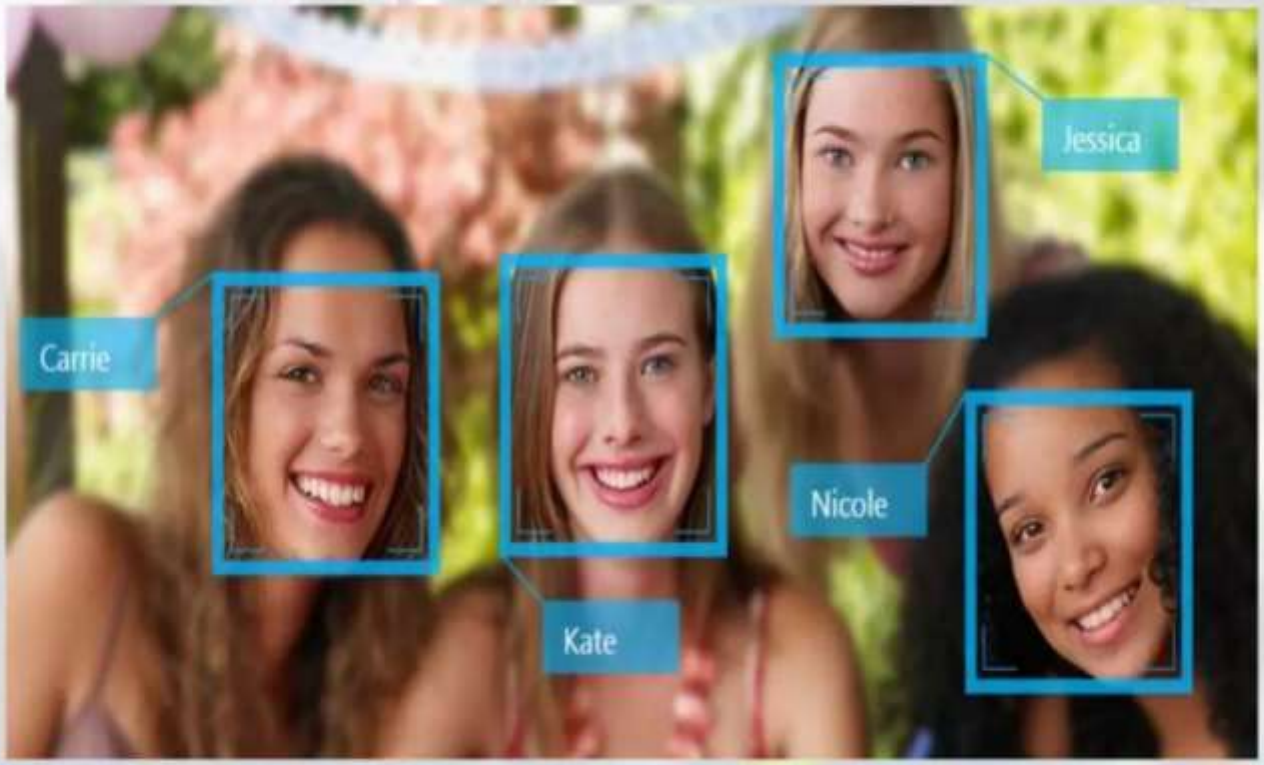
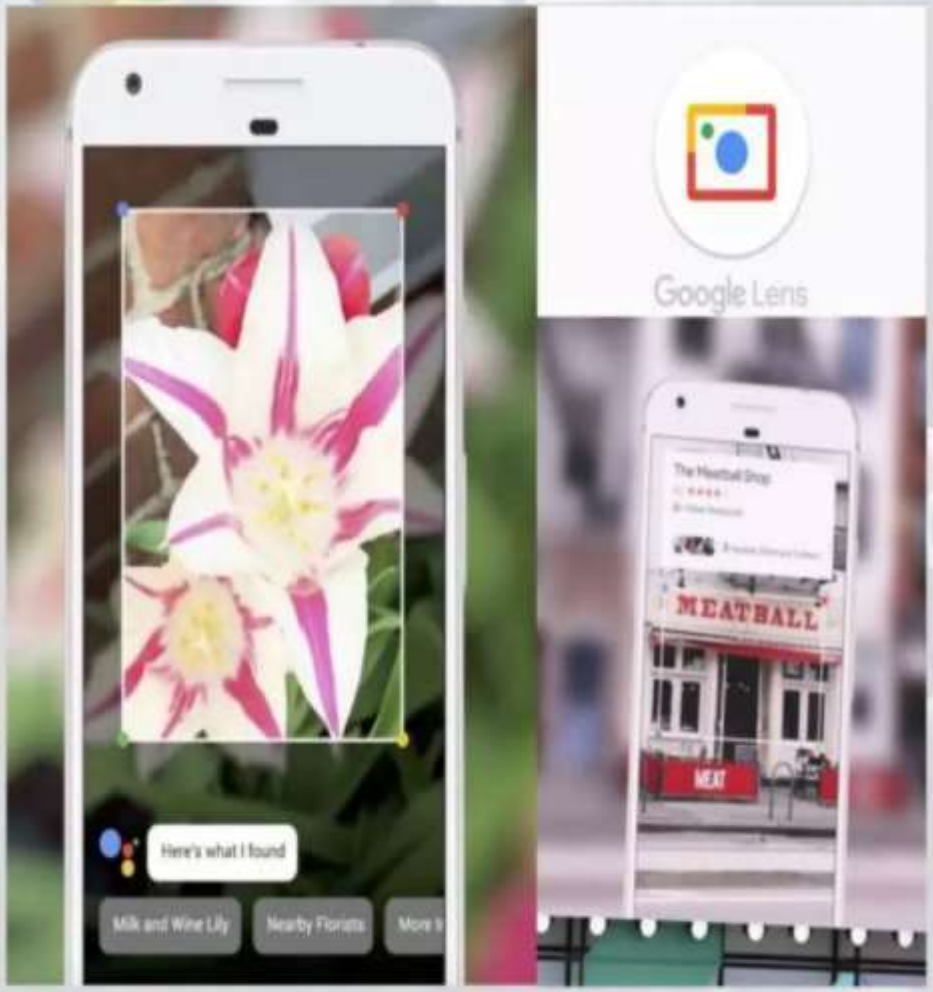


Figure 13: Face Recognition (Adapted from www.edureka.co)

Applications of Deep Learning (Contd)



- Google lens is a set of vision based computing capabilities that allows your smartphone to understand what's going on in a photo, video or a live feed.
- For instance, point your phone at a flower and Google Lens will tell you on the screen which type of flower it is.
- You can aim the camera at a restaurant sign to see reviews and other information.

Figure 14: Google Lens (Adapted from www.edureka.co)

Applications of Deep Learning (Contd)

Translation

- This is a situation where you are given words in some language and you have to translate the words to desired language say english
- This kind of translation is a classical example of image recognition

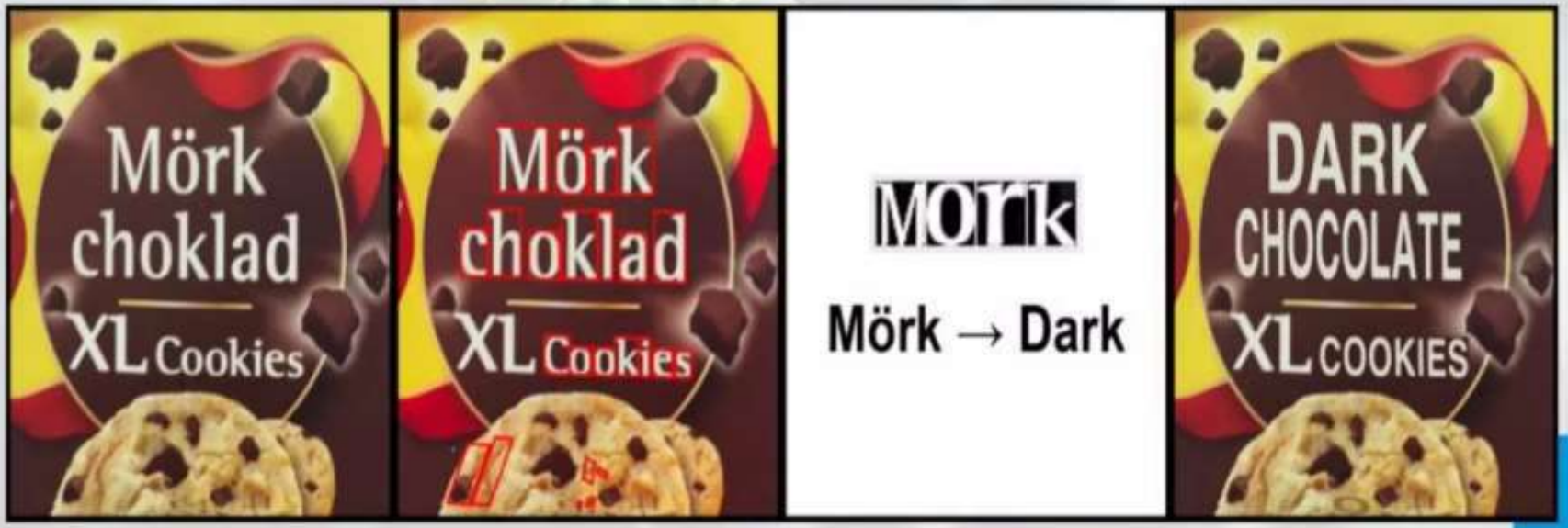


Figure 15: Machine Translation (Adapted from www.edureka.co)

Instant Visual Translation

Figure 16: Instant Visual Translation (Adapted from www.edureka.co)

Self Driving Cars

Figure 17: Self Driving Cars (Adapted from www.edureka.co)

Automatic Machine Translation

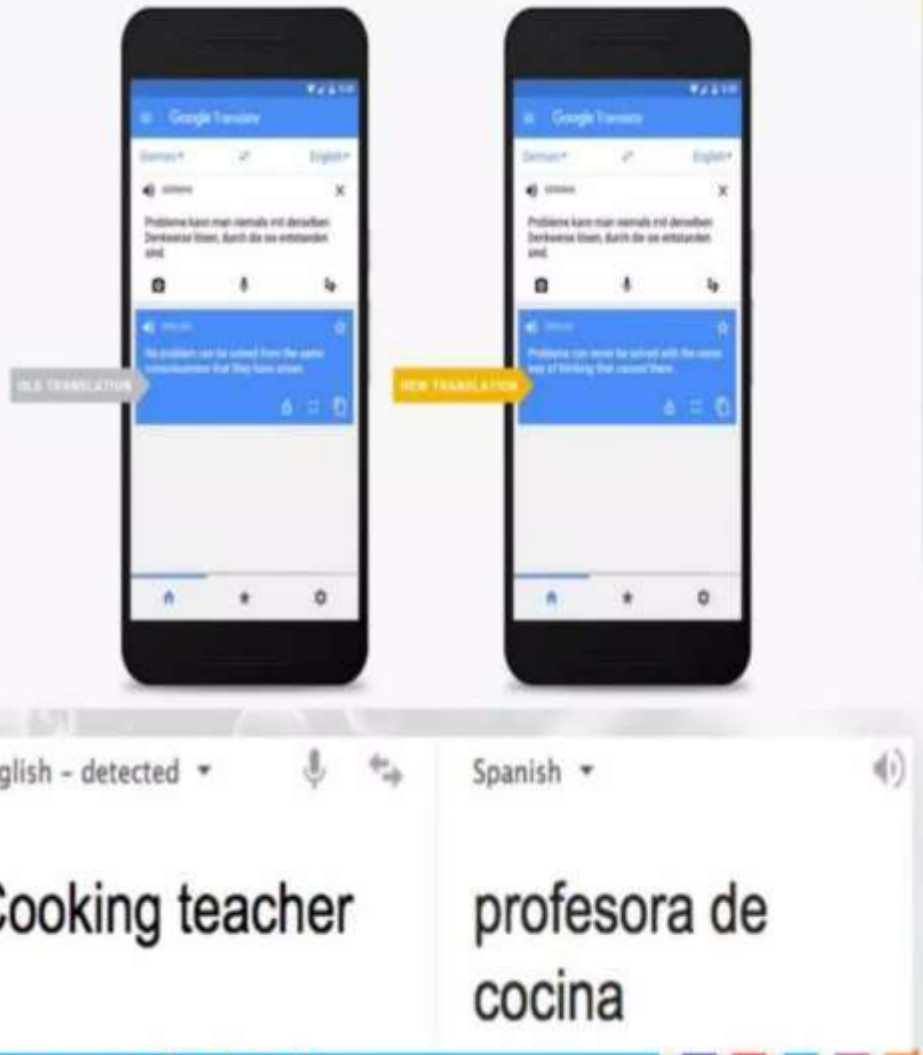


Figure 18: Machine Translation (Adapted from www.edureka.co)

Trends in Deep Learning Research

1. Design of more powerful deep models to learn from fewer training data. (Guo et al, 2016; pasupa et al., 2016 ;Li, et al 2017)
2. Use of better optimization algorithms to adjust network parameters i.e. regularization techniques (zeng et al, 2016; Li, et al 2017)
3. Implementation of deep learning algorithms on mobile devices (Li, et al 2017)
4. Stability analysis of deep neural network (Li, et al 2017)

Trends in Deep Learning Research (Contd)

5. Combining probabilistic , auto-encoder and manifold learning models.(bengio et al., 2013)
6. Applications of deep neural networks in nonlinear networked control systems (NCSs) (Li, et al 2017)
7. Applications of unsupervised, semi-supervised and reinforcement-learning approaches to DNNs for complex systems (Li, et al 2017)
8. Learning deep networks for other machine learning techniques e.g. deep kNN (Zoran et al, 2017), deep SVM (Li, et al 2017).

Research Issues/challenges in Deep Learning

1. High Computational cost/burden in training phase (pasupa et al., 2016)
2. Over-fitting problem when the data-set is small. (pasupa et al., 2016, Guo et al, 2016)
3. Optimization issues due to local minima or use of first order methods. (pasupa et al., 2016)
4. Little or no clear understanding of the underlying theoretical foundation of which deep learning architecture should perform well or outperform other approaches. (Guo et al, 2016)
5. Time complexity (Guo et al, 2016)

Deep Learning – Use Case

Let's look at a use case where we can use DL for image recognition

Practical Application of deep learning in Facial Recognition

Problem Scenario

Suppose we want to create a system that can recognize faces of different people in an image. How do we solve this as a typical machine learning problem and/or using a deep learning approach?

Classical Machine Learning Approach

We will define facial features such as eyes, nose, ears etc. and then, the system will identify which features are more important for which person on its own or by itself.

Deep Learning Approach

Now, deep learning takes this one step ahead.

Deep learning automatically finds out the features which are important for classification because of deep neural networks, whereas in case of Machine Learning we had to manually define these features.

Practical Application of deep learning - Facial Recognition (Contd)

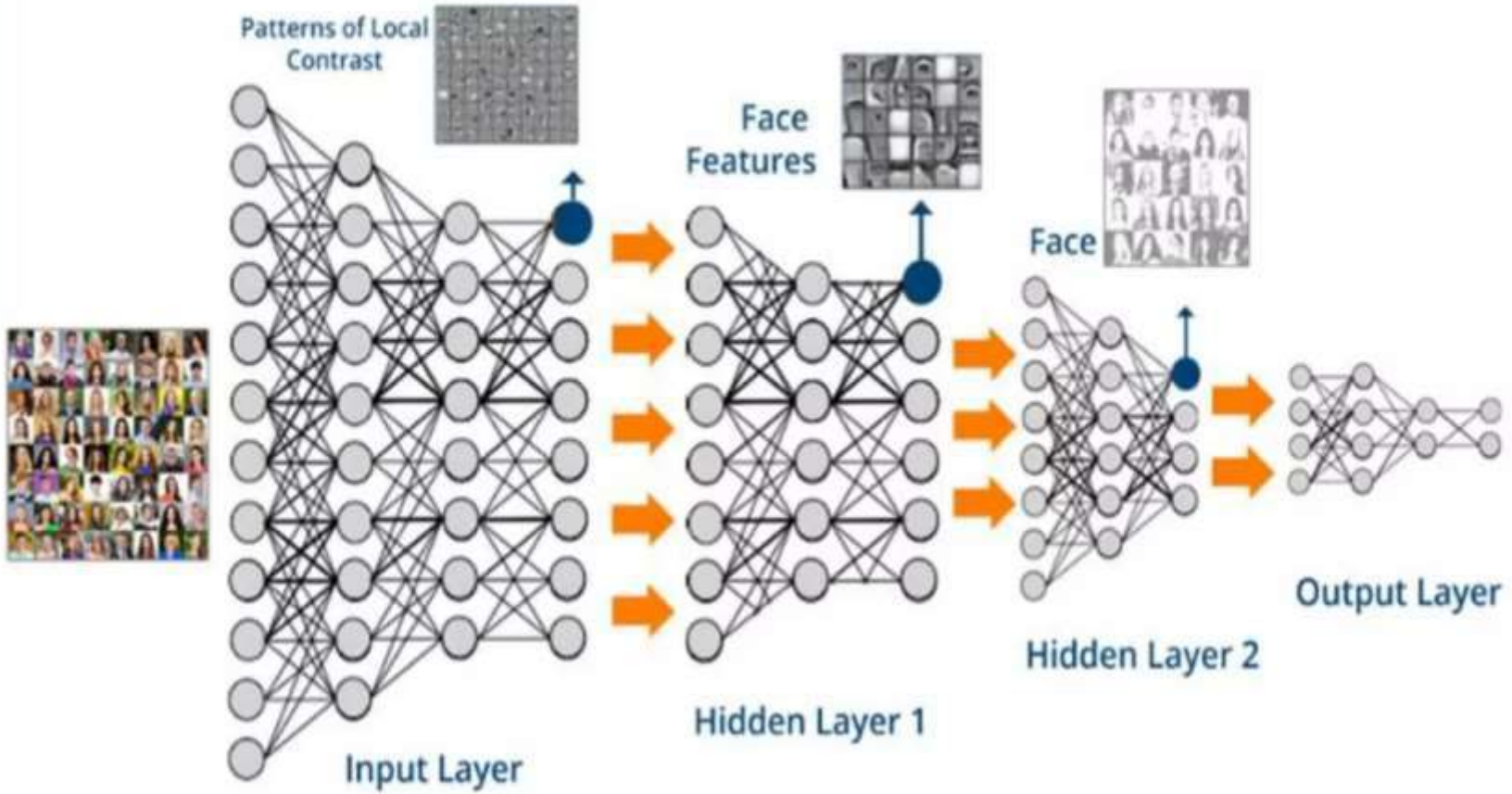


Figure 19: Face Recognition Using deep networks (Source: www.edureka.co)

Deep Face Recognition

Face recognition applications have two parts or phases viz:

(1) Phase-I: Enrollment phase – Model / system is trained using millions of prototype face images and a trained model is generated. Generated face features are stored in database and

(2) Phase-II: Recognition phase – Query face image is given as input to the model generated in phase-I to recognise it correctly.

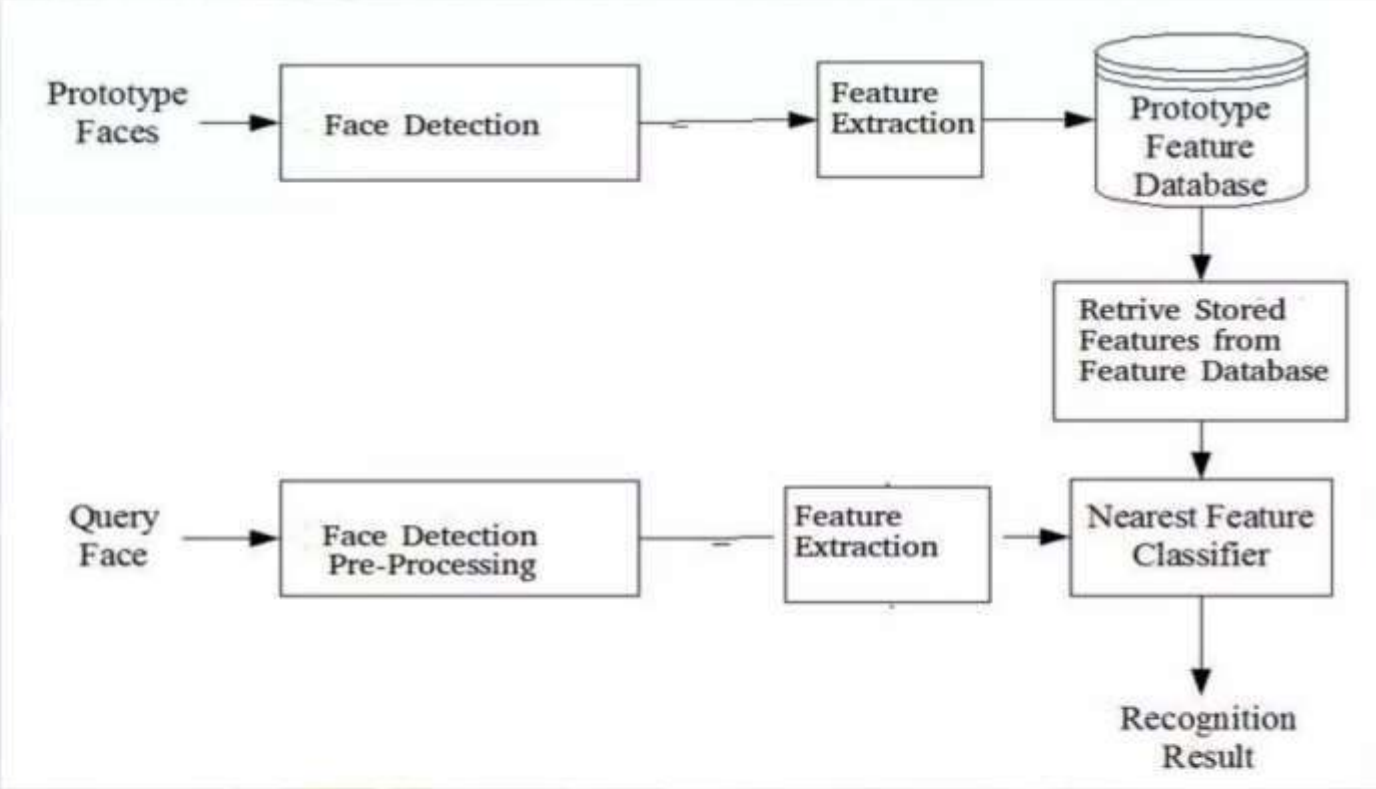


Figure 20: Face Recognition Architecture (Source: aiehive.com)

Deep Face Recognition (Contd)

Steps within Enrollment Phase Includes

1. Face Detection
2. Feature extraction
3. Store Model and extracted feature in Database

Steps within Recognition Phase / Query Phase Includes

1. Face Detection
2. Preprocessing
3. Feature Extraction
4. Recognition

Step 1: Face Detection - Enrollment Phase

Face Detection: Face needs to be located and region of interest is computed.

- Histogram of Oriented Gradients (HOG) is a faster and easier algorithm for face detection.
- Detected faces are given to next step of feature extraction.

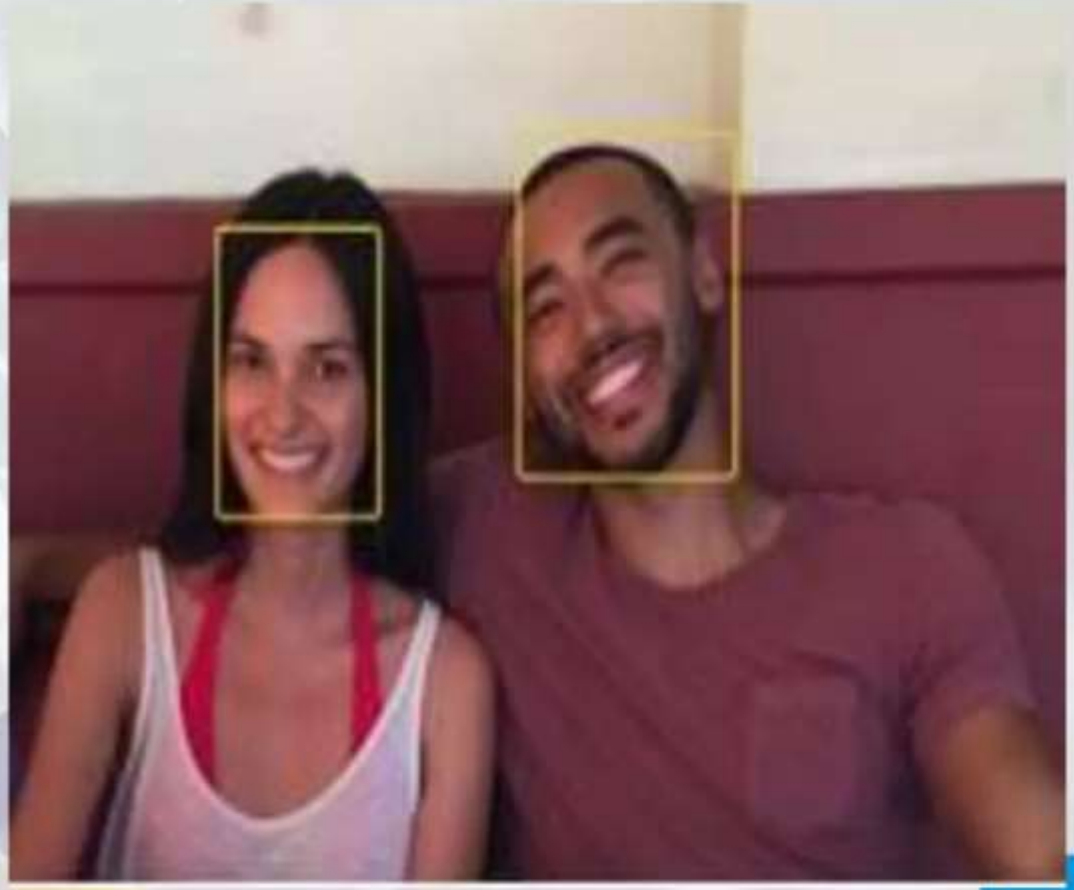
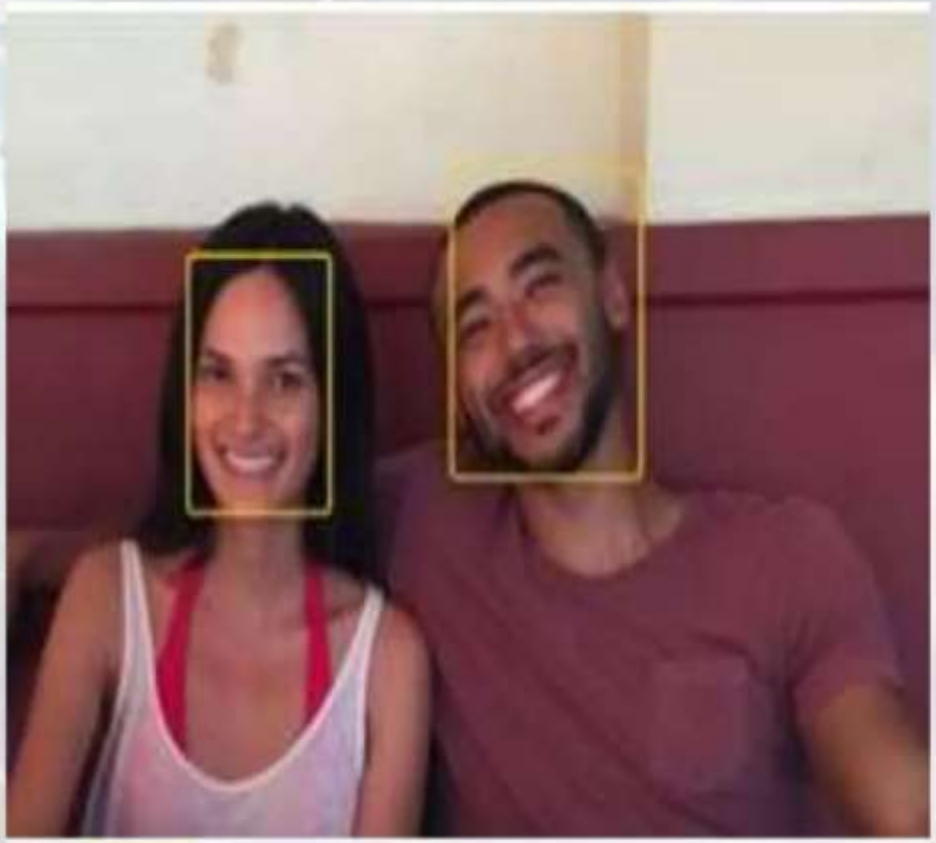


Figure 21: Multiple Face Detection (Source: aiehive.com)

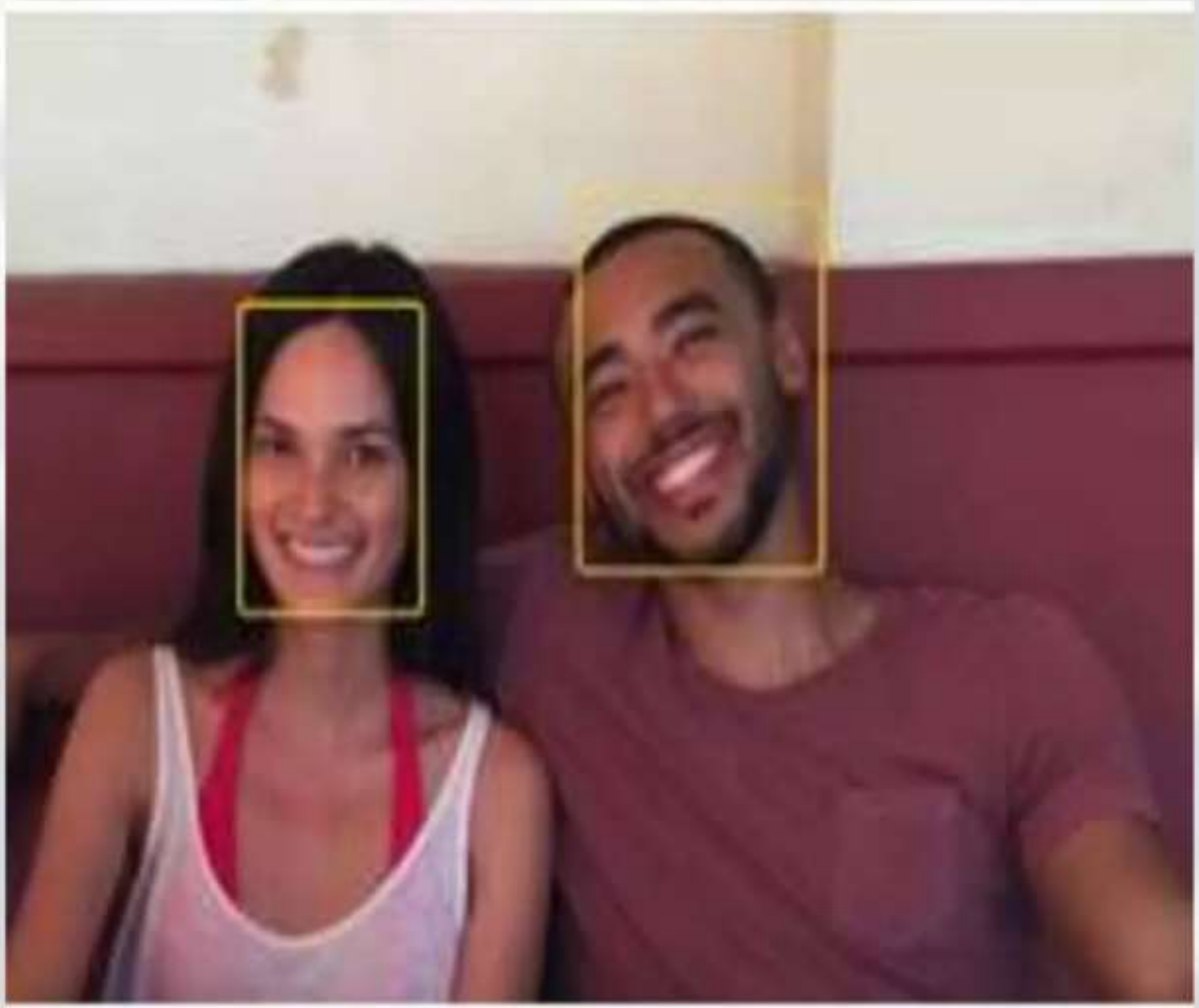
Step 2: Feature Extraction- Enrollment Phase

What is the best feature measure that represents human face in a best way?

Deep learning can determine which parts of a face are important to measure. Deep Convolution Neural Network (DCNN) can be trained to learn important features. (Simonyan et al., 2014)



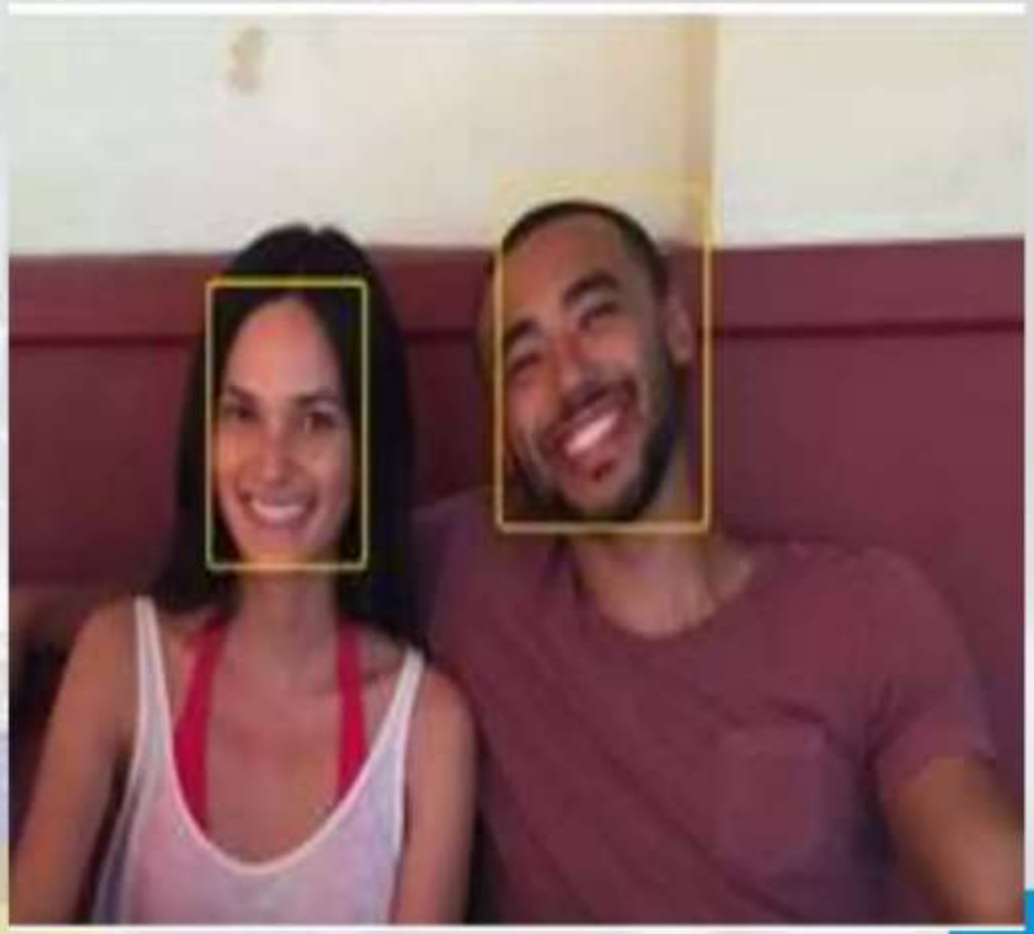
Step-3. Store DCNN model and Feature in Database- Enrollment Phase



Step 1: Face Detection- Recognition Phase

Face Detection: Face needs to be located and region of interest is computed.

- Histogram of Oriented Gradients (HOG) is faster and easier algorithm for face detection.
- Detected faces are given to next step of preprocessing.



Step-2. Pre-processing- Recognition Phase

- Pre-process to overcome issues like noise, illumination using any suitable filters [Kalman Filter, Adaptive Retinex (AR), Multi-Scale Self Quotient (SQI), Gabor Filter, etc.]
- Pose/rotation can be accounted by using 3D transformation or affine transformation or face landmark estimation
- Determine 68 landmark points on every face – the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc.

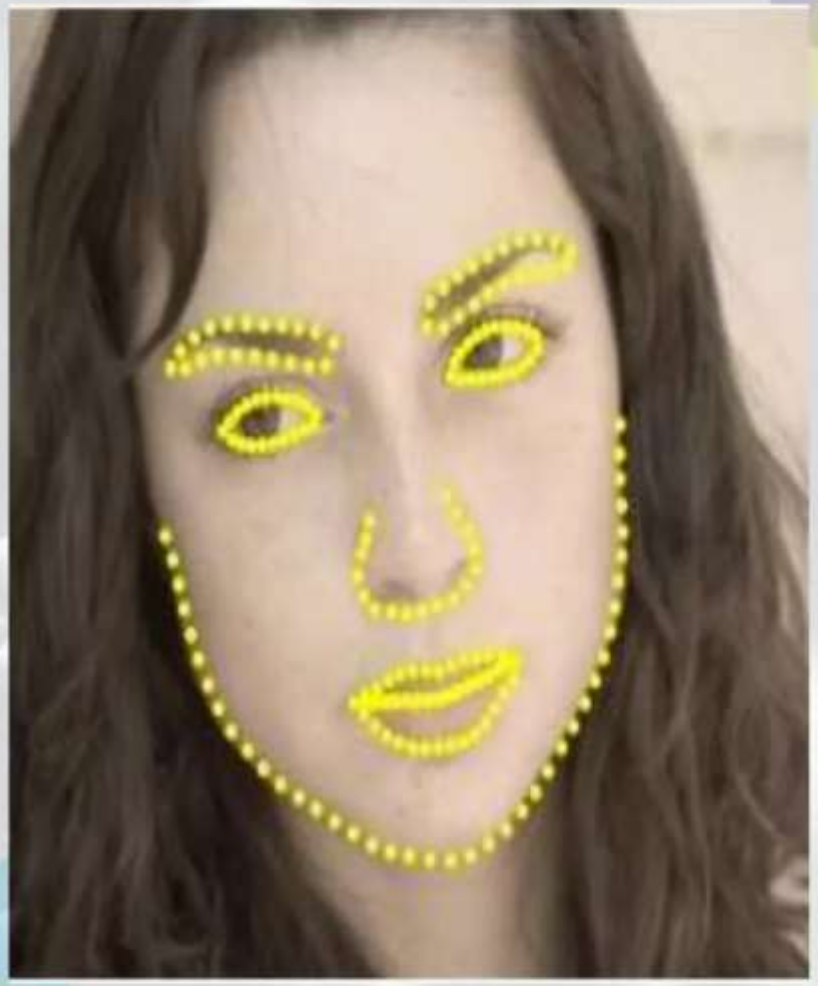


Figure 22: Landmark point estimation *(Source: aiehive.com)*

Step 3: Feature Extraction-Recognition Phase

- In this third step of Deep Face Recognition, we have to use trained DCNN model, which was generated during feature extraction step of enrollment phase
- A query image is given as input.
- The DCNN generates 128 feature values.
- This feature vector is then compared with feature vector stored in database

Step 4: Recognition-Recognition Phase

- This can be done by using any basic machine learning classification algorithm SVM classifier, Bayesian classifier, Euclidean Distance classifier, for matching database feature vector with query feature vector.

- Gives ID of best matching face image from database as a recognition output.

Conclusion

- Deep learning is a representation learning method and the new state-of-the-art technique for performing automatic feature extraction in large unlabeled data
- Various categories of deep learning architectures and basic algorithms together with their related approaches have been discussed
- Several theoretical concepts and practical application areas have been presented
- It is a promising research area for tackling feature extraction for complex real-world problems without having to undergo the process of manual feature engineering.
- With the rapid development of hardware resources and computation technologies, it is certain that deep neural networks will receive wider attention and find broader applications in the future.

References

- Afridi, M. J., Ross, A., & Shapiro, E. M. (2017). On automated source selection for transfer learning in convolutional neural networks. Pattern Recognition.
- Araque, O., Corcuera-Platas, I., Sánchez-Rada, J. F., & Iglesias, C. A. (2017). Enhancing deep learning sentiment analysis with ensemble techniques in social applications. Expert Systems with Applications, 77, 236-246.
- Betru, B. T., Onana, C. A., & Batchakui, B. (2017). A Survey of State-of-the-art: Deep Learning Methods on Recommender System. International Journal of Computer Applications, 162(10).
- Chen, Y. H., Krishna, T., Emer, J. S., & Sze, V. (2017). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 52(1), 127-138.
- Cho, K., Raiko, T., & Ihler, A. T. (2011). Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines. In Proceedings of the 28th International Conference on Machine Learning (ICML-11) (pp. 105-112).

References (Contd).

- Deng, L., & Yu, D. (2014). Deep learning: methods and applications. Foundations and Trends® in Signal Processing, 7(3–4), 197-387.
- Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.
- Erin Liong, V., Lu, J., Wang, G., Moulin, P., & Zhou, J. (2015). Deep hashing for compact binary codes learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2475-2483).
- Evermann, J., Rehse, J. R., & Fettke, P. (2017). Predicting process behaviour using deep learning. Decision Support Systems.
- Gao, S., Tsang, I. W. H., Chia, L. T., & Zhao, P. (2010, June). Local features are not lonely–Laplacian sparse coding for image classification. In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on (pp. 3555-3561). IEEE.

References (Contd).

- Grinblat, G. L., Uzal, L. C., Larese, M. G., & Granitto, P. M. (2016). Deep learning for plant identification using vein morphological patterns. Computers and Electronics in Agriculture, 127, 418-424.
- Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. Neurocomputing, 187, 27-48.
- He, K., Zhang, X., Ren, S., & Sun, J. (2014, September). Spatial pyramid pooling in deep convolutional networks for visual recognition. In European Conference on Computer Vision (pp. 346-361). Springer, Cham.
- Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.
- Kang, B., & Choo, H. (2016). A deep-learning-based emergency alert system. ICT Express, 2(2), 67-70.

References (Contd).

- Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
- Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2011). Unsupervised learning of hierarchical representations with convolutional deep belief networks. Communications of the ACM, 54(10), 95-103.

References (Contd).

- Li, Y., & Zhang, T. (2017). Deep neural mapping support vector machines. Neural Networks, 93, 185-194.
- Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. arXiv preprint arXiv:1702.05747.
- Luo, C., Wu, D., & Wu, D. (2016). A deep learning approach for credit scoring using credit default swaps. Engineering Applications of Artificial Intelligence.
- Makwana, M. A.(2016, Dec)Deep Face Recognition Using Deep Convolutional Neural Network.Retrieved from http://aiehive.com
- Min, R., Stanley, D. A., Yuan, Z., Bonner, A., & Zhang, Z. (2009, December). A deep non-linear feature mapping for large-margin knn classification. In Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on (pp. 357-366). IEEE.
- Nagi, J., Di Caro, G. A., Giusti, A., Nagi, F., & Gambardella, L. M. (2012, December). Convolutional neural support vector machines: hybrid visual pattern classifiers for multi-robot systems. In Machine Learning and Applications (ICMLA), 2012 11th International Conference on (Vol. 1, pp.

References (Contd).

- Ngiam, J., Chen, Z., Koh, P. W., & Ng, A. Y. (2011). Learning deep energy models. In Proceedings of the 28th International Conference on Machine Learning (ICML-11) (pp. 1105-1112).
- Pasupa, K., & Sunhem, W. (2016, October). A comparison between shallow and deep architecture classifiers on small dataset. In Information Technology and Electrical Engineering (ICITEE), 2016 8th International Conference on (pp. 1-6). IEEE.
- Patel, V. (2016). Kalman-based stochastic gradient method with stop condition and insensitivity to conditioning. SIAM Journal on Optimization, 26(4), 2620-2648.
- Polson, N. G., & Sokolov, V. O. (2017). Deep learning for short-term traffic flow prediction. Transportation Research Part C: Emerging Technologies, 79, 1-17.
- Poultney, C., Chopra, S., & Cun, Y. L. (2007). Efficient learning of sparse representations with an energy-based model. In Advances in neural information processing systems (pp. 1137-1144).

References (Contd).

- Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.
- Ren, W., Yu, Y., Zhang, J., & Huang, K. (2014, August). Learning convolutional nonlinear features for k nearest neighbor image classification. In Pattern Recognition (ICPR), 2014 22nd International Conference on (pp. 4358-4363). IEEE.
- Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In Proceedings of the 28th international conference on machine learning (ICML-11) (pp. 833-840).
- Salakhutdinov, R., & Hinton, G. (2009). Semantic hashing. International Journal of Approximate Reasoning, 50(7), 969-978.
- Salakhutdinov, R., & Hinton, G. (2009, April). Deep boltzmann machines. In Artificial Intelligence and Statistics (pp. 448-455).

References (Contd).

- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
- Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of machine learning research, 15(1), 1929-1958.
- Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, February). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147).
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

References (Contd).

- Tang, Y. (2013). Deep learning using support vector machines. CoRR, abs/1306.0239, 2.
- Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.
- Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008, July). Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning (pp. 1096-1103). ACM.
- Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec), 3371-3408.
- Wang, H., Wang, N., & Yeung, D. Y. (2015, August). Collaborative deep learning for recommender systems. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1235-1244). ACM.
- what is deep learning.(Web log post).Retrieved September 6, 2017 from https://www.edureka.co/blog/

References (Contd).

- Yang, J., Yu, K., Gong, Y., & Huang, T. (2009, June). Linear spatial pyramid matching using sparse coding for image classification. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on (pp. 1794-1801). IEEE.
- Yu, K., Zhang, T., & Gong, Y. (2009). Nonlinear learning using local coordinate coding. In Advances in neural information processing systems (pp. 2223-2231).
- Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.
- Zhao, J., & Ho, S. S. (2017). Structural knowledge transfer for learning Sum-Product Networks. Knowledge-Based Systems, 122, 159-166.
- Zhong, G., Xu, H., Yang, P., Wang, S., & Dong, J. (2016, July). Deep hashing learning networks. In Neural Networks (IJCNN), 2016 International Joint Conference on (pp. 2236-2243). IEEE.
- Zhong, S., & Ghosh, J. (2000). Decision boundary focused neural network classifier.

References (Contd).

- Zhou, X., Yu, K., Zhang, T., & Huang, T. S. (2010, September). Image classification using super-vector coding of local image descriptors. In European conference on computer vision (pp. 141-154). Springer, Berlin, Heidelberg.
- Zoran, D., Lakshminarayanan, B., & Blundell, C. (2017). Learning Deep Nearest Neighbor Representations Using Differentiable Boundary Trees. arXiv preprint arXiv:1702.08833.

Acknowledgement

- Almighty God for His sufficient grace.
- I would like to appreciate the HOD, Dr Osamor V.C. and the PG Coordinator, Dr. Azeta for their contribution toward the reality of the presentation today.
- Special recognition to my Mentor, Dr. Olufunke Oladipupo who gave this work the depth of knowledge it possesses
- I also appreciate the entire faculty members in the department for their support.

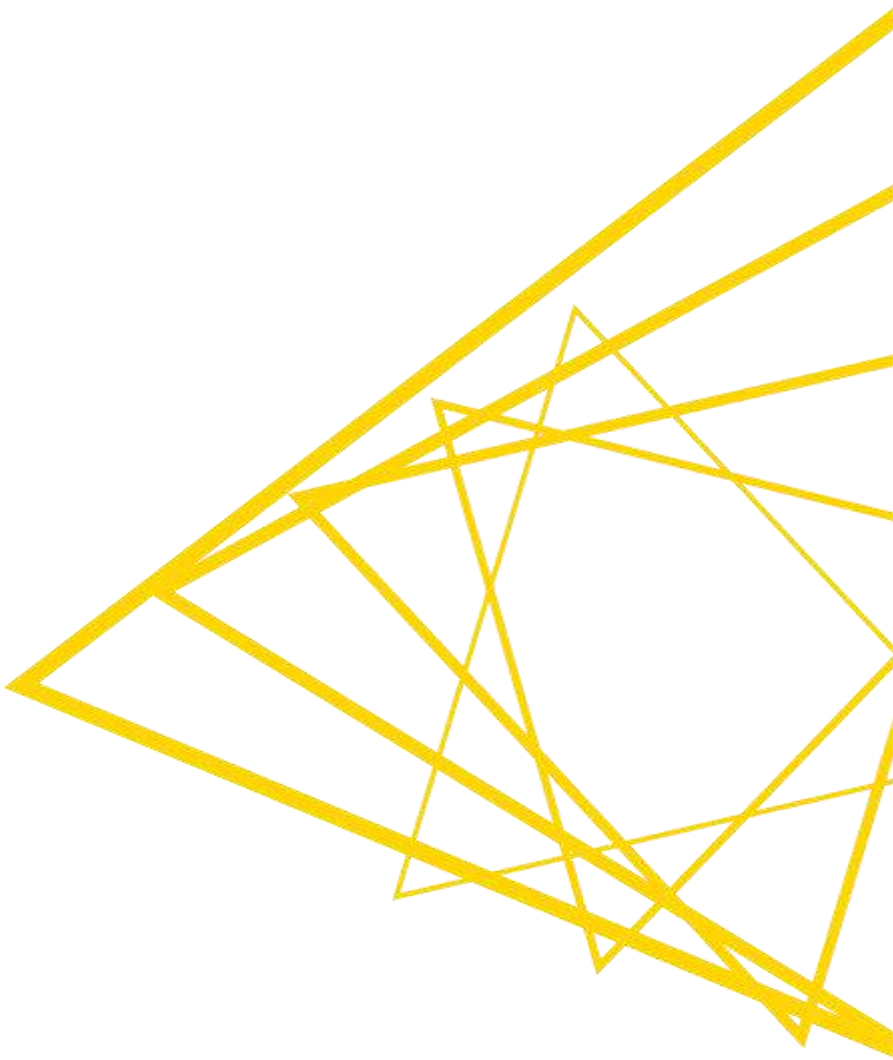
THANK YOU

FOR LISTENING

Open for Innovation
KNIME

[L4-DL] Introduction to Deep Learning – Session 1

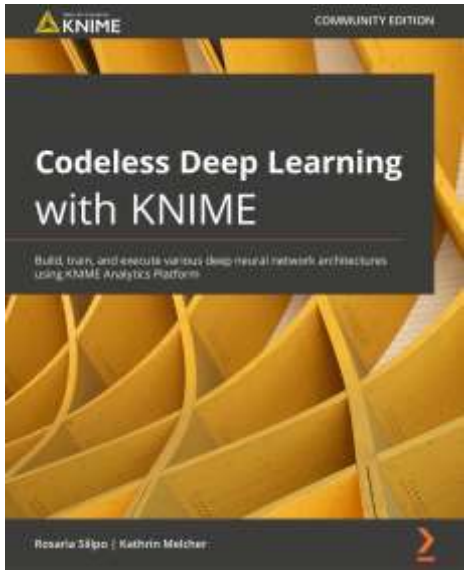
KNIME AG



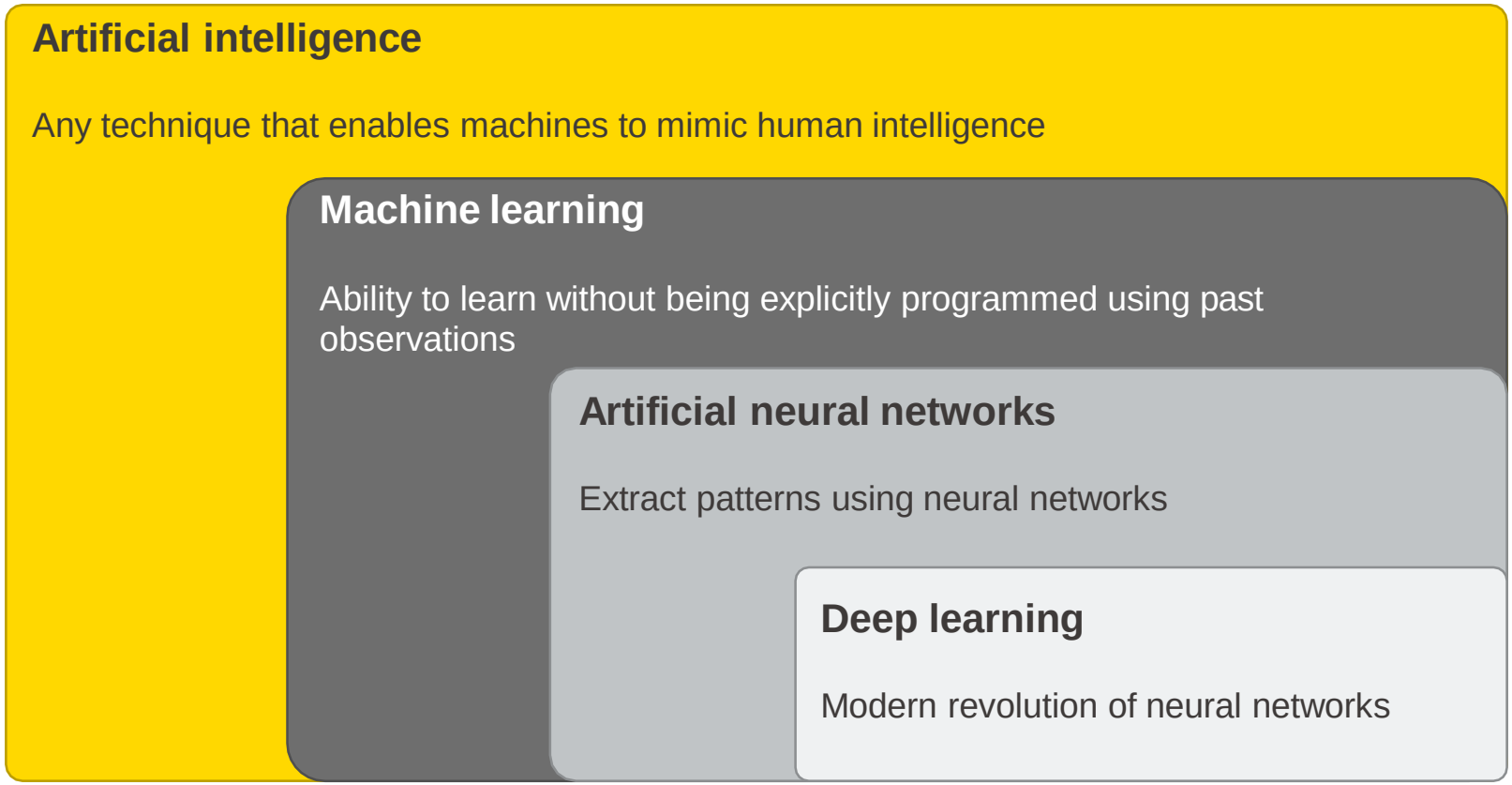
Material

- Guide to Intelligent Data Science—Springer, 2020
  - By Michael R. Berthold, Christian Borgelt, Frank Höppner, Frank Klawonn, Rosaria Silipo

- Codeless Deep Learning with KNIME—Packt, 2020
  - By Rosaria Silipo & Kathrin Melcher



What is Deep Learning?



Artificial Neurons and Networks

Artificial Neural Networks

- **Artificial Neural Networks** (**ANN**) are among the oldest and most intensely studied Machine Learning approaches

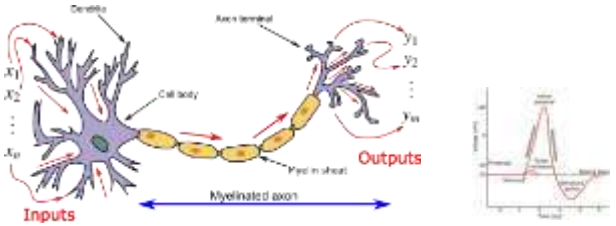
- Inspired by biological neural networks, they mimic the learning process of animals and humans

- However, the model is very coarse, and several improvements to the basic approach have even abandoned the biological analogy

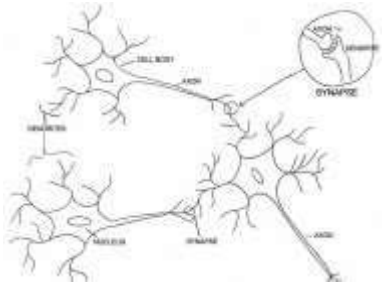
Advantage	Disadvantage
Highly flexible ➔ good performance	Black-box models not easy to interpret

Biological Neuron vs. Perceptron

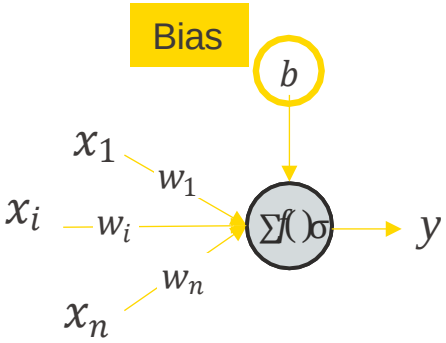
Biological Neuron



Biological Neural Networks



Artificial Neuron (**Perceptron**)



$$y = f(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + b)$$

$$b = w_0, \; x_0 = 1$$

$$y = f\left(\sum_{i=0}^{n} x_i w_i\right)$$

Input:

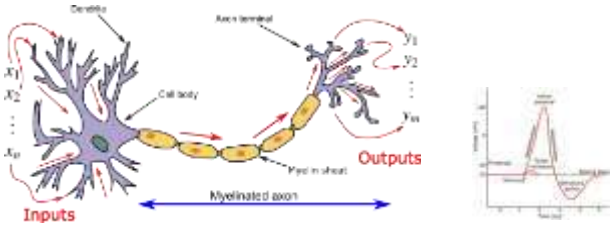
- numerical attributes x_i

Output:

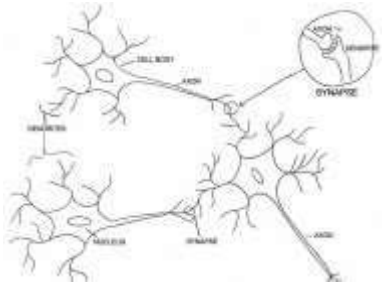
- Probability (0,1)
- Numerical outcome $(-\infty, \infty)$

Biological Neuron vs. Perceptron

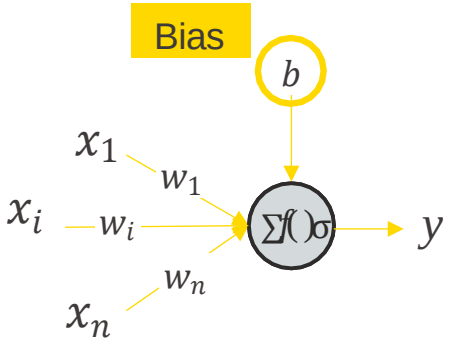
Biological Neuron



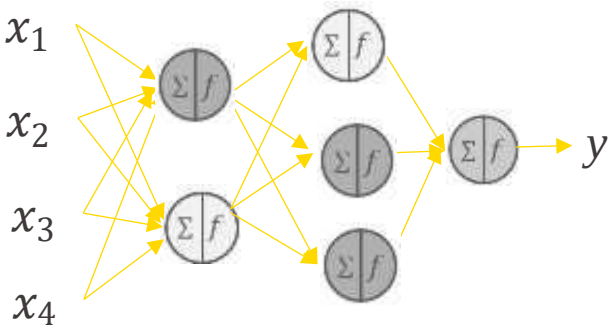
Biological Neural Networks



Artificial Neuron (**Perceptron**)

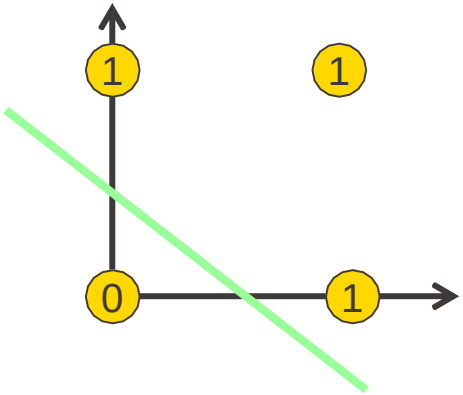


Artificial Neural Networks (**Multilayer Perceptron, MLP**)

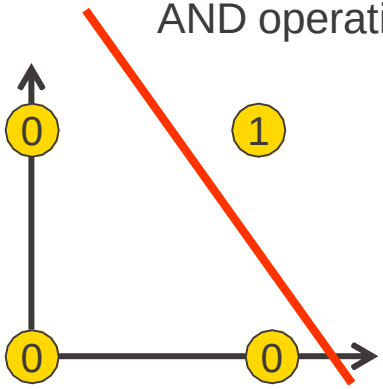


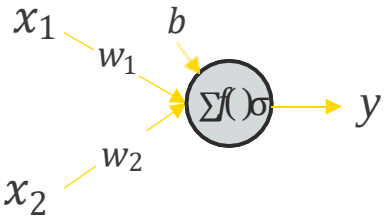
What Can a Single Perceptron Do?

OR operation

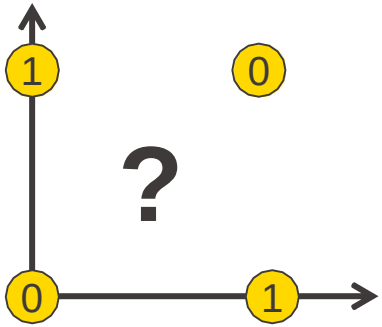


AND operation

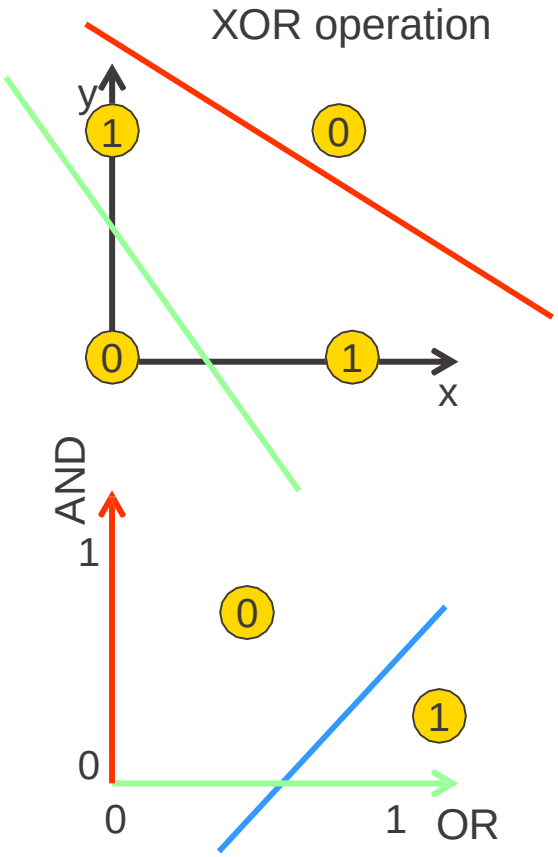


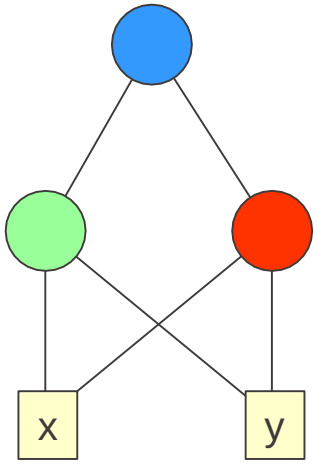


XOR operation



What Can a 3-neuron MLP Do?





Feedforward Neural Networks

Feedforward Neural Networks (FFNN)

A network of artificial neurons with multiple layers

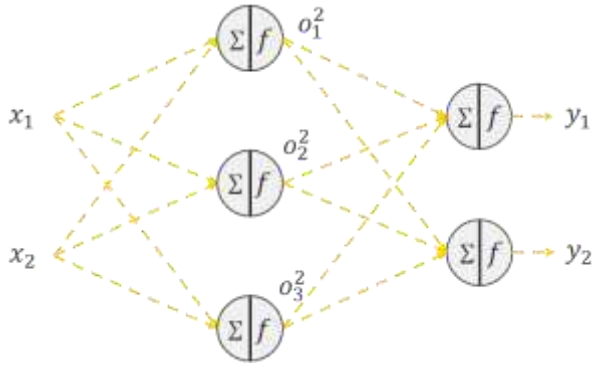
- an ***input layer***
- one or more ***hidden layers***
- an ***output layer***

- Connections exist only between neurons from one layer to the next (***feedforward***)
- MLP is a special case of FFNN, with non-linear activation functions in all layers

FFNN: Example of Architecture / Topology

Let's see an example of an FFNN

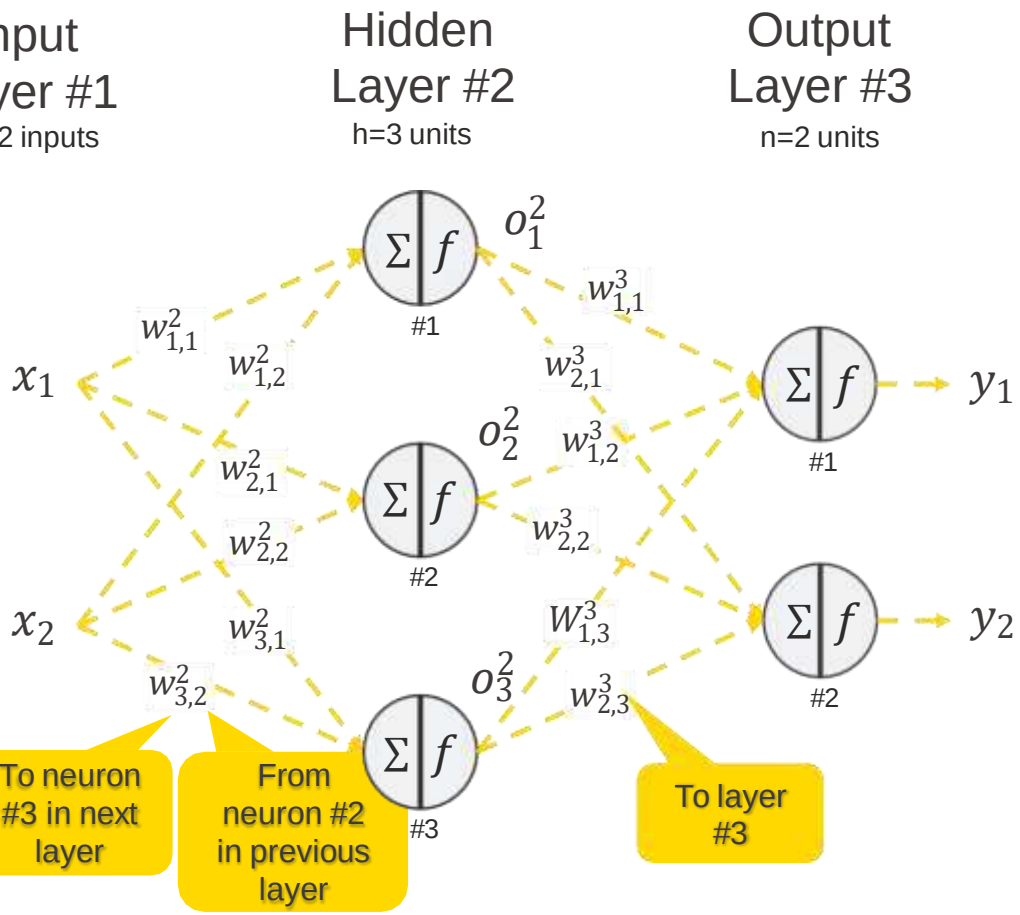
- 1 input layer with *m=2 inputs*
- 1 hidden layer with *h=3 hidden neurons*
- 1 output layer with *n=2 output neurons*



fully connected
feed forward neural network

- **Fully-connected**: Each neuron in one layer is connected to all neurons in the next layers
- **Dense layer**: fully connected to the previous layer

Feed-Forward Neural Networks (FFNN)



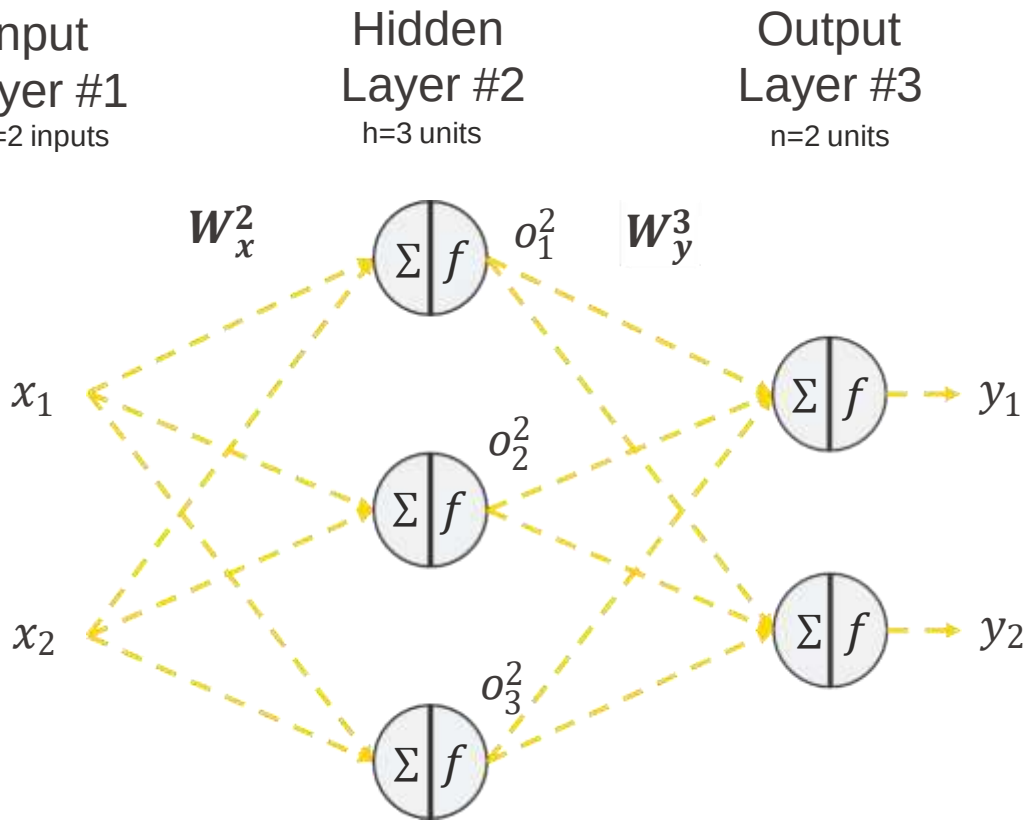
Forward pass:

$$o_{\mathrm{j}}^2 = f\left(\sum_{i=1}^{n} w_{\mathrm{j},i}^2 \, x_i\right)$$

$$y_k = f\left(\sum_{\mathrm{j}=1}^{h} w_{k,\mathrm{j}}^3 \, o_{\mathrm{j}}^2\right)$$

$$y_k = f\left(\sum_{\mathrm{j}=1}^{h} w_{k,\mathrm{j}}^3 \, f\left(\sum_{i=1}^{n} w_{\mathrm{j},i}^2 \, x_i\right)\right)$$

Same with Matrix Notations



Forward pass:

$$\boldsymbol{o} = f(W_x^2 \boldsymbol{x})$$

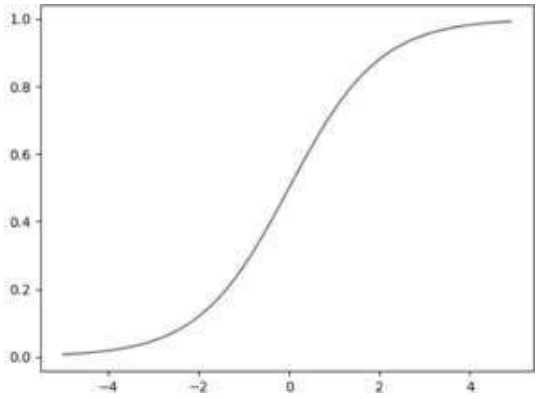
$$y = f(W_y^3 \boldsymbol{o})$$

$$y = f(W_y^3 f\,(W_x^2 \boldsymbol{x}))$$

$f(\,)$ = activation function

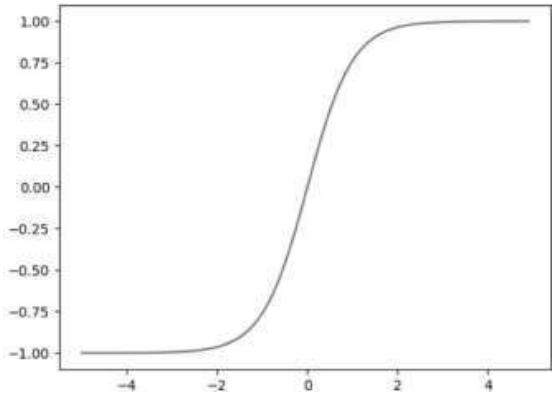
Frequently Used Activation Functions

Sigmoid



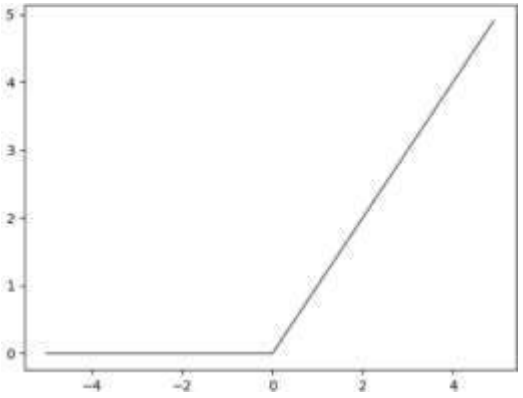
$$f(a) = \frac{1}{1 + e^{-ha}}$$

Tanh



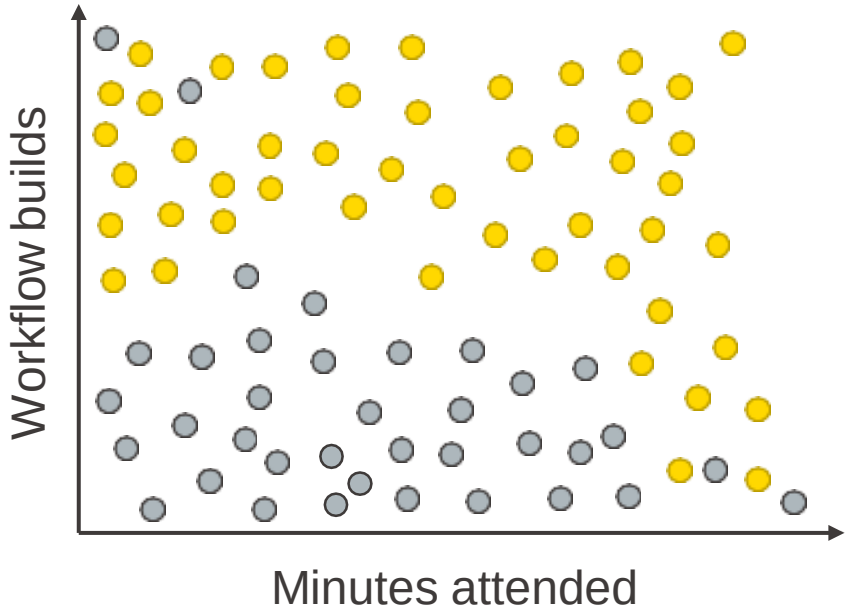
$$f(a) = \frac{e^{2ha} - 1}{e^{2ha} + 1}$$

Rectified Linear Unit (ReLU)



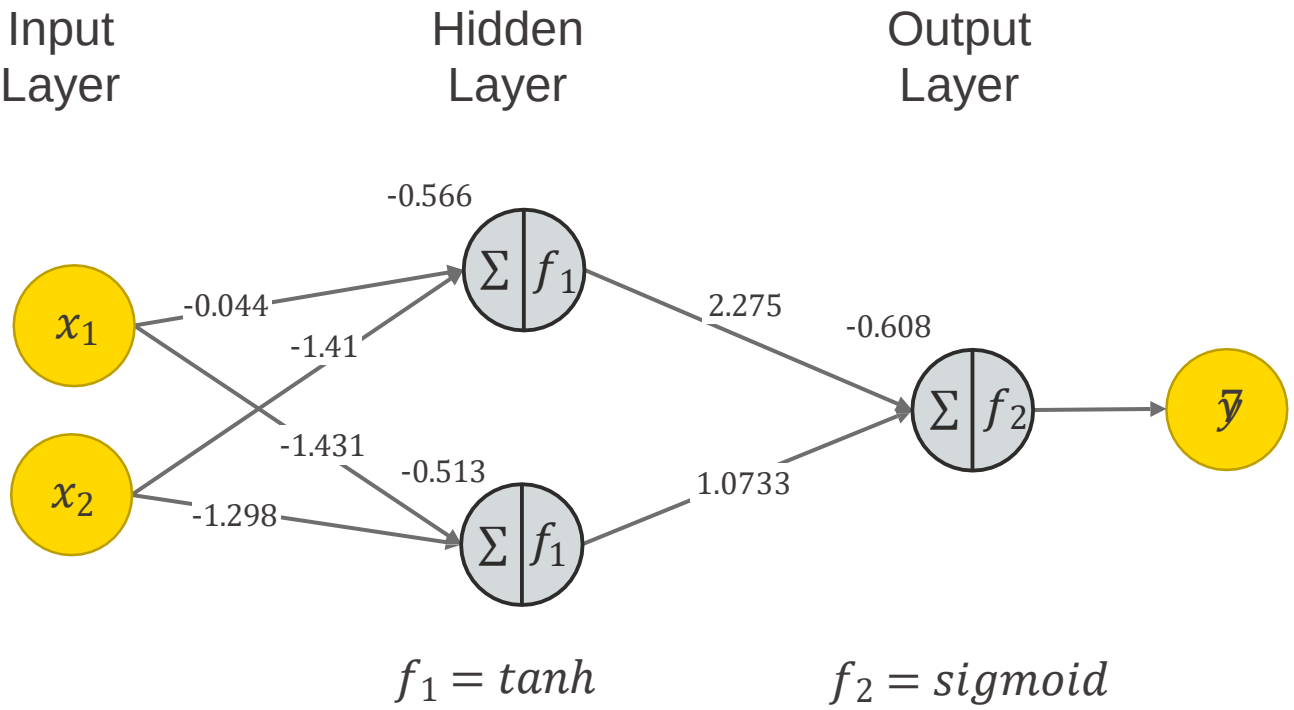
$$f(a) = max\{0, ha\}$$

Example: Passing KNIME L1 Certification



- Passed certification
- Didn’t pass certification

Example: Passing KNIME L1 Certification

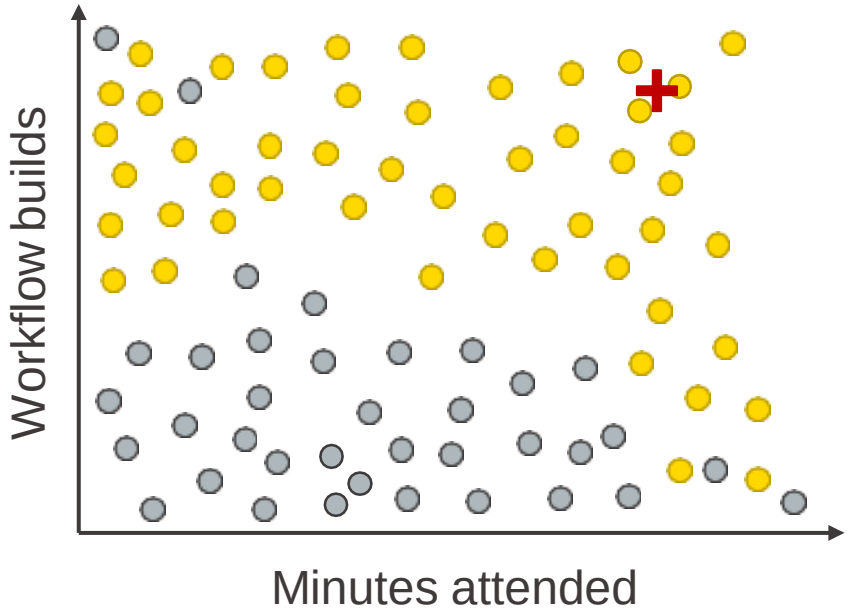


Input features:
x_1= minutes attended
x_2= workflows build

Weights:
Randomly assigned

Output:
$\hat{y}$ =Probability that a person passed
$\hat{y} \geq 0.5 \Rightarrow Passed$ and $\hat{y} < 0.5 \Rightarrow Failed$

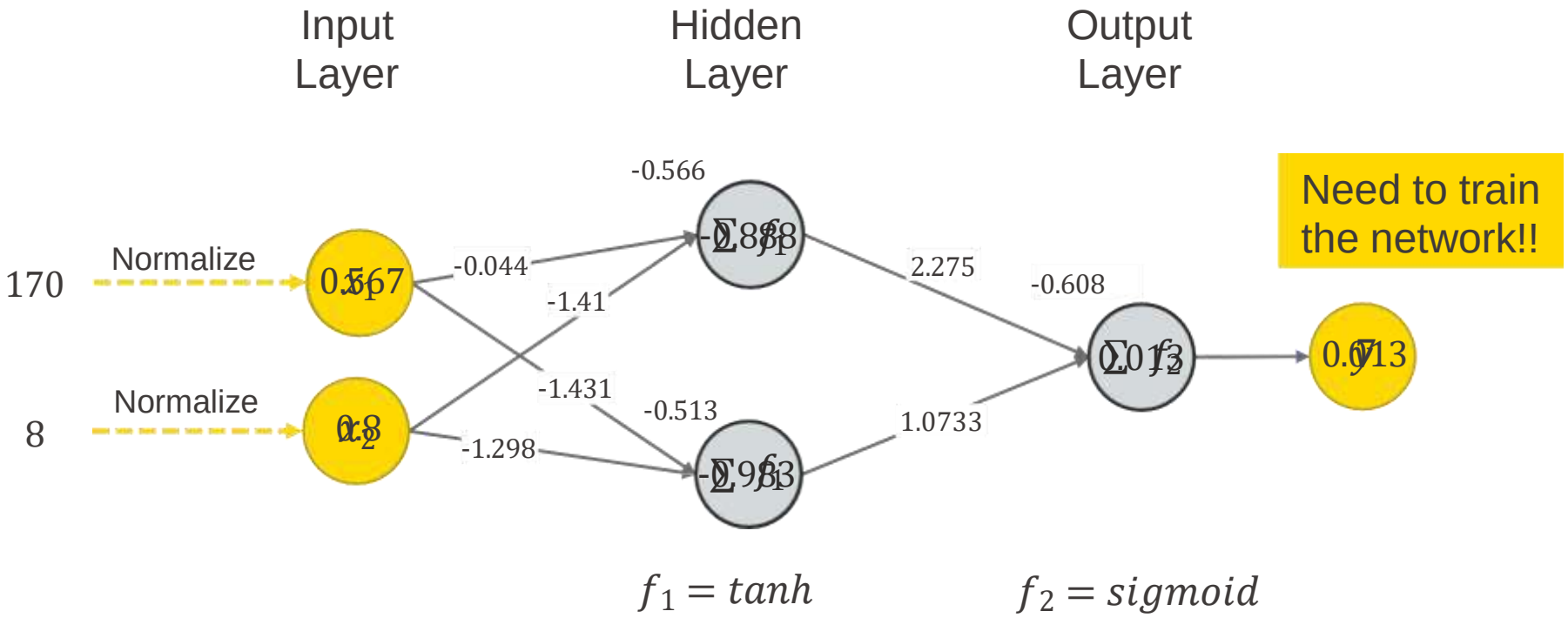
Example: Passing KNIME L1 Certification



- Passed certification
- Didn't pass certification
- New sample

x_1= minutes attended = 170
x_2= workflows build = 8

Example: Passing KNIME L1Certification

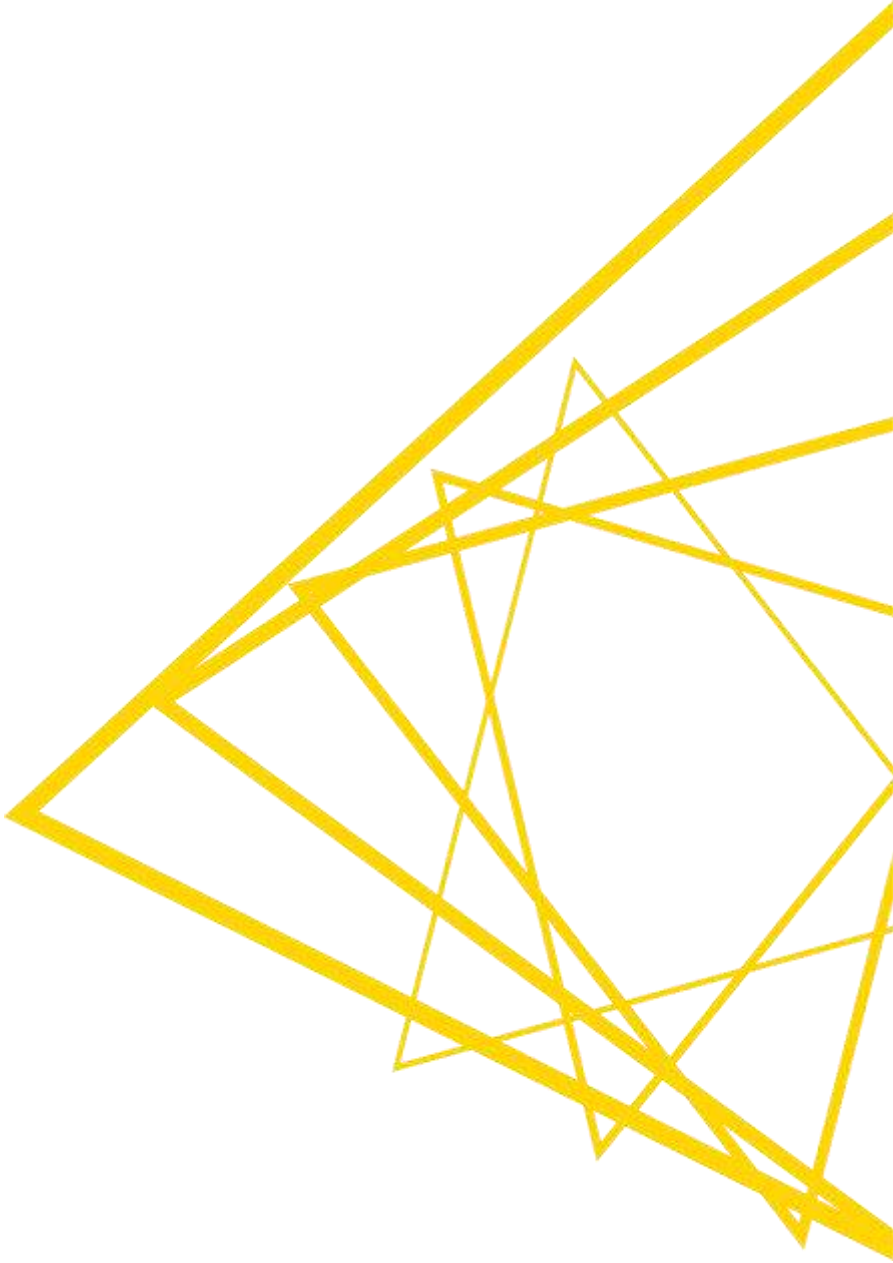


Input features:
x_1= minutes attended
x_2= workflows build

Weights:
Randomly assigned

Output:
$\hat{y}$ =Probability that a person passed
$\hat{y} \geq 0.5 \Rightarrow Passed$ and $\hat{y} < 0.5 \Rightarrow Failed$

Back-Propagation



Training a Feed Forward Neural Network

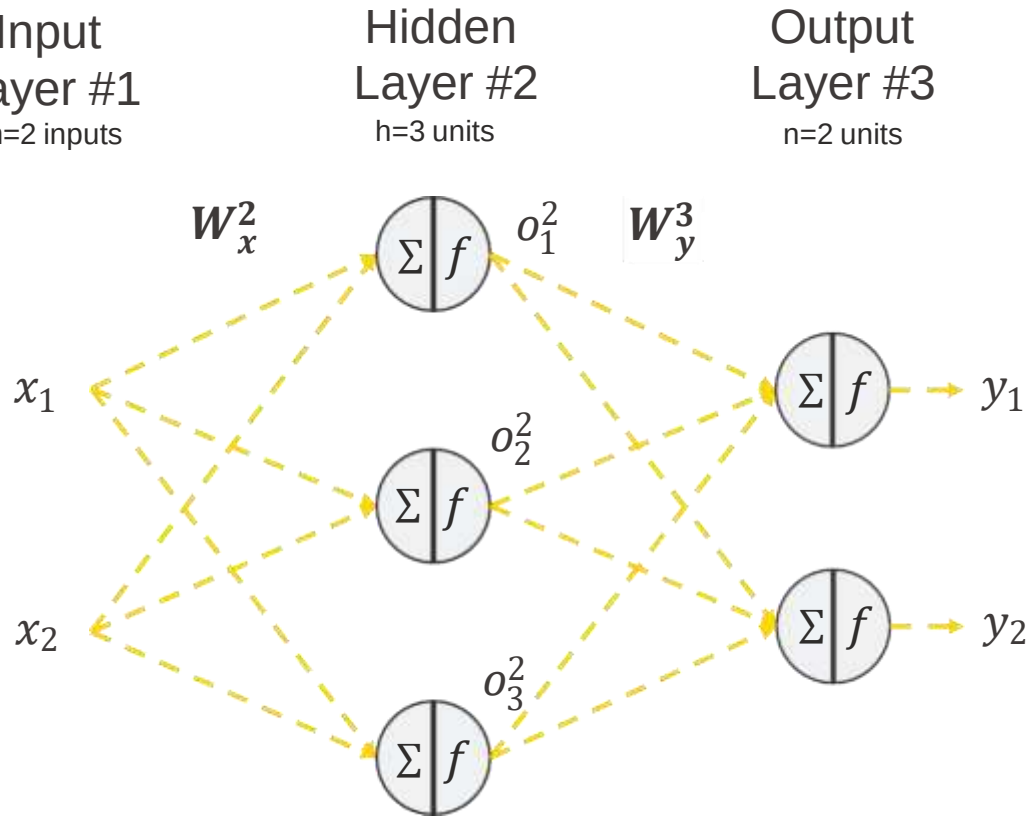
- Teach (ensemble of) neuron(s) a desired input-output behavior
- Show examples from the training set repeatedly
- Networks adjusts parameters to fit underlying function

  → topology

  → weights

  → internal functional parameters

Loss Function



Number of output units

$$\varepsilon = \frac{1}{2} \% \quad \% \quad (y_k - t_k)^2$$

n — $x \in T$ — $k=1$

On the whole Training set

Network output value k

Target value k

Forward pass:

$$\boldsymbol{o} = f(W_x^2 \boldsymbol{x})$$

$$y = f(W_y^3 \boldsymbol{o})$$

Learning Rule from Gradient Descent

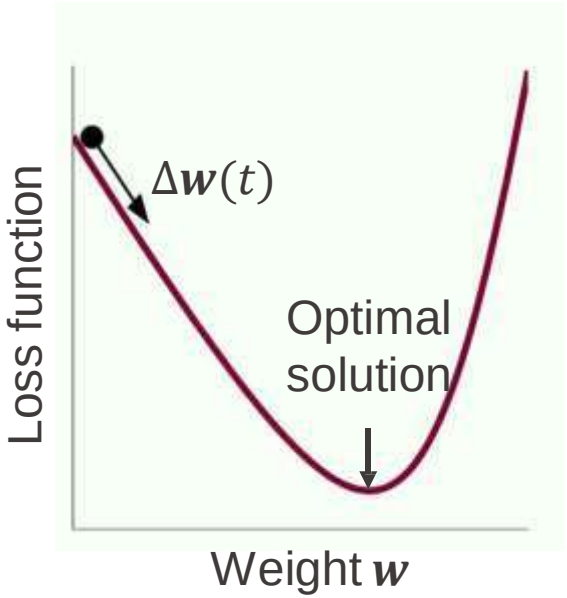
- Adjust the weight for the next step by the adjustment term $\Delta \boldsymbol{w}(t)$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \Delta \boldsymbol{w}(t)$$

Updated weight in next step $t+1$

Weight in current step t

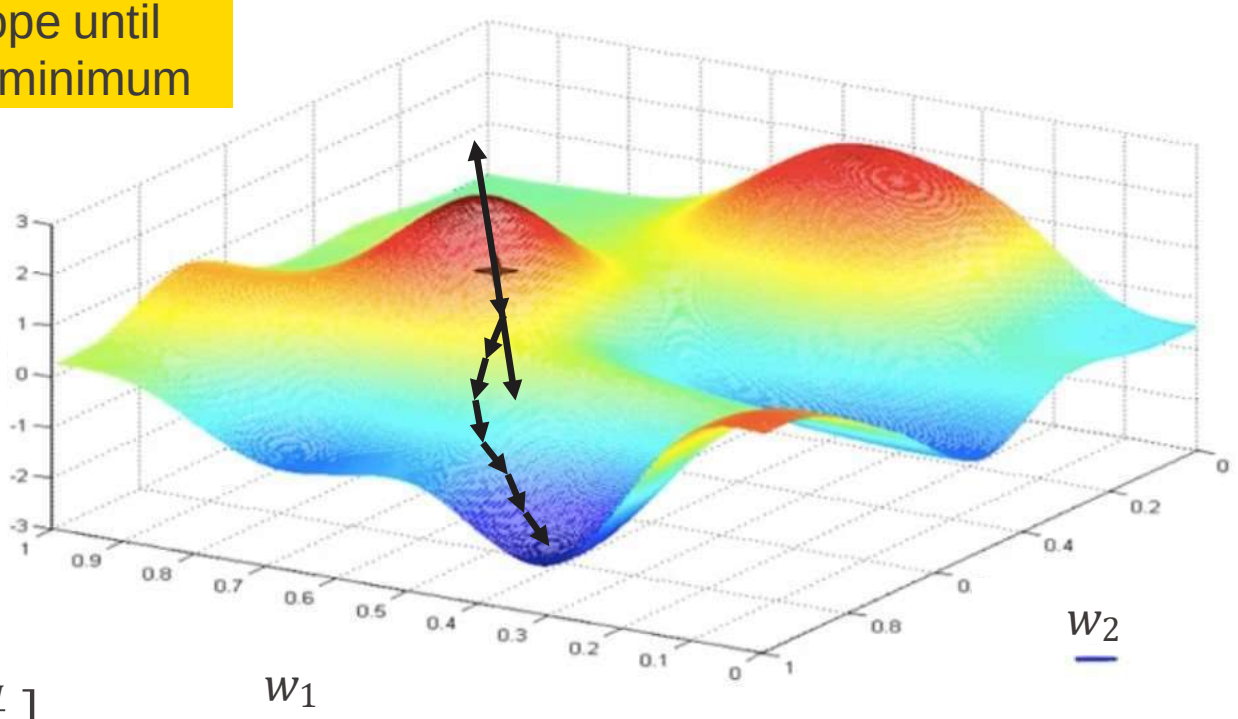
Weight adjustment based on gradient



Idea Behind Gradient Descent

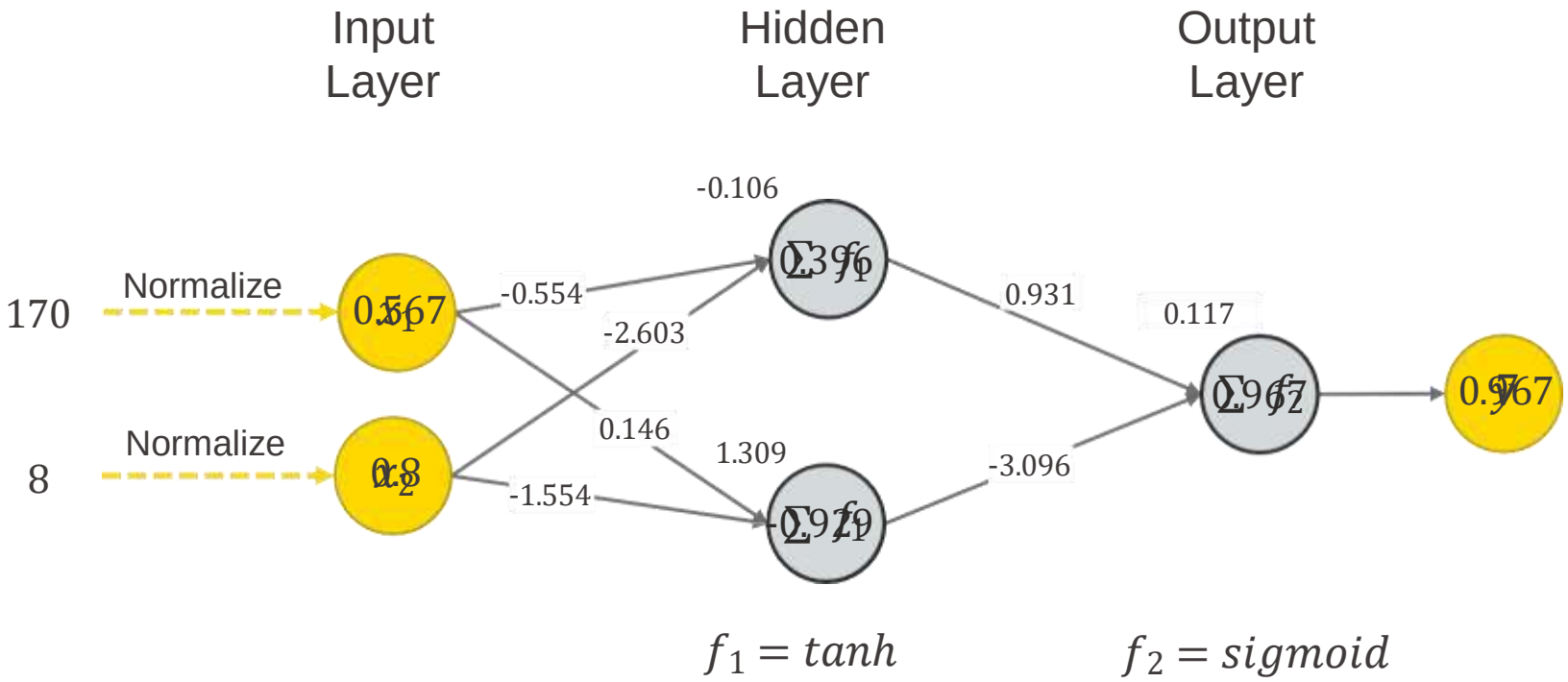
Rolling down the steepest slope until reaching the minimum

$J(w_1, w_2)$



$$\nabla_W J(x, W) = \begin{bmatrix} \frac{\partial J}{\partial 0_)} \\ \frac{\partial J}{\partial 0_*} \end{bmatrix}$$

Example: Passing the KNIME L1 Certification



Input features:
x_1= minutes attended
x_2= workflows build

Weights:
Trained with gradient descent

Output:
y = Probability that a person passed
$y \geq 0.5 \Rightarrow Passed$ and $y < 0.5 \Rightarrow Failed$

Learning Rule from Gradient Descent

- $\Delta \boldsymbol{w}$ has two parts:

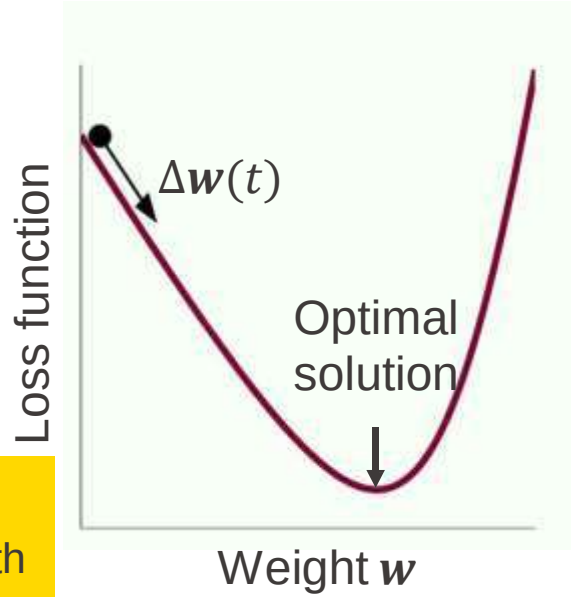
$$\Delta \boldsymbol{w}(t) = -\eta \frac{\partial \varepsilon}{\partial \boldsymbol{w}}$$

Need to calculate as many partial derivatives as the number of all weights!

Learning rate $\eta > 0$:
Dictates how quickly to approach the solution

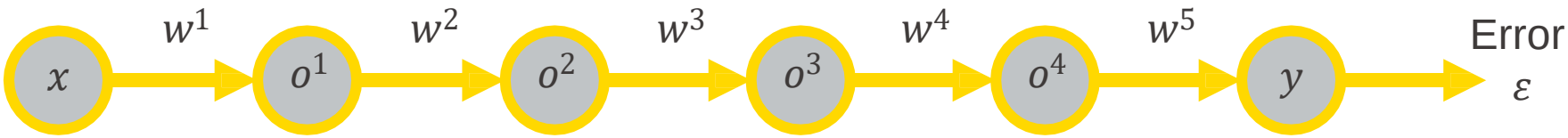
Slope of Loss function

Gradient:
Partial derivatives with respect to weight $\boldsymbol{w}$



Gradients by Chain Rule

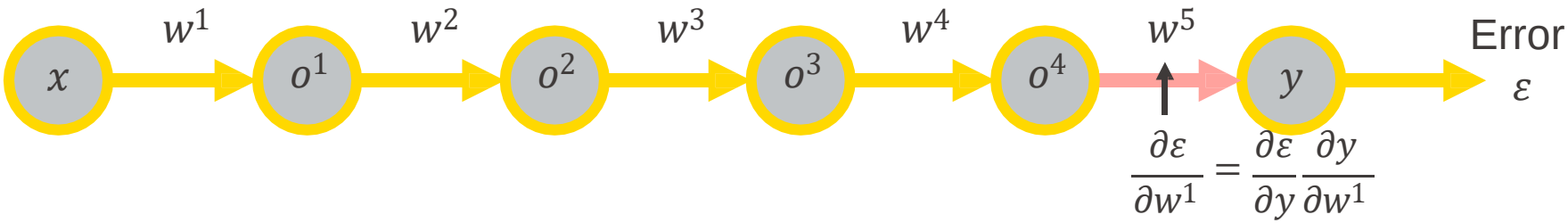
- Gradients can be determined – one layer at a time – by the chain rule



- Consider a simple feedforward network

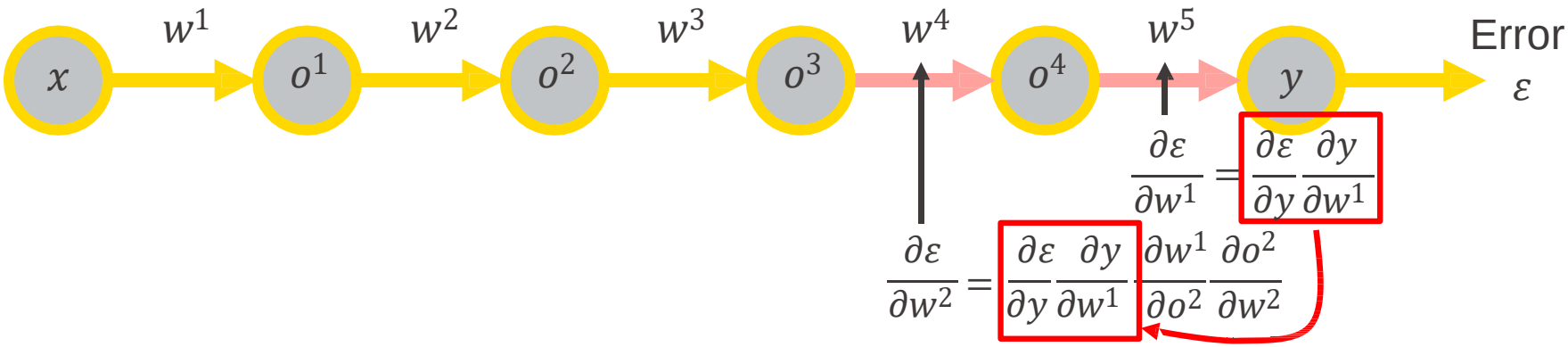
Gradients by Chain Rule

- Gradients can be determined – one layer at a time – by the chain rule



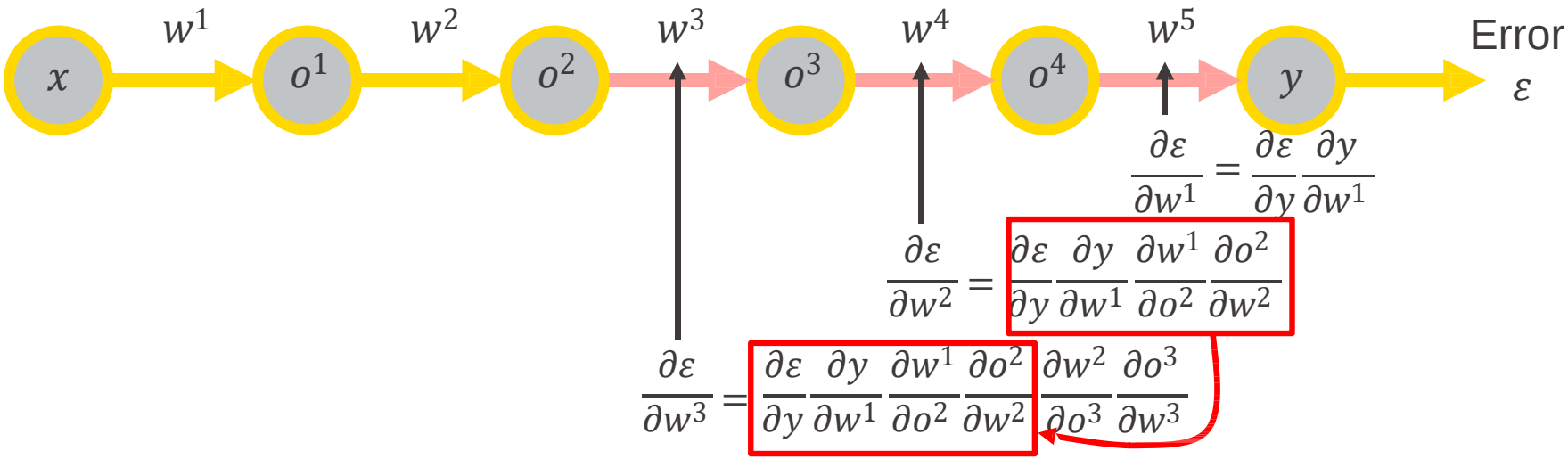
Gradients by Chain Rule

- Gradients can be determined – one layer at a time – by the chain rule



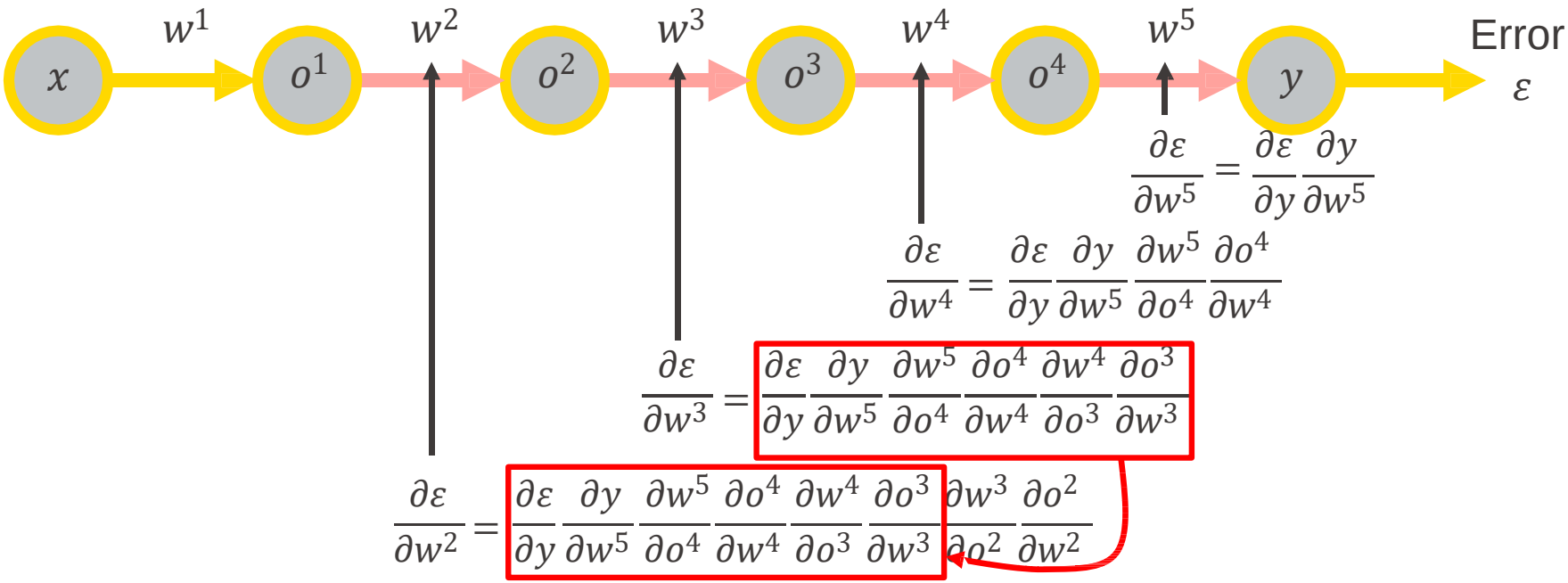
Gradients by Chain Rule

- Gradients can be determined – one layer at a time – by the chain rule



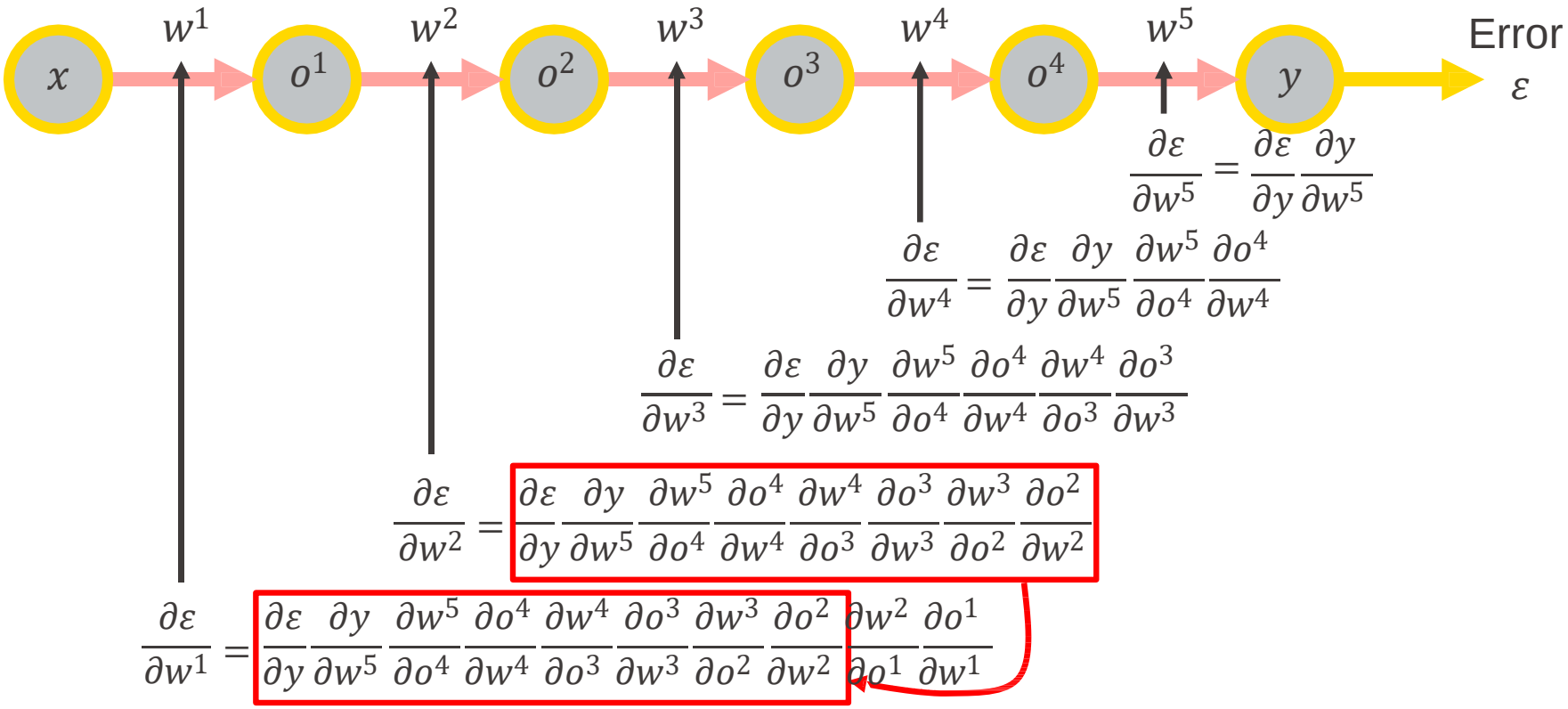
Gradients by Chain Rule

- Gradients can be determined – one layer at a time – by the chain rule



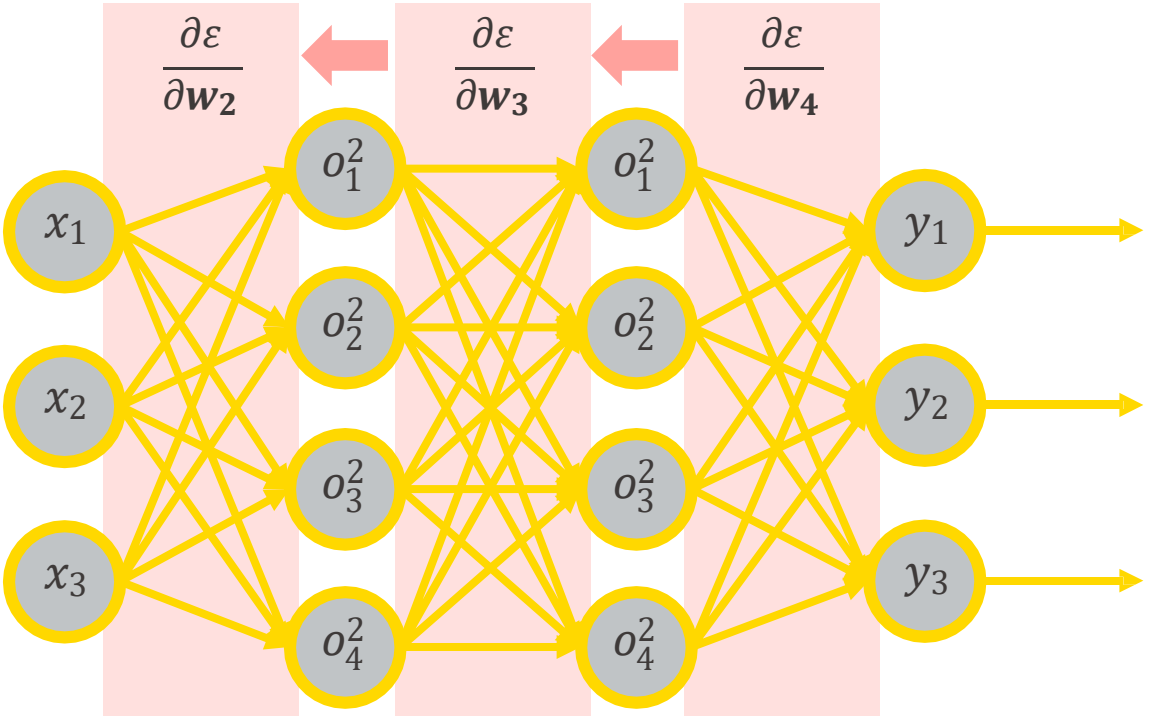
Gradients by Chain Rule

- Gradients can be determined – one layer at a time – by the chain rule



Gradients by Chain Rule

- Gradients can be determined – one layer at a time – by the chain rule
- To determine the gradient at a particular layer, you only need gradients from the subsequent layers ➔ known as ***back-propagation***



Updating Weights

Actual updating of weights:

- Starting from the output layer → previous layer, one layer at a time
- Remember the formula:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \Delta\boldsymbol{w}(t)$$

Updating Weights

Actual updating of weights:

- Updating weight w_{ji}, neuron i to output neuron j with

$$\Delta w_{ji} = -\eta \sum_{x \in E} \delta_j o_i$$

Output from neuron i

- With MSE loss function $\frac{1}{n}\sum_{j=1}^{n}\left(y_j - t_j\right)^2$, δ_j is calculated as

Prediction error
(at the output layer)

δ's and w's from the next layer
(at the hidden layer)

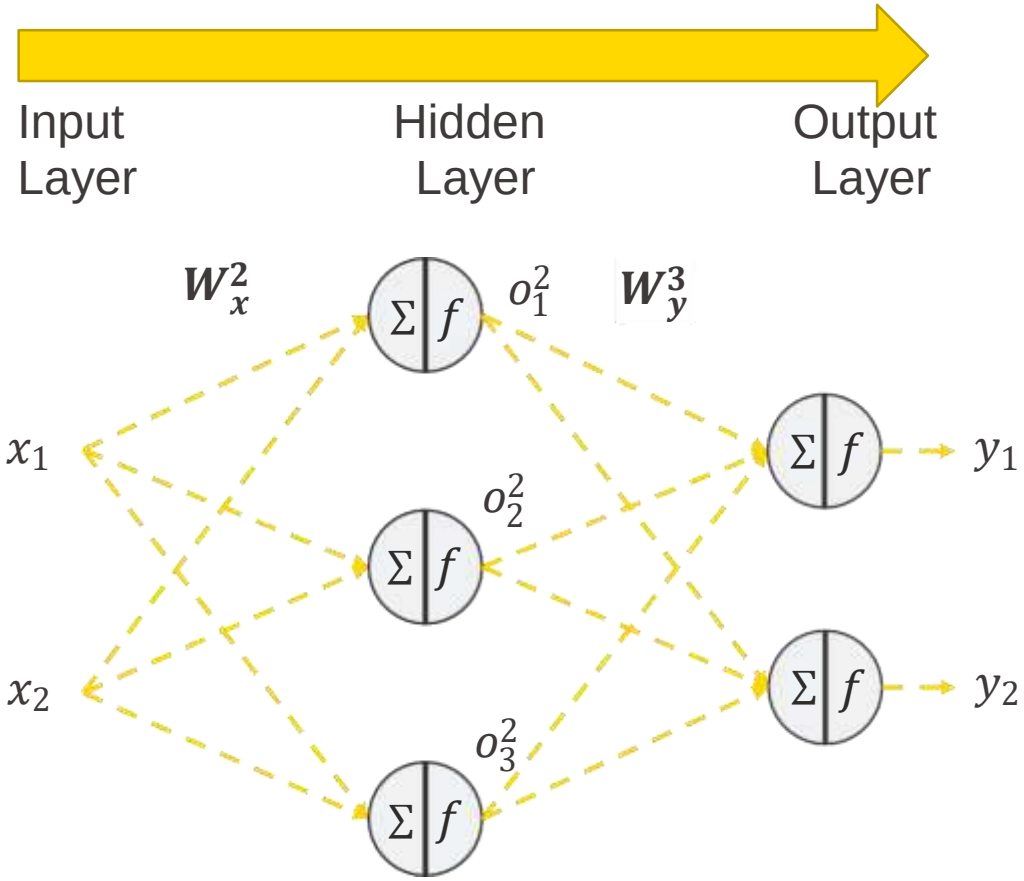
$$\delta_j = \begin{cases} \left(y_j - t_j\right) f'(y_j) \\ \sum_{k=1}^{n} \delta_k^{L+1} w_{jk}^{L+1} f'\left(y_j^L\right) \end{cases}$$

Neuron j is in the output layer

Neuron j is in the hidden layer L

Derivative of activation function
(at the current layer)

Step 1. Forward Pass

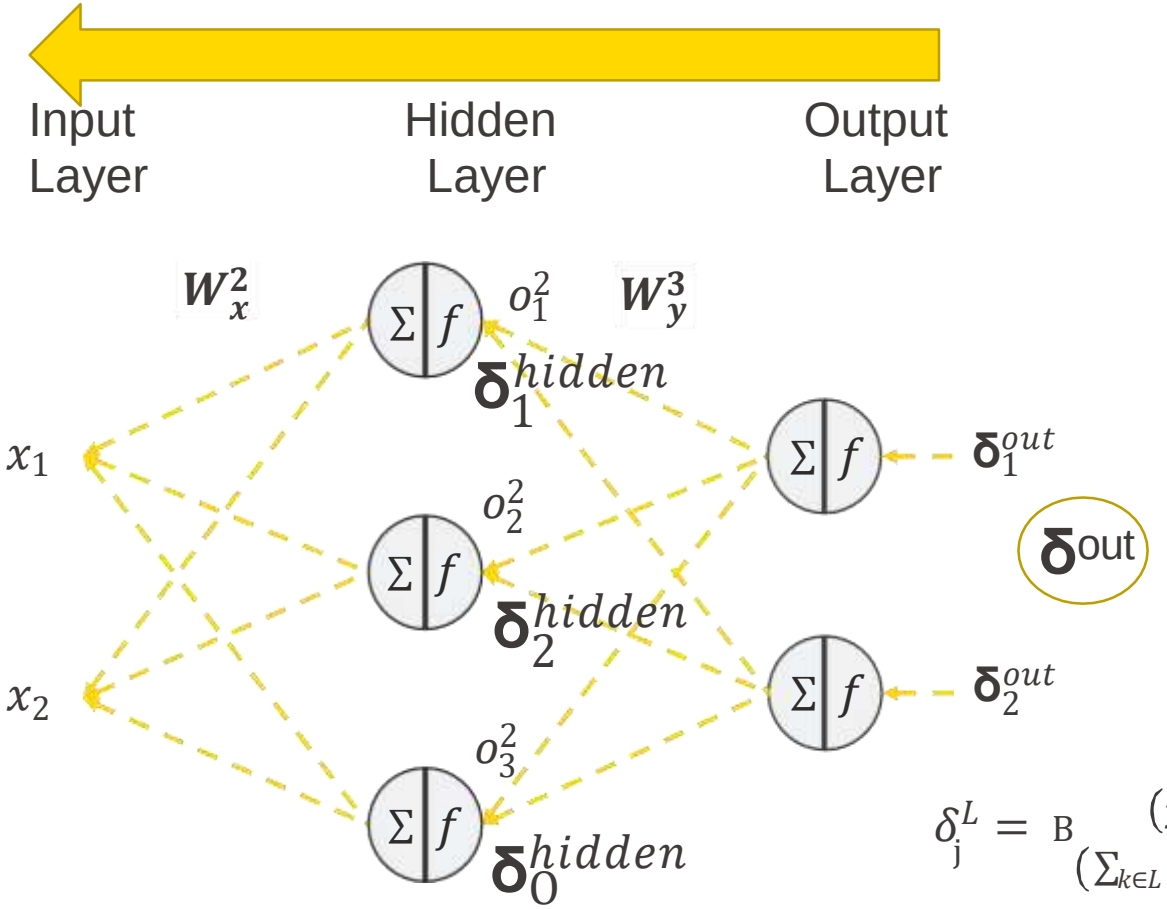


1. Forward pass:

$$o_{\mathrm{j}}^2 = f\left(\sum_{i=1}^{n} w_{\mathrm{j}i}^2 \, x_i\right)$$

$$y_k = f\left(\sum_{\mathrm{j}=1}^{h} w_{k\mathrm{j}}^3 \, o_{\mathrm{j}}^2\right)$$

Step 2. Backward Pass - δ



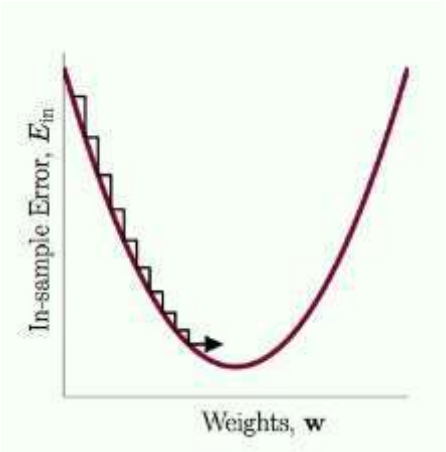
2. Backward pass:

$$\delta_j^L = \begin{cases} (y_+ - t_+)\, f'(y_+) & L = last\ layer \\ \left(\sum_{k \in L} w_{+k}\delta_k\right) f'(y_+) & L = hidden\ layer \end{cases}$$

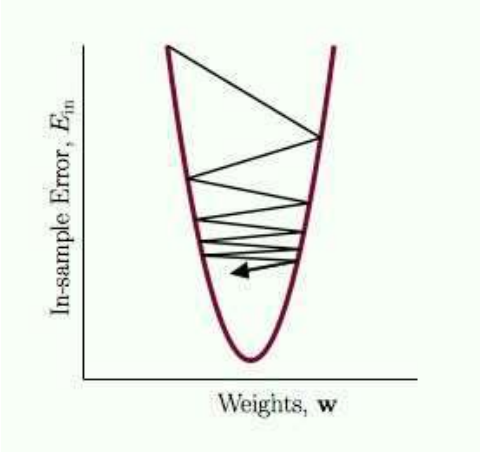
Learning Rate η

- Network weights are updated by back propagation
- How much weight adjustment? → controlled by the learning rate η
- Optimizers → various variants of stochastic gradient descent (SGD)

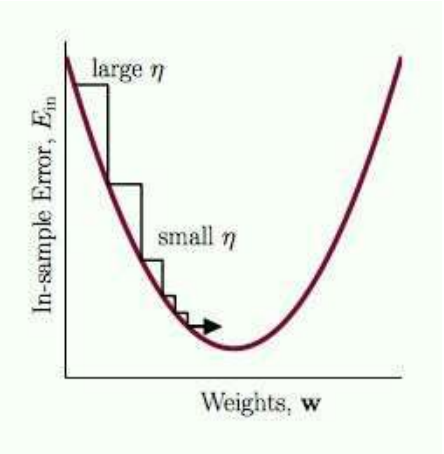
η too small



η too large



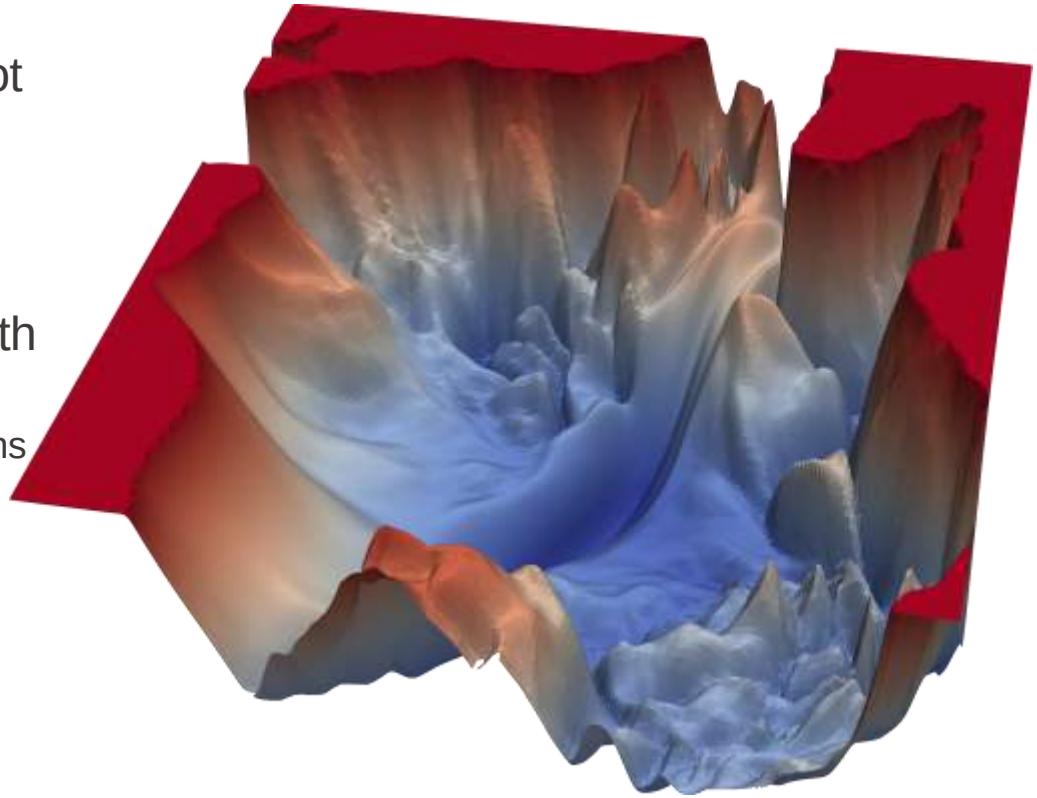
η just right



Optimizers in Keras

- In reality, loss landscape may not be smooth
  - Possibly many local minima

- Different optimizer algorithms with
  - Varying learning rate η
  - History of gradients in previous iterations

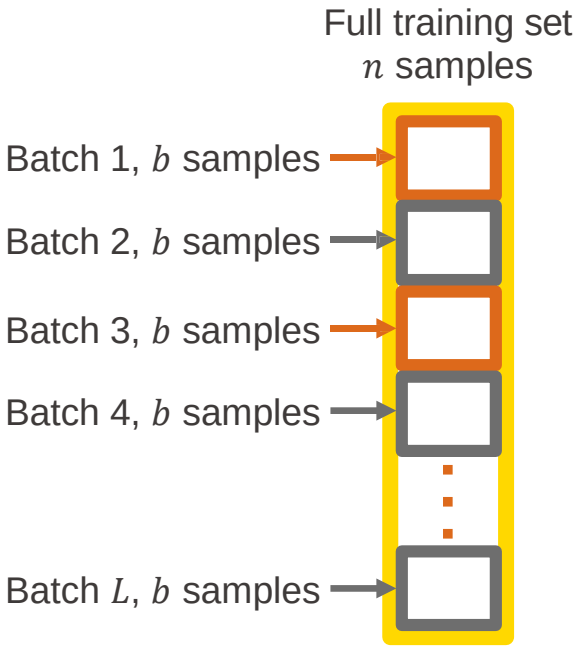


Source: https://www.cs.umd.edu/~tomg/projects/landscapes/

Optimizers in Keras

Optimizer	How it works	Strengths	Weaknesses	When to use
SGD with momentum	Use the previous gradient to accelerate convergence	-Reduces oscillation near maxima	-Const learning rate	
NAG (Nesterov accelerated gradient)	Use the current gradient to predict gradient	-Increased responsiveness	-Additional hyperparameter	RNN
Adagrad	Updating by cumulating sum of sq gradients from past	-Different learning parameters for different features	-Computationally expensive -Shrinking learning rate	Sparse data (e.g. text)
Adadelta	Modified Adagrad with decaying average of sq gradients from past	-Learning rate not dramatically shrinking like Adagrad	-Computationally expensive	Sparse data (e.g. text)
RMSProp	Modified Adagrad with sq gradients added very slowly	-Learning rate not dramatically shrinking like Adagrad		
Adam (Adaptive Moment Estimation)	RMSProp plus decaying average of gradients from past	-Fast convergence	-Computationally expensive	

Batch Size & Epochs



Batch size

- Number of samples in each iteration of training
- Batch size = n → ***batch training***
  - Computationally expensive but correct
- Mini-batch mode: batch size $b < n$ → ***online training***
  - Faster but prone to running into oscillations
- L iterations to go through the entire training data set ➔ referred as an **epoch**

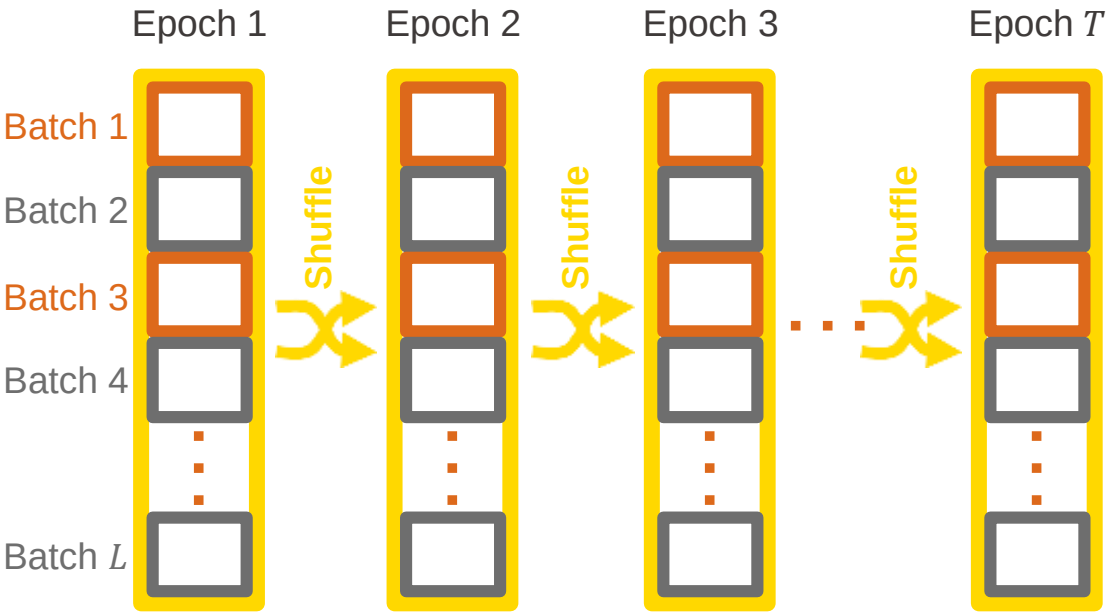
Training data segmented into L batches of size b ($n = b \times L$)

- Often batch sizes are chosen to be powers of 2 (e.g., 16, 32, 64, 128, 256, …)
- Small batches for deeper networks

Batch Size & Epochs

Number of **Epochs**

- Number of cycles to run the full training set
- Training set is shuffled in each epoch → different batches in each epoch

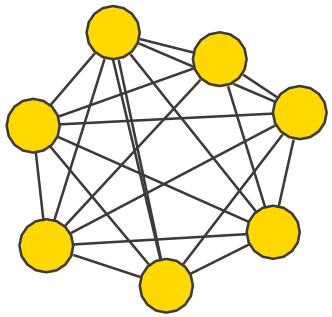


In total, the training algorithm runs

L batches $\times$ T epochs times

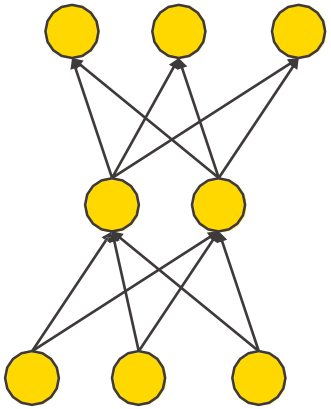
Other Neural Architectures

Completely connected



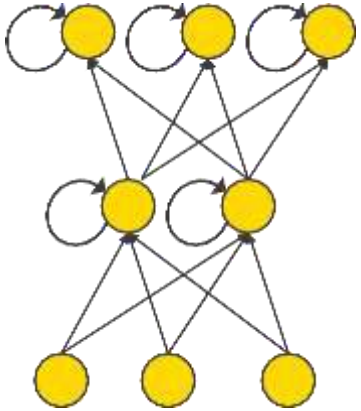
Example:
- associative neural network
- Hopfield

Feedforward (directed, a-cyclic)



Example:
- Multi-Layer Perceptron
- Autoencoder MLP

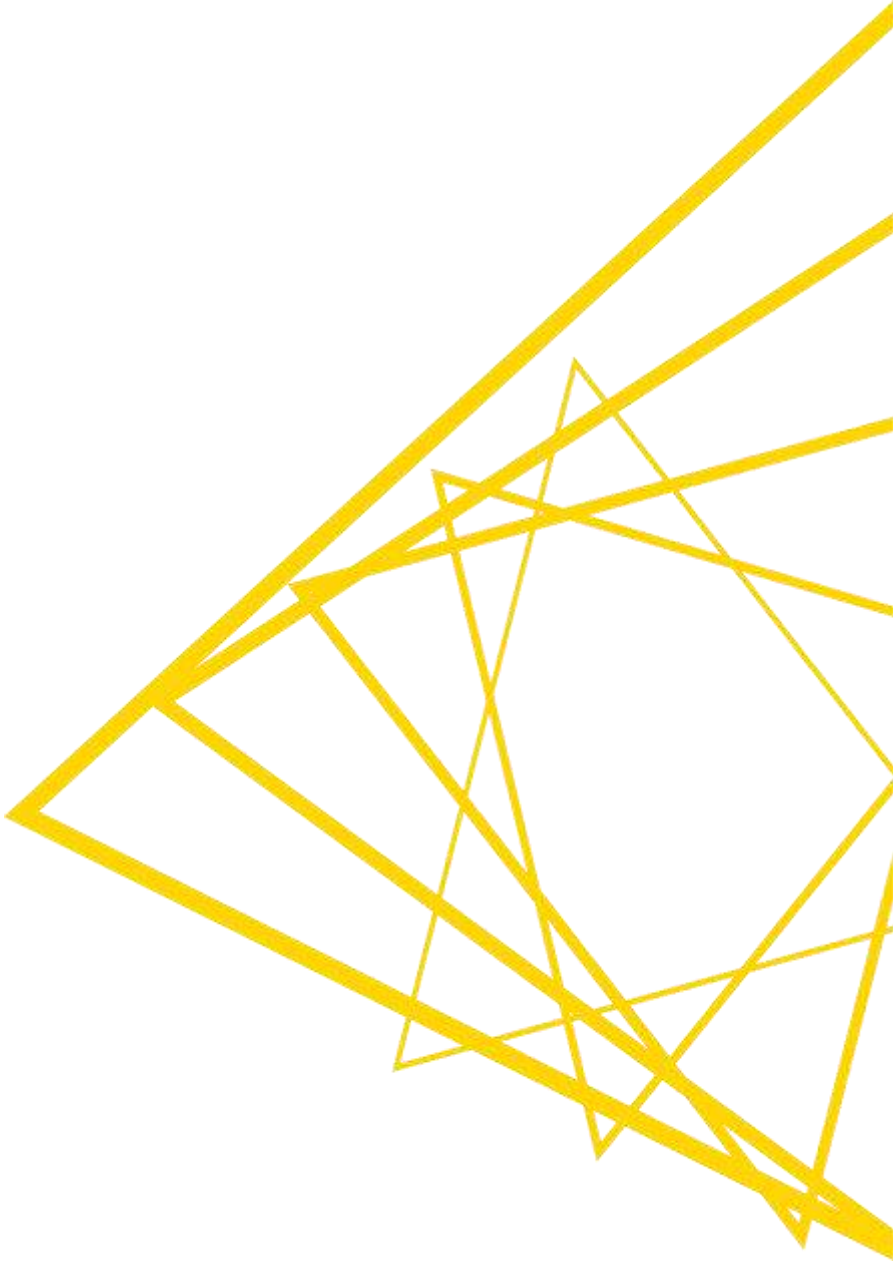
Recurrent (feedback connections)



Example:
-Recurrent Neural Network (for time series recognition)

KNIME Deep Learning Extension

Activation Functions



Why Do We Need An Activation Function?

- Linear combinations of linear functions can only model linear responses
  - Even a deep network could be described as a simple linear transformation, if ***every layer were linear***

$$A\Big(B\Big(C\Big(D\big(E(F\boldsymbol{x})\big)\Big)\Big)\Big) = (ABCDEF)\boldsymbol{x} = M\boldsymbol{x}$$

- Unable to solve non-linear problems
  - For example, $f(x) = x^2$ then $f(a) + f(b) = a^2 + b^2 \neq (a+b)^2 = f(a+b)$

- Non-linear activation functions enable modeling of non-linear problems

Activation Functions

At each neuron, the output is generated by

- 1. Calculating the weighted sum of the inputs

$$z_i = \sum_{j=0}^{n} w_{ji} x_j$$

Input j

Weight from input j to neuron i

$w_{0i}x_0$: bias (constant offset term)

- 2. Applying an activation function to the sum

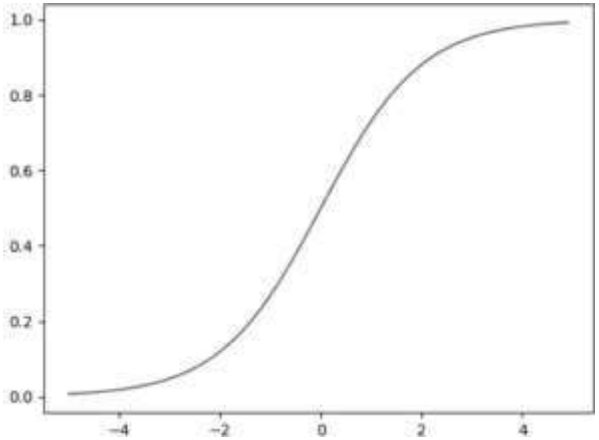
$$f(z_i) = f\left(\sum_{j=0}^{n} w_{ji} x_j\right)$$

- For back-propagation to work, activation functions must be ***differentiable***

Commonly Used Activation Functions

Sigmoid function

$$f(z_i) = \frac{1}{1 + e^{-z_i}} = \frac{e^{z_i}}{e^{z_i} + 1}$$



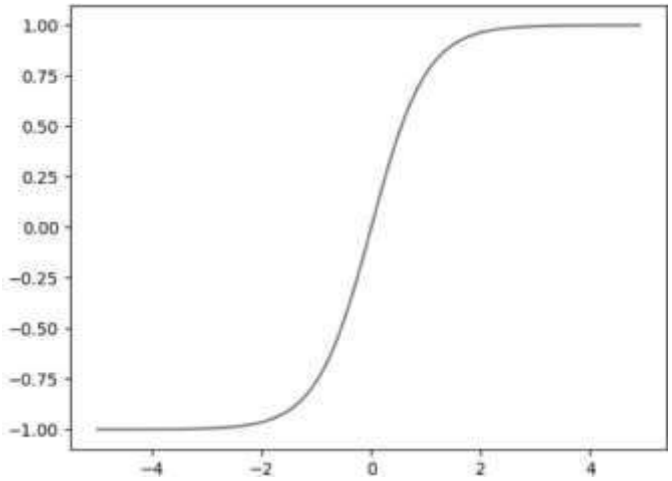
- S-shaped function
- Output range is (0,1) → ideal for describing probabilities
- Used as the last layer in **binary classification** problem
  - E.g., income ≥$50k vs. income <$50k
- Differentiable everywhere, and $f'(z_i) = f(z_i)(1 - f(z_i))$
- Prone to vanishing gradient problem

  → Small gradients (<1) diminish gradients in back-propagation

Commonly Used Activation Functions

Hyperbolic Tangent (Tanh) Function

$$f(z_i) = \frac{1 - e^{-2z_i}}{1 + e^{-2z_i}} = \frac{e^{z_i} - e^{-z_i}}{e^{z_i} + e^{-z_i}}$$

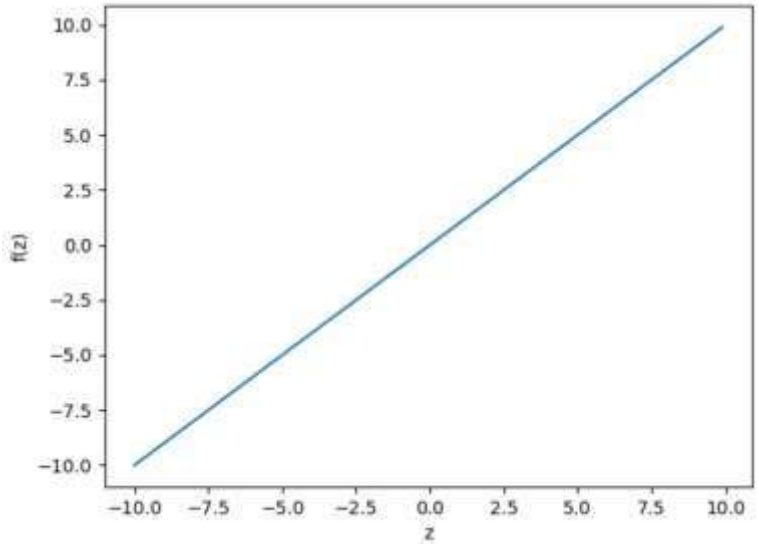


- S-shaped function
- Output range is (-1,1) → useful with the **hinge** loss function
- Differentiable everywhere, and $f'(z_i) = 1 - f^2(z_i)$
- Centered around 0 → helps stabilize the training process
- Prone to vanishing gradient problem

Commonly Used Activation Functions

Linear Function

$$f(z_i) = z_i$$

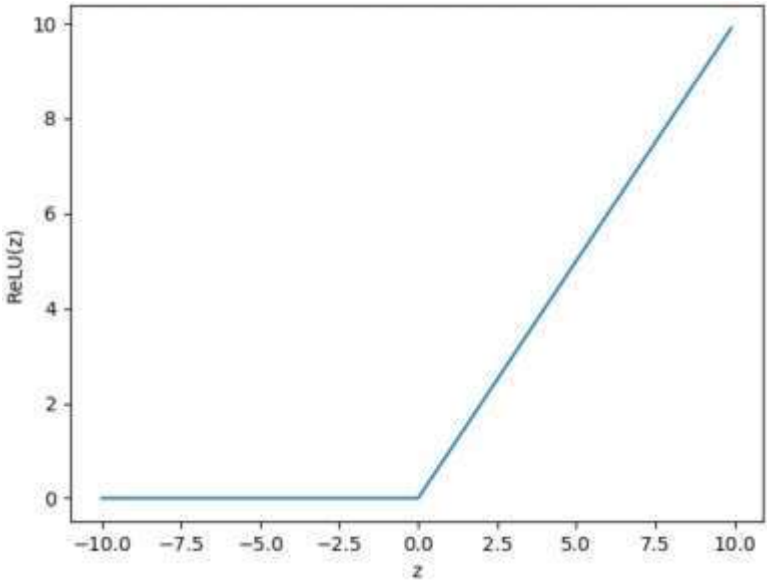


- Output range is the same as z_i
- Can be used as last layer to implement a regression model
- Differentiable everywhere

Commonly Used Activation Functions

Rectified Linear Unit (ReLU)

$$f(z_i) = \max\{0, z_i\}$$



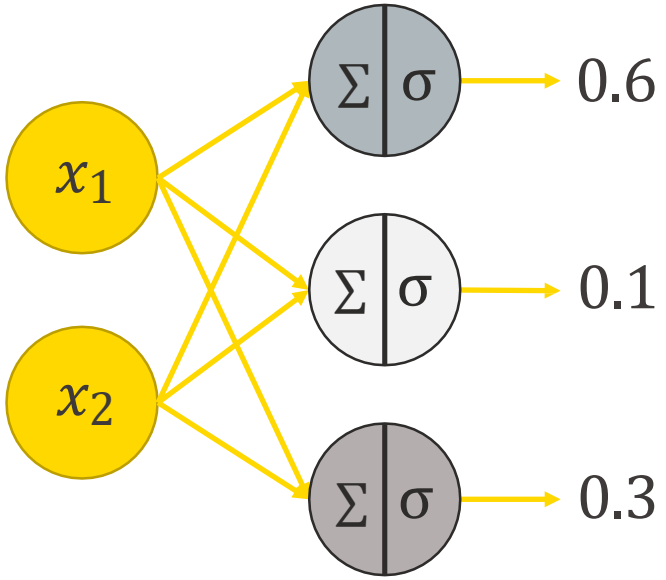
- Identity function for $z_i > 0$
- Appears linear, but it's not linear: $f(-a) + f(a) \neq f(-a + a) = f(0)$
- Able to overcome the vanishing gradient problem
- Not differentiable at $z_i = 0$ ← rare occurrence in practice

Commonly Used Activation Functions

Softmax Function

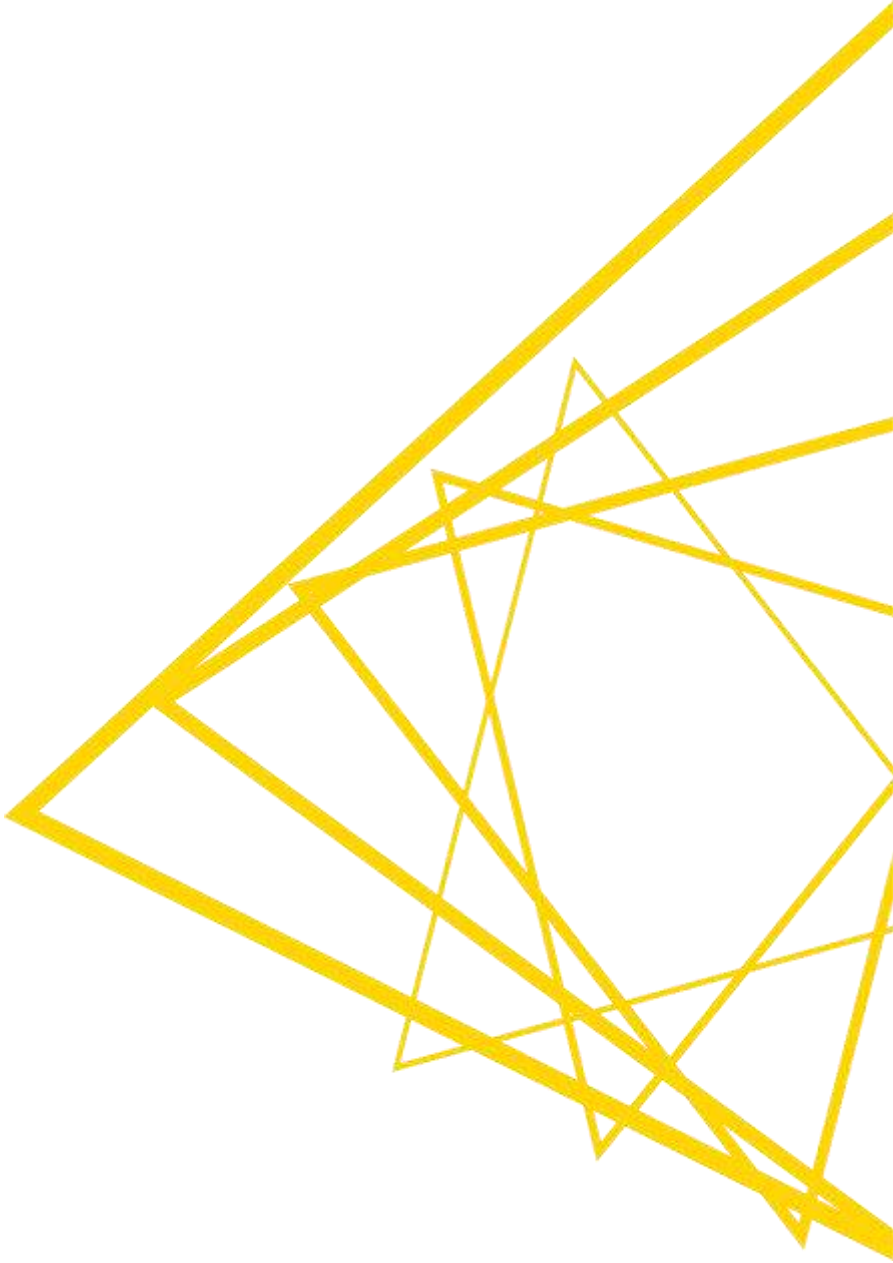
$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Sigmoid Layer with Three Outputs



- Applied to all outputs of a particular layer
- Used as last layer for multiclass classification
- Each output is the class probability (0,1) ➔ Sums up to 1

Loss Functions



Loss Functions

- Quantifies errors or deviations in the network outcome compared to the target
- *We want to minimize the loss!!*
- Different types of loss functions for classification and regression

- *Classification*: We want the predicted category to match the target
- *Regression*: We want to minimize deviation from the target

Loss Functions – Binary Classification

Binary cross entropy

- Target: $y_i = 1$ or $y_i = 0$, for $i = 1,2,\cdots,N$
- Predicted probability $p(y_i)$ for $y_i = 1$, and $1 - p(y_i)$ for $y_i = 0$
- The cross entropy CE is calculated as

$$CE = \frac{1}{N}\% \sum_{i=1}^{N} y_i \log\left(p(y_i)\right) + (1 - y_i) \log(1 - p(y_i))$$

- Suitable for the ***sigmoid*** activation function

Hinge function

- Same as binary CE, except that the two classes are coded by -1 and 1
- Suitable for the ***tanh*** activation function

Loss Functions – Multiclass Classification

Categorical cross entropy

- Target category is k $(k = 1,2,\cdots,C)$
- Target class is encoded by a binary indicator variable $y_{i,k}$
  - $y_{i,k} = 1$ if the i-th observation in category k
  - $y_{i,k} = 0$ otherwise
  - Also known as *one-hot encoding* – more details to come
- Predicted probability $p(y_{i,k})$ for category k
- The categorical cross entropy CE is calculated as

$$CE = \frac{1}{N}\% \sum_{i=1}^{N} \left(y_{i,1} \log\left(p\left(y_{i,1}\right)\right) + y_{i,2} \log\left(p\left(y_{i,2}\right)\right) + \cdots + y_{i,C} \log\left(p\left(y_{i,C}\right)\right)\right)$$

$$= \frac{1}{N}\% \sum_{i=1}^{N} \% \sum_{k=1}^{C} y_{i,k} \log\left(p\left(y_{i,k}\right)\right)$$

Loss Functions – Regression

Mean Squared Error (MSE) Loss

- Let $\hat{y}_i$ be the predicted outcome of the model for y_i $(i = 1,2,\cdots,N)$
- The MSE is calculated as

$$MSE(\boldsymbol{y},\hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Large deviations are given more importance
- Sensitive to outliers

Loss Functions – Regression

Mean Squared Logarithmic Error (MSLE) Loss

- Log-transformed target and predicted outcome
- The MSLE is calculated as

$$MSLE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (\log(y_i + 1) - \log(\hat{y}_i + 1))^2$$

  - Note: $+1$ to avoid $\log(0) = -\infty$
- Large deviations are penalized less than MSE
- Recommended when the range of the target values is large

Loss Functions – Regression

Mean Absolute Error (MAE) Loss

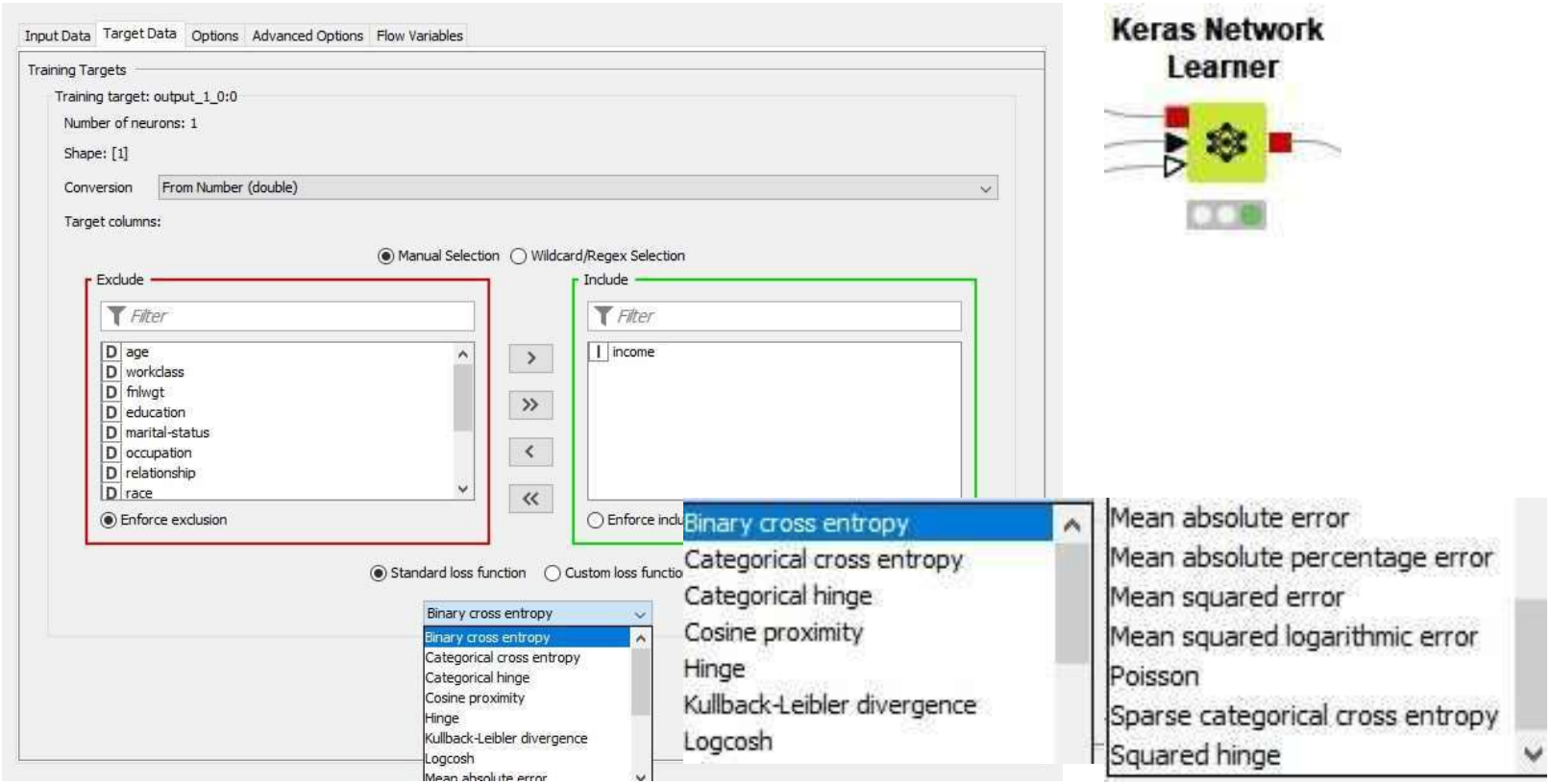
- The MAE is calculated as:

$$MAE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

- Large deviations are penalized less than MSE or MSLE
- Robust against outliers

Specifying a Loss Function

- You can specify a loss function in the Keras Network Learner node



Which Activation Functions? Which Loss Functions?

- Depends on the problem you are working on

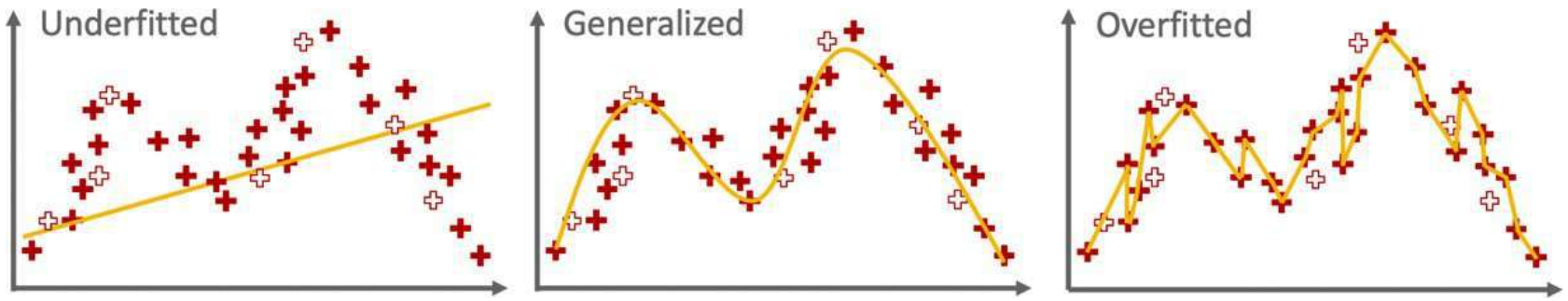
✓ Recommended
ΔCan be used

		Activation Functions								Loss Functions					
		Hidden Layers			Output Layer										
Problems		Sigmoid	Tanh	ReLU	Sigmoid	Tanh	Linear	ReLU	Softmax	Binary CE	Hinge	Categorical CE	MSE	MSLE	MAE
Classification	Binary classification (0 vs 1)	✓	✓	✓	✓					✓					
	Binary classification (-1 vs 1)	✓	✓	✓		✓					✓				
	Multi-class classification	✓	✓	✓					✓			✓			
Regression	Regression	✓	✓	✓	Δ	Δ	✓	Δ					✓		
	Regression (wide range)	✓	✓	✓			✓							✓	
	Regression (possible outliers)	✓	✓	✓			✓								✓

Overfitting and How to Avoid it

Overfitting

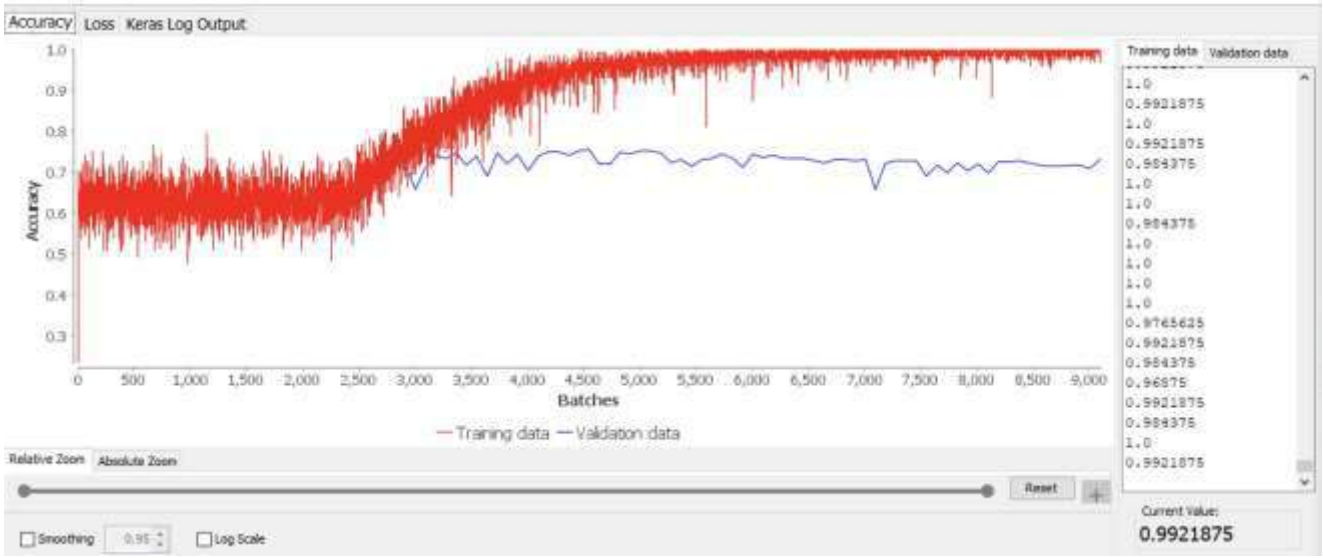
- Large neural network with many parameters trained on small dataset
- Able to fit the training set very well, but unable to generalize to other data
- Known as **overfitting**



Overfitting

One sign of overfitting

- Accuracy continues to improve on the training data but not the validation data



How can we avoid overfitting?

- Norm regularization, dropout, and early stopping

Avoiding Overfitting – Norm Regularization

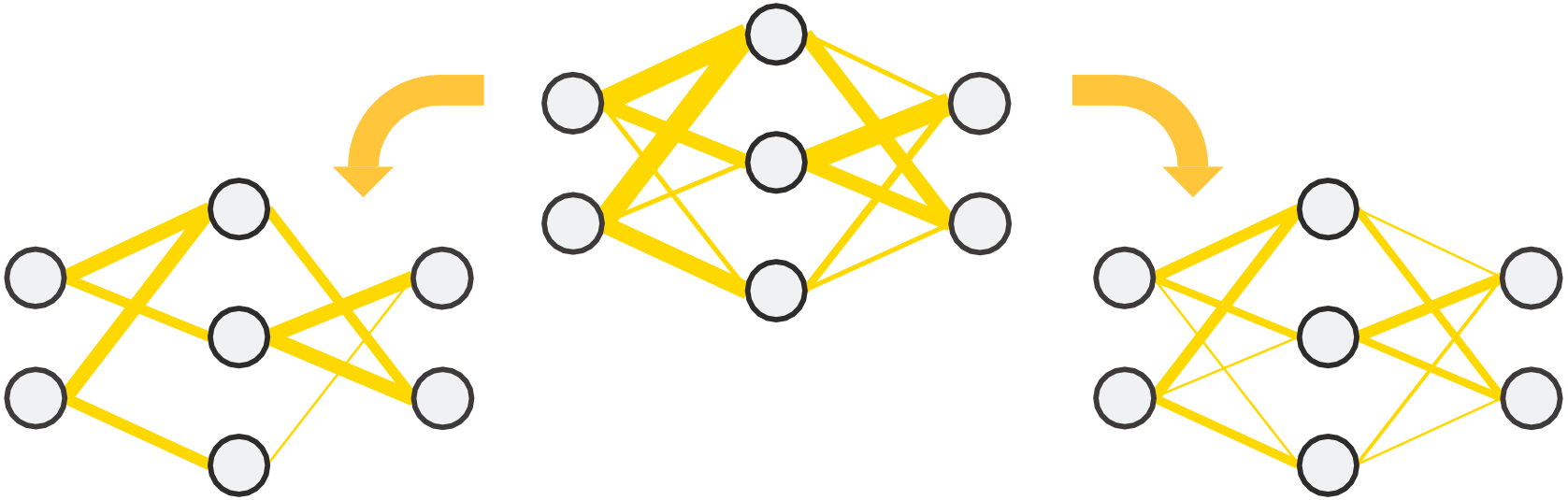
Norm Regularization

- One sign of overfitting – high values of weights $\boldsymbol{W}$
- Solution: penalty term in loss function to avoid large weights

$$\tilde{E}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = E(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \lambda\Omega(\boldsymbol{W})$$

- Parameter λ: controls the penalty effect – strong regularization with larger λ

Avoiding Overfitting – Norm Regularization



L1 norm – a.k.a., Laplace, LASSO

$$\Omega(\boldsymbol{W}) = L_1(\boldsymbol{W}) = \% \sum_{i=1}^{n} |\boldsymbol{W}_i|$$

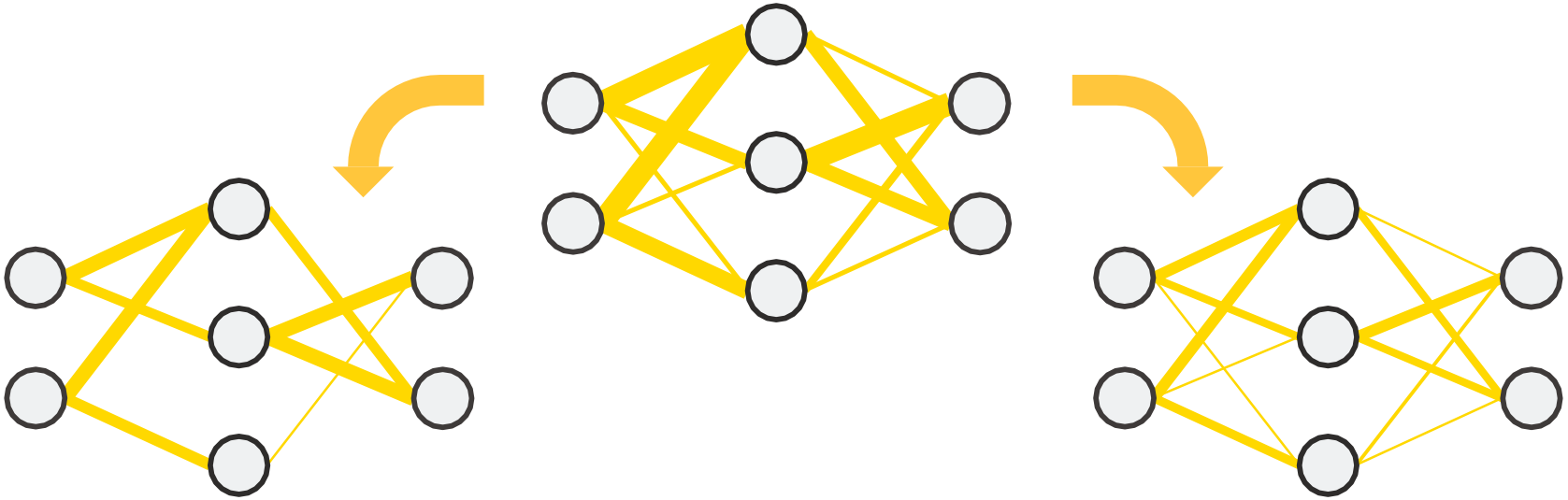
- Can reduce some weights to zero → simplifying the network
- Robust to outliers

L2 norm – a.k.a., Gauss, Ridge

$$\Omega(\boldsymbol{W}) = L_2(\boldsymbol{W}) = \% \sum_{i=1}^{n} \boldsymbol{W}_i^2$$

- All connections are included
- Better prediction
- Sensitive to outliers

Avoiding Overfitting – Norm Regularization



L1 & L2 norm – a.k.a., Elastic Net

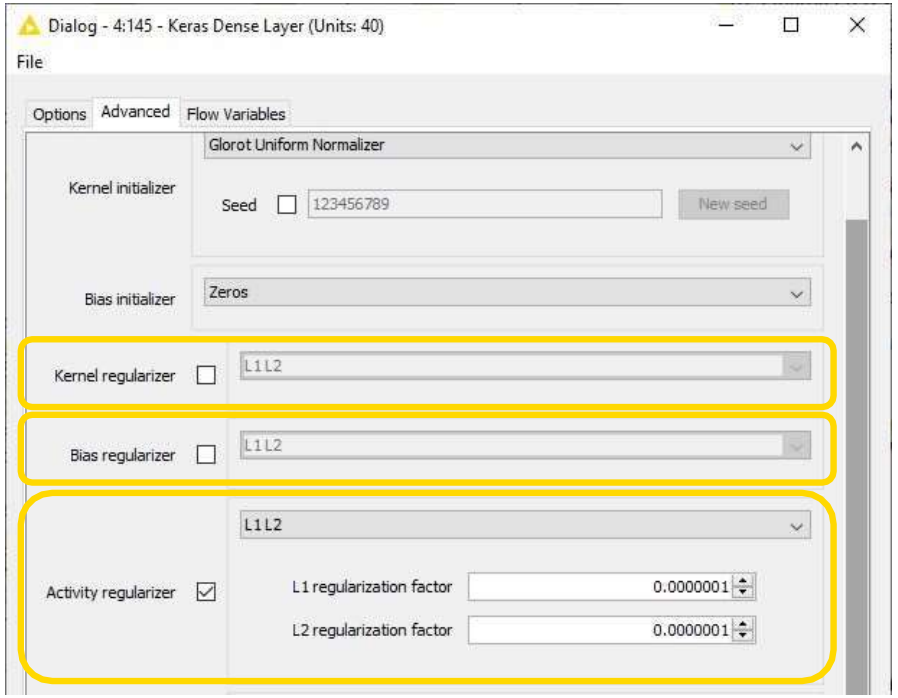
$$\Omega(\boldsymbol{W}) = \alpha L_1(\boldsymbol{W}) + \beta L_2(\boldsymbol{W})$$

- Both L_1 and L_2 regularization terms in the loss function
- Combining the strengths of both

Avoiding Overfitting – Norm Regularization

- Regularization can be specified at each layer
  – L1 only, L2 only, or L1+L2

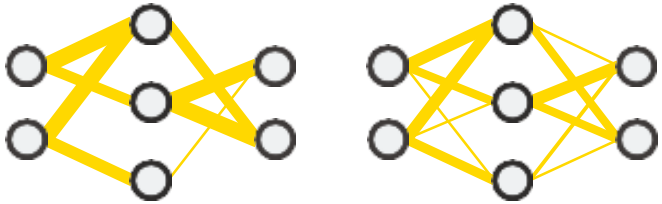
- **Kernel regularizer**
- **Bias regularizer**
- **Activity regularizer**



Avoiding Overfitting – Norm Regularization

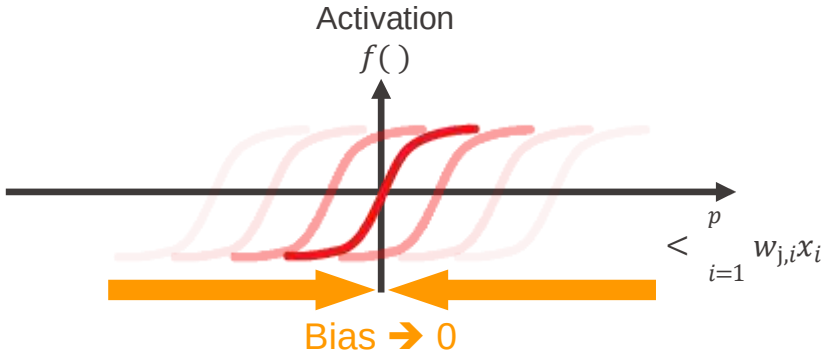
Kernel regularizer

- Regularizes weights, but not biases



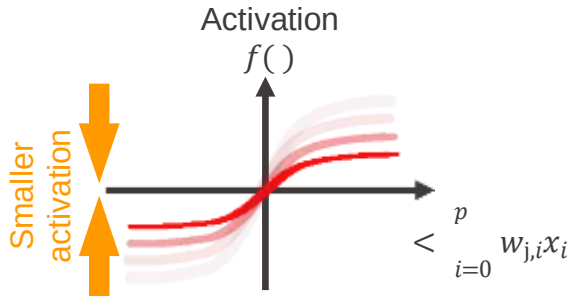
Bias regularizer

- Regularizes biases, but not weights



Activity regularizer

- Leads to smaller activation

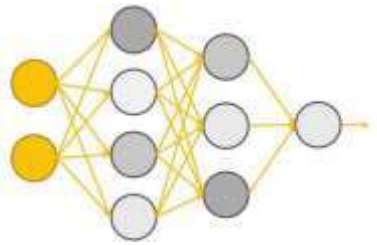


Avoiding Overfitting – Dropout

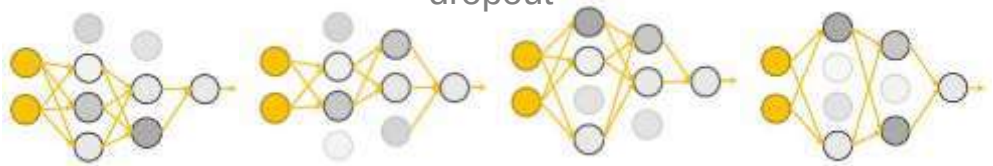
Dropout

- Randomly drop neurons in input or hidden layer – introducing randomness
- So that individual neurons do not rely on the input from a single neuron
- Different neurons are dropped in each iteration

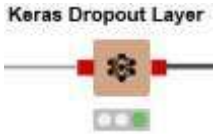
Standard network



Networks in different iterations after applying dropout



- Keras Dropout Layer node – drops neurons randomly in the previous layer

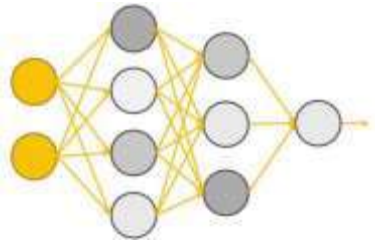


Avoiding Overfitting – Dropout

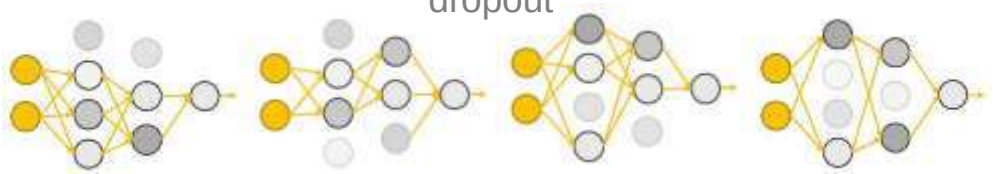
Dropout rate

- Proportion of nodes to be dropped
- Typically, 20-50%
- Too low → does not improve overfitting
- Too high → network no longer able to predict

Standard network

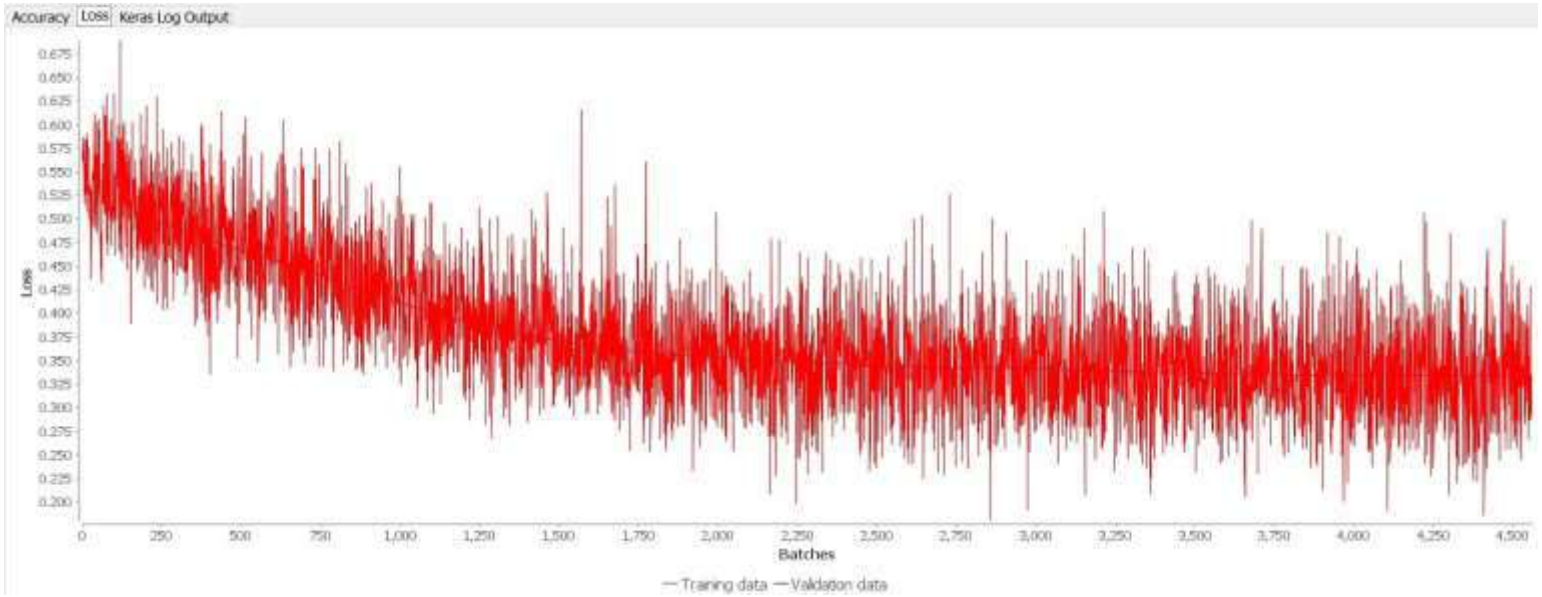


Networks in different iterations after applying dropout



Avoiding Overfitting – Early Stopping

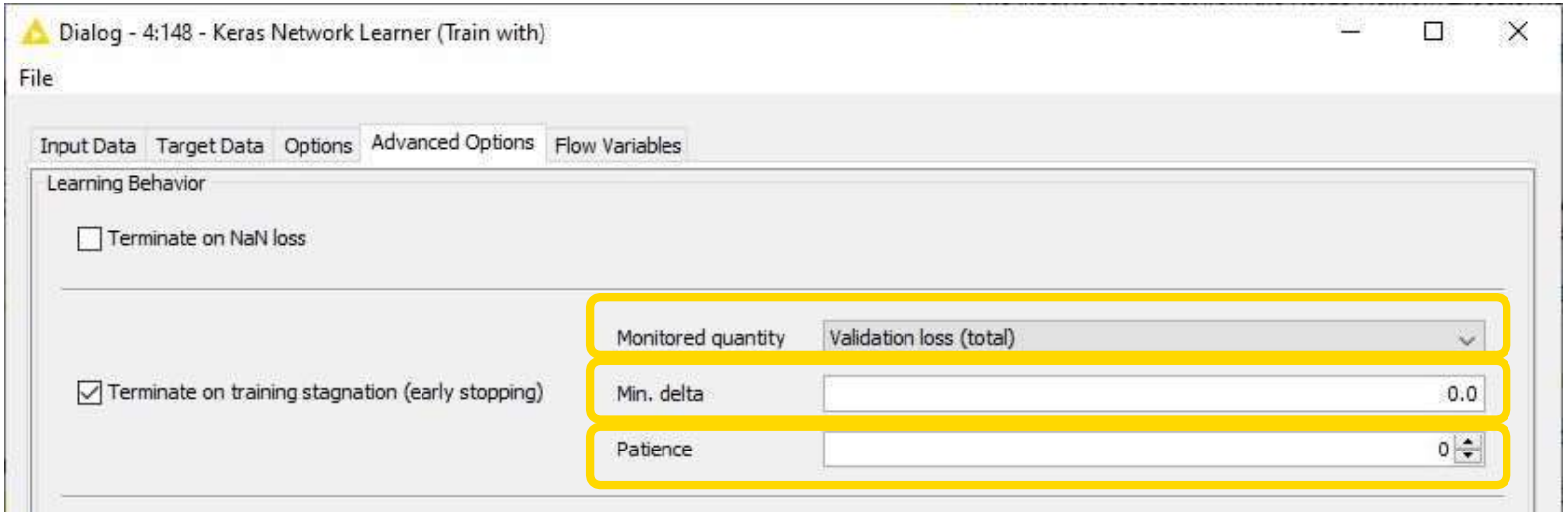
- The loss function may plateau



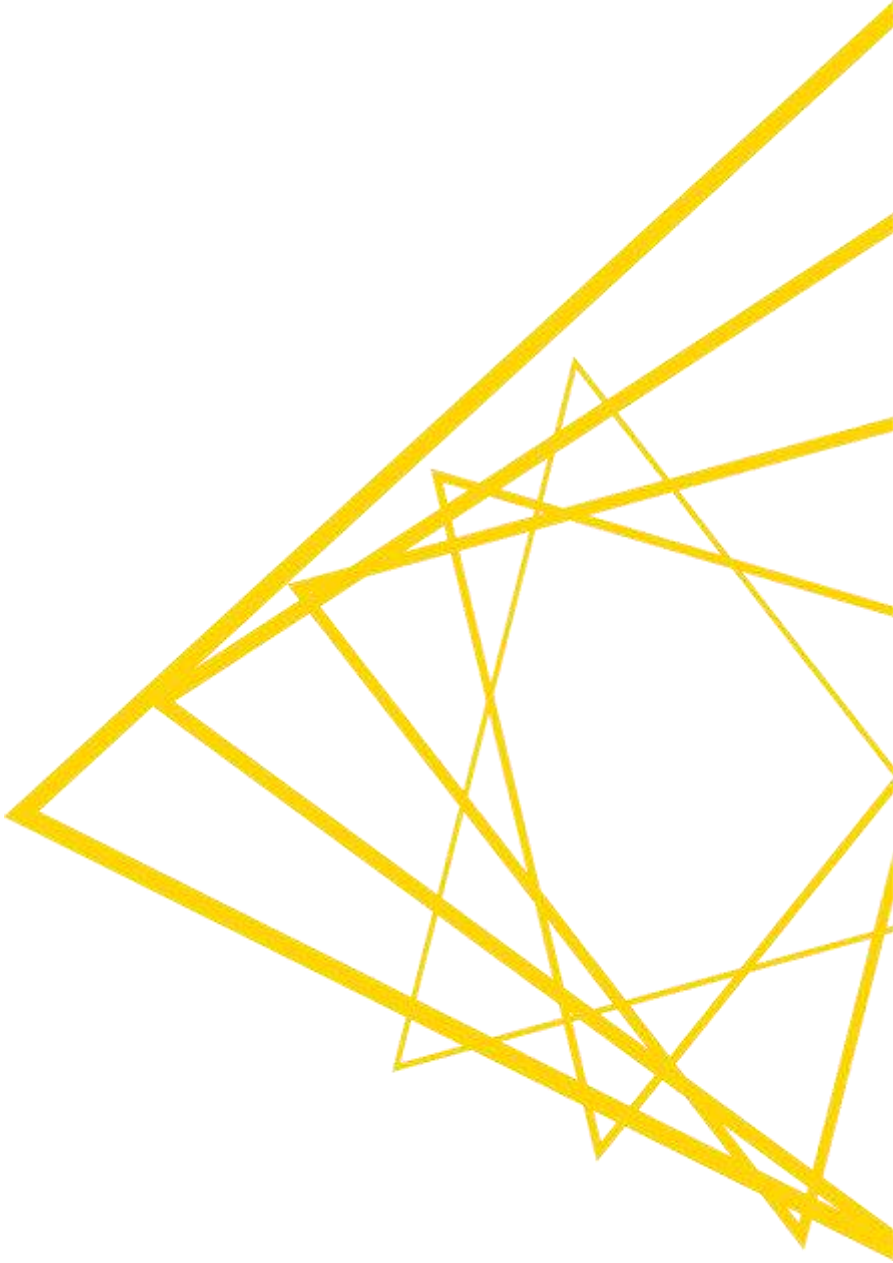
- Stops training the model when the loss function stops improving

Avoiding Overfitting – Early Stopping

- Either in the training data or the validation data
- If the improvement on the loss is below the prespecified value, AND
- Does not improve for a specified number of epochs

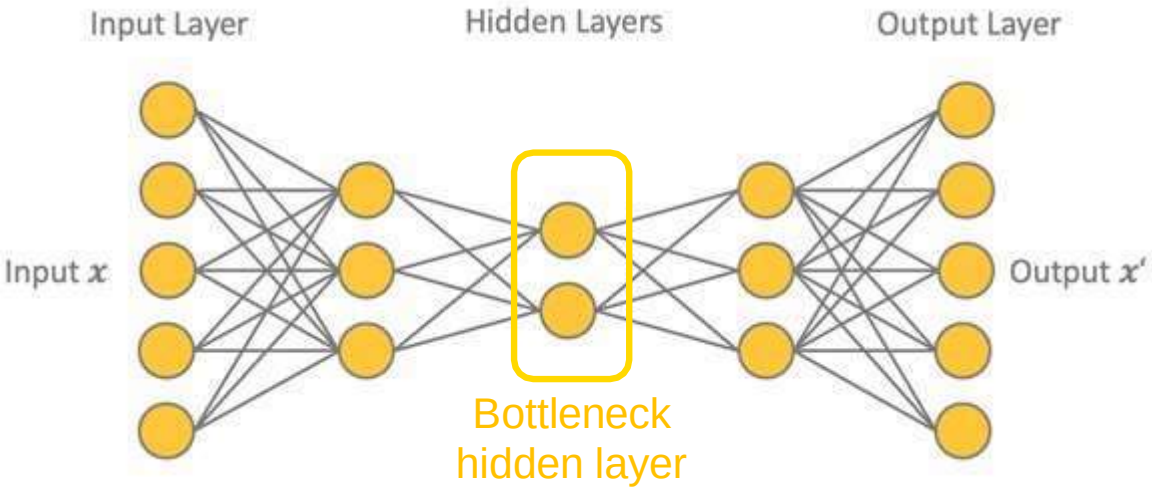


Autoencoders



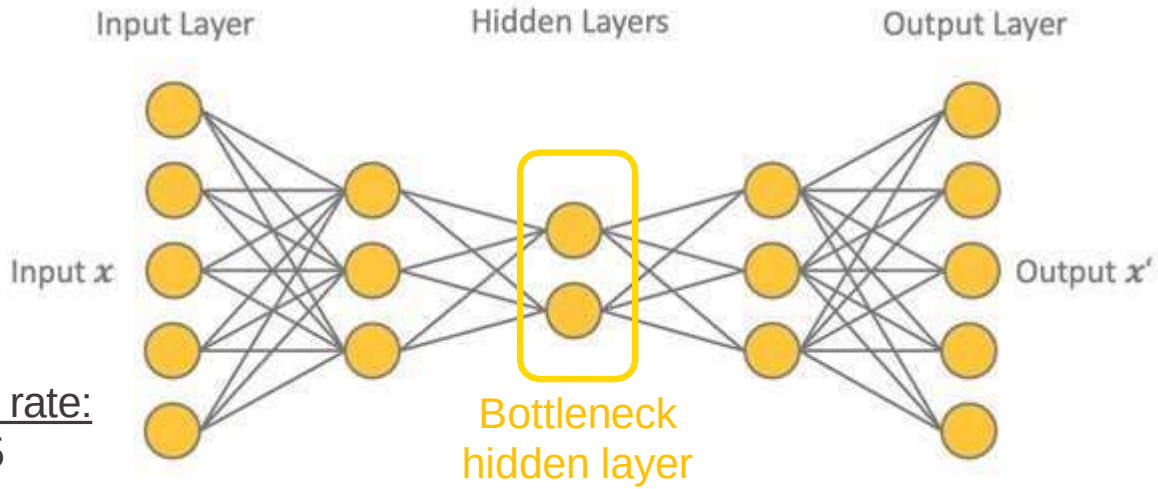
What Is an Autoencoder?

- Multilayer feedforward network
- Reproduces the input at the output layer
- MSE loss function – output as close to the input as possible
- Bottleneck hidden layer – fewer neurons than the input or output layer



What Is an Autoencoder?

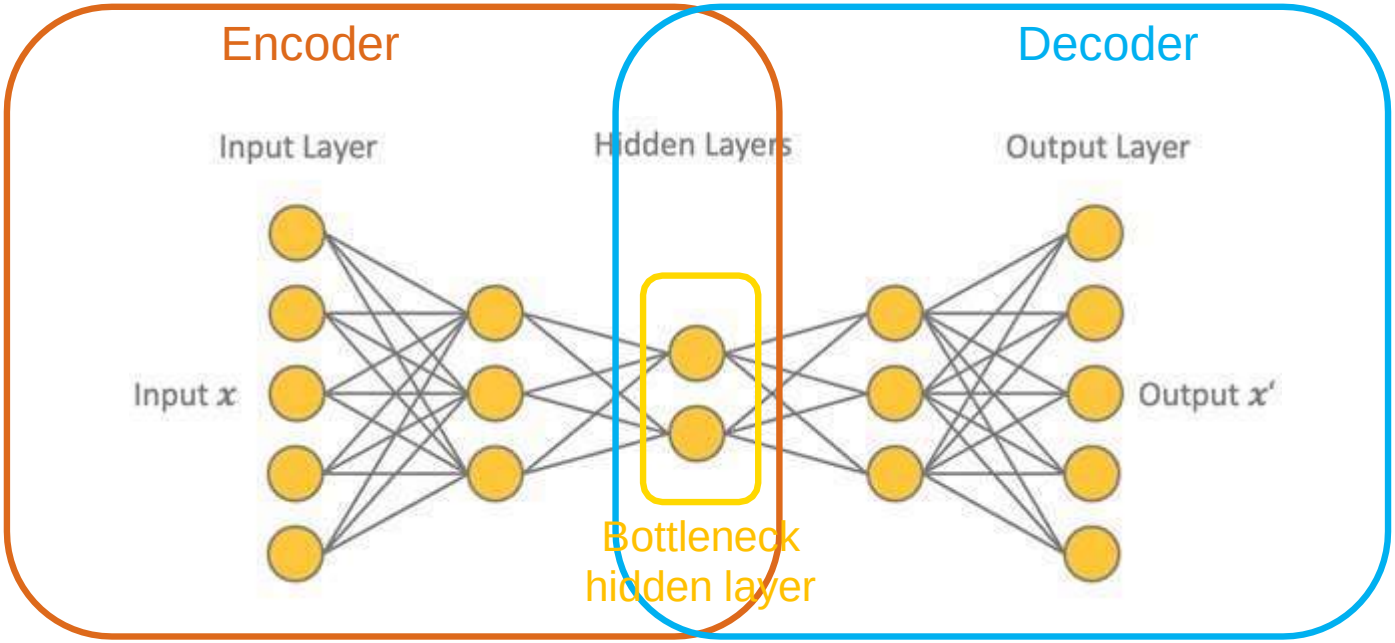
- Can be used for data compression
- Input & output n-dimension
- Bottleneck hidden layer h-dimension
- Compression rate n/h



Compression rate:
5/2 = 2.5

What Is an Autoencoder?

- Consists of two parts
- Encoder: Compresses the input to smaller dimension
- Decoder: Reconstitutes the input from the signal at the bottleneck hidden layer



Anomaly Detection with Autoencoder

- Anomaly: rare and unexpected event
- Examples: cardiac arrhythmia, mechanical breakdown, fraudulent transaction
- Too few occurrences ➔ not enough data to train a standard classifier
- Possible solutions: clustering with DBSCAN, isolation forest technique

Autoencoder is a viable solution

- Train with “normal” samples only → able to reproduce normal data
- Anomalies – the network has not seen such observations
- Unable to reproduce the input → larger MSE compared to normal

Anomaly Detection with Autoencoder

- Say ε_k is the MSE between the input $\boldsymbol{x}_k$ and the reproduced output $\boldsymbol{x'}_k$

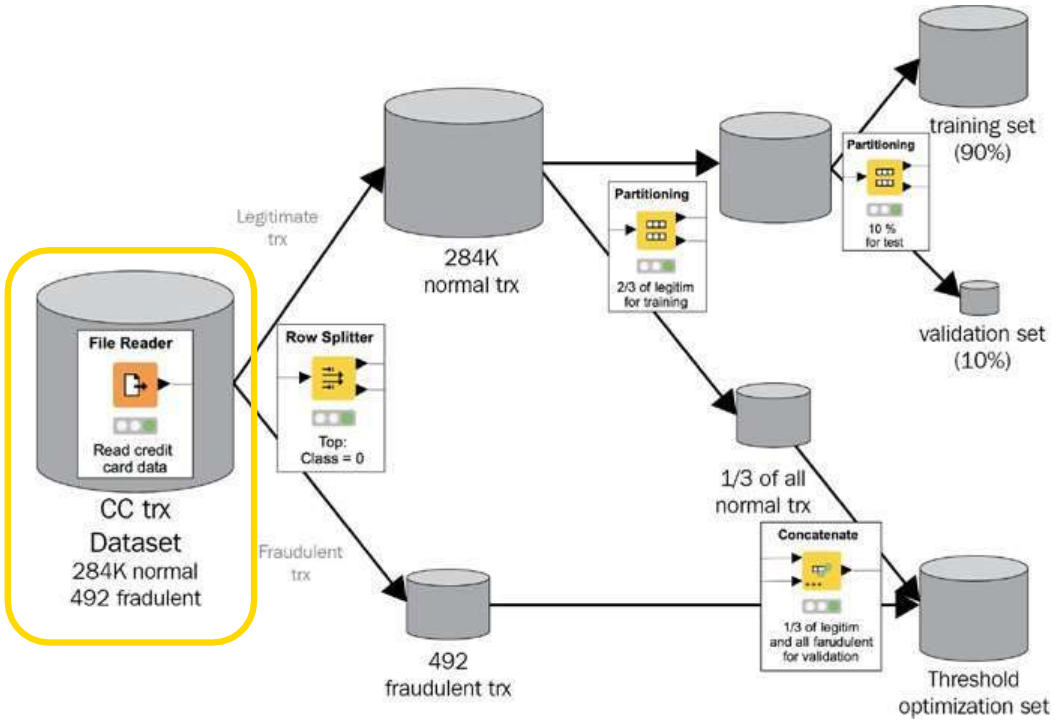
  If $\varepsilon_k < K$, then $\boldsymbol{x}_k$ is “normal”

  If $\varepsilon_k \geq K$, then $\boldsymbol{x}_k$ is “anomaly”

- With a certain threshold K

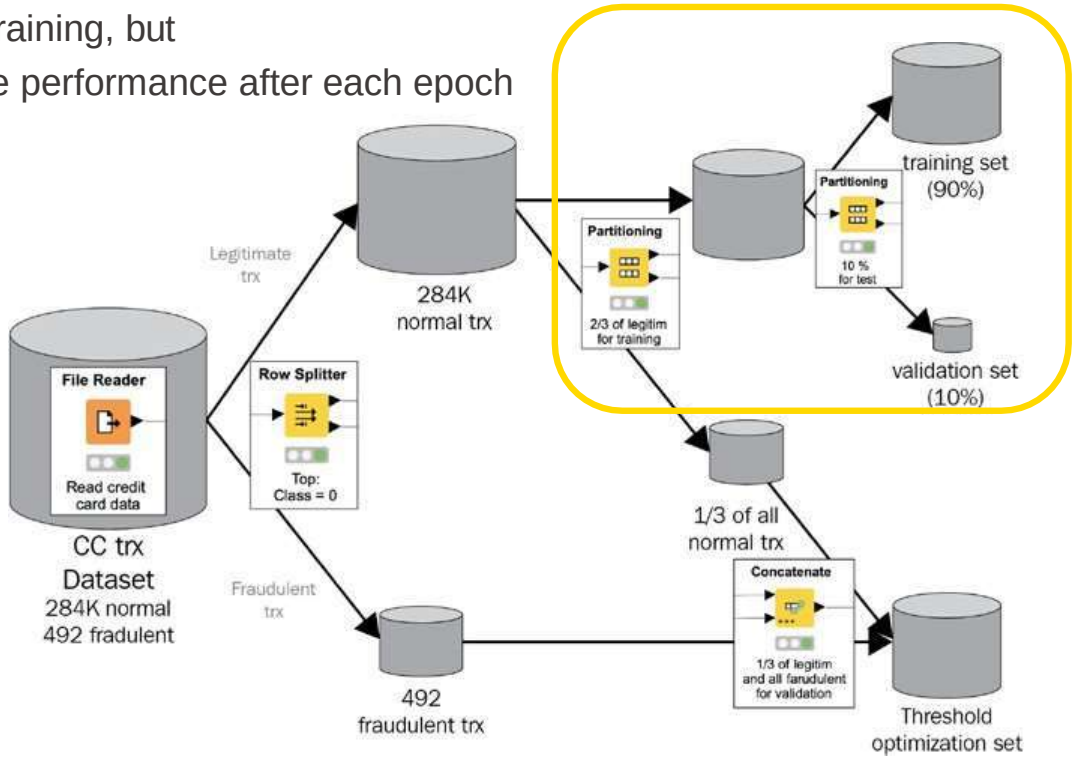
Data for Anomaly Detection with Autoencoder

- Many normal samples
- As many anomaly samples as possible – very few compared to normal samples



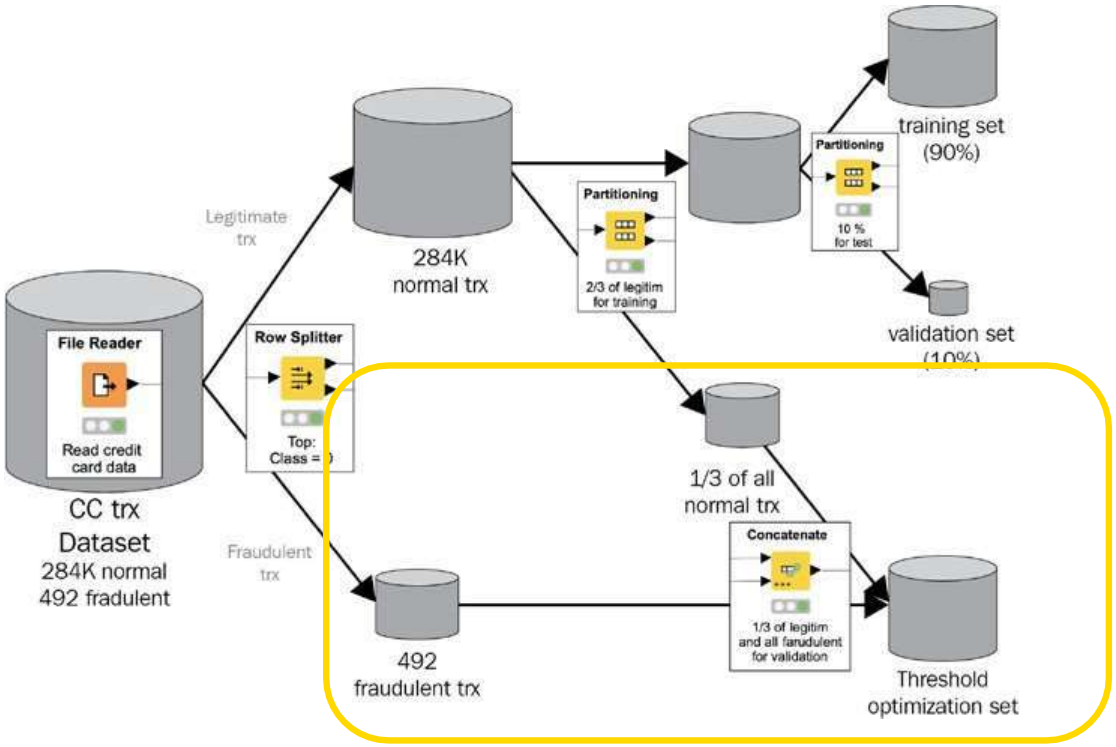
Data for Anomaly Detection with Autoencoder

- A large portion of normal observations are used to train an autoencoder
- With a **validation set**
  - Unused during training, but
  - Used to evaluate performance after each epoch



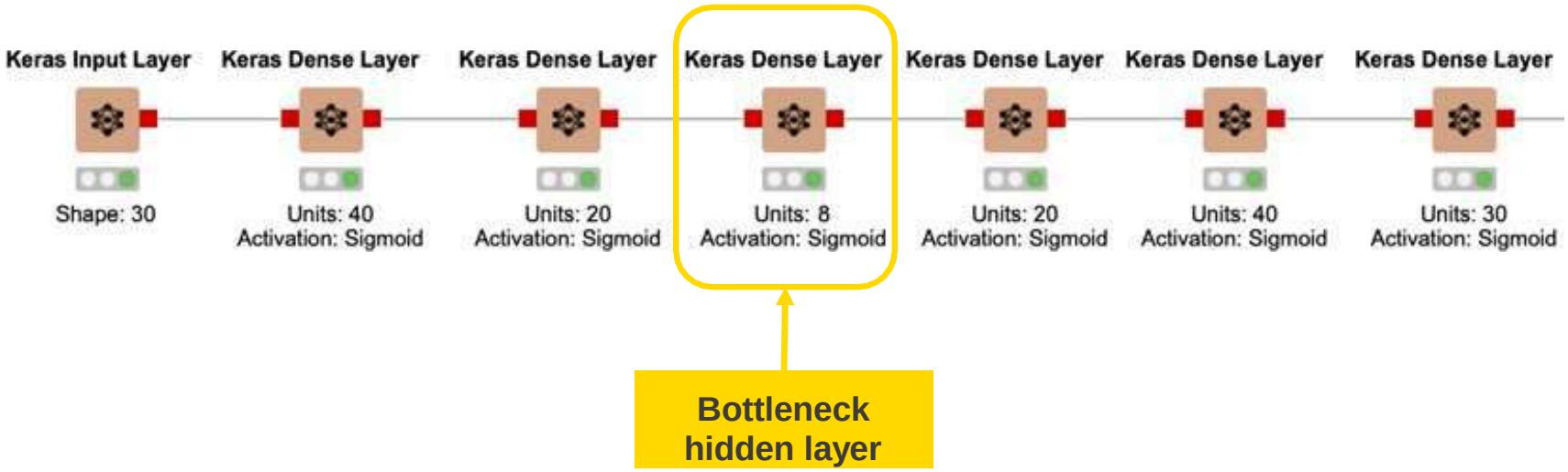
Data for Anomaly Detection with Autoencoder

- The remaining normal samples are mixed with all available anomalies
- Used to optimize the threshold K for anomaly detection



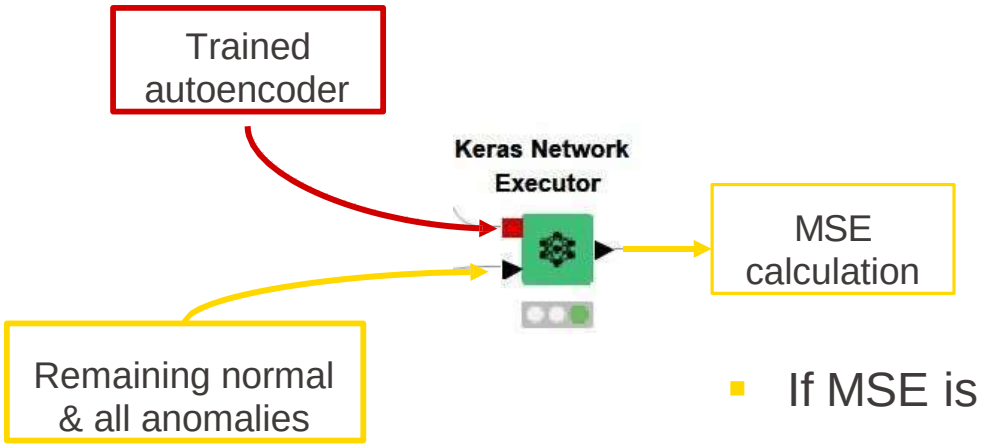
Training the Autoencoder

- Input and output features are the same
- Feedforward network with the bottleneck hidden layer in the middle
- MSE loss function – to minimize the error between the input and reproduced input



Normal and Anomaly Samples on the Autoencoder

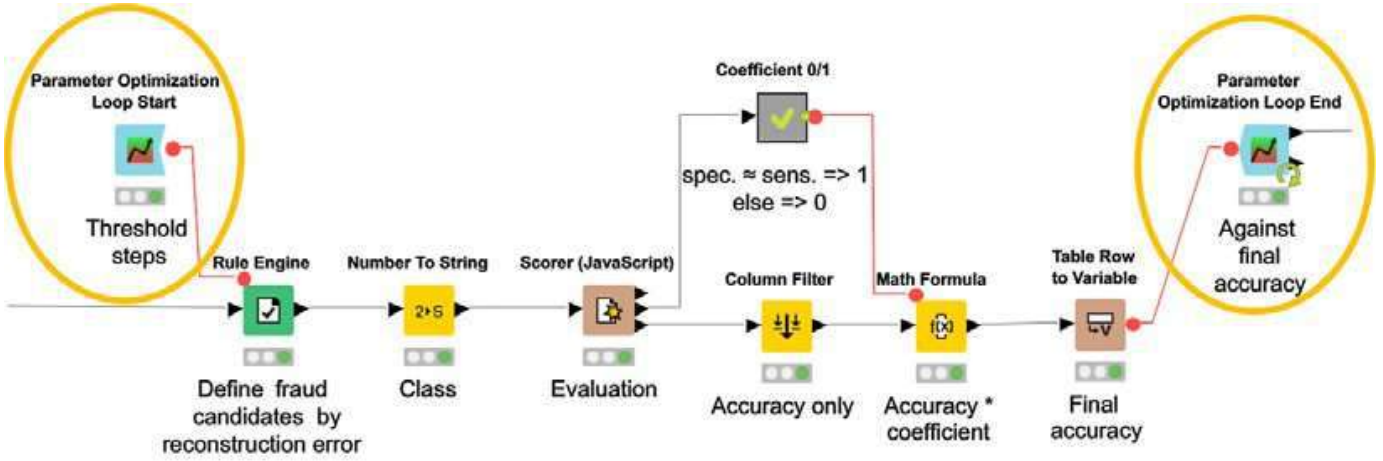
- The optimization set with normal and anomaly samples
- Applied to the trained autoencoder
- Calculate the MSE loss function



- If MSE is larger than K → declared “anomaly”
- How do we find K?

Optimizing the Threshold K

- Try different values of K – Parameter Optimization Loop
- For each K, classify samples as normal or anomaly ($\varepsilon_k < K$ or $\varepsilon_k \geq K$)
- Find the optimal K that maximizes the overall accuracy



- Additional constraint – sensitivity and specificity are similar to each other

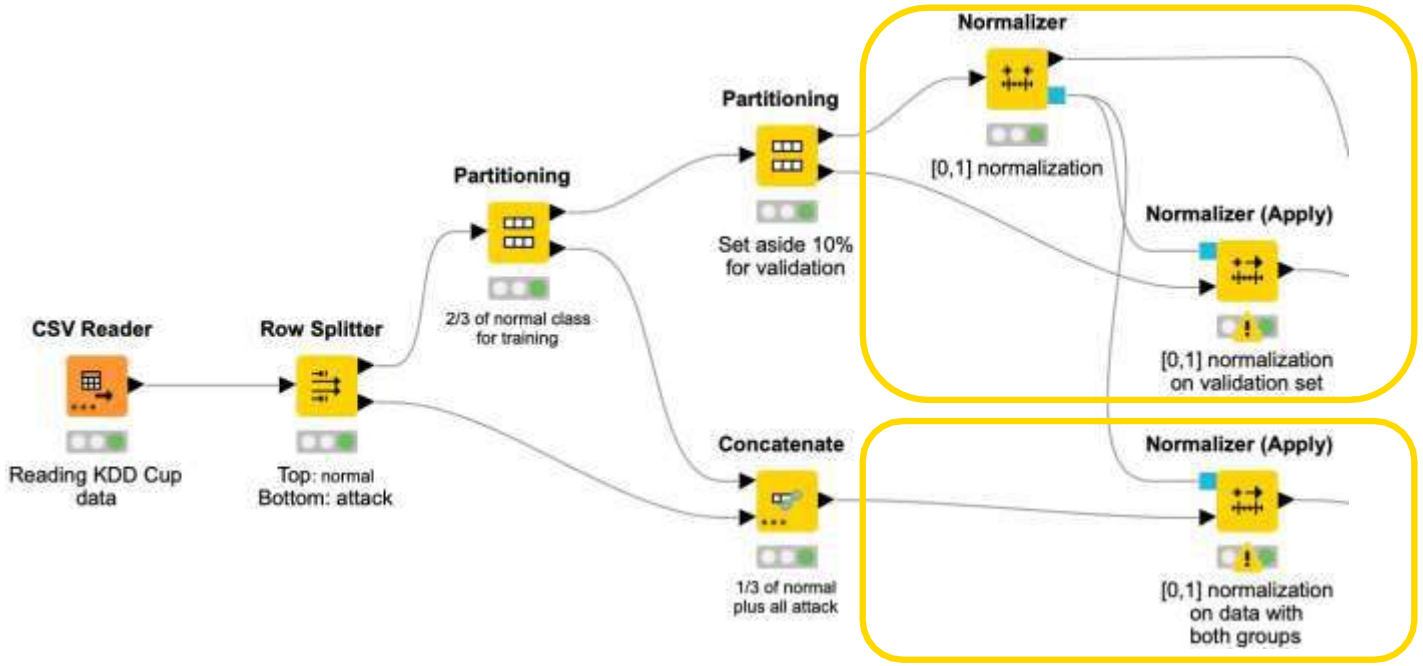
Example – Anomaly Detection

Data

- KDD Cup 1999 data
- Determining whether computer network connections are considered normal or attacks (unauthorized intrusions)
- Attacks are very rare events → anomaly detection with autoencoder
- Http connection data with 3 continuous features and the target (data available from http://odds.cs.stonybrook.edu/http-kddcup99-dataset/)
- There are 565287 normal and 2211 attacks in the data set

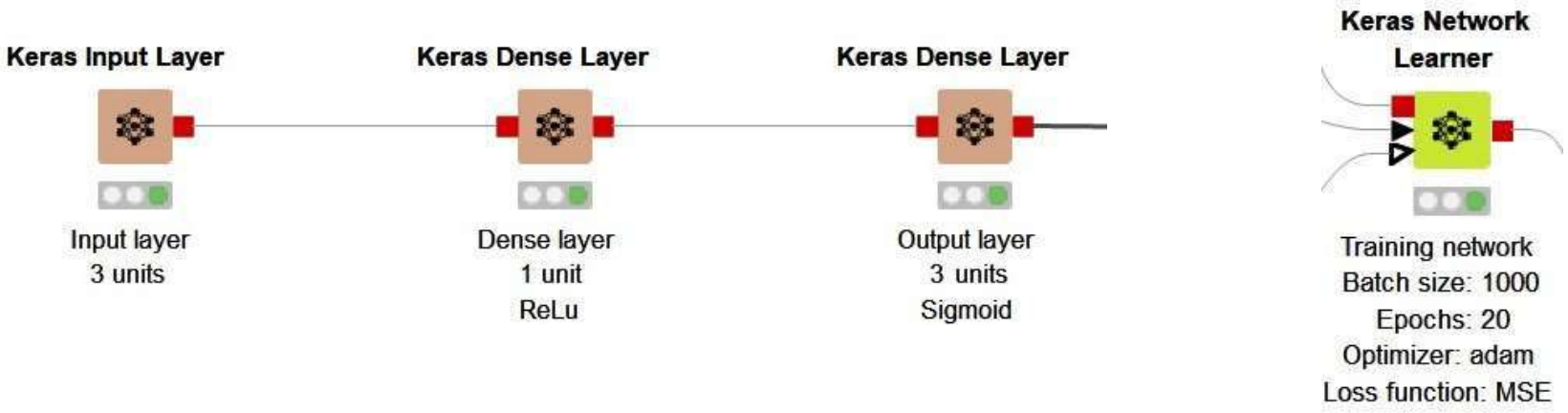
Data Preparation

- 2/3 of all normal samples (of which 10% validation) → train autoencoder
- 1/3 of all normal samples + all attacks → optimize the threshold K



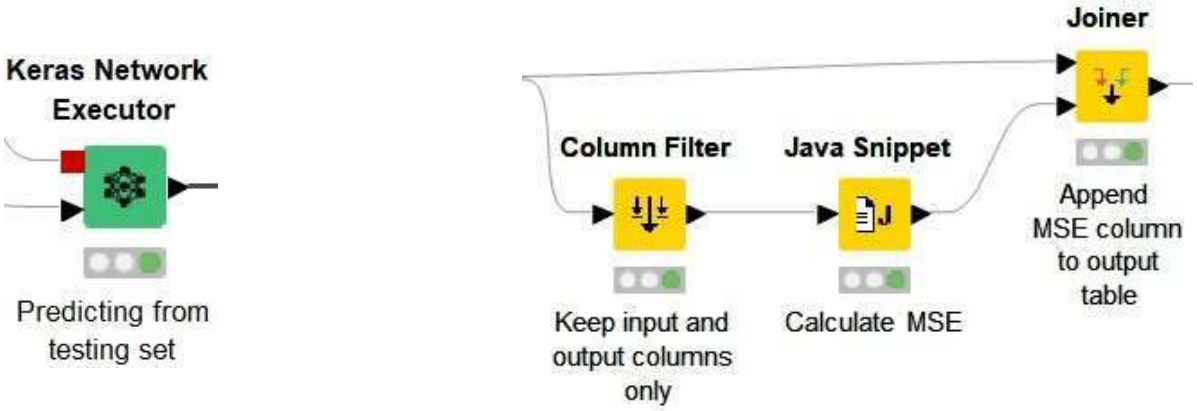
Autoencoder

- 3-1-3 autoencoder
- Keras Network Learner with 20 epochs, batch size 1000, Adam optimizer
- Loss function: MSE



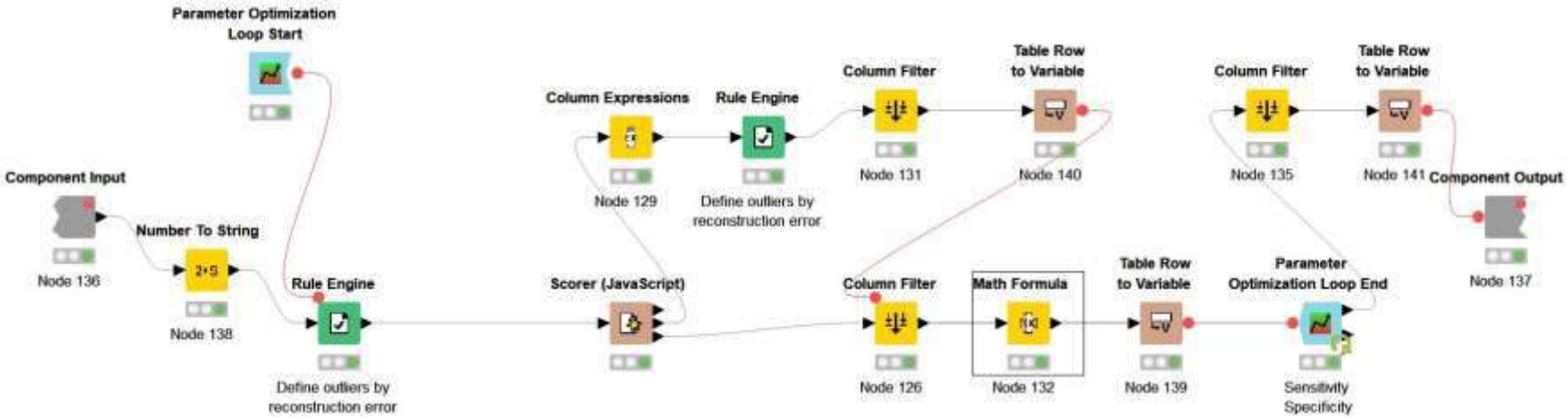
Applying Testing Data

- Testing set with normal and anomaly samples applied to the autoencoder
- MSE calculated (Java Snippet for faster calculation)



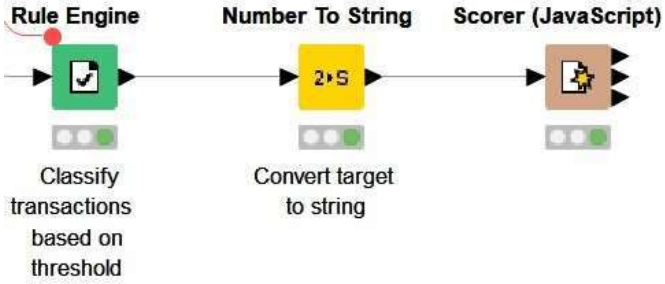
Finding the Optimal Threshold K

- Component for finding the optimal threshold K
- Values between 0.01 and 0.40 were examined



Evaluating the Performance with the Optimal K

- The optimal threshold K is applied to the testing set to examine the anomaly detection performance



Scorer View

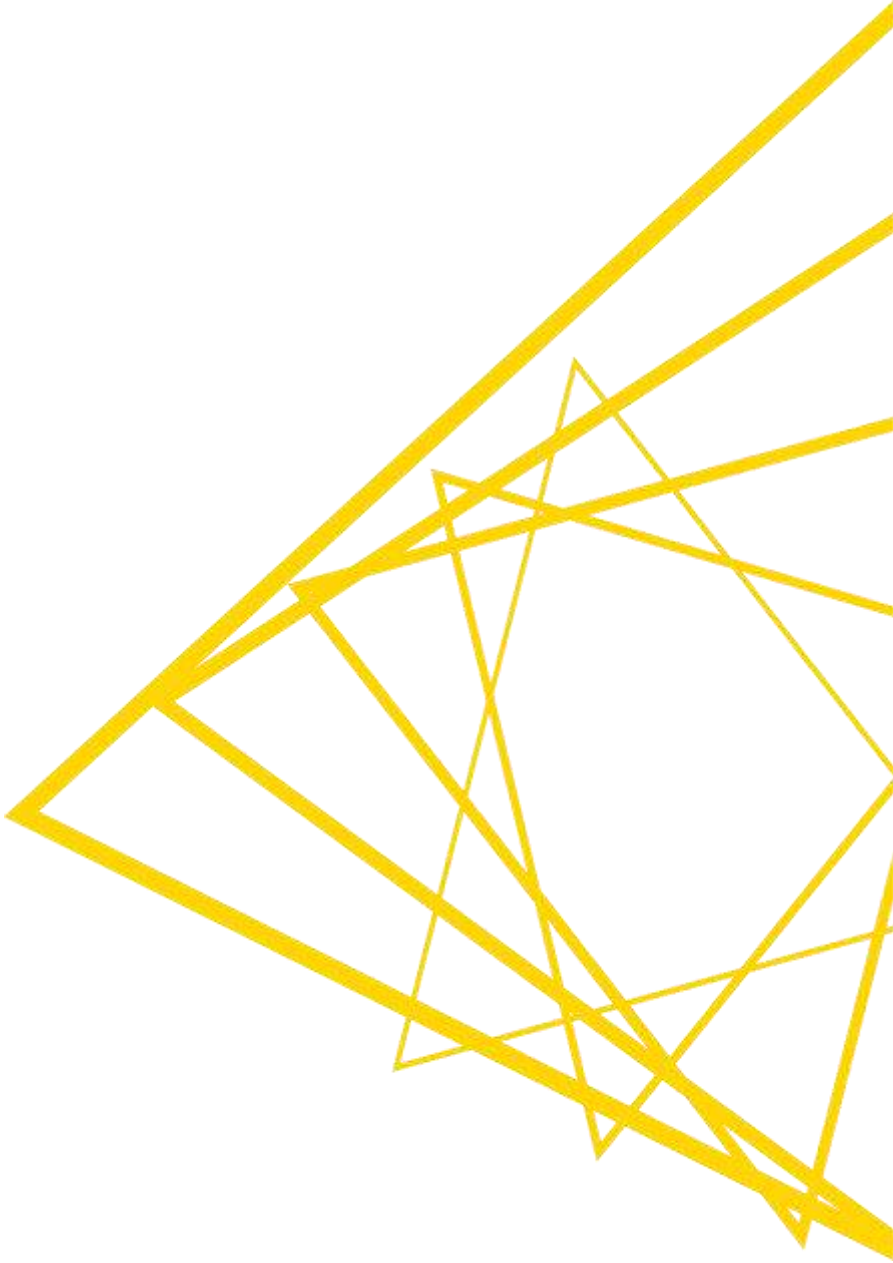
Confusion Matrix

	0 (Predicted)	1 (Predicted)	
0 (Actual)	188237	4	> 99.99%
1 (Actual)	4	2207	99.82%
	> 99.99%	99.82%	

Class Statistics

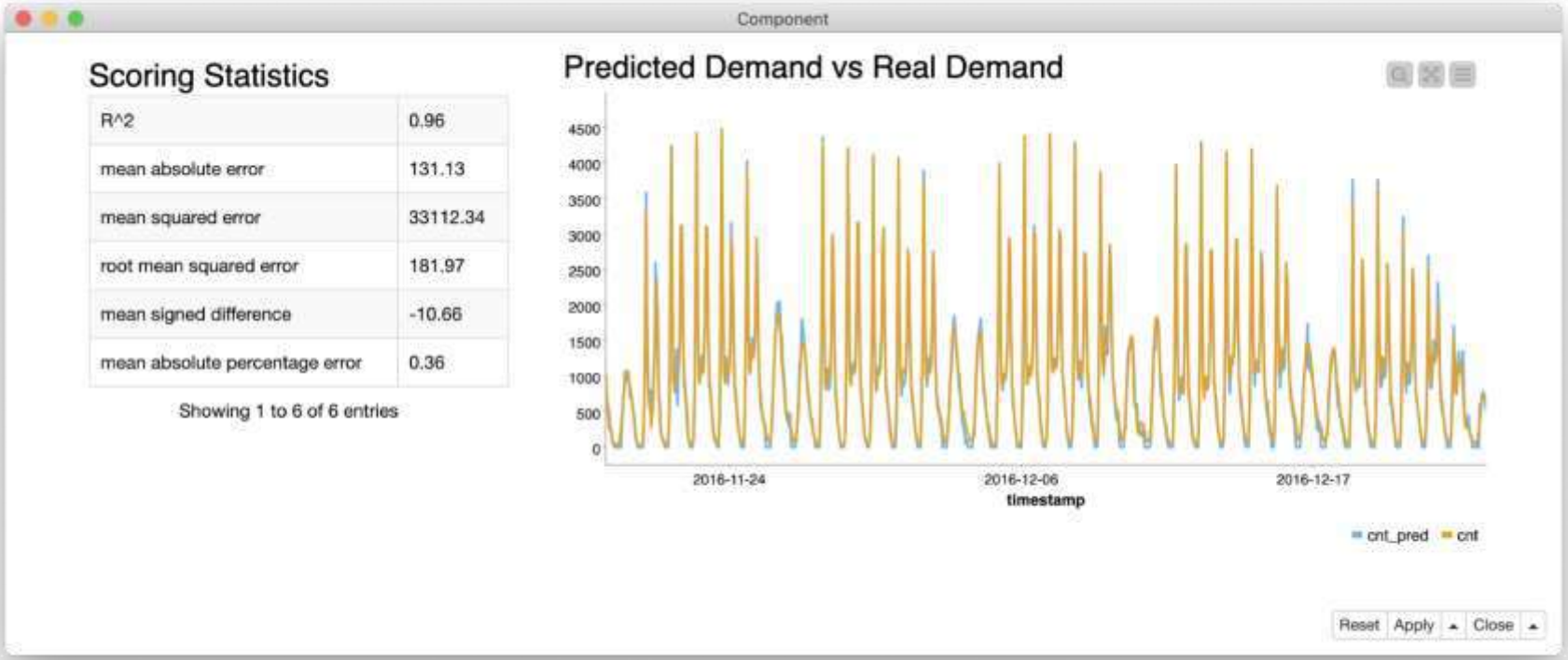
Class	True Positives	False Positives	True Negatives	False Negatives	Recall	Precision	Sensitivity	Specificity
0	188237	4	2207	4	> 99.99%	> 99.99%	> 99.99%	99.82%
1	2207	4	188237	4	99.82%	99.82%	99.82%	> 99.99%

Deep Learning Use Cases



Demand Prediction for London Bike Sharing

- Recurrent Neural Network (RNN) with Long Short Term Memories (LSTM)



https://www.knime.com/blog/multivariate-time-series-analysis-lstm-codeless

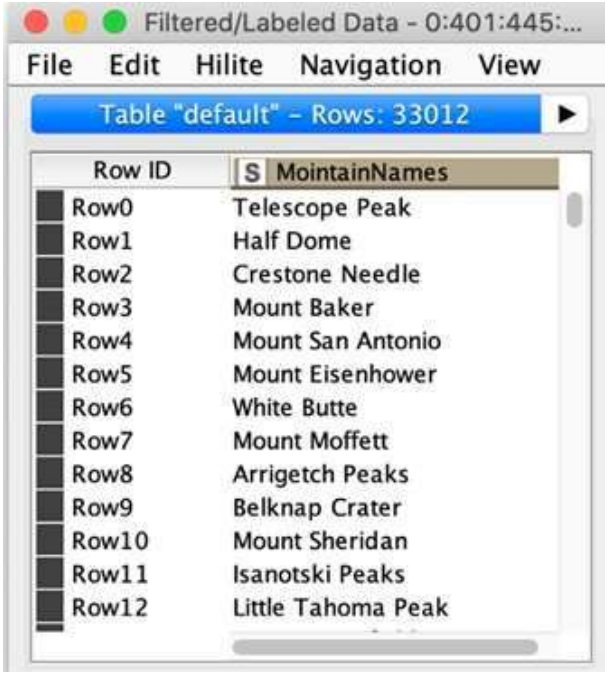
Neural Machine Translation

English	German	Translation
You run.	Du läufst.	Sie laufen.
You won.	Du hast gewonnen.	Sie gens!
She blushed.	Sie errötete.	Sie lief rot an.
Sit with me.	Setz dich zu mir!	Wirgen Sie mir!
Start again.	Fang noch einmal an.	Beginnen Sie noch einmal.
Stop trying.	Probieren Sie es nicht länger.	Probier es nicht länger.
Take a walk.	Geh spazieren!	Geht spazieren!
Talk to Tom.	Sprich mit Tom!	Rune mir es Tom!
That's cool.	Das ist cool.	Das ist geil.
Be brave.	Sei tapfer!	Seid tapfer!

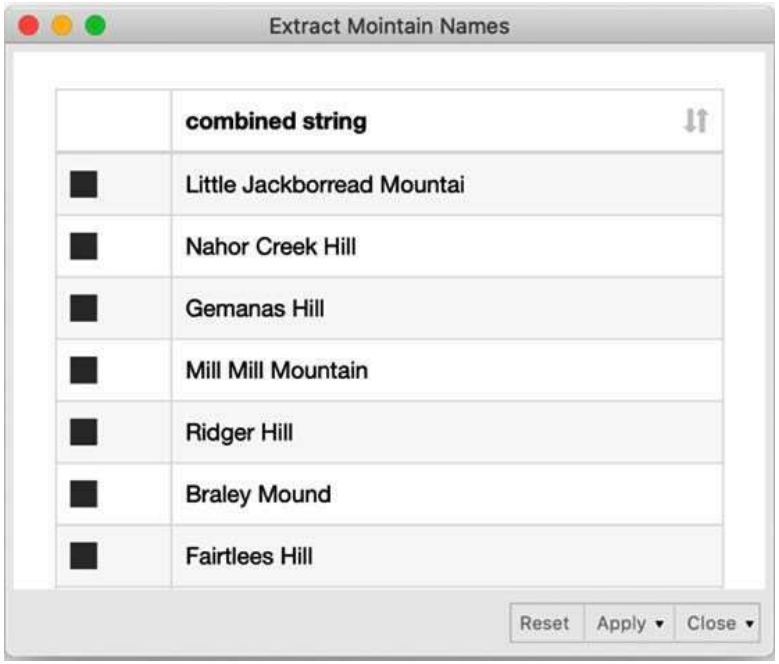
Codeless Deep Learning with KNIME, Chapter 8

Product Name Generation with RNNs

Training set
(US mountain names)



Generated mountain names



Codeless Deep Learning with KNIME, Chapter 7

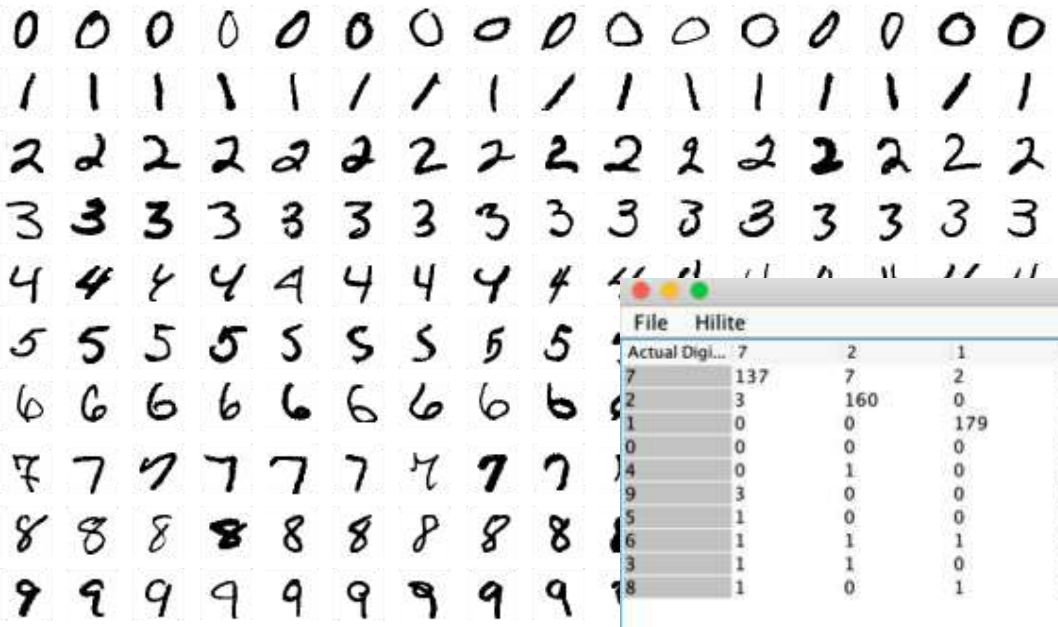
Image Classification

Kaggle Dogs vs Cats Challenge
https://www.kaggle.com/c/dogs-vs-cats/overview

Classification with KNIME
https://kni.me/w/ORyjDq1Xx25DSpFm

Image Classification

MNIST handwritten digits data



Confusion Matrix - 0:101 - Scorer

File Hilite

Actual Digi...	7	2	1	0	4	9	5	6	3	8
7	137	7	2	0	0	3	0	0	1	0
2	3	160	0	1	0	1	0	1	0	6
1	0	0	179	0	0	0	0	0	0	0
0	0	0	0	125	0	0	0	1	0	0
4	0	1	0	0	157	6	0	1	0	2
9	3	0	0	1	1	135	0	0	0	4
5	1	0	0	1	1	1	124	2	2	5
6	1	1	1	3	0	0	2	120	0	3
3	1	1	0	0	0	1	4	1	143	3
8	1	0	1	0	3	1	0	0	3	131

Correct classified: 1,411 Wrong classified: 89

Accuracy: 94.067 % Error: 5.933 %

Cohen's kappa (κ) 0.934

Codeless Deep Learning with KNIME, Chapter 9

Semantic Segmentation

https://kni.me/w/kRZQcv4e7-qoGQkU

Image Captioning

Original caption:
"large clock hanging from wall"

Generated caption:
"clock mounted wall"

Original caption:
"woman holding pizza her hand the middle kitchen"

Generated caption:
"woman sitting table with plate food"

Original caption:
"young child sitting chair next stove top oven"

Generated caption:
"kitchen with stove and refrigerator"

Original caption:
"woman watching baby takes bite cake"

Generated caption:
"woman eating piece cake with fork"

https://kni.me/w/hH01-0kMQfBLrAq4

Neural Style Transfer

Fig. 3a. Misha's input photo

+

Fig. 3b. "Portrait of Dora Maar" by Picasso (Image from WikiArt)

=

Fig. 3c. Misha's photo Picasso-style

Fig. 4a. Rosaria's input photo

+

Fig. 4b. "Medusa" by Caravaggio (Image from Wikipedia)

=

Fig. 4c. Rosaria's photo Caravaggio-style

https://www.knime.com/blog/deploying-the-obscure-python-script-neuro-styling-of-portrait-pictures

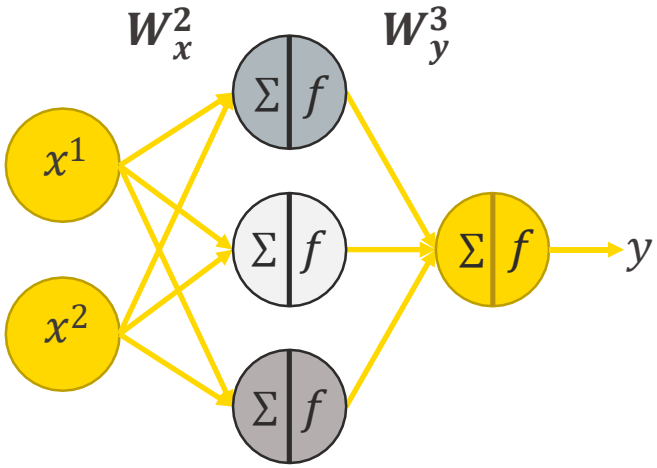
Recurrent Neural Network (RNN)

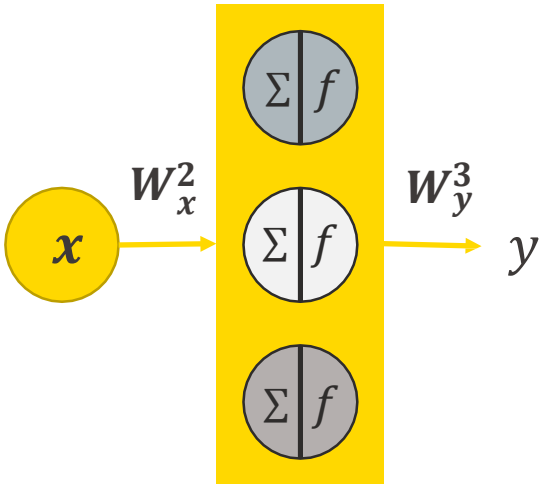
Recurrent Neural Networks

- **R**ecurrent **N**eural **N**etworks (RNNs) are a family of neural networks suitable for processing of sequential data
- Key idea: use a loop connection

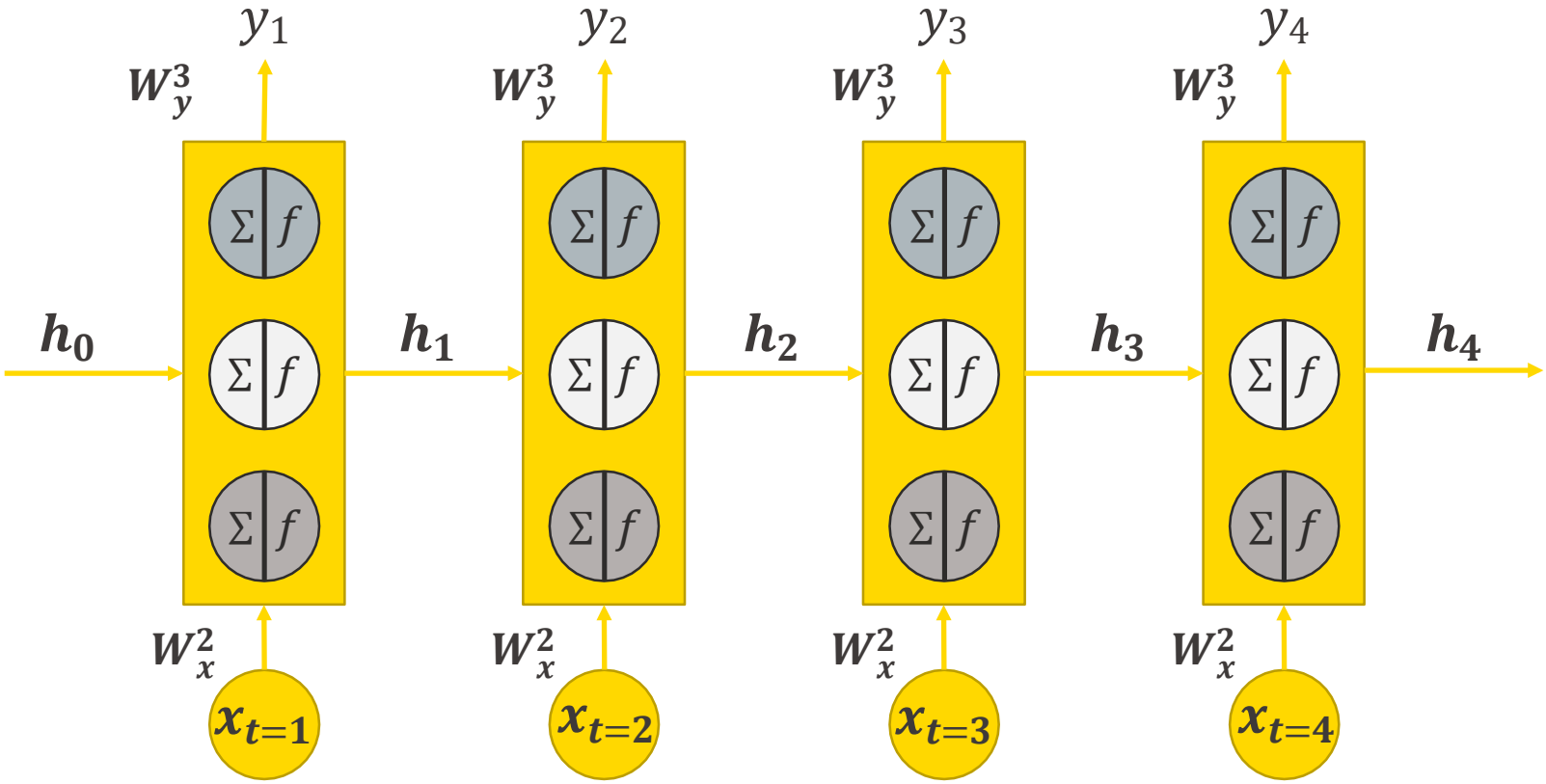
- RNNs are used for all sorts of tasks:
  - Language modeling / Text generation
  - Text classification
  - Neural machine translation
  - Text summarization
  - Image captioning
  - Speech to text
  - Demand prediction
  - Stock price prediction
  - …

From Feed Forward to Recurrent Neural Networks

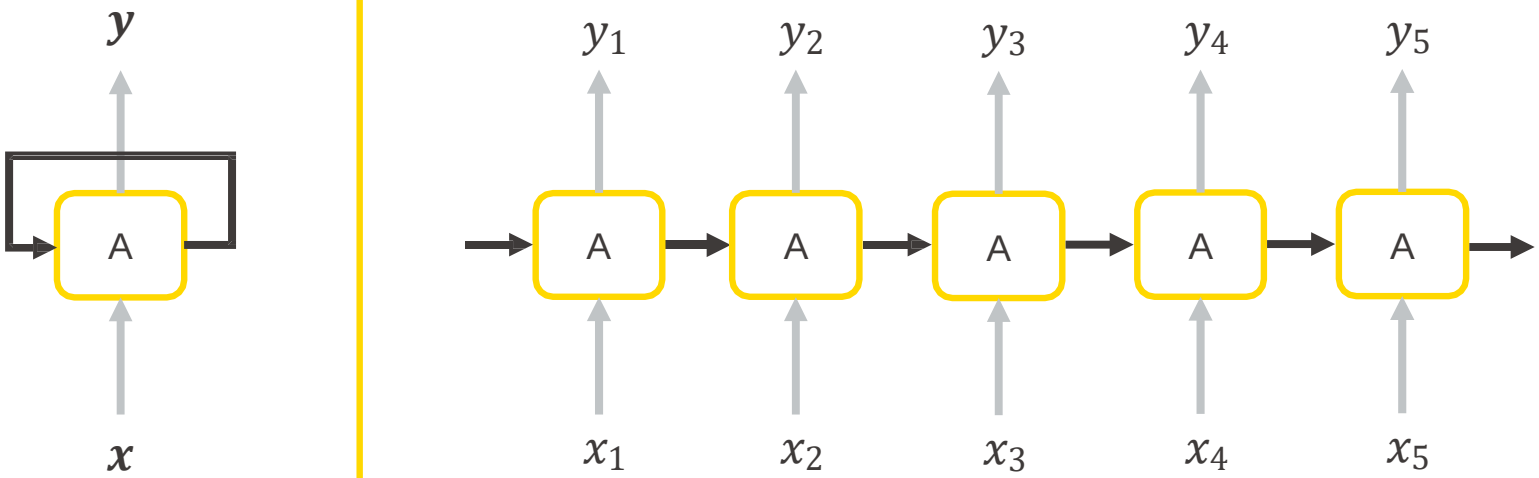




From Feed Forward to Recurrent Neural Networks

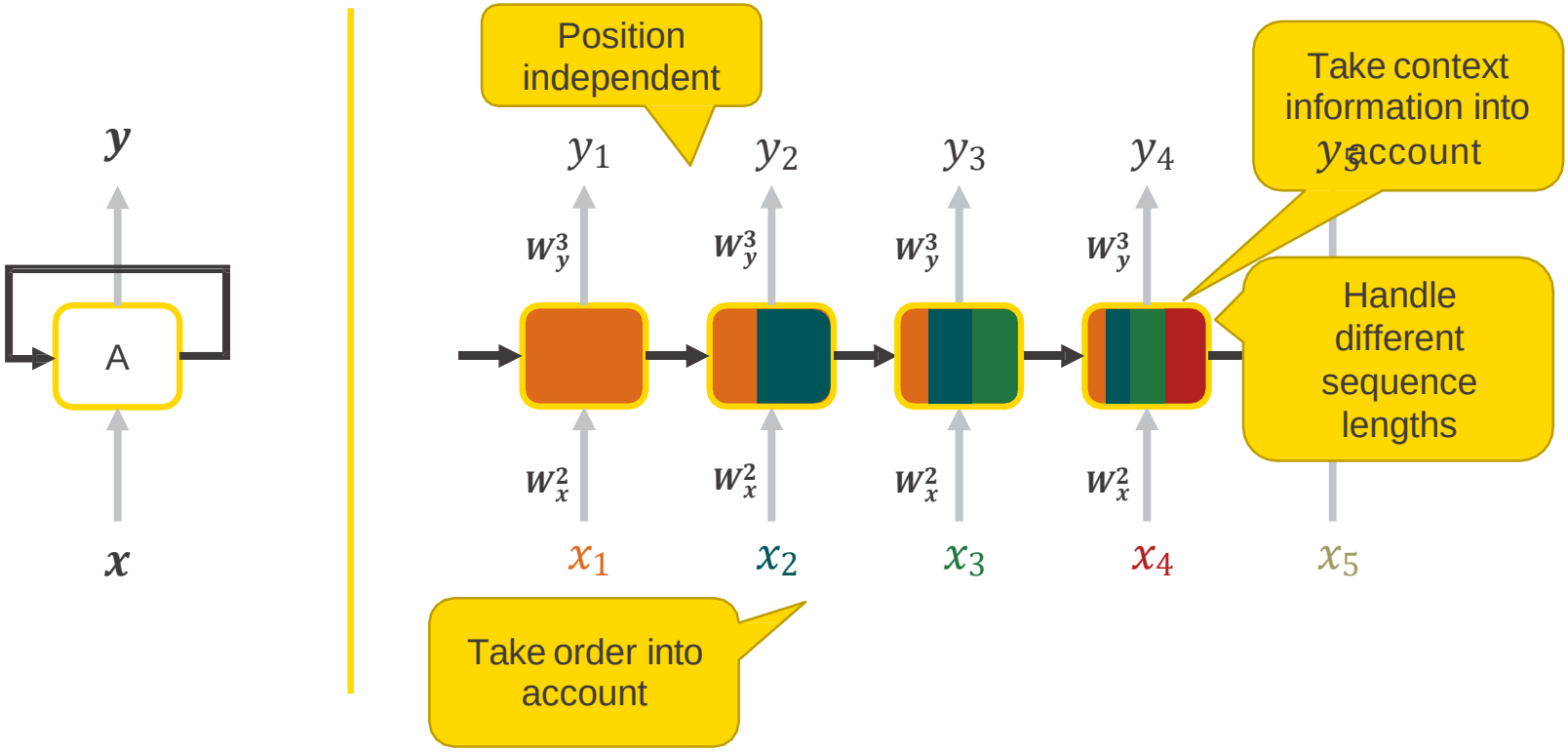


RNN Rolled and Unrolled Representation



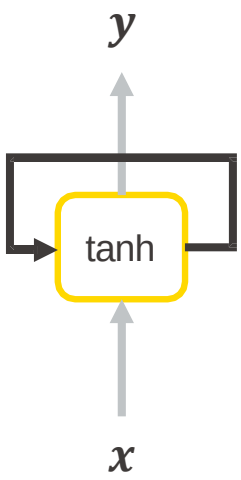
A = A feed forward network with one or multiple layers

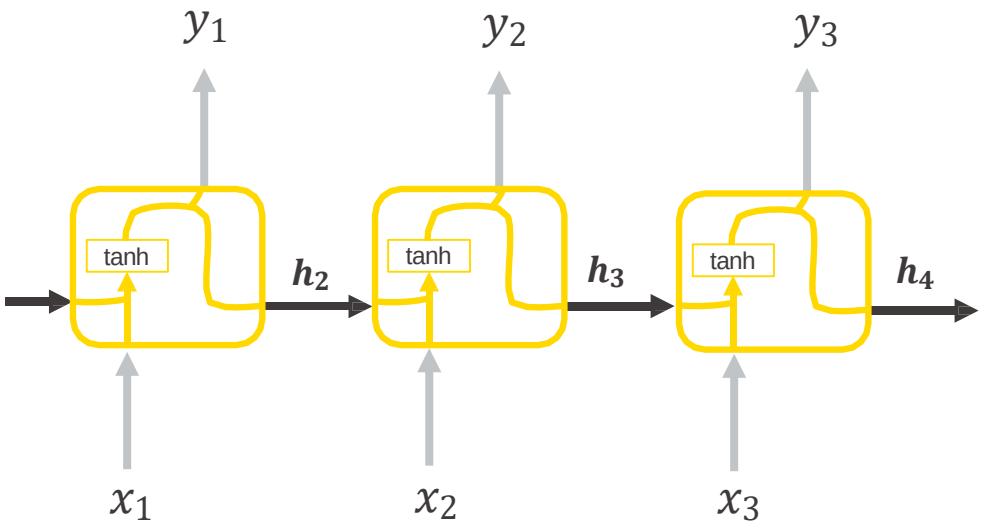
Memory of an RNN



A = A feed forward network with one or multiple layers

The Math Behind An RNN





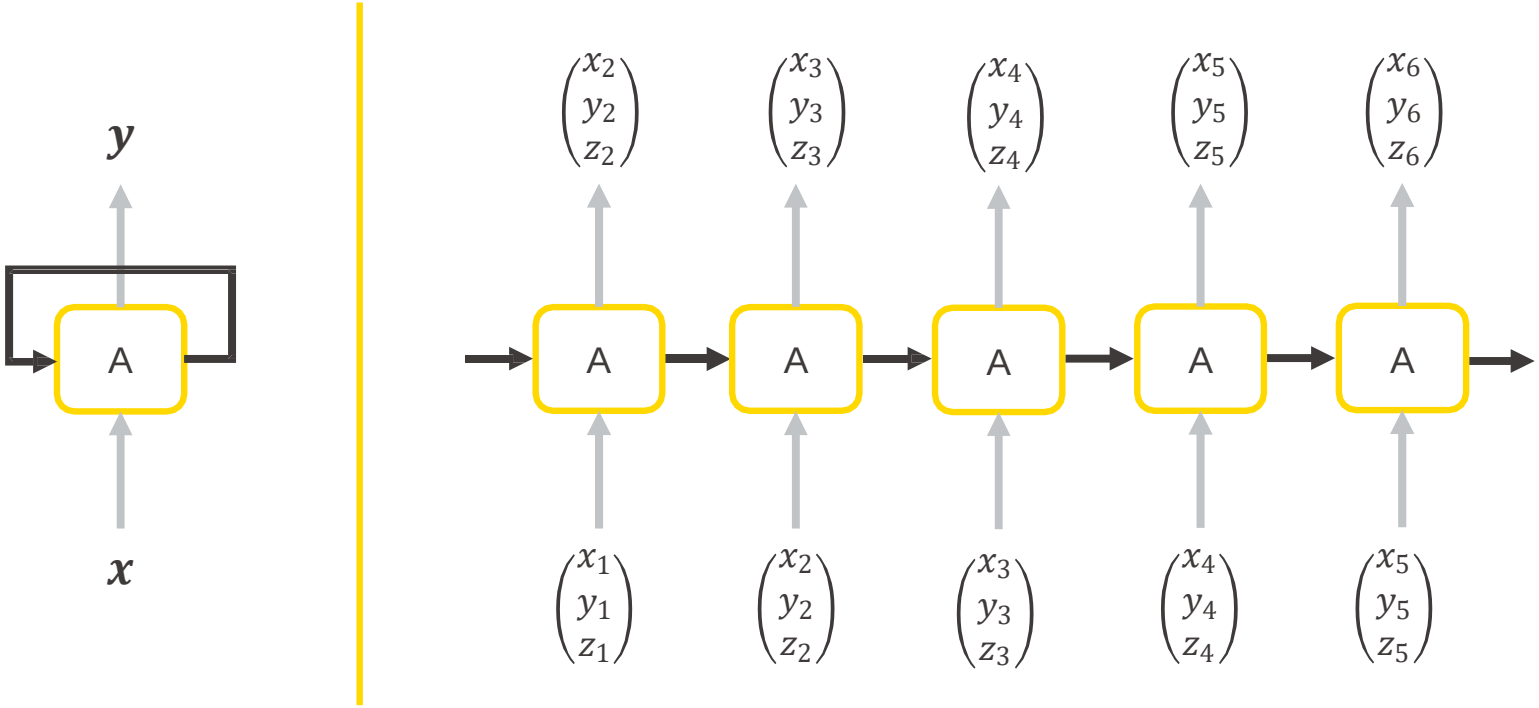
$$y_1 = \tanh(W_x x_1 + W_h h_0)$$

$$y_2 = \tanh(W_x x_2 + W_h h_1)$$

$$\vdots$$

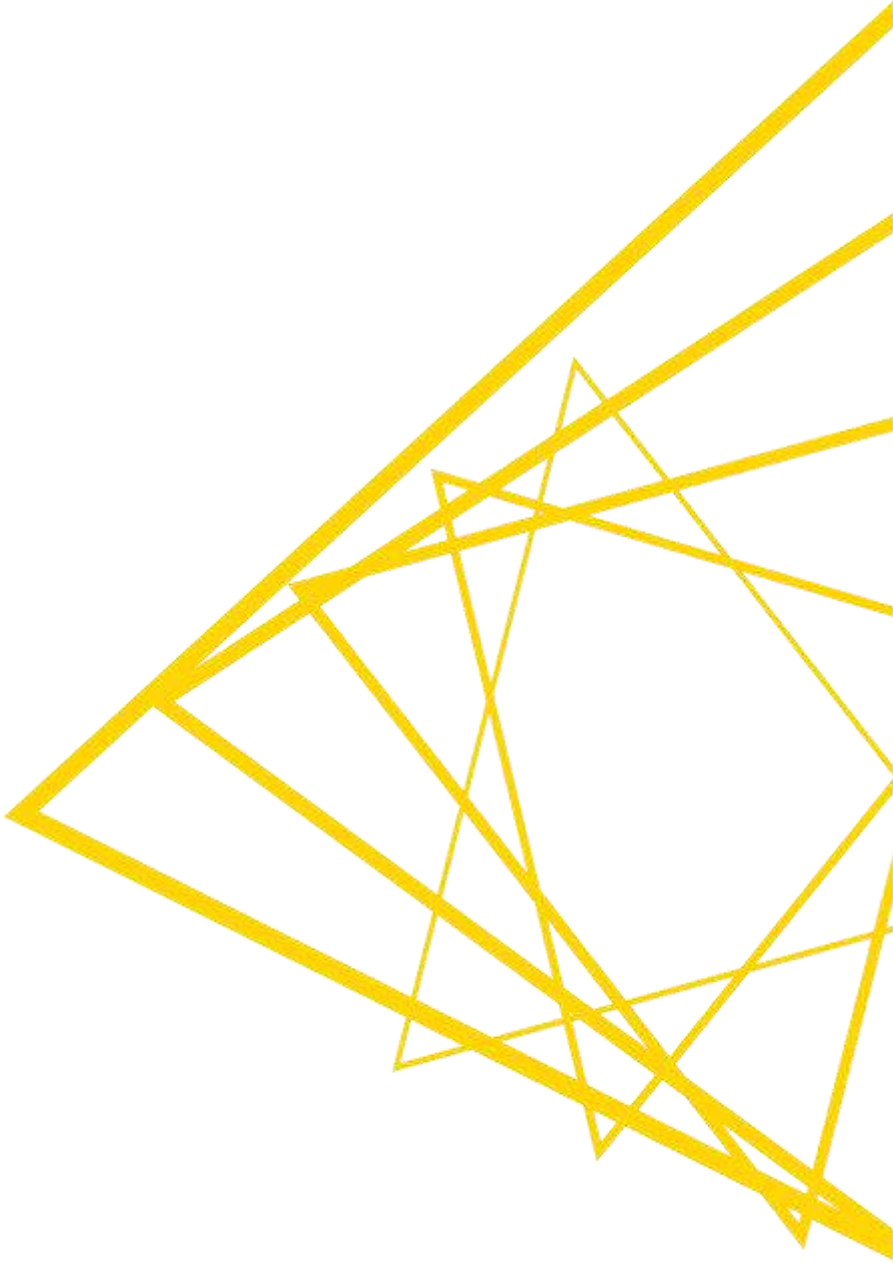
$$y_n = \tanh(W_x x_n + W_h h_{n-1})$$

Example: Trajectory of a Ball

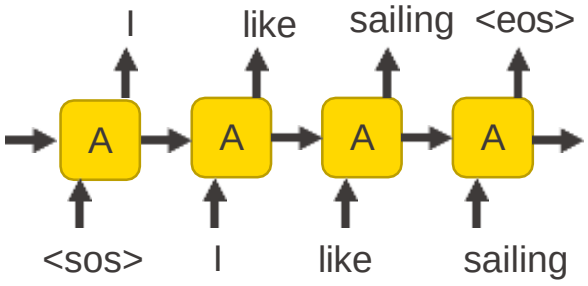


A = A feed forward network with one or multiple layers including the weights

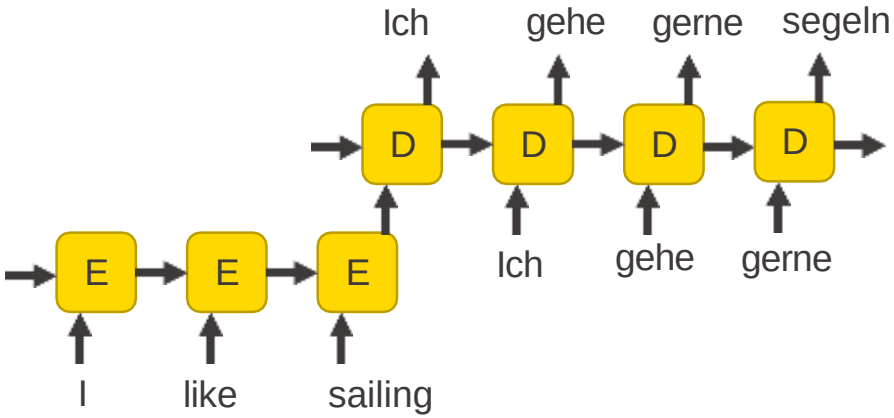
Different RNN Architectures



RNN Architectures: Many-to-Many (Seq2Seq)



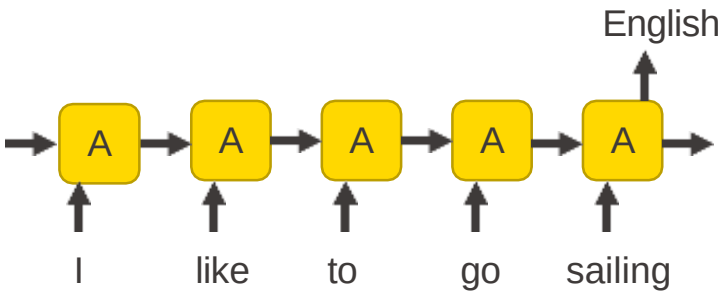
Language model



Neural machine translation

RNN Architectures: Many-to-One & One-to-Many

Many-to-one



Language classification

One-to-many

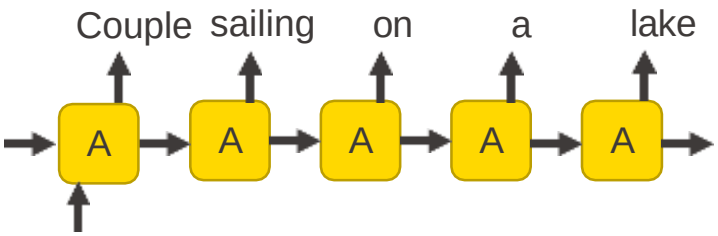
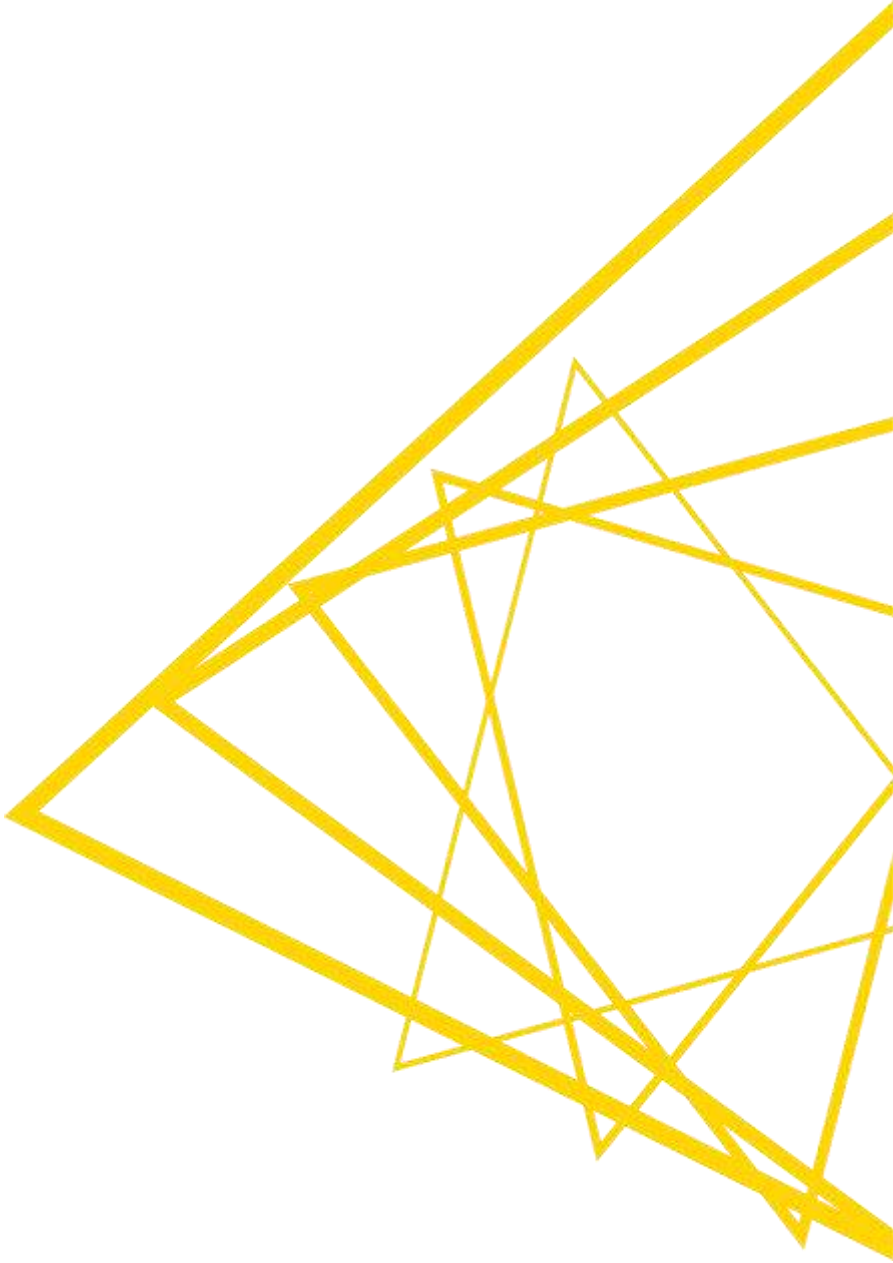


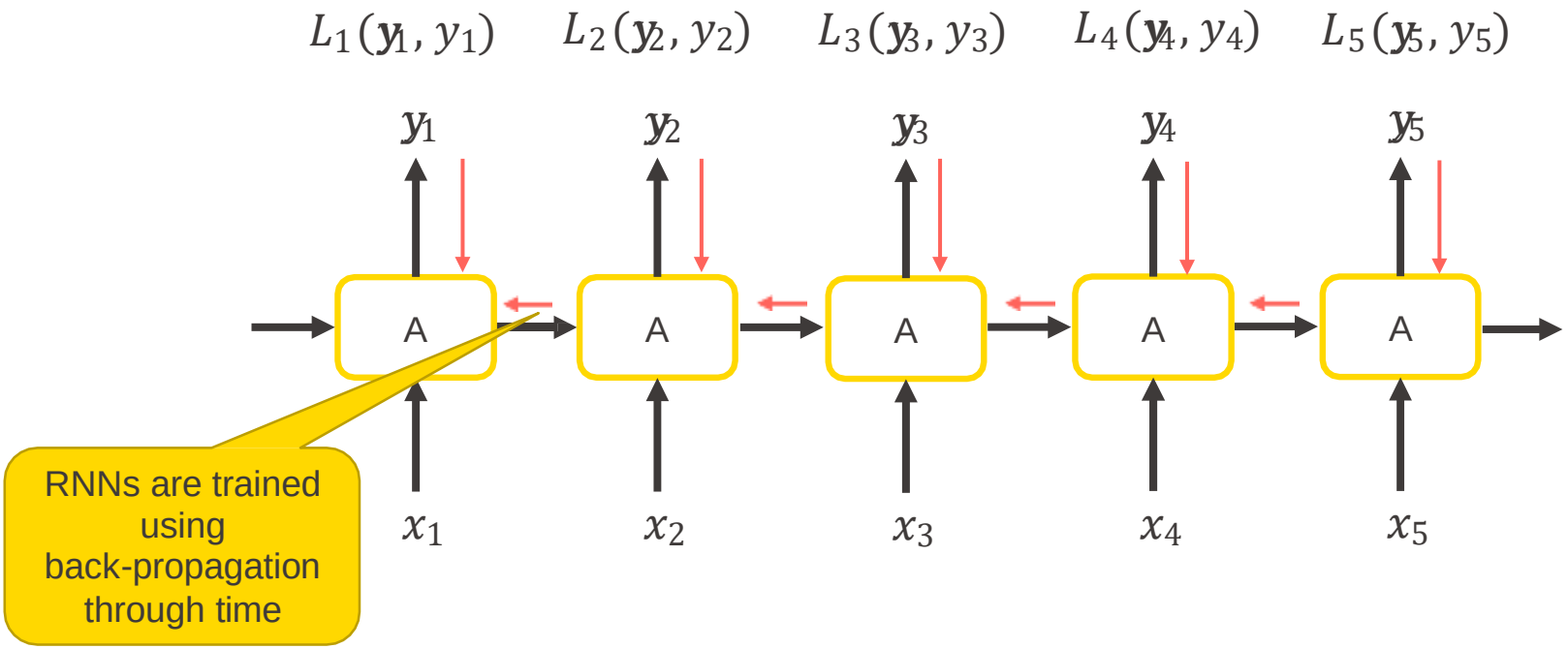
Image captioning

Training an RNN



Training an RNN

$$L = \sum_{i=1}^{5} L_i(\hat{y}_i, y_i)$$



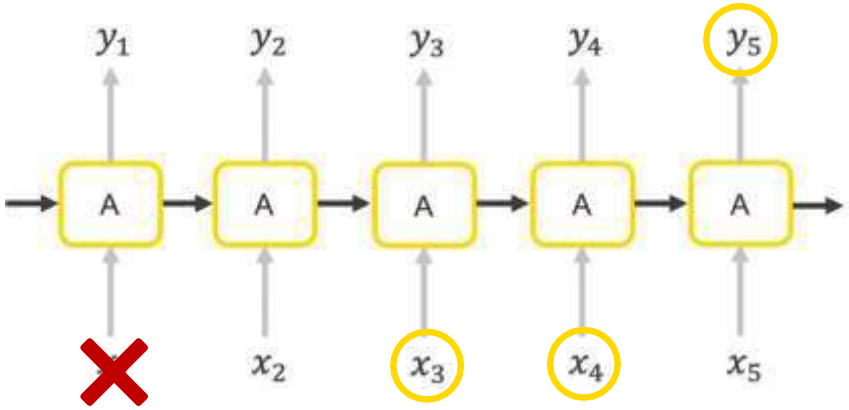
Limitation of Simple RNNs

The “memory” of simple RNNs is sometimes too limited to be useful:

- *“Cars drive on the ____” (road)*

- *“I love the beach.*

  My favorite sound is the crashing of ____“ (cars? glass? waves?)



Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM) Units

Special type of unit with

- an additional cell state
- gates

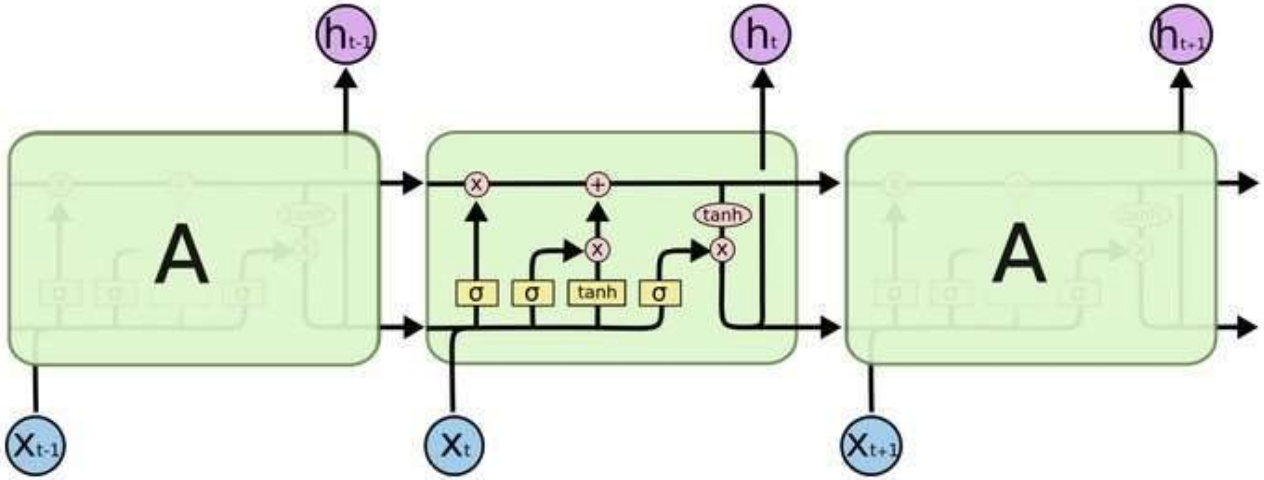


Image Source: Christopher Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

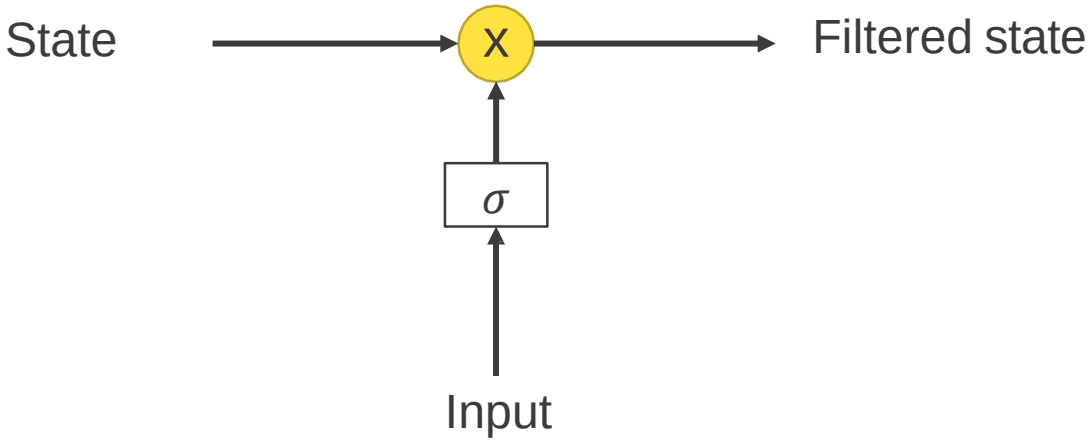
Idea of a Gate

- A gate can be …

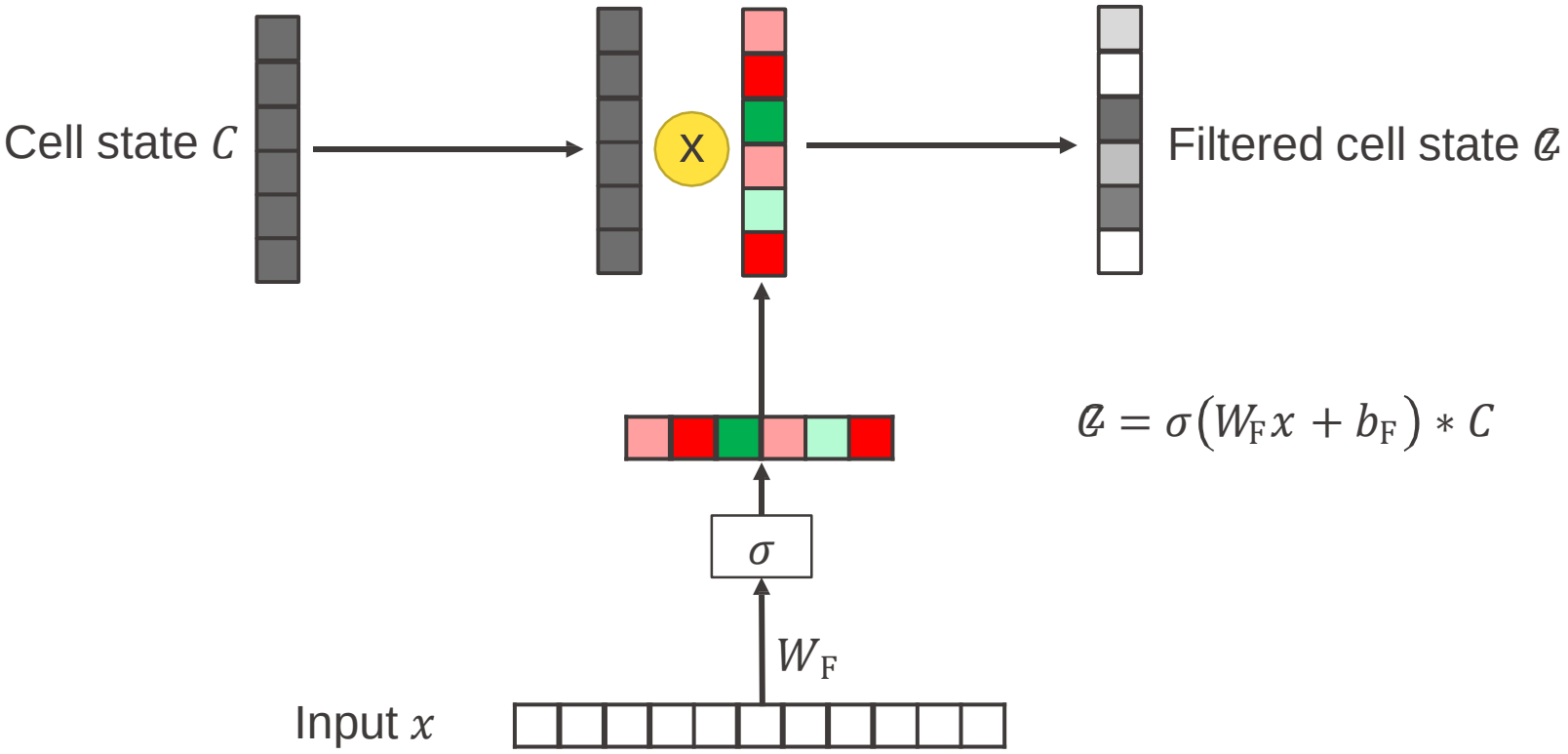
	open	1	Let all information through
	partially open	(0,1)	Let part of the information through
	closed	0	Let no information through

Implementing a Gate

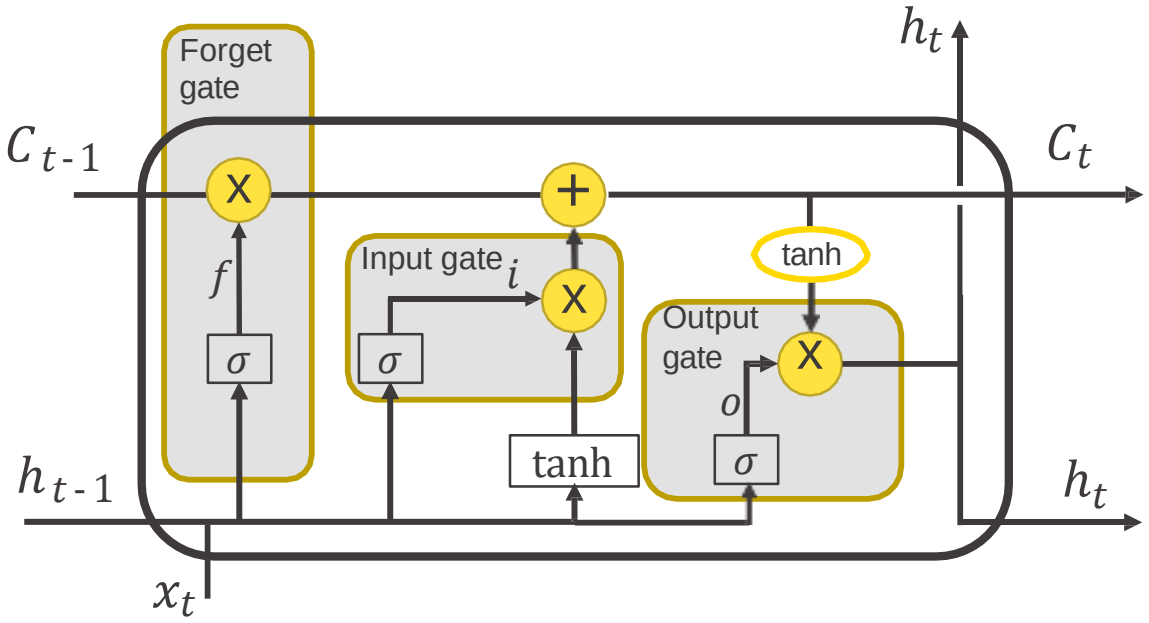
- A gate decides for each dimension whether it is open, partially open, or closed
- A gate is implemented via a sigmoid layer and a pointwise multiplication
  - Sigmoid layer produces values between 0 and 1
  - Pointwise multiplication applies gate to input



Implementing a Gate



Three Gates in an LSTM Unit

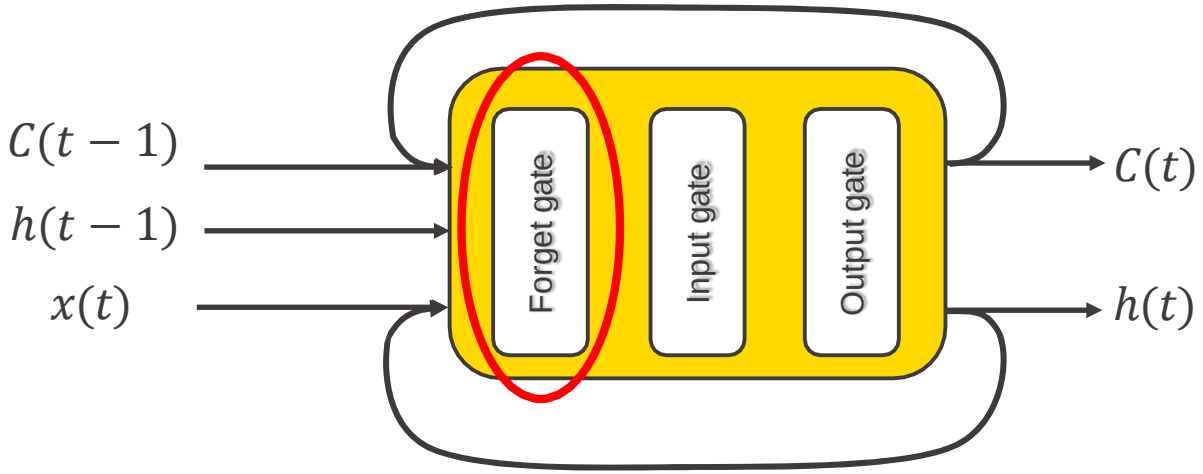


$$f = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$i = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

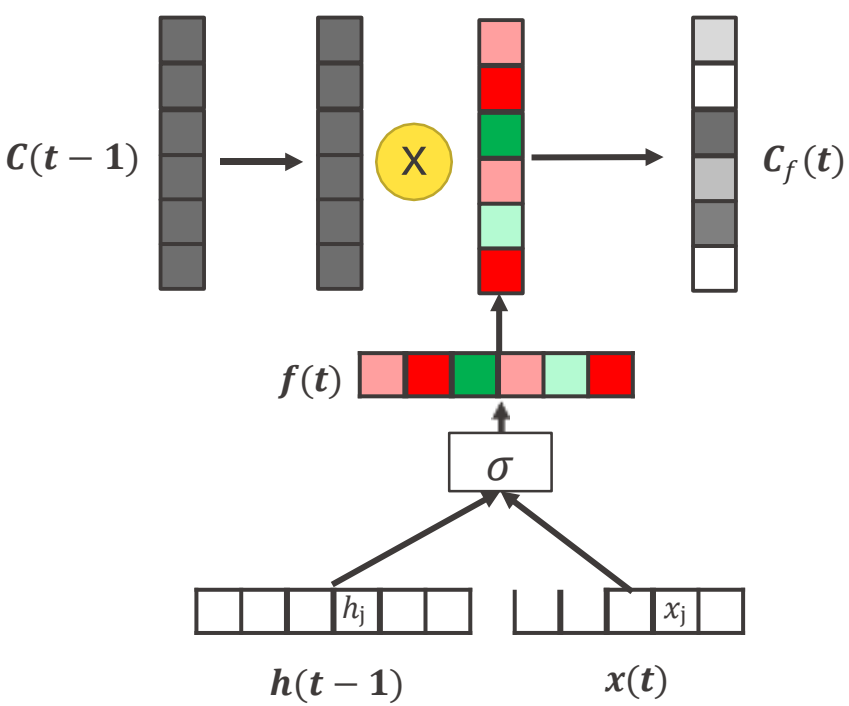
$$o = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

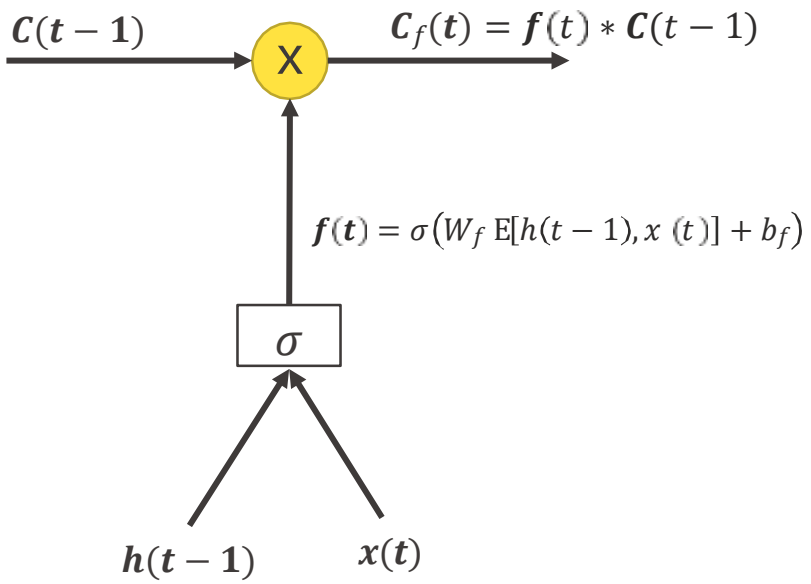
LSTM Unit



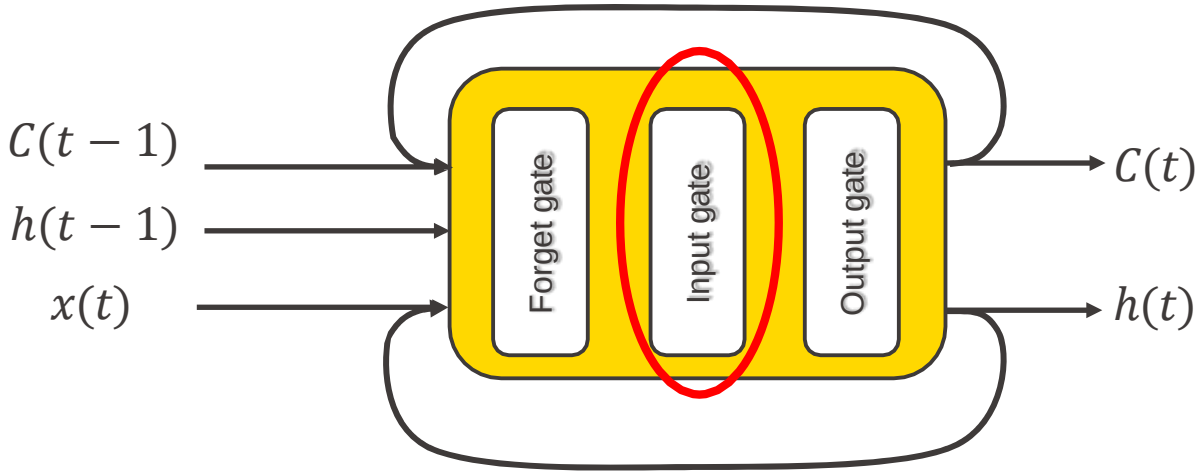
LSTM: Forget Gate

- Forget gate is trained to forget parts of the cell state.
- At time t, the forget gate decides which item of $\boldsymbol{C}(t-1)$ to keep (and how much of it) in $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$



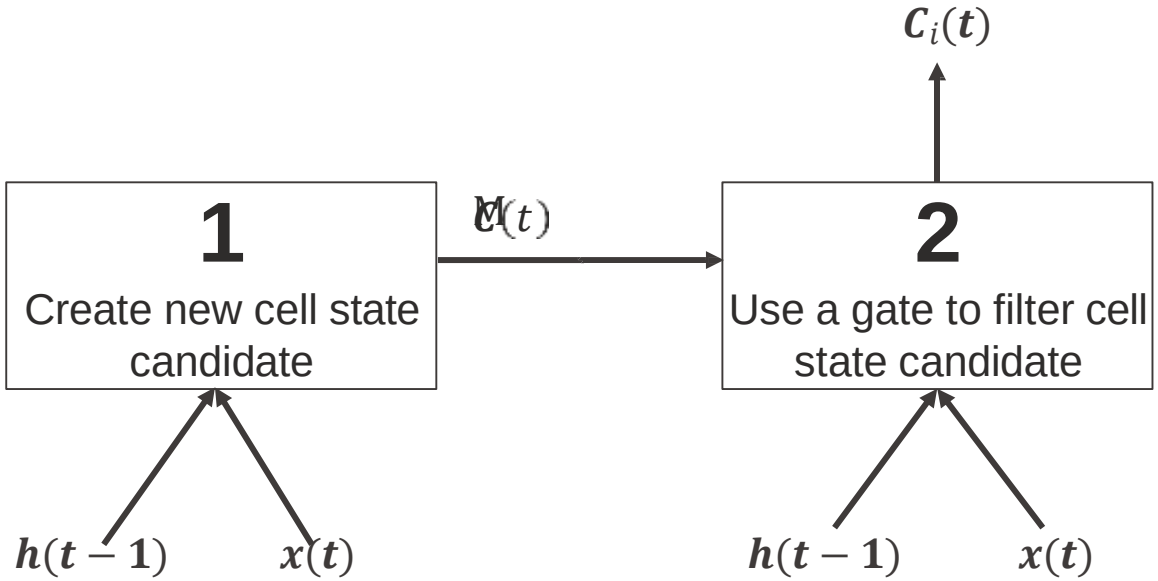


LSTM Unit



LSTM: Input Gate

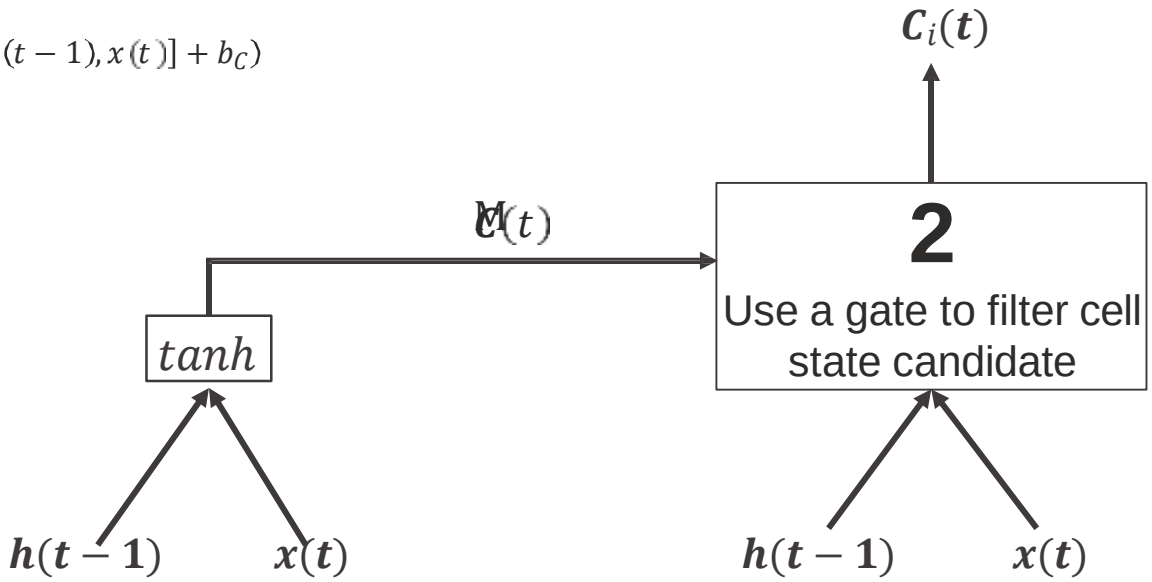
- Input gate is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



LSTM: Input Gate

- Input gate is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

$$\tilde{C}(t) = tanh(W_C \, \mathrm{E}[h(t-1), x(t)] + b_C)$$

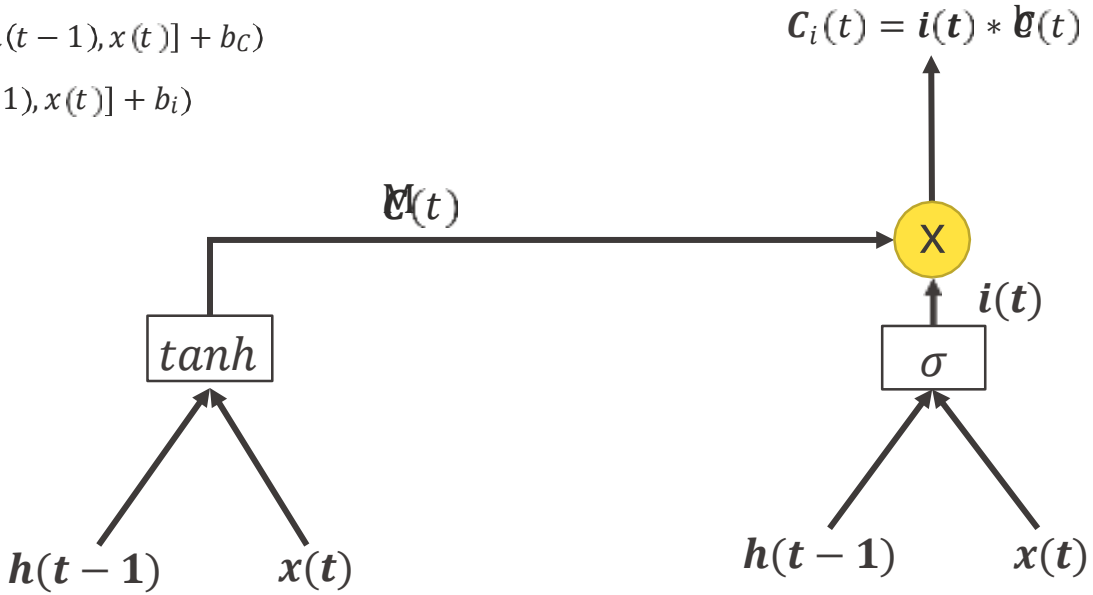


LSTM: Input Gate

- Input gate is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

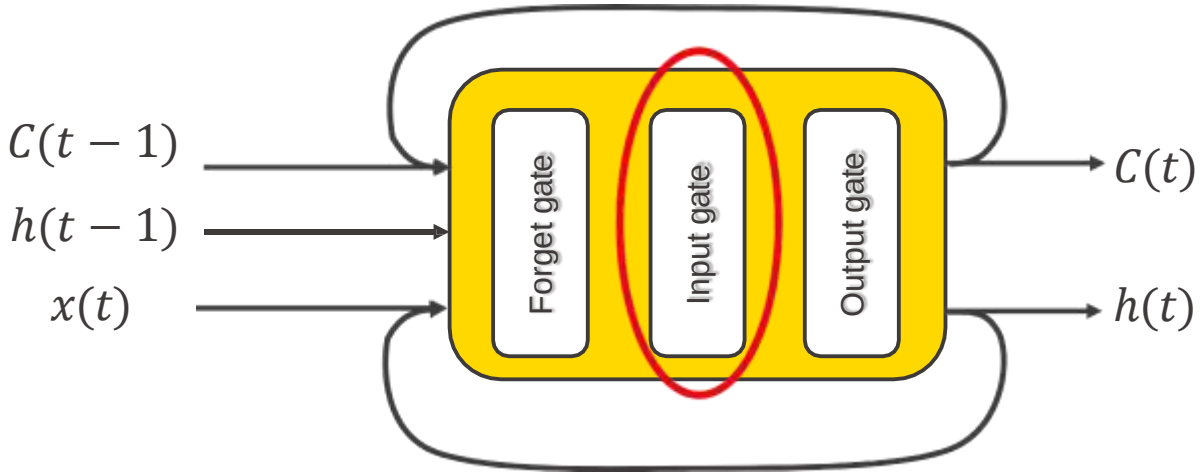
$$\tilde{C}(t) = tanh(W_C \, \mathrm{E}[h(t-1), x(t)] + b_C)$$

$$i_j(t) = \sigma(W_i \, \mathrm{E}[h(t-1), x(t)] + b_i)$$



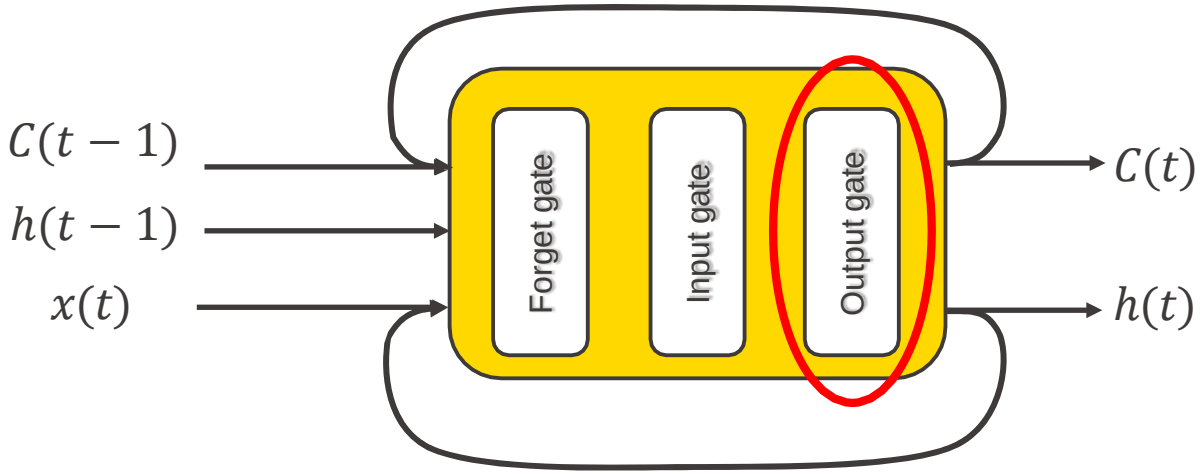
LSTM Unit: Update

Add output from input gate to filtered input from forget gate



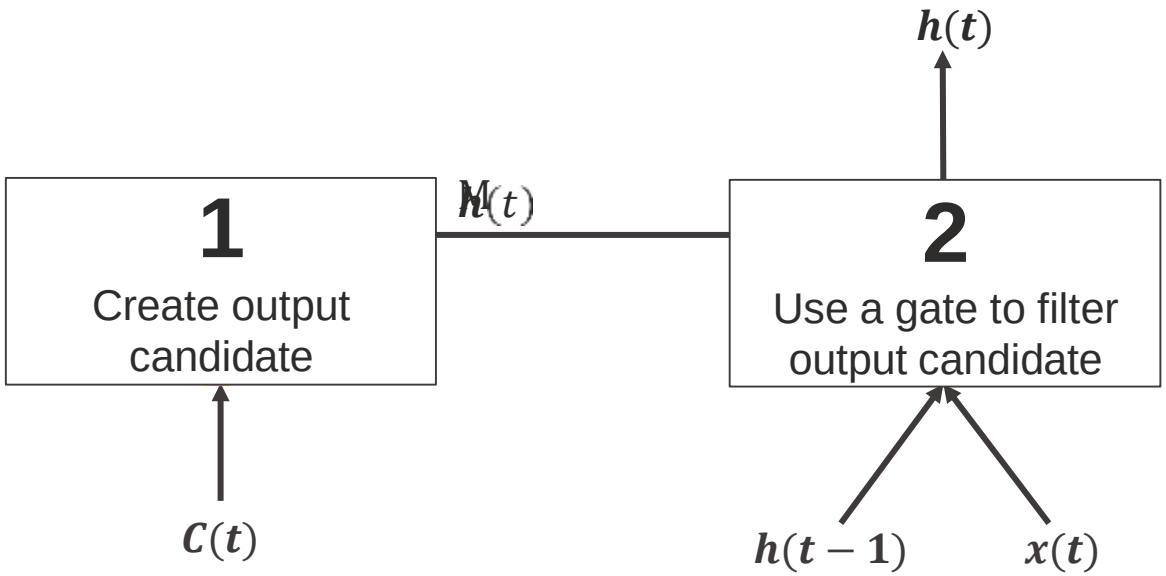
$$\boldsymbol{C}(t) = \boldsymbol{C}_f(t) + \boldsymbol{C}_i(t) = \boldsymbol{f}(t) * \boldsymbol{C}(t-1) + \boldsymbol{i}(t) * \boldsymbol{\tilde{C}}(t)$$

LSTM Unit



LSTM: Output Gate

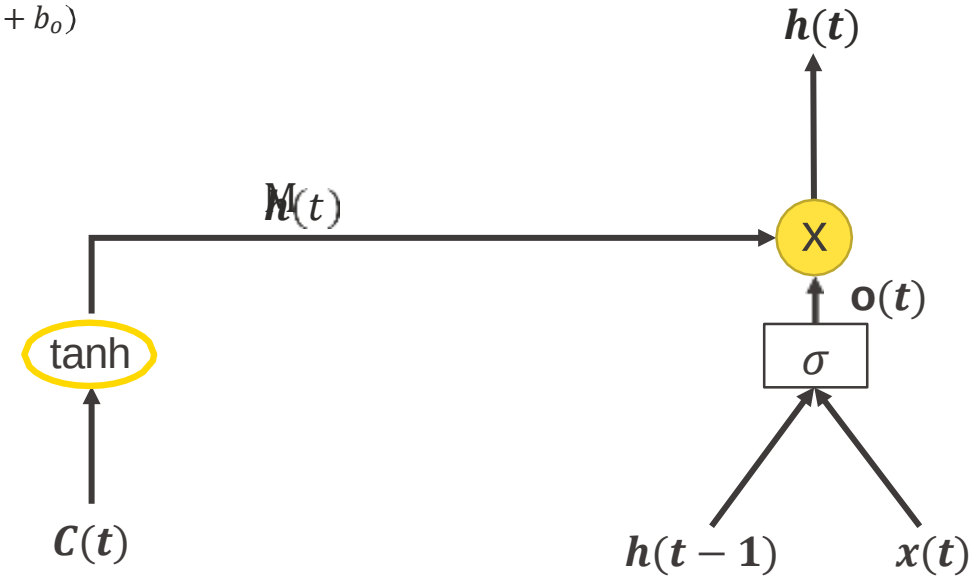
- Output gate is trained to output a reasonable result.
- At time t, output gate decides which parts of status $\boldsymbol{C}(t)$ (and how much of it) will be output, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



LSTM: Output Gate

- Output gate is trained to output a reasonable result.
- At time t, output gate decides which parts of status $\boldsymbol{C}(t)$ (and how much of it) will be output, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

$$\hat{h}(t) = \tanh(C(t))$$
$$o(t) = \sigma(W_o \, \mathrm{E}[h(t-1), x(t)] + b_o)$$
$$h(t) = \hat{h}(t) * o(t)$$



Summary: LSTMs

- Additional cell state makes it easier to remember information
- At each time step
  1. The forget gate removes irrelevant information from the cell state
  2. The input gate decides which information should be added to the cell state
  3. The cell state is updated
  4. The output gate decides which information to output and to send to the next time step

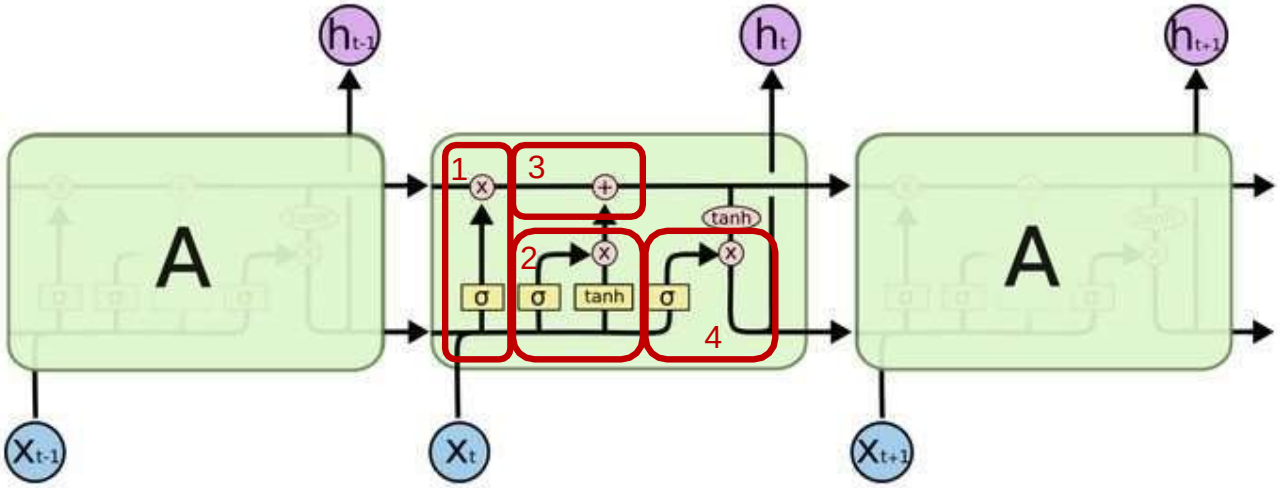
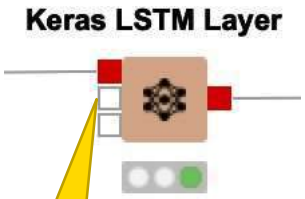
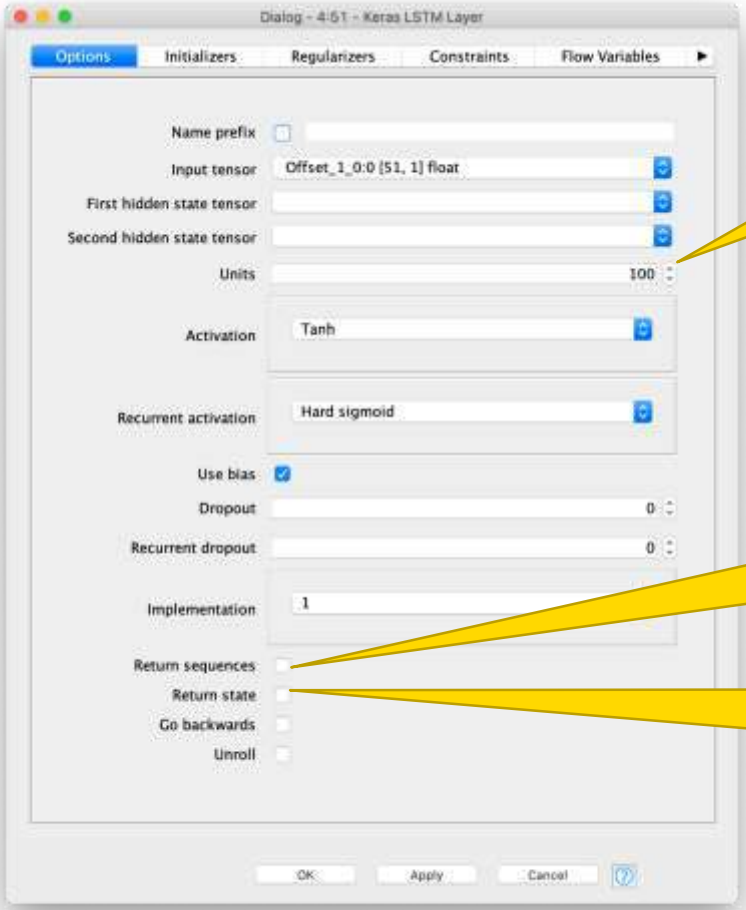


Image Source: Christopher Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Layer Node



Optional input ports for the hidden state tensors

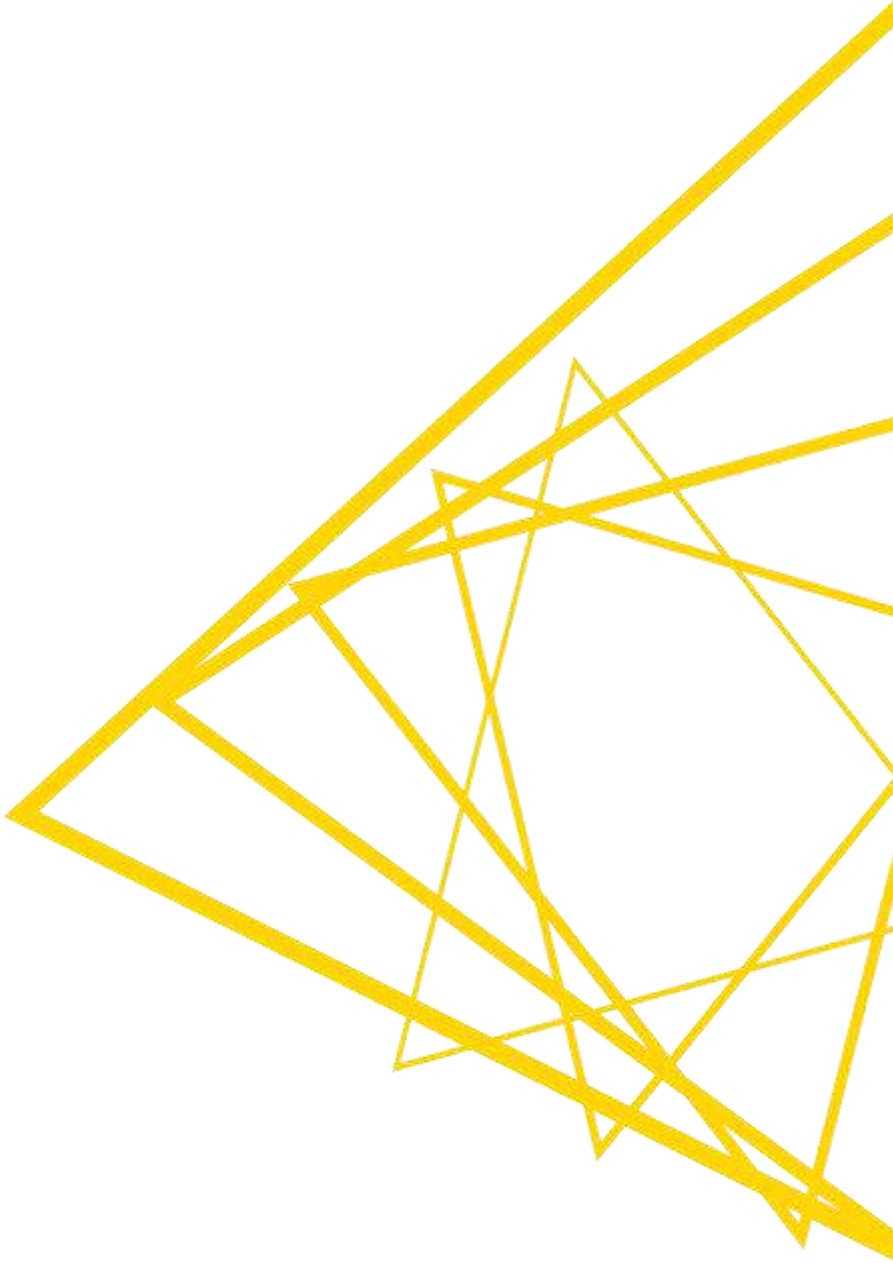


Size of the hidden state vectors

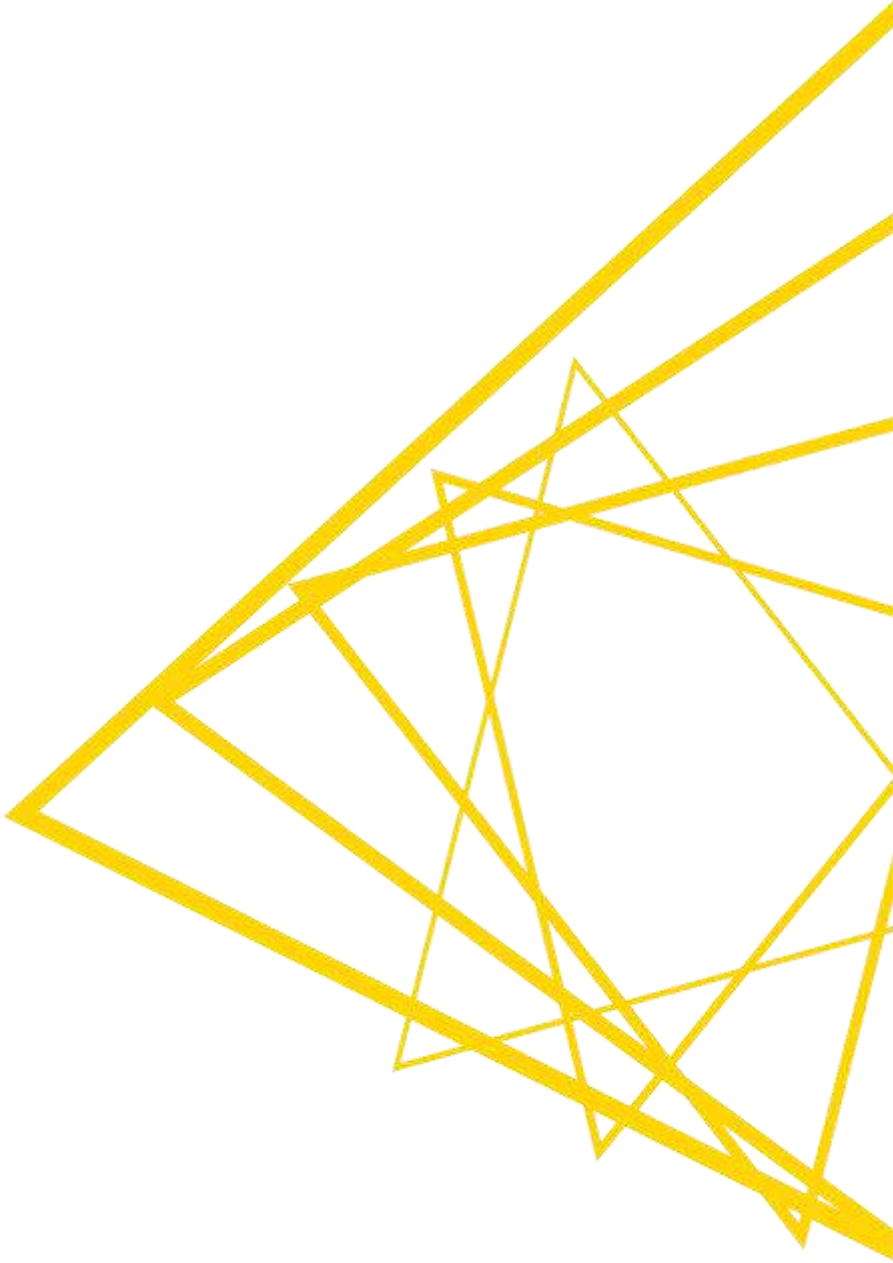
Activate “Return sequences” for seq2seq models

Activate “Return state” to use it during deployment

RNN Examples



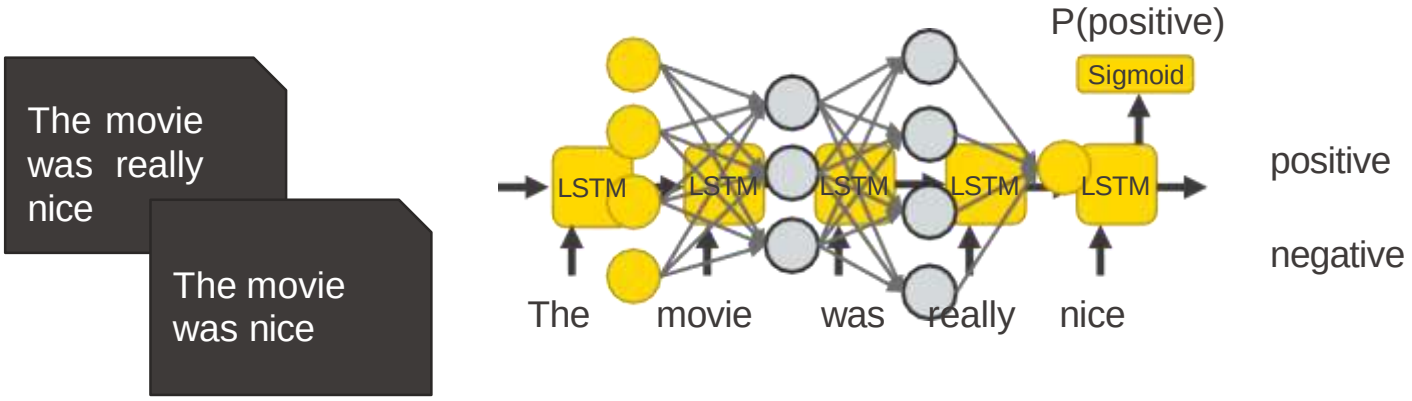
Text Classification – Sentiment Analysis



Text Classification

- Task: Assigning tags or categories to text according to its content
- Examples:
  - Identify the underlying sentiment of movie / restaurant / product reviews, tweets etc.
    - Positive
    - Negative
  - Classify vacation reviews. What is the review about?
    - Hotel
    - Flight
    - Booking process

Use Case: Sentiment Analysis of Movie Reviews



Preprocessing:

- Different sequence lengths
  - Sequences within the same training batch must have the same length
  - Solution: Truncate too long sequences and zero pad too short sequences
- Encoding
  - Index encoding plus embedding layer
  - Large number of different words: Define a fixed dictionary size and assign default (“unknown”) value to all other words

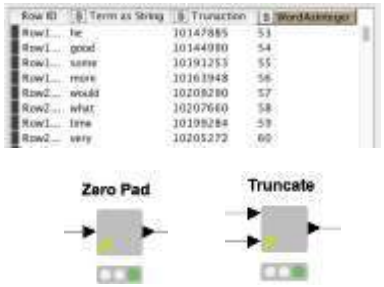
Idea Embedding Layer

- Goal: Represent each word as a vector in a vector space (embedding space)
  - Words with similar meanings have similar vectors
  - The distance between any two vectors captures part of the semantic relationship between the two associated words
- Advantage: Embedding space has a lower dimension
  - Only a few tens or hundreds
  - One-hot vector encodings many thousands
- You can …
  - … train your own embeddings via an Embedding layer
  - … use pretrained embeddings, e.g GloVe

Use Case: Sentiment Analysis of Movie Reviews

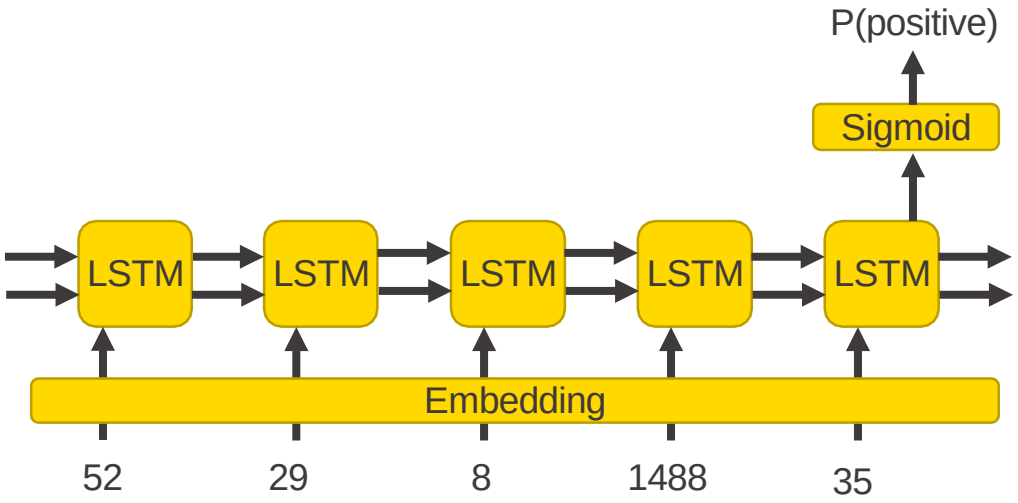
Preprocessing

This film is mediocre at best. Angie Harmon is as funny as a bag of hammers. Her bitchy demeanor from Law and Order carries over in a failed attempt at comedy. Charlie Sheen is the only one to come out unscathed in this horrible anti-comedy. The only positive thing to come out of this mess is Charlie and Denise's marriage. Hopefully that effort produces better results.

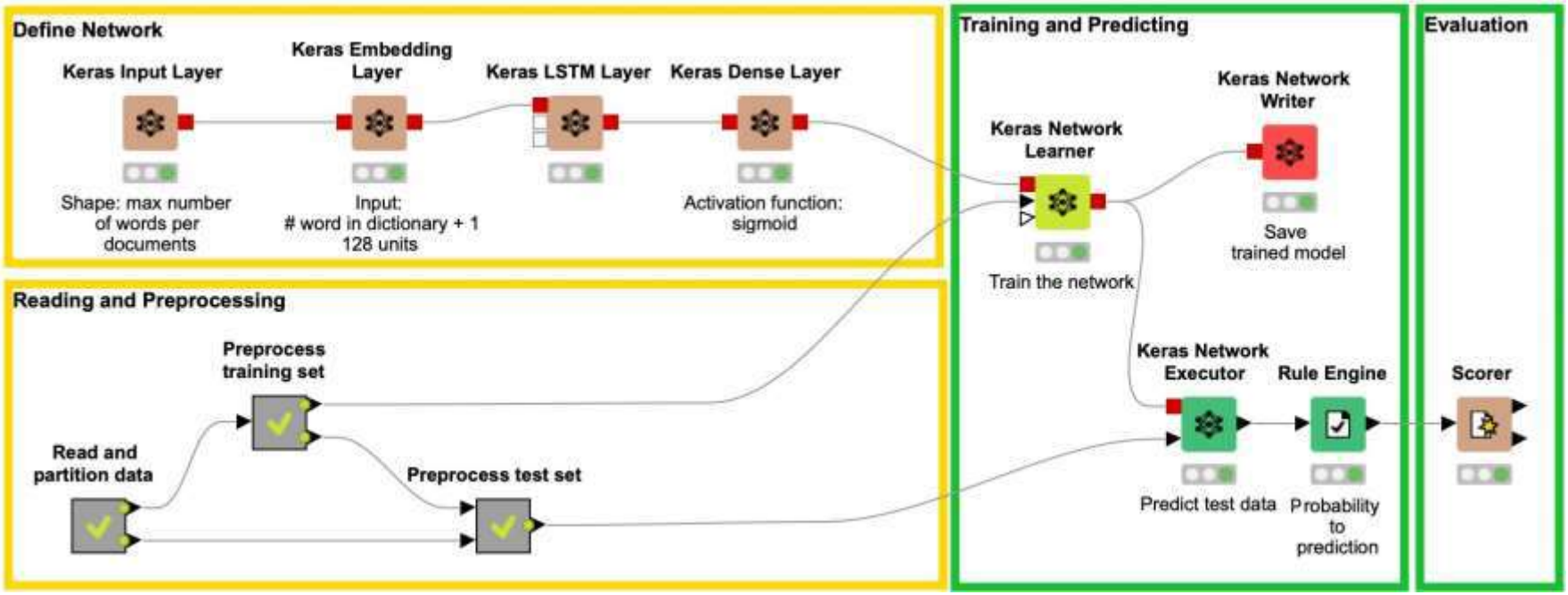


52 29 8 1488 35 130 1 11794 18869 8 20
187 20 3 2984 6 20001 1 883 8167 8409
39 3791 4 8384 2716 148 9 3 1154 585 35
258 1 1987 5781 6 2 65 30 7 221 44
13964 9 10 552 20001 1 18 65 1124 159 7
221 44 6 10 931 8 1987 4 10239 14 1570
1 5136 12 741 7256 131 1829 1

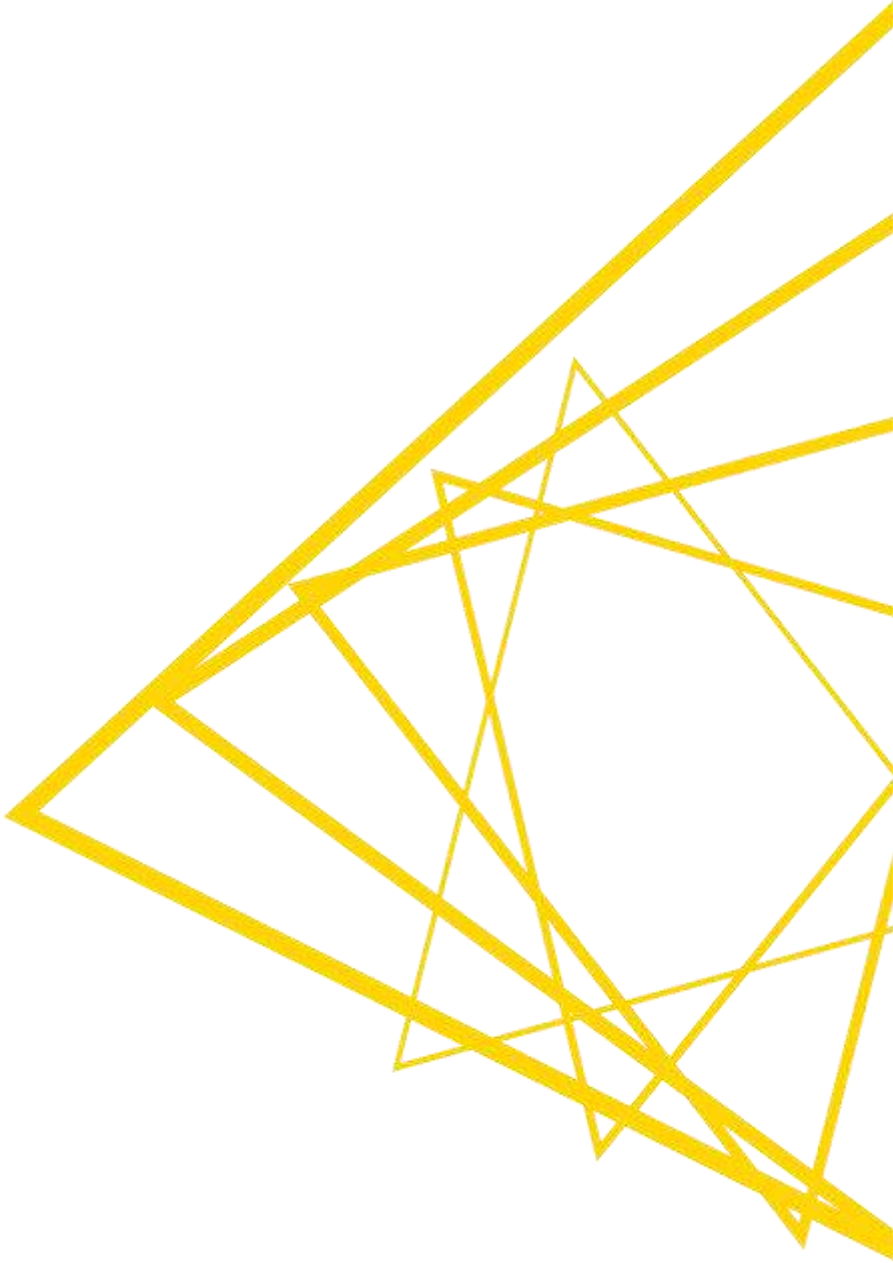
Network



Text Classification: Sentiment Analysis

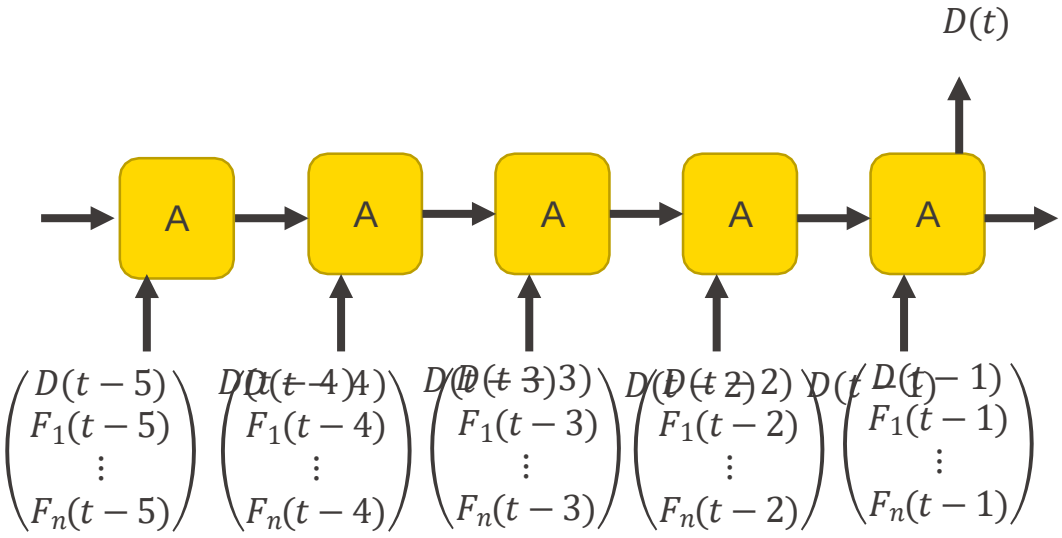


Time Series Prediction



Time Series Prediction

- Goal: Predict the next value in a time series based on recent values
- Examples:
  - Demand prediction
  - Stock price predictions
  - Product price predictions



Training Data for Univariate Time Series Prediction

- Example: Using the last-4-times step

Time series data

Demand
$x_{t=1}$
$x_{t=2}$
$x_{t=3}$
$x_{t=4}$
$x_{t=5}$
$x_{t=6}$
$x_{t=7}$
$x_{t=8}$
$x_{t=9}$

Training data

Demand - 4	Demand - 3	Demand -2	Demand -1	Demand
$x_{t=1}$	$x_{t=2}$	$x_{t=3}$	$x_{t=4}$	$x_{t=5}$
$x_{t=2}$	$x_{t=3}$	$x_{t=4}$	$x_{t=5}$	$x_{t=6}$
$x_{t=3}$	$x_{t=4}$	$x_{t=5}$	$x_{t=6}$	$x_{t=7}$
$x_{t=4}$	$x_{t=5}$	$x_{t=6}$	$x_{t=7}$	$x_{t=8}$
$x_{t=5}$	$x_{t=6}$	$x_{t=7}$	$x_{t=8}$	$x_{t=9}$

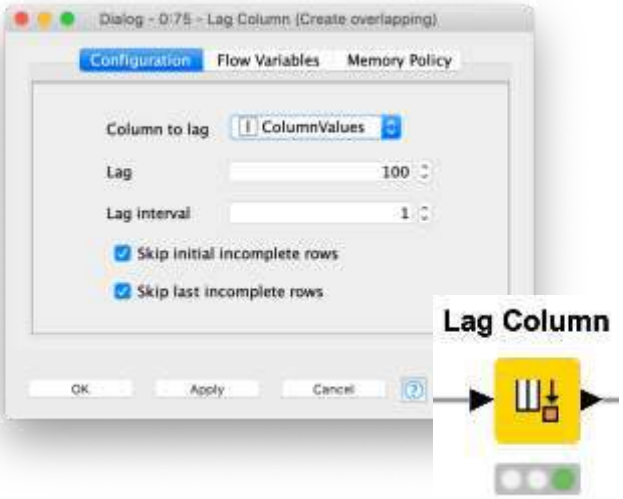
Input sequence

Target

Lag Column Node

- Appends past values as new columns
  - Lag: Number of new columns, aka number of past values that are added to each row
  - Lag interval: Numbers of steps the data is shifted
- Can be used to create overlapping sequences
- Example: Lag = 3; Lag interval = 1

Value
x_1
x_2
x_3
x_4
x_5
x_6



Value	Value -1	Value -2	Value -3
x_4	x_3	x_2	x_1
x_5	x_4	x_3	x_2
x_6	x_5	x_4	x_3

Training Data for Multivariate Time Series Prediction

- Example:
  - Target feature T plus two additional features, F_1 and F_2
  - Using the last 4 time steps

How can we feed the input matrix into the Keras model?

Input	Target
$\begin{pmatrix} T(t=1) & T(t=2) & T(t=3) & T(t=4) \\ F_1(t=1) & F_1(t=2) & F_1(t=3) & F_1(t=4) \\ F_2(t=1) & F_2(t=2) & F_2(t=3) & F_2(t=4) \end{pmatrix}$	$T(t=5)$
$\begin{pmatrix} T(t=2) & T(t=3) & T(t=4) & T(t=5) \\ F_1(t=2) & F_1(t=3) & F_1(t=4) & F_1(t=5) \\ F_2(t=2) & F_2(t=3) & F_2(t=4) & F_2(t=5) \end{pmatrix}$	$T(t=6)$

Preparing the Data in KNIME Analytics Platform

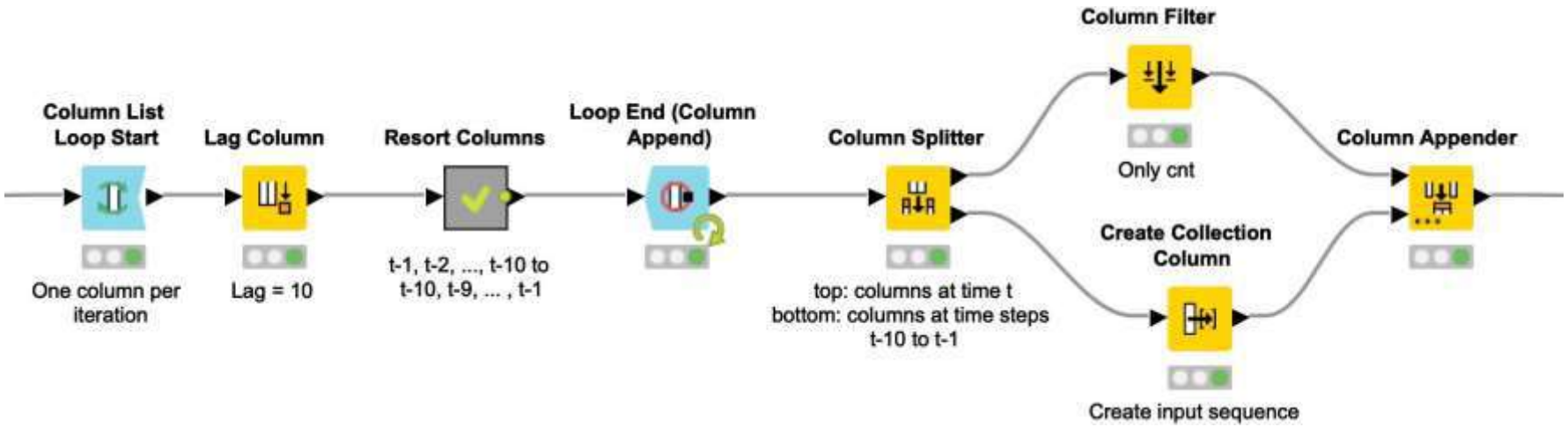
- The Keras Network Learner node can create the input matrix based on a vector obtained by concatenating the cells in the matrix.
  - One feature (row) after the other, with values sorted from the earliest to the most recent

$$\begin{pmatrix} T(t=1) & T(t=2) & T(t=3) & T(t=4) \\ F_1(t=1) & F_1(t=2) & F_1(t=3) & F_1(t=4) \\ F_2(t=1) & F_2(t=2) & F_2(t=3) & F_2(t=4) \end{pmatrix}$$

$$[T(t=1), T(t=2), T(t=3), T(t=4), F_1(t=1), F_1(t=2), F_1(t=3), F_1(t=4), F_2(t=1), F_2(t=2), F_2(t=3), F_2(t=4)]$$

- Important:
  - Define the correct input size in the Keras Input Layer, e.g. 4,3.
  - Store the vector in a collection cell and use the conversion “From Collection of Number (double)”

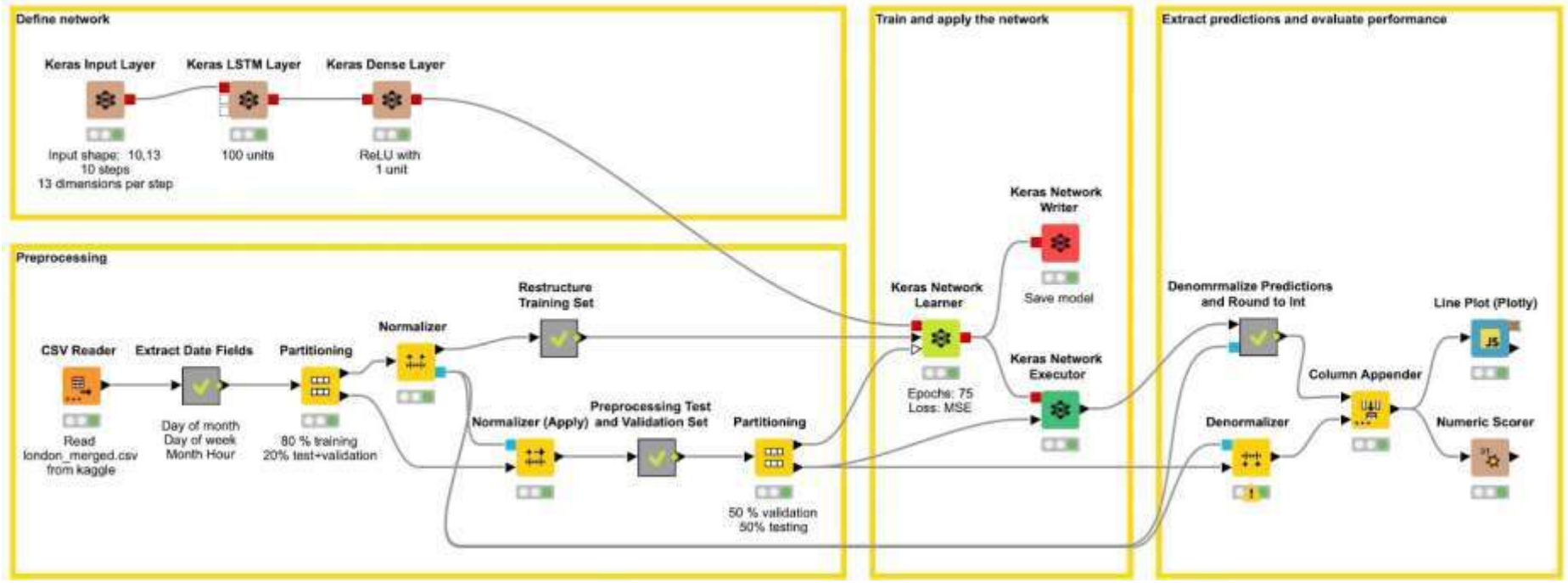
Preparing the Data in KNIME Analytics Platform



Bike Demand Prediction (Multivariate Time Series)

- Dataset: London Bike Sharing dataset from Kaggle
  - TF= "cnt" - the count of a new bike shares
  - F1="t1" - real temperature in C
  - F2="t2" - temperature in C "feels like"
  - F3="hum" - humidity in percentage
  - F4="windspeed" - wind speed in km/h
  - F5="weathercode" - category of the weather
  - F6="isholiday" - boolean field - 1 holiday / 0 non holiday
  - F7="isweekend" - boolean field - 1 if the day is weekend
  - F8="season" - category field meteorological seasons: 0-spring ; 1-summer; 2-fall; 3-winter.
- Goal: Predict the demand for bikes in the next hour based on
  - The demand in the last hours
  - The other feature values in the last hours

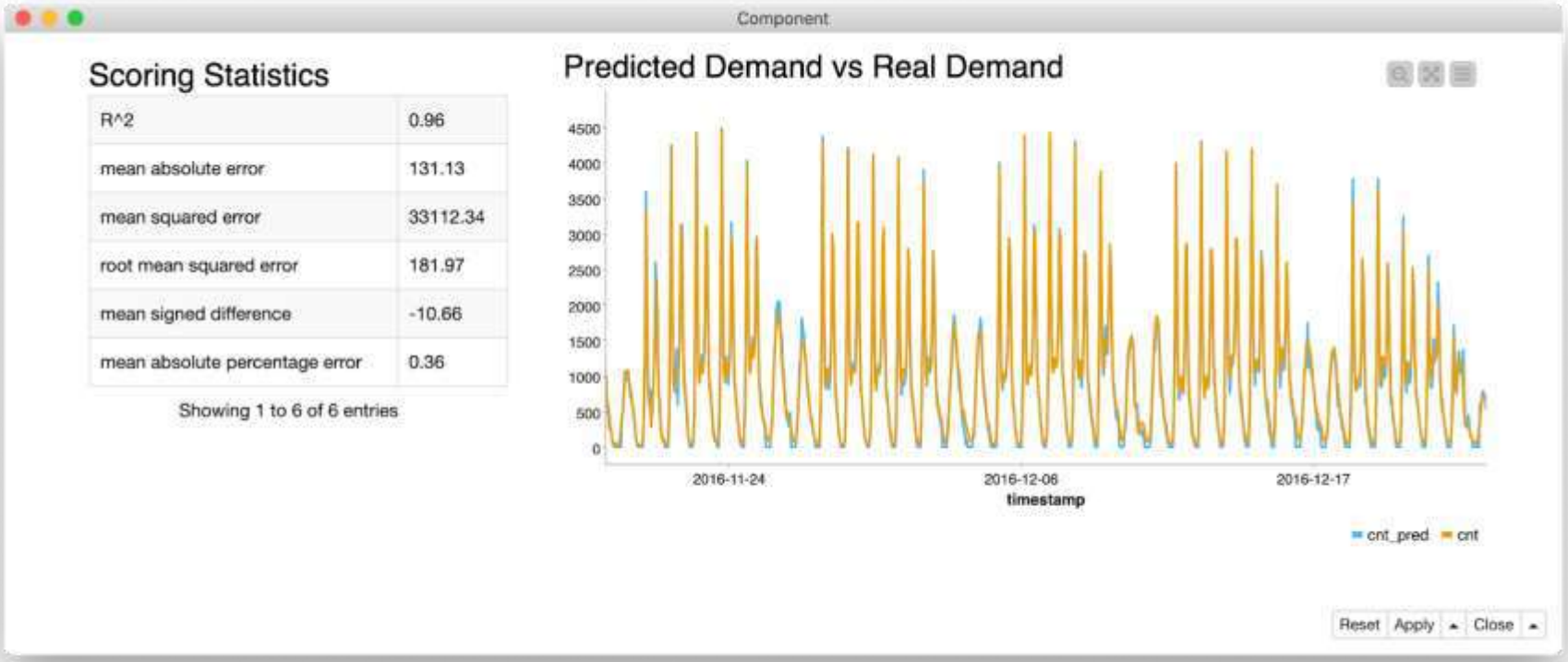
Demand Prediction with Multivariate Time Series Data



Blogpost: https://www.knime.com/blog/multivariate-time-series-analysis-lstm-codeless
Workflow: https://kni.me/s/r-a9pad3savKfRsV

Model Evaluation

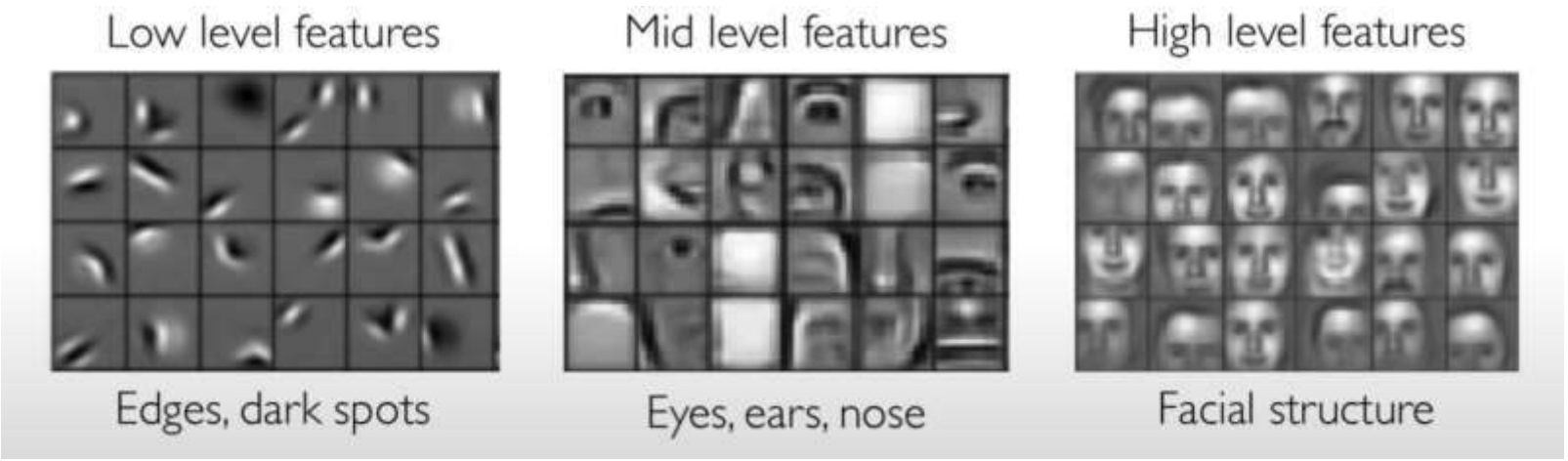
- Training settings:
  - 50 epochs with a training and validation batch size of 32 and Adam as optimizer
  - Shuffling the data before each epoch



Convolution Neural Networks (CNN)

Convolutional Neural Network (CNN)

- A CNN is a neural network with at least one convolutional layer.
- Instead of handcrafting different features a CNN learns a hierarchy of features using multiple convolution layers that detect different features.



Images from: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

How Do Convolutional Layers Works?

- Idea: Instead of connecting every neuron to the new layer a sliding window is used.

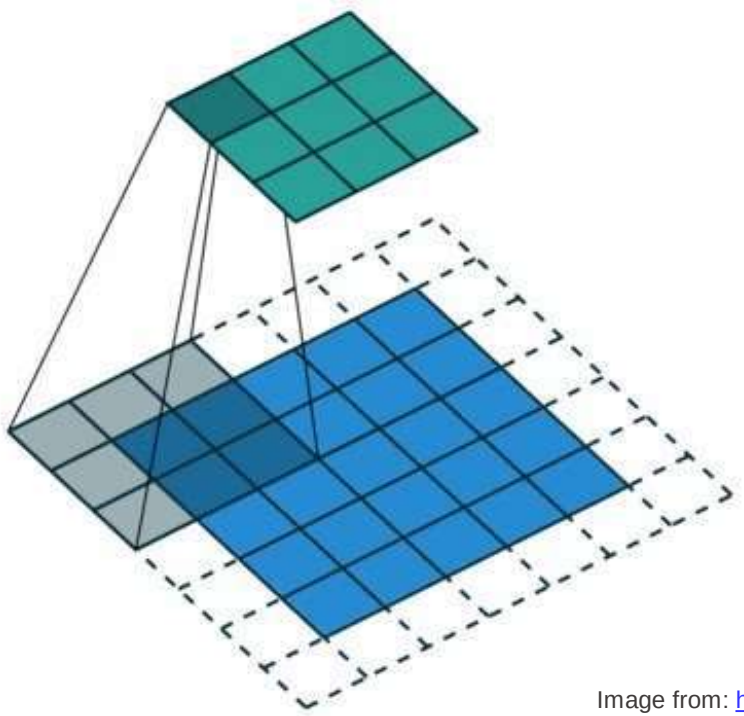
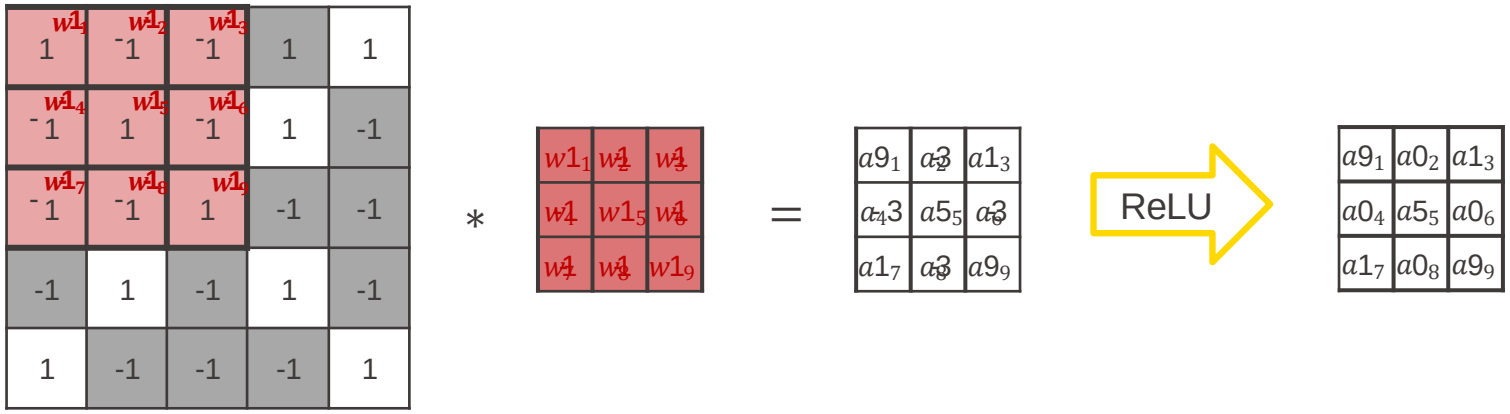


Image from: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

How Do Convolutional Layers Work?

- Idea:
  - Use a kernel / weight matrix and slide it over the image
  - At each position: Apply the convolution and a non-linear activation, e.g. ReLU



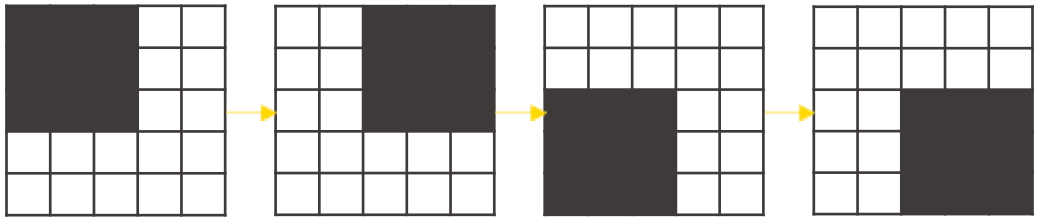
- The weights of the kernel are learned during training
- Note: These are similar steps like in a feed forward neural network
  - Convolution ≠ Weighted sum of inputs

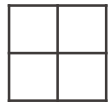
Stride

- Defines by how many pixels the kernel shifts

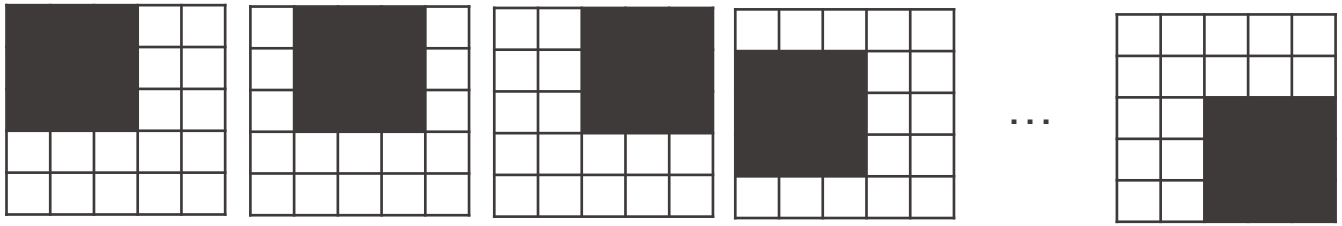
Output size

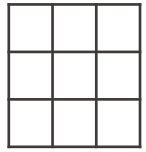
Stride of 2, 2





Stride of 1, 1

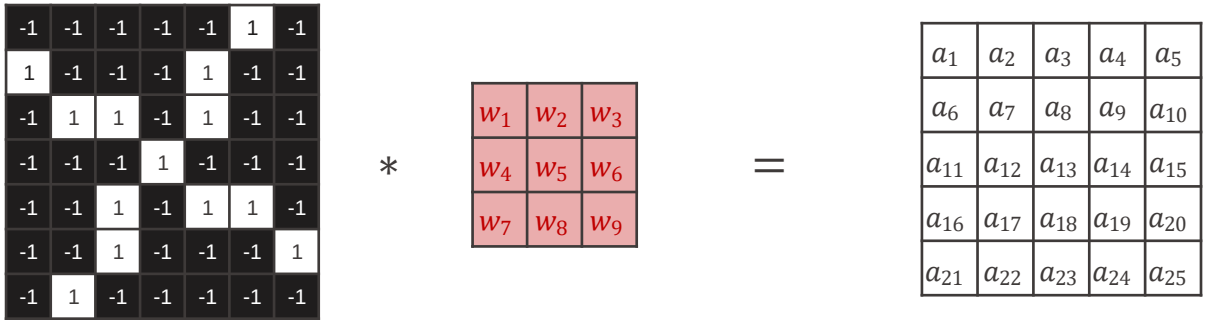




Note: The output dimension of a convolution for a n x n picture with a k x k kernel and a stride of s is $\left\lfloor \frac{n-k}{s} + 1 \right\rfloor$ x $\left\lfloor \frac{n-k}{s} + 1 \right\rfloor$

Motivation for Padding

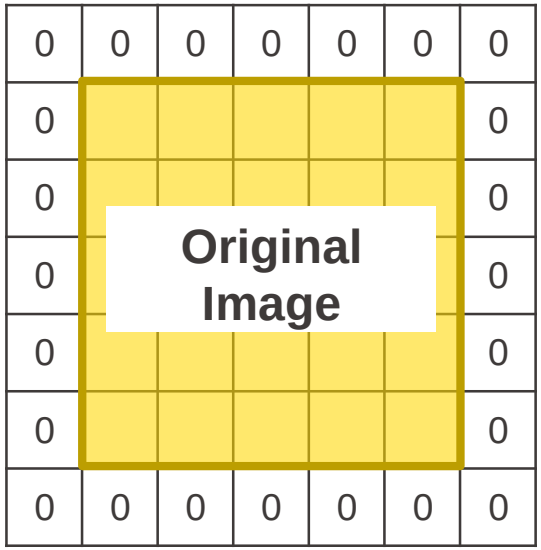
- Image shrinks with each conv layer, e.g., kernel of 3,3, and stride of 1,1



- Information from the edges is used less often (stride 2,2)

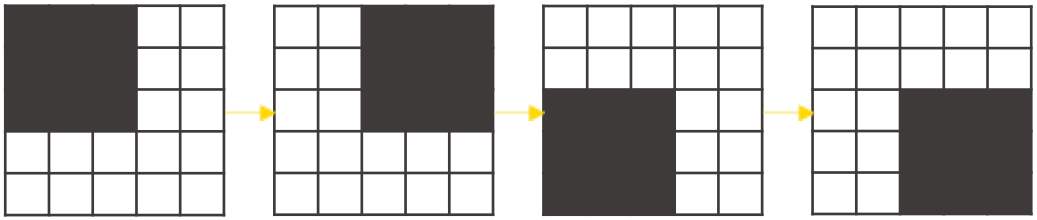
Padding

- Idea: Add additional cells with 0s
- Two common setting options
  - **Valid** means no padding is performed
  - **Same** means zero padding is performed, such that the output dimension of the feature map is the same as the input dimension.

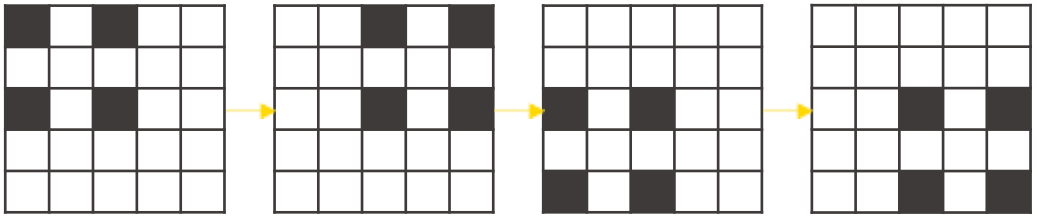


Dilation Rate

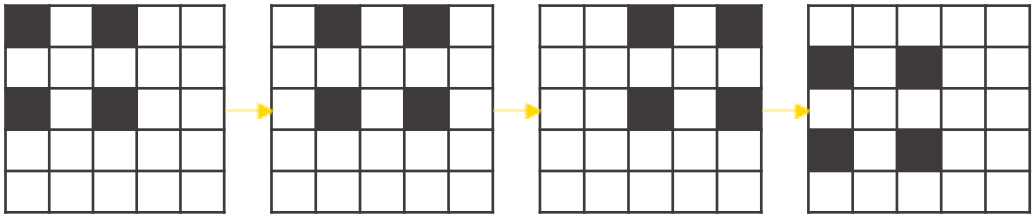
Stride = 2, 2 ; Dilation rate = 1, 1; Kernel 3 x 3



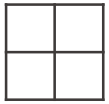
Stride = 2, 2 ; Dilation rate = 2, 2; Kernel 2 x 2

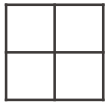


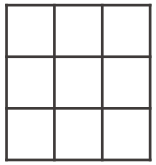
Stride = 1, 1 ; Dilation rate = 2, 2; Kernel 2 x 2



Output size

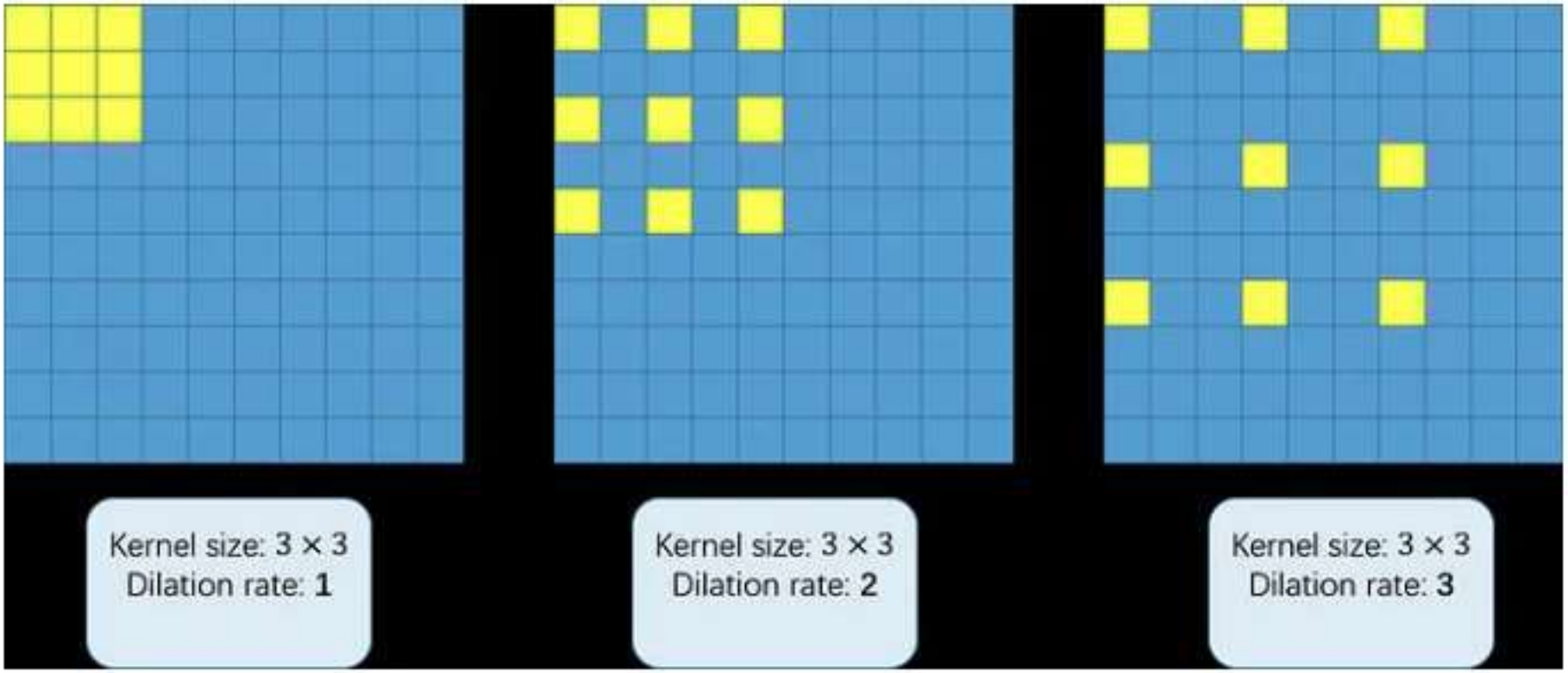




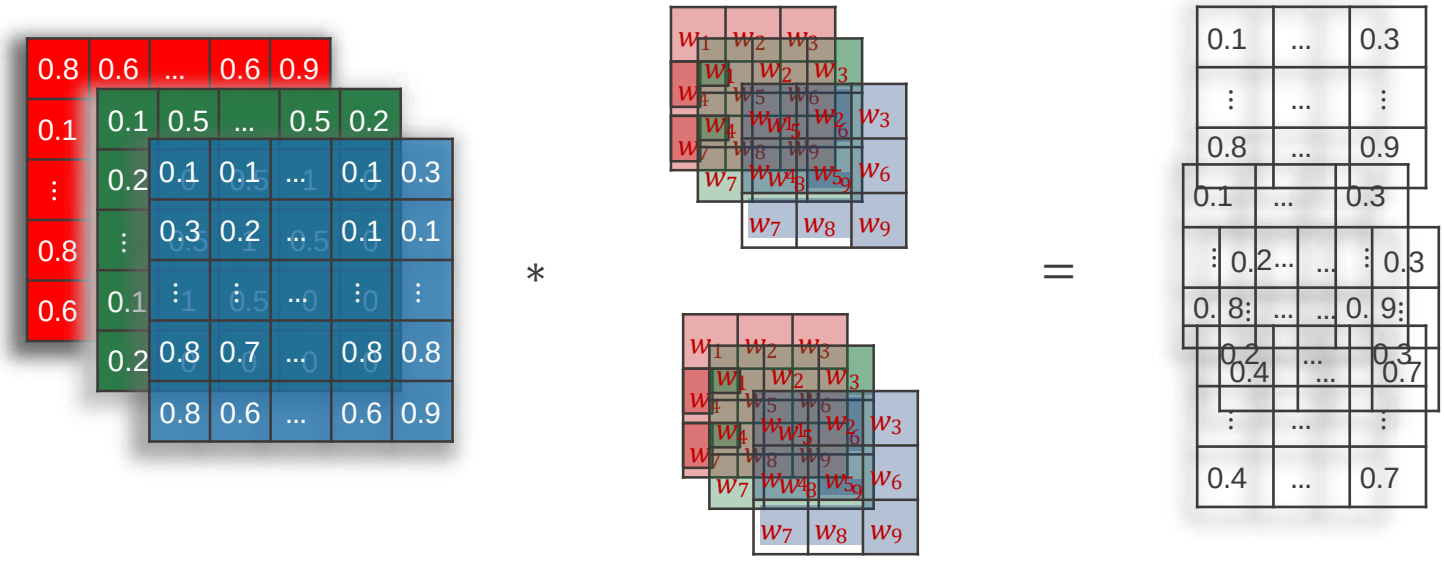


etc.

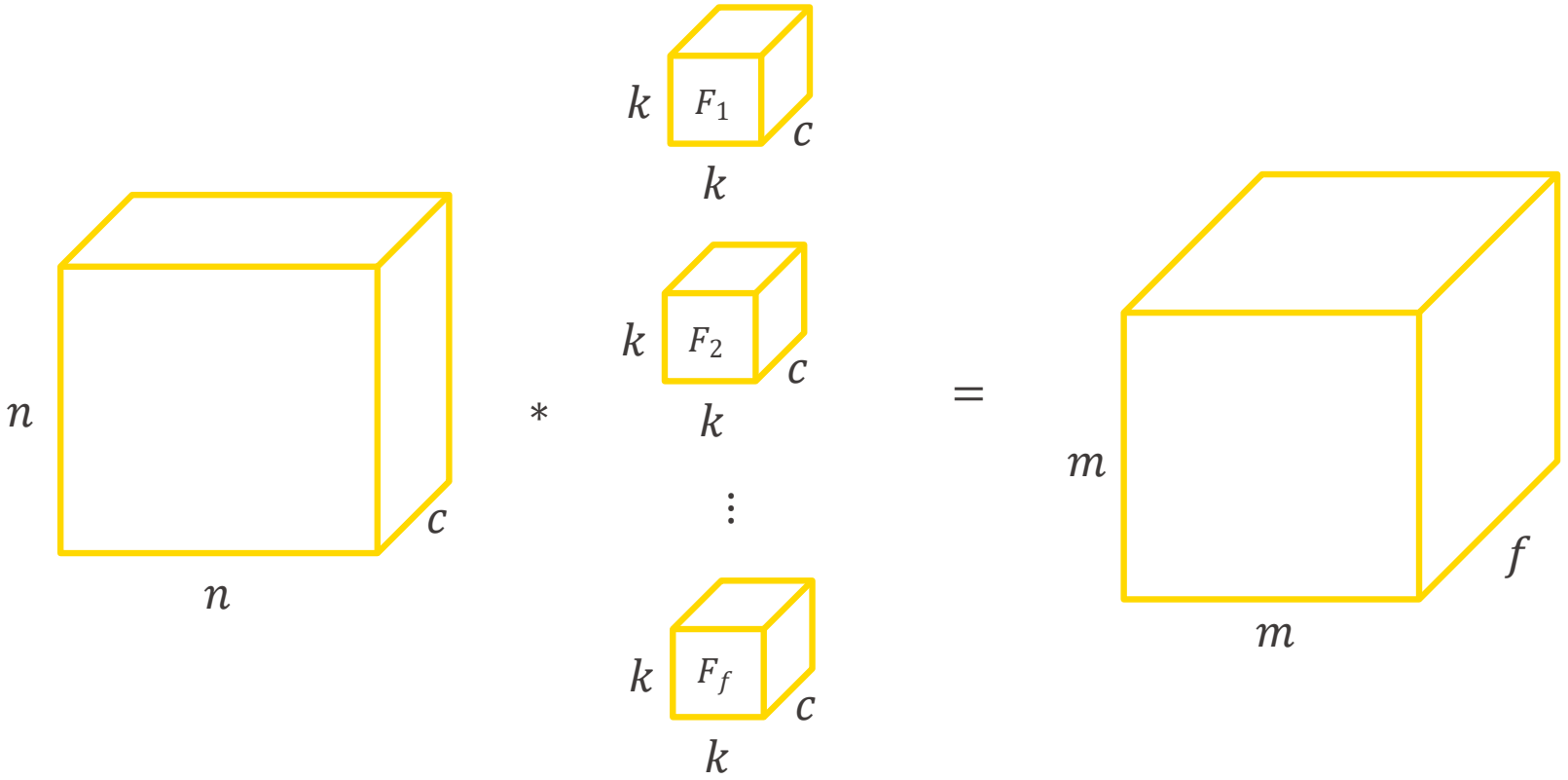
Dilation rate in convolutional neural networks is like adding space between the elements of your drawing tools, allowing you to see and work with larger areas of your drawing at once. It's a powerful technique for capturing and emphasizing larger patterns in the data, enhancing the ability of the network to understand complex structures and relationships within the input.



Training Many Filters for Many Dimension



An Often-used Alternative Visualization



n x n x c

k x k x c, f times

m x m x f

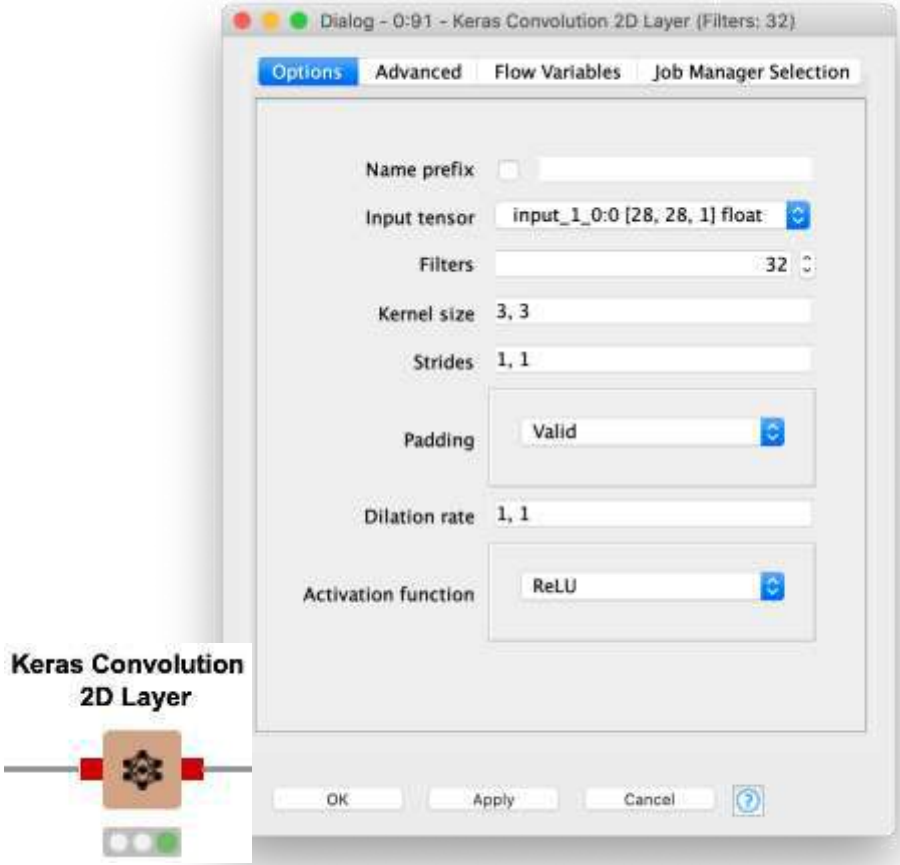
Layers used to build ConvNets

As we described above, a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet **architecture**.

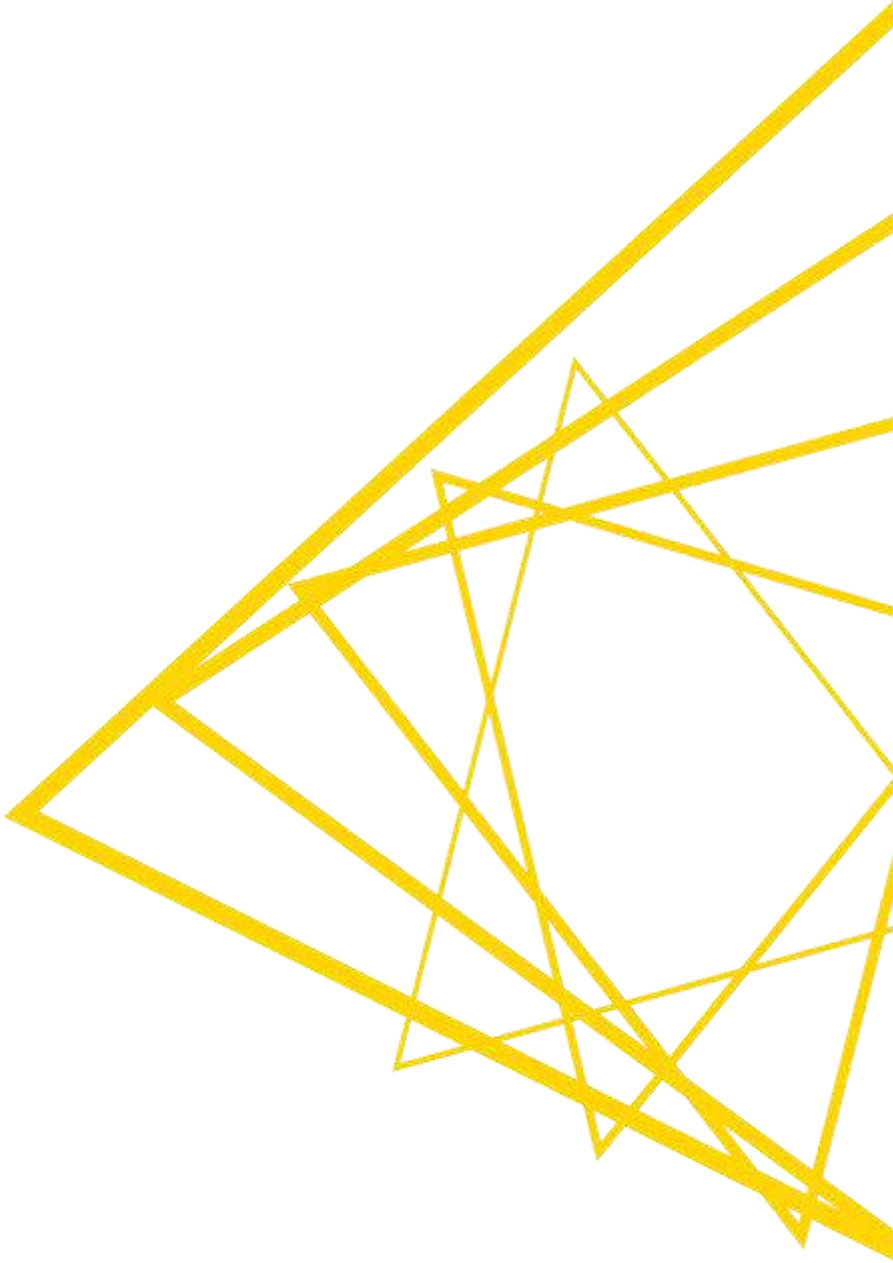
Example Architecture: Overview. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail:

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the $\boldsymbol{max(0, x)}$ thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

Keras Convolutional 2D Layer

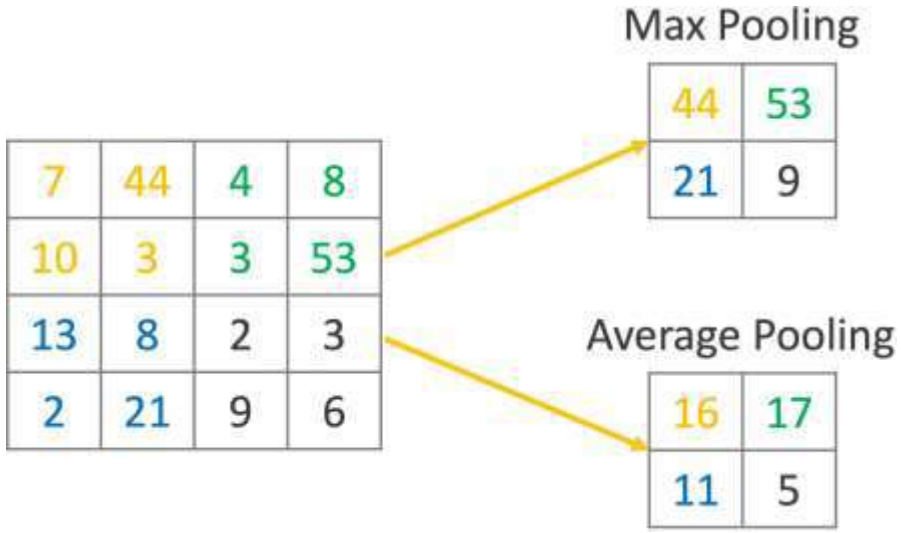


Pooling Layer

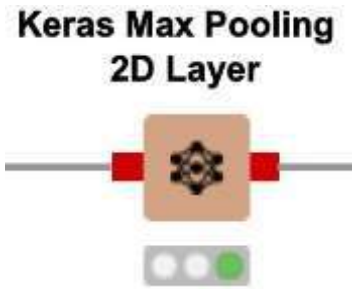


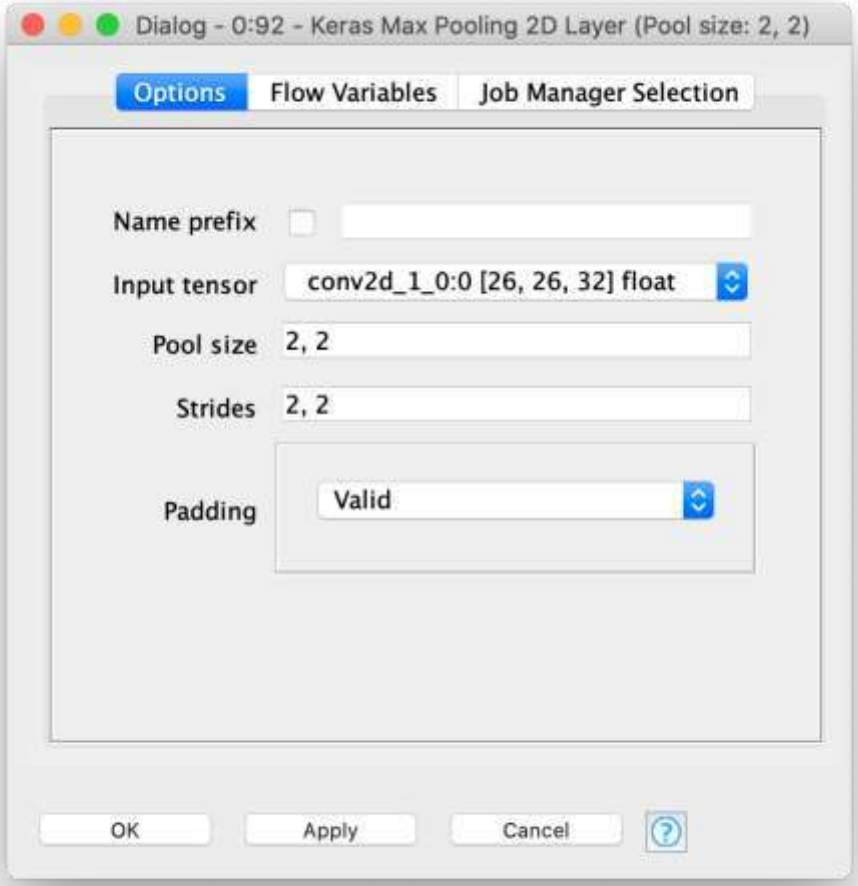
Pooling Layer

- Idea: Replace the area of an image or feature map with a summary statistic.
- Example: Replace each 2x2 area with the
  - Maximum value (Max pooling)
  - Mean value (Average pooling)
- Pooling layers are often used in between convolutional layers to
  - Increase the receptive field of the following layers
  - Reduce computational complexity
- No parameters to learn



Keras Max Pooling 2D Layer Node





CNN for Image Classification

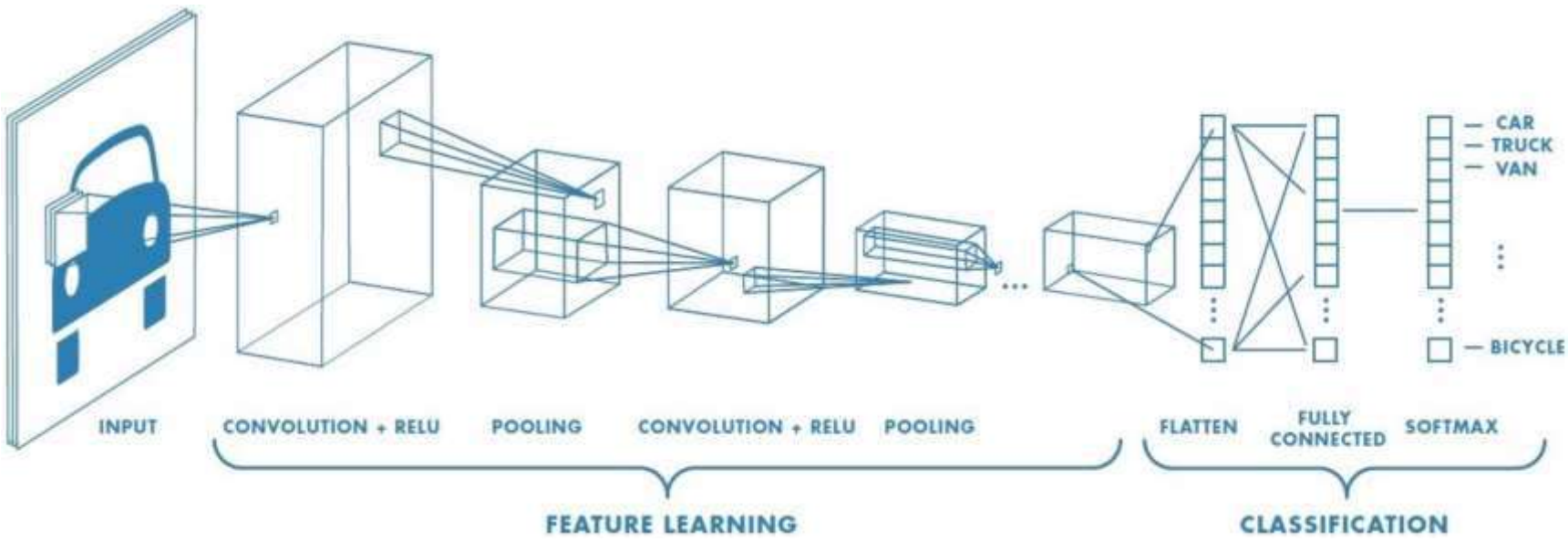


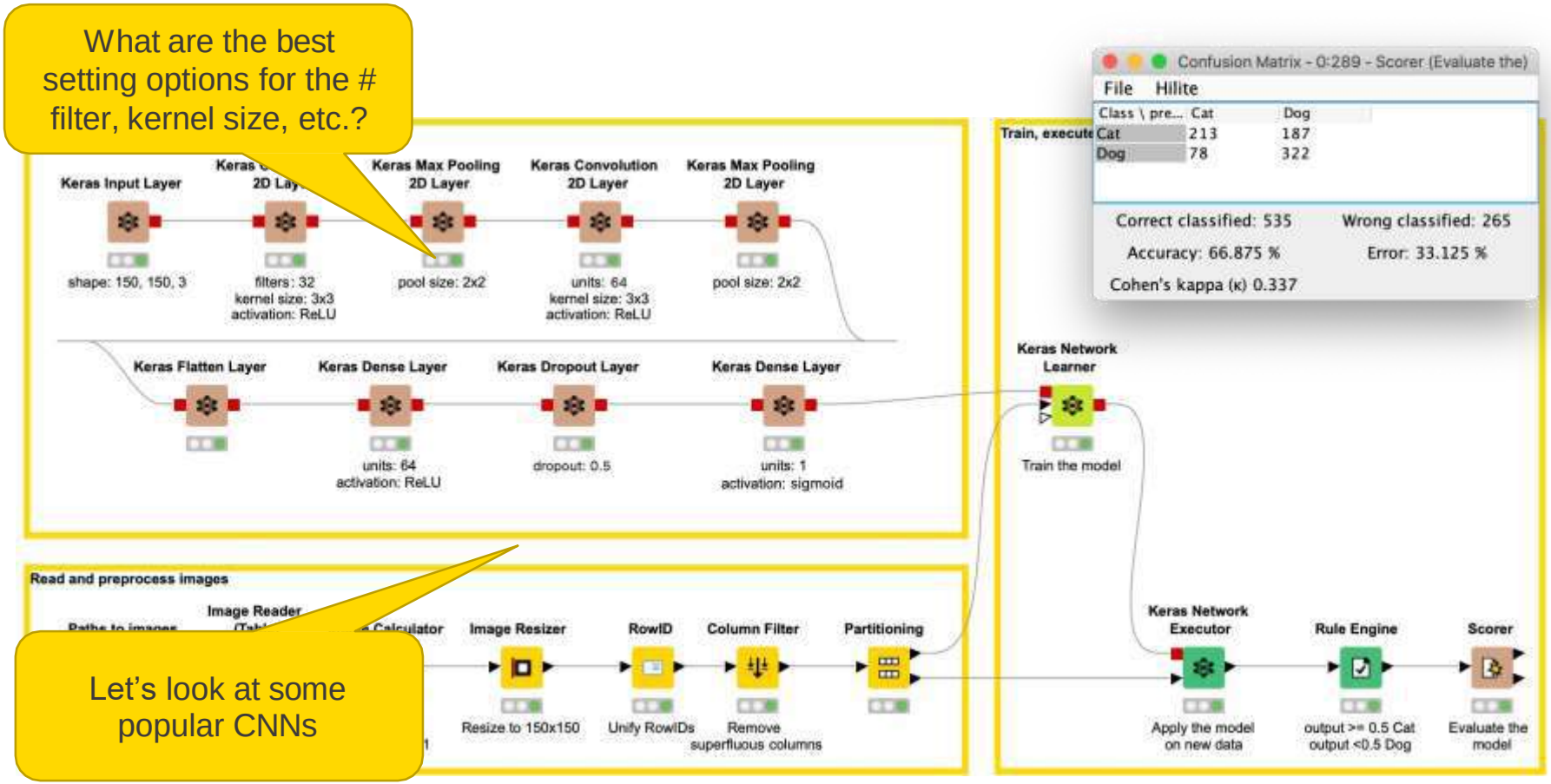
Image from: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Image Classification: Cats & Dogs Data

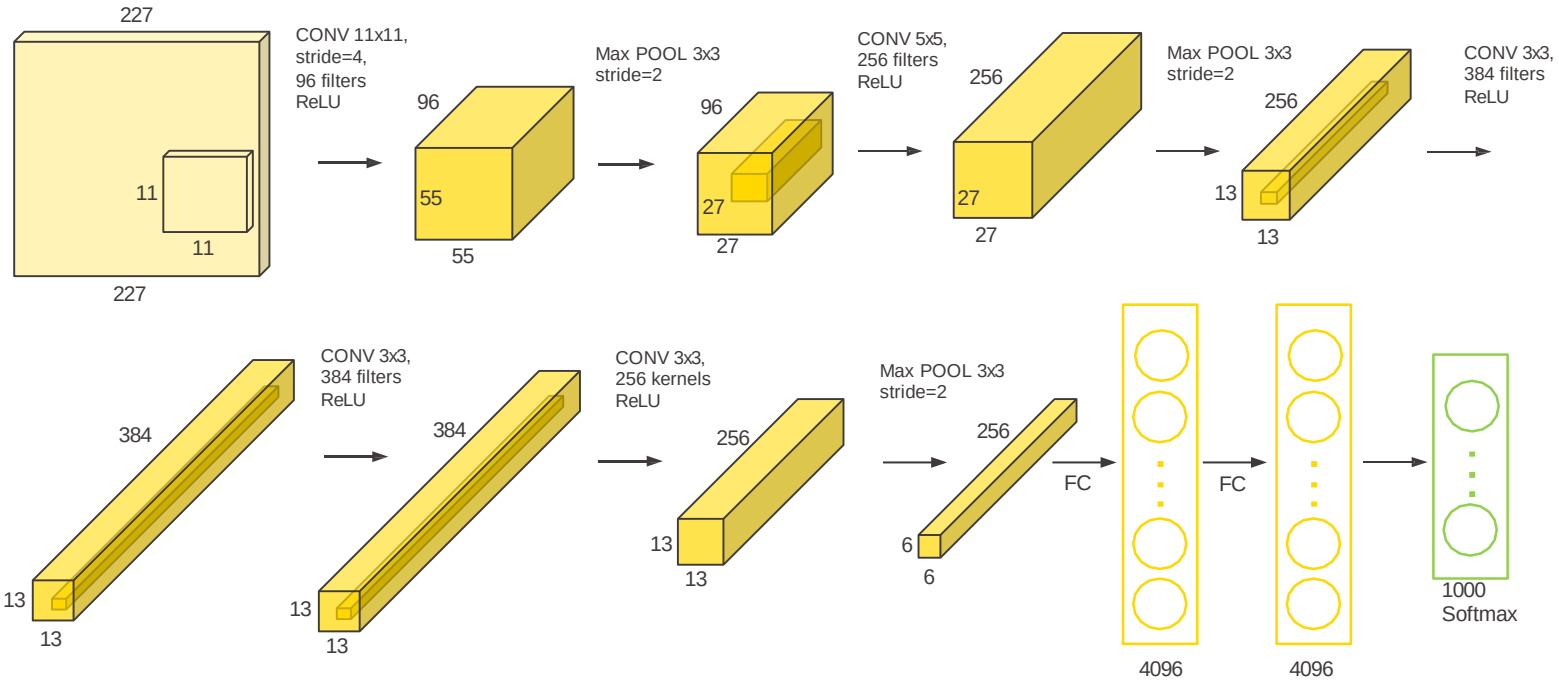



https://www.kaggle.com/c/dogs-vs-cats/overview

Simple CNN for Image Classification



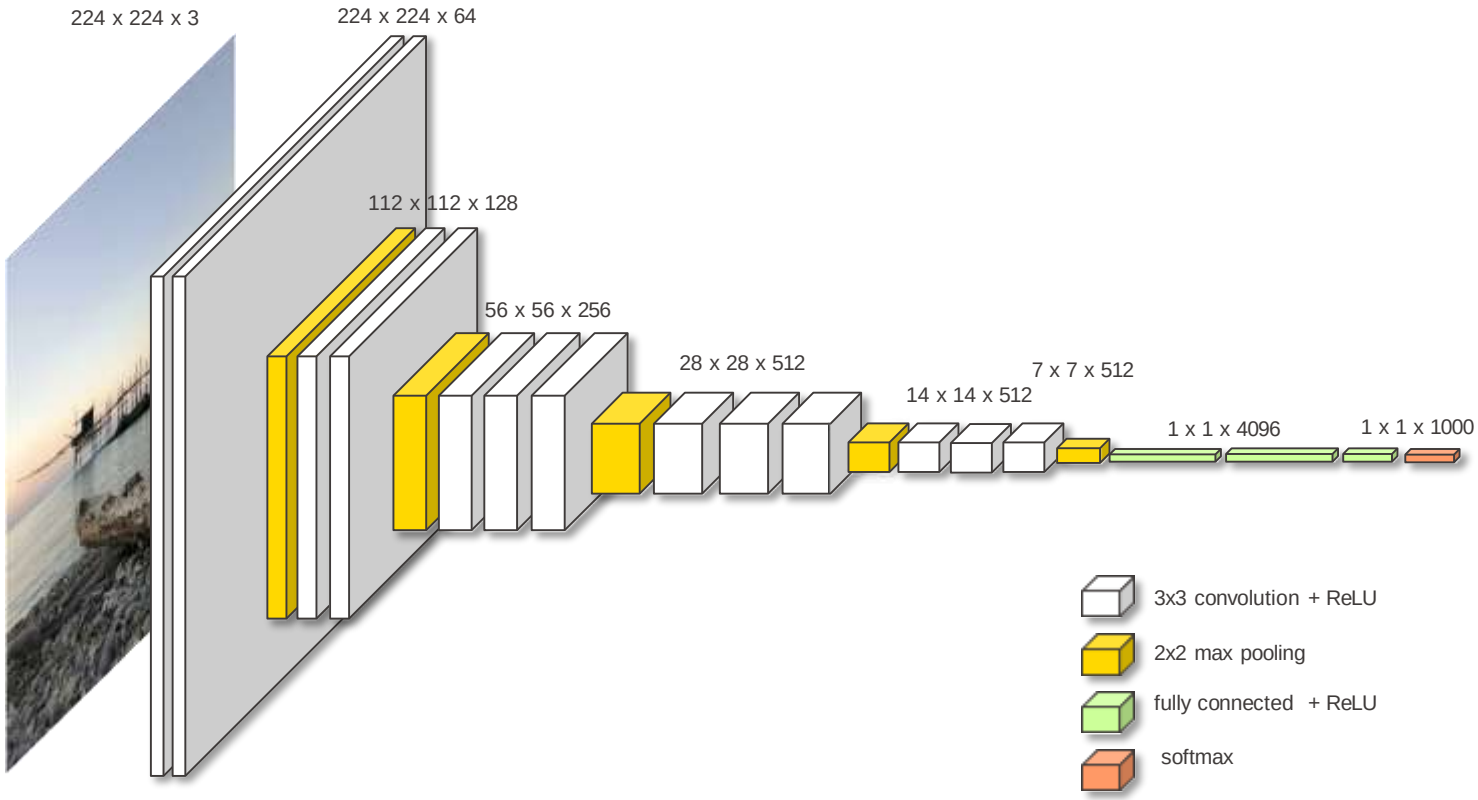
Popular CNNs: AlexNet (2012)



Source: https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

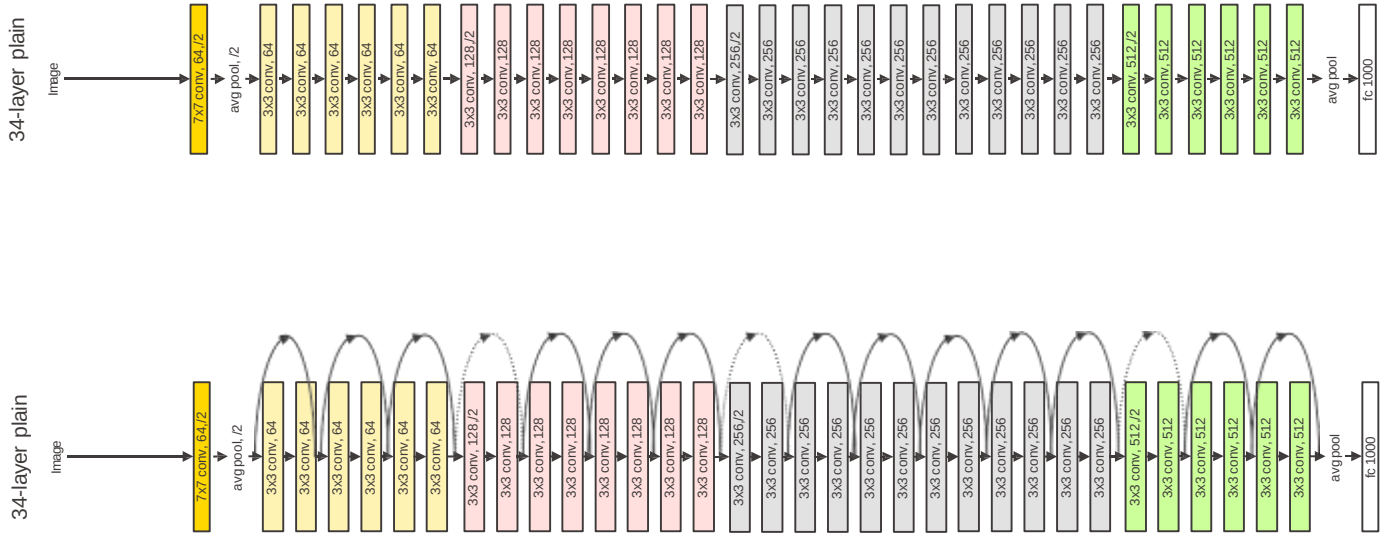
Paper: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Popular CNN: VGG-16 (2015)



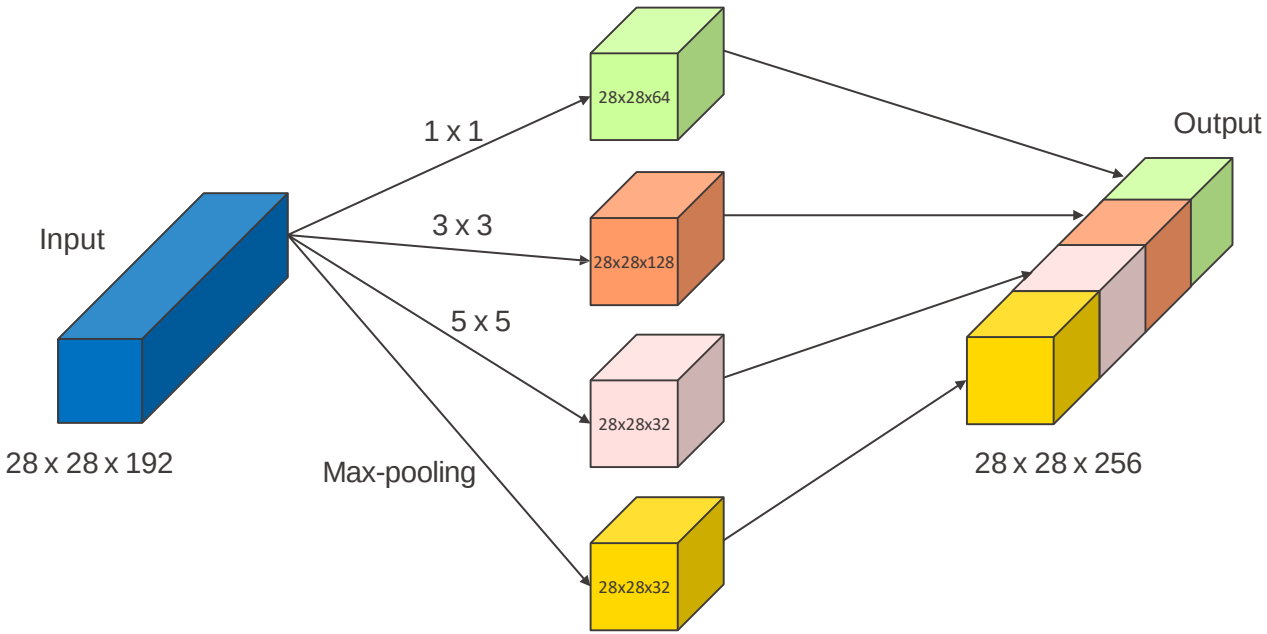
Outlook: Residual CNNs

- ResNet
  - Problem: Really deep CNNs perform worse than deep CNNs
  - Idea: Use skip connections

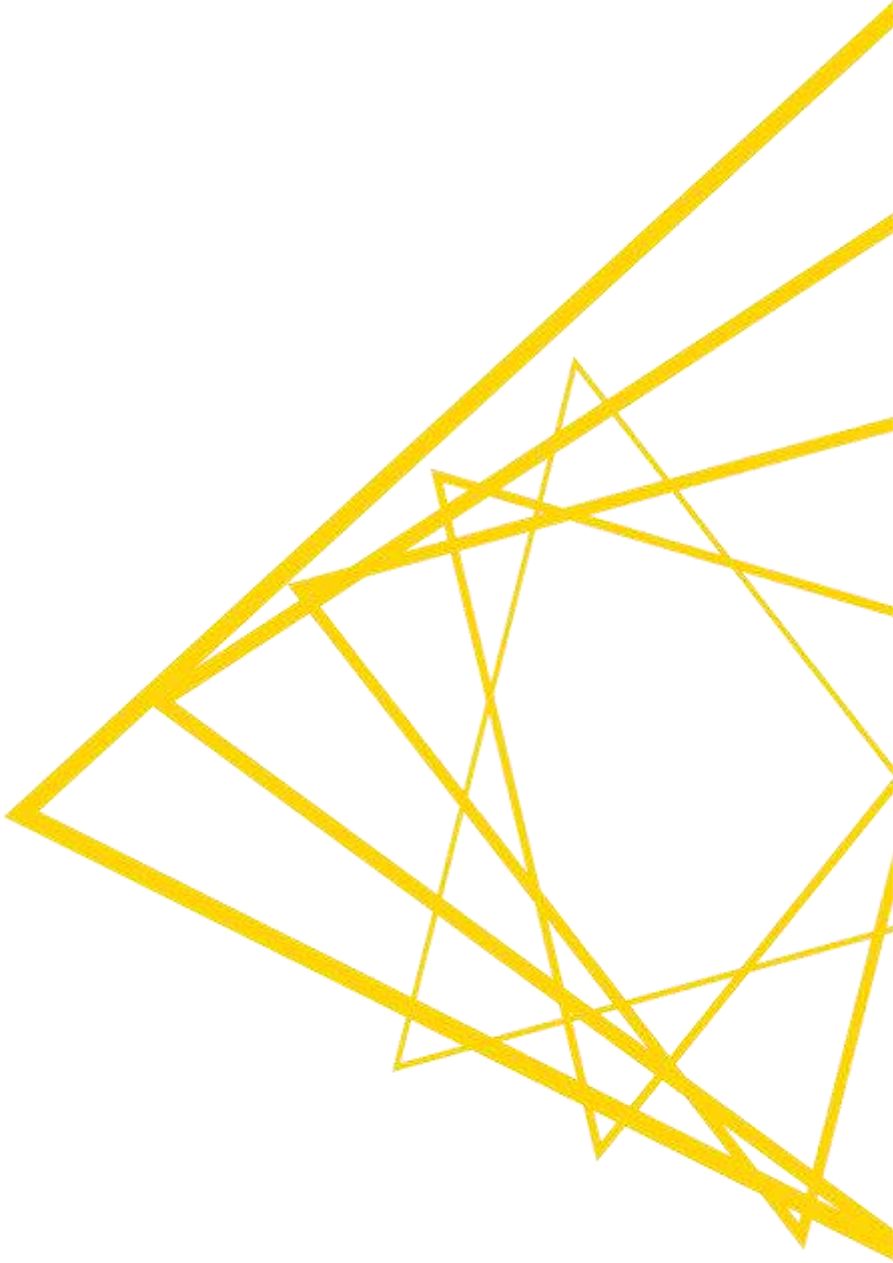


Outlook: Inception Network

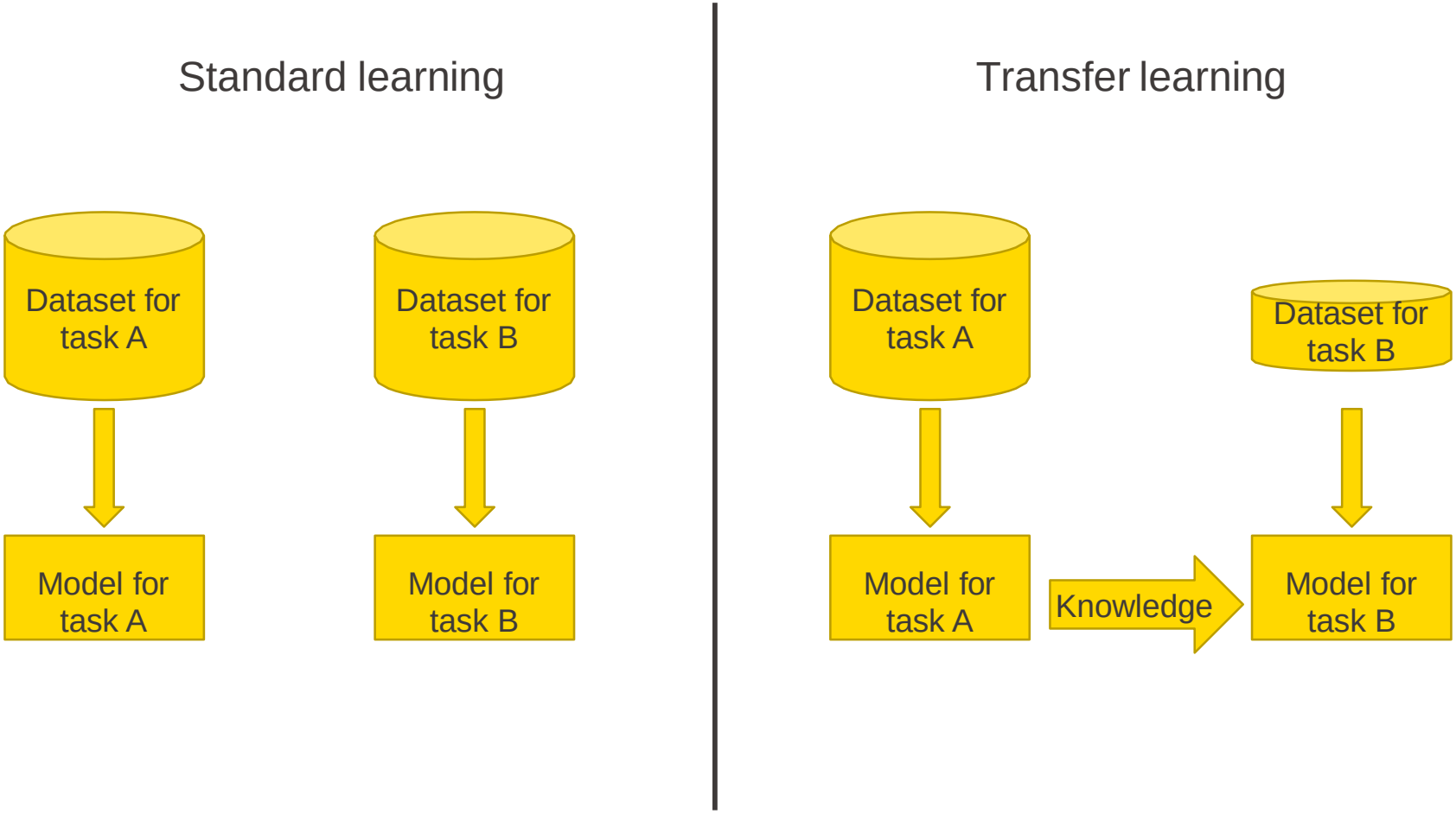
- Problem: Many design options to choose from, kernel size, stride, pooling, etc.
- Idea: Use different layer types at the same time and combine them



Transfer Learning

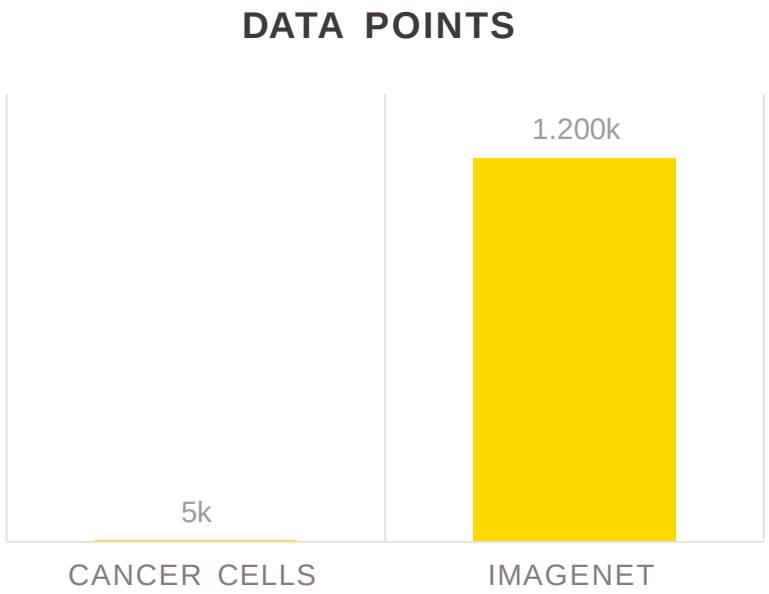


Standard vs. Transfer Learning

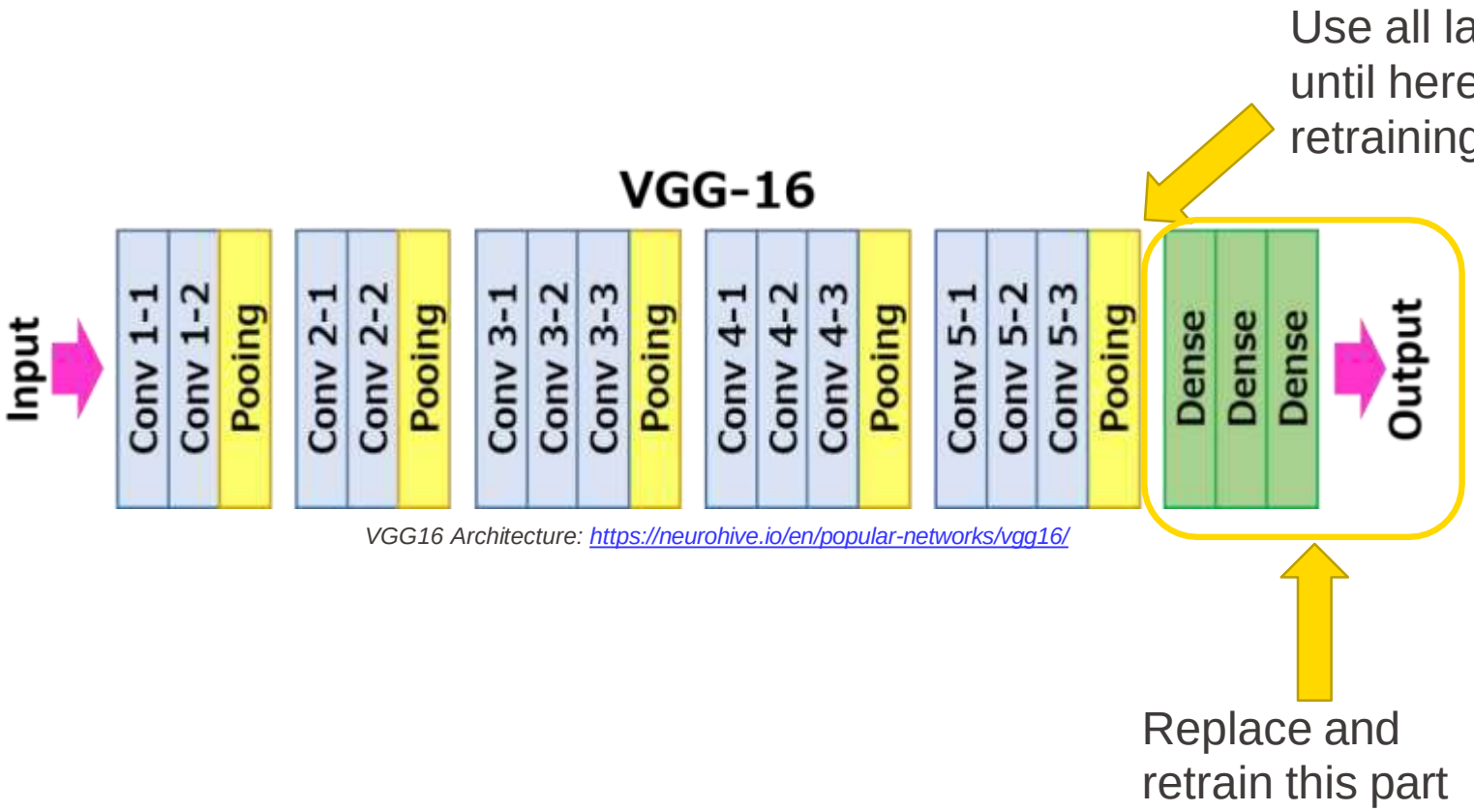


Transfer Learning for Image Classification

- Idea: Reuse existing architecture, pretrained on a similar task
  - E.g., use VGG16 as starting point to solve the cats and dogs classification
- Many ways how the trained network can be used
  - Reuse only network structure
  - Reuse network structure and weights and
    - retrain only some layer
    - retrain all layers
    - add some layers on top
- Why is it helpful?
  - Image classification requires tons of data
    - Often not available
  - VGG16 was trained on more than 1,000,000 images from ImageNet dataset.

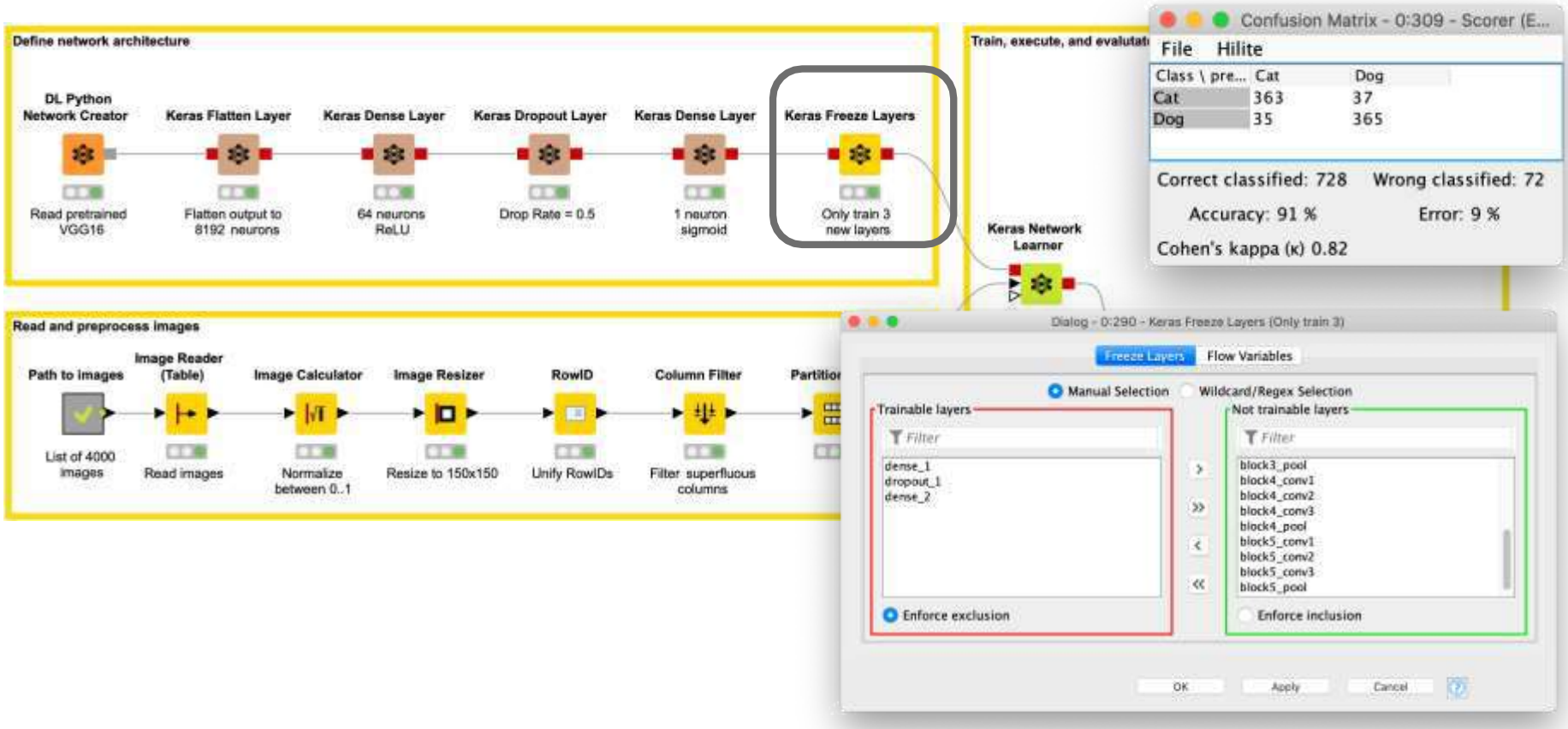


Transfer Learning for Image Classification



VGG16 Architecture: https://neurohive.io/en/popular-networks/vgg16/

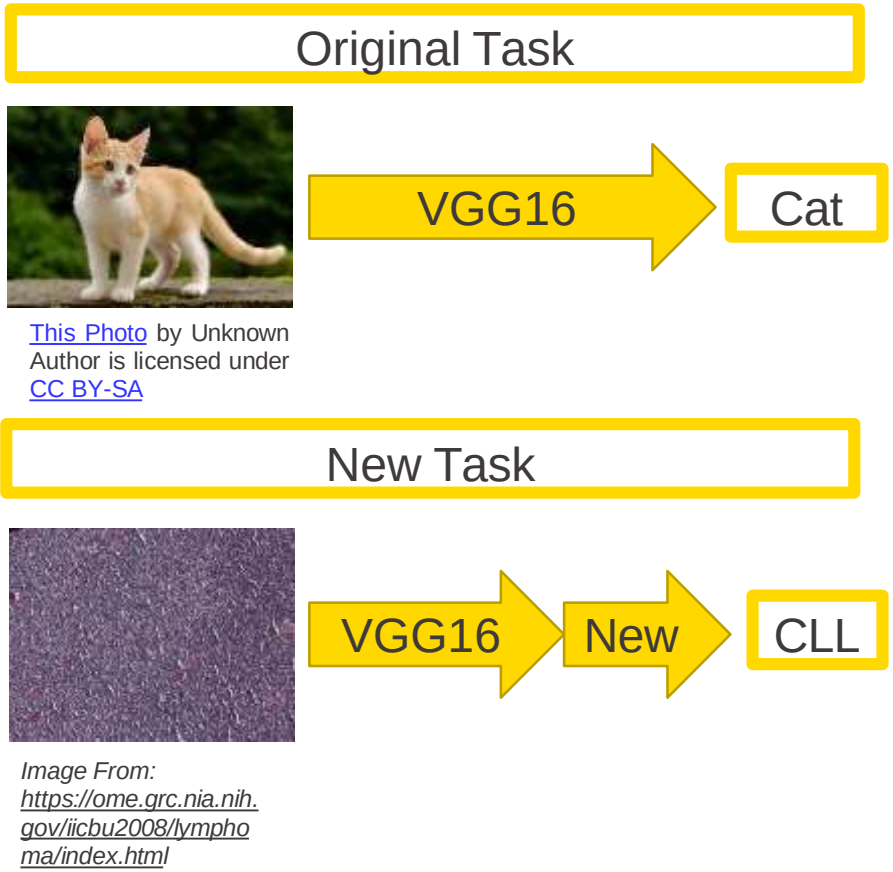
Transfer Learning for Image Classification



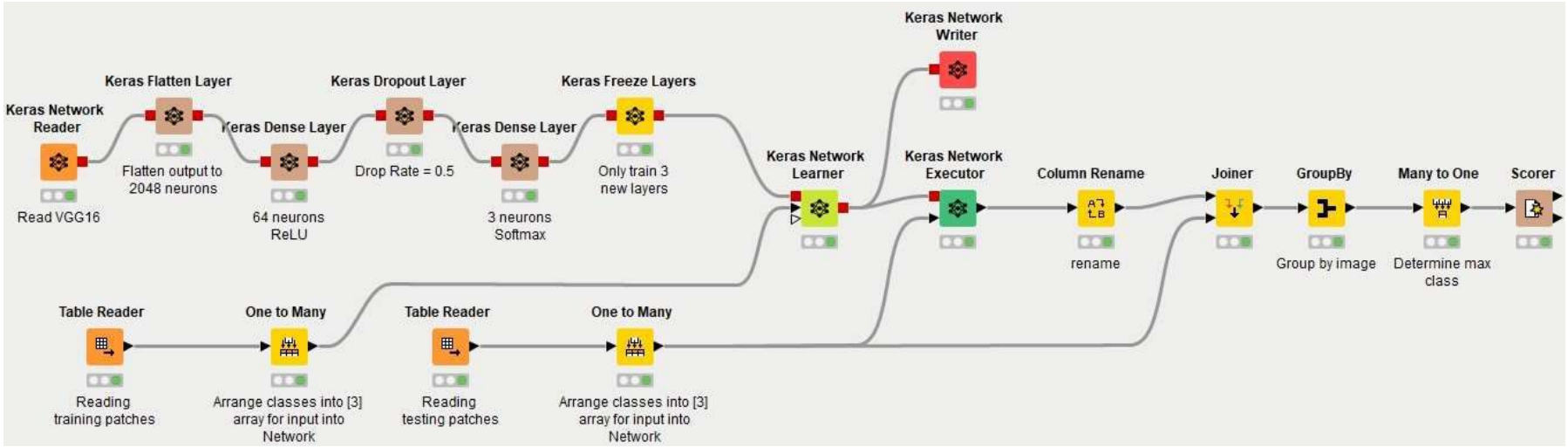
Transfer Learning for Image Classification: Option 2

Cancer Cell Classification with Transfer Learning

- Transfer learning can be adapted to a wide range of image classification problems
- Task: Classify histopathology slide images and about the type of lymphoma
  - chronic lymphocytic leukemia (CLL)
  - follicular lymphoma (FL)
  - mantle cell lymphoma (MCL)
- Reuse VGG16 network

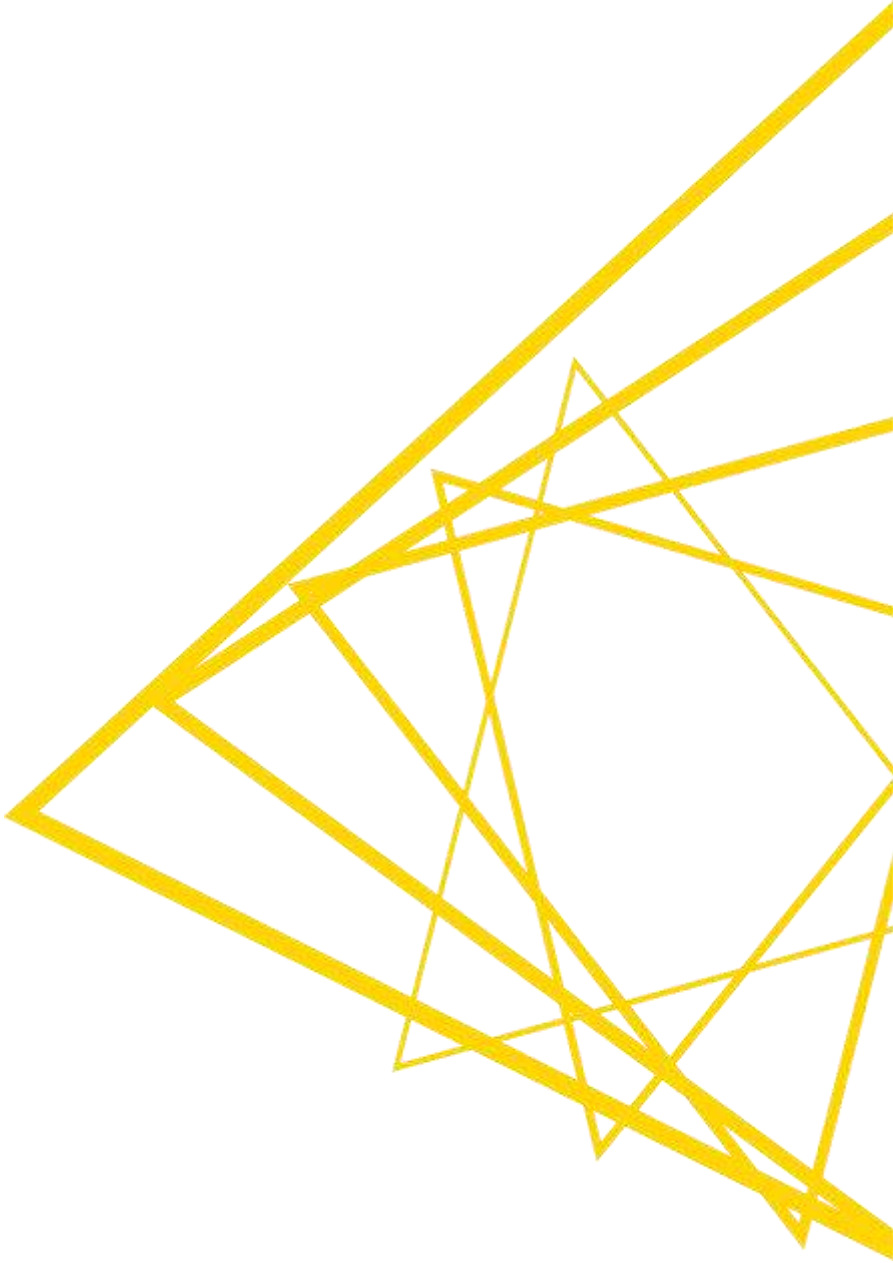


Transfer Learning with KNIME Analytics Platform



Workflow available on the KNIME Hub: https://kni.me/s/yMp8GBkT0Xwzx5X2

Other Use Cases for CNNs



Semantic Segmentation

Encoder-decoder Structure

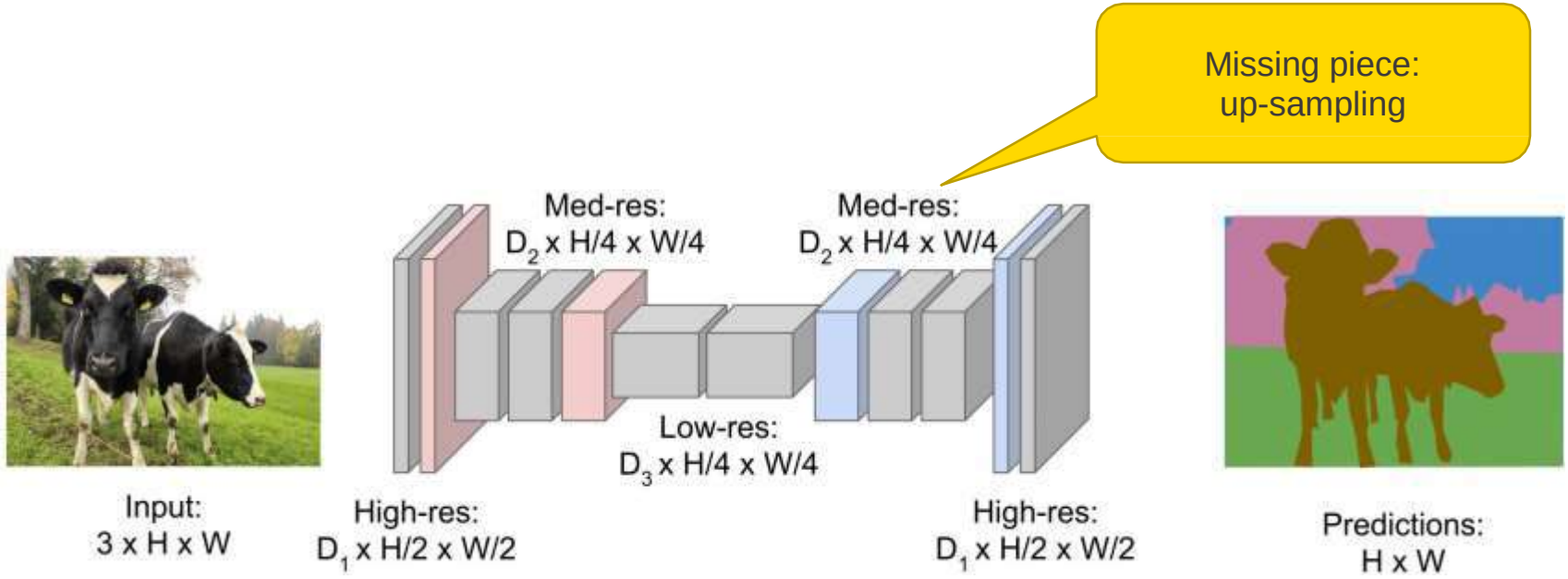


Image Source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Upsampling? Transpose Convolution

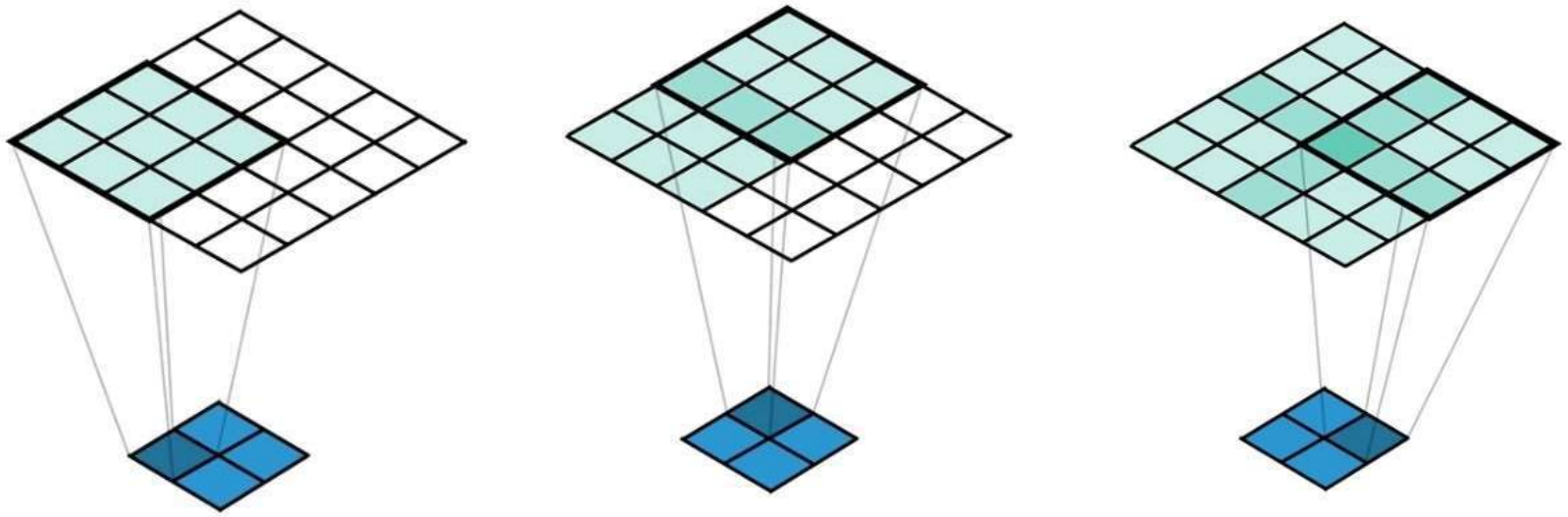
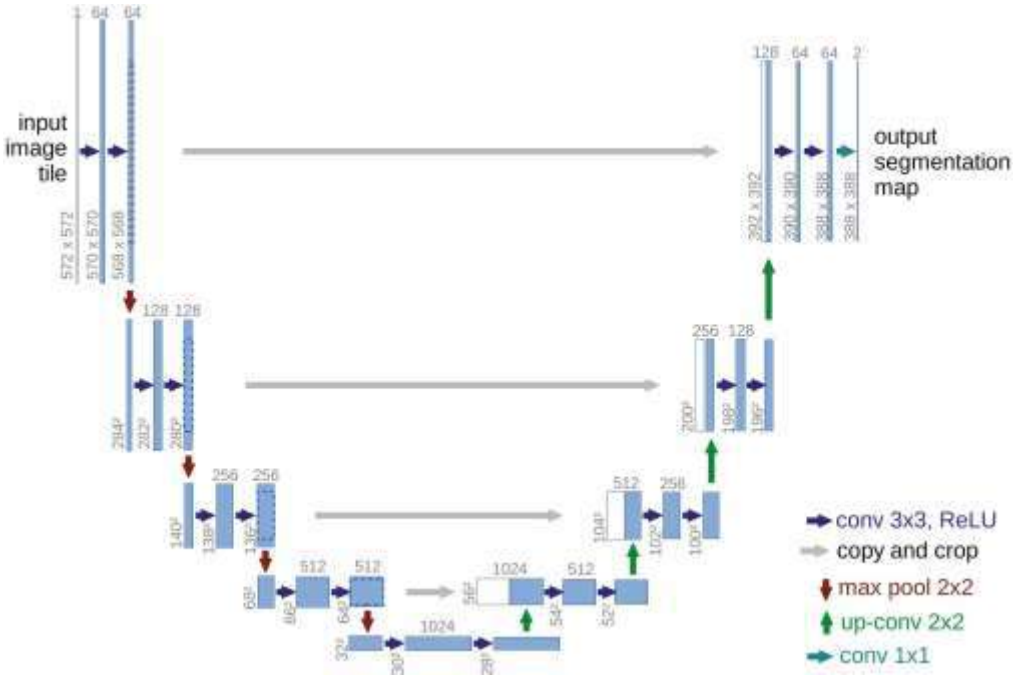
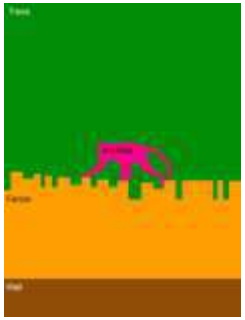


Image Source: https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

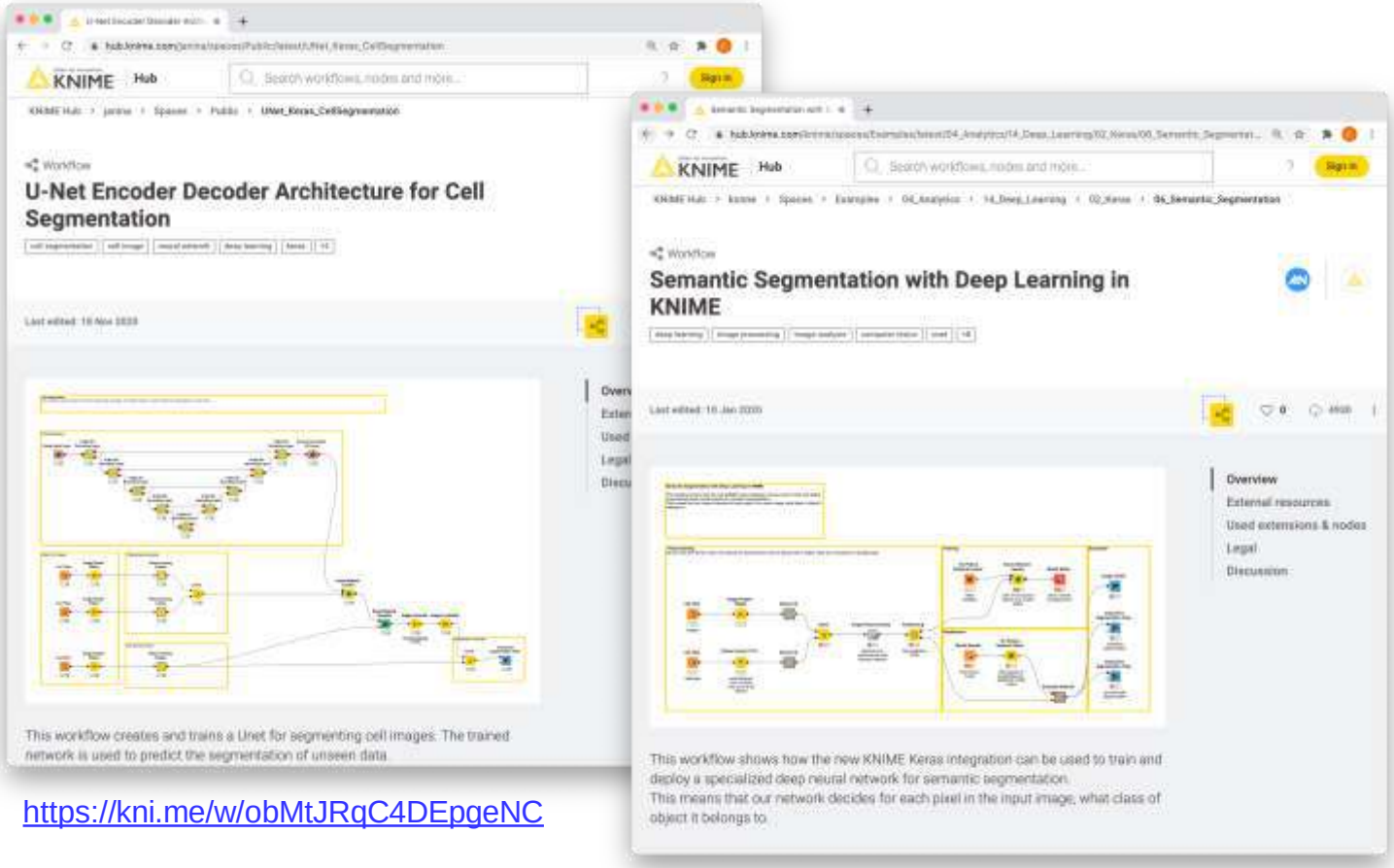
U-Net





Ronneberger et al.
https://arxiv.org/pdf/1505.04597.pdf

Example Workflows for Semantic Segmentation

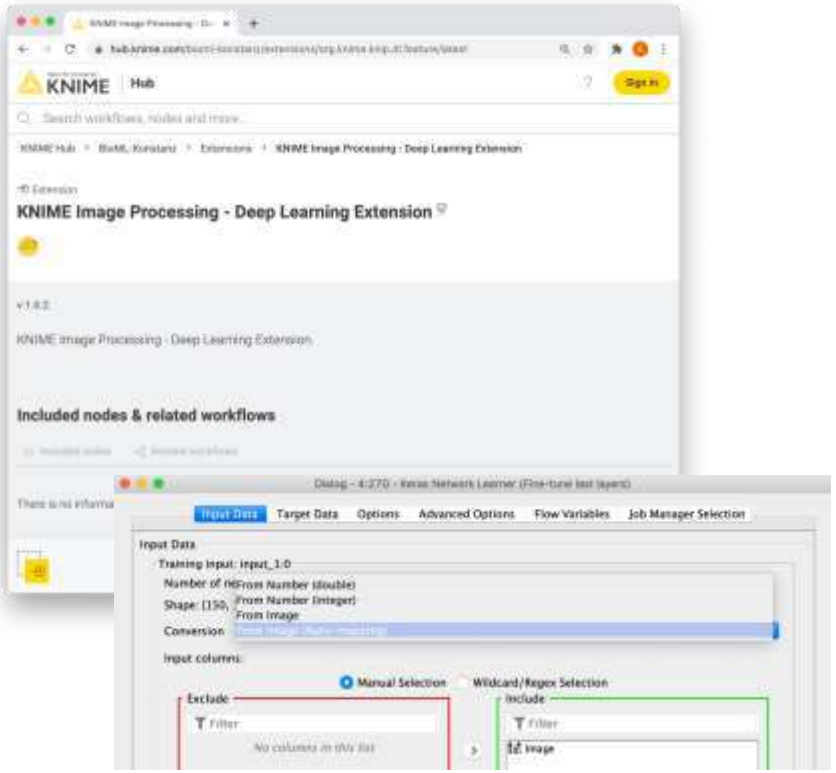


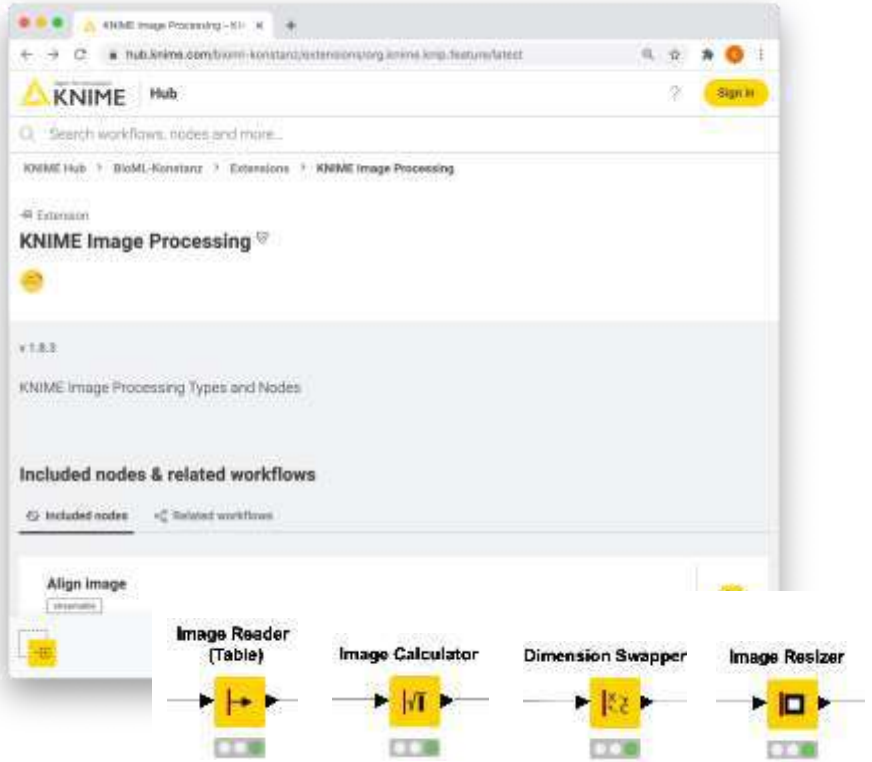
https://kni.me/w/obMtJRqC4DEpgeNC

https://kni.me/w/kRZQcv4e7-qoGQkU

Working with Images in KNIME Analytics Platform

Extension

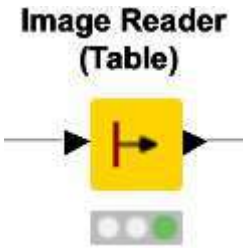




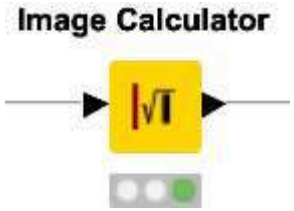
https://kni.me/e/JMlIEafbwxOPn652

https://kni.me/e/Uq6QE1IQIqG4q_mp

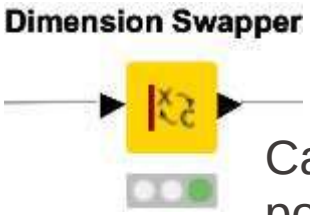
Helpful Nodes when Working with Images



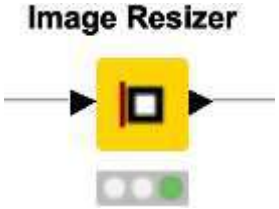
Reads images from various file formats based on a table with file paths or URLs.



Evaluates a mathematical expression based on the images. Can be used to normalize images, e.g. Image / 255

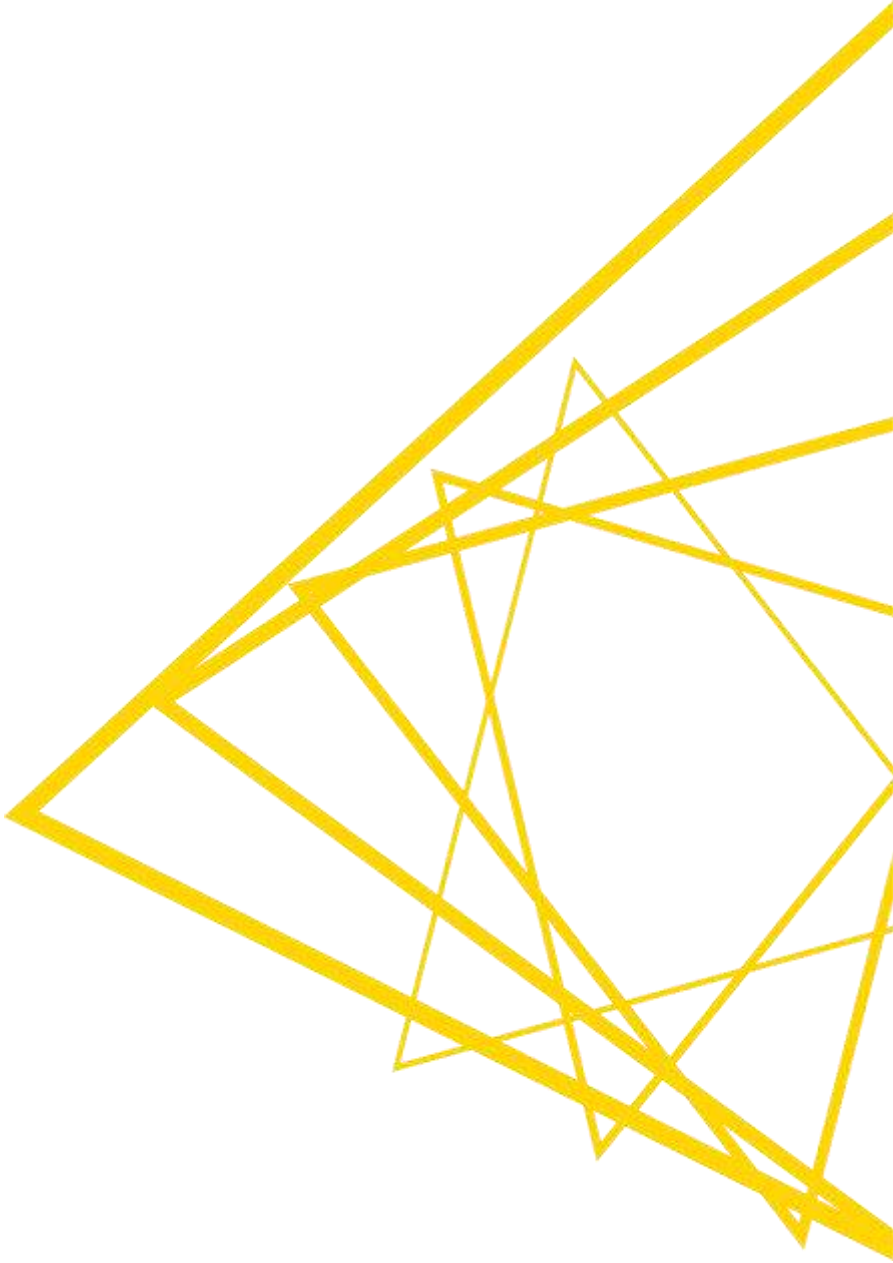


Swaps the dimensions of an image.
Can be used to change the position of the channel.



Resizes the image in each dimension. Different resize strategies are available, e.g. linear interpolation.

Combining RNN and CNN Image Captioning



What is Image Captioning?

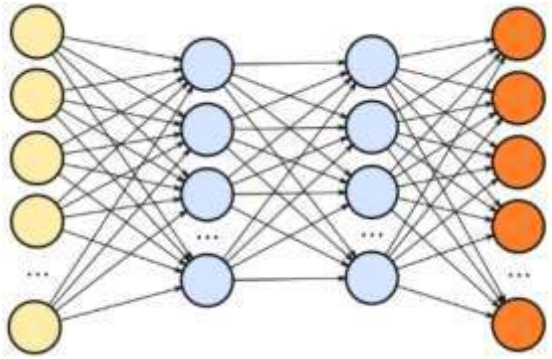
Task:
Describe the contents of an image

Example captions:

- A fancy dessert on a plate with a twisted orange.
- A plate has a dessert and orange slices on it.
- Some ice cream sitting next to some orange slices

…

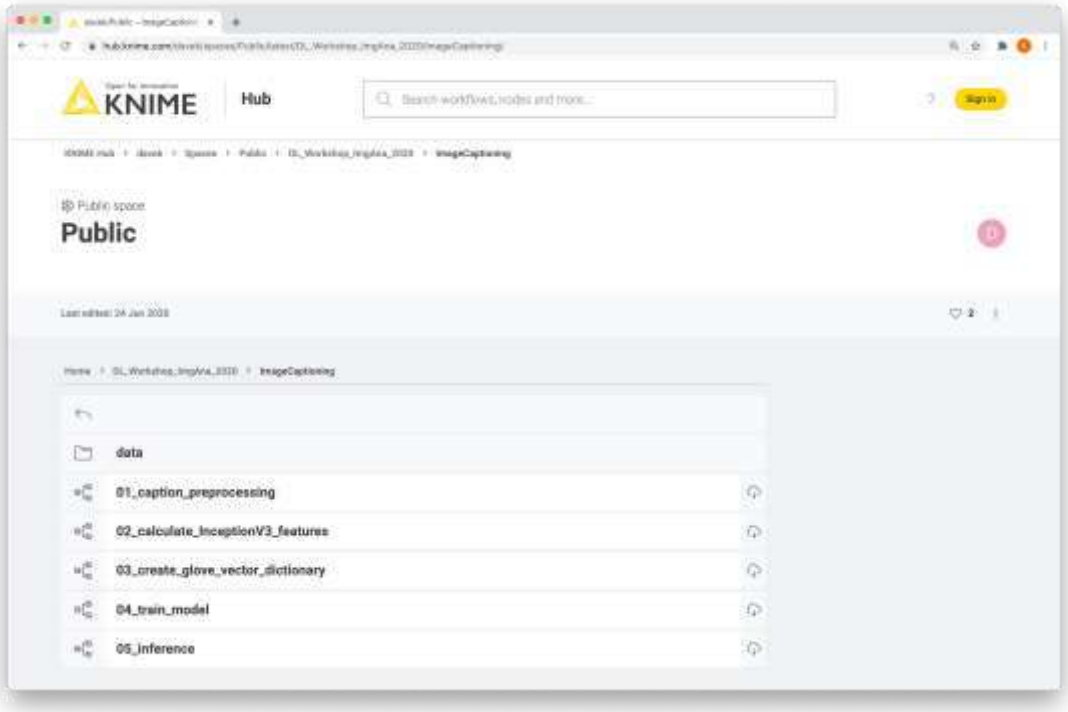
Image Captioning with Deep Learning



'A fancy dessert on a plate with a twisted orange.'

Does this work?

Yes – Workflows Available on the KNIME Hub



- Workflow: https://kni.me/s/RpxRWvlX1wsH0clx
  - Note: Sign in to download the whole workflow group
- Webinar: https://www.youtube.com/watch?v=_Fmkqwrt_Gk&t=4400s

Exercise: MNIST Image Classification

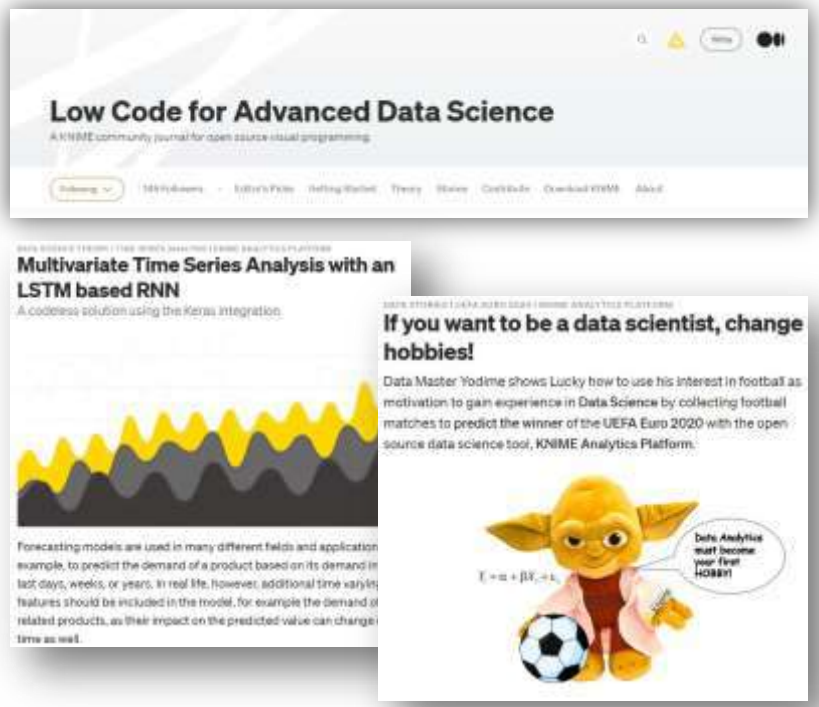
- Dataset: MNIST dataset, which contains images of handwritten digits as well as the corresponding number.
  Source http://yann.lecun.com/exdb/mnist/
- Goal: Train a CNN able to recognize the correct digit given the picture of the handwritten digit.

Stay Up-To-Date and Contribute

- Follow the KNIME Community Journal on Medium
  [Low Code for Advanced Data Science]

- Daily content on data stories, data science theory, getting started with KNIME and more
  for the community by the community

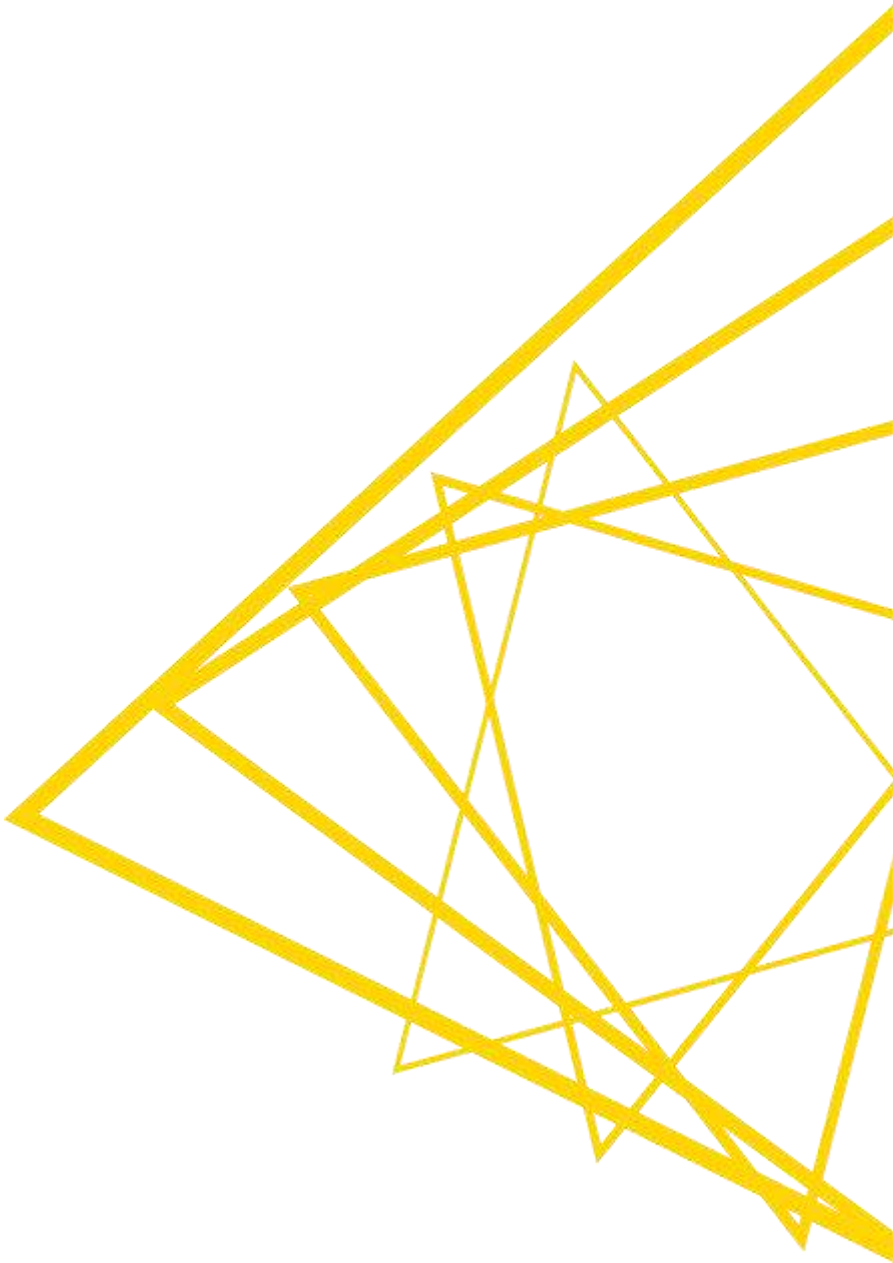
- Would you like to share your data story with the KNIME community?



Contributions are always welcome!

Open for Innovation
KNIME

Thank You!



Thank you for your attention