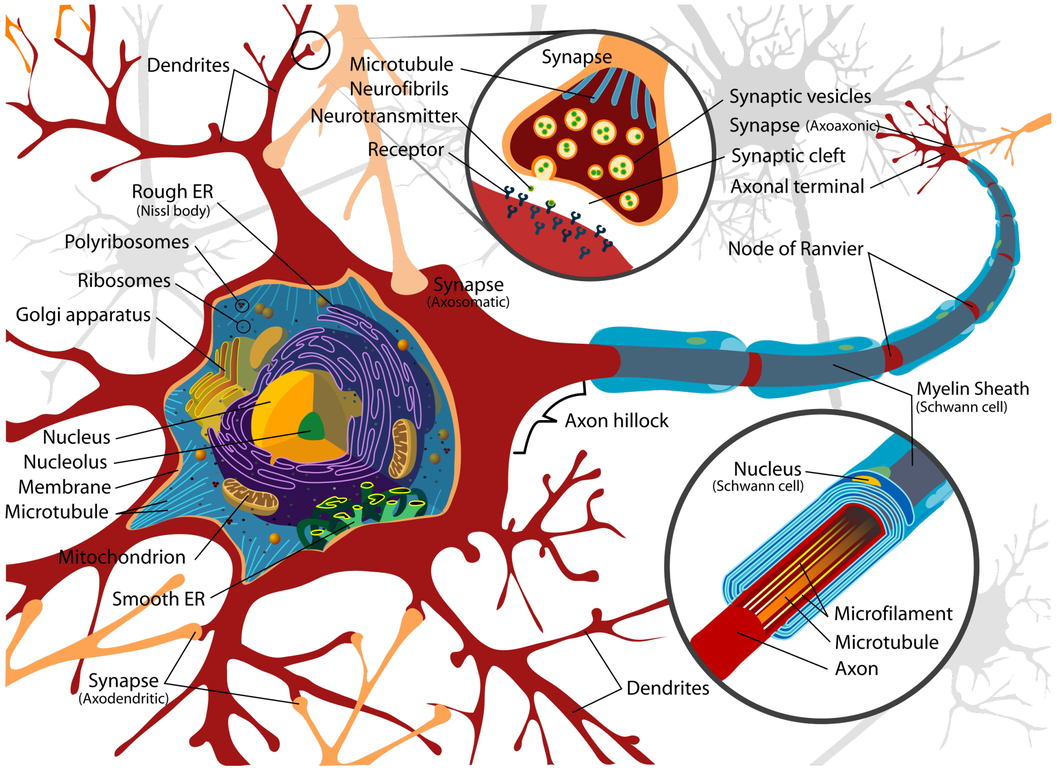
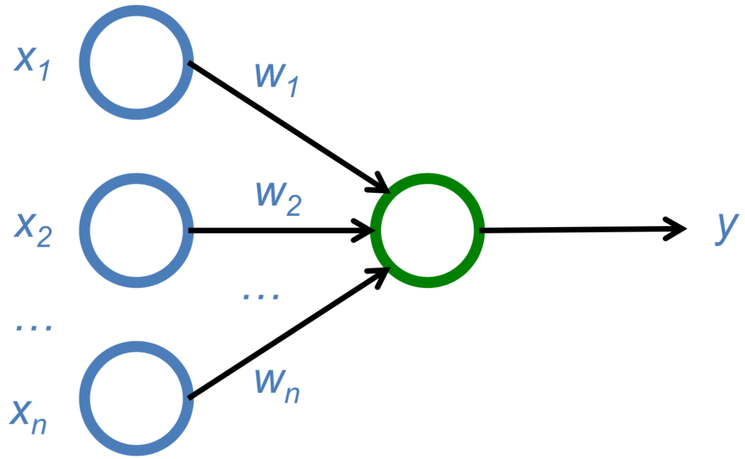
Deep Learning Theory

Dr. Claus Horn,
HSLU Deep Learning Bootcamp

Biological vs artificial Neuron



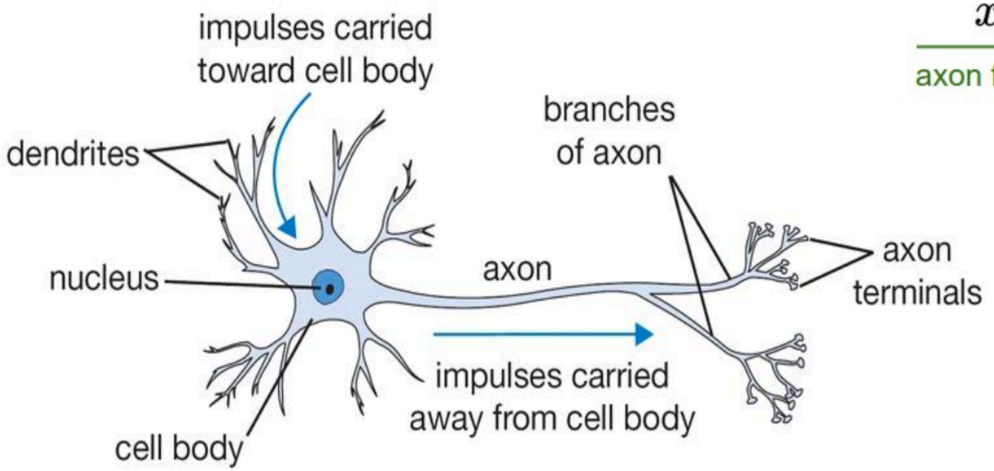
Source: https://en.wikipedia.org/wiki/Neuron

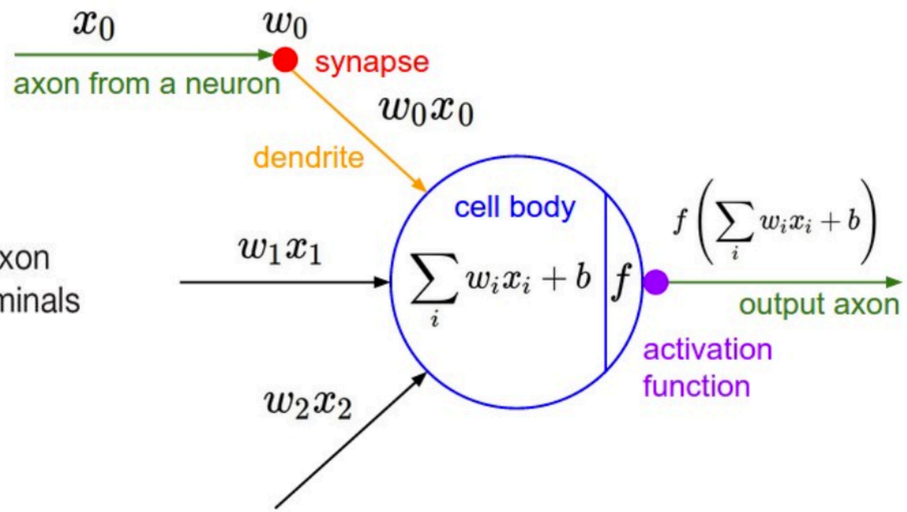


Obviously an oversimplification
Main idea: Connectionism

Biological vs artificial Neuron

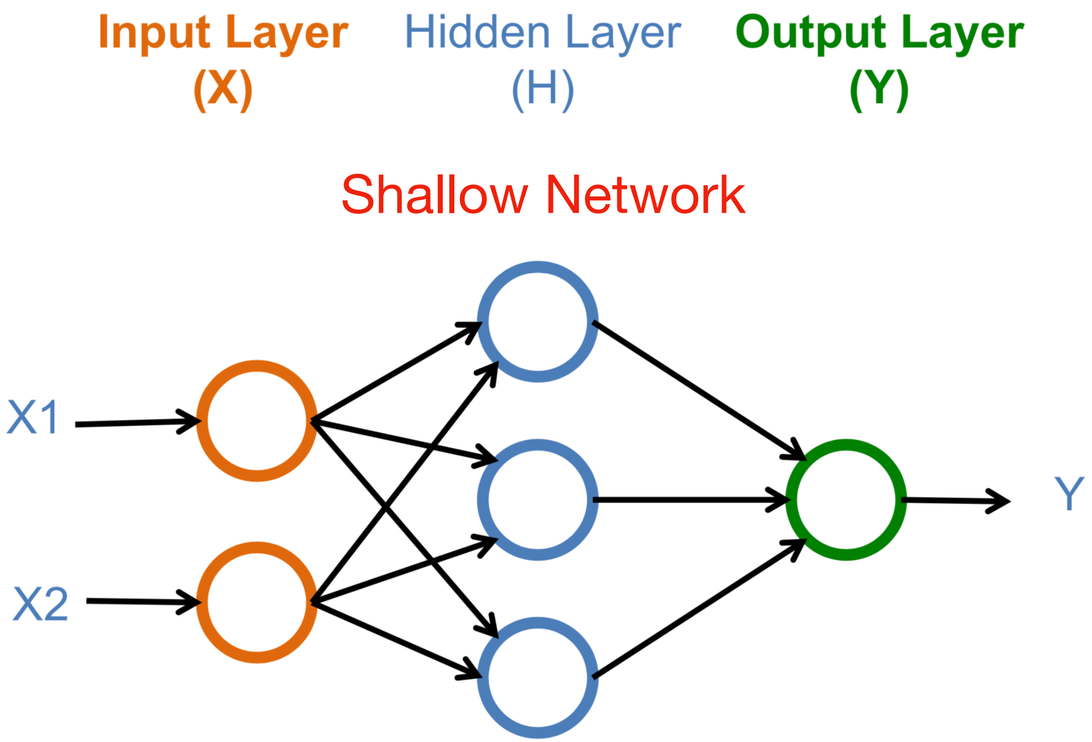
But some characteristics match

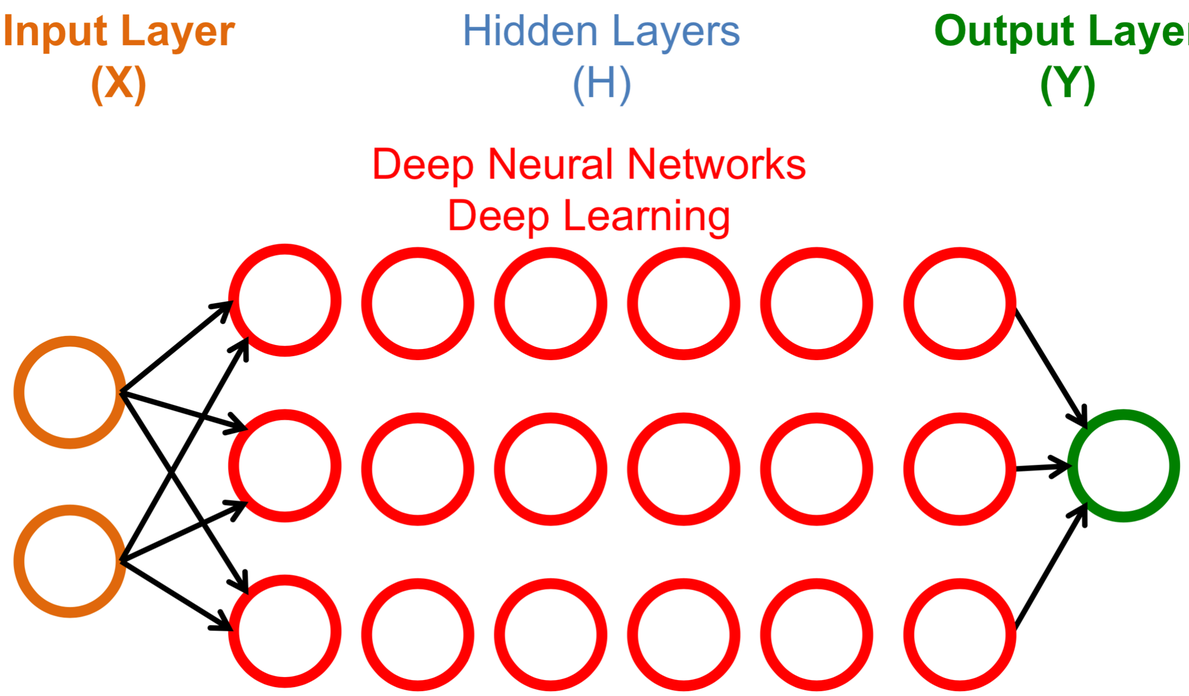




Input currents summ up
The output has a threshold-like non-linear dependence

Deepness



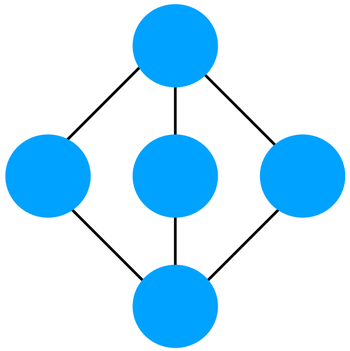


Advantages:
Re-use of lower-level features
Learning of more abstract higher-level features

Number of Learnable Parameters

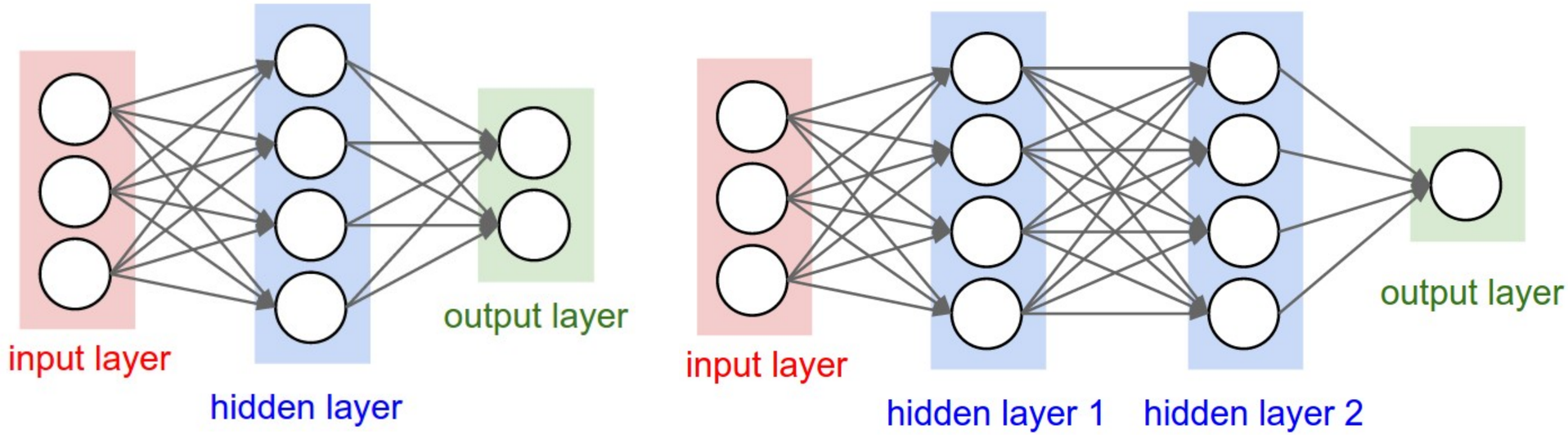
- The number of learnable parameters depends on the architecture
- Networks with fewer neurons can have more parameters

Neurons, biases = 5
Weights = 4
Learnable parameters = 9



Neurons, biases = 5
Weights = 6
Learnable parameters = 11

The Importance of NN Architecture



Source: http://cs231n.github.io/neural-networks-1/#bio

- The capacity of a network can be increased with the number of layers and units per layer.
- As a rule of thumb, going deeper results in more expressive networks, while going wider may lead to overfitting.
- More layers lead to more nested functions and non-linearities that increase the abstraction power, while more units in the same layer usually add features of the same complexity, which might lead to redundancy.

Universal Approximation Theorem

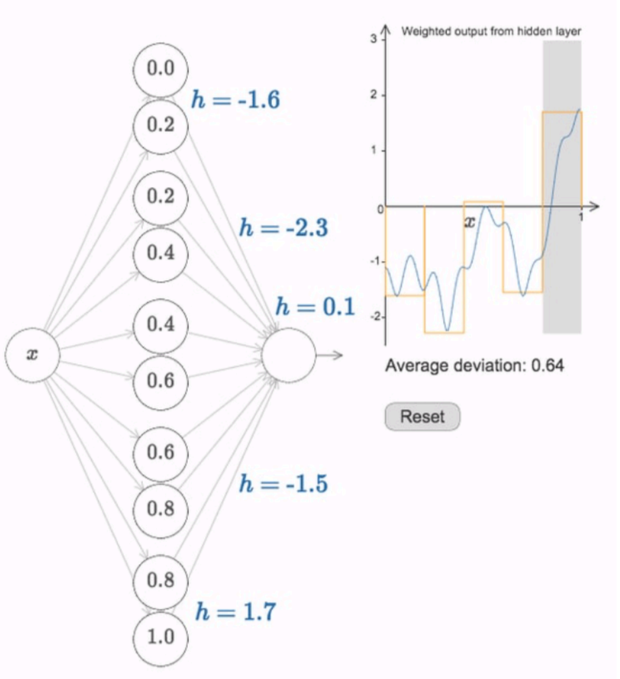
A multilayered network of neurons with a single hidden layer (=shallow network) can be used to approximate any continuous function to any desired precision.

Also, most machine learning tools (such as SVM, boosting, and KNN) can be approximated as neural networks with one or two hidden layers.

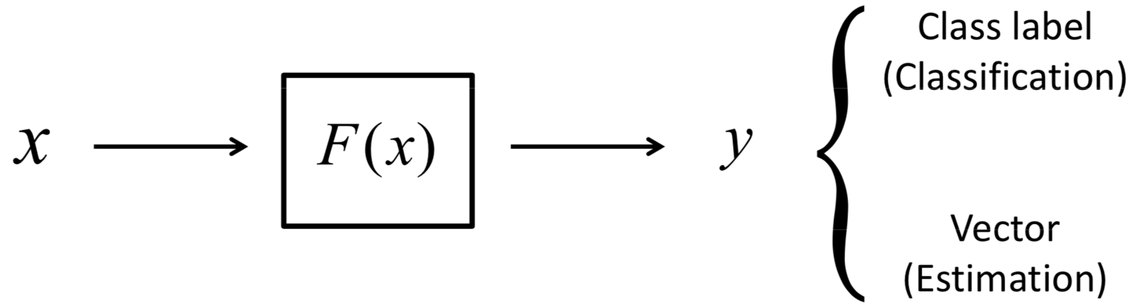
But deep networks are more efficient (for certain problems)

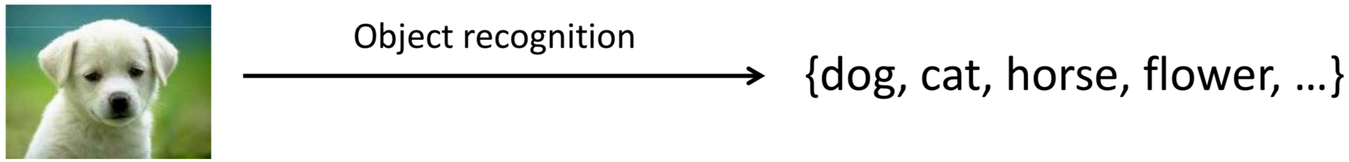
potentially exponentially more efficient (Hastad 1986, Hastad and Goldmann 1991)

- more layers (more sequential computation)
- less hardware (less parallel computation)



Functional View of Deep Learning





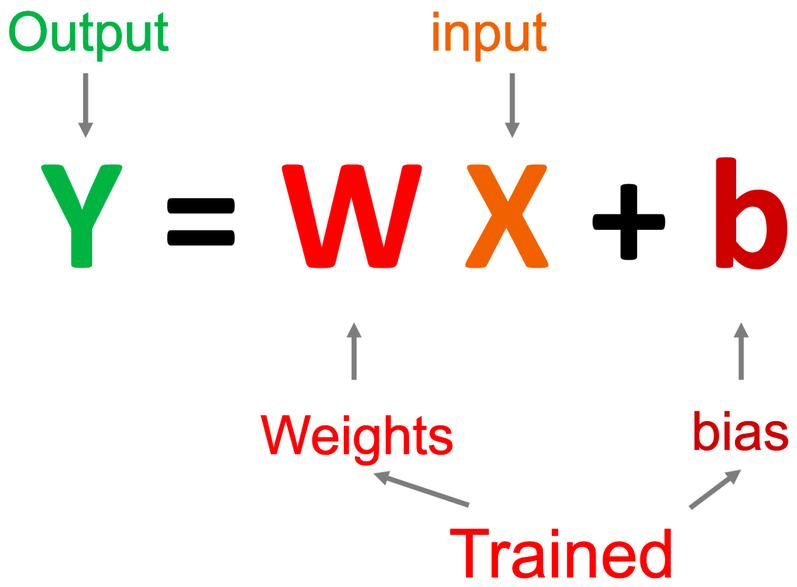
- Deep Networks are complex function approximators
  - Facial Recognition: Map a bunch of pixels to a name
  - Handwriting Recognition: Map an image to a character
- Functions are combination of linear and non-linear components

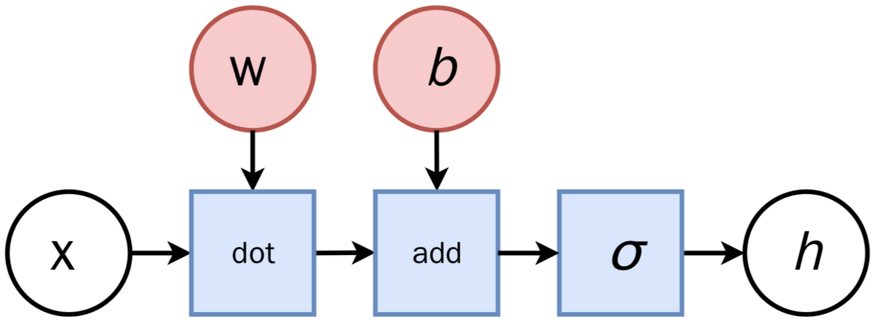
$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$

F: Feature extractor

How does a Neural Network Learn?

- The training data is used to adjust the weights and biases





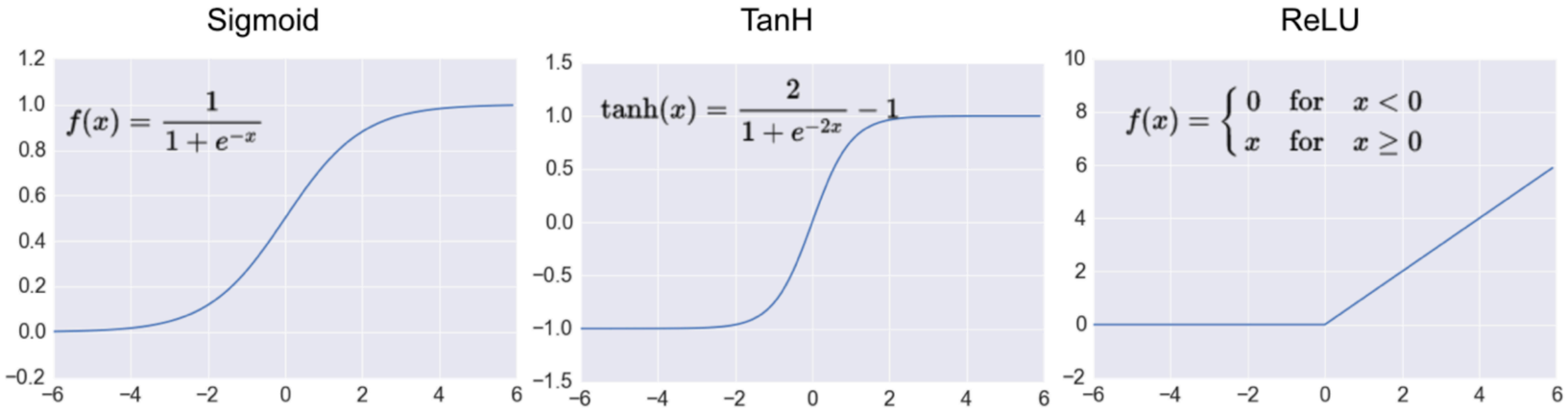
This unit is the lego brick of all neural networks!

Source: Prof. Gilles Louppe

Activation Functions

Why do we need an activation function?

- There will always be this sudden change in the decision (from 0 to 1) when the sum of weighted inputs crosses the threshold [case of the original Perceptron]
- For most real world applications we would expect a smoother decision function which gradually changes from 0 to 1
- Non-linearities allows us to make use of multiple layers



Loss Function

- The loss function measures how well a neural network predicts the expected outputs given the training samples.

- Common Loss Functions:

  - Mean square error (aka maximum likelihood and sum squared error):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

  - Cross-entropy (aka Bernoulli negative log-likelihood and binary cross-entropy):

$$\mathrm{H}(P, Q) \ = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

  - Kullback–Leibler divergence (aka information divergence, information gain and relative entropy):

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \quad = \mathrm{H}(P, Q) - \mathrm{H}(P)$$

    Measures how different two probability distributions are.

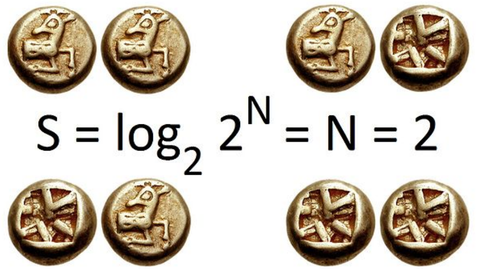
Recap: Information Theory

- In Thermodynamics, Entropy measures the disorder of a system. It increases over time. $\Delta S \geq 0$.
- Claude Shannon derived the properties of Information:

$$S = -k_{\mathrm{B}} \sum p_i \ln p_i$$

p_i: probability of micro-sate

1. $\mathrm{I}(p)$ is monotonically decreasing in p : an increase in the probability of an event decreases the information from an observed event, and vice versa.
2. $\mathrm{I}(p) \geq 0$: information is a non-negative quantity.
3. $\mathrm{I}(1) = 0$: events that always occur do not communicate information.
4. $\mathrm{I}(p_1\, p_2) = \mathrm{I}(p_1) + \mathrm{I}(p_2)$: information due to independent events is additive.



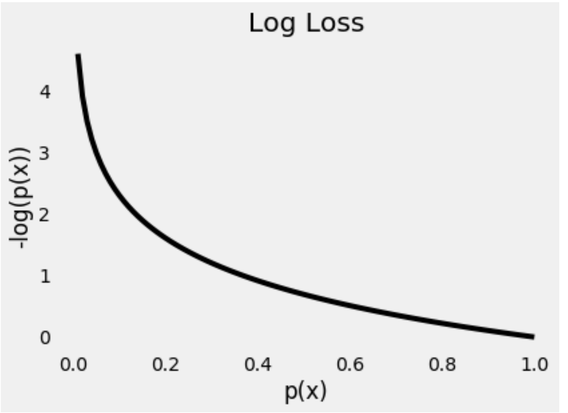
$S = \log_2 2^N = N = 2$

- From which we can derive: S=H=I $= -\sum_i P_i \log P_i = -\,\mathrm{E}_P[\log P]$
- Information entropy is the average amount of information conveyed by an event.
- The more unlikely an event, the more information it carries. $-\log_2(p(E)) = \log_2(1/p(E))$
- The uniform probability distribution (complete randomness) has maximum entropy.

Cross Entropy and KL-Divergence

- Both measure the difference between tow probability distributions P = Truth, and Q = model.
- They measure the information lost, if Q is used instead of P.
- They only differ by a term H(P).
- D makes more sense for a difference measure since D(P|P)=0.
- Minimizing the cross-entropy with respect to Q is equivalent to minimizing the KL divergence (because H(P) does not depend on Q).
- Binary cross entropy is used when the labels are binary y e {1,0}, with probabilities p and (1-p).

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$



Recap: Problem -> Loss Function

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Training a Neural Network

Training a Network

=

Minimize the Cost Function

Minimize the Loss Function

While not done:

Pick a random training example "(input, label)"
Run neural network on "input"

Adjust weights on edges to make output closer to "label"

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016

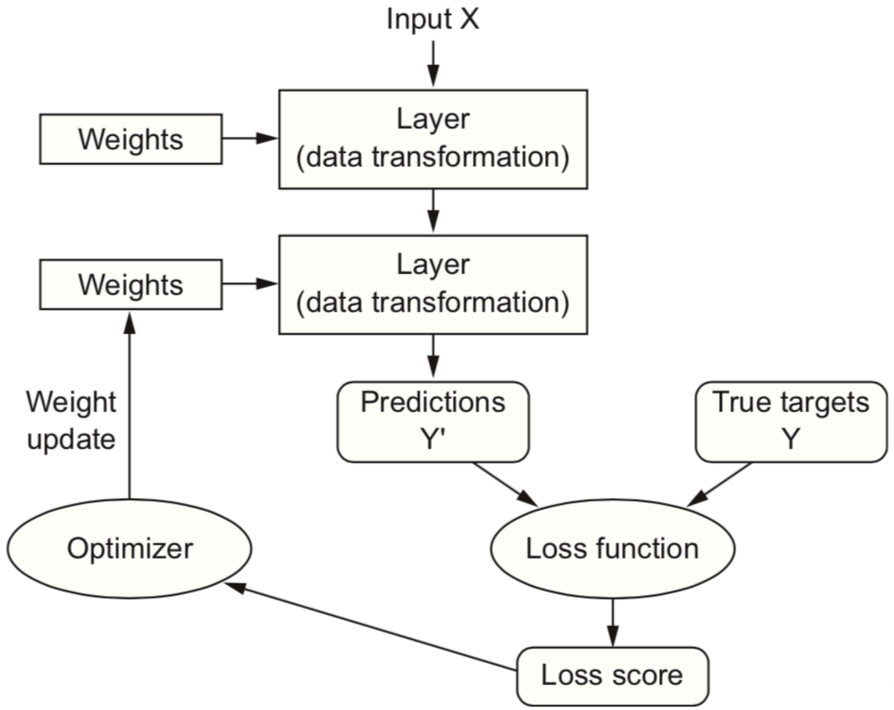


Figure 1.9 The loss score is used as a feedback signal to adjust the weights.

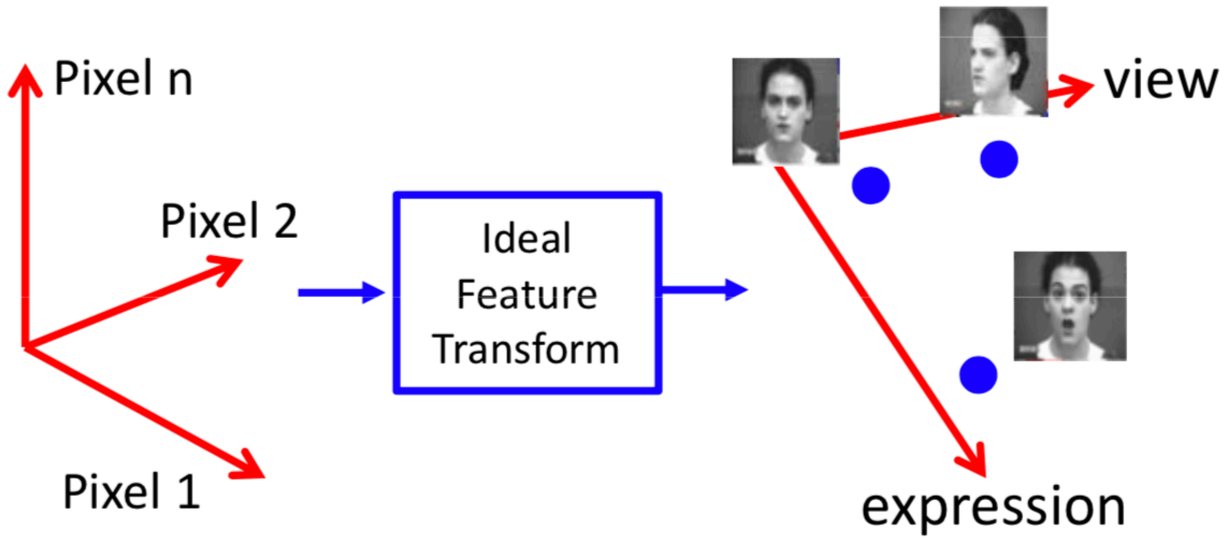
Deep Networks

Advantages of feature learning:

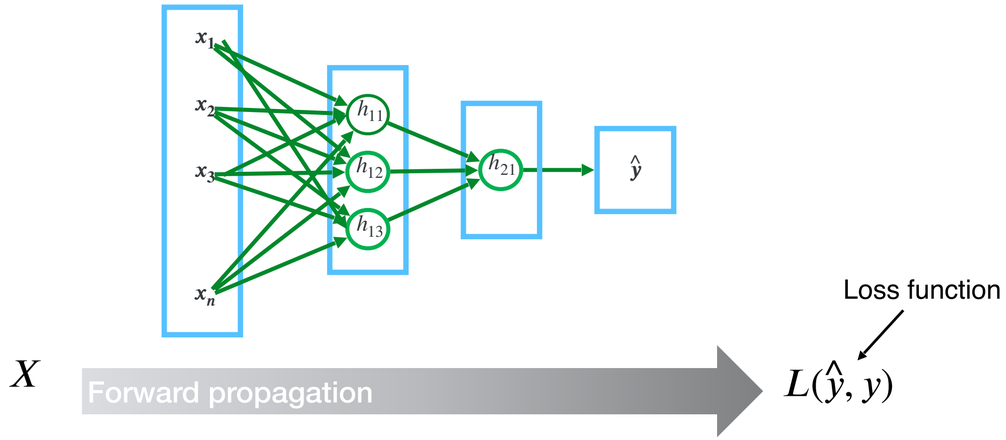
– Make better use of big data

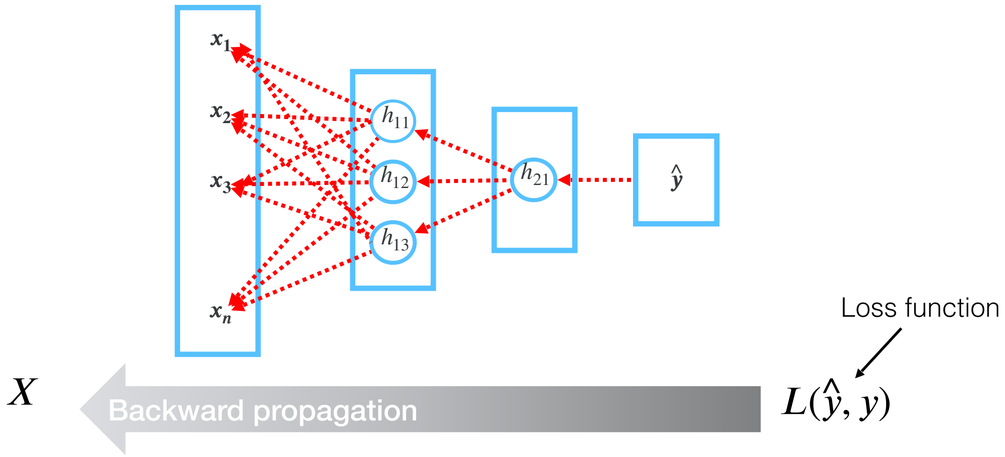
– Jointly learning feature transformations and classifiers makes their integration optimal

– Faster to get feature representations for new applications



Backpropagation of Errors



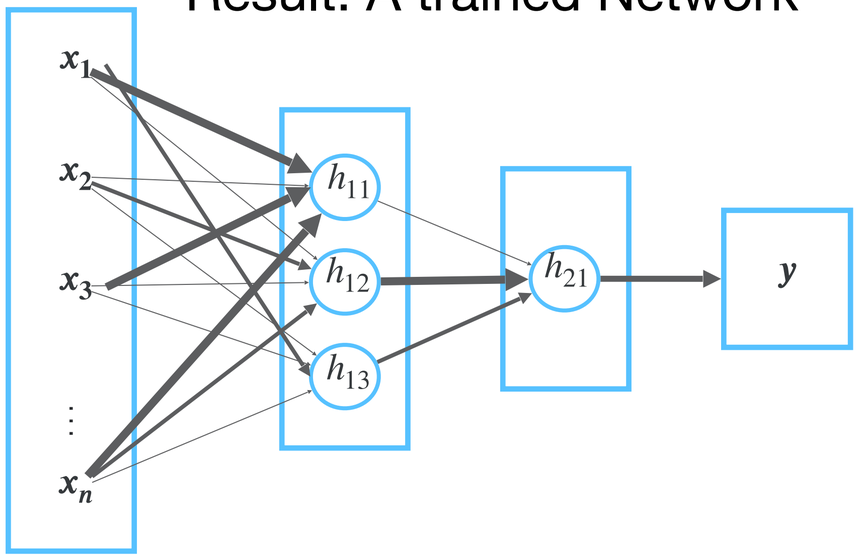


Weigert update

$$w_{11}^{1}(new) = w_{11}^{1} + \eta\ \delta_{11} \frac{\partial}{\partial w_{ij}^{k}} J(\boldsymbol{w})\ x_1$$

learning rate

Result: A trained Network



Backpropagation

- We change the weights so that the error gets reduced:

$$w_{ij}^{\text{neu}} = w_{ij}^{\text{alt}} + \Delta w_{ij} \qquad \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

$$E = \frac{1}{2} \sum_{i=1}^{n} (t_i - o_i)^2$$

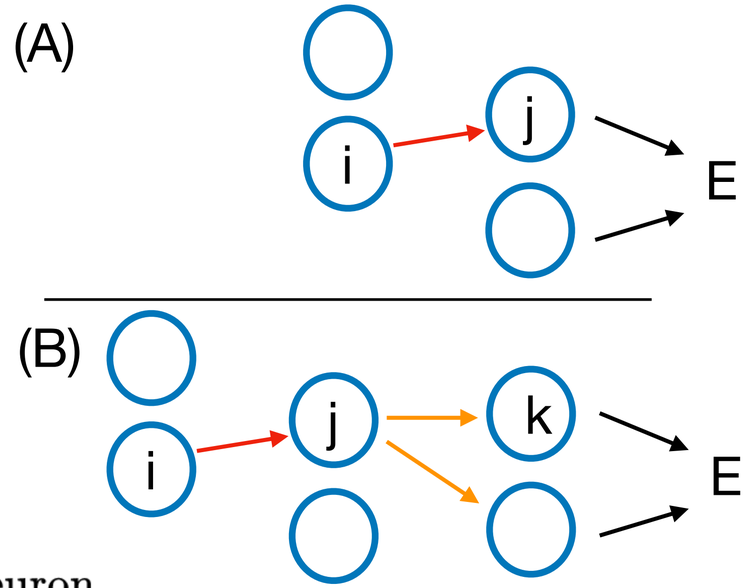
- The neural network is described by:

$$o_j = \varphi(\text{net}_j) \qquad \text{net}_j = \sum_{i=1}^{n} x_i w_{ij}$$

- Application of the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ij}}$$

$$= \delta_j o_i$$



Error signal of neutron j:

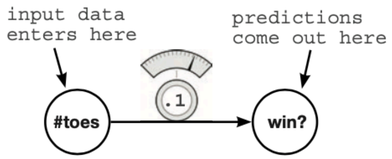
$$\delta_j = \begin{cases} \varphi'(\text{net}_j)(o_j - t_j) & \text{(A) if } j \text{ is an output neuron,} \\ \varphi'(\text{net}_j) \sum_k \delta_k w_{jk} & \text{(B) if } j \text{ is an inner neuron.} \end{cases}$$

https://de.wikipedia.org/wiki/Backpropagation

One Iteration of Gradient Descent

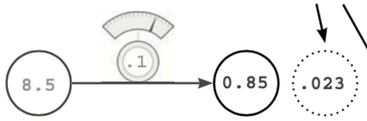
This performs a weight update on a single "training example" (input->true) pair

1 An Empty Network




```
weight = 0.1

alpha = 0.01

def neural_network(input, weight):

    prediction = input * weight

    return prediction
```

2 PREDIT: Making A Prediction And Evaluating Error



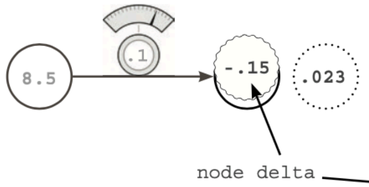

```
number_of_toes = [8.5]
win_or_lose_binary = [1] // (won!!!)

input = number_of_toes[0]
goal_pred = win_or_lose_binary[0]

pred = neural_network(input,weight)

error = (pred - goal_pred) ** 2
```

The "error" is simply a way of measuring "how much we missed". There are multiple ways to calculate error as we will learn later. This one is "Mean Squared Error"

raw error

Forces the raw error to be positive by multiplying it by itself. Negative error wouldn't make sense.

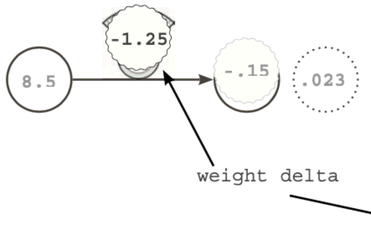
3 COMPARE: Calculating "Node Delta" and Putting it on the Output Node




```
number_of_toes = [8.5]
win_or_lose_binary = [1] // (won!!!)

input = number_of_toes[0]
goal_pred = win_or_lose_binary[0]

pred = neural_network(input,weight)

error = (pred - goal_pred) ** 2

delta = pred - goal_pred
```

Delta is a measurement of "how much this node missed". Thus, since the true prediction was 1.0, and our network's prediction was 0.85, the network was too ***low*** by 0.15. Thus, delta is ***negative*** 0.15.

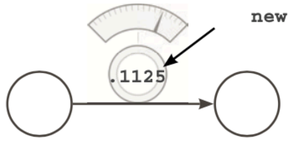
4 LEARN: Calculating "Weight Delta" and Putting it on the Weight




```
number_of_toes = [8.5]
win_or_lose_binary = [1] // (won!!!)

input = number_of_toes[0]
goal_pred = win_or_lose_binary[0]

pred = neural_network(input,weight)

error = (pred - goal_pred) ** 2

delta = pred - goal_pred

weight_delta = input * delta
```

Weight delta is a measure of "how much this weight caused the newtork to miss". We calculate it by multiplying the weight's output "Node Delta" by the weight's input. Thus, we create each "Weight Delta" by *scaling* it's output "Node Delta" by the weight's input. This accounts for the 3 aforementioned properties of our "direction_and_amount", scaling, negative reversal, and stopping.

5 LEARN: Updating the Weight



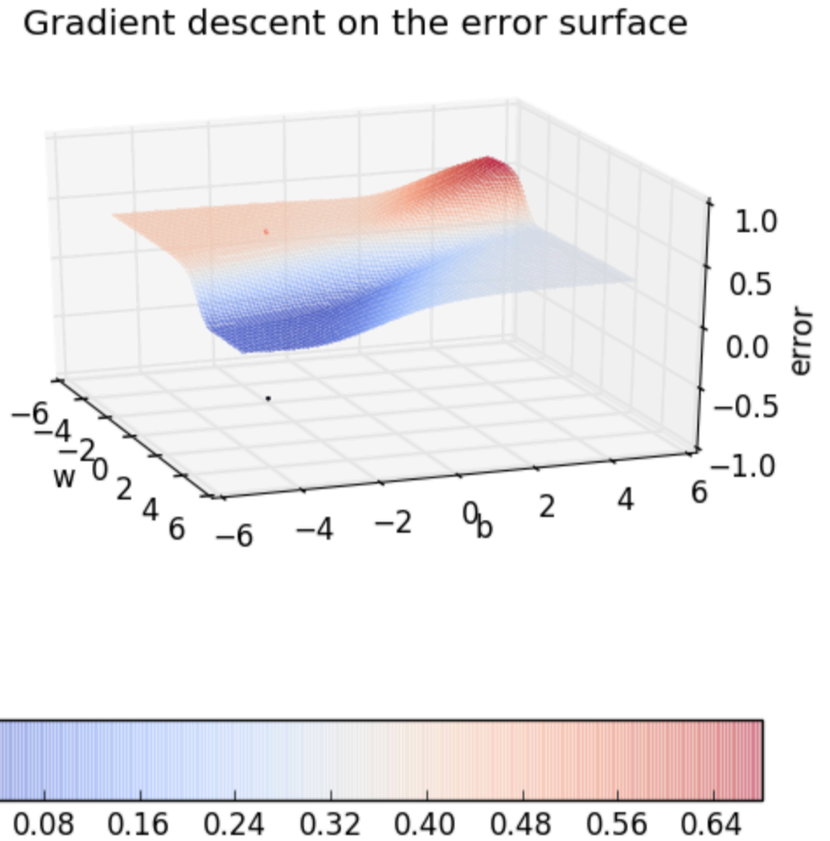

```
number_of_toes = [8.5]
win_or_lose_binary = [1] // (won!!!)

input = number_of_toes[0]
goal_pred = win_or_lose_binary[0]

pred = neural_network(input,weight)

error = (pred - goal_pred) ** 2

delta = pred - goal_pred

weight_delta = input * delta

alpha = 0.01 // fixed before training

weight -= weight_delta * alpha
```

We multiply our weight_delta by a small number "alpha" before using it to update our weight. This allows us to control how fast the network learns. If it learns too fast, it can update weights too aggressively and overshoot. More on this later. Note that the weight update made the same change (small increase) as Hot and Cold Learning

Source: Grokking Deep Learning

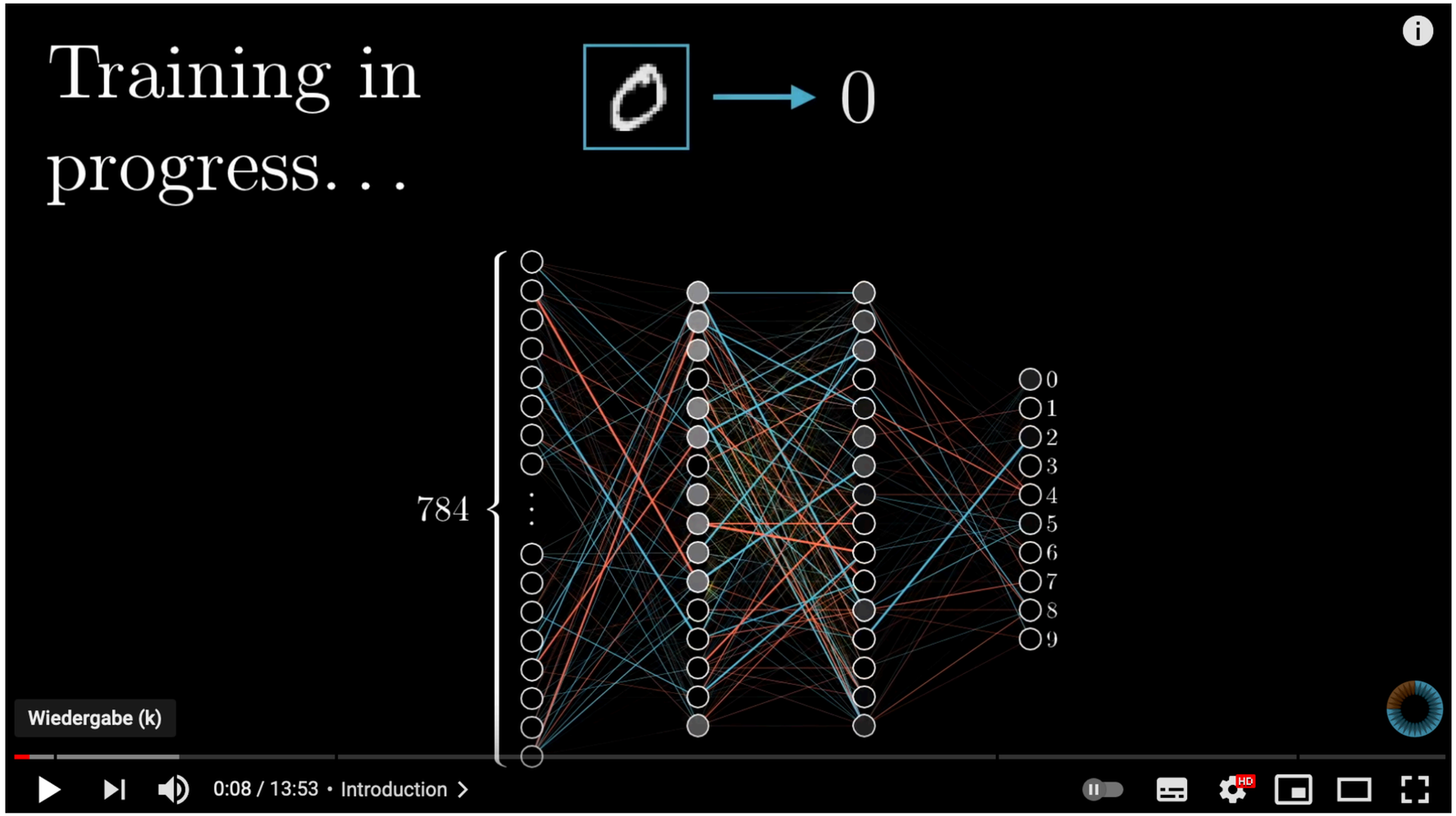
Gradient Descent

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



Backpropagation explained by 3Blue1Brown

https://www.youtube.com/watch?v=Ilg3gGewQ5U

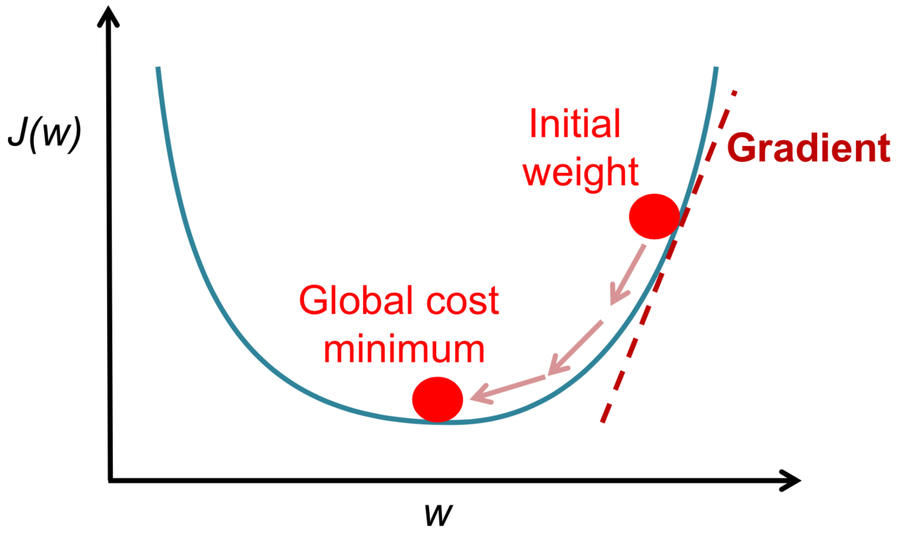


Optimizers

Stochastic gradient descent:

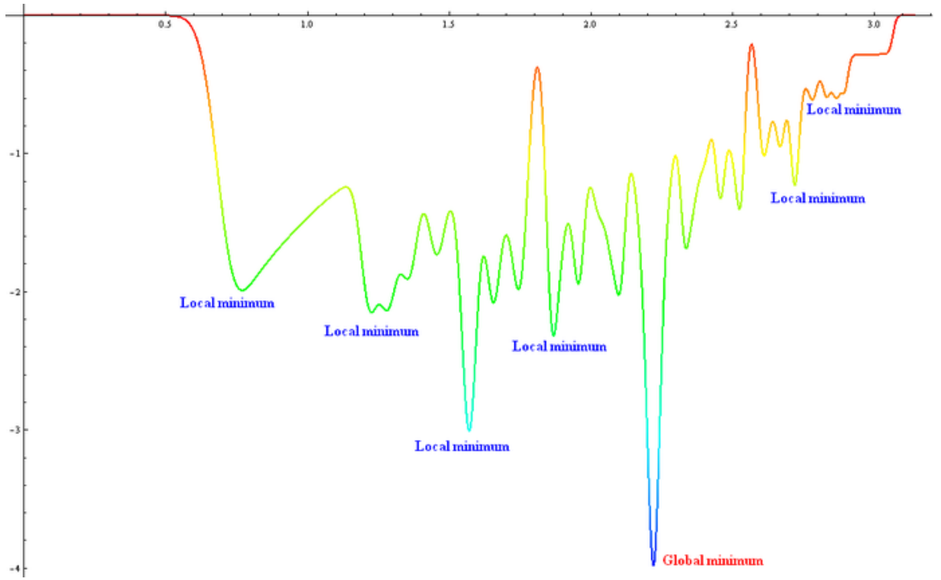
1. Choose an initial vector of parameters
2. Repeat until convergence:

- Randomly shuffle training examples
- Move the weight vector towards the direction of steepest descent by learning rate η



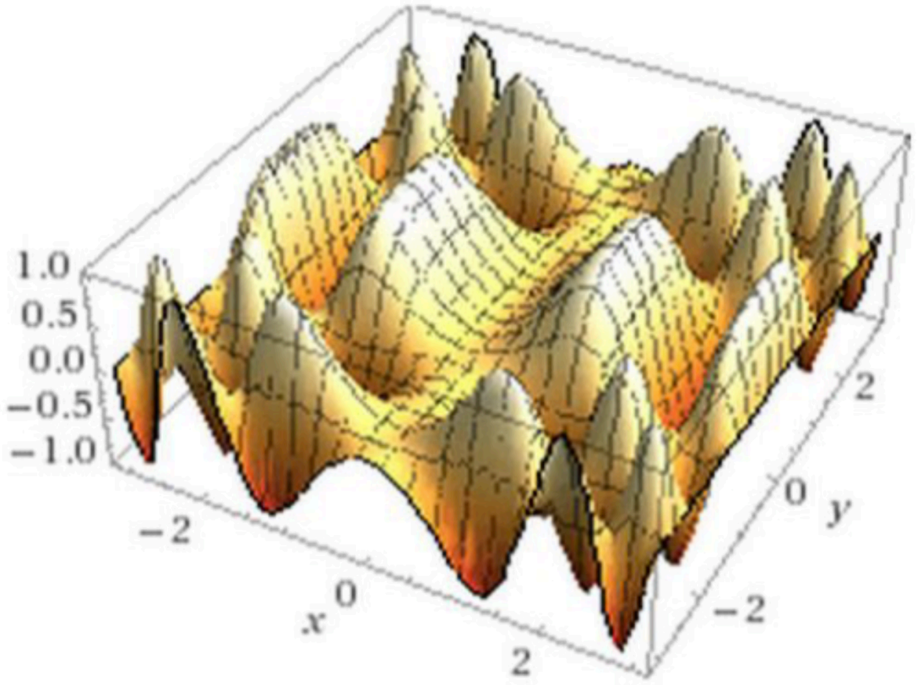
Advanced choices:

- Momentum: save the update at each iteration, and determine the next update as a linear combination of the gradient and the previous update
- Adaptive learning rate methods: RMSprop (2013), Adagrad (2011), Adam (2015)



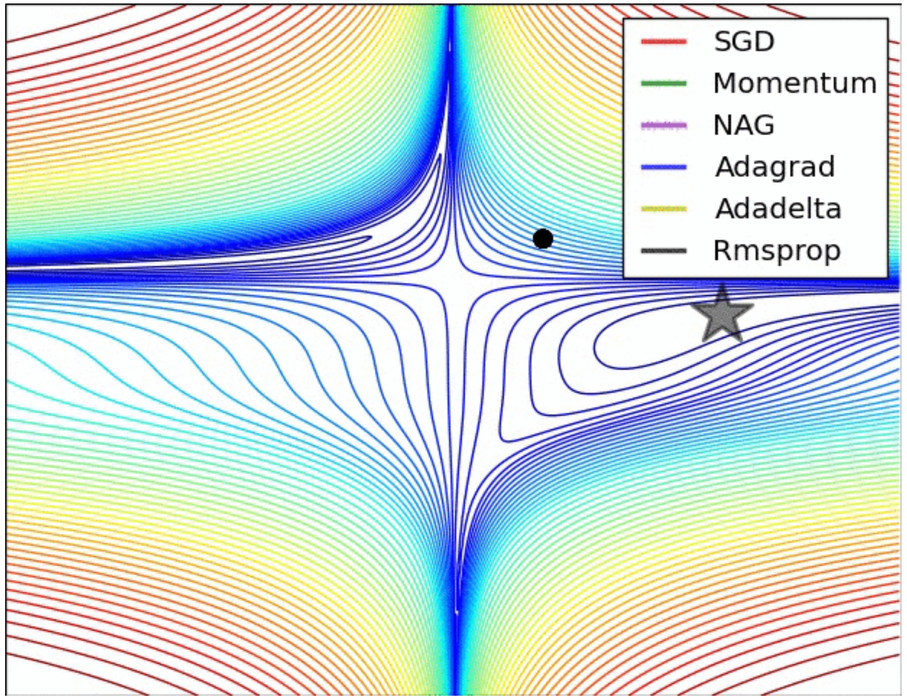
http://www.glopt.net/what-is-global-optimization.html

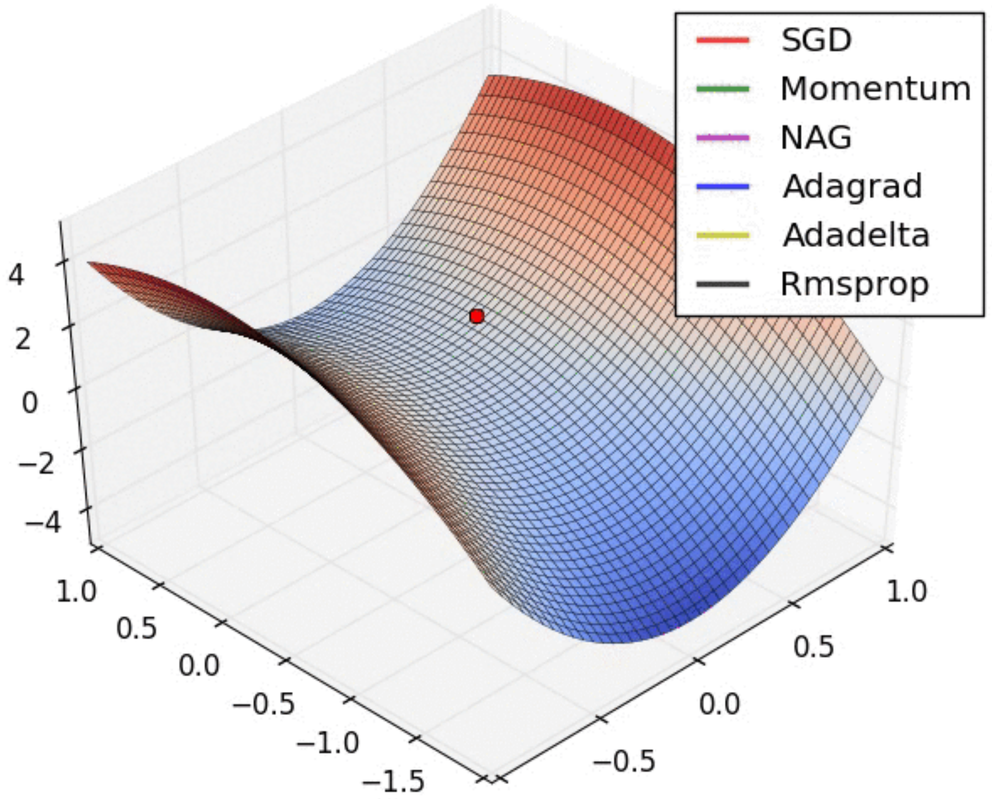
Optimization in many dimensions



Example function with two variables.
Real neural nets may have millions of variables.

Comparison of different Optimizers





How to avoid Overfitting: Regularization

Overfitting occurs when a model with high capacity fits the noise in the data instead of the general underlying relationship.

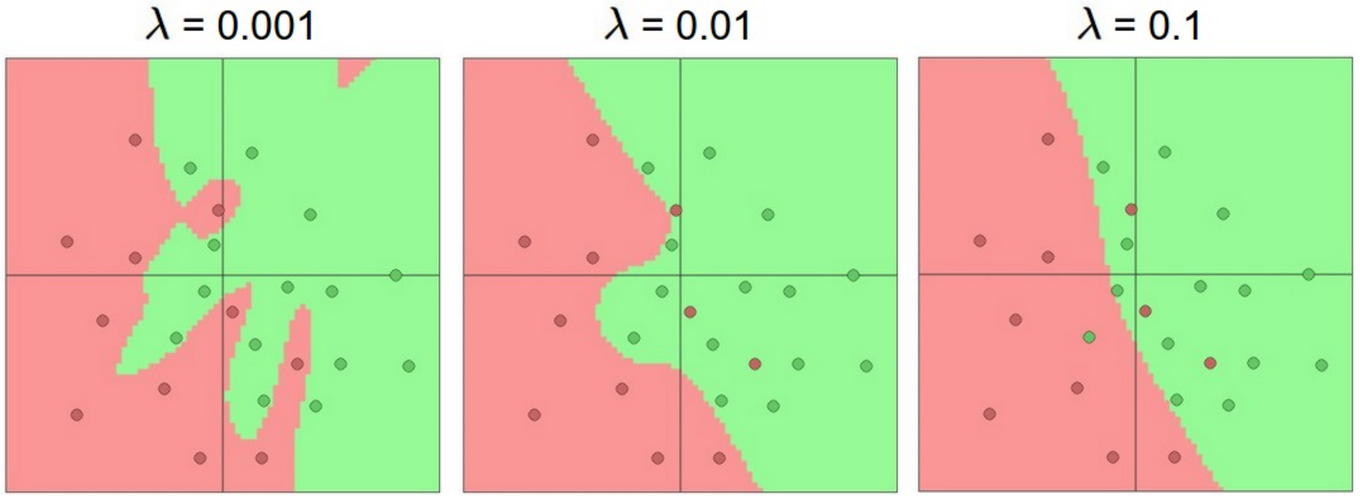
- Classical approach: Create weight penalties L1 and L2
- Dataset augmentation:
  - Create fake data and add it to the training set.
- Noise addition:
  - Add noise to data (example: denoising autoencoder): The network is trained to reconstruct the input from a corrupted version of it.
- Dropout
- BatchNormalization
- Earl Stopping

Effect of L1/L2 Regularization

L1 / L2 regularization reduces the size of the used weight values.

$$E = \frac{1}{2} * \sum (t_k - o_k)^2 + \frac{\lambda}{2} * \sum {w_i}^2$$

plain error — weight penalty

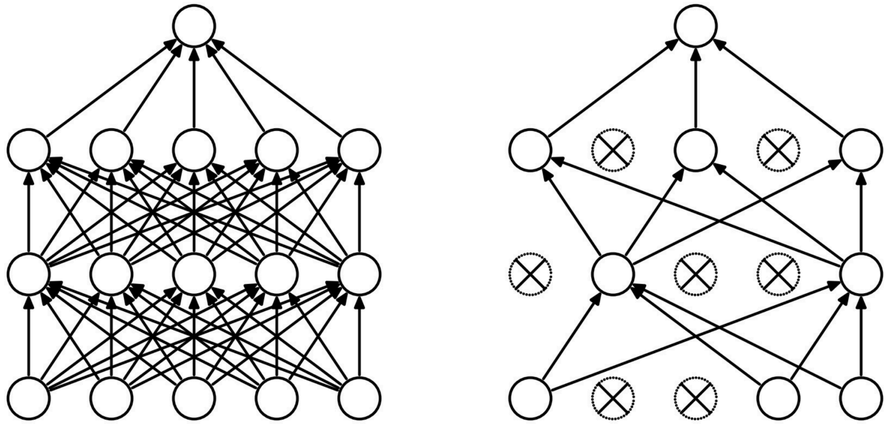


The effects of regularization strength:

- Each neural network above has 20 hidden neurons, but increasing the regularization strength makes its final decision regions smoother.

Dropout

Dropout: dropping out units (both hidden and visible) in a neural network.



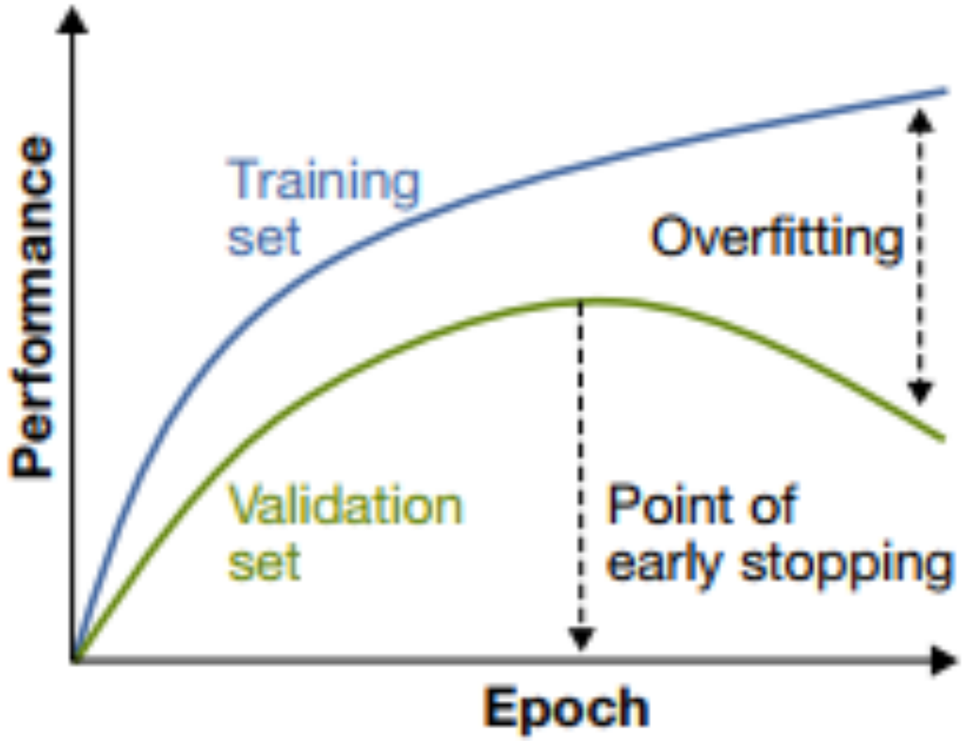
(a) Standard Neural Net (b) After applying dropout.

- At each training iteration a dropout layer randomly removes some nodes in the network with probability p along with all of their incoming and outgoing connections.
- Dropout can be applied to hidden or input layer.
- Why it works:
  - Prevents co-adaptation between neurons.
  - Dropout is an example of ensemble technique, where multiple thinned networks with shared parameters are averaged out.

Source: Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting."Journal of machine learning research15, no. 1 (2014): 1929-1958.

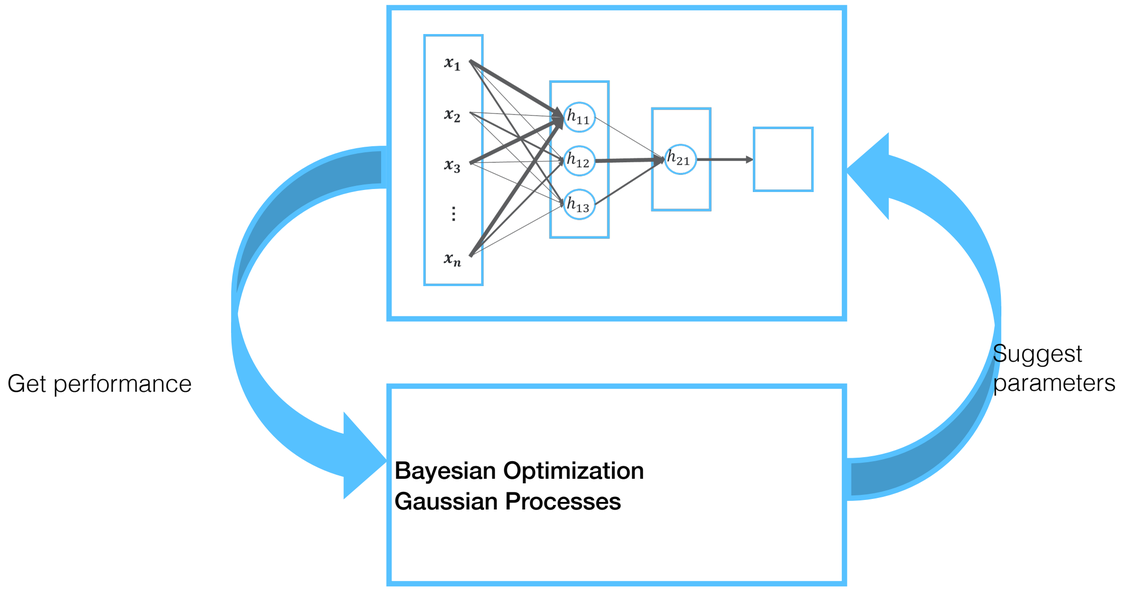
Early Stopping

Stop training as so on as the error on the validation set is higher than it was the last time it was checked.



Hyperparameter Tuning

- Grid Search
- Random Search
- Corse to fine
- Bayesian Optimization

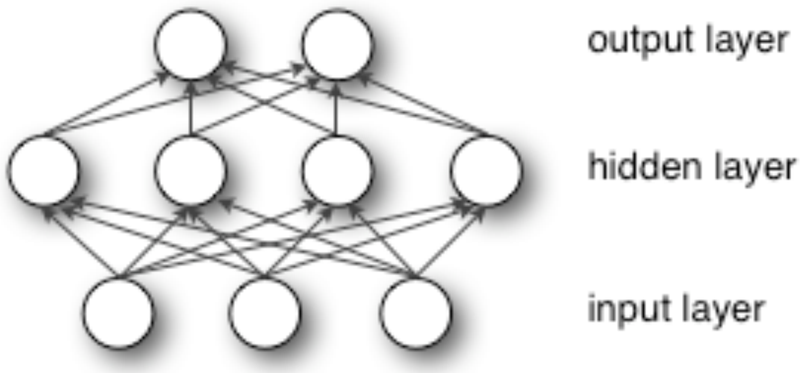


DNNs can involve many hyperparameters. The most common include:
- initial learning rate
- momentum
- regularization strength (L2 penalty, dropout strength, etc)

A grid search exploration of all possible parameter combination might not be the most efficient way of tuning the DNN!

Part 2: Specific architectures

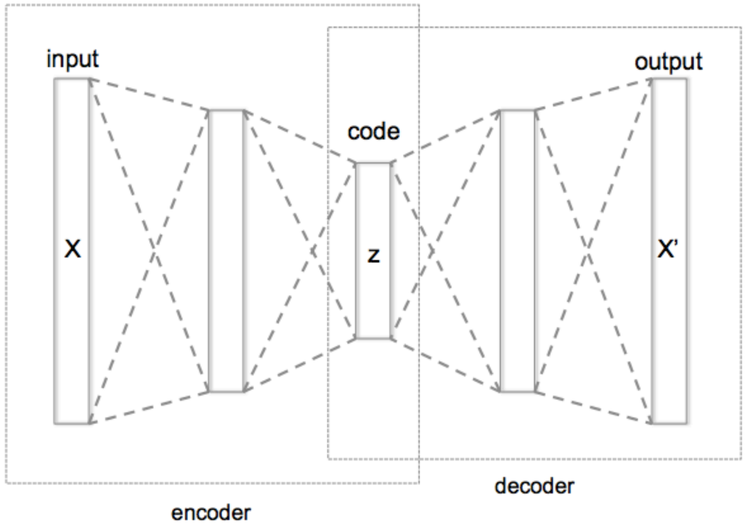
Multi-layer Perceptron



- An MLP is a DNN that:
  - Consists at least of three layers of nodes (i.e. there is at least one hidden layer).
  - It is always feedforward (no loops are allowed).
  - Consecutive layers are fully connected.

Source: http://deeplearning.net/tutorial/mlp.html

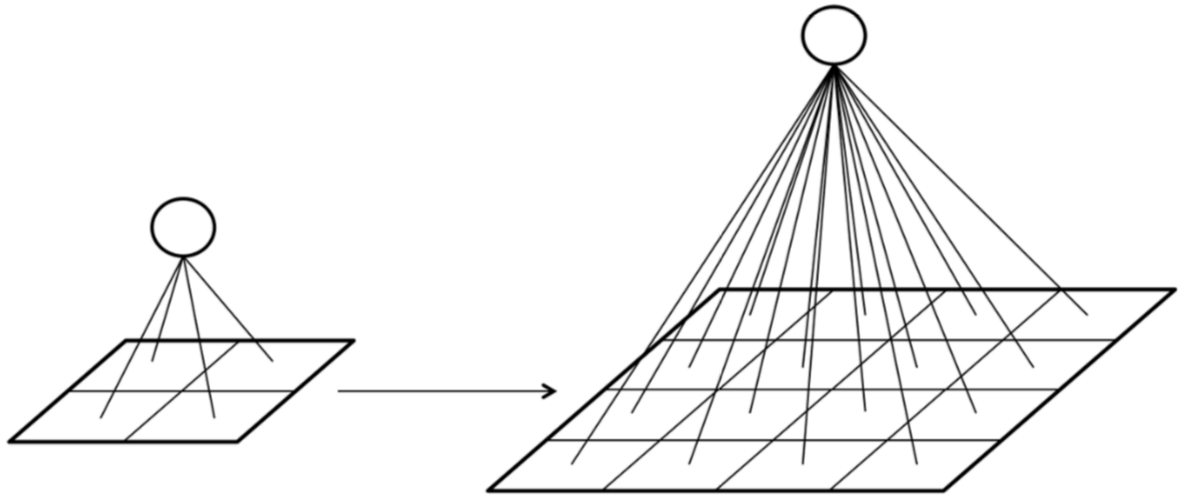
Autoencoders



$$\phi : \mathcal{X} \to \mathcal{F}$$
$$\psi : \mathcal{F} \to \mathcal{X}$$
$$\phi, \psi = \arg\min_{\phi,\psi} \|X - (\psi \circ \phi)X\|^2$$

- AEs are DNNs used for unsupervised learning.
- AEs try to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.
- If linear activations are used, or only a single sigmoid hidden layer, then the optimal solution to an autoencoder is strongly related to principal component analysis (PCA)

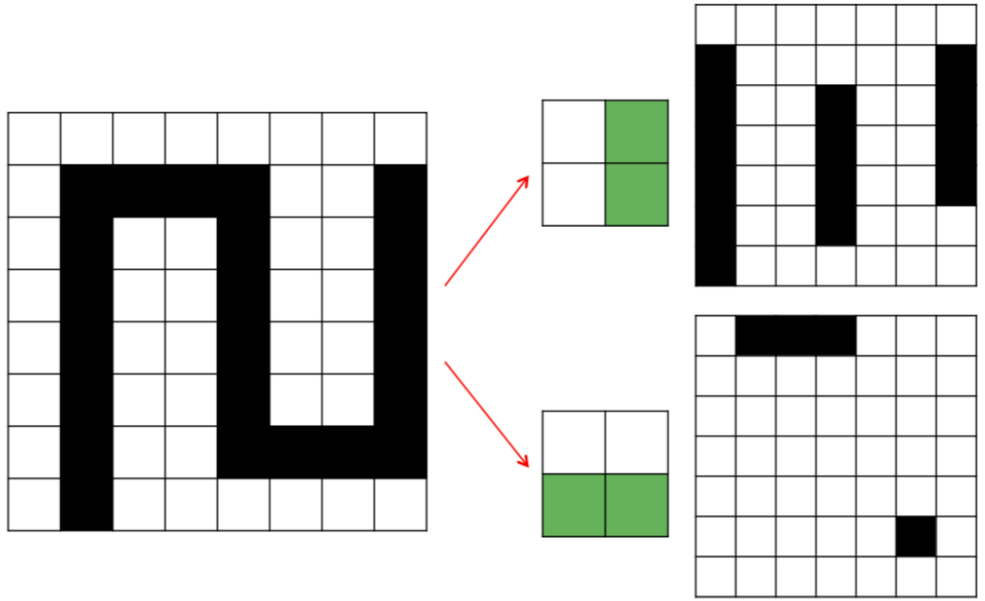
Towards Convolution



The fully connected architecture becomes intractable.
A 200x200 picture would have 120k weights (with 3 colours) per neuron in the next layer.
And we need many neurons in many layers.

Source: Fundamentals of Deep Learning

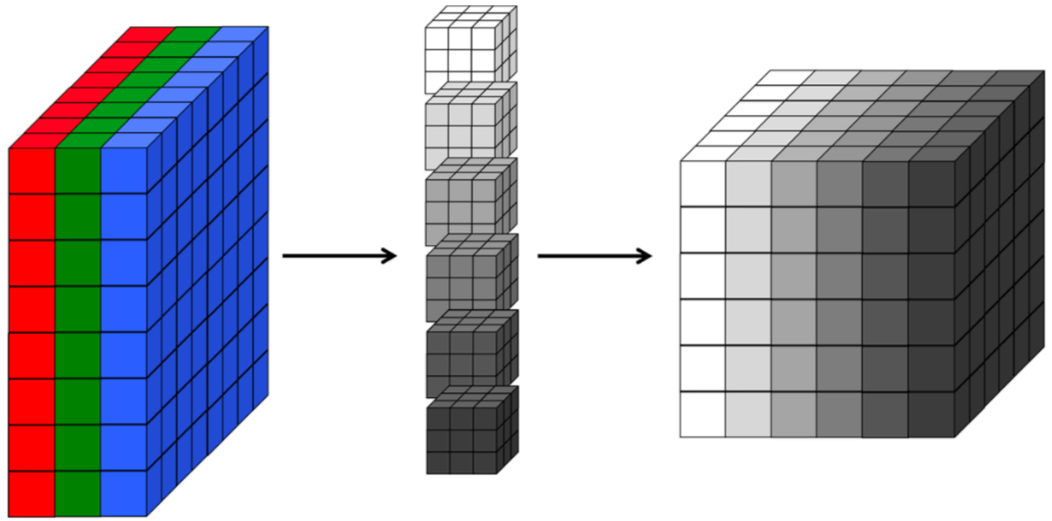
Filters



Example of vertical and horizontal line detector filters.
(Motivated by the visual cortex.)

Source: Fundamentals of Deep Learning

Sets of 3D Filters



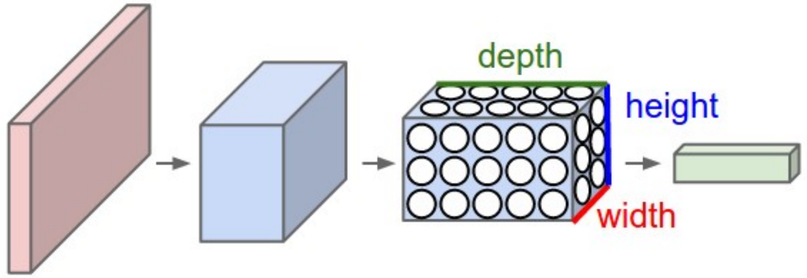
Tensor dimensions: (n_{Height}, n_{Width}, $n_{Channels}$) -> (n_{Height}, n_{Width}, $n_{Filters}$)

Example of brightens filtering:

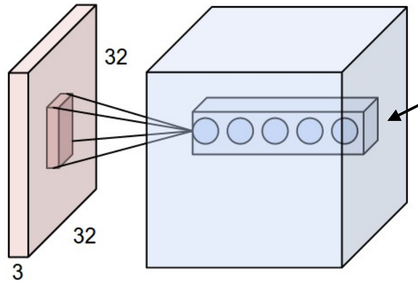
- Each neuron connects only to local region
- Shifting the input region does have shared weights.
- Each brightness filter considers the combination of all 3 color inputs.
- The output has one slide per filter.

Source: Fundamentals of Deep Learning

Convolution

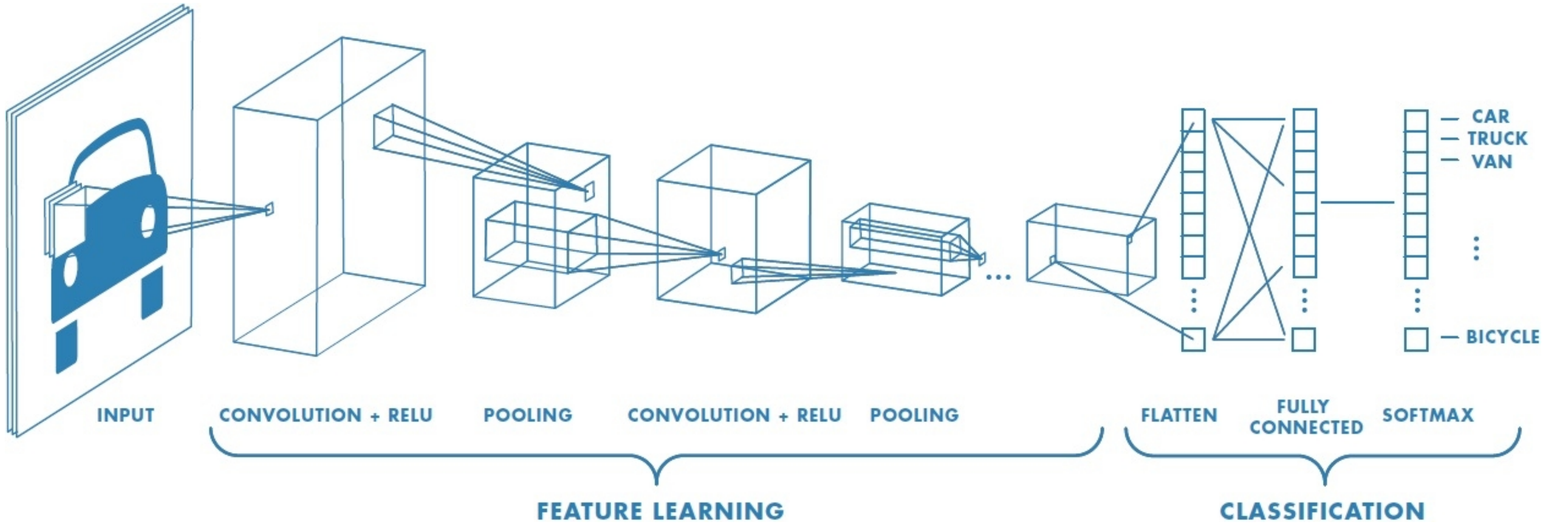


Source: http://cs231n.github.io/convolutional-networks/



One fibre: set of filters
each looking to different features

- A CNN arranges neurons in three dimensions (width, height, depth).
- In general a filter combines information from all features that have been learned.

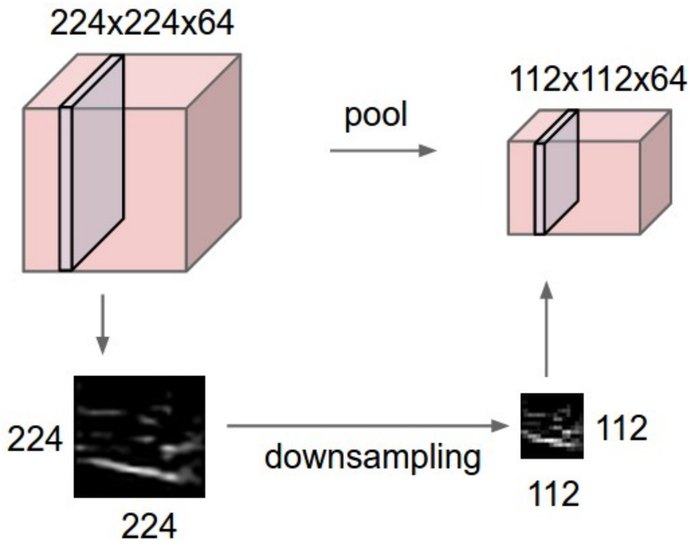


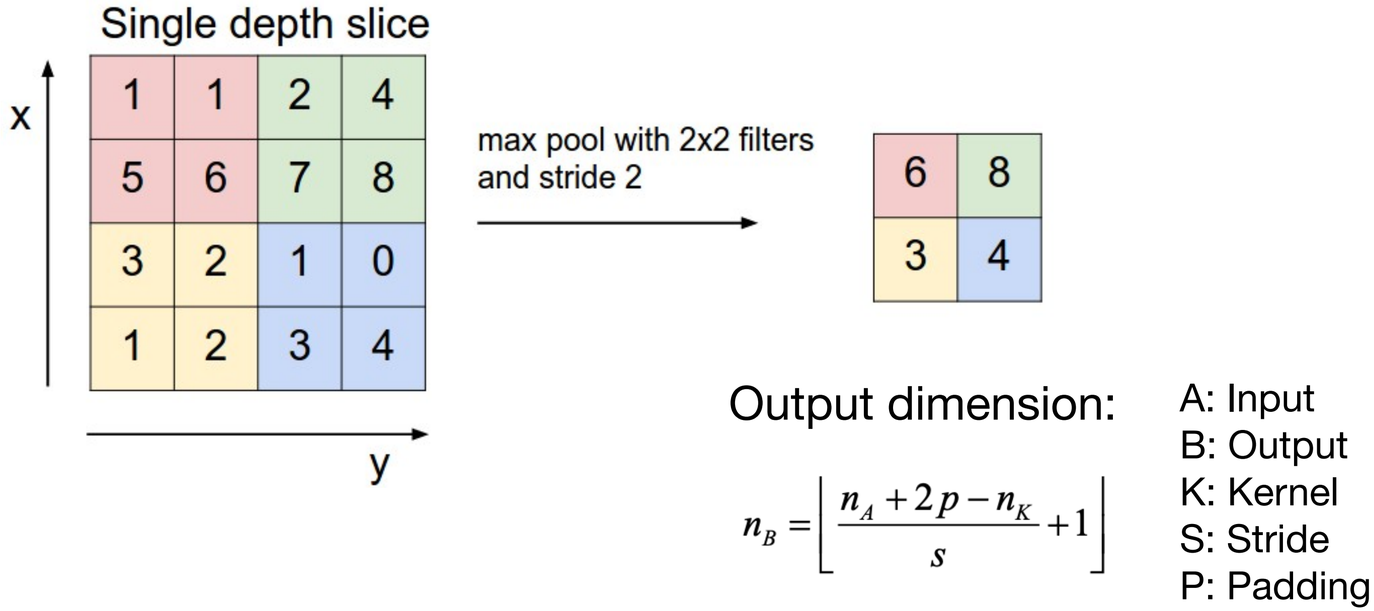
Source: https://de.mathworks.com/discovery/convolutional-neural-network.html

Pooling Layers

Hubel & Wiesel 1962:

simple cells detect local features
complex cells "pool" the outputs of simple cells within a retinotopic neighborhood





Output dimension:

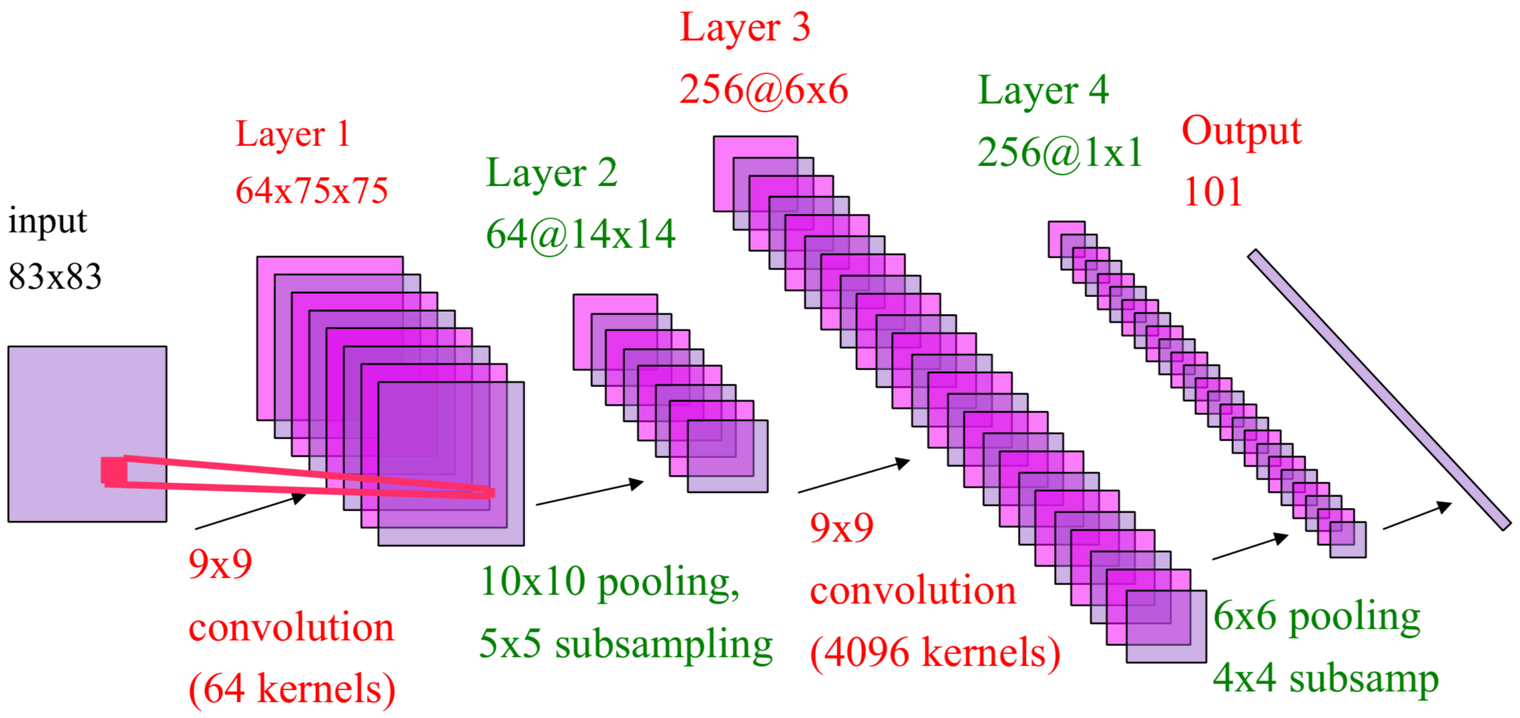
$$n_B = \left\lfloor \frac{n_A + 2p - n_K}{s} + 1 \right\rfloor$$

A: Input
B: Output
K: Kernel
S: Stride
P: Padding

Max-pooling is a form of non-linear down-sampling. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value.

http://cs231n.github.io/convolutional-networks/

CNN Parameters



OverFeat Network, 2014

Layer	1	2	3	4	5	6	7	8	Output 9
Stage	conv + max	conv + max	conv	conv	conv	conv + max	full	full	full
# channels	96	256	512	512	1024	1024	4096	4096	1000
Filter size	7x7	7x7	3x3	3x3	3x3	3x3	-	-	-
Conv. stride	2x2	1x1	1x1	1x1	1x1	1x1	-	-	-
Pooling size	3x3	2x2	-	-	-	3x3	-	-	-
Pooling stride	3x3	2x2	-	-	-	3x3	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	221x221	36x36	15x15	15x15	15x15	15x15	5x5	1x1	1x1

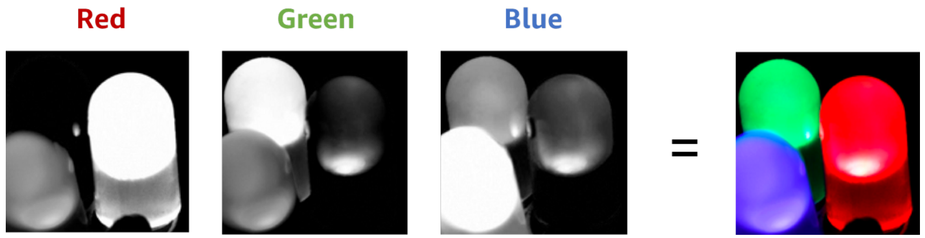
How are different Channels Combined?

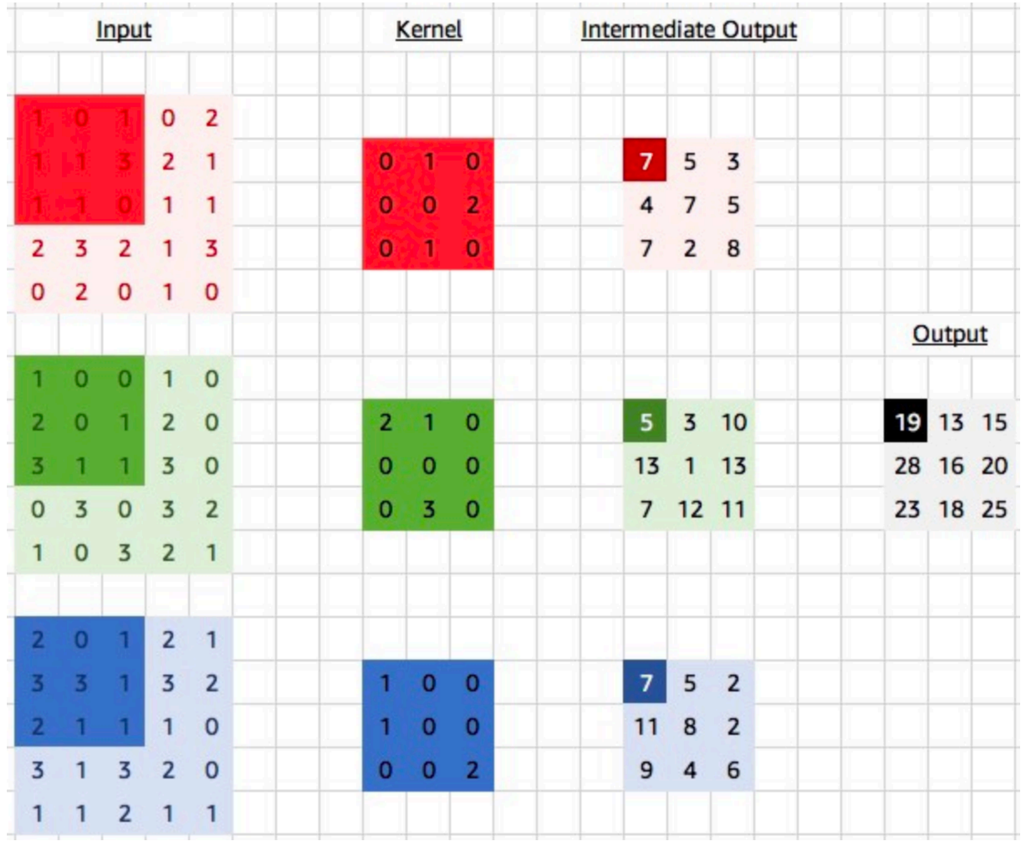
If the data has several channels:

- Filters are applied per channel.

  (with different weights per channel)
- Then added at the end.

  (Thus, this is equivalent to a 3D kernel!)





Source: https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108

1D Convolution with Multiple Channels

	Input					
Date	1st	2nd	3rd	4th	5th	6th
Price	1	3	3	0	1	2
Marketing spend	0	2	1	1	2	0
Outside temperature	3	2	2	3	1	1
Weekend	0	1	1	0	0	0

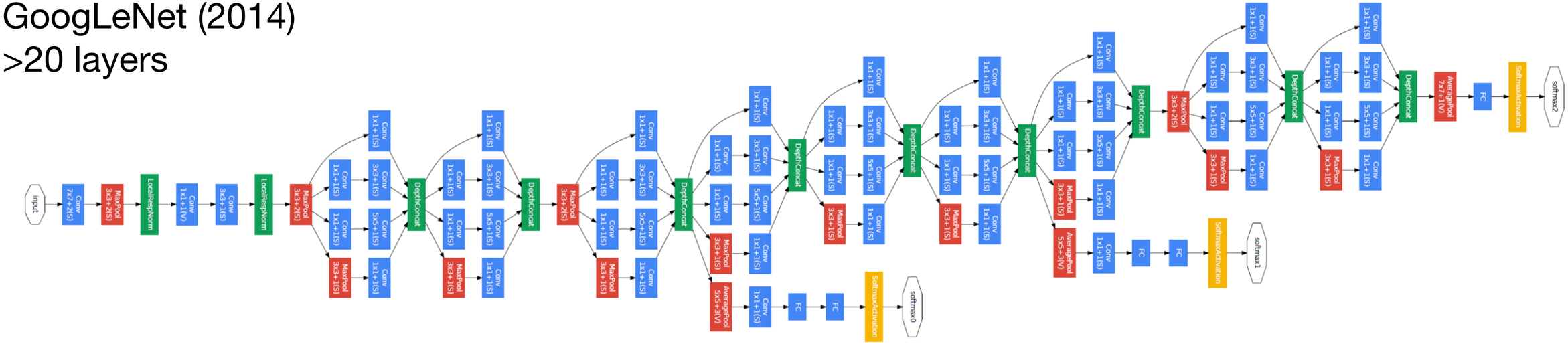
- Even though we are using a 1D Convolution, we have a 2D kernel! A 1D convolution just means we slide the kernel along one dimension. The shape of the kernel depends also on the number of input channels.
- One could achieve the same with a 2D convolution (and 1 channel),

  but 1D is prefered, since:

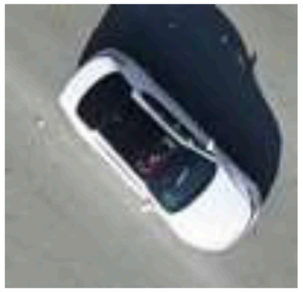
  - With a 1D Convolution you don’t need to specify the channel dimension.
  - 2D results might differ when adding padding, stride and dilation.

Bigger CNNs

GoogLeNet (2014)
>20 layers



Application: Object Detection & Classification

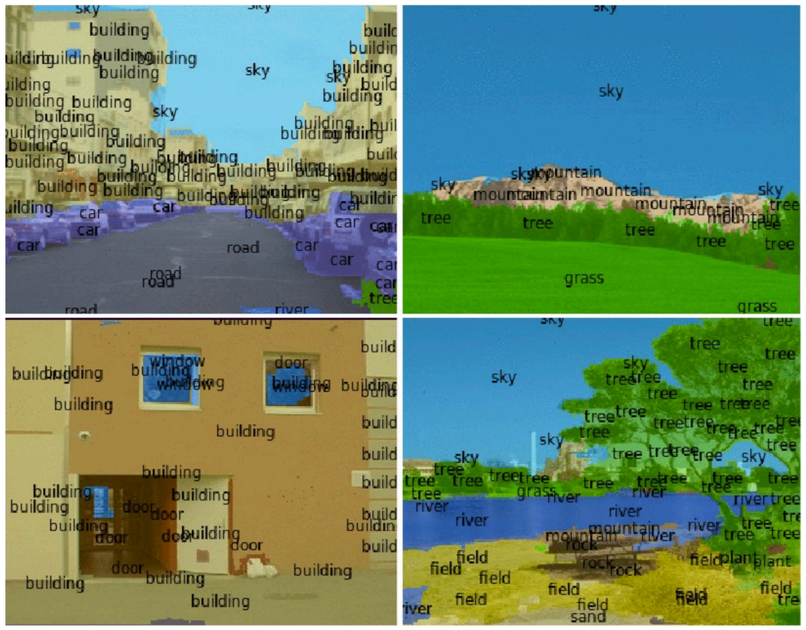


Famous ConvNet Architechures

- **LeNet**. The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990's. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.
- **AlexNet**. The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).
- **ZF Net**. The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the ZFNet (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.
- **GoogLeNet**. The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google. Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M). Additionally, this paper uses Average Pooling instead of Fully Connected layers at the top of the ConvNet, eliminating a large amount of parameters that do not seem to matter much. There are also several followup versions to the GoogLeNet, most recently Inception-v4.
- **VGGNet**. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. Their pretrained model is available for plug and play use in Caffe. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.
- **ResNet**. Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015. It features special *skip connections* and a heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network. The reader is also referred to Kaiming's presentation (video, slides), and some recent experiments that reproduce these networks in Torch. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 10, 2016). In particular, also see more recent developments that tweak the original architecture from Kaiming He et al. Identity Mappings in Deep Residual Networks (published March 2016).

Image Segmentation

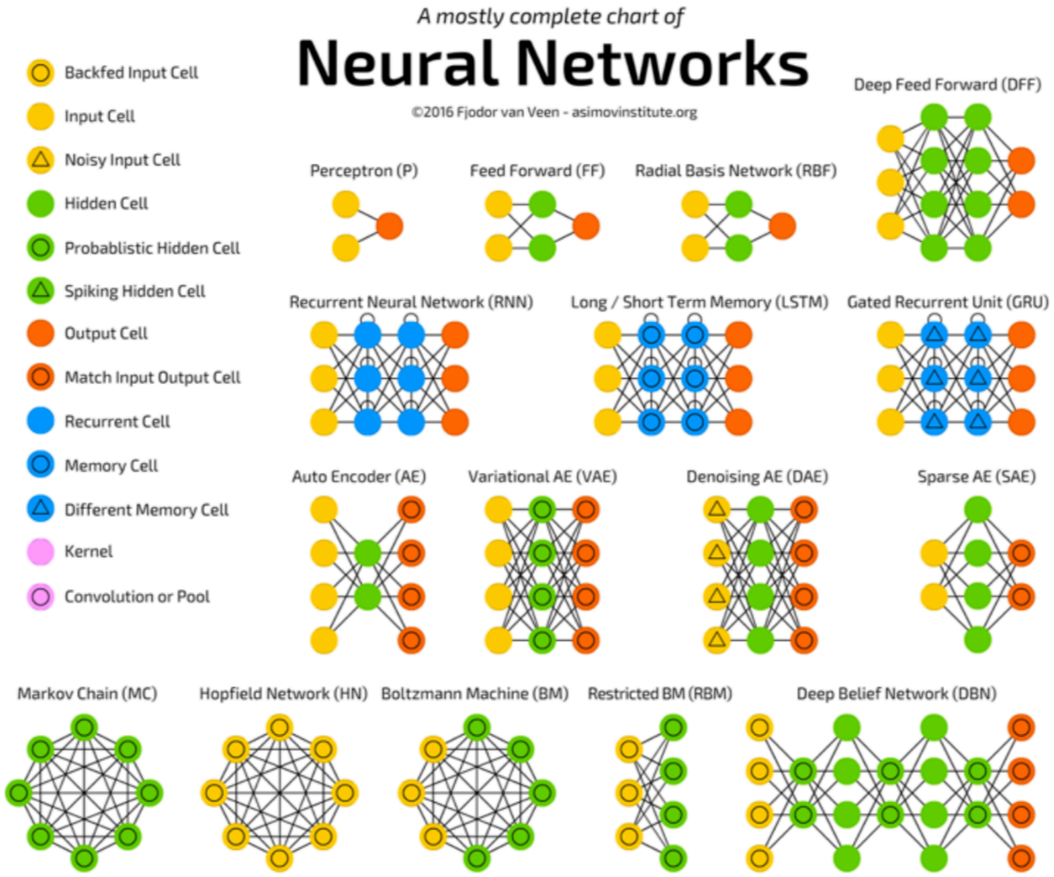
Labeling every pixel with the object it belongs to

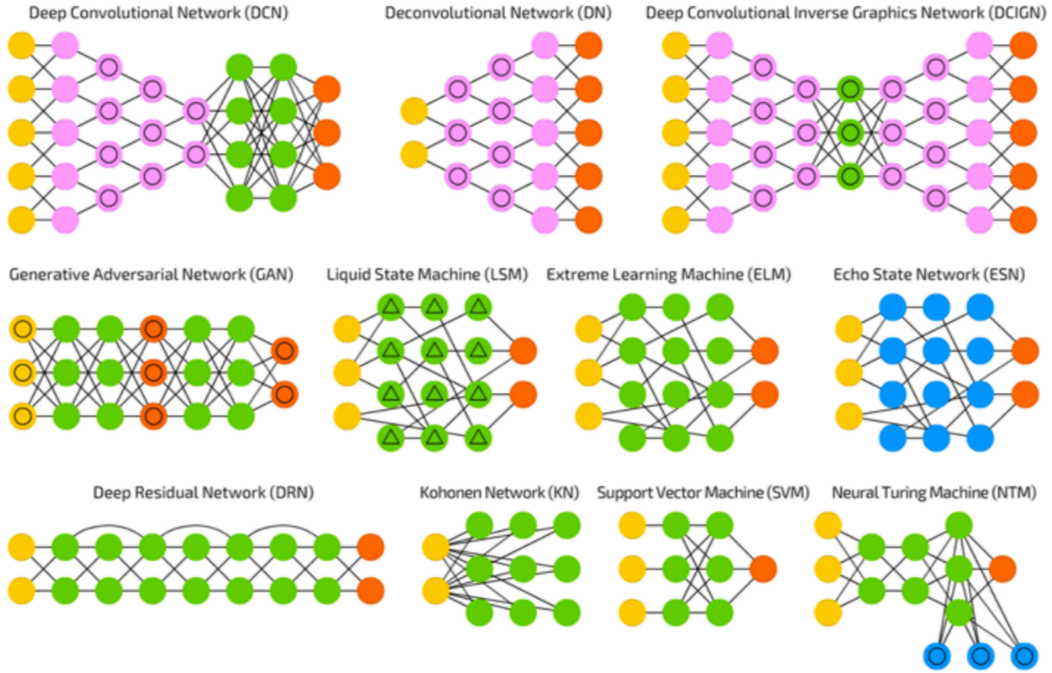


Clement Farabet, Camille Couprie, Laurent Najman and Yann LeCun: Learning Hierarchical Features for Scene Labeling, IEEE Transactions on Pattern Analysis and Machine Intelligence, August, 2013

https://www.youtube.com/watch?v=KkNhdlNs13U

Neural Network Architectures





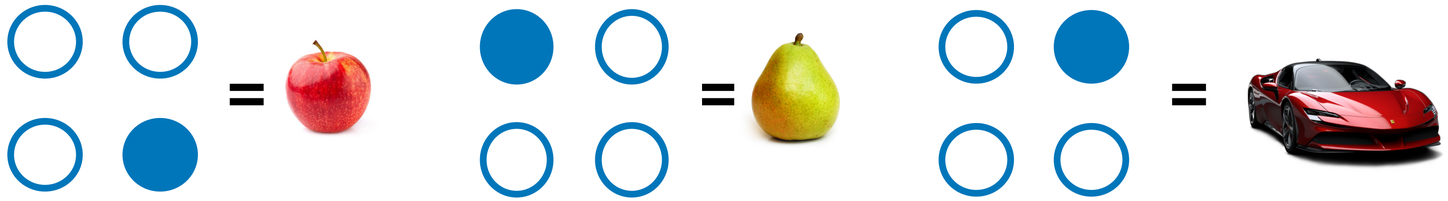
Source: http://www.asimovinstitute.org/neural-network-zoo/

Basis of DL: Distributed Representations

- Sparse: (1.54,0,0,0,0) Dense: (1.23,1.91,1.54,1.76,1.36)
- Sparse and non-distributed: (1,0,0,0,0) (0,1,0,0,0) (0,0,1,0,0) (0,0,0,1,0) (0,0,0,0,1)
  - Up to N
- Sparse and distributed: (1,0,1,1,0). (1,1,1,0,0). (1,0,0,1,1). (0,0,1,1,1) …
  - Up to 2^N

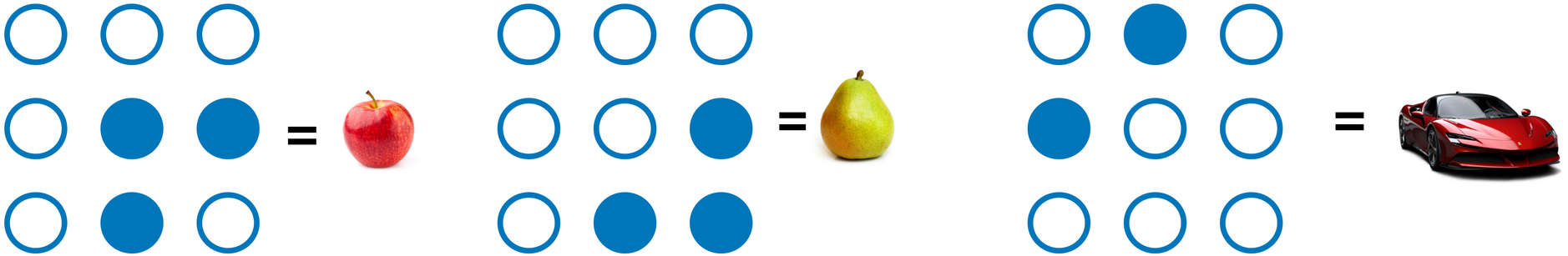
Distributed Representations

Not distributed:



Limited capacity

Distributed (a many-to-many relationship):



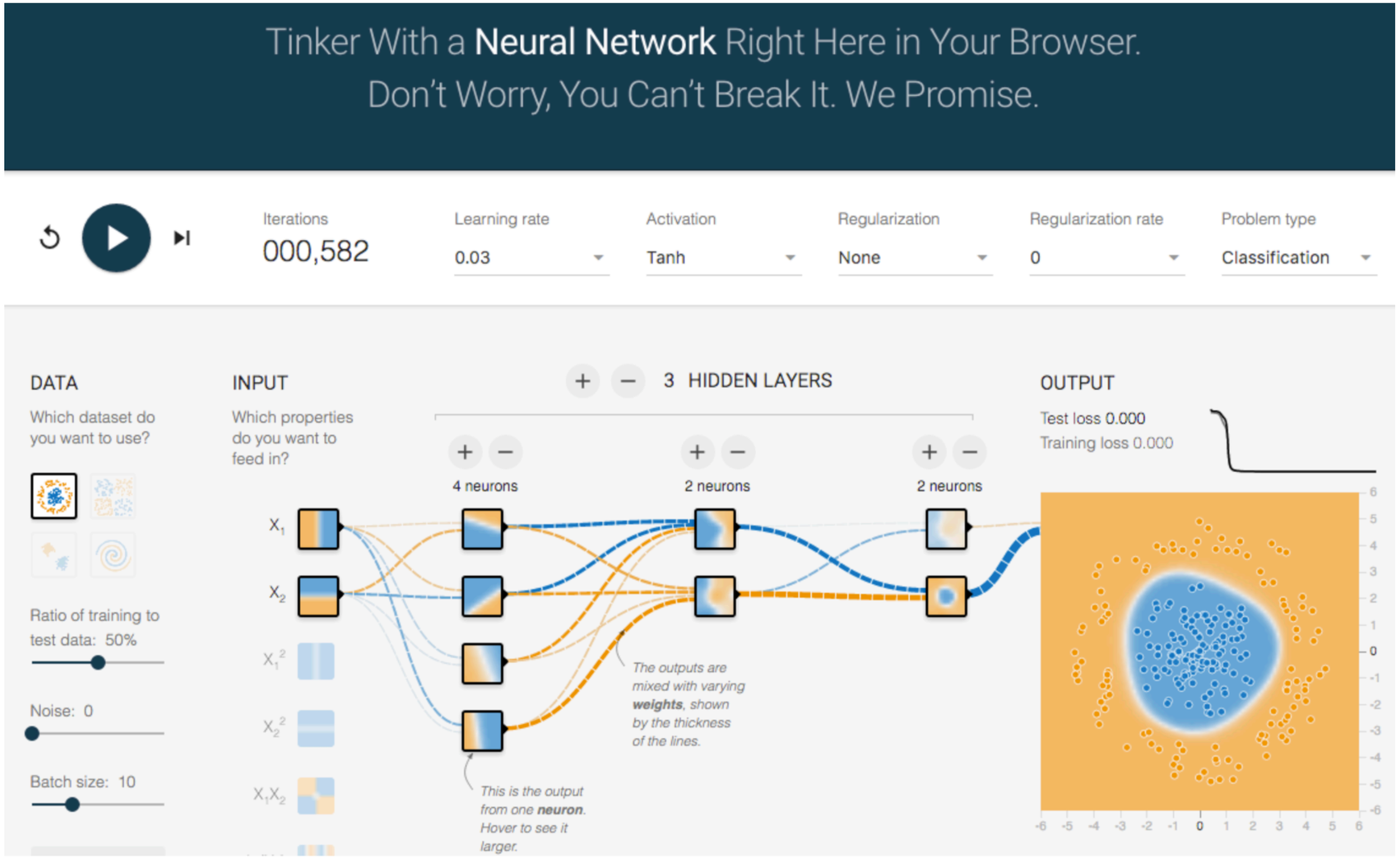
Sparse distributed representations can be used to represent probabilities

Representation Learning

- We wont to compress input data to a small feature space with high signal-to-noise
- Kernel methods may build on top of these, but suffer from the curse of dimensionality, since they only exploit *local-generalisation*
- Good representations probably posses a number of other priors
- Deep learning is more efficient: allows to represent certain functions with fewer parameters
- Abstraction (like produced by pooling): has more invariance to input, is more non-linear, allows wider generalisation
- But we want to disentangle factors of variation (like face direction, lightning, expression)

Tensorflow playground

Hands On Exercise



http://playground.tensorflow.org/

Things to Try

Hands On Exercise

- How to evaluate: Loss on test sample; output distribution (use “Discretize output”)
- See how results improve when running over epochs
- Compare results between different data sets
- Compare results between different activation functions
- Vary the number of hidden layers
- Vary the number of neurons per layer
- Use different input features

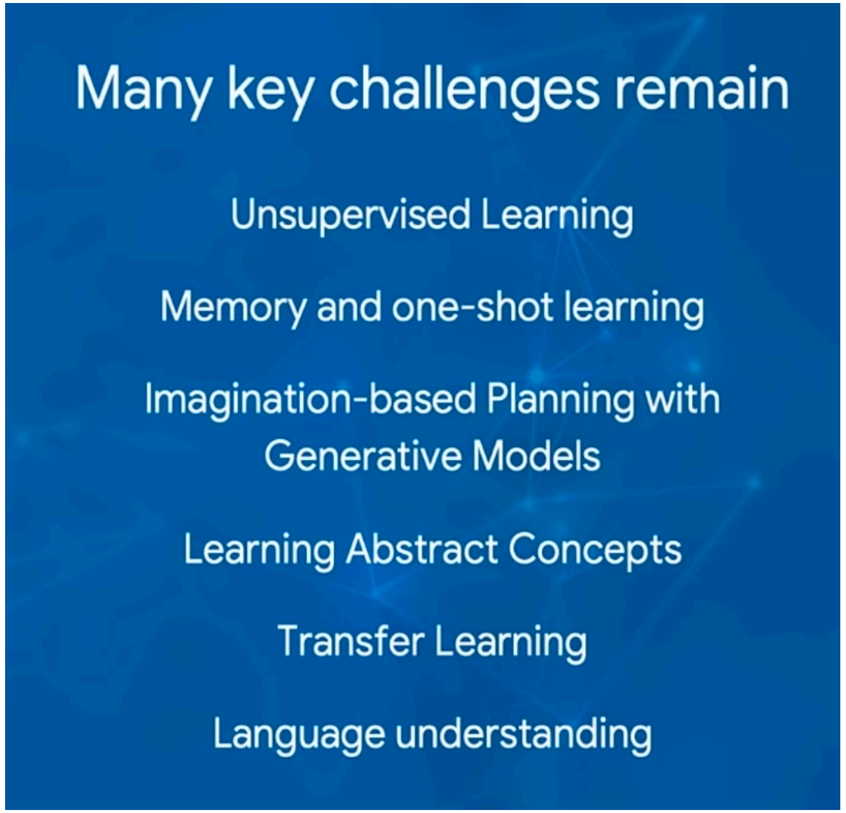
Observations

- ReLu learns faster
- Higher training set ratio learns faster
- Adding layers makes learning slower
- Linear unit can not solve entangled problems, also more layers/more neurons do not help
- How many neurons are needed to solve a problem: circle needs only one layer; spiral needs 2 layers
- Sometimes spiky learning curves appear

Deep Learning Frontiers

- We don’t know the minimum number of training examples needed to get a certain level of accuracy. All we know is, the more the better!
- Lack of interpretability of the models that are learned.
- We just don’t know how to do unsupervised learning with deep learning (besides autoencoders).
- Only learn specific cause effect relation. Can not predict behavior of other complex systems.
- No meta-level reasoning.

Source: Y. LeCun



Source: Demis Habits

References

- Martin Gorner (2017), TensorFlow and Deep Learning without a PhD, Part 1 (Google Cloud Next '17), https://www.youtube.com/watch?v=u4alGiomYP4
- Martin Gorner (2017), TensorFlow and Deep Learning without a PhD, Part 2 (Google Cloud Next '17), https://www.youtube.com/watch?v=fTUwdXUFfI8
- Martin Gorner (2017), TensorFlow and Deep Learning without a PhD, https://goo.gl/pHeXe7, https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist
- Deep Learning Basics: Neural Networks Demystified, https://www.youtube.com/playlist?list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
- Deep Learning SIMPLIFIED, https://www.youtube.com/playlist?list=PLjJh1vlSEYgvGod9wWiydumYl8hOXixNu
- 3Blue1Brown (2017), But what *is* a Neural Network? | Chapter 1, deep learning, https://www.youtube.com/watch?v=aircAruvnKk
- 3Blue1Brown (2017), Gradient descent, how neural networks learn | Chapter 2, deep learning, https://www.youtube.com/watch?v=IHZwWFHWa-w
- 3Blue1Brown (2017), What is backpropagation really doing? | Chapter 3, deep learning, https://www.youtube.com/watch?v=Ilg3gGewQ5U
- TensorFlow: https://www.tensorflow.org/
- Keras: http://keras.io/
- Deep Learning Studio: Cloud platform for designing Deep Learning AI without programming, http://deepcognition.ai/
- Natural Language Processing with Deep Learning (Winter 2017), https://www.youtube.com/playlist?list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6
- Udacity, Deep Learning, https://www.youtube.com/playlist?list=PLAwxTw4SYaPn_OWPFT9ulXLuQrImzHfOV
- http://p.migdal.pl/2017/04/30/teaching-deep-learning.html
- https://github.com/leriomaggio/deep-learning-keras-tensorflow