NEURAL NETWORKS

Deep learning = Deep neural networks =
neural networks

DNNs (Deep Neural Networks)

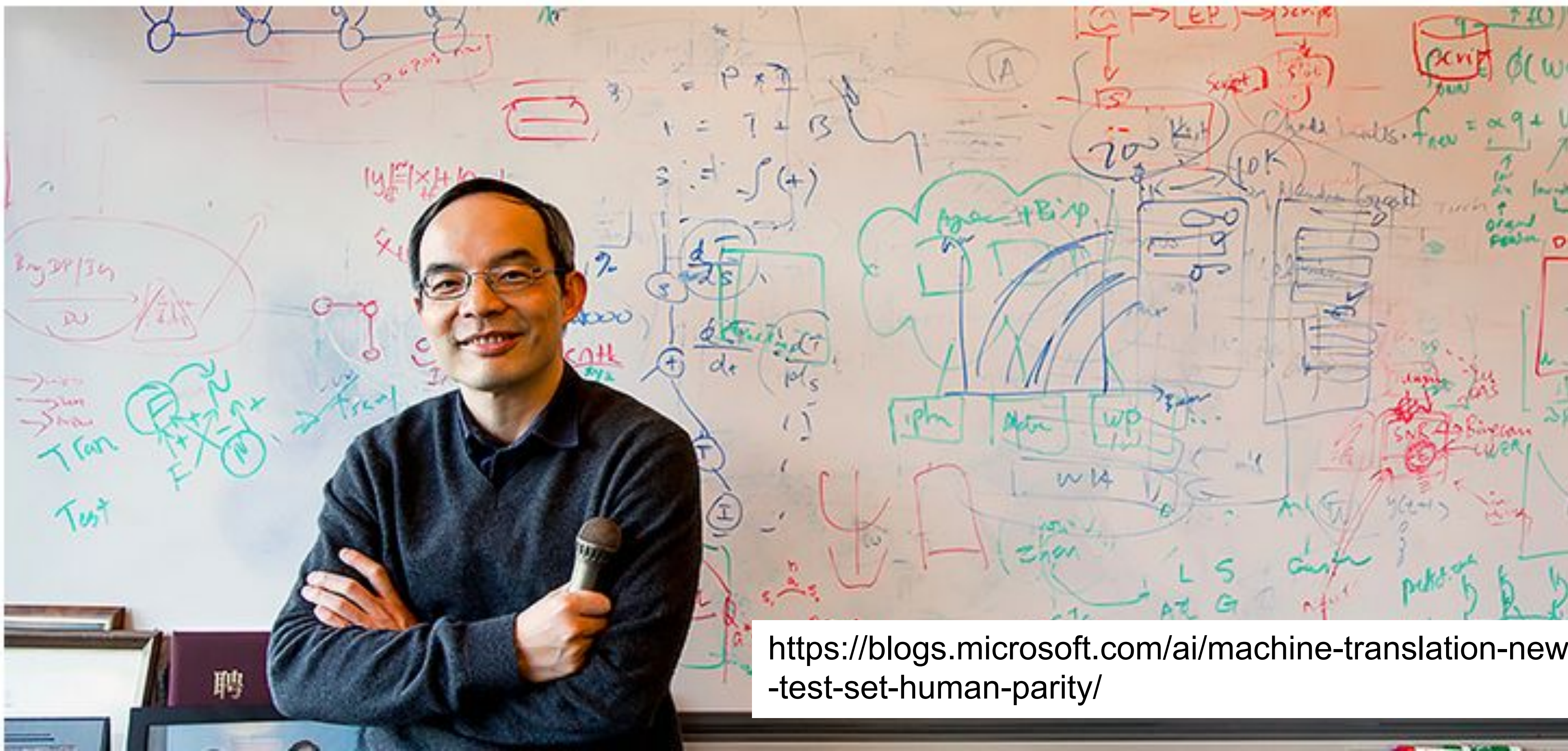
- Why deep learning?
- Greatly improved performance in ASR and other tasks (Computer Vision, Robotics, Machine Translation, NLP, etc.)
- Surpassed human performance in many tasks

Task	Previous state-of-the-art	Deep learning (2012)	Deep learning (2019)
Switchboard	23.6%	16.1%	5.0%
Google voice search	16.0%	12.3%	4.9%
MOOC (Thai)	38.7%		19.6%

Geoffrey Hinton, Deep Neural Networks for Acoustic Modeling in Speech Recognition, 2012

Microsoft reaches a historic milestone, using AI to match human performance in translating news from Chinese to English

Mar 14, 2018 | Allison Linn



https://blogs.microsoft.com/ai/machine-translation-news-test-set-human-parity/

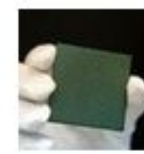
Google's AlphaGo Defeats Chinese Go Master in Win for A.I.

点击查看本文中文版

By **PAUL MOZUR** MAY 23, 2017

RELATED COVERAGE

A.I. I
Repla



China
FEB. 3,

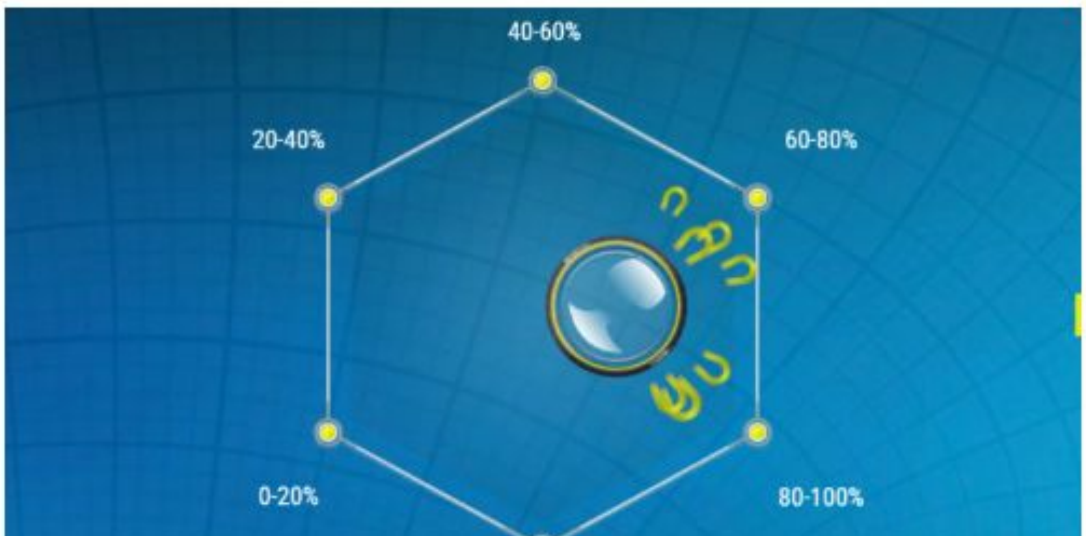
THE FU
The F

Mast
Goog

https://www.nytimes.com/2017/05/23/business/google-deepmind-alphago-go-champion-defeat.html

Artificial swarm intelligence diagnoses pneumonia better than individual computer or doctor

by Fan Liu — September 27, 2018 0



Courtesy of Unanimous AI

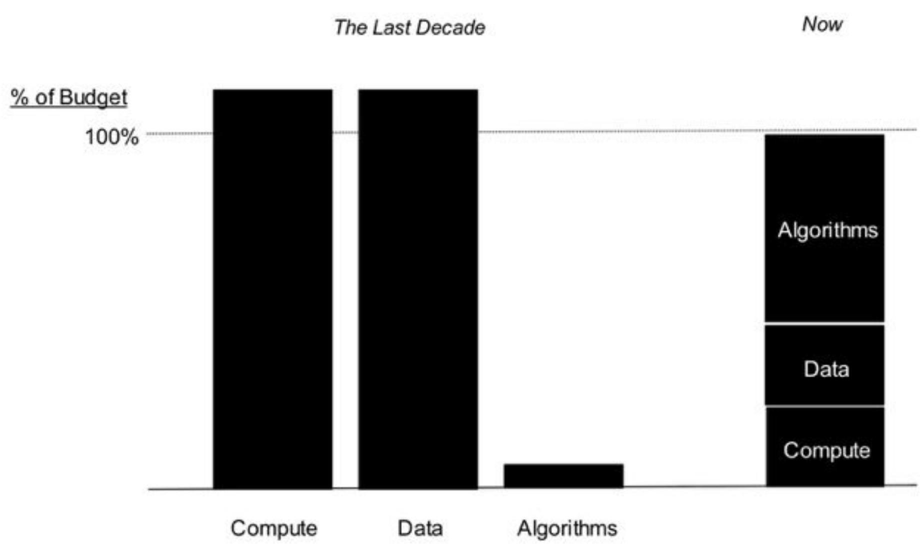
Popular Posts

Artificial swarm intelligence diagnoses pneumonia better than individual computer or

https://www.stanforddaily.com/2018/09/27/artificial-swarm-intelligence-diagnoses-pneumonia-better-than-individual-computer-or-doctor/

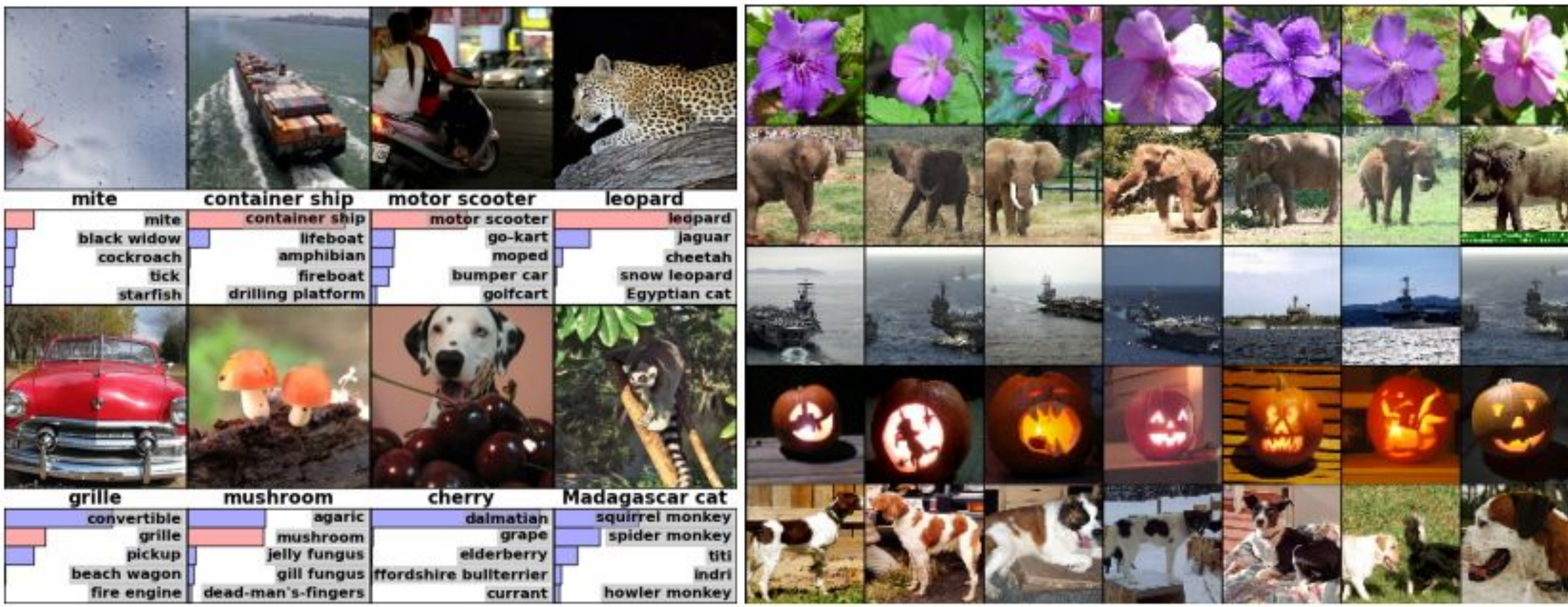
Why now

- Neural Networks has been around since 1990s
- Big data – DNN can take advantage of large amounts of data better than other models
- GPU – Enable training bigger models possible
- Deep – Easier to avoid bad local minima when the model is large



http://www.kdnuggets.com/2017/06/practical-guide-machine-learning-understand-differentiate-apply.html

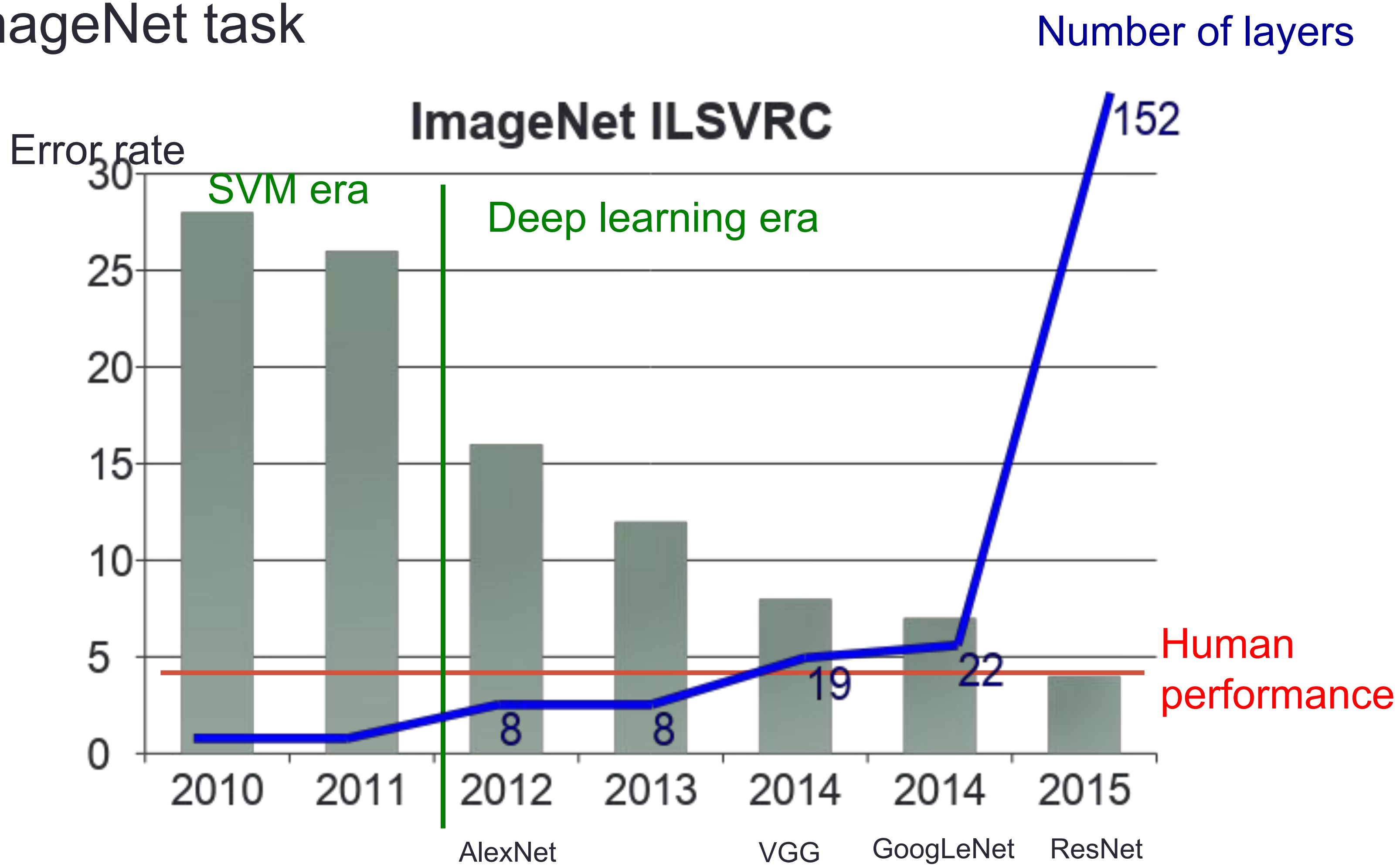
ImageNet - Object classification



Alex, Krizhevsky, Imagenet classification with deep convolutional neural networks, 2012

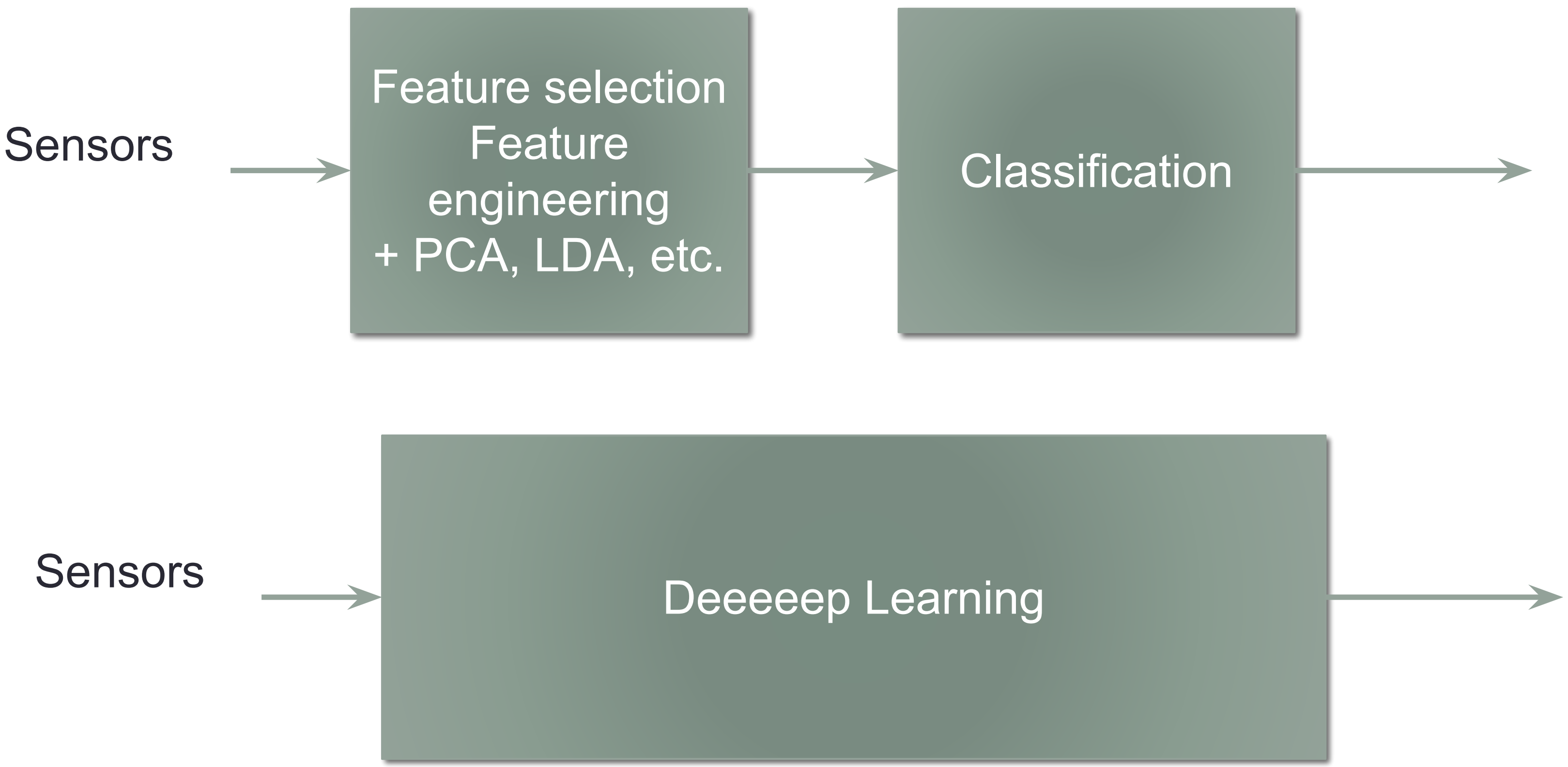
Wider and deeper networks

- ImageNet task

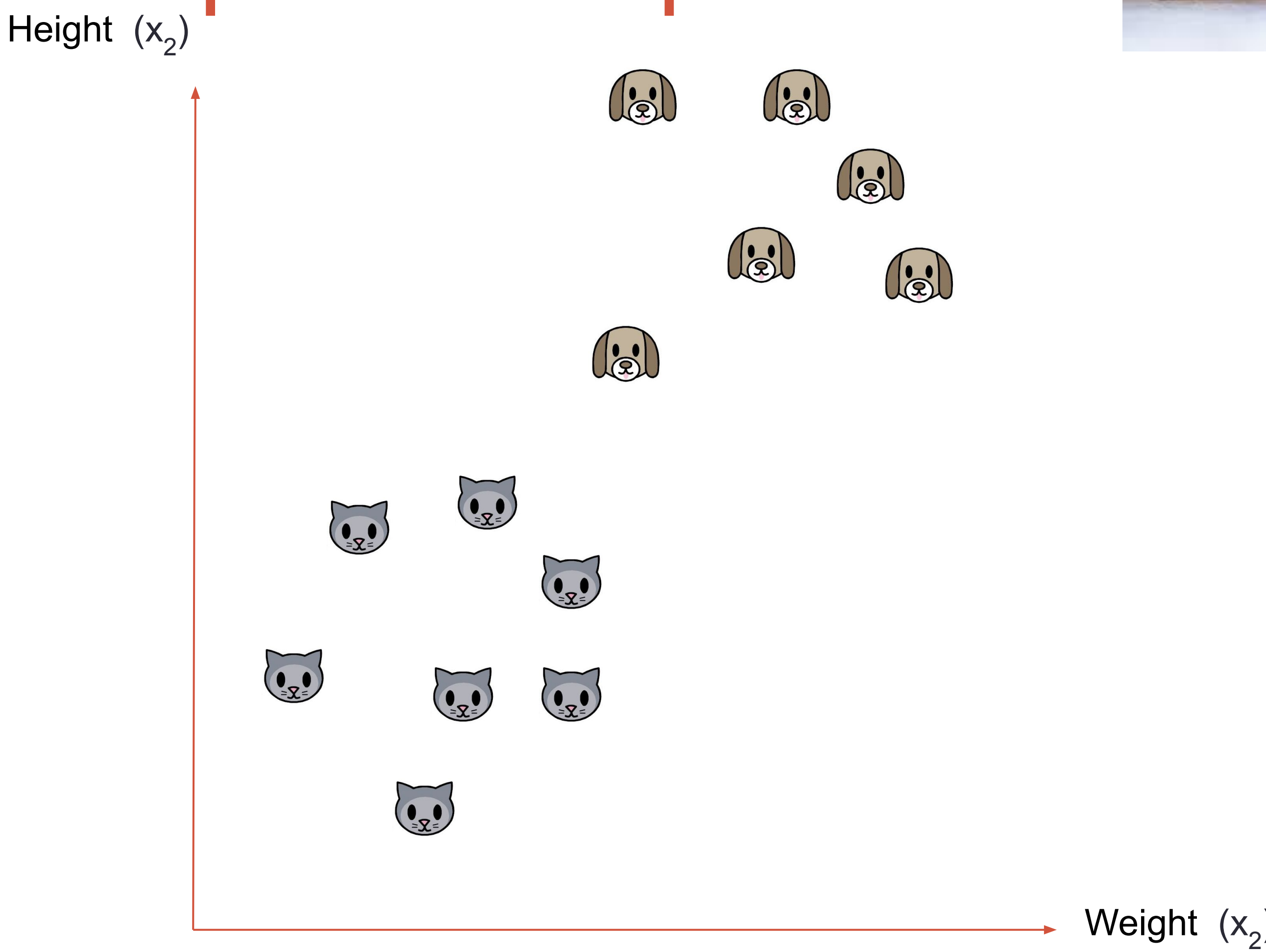


Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

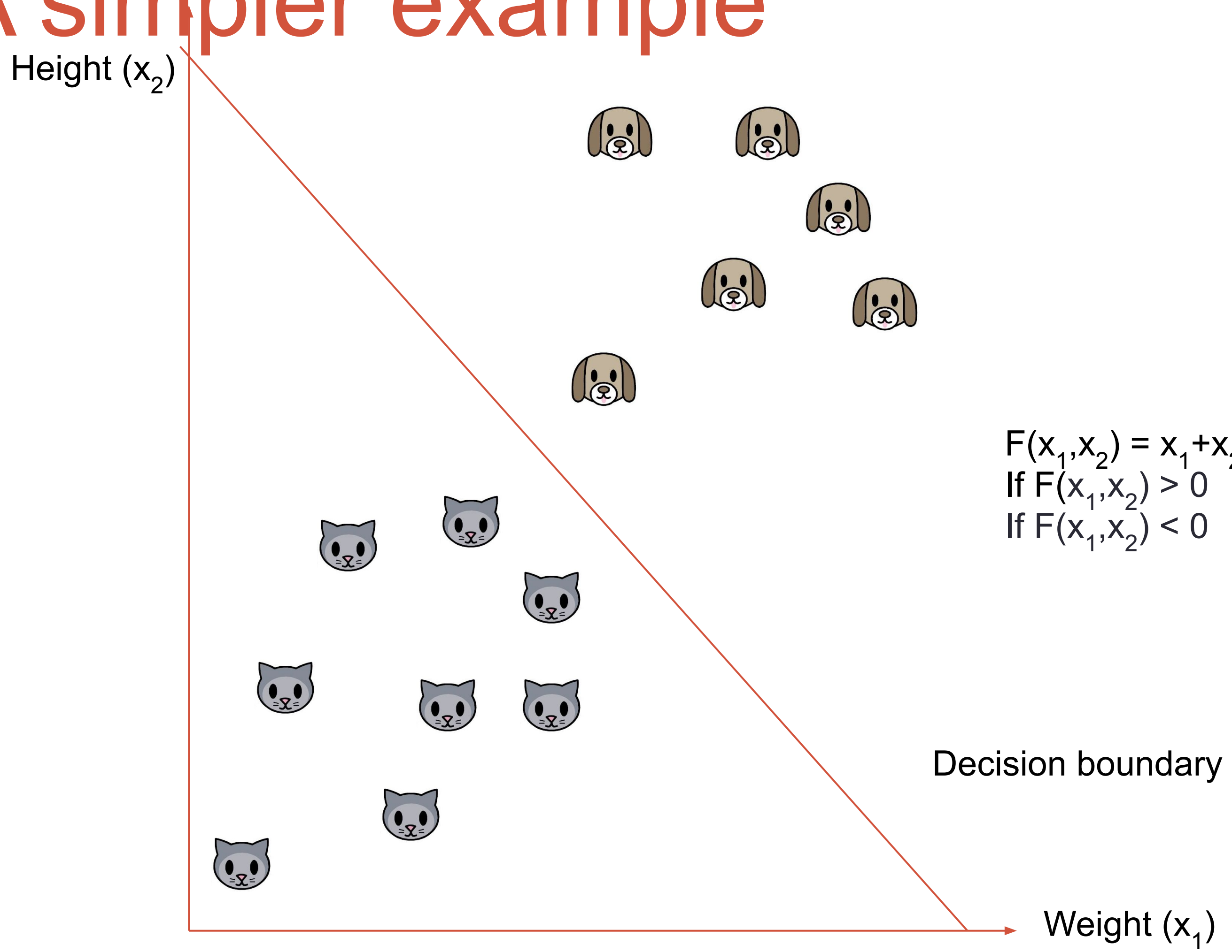
Traditional VS Deep learning



A simpler example



A simpler example

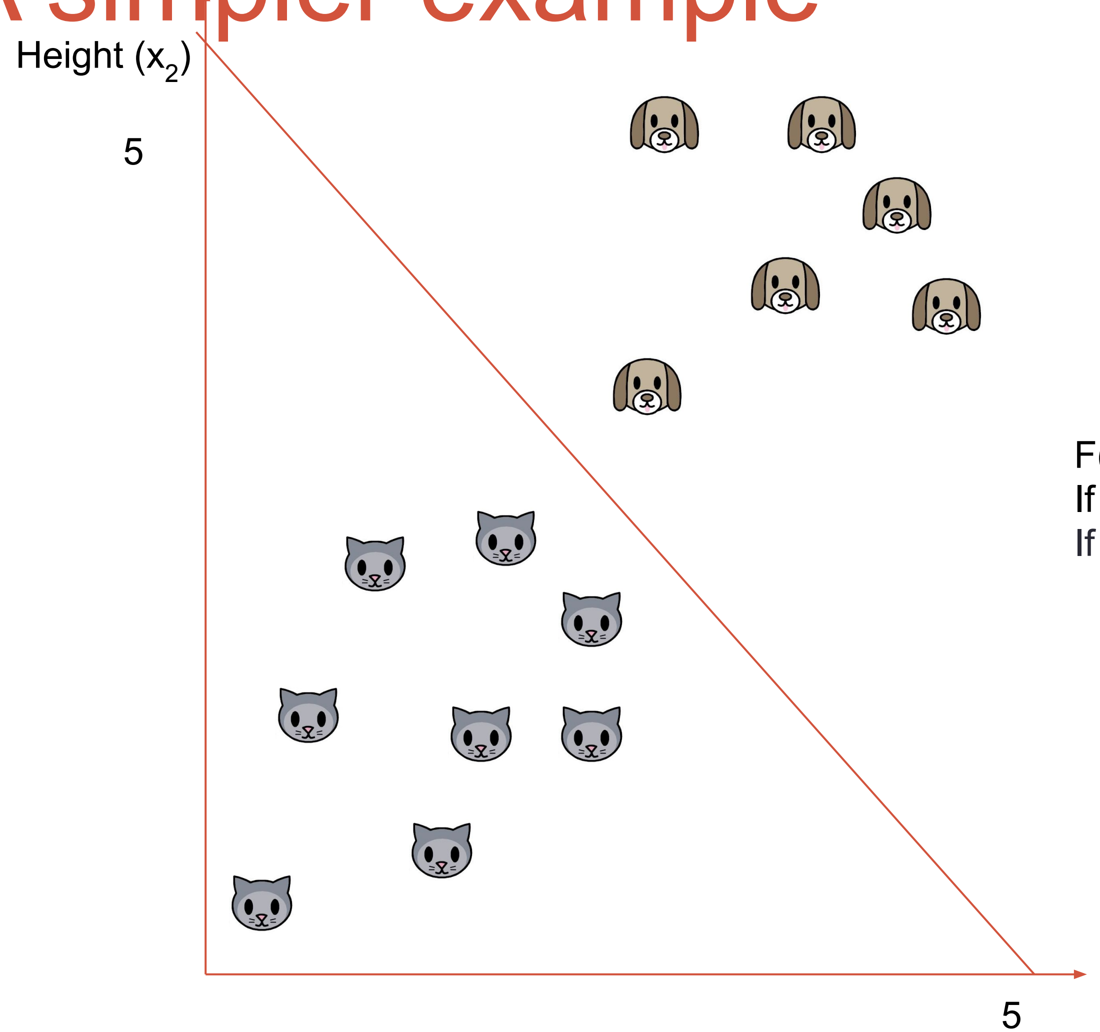


$F(x_1,x_2) = x_1+x_2-5$

If $F(x_1,x_2) > 0$ answer dog

If $F(x_1,x_2) < 0$ answer cat

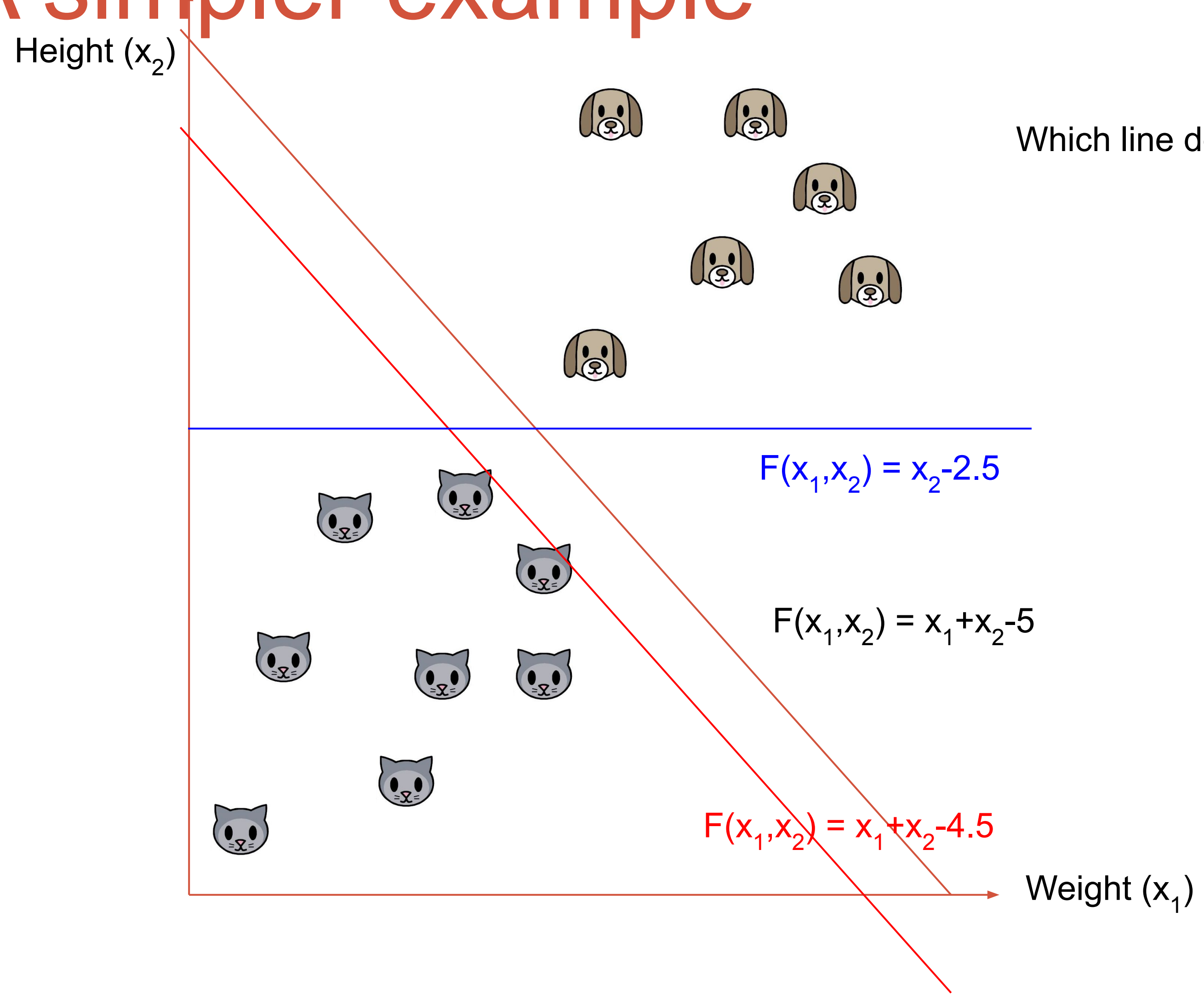
A simpler example



$F(x_1,x_2) = x_1+x_2-5$
If $F(x_1,x_2) > 0$ answer dog
If $F(x_1,x_2) < 0$ answer cat

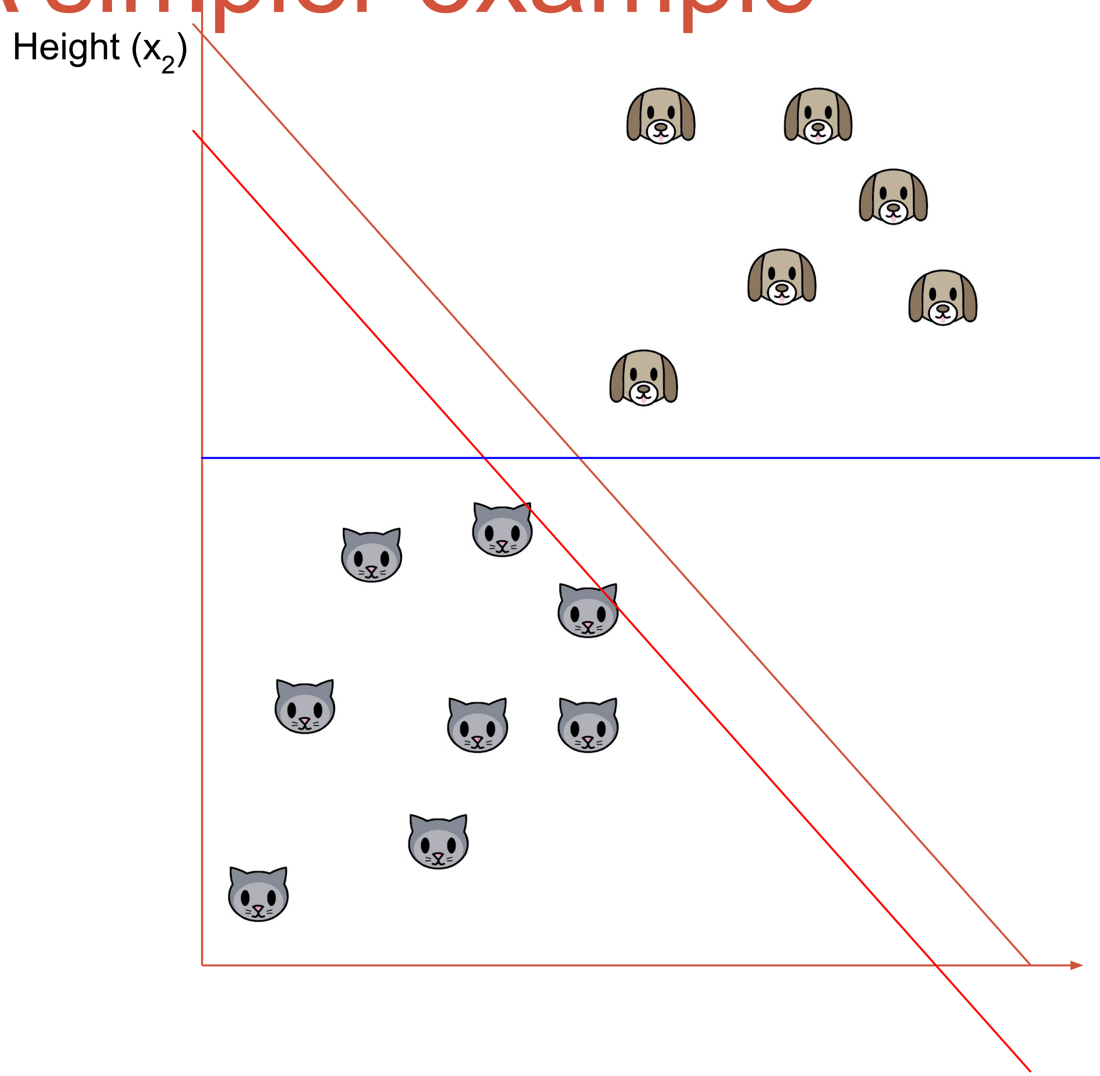
Linear classifier
Ex: logistic regression

A simpler example



Which line do we select?

A simpler example



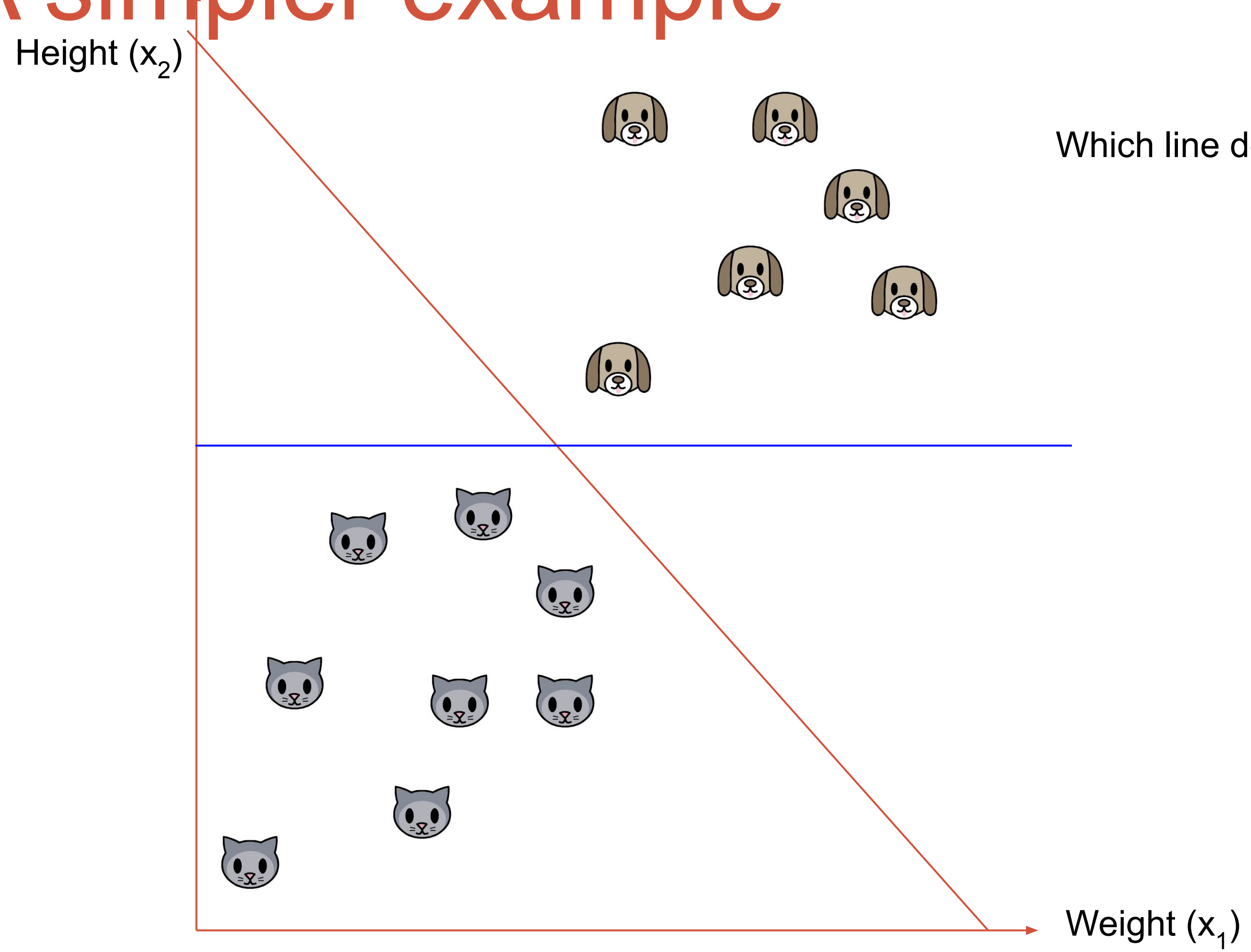
Which line do we select?

Create an objective function and optimize it

$F(x_1, x_2)$ - $F(x_1, x_2)$

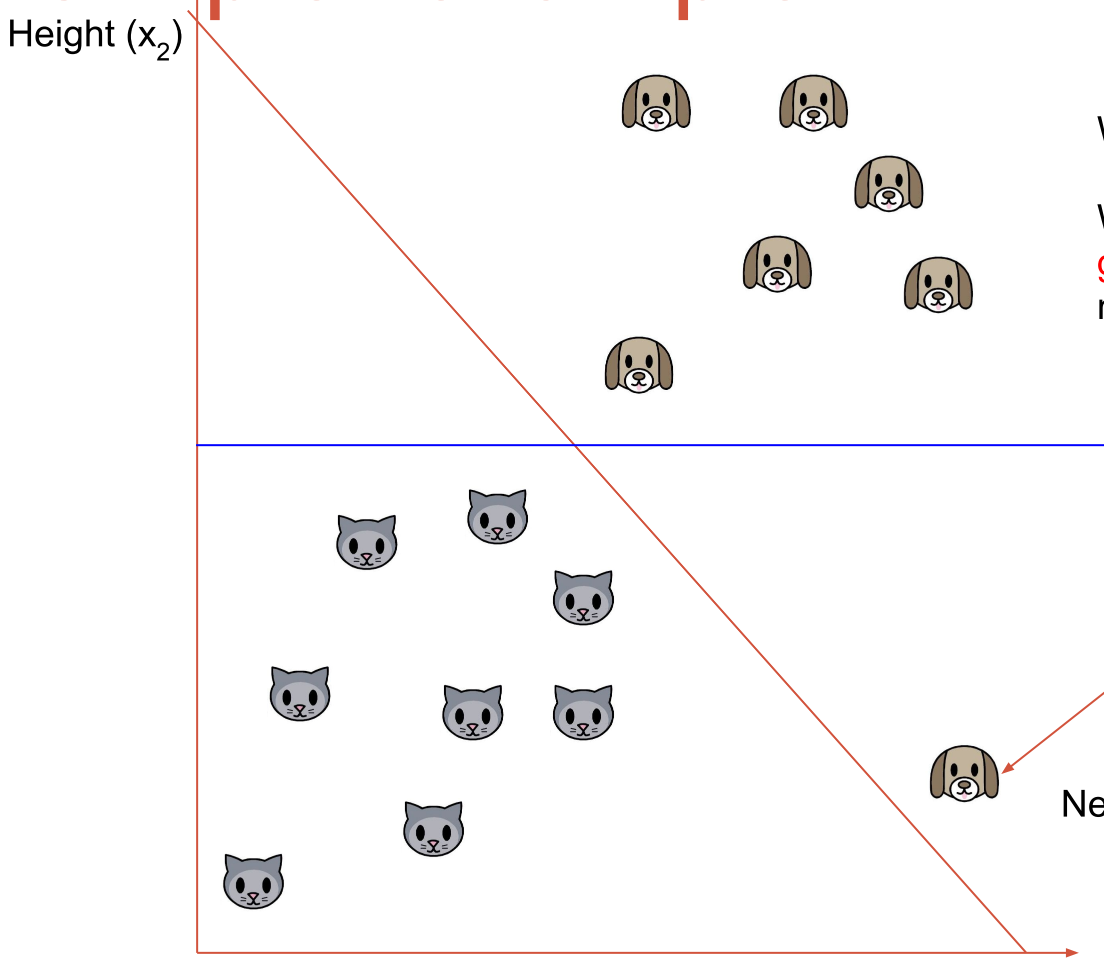
For all 🐶 For all 🐱

A simpler example



Which line do we select?

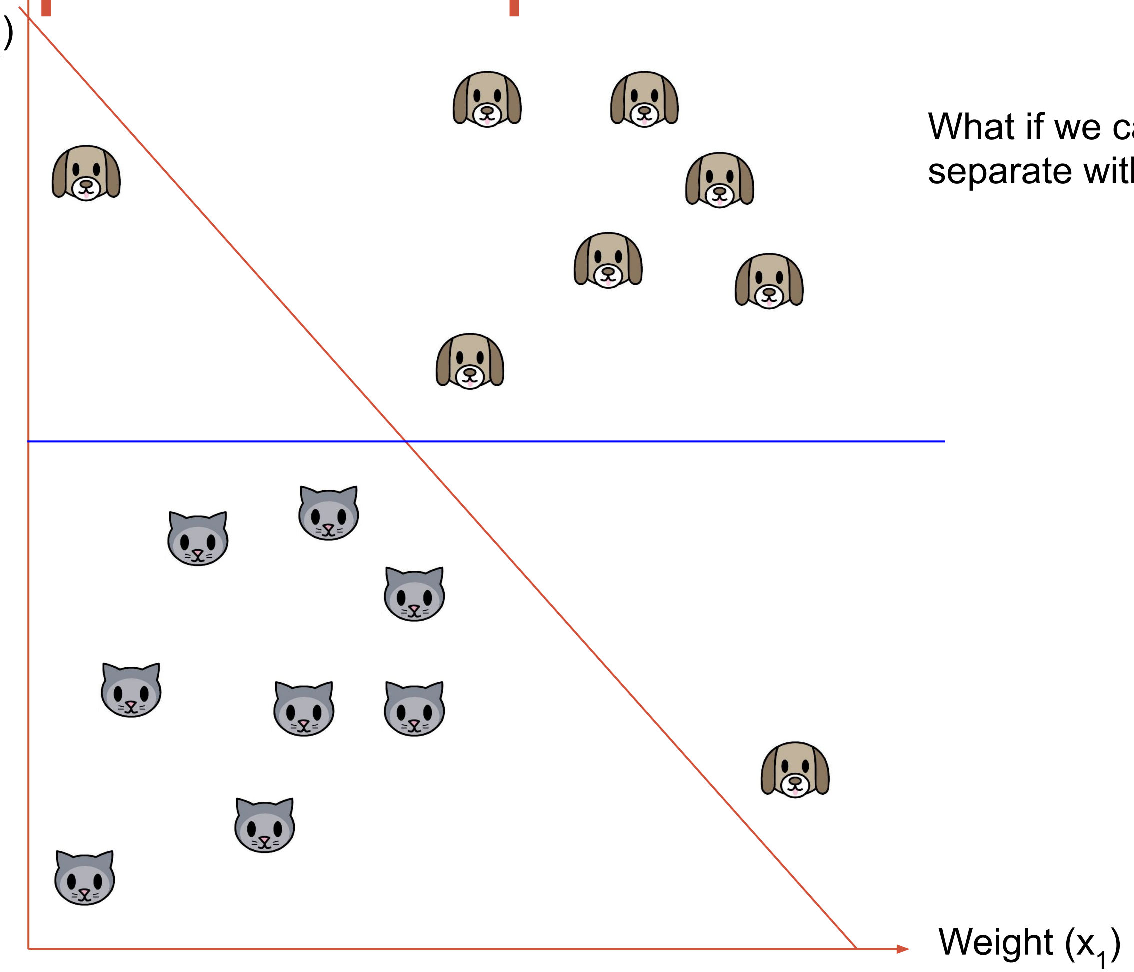
A simpler example



Which line do we select?

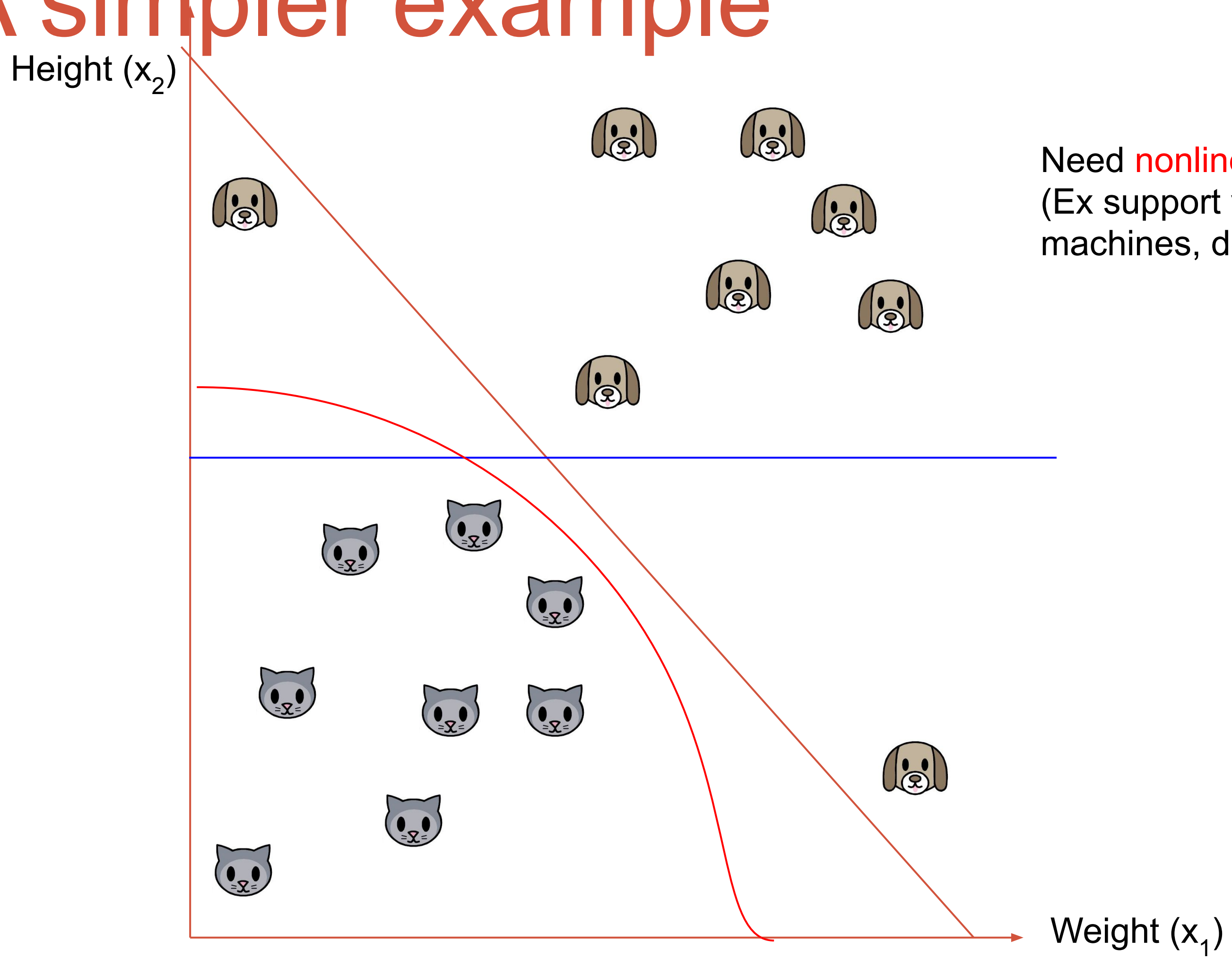
We want the line that generalize better (works on new data)

A simpler example



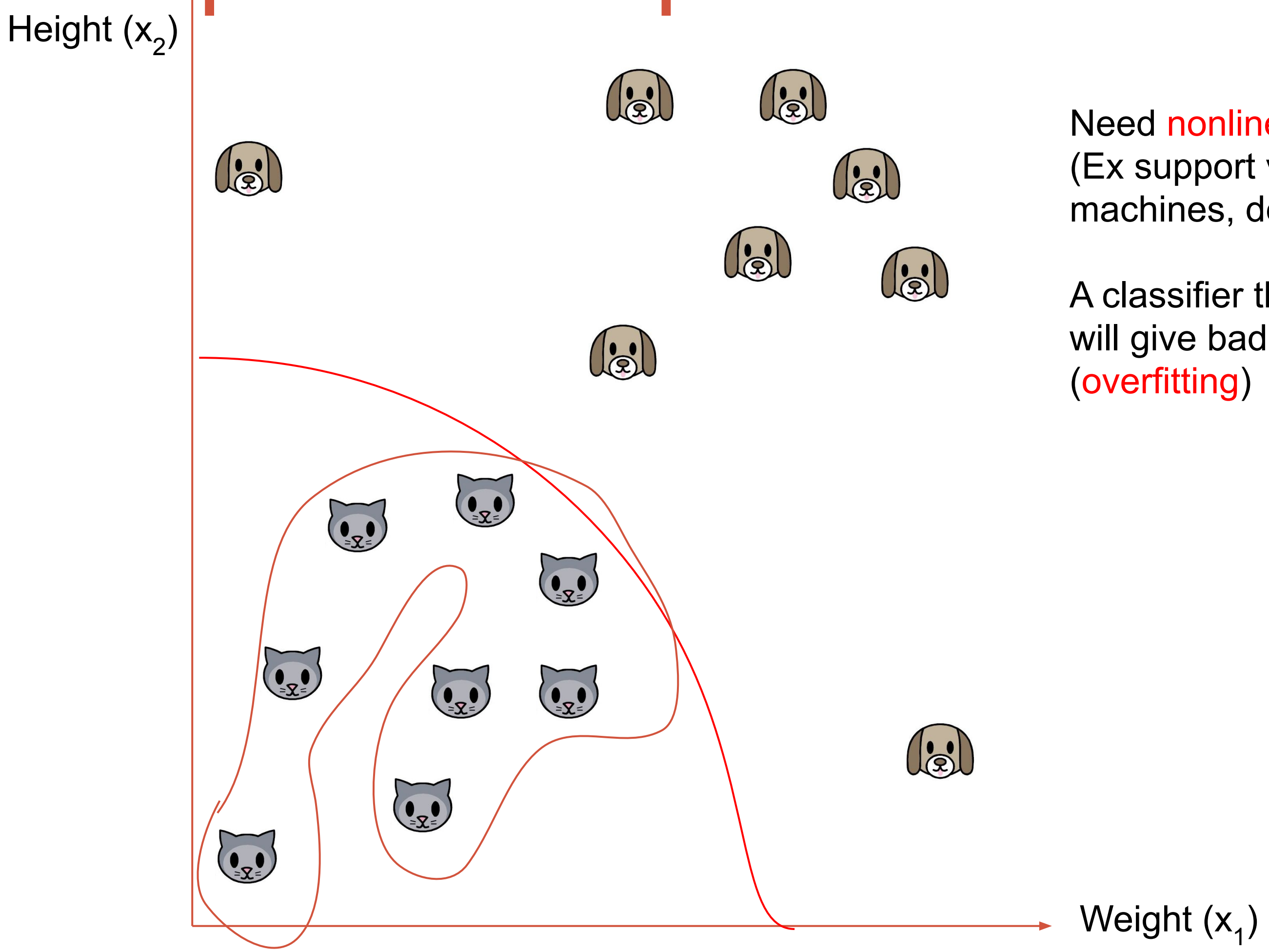
What if we cannot separate with a line?

A simpler example



Need nonlinear classifiers (Ex support vector machines, deep learning)

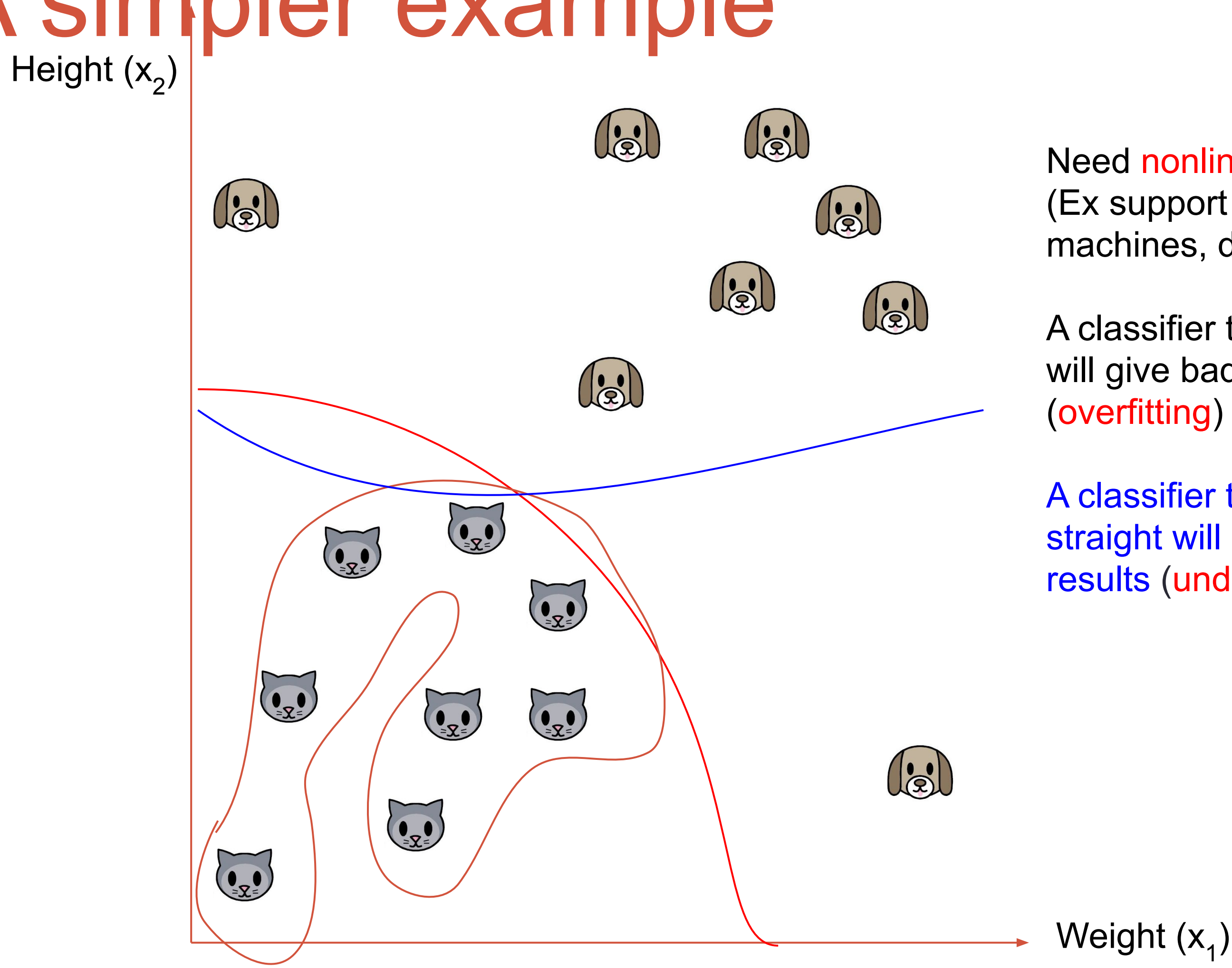
A simpler example



Need nonlinear classifiers (Ex support vector machines, deep learning)

A classifier that is too curvy will give bad results (overfitting)

A simpler example



Need nonlinear classifiers (Ex support vector machines, deep learning)

A classifier that is too curvy will give bad results (overfitting)

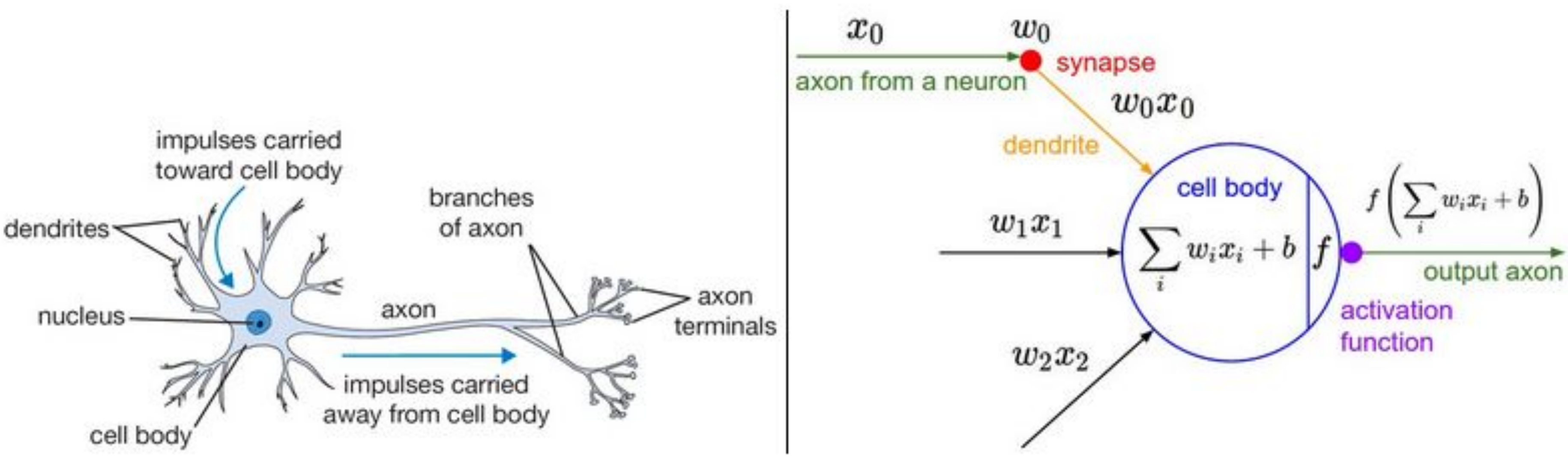
A classifier that is too straight will give bad results (underfitting)

Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- Demos
  - Tensorflow, Keras

Fully connected networks

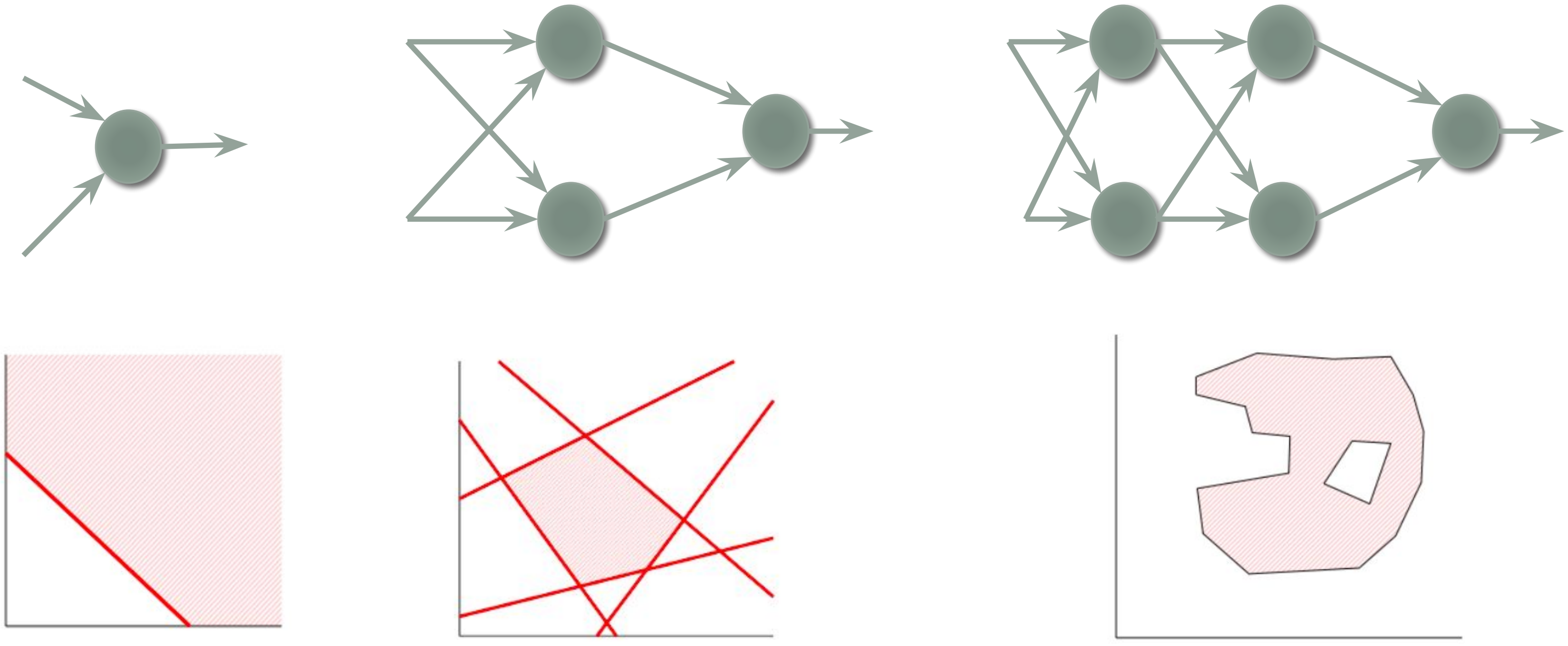
- Many names: feed forward networks or deep neural networks or multilayer perceptron or artificial neural networks
- Composed of multiple neurons



http://cs231n.github.io/neural-networks-1/

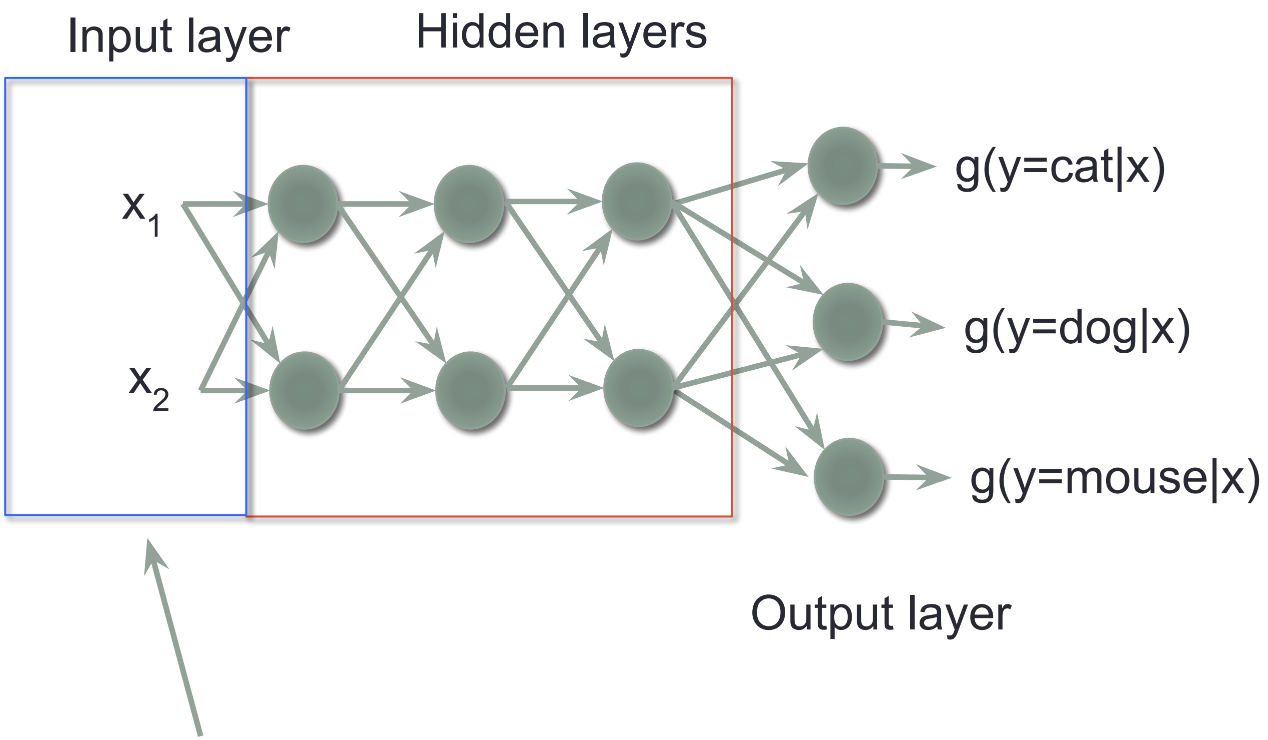
Combining neurons

- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting



Terminology

Deep in Deep neural networks means many hidden layers



Input should be scaled to have zero mean unit variance

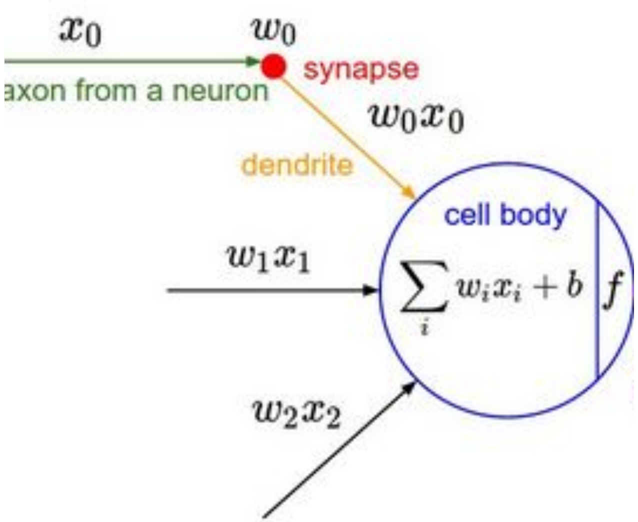
Matrices

- Inputs

features

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

X



W

weights

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$W^T X + b$

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b \end{bmatrix}$$

Matrices

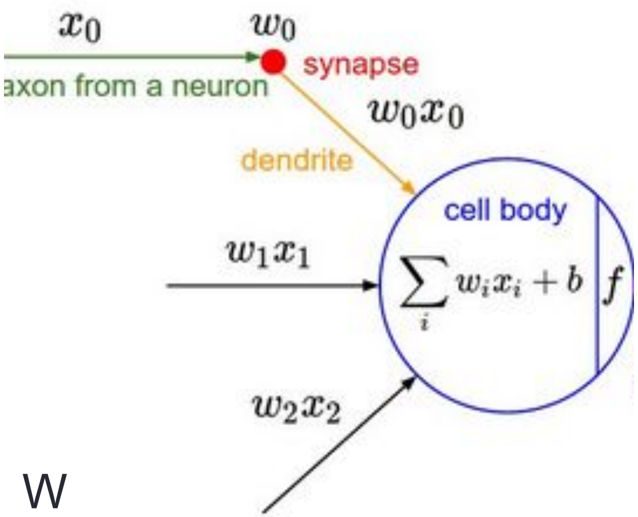
- Inputs

samples

features

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix}$$

X



W

weights

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$W^T X + b$

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix} + \begin{bmatrix} b \end{bmatrix}$$

Matrices

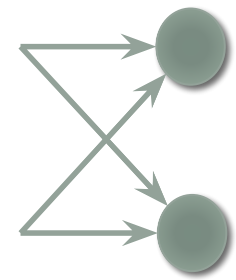
- Inputs

samples

features

$$\begin{matrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{matrix}$$

X



neorons

W

weights

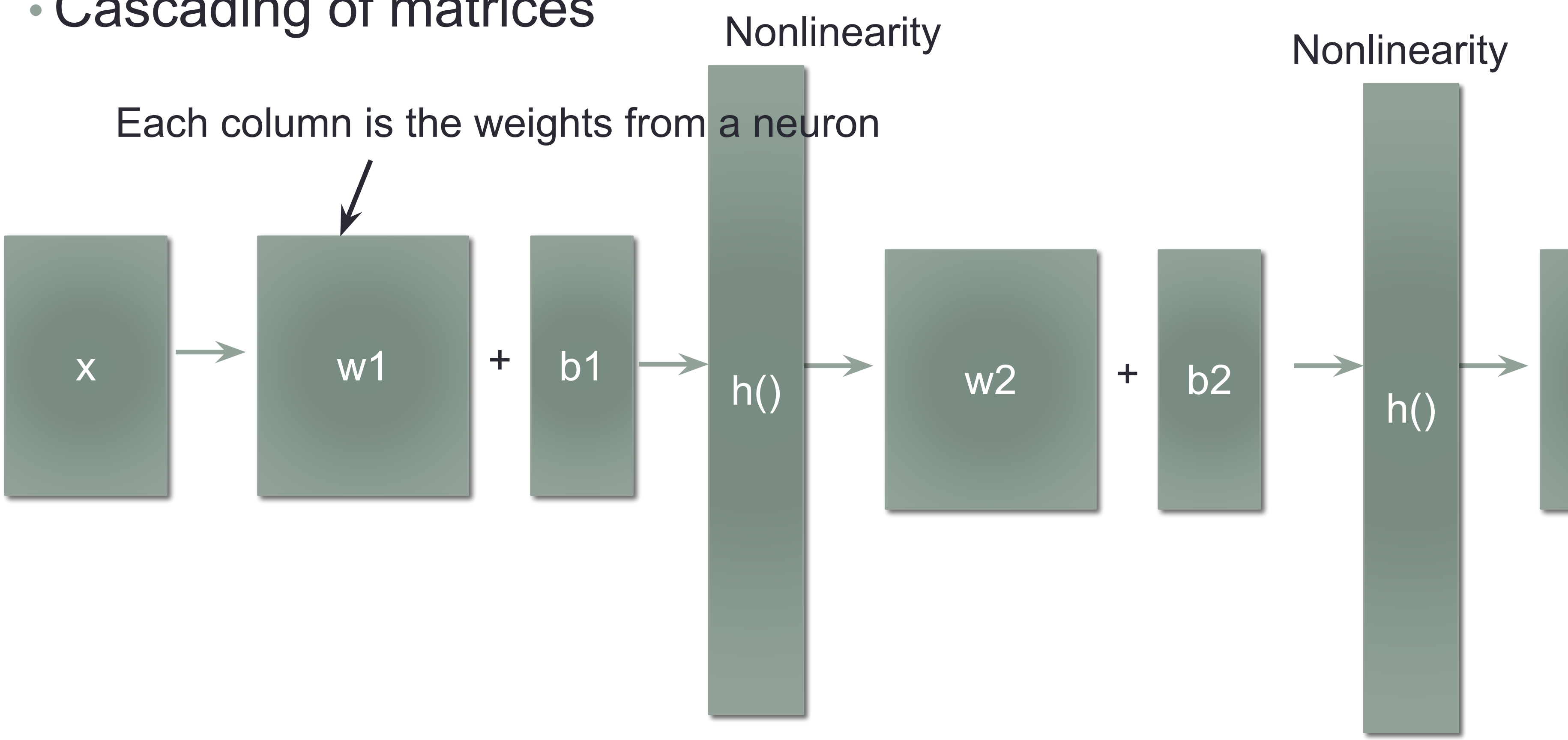
$$\begin{matrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{matrix}$$

$W^TX + b$

$$\begin{matrix} w_{11} & w_{21} & w_{31} \\ w_{21} & w_{22} & w_{23} \end{matrix} \quad \begin{matrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{matrix} \quad + \quad \begin{matrix} b_1 \\ b_2 \end{matrix}$$

More linear algebra

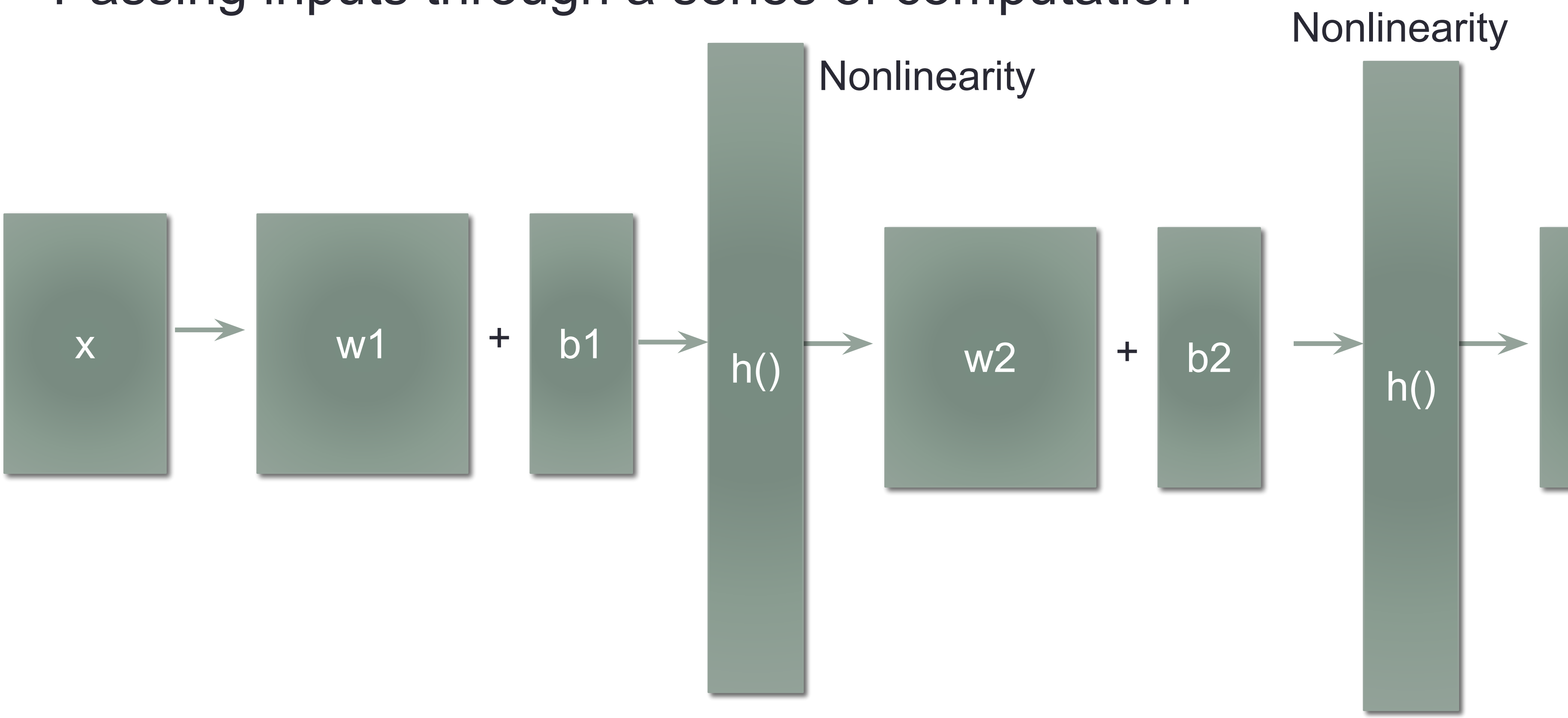
- Cascading of matrices



$h(W_2^T\, h(W_1^T X + \mathbf{b_1}) + \mathbf{b_2})$

Computation graph

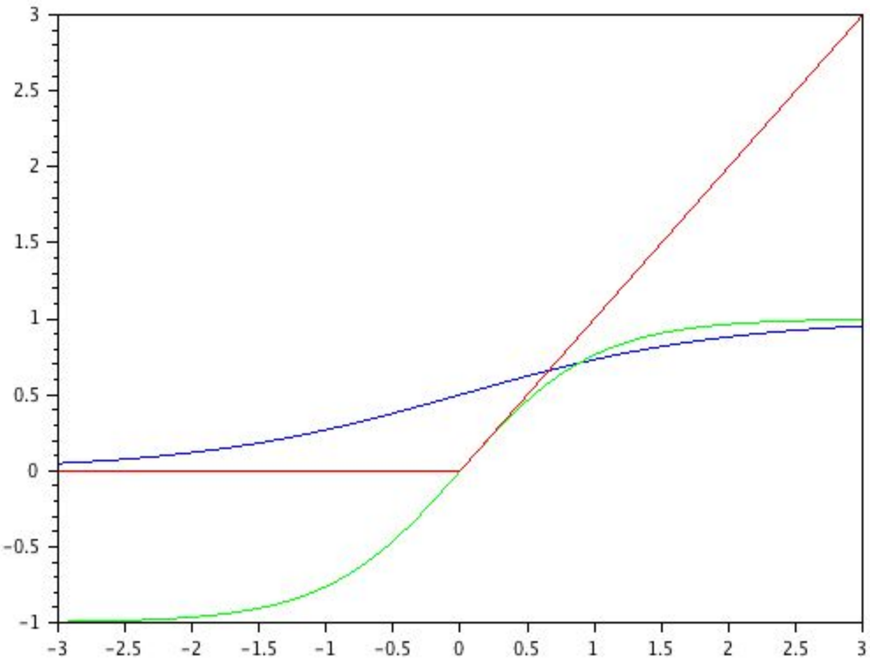
- Passing inputs through a series of computation



$h(W_2^T \, h(W_1^T X + \mathbf{b}_1) + \mathbf{b}_2)$

Non-linearity

- The Non-linearity is important in order to stack neurons
- Sigmoid or logistic function
- tanh
- Rectified Linear Unit (ReLU)
- Most popular is ReLU and its variants (Fast to train, and more stable)



Non-linearity

- Sigmoid

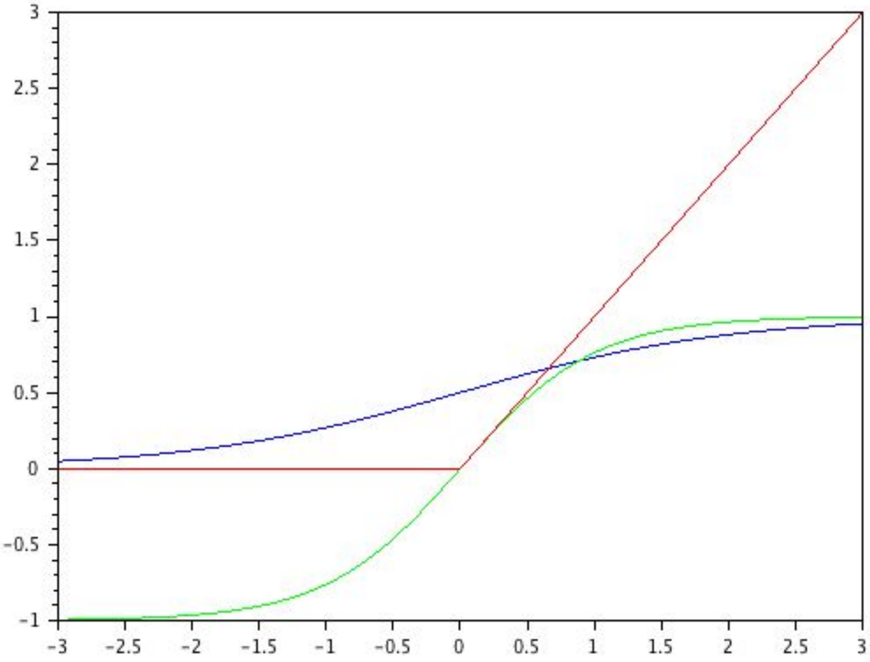
$$\frac{1}{1+e^{-x}}$$

- tanh

$$tanh(x)$$

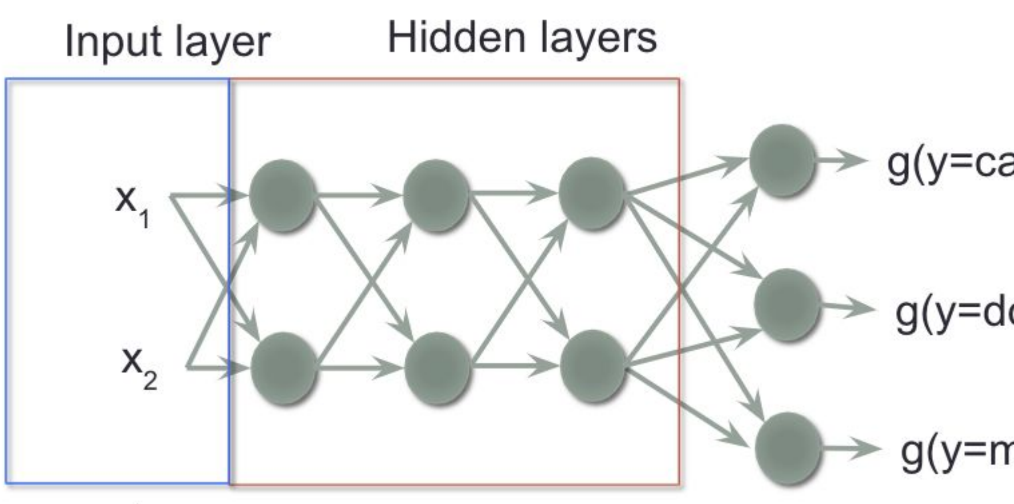
- Rectified Linear Unit (ReLU)

$$max(0,x)$$



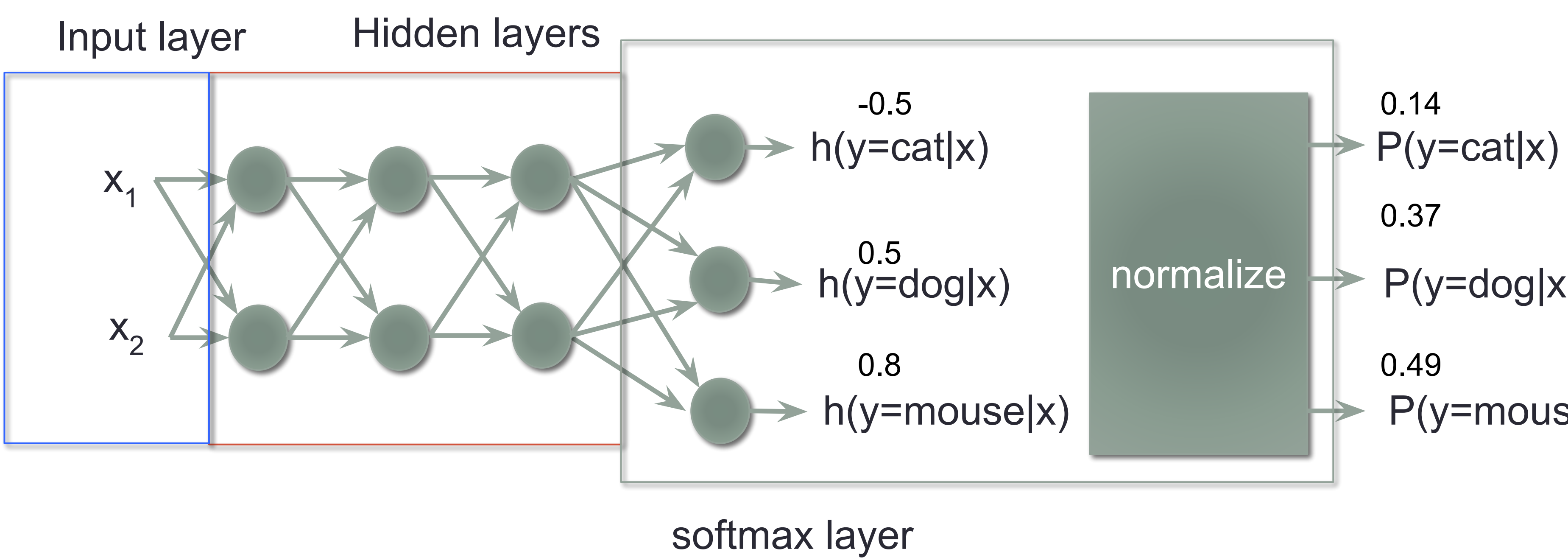
Output layer – Softmax layer

- We usually wants the output to mimic a probability function (0<=P<=1,sums to 1)
- Current setup has no such constraint
- The current output should have highest value for the correct class.
  - Value can be positive or negative number
- Takes the exponent
- Add a normalization



Softmax layer

$$P(y=j|x) = \frac{e^{h(y=j|x)}}{\Sigma_y e^{h(y|x)}}$$



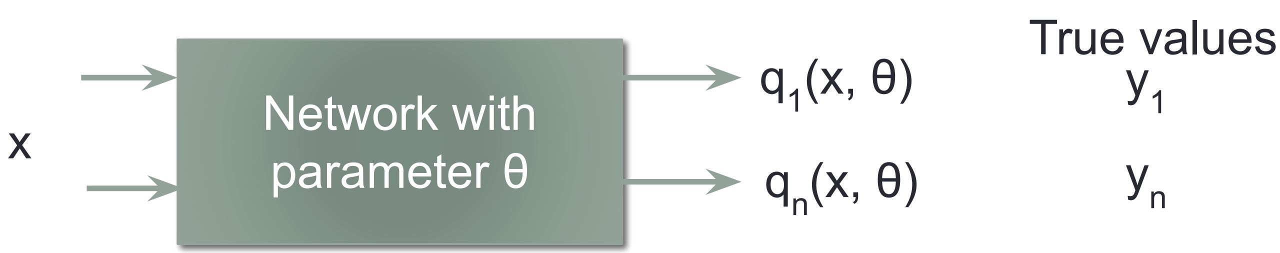
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- Demos
  - Tensorflow, Keras

Objective function (Loss function)

- Can be any function that summarizes the performance into a single number
- Cross entropy

- Sum of squared errors



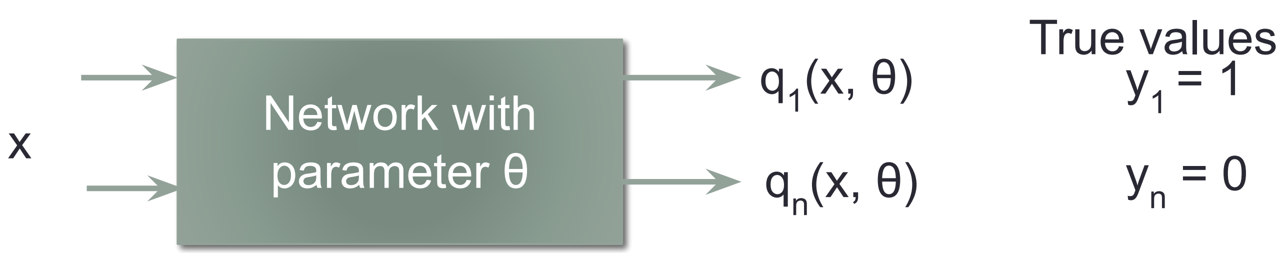
Cross entropy loss

- Used for softmax outputs (probabilities), or classification tasks

$$L = -\Sigma_n y_n log q_n(x, \theta)$$

- Where y_n is 1 if data x comes from class n
  0 otherwise

- L only has the term from the correct class
- L is non negative with highest value when the output matches the true values, a “loss” function

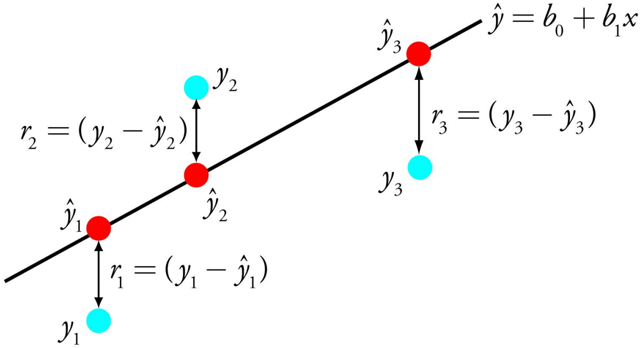


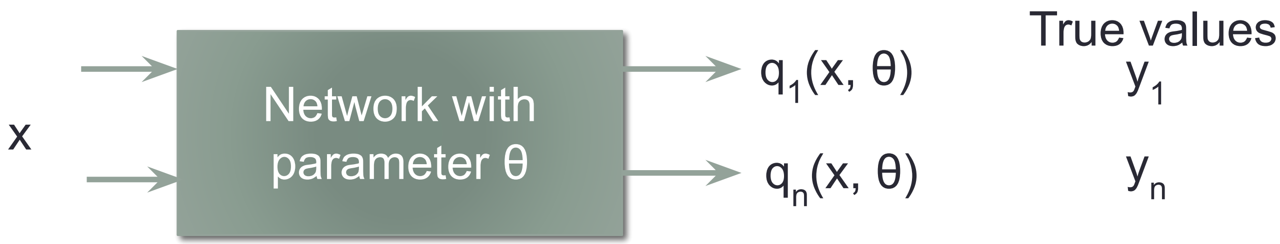
Sum of squared errors (MSE)

- Used for any real valued outputs such as regression

$$L = \frac{1}{2}\Sigma_n(y_n - q_n(x,\theta))^2$$

- Non negative
- The better the lower the loss

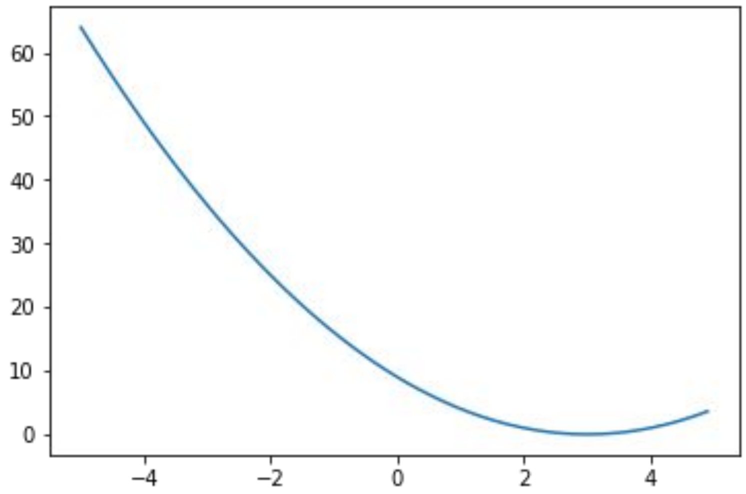




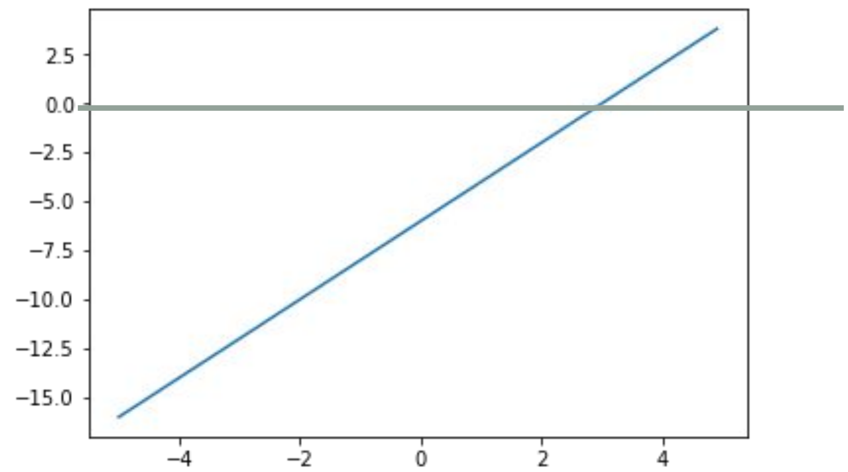
Minimizing a function

- You have a function
  - $y = (x - a)^2$
- You want to minimize Y with respect to x
  - $dy/dx = 2x - 2a$
  - Take the derivative and set the derivative to 0
    - (And maybe check if it's a minima, maxima or saddle point)
- We can also go with an iterative approach

Gradient descent



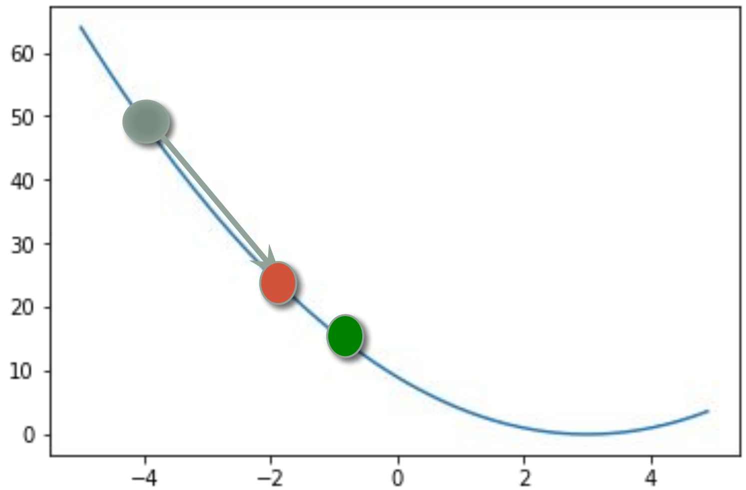
y



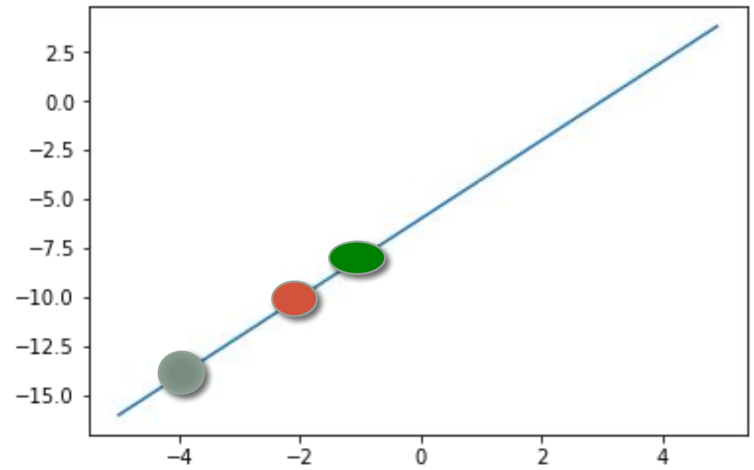
dy/dx

First what does dy/dx means?

Gradient descent



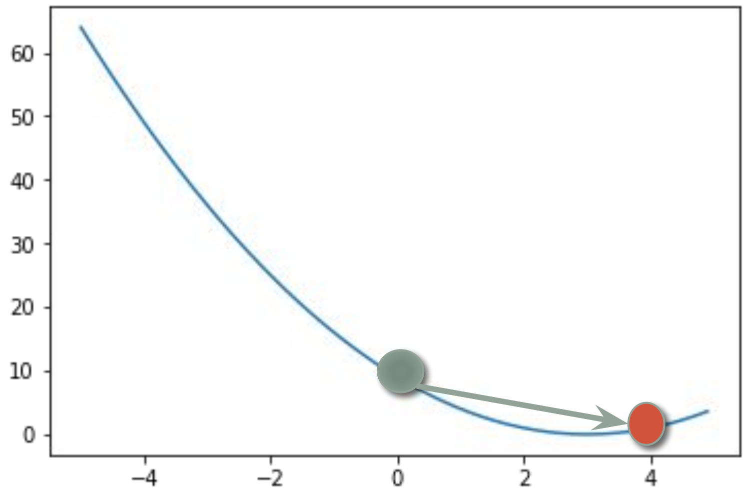
y



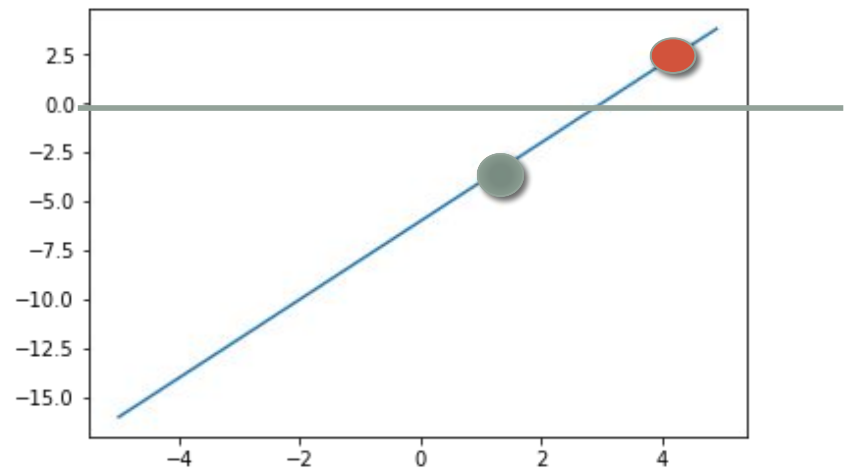
dy/dx

Move along the negative direction of the gradient
The bigger the gradient the bigger step you move

Gradient descent



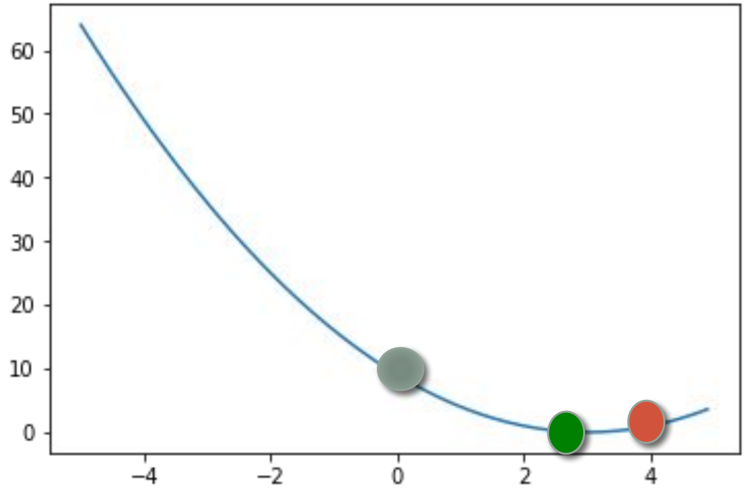
y



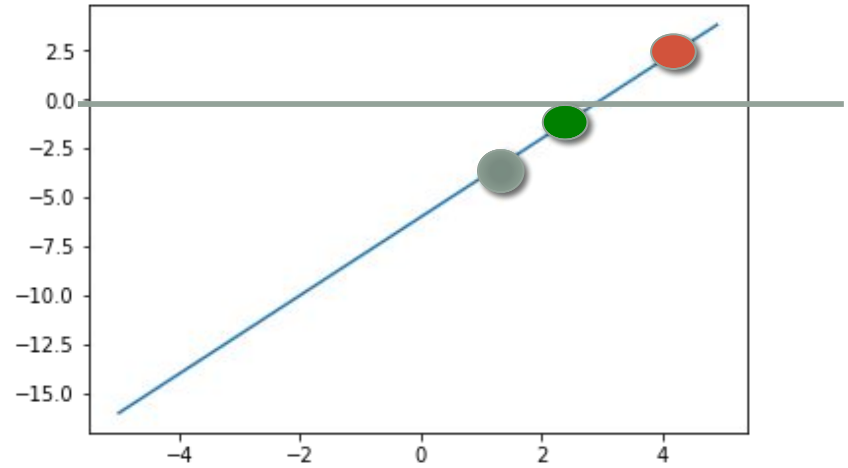
dy/dx

What happens when you overstep?

Gradient descent



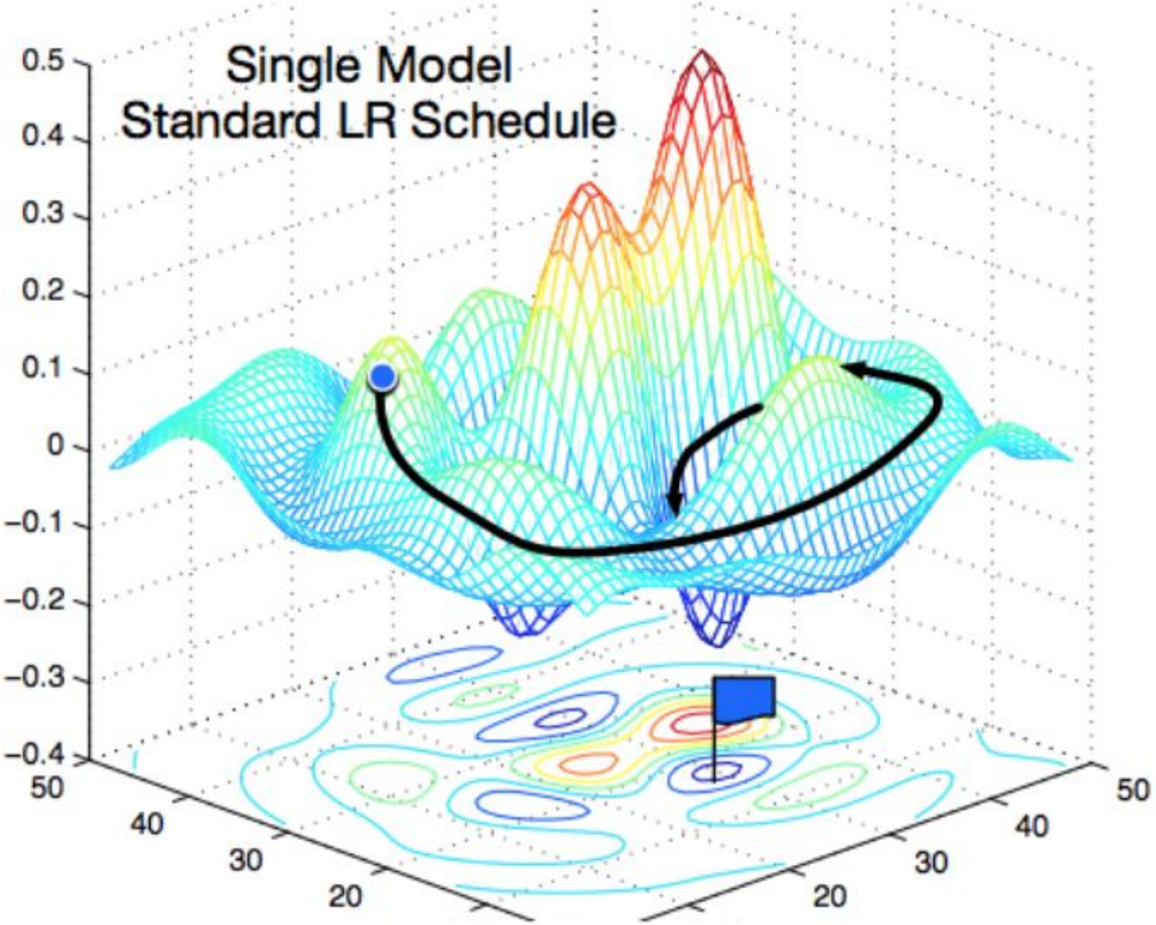
y



dy/dx

If you over step you can move back

Gradient descent in 3d



https://openreview.net/pdf?id=BJYwwY9ll

Formal definition

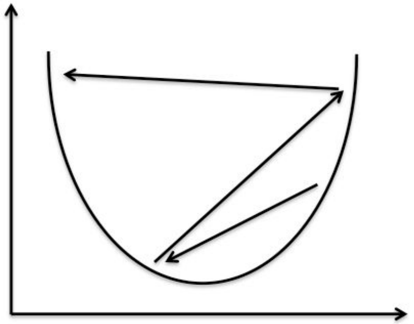
- $y = f(x)$

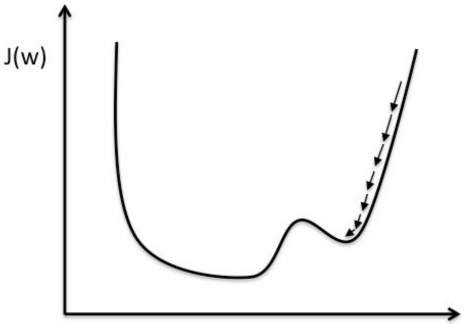
- Pick a starting point x_0

- Moves along $-dy/dx$
- $x_{n+1} = x_n - r * dy/dx$
- Repeat till convergence
- r is the learning rate

    Big r means you might overstep

    Small r and you need to take more steps





Differentiating a neural network model

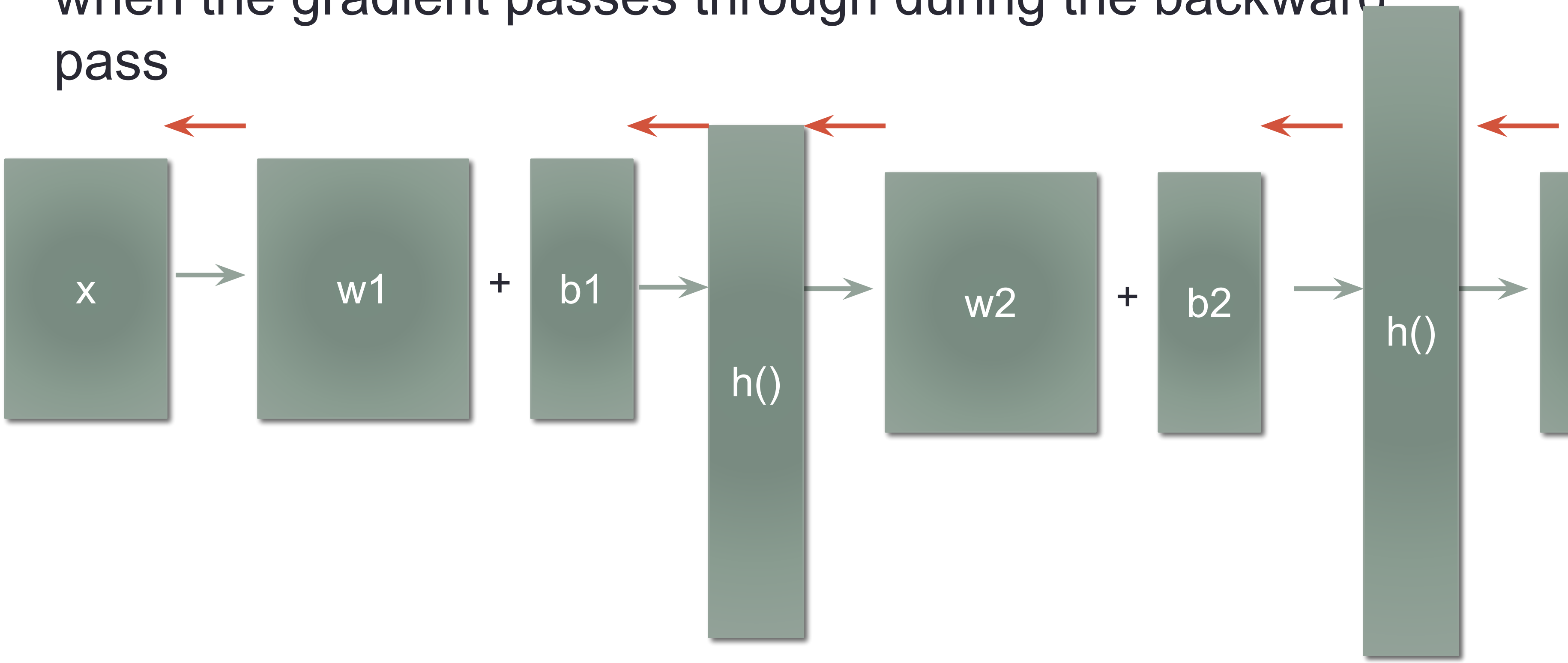
- We want to minimize loss by gradient descent
- A model is very complex and have many layers! How do we differentiate this!!?

Back propagation

- Forward pass
  - Pass the value of the input until the end of the network
- Backward pass
  - Compute the gradient starting from the end and passing down gradients using chain rule

Backprop and computation graph

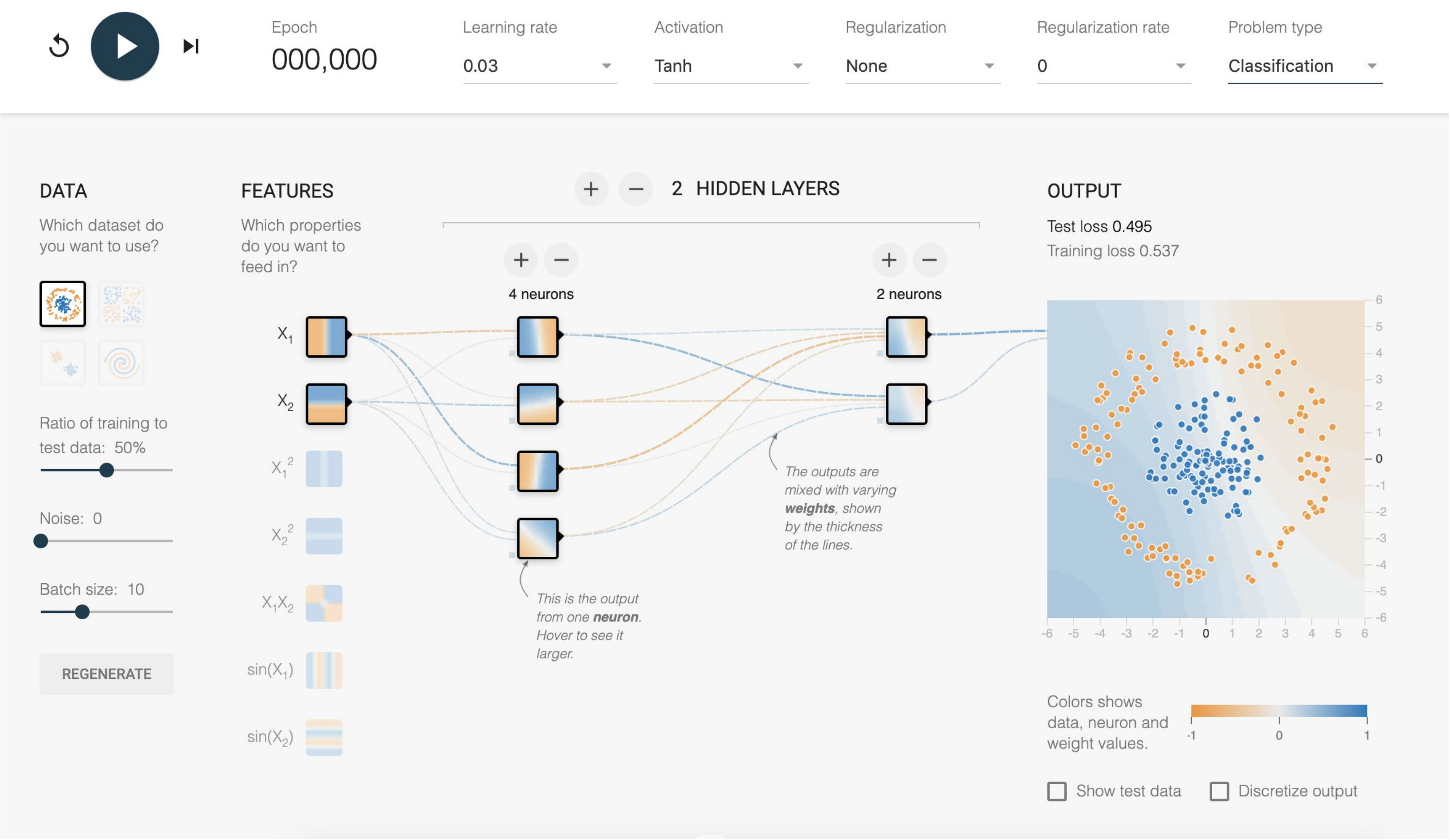
- We can also define what happens to a computing graph when the gradient passes through during the backward pass



This lets us to build any neural networks without having to redo all the derivation as long as we define a forward and backward computation for the block.

Tensorflow playground

https://playground.tensorflow.org/



Regularization

There two main approaches to regularize neural networks

- Explicit regularization
  - Deals with the loss function
- Implicit regularization
  - Deals with the network

Regularization in one slide

- What?
  - Regularization is a method to lower the model variance (and thereby increasing the model bias)
- Why?
  - Gives more generalizability (lower variance)
  - Better for lower amounts of data (reduce overfitting)
- How?
  - Introducing regularizing terms in the original loss function
    - Can be anything that make sense

Famous types of regularization

- L1 regularization: Regularizing term is a sum
  - **Original loss** + $C\Sigma|w_i|$
- L2 regularization: Regularizing term is a sum of squares
  - **Original loss** + $C\Sigma w_i^2$

http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/

Numerical example

Training data x = [3, 2, 1] y = 10, regression task

Objective: $10 = w_1*3 + w_2*2 + w_3*1$ Find w_1, w_2, w_3

w_1	w_2	w_3	L1 reg loss	L2 reg loss
3	0.25	0.5	3.75	9.31
5	-2	-1	8	30
3.33	0	0	3.33	11.11
2.14	1.42	0.71	4.29	7.14

L1 does feature selection (makes most numbers 0) - can be used to do feature selection
L2 spreads the numbers (no 0) - gives generalization

L1 L2 regularization notes

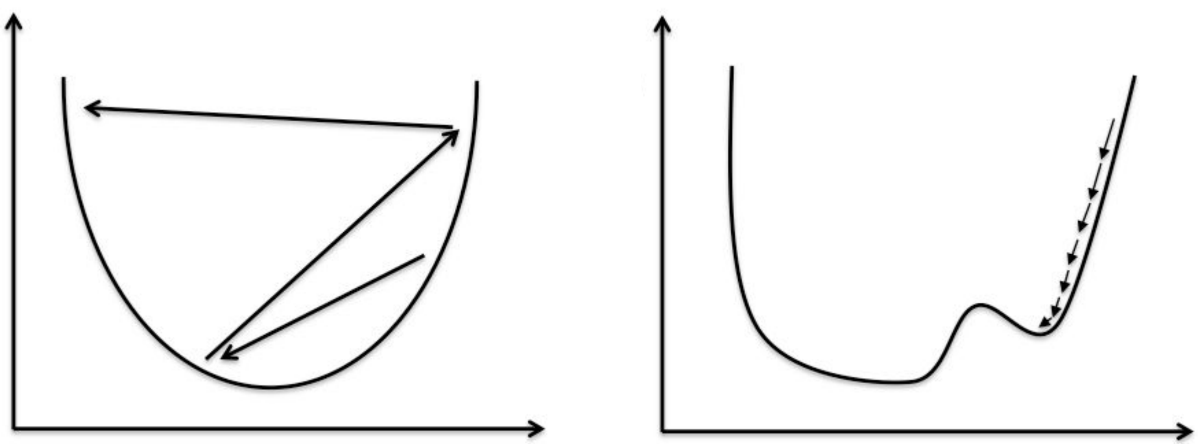
- Can use both at the same time
  - People claim L2 is superior (called weight decay by some community)
- Mostly ignored nowadays
- Other regularization methods exist (we will go over these later)

Stochastic gradient descent (SGD)

- Consider you have one million training examples
  - Gradient descent computes the objective function of all samples, then decide direction of descent
    - Takes too long
  - SGD computes the objective function on subsets of samples
    - The subset should not be biased and properly randomized to ensure no correlation between samples
- The subset is called a mini-batch
- Size of the mini-batch determines the training speed and accuracy
  - Usually somewhere between 32-1024 samples per mini-batch
- Definition: 1 batch vs 1 epoch

Learning rate

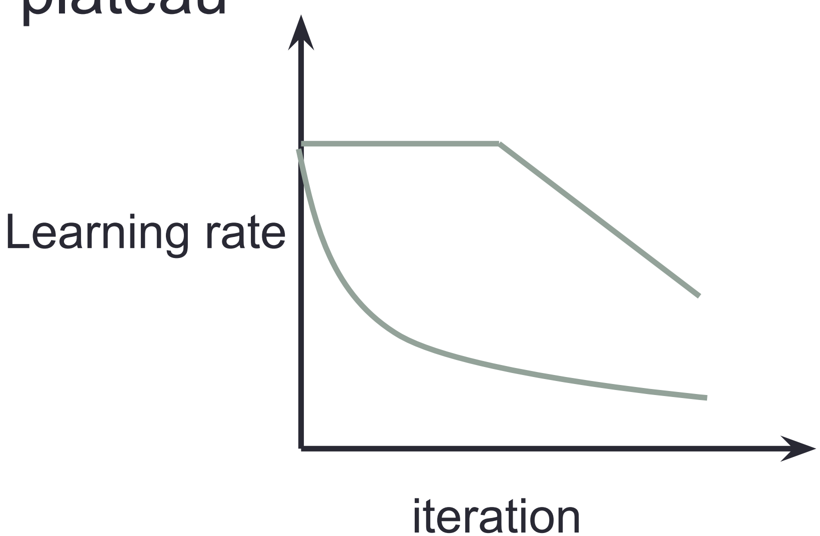
- How fast to go along the gradient direction is controlled by the learning rate
- Too large models diverge
- Too small the model get stuck in local minimas and takes too long to train



http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

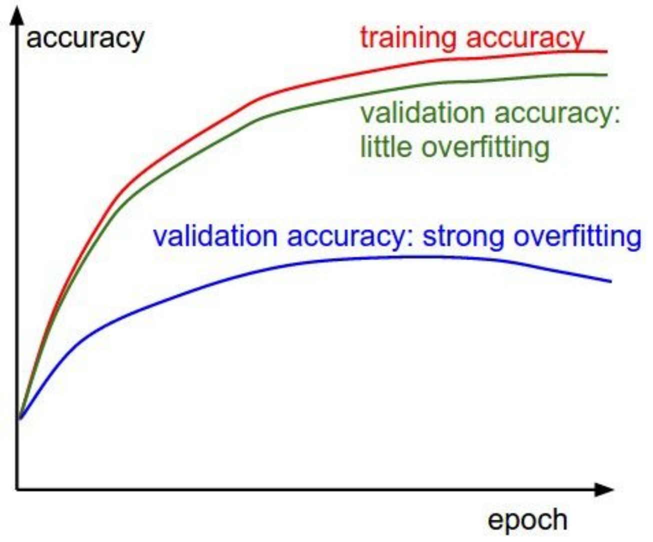
Learning rate scheduling (annealing)

- Usually starts with a large learning rate then gets smaller later
- Depends on your task
- Automatic ways to adjust the learning rate : Adagrad, Adam, etc. (still need scheduling still)
- For beginners, use Adam and learning rate decay on plateau



Overfitting

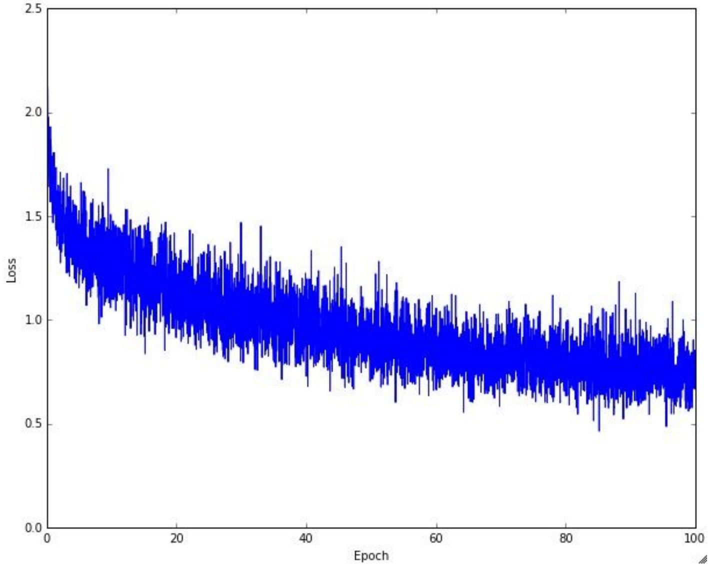
- You can keep doing back propagation forever!
- The training loss will always go down
- But it overfits
- Need to monitor performance on a held out set
- Stop or decrease learning rate when overfit happens



http://cs231n.github.io/neural-networks-3/

Monitoring performance

- Monitor performance on a dev/validation set
  - This is NOT the test set
- Can monitor many criterions
  - Loss function
  - Classification accuracy
- Sometimes these disagree
- Actual performance can be noisy, need to see the trend



http://cs231n.github.io/neural-networks-3/

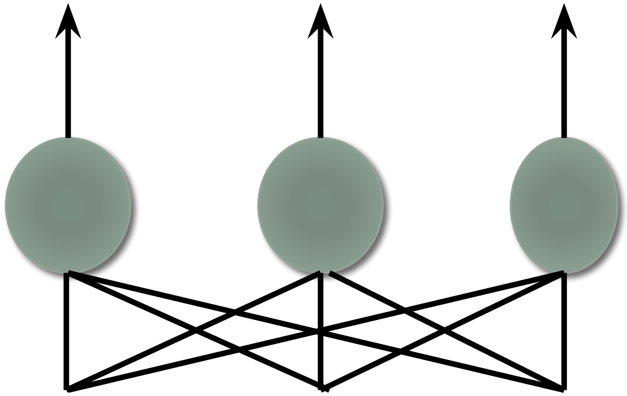
Dropout

A implicit regularization technique for reducing overfitting

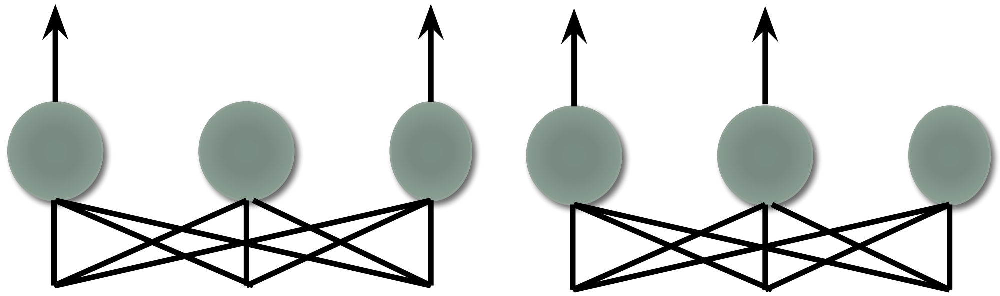
Randomly turn off different subset of neurons during training

- Network no longer depend on any particular neuron
- Force the model to have redundancy – robust to any corruption in input data

A form of performing model averaging (ensemble of experts)



Model

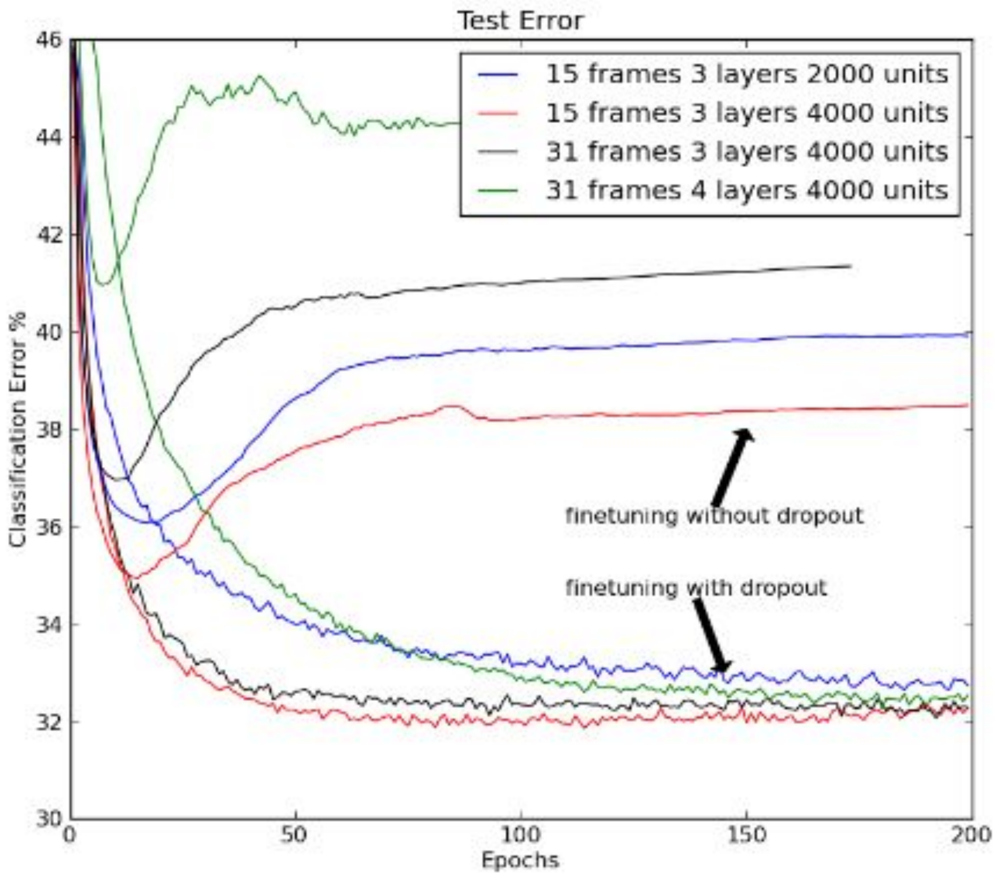


Dropout rate of 0.33

Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

Dropout on TIMIT

- A phoneme recognition task



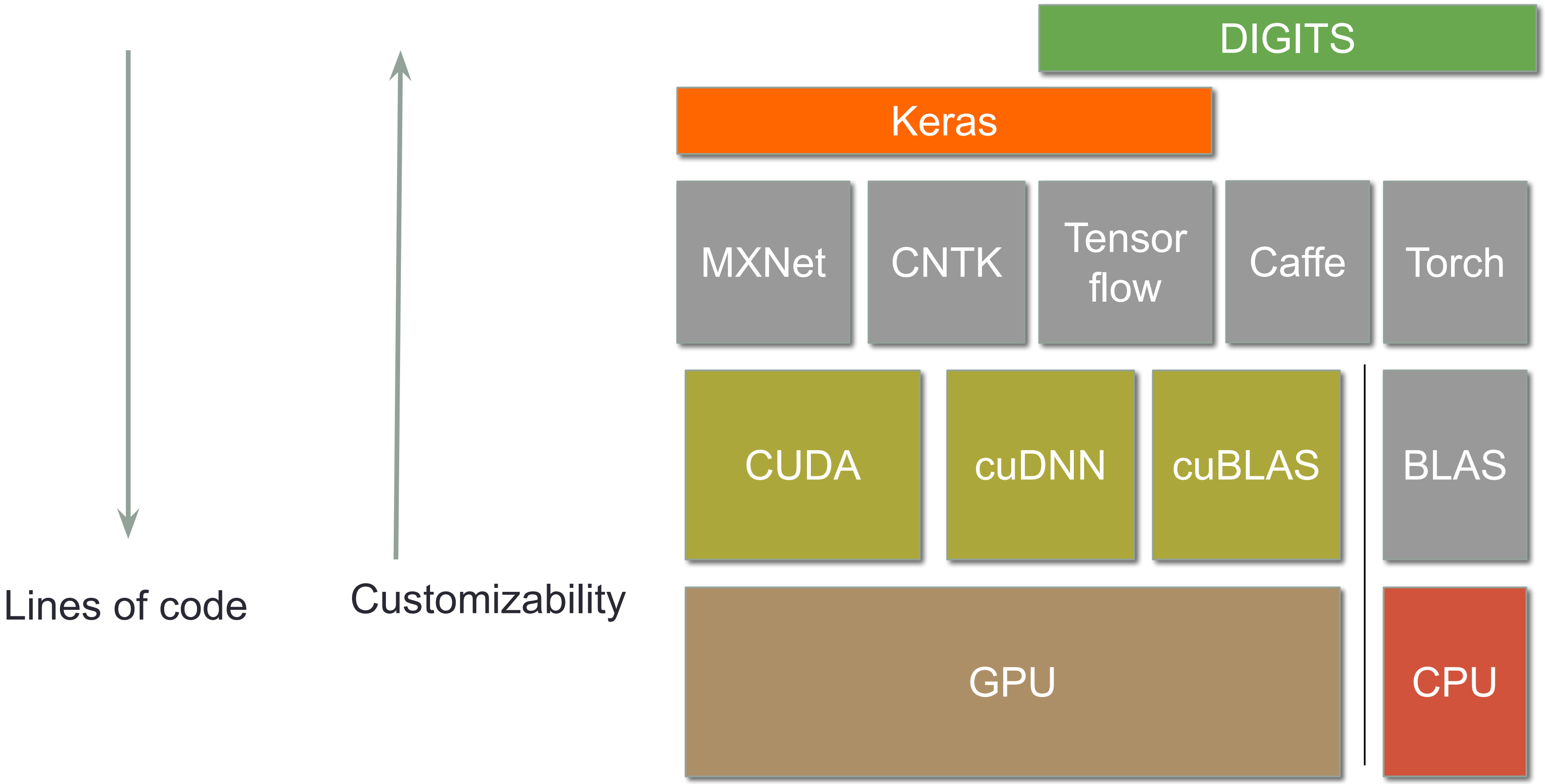
Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

Neural networks

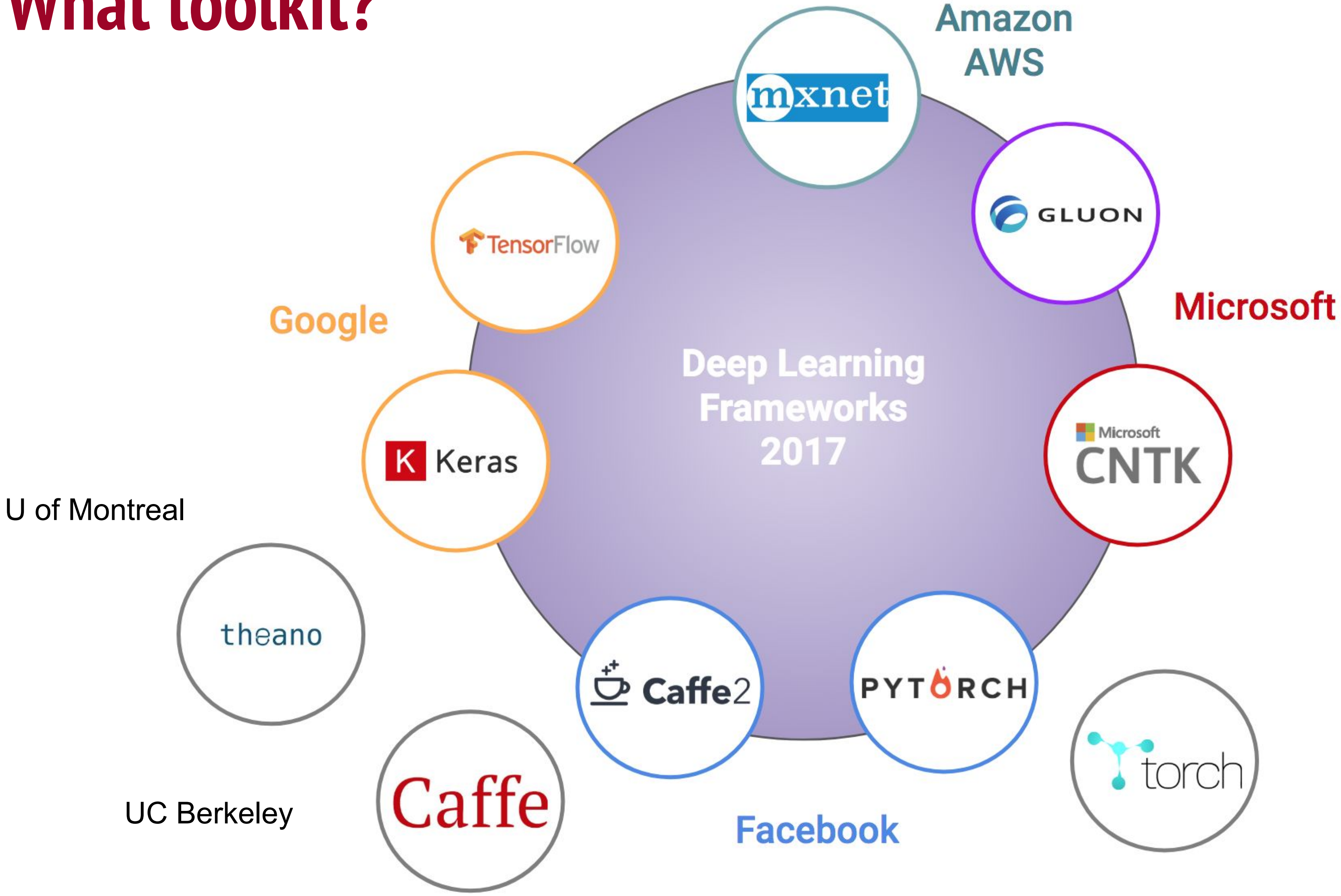
- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- Demos
  - Tensorflow, Keras

What toolkit

Tradeoff between customizability and ease of use



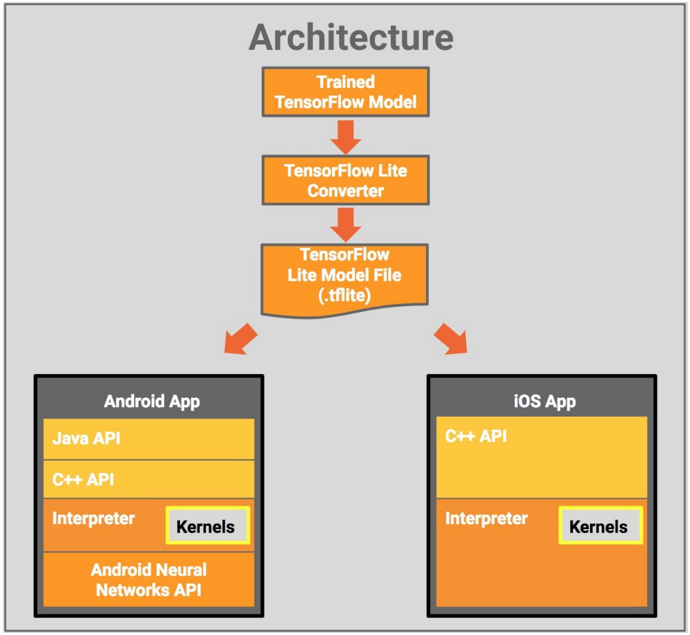
What toolkit?



https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

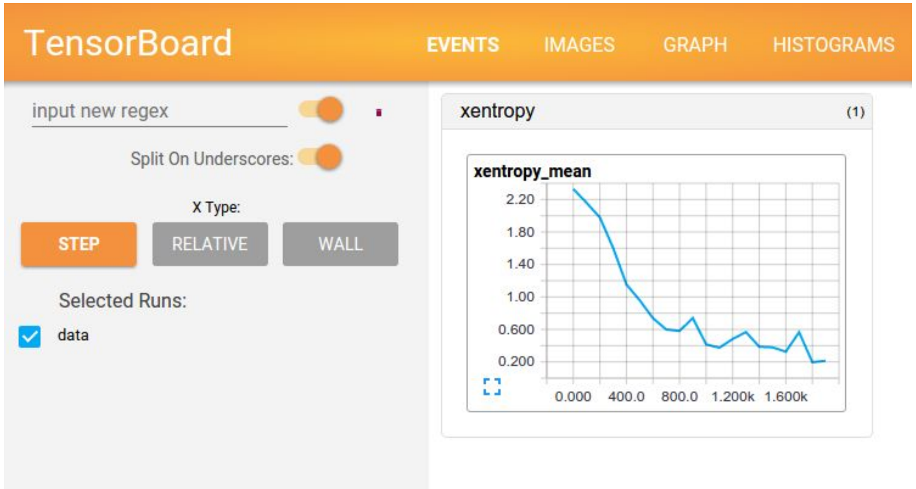
Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
- Best tools: TensorFlow
  - Tensorboard



Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
- Best tools: TensorFlow
  - Tensorboard
- Community: TensorFlow



Keras steps

- Define the network
- Compile the network
- Fit the network

```
def get_feedforward_nn():
  input1 = Input(shape=(21,))
  x = Dense(100, activation='relu')(input1)
  x = Dense(100, activation='relu')(x)
  x = Dense(100, activation='relu')(x)
  out = Dense(1, activation='sigmoid')(x)

  model = Model(inputs=input1, outputs=out)
  model.compile(optimizer=Adam(),
                loss='binary_crossentropy',
                metrics=['acc'])
  return model
```

```
  model_feedforward_nn.fit(x_train_char, y_train, epochs=epochs, batch_size=batch_size, verbose=ve
rbose,
                           callbacks=callbacks_list_feedforward_nn,
                           validation_data=(x_val_char, y_val))
```

Keras is easy!

Dense

[source]

```
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializ
```

Just your regular densely-connected NN layer.

`Dense` implements the operation: `output = activation(dot(input, kernel) + bias)` where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only applicable if `use_bias` is `True`).

- **Note**: if the input to the layer has a rank greater than 2, then it is flattened prior to the initial dot product with `kernel`.

Example

```
# as first layer in a sequential model:
model = Sequential()
model.add(Dense(32, input_shape=(16,)))
# now the model will take as input arrays of shape (*, 16)
# and output arrays of shape (*, 32)

# after the first layer, you don't need to specify
# the size of the input anymore:
model.add(Dense(32))
```

Dropout

[source]

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

Applies Dropout to the input.

Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape `(batch_size, timesteps, features)` and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch_size, 1, features)`.
- **seed**: A Python integer to use as random seed.

Lab

https://colab.research.google.com/drive/1X0aDtVesSqbrlbK9HBoJb20yKRV6qbej

Either **File -> Save a copy in drive** or

Open in playground