NEURAL NETWORKS

GMM/Gaussian fitting

- How many parameters are there in a 2x2 covariance matrix?
- How many data points do you need to estimate a 2v2 covariance matrix (at least)?

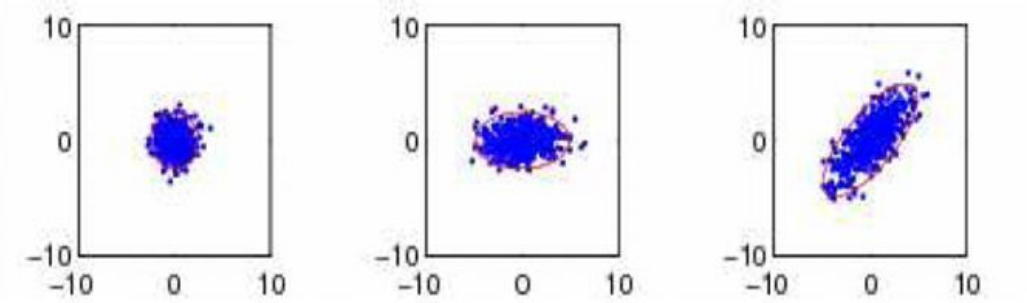
$$m_j = \frac{1}{N}\Sigma_n w_{n,j}$$

$$\vec{\mu_j} = \frac{\Sigma_n w_{n,j}\vec{x_n}}{\Sigma_n w_{n,j}}$$

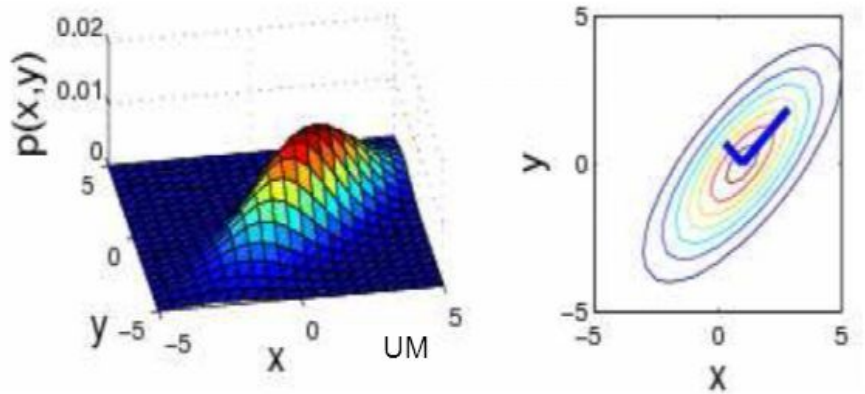
$$\mathbf{\Sigma}_j = \frac{\Sigma_n w_{n,j}(\vec{x_n} - \vec{\mu_j})(\vec{x_n} - \vec{\mu_j})^T}{\Sigma_n w_{n,j}}$$

Many forms of covariance matrix

Spherical, diagonal, full covariance



$$\Sigma = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$



http://slideplayer.com/slide/6258331/

Whitening and GMM fitting

- Spherical/diagonal covariance are less prone to overfitting (less parameters)
- Data are not always distributed like that
- Use whitening to help make them spherical/diagonal distributed
  - Still not quite true, but oh well

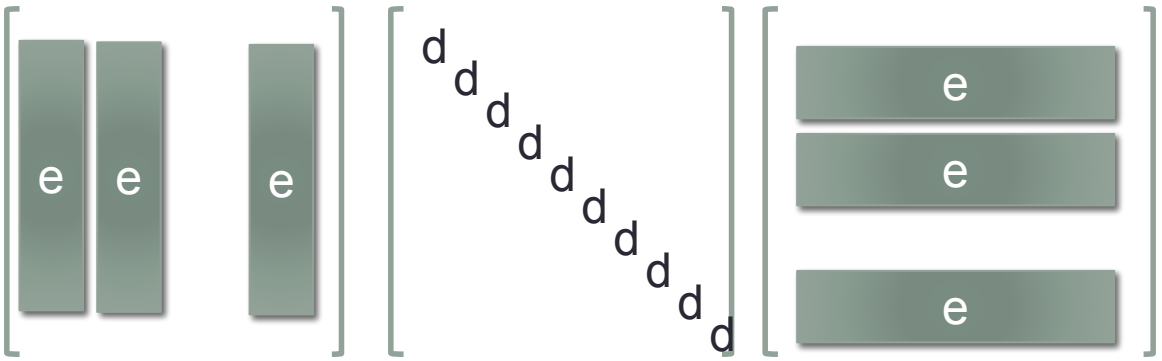
PCA as a feature normalization technique

- We said it's good to normalize features to [0,1],[-1,1], N(0,1).
  - Normalize each dimension independently
- Can we do better?

Whitening (PCA)

- Find the project along the dimensions that has the highest variance in the data
- Let $\sum$ be the covariance matrix. E is the matrix of eigenvectors, and D has eigenvalues along the diagonal. With eigen decomposition:
  - Note for covariance matrices, $E^t = E^{-1}$

$$\Sigma = EDE'$$

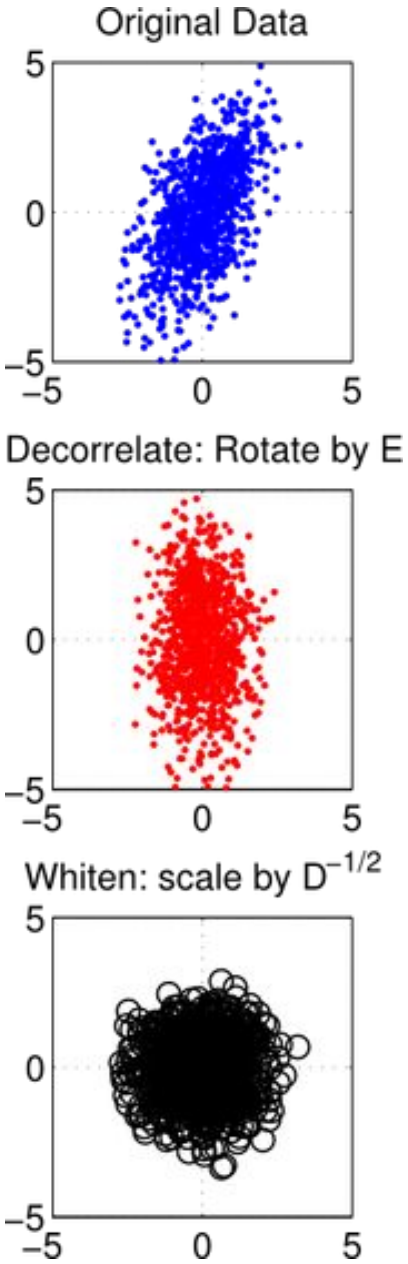


Whitening (PCA)

- Whitening decorrelates and scale

$$Y = D^{-1/2}E'X$$

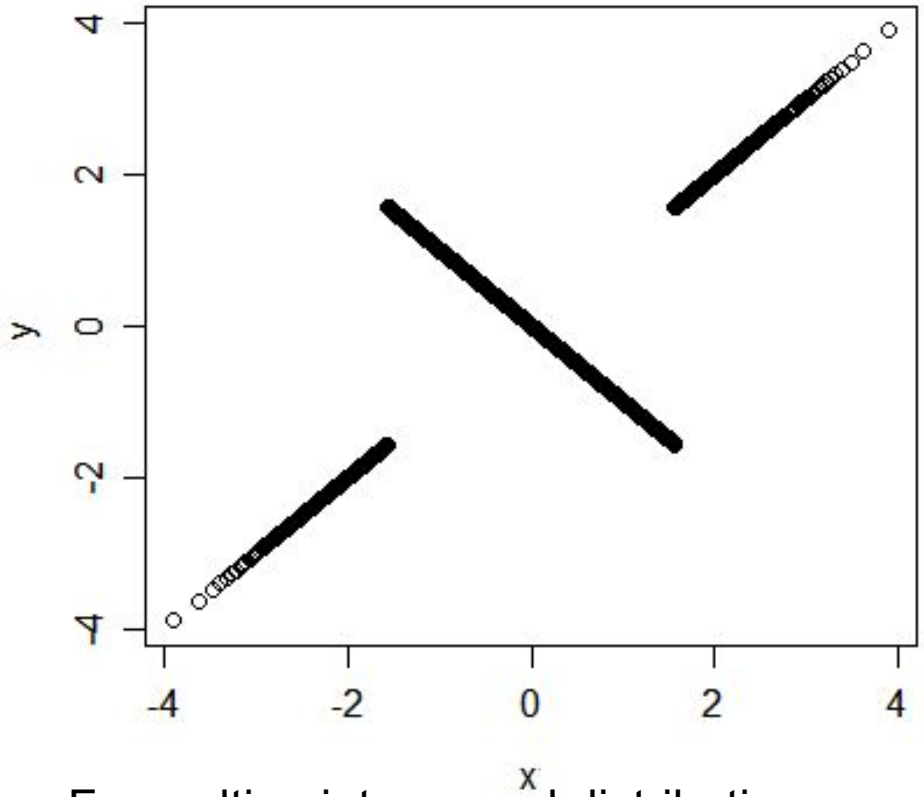
- In homework we only use the decorrelates part (rotation)
- Some models prefer features to be of equal variance (SVMs, Neural networks)
- Scale according to the inverse of the variance.
- This decorrelates the features (on the global scale)
  - Correlations can still exist given class
  - Uncorrelated-ness does not imply independence
    - We usually assume so though
- It is often a good idea to normalize the variance of each feature first before doing PCA dimensionality reduction

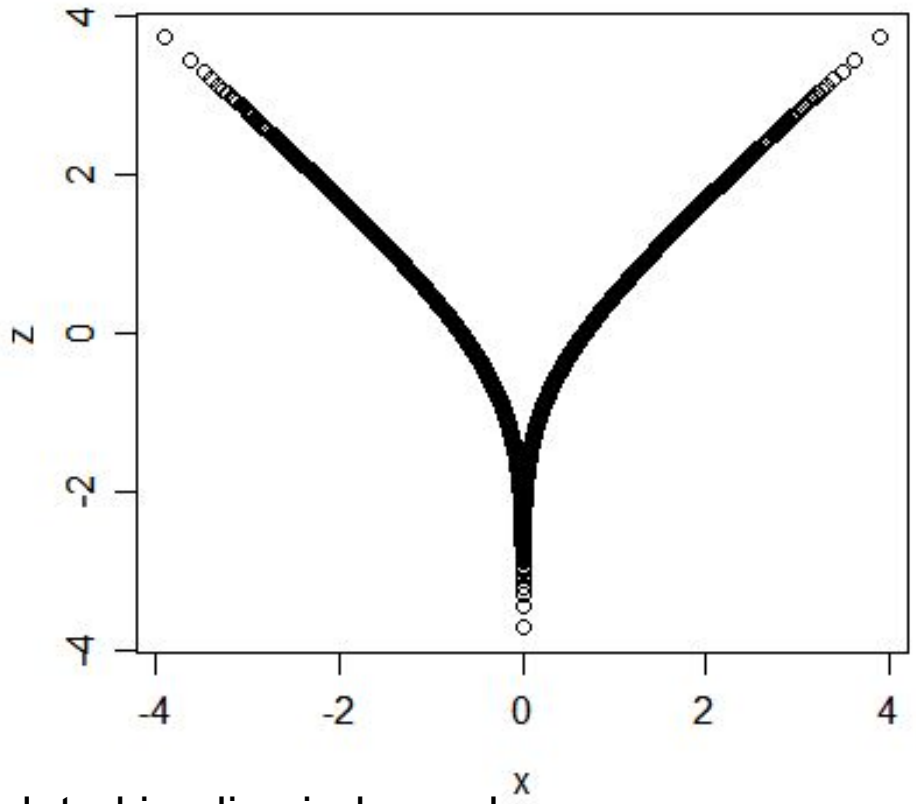


https://theclevermachine.wordpress.com/2013/03/30/the-statistical-whitening-transform/

Uncorrelated but dependence

- Below are example of variables with 0 correlation but definitely not independent





For multivariate normal distribution, uncorrelated implies independence

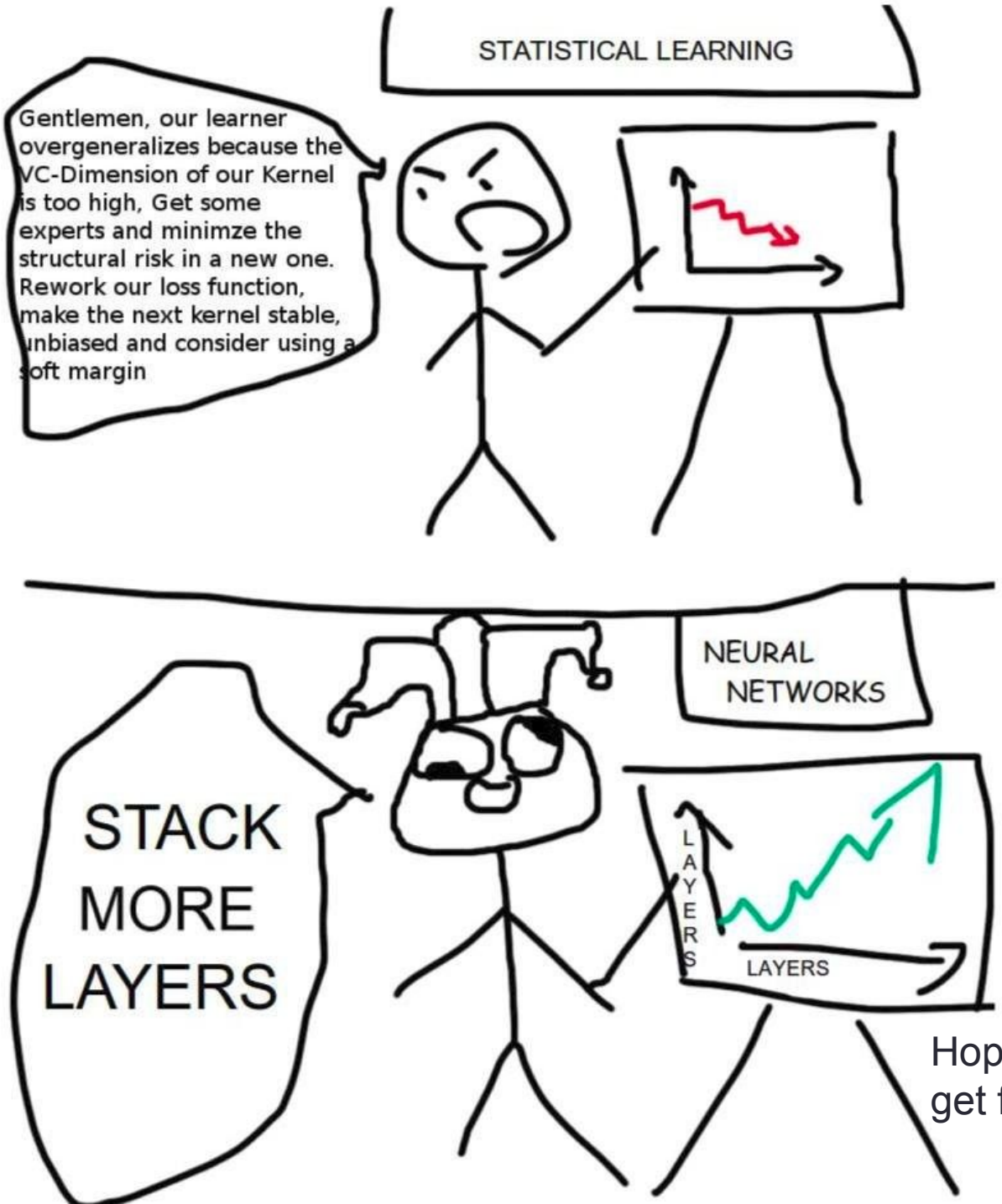
Whitening (PCA)

- What is the covariance matrix of data rotated by PCA?
- What is the covariance matrix of data whiten by PCA?

NEURAL NETWORKS

Deep learning = Deep neural networks = neural networks

https://xkcd.com/1838/



Hopefully this won't be all you get from this class

DNNs (Deep Neural Networks)

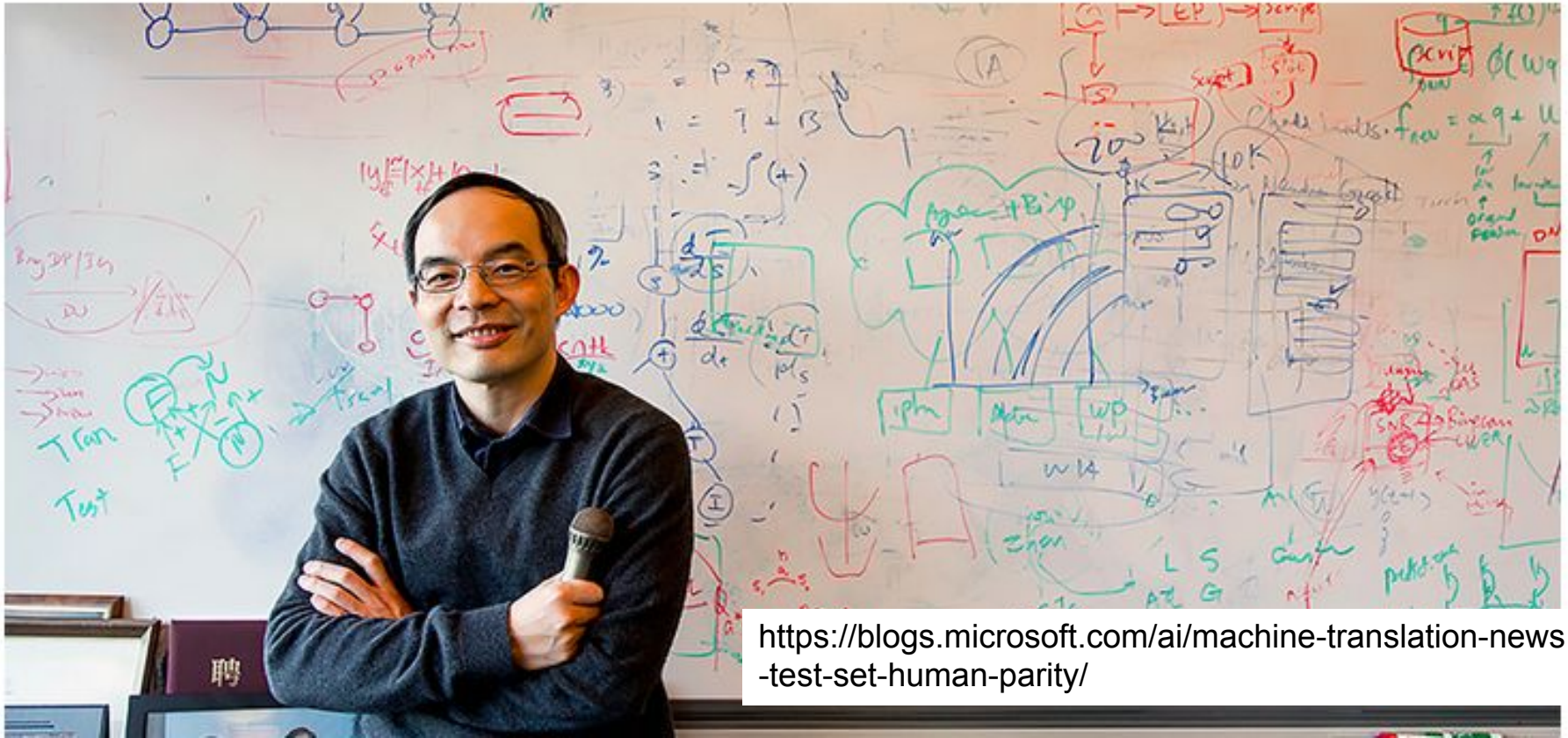
- Why deep learning?
- Greatly improved performance in ASR and other tasks (Computer Vision, Robotics, Machine Translation, NLP, etc.)
- Surpassed human performance in many tasks

Task	Previous state-of-the-art	Deep learning (2012)	Deep learning (2017)
TIMIT	24.4%	20.0%	17.0%
Switchboard	23.6%	16.1%	5.5%
Google voice search	16.0%	12.3%	4.9%

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/

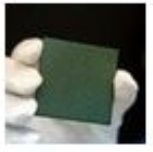
Google's AlphaGo Defeats Chinese Go Master in Win for A.I.

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

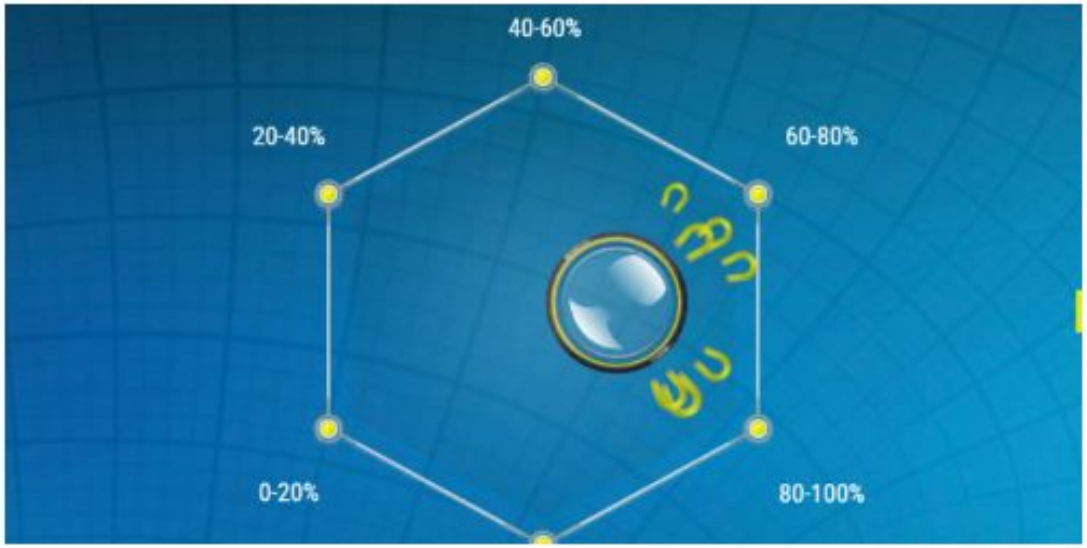

Maste
Goog

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

The Stanford Daily

Artificial swarm intelligence diagnoses pneumonia better than individual computer or doctor

by Fan Liu — September 27, 2018 0



Courtesy of Unanimous AI

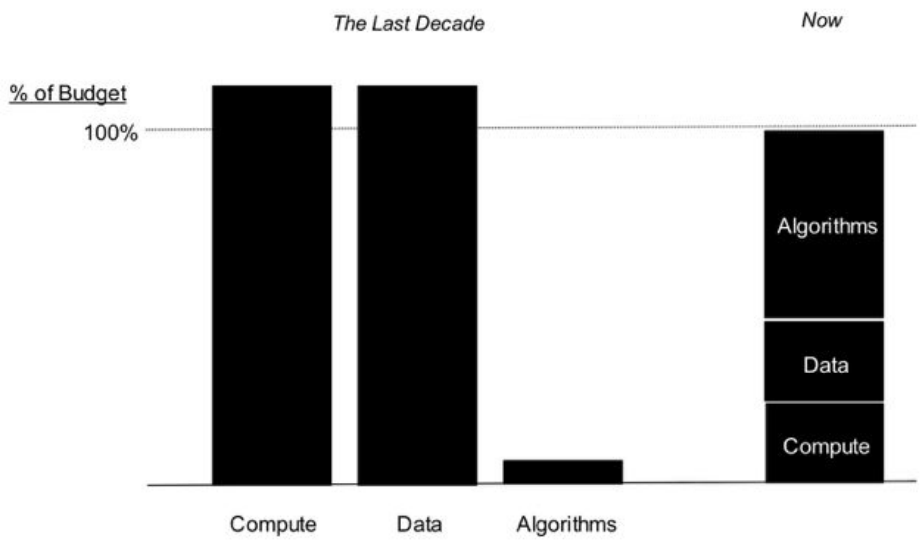
Popular Posts

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



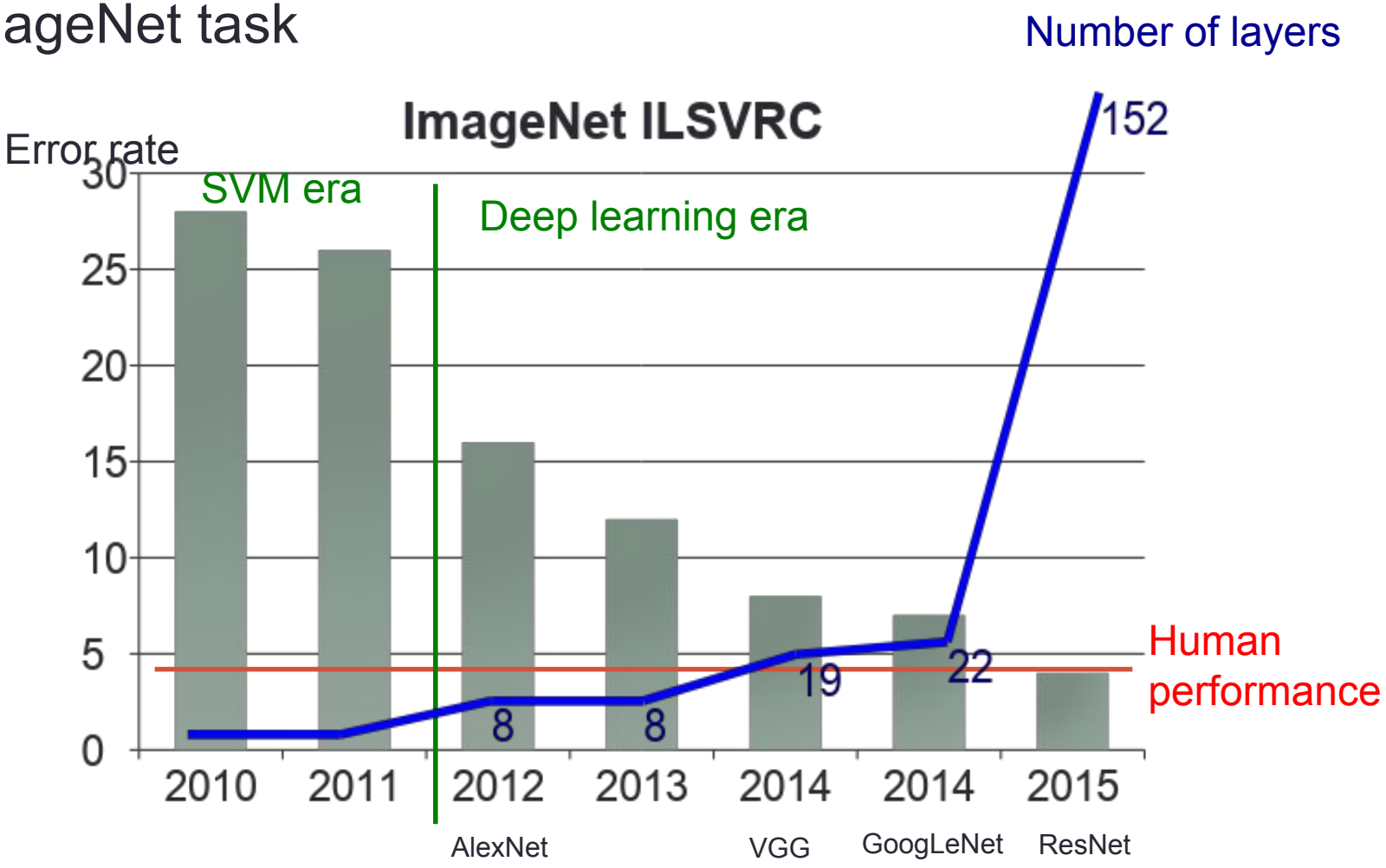
http://www.kdnuggets.com/2017/06/practical-guide-machine-learning-understand-differentiate-apply.html

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

Statistics > Machine Learning

Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, Jeffrey Pennington

(Submitted on 14 Jun 2018)

In recent years, state-of-the-art methods in computer vision have utilized increasingly deep convolutional neural network architectures (CNNs), with some of the most successful models employing hundreds or even thousands of layers. A variety of pathologies such as vanishing/exploding gradients make training such deep networks challenging. While residual connections and batch normalization do enable training at these depths, it has remained unclear whether such specialized architecture designs are truly necessary to train deep CNNs. In this work, we demonstrate that it is possible to train vanilla CNNs with ten thousand layers or more simply by using an appropriate initialization scheme. We derive this initialization scheme theoretically by developing a mean field theory for signal propagation and by characterizing the conditions for dynamical isometry, the equilibration of singular values of the input-output Jacobian matrix. These conditions require that the convolution operator be an orthogonal transformation in the sense that it is norm-preserving. We present an algorithm for generating such random initial orthogonal convolution kernels and demonstrate empirically that they enable efficient training of extremely deep architectures.

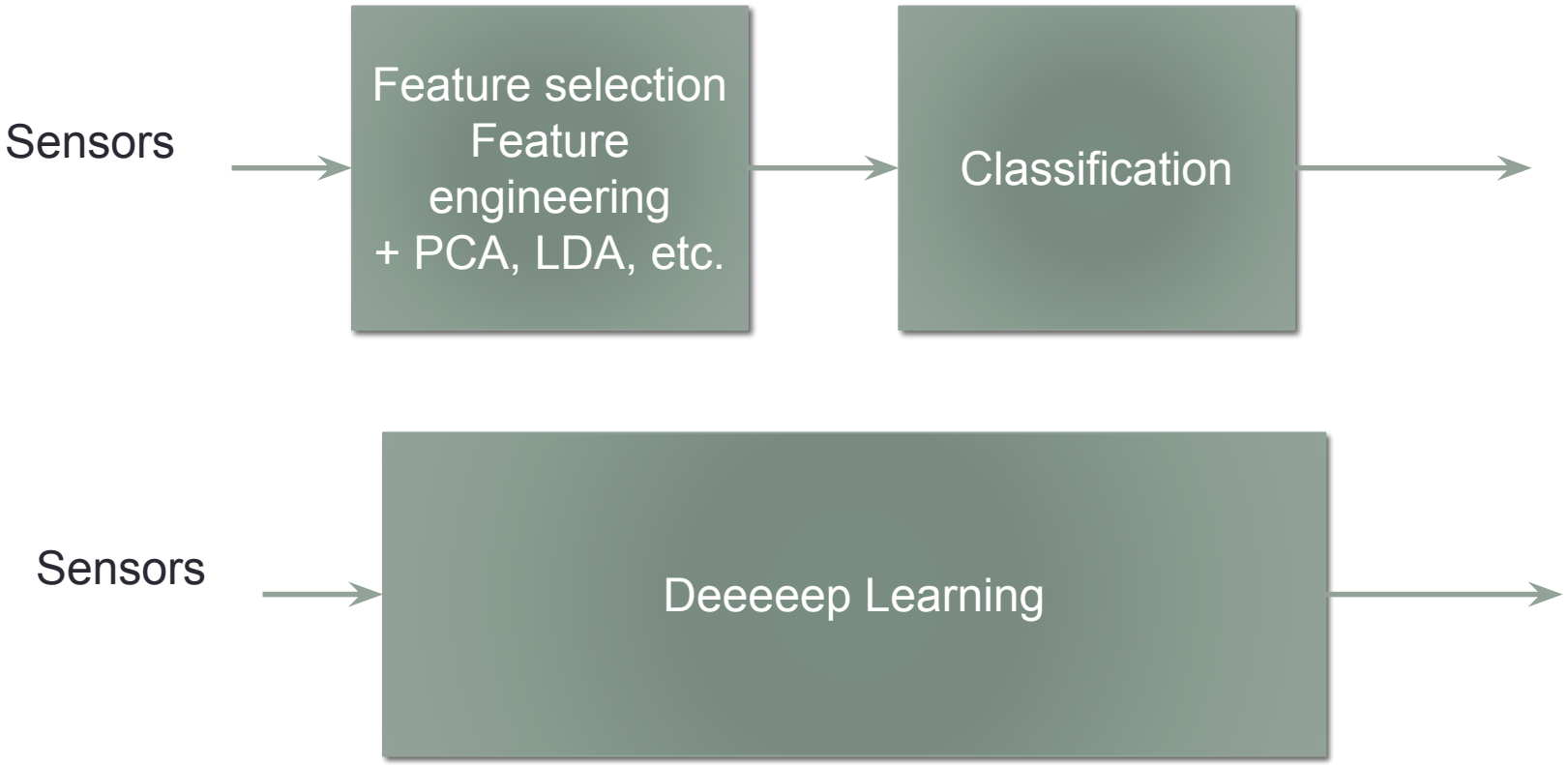
Comments: ICML 2018 Conference Proceedings
Subjects: **Machine Learning (stat.ML)**; Machine Learning (cs.LG)
Cite as: arXiv:1806.05393 **[stat.ML]**
(or **arXiv:1806.05393v1 [stat.ML]** for this version)

Submission history

From: Samuel Schoenholz [view email]
[v1] Thu, 14 Jun 2018 07:04:15 GMT (6734kb,D)

Which authors of this paper are endorsers? | Disable MathJax (What is MathJax?)

Traditional VS Deep learning

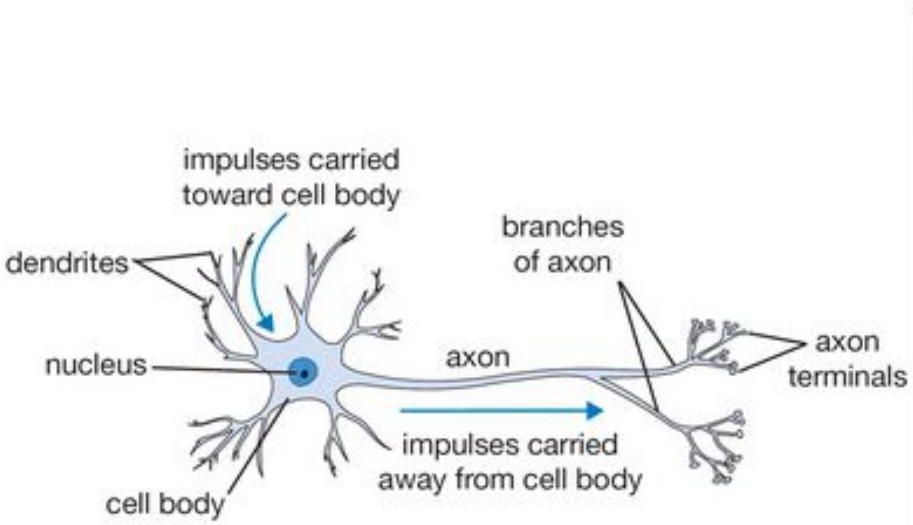


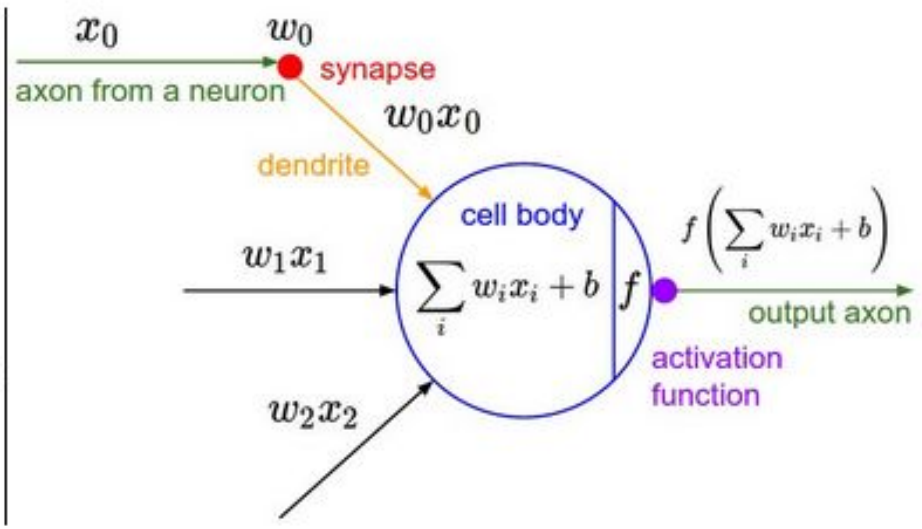
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- Demos
  - Tensorflow, Gcloud, Keras
- CNN, RNN, LSTM, GRU <- Next class

Fully connected networks

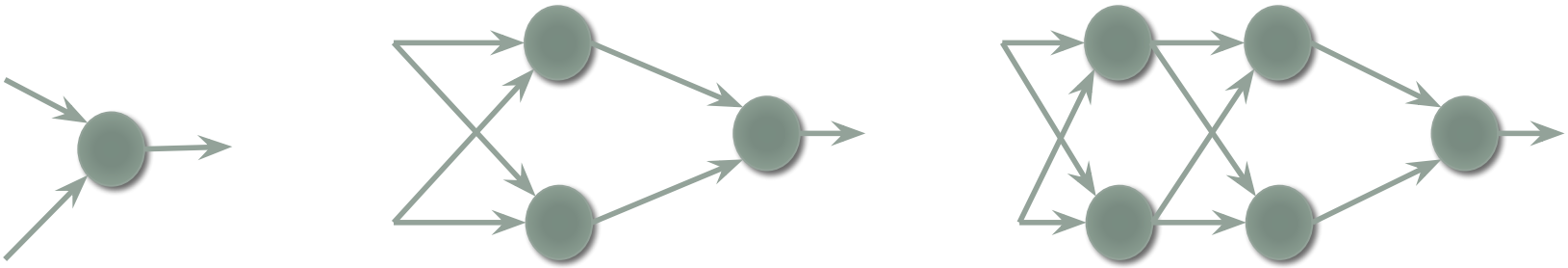
- Many names: feed forward networks or deep neural networks or multilayer perceptron or artificial neural networks
- Composed of multiple neurons

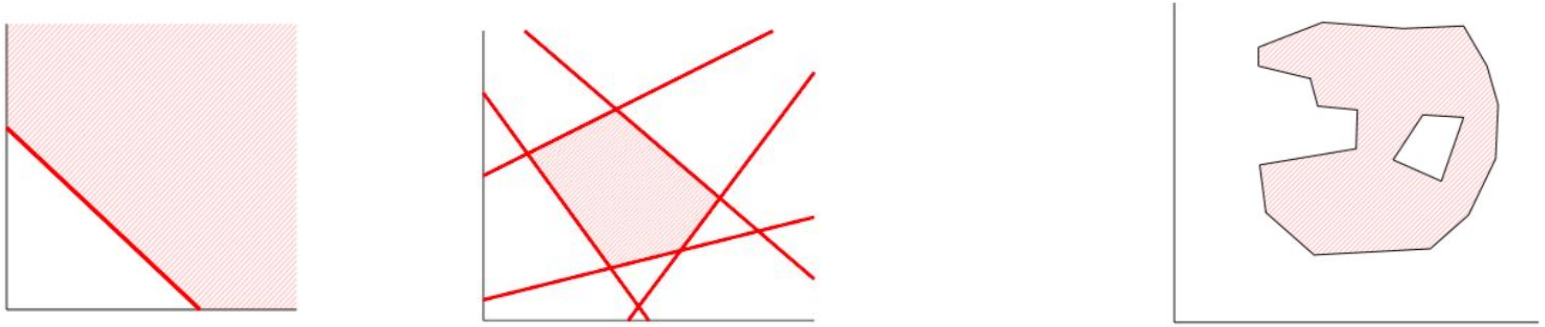




Combining neurons

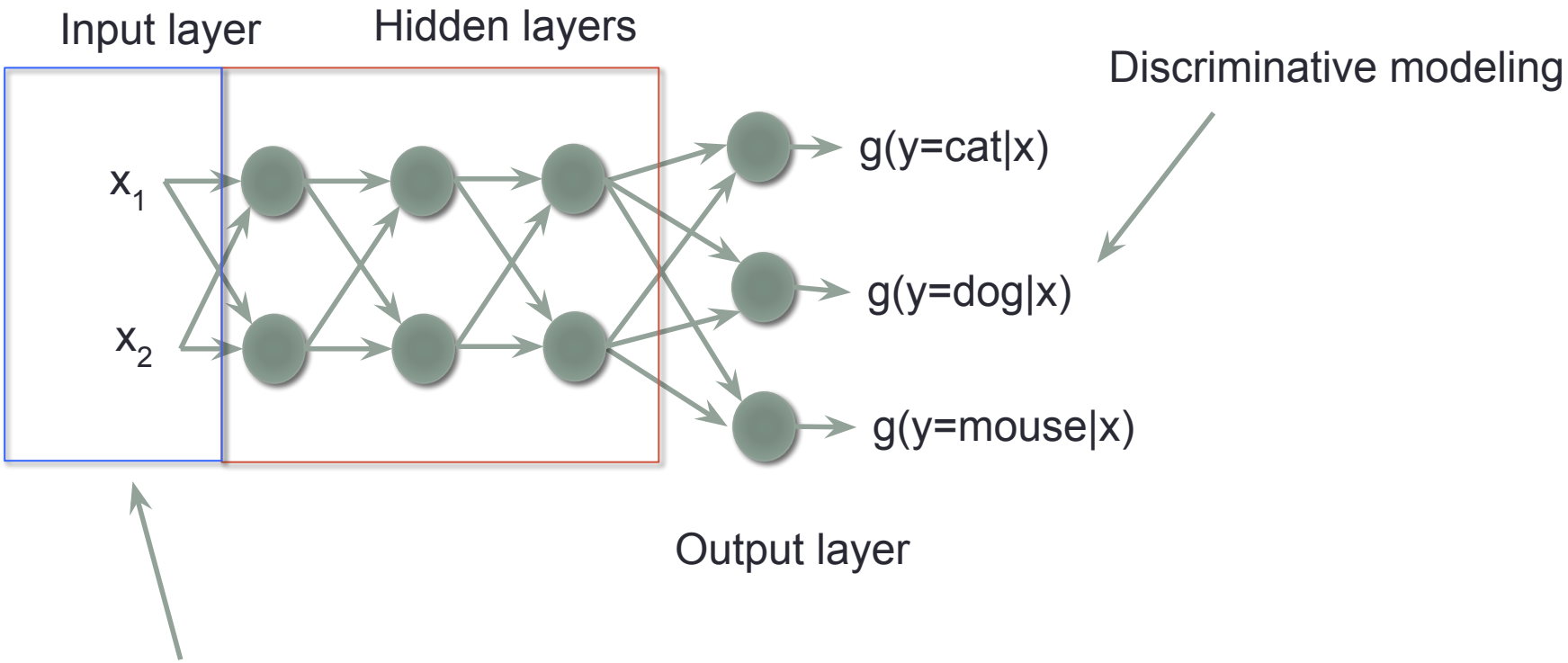
- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting





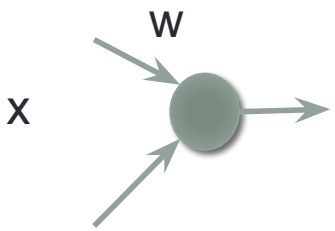
Terminology

Deep in Deep neural networks means many hidden layers



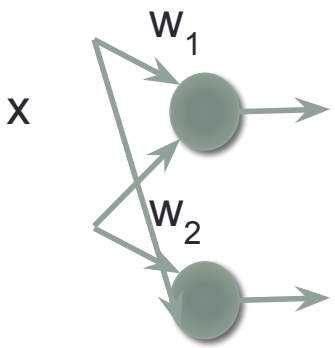
Projections and Neural network weights

- $w^T x$



Projections and neural network weights

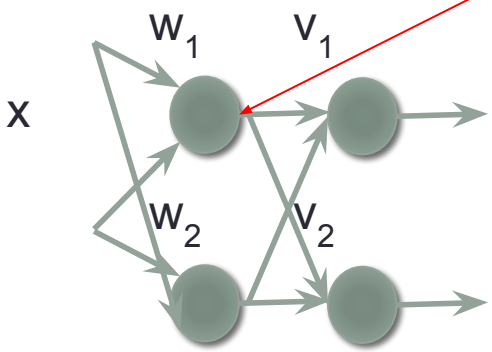
- $W^T x$



Neural network layer acts as nonlinear feature transform

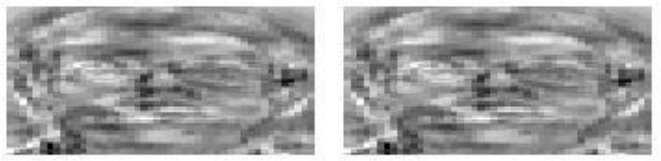
- $W^T x$

Without the nonlinearity the two matrices combine into one operation

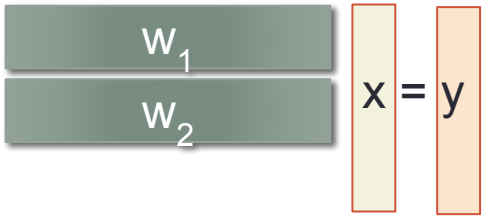


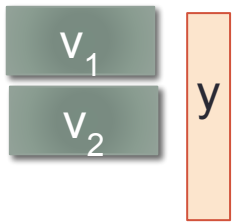
fischer projection = $V^T W^T x$
$= (WV)^T x$

Wv_1 Wv_2



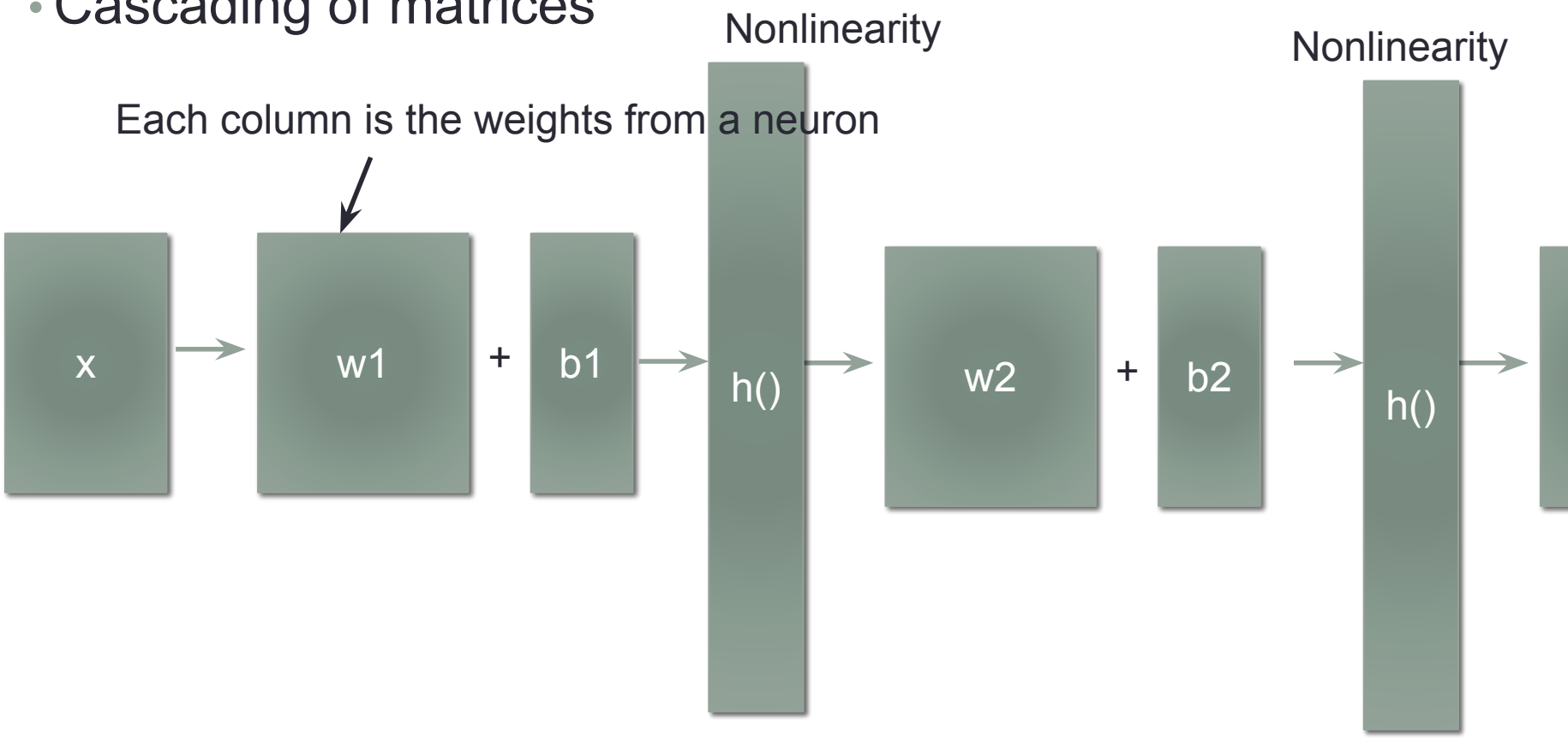
LDA projections





More linear algebra

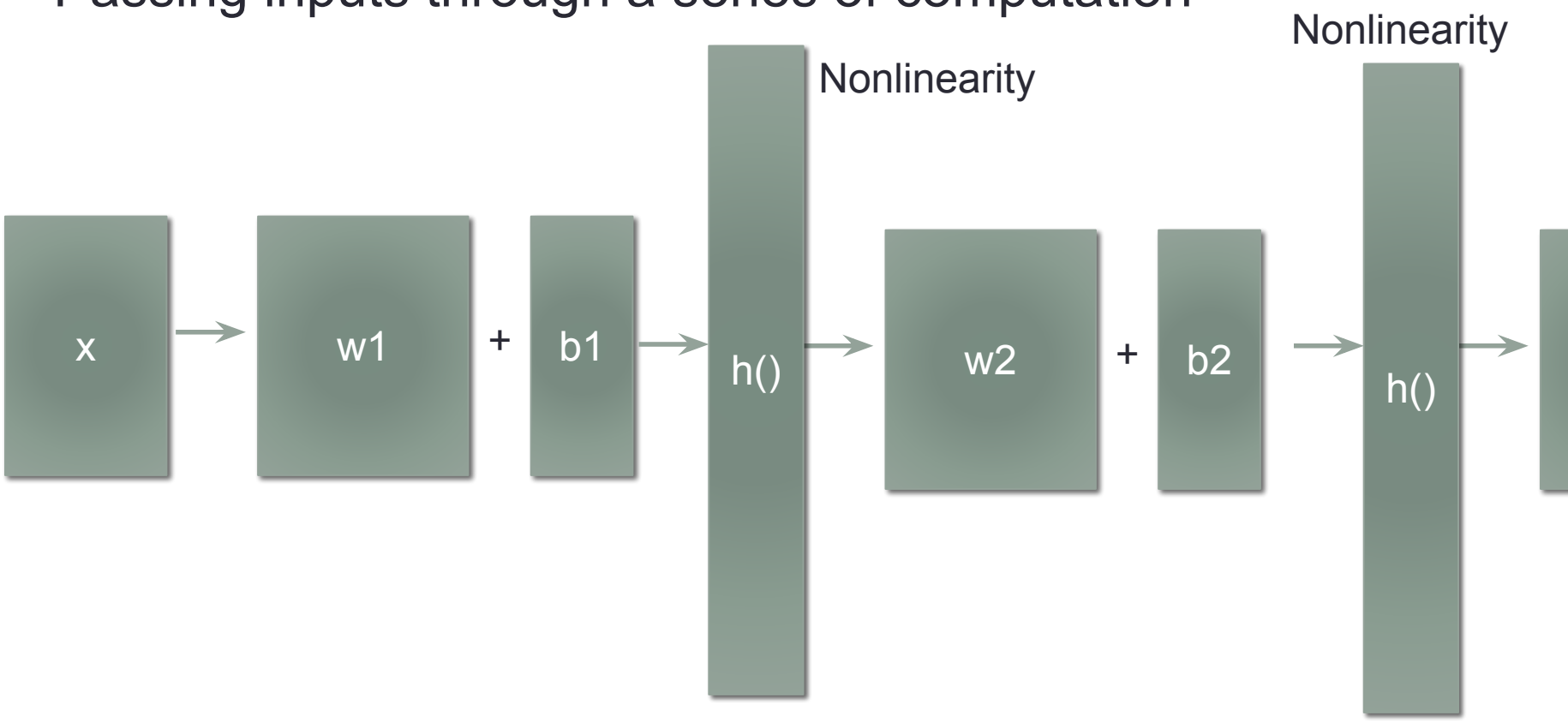
- Cascading of matrices



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

Computation graph

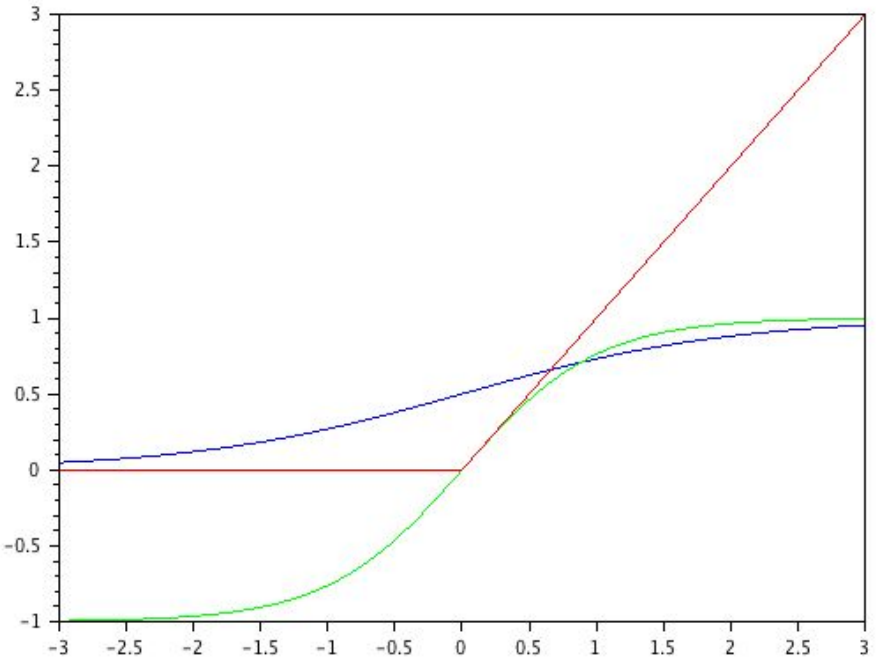
- Passing inputs through a series of computation



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

Non-linearity

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



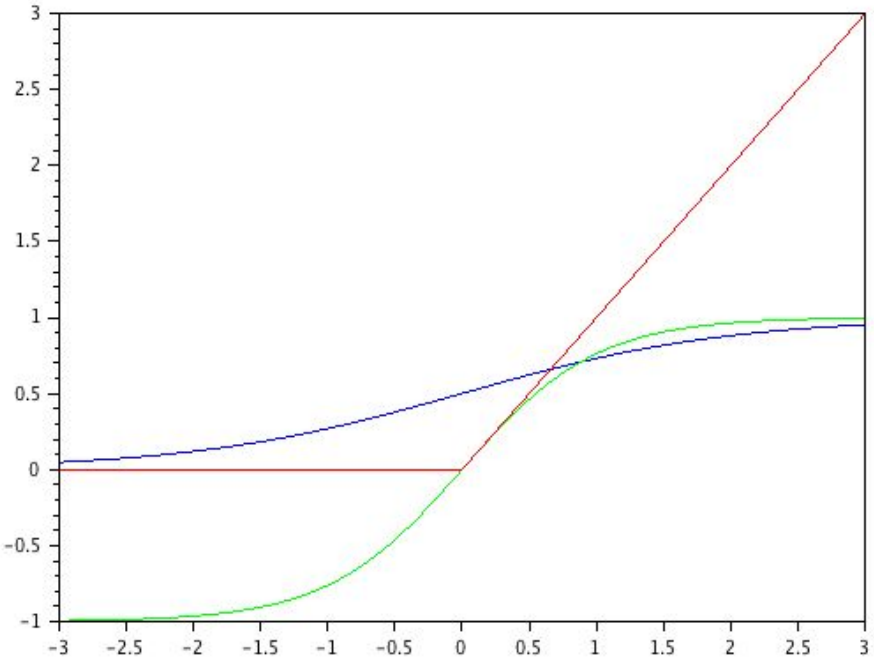
Non-linearity

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

$$tanh(x)$$

- Rectified Linear Unit (ReLU)

$$max(0, x)$$



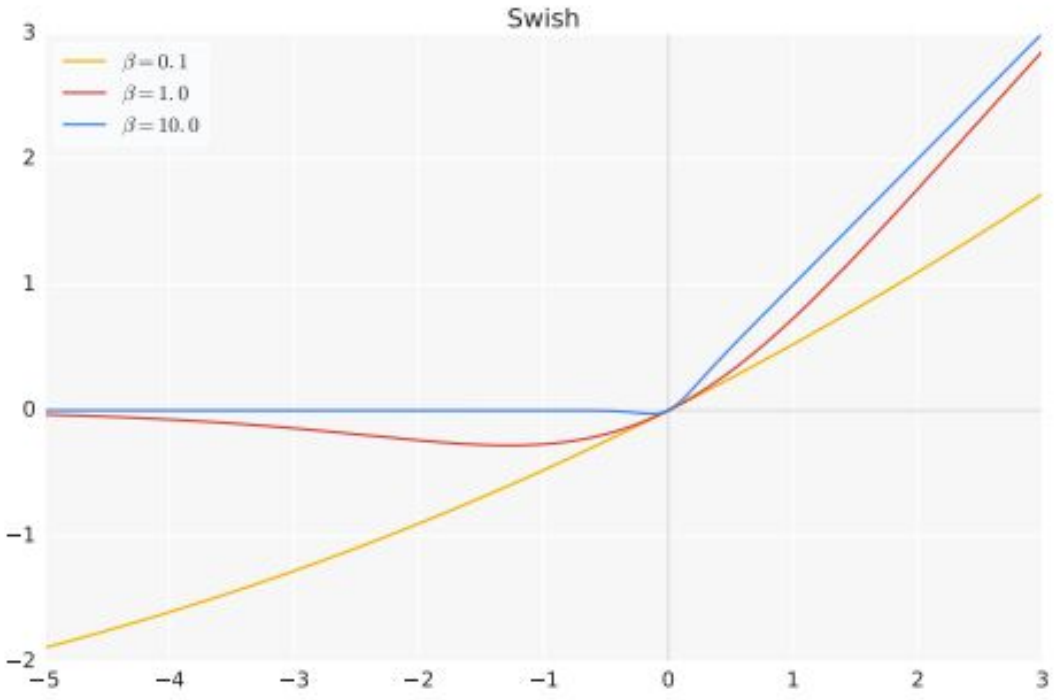
Swish

Found through reinforcement learning to be the best general non-linearity

$$x \cdot sig(\beta x)$$

sig refers to a sigmoid function
Beta is a learnable parameter or can be set to 1 for slightly worse performance

Beta -> inf, then Swish -> ReLu



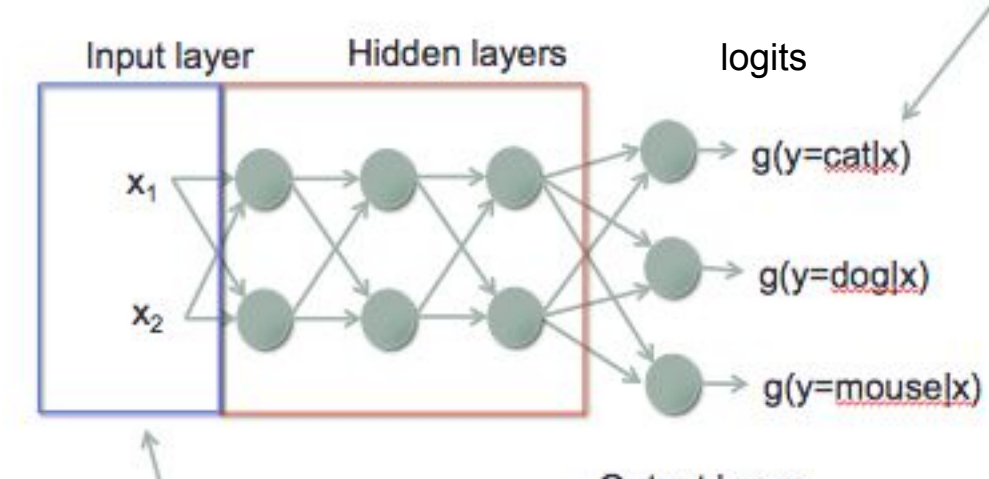
Searching for Activation Functions - arXiv

Proven theoretically to be optimal

Expectation propagation: a probabilistic view of Deep Feed Forward Networks

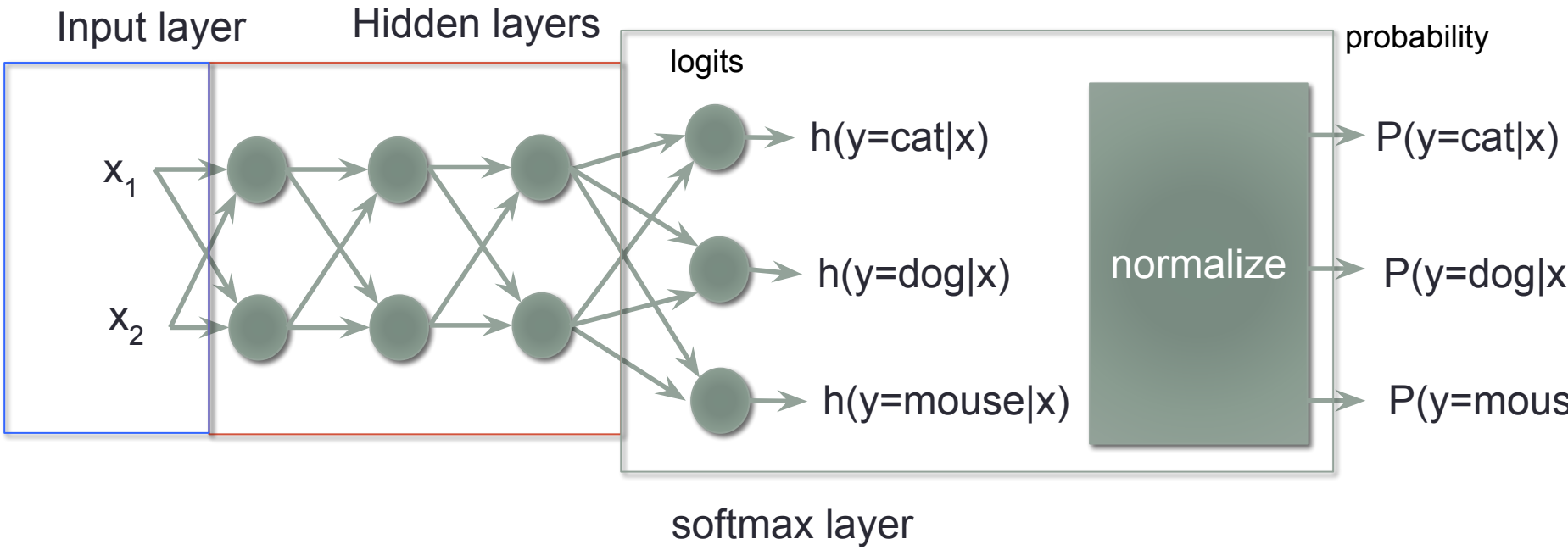
Output layer – Softmax layer

- 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

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



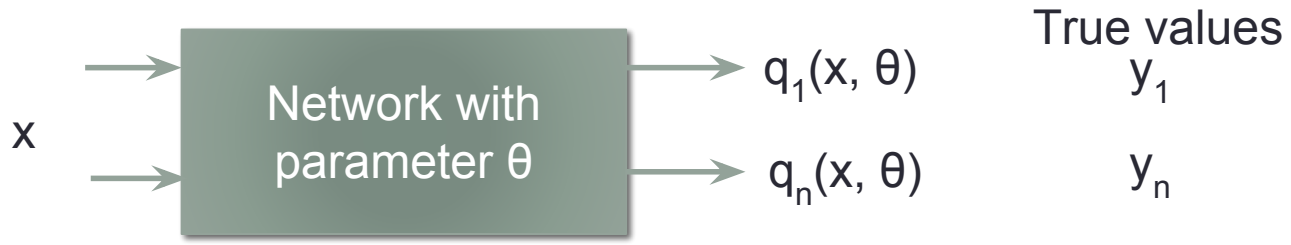
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- Demos
  - Tensorflow, Gcloud, Keras
- CNN, RNN, LSTM, GRU <- Next class

Objective function (Loss function)

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

- Sum of squared errors



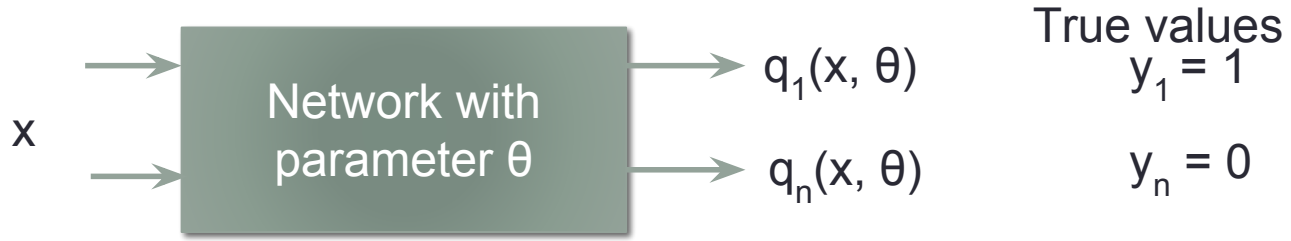
Cross entropy loss

- 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

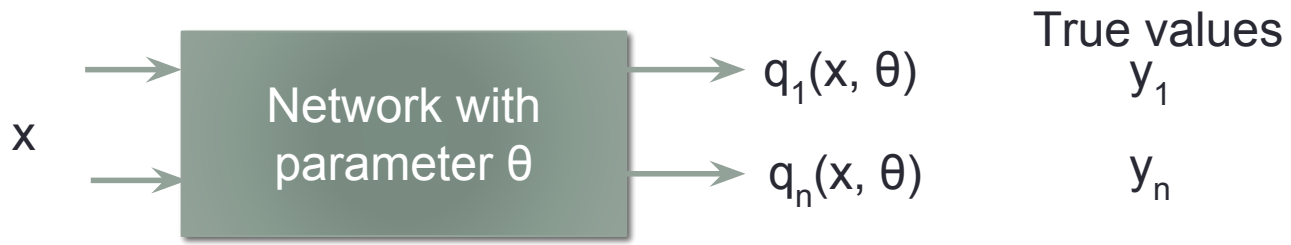


Sum of squared errors (MSE)

- 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



Cross entropy loss & Logarithmic Loss (log loss)

- Minimizing the CE can be considered as the maximizing the log likelihood

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

- Where y_n is 1 if data x comes from class n
  0 otherwise
- For binary class: $L(x_n) = \begin{cases} -\log(h(x_n)) & \text{, if } y_n = 1 \\ -\log(1-h(x_n)) & \text{, if } y_n = 0 \end{cases}$

$$L = [y_n log(h(x_n))] + [(1 - y_n) log(1 - h(x_n))]$$

Same as log likelihood of logistic regression
Negative in front because we are minimizing the loss vs maximizing the probability

$$p(y \mid x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

Probabilistic view of Logistic Regression

- Let's assume, we'll classify as 1 with probability in accordance to the output of

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

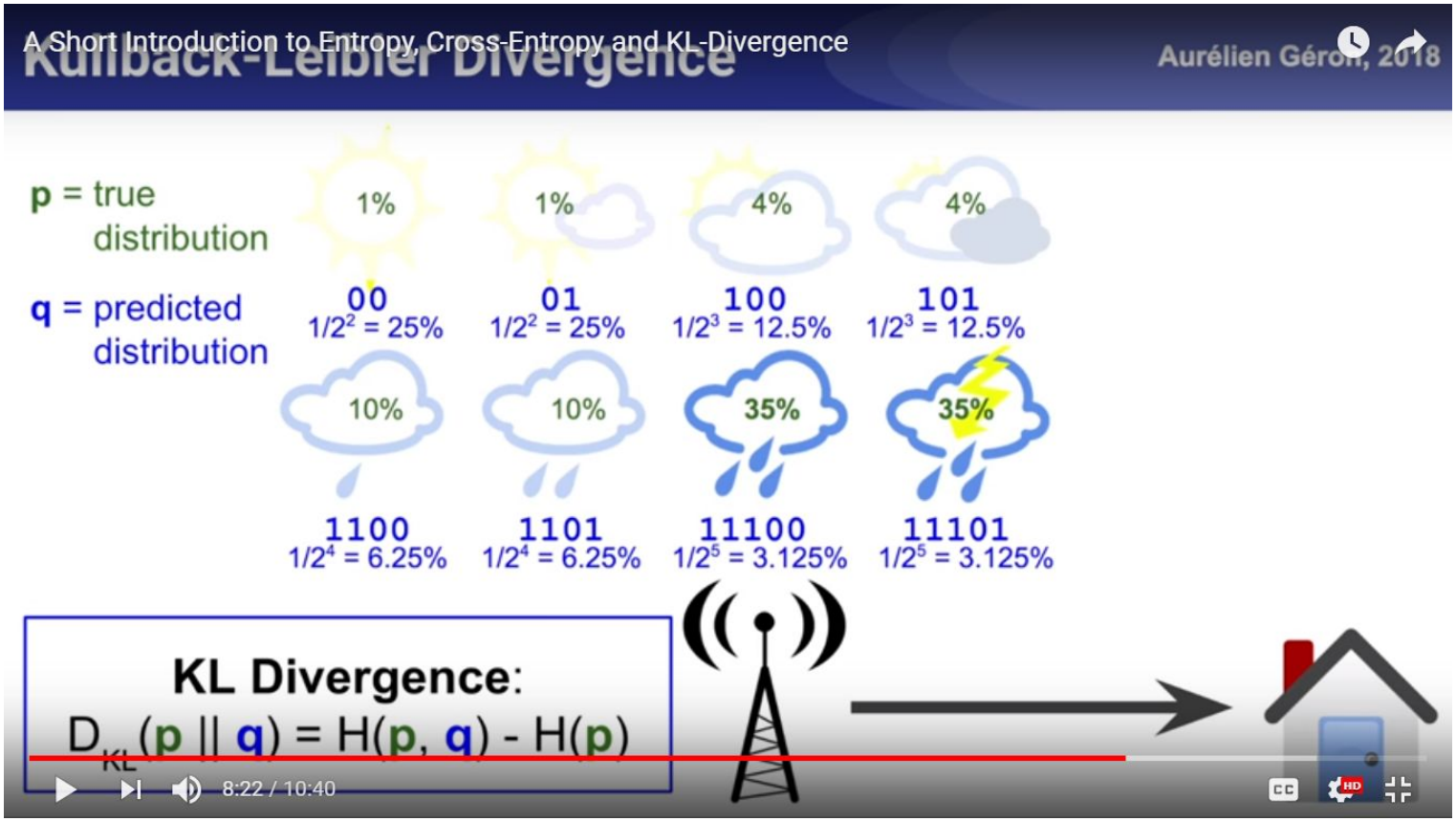
or

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

Other views of CE loss

Relationship between Entropy, CE, and KL Divergence

https://www.youtube.com/watch?v=ErfnhcEV1O8



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

$$\mathbf{w}^T\mathbf{w} + C\Sigma\varepsilon_i$$

MAP estimate is MLE with regularization (the prior term)

Famous types of regularization

- L1 regularization: Regularizing term is a sum
  - $\mathbf{w}^T\mathbf{w} + C\Sigma\varepsilon_i$
- L2 regularization: Regularizing term is a sum of squares
  - $\mathbf{w}^T\mathbf{w} + C\Sigma\varepsilon_i^2$

L2 regularization	L1 regularization
Computational efficient due to having analytical solutions	Computational inefficient on non-sparse cases
Non-sparse outputs	Sparse outputs
No feature selection	Built-in feature selection

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

Regularization in neural networks L2

- We want to improve generalization somehow.
- Observation, models are better when the weights are spread out (no peaky weights).
  - Try to use every part of the model.
- Add a cost if we put some value to the weights

- Regularized loss = Original loss + $0.5\ C\Sigma w^2$

- we sum the square of weights of the whole model
- 0.5 is for prettiness when we take derivative
- C is a hyperparameter weighting the regularization term

Regularization in neural networks

- We want to improve generalization somehow.
- Observation, models behave better when we force the weights to be sparse.
  - Sparse means many weights are zero or close to zero
  - Force the model to focus on only important parts
  - Less prone to noise
- Add a cost if we put some value to the weights
- Regularized loss = Original loss + 0.5 $C\Sigma|w|$

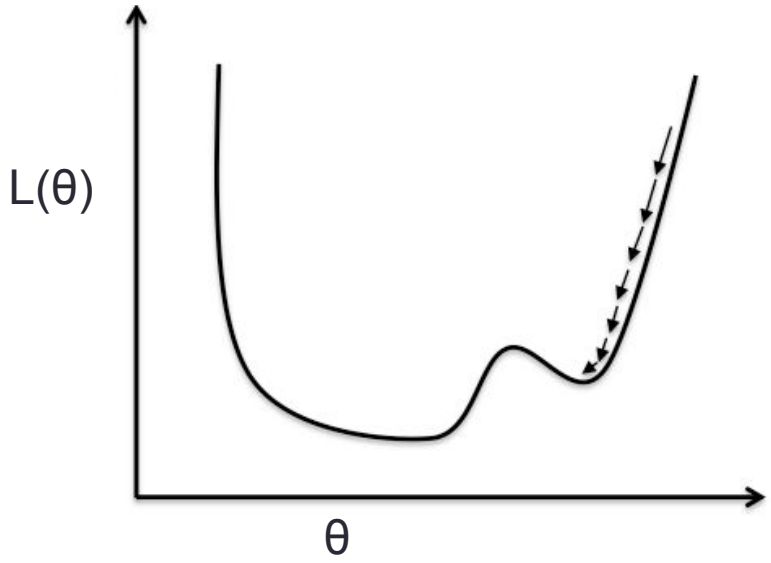
- we sum the absolute weights of the whole model
- 0.5 is for prettiness when we take derivative
- C is a hyperparameter weighting the regularization term

L1 L2 regularization notes

- Can use both at the same time
  - People claim L2 is superior
- I found them useless in practice for deep neural networks
  - Maybe there are some tasks out there that benefit from this? I don't know
- Other regularization methods exist (we will go over these later)

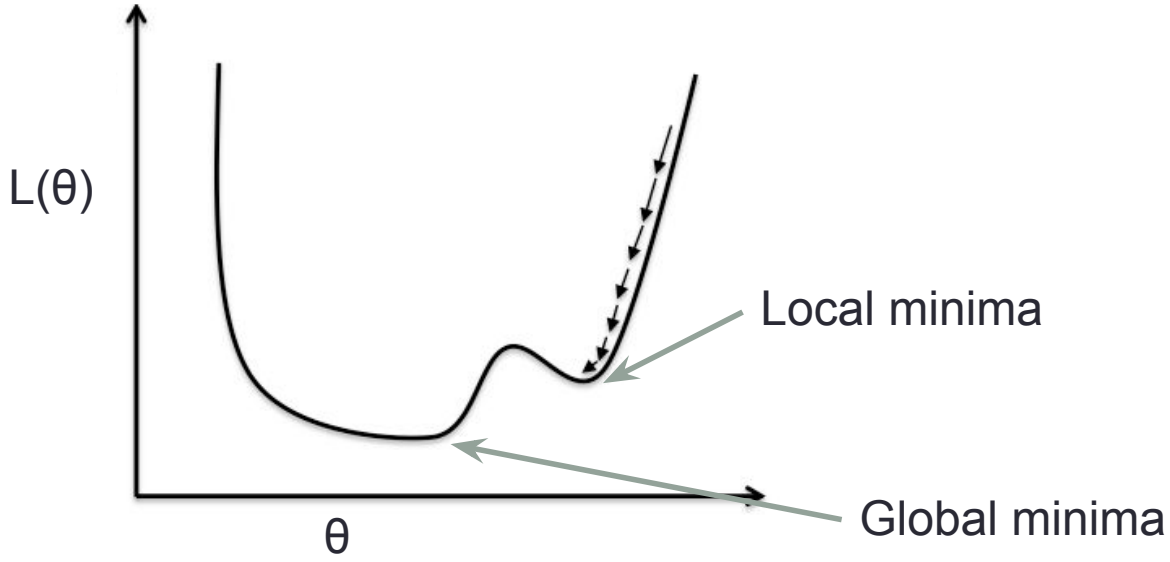
Minimization using gradient descent

- We want to minimize L with respect to θ (weights and biases)
  - Differentiate with respect to θ
  - Gradients passes through the network by Back Propagation



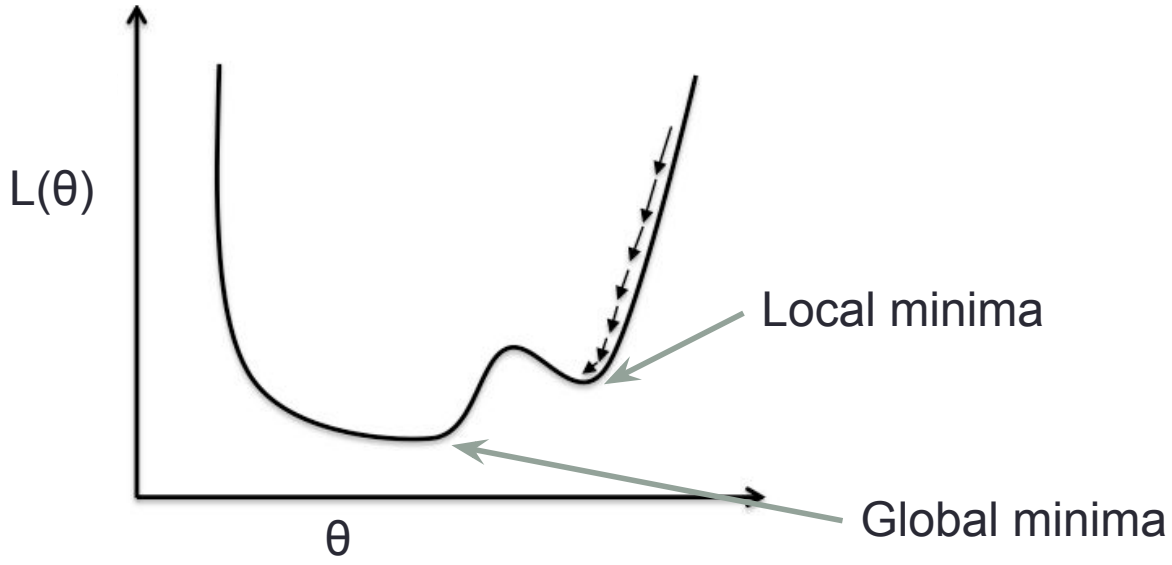
Deep vs Shallow

- The loss function of neural network is non-convex (and non-concave)
  - Local minimas can be avoided with convexity
    - Linear regression, SVM are convex optimization
  - Convexity gives easier training
    - Does not imply anything about the generalization of the model
    - The loss is optimized by the training set



Deep vs Shallow

- If deep, most local minimas are the global minima!
  - Always a way to lower the loss in the network with millions of parameters
  - Enough parameters to remember every training examples
  - Does not imply anything about generalization



Kenji Kawaguchi,Deep Learning without Poor Local Minima, 2016 https://arxiv.org/abs/1605.07110

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

Examples to read

https://alonalj.github.io/2016/12/10/What-is-Backpropagation/

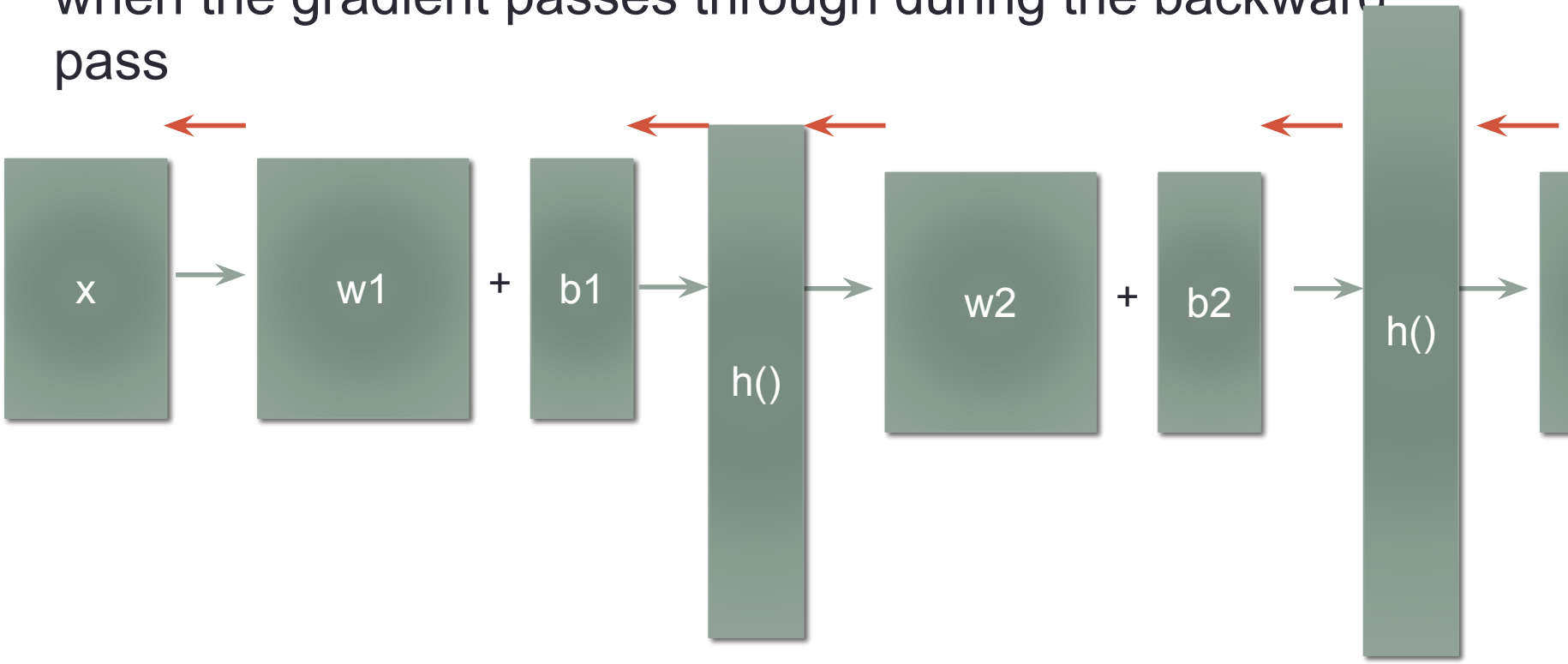
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Back propagation

- Regularization terms only appears at the particular weights when doing the derivative
- What about cross entropy?

Backprop and computation graph

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

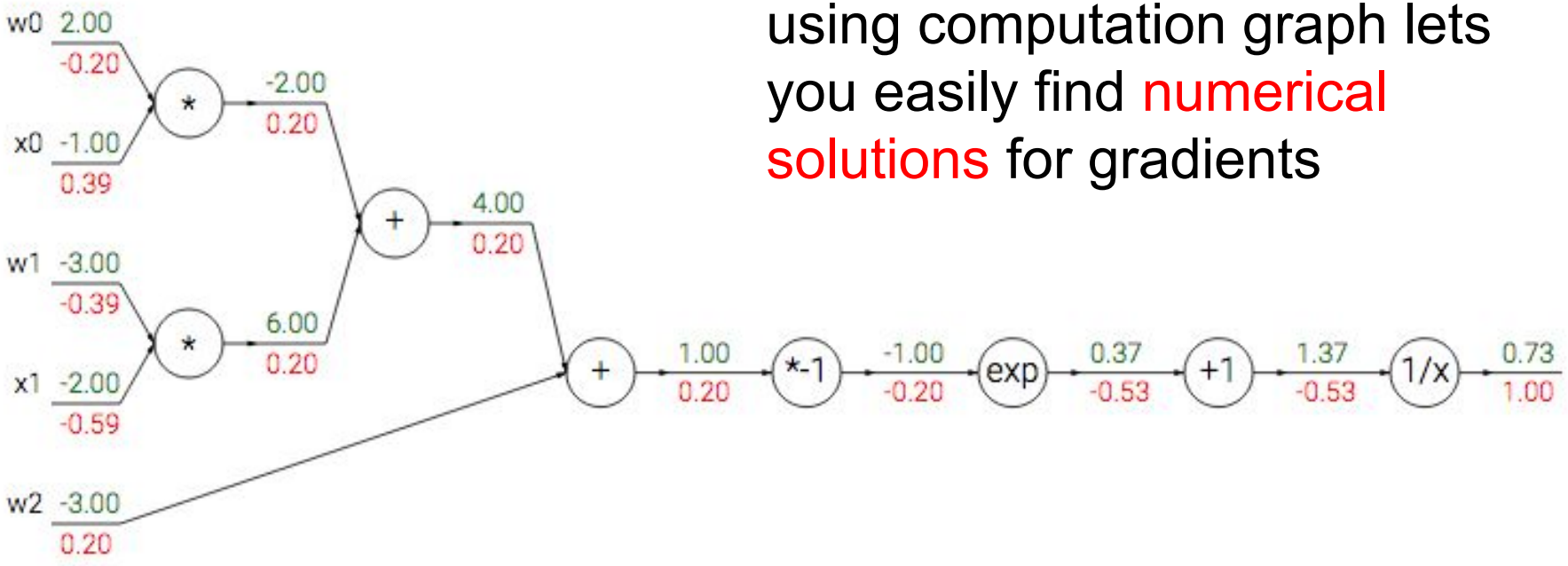


This lets us to build any neural networks without having to redo all the derivation as long as we define a forward and backward computation for the block.

Numerical gradient flow

- Let's find the gradient of $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

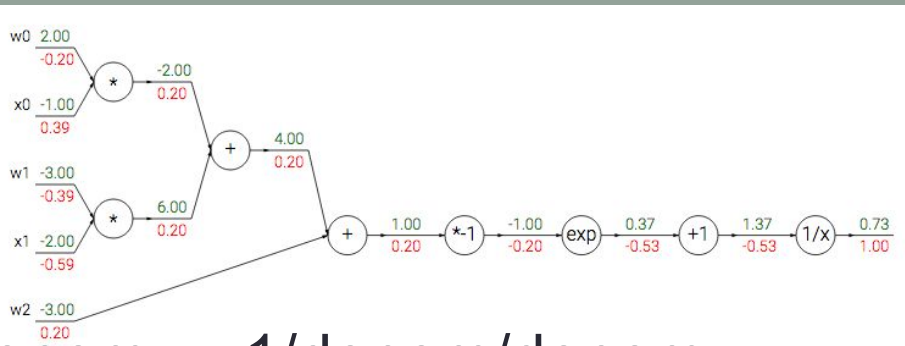
Computation graph



Doing backprop (chain rule) by using computation graph lets you easily find numerical solutions for gradients

http://cs231n.github.io/optimization-2/

$$f(w,x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

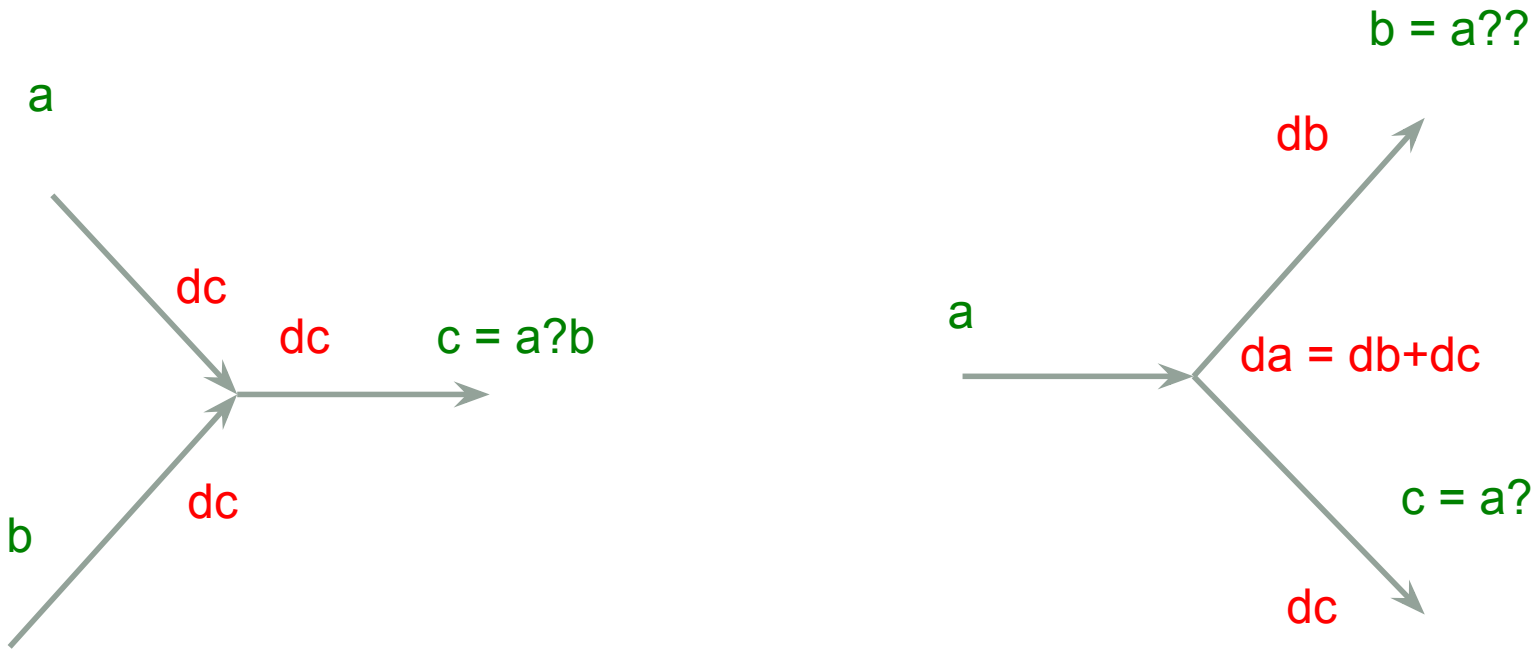


- w = [0, -3, -3]
- x = [-1, -2]
- $t_0 = w[0]*x[0]$
- $t_1 = w[1]*x[1]$
- $t_{01} = t_0 + t_1$
- $t_{012} = t_{01} + w[2]$
- $n_t = -t_{012}$
- $e = \exp(n_t)$
- denom = e+1
- f = 1/denom

- ddenom = -1/denom/denom
- de = 1*ddenom
- $dn_t = \exp(n_t)*de$
- $dt_{012} = -dn_t$
- $dw_2 = 1*dt_{012}$
- $dt_{01} = 1*dt_{012}$
- $dt_0 = 1*dt_{01}$
- $dt_1 = 1*dt_{01}$
- $dw_1 = x[1]dt_1$
- $dx_1 = w[1]dt_1$
- $dw_0 = x[0]dt_0$; $dx_0 = w[0]dt_0$

Perform backward pass in reverse order. No need to explicitly find overall derivative

Gradient flow at forks



Forward and backward pass acts differently at forks

Gradient and non-linearities

We can now talk about how good a non-linearity is by looking at the gradients.

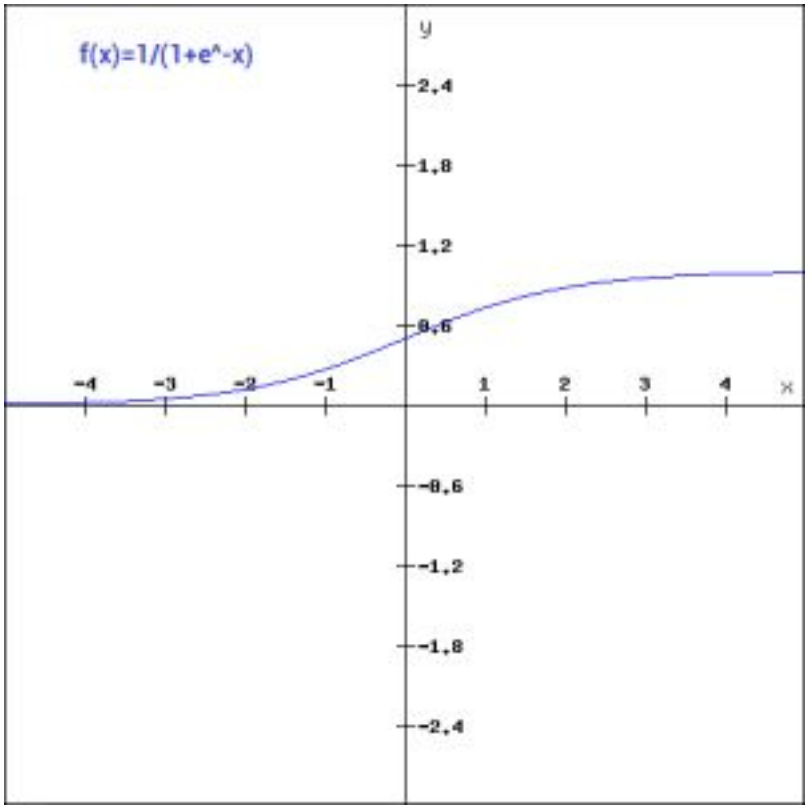
We want

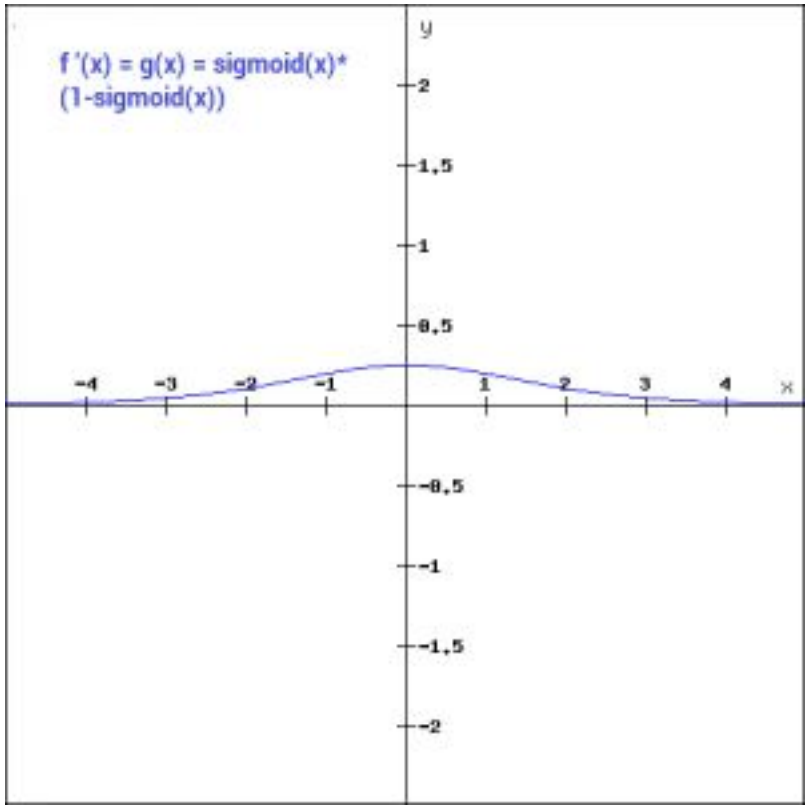
- Something that is differentiable numerically
- Cheap to compute
- Big gradients at every point

Notes on non-linearity

- Sigmoid

  Models get stuck if fall go far away from 0. Output always positive



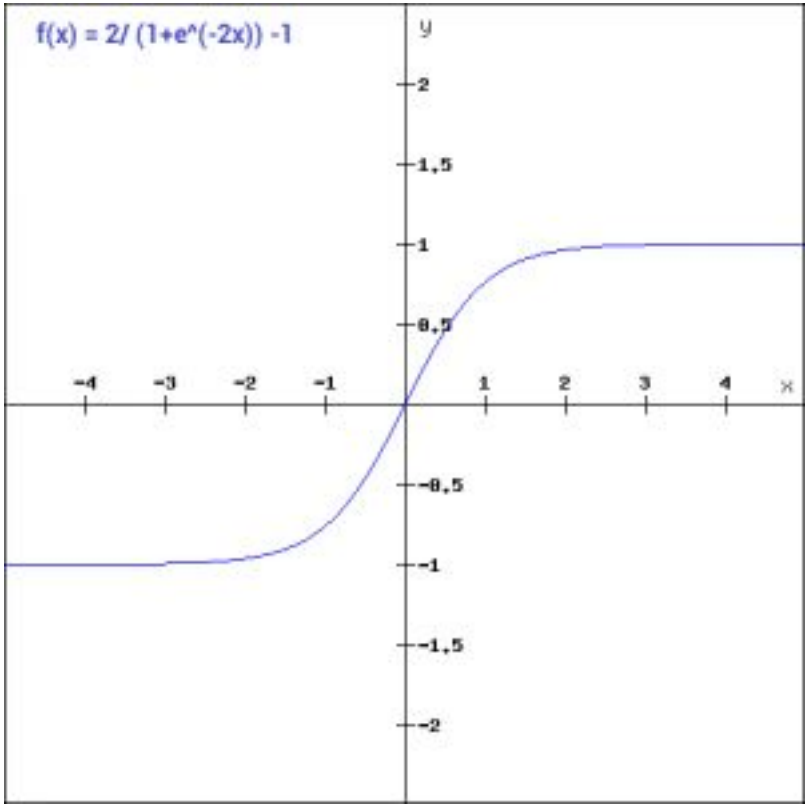


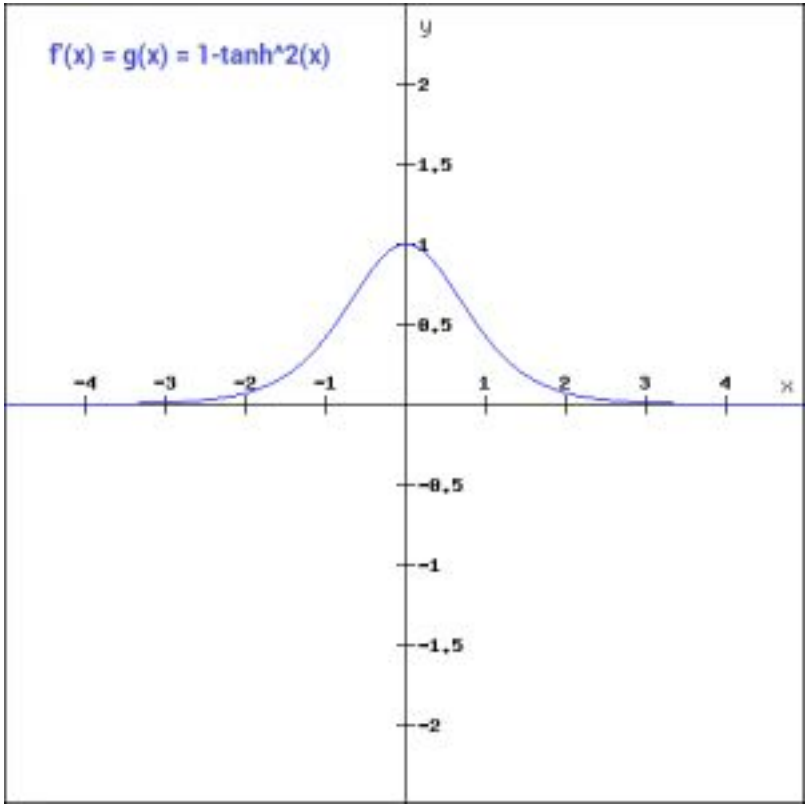
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use

Notes on non-linearity

- Tanh

Output can be +-. Models get stuck if far away from 0



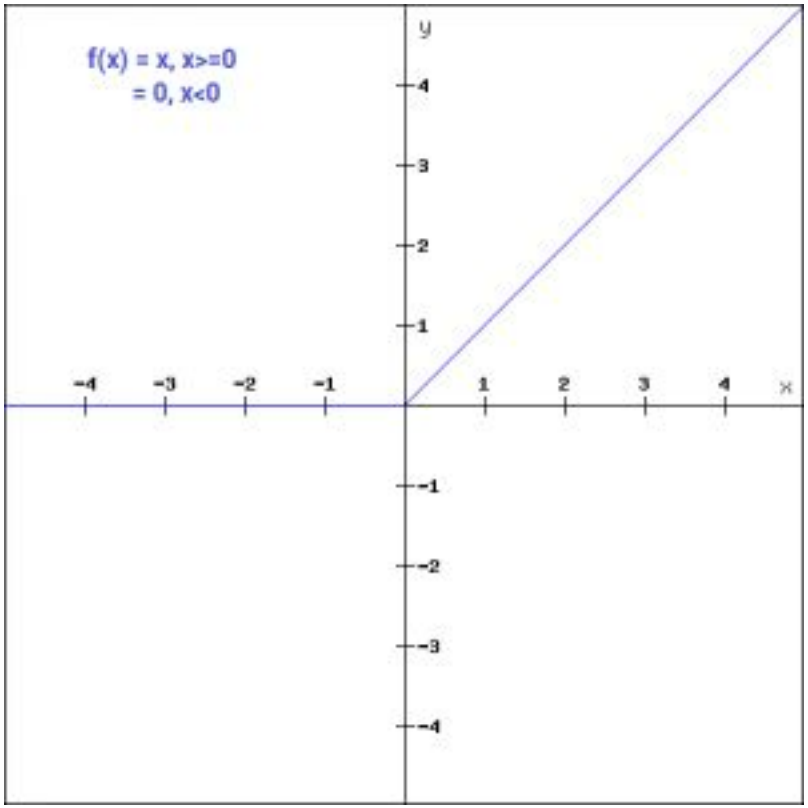


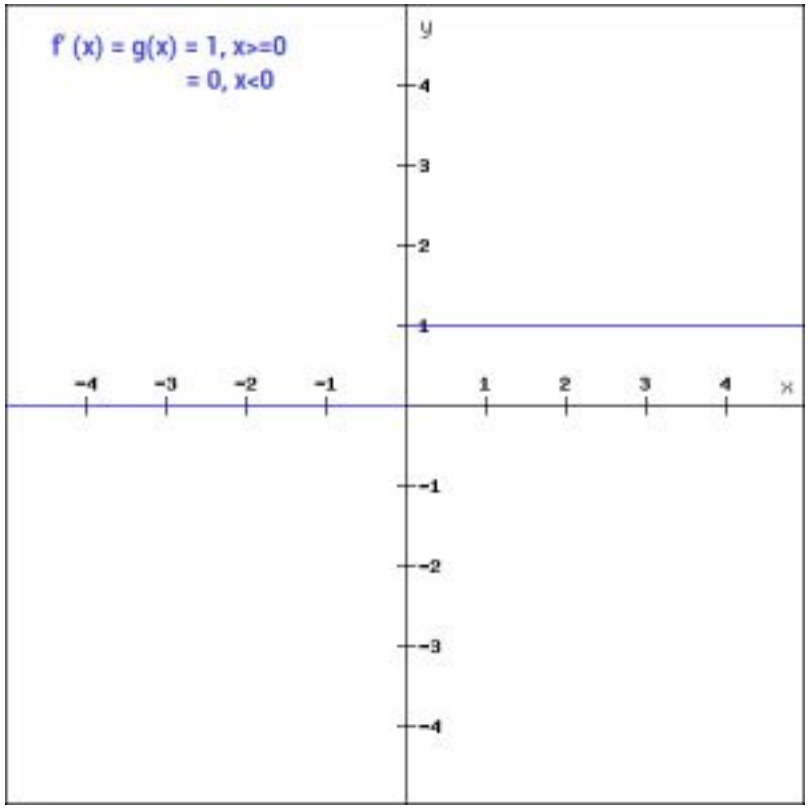
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use

Notes on non-linearity

- ReLU

  High gradient in positive. Fast compute. Gradient doesn't move in negative



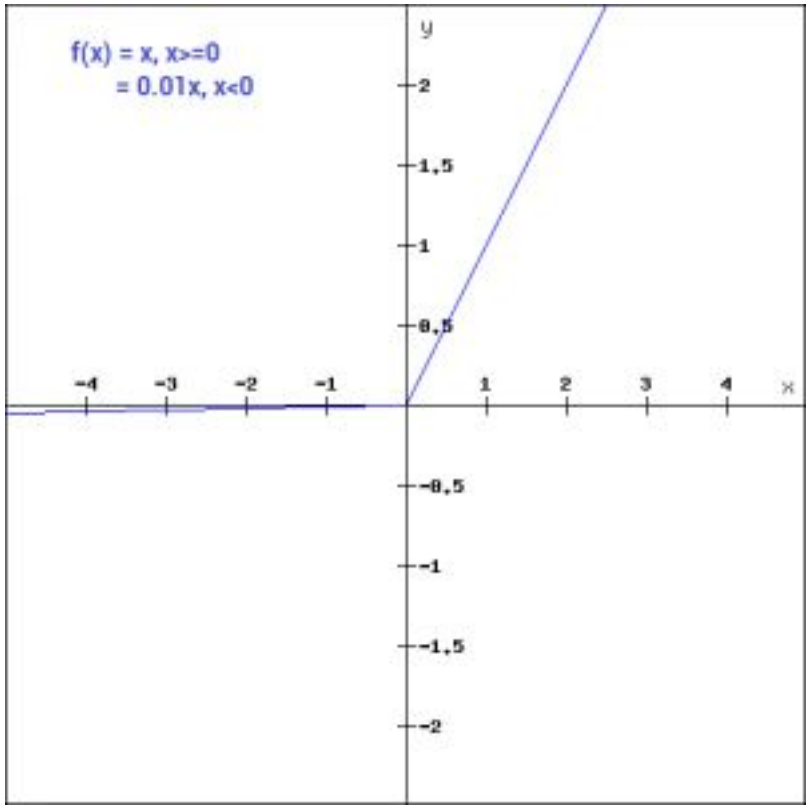


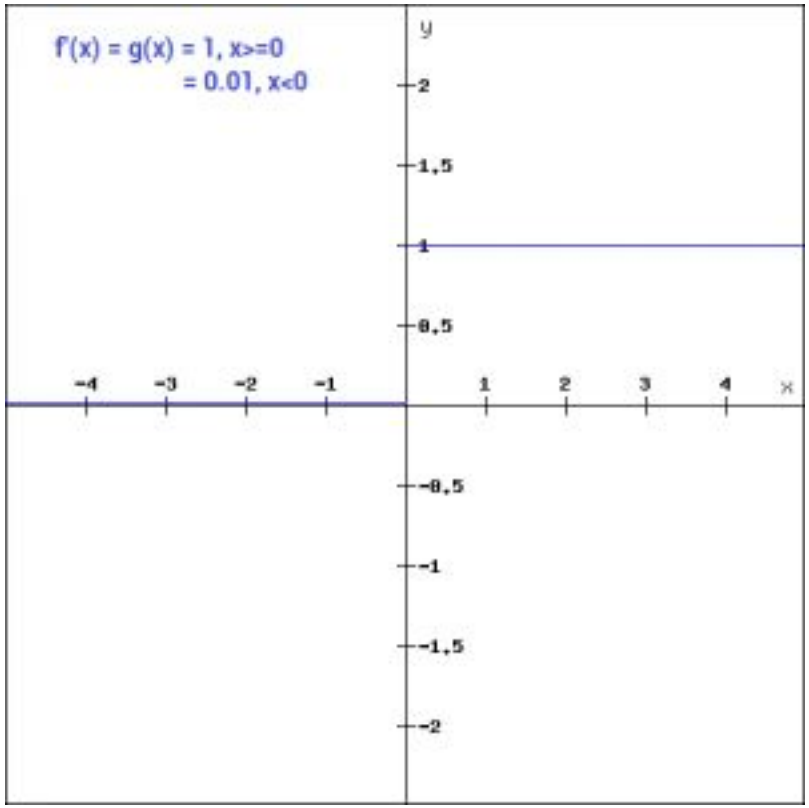
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use

Notes on non-linearity

- Leaky ReLU

  Negative part now have some gradient. Small improvements depending on tasks



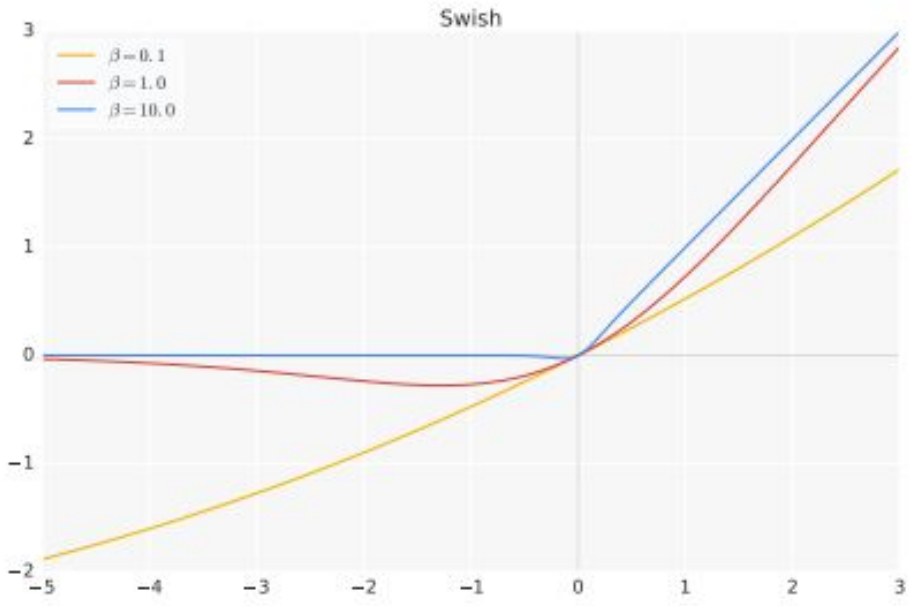


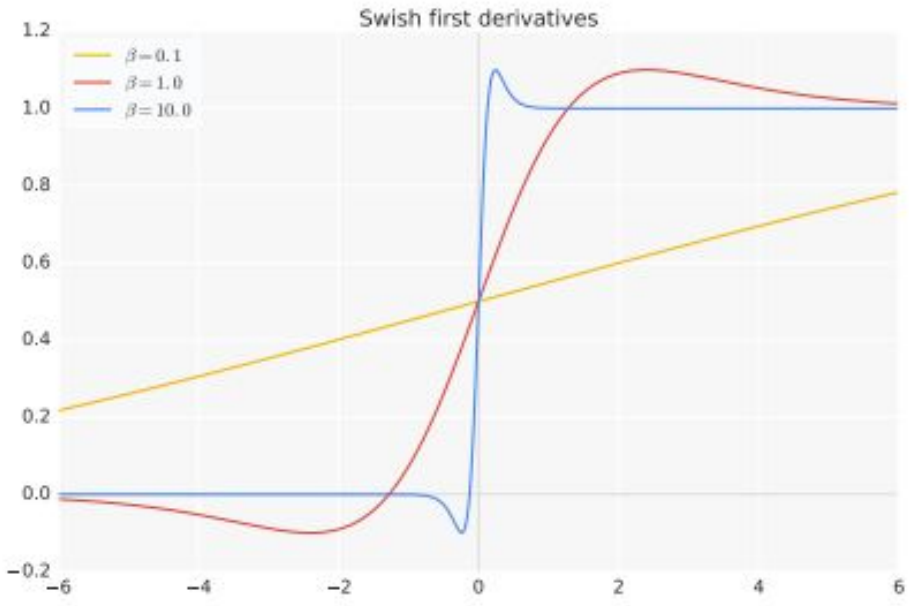
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use

Notes on non-linearity

- Swish

  Nonnegative everywhere. Not monotonic.





[Searching for Activation Functions - arXiv]

Initialization

- The starting point of your descent
- Important due to local minimas
- Not as important with large networks AND big data
- Now usually initialized randomly
  - One strategy

$$W \sim \text{Uniform}(0, \frac{1}{\sqrt{\text{FanIn} + \text{FanOut}}})$$

  - For ReLUs

```
w = np.random.randn(n) * sqrt(2.0/n)
```

  - Or use a pre-trained network as initialization

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010

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

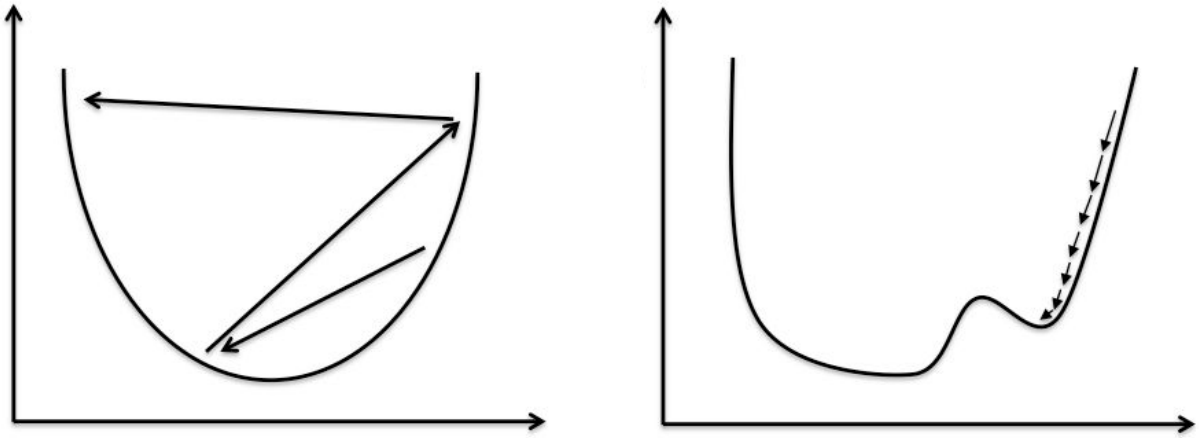
Self regularizing property of SGD

- SGD by its randomized nature does not overfit (as fast)
  - Considered as an implicit regularization (no change in the loss)

https://cbmm.mit.edu/sites/default/files/publications/CBMM-Memo-067-v3.pdf

Learning rate

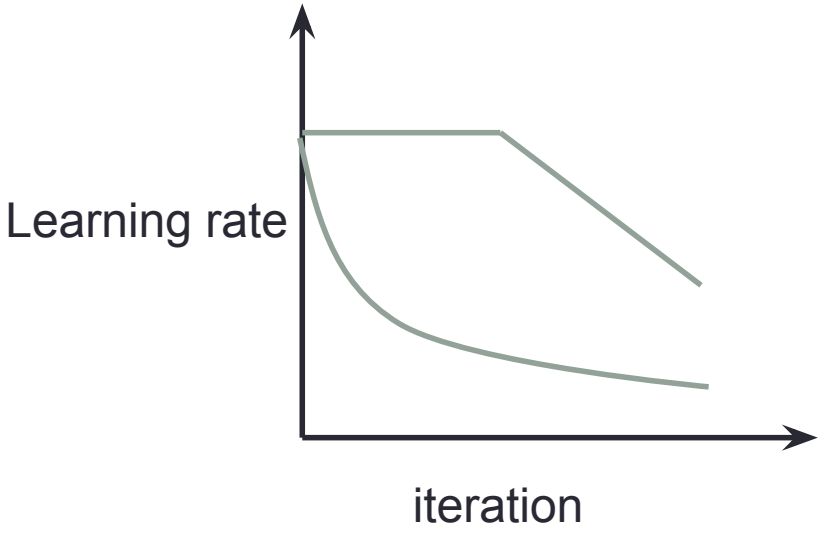
- How fast to go along the gradient direction is controlled by the learning rate
- Too large models diverge
- Too small the model get stuck in local minimas and takes too long to train



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

Learning rate scheduling

- 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)



Learning rate strategies (annealing)

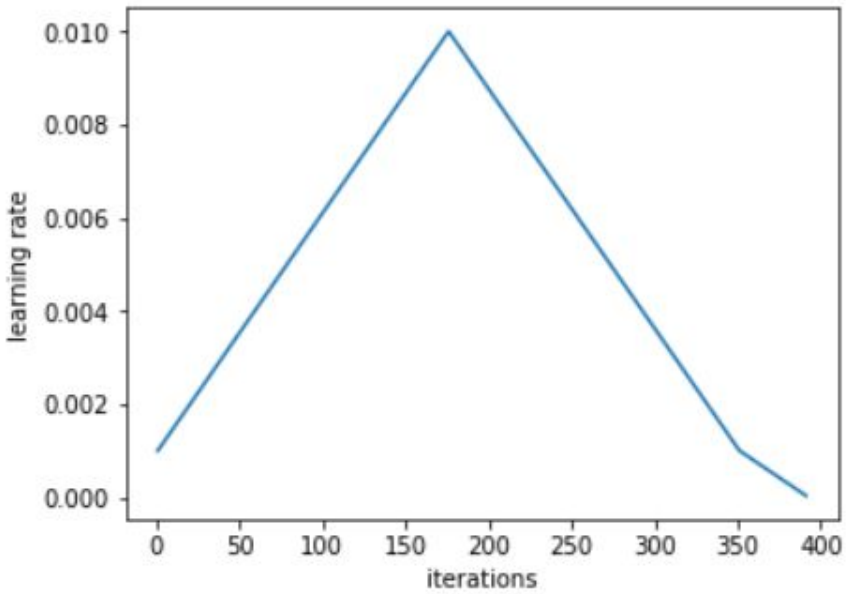
- Step decay: reduce learning rate by x after y epochs
- New bob method: half learning rate every time the validation error goes up. Only plausible in larger tasks
- Exponential decay: multiplies the learning rate by exp(-rate*epoch number)

Learning rate warm up

Initial point of the network can be at a bad spot.

Try not to go to fast - has a warm up period.

Useful for large datasets, or adaption (transfer learning)



Potentially leads to faster convergence and better accuracy

See links below for methods to select the shape of the triangle

https://sgugger.github.io/the-1cycle-policy.html#the-1cycle-policy

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Cyclical Learning Rates for Training Neural Networks

Momentum

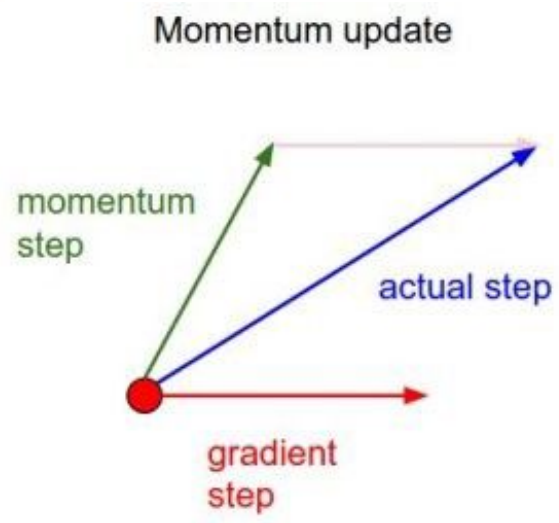
- Gradient descent can get stuck on small local minimas
  - Or slow down at saddle points
- Have concept of speed

speed Momentum rate gradient

$$V_t = \beta V_{t-1} + (1-\beta)\nabla_w L(W, X, y)$$

$$W = W - \alpha V_t$$

learning rate



https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
http://cs231n.github.io/neural-networks-3/

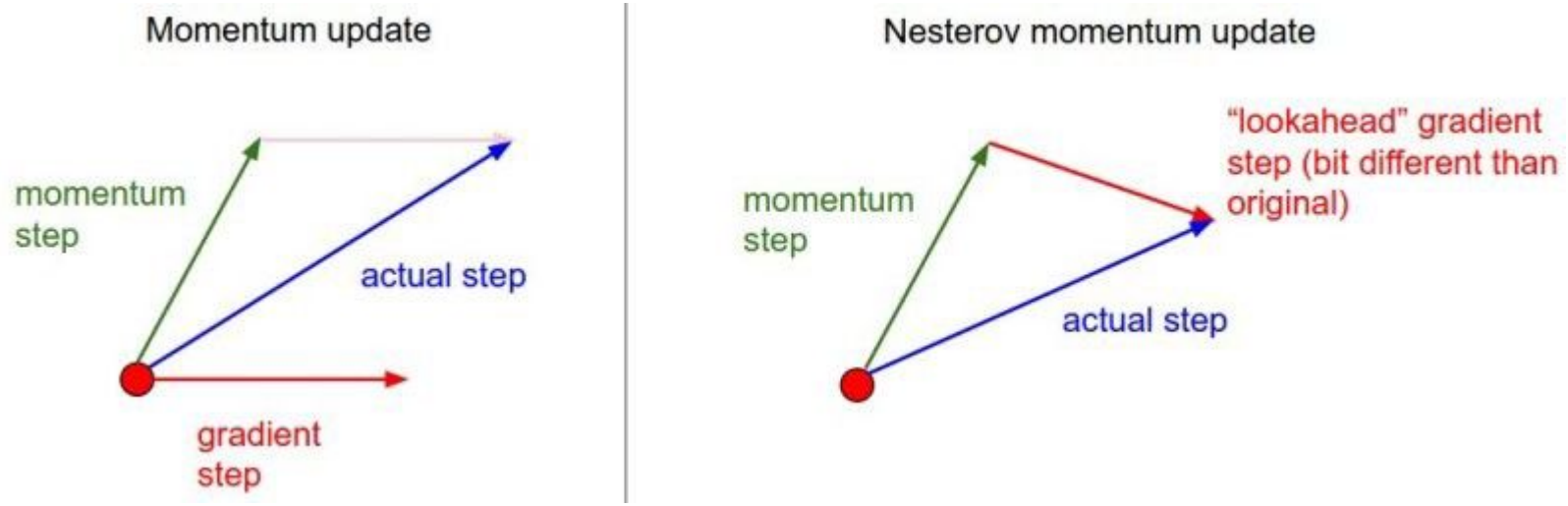
Nesterov Momentum

- Momentum with look ahead.
  - Compute gradient as if we took an additional step

$$V_t = \beta V_{t-1} + \alpha \nabla_w L(W - \beta V_{t-1}, X, y)$$

$$W = W - V_t$$

gradient is computed as if we took a step



https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
http://cs231n.github.io/neural-networks-3/

Adaptive learning rates

How to have the updates be different for different layers?

How to have the momentum estimates take into account of higher moments (acceleration)?

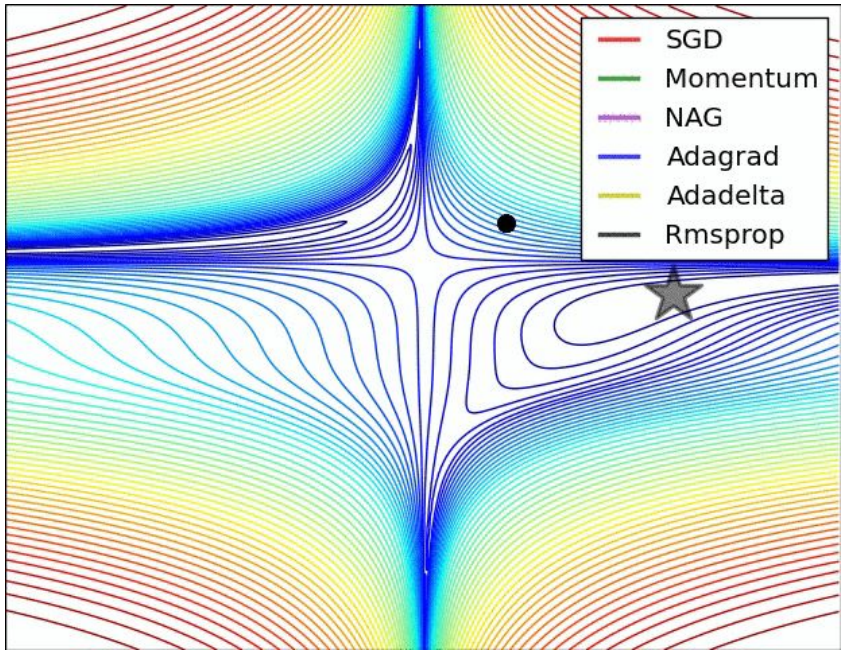
RMSProp and ADAM

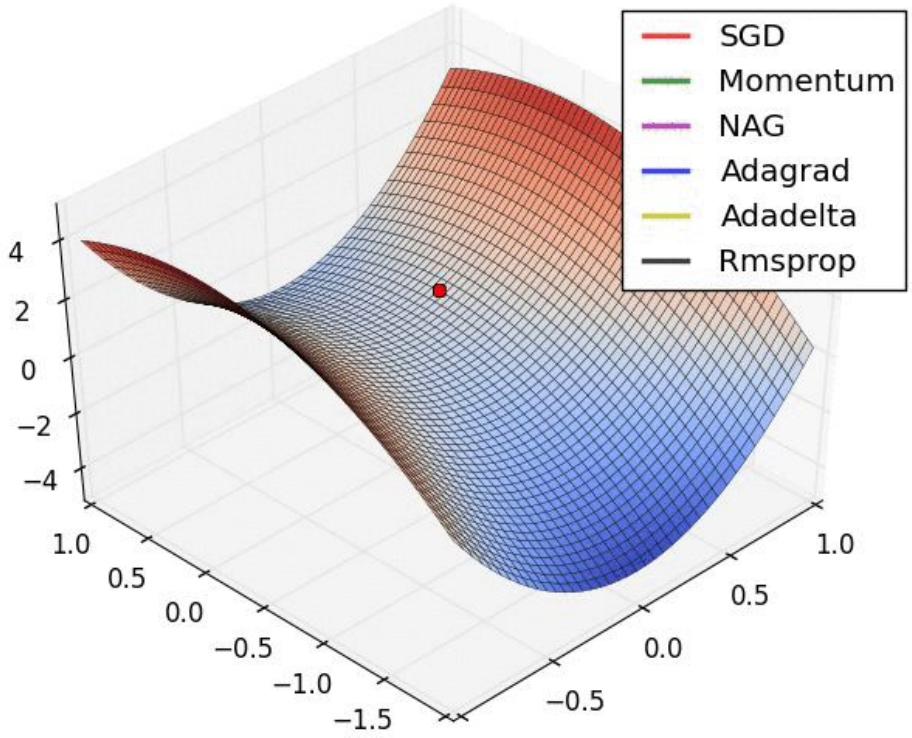
You still need to do learning rate scheduling

More details see

http://ruder.io/optimizing-gradient-descent/index.html#whichoptimizertochoose

Optimization method and speed





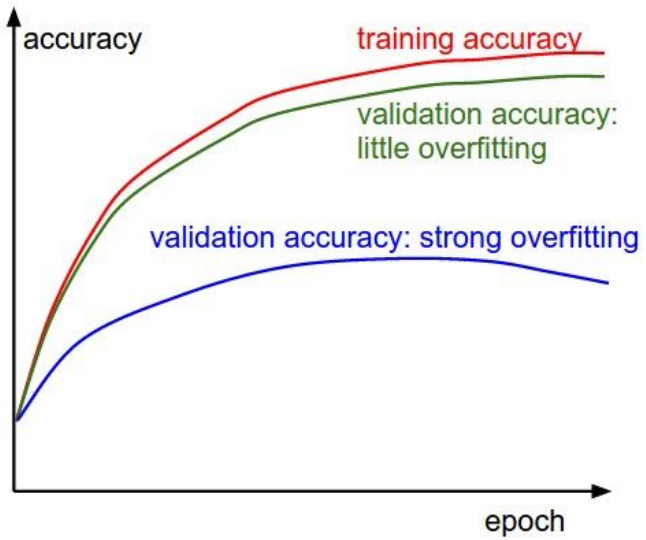
http://ruder.io/optimizing-gradient-descent/index.html#whichoptimizertochoose

Learning rate tricks

- At least decay the learning rate
  - Monitor validation set performance
- If the loss never goes down -> decrease the learning rate (by factor of 10)
- Start with ADAM. Also try RMSprop and SGD with Nesterov Momentum if you have time

Overfitting

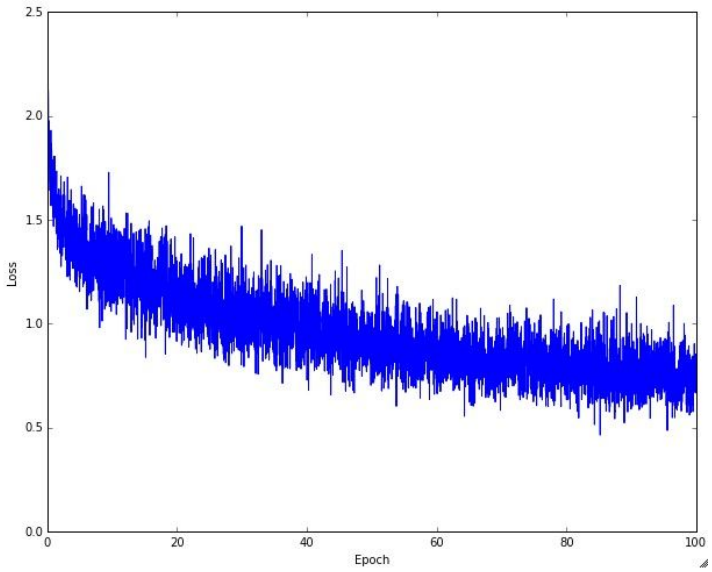
- You can keep doing back propagation forever!
- The training loss will always go down
- But it overfits
- Need to monitor performance on a held out set
- Stop or decrease learning rate when overfit happens



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

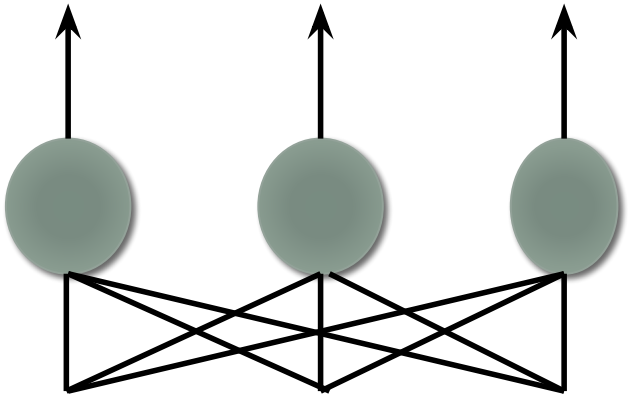
A implicit regularization technique for reducing overfitting

Randomly turn off different subset of neurons during training

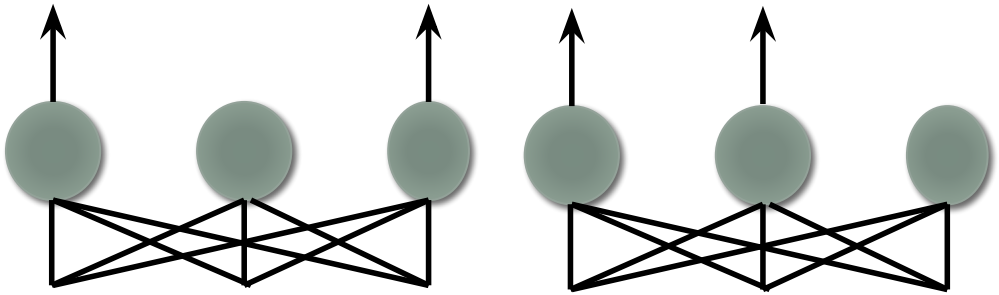
Network no longer depend on any particular neuron

Force the model to have redundancy – robust to any corruption in input data

A form of performing model averaging (ensemble of experts)



Model

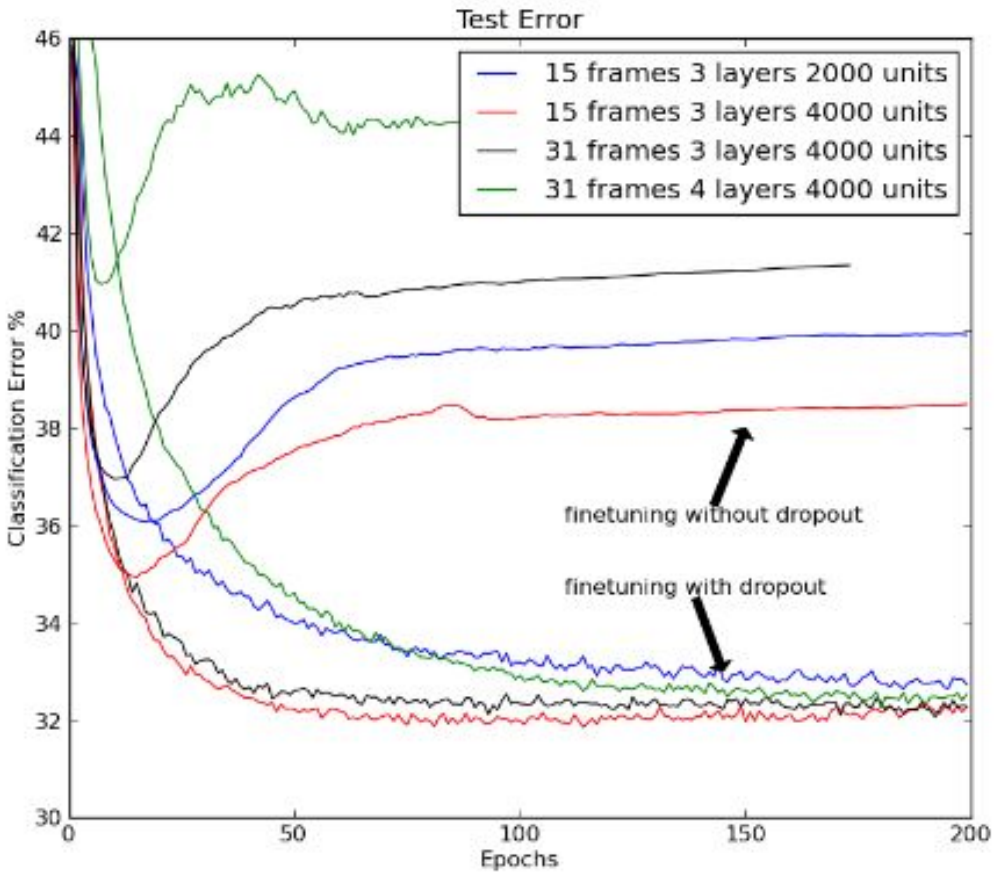


Dropout rate of 0.33

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

Batch normalization

- Recent technique for (implicit) regularization
- Normalize every mini-batch at various batch norm layers to standard Gaussian (different from global normalization of the inputs)
- Place batch norm layers before non-linearities
- Faster training and better generalizations

For each mini-batch that goes through batch norm

1. Normalize by the mean and variance of the mini-batch for each dimension
2. Shift and scale by learnable parameters

Replaces dropout in some networks

https://arxiv.org/abs/1502.03167

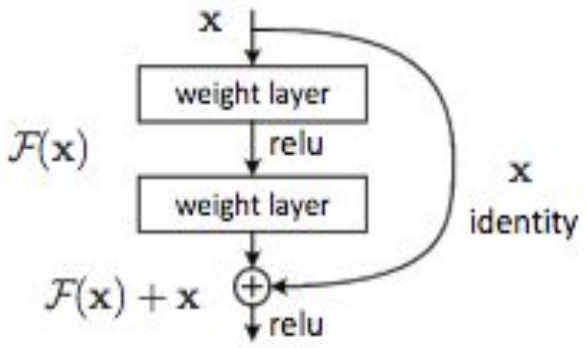
$$\hat{x} = \frac{x - \mu_b}{\sigma_b}$$

$$y = \alpha\hat{x} + \beta$$

Vanishing/Exploding gradient

- Backprop introduces many multiplications down chain
- The gradient value gets smaller and smaller
  - The deeper the network the smaller the gradient in the lower layers
  - Lower layers changes too slowly (or not at all)
  - Hard to train very deep networks (>6 layers)
- The opposite can also be true. The gradient explodes from repeated multiplication
  - Put a maximum value for the gradient (Gradient clipping)

- How to deal with this?
  - Residual connection



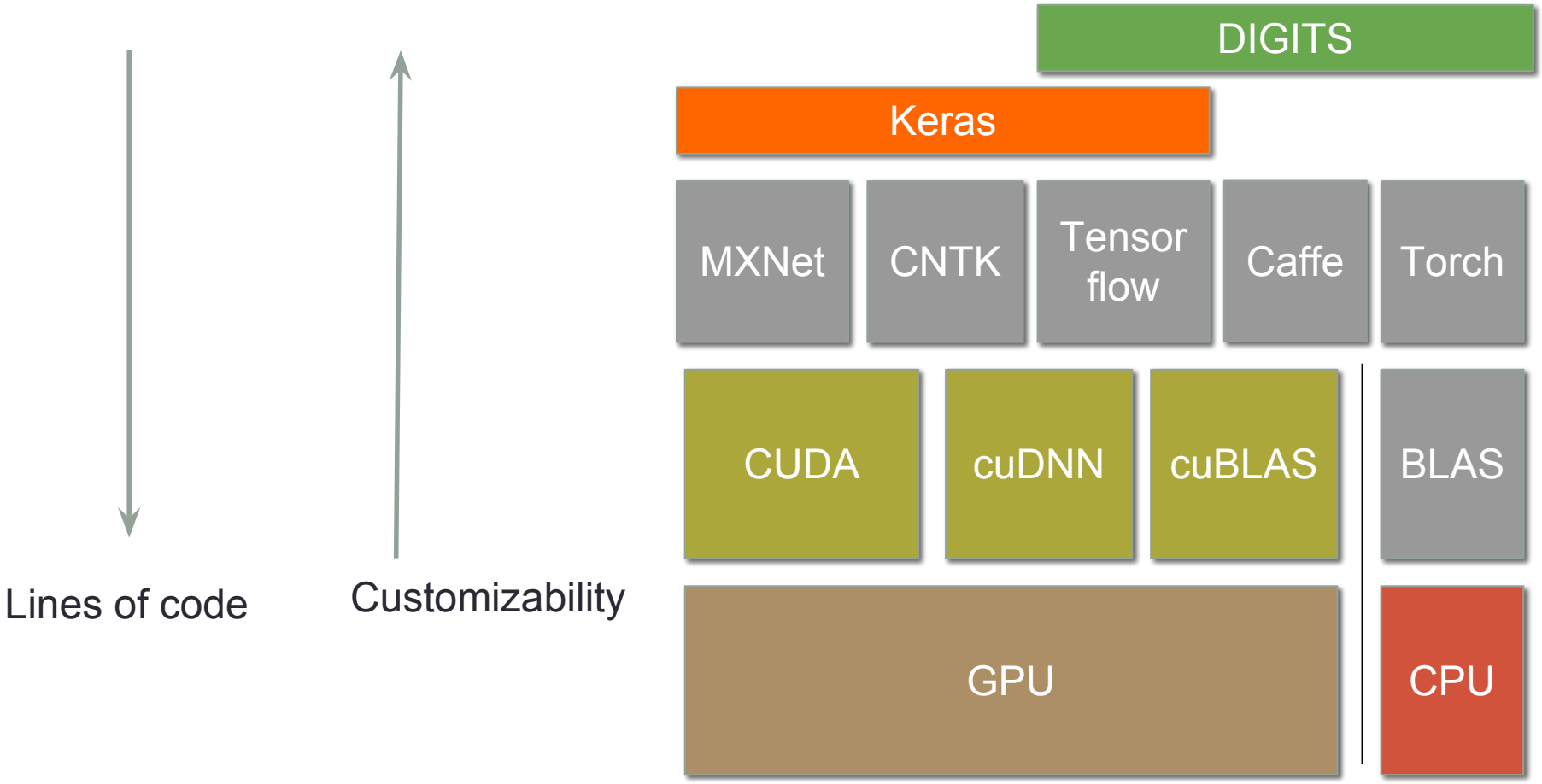
https://arxiv.org/pdf/1512.03385.pdf

Neural networks

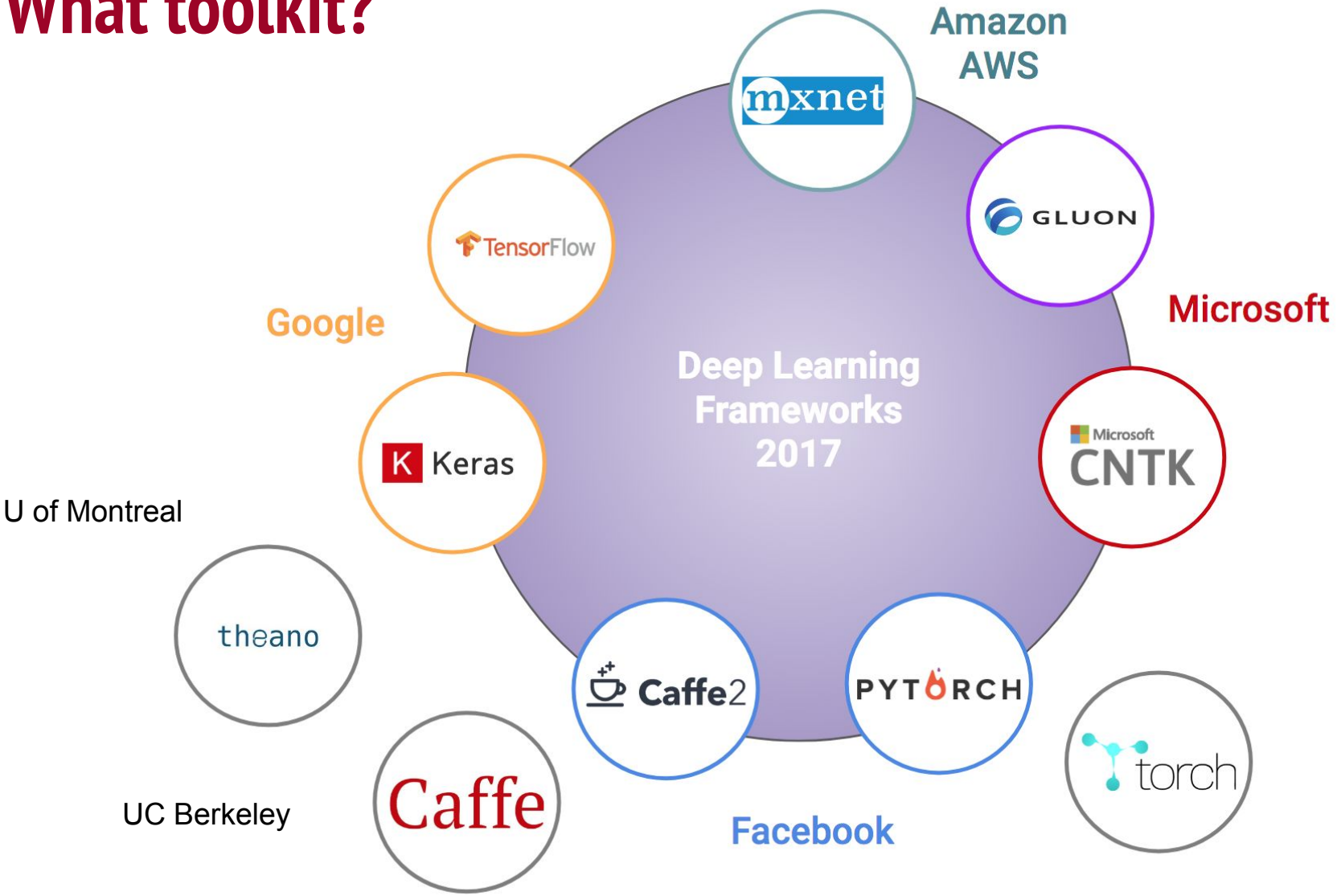
- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- Demos
  - Tensorflow, Gcloud, Keras
- CNN, RNN, LSTM, GRU <- Next class

What toolkit

Tradeoff between customizability and ease of use



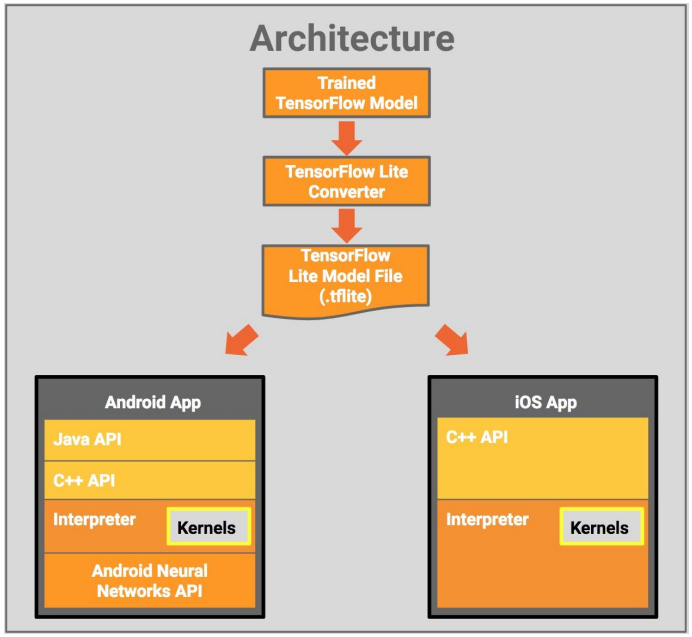
What toolkit?



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

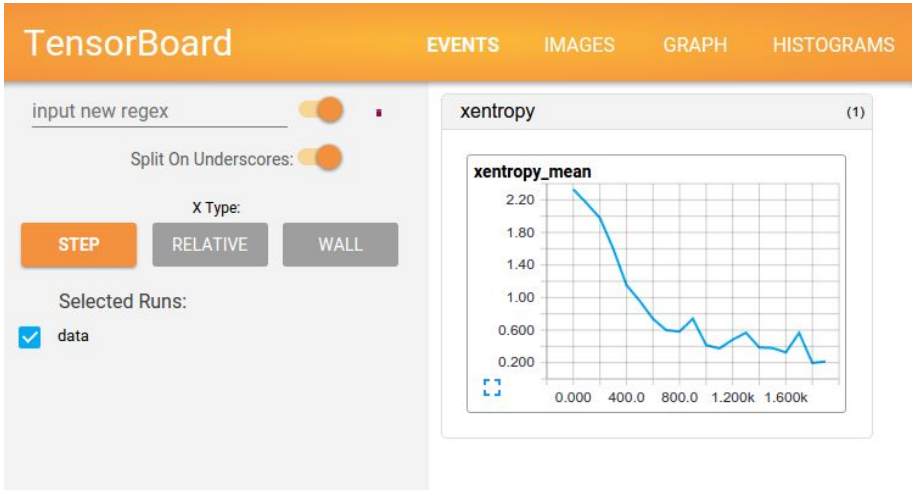
Which?

- 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.

Demos

- Tensorboard
  - What's a tensor?
- Gcloud