Python for Data Science 2

Lecture 22- Deep Learning

Amir Farbin

Deep Neural Network Architectures

- The most generic Deep Neural Network is a composition of functions:
  - Define n^{th} layer as $F_n(\mathbf{X}) = f\ (\ \mathbf{W_n}\ \mathbf{X} + \mathbf{b_n}\ )$
  - n layer network: $\mathbf{Y} = F_n(F_{n-1}(F_{n-2}(\ldots F_1(\mathbf{X}))))$
- The number of layers and the number of neurons per layer (size of matrices) are hyper-parameters.
- Consider Convolutional Neural Net (described in previous lecture)
  - can be expressed in the same way with addition that various parameters are shared between the different layers.
  - this a type of Neural Network Architecture
- A large variety Neural Network Architectures exist… depends on type of input data.
- In addition, there are various problem formulations
  - What are inputs and outputs
  - Training protocols
  - …

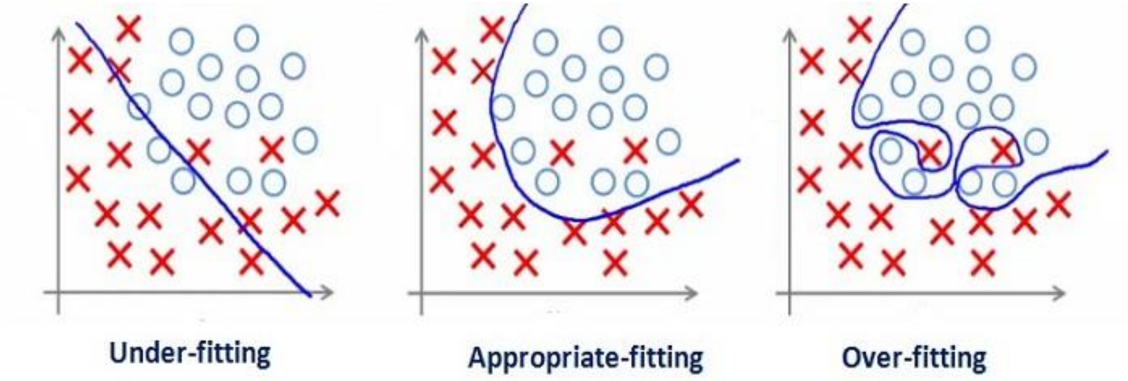
- Tensorflow Playground

Overfitting

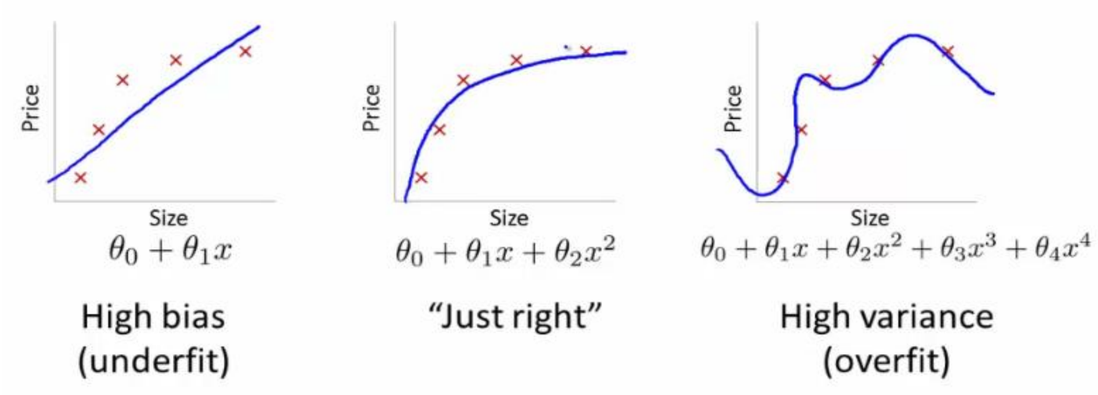
Over Fitting

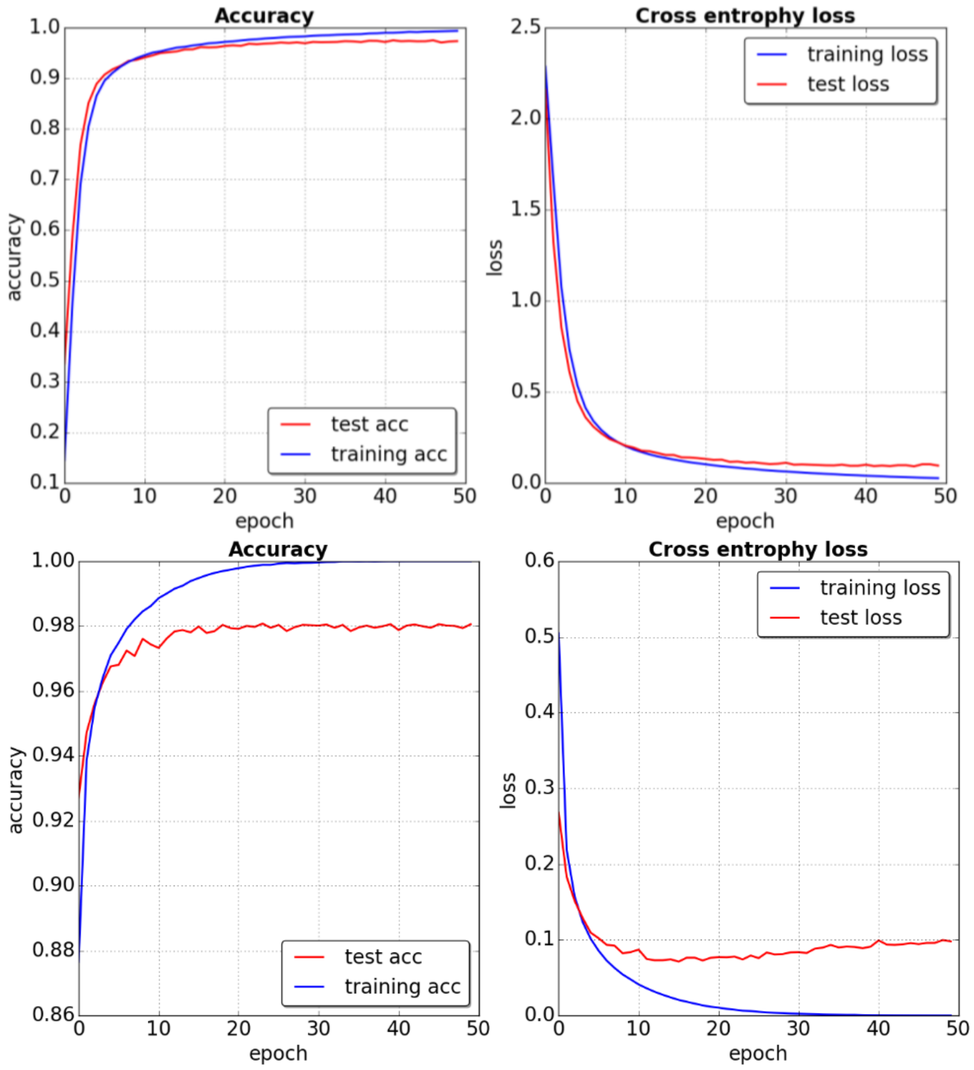
- **Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit has poor predictive performance, as it overreacts to minor fluctuations in the training data.**

Classification:



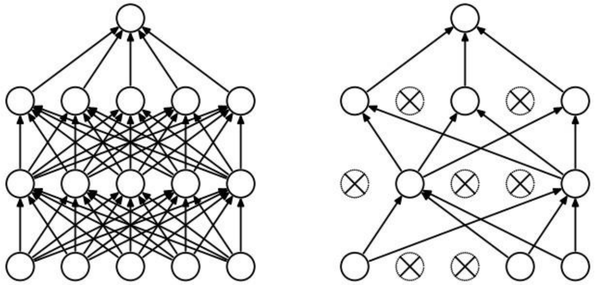
Regression:




Accuracy
accuracy
epoch
test acc
training acc
Cross entrophy loss
loss
epoch
training loss
test loss
Accuracy
accuracy
epoch
test acc
training acc
Cross entrophy loss
loss
epoch
training loss
test loss

Regularization - Dropout

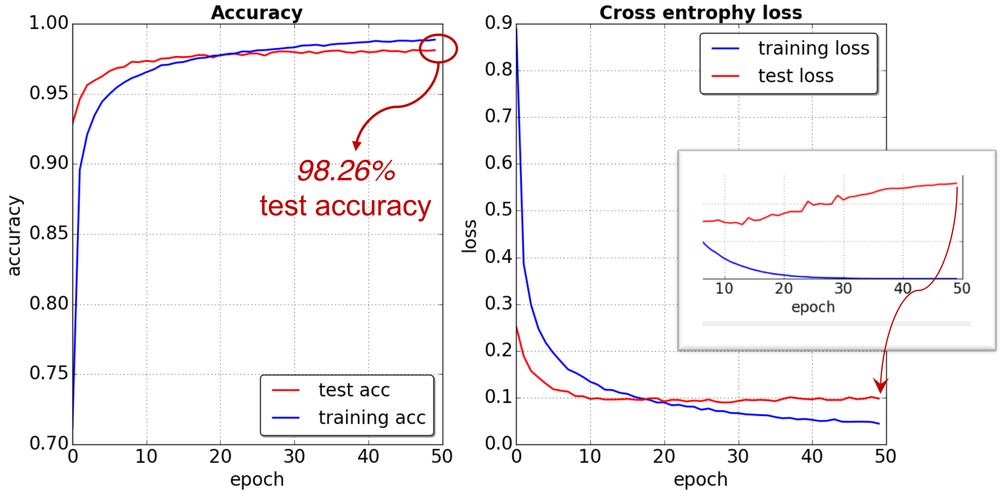
- **Dropout is an extremely effective, simple and recently introduced regularization technique by Srivastava et al (2014).**



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

- **While training, dropout is implemented by only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise.**
- **It is quite simple to apply dropout in Keras.**

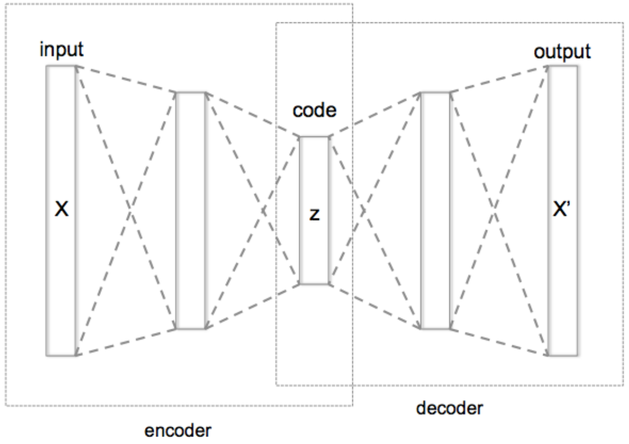
```
# apply a dropout rate 0.25 (drop 25% of the neurons)
model.add(Dropout(0.25))
```

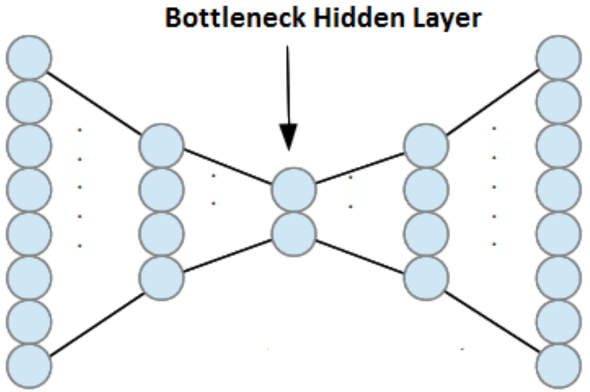


Unsupervised Learning

Semi-supervised Learning

- Basic idea: Train network to ***reproduce the input***.
- Example: ***Auto-encoders***
  - ***De-noising auto-encoders***: add noise to input only.
  - ***Sparse auto-encoders***:
    - ***Sparse latent (code) representation*** can be exploited for ***Compression, Clustering, Similarity testing,*** …
  - ***Anomaly Detection***
    - Reconstruction Error
    - Outliers in latent space





(a) Azimuth (pose)

(b) Elevation

(c) Lighting

(d) Wide or Narrow

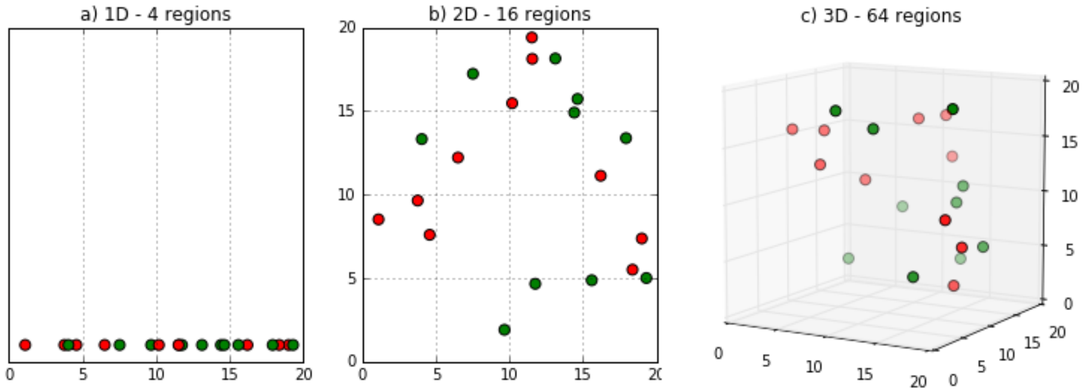
Figure 3: **Manipulating latent codes on 3D Faces:** We show the effect of the learned continuous latent factors on the outputs as their values vary from -1 to 1. In (a), we show that one of the continuous latent codes consistently captures the azimuth of the face across different shapes; in (b), the continuous code captures elevation; in (c), the continuous code captures the orientation of lighting; and finally in (d), the continuous code learns to interpolate between wide and narrow faces while preserving other visual features. For each factor, we present the representation that most resembles prior supervised results [7] out of 5 random runs to provide direct comparison.

https://arxiv.org/pdf/1606.03657.pdf

Feature Learning

Motivations

- Curse of Dimensionality
- Smoothness
- Manifold Learning



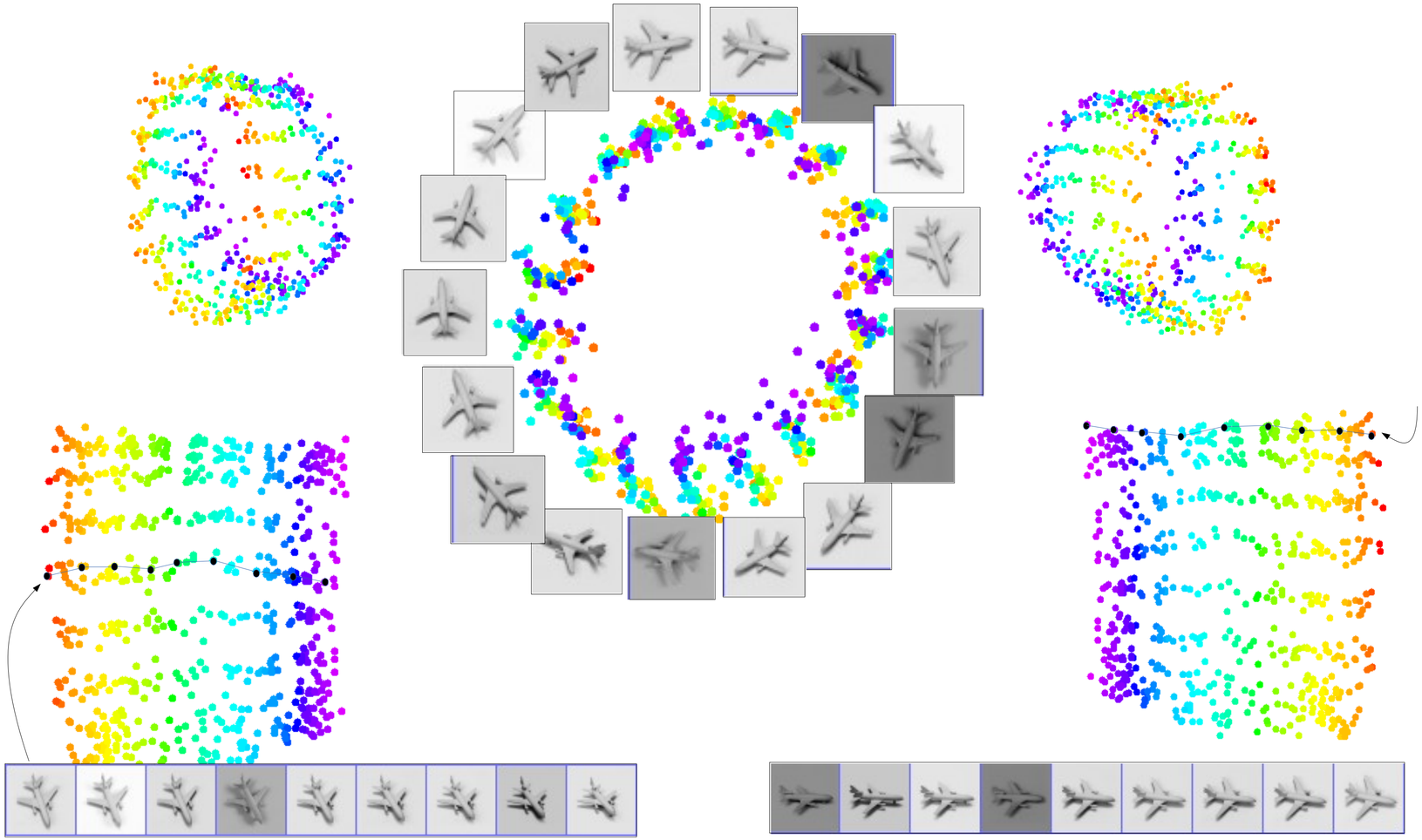
Reference

- Natural Data Lives in low dimensional (Non-Linear) Manifold.

Data Manifold & Invariance: Some variations must be eliminated

Y LeCun
MA Ranzato

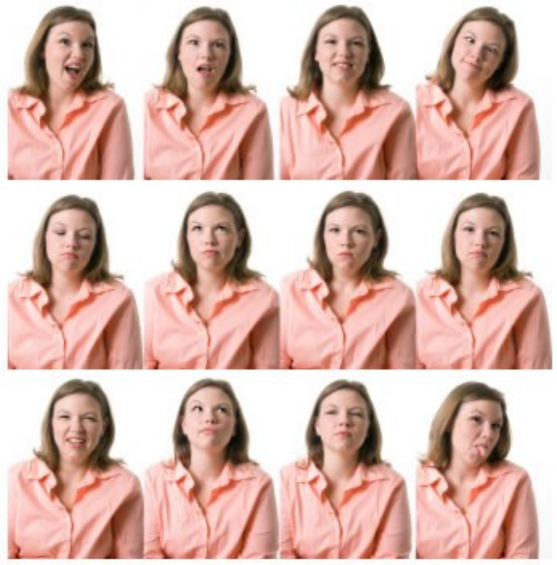
- Azimuth-Elevation manifold. Ignores lighting. [Hadsell et al. CVPR 2006]

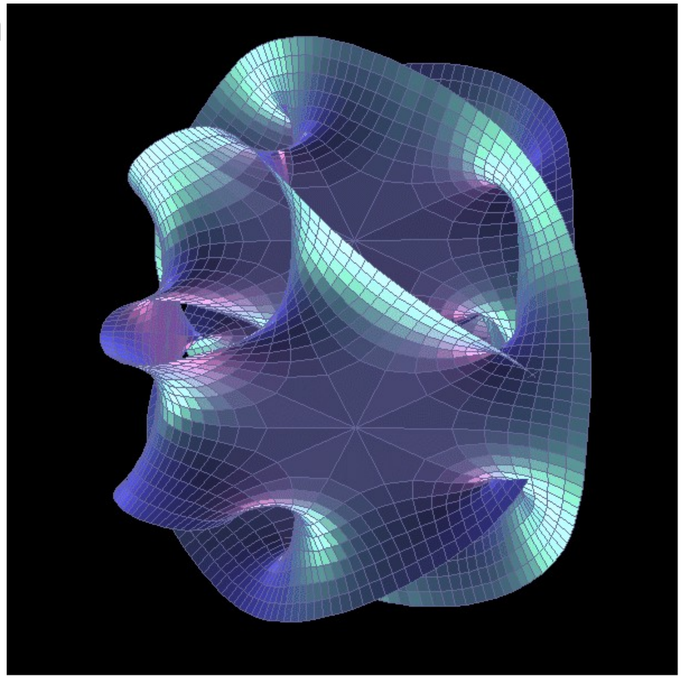


Discovering the Hidden Structure in High-Dimensional Data
The manifold hypothesis

Y LeCun
MA Ranzato

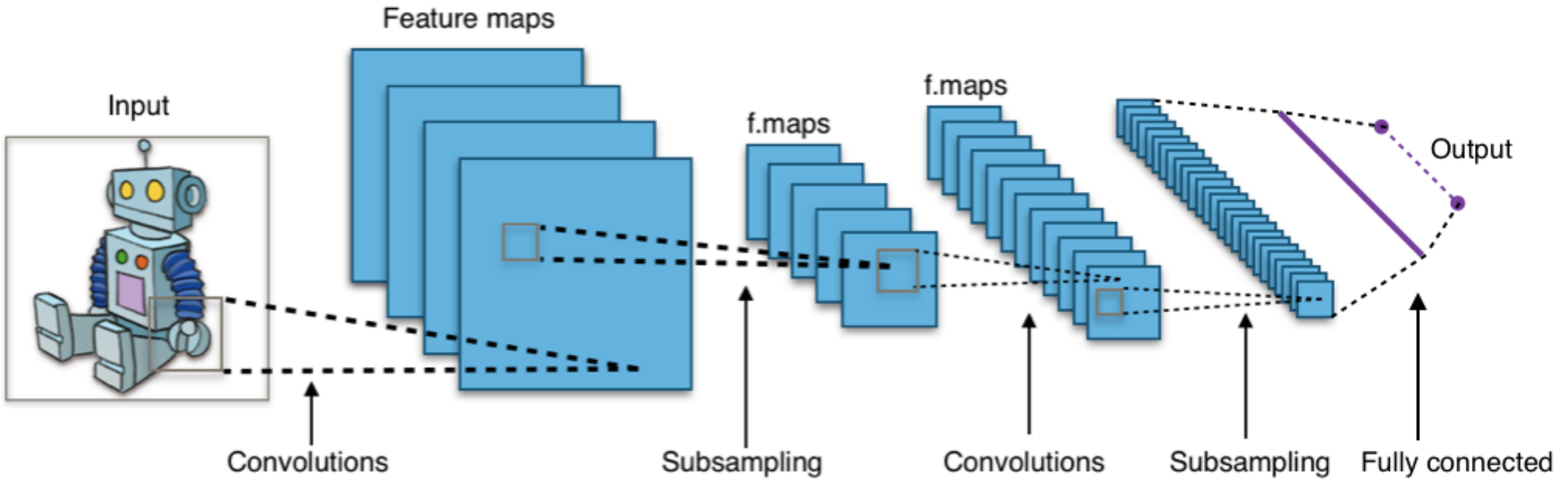
- Learning Representations of Data:
  - **Discovering & disentangling the independent explanatory factors**
- The Manifold Hypothesis:
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data

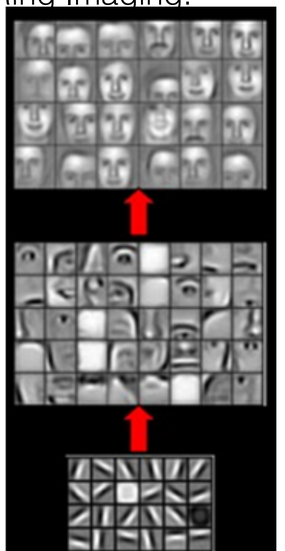




Feature Learning

- ***Feature Engineering***: e.g. Event Reconstruction ~ Feature Extraction, Pattern Recognition, Fitting, …
- Deep Neutral Networks can ***Learn Features*** from ***raw data***.
- Example: ***Convolutional Neural Networks*** - Inspired by visual cortex
  - ***Input***: Raw data… for example 1D = Audio, 2D = Images, 3D = Video
  - ***Convolutions*** ~ learned feature detectors
  - ***Feature Maps***
  - ***Pooling*** - dimension reduction / invariance
  - ***Stack***: Deeper layers recognize higher level concepts.
- Over the past few years, CNNs have lead to ***exponential improvement / superhuman performance on Image classification*** challenges. Current best > 150 layers.
- Obvious HEP application: ***"Imaging" Detectors*** such as TPCs, High Granularity Calorimeters, or Cherenkov Ring Imaging.

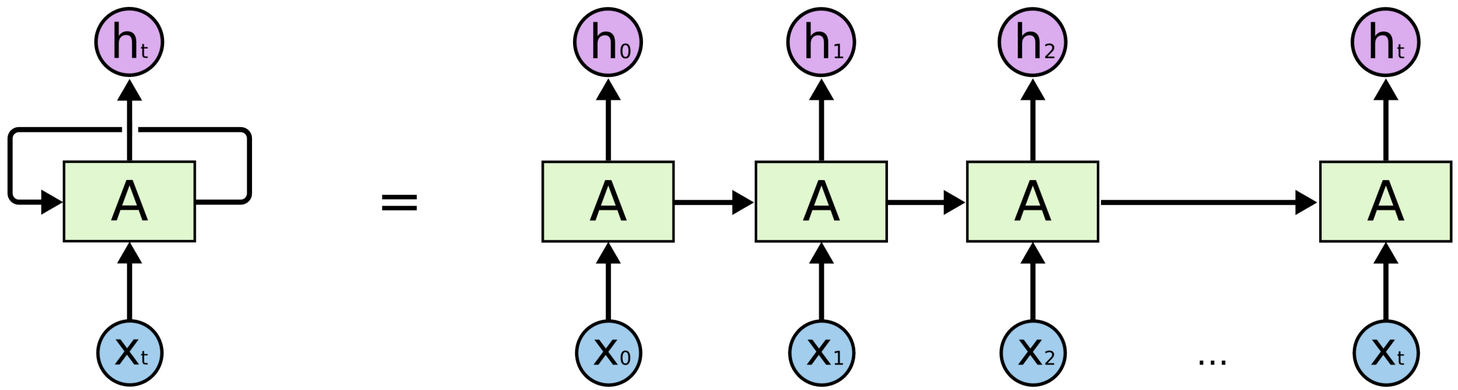




Recurrent DNNs

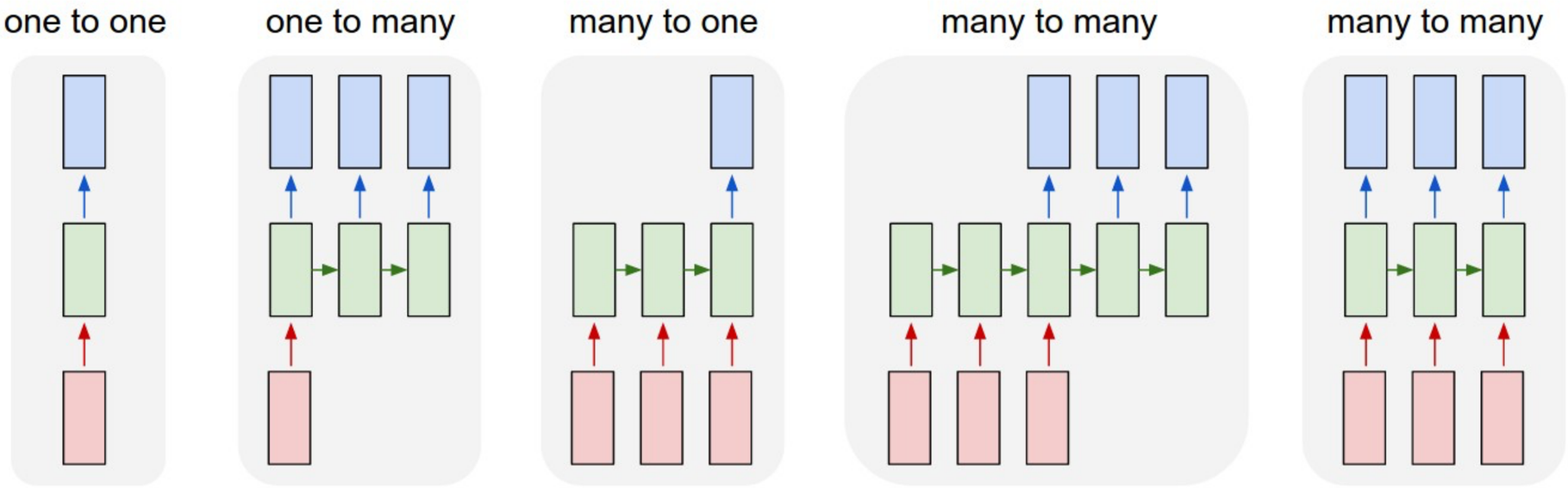
Motivation

- DNN inputs:
  - Fixed size: images, video, raw data from detector…
  - Variable size: audio, text, particles in event… sequences.
- Usual NNs map input to output.
  - No memory of previous information.
- Recurrent NN: feed some output back into self.
- DNNs can represent arbitrary functions… RNNs can represent arbitrary programs.



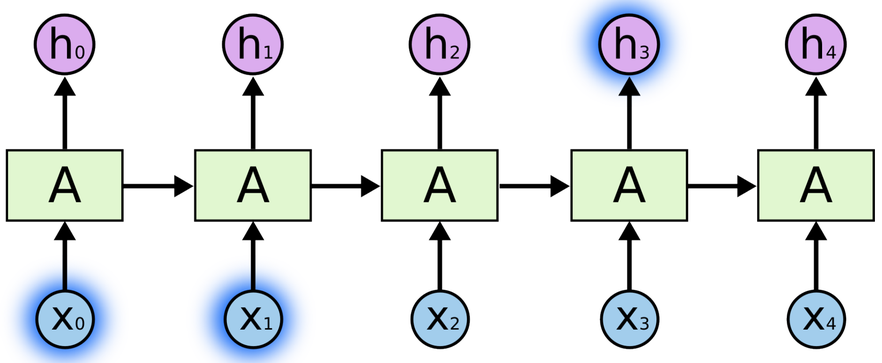
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN input/output



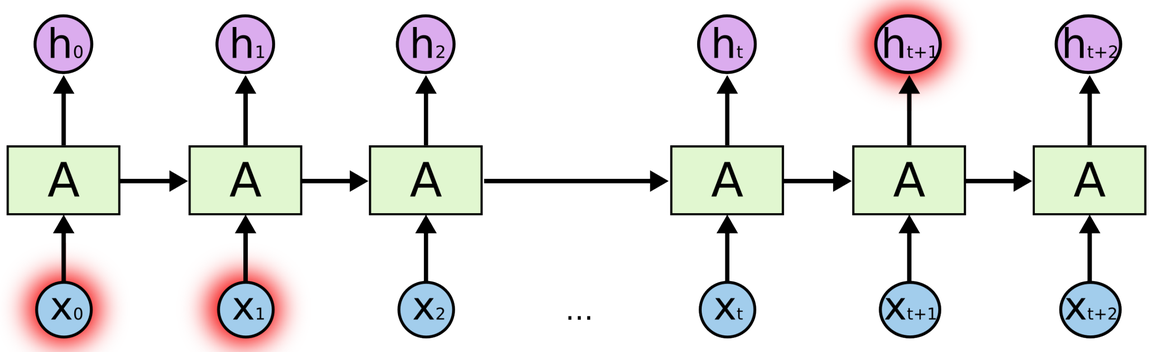
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

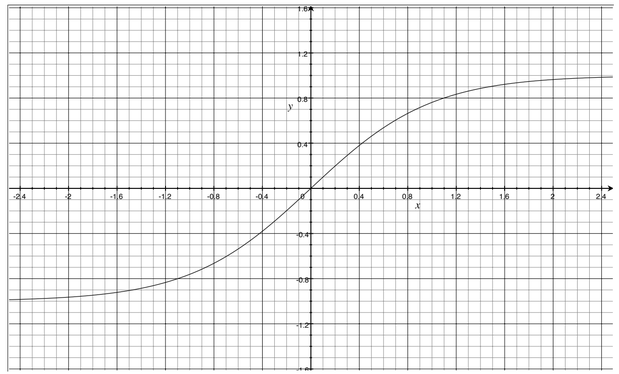
Basic RNN

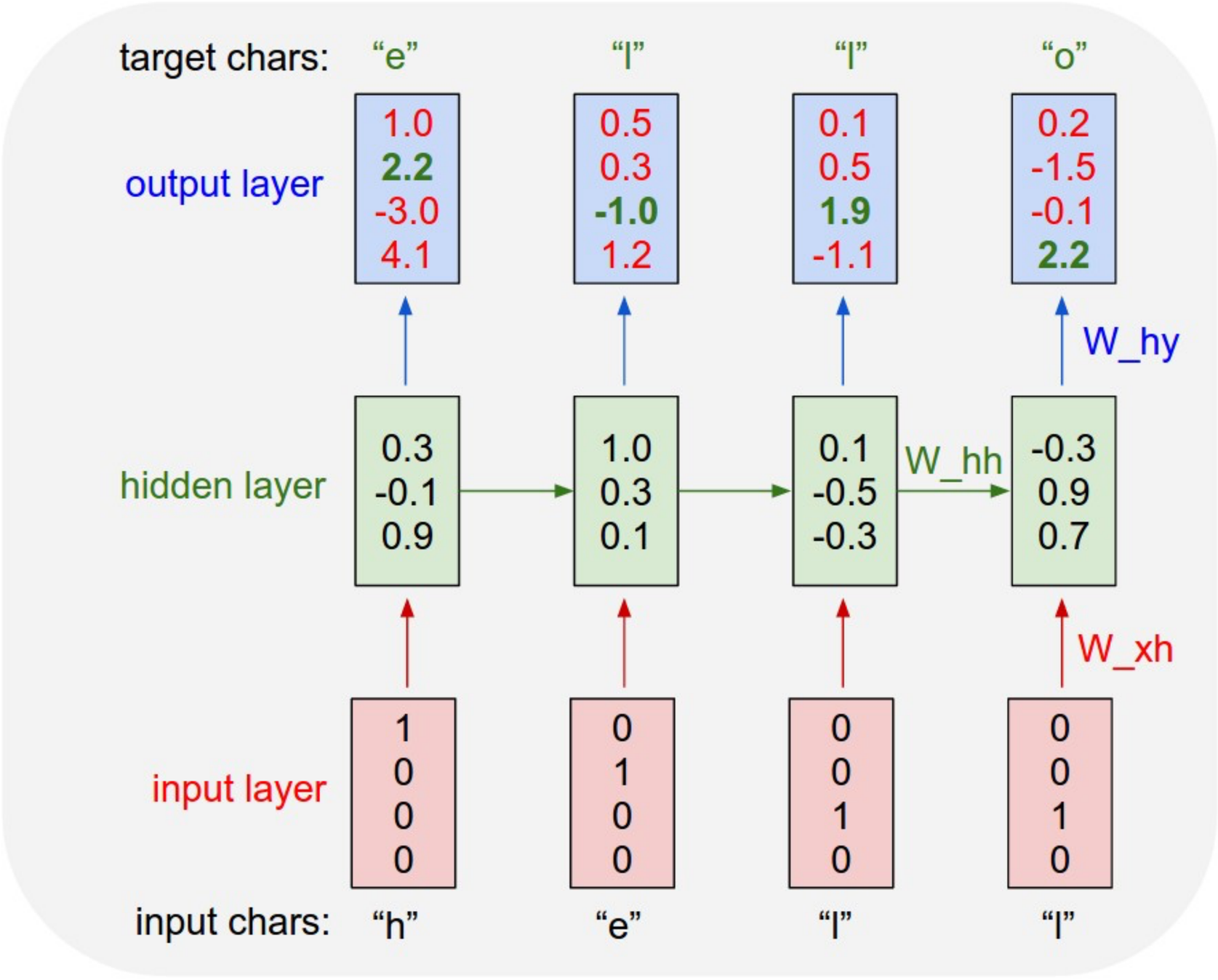


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

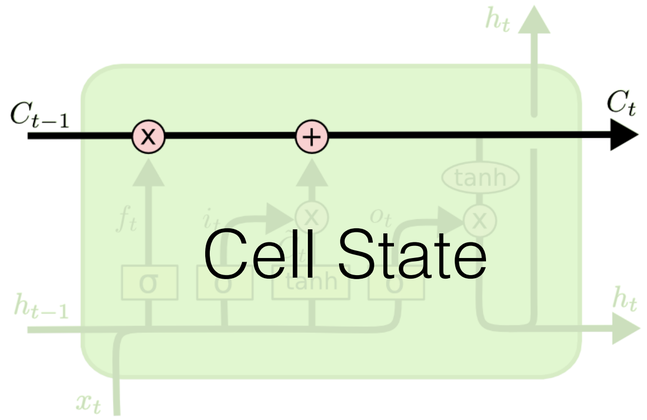
$$y = W_{hy} \cdot h_t$$



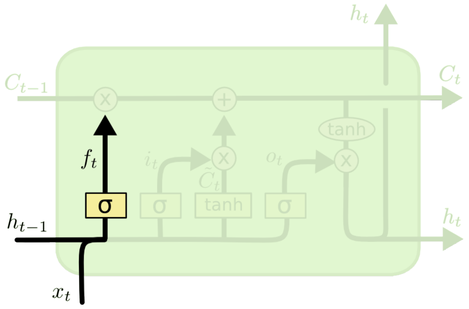



target chars:
“e”
“l”
“l”
“o”
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
“h”
“e”
“l”
“l”

LSTM

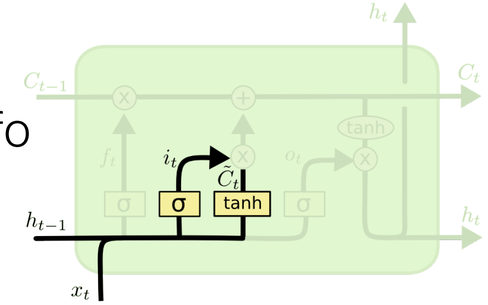


Forget



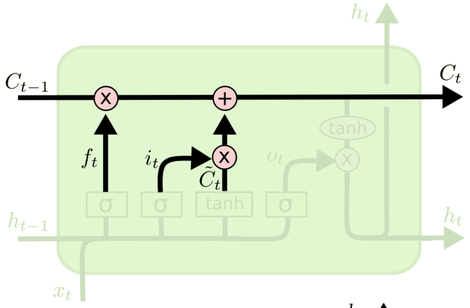
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

New info



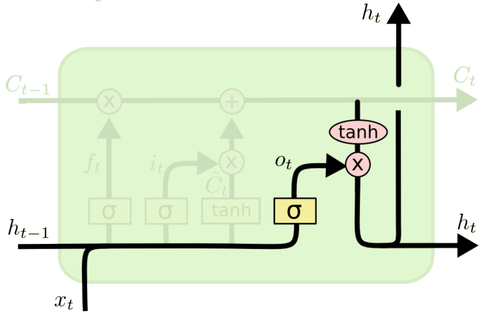
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

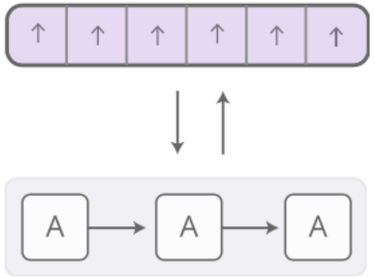
Construct output



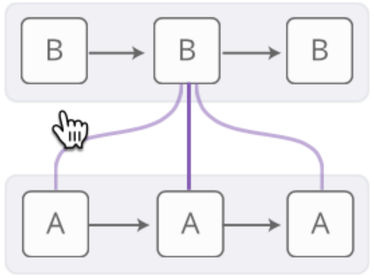
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

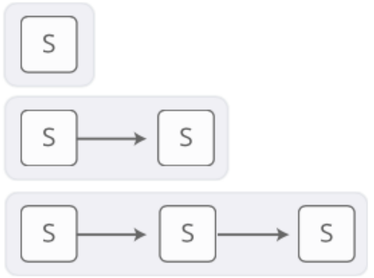
Other RNNs



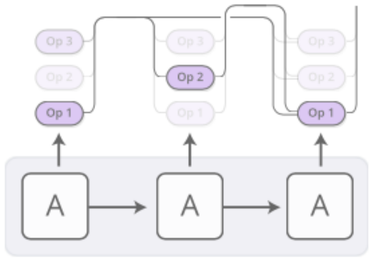
Neural Turing Machines
have external memory that they can read and write to.



Attentional Interfaces
allow RNNs to focus on parts of their input.



Adaptive Computation Time
allows for varying amounts of computation per step.



Neural Programmers
can call functions, building programs as they run.

Generative Models

Generative Models

- DNNs Generative Models enable building simulations purely from examples.
  - ***Generative Adversarial Nets*** (Goodfellow, et. al. arxiv:1406.2661). Simultaneously train 2 Networks:
    - ***Discriminator*** (D) that tries to distinguish output and real examples.
    - ***Generator*** (G) that generate the output that is difficult to distinguish.
  - ***Variational Auto-encoders***:
    - Learn a ***latent variable probabilistic model*** of the input dataset.
    - ***Sample latent space*** and use ***decoder to generate data***.

Auto-Encoding Variational Bayes

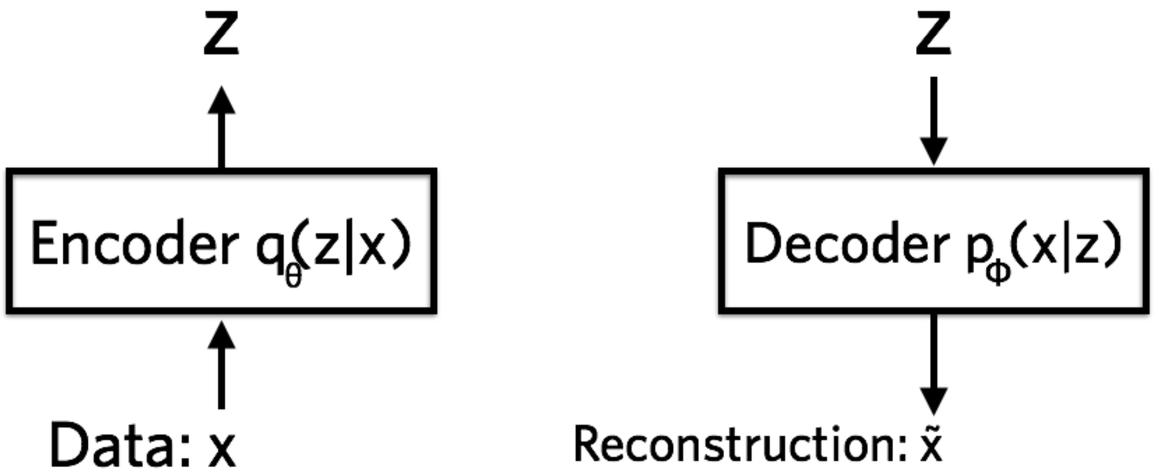
Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions is two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

https://arxiv.org/abs/1312.6114



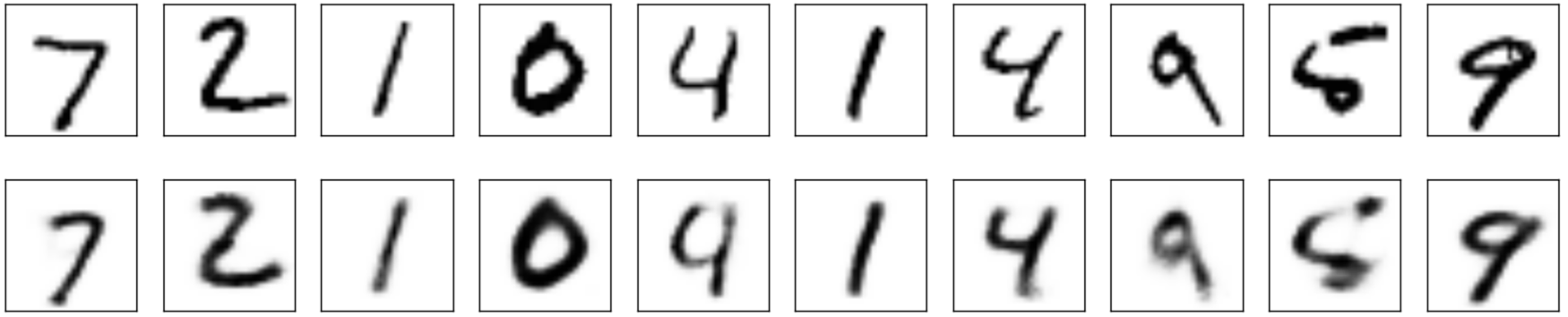
- a probabilistic encoder $q_\phi(z|x)$, approximating the true (but intractable) posterior distribution $p(z|x)$, and
- a generative decoder $p_\theta(x|z)$, which notably does not rely on any particular input x.

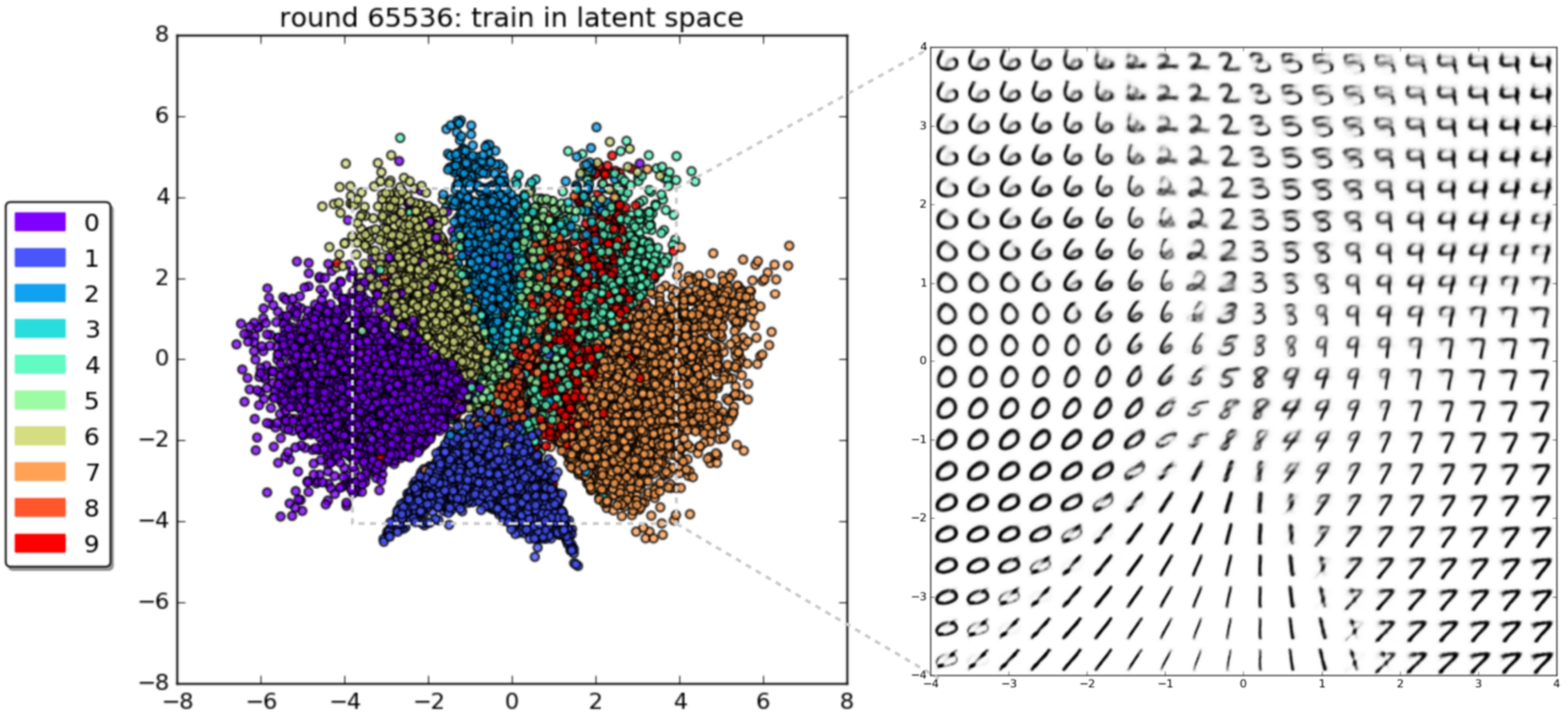
Both the encoder and decoder are artificial neural networks (i.e. hierarchical, highly nonlinear functions) with tunable parameters ϕ and θ, respectively.

Learning these conditional distributions is facilitated by enforcing a plausible mathematically-convenient prior over the latent variables, generally a standard spherical Gaussian: $z \sim \mathcal{N}(0, I)$.

$$l_i(\theta, \phi) = -E_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + KL(q_\theta(z|x_i)||p(z))$$

http://blog.fastforwardlabs.com/2016/08/12/introducing-variational-autoencoders-in-prose-and.html




round 65536: train in latent space
0
1
2
3
4
5
6
7
8
9

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie[*], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[†], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

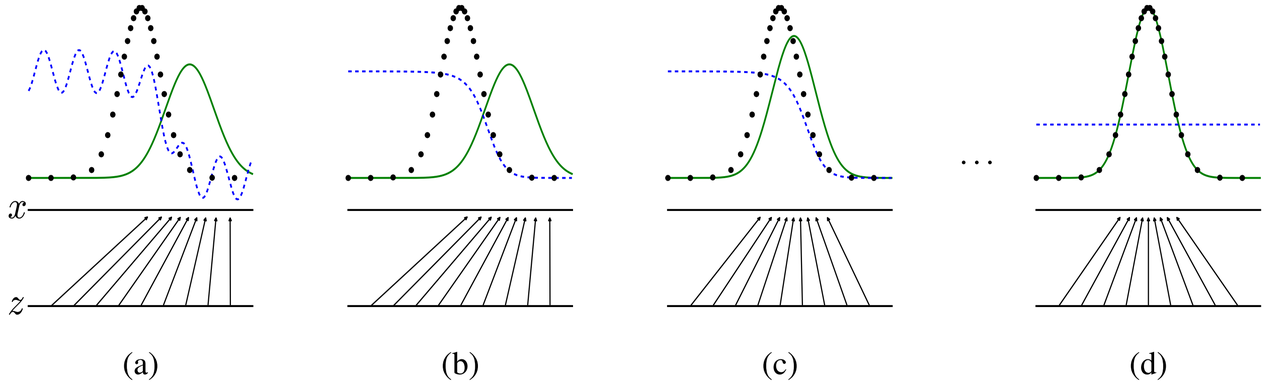


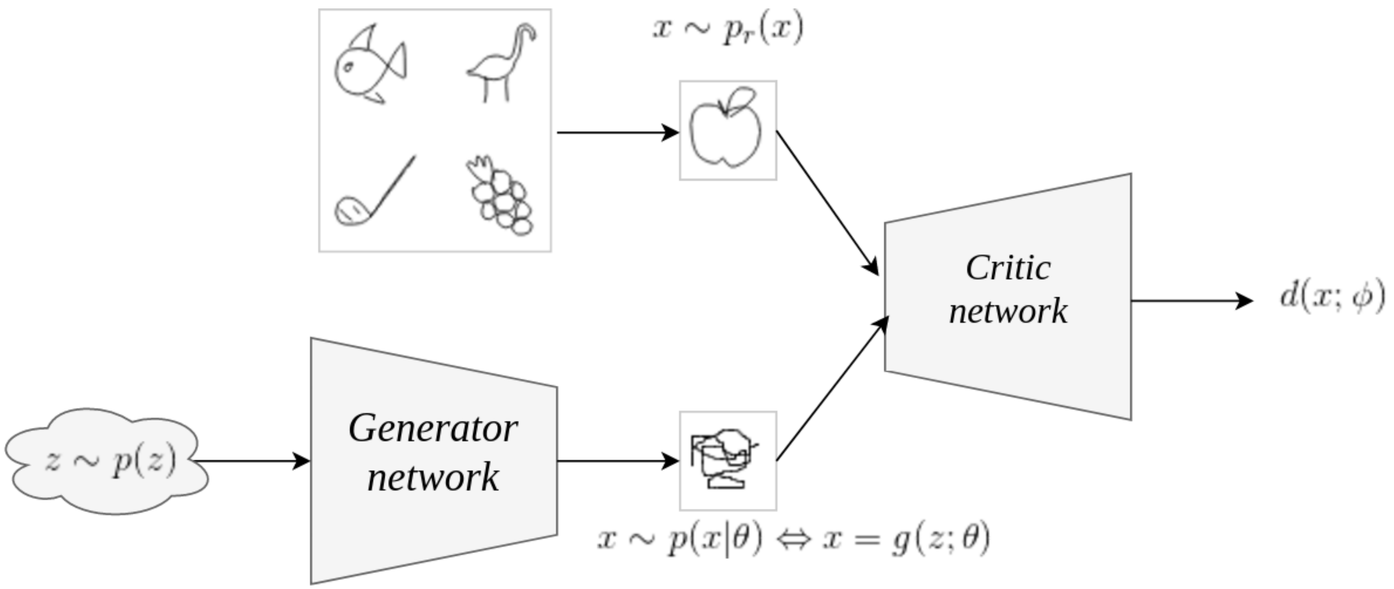
Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

https://arxiv.org/abs/1406.2661

Example: http://cs.stanford.edu/people/karpathy/gan/

Generative adversarial networks

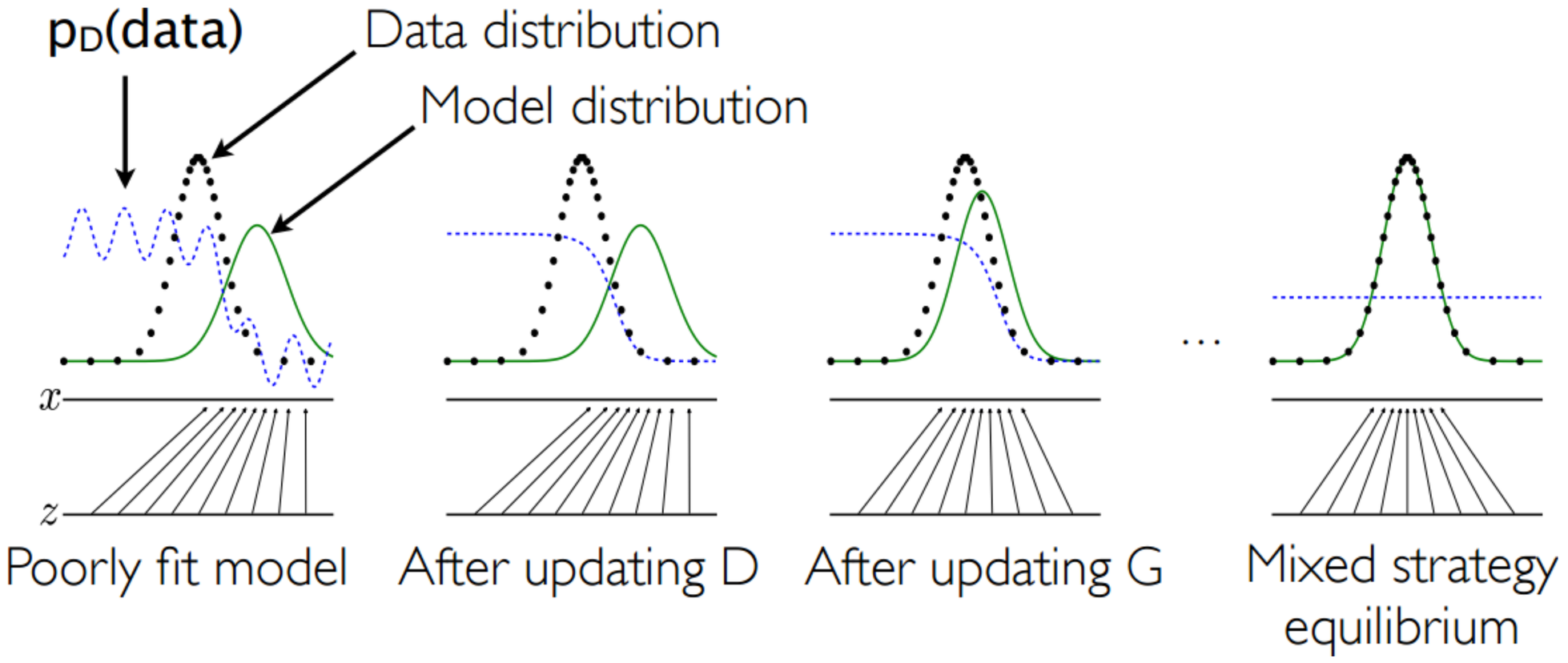
Goodfellow et al, 2014, arXiv:1406.2661
Arjovsky et al, 2017, arXiv:1701.07875



$$\mathcal{L}_d(\phi) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\theta)}[d(\mathbf{x};\phi)] - \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}[d(\mathbf{x};\phi)] + \lambda\Omega(\phi)$$

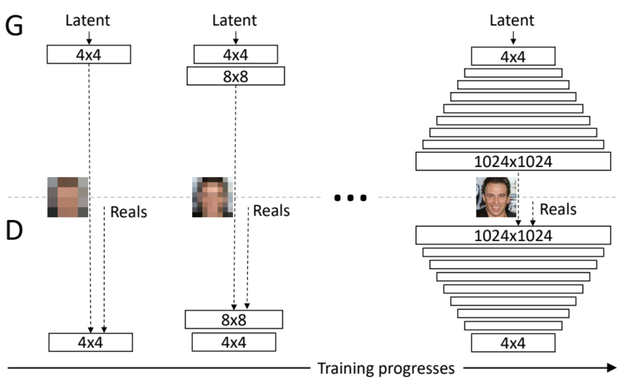
$$\mathcal{L}_g(\theta) = -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\theta)}[d(\mathbf{x};\phi)]$$

(Wasserstein GAN + Gradient Penalty)


p_D(data)
Data distribution
Model distribution
x
z
...
Poorly fit model
After updating D
After updating G
Mixed strategy equilibrium

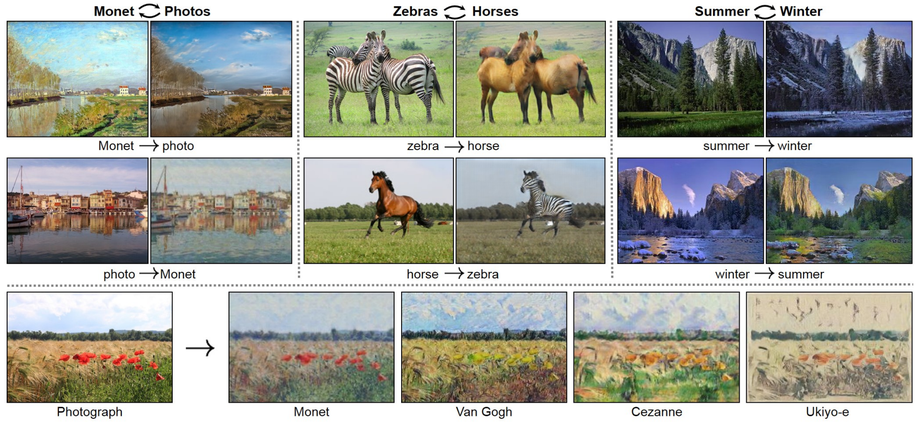
State-of-the-art

Karras et al, 2017, arXiv:1710.10196



Style transfer

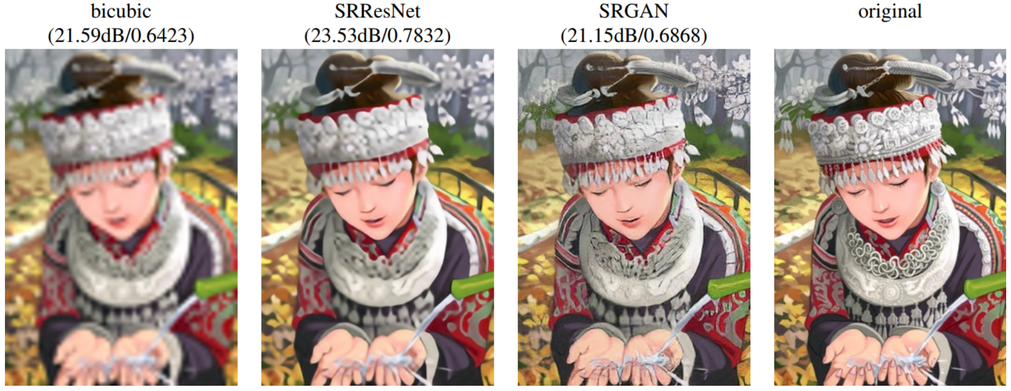
Zhu et al, 2017, arXiv:1703.10593



From simulated data to realistic data?

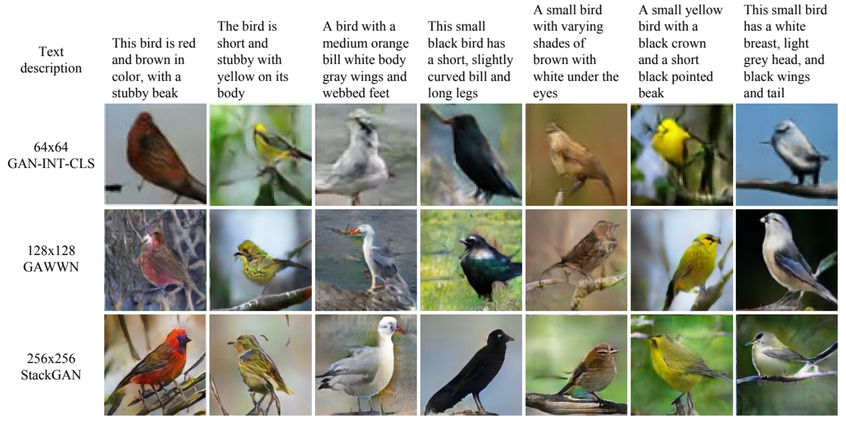
Super-resolution

Ledig et al, 2016, arXiv:1609.04802



Text-to-image synthesis

Zhang et al, 2016, arXiv:1612.03242



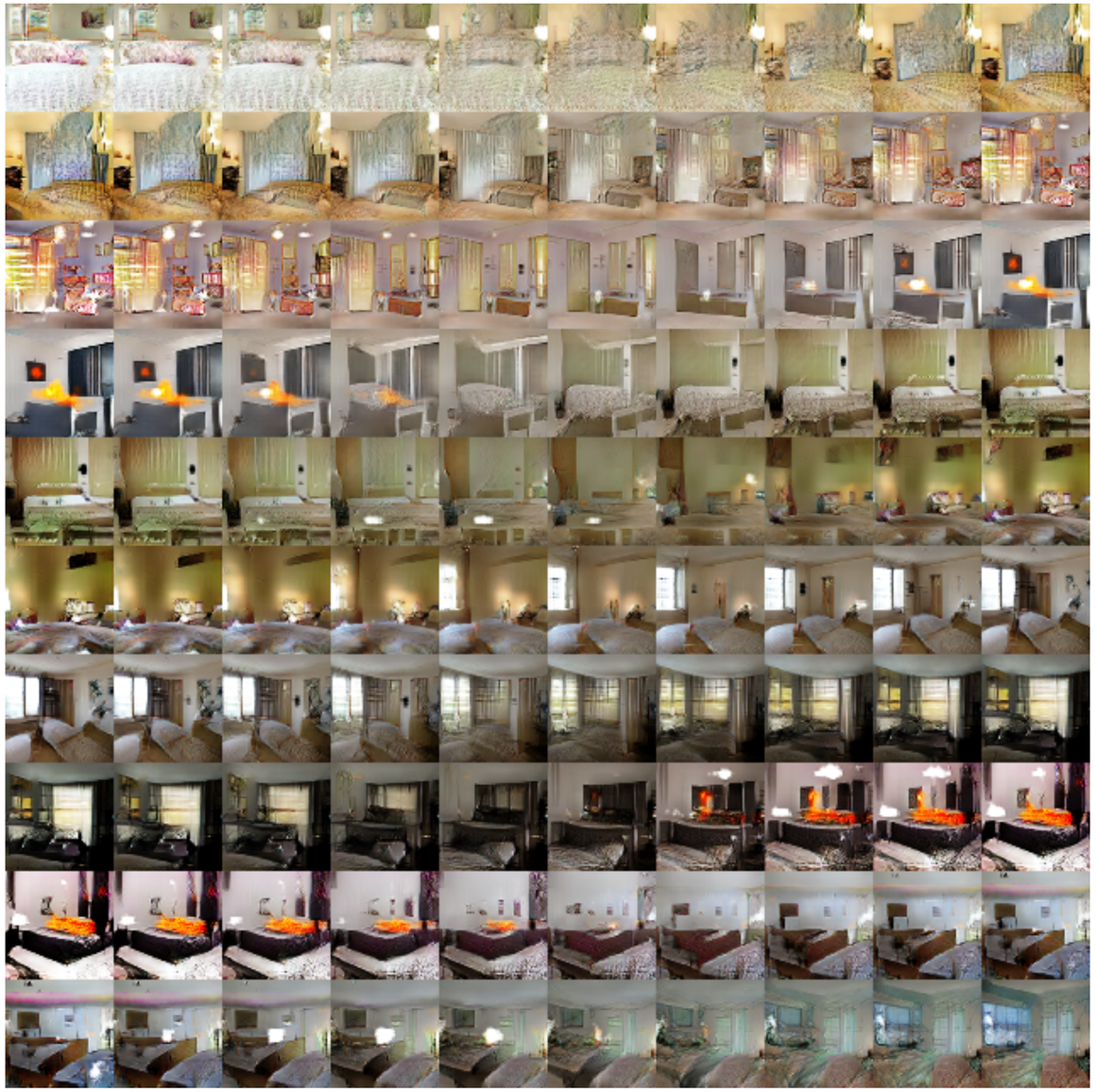


Figure 4: Top rows: Interpolation between a series of 9 random points in Z show that the space learned has smooth transitions, with every image in the space plausibly looking like a bedroom. In the 6th row, you see a room without a window slowly transforming into a room with a giant window. In the 10th row, you see what appears to be a TV slowly being transformed into a window.

https://arxiv.org/pdf/1511.06434v2.pdf

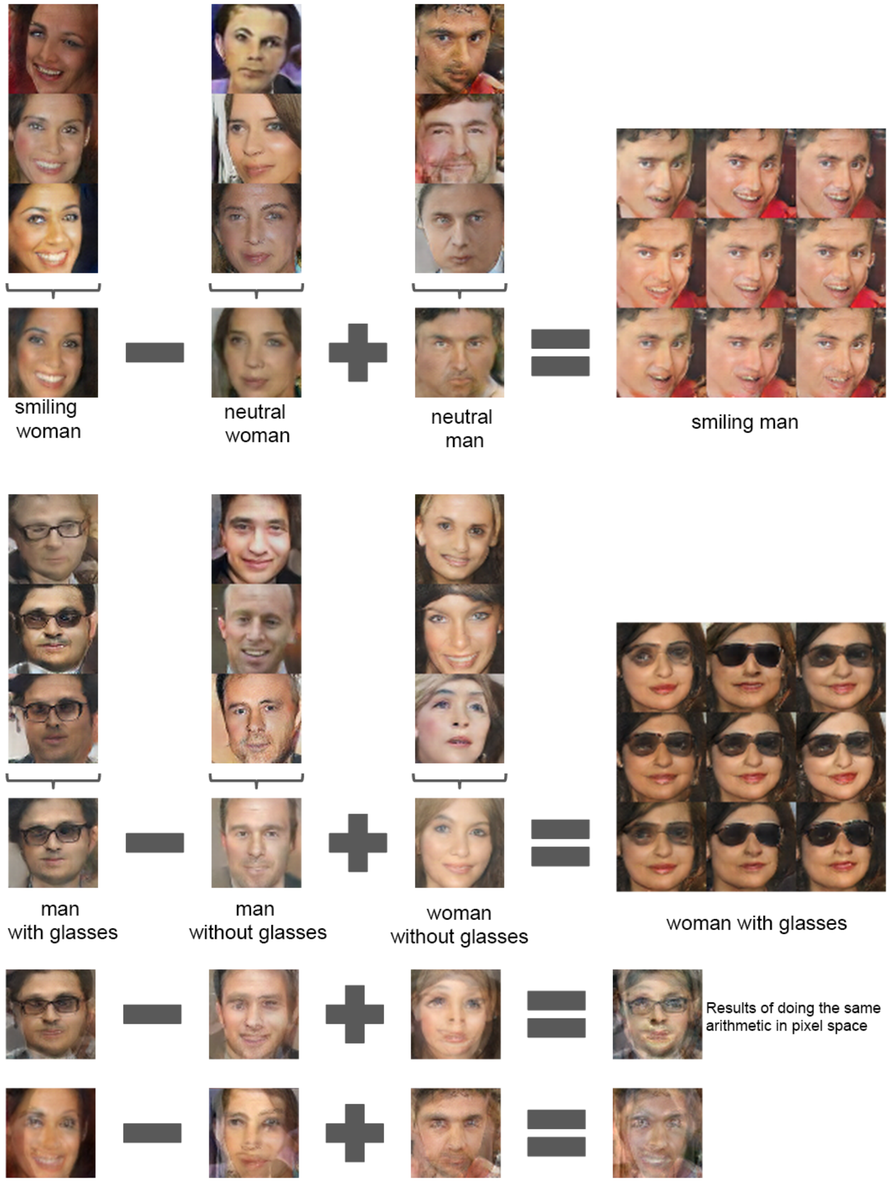


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples) results in noisy overlap due to misalignment.

Figure 8: A ”turn” vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

(a) Azimuth (pose)

(b) Elevation

(c) Lighting

(d) Wide or Narrow

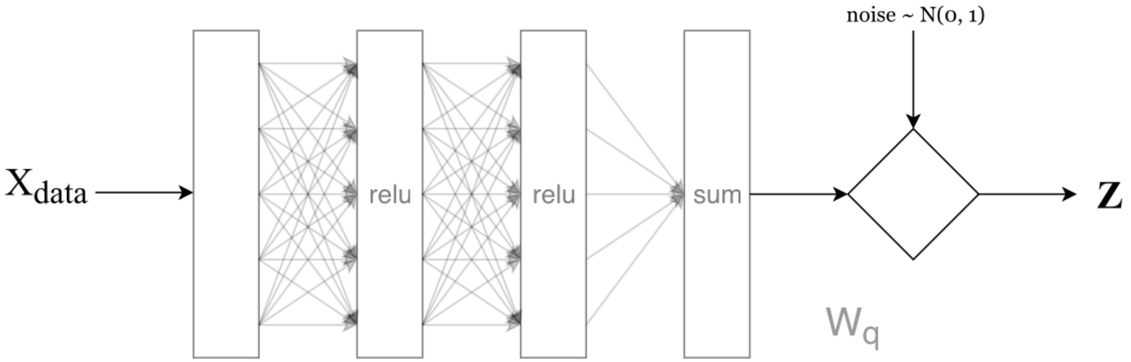
Figure 3: **Manipulating latent codes on 3D Faces:** We show the effect of the learned continuous latent factors on the outputs as their values vary from -1 to 1. In (a), we show that one of the continuous latent codes consistently captures the azimuth of the face across different shapes; in (b), the continuous code captures elevation; in (c), the continuous code captures the orientation of lighting; and finally in (d), the continuous code learns to interpolate between wide and narrow faces while preserving other visual features. For each factor, we present the representation that most resembles prior supervised results [7] out of 5 random runs to provide direct comparison.

https://arxiv.org/pdf/1606.03657.pdf

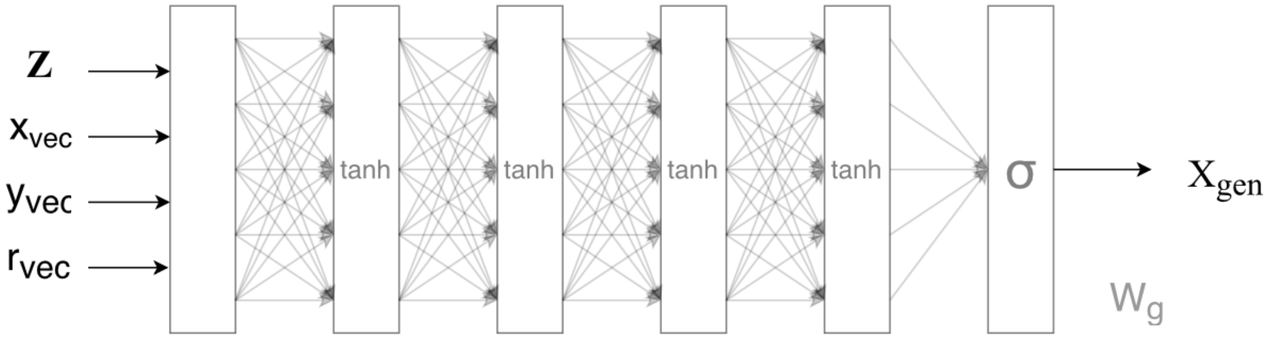
VAE vs GAN

- GAN
  - Generates from noise… can’t select what you generate.
  - Adversarial network only discriminates… results in current “style” but no object.
- VAE
  - Cost function instead of adversarial network… results in blurry images.

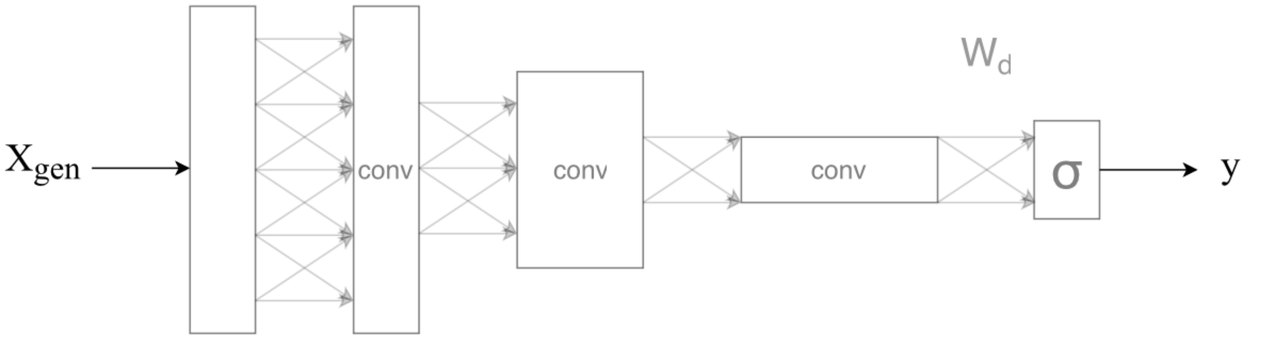
Variational Encoder Network



Generator Network



Discriminator Network



http://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/

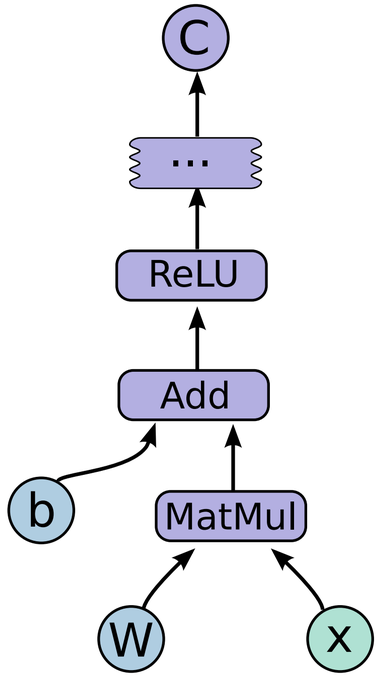
DL Software and Technical Challenges

numpy, TensorFlow, Keras

- Numpy
  - Provides a tensor representation.
  - It's interface has been adopted by everyone.
    - e.g. HDF5, Then, TensorFlow, … all have their own tensors.
    - You can use other tensors, for the most part interchangeably with numpy.
  - Provides extensive library of tensor operations.
    - D = A * B + C, immediately computes the product of A and B matrices, and then computes the sum with C.
- TensorFlow / PyTorch
  - Allows you write tensor expressions symbolically.
    - A * B + C is an expression.
  - Compiles the expression into fast executing code on CPU/GPU: F(A,B,C)
  - You apply the Compiled function to data get at a result.
    - D=F(A,B,C)
- Keras
  - Neutral Networks can be written as a Tensor mathematical expression.
  - Keras writes the expression for you.

DNN Software

- Basic steps
  - Prepare data
  - Build Model
  - Define Cost/Loss Function
  - Run training (most commonly Gradient Decent)
  - Assess performance.
  - Run lots of experiments…
- 2 Classes of DNN Software: (Both build everything at runtime)
  - Hep-Framework-Like: e.g. Torch, Caffe, …
    - C++ Layers (i.e. Algorithms) steered/configured via interpreted script:
  - General Computation Frameworks: Theano and TensorFlow
    - Everything build by building mathematical expression for Model, Loss, Training from primitive ops on Tensors
    - Symbolic derivatives for the Gradient Decent
    - Builds Directed Acyclic Graph of the computation, performs optimizations
    - Theano-based High-level tools make this look like HEP Frameworks (e.g. pylearn2, Lasagna, Keras, …)



Technical Challenges

- Datasets are too large to fit in memory.
- Data comes as many h5 files, each containing O(1000) events, organized into directories by particle type.
- For training, data needs to be read, mixed, “labeled”, possibly augmented, and normalized…. can be time consuming.
- Very difficult to keep the GPU fed with data. GPU utilization often < 10%, rarely > 50%.
- Keras python multi-process generator mechanism has limitations…

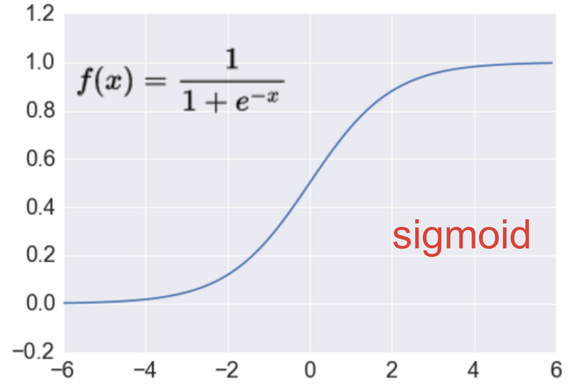
DL Model Components

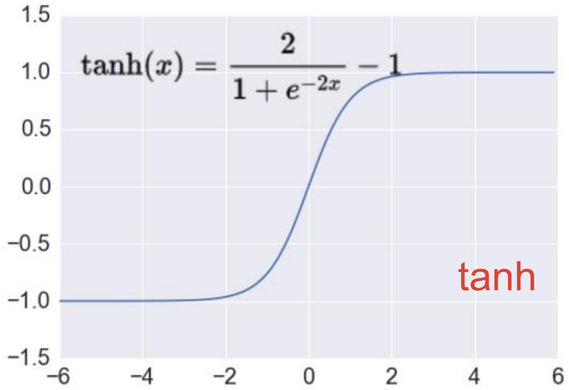
Keras

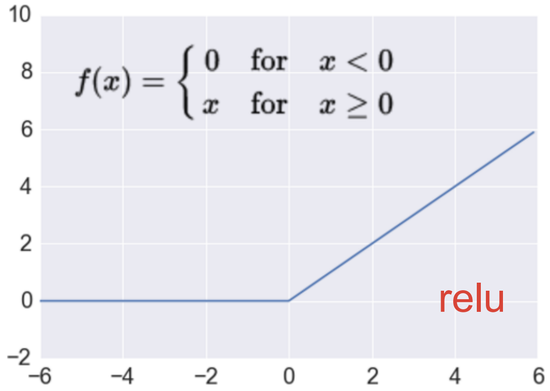
https://keras.io/

Activations

- Vanishing Gradients
- Sigmoid
  - Saturate
  - non-Zero centered
- Relu
  - Larger Gradient
  - Simple to compute
  - If learning rate too high, neurons can “Die” (never activate)
- Leaky Relu





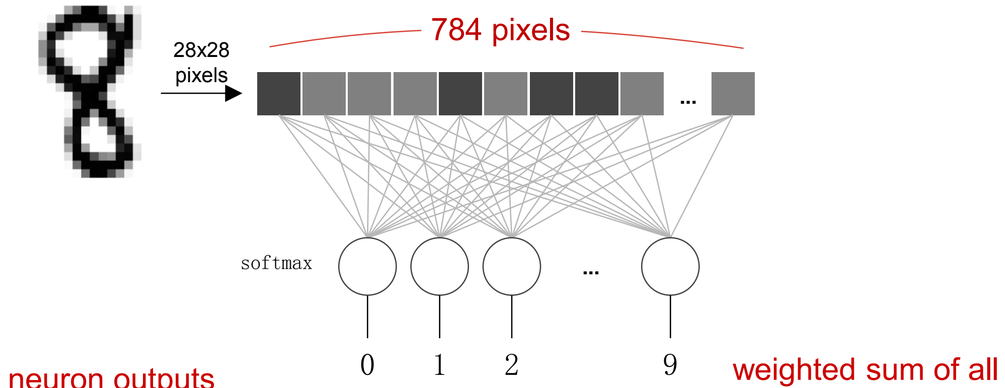


Output

- Classification: One-hot representation
- SoftMax
  - Boltzmann distribution: $e^{-E/kT}$
  - Takes vector of arbitrary values and maps to
    - vector with values in range 0 to 1
    - vector components sum to 1.
  - Uses:
    - Prob output from Multiclass classification
    - Normalization of data.

The softmax layer

➢ **The output from the softmax layer is a set of probability distribution, positive numbers which sum up to 1.**

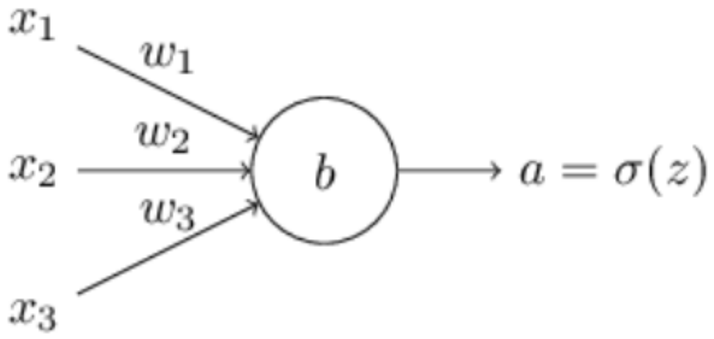


$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Cost/Loss

- MSE- Means square error.
  - Proven to give right y for given x.
- MAE-
  - Proven to give right median y for given x.
- Often train poorly: saturating outputs give small gradients.
  - Use cross-entropy for classification

Cost Functions



$$z = \sum_j w_j x_j + b$$

$$C = \frac{(y-a)^2}{2}$$

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$

$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

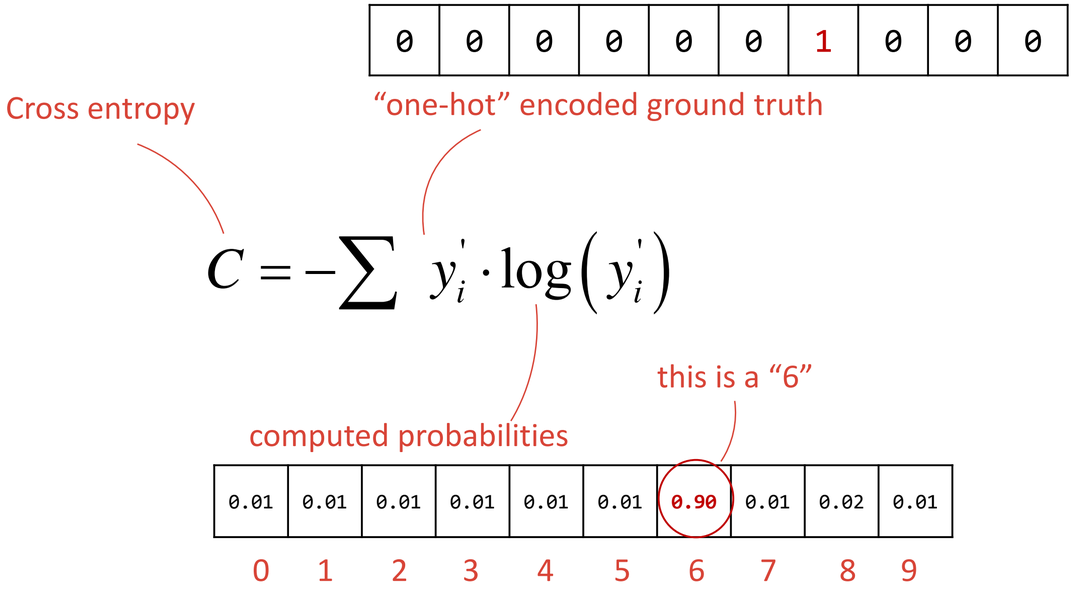
$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z)-y).$$

$$C = -\frac{1}{n}\sum [y \ln a + (1-y)\ln(1-a)]$$

$$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_x (\sigma(z)-y).$$

The Cross-Entropy Cost Function

- **For classification problems, the Cross-Entropy cost function works better than quadratic cost function.**
- **We define the cross-entropy cost function for the neural network by:**



$$H(p,q) = -\sum_x p(x)\ \log q(x).$$

- In information theory, the cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "unnatural" probability distribution q rather than the "true" distribution p