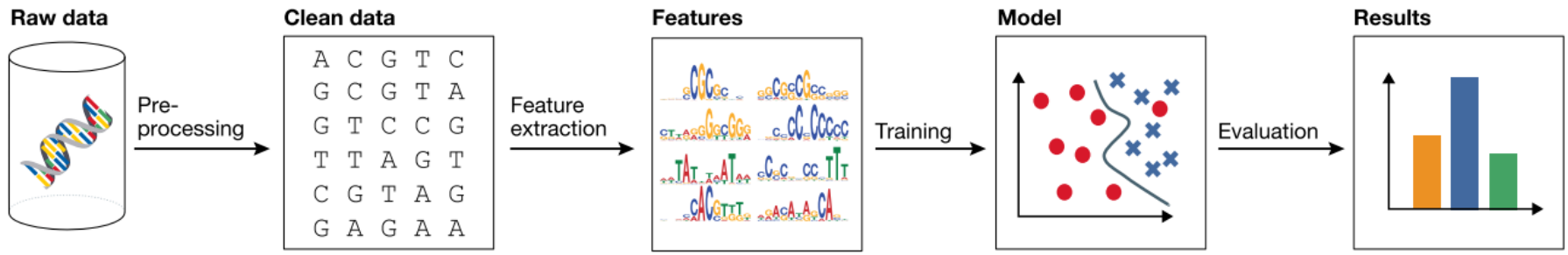
CSE549

DNN Applications to Bioinformatics

Sael Lee
Department of Computer Science,
SUNY Korea, Incheon 21985, Korea

Benefits of DNN Learning

Classical Machine Learning Pipeline in Comp Bio



Deep Learning in Comp Bio.

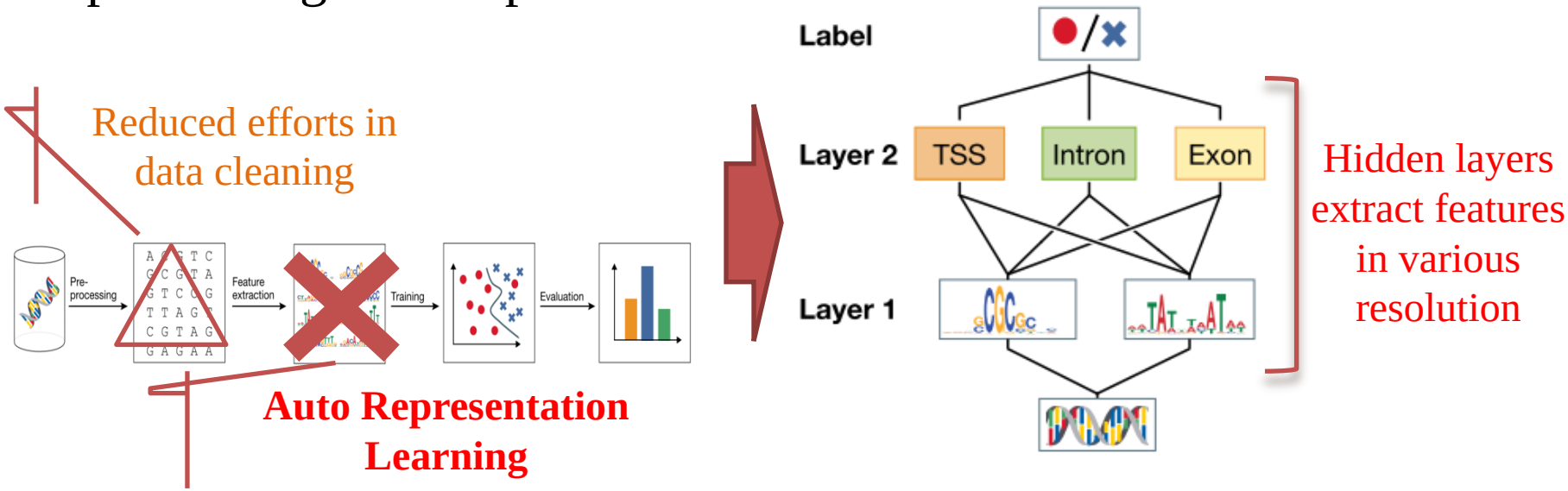


Fig 1A,D from Angermueller et al. (2016) *Molecular Systems Biology*, (12), 878.

Various Applications

- Regulatory Genomics
  - Alternative Splicing (Leung et al 2014; Xiong et al, 2015)
  - Accessible Genome Analysis (Zhous & Troyanskaya, 2015; Kelley et al, 2016)
  - Protein-Nucleic Acid Binding Prediction (Alipananhi et al, 2015)
  - Variant Analysis
- Protein Structure Prediction
  - Secondary structure Prediction
  - Order/Disorder Region Prediction
  - Residue-Residue Contact Prediction
- Applications on High throughput Data
  - QSAR Prediction
  - Circadian Rhythms
- Other Topics Not Covered
  - Cellular Image Analysis
  - Medical Time Series Data

Early works of DNN in Alternative Splicing

Deep Feedforward NN

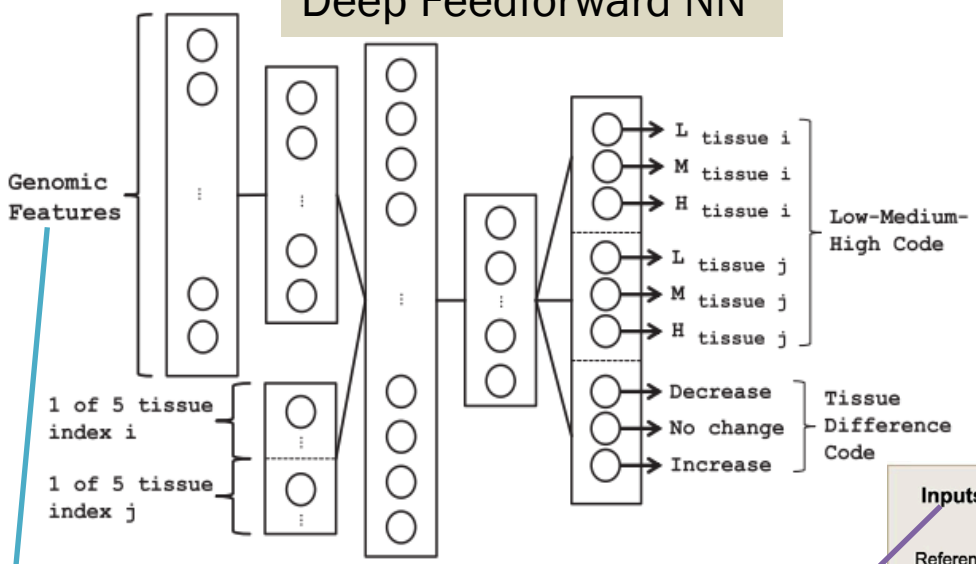


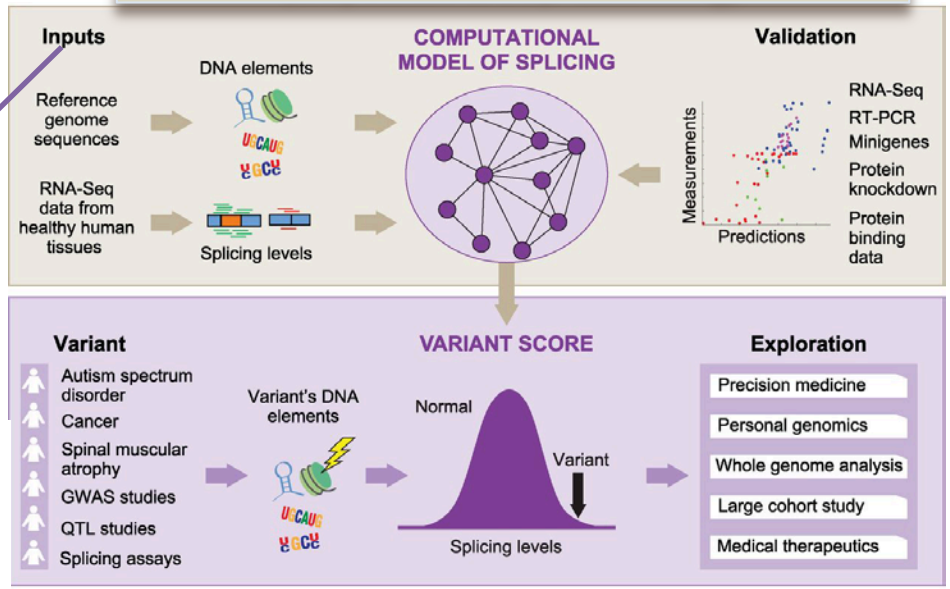
Fig 1 of Leung et al. (2014) Bioinformatics 30(12) 121-129

"Deep learning of the tissue-regulated splicing code"

1393 features extracted from each exon of 5 different tissue types

Early works still utilize **selected (large size) features**

Fully connected Feedforward NN (Bayesian Deep Learning)



1000 predetermined features from candidate exon and adjacent introns

Fig 1 of Xiong et al. (2015) Science 347(6218):1254806

Feature listing

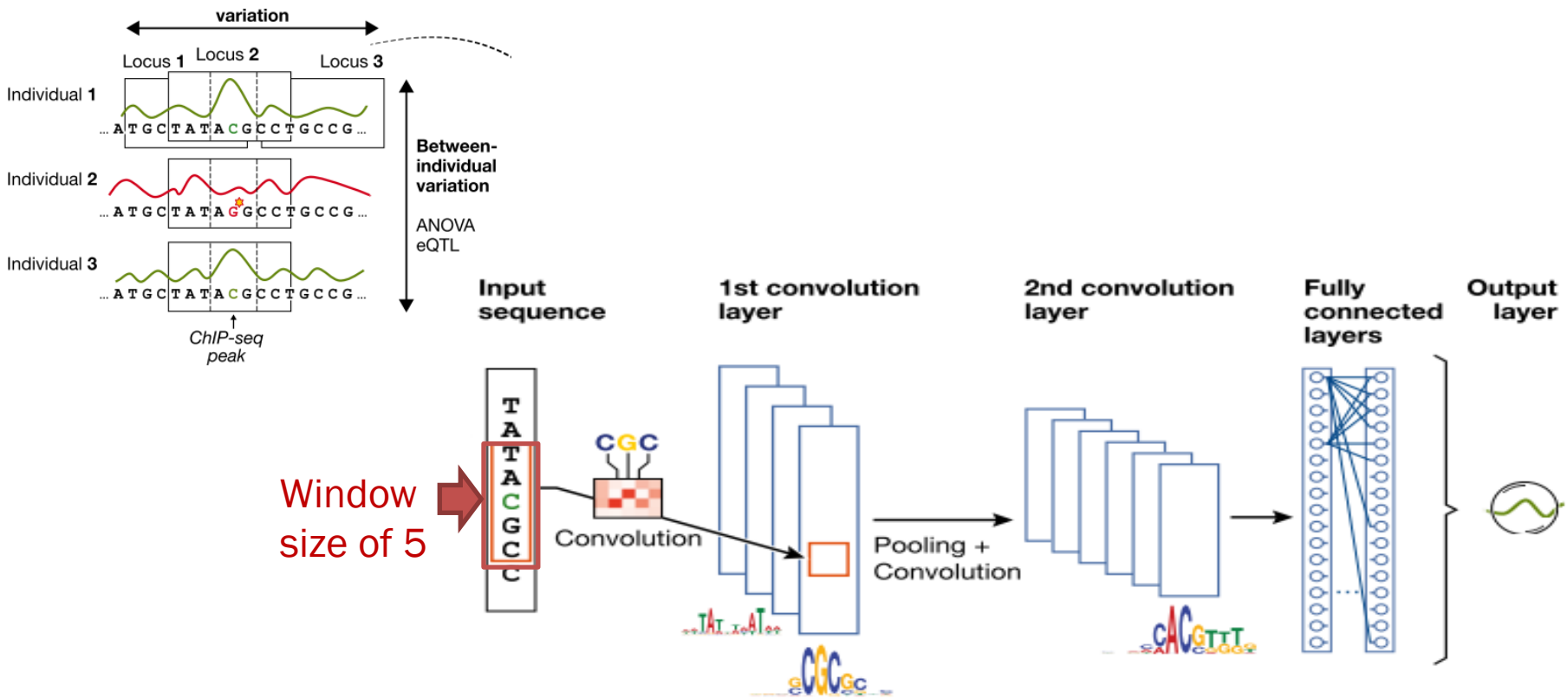
Leung et al. (2014)
Bioinformatics 30(12)
121-129

Group #	Name	Description	Type	# of Features
01	short-seq-1mer	Frequency of nucleotide patterns of different lengths (1 to 3).	real (0-1)	28
02	short-seq-2mer			112
03	short-seq-3mer			320
04	translatable-C1	Describes whether exons can be translated without a stop codon in one of three possible reading frames. For example, C1A means the exons of interest are C1 + A.	binary	1
05	translatable-C1A			1
06	translatable-C1AC2			1
07	translatable-C1C2			1
08	mean-con-score-AI2	Mean conservation score.	real (0-1)	1
09	mean-con-score-I1A			1
10	mean-con-score-I2C2			1
11	mean-con-score-C1I1			1
12	log-length	Log base 10 lengths of exons.	real	5
13	log-length-ratio	Log base 10 length ratios of exons.	real	3
14	acceptor-site-strength	Strength of acceptor and donor sites.	real	2
15	donor-site-strength			2
16	frameshift-exonA	Whether exon A introduces frame shift.	binary	1
17	rna-sec-struct	RNA secondary structures.	real (0-1)	32
18	5mer-motif-down	Counts of motif clusters of different lengths (5 to 7) weighted by conservation upstream and downstream from alternative exon.	real	54
19	6mer-motif-down			76
20	7mer-motif-down			28
21	5mer-motif-up			49
22	6mer-motif-up			78
23	7mer-motif-up			29
24	ese-ess-A	Counts of exonic splicing enhancers and silencers.	real	4
25	ese-ess-C1			4
26	ese-ess-C2			4
27	pssm-SC35	PSSM scores of SC35 splicing regulator protein.	real	5
28	pssm-ASF-SF2	PSSM scores of ASF/SF2 splicing regulator protein.		5
29	pssm-SRp40	PSSM scores of SRp40 splicing regulator protein.		10
30	nucleosome-position	Nucleosome positioning.	real	4
31	PTB	Phosphotyrosine-binding domain.	real	50
32	Nova-counts	Counts of Nova motif.	integer	27
33	Nova-cluster	Nova cluster score.	real	8
34	T-rich	Counts of motif with and without weighting by conservation.	real	24
35	G-rich			8
36	UG-rich			16
37	GU-rich			32
38	Fox	Counts of motif with and without weighting by conservation.	real	24
39	Quak			8
40	SC35			22
41	SRm160			11
42	SRrp20/30/38/40/55/75			77
43	CELF-like			2
44	CUGBP			16
45	MBNL			24
46	TRA2-alpha			22
47	TRA2-beta			22
48	hnRNP-A			44
49	hnRNP-H			22
50	hnRNP-G			22
51	9G8			22
52	ASF/SF2			11
53	Sugnet			2
54	alt-AG-pos	Position of the alternative AG and GT position.	integer	2
55	Alu-AI2	Counts of ALU repeats.	integer	12

C1 and *C2* denote the flanking constitutive exons ; *A* denotes the alternative exon ; *I1* denotes the intron between *C1* and *A* ; *I2* denotes the intron between *A* and *C2*

DNA/RNA Sequence Analysis with Deep CNN

Convolution step in Deep CNN resembles traditional sequence "**windowing**" scheme

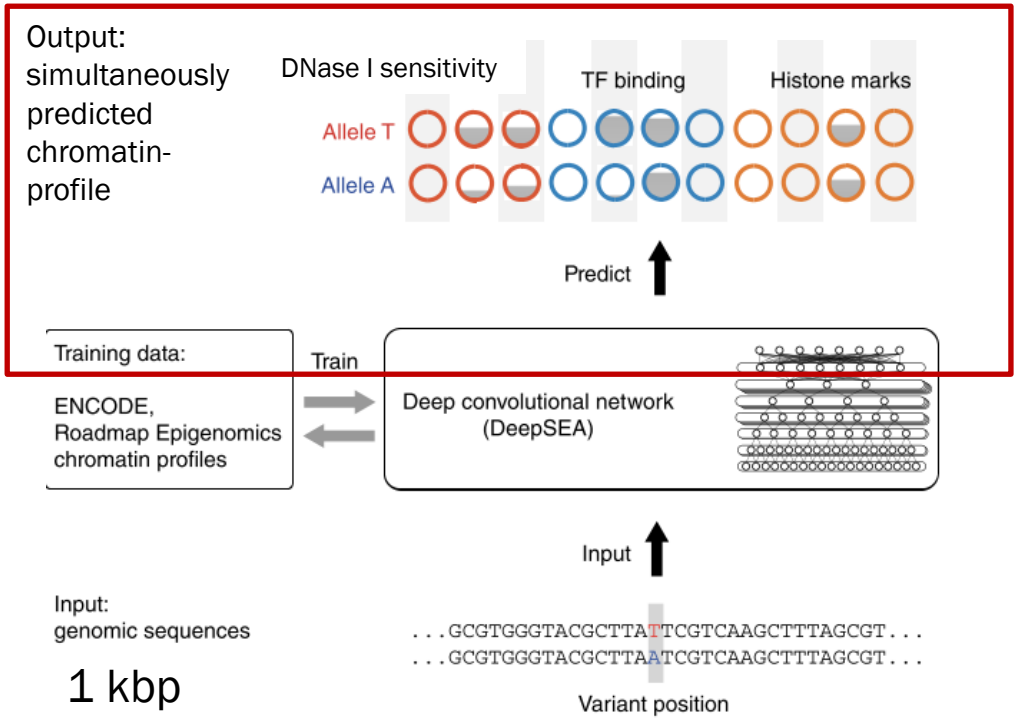


Angermueller et al. (2016) *Molecular Systems Biology*, (12), 878.

DeepSEA: CNN-based noncoding variant effect prediction

GOAL: Identifying functional effects of noncoding variants

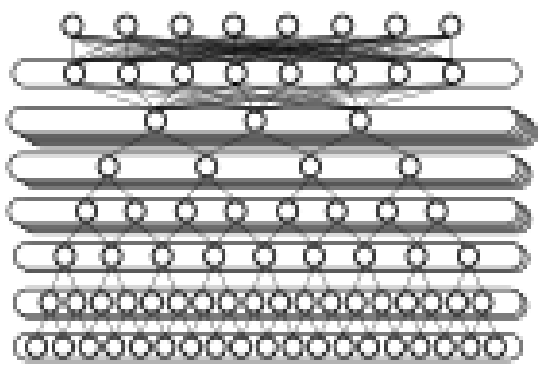
DeepSEA CNN structure



Innovative points:

1. Use long seq. 1kbp
2. **multitask architecture**

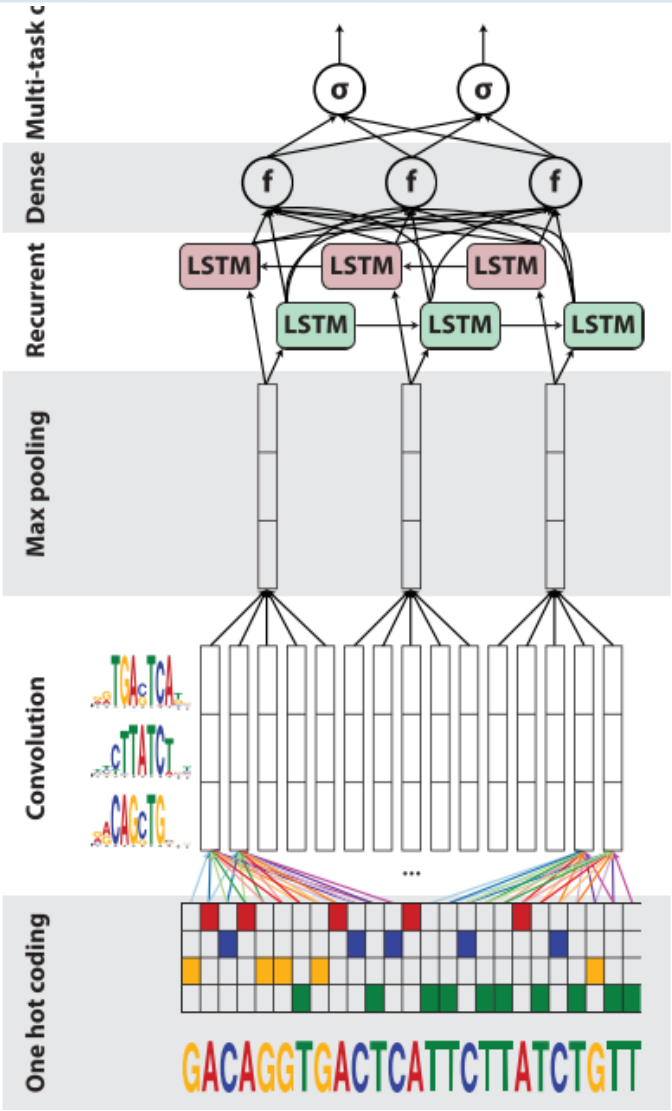
-> multiple output variables
919 chromatin features (125 DNase features, 690 TF features, 104 histone features)



Zhou, J., & Troyanskaya, O. G. (2015). *Nature Methods*, *12*(10), 931–4.

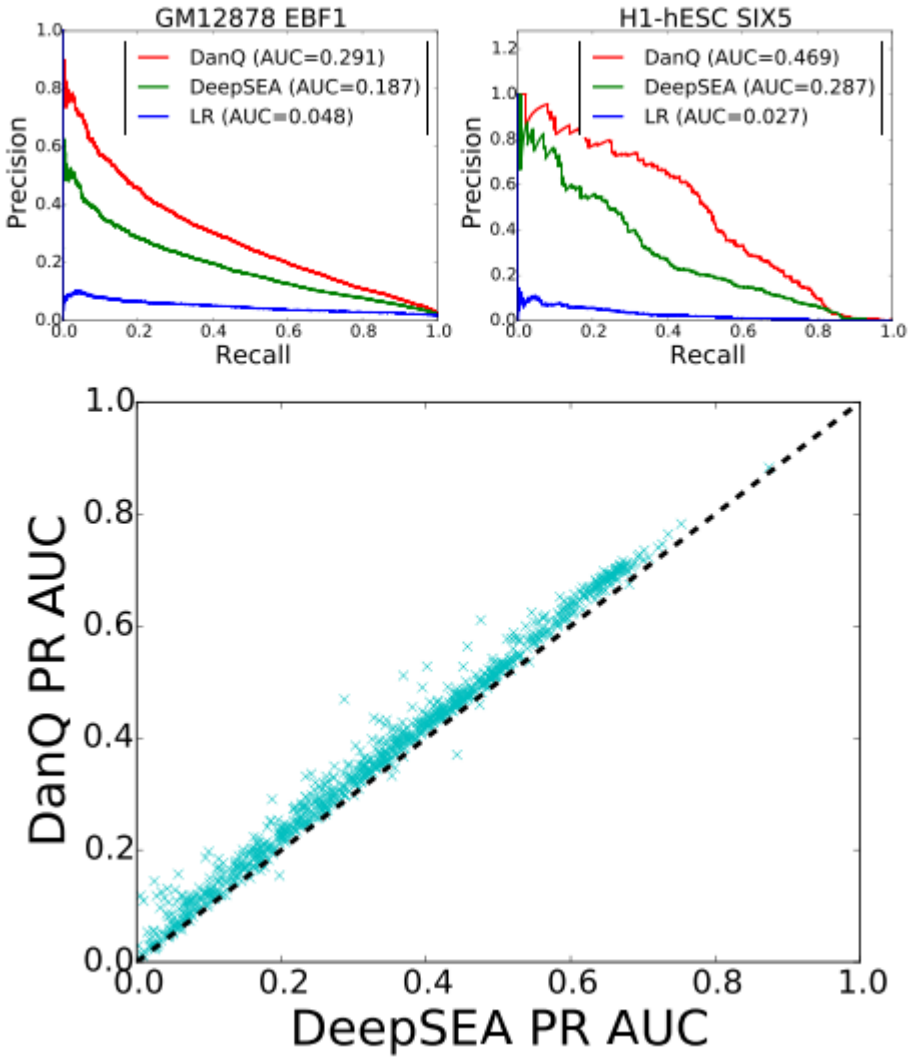
DanQ: Quantifying the Function of DNA

- Motivation: Over 98% of the human genome is non-coding and 93% of disease-associated variants lie in non-coding regions.
- Proposed: DanQ, hybrid convolutional and bi-directional long short-term memory recurrent neural network predicting non-coding function.
- Data:
  - Input: GRCh37 reference genome segmented into non-overlapping 200-bp bins.
  - Labels: Intersecting 919 ChIP-seq and DNase-seq peak sets from uniformly processed ENCODE and Roadmap Epigenomics data

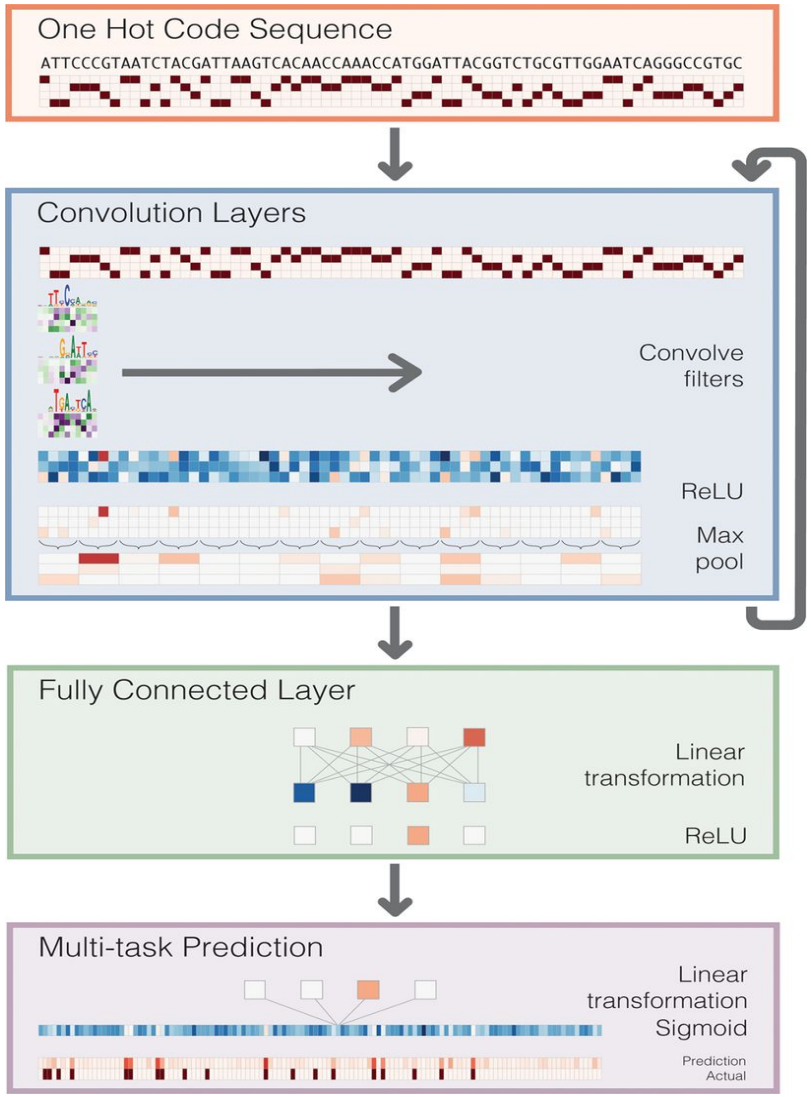


Daniel Quang and Xiaohui Xie. 2016. DanQ: A hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Research* 44, 11.

DanQ vs DeepSEA



Basset: CNN-based Accessible Genome Analysis



1. convert the sequence to a "one hot code" representation

2. scanning weight matrices across the input matrix to produce an output matrix with a row for every convolution filter and a column for every position in the input

3. linear transformation of the input vector and apply a ReLU.

4. linear transformation to a vector of 164 elements that represents the target cells

Kelley et al. (2016). *Genome Research*, *26*(7), 990–999

DeepBind: Protein–Nucleic acid Binding Site Prediction

DeepBind is a CNN based supervised learning where

Input: segments of sequences and

labels (output): experimentally determined binding score (ex. ChIP-seq peaks)

$$f(s) = \mathrm{net}_W\left(\mathrm{pool}\left(\mathrm{rect}_b\left(\mathrm{conv}_M(s)\right)\right)\right)$$

$$X = \mathrm{conv}_M(s)$$

$$X_{i,k} = \sum_{j=1}^{m}\sum_{l=1}^{4} S_{i+j,l} M_{k,j,l}$$

$$Y = \mathrm{rectify}_b(X)$$

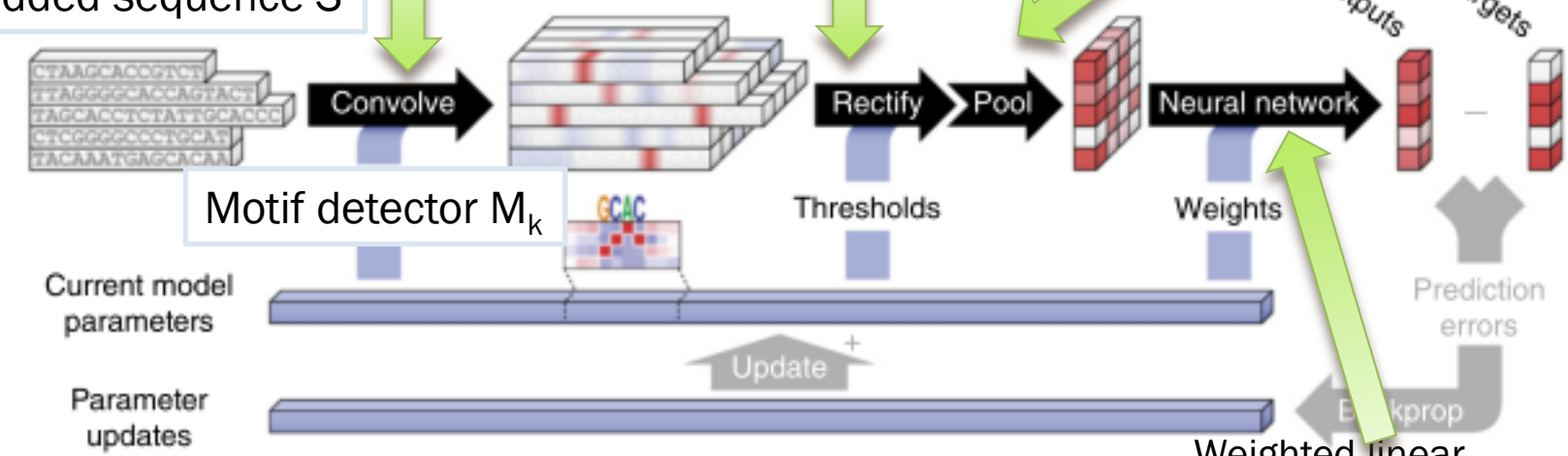
$$Y_{i,k} = \max(0, X_{i,k} - b_k).$$

Threshold of each motif detector k

$$z = \mathrm{max\_pool}(Y)$$

$$z_k = \max(Y_{1,k}, \ldots, Y_{n,k}).$$

padded sequence S



Motif detector M_k

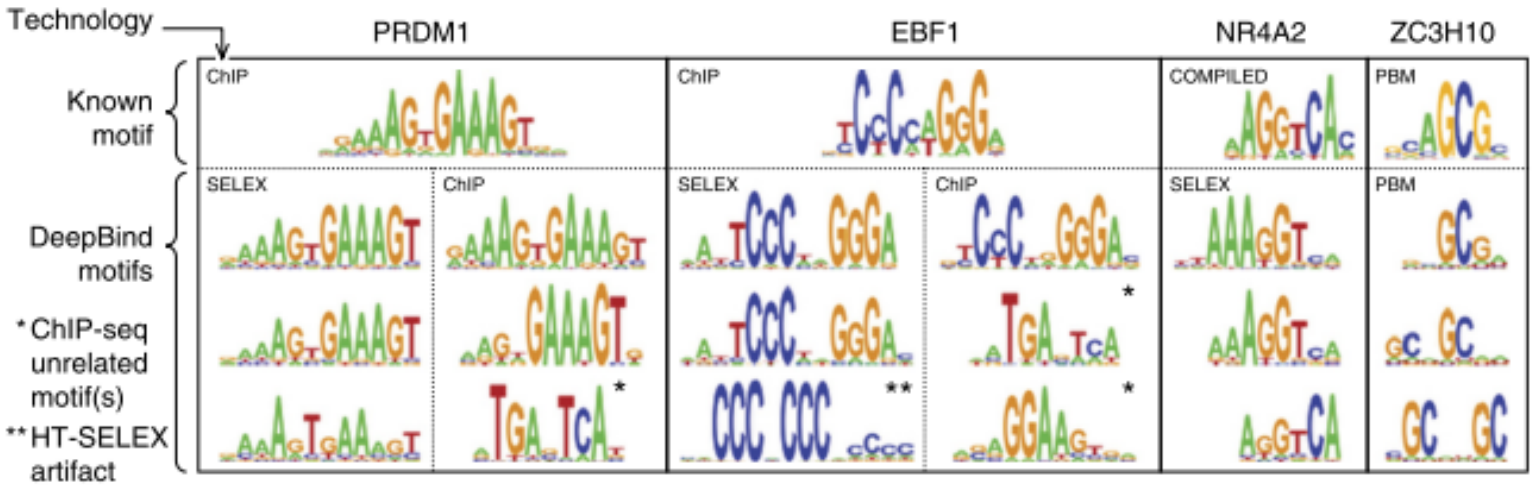
Update parameter by stochastic gradient descent

Weighted linear combination of pooled features

Alipanahi et al (2015) *Nature Biotechnology*, *33*(8), 831–838.

Motif Extraction capability of DEEPBIND

The trained motif detector M_k and visualization with sequence logo



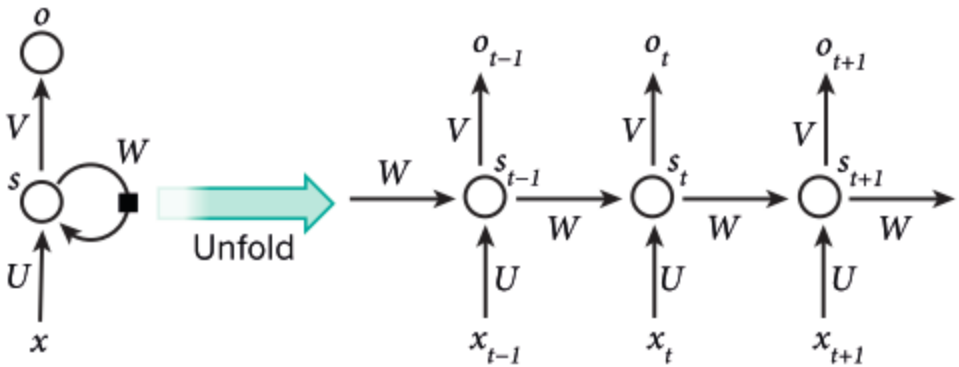
Generating sequence logo to find motifs

1. Feed all sequences from the test set through the convolutional and rectification stages of the DeepBind model,
2. Align all the sequences that passed the activation threshold for at least one position *i*.
3. Generate a position frequency matrix (PFM) and transform it into a sequence logo.

Alipanahi et al (2015) *Nature Biotechnology*, *33*(8), 831–838.

RNN for variable length Seq. Input

- Recurrent Neural Network
  - Able to work with sequence input of variable length
  - Capture long range interactions within the input sequences and across outputs.
  - Difficult to work with and train



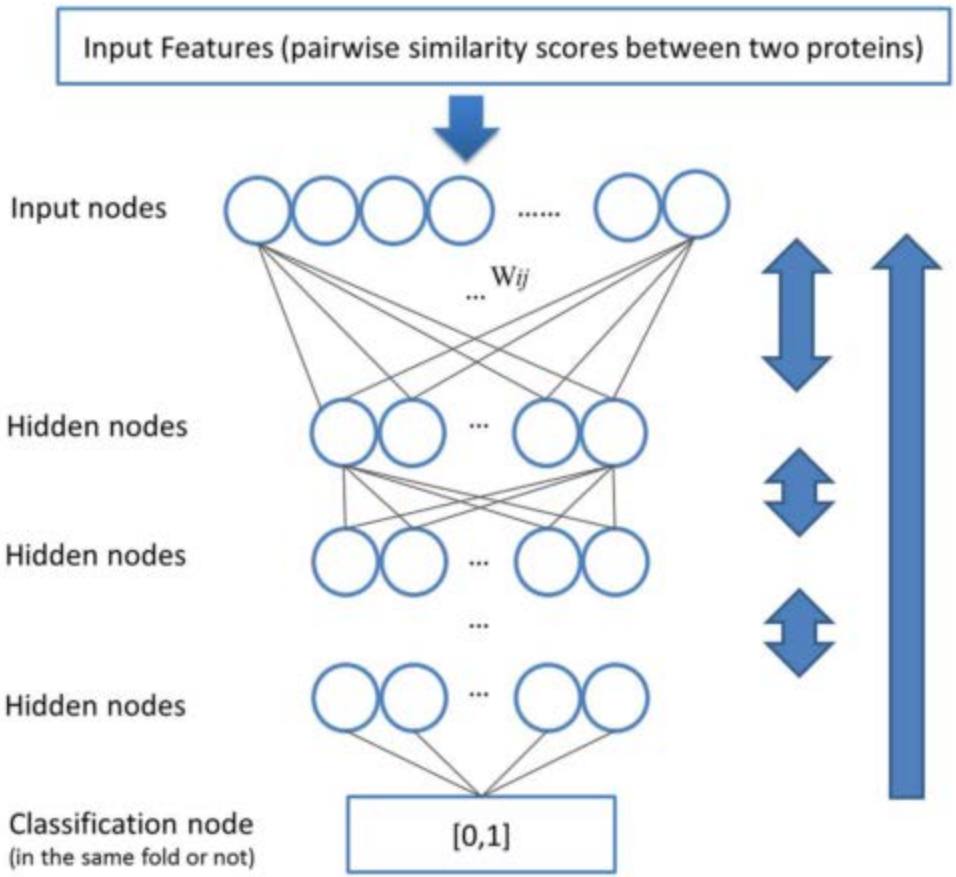
A recurrent neural network and the unfolding in time of the computation involved in its forward computation. (fig 5 of LeCun et al. 2015 Nature)

  - Not many success here

Protein Structure Prediction

- Protein structure prediction methods tend to apply unsupervised method or combination of NN methods
- Types of unsupervised DNN methods:
  - Restricted Boltzmann Machines (RBM)
  - Deep Belief Networks
- Combination methods
  - Deep Conditional Neural Fields

Stacking RBM in Protein Fold Recognition



84 features from five types of sequence alignment and/or protein structure prediction tools

Layer by layer learning with restricted Boltzmann machine (RBM).

Same fold or not

Jo et al. (2015). *Scientific Reports*, 5, 17573.

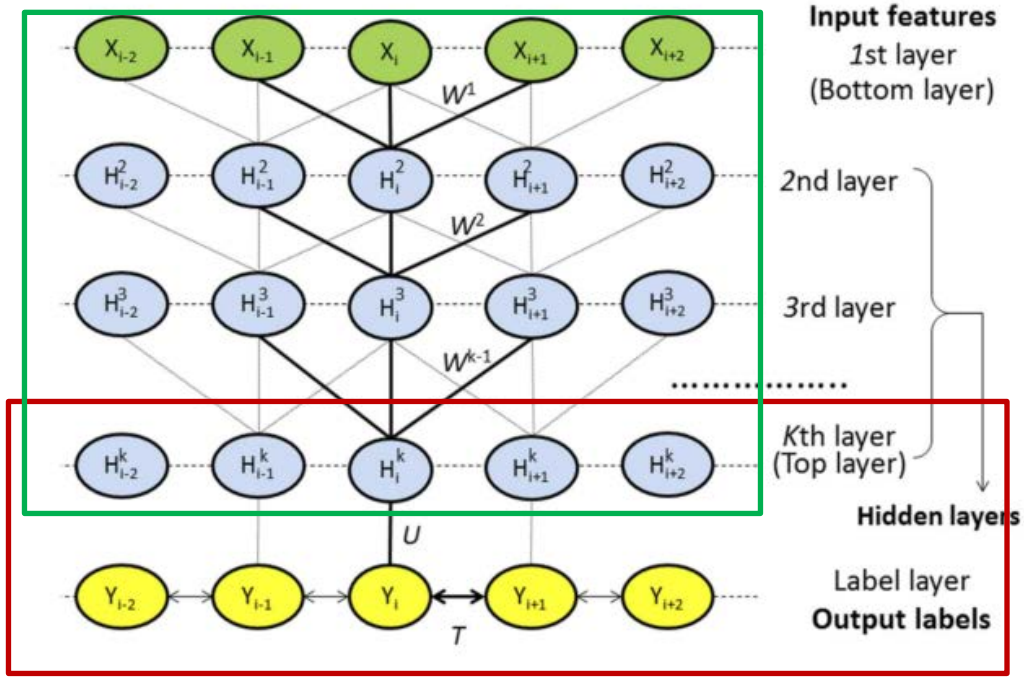
DEEPCNF: Secondary Structure Prediction

The architecture of Deep Convolutional Neural Field

fixed window size of 11:
average length of an alpha helix is around eleven residues and that of a beta strand is around six

Xi the associated input features of residue i.

5-7 layer CNN



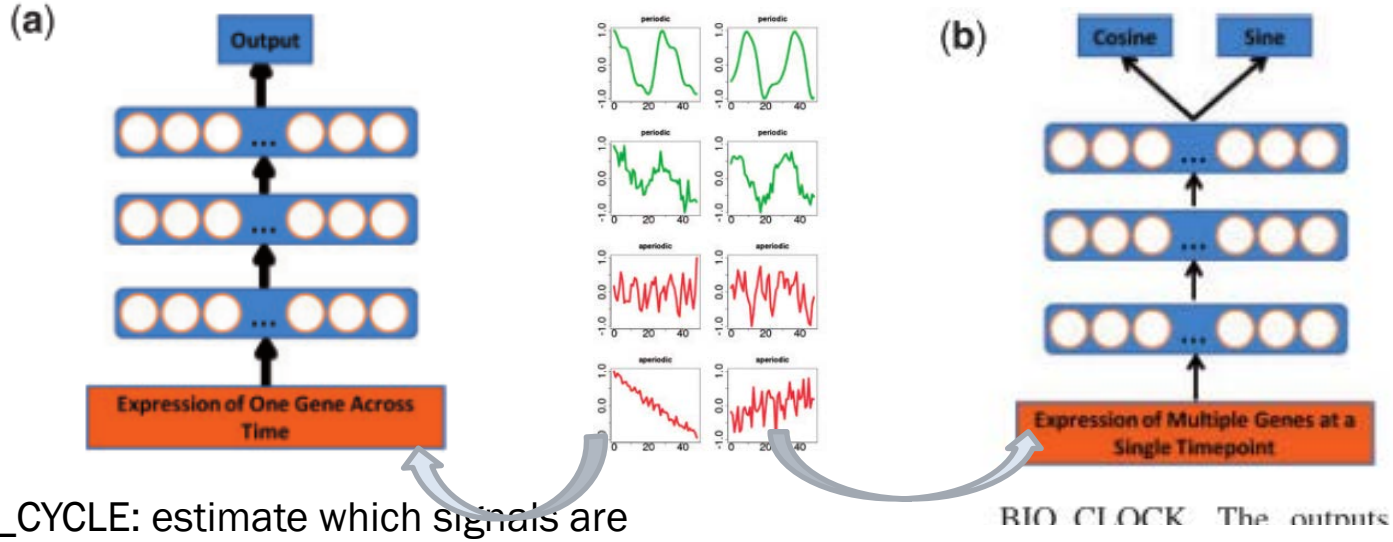
conditional random field (CRF) with U and T being the model parameters.

Calculates conditional probability of SS labels on input features

Wang et al. (2016) *Scientific Reports*, *6*(January), 18962.

Circadian Rhythms

GOAL: inferring whether a given genes oscillate in circadian fashion or not and inferring the time at which a set of measurements was taken



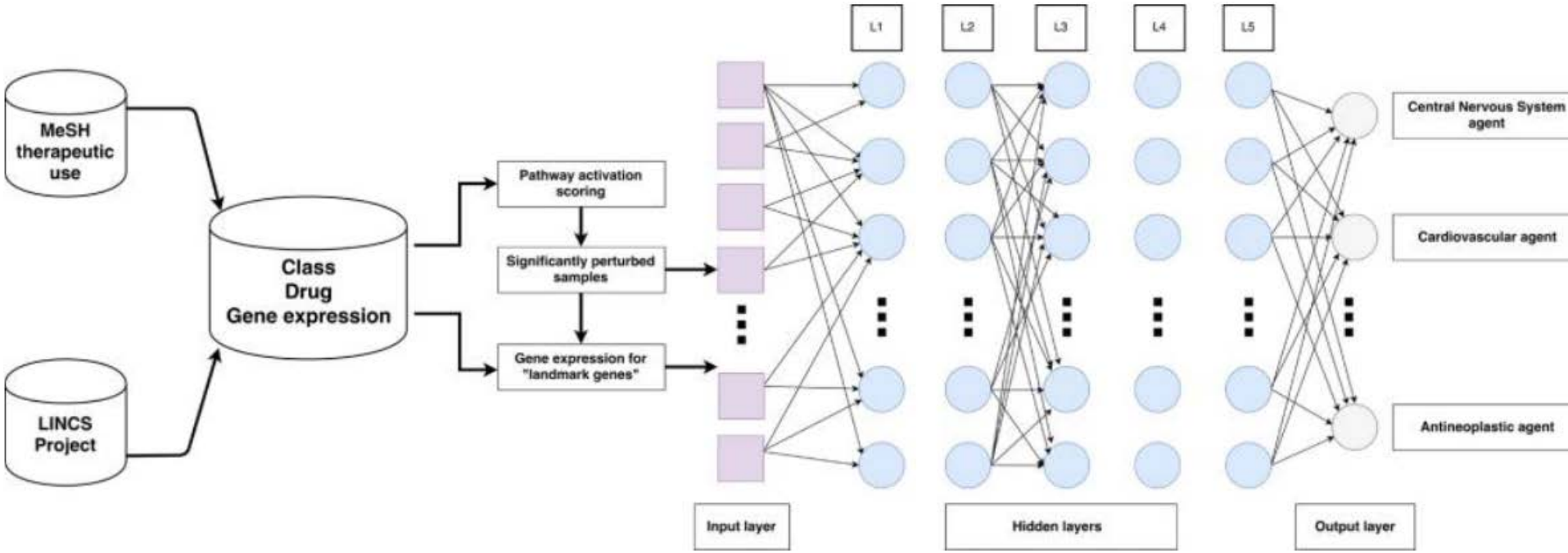
BIO_CYCLE: estimate which signals are periodic in high-throughput circadian experiments, producing estimates of amplitudes, periods, phases, as well as several statistical significance measures.
DATA: data sampled over 24 and 48h

BIO_CLOCK: estimate the time at which a particular single-time-point transcriptomic experiment was carried

Agostinelli, et al. (2016). *Bioinformatics*, *32*(12), i8–i17.

Predicting Properties of Drugs

- Input: transcriptional response data sets (transcriptional profile)
- Goal: classify various drugs to therapeutic categories

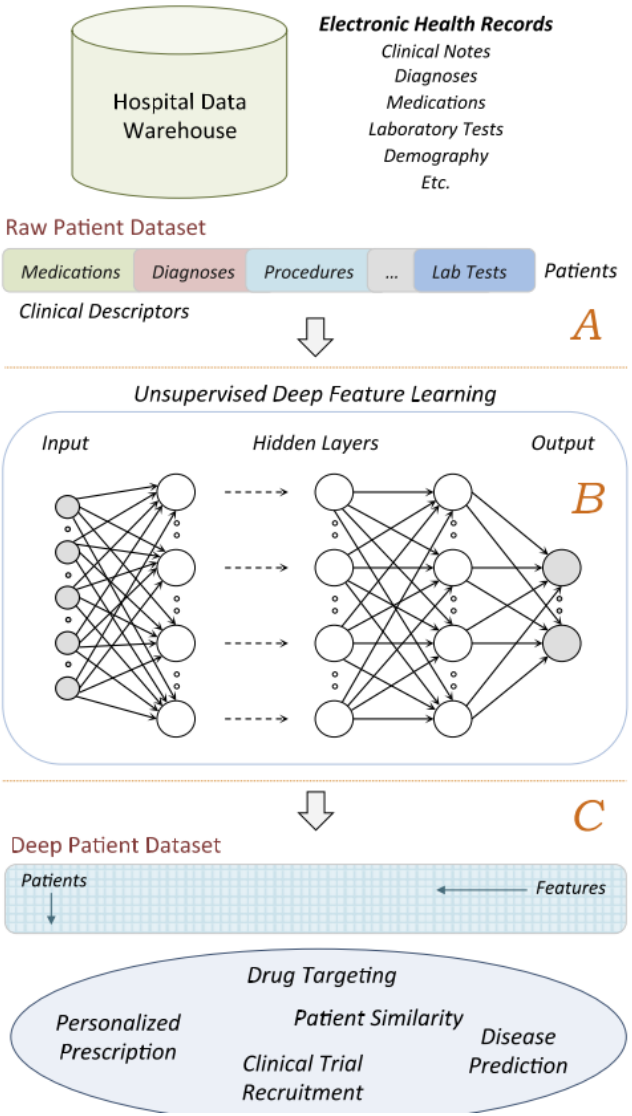


input layers of 977 and 271 neural nodes,

A. Aliper, et al. 2016. Deep learning applications for predicting pharmacological properties of drugs and drug repurposing using transcriptomic data. *Molecular Pharmaceutics* 13, 7.

Deep Patient: Unsupervised Prognostic Prediction based on EHR

- Feature learning:
  - three-layer stack of denoising autoencoders
- Data: EHRs of
  - about 700,000 patients from the Mount Sinai data warehouse.
  - evaluation using 76,214 test patients comprising 78 diseases from diverse clinical domains and temporal windows
- Prediction: random forest classifier



R. Miotto et al. 2016. Deep Patient: An Unsupervised Representation to Predict the Future of Patients from the Electronic Health Records. *Scientific reports* 6, April.

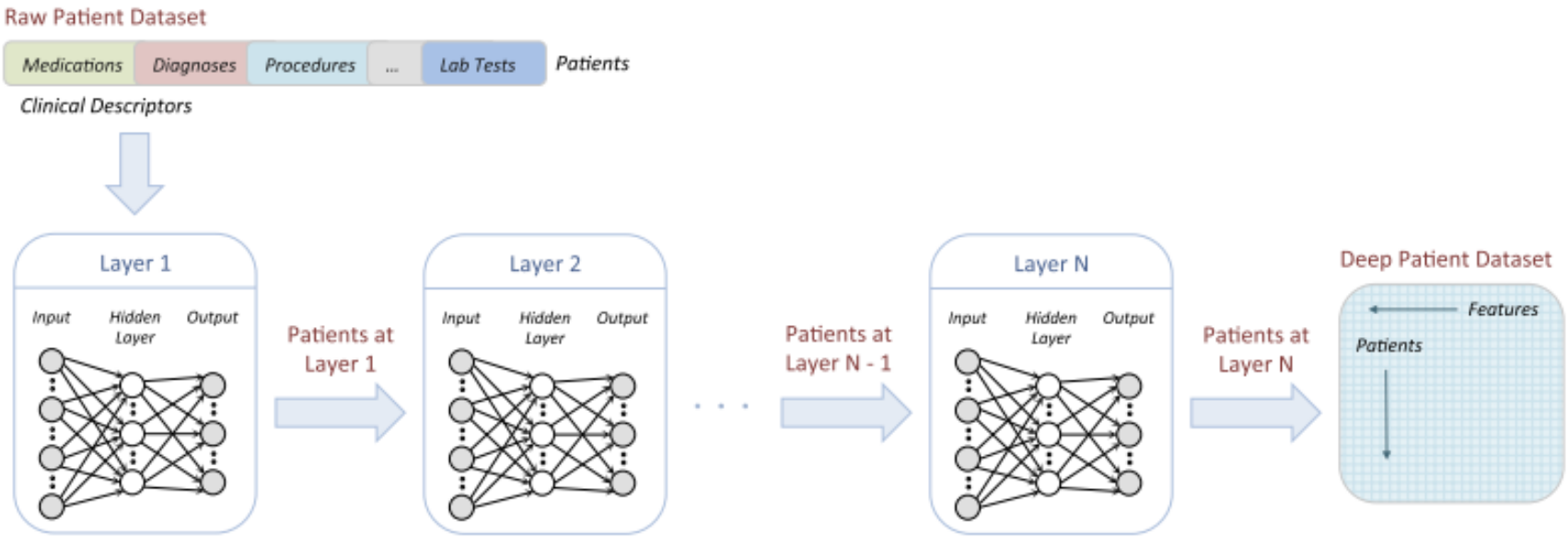


Figure 2. Diagram of the unsupervised deep feature learning pipeline to transform a raw dataset into the deep patient representation through multiple layers of neural networks. Each layer of the neural network is trained to produce a higher-level representation from the result of the previous layer.

Disease classification results

Time Interval = 1 year (76,214 patients)			
		Classification Threshold = 0.6	
Patient Representation	**AUC-ROC**	**Accuracy**	**F-Score**
RawFeat	0.659	0.805	0.084
PCA	0.696	0.879	0.104
GMM	0.632	0.891	0.072
K-Means	0.672	0.887	0.093
ICA	0.695	0.882	0.101
DeepPatient	**0.773***	**0.929***	**0.181***

Disease classification experiment

Time Interval = 1 year (76,214 patients)			
	Area under the ROC curve		
Disease	**RawFeat**	**PCA**	**DeepPatient**
Diabetes mellitus with complications	0.794	0.861	**0.907**
Cancer of rectum and anus	0.863	0.821	**0.887**
Cancer of liver and intrahepatic bile duct	0.830	0.867	**0.886**
Regional enteritis and ulcerative colitis	0.814	0.843	**0.870**

Reference

1. Alipanahi, B., Delong, A., Weirauch, M. T., & Frey, B. J. (2015). Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nature Biotechnology*, *33*(8), 831–838.
2. Dahl, G., Jaitly, N., & Salakhutdinov, R. (2014). Multi-task Neural Networks for QSAR Predictions. *arXiv Preprint arXiv:1406.1231*, 1–21.
3. Eickholt, J., & Cheng, J. (2012). Predicting protein residue-residue contacts using deep networks and boosting. *Bioinformatics*, *28*(23), 3066–3072.
4. Eickholt, J., & Cheng, J. (2013). DNdisorder: predicting protein disorder using boosting and deep networks. *BMC Bioinformatics*, *14*(1), 88.
5. Gawehn, E., Hiss, J. A., & Schneider, G. (2016). Deep Learning in Drug Discovery. *Molecular Informatics*, *35*(1), 3–14.
6. Jo, T., Hou, J., Eickholt, J., & Cheng, J. (2015). Improving Protein Fold Recognition by Deep Learning Networks. *Scientific Reports*, *5*, 17573.
7. Kelley, D. R., Snoek, J., & Rinn, J. L. (2016). Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Research*, *26*(7), 990–999.
8. Leung, M. K. K., Xiong, H. Y., Lee, L. J., & Frey, B. J. (2014). Deep learning of the tissue-regulated splicing code. *Bioinformatics*, *30*(12), 121–129.
9. Sønderby, S. K., & Winther, O. (2014). Protein Secondary Structure Prediction with Long Short Term Memory Networks. Retrieved from http://arxiv.org/abs/1412.7828
10. Wang, S., Peng, J., Ma, J., & Xu, J. (2016). Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields. *Scientific Reports*, *6*(January), 18962.
11. Wang, S., Weng, S., Ma, J., & Tang, Q. (2015). DeepCNF-D: Predicting Protein Order/Disorder Regions by Weighted Deep Convolutional Neural Fields. *International Journal of Molecular Sciences*, *16*(8), 17315–17330.
12. Zhang, S., Zhou, J., Hu, H., Gong, H., Chen, L., Cheng, C., & Zeng, J. (2015). A deep learning framework for modeling structural features of RNA-binding protein targets. *Nucleic Acids Research*, *44*(4), 1–14.
13. Zhou, J., & Troyanskaya, O. G. (2015). Predicting effects of noncoding variants with deep learning-based sequence model. *Nature Methods*, *12*(10), 931–4.
14. A. Aliper, et al. 2016. Deep learning applications for predicting pharmacological properties of drugs and drug repurposing using transcriptomic data. *Molecular Pharmaceutics* 13, 7.
15. R. Miotto et al. 2016. Deep Patient: An Unsupervised Representation to Predict the Future of Patients from the Electronic Health Records. *Scientific reports* 6, April.

Reference to Reviews

1. **Angermueller, C., Pärnamaa, T., Parts, L., & Oliver, S. (2016). Deep Learning for Computational Biology. *Molecular Systems Biology*, (12), 878.**
2. Gawehn, E., Hiss, J. A., & Schneider, G. (2016). Deep Learning in Drug Discovery. *Molecular Informatics*, *35*(1), 3–14.
3. Mamoshina, P., Vieira, A., Putin, E., & Zhavoronkov, A. (2016). Applications of Deep Learning in Biomedicine. *Molecular Pharmaceutics*, acs.molpharmaceut.5b00982.
4. Ladislav Rampasek and Anna Goldenberg. 2016. TensorFlow: Biology's Gateway to Deep Learning? *Cell Systems* 2, 1: 12–14.

Tensor Flow Tutorial

Contents and examples extended from **Udacity Deep Learning** by Google
https://classroom.udacity.com/courses/ud730/

Off-the-shelf Deep learning Tools

4x slower than competitors
but it's expected to be improved.

Table 1. Overview of existing deep learning frameworks, comparing four widely used software solutions.

	Caffe	Theano	Torch7	TensorFlow
Core language	C++	Python, C++	LuaJIT	C++
Interfaces	Python, Matlab	Python	C	Python
Wrappers		Lasagne, Keras, sklearn-theano		Keras, Pretty Tensor, Scikit Flow
Programming paradigm	Imperative	Declarative	Imperative	Declarative
Well suited for	CNNs, Reusing existing models, Computer vision	Custom models, RNNs	Custom models, CNNs, Reusing existing models	Custom models, Parallelization, RNNs

Table 1 in Angermueller et al. (2016) *Molecular Systems Biology*, (12), 878.

Installing

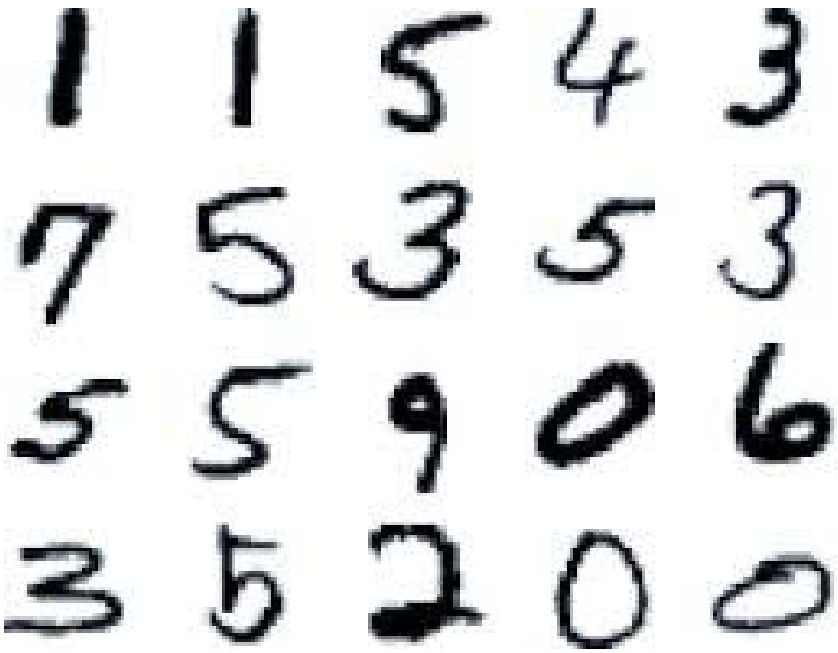
- **Install 64-bit Python 3.5 & pip (or** Anaconda3-4.2.0-Windows-x86_64)
- **Install virtualenv:**
  - CMD: pip install virtualenv
  - CMD: pip install virtualenvwrapper-win
- Create virtual environment
  - CMD: mkvirtualenv tensorflowCPU
- Install the CPU-only version of TensorFlow in the virtual environment
  - (TENSOR~) C:\Users\Name> pip install --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-0.12.1-cp35-cp35m-win_amd64.whl

- The role of the Python code in TensorFlow is to build this external computation graph, and to dictate which parts of the computation graph should be run.

- Other heavy lifting such as numerical computations are don outside Python.

Mnist data

- 10 labels
- 1 channel
- 28x28 images



Trying out MNIST tutorials in Tensorflow.org

GOTO: https://www.tensorflow.org/tutorials/mnist/pros/

Load MNIST Data

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

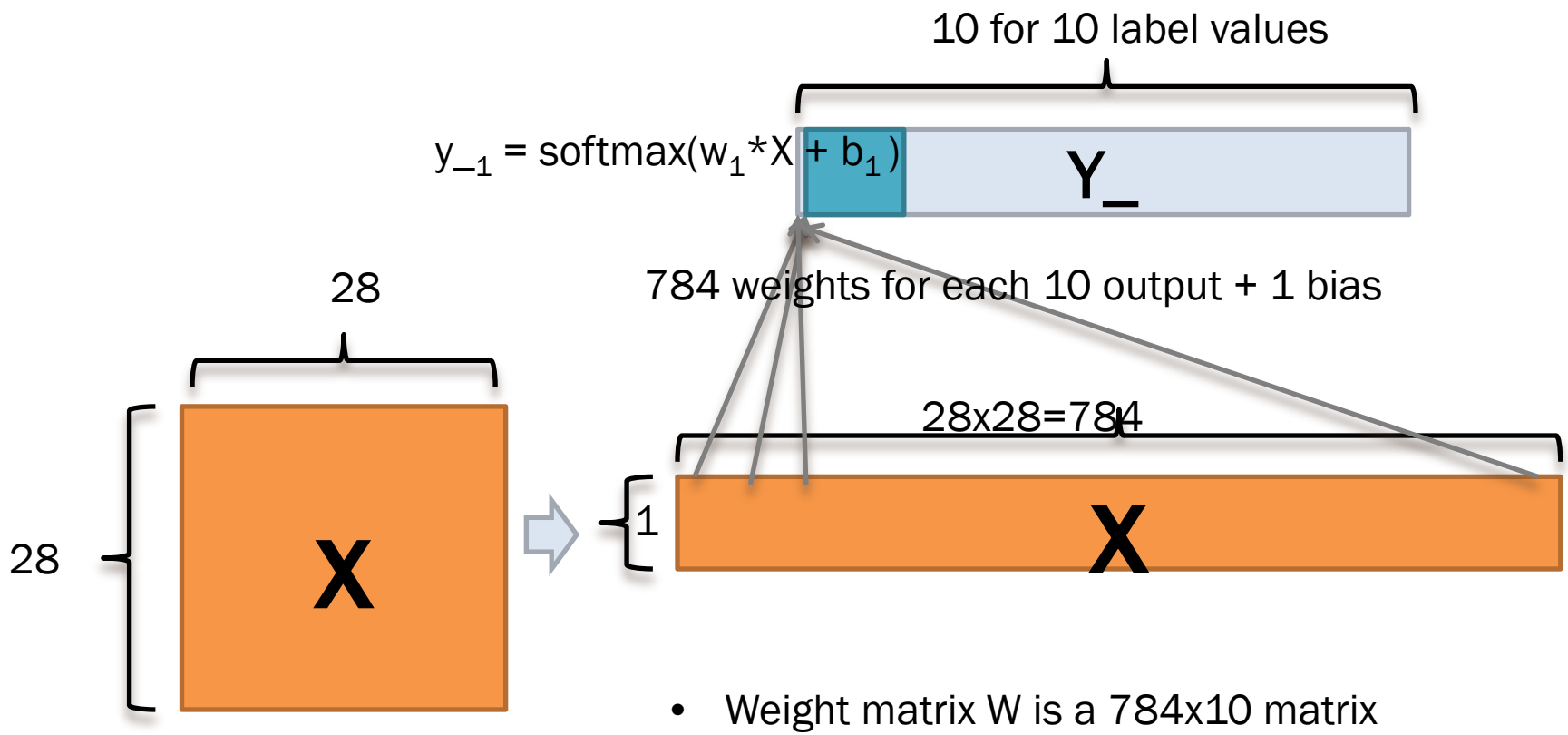
stores the training, validation, and testing sets

Start TensorFlow InteractiveSession

```
import tensorflow as tf
sess = tf.InteractiveSession()
```

It allows you to interleave operations which build a computation graph with ones that run the graph.

MODEL1: Build a Softmax Regression Model



- Weight matrix W is a 784x10 matrix
  - we have 784 input features fully connected to 10 outputs
- Bias vector b is a 10-dimensional vector
  - we have 10 classes

Placeholders: create nodes for the input images and target output classes.

```
x = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])
```

Variables: define & initalize weights W and bias b variables

```
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))

sess.run(tf.global_variables_initializer())
```

Define the regression model.

```
z = tf.matmul(x,W) + b
```

Define the loss function : one used to update W and bias

```
cross_entropy =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(z, y_))
```

Applies the softmax on the model's unnormalized model prediction (z) and sums across all classes

Takes average over the sums across 10 classes

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Training Step

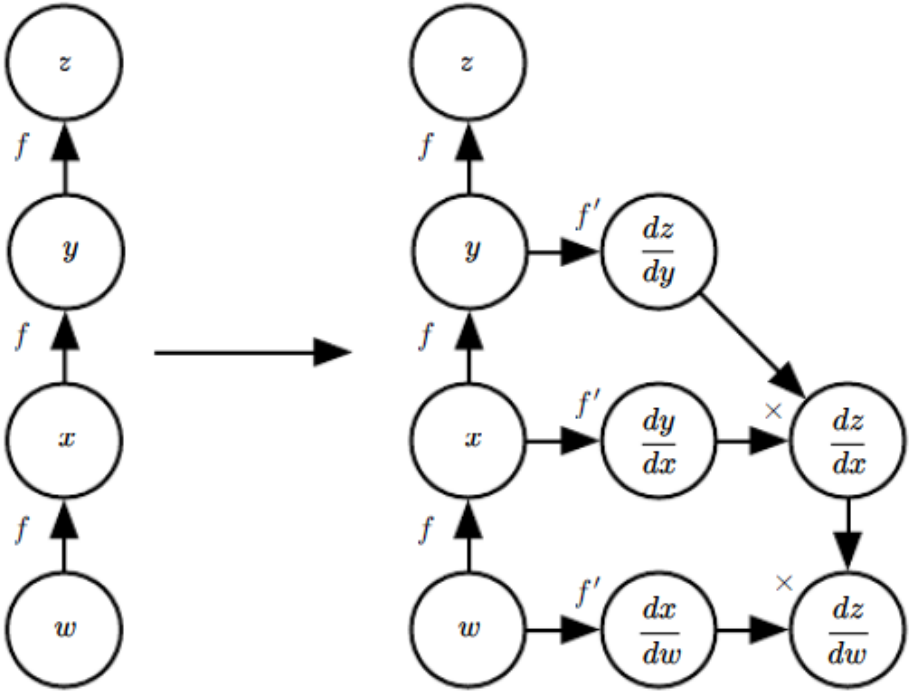
```
train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Steepest gradient descent, with a step length of 0.5, to descend the cross entropy.
Other built-in optimization functions:
https://www.tensorflow.org/api_docs/python/train/#optimizers

- TensorFlow actually added set of new operations to the computation graph.
  - Ones to compute gradients,
  - Ones to compute parameter update steps, and
  - Ones apply update steps to the parameters.

TensorFlow Back-propagation approach

TensorFlow take a computational graph and add additional nodes to the graph that provide a symbolic description of the desired derivatives.



symbol-to-symbol approach to computing derivatives

Training iteration

```
for i in range(1000):
  batch = mnist.train.next_batch(100)
  train_step.run(feed_dict={x: batch[0], y_: batch[1]})
```

Evaluate model

Predicted label

true label

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

Take the avg.

Change Bool to float

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction,
tf.float32))
```

evaluate our accuracy on the test data

```
print(accuracy.eval(feed_dict={x: mnist.test.images, y_:
mnist.test.labels}))
```

```
C:\Users\Sael Lee>workon tensorflowCPU
(TENSOR~1) C:\Users\Sael Lee>python
Python 3.5.2 |Continuum Analytics, Inc.| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from tensorflow.examples.tutorials.mnist import input_data
>>> mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting MNIST_data\train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting MNIST_data\train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz
>>>
>>> import tensorflow as tf
>>> sess = tf.InteractiveSession()
>>> x = tf.placeholder(tf.float32, shape=[None, 784])
>>> y_ = tf.placeholder(tf.float32, shape=[None, 10])
>>> W = tf.Variable(tf.zeros([784,10]))
>>> b = tf.Variable(tf.zeros([10]))
>>> sess.run(tf.global_variables_initializer())
>>>
>>>
>>> y = tf.matmul(x,W) + b
>>> cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
>>> train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
>>> for i in range(1000):
...   batch = mnist.train.next_batch(100)
...   train_step.run(feed_dict={x: batch[0], y_: batch[1]})
...
>>> correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
>>> accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
>>> print(accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
0.9165
```

Get 92% accuracy => very bad for MNIST

MODEL2: Build a Multilayer Convolutional Network

Weight Initialization

```
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)
```

One way to randomize. initialize weights with a small amount of noise for symmetry breaking, and to prevent 0 gradients.

Since we're using ReLU neurons, we should initialize them with a slightly positive initial bias to avoid "dead neurons"

Define Convolution and Pooling function

Model:

- Convolution stride of 1 and are zero padded so that the output is the same size as the input (same padding).
- Pooling: max pooling over 2x2 blocks.

```
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

```
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
```

*Computes a 2-D convolution given 4-D input and filter tensors.

tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None, data_format=None, name=None)

1. Flattens the filter to a 2-D matrix with shape [filter_height * filter_width * in_channels, output_channels].
2. Extracts image patches from the input tensor to form a *virtual* tensor of shape [batch, out_height, out_width, filter_height * filter_width * in_channels].
3. For each patch, right-multiplies the filter matrix and the image patch vector.

https://www.tensorflow.org/api_docs/python/nn/convolution#conv2d

```
def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

```
tf.nn.max_pool(value, ksize, strides, padding,
                        data_format='NHWC', name=None)
```

ARGUMENTS:

- **value**: A 4-D Tensor with shape [batch, height, width, channels] and type tf.float32.
- **ksize**: A list of ints that has length >= 4. The size of the window for each dimension of the input tensor.
- **strides**: A list of ints that has length >= 4. The stride of the sliding window for each dimension of the input tensor.
- **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm.
- **data_format**: A string. 'NHWC' and 'NCHW' are supported.
- **name**: Optional name for the operation.

1st Convolutional Layer

patch size, #input channel, # output channel

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
```

convolution will compute 32 features for each 5x5 patch

Bias per each 32 output channel

```
x_image = tf.reshape(x, [-1,28,28,1])
```

Reshape x to 4d tensor
2nd&3rd 2d image dim. 4th #of input channel

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

Convolve X_image with the weight tensor, add the bias, apply the ReLU function

reduce the image size to 14x14.

2nd Convolutional Layer

```
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

image size has been reduced to 7x7

Densely Connected Layer

```
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

fully-connected layer with 1024 neurons to allow processing on the entire image.

Add Dropout

To reduce overfitting, apply **dropout** before the readout layer.

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

Create placeholder for probability that a neuron's output is kept during dropout.

`tf.nn.dropout` op automatically handles scaling neuron outputs in addition to masking them

Readout Layer

```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

Train and Evaluate the Model

Almost similar the SoftMax example with the following differences:

- Replace the steepest gradient descent optimizer with the more sophisticated ADAM optimizer.
- Include the additional parameter keep_prob in feed_dict to control the dropout rate.
- Add logging to every 100th iteration in the training process.

WARNING but it does 20,000 training iterations and may take a while (possibly up to half an hour), depending on your processor.

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y_conv, y_))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
sess.run(tf.global_variables_initializer())
for i in range(20000):
  batch = mnist.train.next_batch(50)
  if i%100 == 0:
    train_accuracy = accuracy.eval(feed_dict={
        x:batch[0], y_: batch[1], keep_prob: 1.0})
    print("step %d, training accuracy %g"%(i, train_accuracy))
  train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

print("test accuracy %g"%accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

Let's change this to 2000 not to crush your laptop

```
step 12600, training accuracy 1
step 12700, training accuracy 0.98
step 12800, training accuracy 1
step 12900, training accuracy 1
step 13000, training accuracy 1
step 13100, training accuracy 1
step 13200, training accuracy 1
step 13300, training accuracy 1
step 13400, training accuracy 1
step 13500, training accuracy 1
step 13600, training accuracy 1
step 13700, training accuracy 1
step 13800, training accuracy 1
step 13900, training accuracy 1
step 14000, training accuracy 0.98
step 14100, training accuracy 1
step 14200, training accuracy 1
step 14300, training accuracy 1
step 14400, training accuracy 1
step 14500, training accuracy 1
step 14600, training accuracy 1
step 14700, training accuracy 0.98
step 14800, training accuracy 1
step 14900, training accuracy 1
step 15000, training accuracy 1
step 15100, training accuracy 1
step 15200, training accuracy 1
step 15300, training accuracy 0.98
step 15400, training accuracy 1
step 15500, training accuracy 0.98
step 15600, training accuracy 1
step 15700, training accuracy 1
step 15800, training accuracy 1
step 15900, training accuracy 1
step 16000, training accuracy 1
step 16100, training accuracy 1
step 16200, training accuracy 1
step 16300, training accuracy 1
step 16400, training accuracy 1
step 16500, training accuracy 1
step 16600, training accuracy 1
step 16700, training accuracy 1
step 16800, training accuracy 1
step 16900, training accuracy 1
step 17000, training accuracy 1
step 17100, training accuracy 1
step 17200, training accuracy 1
step 17300, training accuracy 1
step 17400, training accuracy 1
step 17500, training accuracy 1
step 17600, training accuracy 1
step 17700, training accuracy 0.98
step 17800, training accuracy 1
step 17900, training accuracy 1
step 18000, training accuracy 1
step 18100, training accuracy 1
step 18200, training accuracy 1
step 18300, training accuracy 1
step 18400, training accuracy 1
```

```
step 18900, training accuracy 1
step 19000, training accuracy 1
step 19100, training accuracy 1
step 19200, training accuracy 0.98
step 19300, training accuracy 1
step 19400, training accuracy 1
step 19500, training accuracy 1
step 19600, training accuracy 1
step 19700, training accuracy 1
step 19800, training accuracy 1
step 19900, training accuracy 1
>>> print("test accuracy %g"%accuracy.eval(feed_dict={
...     x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
test accuracy 0.9924
>>>
```