Neural Networks

lecturer: Jiří Matas, matas@cmp.felk.cvut.cz

authors: O. Drbohlav , J. Matas, D. Mishkin

Czech Technical University, Faculty of Electrical Engineering
Department of Cybernetics, Center for Machine Perception
121 35 Praha 2, Karlovo nám. 13, Czech Republic

http://cmp.felk.cvut.cz

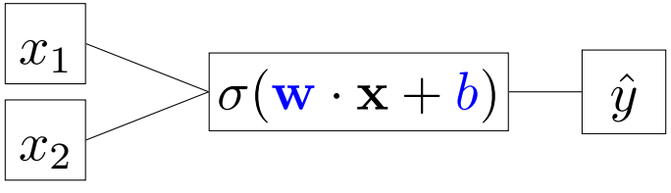
Last update: Nov 2020

Neural Networks - Motivation (1)

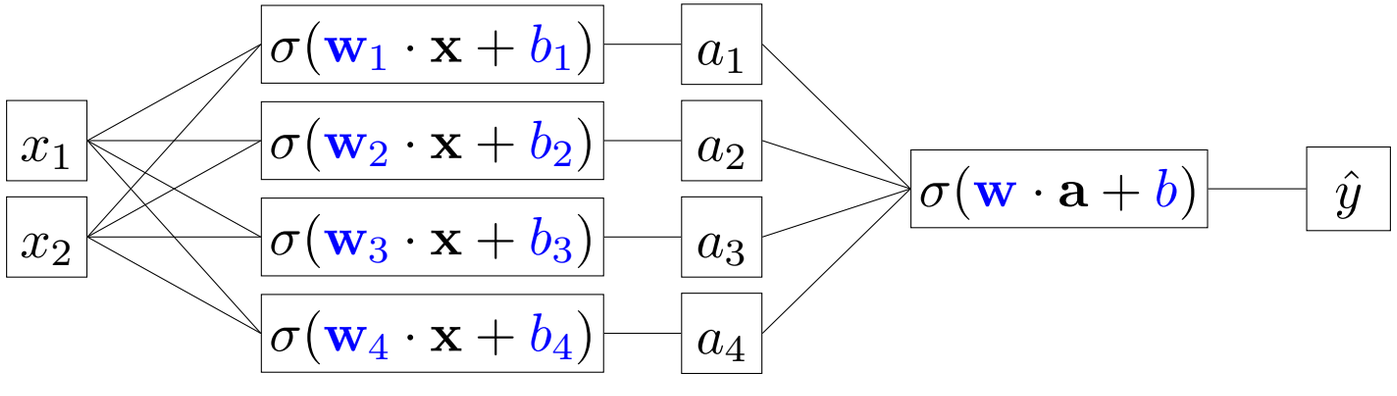
Recall the Perceptron: given the perceptron parameters $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$, the classification $\hat{y}$ to two classes $\{-1, 1\}$ for a vector $\mathbf{x} \in \mathbb{R}^n$ is performed as

$$\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)\,, \tag{1}$$

which is an affine (often called linear) function $l(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, followed by a non-linear function $\sigma : \mathbb{R} \to \mathbb{R}$, $\sigma(z) = \text{sign}(z)$. The perceptron is also often depicted as follows:



to make the linear combination of elements of the input vector explicit. Perceptron is a linear classifier; these are well understood, have low VC dimension, etc. and thus it is a natural next step to combine them to a more complex classifiers: By letting more of them sharing the input, as well as using their outputs a_i's as inputs to other perceptrons:



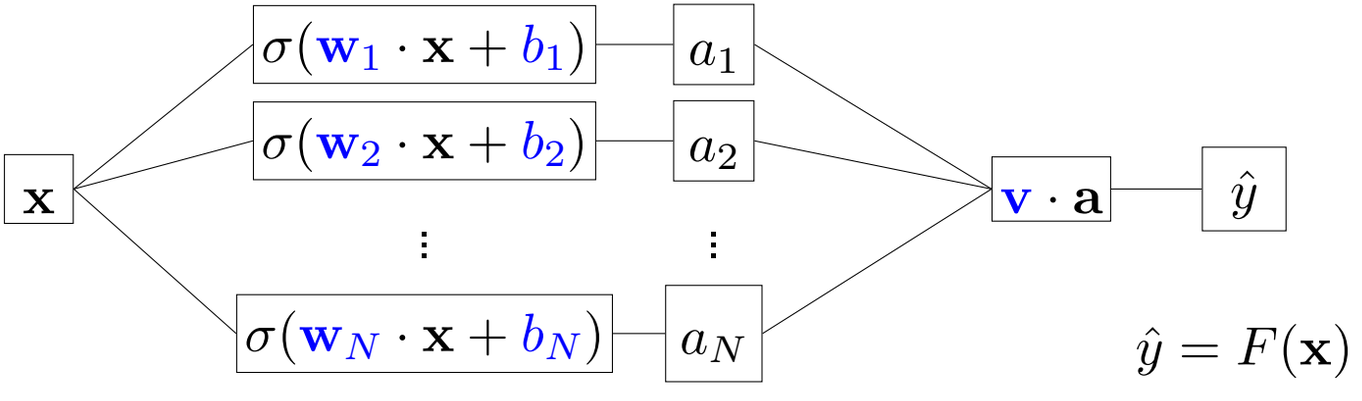
Neural Networks - Motivation (2)

Universal Approximation Theorem. Another strong motivation for forming such combinations of simple classifiers is the theorem which states that if $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonconstant, bounded and continuous function and f is a continuous function on unit hypercube $[0, 1]^n$ then for any $\epsilon > 0$ there exists $N \in \mathbb{N}$, $v_i, b_i \in \mathbb{R}$ and $\mathbf{w}_i \in \mathbb{R}^n$ such that

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)\,, \text{ and} \tag{2}$$

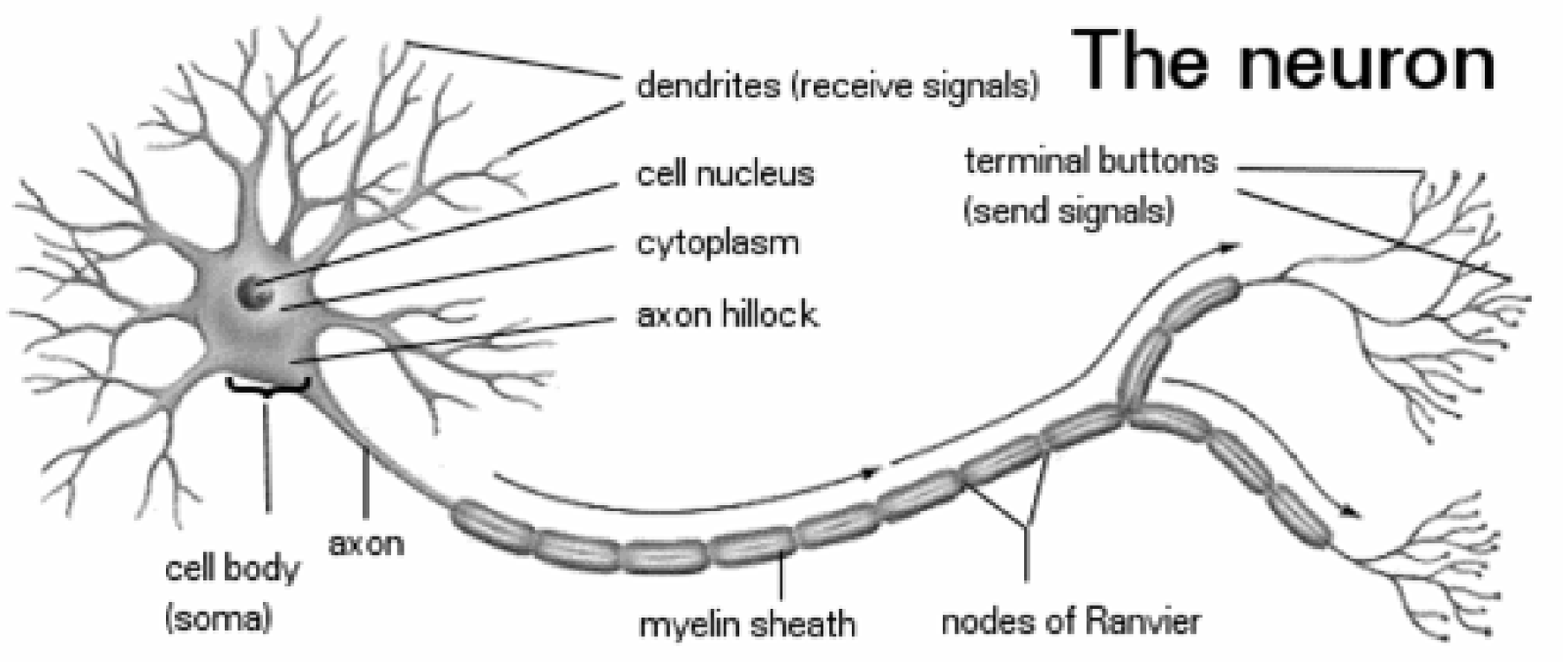
$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon\,, \quad \forall \mathbf{x} \in [0, 1]^m \tag{3}$$

By comparison, we see that the approximation is exactly captured by the following network with single hidden layer and linear output:



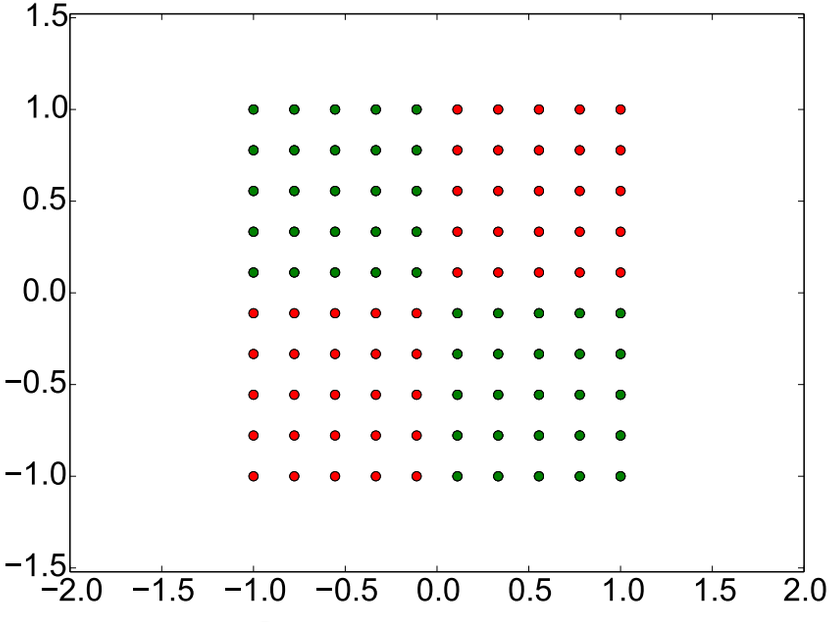
Neural Networks - Motivation (3)

'Biological' motivation - a real neuron is known to combine inputs to an output which is then passed to other neurons.



Historical Note

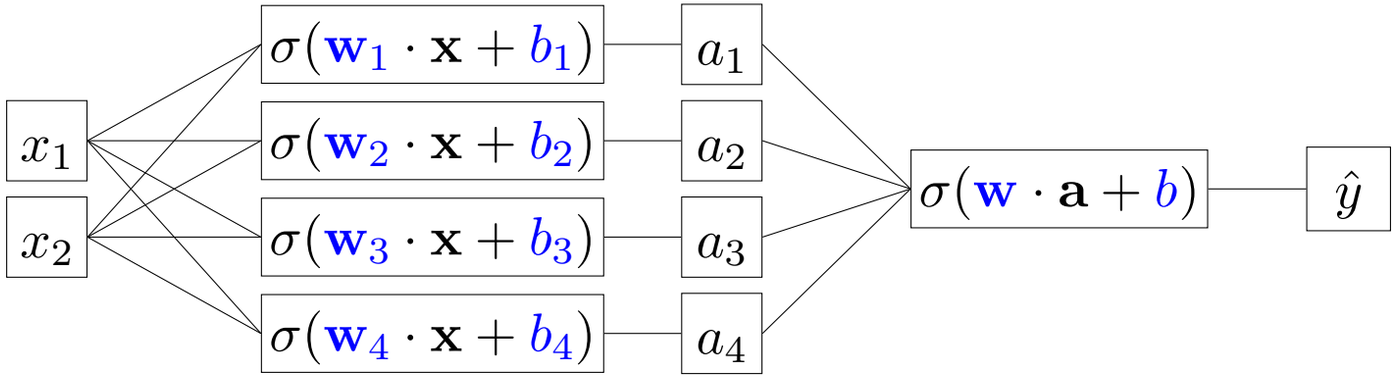
- Perceptron (Rosenblatt, 1956) with its simple learning algorithm generated a lot of excitement
- Minsky and Papert (1969) showed that even a simple XOR cannot be learnt by a perceptron
- But chaining perceptrons to a network (Multi-Layer Perceptron, MLP) *can* learn XOR



XOR arrangement

Layers, Concise Equivalent Representation

The network



input layer hidden layer output layer

can be rewritten in a more concise form as follows:

$$\text{(in)} \xrightarrow{\mathbf{x}\in\mathbb{R}^2} \boxed{\sigma(\overset{\mathbf{W}\in\mathbb{R}^{4\times 2},\mathbf{b}\in\mathbb{R}^4}{\mathbf{W}\mathbf{x}+\mathbf{b}})} \xrightarrow{\mathbf{a}\in\mathbb{R}^4} \boxed{\sigma(\overset{\mathbf{w}\in\mathbb{R}^4,b\in\mathbb{R}}{\mathbf{w}\cdot\mathbf{a}+b})} \xrightarrow{\hat{y}\in\mathbb{R}} \text{(out)} \tag{4}$$

where we introduced a convention that for $\sigma\colon \mathbb{R} \to \mathbb{R}$ and $\mathbf{z} \in \mathbb{R}^k$,
$\sigma(\mathbf{z}) = [\sigma(z_1), \sigma(z_2), ..., \sigma(z_k)]^\top$, i.e. the function is applied per component of the vector $\mathbf{z}$.

A network with M hidden layers can be written as (here shown with affine output layer)

$$\text{(in)} \xrightarrow{\mathbf{a}_1=\mathbf{x}\in\mathbb{R}^n} \Bigg[\overset{\textbf{hidden layers}}{\boxed{\sigma(\mathbf{W}_i\mathbf{a}_i+\mathbf{b}_i)}} \xrightarrow{\mathbf{a}_{i+1}}\Bigg]_{i=1}^{M} \rightarrow \overset{\textbf{output layer}}{\boxed{\mathbf{W}_{M+1}\mathbf{a}_{M+1}+\mathbf{b}_{M+1}}} \xrightarrow{\hat{\mathbf{y}}\in\mathbb{R}^K} \text{(out)} \tag{5}$$

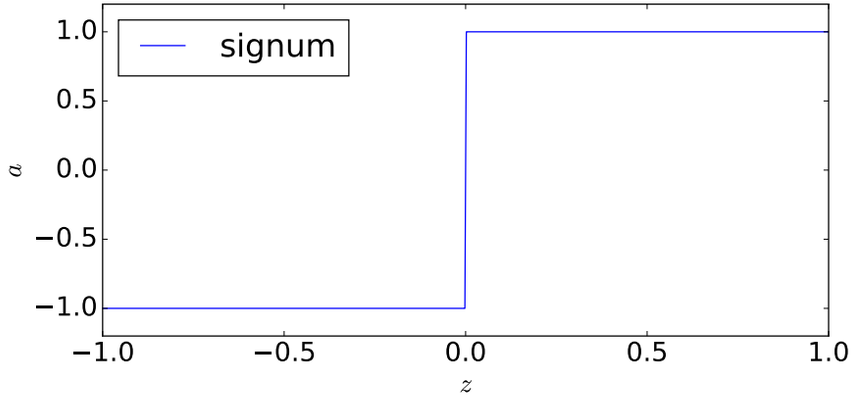
Layer Anatomy

$$\text{(in)} \xrightarrow{\mathbf{a}_1=\mathbf{x}\in\mathbb{R}^n} \Big[\underbrace{\boxed{\sigma(\mathbf{W}_i\mathbf{a}_i+\mathbf{b}_i)}}_{\textbf{hidden layers}} \xrightarrow{\mathbf{a}_{i+1}}\Big]_{i=1}^{M} \rightarrow \underbrace{\boxed{\mathbf{W}_{M+1}\mathbf{a}_{M+1}+\mathbf{b}_{M+1}}}_{\textbf{output layer}} \xrightarrow{\hat{\mathbf{y}}\in\mathbb{R}^K} \text{(out)} \quad (6)$$

Each hidden layer of an NN is composed of an affine function, followed by a non-linearity.

Non-linear functions

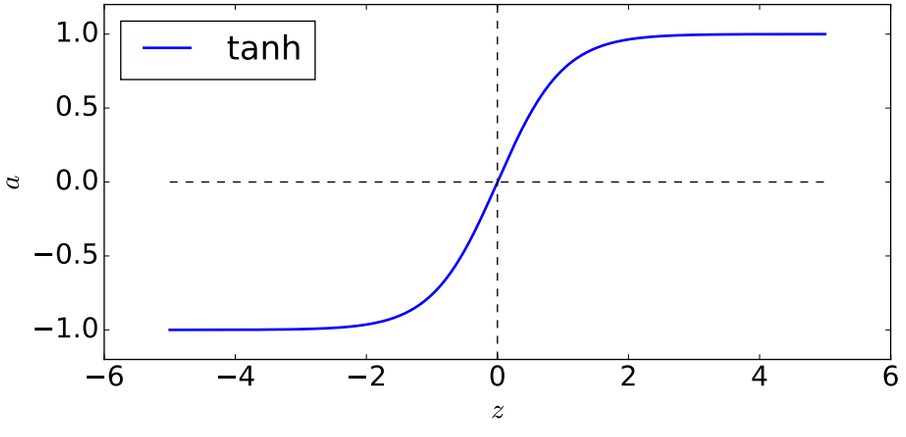
- $\sigma(z) = \text{sign}(z)$: used for the original perceptron. Unusable for an NN because this non-linear function is discontinuous and not differentiable at 0, and everywhere else has zero gradient. So, once we would like to optimize the parameters of the combination of perceptrons, gradient descent and other methods based on first and second order approximations could not proceed. This is why the original sign function has been replaced by functions with 'nicer' properties.

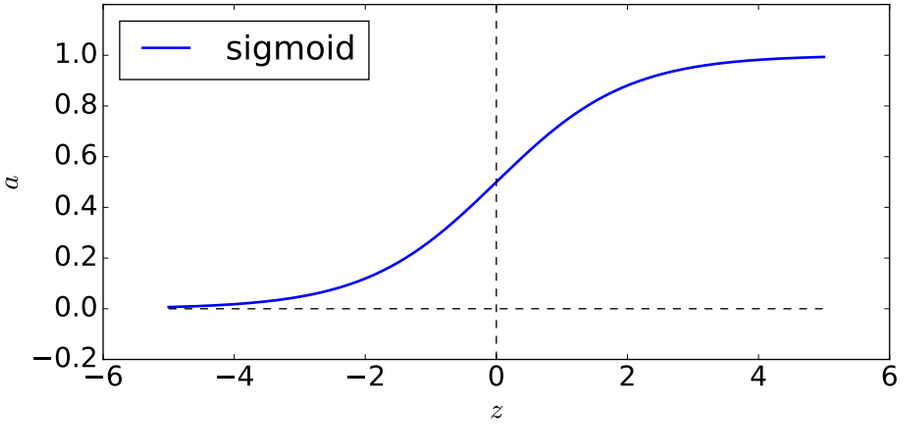


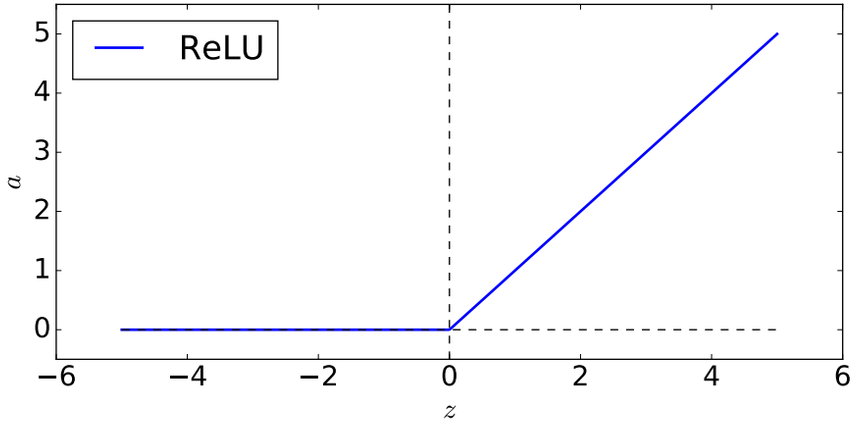
Layer Anatomy, Non-Linear Functions

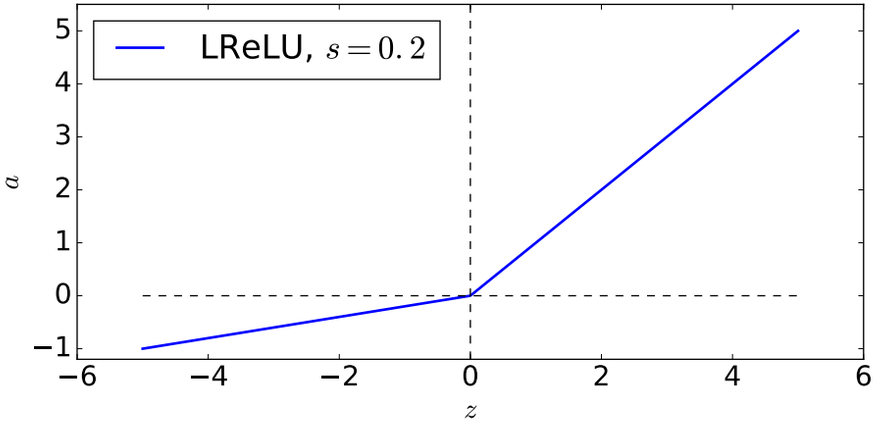
Non-linear functions

- $\sigma(z) = 1/(1+e^{-z})$: logistic sigmoid, $\sigma : \mathbb{R} \to [0, 1]$
- $\sigma(z) = \tanh(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$: tanh sigmoid, $\sigma : \mathbb{R} \to [-1, 1]$
- $\sigma(z) = \max(0, z)$: ReLU (rectified linear unit)
- $\sigma(z) = \max(0, z) + \min(0, sz)$ $(0 < s < 1)$: Leaky ReLU
- Many other









Output Layer and Loss Functions. K-class Classification (1)

So far, we have made a general assumption that the output of NN is a K-element vector $\hat{\mathbf{y}} \in \mathbb{R}^K$. Now we will discuss what form the output should take and how we will measure how good the output is.

K-class Classification. We may formally number the classes as $1, 2, ..., K$. But if classes are not ordinal, these numbers are really just labels; it doesn't mean that objects from classes 2 and 3 are in some sense closer than objects from classes 1 and K, say. Therefore, it would make no sense to have a one-dimensional output $\hat{y}$ trying to predict the class label. Instead, the output layer will consist of an affine function producing a K-dimensional vector, followed by the **softmax** which will convert it to class probabilities:

output layer

$$\xrightarrow{\mathbf{a}_{M+1}} \boxed{\mathbf{W}_{M+1}\mathbf{a}_{M+1} + \mathbf{b}_{M+1}} \xrightarrow{\mathbf{z}\in\mathbb{R}^K} \boxed{\text{softmax}} \xrightarrow{\hat{\mathbf{y}}\in\mathbb{R}^K} (\text{out}) \tag{7}$$

with

$$[\text{softmax}(\mathbf{z})]_k = \frac{\exp z_k}{\sum_{l=1}^{K} \exp z_l}\,. \tag{8}$$

For representing the class y of the training data point $(\mathbf{x}, y)$, **one-hot** representation is used which simply makes a K-dimensional vector which is everywhere zero except for the y-th component which is 1:

$$\text{onehot}(y) = [\delta_{1y}, \delta_{2y}, ...\delta_{Ky}]^\top = [0, 0, ..., \underset{\substack{\uparrow\\ \text{y-th place}}}{1}, ..., 0]^\top \in \mathbb{R}^K \tag{9}$$

Output Layer and Loss Functions. K-class Classification (2)

For a training data point $(\mathbf{x}, y)$, how to measure how far the prediction $\hat{\mathbf{y}} \in [0,1]^K$ for $\mathbf{x}$ is from the target vector $\mathbf{y} = \text{onehot}(y) \in \{0,1\}^K$?

- Squared difference:
$$J(\hat{\mathbf{y}}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2 . \tag{10}$$
This loss $= 0$ when the prediction matches the target and > 0 otherwise.
- Negative log-likelihood. To avoid confusion, let us denote the index of the target class l for a training data point $(\mathbf{x}, l)$; then

$$J(\hat{\mathbf{y}}, \mathbf{y}) = -\mathbf{y}^\top \log \hat{\mathbf{y}} = -\sum_{i=1}^{K} y_i \log \hat{y}_i = -\log \hat{y}_l . \tag{11}$$

Example: K-class logistic regression. The class conditionals $\hat{\mathbf{y}}(\mathbf{x}) = [p(1|\mathbf{x}), p(2|\mathbf{x}), ..., p(K|\mathbf{x})] \in \mathbb{R}^K$ can be written as ($\mathbf{W} \in \mathbb{R}^{n \times K}$, $\mathbf{b} \in \mathbb{R}^K$):

$$\text{(in)} \xrightarrow{\mathbf{x} \in \mathbb{R}^n} \boxed{\mathbf{W}\mathbf{x} + \mathbf{b}} \xrightarrow{\mathbf{z} \in \mathbb{R}^K} \boxed{\text{softmax}} \xrightarrow{\hat{\mathbf{y}} \in \mathbb{R}^K} \text{(out)} \tag{12}$$

Output Layer and Loss Functions. 2-class Classification

For two classes, the output $\hat{y}$ of the NN can be 1-dimensional, modeling the class posterior $p(1|\mathbf{x}) \in [0, 1]$. The posterior $p(2|\mathbf{x})$ is computed simply as $1 - \hat{y}$.

Example: 2-class logistic regression. The posteriors $p(1|\mathbf{x})$, $p(2|\mathbf{x})$ are modeled as $(\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R})$:

$$p(1|\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}, \qquad p(2|\mathbf{x}) = 1 - p(1|\mathbf{x}) \tag{13}$$

This can be written as a single-neuron NN with output $\hat{y} = p(1|\mathbf{x})$:

$$\text{(in)} \xrightarrow{\mathbf{x}} \boxed{\sigma(\mathbf{w} \cdot \mathbf{x} + b)} \xrightarrow{\hat{y}} \text{(out)} \tag{14}$$

where $\sigma(z) = 1/(1 + e^{-z})$ is the logistic sigmoid non-linearity.

Recall that negative log-likelihood loss has been used for the 2-class logistic regression:

$$J(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})] \tag{15}$$

where class labels y have conveniently been changed as $(1, 2) \mapsto (1, 0)$. The procedure for finding the optimal parameters $\boldsymbol{\theta} = \{\mathbf{w}, b\}$ was the gradient descent.

Output Layer and Loss Functions. Regression

The NN can of course also be used for regression, that is, finding a function which predicts a target $\mathbf{y} \in \mathbb{R}^K$ which is not constrained to represent probability distribution.

Squared difference is an obvious choice for such a problem.

Training the NN

Let the training data be $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$, where without loss of generality we assume that class labels y_i's have been converted to their one-hot representations $\mathbf{y}_i$'s if needed. Let $\boldsymbol{\theta}$ denote all parameters of the NN. We want to minimize

$$J(\mathcal{T}; \boldsymbol{\theta}) = \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} J(\hat{\mathbf{y}}(\mathbf{x}), \mathbf{y}) \,. \tag{16}$$

Gradient-based methods are used most of the time for minimizing the loss function $J(\mathcal{T}; \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$. We need to evaluate the gradient of loss w.r.t. the NN parameters, $\frac{\partial J(\hat{\mathbf{y}}(\mathbf{x}), \mathbf{y})}{\partial \boldsymbol{\theta}}$, in order to use it for updates of the gradient-descent type:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \mu \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}' \subseteq \mathcal{T}} \frac{\partial J(\hat{\mathbf{y}}(\mathbf{x}), \mathbf{y})}{\partial \boldsymbol{\theta}} \tag{17}$$

where μ is the learning rate and the summation is not necessarily over the entire dataset.

Computing gradient (1)

When computing the gradient, we will make use of the chain rule. Let $f\colon \mathbb{R}^n \to \mathbb{R}$ and $g\colon \mathbb{R}^m \to \mathbb{R}^n$. For $\mathbf{x} \in \mathbb{R}^m$, let $\mathbf{y} = g(\mathbf{x})$. Let us consider the composition of these functions, $f(g(\mathbf{x})) = f(\mathbf{y})$ ($f \circ g\colon \mathbb{R}^m \to \mathbb{R}$). There holds

$$\frac{\partial f(g(\mathbf{x}))}{\partial x_k} = \sum_{i=1}^{n} \frac{\partial f}{\partial y_i} \frac{\partial y_i}{\partial x_k}. \tag{18}$$

This can be written in a matrix form,

$$\frac{\partial f(g(\mathbf{x}))}{\partial \mathbf{x}} = \frac{\partial f}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = f' \mathbf{y}', \tag{19}$$

where $f' = \partial f / \partial \mathbf{y}$ and $\mathbf{y}' = \partial \mathbf{y} / \partial \mathbf{x}$ are Jacobian matrices:

$$f' = \left[\frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial y_2}, \ldots, \frac{\partial f}{\partial y_n} \right], \tag{20}$$

$$\mathbf{y}' = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_m} \\ \vdots & & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix} \tag{21}$$

Example: Gradient by Back Propagation (1)

Let us have the following network:

$$\xrightarrow[\mathbf{a}_1=\mathbf{x}]{\mathbf{a}_1\in\mathbb{R}^n} \boxed{\mathbf{W}_1\mathbf{a}_1+\mathbf{b}_1} \xrightarrow{\mathbf{z}_2\in\mathbb{R}^{m_2}} \boxed{\sigma} \xrightarrow{\mathbf{a}_2\in\mathbb{R}^{m_2}} \boxed{\mathbf{W}_2\mathbf{a}_2+\mathbf{b}_2} \xrightarrow{\mathbf{z}_3\in\mathbb{R}^{m_3}} \dots \tag{22}$$

$$\boxed{\sigma} \xrightarrow{\mathbf{a}_3\in\mathbb{R}^{m_3}} \boxed{\mathbf{W}_3\mathbf{a}_3+\mathbf{b}_3} \xrightarrow{\hat{\mathbf{y}}\in\mathbb{R}^K} \boxed{J(\hat{\mathbf{y}},\mathbf{y})}$$

where the non-linearities have been explicitly put to the chain. The set of NN parameters is $\boldsymbol{\theta}=\{\mathbf{W}_1,\mathbf{W}_2,\mathbf{W}_3,\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3\}$. For the loss, let us consider $J(\hat{\mathbf{y}},\mathbf{y})=\|\hat{\mathbf{y}}-\mathbf{y}\|^2$.

1. Compute $\mathbf{v}_3=\frac{\partial J}{\partial\hat{\mathbf{y}}}\big|_{\hat{\mathbf{y}}}$. This is a row vector, $\mathbf{v}_3\in\mathbb{R}^K$, $\mathbf{v}_3=2(\hat{\mathbf{y}}-\mathbf{y})^\top$.
2. $\frac{\partial J}{\partial\mathbf{b}_3}=\mathbf{v}_3\frac{\partial\hat{\mathbf{y}}}{\partial\mathbf{b}_3}\big|_{\mathbf{a}_3}=\mathbf{v}_3\mathbb{1}=\mathbf{v}_3$
3. $\frac{\partial J}{\partial(\mathbf{W}_3)_{kl}}=\mathbf{v}_3\frac{\partial\hat{\mathbf{y}}}{\partial(\mathbf{W}_3)_{kl}}=\mathbf{v}_3[0,0,...,(\mathbf{a}_3)_l,...,0,0]^\top=(\mathbf{v}_3)_k(\mathbf{a}_3)_l$

   $\uparrow$

   at k-th element

   So, arranging partial derivatives of J w.r.t. elements of $\mathbf{W}_3\in\mathbb{R}^{K\times m_3}$ to a matrix of the same dimensions then we can write

$$\frac{\partial J}{\partial\mathbf{W}_3}=\begin{bmatrix}\frac{\partial J}{\partial(\mathbf{W}_3)_{11}} & \frac{\partial J}{\partial(\mathbf{W}_3)_{12}} & \dots & \frac{\partial J}{\partial(\mathbf{W}_3)_{1m_1}}\\ \frac{\partial J}{\partial(\mathbf{W}_3)_{21}} & \frac{\partial J}{\partial(\mathbf{W}_3)_{22}} & \dots & \frac{\partial J}{\partial(\mathbf{W}_3)_{2m_1}}\\ \vdots & \ddots & & \\ \frac{\partial J}{\partial(\mathbf{W}_3)_{K1}} & \frac{\partial J}{\partial(\mathbf{W}_3)_{K2}} & \dots & \frac{\partial J}{\partial(\mathbf{W}_3)_{Km_1}}\end{bmatrix}=\mathbf{v}_3{}^\top\mathbf{a}_3{}^\top \tag{23}$$

Example: Gradient by Back Propagation (2)

Let us have the following network:

$$\xrightarrow[\mathbf{a}_1=\mathbf{x}]{\mathbf{a}_1\in\mathbb{R}^n} \boxed{\mathbf{W}_1\mathbf{a}_1+\mathbf{b}_1} \xrightarrow{\mathbf{z}_2\in\mathbb{R}^{m_2}} \boxed{\sigma} \xrightarrow{\mathbf{a}_2\in\mathbb{R}^{m_2}} \boxed{\mathbf{W}_2\mathbf{a}_2+\mathbf{b}_2} \xrightarrow{\mathbf{z}_3\in\mathbb{R}^{m_3}} \dots \tag{24}$$

$$\boxed{\sigma} \xrightarrow{\mathbf{a}_3\in\mathbb{R}^{m_3}} \boxed{\mathbf{W}_3\mathbf{a}_3+\mathbf{b}_3} \xrightarrow{\hat{\mathbf{y}}\in\mathbb{R}^K} \boxed{J(\hat{\mathbf{y}},\mathbf{y})}$$

4. Compute $\mathbf{v}_2 = \mathbf{v}_3\frac{\partial\hat{\mathbf{y}}}{\partial\mathbf{a}_3}\Big|_{\mathbf{a}_3}\frac{\partial\mathbf{a}_3}{\partial\mathbf{z}_3}\Big|_{\mathbf{z}_3} = \mathbf{v}_3\mathbf{W}_3\text{diag}[\sigma'(\mathbf{z}_3)] = (\mathbf{v}_3\mathbf{W}_3)\odot\sigma'(\mathbf{z}_3)$;

   Here $\sigma'(\mathbf{x}) = [\sigma'(x_1),\sigma'(x_2),...,\sigma'(x_n)]$ with $\mathbf{x}\in\mathbb{R}^n$ and σ' the derivative of σ, $\text{diag}(\mathbf{x})$ for $\mathbf{x}\in\mathbb{R}^n$ forms an n-by-n diagonal matrix with elements of $\mathbf{x}$ on the diagonal, and $\odot$ is the Hadamard (element-wise) product.

5. $\frac{\partial J}{\partial\mathbf{b}_2} = \mathbf{v}_2$

6. $\frac{\partial J}{\partial\mathbf{W}_2} = \mathbf{v}_2{}^\top\mathbf{a}_2{}^\top$

7. Compute $\mathbf{v}_1 = \mathbf{v}_2\mathbf{W}_2\text{diag}[\sigma'(\mathbf{z}_2)] = (\mathbf{v}_2\mathbf{W}_2)\odot\sigma'(\mathbf{z}_2)$

8. $\frac{\partial J}{\partial\mathbf{b}_1} = \mathbf{v}_1$

9. $\frac{\partial J}{\partial\mathbf{W}_1} = \mathbf{v}_1{}^\top\mathbf{a}_1{}^\top$

Gradient-based Optimization, Back Propagation

- As it has been just shown, gradient for all parameters can be efficiently computed (without repeating already performed computations) by computation flow from the last layer to the first – hence the name Back Propagation (aka backprop)
- Recall the update rule:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \mu \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}'\subseteq\mathcal{T}} \frac{\partial J(\hat{\mathbf{y}}(\mathbf{x}),\mathbf{y})}{\partial \boldsymbol{\theta}} \tag{25}$$

  - How to choose the learning rate μ? The rate is often changed adaptively during learning, based on monitoring of the learning process
  - The summation is often done over a subset $\mathcal{T}'$ of the training data. This provides an estimate of the actual gradient and the technique is called Stochastic Gradient Descent (SGD). Many alternatives (important one: SGD with momentum)
  - How to initialize the parameters? Rule of thumb (core idea, many variants): Initialize them such that variance of outputs is equal to 1 for all layers.

Deep Learning

Deep Learning

- Extremely successful branch of Machine Learning methods
- Lot of progress in recent (>10) years
- Includes:
  - Convolutional Neural Nets (CNNs): suitable for inputs which are translation-invariant and 'warpable'. Typical examples are visual signals (images)
  - Recurrent neural networks (RNNs): allow previous outputs to be used as inputs; recognition of time-series signals, speech recognition
  - Autoencoders: The goal is to output the inputs; The hidden layers have a bottleneck (fewer neurons than input layer) and thus the network is forced to learn 'compressed' representation of the input
  - ...

Example: Convolutional Neural Networks

Consider an image (an input layer) which is 64×64 pixels large. Let the next layer be a layer of the same size. If this is modeled as a fully connected network, the number of connections is $(64^2)^2 \approx 16M$.

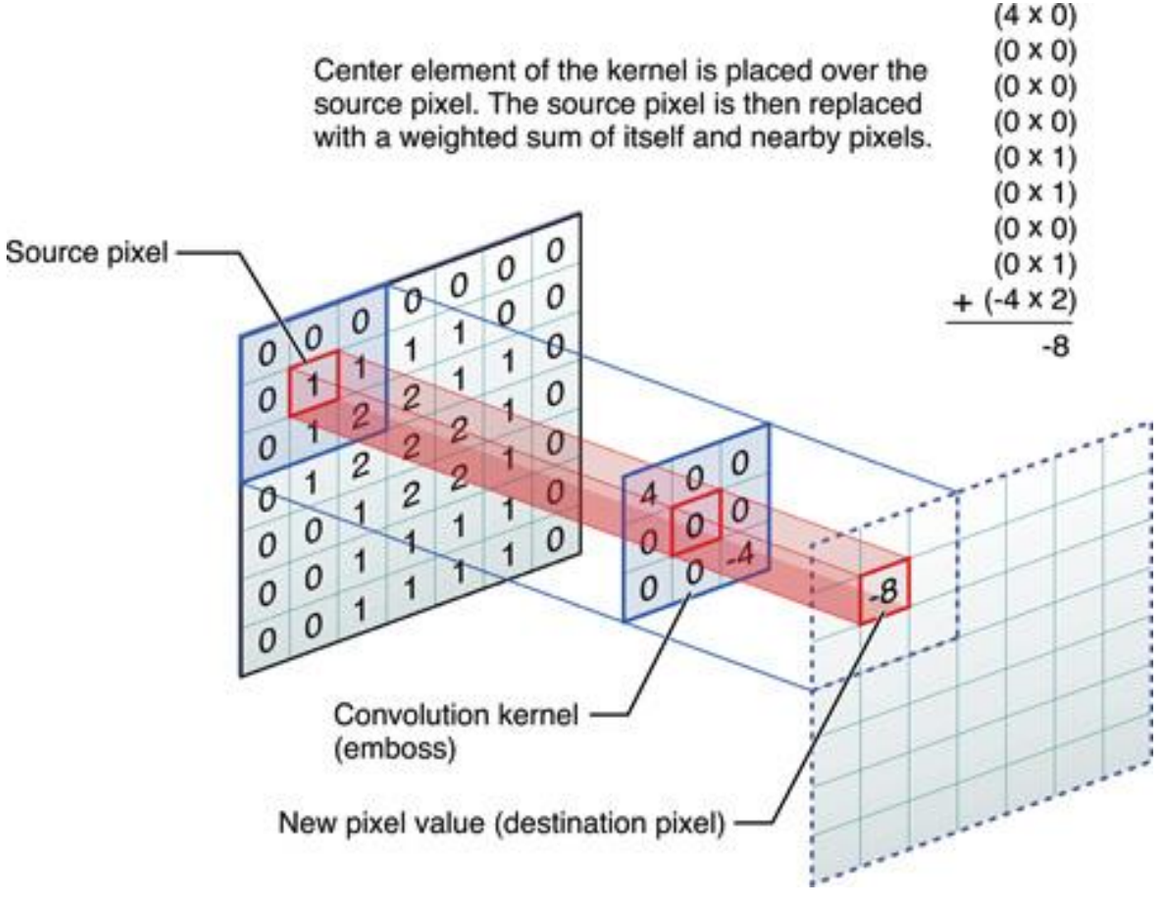
In contrast to that,

1. Make the connections only in a 5×5 neighbourhood of each neuron in the second layer; This would lower the number of parameters to $64^2 \cdot 25 \approx 102k$
2. Make the parameters of all 5×5 connections *shared*; This lowers the number of parameters to only 25.

Doing this corresponds to learning a *convolutional filter* of size 5×5. In practice, N (e.g. $N = 32$) filters are learnt in the first layer, forming N-channel output. The next layer then operates on all N channels, thus when e.g. $N = 32$ and the receptive field is 3×3, each of the next convolutional filters has $32 \cdot 3^2$ parameters.

Convolutional structure reduces the number of parameters by orders of magnitude.

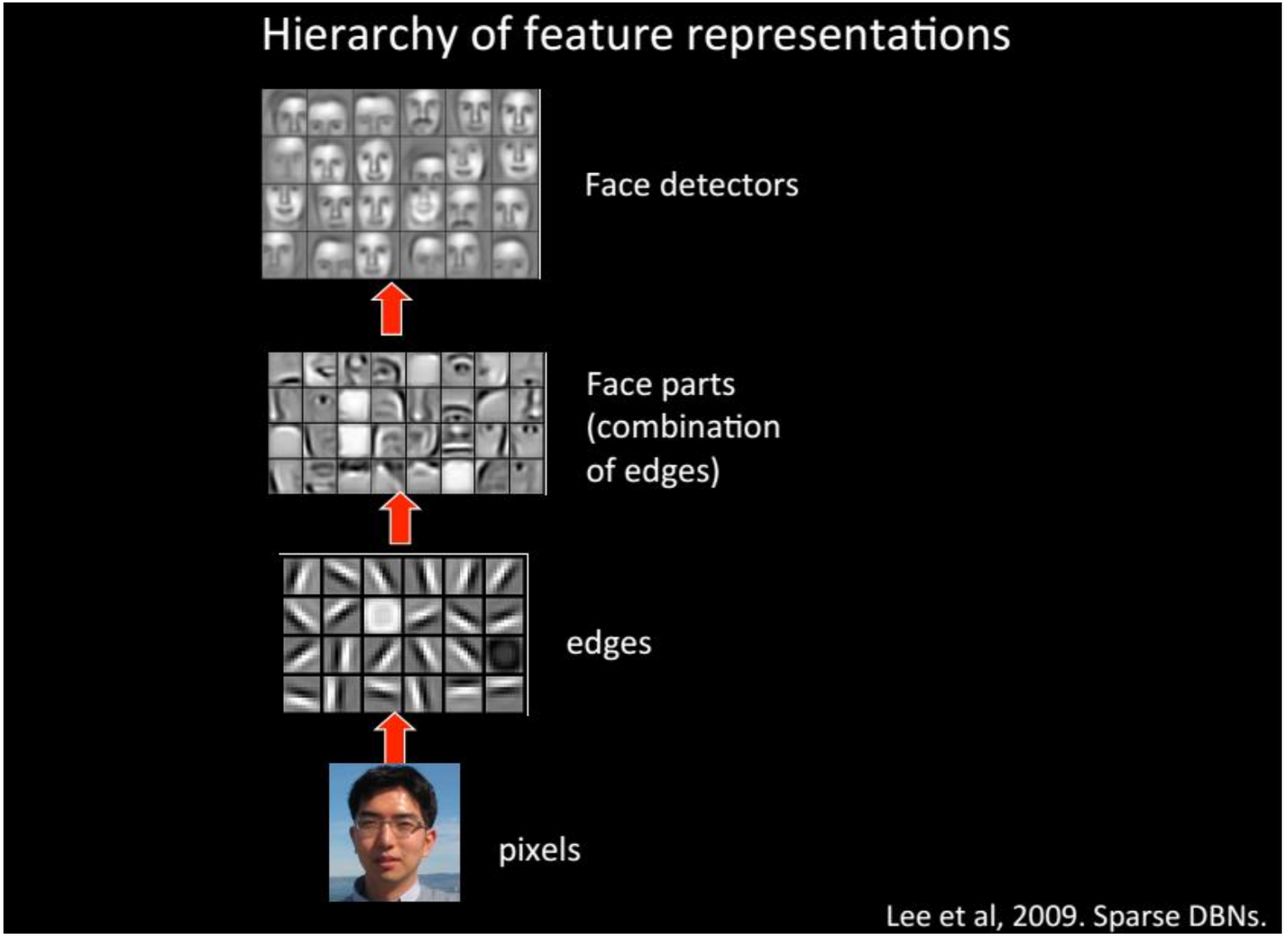
What is convolution



Classical NN for image is convolution with image size kernel

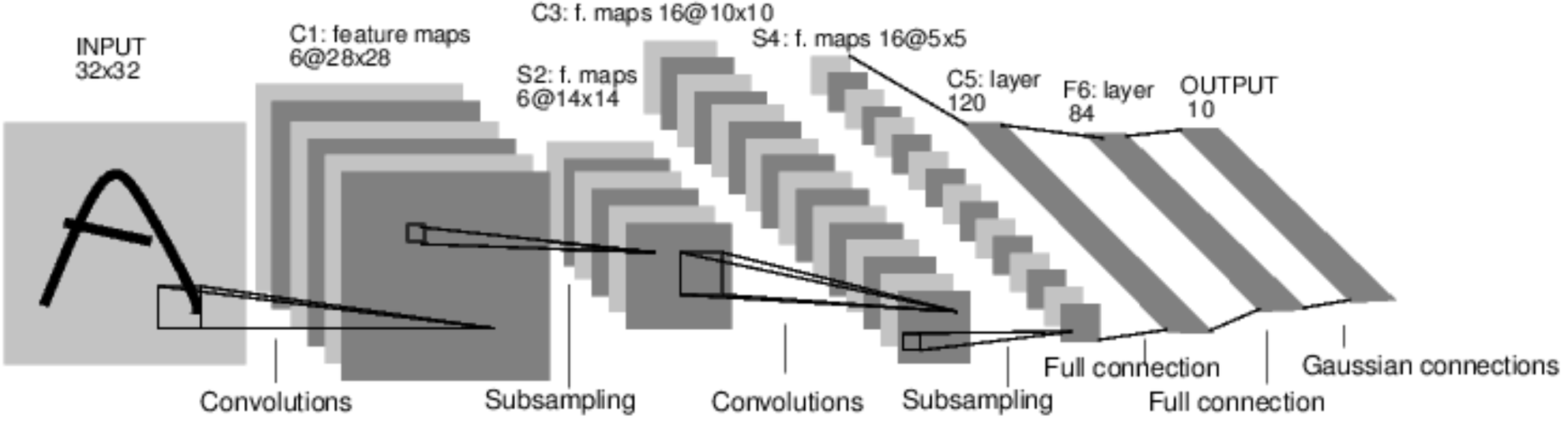
https://developer.apple.com/library/ios/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html

DEEP LEARNING IS HIERARCHICAL REPRESENTATION LEARNING



Quoc.V.Le et.al.,2011. Building high-level features using large scale unsupervised learning

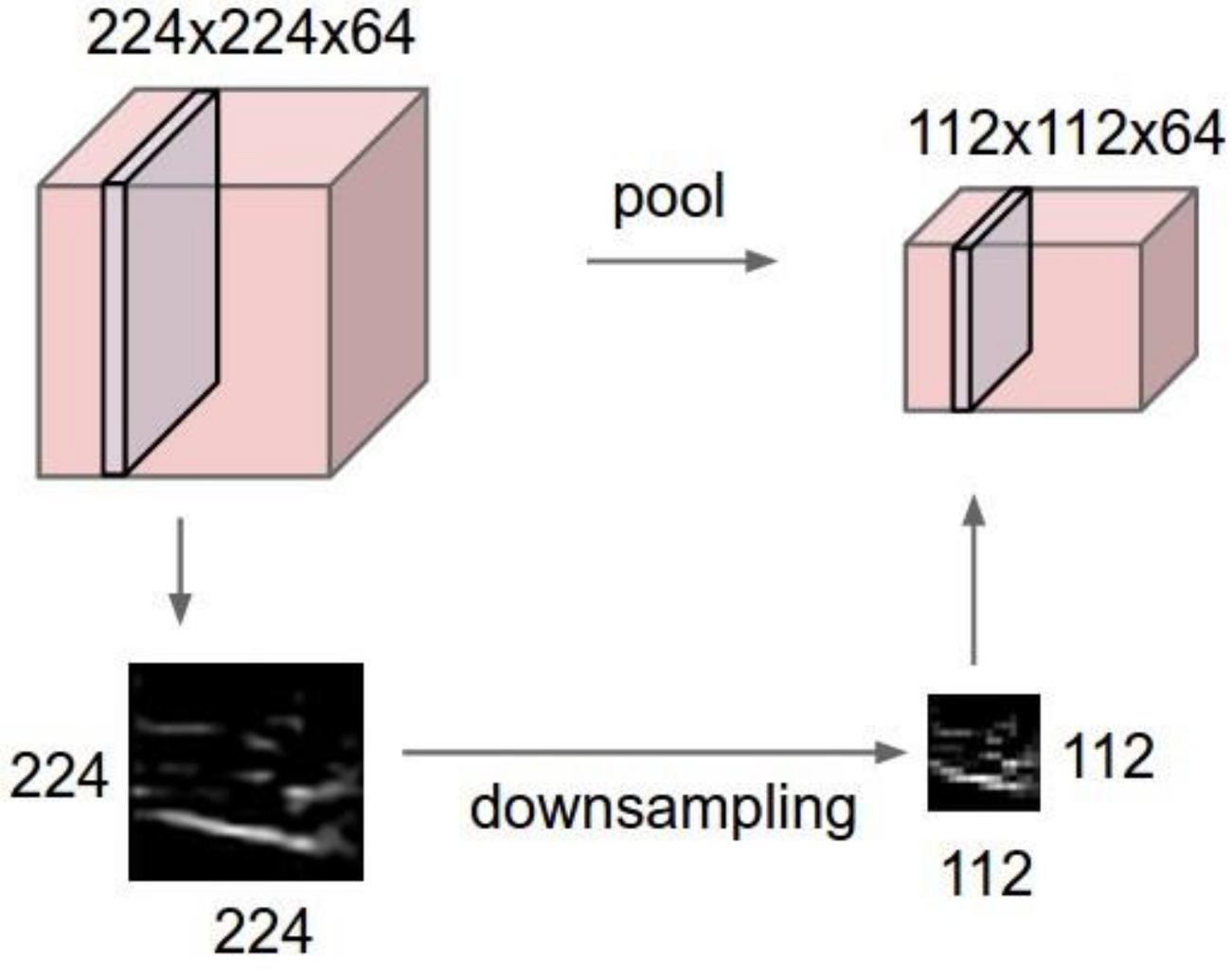
TYPICAL CNN STRUCTURE (LENET-5)



- (Conv-ReLU-Pool)xN Softmax. Simple
- (Conv-ReLU)xN-Pool- (Conv-Relu)x2N-Pool….Softmax. Popular.
- Some Inception arch. Have fun :)

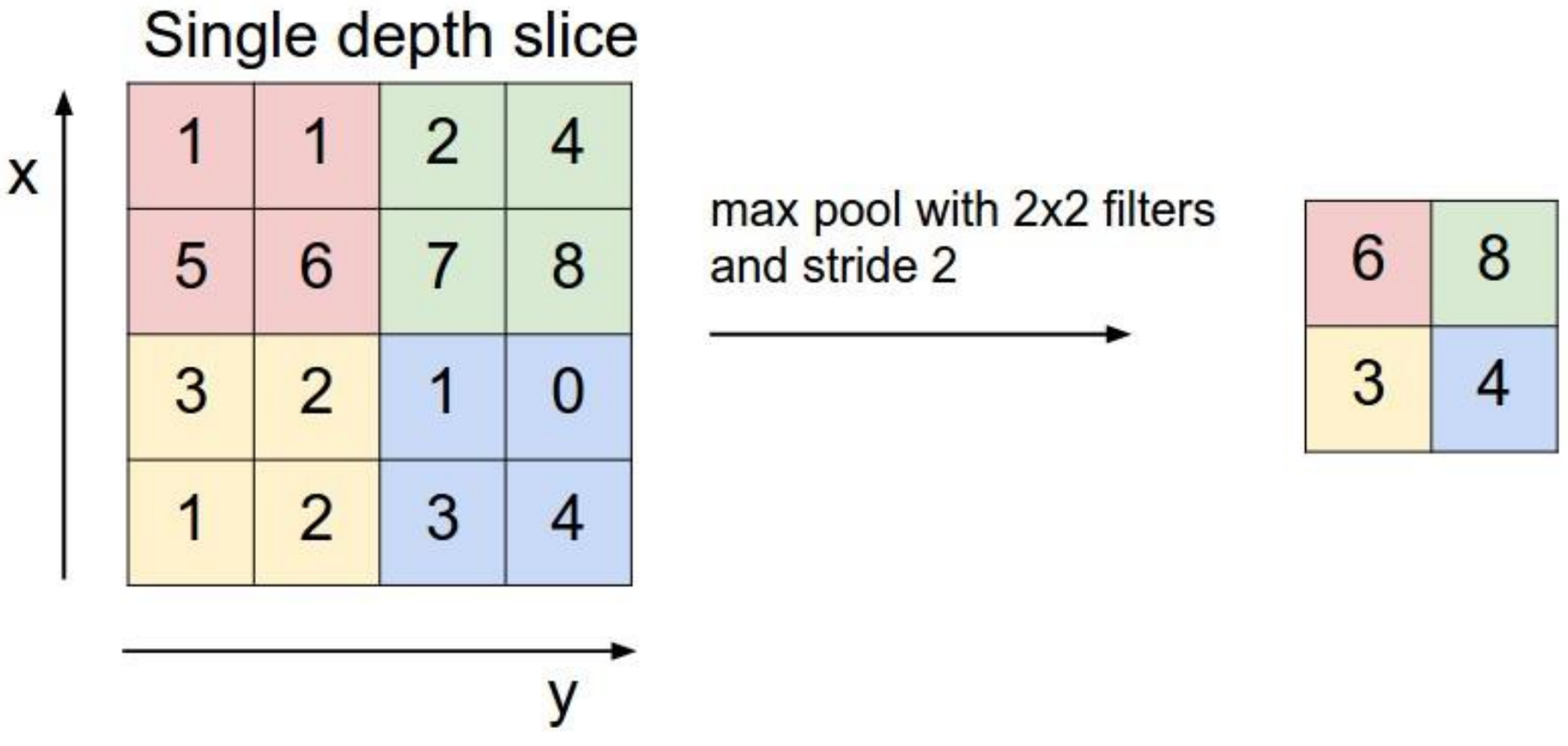
http://eblearn.sourceforge.net/lib/exe/lenet5.png

POOLING



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

MAX POOLING

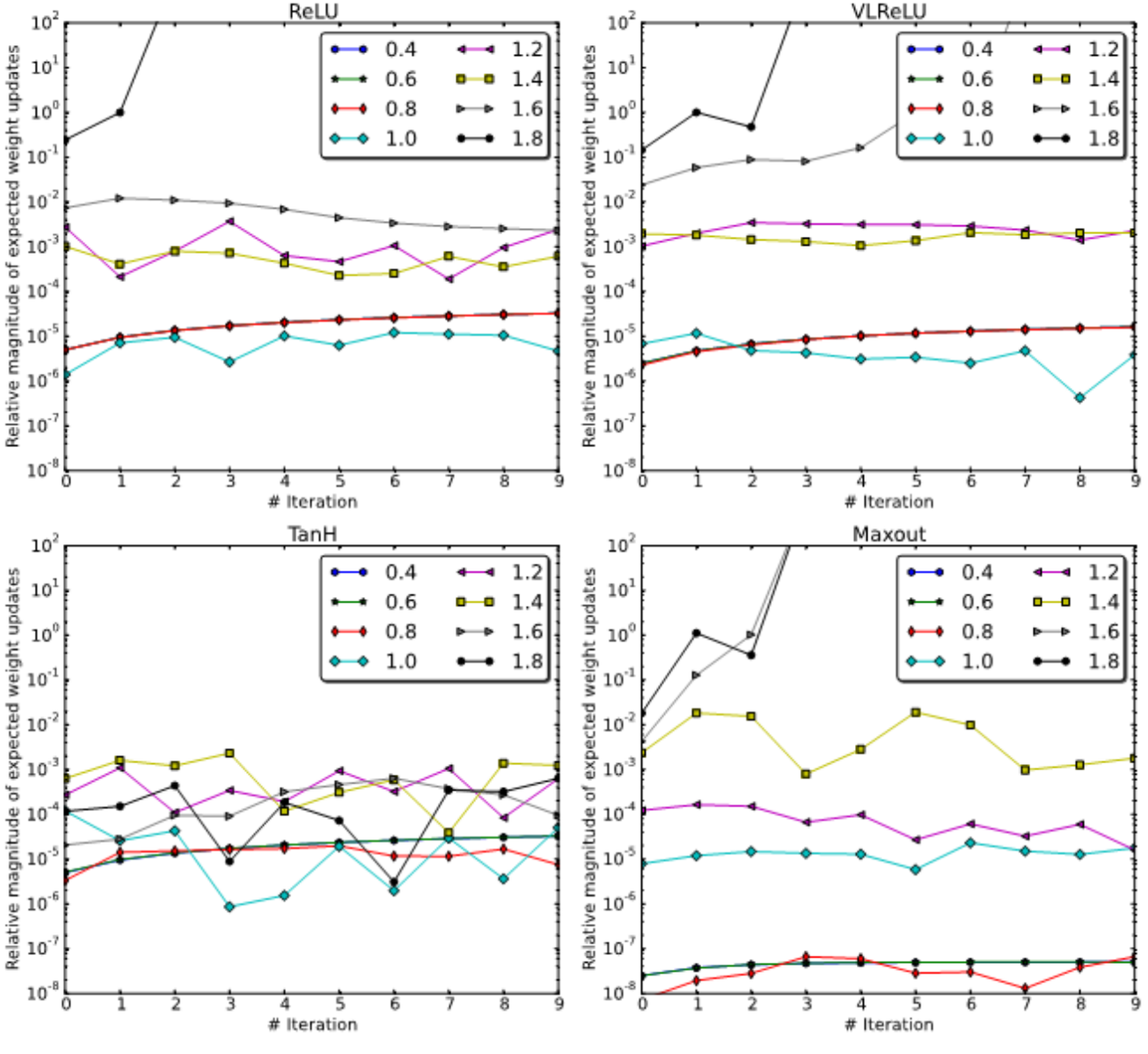


http://cs231n.github.io/convolutional-networks/

CNNS: Important details/concepts

- weights init
- dropout
- batch normalization
- data augmentation
- padding

WEIGHTS INITIALIZATION



- Preserve var=1 of all layers output.

How?

There are lots of papers with variants

Mishkin and Matas. All you need is a good init. ICLR, 2016

Weights initialization

- Gaussian noise with some coefficient:
  - Xavier: $n_l Var[w_l] = 1, \quad \forall l.$
  - He (0.5 * Xavier for ReLU)
- Orthonormal (Saxe et.al. 2013)
- Data-dependent: LSUV

Algorithm 1 Layer-sequential unit-variance orthogonal initialization. L – convolution or full-connected layer, W_L - its weights, B_L - its output blob., Tol_{var} - variance tolerance, T_i – current trial, T_{max} – max number of trials.

Pre-initialize network with orthonormal matrices as in Saxe et al. (2014)
for each layer L **do**
  while $|Var(B_L) - 1.0| \geq Tol_{var}$ and $(T_i < T_{max})$ **do**
    do Forward pass with a mini-batch
    calculate $Var(B_L)$
    $W_L = W_L \,/\, \sqrt{Var(B_L)}$
  end while
end for

Mishkin and Matas. All you need is a good init. ICLR, 2016

BATCH NORMALIZATION

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

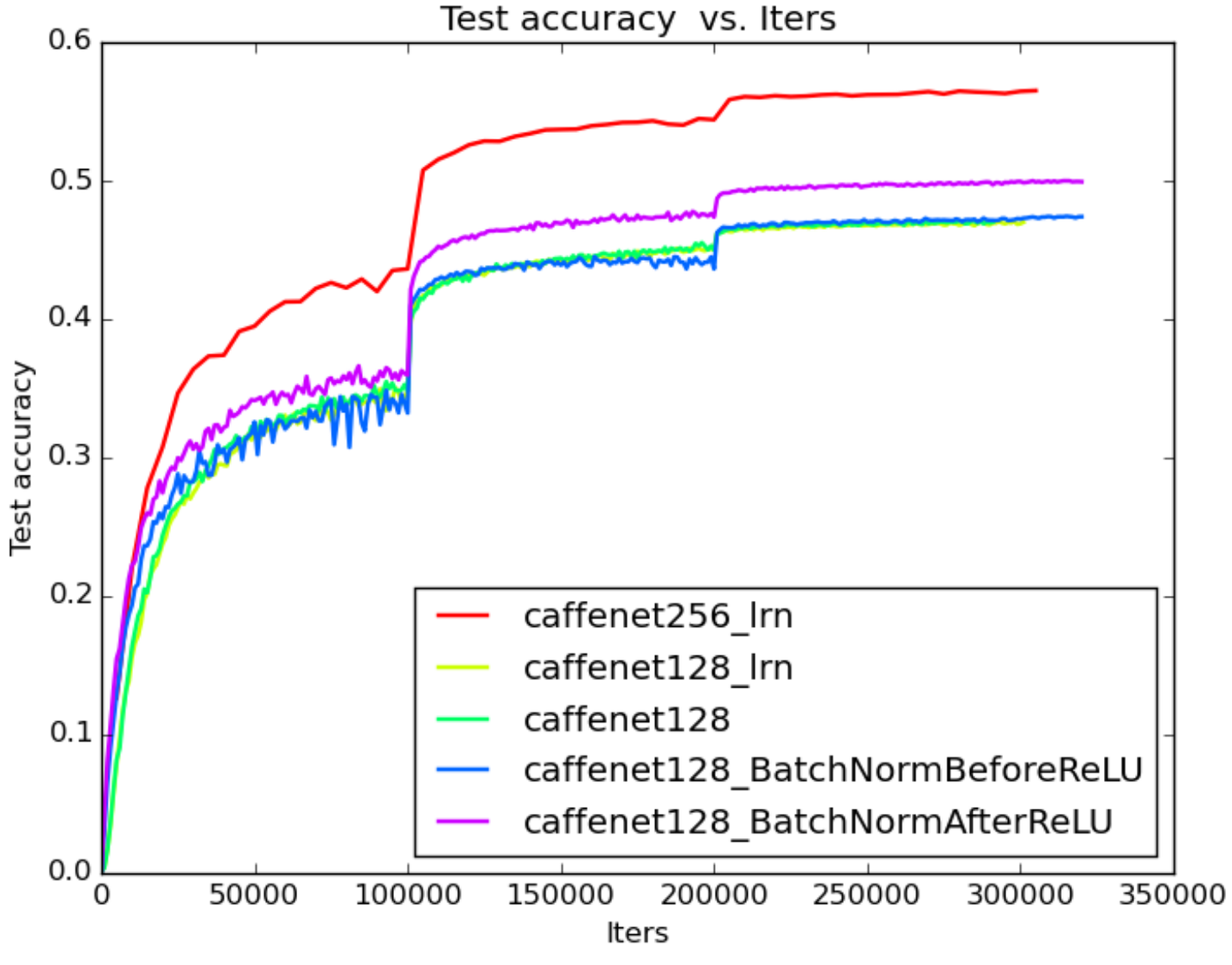
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

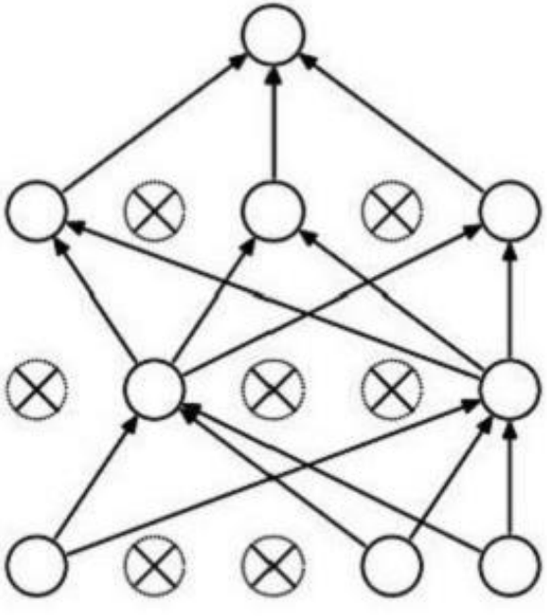
Ioffe et.al 2015

Batch normalization



DROPOUT

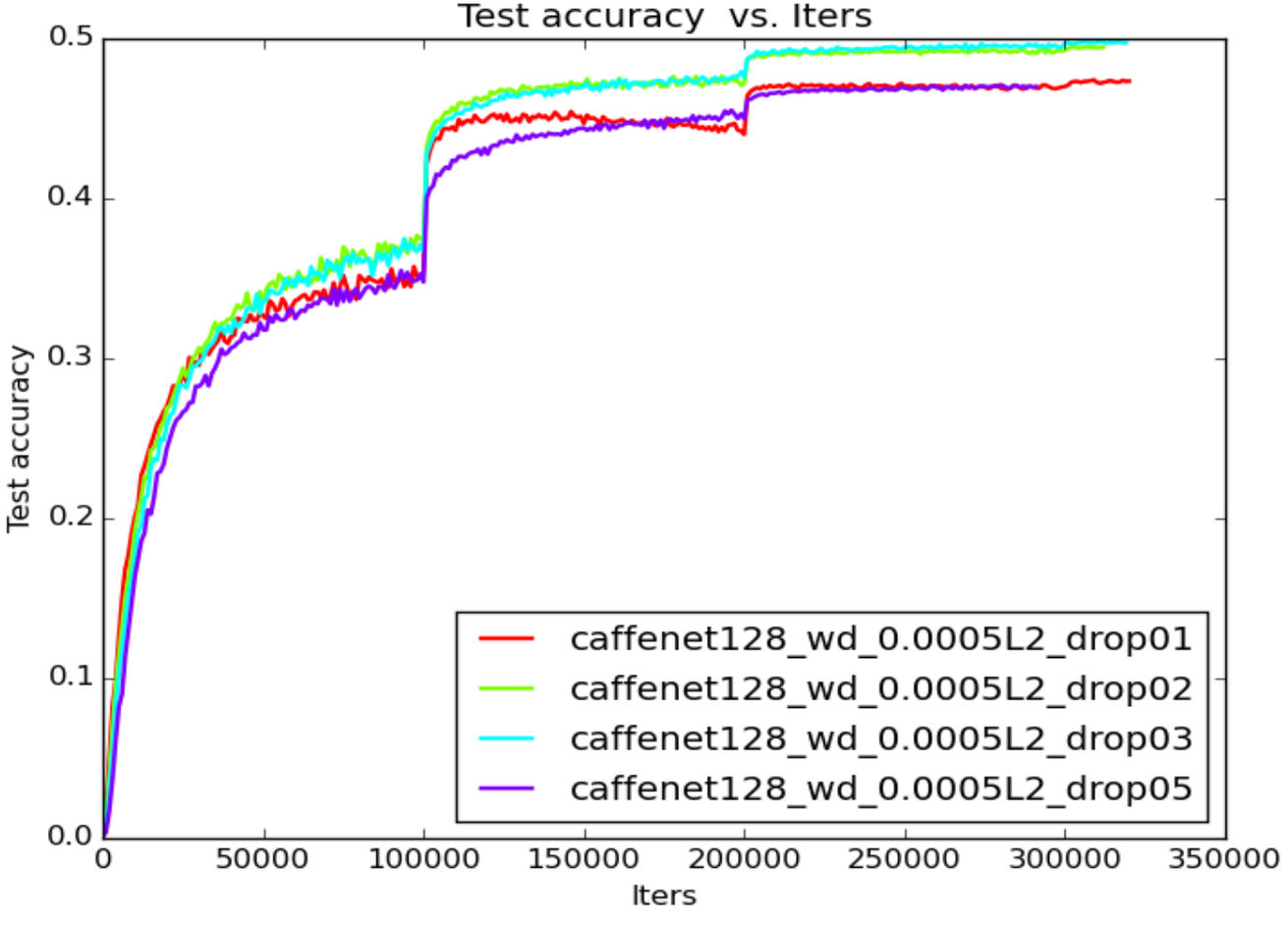
Dropout



Forces the network to have a redundant representation.

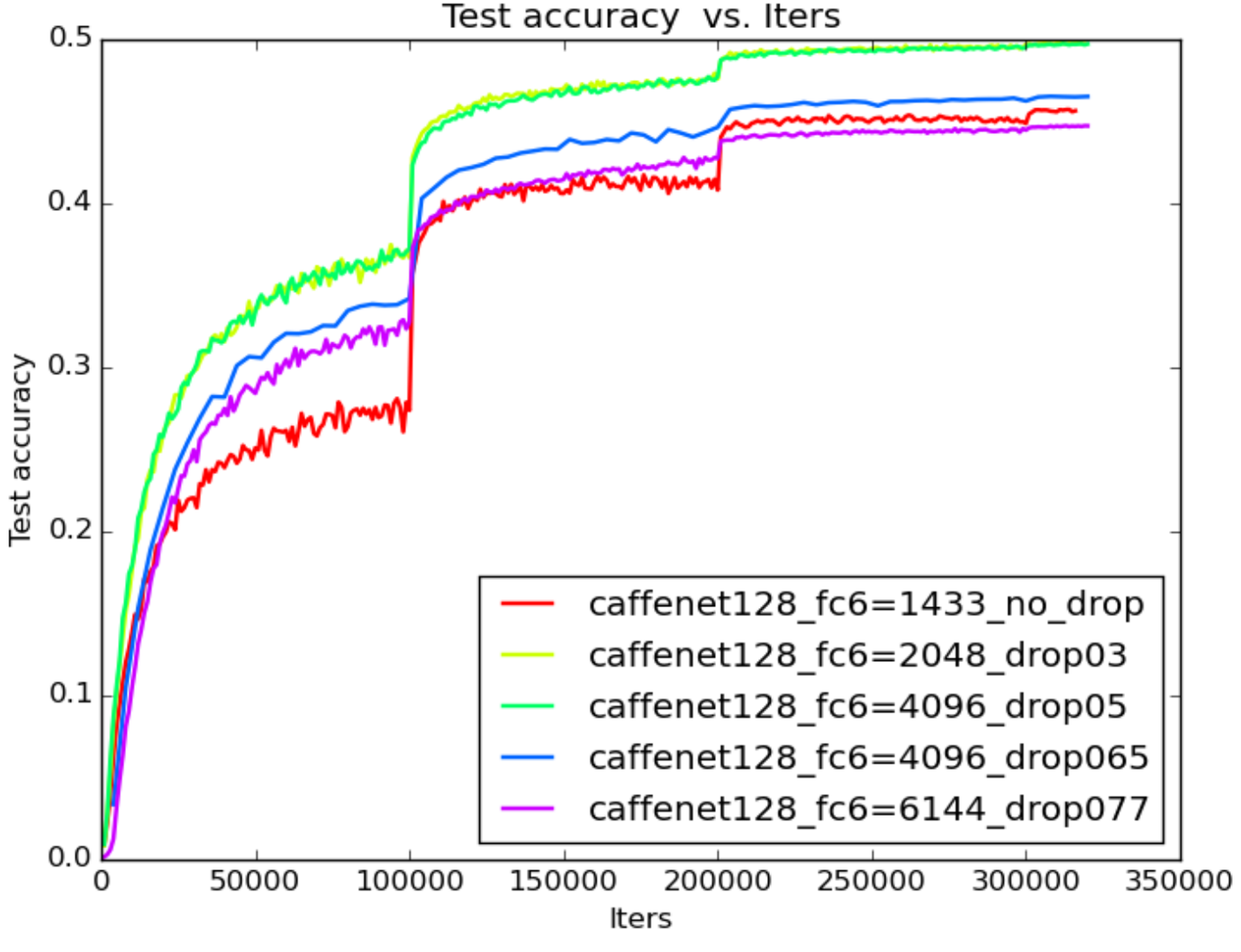
Dropout

- Play with rates. 0.5 is rarely optimal choice (but often good)



Dropout

- Dropout_rate * width = constant – doesn`t work!

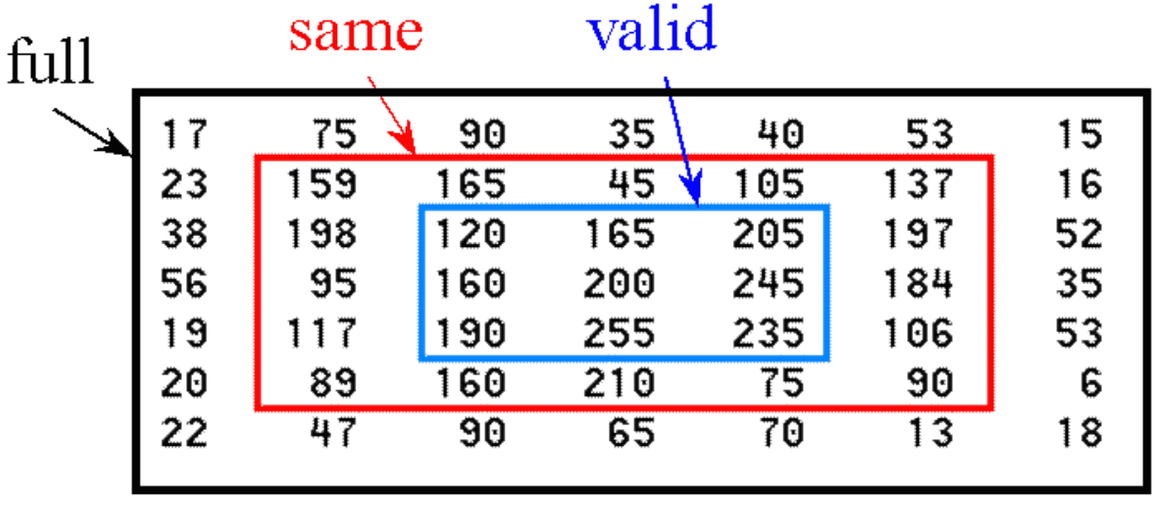


Data augmentation

- Common (helps 99% cases):
  - Random crop: e.g., 227x227 from 256x256 px (AlexNet)
  - Horizontal mirror
- Dataset dependent:
  - Random rotation
  - Affine transform
  - Random scale
  - Color augmentation
  - Noise input
  - Thin plate deformation
  - Unleash your imagination

Padding. Valid and same convolution

$$\begin{pmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix} * \begin{pmatrix} 1 & 3 & 1 \\ 0 & 5 & 0 \\ 2 & 1 & 2 \end{pmatrix}$$



Same = padding with zeros by ½ kernel size.
The most common choice

http://www.johnloomis.org/ece563/notes/filter/conv/convolution.html

PADDING

- Padding:
  - Preserving spatial size, not “washing out” information
  - Dropout-like augmentation by zeros

Caffenet128

with conv padding: **47**% top-1 acc

w/o conv padding: **41**% top-1 acc.

It is huge difference

Notable approaches & Applications

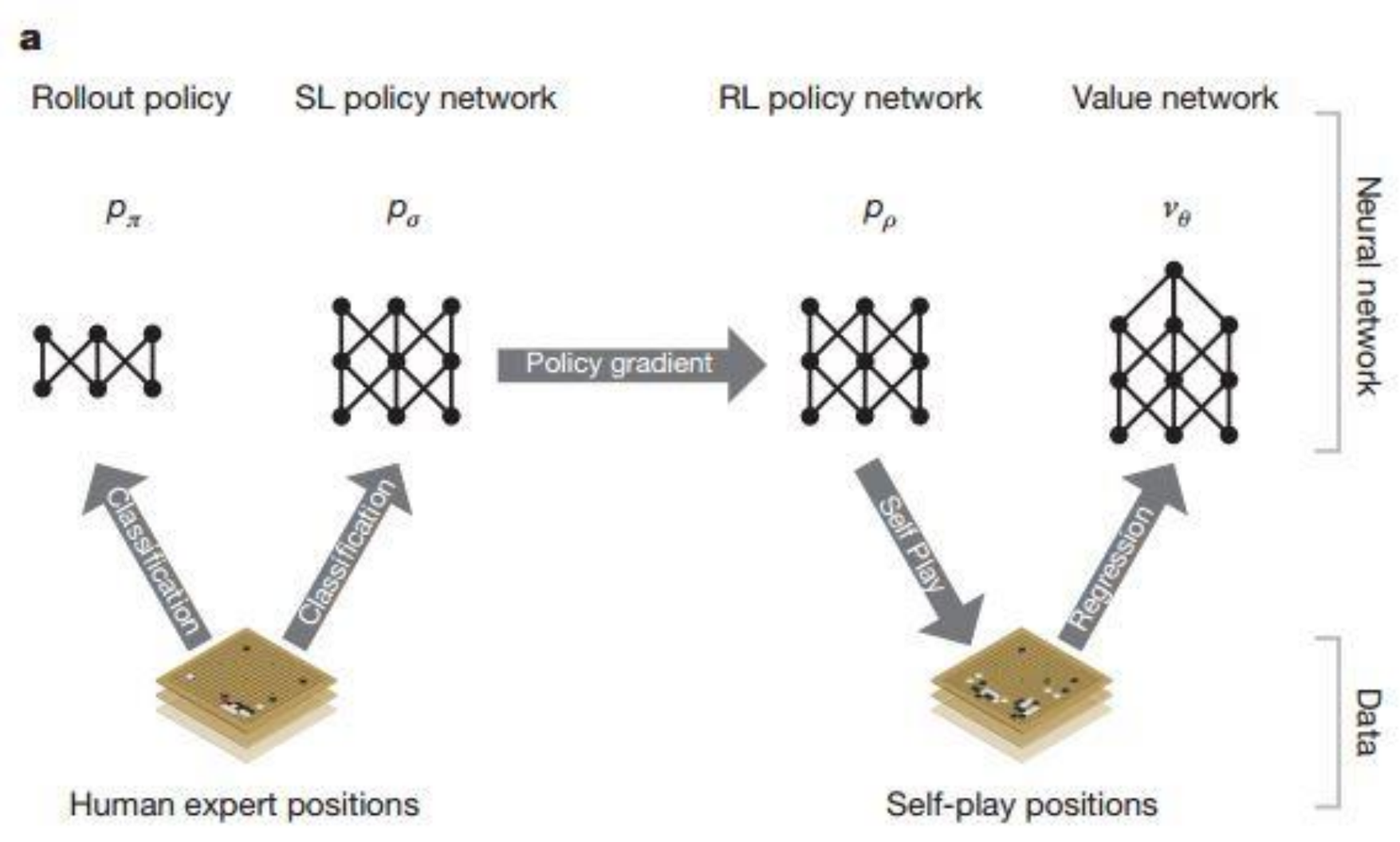
How to do – let`s go to whiteboard

- **Image retrieval** Babenko et. al (2014)
- **Person identification** Chopra et. al 2006
- **Ranking** Wang et.al 2014
- Playing games. Atari (2013) Go (2016)
- Text generation https://github.com/karpathy/char-rnn
- Image generation Radford et.al 2016
- Action recognition Simonyan et.al 2014
- **Anomaly detection** https://www.youtube.com/watch?v=ds73ULGjnpc&feature=youtu.be
- Translation Cho et al 2014
- Fraud detection at PayPal http://university.h2o.ai/cds-lp/cds02.html

Deep learning applications

- Alpha Go :)
- Image recognition
- Speech Recognition. Cortana, Siri
- Translation
- Anomaly detection
- Fraud detection
- Video recognition
- Robotics
- Recommendation systems
- DNA, biology, and more..

AlphaGo



Mastering the game of Go with deep neural networks and tree search
Silver et.al 2016

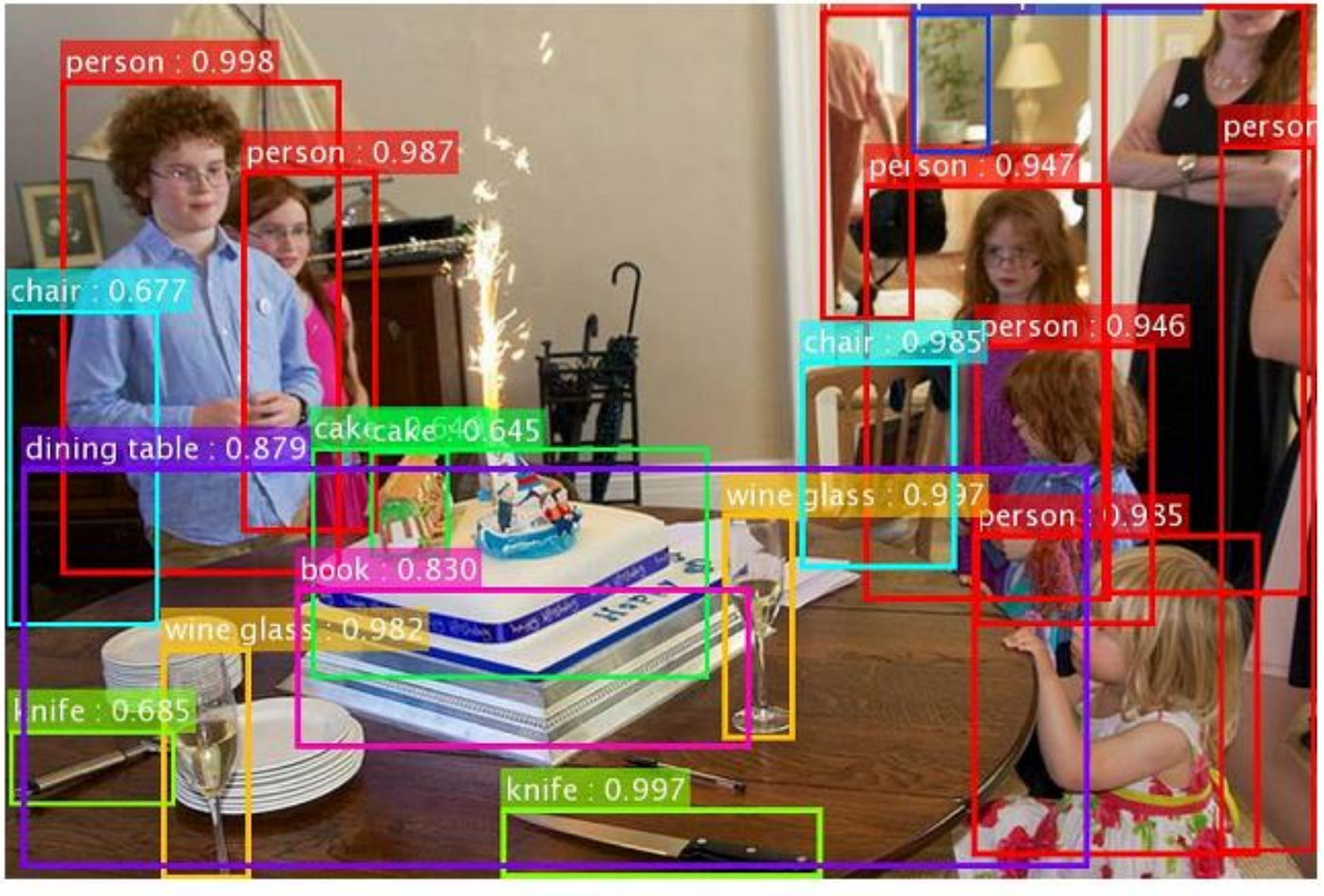
IMAGE CLASSIFICATION

Select all dogs. Our assignment…almost :)

State-of-art since 2012. Krizhevsky et.al 2012
Superhuman level an ImageNet classification since 2015.
He et.al 2015, Szegedy et.al 2015

OBJECT DETECTION

Microsoft Research



Our results on COCO – too many objects, let's check carefully!

*the original image is from the COCO dataset

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.
Shaoqing Ren, Kaiming He, Ross Girshick, & Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". NIPS 2015.

ICCV15 International Conference on Computer Vision

Speech recognition

- Cortana
- Siri
- OK, Google

Figure 1. CNN architecture for speech recognition

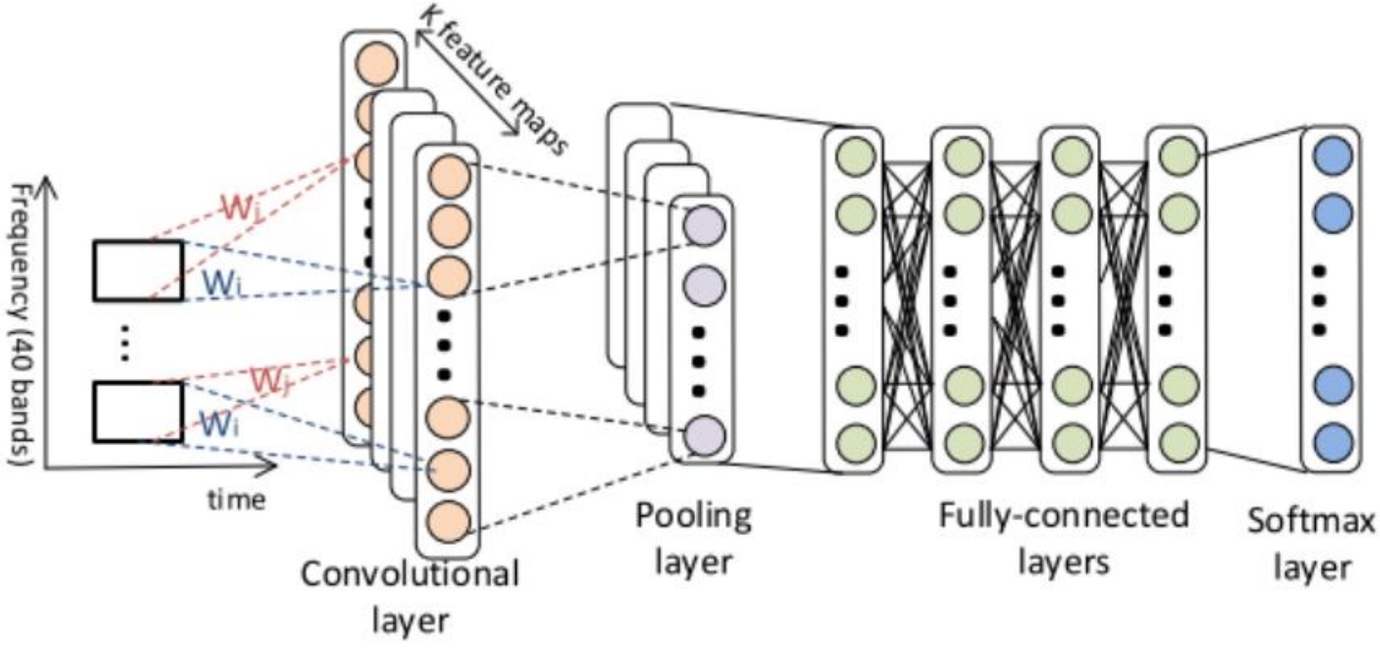
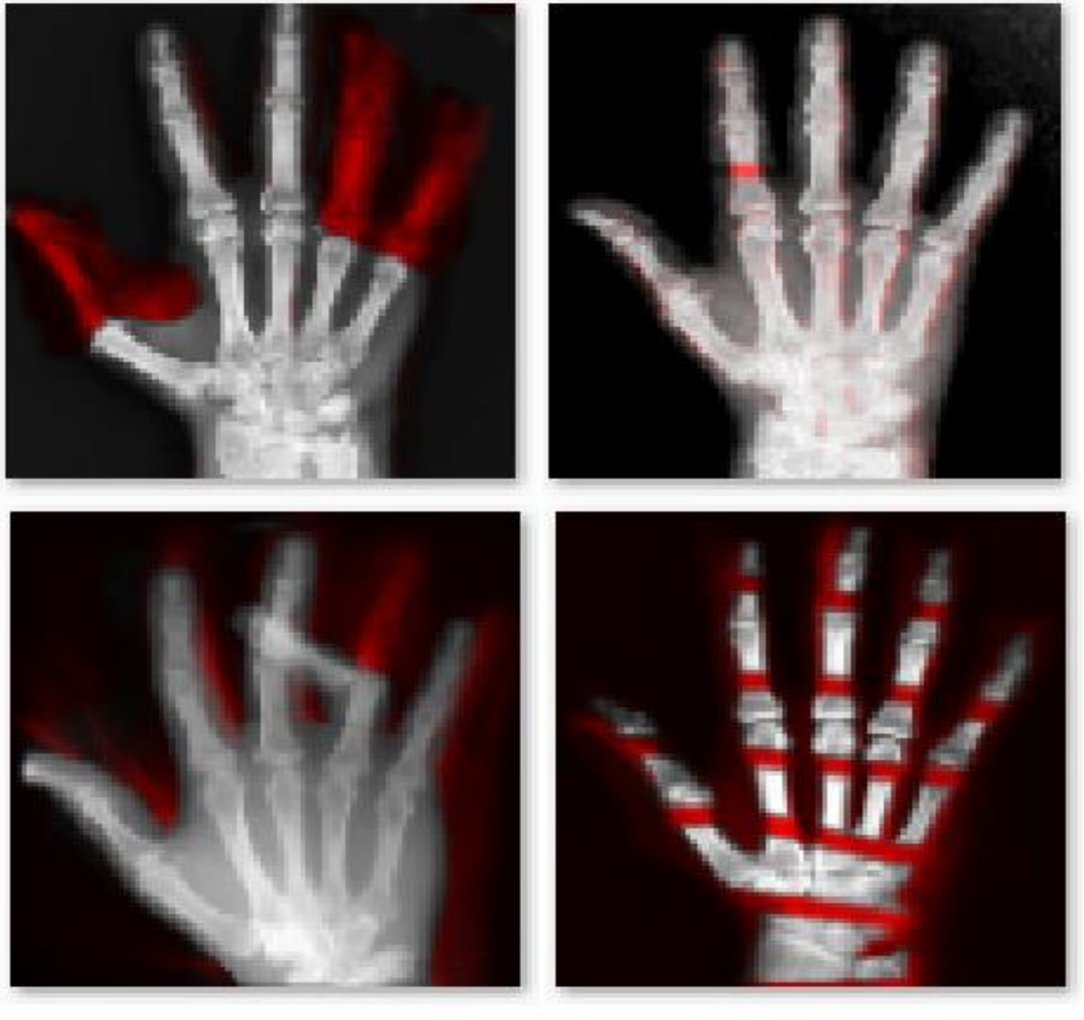


Figure from Huang et.al. 2015.

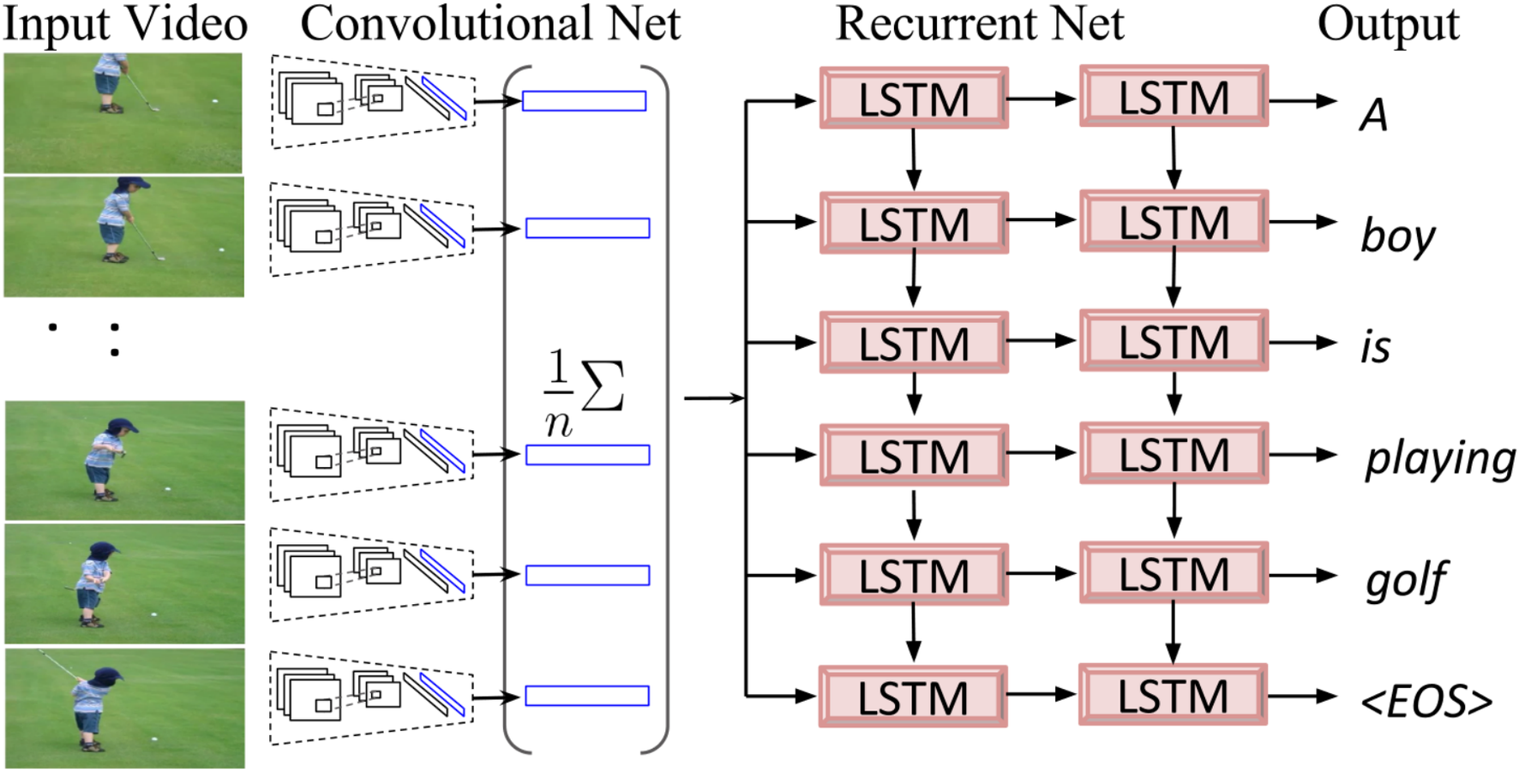
Anomaly detection

Virtual doctor: Image Recognition for Healthcare

Igor Kostiuk



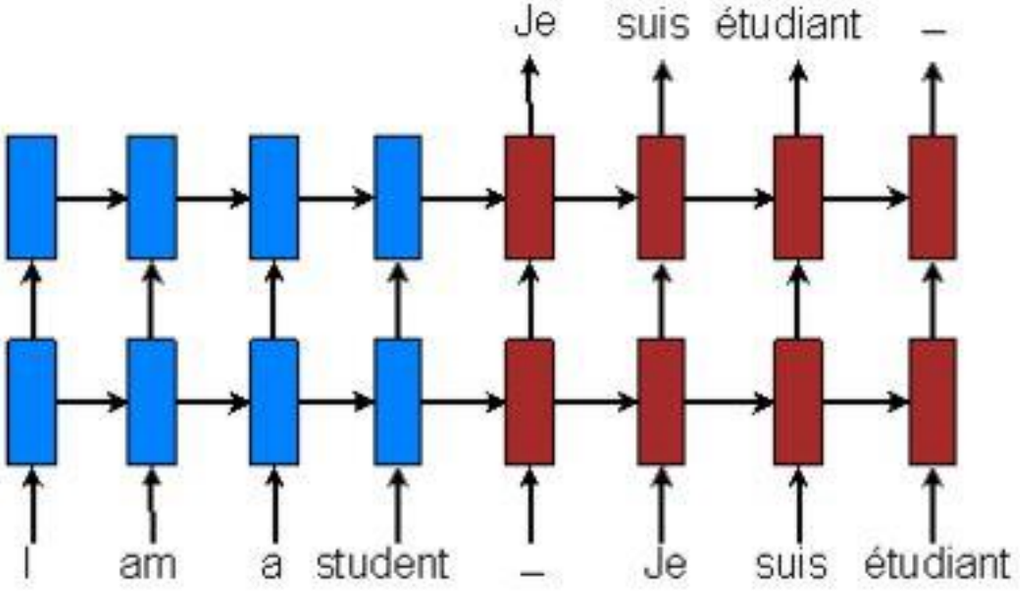
VIDEO CAPTIONING



Translating Videos to Natural Language Using Deep Recurrent Neural Networks. Venugopalan et.al. 2015

TEXT TRANSLATION

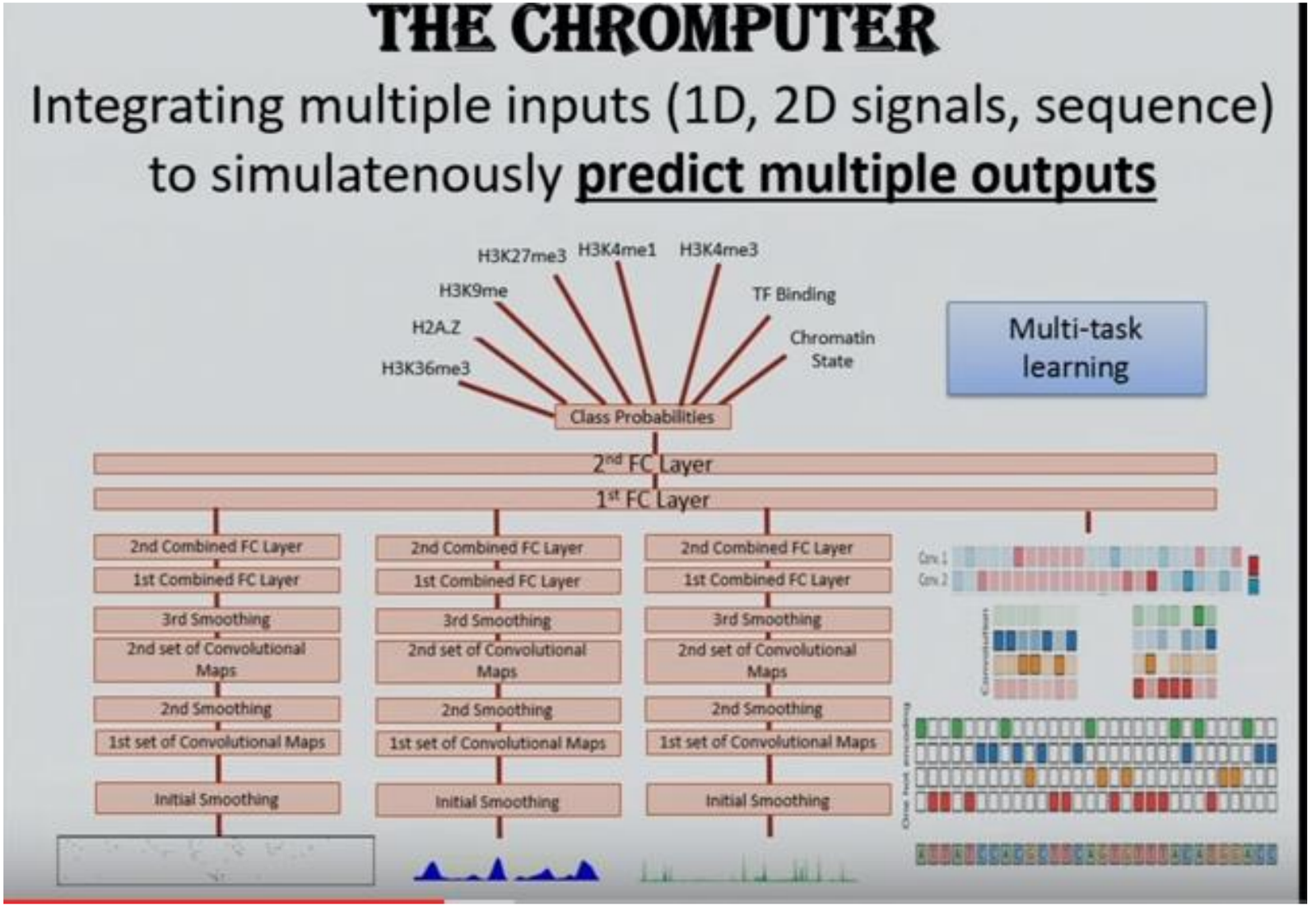
Neural Machine Translation (NMT)



- RNNs trained end-to-end (Sutskever et al., 2014).

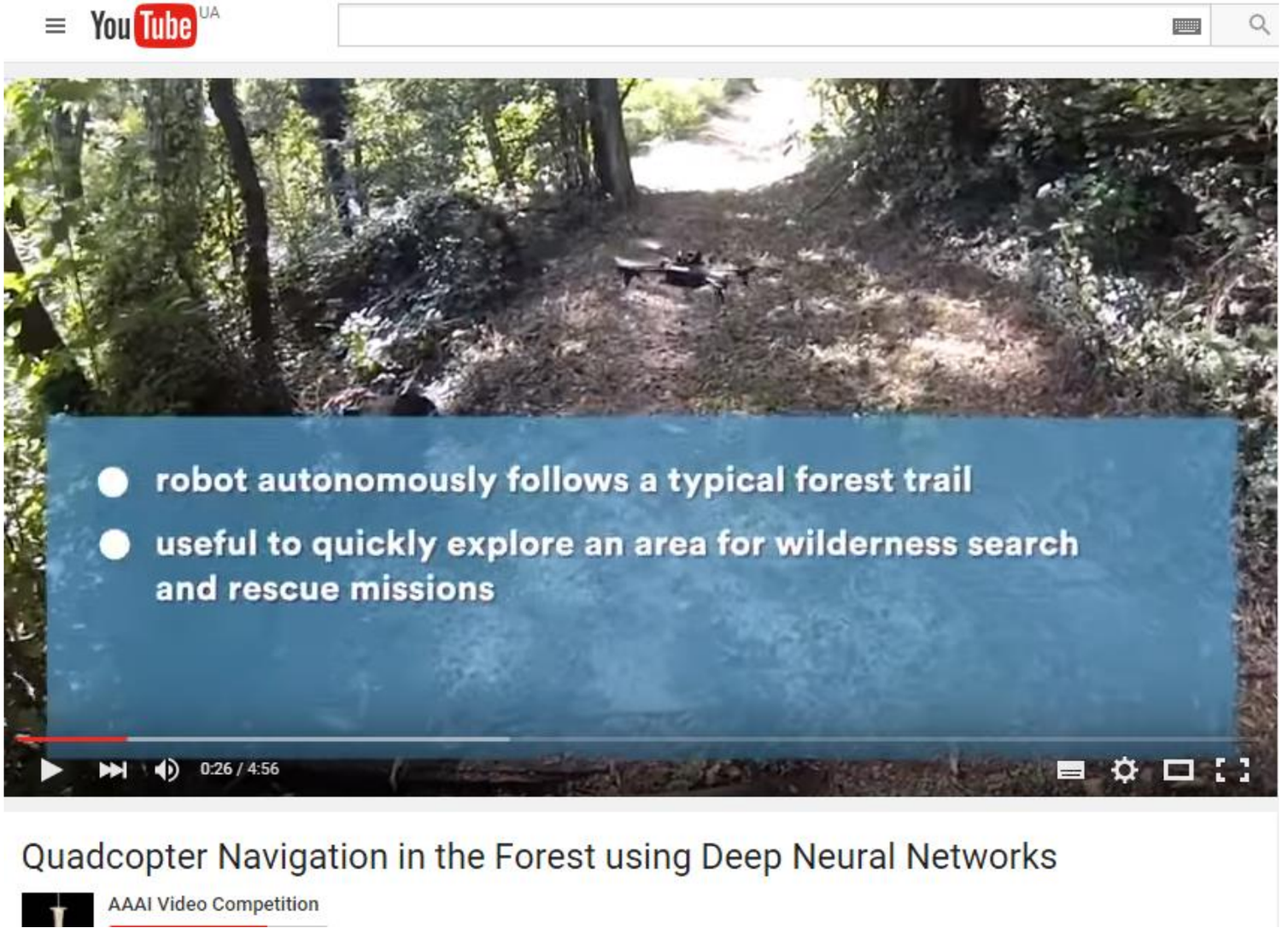
From [Bahadanau et al., 2015] slides at ICLR 2015.

Deep Learning Frameworks for Regulatory Genomics and Epigenomics



https://www.youtube.com/watch?v=2vpKB3j-OY0

ROBOTICS: NAVIGATION



https://www.youtube.com/watch?v=umRdt3zGgpU

Fraud Detection

Experimental Design

10 million rows/1500 features (60% training; 20% validation; 20% test)

Parameter	Range
# of hidden layers	2, 4, 6, 8
# of neurons	200, 300, 400, 500, 600, 700
activation function	Rectifier; Tanh; Maxout; RectifierWithDropout
feature subset	All, subset1 – subset7
test data set	All, week4 – week8
L1/L2 regularization	0 - 1
epoch	500

Results

Best performance with 6 layers

How much depth is required?

# of hidden layers (Rectifier, 2 layer, 200 neurons, 500 epoch, L1/L2 = 0)	Area Under ROC Curve (AUC)
2	0.762
4	0.821
6	0.839
8	**0.839**

As simple classification
http://www.slideshare.net/0xdata/paypal-fraud-detection-with-deep-learning-in-h2o-presentationh2oworld2014

Neural Networks Summary

+ Flexible technique; can be employed both for classification and regression
+ Multiclass classification
+ Outputs probabilities of all classes, giving access to confidence of prediction
– Interpretability
– Objective function has multiple extrema and gradient-based learning is not guaranteed to find the global minimum