Training neural network

zsc@

Jan. 2017

Neural Network

Machine Learning as Optimization

- Supervised learning
  - θ is the parameter
  - $\hat{y}$ is output, X is input
  - y is ground truth
  - d is the objective function

$$\min_\theta d(y, \hat{y})$$
$$\text{where } \hat{y} = f(X, \theta)$$

- Unsupervised learning
  - some oracle function **r**: low rank, sparse, K

$$\min_\theta r(\hat{y})$$
$$\text{where } \hat{y} = f(X, \theta)$$

Machine Learning as Optimization

- Regularized supervised learning
  - $\min_\theta d(y, \hat{y}) + r(\hat{y})$
    where $\hat{y} = f(X, \theta)$
- Probabilistic interpretation
  - *d* measures conditional probability
  - *r* measures prior probability
  - Probability approach is more constrained than the optimization-approach due to normalization problem
    - Not easy to represent uniform distribution over [0, \infty]

$$d(y, \hat{y}) = -\log p(y|\hat{y})$$
$$r(\hat{y}) = -\log p(\hat{y})$$
$$\Rightarrow d(y, \hat{y}) + r(\hat{y}) = -\log p(y)$$

Gradient descent

$$\min_\theta d(y, \hat{y}) + r(\hat{y})$$
$$\text{where } \hat{y} = f(X, \theta)$$

- Can be solved by an ODE: $\dot{\theta} = -\frac{\partial C(\theta(t))}{\partial \theta}$
  - Discretizing with step length λ we get gradient descent with learning rate λ
  - $\dot{\theta} \approx \frac{\theta_{t+\lambda} - \theta_t}{\lambda} \xrightarrow{\text{Derive}} \theta_{t+\lambda} = \theta_t - \lambda \frac{\partial C(\theta)}{\partial \theta}$
- Convergence proof

$$\frac{\mathrm{d}C(\theta)}{\mathrm{d}t} = \frac{\partial C(\theta)}{\partial \theta} \frac{\mathrm{d}\theta}{\mathrm{d}t} = -\left(\frac{\partial C(\theta)}{\partial \theta}\right)^2 \leq 0$$

Trapping in Local Minimum

$f(x) = (x + 1/x - 1)^2$

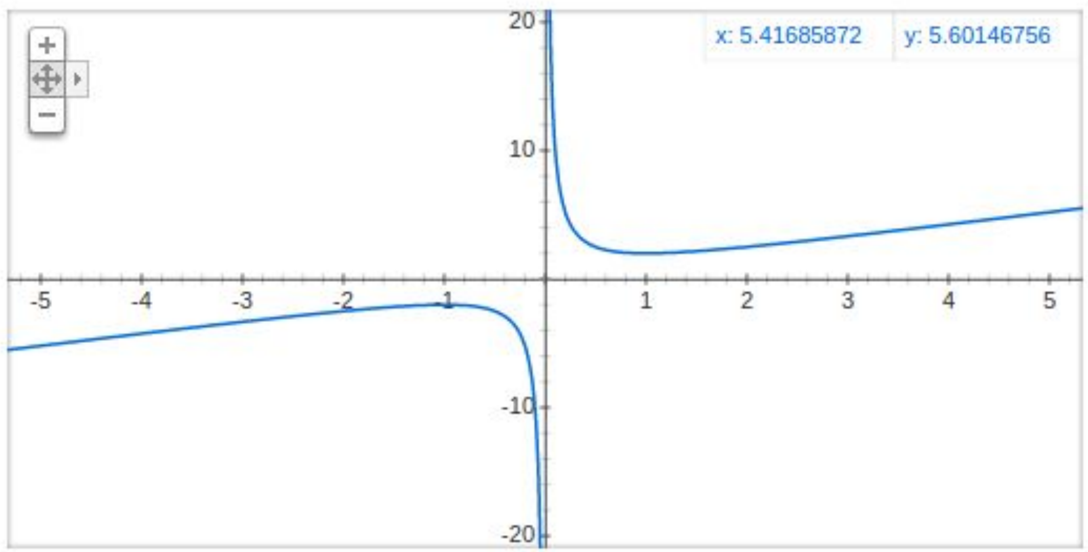
$g(x) = 2(x+1/x-1)(1-x^{-2})$

$g(-1) = 0$，而$f(-1)$非最优

$g(-1/2) = 21$

$g(-3/2) = -3.52$

“x+1/x”的图表



Discretized Gradient Descent may not converge

- Let learning rate be 1, $f(x) = x^2$
  - $f'(1) = 2$, $f'(-1) = -2$
  - $x_{t+1} = x_t - f'(x_t)$
    - $x_{t+1} = -x_t$, will oscillate between 1 and -1.
  - Diverges if **lr > 1**: $x_{t+1} = -(2\,\mathbf{lr} - 1)\,x_t$
- No matter how small is the learning rate
  - If learning rate is 10^{-6}, i.e. $x_{t+1} = x_t - 10^{-6} f'(x\_t)$
  - When $f(x) = 10^6 x^2$, x will oscillate between 10^{-6} and -10^{-6}.

Beyond Gradient Descent: second-order method

- $f(x) = 10^6 x^2$
- Can be solved by Newton's method trivially
  - $x_{t+1} = x_t - H^{-1} f'(x)$
  - $H^{-1} f'(x) = 5e\text{-}7 * 1e6 * 2x = x$
    - $x_{t+1} = 0$
    - Get to optimum in one-step
- A little regularization will stop divergence
- We should use second-order method for last-round tuning?

$$|f(x) - f(y) - \nabla f(y)^T (x-y)|$$

$$= \left| \int_0^1 \nabla f(y + (x-y)t)^T (x-y)\,dt - \nabla f(y)^T (x-y) \right|$$

$$= \left| \int_0^1 \nabla f(y + (x-y)t)^T (x-y)\,dt - \int_0^1 \nabla f(y)^T (x-y)\,dt \right|$$

$$\le \int_0^1 \left| (\nabla f(y + (x-y)t)^T - \nabla f(y)^T)(x-y) \right| dt$$

(triangle inequality)

$$\le \int_0^1 \| \nabla f(y + (x-y)t)^T - \nabla f(y)^T \| \, \|x-y\| \, dt$$

(Cauchy - Schwartz)

$$\le \int_0^1 \beta \, \|(x-y)t\| \, \|x-y\| \, dt \quad (\beta\text{-smooth})$$

$$= \frac{\beta}{2} \|x-y\|^2$$

$$f(x_{t+1}) \le f(x_t) + \nabla f(x_t)^T (x_{t+1} - x_t) + \frac{\beta}{2} \|x_{t+1} - x_t\|^2$$

$$= f(x_t) - \eta \|\nabla f(x_t)\|^2 + \frac{\beta \eta^2}{2} \|\nabla f(x_t)\|^2$$

$$= f(x_t) - \frac{\beta}{2} \eta \left(\frac{2}{\beta} - \eta\right) \|\nabla f(x_t)\|^2$$

只要 $\beta\eta < 2$ 就可下降

$\beta\eta = 1$ 时 bound 最紧

when bound is most tight:

$$f(x_{t+1}) \le f(x_t) - \frac{\eta}{2} \|\nabla f(x_t)\|^2$$

This inequality is still scale-invariant, as \eta \beta = 1 .

Newton's method from Differential Geometry perspective

- First order method does not respect tensor variance
  - adding pseudo-vector to vector

$$X_{t+1} = X_t - \lambda \frac{\partial f(X)}{\partial X}$$

- Hessian corresponds to metric tensor

$$X_{t+1} = X_t - H^{-1} \frac{\partial f(X)}{\partial X}$$

- [Gauss-Newton algorithm]()

$$H \approx \frac{\partial f(X)}{\partial X}^\top \frac{\partial f(X)}{\partial X}$$

But Hessian need not be positive-semidefinite.

$$X_{t+1} = X_t - \frac{\partial f(X)}{\partial X}^+$$

Linear Regression

- $\min_W \|y - Wx\|^2$
  - $\hat{y} = f(X, \theta) = WX$
  - $d(y, \hat{y}) = \|y - \hat{y}\|^2$
  - *x* is input, *\hat{y}* is prediction, *y* is ground truth.
  - ***W*** with dimension (m,n)
  - #param = *m n*, #OPs = *m n*

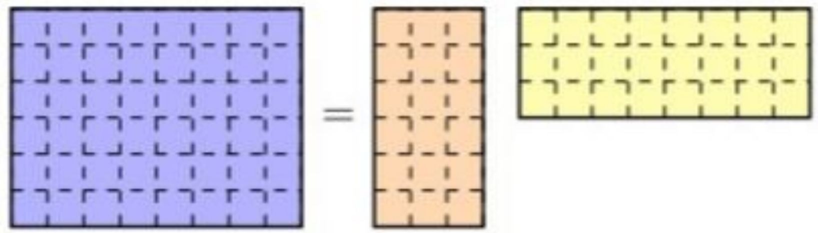
$$y \approx \hat{y} = Wx$$

Linear Regression

- Least square solution
  - For A x = b, we have: $x = A^{+}b + [I - A^{+}A]w$
  - Hence $W = \hat{y} x^{+} + c (I - x x^{+})$
    - x^{+} is pseudo-inverse.
    - c is free variable. Using regularization can suppress it.

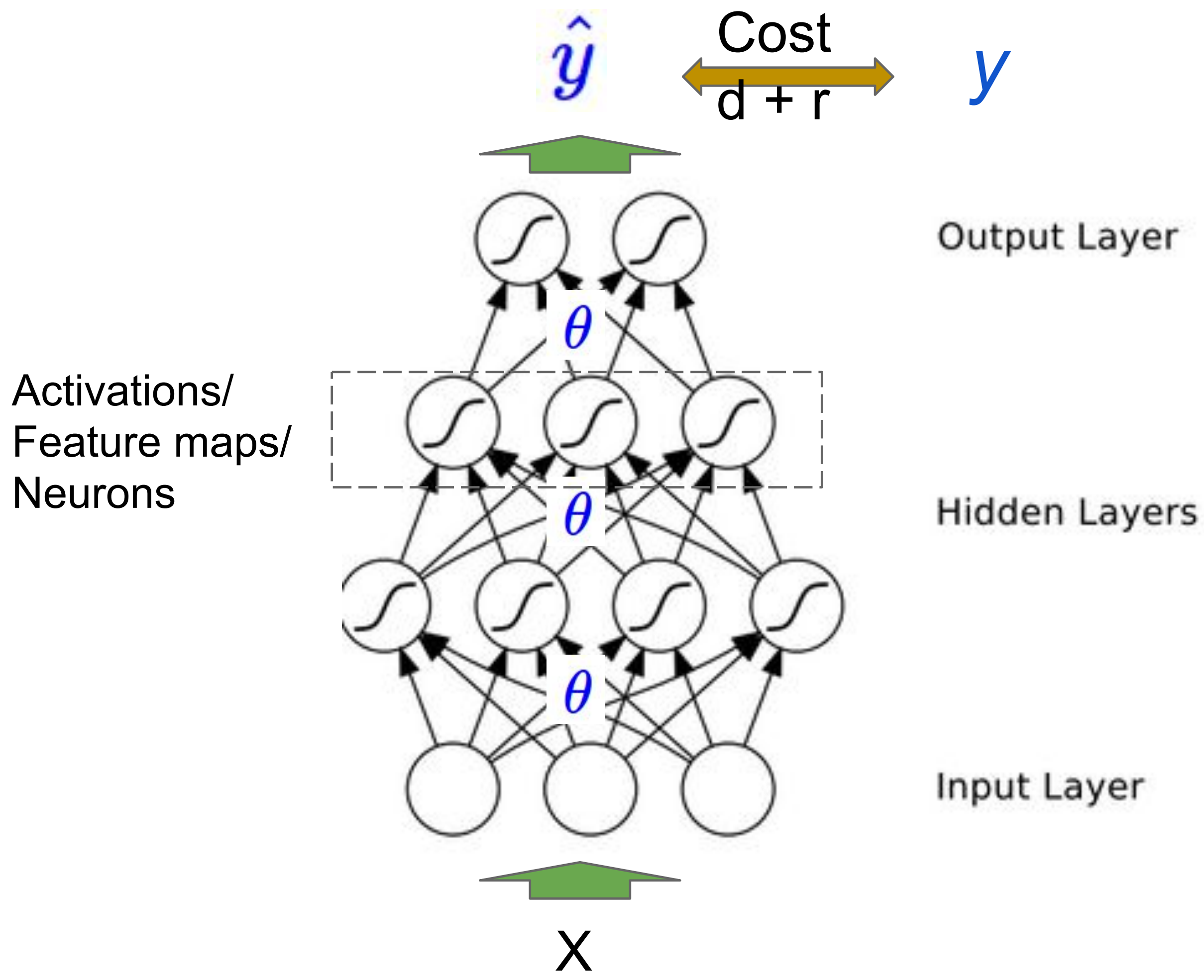
Fully-connected

- $y \approx W_2 f(W_1(x))$
  - In general, will use nonlinearity to increase "model capacity".
  - Make sense if ***f*** is identity? I.e. f(x) = x?
    - Sometimes, if W_2 is ***m*** by ***r*** and W_1 is ***r*** by ***n***, then W_2 W_1 is a matrix of rank ***r***, which is different from a ***m*** by ***n*** matrix.

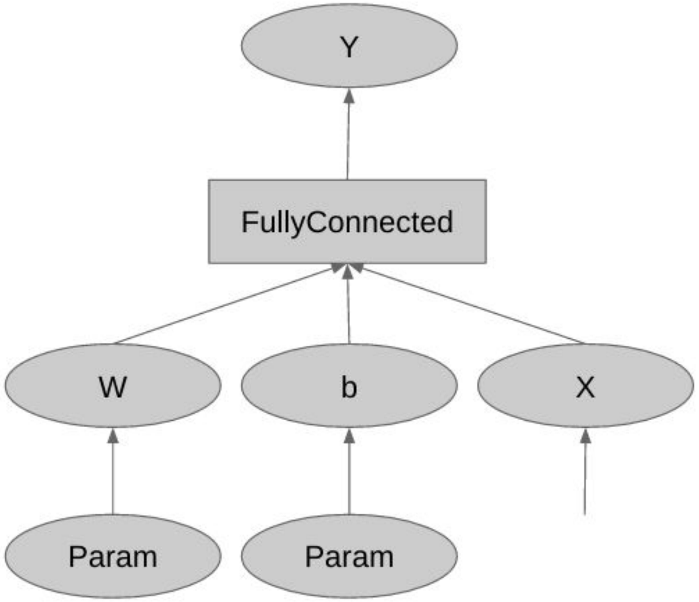


- $y \approx W_3(f(W_2 f(W_1(x))))$
  - Deep learning!

Neural Network



Computation Graph



Chain rule

- For training, need to save all intermediate feature maps.
  - Can compute from scratch instead of saving, especially for deep nets.

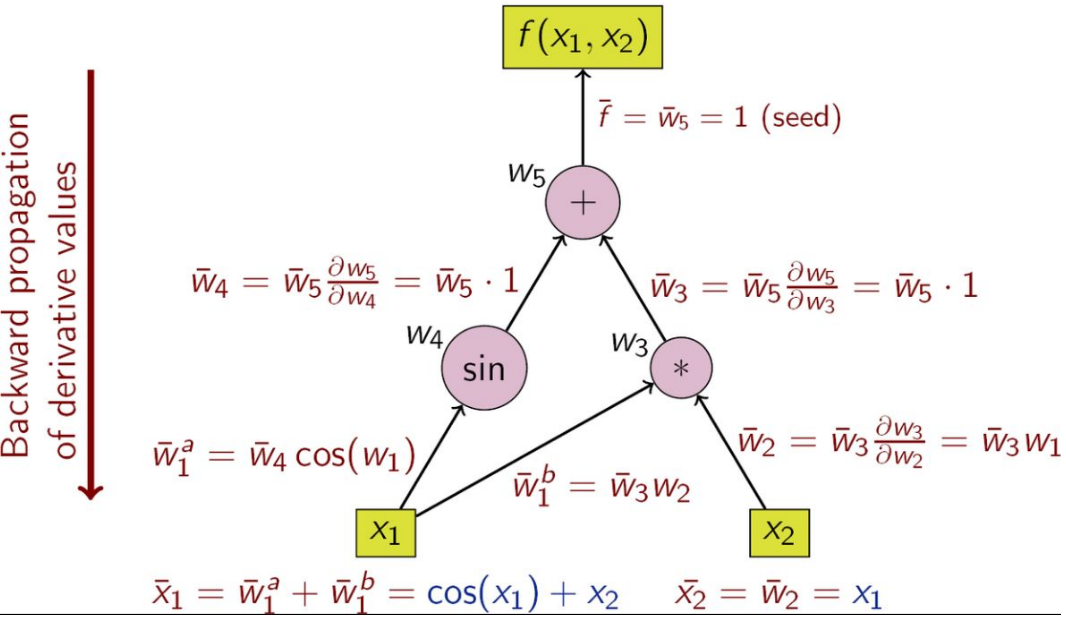
$$\frac{\partial Wx}{\partial x} = W^\top$$

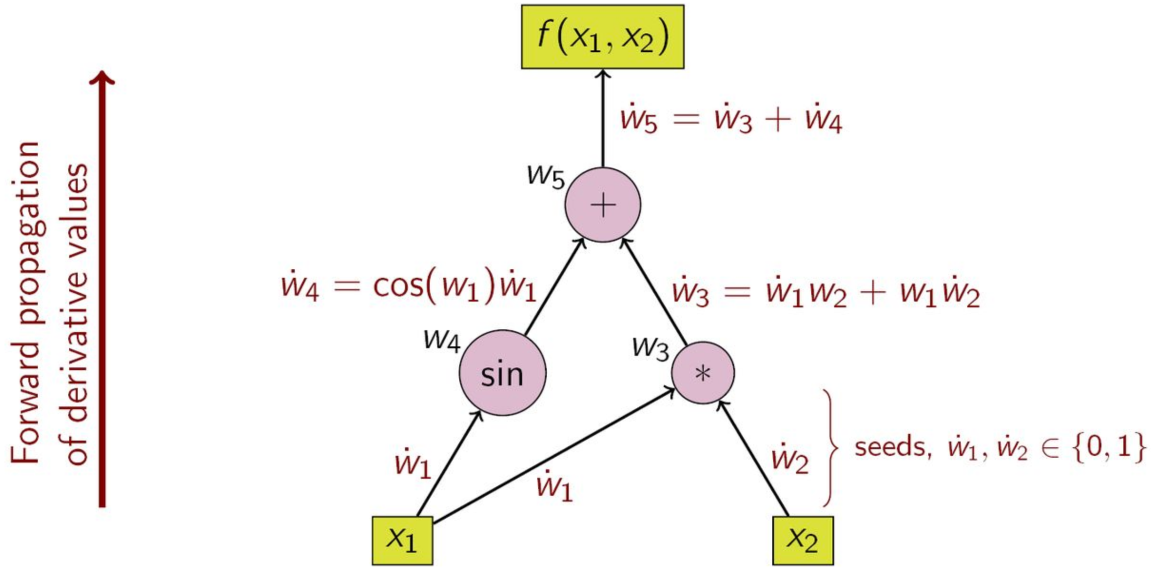
$$\frac{\partial f}{\partial x} = \frac{\partial Wx}{\partial x}\frac{\partial f}{\partial Wx} = W^\top \frac{\partial f}{\partial Wx}$$

$$\frac{\partial Wx}{\partial W} = (\frac{\partial Wx}{x})^\top x$$

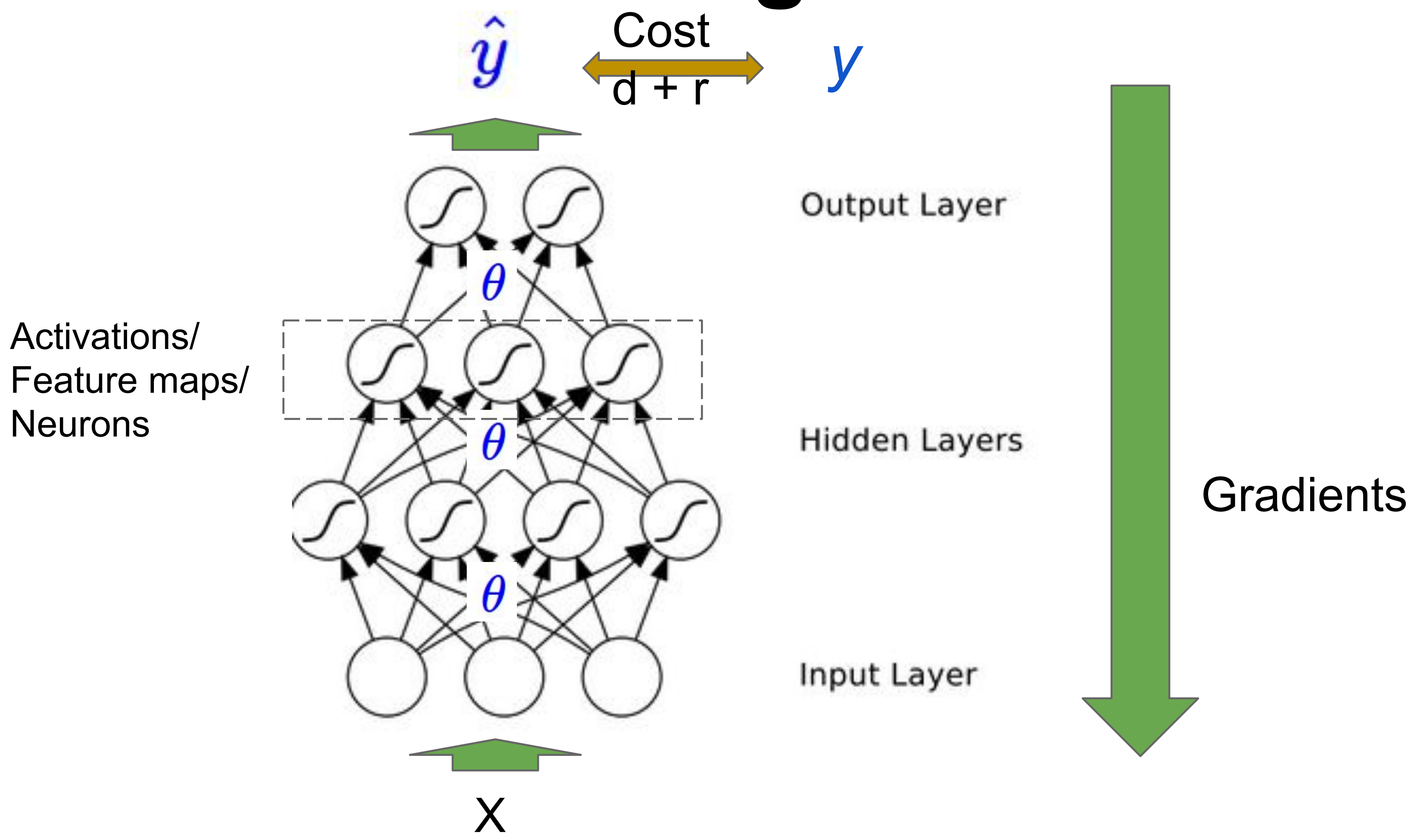
Backpropagation: Reverse Accumulation

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \cdots$$



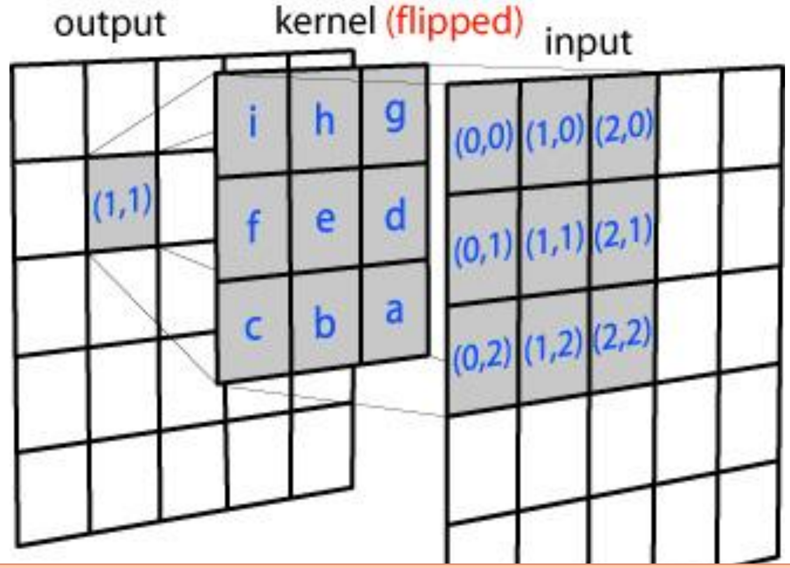


Neural Network Training



Convolutional Neural Network

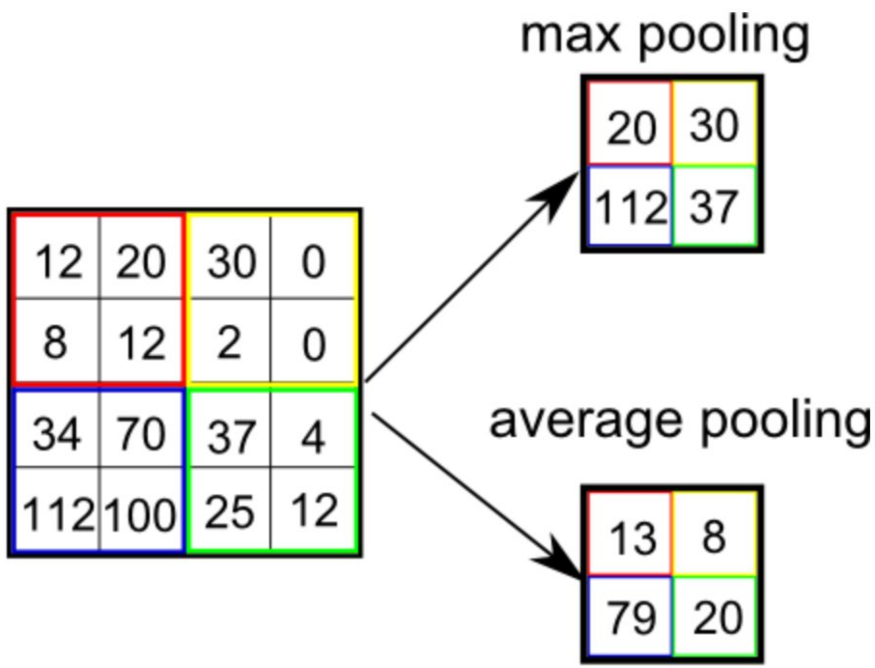
2D Convolution, Pooling, MaxOut



$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

2D Convolution

Convolution mode: flip kernel
Correlation mode: don’t flip kernel



Pooling is channel-wise nonlinear convolution with fixed params

Method 1: Convolution as Sparse Matrix Product

- \vect{M conv $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ Laplacian } = A \vect{M}
  - Because convolution is linear, it can be represented by a matrix
  - A is the sparse Poisson Matrix

$$A = \begin{bmatrix} D & -I & 0 & 0 & 0 & \dots & 0 \\ -I & D & -I & 0 & 0 & \dots & 0 \\ 0 & -I & D & -I & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & -I & D & -I & 0 \\ 0 & \dots & \dots & 0 & -I & D & -I \\ 0 & \dots & \dots & \dots & 0 & -I & D \end{bmatrix}, \quad D = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 4 & -1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 4 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 4 & -1 & 0 \\ 0 & \dots & \dots & 0 & -1 & 4 & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 4 \end{bmatrix},$$

Side note: Poisson Matrix

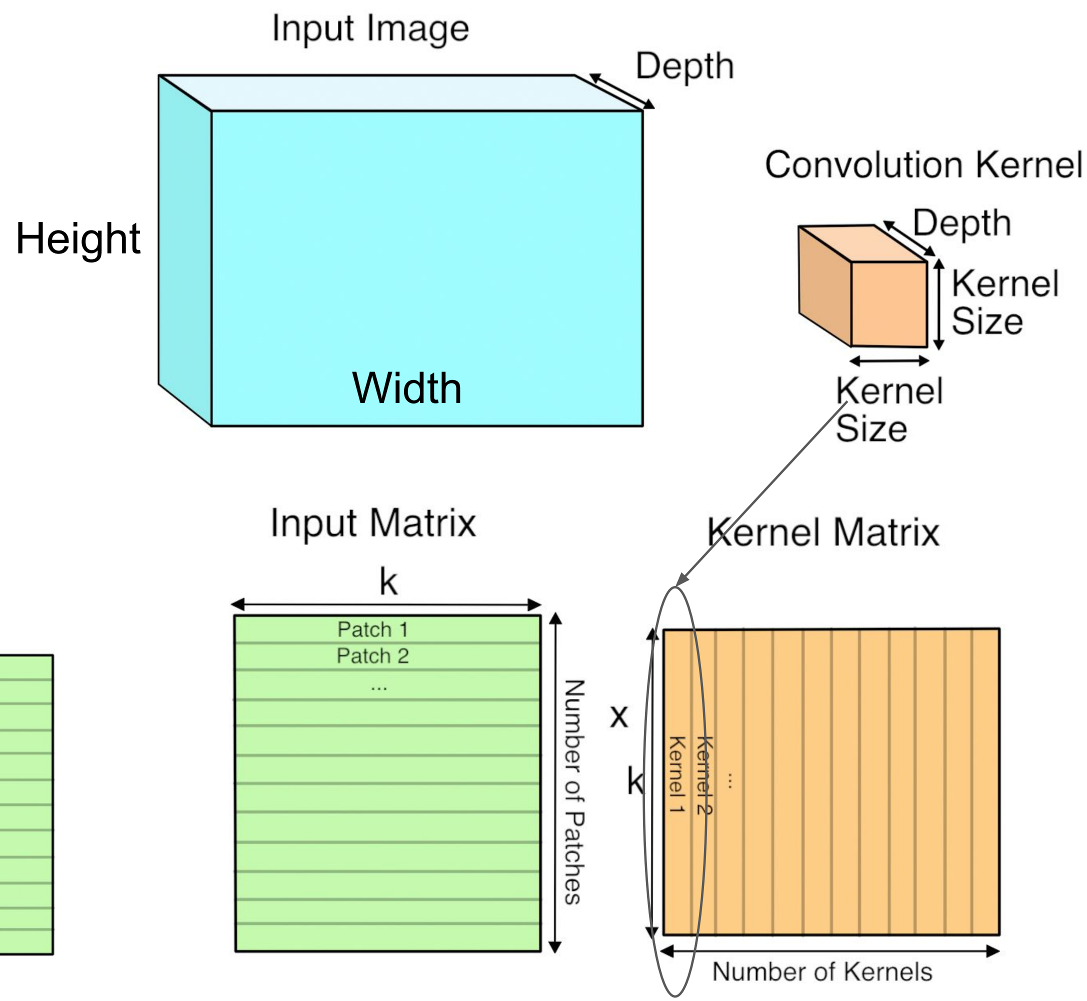
- $\mathcal{P}(m,n) = \mathcal{L}(n) \oplus \mathcal{L}(m)$
  - \oplus is Kronecker sum
- \vect{M} P = L_1 M + M L_2
  - Another way of saying 2D Laplacian is separable to x-axis Laplacian and y-axis Laplacian
  - Laplacian matrix is $\begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}$

Convolutional layer

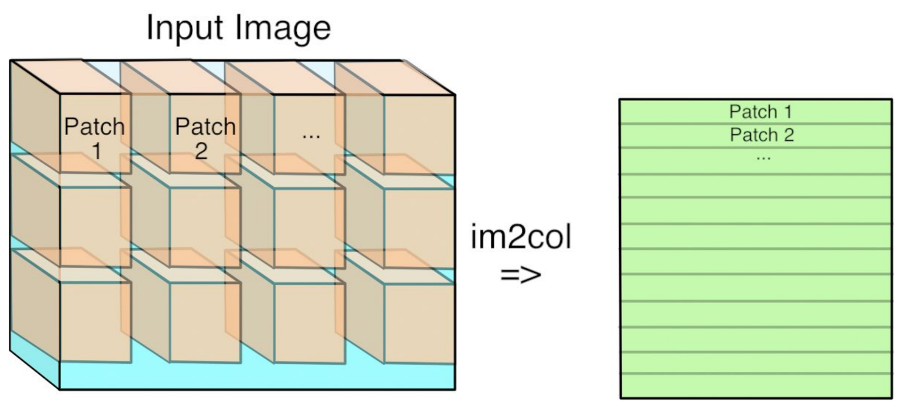
- Translation-invariance
  - Less free #param than Fully-Connected
    - Will have worse global minimum
    - But empirically can find better local minimum
    - **The most important reason behind network architecture engineering**

Method 2: Convolution as matrix product

- Convolution
  - feature map <N, C, H’, W’>
  - weights <K, C, H, W>
- Convolution as FC
  - under proper padding, can extract patchs
  - feature map <N H’ W’, C H W>
  - weights <C H W, K>



Kernel stride determines how much overlap



Locally-Connected

- Convolution is a block triband sparse matrix with shared parameters
- LC layers have unshared parameters, though the same nnz's as Convolutions
- Same amount of computation
- H' W' more parameters
- Useful for Algined Objects (faces)

Pooling

- Can use stride-2 conv to replace pooling for reducing #OP
  - Neupack requires special combination, like kernel_shape=4, padding=(1,1), stride=2
  - Completely no-overlap seems bad: kernel_shape=4, stride=4

Nonlinearity

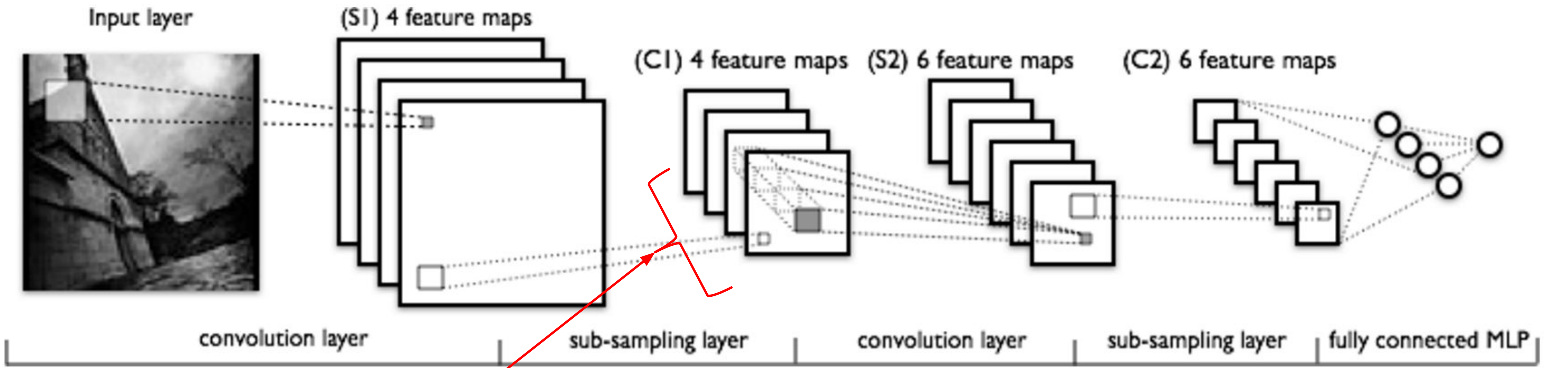
- f in $y \approx W_2 f(W_1(x))$
- Tanh/ReLU/MaxOut/Sin/Abs/AbsTanh all to some extent work
  - ReLU has point-wise scale invariance:
    - max(0, k x) = k max(0, x)
    - **W** and **k W** are both local minima
  - MaxOut has point-wise affine invariance:
    - max(k x_1 + b, k x_2 + b) = k max(x_1, x_2) + b
    - **W** and **k W + b** are both local minima

Nonlinearity

- Avoid tanh family as costly on CPU
- CappedReLU, CappedAbs: seems better
- ELU

MaxOut

- Simply max pooling over the channels
- Can effective reduce #output channels
- (?) In general don't mix maxout with other non-linearity



MaxOut is Max pooling over channels

MaxOut

- Maxout before softmax to boost quality
  - FC to 4*nr_class -> Maxout(4) ->Softmax
    - or 2*nr_class

Softmax

- Make anything like a probability $$y_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$$
- Boltzmann distribution
  - beta is reciprocal of temperature

$$y_i = \frac{exp(-\beta x_i)}{\sum_j exp(-\beta x_j)}$$

1, 2, -3 => 0.4, 0.6, 0.1

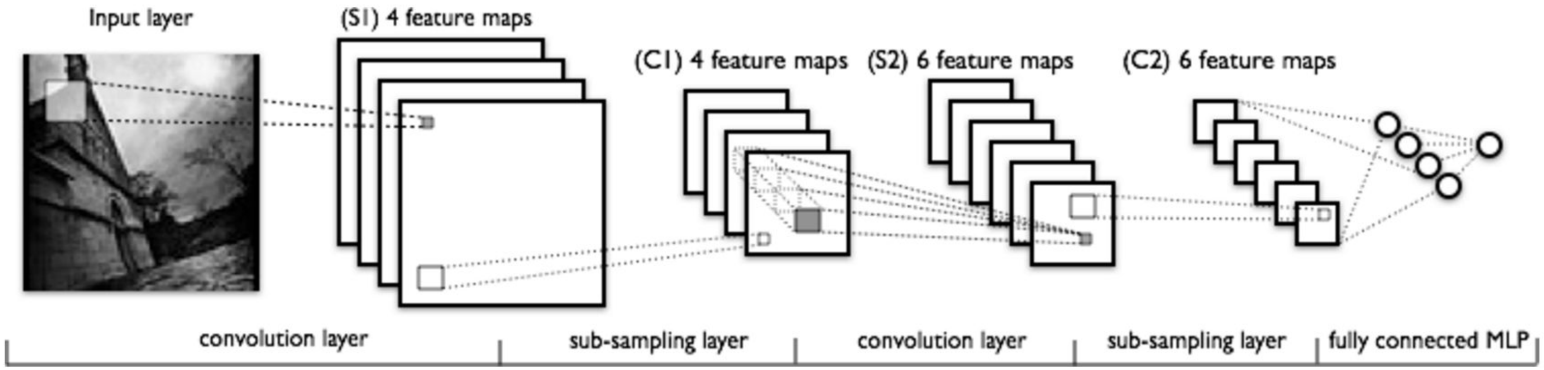
10, 20, -30 ==> 0, 1, 0

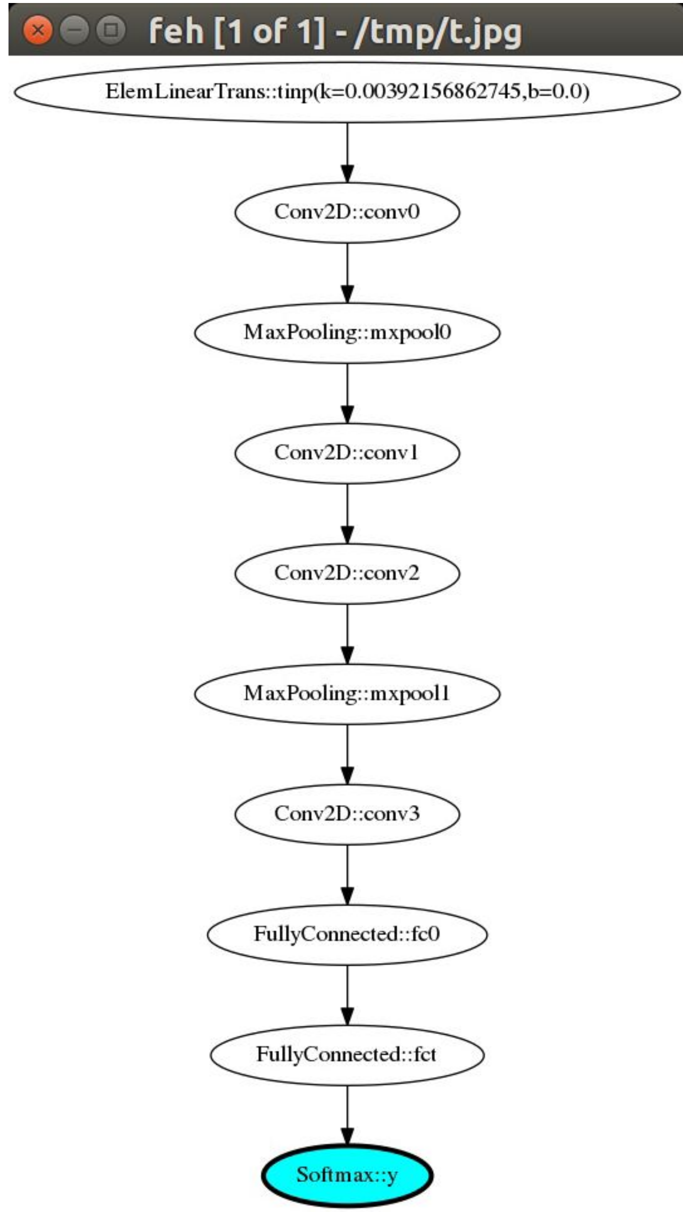
Softmax

- T_i
  - Partition training corpus and limits to smaller set of positive classes.

	Positive training classes associated with training example (x_i, T_i): $POS_i =$	Negative training classes associated with training example (x_i, T_i): $NEG_i =$	Input to Training Loss $G(x,y) =$	Training Loss	$F(x,y)$ gets trained to approximate:
Noise Contrastive Estimation (NCE)	T_i	S_i	$F(x,y) - log(Q(y\|x))$	Logistic	$log(P(y\|x))$
Negative Sampling	T_i	S_i	$F(x,y)$	Logistic	$log\left(\frac{P(y\|x)}{Q(y\|x)}\right)$
Sampled Logistic	T_i	$(S_i - T_i)$	$F(x,y) - log(Q(y\|x))$	Logistic	$logodds(y\|x) = log\left(\frac{P(y\|x)}{1-P(y\|x)}\right)$
Full Logistic	T_i	$(L - T_i)$	$F(x,y)$	Logistic	$log(odds(y\|x)) = log\left(\frac{P(y\|x)}{1-P(y\|x)}\right)$
Full Softmax	$T_i = \{t_i\}$	$(L - T_i)$	$F(x,y)$	Softmax	$log(P(y\|x)) + K(x)$
Sampled Softmax	$T_i = \{t_i\}$	$(S_i - T_i)$	$F(x,y) - log(Q(y\|x))$	Softmax	$log(P(y\|x)) + K(x)$

CNN: Alexnet-like





Importance of Convolutions and FC

Neupack: inspect_model.py
NeuPeak: npk-model-manip XXX info

param_name	shape	#floats	size	perc	
conv0_W	(24, 3, 5, 5)	1800	7.0 KiB	0.21%	
conv0_b	(24,)	24	96.0 B	0.00%	
conv1_W	(32, 24, 3, 3)	6912	27.0 KiB	0.80%	
conv1_b	(32,)	32	128.0 B	0.00%	
conv2_W	(32, 32, 3, 3)	9216	36.0 KiB	1.07%	
conv2_b	(32,)	32	128.0 B	0.00%	
conv3_W	(64, 32, 3, 3)	18432	72.0 KiB	2.14%	
conv3_b	(64,)	64	256.0 B	0.01%	
fc0_W	(1600, 512)	819200	3.1 MiB	95.11%	####################
fc0_b	(512,)	512	2.0 KiB	0.06%	
fct_W	(512, 10)	5120	20.0 KiB	0.59%	
fct_b	(10,)	10	40.0 B	0.00%	
total size		861354	3.3 MiB		

Most storage size

Feature map size

layer_name	ispace	ospace	osize	#ops	readable	perc	
tinp	(3, 40, 40)	(3, 40, 40)	4800	0	0.0	0.00%	
conv0	(3, 40, 40)	(24, 36, 36)	31104	2332800	2.2 Mi	32.43%	####################
mxpool0	(24, 36, 36)	(24, 18, 18)	7776	0	0.0	0.00%	
conv1	(24, 18, 18)	(32, 16, 16)	8192	1769472	1.7 Mi	24.60%	###############
conv2	(32, 16, 16)	(32, 14, 14)	6272	1806336	1.7 Mi	25.11%	###############
mxpool1	(32, 14, 14)	(32, 7, 7)	1568	0	0.0	0.00%	
conv3	(32, 7, 7)	(64, 5, 5)	1600	460800	450.0 Ki	6.41%	###
fc0	(64, 5, 5)	(512, 1, 1)	512	819712	800.5 Ki	11.39%	#######
fct	(512, 1, 1)	(10, 1, 1)	10	5130	5.0 Ki	0.07%	
y	(10, 1, 1)	(10, 1, 1)	10	0	0.0	0.00%	
total OPs				7194250	6.9 MiOps		

Most Computation

The Matrix View of Neural Network

- Weights of FullyConnected and Convolutions layers
  - take up most computation and storage size
  - are representable as matrices
- Approximating the matrices approximates the network
  - The approximation error accumulates.

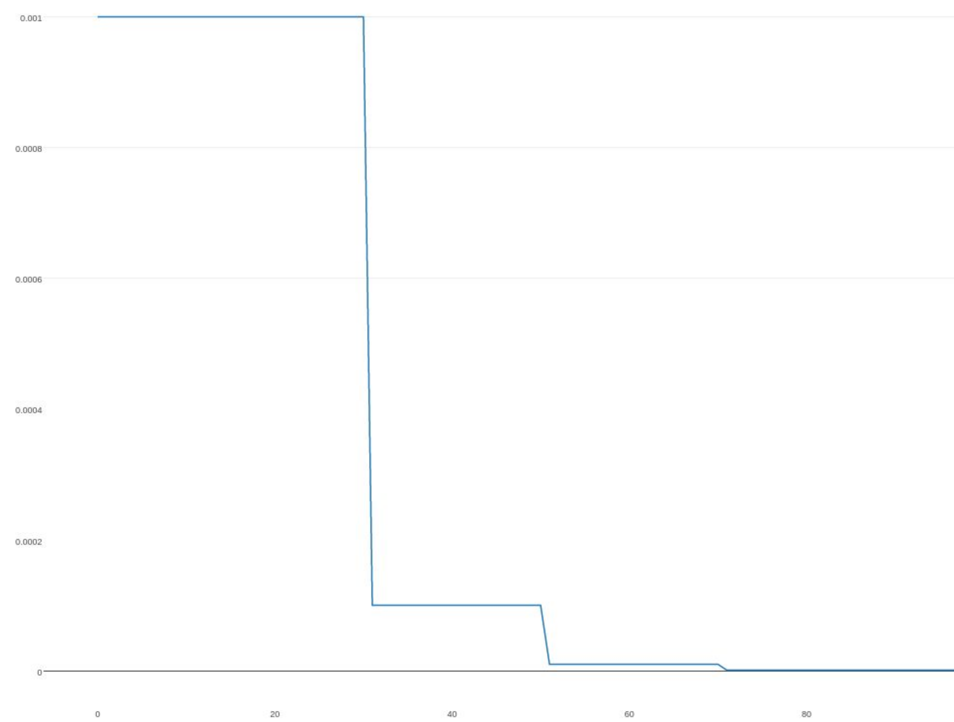
$$W_a \approx W \Rightarrow f(X, W_a) \approx f(X, W)$$

BatchNormalization $\alpha\frac{X-\mathrm{mean}(X)}{\mathrm{std}(X)}+\beta$

- Mechanism
  - Standardization of data by computing mean and std of mini-batch (In Megbrain, if multiple card, only the part on first-card is computed.)
  - recompute stats for inference: using moving-average or plain average over samples (latter more stable)
  - Followed by an affine layer to deal with nonlinearites that are not scale-invariant. **alpha**, **beta** are learnt.
- Effect
  - Stabilizer of model training
  - Faster convergence as we can use larger **lr**
  - **npk-sanitize-batch-normalization**
- Fine-tuning
  - BN有freezed

Tweaking learning rate

- Learning rate are “step lengths”
  - Sum of step length need be infinity, to be independent of initial value.
    - Good initial value saves distance.
  - It helps to have learning rate “decay” to do local search.



Learning rule

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w),$$

Momentum

$$\Delta w := -\eta \nabla Q_i(w) + \alpha \Delta w$$
$$w := w + \Delta w$$

AdaGrad

$$w := w - \eta \, \mathrm{diag}(G)^{-\frac{1}{2}} \circ g$$

RMSProp

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$
$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

ADAM Learning rule

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)

end while

Model & Training

CNN: Alexnet-like

- Works for almost-square Image
  - 28x28, 1 or 3 channel
  - Conv (5x5), 32 channel, nonlinearity
  - 24x24, 32 channel
  - pool(2x2)
  - 12x12, 32 channel
  - Conv (3x3), 32 channel, nonlinearity
    - often speed bottleneck as input also 32 channel
  - 10x10, 32 channel
  - pool(2x2)
  - 5x5: should not be too small
  - FC (512) + nonlinearity: often speed bottle neck and most #param
  - FC(10) + softmax

In fact a 5x5 conv without zero padding.

Model design rule-of-thumb

Neupack: inspect_model.py
NeuPeak: npk-model-manip XXX info

param_name	shape	#floats	size	perc	
conv0_W	(24, 3, 5, 5)	1800	7.0 KiB	0.21%	
conv0_b	(24,)	24	96.0 B	0.00%	
conv1_W	(32, 24, 3, 3)	6912	27.0 KiB	0.80%	
conv1_b	(32,)	32	128.0 B	0.00%	
conv2_W	(32, 32, 3, 3)	9216	36.0 KiB	1.07%	
conv2_b	(32,)	32	128.0 B	0.00%	
conv3_W	(64, 32, 3, 3)	18432	72.0 KiB	2.14%	
conv3_b	(64,)	64	256.0 B	0.01%	
fc0_W	(1600, 512)	819200	3.1 MiB	95.11%	####################
fc0_b	(512,)	512	2.0 KiB	0.06%	
fct_W	(512, 10)	5120	20.0 KiB	0.59%	
fct_b	(10,)	10	40.0 B	0.00%	
total size		861354	3.3 MiB		

Feature map size

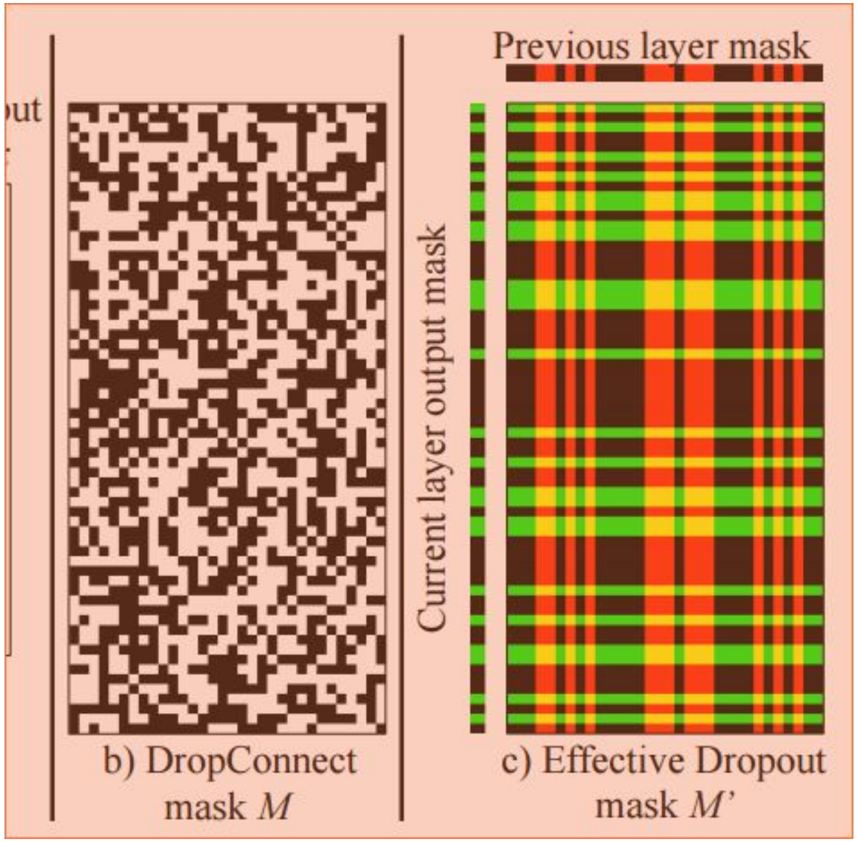
layer_name	ispace	ospace	osize	#ops	readable	perc	
tinp	(3, 40, 40)	(3, 40, 40)	4800	0	0.0	0.00%	
conv0	(3, 40, 40)	(24, 36, 36)	31104	2332800	2.2 Mi	32.43%	####################
mxpool0	(24, 36, 36)	(24, 18, 18)	7776	0	0.0	0.00%	
conv1	(24, 18, 18)	(32, 16, 16)	8192	1769472	1.7 Mi	24.60%	###############
conv2	(32, 16, 16)	(32, 14, 14)	6272	1806336	1.7 Mi	25.11%	###############
mxpool1	(32, 14, 14)	(32, 7, 7)	1568	0	0.0	0.00%	
conv3	(32, 7, 7)	(64, 5, 5)	1600	460800	450.0 Ki	6.41%	###
fc0	(64, 5, 5)	(512, 1, 1)	512	819712	800.5 Ki	11.39%	#######
fct	(512, 1, 1)	(10, 1, 1)	10	5130	5.0 Ki	0.07%	
y	(10, 1, 1)	(10, 1, 1)	10	0	0.0	0.00%	
total OPs				7194250	6.9 MiOps		

CNN: Alexnet

- SGD
  - Momentum
  - AdamV8(1e-4) seems fine
- Pooling: max/meanpooling
  - shape 3 stride 2(better on SVHN-cropped)

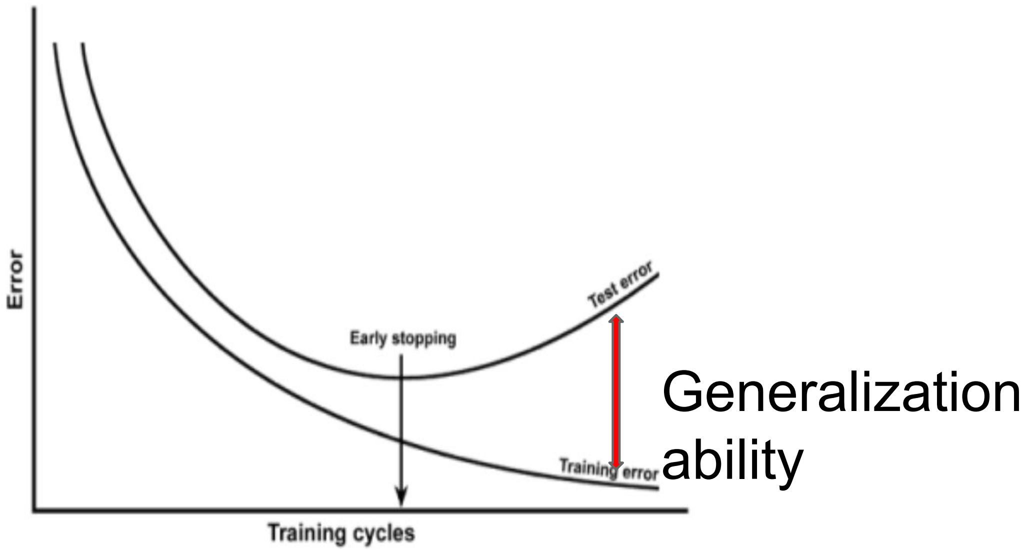
Dropout

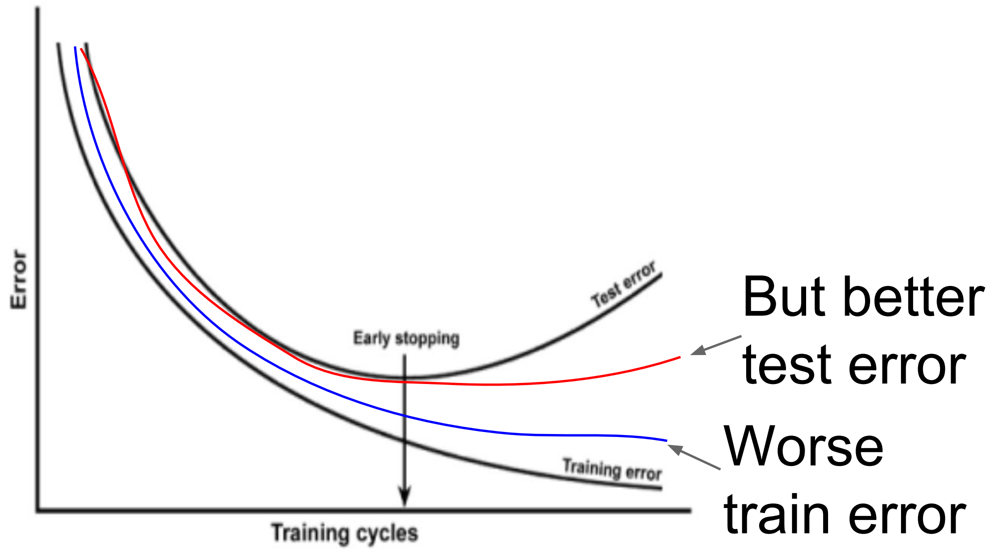
- Dropout
  - Multiple **W** element-wise with Bernoulli distribution.
  - p=0.5: Good on FC, but never on output FC
    - maybe good on input
    - or p=0.7 (better on SVHN-cropped)



Generalization Ability

- *Wisdom*: torture NN, but don't kill it, and it will have better generalizability.





CNN: Maxout

- Works for square like object, and faster
    - 28x28, 1 or 3 channel
    - Conv (5x5), 32 channel + Identity + Maxout(2)
    - 24x24, **16** channel
    - pool(2x2)
    - Feature map 12x12, **16** channel
    - Conv (3x3), 32 channel + Identity + Maxout(2)
        - faster as input only also 16 channel
    - Feature map 10x10, **16** channel
    - pool(2x2)
    - Feature map 5x5
    - FC (512) + nonlinear: often most #param
    - FC(10) + softmax

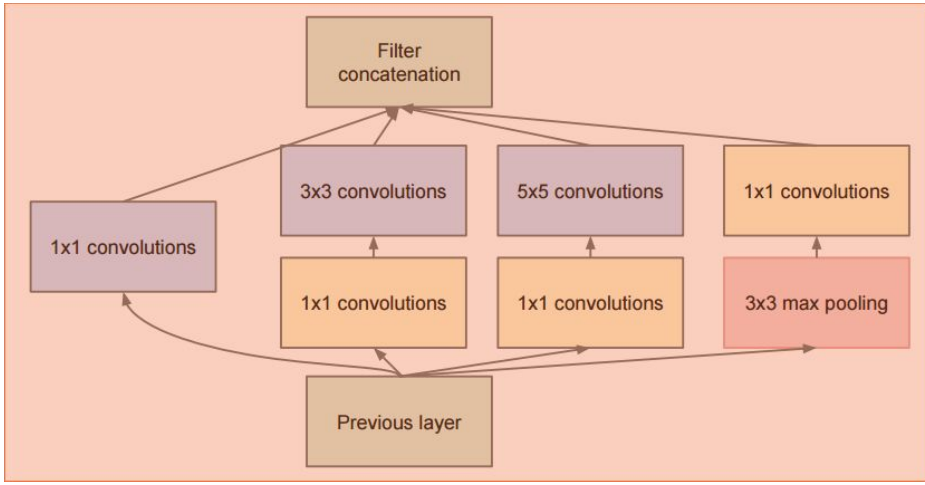
CNN: word recognition

- Softmax replacement:
  - MultiSoftmax: 0010 0100 0001
    - Use epsilon to encode var-length labels: **(label | epsilon)^n**
- Non-square pooling
  - Final feature map size should not be too small, but height may be much smaller than width
  - MaxPooling('pool2', pool_shape=(2, 3), padding=(0,2), pool_stride=(2,2)))
  - out_shape = (in_shape + padding - pool_shape) / pool_stride + 1
    - 4,13 => 2,7

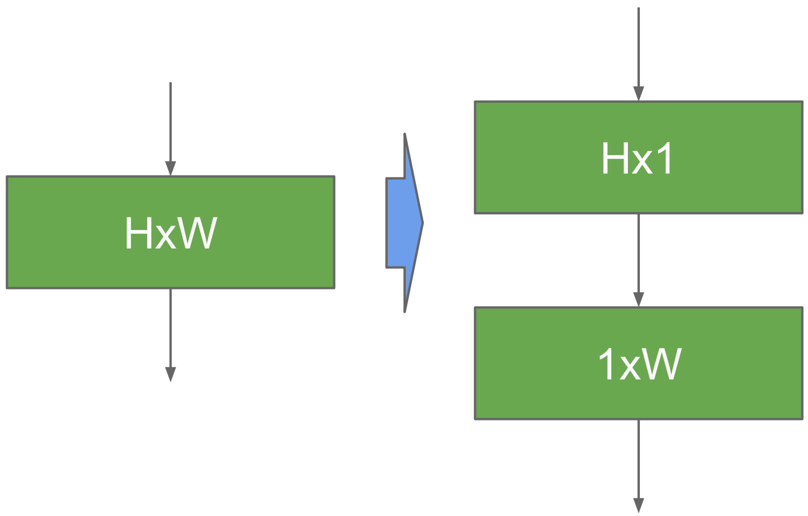
CNN: word recognition

- Fully-Connected
  - X = #instance, #channel_in
  - W = #channel_in, #channel_out
- Columwise Fully-Connected
  - Less #param and invarint to X-dimension translation
  - X = #instance, width, #channel_in
  - W = #channel_in, #channel_out
  - Faster than Recurrent NN

CNN: GoogLeNet

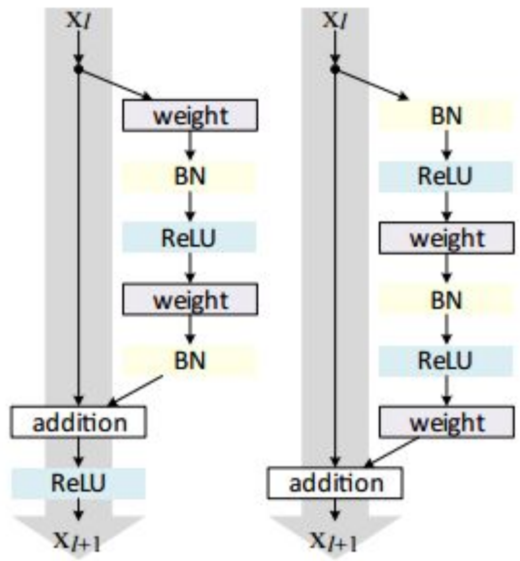


- Inception layer
  - Less #OP does not necessarily mean faster
    - especially on GPU as GPU loves regularity. CPU suffers less as has less #parallelism.
- A final large average pooling turns feature map size to 1x1
  - eliminates large FC, significantly reduce storage size and #OP
  - But may hurt performance
  - Two rank-1 conv's may be better

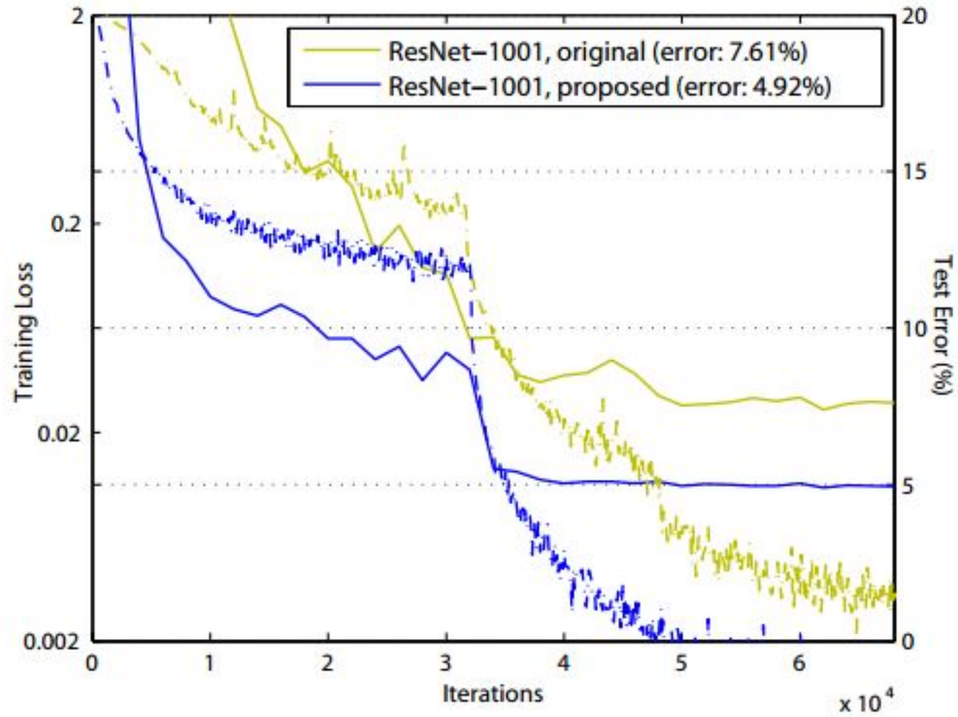


CNN: ResNet

- Original and "Identity mapping" variant

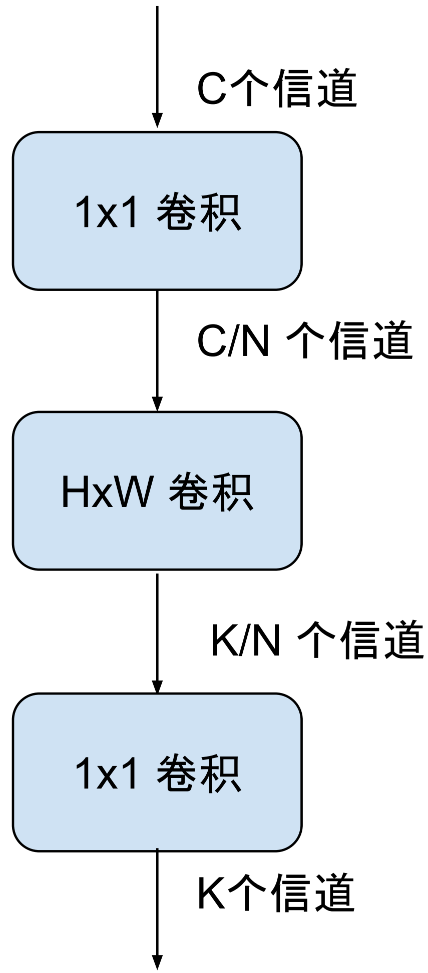


(a) original (b) proposed

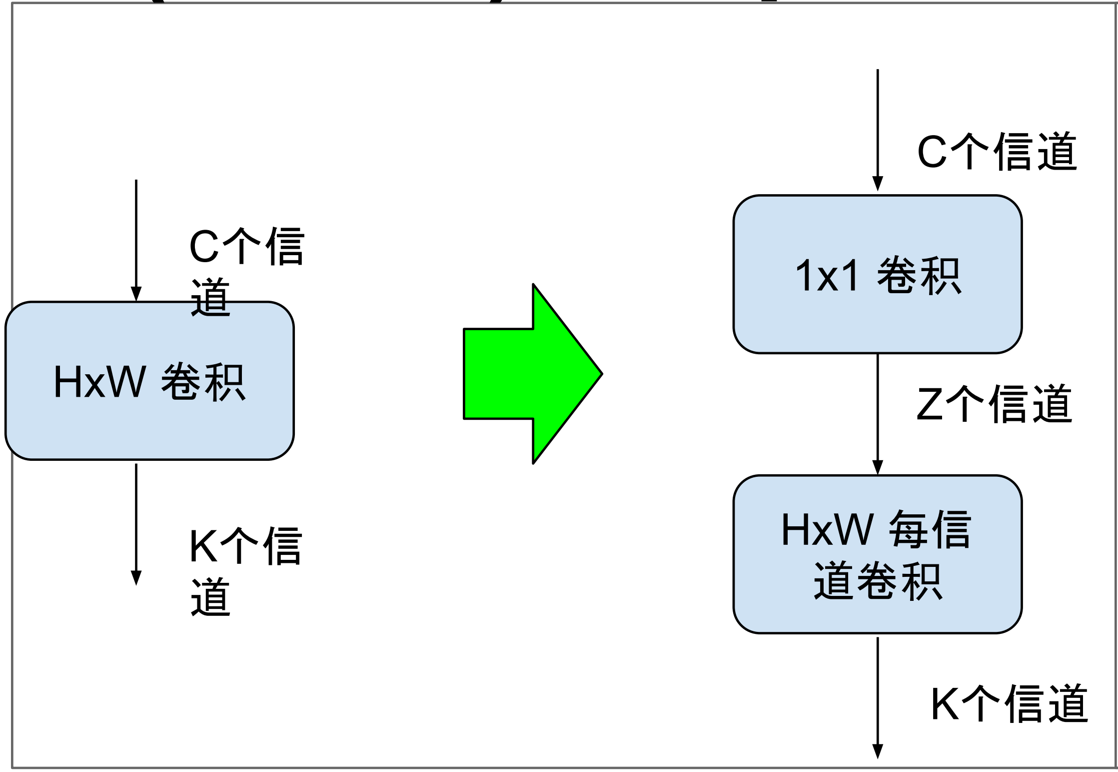


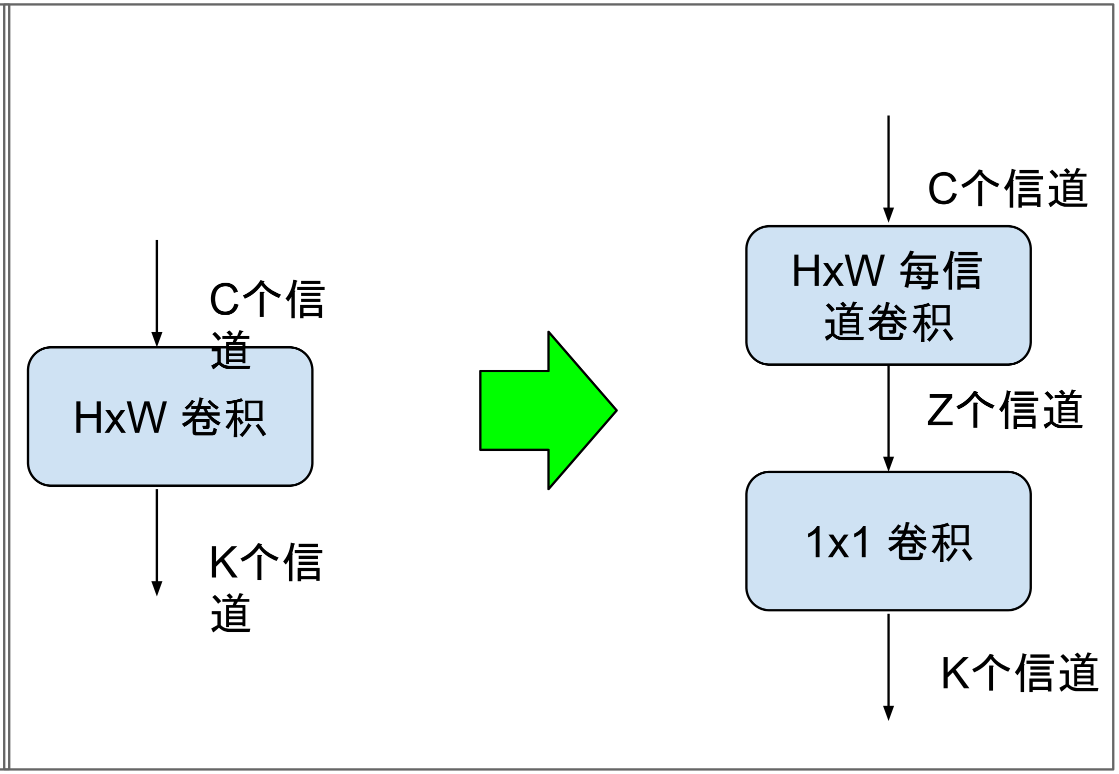
Bottleneck Structure

- ResNet-50 and above
- If channels-wise HxW conv
  - CP decomposition



(Shard) Xception





计算量极小，Receptive-field担当

Training Model

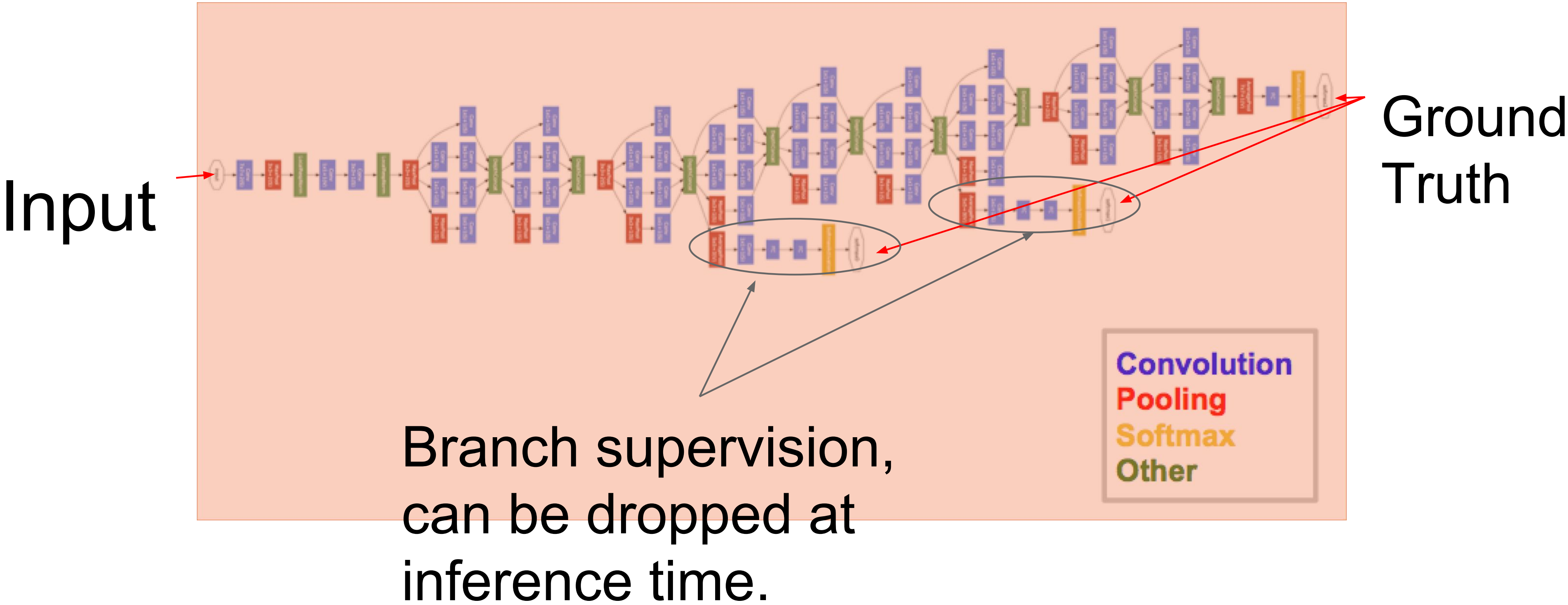
Before training

- Check list
    - data ***must*** be shuffled
        - Stochastic Gradient Descent relies on *unordered* order of data.
    - color images are not in BGR
        - The human face should not be blue
    - make sure ***have not*** mixed training/validation/test
    - read the work log if there is one
    - check #OP and learnable params in your model
    - If your model is big, don’t save too often.

Model Death (misclassify 0.9748)

- Tweak learning rate (larger or smaller)
- May add Batch normalization layers
- May try branch supervisions
- Check value range of parameters
  - If strange, change initial value range.
- Lower dropout
- Try wider & deeper model
  - but too wide & deep also may cause death

Branch Supervisions



Model Death (misclassify 0.9748)

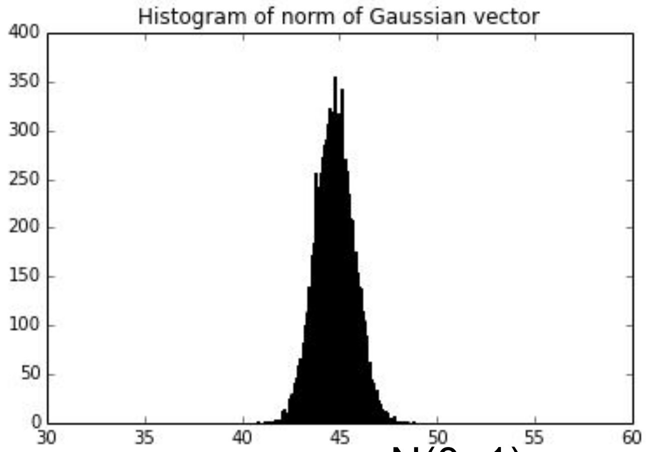
- Init’ing from a working model
  - Can steal some layers
- Curriculum learning (Doom)
  - Initiating a model on harder data with a model that works on easier data
  - w.r.t. number of class
    - Train 100-class before training 1000-class.
  - w.r.t. noise level
    - Reduce level of augmentation

Working Mechanism

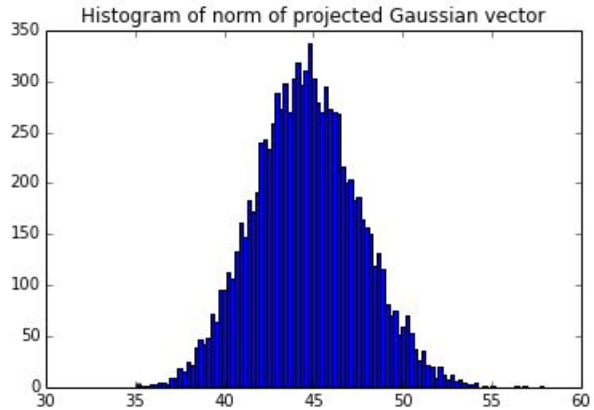
Dimension Reduction

- Symmetric (zero-mean) distribution perserves norm
  - whether Gaussian/Bernoulli/ternary
- Even random projection works somehow
  - Good initializers

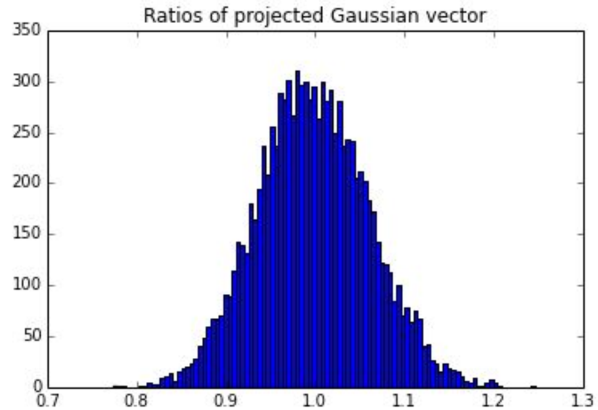
Norm and Inner-product



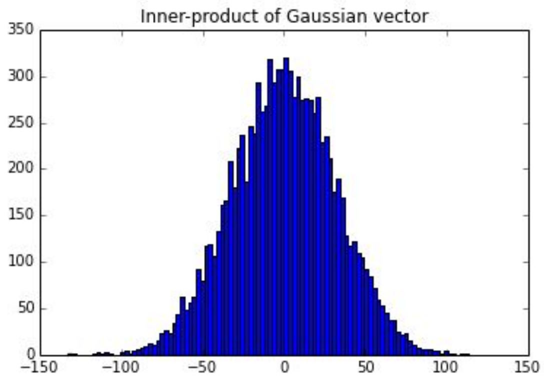
u - v, u, v ~ N(0, 1), ||u||_0 = 10000



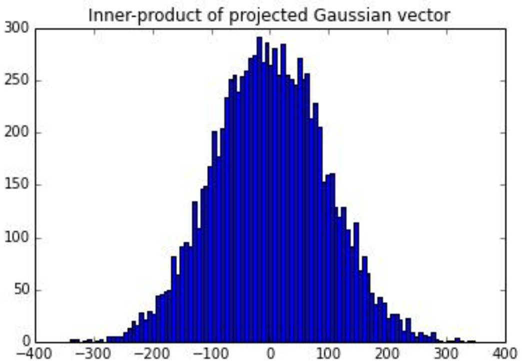
M(u - v), M \in R^{1250, 10000}



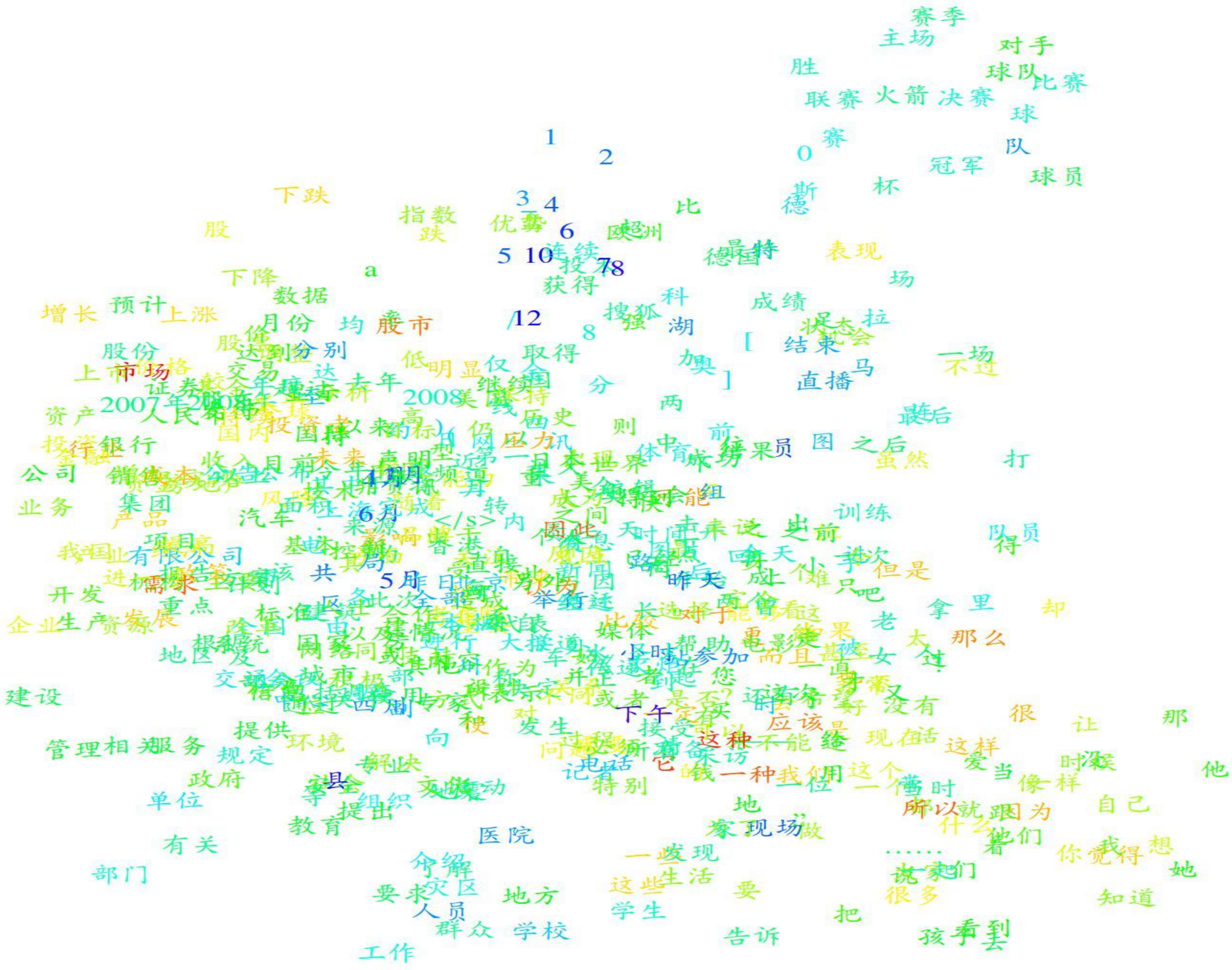
||M(u - v)|| / ||u -v||



<u, v>



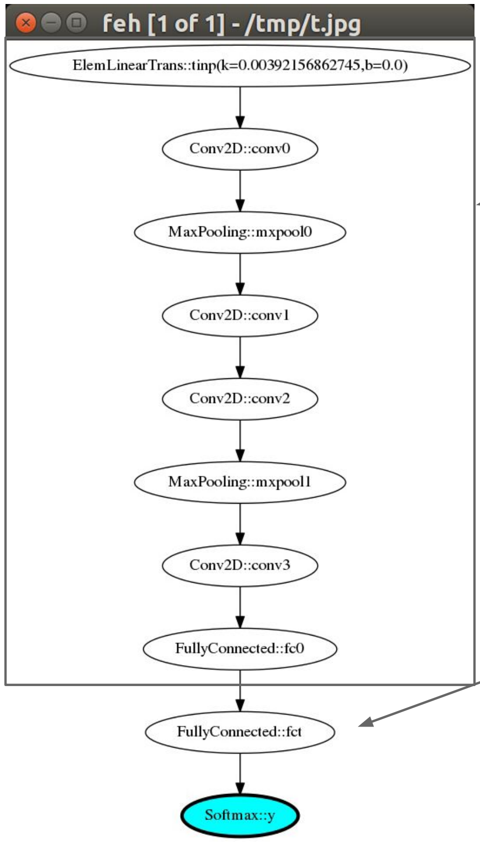
<M u, M v>



赛季
主场
对手
胜
球队
比赛
联赛
火箭
决赛
球
赛
队
冠军
球员
杯
1
2
0
下跌
指数
股
跌
优势
3
4
6
比
德
5
10
78
表现
下降
a
数据
获得
场
增长
预计
上涨
股市
12
科
成绩
搜狐
湖
拉
月份
均
8
结束
股份
分别
一场
不过
直播
马
取得
明显
低
上市
市场
2008
2007年
资产
两
之后
图
投资
银行
最后
历史
压力
虽然
打
公司
集团
业务
训练
汽车
产品
队员
有限公司
之前
开发
但是
重点
拿
里
却
企业
那么
地区
建设
城市
提供
管理
服务
环境
很
让
那
规定
政府
这样
单位
医院
教育
有关
部门
要求
灾区
地方
人员
群众
学校
工作
学生
要
告诉
把
孩子
看到
知道
她
他
自己
你
觉得
想
时候
像
样

Good practices

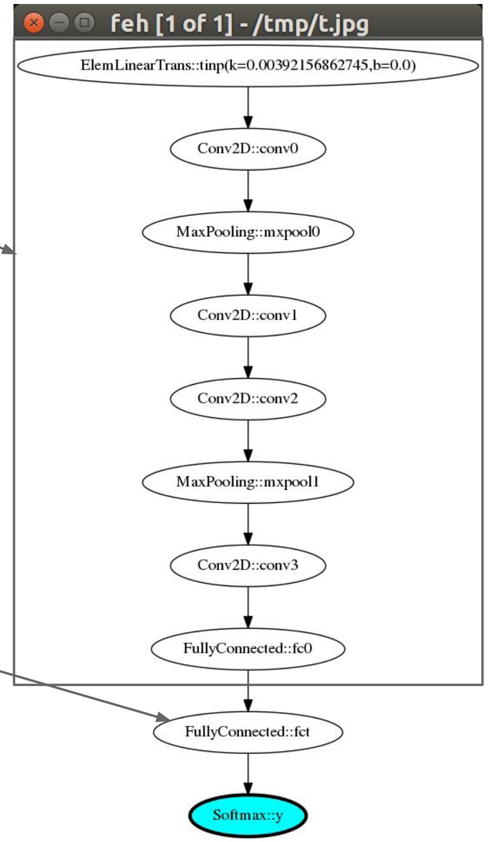
100-class to 1000-class



100-class

Not affected by #class and can be reused.

Can pad zero to get larger output matrix.



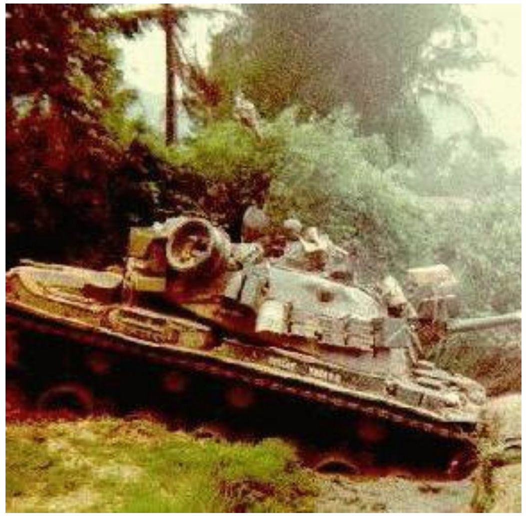
1000-class

Multi-task learning

- Wisdom: weak supervsion signal may be problematic
  - 2-class detection may be harder to train than n-class recognition

No

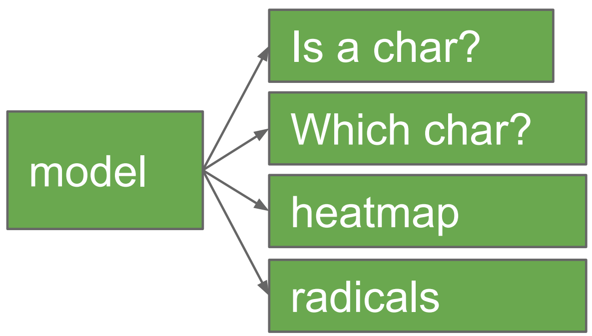
Tank classifier,
or weather
classifier?



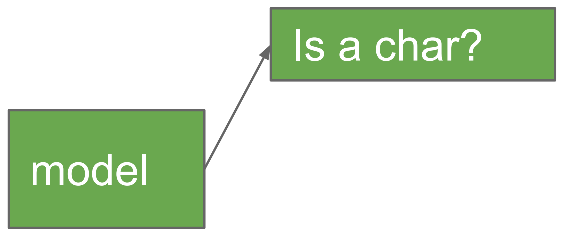
Yes

Multi-task learning

- Can use additional supervision signal for training, but can omit when inference.



Train time



Inference time, no extra computation

Need balancing the order of magnitude of losses, otherwise the smaller losses will be “drowned” in the fluctuations of the larger losses.

Pairwise Loss, Triplet Loss

- When many classes with few per-class instances, need pairwise loss

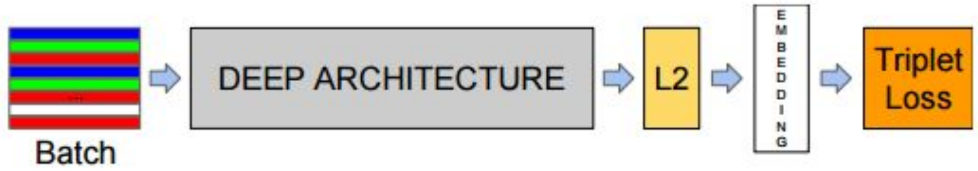


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

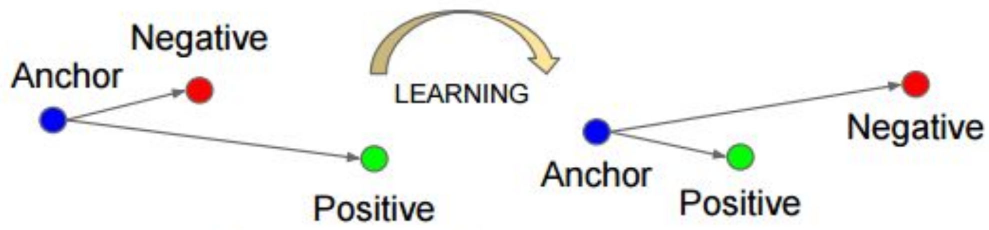


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

Hard Example Mining

- With pairwise/triplet loss, random sampling will become less efficient
  - Both of below work

- Generate triplets offline every n steps, using the most recent network checkpoint and computing the argmin and argmax on a subset of the data.
- Generate triplets online. This can be done by selecting the hard positive/negative exemplars from within a mini-batch.

Selecting the hardest negatives can in practice lead to bad local minima early on in training, specifically it can result in a collapsed model (*i.e.* $f(x) = 0$). In order to mitigate this, it helps to select x_i^n such that

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 \ . \tag{4}$$

We call these negative exemplars *semi-hard*, as they are fur-


基本法

CNN: further

- Quality-speed tick-tock
  - speed: smaller input (O(n^2)), less channels, shallower, maxout
  - quality: more data, larger input, more channels, deeper, more dropout
    - synthesizing, multitask
- Be prepared for strong unpredicability in convergence/quality

Model design rule-of-thumb

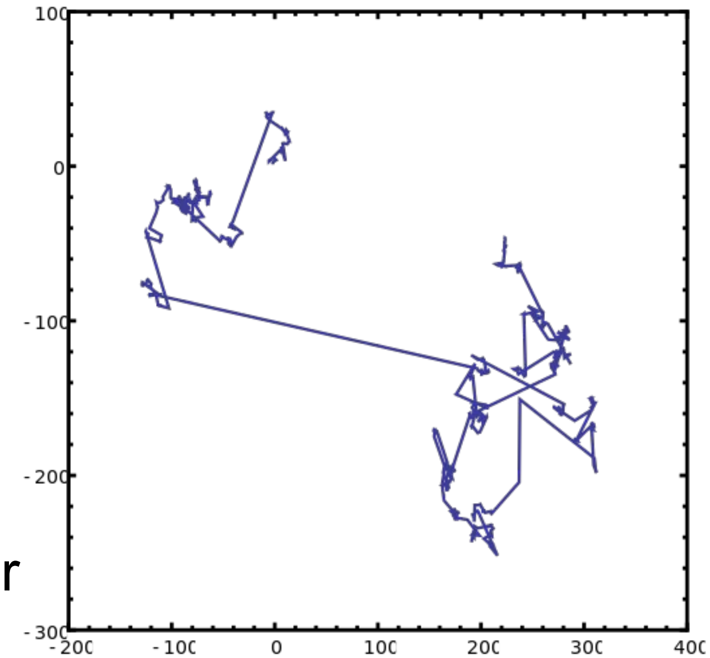
- Determine input image size
- Roughly compute #layer and #channel
  - Figure out how many #OP you can afford
  - Don’t concentrate your #OP in one layer
- Feature map should be gradually smaller
  - Don’t create a bottleneck
  - Determine #pooling layers
- Start with a fatter-than-requirement model
  - #OP should be at the same order of magnitude
  - Later reduce model by cutting #channel
- If no idea, just create a model. It will change a lot anyway.

Searching for good model

- Analyze the property of the problem
  - translation invariance? More conv and less FC.
  - sensitive to scale? Sprite like model.
  - sequence? Multi-softmax, colfc, RNN.
- Boostrap by modifying a proven-to-work model
  - Most models don’t work. Better reuse old wisdom.

Searching for good model

- Repeat: big-step, many baby-step’s
  - Big steps helps you explore design space
    - Make wild changes and hope to get better
  - Baby step locally search for local optimum
    - Mostly in the form of control experiments where only one factor changes, to allow for later combination of factors
    - Should be densely logged

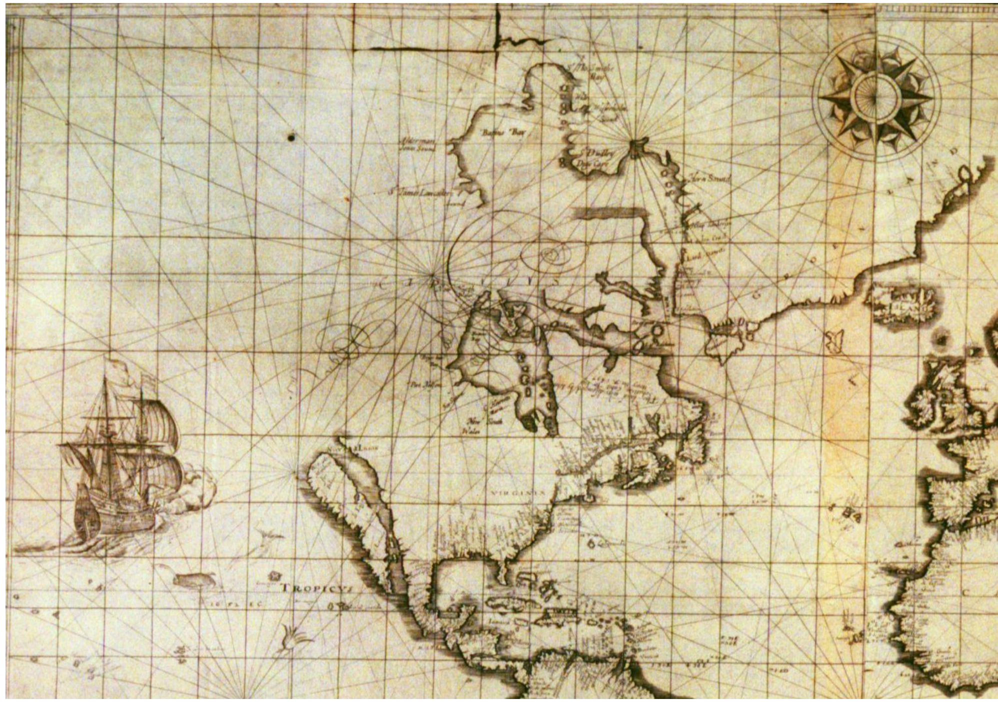


Lévy flight, one way animals find food.

Systematic naming of experiments

- Naming
  - 中浙优8号
    - 中稻, 浙江产, 优, 8号
  - 隆平稻 (reserved for exceptional good ones)
    - serve to shorten name
- config/quarter_fc_noepsilon_nodupe/model.py
  - quarter_fc: FC only has quarter size of the original
  - noepsilon + nodupe: remove epsilon and dupes in labels before matching

Work log for record



- The way Colombus discovers America
- Scripts:
  - Monitoring progress
    - watch "python neupeak/scripts/gen_work_log.**sh** train_log/log.txt|tail"
  - Producing record
    - neupeak/scripts/gen_work_log.**py** train_log/log.txt

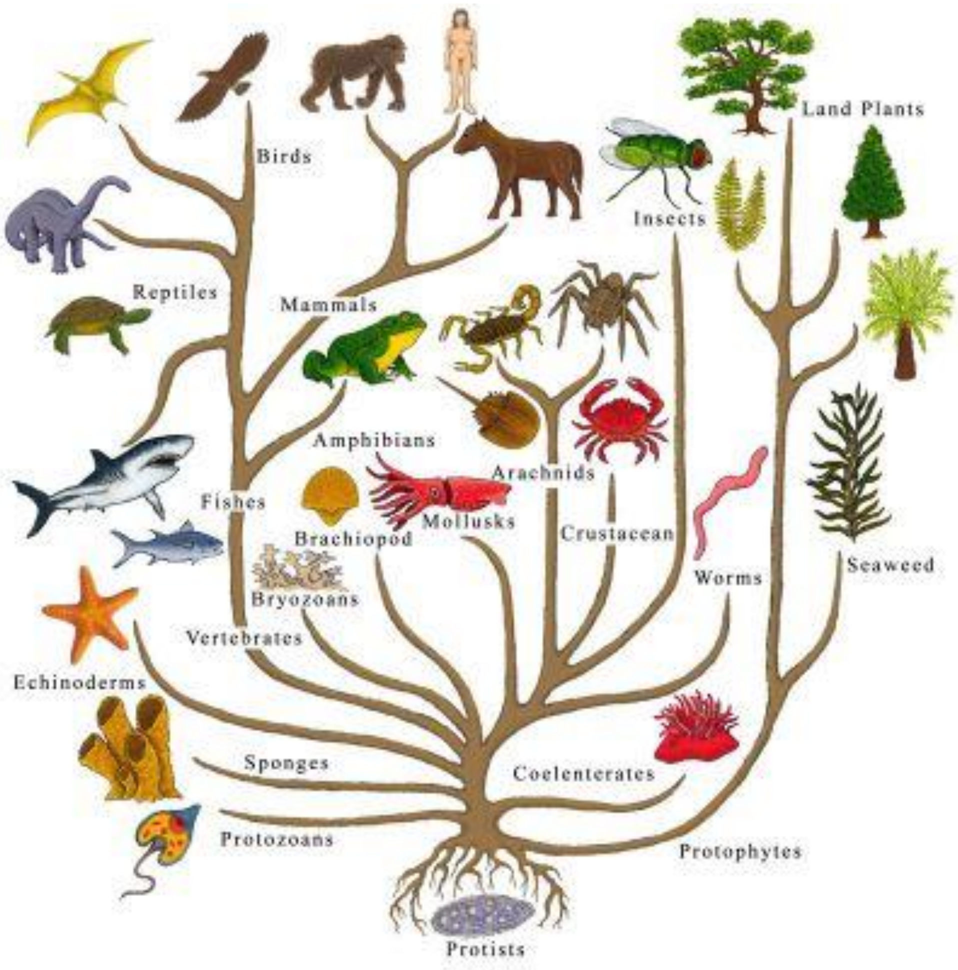
Work philosophy

- Never stare at training process unless for MNIST
  - doing training asynchronously
- Find a quick numeric way to check result
  - a bad metric is better than no metric
  - or define a tiny representative set (that can be human-eval'ed in <1minute)
- Check before sleep whether the job is alive
  - Check if consuming too much memory
  - Check deadlock

每天工作时间安排

时　间	工作安排
08:30～09:00	开新反应A　（黄金30分）
09:00～15:00	反应后处理和产品纯化（过夜反应和反应A）
15:00～16:00	开过夜反应B　（关键）
16:00～17:00	制定次日反应方案，准备反应所需材料。

Parallel experiments



Evolution of a NN’ist

- 我一直在搞NN，开始是Nearest Neighbor,然后是Nuclear Norm,现在的Neural Network.
- Evolution
  - Beginner: read Kevin Murphy’s book and anxiously watch MNIST training. Loves trying different nonlinearity. Uses Mathematica for NN.
  - Junior: Finds Inception Layer to be great. Discovers Neural Turing Machine. Loves making everything differentiable in Theano. Thinks coffee is a misspelled word.
  - Senior: Doing tensor decomposition on convolution weights. Building up bit NN runtime. Trains VGG-16 in proprietry NN framework.
  - ...
  - Master: If anything, send an E-mail to zxy@.