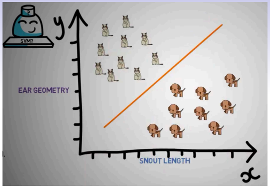
上节课回顾

Support Vector Machines



Here we approach the two-class classification problem in a direct way:

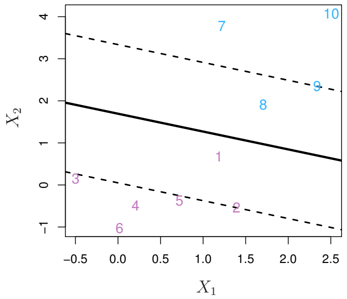
We try and find a plane that separates the classes in feature space.

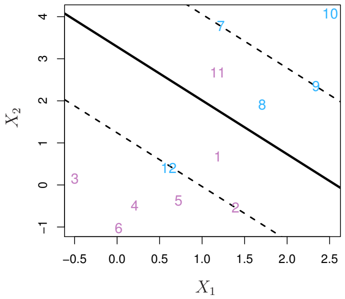
If we cannot, we get creative in two ways:

- We soften what we mean by “separates”, and
- We enrich and enlarge the feature space so that separation is possible.

上节课回顾

Support Vector Classifier





$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

上节课回顾　对偶问题

SVM 的优化思路是 **样本点到超平面的间隔最大化**，在原空间上SVM 要优化的问题的拉格朗日函数如下所示，即我们要做的就是在 $\lambda_i \geq 0$ 的约束下求解 $L(w,b,\lambda)$ 函数的最小值，（详细的推导过程请参考我的前一篇博客 $\rightarrow$ 软硬SVM）

$$L(w,b,\lambda) = \frac{1}{2}||w||^2 + \sum_{i}^{m} \lambda_i [1 - y^{(i)}(w^T x^{(i)} + b)]$$
$$\text{s.t. } \lambda_i \geq 0$$

我们将其转换为对偶问题 (即先对 w, b 求 L 的最小值，在对 λ 求 L 的最大值)，对 w, b 的求导得到，

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{m} \lambda_i y^{(i)} \phi(x^{(i)}), \quad \frac{\partial L}{\partial b} = -\sum_{i=1}^{m} \lambda_i y^{(i)}$$

令 w, b 的偏导数为 0，我们得到 $w^* = \sum_{i=1}^{m} \lambda_i y^{(i)} \phi(x^{(i)})$ 以及 $\sum_{i=1}^{m} \lambda_i y^{(i)} = 0$。我们将它们带入到 L 中可得到其最小值 $L_{min}(w,b,\lambda) = \theta(\lambda)$，最优化问题转变为求解 $\theta(\lambda)$ 的最大值问题，即

$$\theta(\lambda) = -\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y^{(i)} y^{(j)} \boxed{[\phi(x_i)]^T \phi(x_j)} + \sum_{i=1}^{m} \lambda_i$$
$$\text{s.t. } \sum_{i=1}^{m} \lambda_i y^{(i)} = 0, \quad \lambda_i \geq 0, i = 1 \ldots m.$$

SMO序列最小优化算法

由于$\phi(x_i), \phi(x_j)$ 都是 (nx1) 维向量，因此上式中的 $[\phi(x_i)]^T \phi(x_j)$ 也可以写为内积的形式 $\phi(x_i) \cdot \phi(x_j)$，故 $\theta(\lambda)$ 也可以写成 $\theta(\lambda) = -\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y^{(i)} y^{(j)} \phi(x_i) \phi(x_j) + \sum_{i=1}^{m} \lambda_i$。

查看链接https://blog.csdn.net/chikily_yongfeng/article/details/105645955

Feature Expansion

- Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,.... Hence go from a p-dimensional space to a $M > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Example: Suppose we use $(X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2)$ instead of just (X_1, X_2). Then the decision boundary would be of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0$$

This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

上节课回顾

Equation 5-9. Kernel trick for a 2nd-degree polynomial mapping

$$\phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) = \begin{pmatrix} a_1^2 \\ \sqrt{2}\, a_1 a_2 \\ a_2^2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}\, b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2 a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

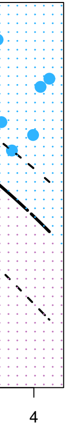
$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)^2 = \left(\mathbf{a}^T \cdot \mathbf{b}\right)^2$$

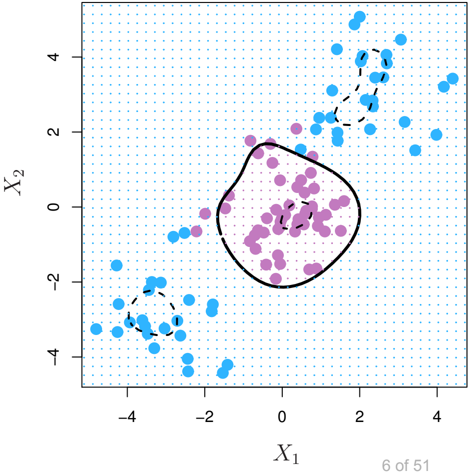
On the left-hand side, we have the dot product of the transformed feature vectors, which is equal to our 2nd-degree polynomial kernel function.

上节课回顾

Radial Kernel

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2).$$



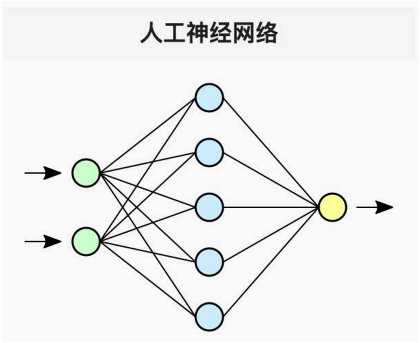


$$f(x) = \beta_0 + \sum_{i\in\mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

Implicit feature space; very high dimensional.

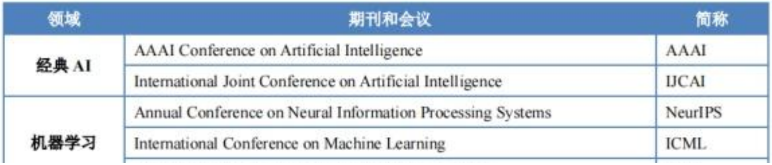
Controls variance by squashing down most dimensions severely

Deep Learning



Deep Learning

Neural networks became popular in the 1980s.
Lots of successes, hype, and great conferences: NeurIPS, Snowbird.



领域	期刊和会议	简称
经典 AI	AAAI Conference on Artificial Intelligence	AAAI
	International Joint Conference on Artificial Intelligence	IJCAI
机器学习	Annual Conference on Neural Information Processing Systems	NeurIPS
	International Conference on Machine Learning	ICML

Deep Learning

Neural networks became popular in the 1980s.
Lots of successes, hype, and great conferences: NeurIPS, Snowbird.

Then along came SVMs, Random Forests and Boosting in the 1990s, and Neural Networks took a back seat.

Deep Learning

Neural networks became popular in the 1980s.
Lots of successes, hype, and great conferences: NeurIPS, Snowbird.

Then along came SVMs, Random Forests and Boosting in the 1990s, and Neural Networks took a back seat.

Re-emerged around 2010 as *Deep Learning.*
By 2020s very dominant and successful.

Part of success due to vast improvements in computing power, larger training sets, and software: Tensorflow and PyTorch.

Deep Learning

Neural networks became popular in the 1980s.
Lots of successes, hype, and great conferences: NeurIPS, Snowbird.

Then along came SVMs, Random Forests and Boosting in the 1990s, and Neural Networks took a back seat.

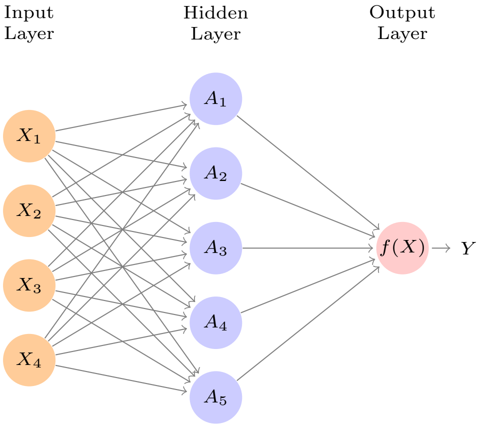
Re-emerged around 2010 as *Deep Learning.*
By 2020s very dominant and successful.

Part of success due to vast improvements in computing power, larger training sets, and software: Tensorflow and PyTorch.

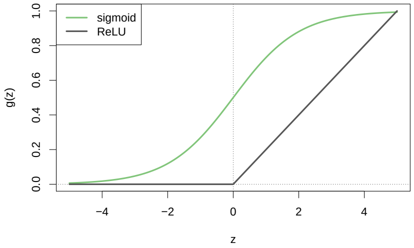
Much of the credit goes to three pioneers and their students: Yann LeCun, Geoffrey Hinton and Yoshua Bengio, who received the 2019 ACM Turing Award for their work in Neural Networks.

Single Layer Neural Network

$$\begin{aligned} f(X) &= \beta_0 + \textstyle\sum_{k=1}^{K} \beta_k h_k(X) \\ &= \beta_0 + \textstyle\sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j). \end{aligned}$$

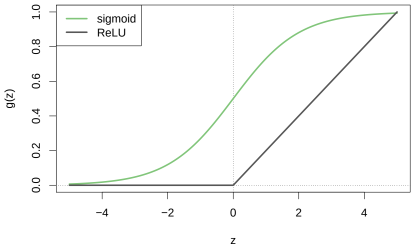


Details



- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj}X_j)$ are called the *activations* in the *hidden layer*.

Details

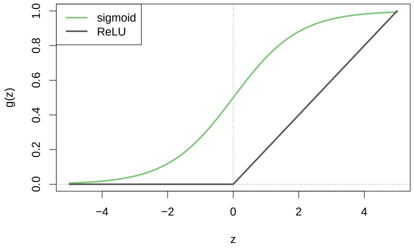


- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.

sigmoid: $$\sigma(x) = \frac{1}{1+e^{-x}}$$

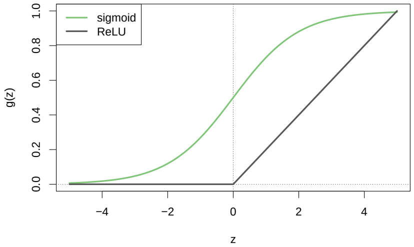
ReLU: $$f(x) = x^+ = \max(0, x) = \frac{x+|x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise,} \end{cases}$$

Details



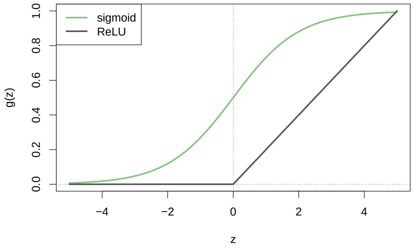
- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.
- Activation functions in hidden layers are typically nonlinear, otherwise the model collapses to a linear model.

Details



- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.
- Activation functions in hidden layers are typically nonlinear, otherwise the model collapses to a linear model.
- So the activations are like derived features — nonlinear transformations of linear combinations of the features.

Details



- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.
- Activation functions in hidden layers are typically nonlinear, otherwise the model collapses to a linear model.
- So the activations are like derived features — nonlinear transformations of linear combinations of the features.
- The model is fit by minimizing $\sum_{i=1}^{n} (y_i - f(x_i))^2$ (e.g. for regression).

Optimization Algorithm

Epoch: The number of times the algorithm runs over the whole training dataset

Sample: A subset of dataset

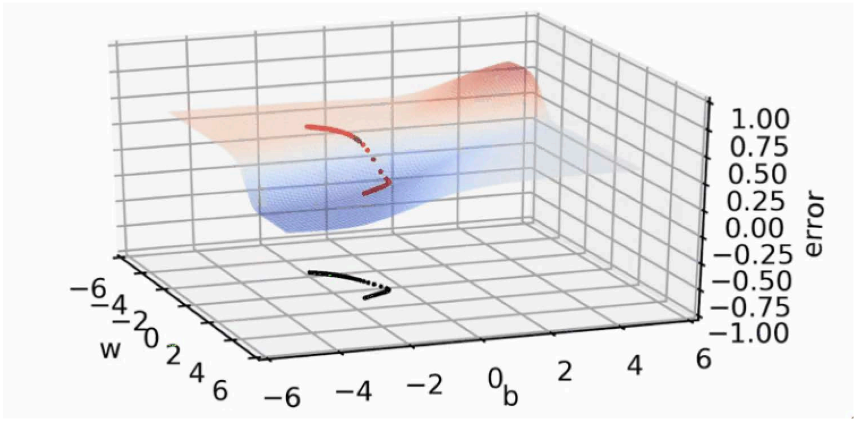
Batch: It denotes the number of records to be considered for updating the model parameters

Learning Rate: It is a parameter that provides the model a scale of how much model weights should be updated

Cost / Loss Function: A cost function is used to calculate the cost , which is the difference between the predicted value and actual value

Weights/Bias: The learnable parameters in a model that controls the signal between two neurons

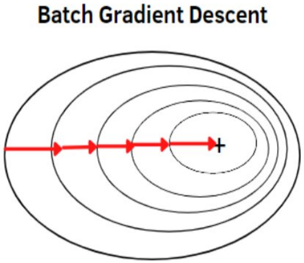
Gradient Descent



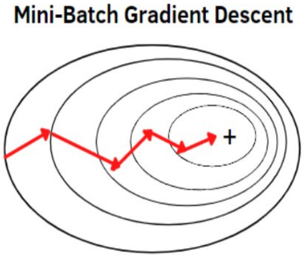
Backpropagation

$$w_{t+1} = w_t - \eta \nabla w_t$$
$$b_{t+1} = b_t - \eta \nabla b_t$$
$$where \nabla w_t = \frac{\partial L(w,b)}{\partial w}_{w=w_t, b=b_t}, \nabla b_t = \frac{\partial L(w,b)}{\partial b}_{w=w_t, b=b_t}$$

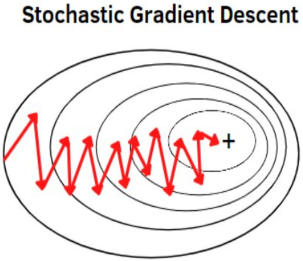
Gradient Descent Update Rule



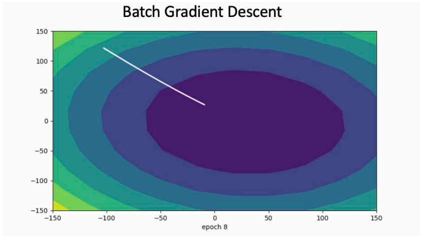
all training sample
each epoch

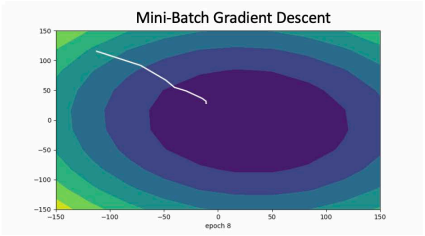


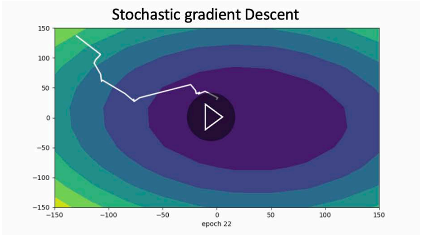
small random subset
each epoch



one random training
sample each epoch







See another PPT

Backpropagation Algorithm

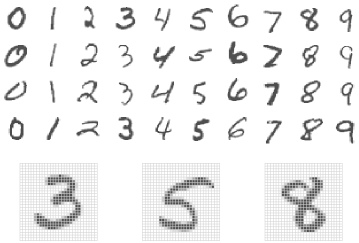
Shree K. Nayar

Columbia University

Topic: Neural Networks, Module: Perception

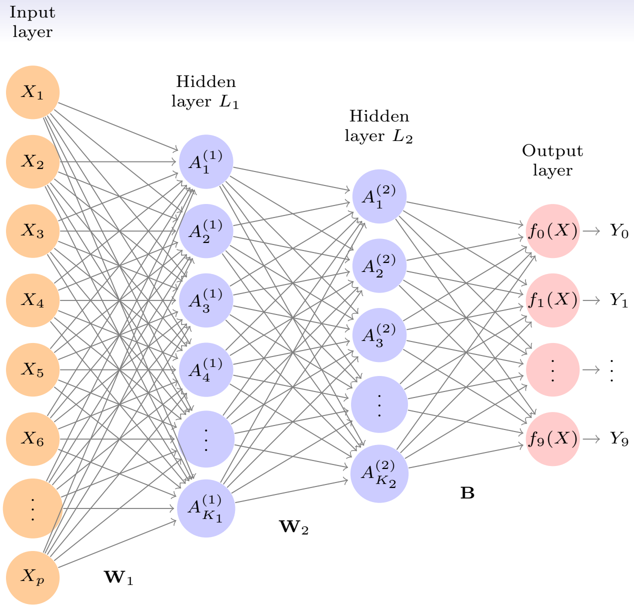
First Principles of Computer Vision

Example: MNIST Digits



Handwritten digits
28×28 grayscale images
$60K$ train, $10K$ test images
Features are the 784 pixel grayscale values $\in (0, 255)$
Labels are the digit class 0–9

- Goal: build a classifier to predict the image class.
- We build a two-layer network with 256 units at first layer, 128 units at second layer, and 10 units at output layer.
- Along with intercepts (called *biases*) there are 235,146 parameters (referred to as *weights*)


Input layer
X_1
X_2
X_3
X_4
X_5
X_6
$\vdots$
X_p
Hidden layer L_1
$A_1^{(1)}$
$A_2^{(1)}$
$A_3^{(1)}$
$A_4^{(1)}$
$\vdots$
$A_{K_1}^{(1)}$
Hidden layer L_2
$A_1^{(2)}$
$A_2^{(2)}$
$A_3^{(2)}$
$\vdots$
$A_{K_2}^{(2)}$
Output layer
$f_0(X)$
$f_1(X)$
$\vdots$
$f_9(X)$
Y_0
Y_1
$\vdots$
Y_9
$\mathbf{W}_1$
$\mathbf{W}_2$
$\mathbf{B}$

Details of Output Layer

- Let $Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)},\ m = 0, 1, \ldots, 9$ be 10 linear combinations of activations at second layer.
- Output activation function encodes the *softmax* function

$$f_m(X) = \Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_\ell}}.$$

Details of Output Layer

- Let $Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}$, $m = 0, 1, \ldots, 9$ be 10 linear combinations of activations at second layer.
- Output activation function encodes the *softmax* function

$$f_m(X) = \Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_\ell}}.$$

- We fit the model by minimizing the negative multinomial log-likelihood (or cross-entropy):

$$-\sum_{i=1}^{n} \sum_{m=0}^{9} y_{im} \log(f_m(x_i)).$$

- y_{im} is 1 if true class for observation i is m, else 0 — i.e. *one-hot encoded*.

Results

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

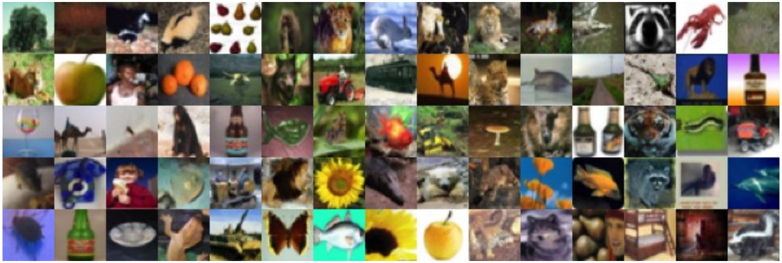
- Early success for neural networks in the 1990s.
- With so many parameters, regularization is essential.
- Some details of regularization and fitting will come later.

Results

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

- Early success for neural networks in the 1990s.
- With so many parameters, regularization is essential.
- Some details of regularization and fitting will come later.
- Very overworked problem — best reported rates are $< 0.5\%$!
- Human error rate is reported to be around 0.2%, or 20 of the 10K test images.

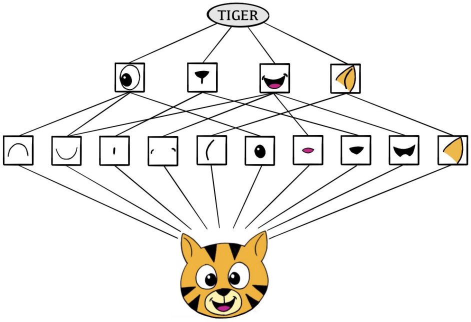
Convolutional Neural Network — CNN



- Major success story for classifying images.
- Shown are samples from CIFAR100 database. 32×32 color natural images, with 100 classes.
- $50K$ training images, $10K$ test images.

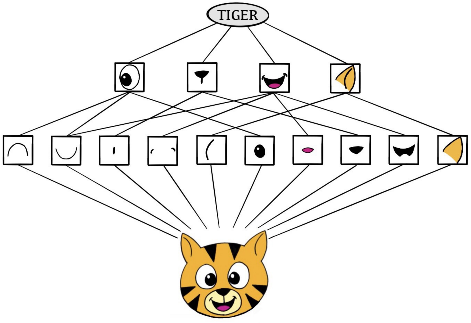
Each image is a three-dimensional array or *feature map:* $32 \times 32 \times 3$ array of 8-bit numbers. The last dimension represents the three color channels for red, green and blue.

How CNNs Work



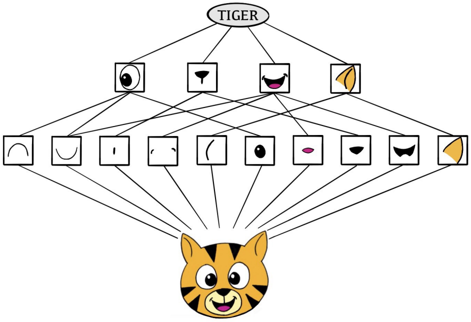
- The CNN builds up an image in a hierarchical fashion.

How CNNs Work



- The CNN builds up an image in a hierarchical fashion.
- Edges and shapes are recognized and pieced together to form more complex shapes, eventually assembling the target image.

How CNNs Work



- The CNN builds up an image in a hierarchical fashion.
- Edges and shapes are recognized and pieced together to form more complex shapes, eventually assembling the target image.
- This hierarchical construction is achieved using *convolution* and *pooling* layers.

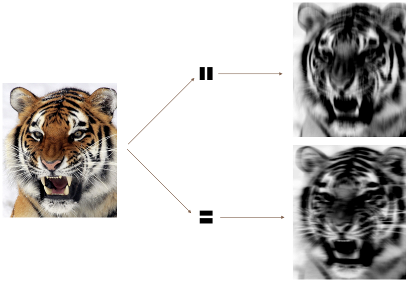
Convolution Filter

$$\text{Input Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix} \quad \text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$$

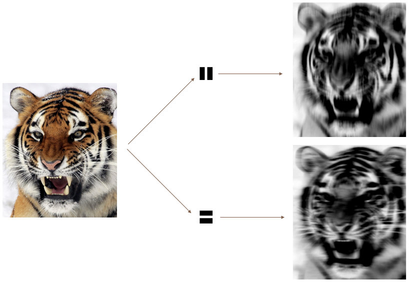
- The filter is itself an image, and represents a small shape, edge etc.
- We slide it around the input image, scoring for matches.
- The scoring is done via *dot-products*, illustrated above.
- If the subimage of the input image is similar to the filter, the score is high, otherwise low.
- The filters are *learned* during training.

Convolution Example



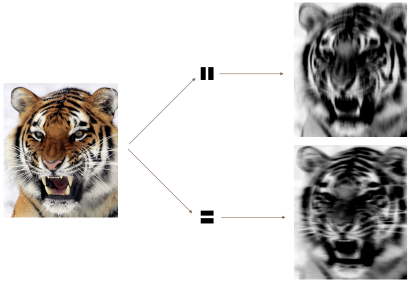
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.

Convolution Example



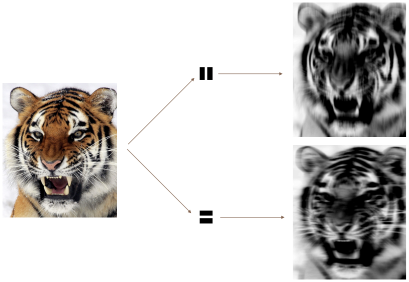
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.

Convolution Example



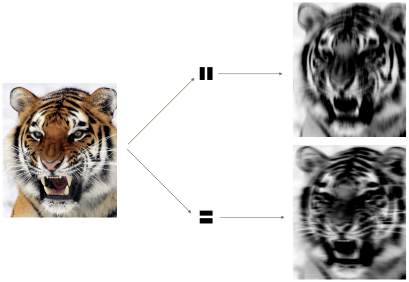
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.
- The result of the convolution is a new feature map.

Convolution Example



- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.
- The result of the convolution is a new feature map.
- Since images have three colors channels, the filter does as well: one filter per channel, and dot-products are summed.

Convolution Example



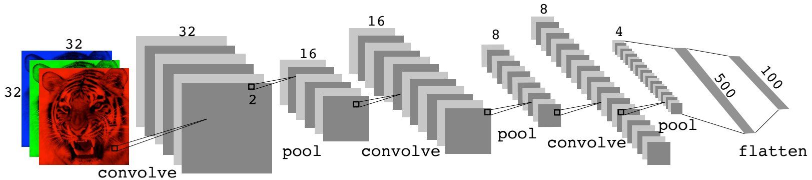
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.
- The result of the convolution is a new feature map.
- Since images have three colors channels, the filter does as well: one filter per channel, and dot-products are summed.
- The weights in the filters are *learned* by the network.

Pooling

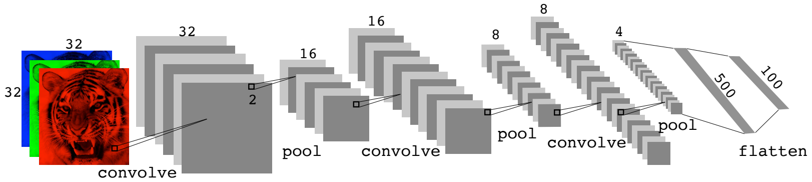
$$\text{Max pool} \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \to \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

- Each non-overlapping 2×2 block is replaced by its maximum.
- This sharpens the feature identification.
- Allows for locational invariance.
- Reduces the dimension by a factor of 4 — i.e. factor of 2 in each dimension.

Architecture of a CNN

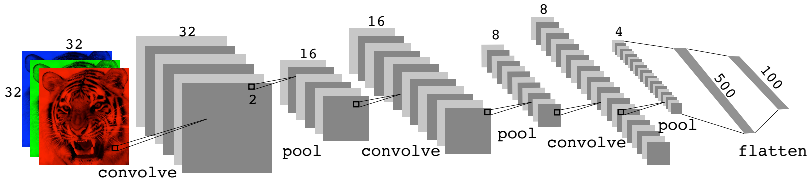


Architecture of a CNN



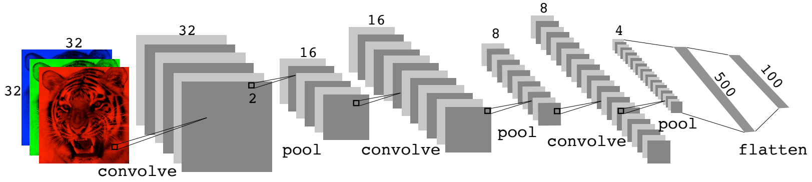
- Many convolve + pool layers.

Architecture of a CNN



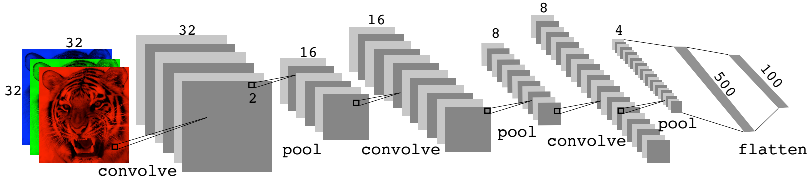
- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3.

Architecture of a CNN



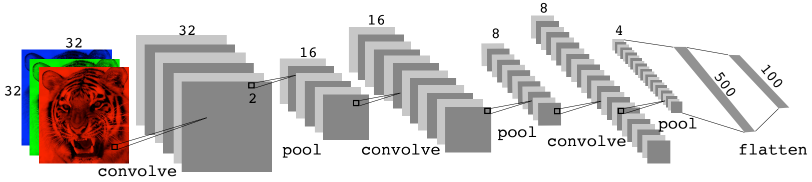
- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3.
- Each filter creates a new channel in convolution layer.

Architecture of a CNN



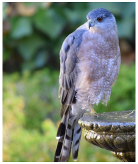
- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3.
- Each filter creates a new channel in convolution layer.
- As pooling reduces size, the number of filters/channels is typically increased.

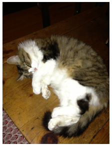
Architecture of a CNN



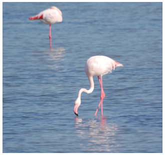
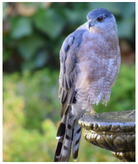
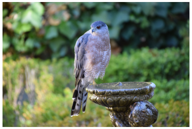
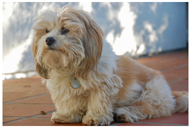
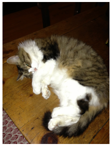
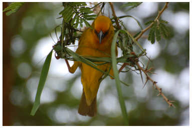
- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3.
- Each filter creates a new channel in convolution layer.
- As pooling reduces size, the number of filters/channels is typically increased.
- Number of layers can be very large. E.g. `resnet50` trained on `imagenet` 1000-class image data base has 50 layers!

Using Pretrained Networks to Classify Images





Using Pretrained Networks to Classify Images

flamingo		Cooper's hawk		Cooper's hawk	
flamingo	0.83	kite (raptor)	0.60	fountain	0.35
spoonbill	0.17	great grey owl	0.09	nail	0.12
white stork	0.00	robin	0.06	hook	0.07
Lhasa Apso		**cat**		**Cape weaver**	
Tibetan terrier	0.56	Old English sheepdog	0.82	jacamar	0.28
Lhasa	0.32	Shih-Tzu	0.04	macaw	0.12
cocker spaniel	0.03	Persian cat	0.04	robin	0.12

Here we use the 50-layer `resnet50` network trained on the 1000-class `imagenet` corpus to classify some photographs.

JAMES | Journal of Advances in Modeling Earth Systems®

RESEARCH ARTICLE
10.1029/2022MS003596

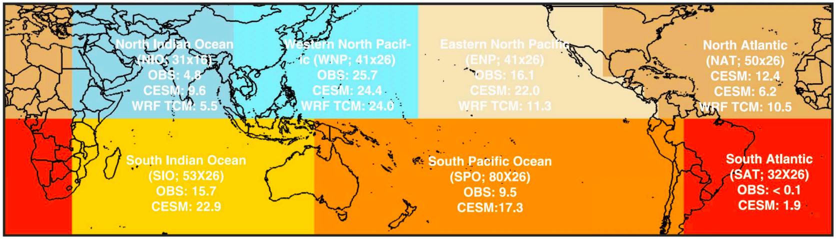
Special Section:
Machine learning application to Earth system modeling

Using Convolutional Neural Network to Emulate Seasonal Tropical Cyclone Activity

Dan Fu[1], Ping Chang[1,2], and Xue Liu[1]

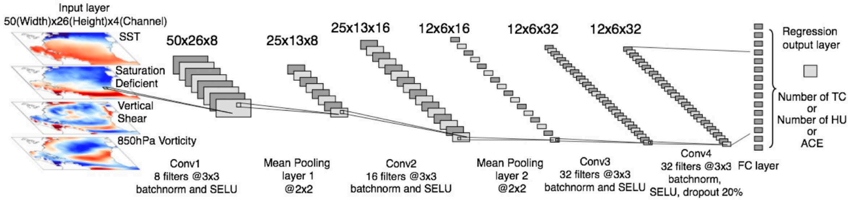
[1]Department of Oceanography, Texas A&M University, College Station, TX, USA, [2]Department of Atmospheric Sciences, Texas A&M University, College Station, TX, USA

a



North Indian Ocean
(NIO; 31x16)
OBS: 4.8
CESM: 9.6
WRF TCM: 5.5
Western North Pacif-ic (WNP; 41x26)
OBS: 25.7
CESM: 24.4
WRF TCM: 24.0
Eastern North Pacific
(ENP; 41x26)
OBS: 16.1
CESM: 22.0
WRF TCM: 11.3
North Atlantic
(NAT; 50x26)
CESM: 12.4
CESM: 6.2
WRF TCM: 10.5
South Indian Ocean
(SIO; 53X26)
OBS: 15.7
CESM: 22.9
South Pacific Ocean
(SPO; 80X26)
OBS: 9.5
CESM:17.3
South Atlantic
(SAT; 32X26)
OBS: < 0.1
CESM: 1.9

b



Input layer
50(Width)x26(Height)x4(Channel)
SST
Saturation Deficient
Vertical Shear
850hPa Vorticity
50x26x8
25x13x8
25x13x16
12x6x16
12x6x32
12x6x32
Regression output layer
Number of TC or Number of HU or ACE
Conv1 8 filters @3x3 batchnorm and SELU
Mean Pooling layer 1 @2x2
Conv2 16 filters @3x3 batchnorm and SELU
Mean Pooling layer 2 @2x2
Conv3 32 filters @3x3 batchnorm and SELU
Conv4 32 filters @3x3 batchnorm, SELU, dropout 20%
FC layer

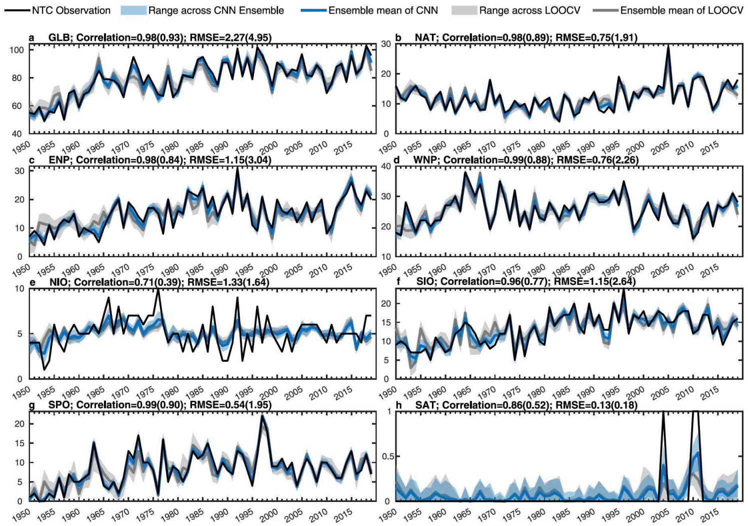


Figure 2. Time-series of seasonal mean number of TC (NTC) for (a) global (defined as the sum of individual ocean basins), (b) NAT, (c) ENP, (d), WNP, (e) NIO, (f) SIO, (g) SPO and (h) SAT. Observed NTC is shown in black line. Ensemble mean CNN emulated and LOOCV results are shown in blue and gray lines, respectively. Ranges across CNN ensembles are shown in shadings. Numbers shown in each panel denote the Pearson correlation coefficients and root mean square errors (RMSE) between observation and CNN model results. Leave-one-out cross validation (LOOCV) results are listed in the parentheses.

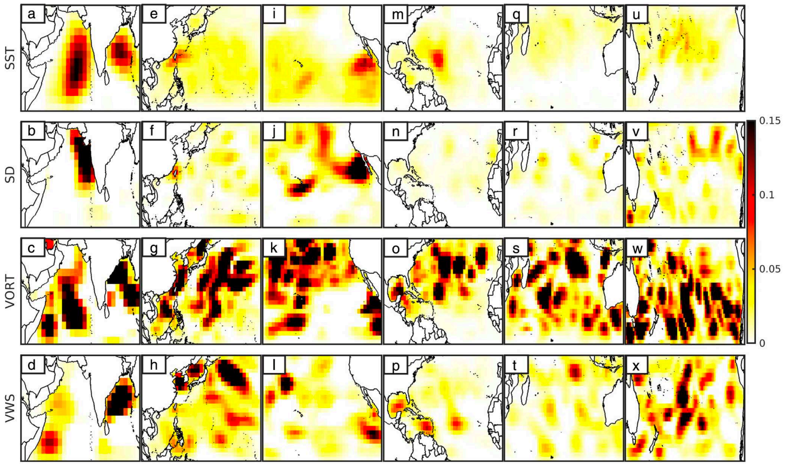


Figure 5. Occlusion sensitivity maps that highlight the relative importance in emulating seasonal NTC in different ocean basins. (a–d): Relative importance of SST, saturation deficit (SD), 850 hPa vorticity and vertical wind shear in NIO, respectively. (e–h), (i–l), (m–p), (q–t), And (u–x) are similar, but for the relative importance of 4 variables in WNP, ENP, NAT, SIO, and SPO, respectively. Areas in the map with higher values correspond to regions of input variables that contribute more significantly to impact the CNN prediction skills. Intuitively, the sensitivity map shows which area most affect the prediction RMSE when changed. Refer main text for details.

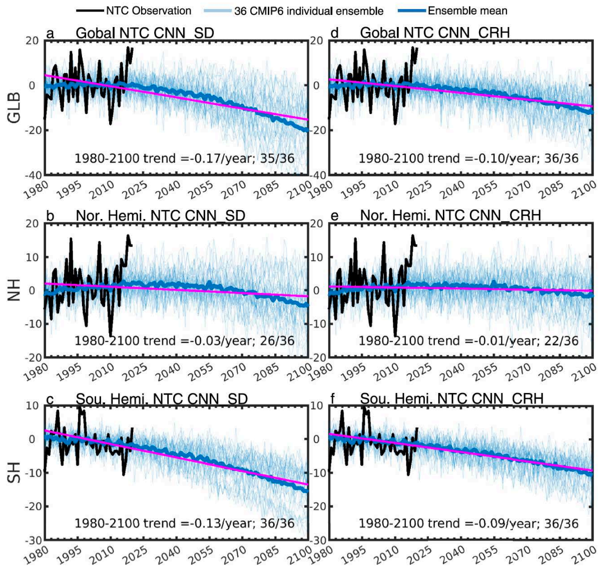


Figure 11. Time-series of CNN_SD emulated anomalous NTC for (a) global, (b) Northern Hemisphere and (c) Southern Hemisphere integration with the large-scale environmental conditions projected by 36 different Coupled Model Intercomparison Project (CMIP6) model under historical forcing and shared socio-economic pathway 5–8.5 (SSP585). Anomalies are computed as the departures from their 1980–1999 climatology. Black lines denote observation during 1980–2020, thick dark blue lines denote the multi-model mean of the CMIP6 models, and thin light blue lines denote the individual 36 CMIP6 model. Linear trends during 1980–2100 are plotted in magenta and listed in each panel; following by the fraction demonstrating number of individual models showing consistent sign of trend as to the multi-model mean. For example, 35 of 36 CMIP6 model project a decreasing trend with the mean trend of –0.17 per year emulated by the CNN_SD model. Note that, all multi-model mean linear trends are significant at 95% confidence level based on the Mann-Kendall trend test. (e–f) Are similar, but for the CNN_CRH emulated NTC projection.