Overview of Modern ML Applications: Convolutional Neural Network (CNN)

Learning objectives

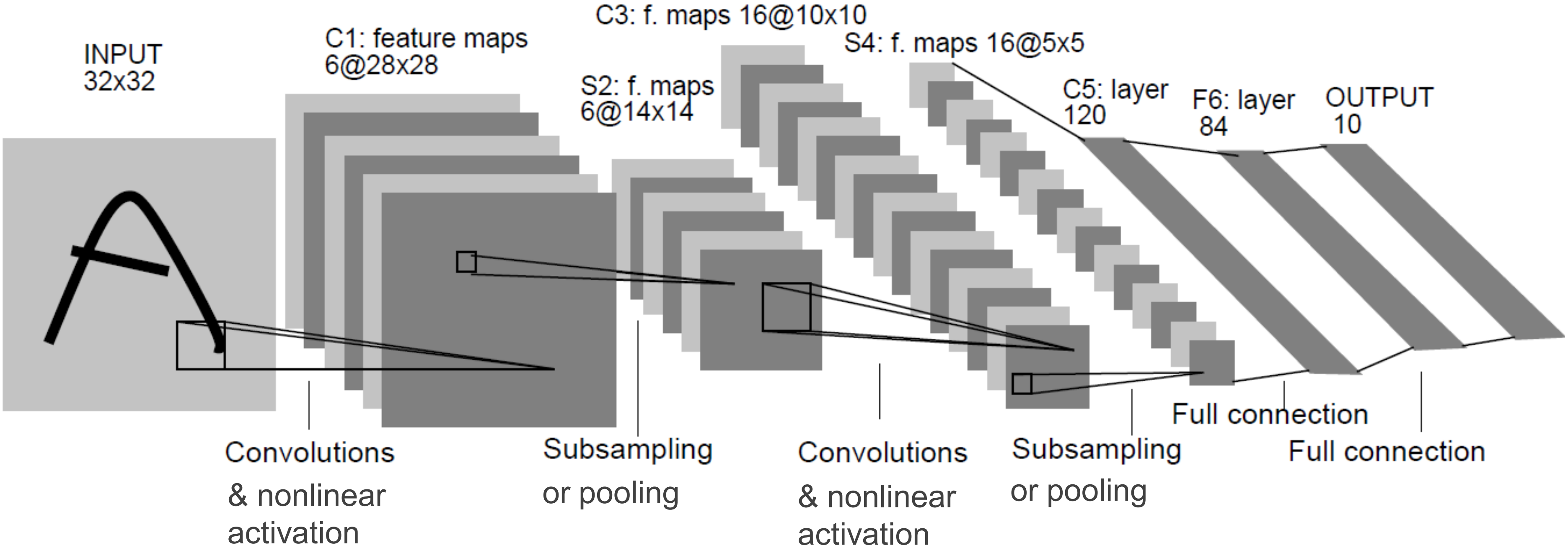
- Describe the structure of CNN
- Build and train simple CNNs using a deep learning package

(Ref: Ch 9 of Goodfellow et al. 2016)

Acknowledgment: Some graphics and slides were adapted from Stanford's CS231n by Fei-Fei Li et al.: http://cs231n.stanford.edu/

Convolutional Neural Network (CNN)

The **single** most important technology that fueled the rapid development of ***deep learning*** and ***big data*** in the past decade.

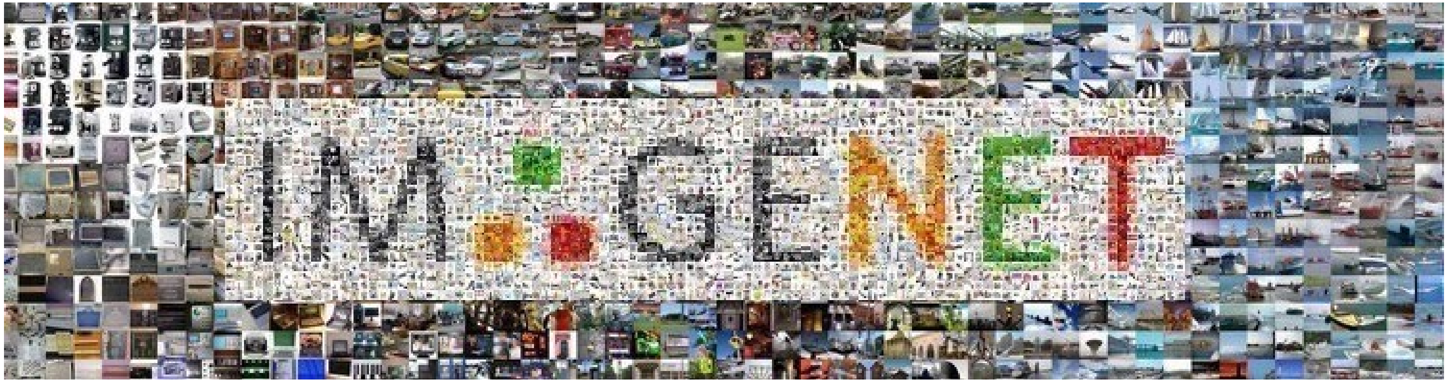


LeCun, Bottou, Bengio, Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proc. IEEE*, **1998**.

Why is Deep Learning so Successful?

1. **Improved model:** convolutional layer, more layers ("deep"), simpler activation (i.e., ReLU), skip/residual connection (i.e., ResNet), attention (i.e., Transformer)
2. **Big data:** huge dataset, transfer learning
3. **Powerful computation:** graphical processing units (GPUs)

◆ Example of big data: ImageNet (22K categories, 15M images)



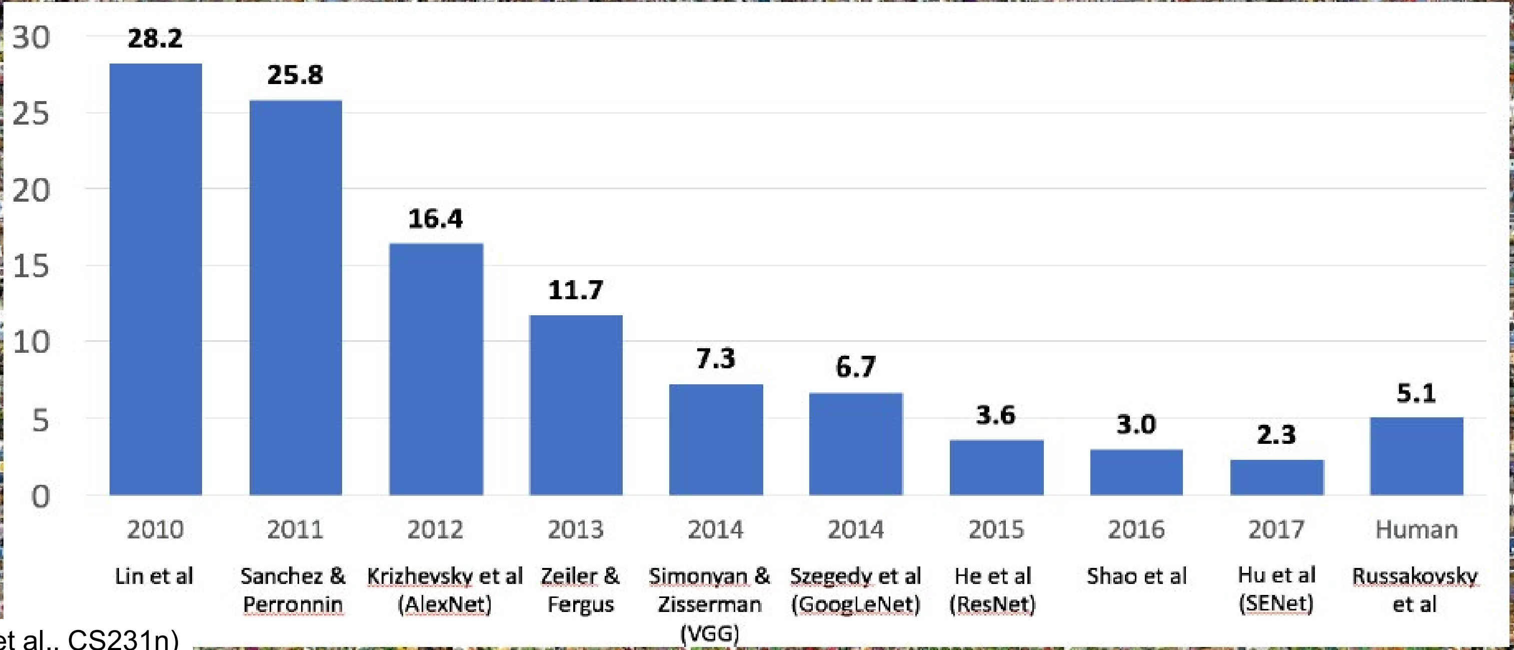
Deng, Dong, Socher, Li, Li & Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," *IEEE CVPR*, 2009.

IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:

1,000 object classes

1,431,167 images



(Fei-Fei Li et al., CS231n)

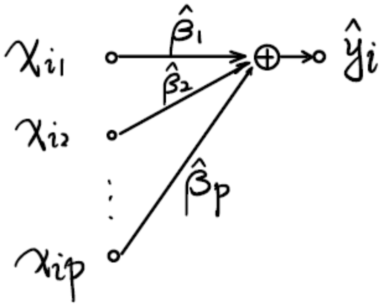
Linear Model to Neural Network

Recall linear model w/ multiple predictors / features / inputs.

$$\underbrace{y_i}_{\text{true output}} = \sum_{j=1}^{p} x_{ij}\beta_j + e_i = \underbrace{[\beta_1, \dots, \beta_p]}_{\text{true weights}} \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix} + e_i \quad , i=1, \dots, n.$$

$$\underbrace{\hat{y}_i}_{\text{predicted output}} = \sum_{j=1}^{p} x_{ij}\hat{\beta}_j = \underbrace{[\hat{\beta}_1, \dots, \hat{\beta}_p]}_{\text{estimated weights}} \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix} \quad , i=n+1, \dots, n+m.$$

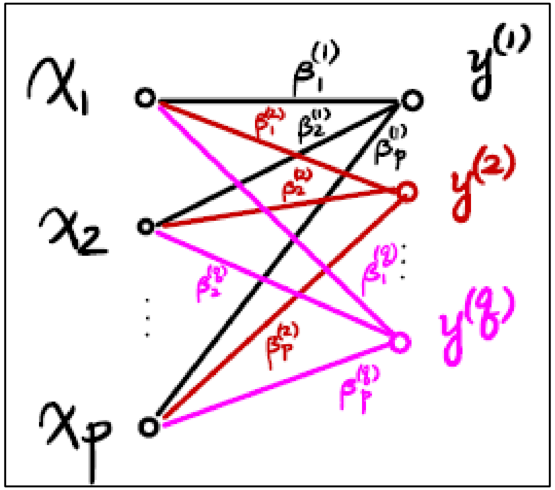
Graphically we have:



① Use multiple linear models

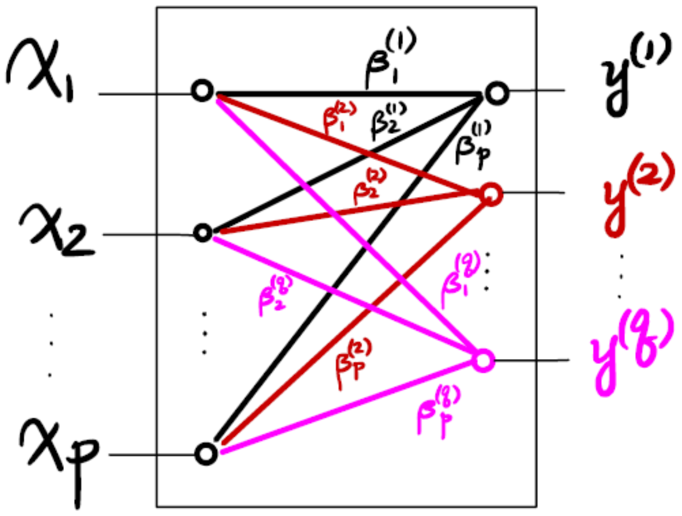
$\longrightarrow$

② Simplify the notations.



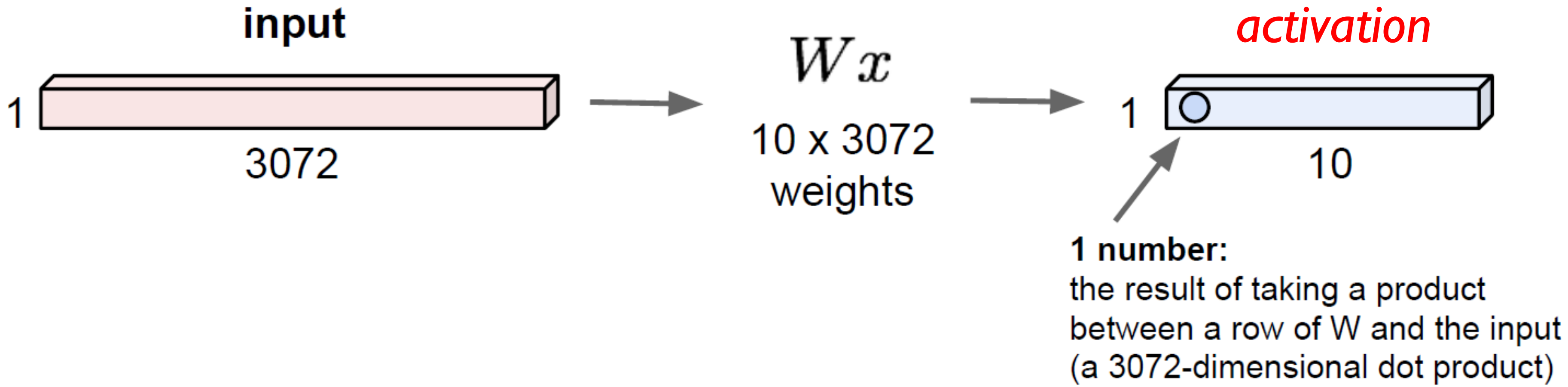
Fully-Connected Layer for 1D Signal

$$\underbrace{\left[\begin{array}{c} y^{(1)} \\ \hline y^{(2)} \\ \hline \vdots \\ \hline y^{(q)} \end{array}\right]}_{\substack{\text{layer} \\ \text{output,} \\ \underset{\sim}{y} \in \mathbb{R}^{q}}} = \underbrace{\left[\begin{array}{cccc} \beta_1^{(1)} & \beta_2^{(1)} & \cdots & \beta_p^{(1)} \\ \hline \beta_1^{(2)} & \beta_2^{(2)} & \cdots & \beta_p^{(2)} \\ \hline \vdots & & & \vdots \\ \hline \beta_1^{(q)} & \beta_2^{(q)} & \cdots & \beta_p^{(q)} \end{array}\right]}_{\substack{\text{dense weight} \\ \text{matrix} \\ B \in \mathbb{R}^{q \times p}}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \\ x_p \end{bmatrix}}_{\substack{\text{layer input,} \\ \underset{\sim}{x} \in \mathbb{R}^{p}}}$$



Fully-Connected Layer for RGB Image

32x32x3 image -> stretch to 3072 x 1



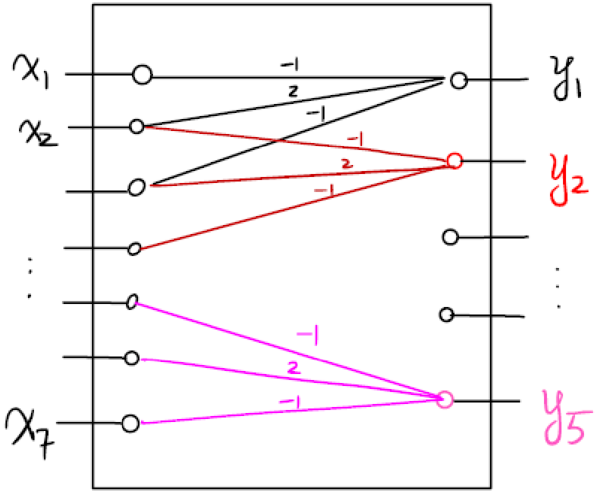
1 number:
the result of taking a product between a row of W and the input (a 3072-dimensional dot product)

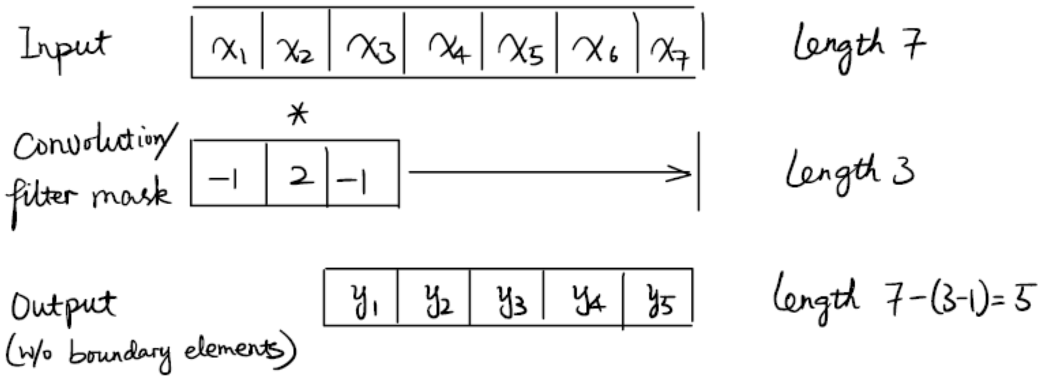
(Fei-Fei Li et al., CS231n)

Convolutional Layer for 1D Signal

$$\begin{bmatrix} y_1 \\ \vdots \\ y_5 \end{bmatrix} = \begin{bmatrix} -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ \vdots & & & -1 & 2 & -1 & \vdots \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_7 \end{bmatrix}$$

Sparse weight matrix





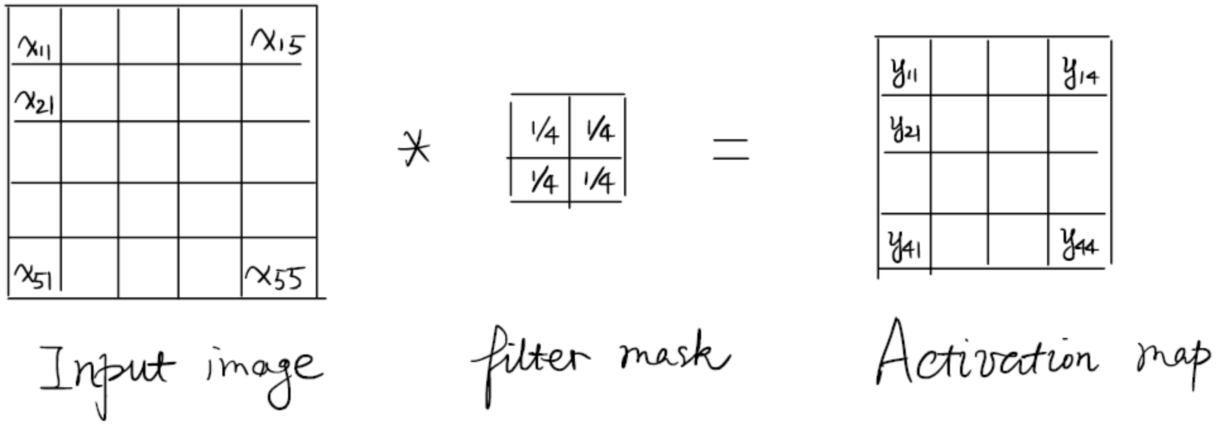
Input x_1 | x_2 | x_3 | x_4 | x_5 | x_6 | x_7 length 7

$*$

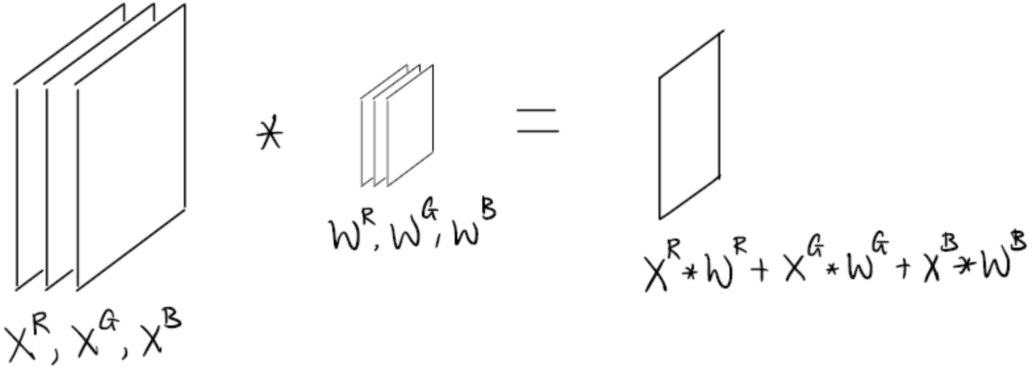
Convolution filter mask -1 | 2 | -1 length 3

Output (w/o boundary elements) y_1 | y_2 | y_3 | y_4 | y_5 length 7-(3-1)=5

Convolutional Layer for 2D Matrix/Image

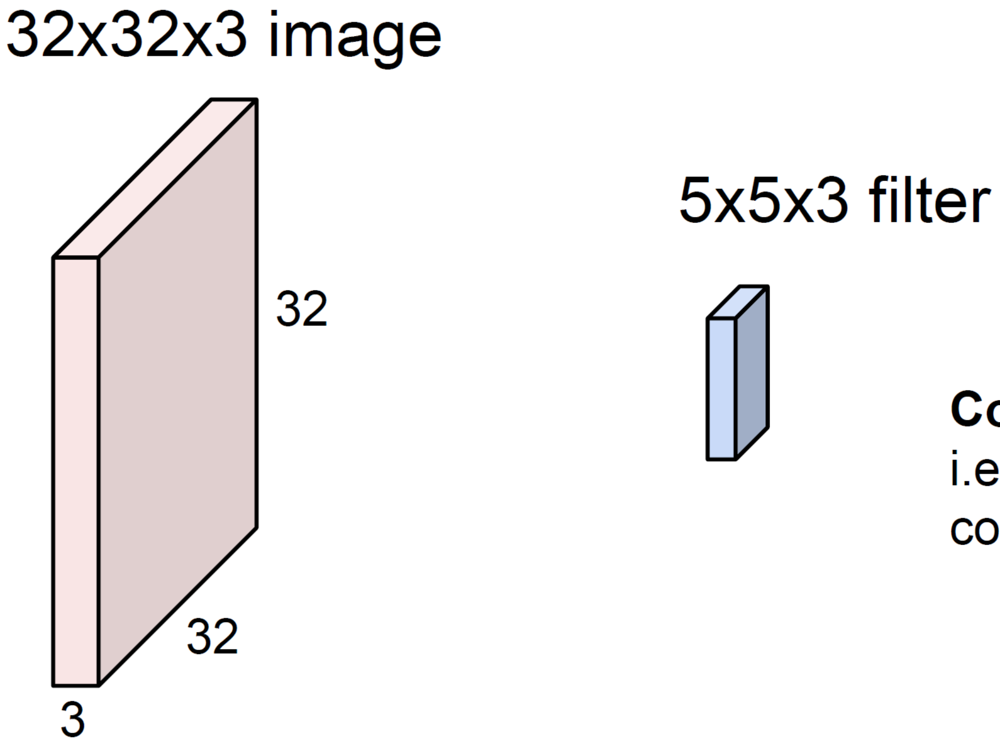


2D Convolution



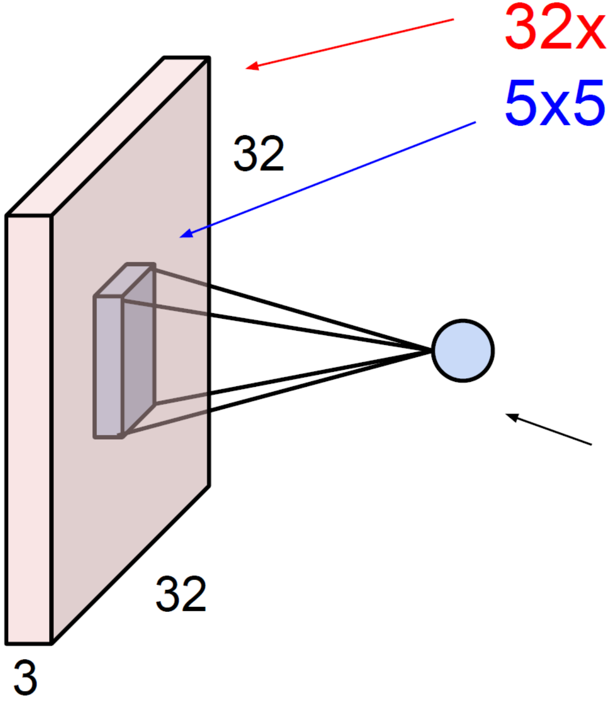
Multiple color channels
need multiple filter masks

Convolutional Layer for RGB Image



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

(Fei-Fei Li et al., CS231n)



32x32x3 image

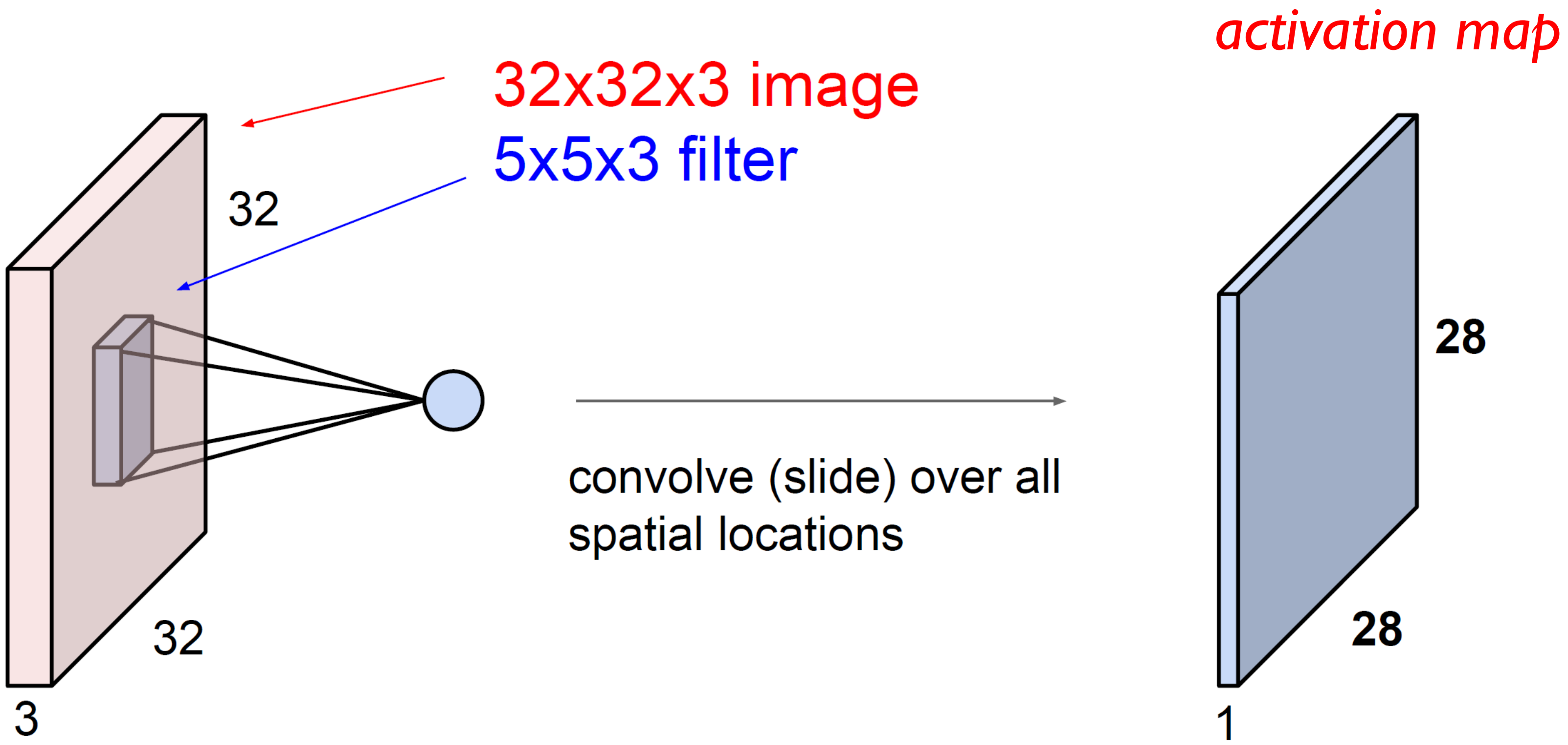
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

(Fei-Fei Li et al., CS231n)

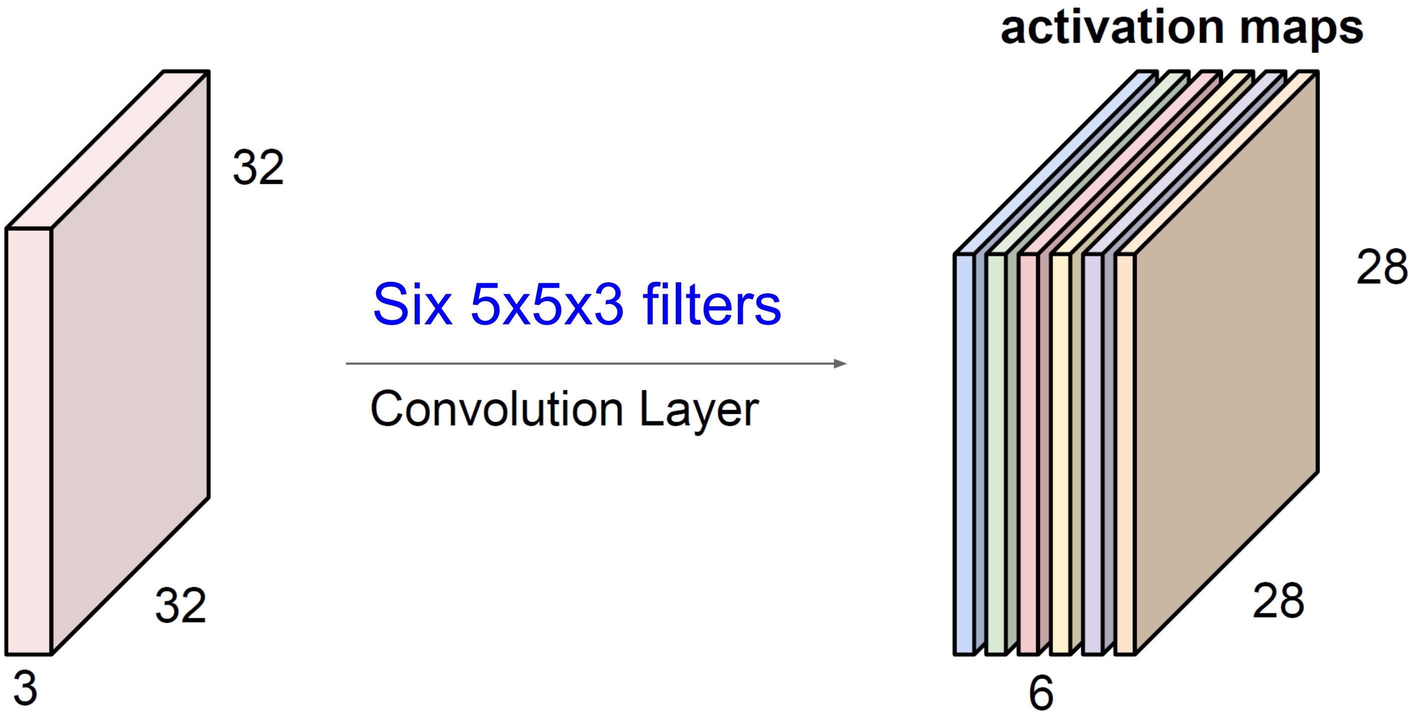
A closer look at spatial dimensions:



convolve (slide) over all spatial locations

(Fei-Fei Li et al., CS231n)

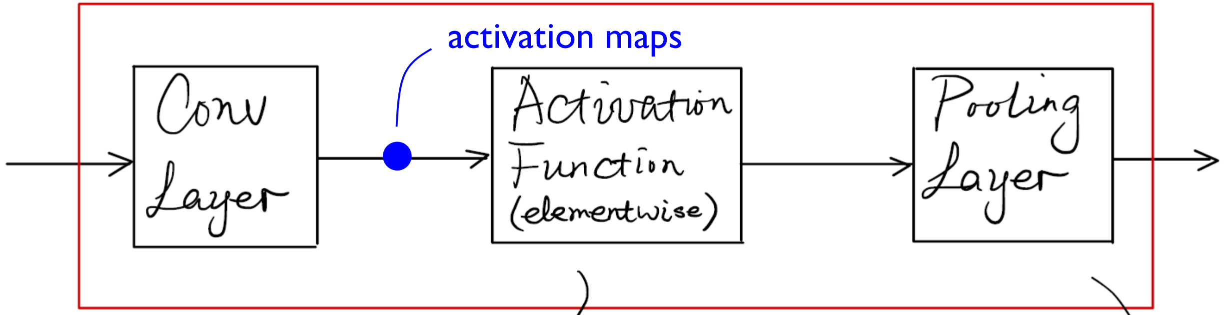
For example, if we had six 5x5 filters, we'll get six separate *activation maps*:



We stack these up to get a "new image" of size 28x28x6!

(Fei-Fei Li et al., CS231n)

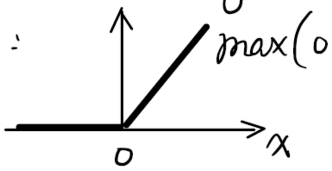
Building Block for Modern CNN



(Rectified Linear Unit)

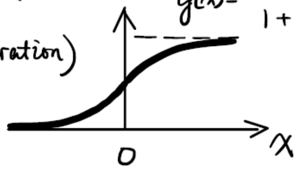
ReLU: (modern)

$g(x) = \max(0, x)$



Logistic: (prior generation)

$g(x) = \frac{1}{1+e^{-x}}$



Ex:

x_{11}	x_{12}
x_{21}	x_{22}

: $\{x_{ij}\}_{i,j=1}^{2}$

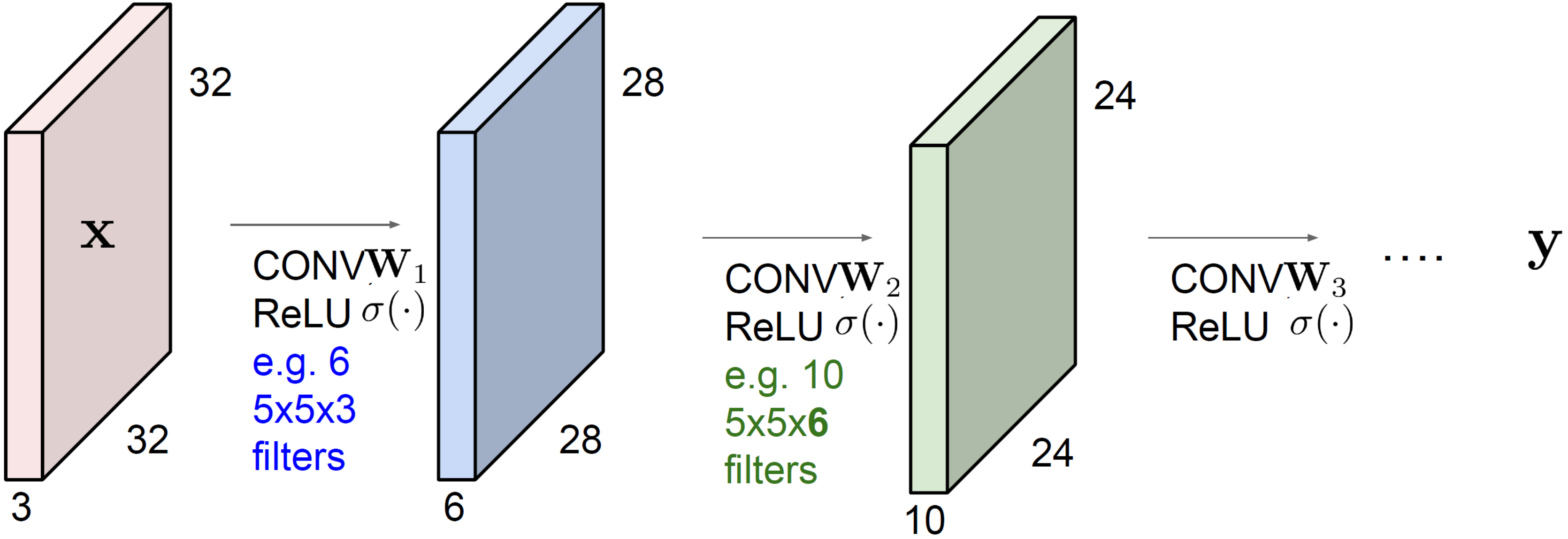
Max pooling:

$$g(\{x_{ij}\}) = \max(\{x_{ij}\})$$

Average pooling:

$$g(\{x_{ij}\}) = \frac{1}{|\{x_{ij}\}|} \sum_{i,j} x_{ij}$$

CNN is composed of a sequence of convolutional layers, interspersed with activation functions (ReLU, in most cases).



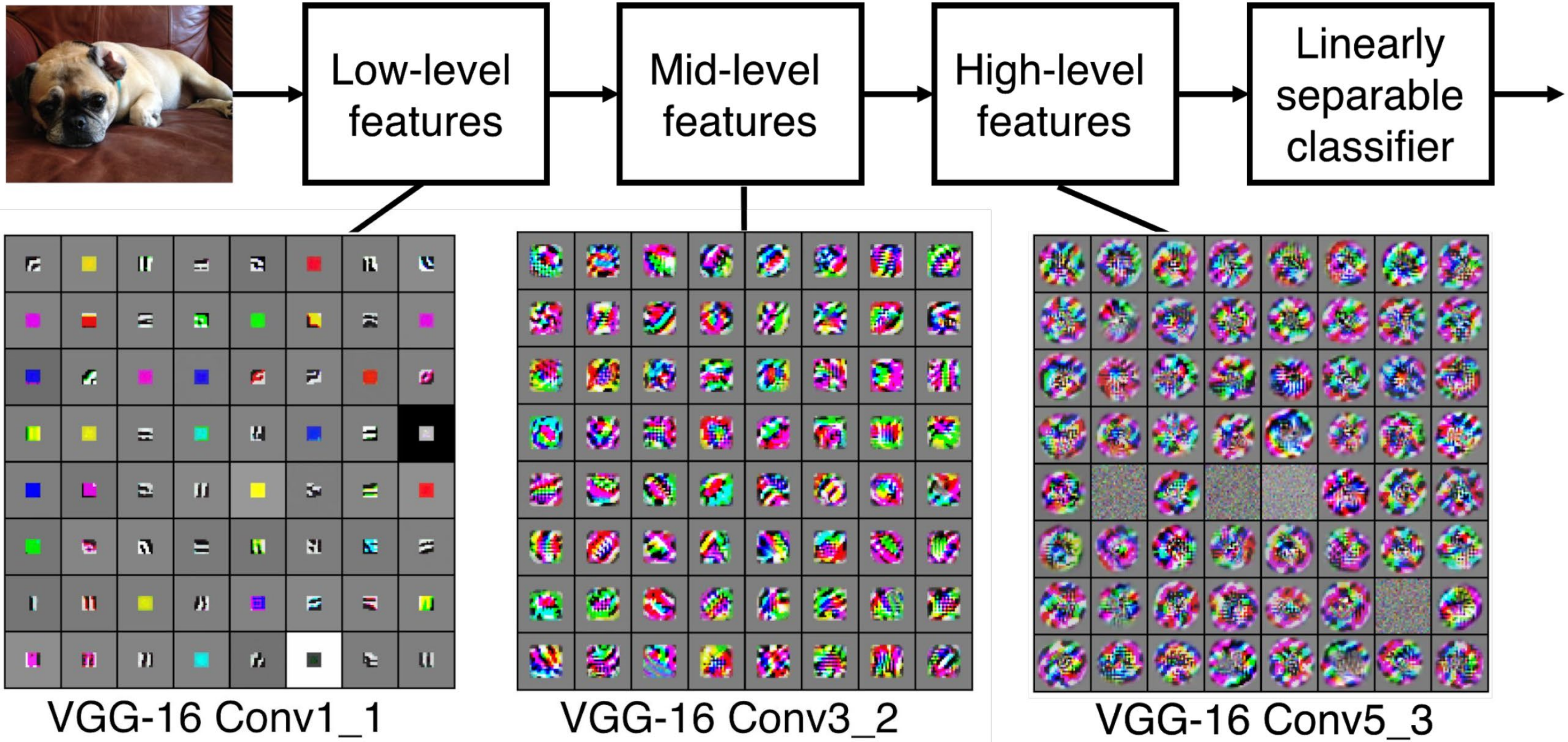
$$\mathbf{y} = \cdots \sigma\Big(\mathbf{W}_3\, \sigma\big(\mathbf{W}_2\, \sigma(\mathbf{W}_1 \mathbf{x})\big)\Big) \cdots$$

Source of nonlinearity, ReLU: $\sigma(x) := \max(0, x)$

(Fei-Fei Li et al., CS231n)

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

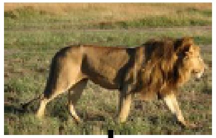


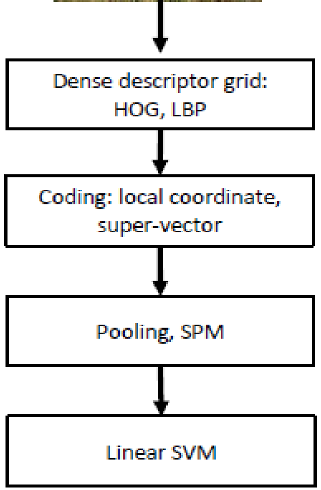
(Fei-Fei Li et al., CS231n)

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC





Dense descriptor grid: HOG, LBP

Coding: local coordinate, super-vector

Pooling, SPM

Linear SVM

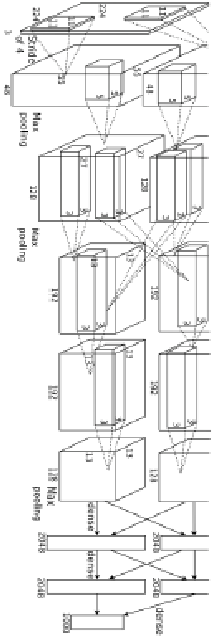
[Lin CVPR 2011]

Lion image by Swissfrog is licensed under CC BY 3.0

AlexNet

Year 2012

SuperVision

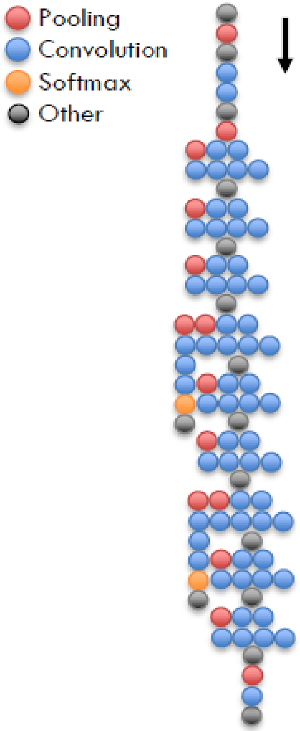


[Krizhevsky NIPS 2012]

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Year 2014

GoogLeNet



[Szegedy arxiv 2014]

VGG

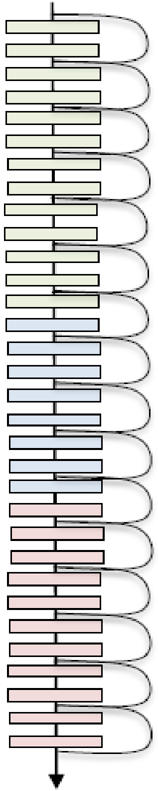
Image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
fc-4096
fc-4096
fc-1000
softmax

[Simonyan arxiv 2014]

ResNet

Year 2015

MSRA

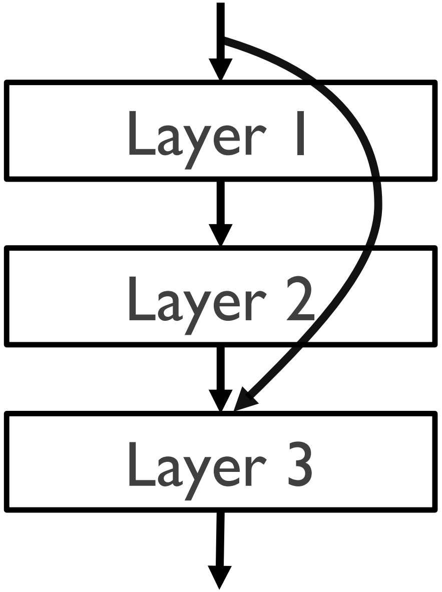


[He ICCV 2015]

(Fei-Fei Li et al., CS231n)

Residual Neural Network (ResNet) (Kaiming He et al., 2015)

- ***Skip connections*** or ***shortcuts*** are added.
- They can
  - avoid "vanishing gradients", and
  - make optimization landscape flatter.
- From Taylor expansion perspective, the neural network only learns the higher-order error terms beyond the linear term $\mathbf{x}$.
- Has interpretations in PDE.
- Preferred modern NN structure.

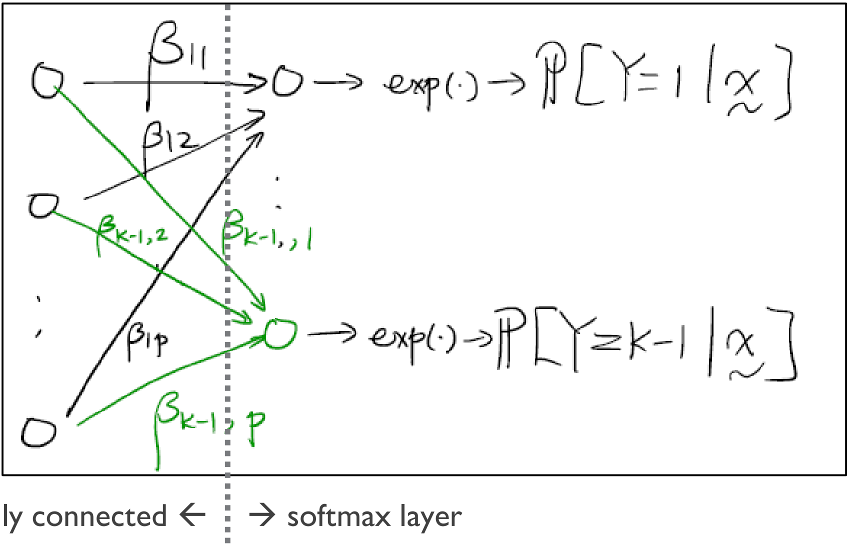


$$\mathbf{y} = \cdots \sigma\Big(\mathbf{W}_3 \underbrace{\big[\mathbf{x}+\sigma\big(\mathbf{W}_2\,\sigma(\mathbf{W}_1\mathbf{x})\big)\big]}_{\mathrm{g}(\mathbf{x})}\Big)\cdots$$

When Output is Categorical / Qualitative

- ◆ A ***softmax layer*** is needed:
- ◆ *Softmax function:*

$$\sigma_i(\underset{\sim}{z}) = \frac{e^{\beta z_i}}{\sum_{j=1}^{K} e^{\beta z_j}}$$



conv/fully connected ← → softmax layer

- ◆ Ex:

$$K=2. \quad \sigma_1 = \frac{e^{\beta z_1}}{e^{\beta z_1} + e^{\beta z_2}}$$

$$= \frac{1}{1+e^{\beta(z_2-z_1)}}$$

When β very large,

$z_2 > z_1$ leads to $\begin{cases} \sigma_1 = 0 \\ \sigma_2 = 1 \end{cases}$

Winner takes all!

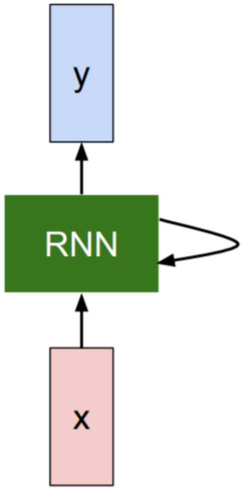
Other Essential Aspects of CNN

- Due to time constraints, this overview lecture covered only the structural elements of CNNs. Other essential aspects are:
    - Cost function/loss, e.g., MSE, cross entropy.
    - How to train CNNs or estimate the weights (will only give practice code), i.e., *backpropagation* (will cover in the next two lectures).
    - Practical training considerations including
        - How to determine *number of hidden units/channels* to be used,
        - How to tune *learning rate* and *batch size*, and
        - When to stop training (number of *epochs*).
- For a more complete treatment on CNN, refer to the dedicate courses such as CS231n CNNs for Visual Recognition or ECE 542/492.

Machine Learning (ML) and Data Science (DS)

- Follow-up machine learning / data science courses:
    - ECE 542/492 Neural Networks (S'23)
    - ECE 792-□ Advanced Topics in Machine Learning (S'23)
    - ECE 592-61 Data Science (each fall)
    - ECE 759 Pattern Recognition and Machine Learning (S'24)
    - ECE 763 Computer Vision (S'24)
    - Any courses/videos on YouTube, Coursera, etc.
- State-of-the-art theory & applications: ICML, NeurIPS, ICLR, AAAI
- Data science competitions: kaggle.com
- Programming languages for ML/DS: Python, R, Matlab

Overview of Modern ML Applications: Recurrent Neural Network (RNN) and LSTM



Acknowledgment: Some graphics and slides were adapted from Stanford's CS231n by Fei-Fei Li et al.: http://cs231n.stanford.edu/

A Sequence of Identical Neural Network Modules

Force the neural nets (in green) to be the same to lower the complexity!

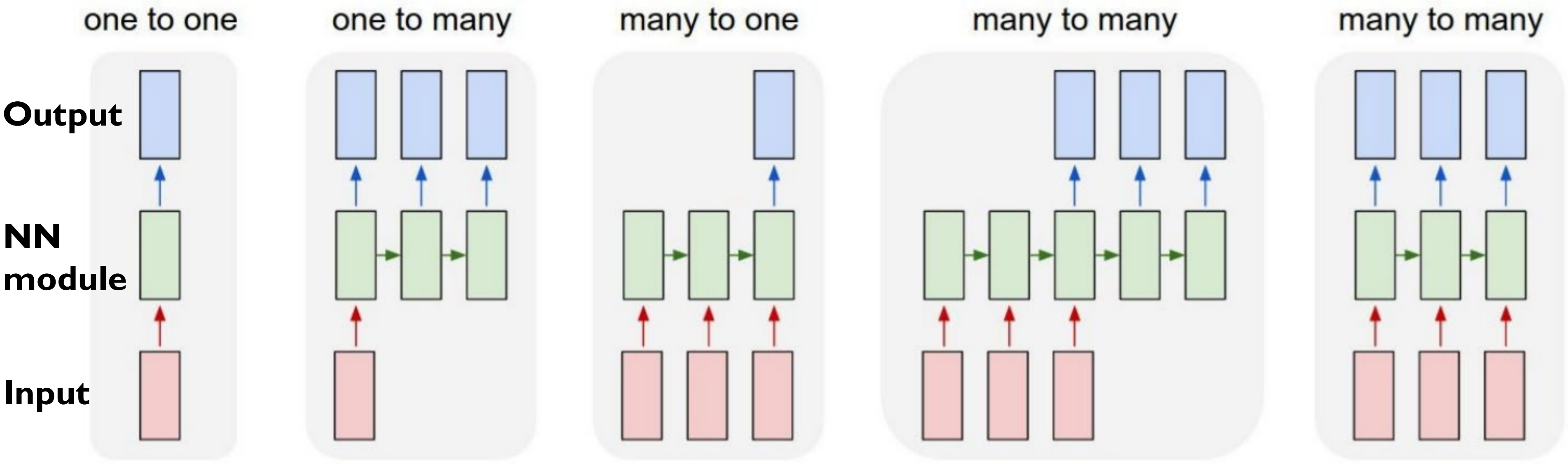


Image Captioning
image -> seq of words

Emotion Classification
seq of words -> emotion

Machine Translation
seq of words -> seq of words

Video classification
at frame level

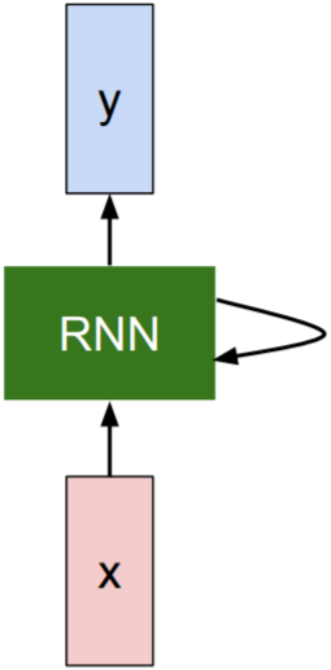
Recurrent Neural Network (RNN): Definition

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

h_t: state, x_t: input
f_W: neural network

$$y_t = W_{hy} h_t$$



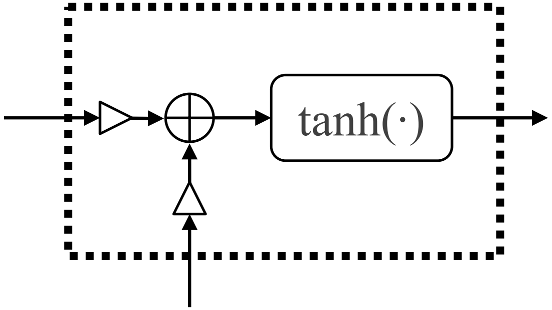
Recurrent Neural Network (RNN): Implementation

$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

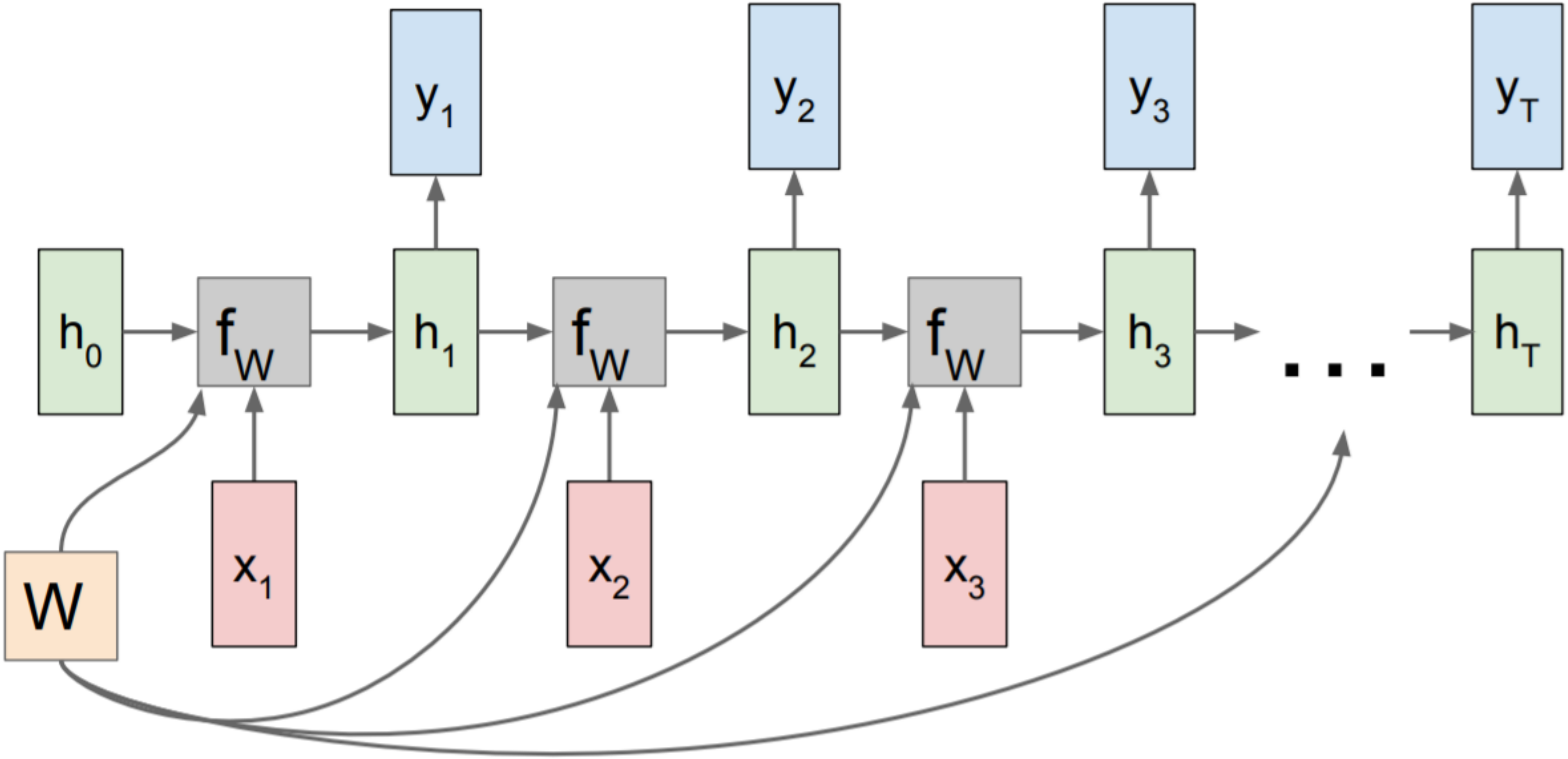
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Diagram for f_W.
Can you label the details?



f_W is implemented via
- linear transforms W_{hh} and W_{xh} and
- elementwise nonlinear function $\tanh(\cdot)$

Recurrent Neural Network (RNN): Unrolled

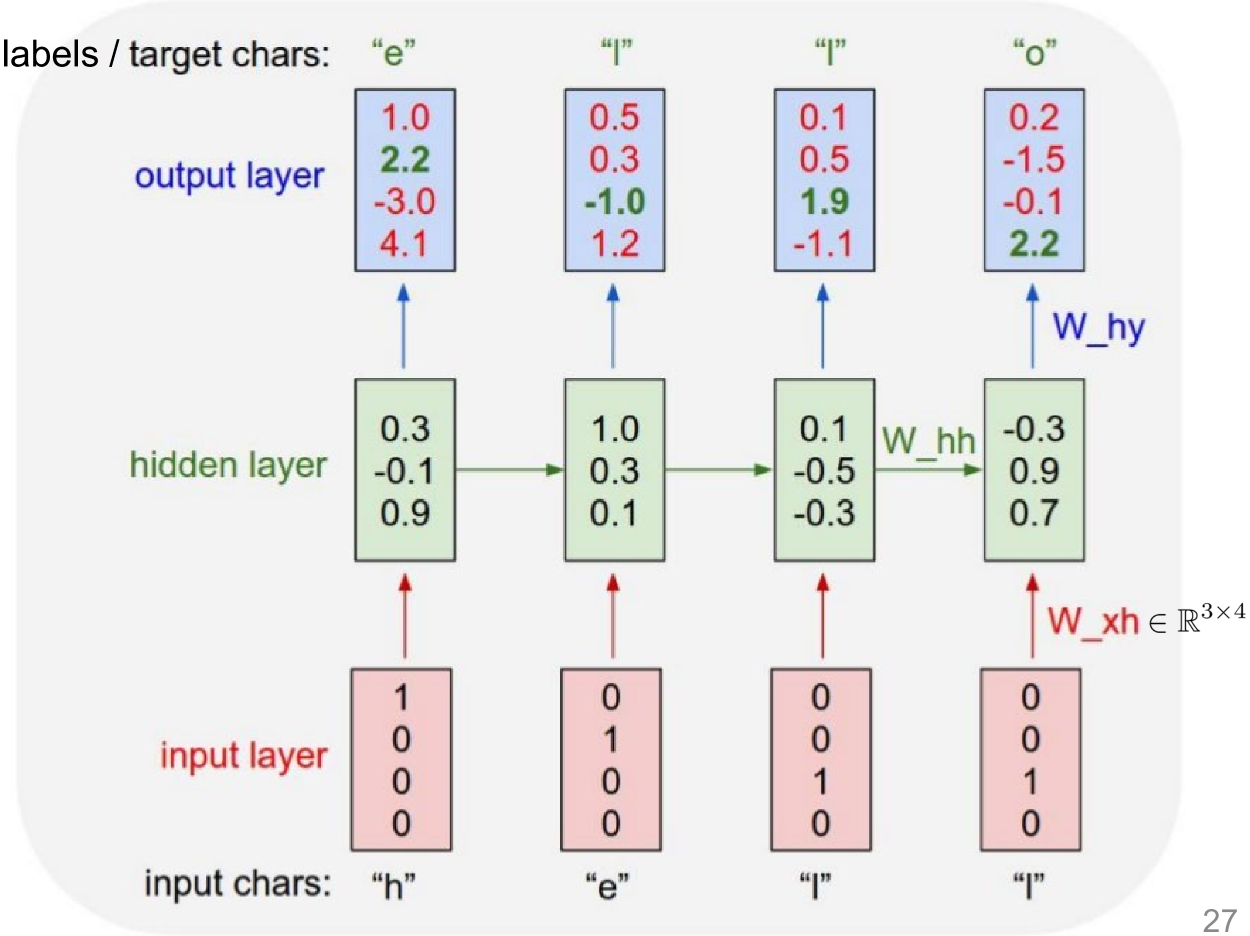


Example: Character-Level Language Model

Vocabulary:
[h, e, l, o]

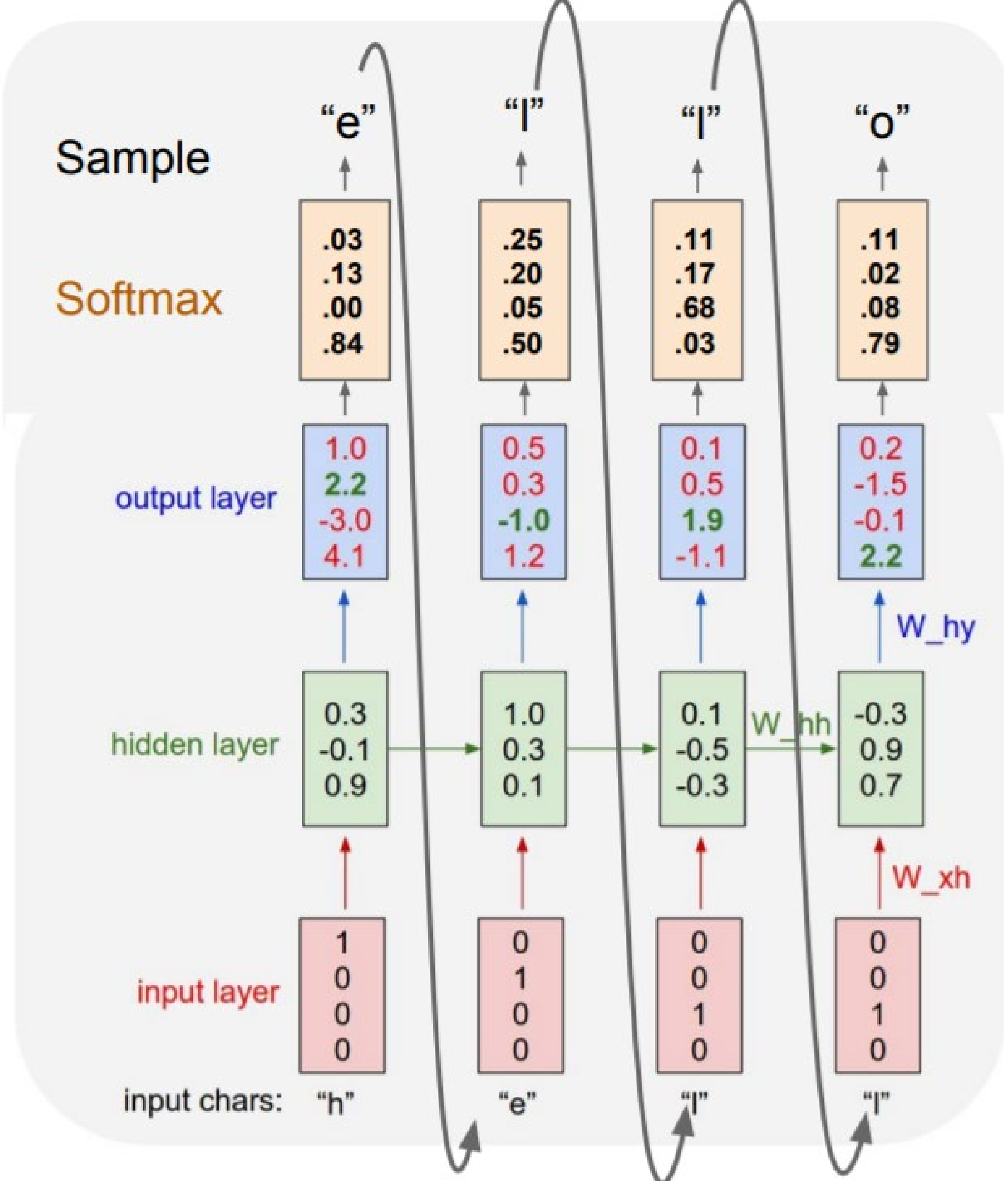
Embeddings:

Example training sequence:
“hello”



Vocabulary:
[h, e, l, o]

At test time, sample characters one at a time, feed back to model



Long Short-Term Memory (LSTM) Network

- RNN has the "vanishing gradient" problem!
- Resolved by long short-term memory (LSTM) units.

RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

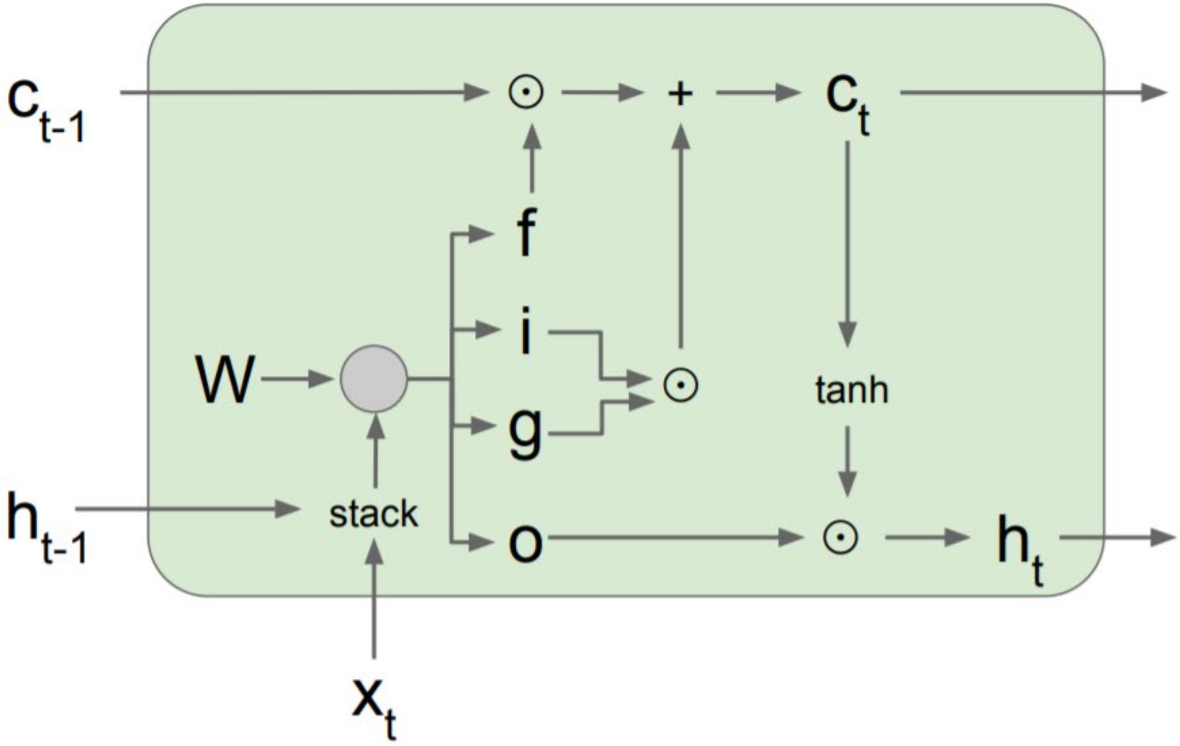
LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

An LSTM Unit [Hochreiter et al., 1997]

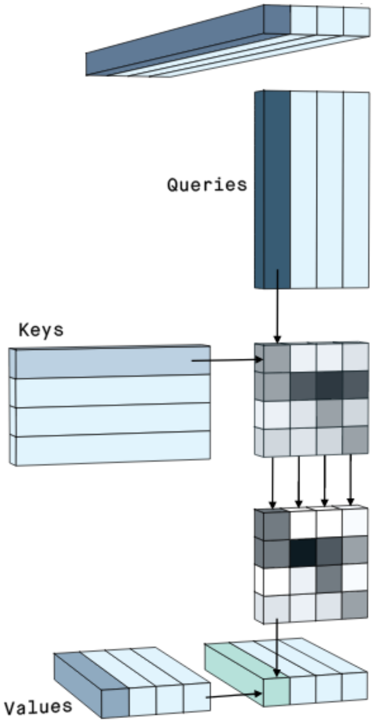


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Overview of Modern ML Applications: Transformers and BERT



Acknowledgment: Some graphics and slides were adapted from
https://peltarion.com/blog/data-science/self-attention-video

How to make good sense of language?

- Reading comprehension: If you were Google, what result(s) should you return for “brazil traveler to usa need a visa”?
  1. A webpage on U.S. citizens traveling to Brazil
  2. A webpage of the U.S. embassy/consulate in Brazil

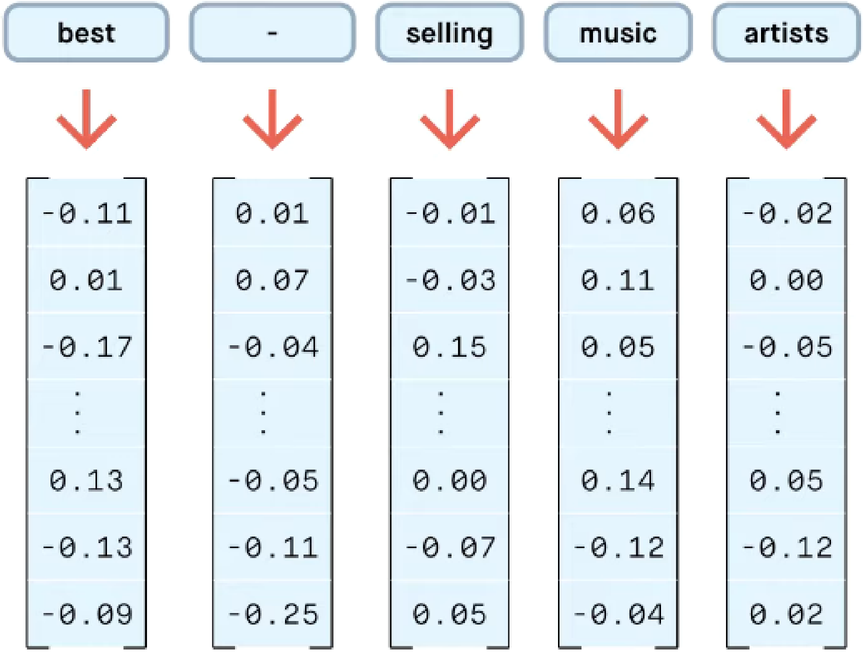
- Contextualization is the key!
  - ✦ A nice walk by the river bank.
  - ✦ Walk to the bank and get cash.

Word Embeddings in Natural Language Processing (NLP)

WordPiece tokens:

Embedding vectors:

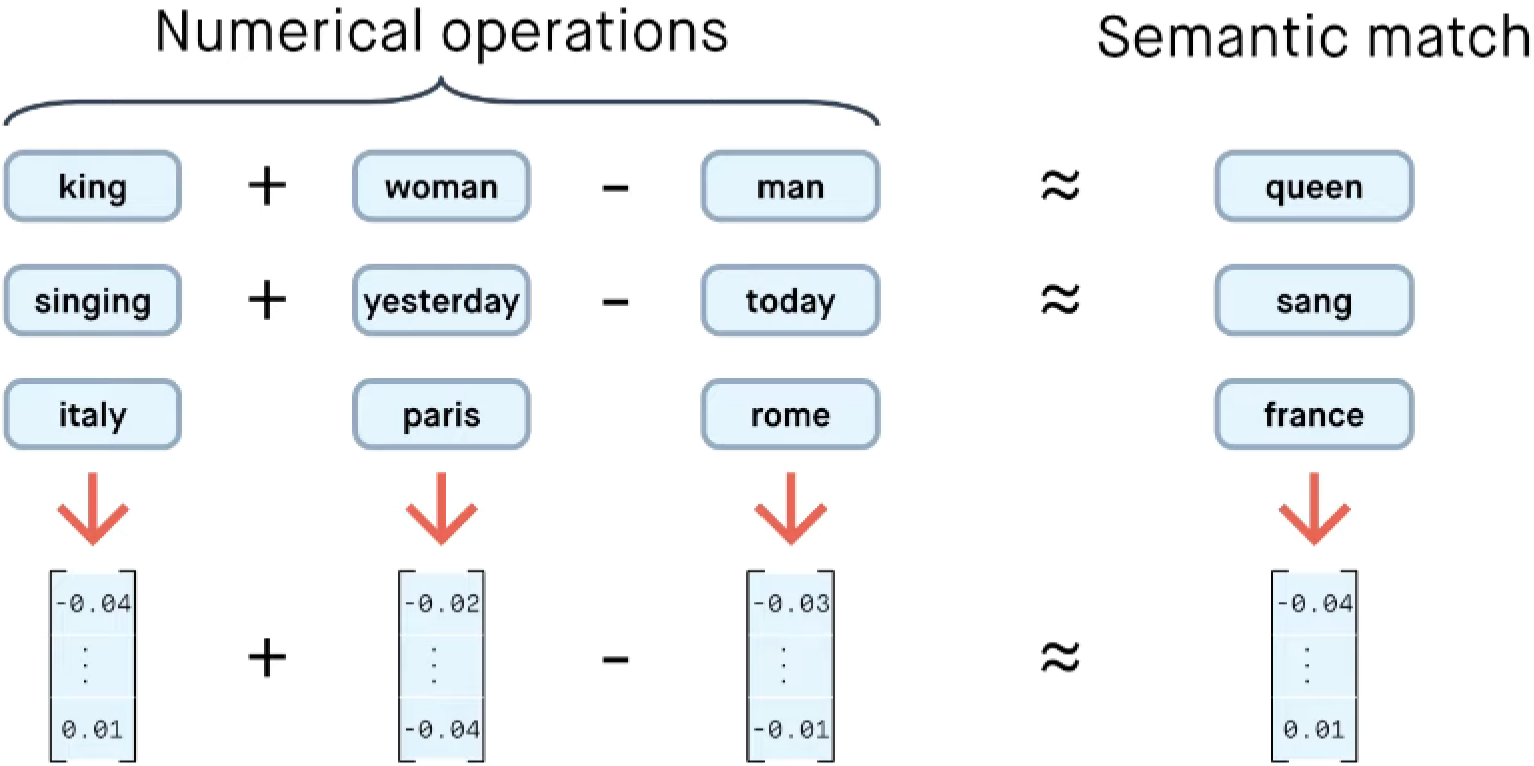
Values are *pretrained*



best	-	selling	music	artists
-0.11	0.01	-0.01	0.06	-0.02
0.01	0.07	-0.03	0.11	0.00
-0.17	-0.04	0.15	0.05	-0.05
⋮	⋮	⋮	⋮	⋮
0.13	-0.05	0.00	0.14	0.05
-0.13	-0.11	-0.07	-0.12	-0.12
-0.09	-0.25	0.05	-0.04	0.02

Embedding examples: Bag of Words (BoW), Word2Vec, …

Word Embeddings are Meaningful Under “+” and “–”



Numerical operations

Semantic match

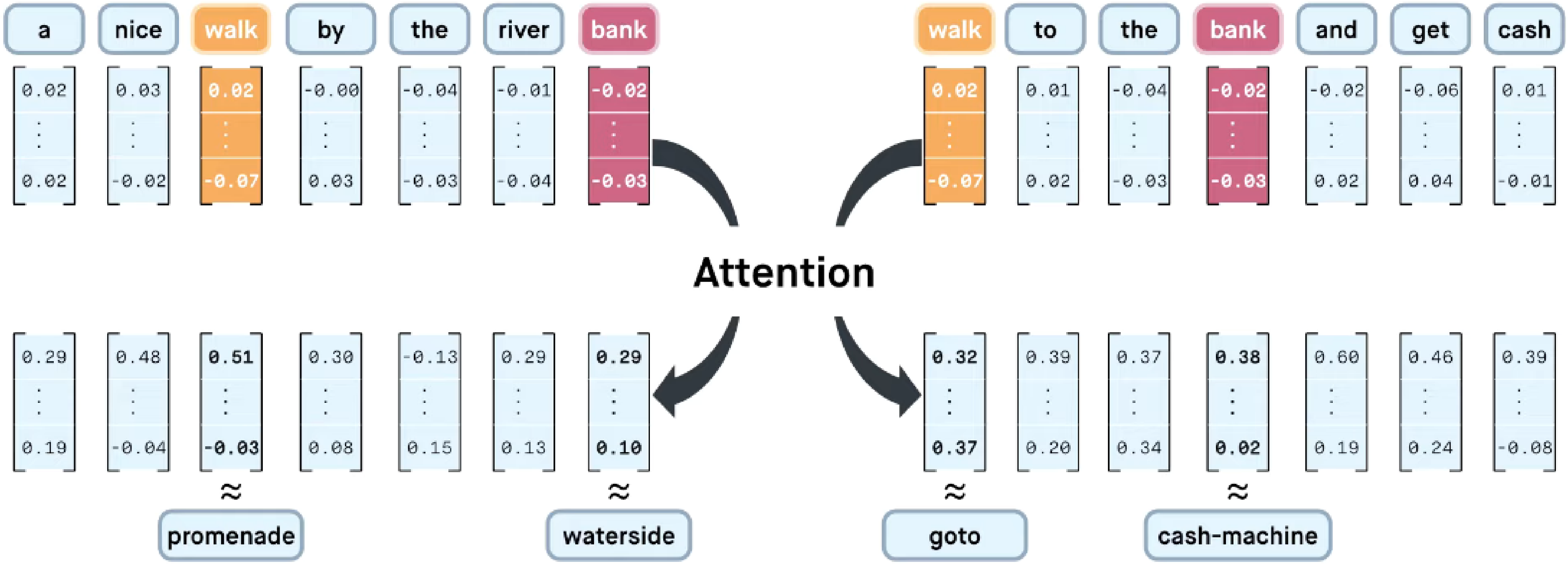
king + woman – man ≈ queen

singing + yesterday – today ≈ sang

italy paris rome france

$$\begin{bmatrix} -0.04 \\ \vdots \\ 0.01 \end{bmatrix} + \begin{bmatrix} -0.02 \\ \vdots \\ -0.04 \end{bmatrix} - \begin{bmatrix} -0.03 \\ \vdots \\ -0.01 \end{bmatrix} \approx \begin{bmatrix} -0.04 \\ \vdots \\ 0.01 \end{bmatrix}$$

Contextualization by "Attention"

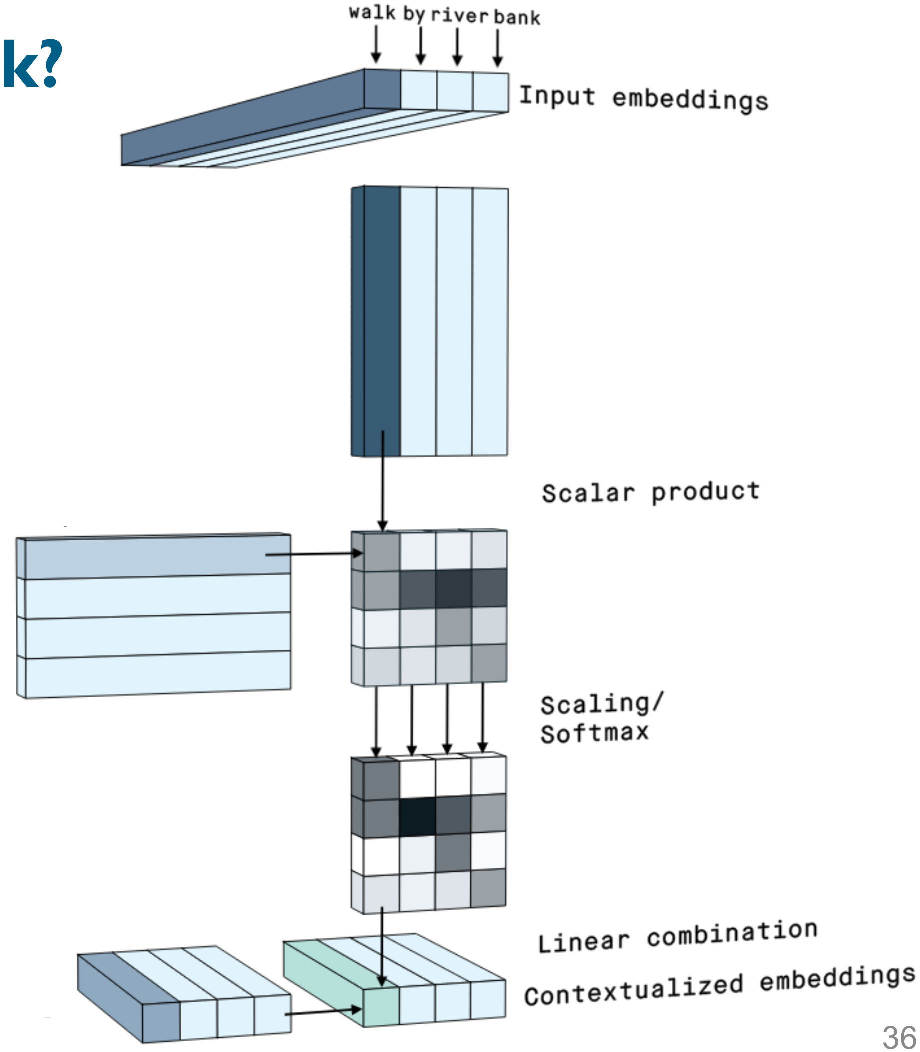


How does “attention” work?

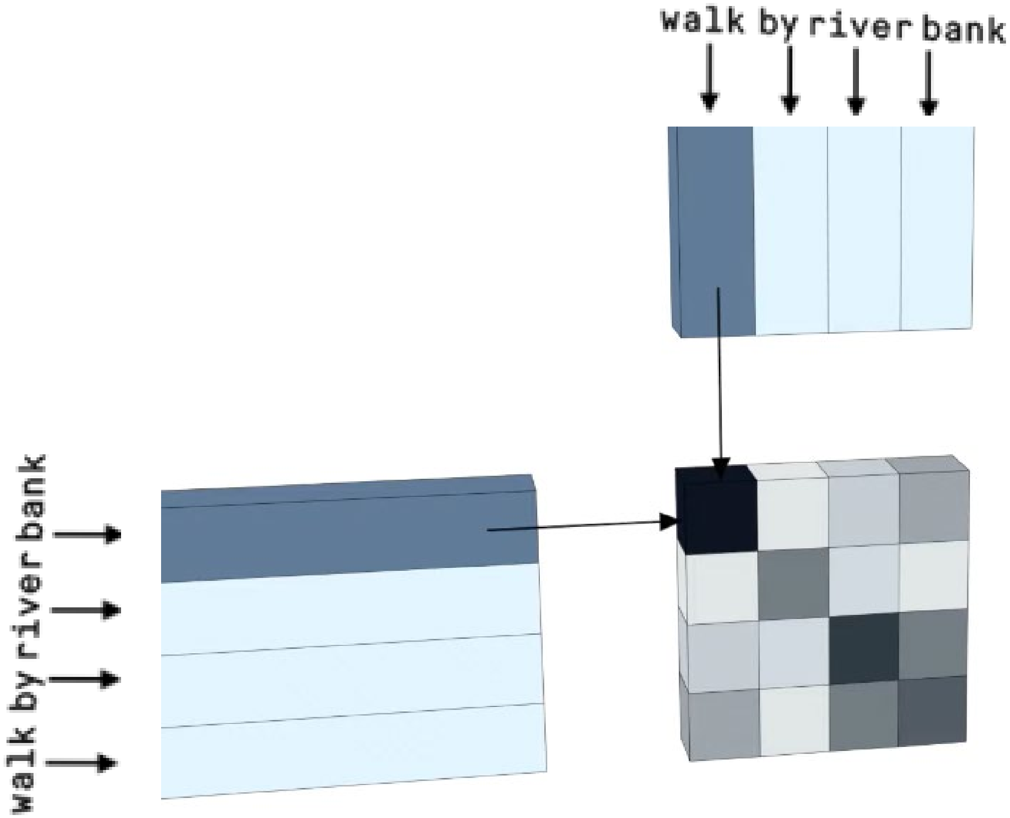
Similarity measure

Exponential saturation & normalization

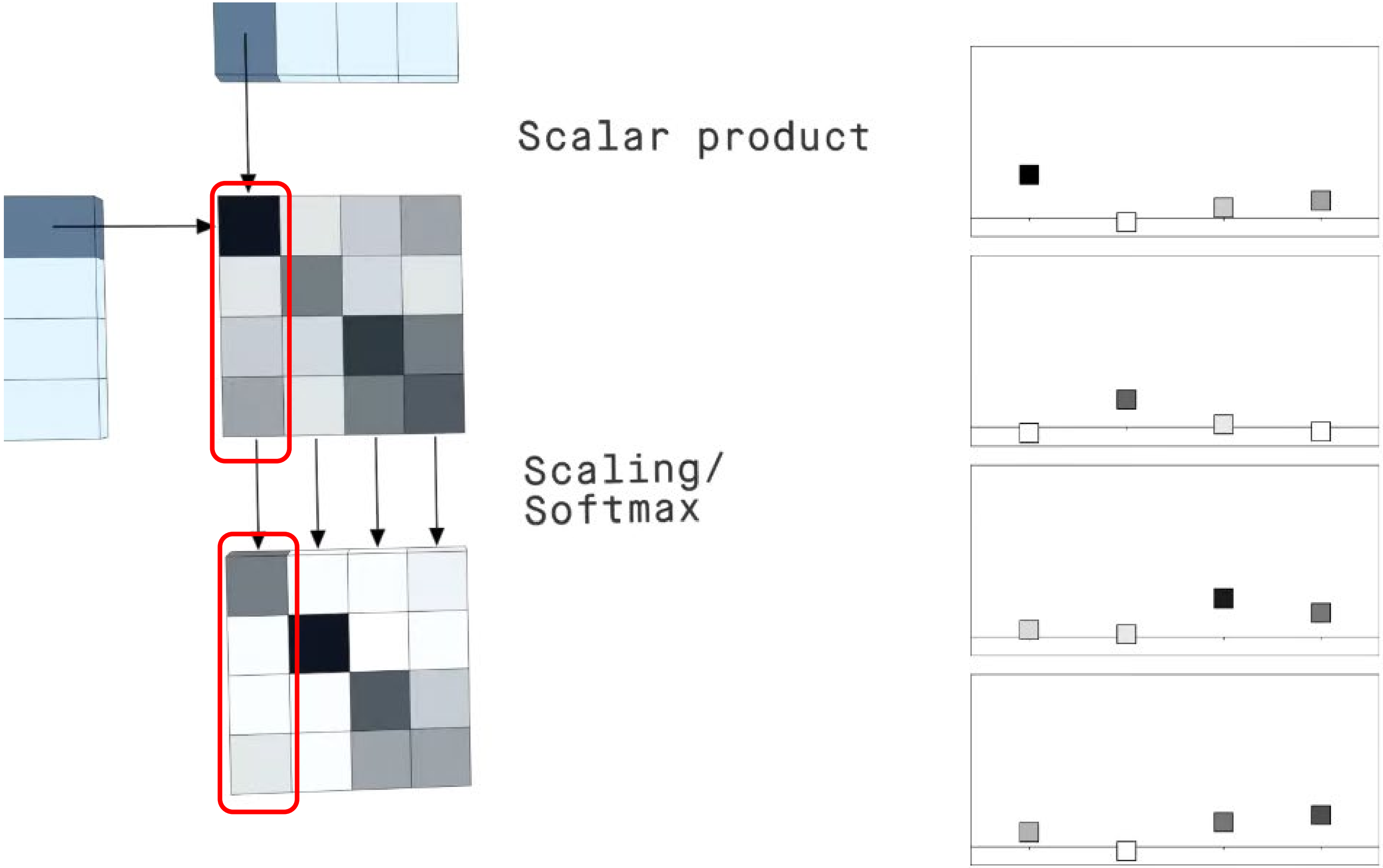
Linear combination



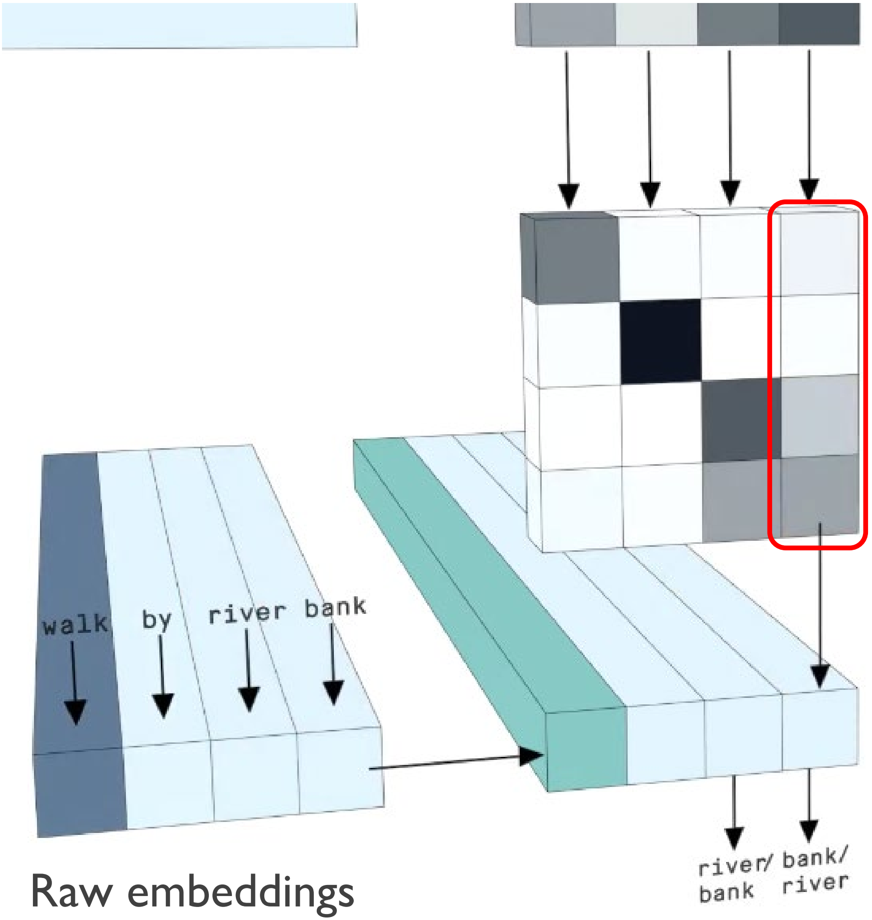
Similarity Measure via Inner Product



Exponential Saturation & Normalization via Softmax



Contextualization via Linear Combination



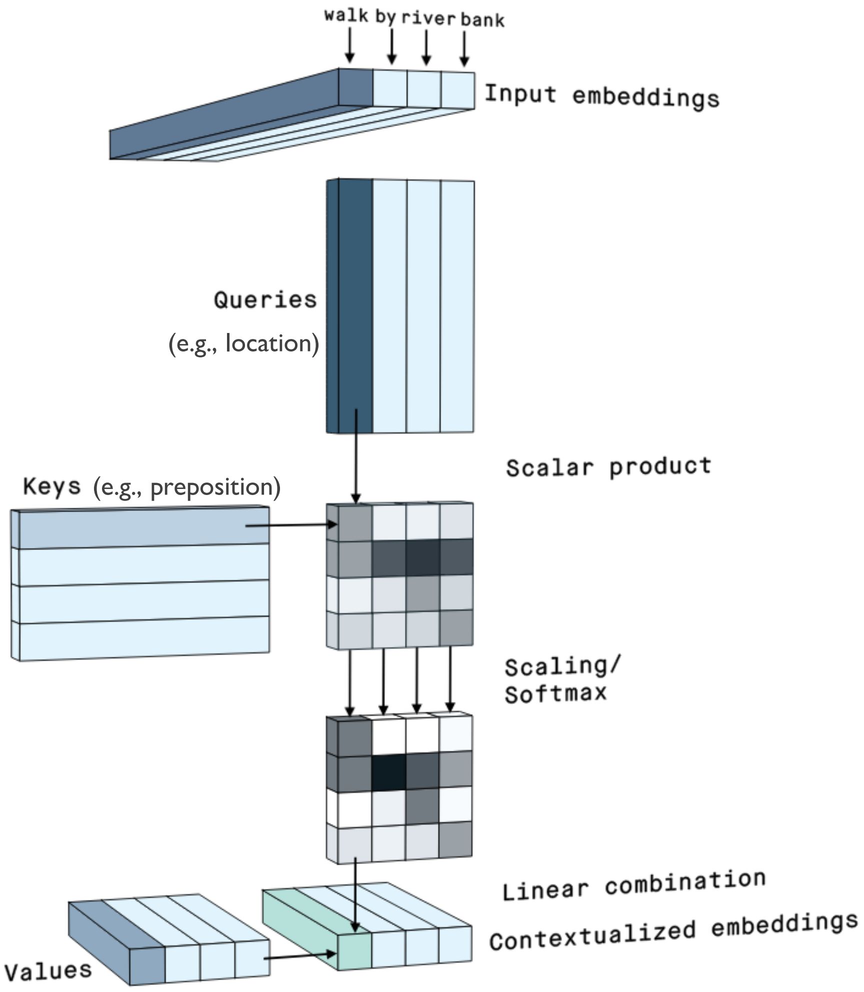
Weights for linear combination

Contextualized embeddings

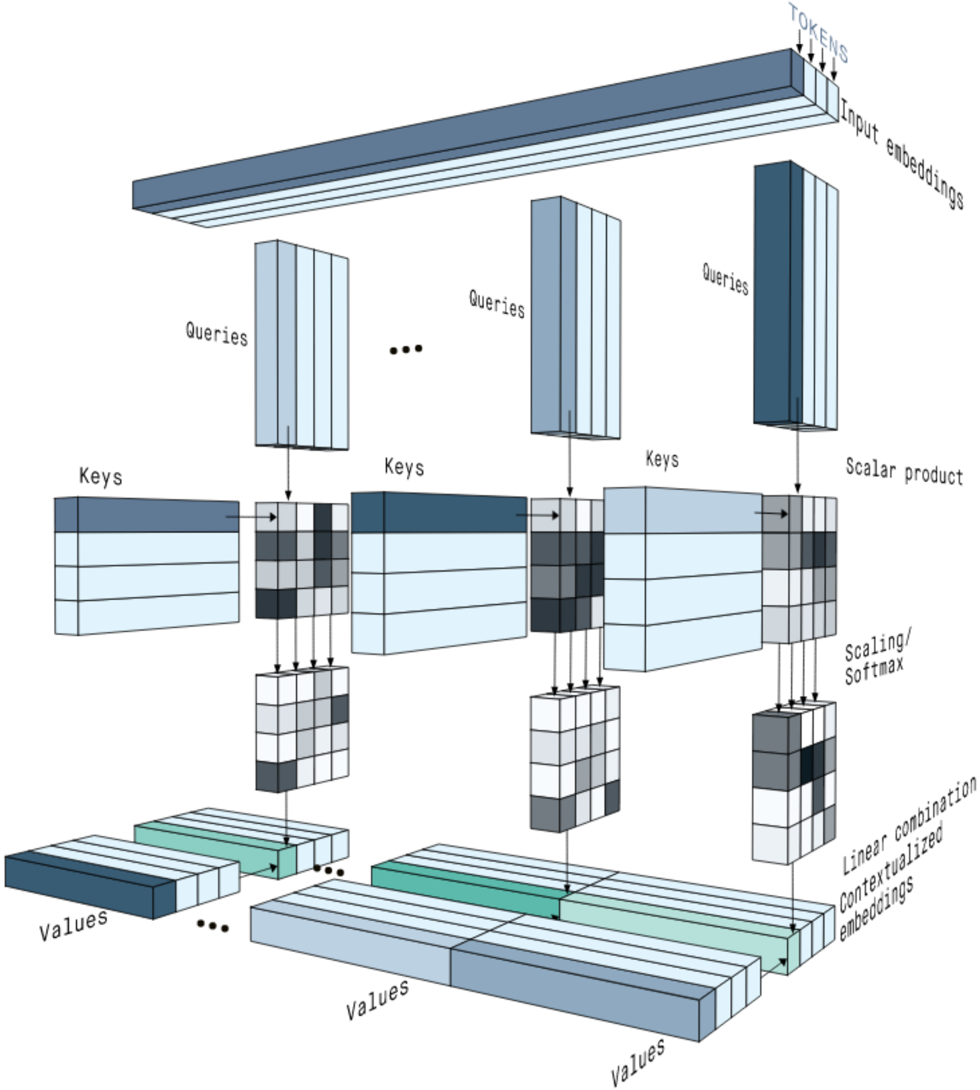
Raw embeddings

Key, Value, and Query

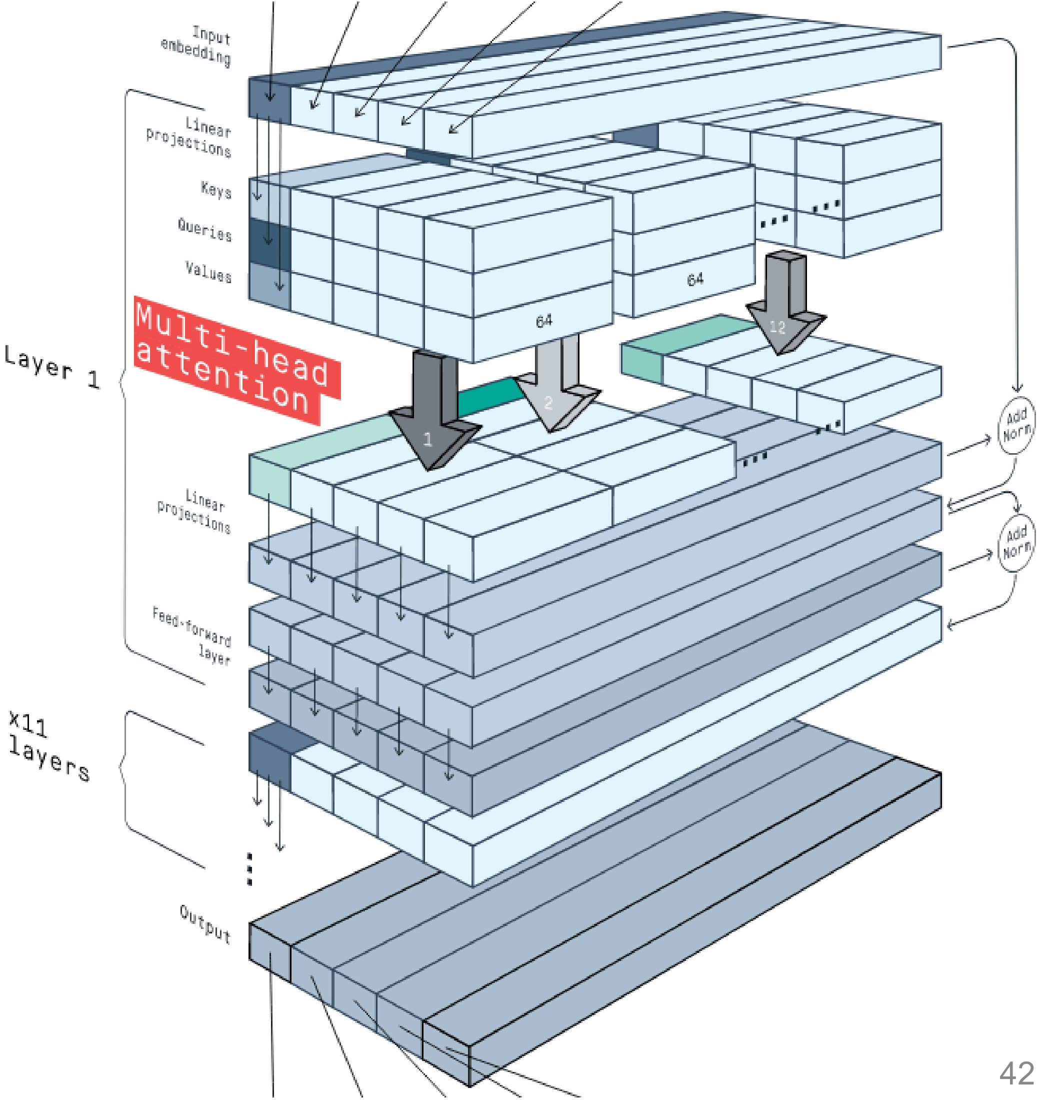
- "Key", "value", and "query" are three projections of an input embedding to three vector subspaces.
- Each subspace represents a unique semantic aspect.
- The projection operators / matrices provide trainable parameters for Transformer neural networks.



Multi-Head Attention



Bidirectional Encoder Representations from Transformers (BERT)



Neural Network Training: Backpropagation

Acknowledgment: Some graphics and slides were adapted from Profs. Jain (MSU), Min Wu (UMD), Fei-Fei Li (Stanford)
Some figures are from Duda-Hart-Stork textbook, Fei-Fei Li's slides

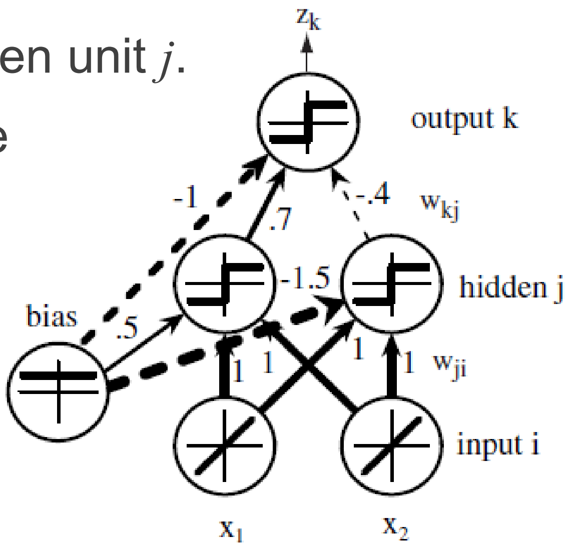
3-Layer Neural Network Structure

- A single "bias unit" is connected to each unit in addition to the input units
- Net activation: $$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv w_j^t . x,$$

  where the subscript i indexes units in the input layer, j indexes units in the hidden layer;

  w_{ji} denotes the input-to-hidden layer weights at hidden unit j.
  - In neurobiology, such weights or connections are called "synapses"
- Each hidden unit emits an output that is a nonlinear function of its activation

$$y_j = f(net_j)$$



Training Neural Networks / Estimating Weights

- Notations: t_k ~ the *k*th target (or desired) output,
  z_k ~ the *k*th estimated/computed output with $k = 1, \ldots, c$.
  w_{ij} ~ weight of the network
- Squared cost func:

$$J(w) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$$

- Learning based on gradient descent by iteratively updating the weights:

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m),$$

$$\Delta w = -\eta \frac{\partial J}{\partial w}$$

  - The weights are initialized with random values, and updated in a direction to reduce the error.
  - **Learning rate**, η, controls the step size of the update in weights.

Walking man image is CC0 1.0 public domain

Figure source: Stanford CS231n by Fei-Fei Li

Gradient Descent

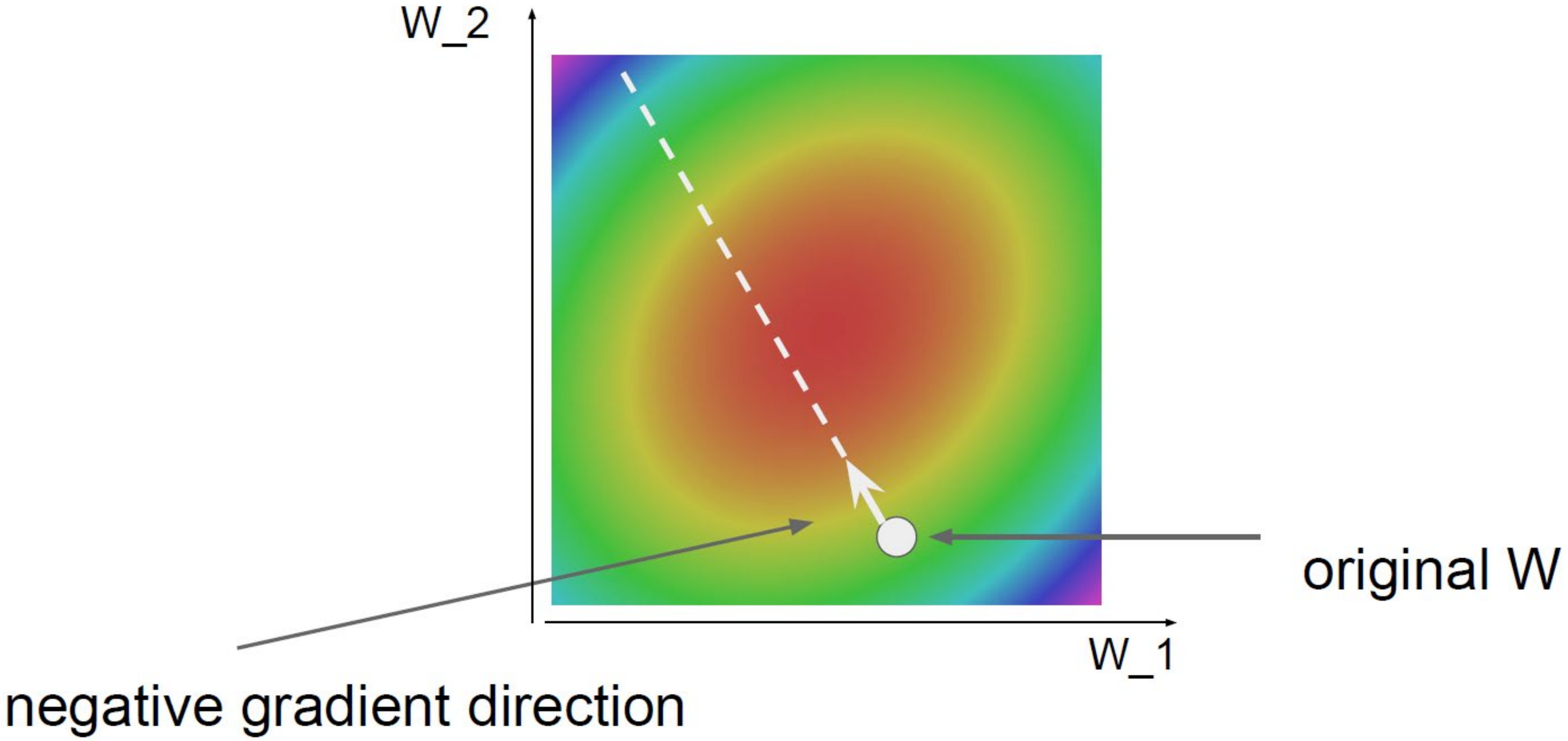


Figure source: Stanford CS231n by Fei-Fei Li

Efficient Gradient Calculation: Backpropagation

- Computes $\partial J / \partial w_{ji}$ for a single input-output pair.
- Exploit the chain rule for differentiation, e.g.,

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

- Computed by ***forward*** and ***backward*** sweeps over the network, keeping track only of quantities local to each unit.
- Iterate backward one unit at a time from last layer. Backpropagation avoids redundant calculations.

Backpropagation (BP): An Example

Backpropagation: a simple example

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

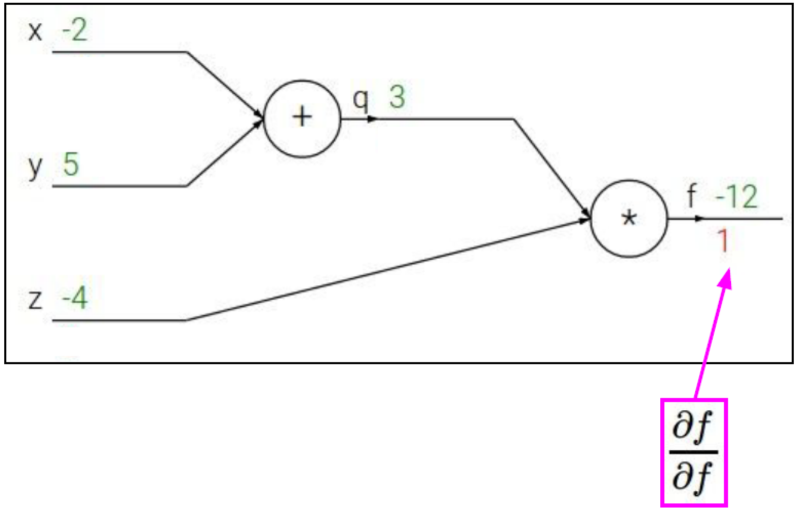
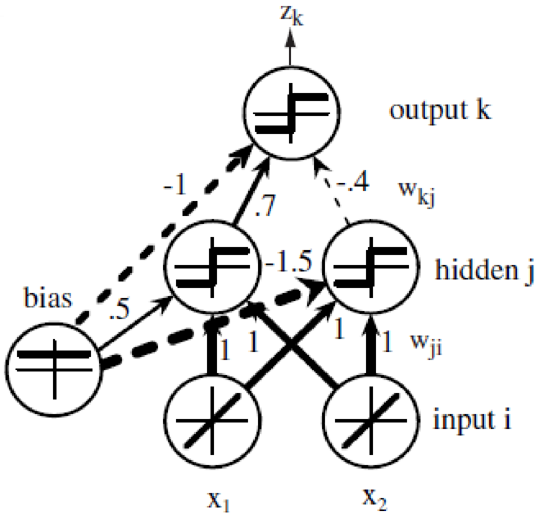


Figure source: Stanford CS231n by Fei-Fei Li

Network Learning by BP (cont'd)



- Error on hidden-to-output weight:

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

- δ_k, the sensitivity of unit k :
  describes how the overall error changes with the activation of the unit's net activation

$$\delta_k \equiv -\frac{\partial J}{\partial net_k}$$

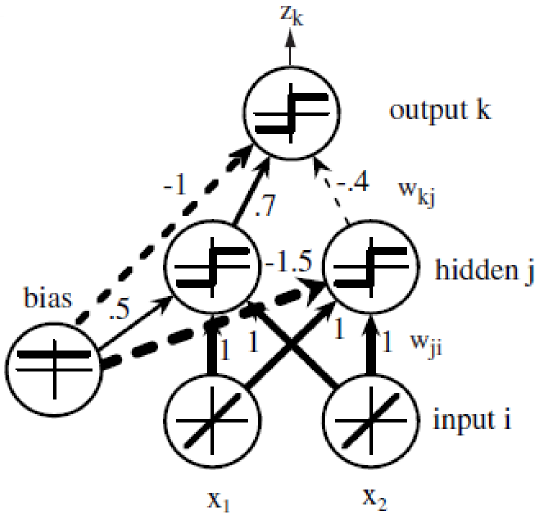
$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

- Since $net_k = \mathbf{w}_k{}^t \mathbf{y}$, we have $\frac{\partial net_k}{\partial w_{kj}} = y_j$

- **Summary 1:** weight update (or learning rule) for the hidden-to-output weight is:

$$\Delta w_{kj} = \eta \, (t_k - z_k) \, f'(net_k) \, y_j = \eta \, \delta_k y_j$$

Network Learning by BP (cont'd)



- Error on input-to-hidden weight:
  - chain rule: $$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$
  - $$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$
$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{kj}$$

- Sensitivity of a hidden unit:

  (Similarly defined as earlier)

$$\delta_j \equiv \frac{\partial J}{\partial net_j} = f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

- **Summary 2:** Learning rule for the input-to-hidden weight is:

$$\Delta w_{ji} = \eta \underbrace{\left[f'(net_j)\sum w_{kj}\delta_k\right]}_{\delta_j} x_i = \eta\delta_j x_i$$

Sensitivity at Hidden Node

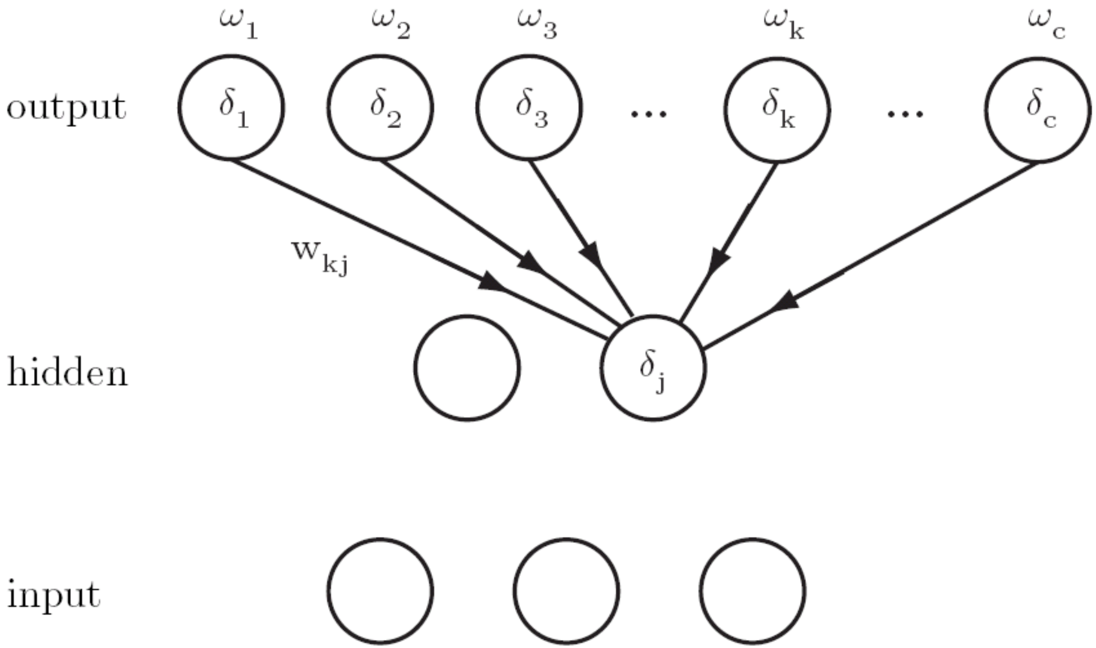


Figure 6.5: The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units: $\delta_j = f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k$. The output unit sensitivities are thus propagated "back" to the hidden units.

BP Algorithm: Training Protocols

- Training protocols:
  - **Batch**: Present all patterns before updating weights
  - **Stochastic**: patterns/input are chosen randomly from training set; network weights are updated for each pattern
  - **Online**: present each pattern once & only once (no memory for storing patterns)
- Stochastic backpropagation algorithm:

```
Begin  initialize $n_H$; w, criterion thres, $\eta$, m ← 0
          do m ← m + 1
                 $x^m$ ← randomly chosen pattern
                 $w_{ji} \leftarrow w_{ji} + \eta\delta_j x_i$; $w_{kj} \leftarrow w_{kj} + \eta\delta_k y_j$
          until $||\nabla J(w)|| <$ thres
          return w
End
```

BP Algorithm: Stopping Criterion

- Algorithm terminates when the change in criterion function $J(w)$ is smaller than some preset thres
  - Also exist other stopping criteria with better performance
- A weight update may reduce the error on the single pattern being presented, but can increase the error on the full training set
  - In stochastic backpropagation and batch propagation, we should make several passes (epoches) through the training data

Learning Curves

- Before training starts, the error on the training set is high; as the learning proceeds, error becomes smaller
- Error per pattern depends on the amount of training data and expressive power (e.g. # of weights) in the network
- Average error on an independent test set is always higher than on the training set, and it can decrease or increase

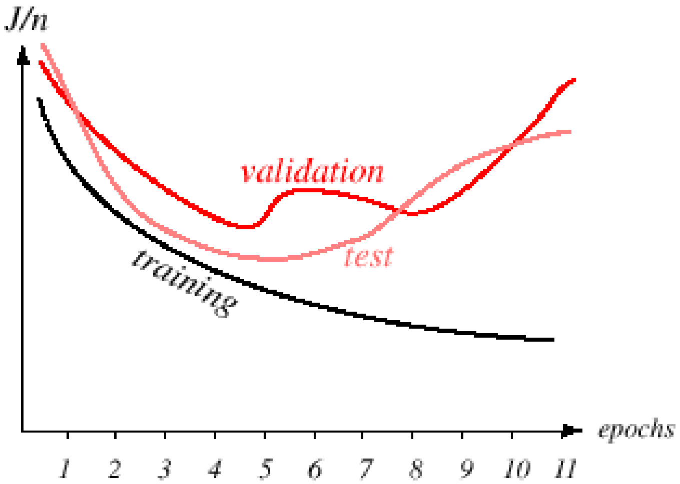


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n\sum_{p=1}^{n} J_p$. The validation error and the test or generalization error per pattern are virtually always higher than the training error. In some protocols, training is stopped at the first minimum of the validation set. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

- A validation set is used in order to decide when to stop training:
  - Avoid overfitting the network and decrease the power of the classifier's generalization

"Stop training when the error on the validation set is minimum"

Practical Considerations: Learning Rate

- Learning Rate
  - Small learning rate: slow convergence
  - Large learning rate: high oscillation and slow convergence

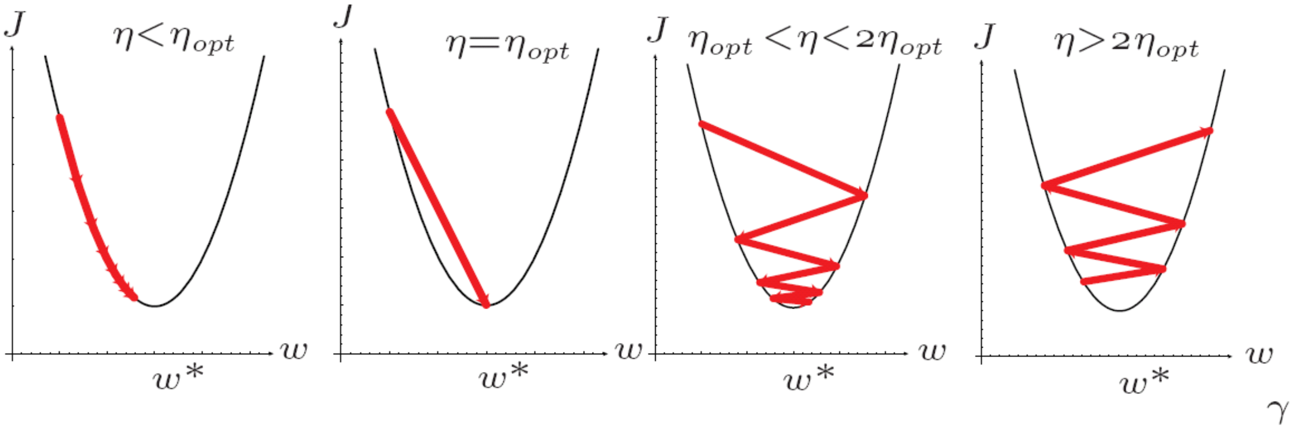


Figure 6.18: Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges.