CS 5565 - Intro to Statistical Learning

Lecture 8: Deep Learning

Adu Baffour, PhD

Lecture Objectives

- Describe the structure of a single-layer neural network.
- Describe the structure of a multilayer neural network.
- Describe the structure of a convolutional neural network.
- Describe the structure of a recurrent neural network.
- Compare deep learning to simpler models.
- Recognize the process by which neutral networks are fit.

Deep Learning

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

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.

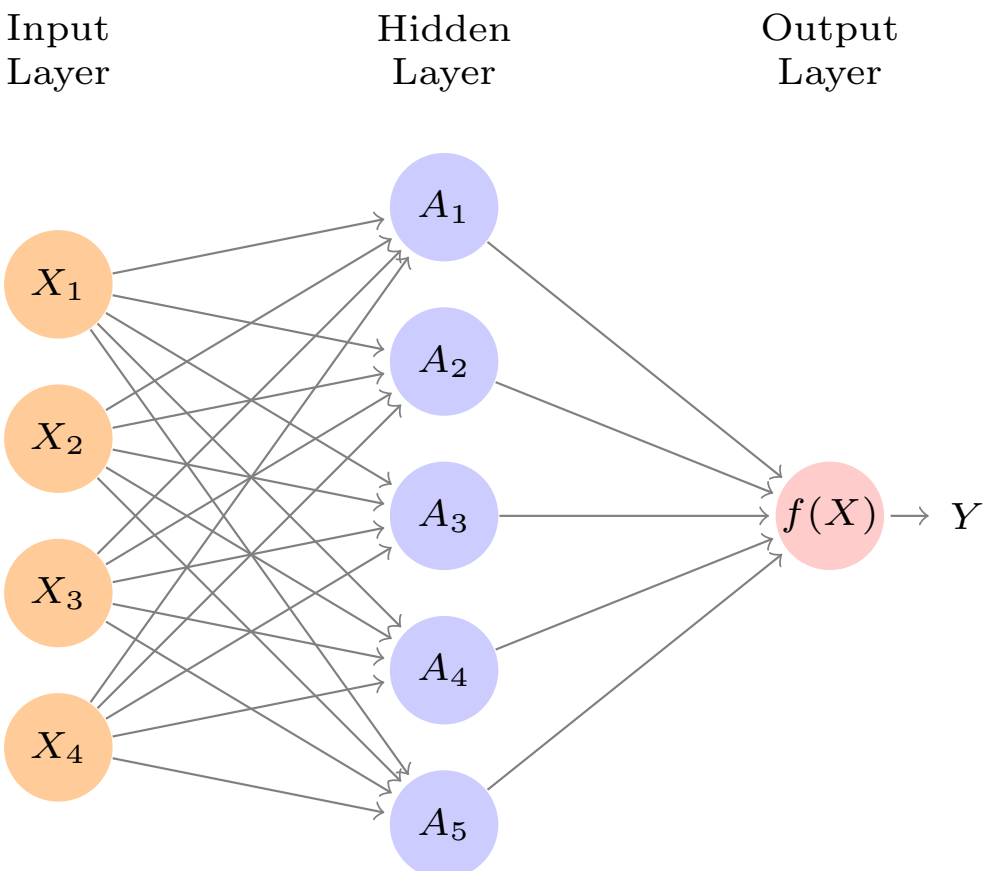
Re-emerged around 2010 as *Deep Learning.*
By 2020s very dominant and successful.

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

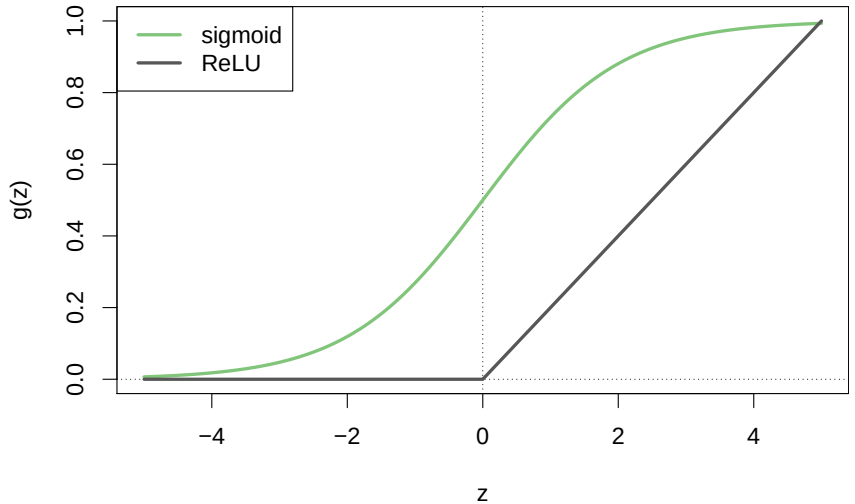
Much of the credit goes to three pioneers and their students: Yann LeCun, Geoffrey Hinton and Yoshua Bengio, who received the 2019 ACM Turing Award for their work in Neural Networks.

Single Layer Neural Network

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

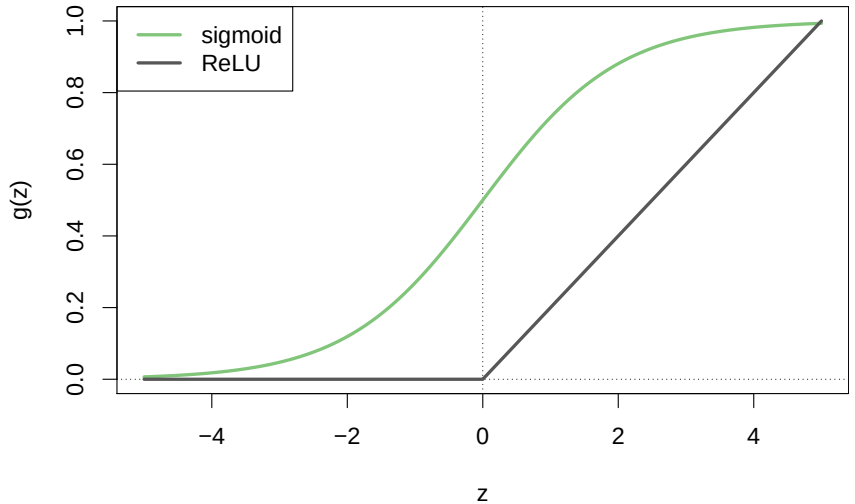


Details



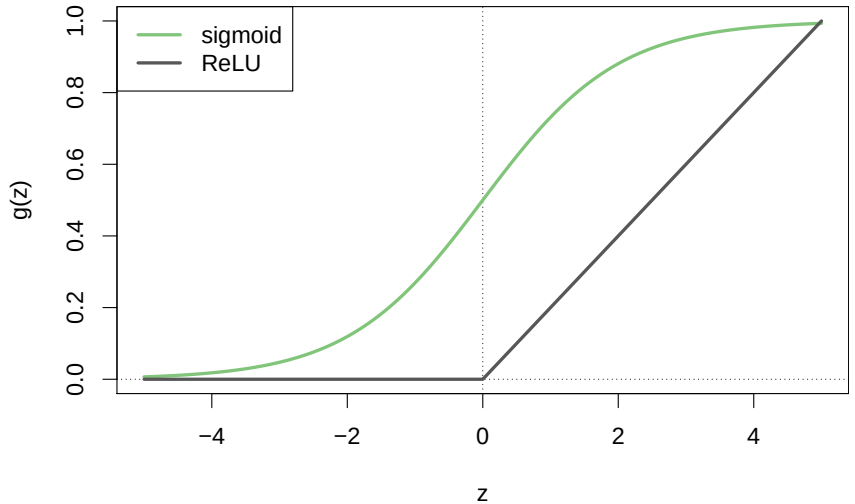
- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.

Details



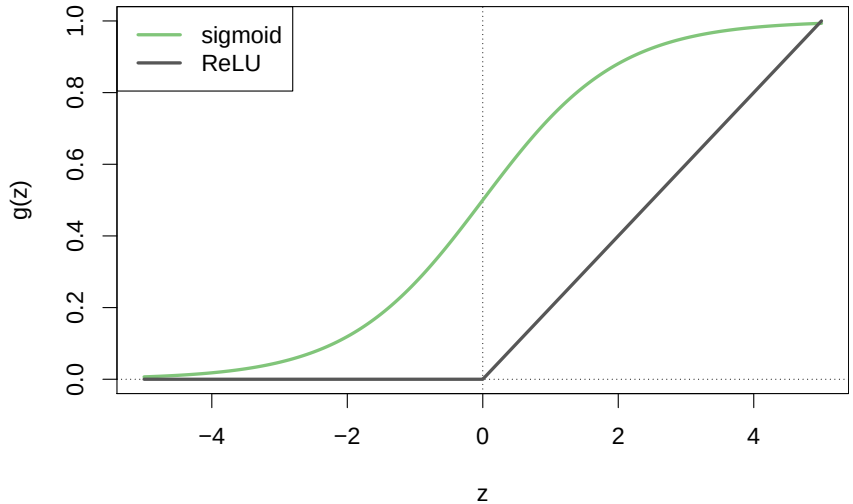
- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.

Details



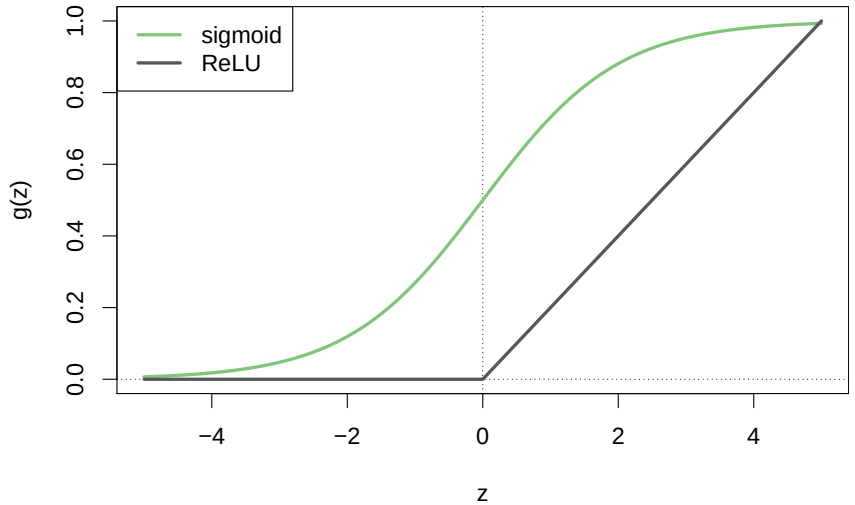
- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.
- Activation functions in hidden layers are typically nonlinear, otherwise the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

Details



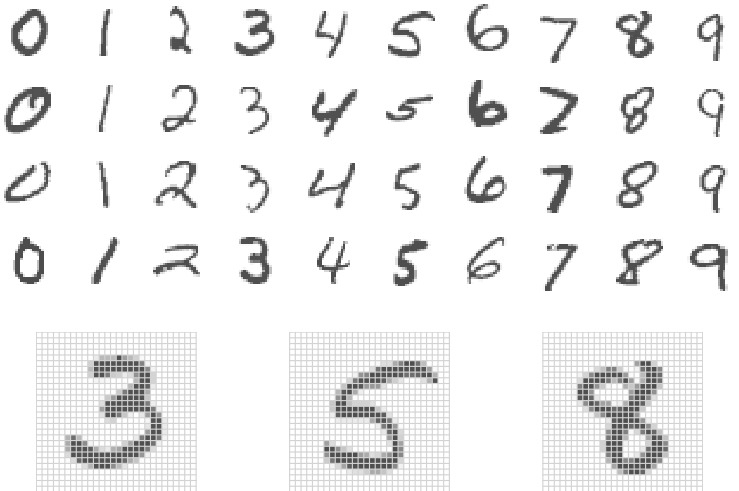
- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.
- Activation functions in hidden layers are typically nonlinear, otherwise the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

Details



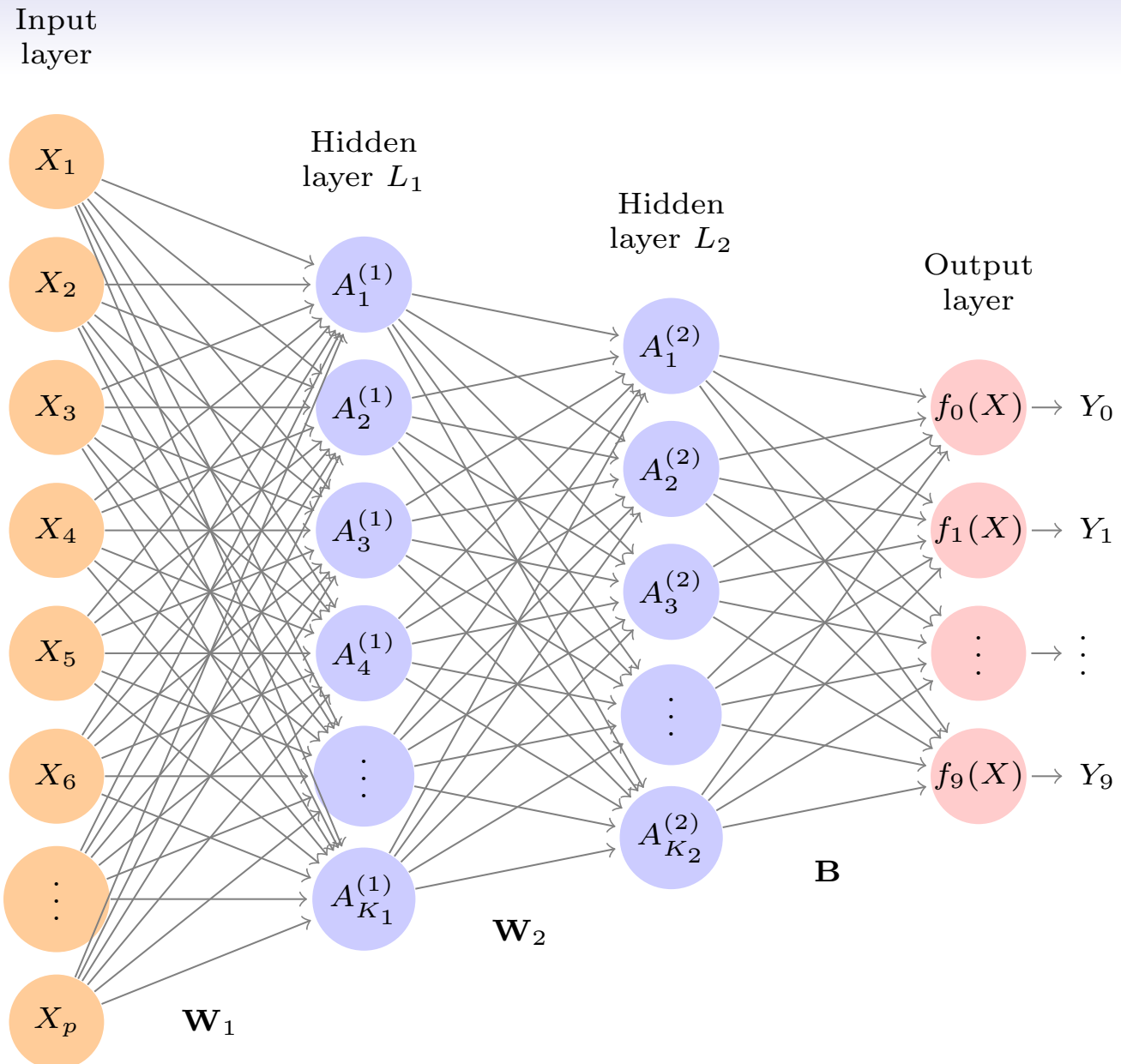
- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.
- Activation functions in hidden layers are typically nonlinear, otherwise the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

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

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

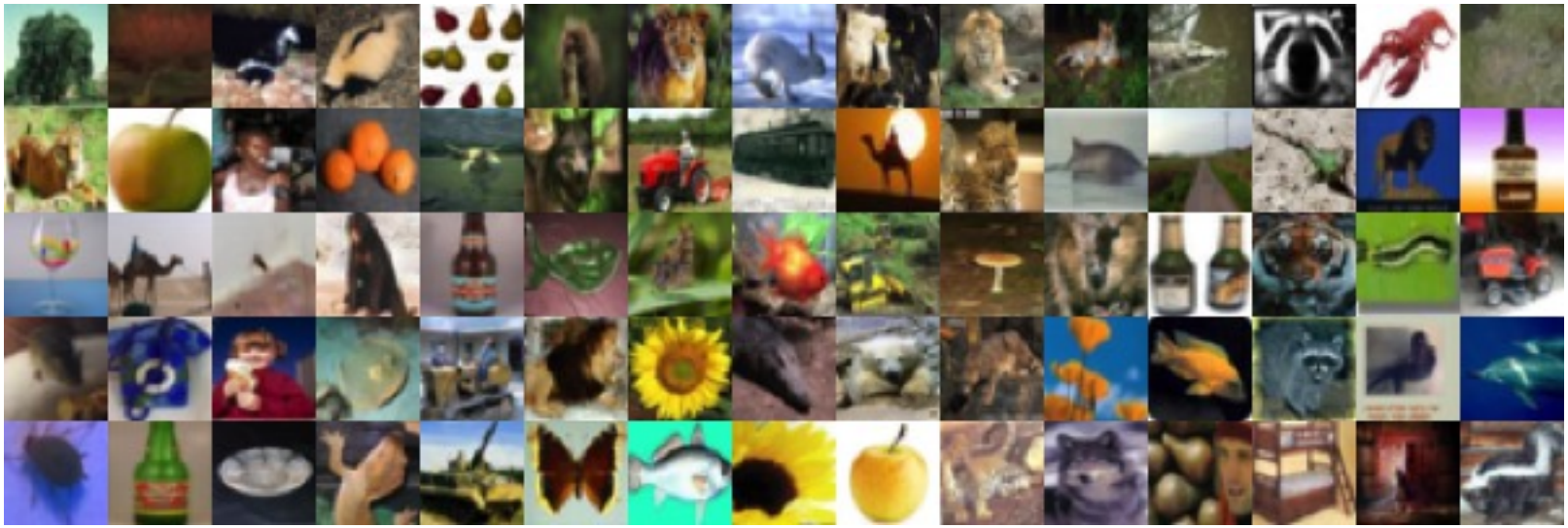
$$f_m(X) = \Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_\ell}}.$$

- 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*.

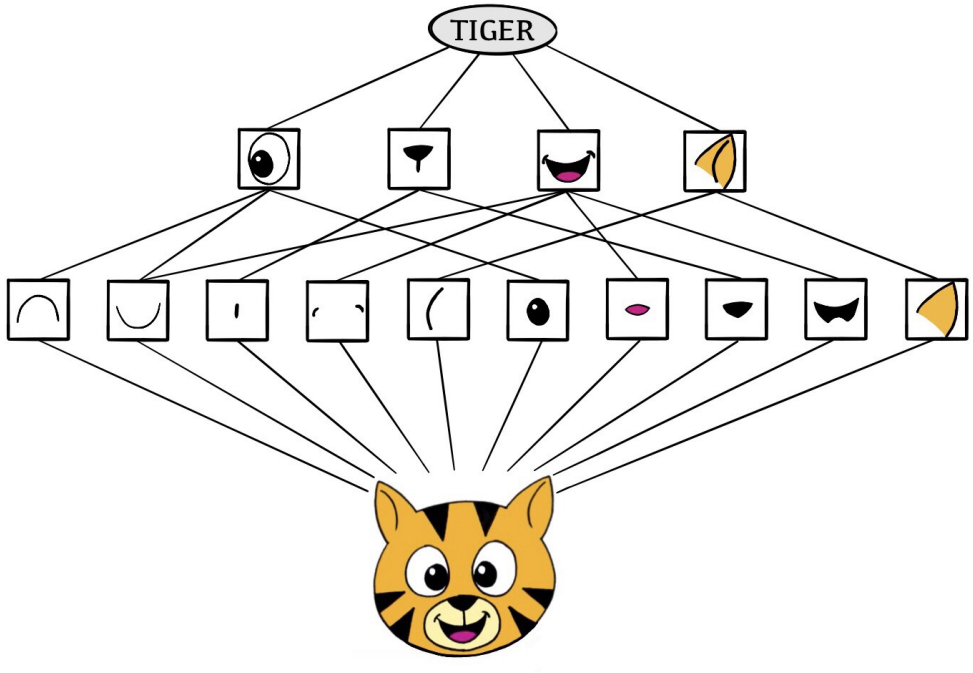
Convolutional Neural Network — CNN



- 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.

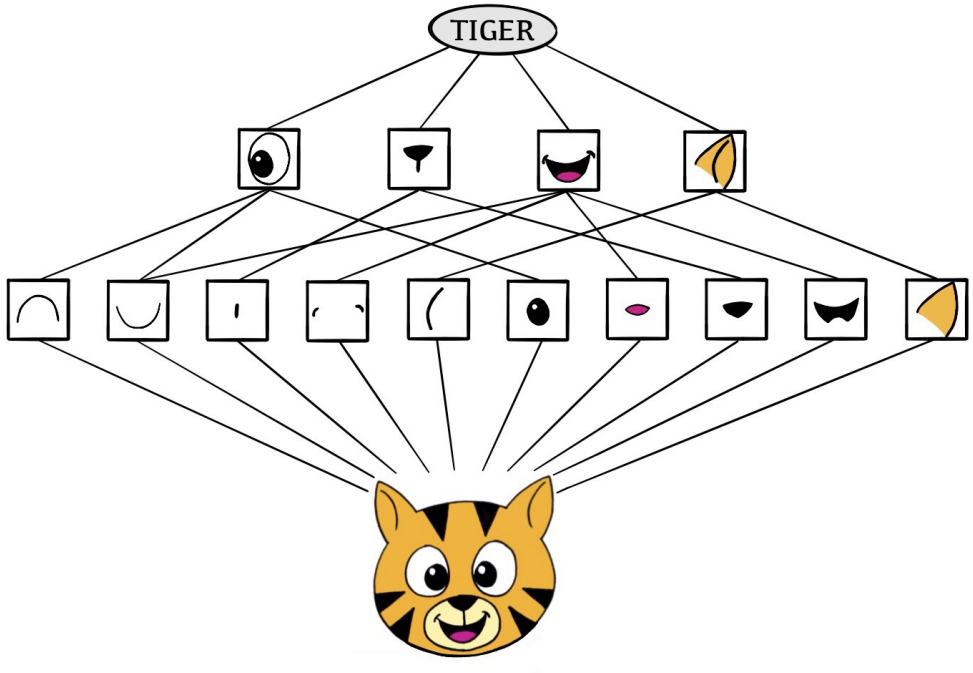
Each image is a three-dimensional array or *feature map:* $32 \times 32 \times 3$ array of 8-bit numbers. The last dimension represents the three color channels for red, green and blue.

How CNNs Work



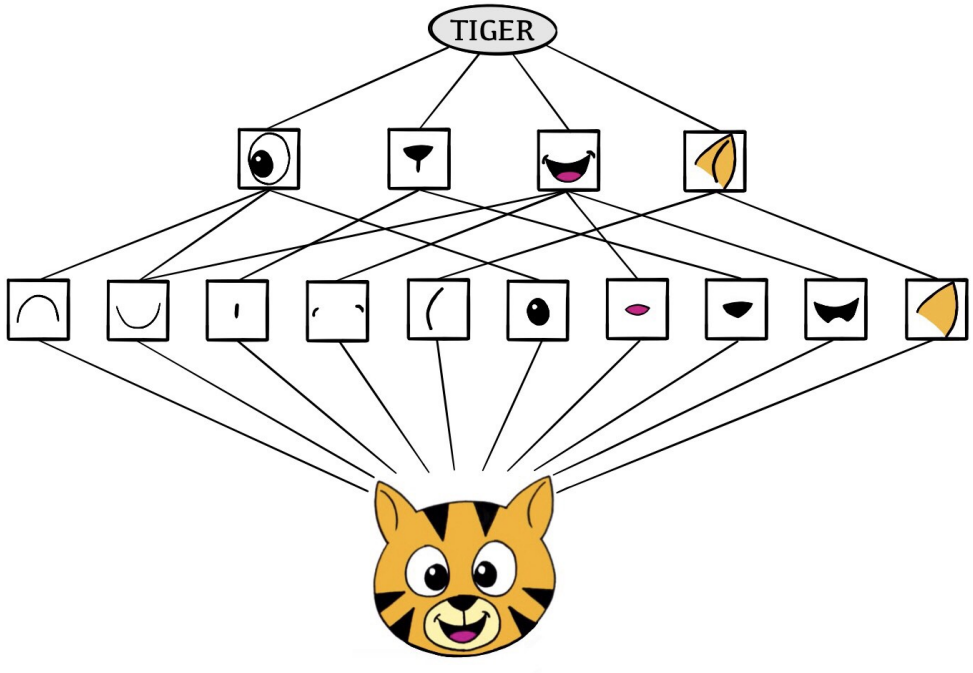
- The CNN builds up an image in a hierarchical fashion.

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.

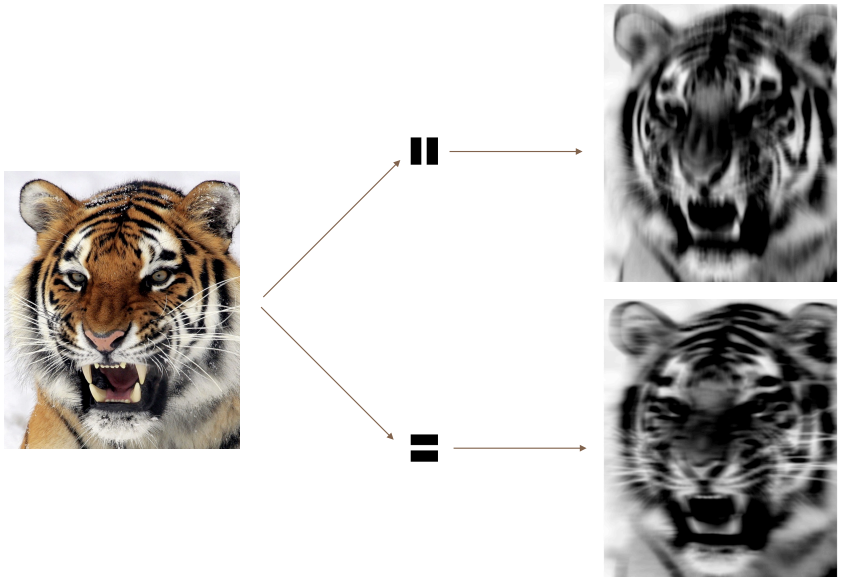
Convolution Filter

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

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

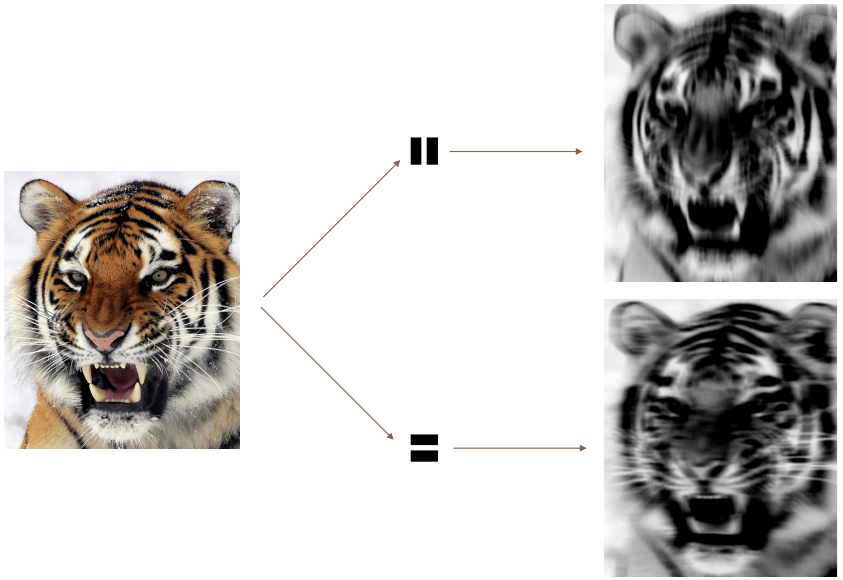
- The filter is itself an image, and represents a small shape, edge etc.
- We slide it around the input image, scoring for matches.
- The scoring is done via *dot-products*, illustrated above.
- If the subimage of the input image is similar to the filter, the score is high, otherwise low.
- The filters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

Convolution Example



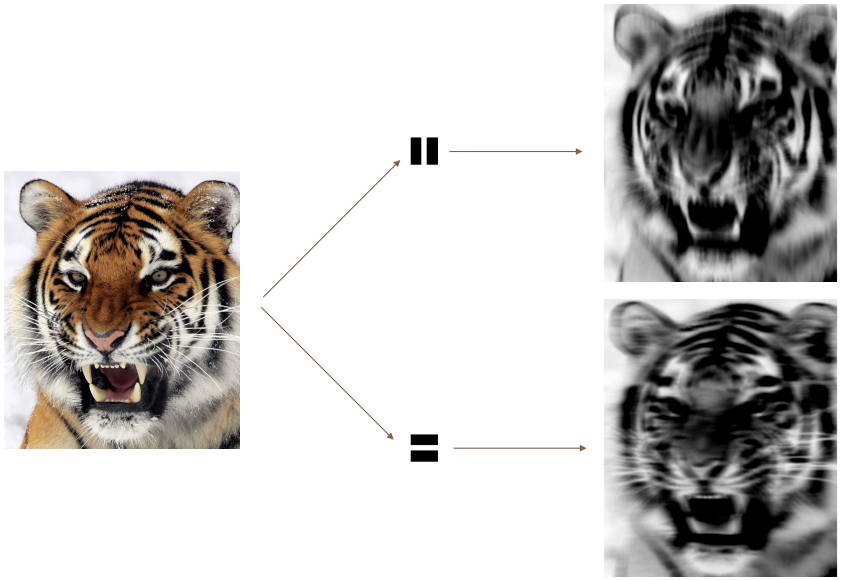
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.

Convolution Example



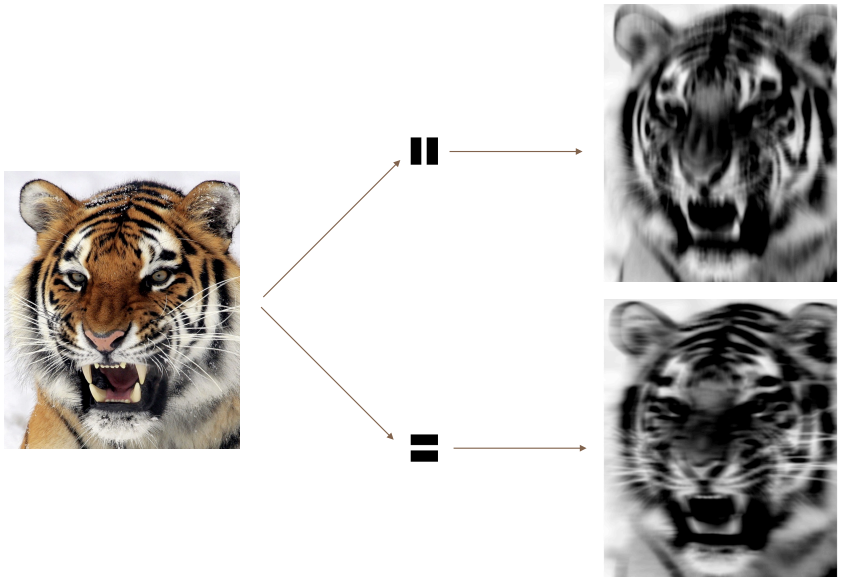
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.

Convolution Example



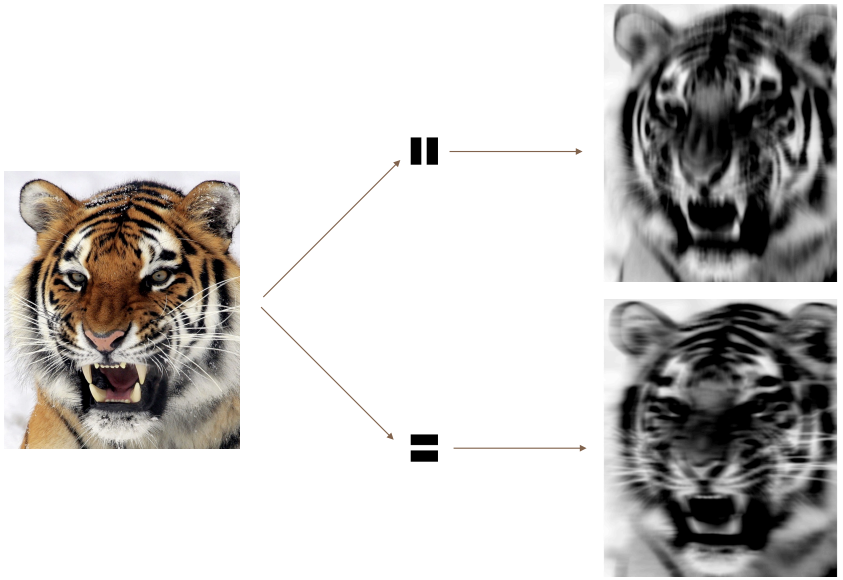
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.
- The result of the convolution is a new feature map.

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



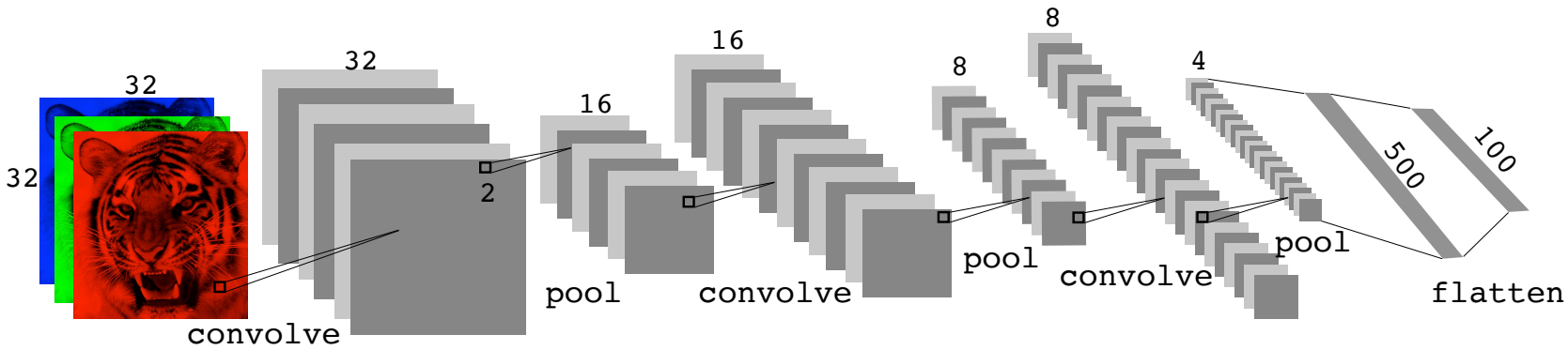
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.
- The result of the convolution is a new feature map.
- Since images have three colors channels, the filter does as well: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

Pooling

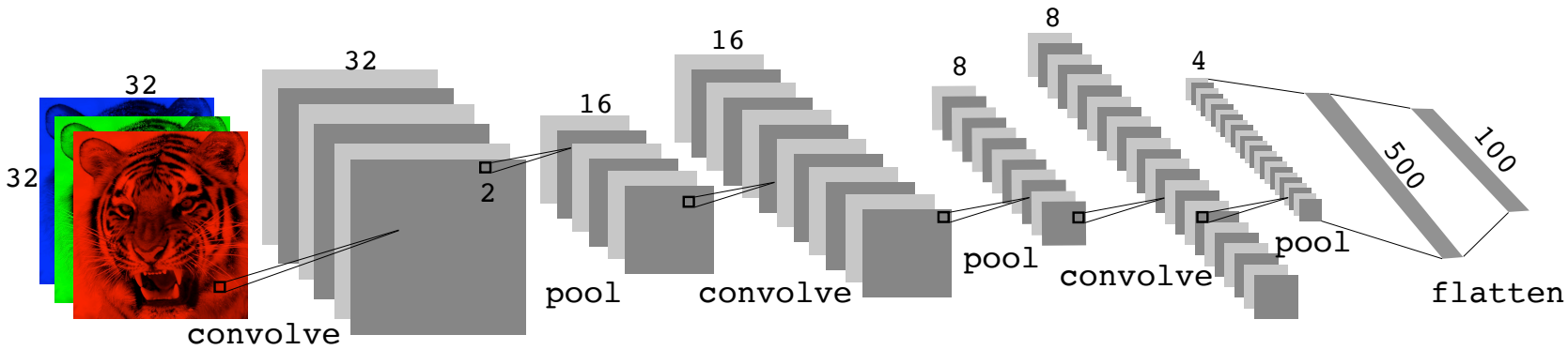
$$\text{Max pool} \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

- 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

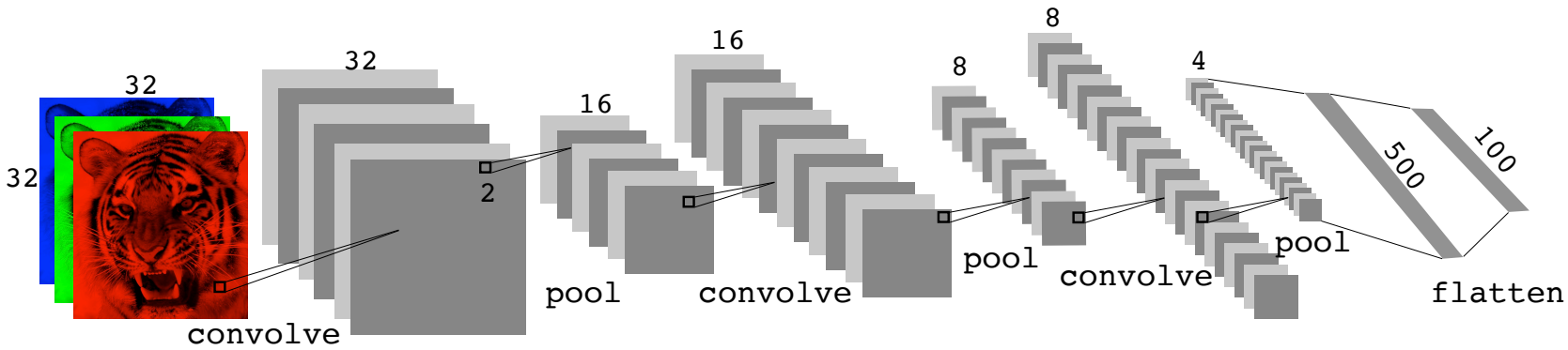


Architecture of a CNN



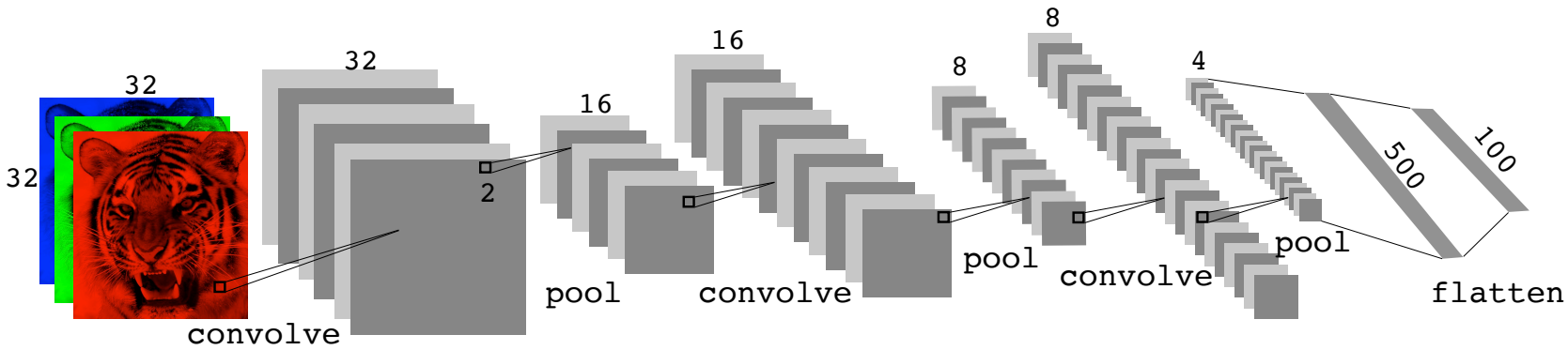
- Many convolve + pool layers.

Architecture of a CNN



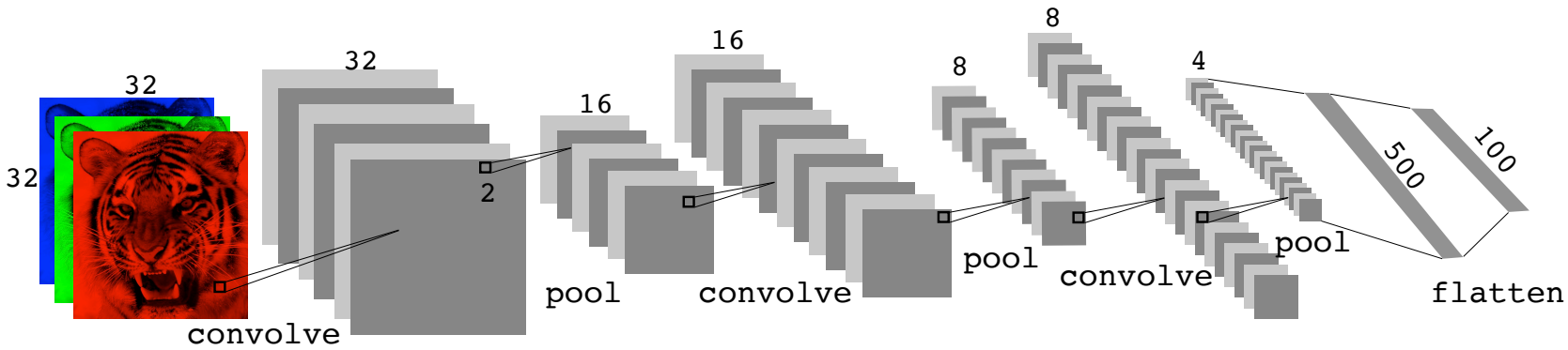
- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3.

Architecture of a CNN



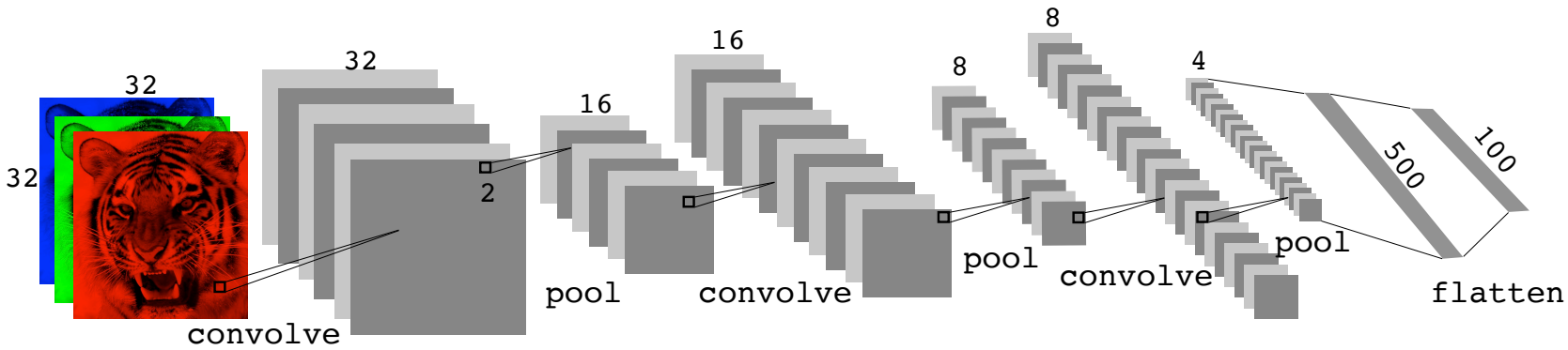
- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3.
- Each filter creates a new channel in convolution layer.

Architecture of a CNN



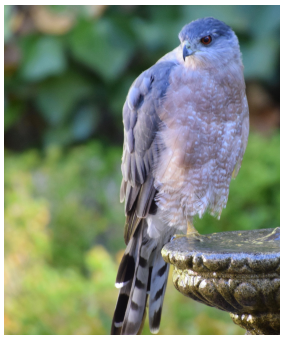
- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3.
- Each filter creates a new channel in convolution layer.
- As pooling reduces size, the number of filters/channels is typically increased.

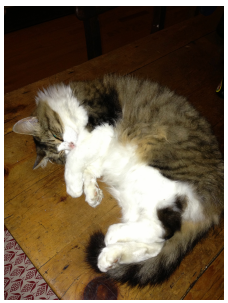
Architecture of a CNN



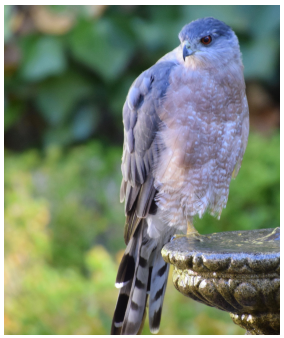
- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3.
- Each filter creates a new channel in convolution layer.
- As pooling reduces size, the number of filters/channels is typically increased.
- Number of layers can be very large. E.g. `resnet50` trained on `imagenet`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

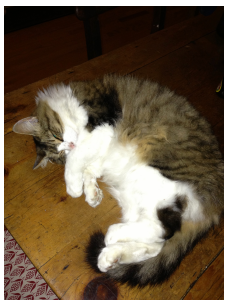
Using Pretrained Networks to Classify Images


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.

Document Classification: IMDB Movie Reviews

The `IMDB` corpus consists of user-supplied movie ratings for a large collection of movies. Each has been labeled for `sentiment` as `positive` or `negative`. Here is the beginning of a negative review:

> This has to be one of the worst films of the 1990s. When my friends & I were watching this film (being the target audience it was aimed at) we just sat & watched the first half an hour with our jaws touching the floor at how bad it really was. The rest of the time, everyone else in the theater just started talking to each other, leaving or generally crying into their popcorn ...

We have labeled training and test sets, each consisting of 25,000 reviews, and each balanced with regard to sentiment.

Document Classification: IMDB Movie Reviews

The `IMDB` corpus consists of user-supplied movie ratings for a large collection of movies. Each has been labeled for `sentiment` as `positive` or `negative`. Here is the beginning of a negative review:

> This has to be one of the worst films of the 1990s. When my friends & I were watching this film (being the target audience it was aimed at) we just sat & watched the first half an hour with our jaws touching the floor at how bad it really was. The rest of the time, everyone else in the theater just started talking to each other, leaving or generally crying into their popcorn ...

We have labeled training and test sets, each consisting of 25,000 reviews, and each balanced with regard to sentiment.

We wish to build a classifier to predict the sentiment of a review.

Featurization: Bag-of-Words

Documents have different lengths, and consist of sequences of words. How do we create features X to characterize a document?

- From a dictionary, identify the $10K$ most frequently occurring words.
- Create a binary vector of length $p = 10K$ for each document, and score a 1 in every position that the corresponding word occurred.
- With n documents, we now have a $n \times p$ *sparse* feature matrix $\mathbf{X}$.
- Bag-of-words are *unigrams*. We can instead use *bigrams* (occurrences of adjacent word pairs), and in general *m-grams*.

Recurrent Neural Networks

Often data arise as sequences:

- Documents are sequences of words, and their relative positions have meaning.
- Time-series such as weather data or financial indices.
- Recorded speech or music.
- Handwriting, such as doctor's notes.

RNNs build models that take into account this sequential nature of the data, and build a memory of the past.

Recurrent Neural Networks

Often data arise as sequences:

- Documents are sequences of words, and their relative positions have meaning.
- Time-series such as weather data or financial indices.
- Recorded speech or music.
- Handwriting, such as doctor's notes.

RNNs build models that take into account this sequential nature of the data, and build a memory of the past.

Recurrent Neural Networks

Often data arise as sequences:

- Documents are sequences of words, and their relative positions have meaning.
- Time-series such as weather data or financial indices.
- Recorded speech or music.
- Handwriting, such as doctor's notes.

RNNs build models that take into account this sequential nature of the data, and build a memory of the past.

- The feature for each observation is a *sequence* of vectors $X = \{X_1, X_2, \ldots, X_L\}$.

Recurrent Neural Networks

Often data arise as sequences:

- Documents are sequences of words, and their relative positions have meaning.
- Time-series such as weather data or financial indices.
- Recorded speech or music.
- Handwriting, such as doctor's notes.

RNNs build models that take into account this sequential nature of the data, and build a memory of the past.

- The feature for each observation is a *sequence* of vectors $X = \{X_1, X_2, \ldots, X_L\}$.
- The target Y is often of the usual kind — e.g. a single variable such as `Sentiment`, or a one-hot vector for multiclass.

Recurrent Neural Networks

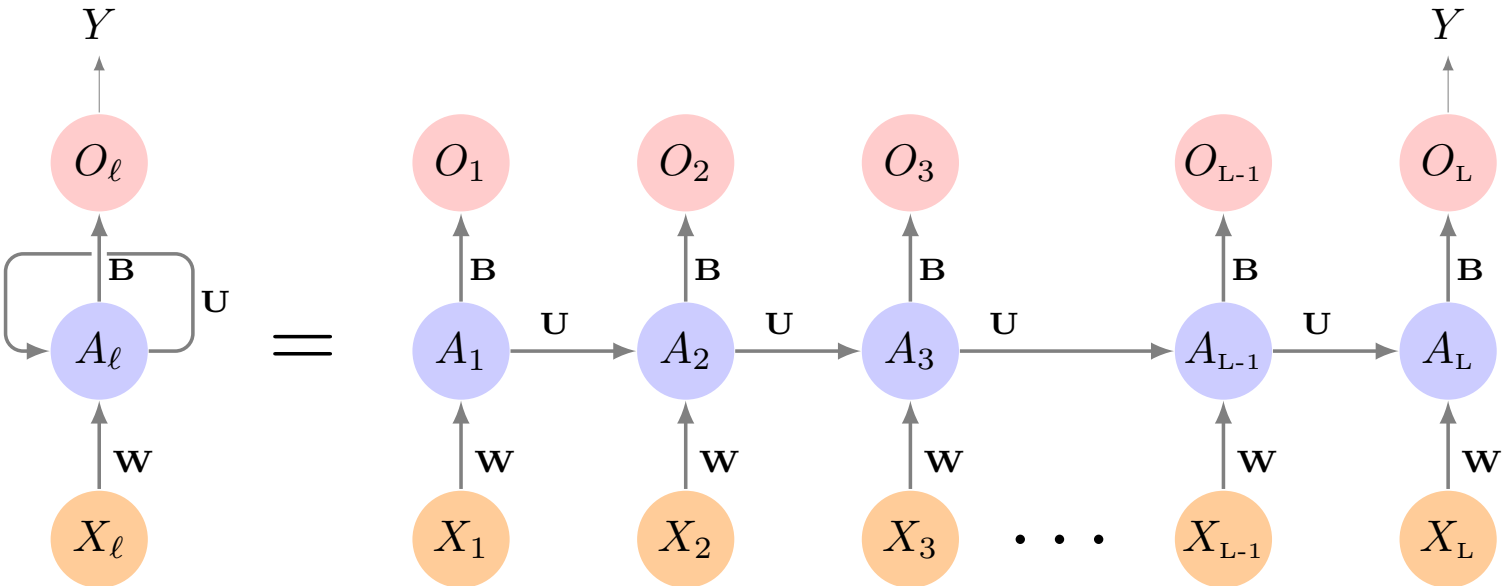
Often data arise as sequences:

- Documents are sequences of words, and their relative positions have meaning.
- Time-series such as weather data or financial indices.
- Recorded speech or music.
- Handwriting, such as doctor's notes.

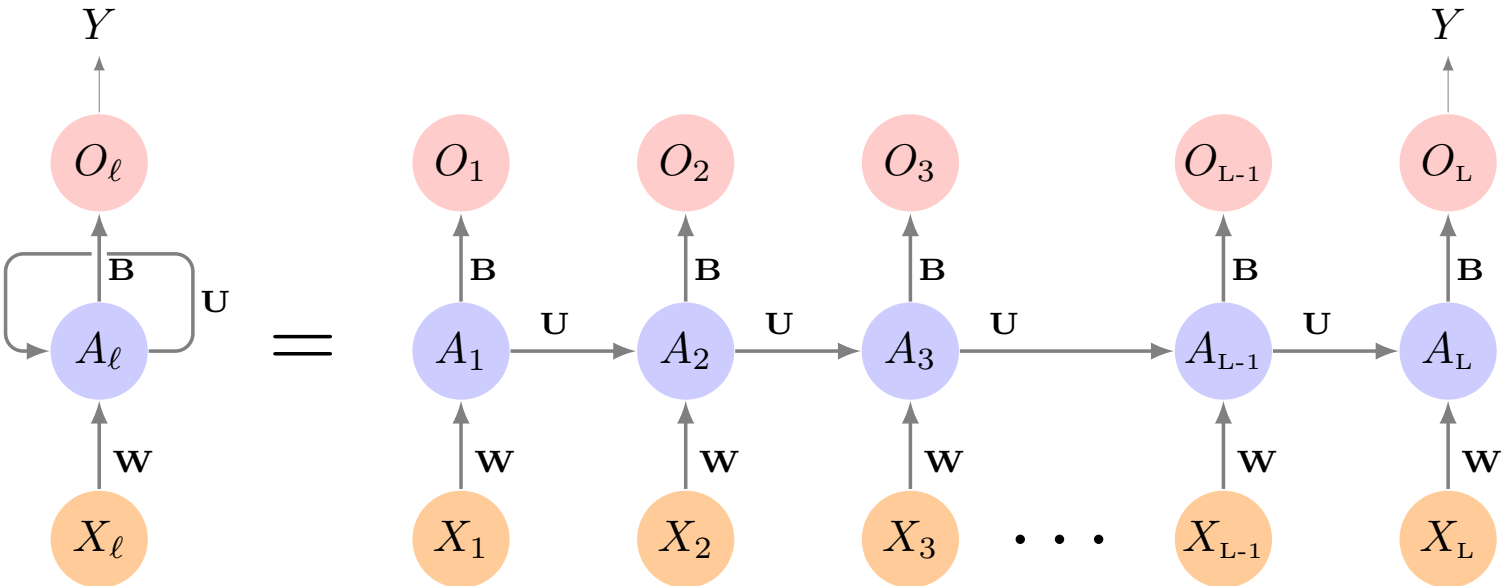
RNNs build models that take into account this sequential nature of the data, and build a memory of the past.

- The feature for each observation is a *sequence* of vectors $X = \{X_1, X_2, \ldots, X_L\}$.
- The target Y is often of the usual kind — e.g. a single variable such as `Sentiment`, or a one-hot vector for multiclass.
- However, Y can also be a sequence, such as the same document in a different language.

Simple Recurrent Neural Network Architecture

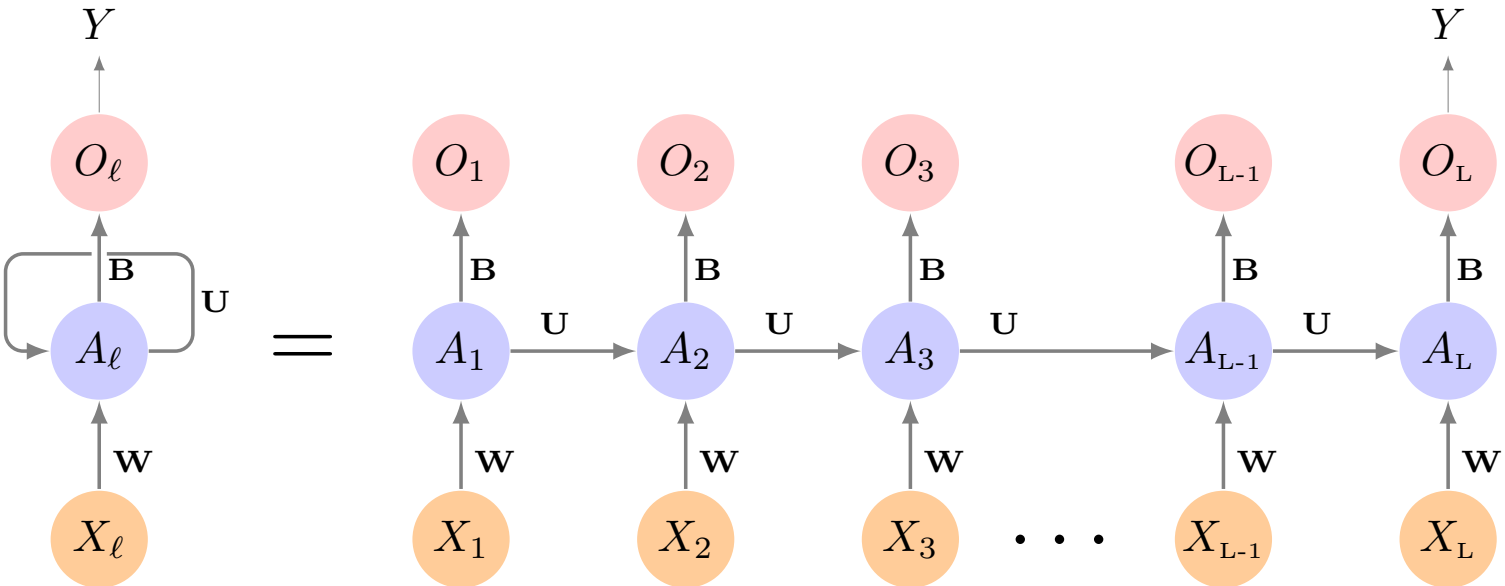


Simple Recurrent Neural Network Architecture



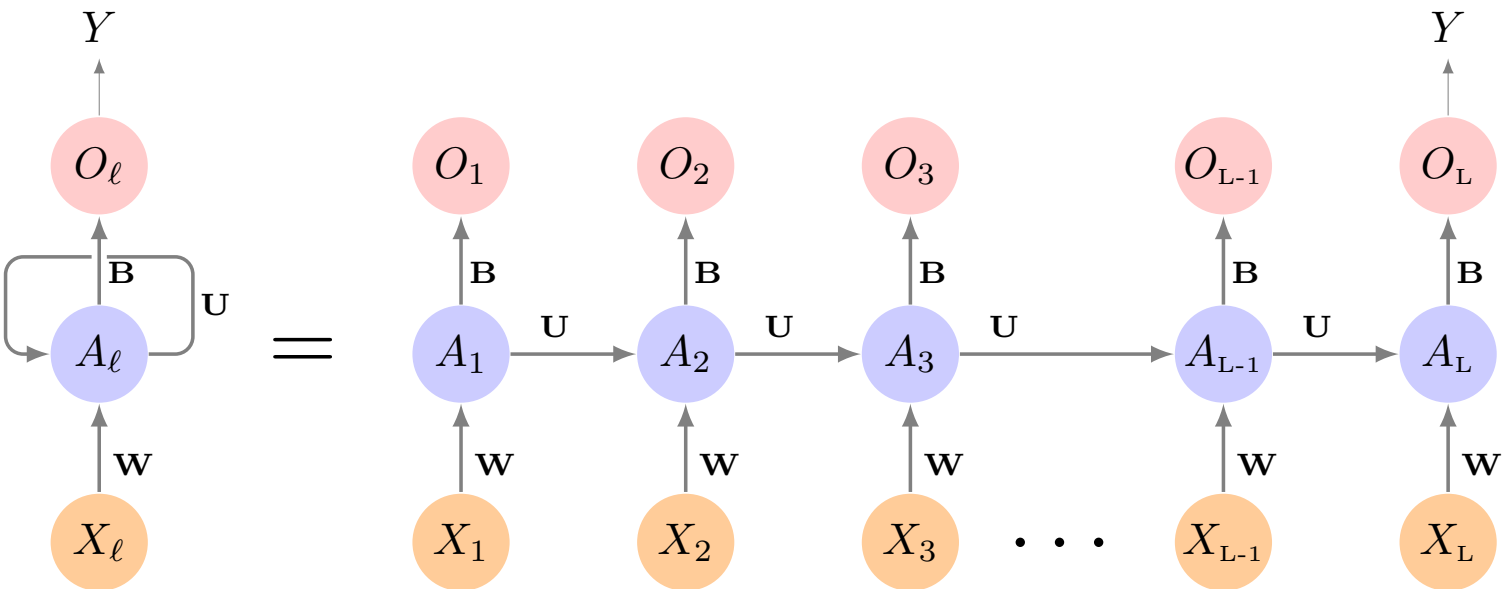
- The hidden layer is a sequence of vectors A_ℓ, receiving as input X_ℓ as well as $A_{\ell-1}$. A_ℓ produces an output O_ℓ.

Simple Recurrent Neural Network Architecture



- The hidden layer is a sequence of vectors A_ℓ, receiving as input X_ℓ as well as $A_{\ell-1}$. A_ℓ produces an output O_ℓ.
- The *same* weights $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{B}$ are used at each step in the sequence — hence the term *recurrent*.

Simple Recurrent Neural Network Architecture

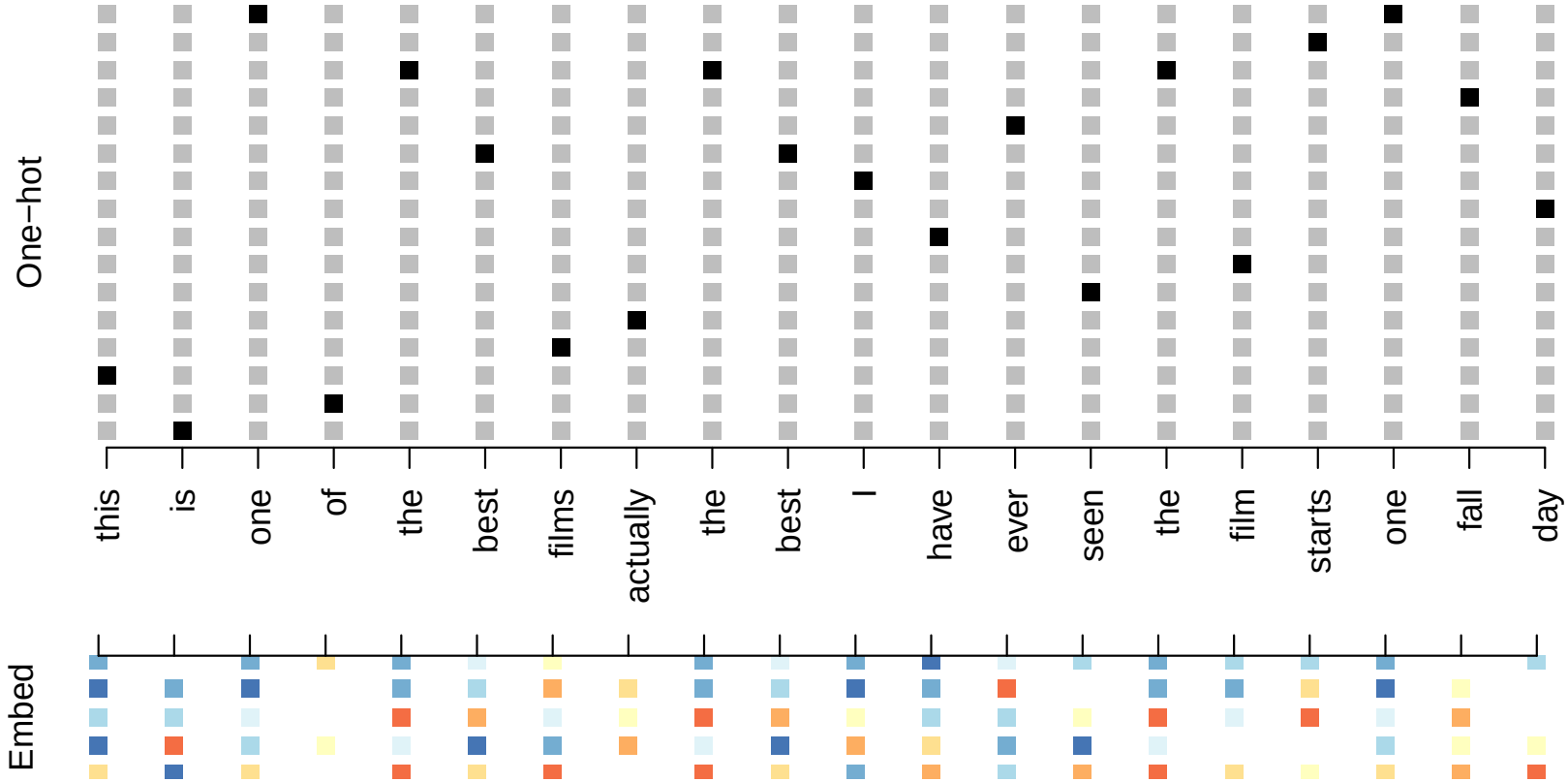


- The hidden layer is a sequence of vectors A_ℓ, receiving as input X_ℓ as well as $A_{\ell-1}$. A_ℓ produces an output O_ℓ.
- The *same* weights $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{B}$ are used at each step in the sequence — hence the term *recurrent*.
- The A_ℓ sequence represents an evolving model for the response that is updated as each element X_ℓ is processed.

RNN and IMDB Reviews

- The document feature is a sequence of words $\{\mathcal{W}_\ell\}_1^L$. We typically truncate/pad the documents to the same number L of words (we use $L = 500$).
- Each word $\mathcal{W}_\ell$ is represented as a *one-hot encoded* binary vector X_ℓ (dummy variable) of length $10K$, with all zeros and a single one in the position for that word in the dictionary.
- This results in an extremely sparse feature representation, and would not work well.
- Instead we use a lower-dimensional pretrained *word embedding* matrix $\mathbf{E}$ ($m \times 10K$, next slide).
- This reduces the binary feature vector of length $10K$ to a real feature vector of dimension $m \ll 10K$ (e.g. m in the low hundreds.)

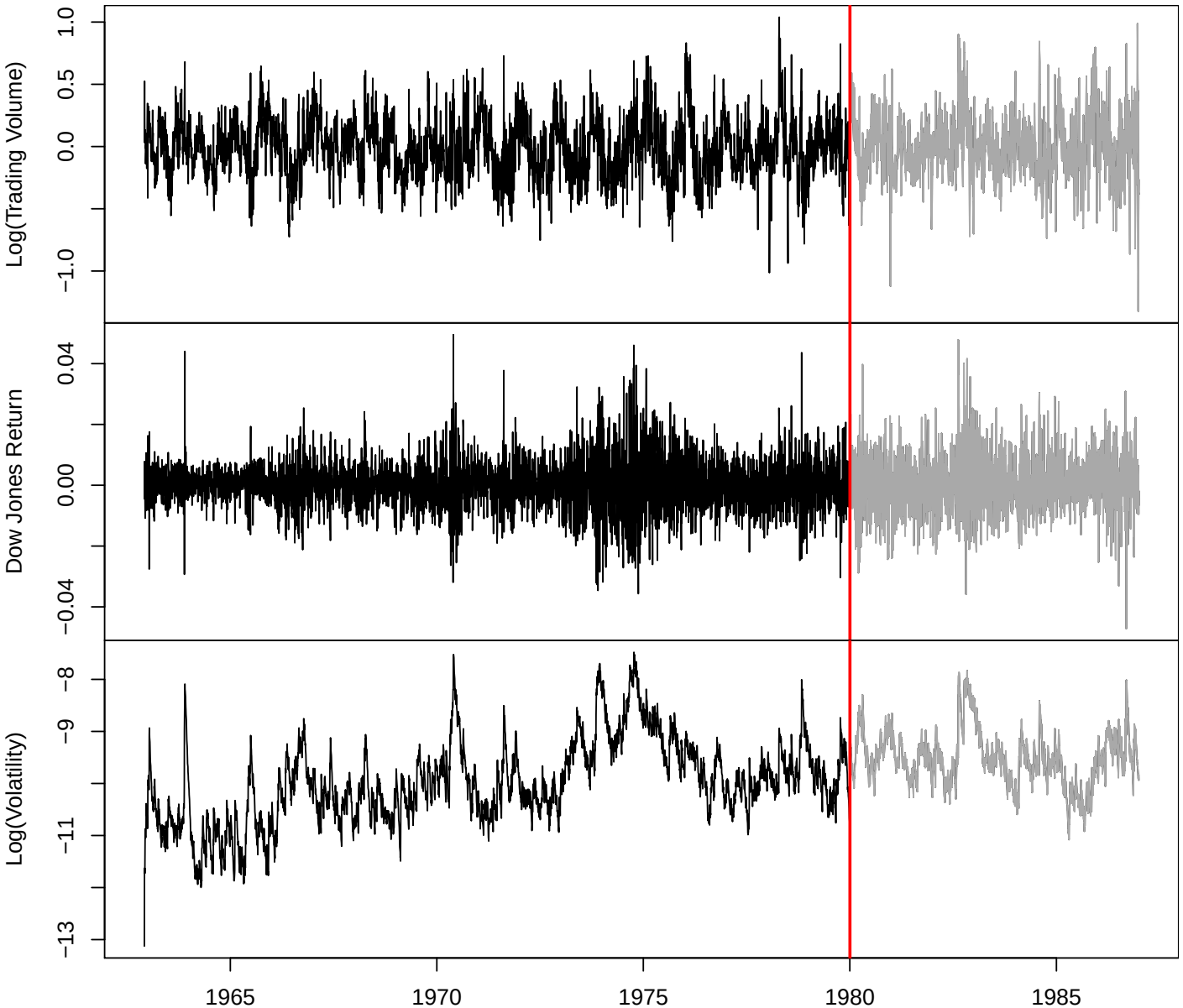
Word Embedding



this is one of the best films actually the best I have ever seen the film starts one fall day $\cdots$.

Embeddings are pretrained on very large corpora of documents, using methods similar to principal components. `word2vec` and `GloVe` are popular.

Time Series Forecasting



Summary of RNNs

- We have presented the simplest of RNNs. Many more complex variations exist.
- One variation treats the sequence as a one-dimensional image, and uses CNNs for fitting. For example, a sequence of words using an embedding representation can be viewed as an image, and the CNN convolves by sliding a convolutional filter along the sequence.
- Can have additional hidden layers, where each hidden layer is a sequence, and treats the previous hidden layer as an input sequence.
- Can have output also be a sequence, and input and output share the hidden units. So called `seq2seq` learning are used for language translation.

When to Use Deep Learning

- CNNs have had enormous successes in image classification and modeling, and are starting to be used in medical diagnosis. Examples include digital mammography, ophthalmology, MRI scans, and digital X-rays.

When to Use Deep Learning

- CNNs have had enormous successes in image classification and modeling, and are starting to be used in medical diagnosis. Examples include digital mammography, ophthalmology, MRI scans, and digital X-rays.
- RNNs have had big wins in speech modeling, language translation, and forecasting.

When to Use Deep Learning

- CNNs have had enormous successes in image classification and modeling, and are starting to be used in medical diagnosis. Examples include digital mammography, ophthalmology, MRI scans, and digital X-rays.
- RNNs have had big wins in speech modeling, language translation, and forecasting.

Should we always use deep learning models?

When to Use Deep Learning

- CNNs have had enormous successes in image classification and modeling, and are starting to be used in medical diagnosis. Examples include digital mammography, ophthalmology, MRI scans, and digital X-rays.
- RNNs have had big wins in speech modeling, language translation, and forecasting.

Should we always use deep learning models?

- Often the big successes occur when the *signal to noise ratio* is high — e.g. image recognition and language translation. Datasets are large, and overfitting is not a big problem.

When to Use Deep Learning

- CNNs have had enormous successes in image classification and modeling, and are starting to be used in medical diagnosis. Examples include digital mammography, ophthalmology, MRI scans, and digital X-rays.
- RNNs have had big wins in speech modeling, language translation, and forecasting.

Should we always use deep learning models?

- Often the big successes occur when the *signal to noise ratio* is high — e.g. image recognition and language translation. Datasets are large, and overfitting is not a big problem.
- For noisier data, simpler models can often work better.
  - On the `NYSE` data, the AR(5) model is much simpler than a RNN, and performed as well.
  - On the `IMDB` review data, the linear model fit by `glmnet` did as well as the neural network, and better than the RNN.

When to Use Deep Learning

- CNNs have had enormous successes in image classification and modeling, and are starting to be used in medical diagnosis. Examples include digital mammography, ophthalmology, MRI scans, and digital X-rays.
- RNNs have had big wins in speech modeling, language translation, and forecasting.

Should we always use deep learning models?

- Often the big successes occur when the *signal to noise ratio* is high — e.g. image recognition and language translation. Datasets are large, and overfitting is not a big problem.
- For noisier data, simpler models can often work better.
  - On the `NYSE` data, the AR(5) model is much simpler than a RNN, and performed as well.
  - On the `IMDB` review data, the linear model fit by `glmnet` did as well as the neural network, and better than the RNN.
- We endorse the *Occam's razor* principal — we prefer simpler models if they work as well. More interpretable!

Tricks of the Trade

- *Slow learning*. Gradient descent is slow, and a small learning rate ρ slows it even further. With *early stopping*, this is a form of regularization.

Tricks of the Trade

- *Slow learning*. Gradient descent is slow, and a small learning rate ρ slows it even further. With *early stopping*, this is a form of regularization.
- *Stochastic gradient descent*. Rather than compute the gradient using *all* the data, use a small *minibatch* drawn at random at each step. E.g. for `MNIST` data, with $n = 60K$, we use minibatches of 128 observations.

Tricks of the Trade

- *Slow learning*. Gradient descent is slow, and a small learning rate ρ slows it even further. With *early stopping*, this is a form of regularization.
- *Stochastic gradient descent*. Rather than compute the gradient using *all* the data, use a small *minibatch* drawn at random at each step. E.g. for `MNIST` data, with $n = 60K$, we use minibatches of 128 observations.
- An *epoch* is a count of iterations and amounts to the number of minibatch updates such that n samples in total have been processed; i.e. $60K/128 \approx 469$ for `MNIST`.

Tricks of the Trade

- *Slow learning*. Gradient descent is slow, and a small learning rate ρ slows it even further. With *early stopping*, this is a form of regularization.
- *Stochastic gradient descent*. Rather than compute the gradient using *all* the data, use a small *minibatch* drawn at random at each step. E.g. for `MNIST` data, with $n = 60K$, we use minibatches of 128 observations.
- An *epoch* is a count of iterations and amounts to the number of minibatch updates such that n samples in total have been processed; i.e. $60K/128 \approx 469$ for `MNIST`.
- *Regularization*. Ridge and lasso regularization can be used to shrink the weights at each layer. Two other popular forms of regularization are *dropout* and *augmentation*, discussed next.

Dropout Learning

- At each SGD update, randomly remove units with probability ϕ, and scale up the weights of those retained by $1/(1-\phi)$ to compensate.

Dropout Learning

- At each SGD update, randomly remove units with probability ϕ, and scale up the weights of those retained by $1/(1-\phi)$ to compensate.
- In simple scenarios like linear regression, a version of this process can be shown to be equivalent to ridge regularization.

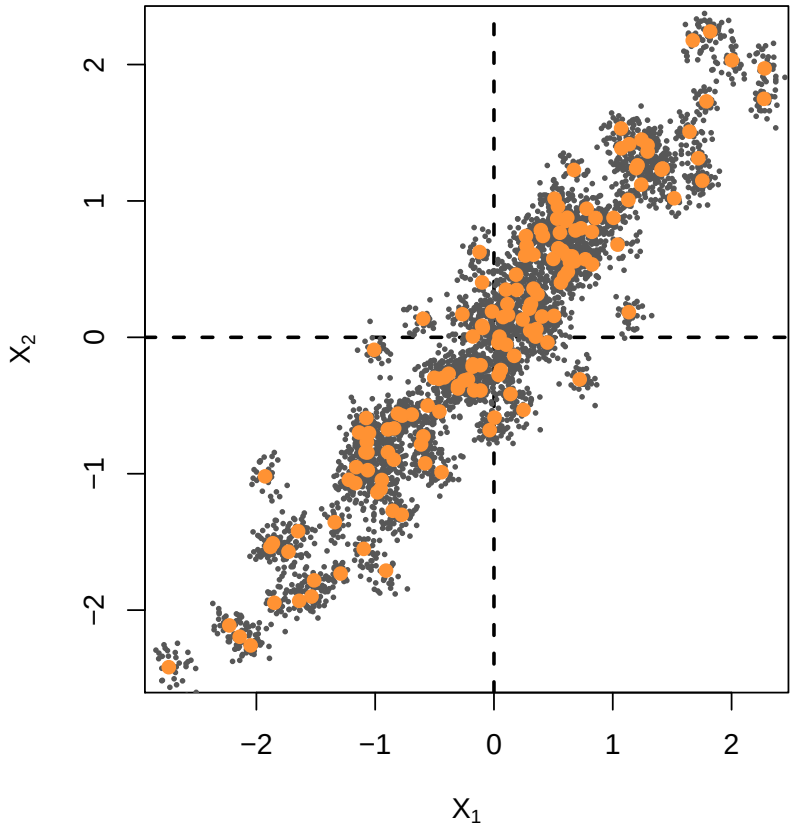
Dropout Learning

- At each SGD update, randomly remove units with probability ϕ, and scale up the weights of those retained by $1/(1-\phi)$ to compensate.
- In simple scenarios like linear regression, a version of this process can be shown to be equivalent to ridge regularization.
- As in ridge, the other units *stand in* for those temporarily removed, and their weights are drawn closer together.

Dropout Learning

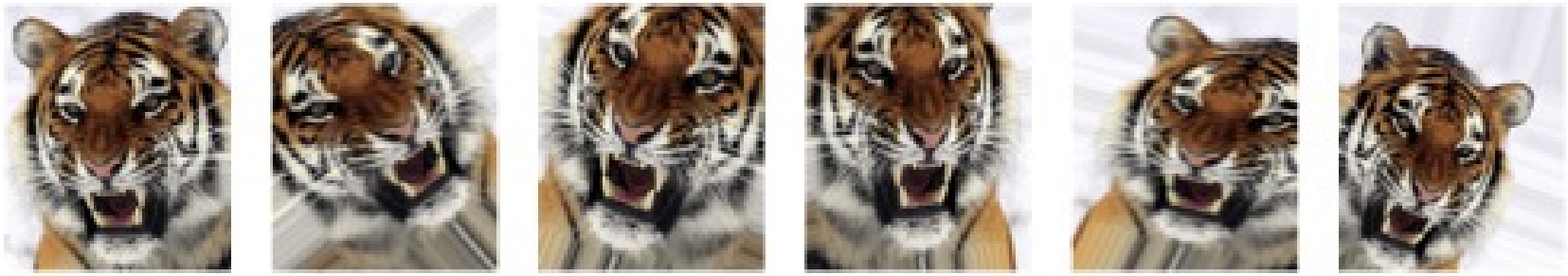
- At each SGD update, randomly remove units with probability ϕ, and scale up the weights of those retained by $1/(1-\phi)$ to compensate.
- In simple scenarios like linear regression, a version of this process can be shown to be equivalent to ridge regularization.
- As in ridge, the other units *stand in* for those temporarily removed, and their weights are drawn closer together.
- Similar to randomly omitting variables when growing trees in random forests (Chapter 8).

Ridge and Data Augmentation



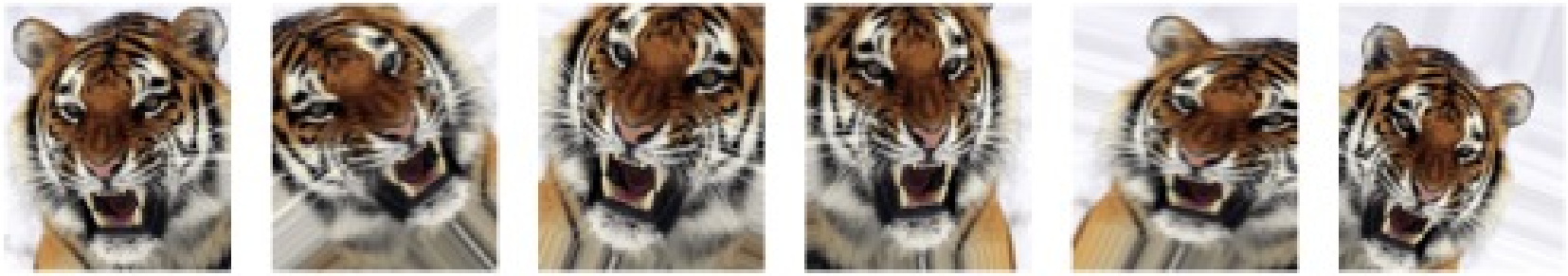
- Make many copies of each (x_i, y_i) and add a small amount of Gaussian noise to the x_i — a little cloud around each observation — but *leave the copies of y_i alone!*
- This makes the fit robust to small perturbations in x_i, and is equivalent to ridge regularization in an OLS setting.

Data Augmentation on the Fly



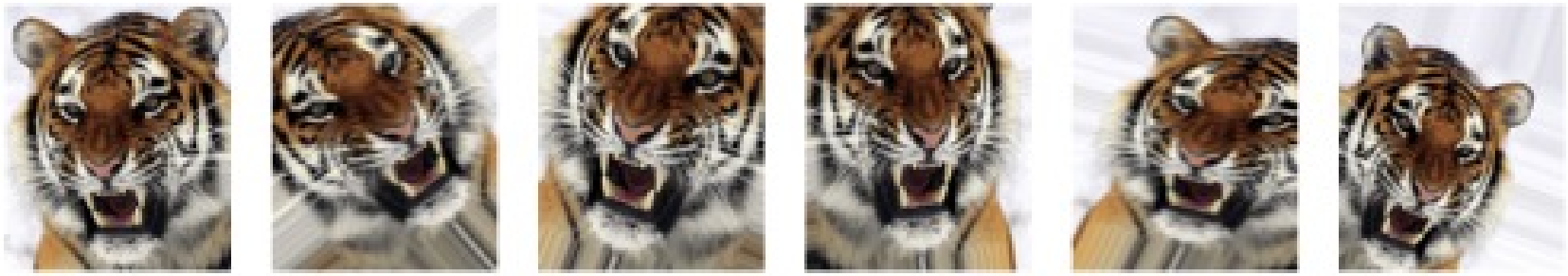
- Data augmentation is especially effective with SGD, here demonstrated for a CNN and image classification.

Data Augmentation on the Fly



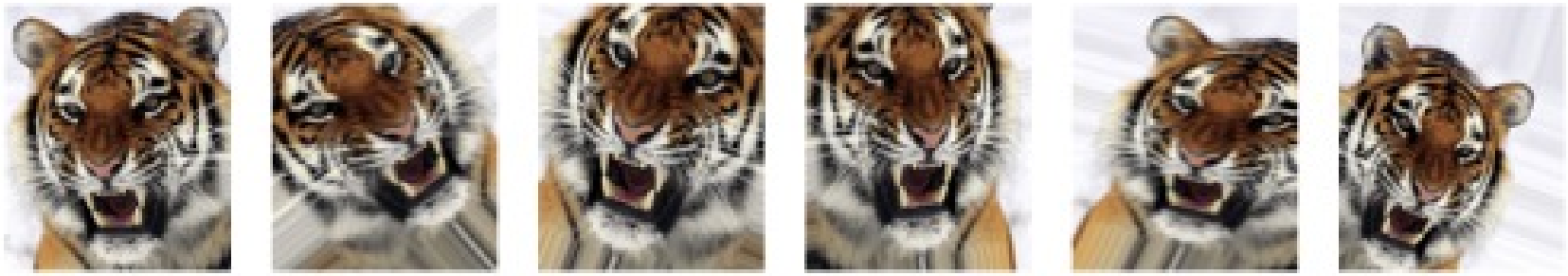
- Data augmentation is especially effective with SGD, here demonstrated for a CNN and image classification.
- Natural transformations are made of each training image when it is sampled by SGD, thus ultimately making a cloud of images around each original training image.

Data Augmentation on the Fly



- Data augmentation is especially effective with SGD, here demonstrated for a CNN and image classification.
- Natural transformations are made of each training image when it is sampled by SGD, thus ultimately making a cloud of images around each original training image.
- The label is left unchanged — in each case still `tiger`.

Data Augmentation on the Fly



- Data augmentation is especially effective with SGD, here demonstrated for a CNN and image classification.
- Natural transformations are made of each training image when it is sampled by SGD, thus ultimately making a cloud of images around each original training image.
- The label is left unchanged — in each case still `tiger`.
- Improves performance of CNN and is similar to ridge.

Software

- Wonderful software available for neural networks and deep learning. `Tensorflow` from Google and `PyTorch` from Facebook. Both are `Python` packages.
- In the Chapter 10 lab we demonstrate `tensorflow` and `keras` packages in `R`, which interface to Python. See textbook and online resources for `Rmarkdown` and `Jupyter` notebooks for these and all labs for the second edition of ISLR book.
- The `torch` package in R is available as well, and implements the `PyTorch` dialect. The Chapter 10 lab will be available in this dialect as well; watch the resources page at `www.statlearning.com`.

Summary

- Deep learning is an area of research in machine learning.
- Multilayer neural networks consist of multiple hidden layers, each containing numerous interconnected units called neurons.
- Convolutional neural network (CNNs) evolved for classifying images by recognizing specific features, distinguishing each particular object class.
- Recurrent Neural Network is used for predictive models for sequential data in nature

Thank you.
Any questions?