Chapter 5: Machine Learning

Quan Xiao
University of Science and Technology

Presented at USTC

May 29, 2019

Outline of this presentation

1. Strong and Weak learning-Boosting
2. Stochastic Gradient Descent
3. Deep learning
4. Further Current Directions

Outline

1. Strong and Weak learning-Boosting
2. Stochastic Gradient Descent
3. Deep learning
4. Further Current Directions

- The strong learner for a problem is an algorithm that with high probability is able to achieve any desired error rate ϵ using a number of samples that may depend polynomially on $1/\epsilon$.
- The weak learner for a problem is an algorithm that does just a little bit better than random guessing. It is only required to get with high probability an error rate less than of equal to $\frac{1}{2} - \gamma$ for some $0 < \gamma \leq \frac{1}{2}$.
- Boosting is a method to construct a strong learner by taking the majority vote of many weak learning algorithms.

Some notations:

- A: a weak learning algorithm
- $\mathcal{H}$: hypotheses class
- t_0: number of learning rounds
- $\mathrm{MAJ}\left(h_1, \ldots, h_{t_0}\right)$: The function taking the majority vote of the hypotheses returned by the weak learner

Assumption:

- When presented with a weighting of the points in our training sample, A always produces a hypothesis that performs slightly better than random guessing with respect to the distribution induced by weighting.

Intuitive notion:

- If an example was misclassified, one needs to pay more attention to it.

Boosting Algorithm

- Given a sample S of n labeled examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$, initialize each example $\mathbf{x}_i$ to have a weight $\mathrm{w}_i = 1$. Let $\mathbf{w} = (w_1, \ldots, w_n)$.
- For $t = 1, 2, \ldots, t_0$:
  - Call the weak learner on the weighted sample $(S, \mathbf{w})$, receiving hypothesis h_t.
  - Mutiply the weight of each example that was misclassified by h_t by $\alpha = \frac{\frac{1}{2}+\gamma}{\frac{1}{2}-\gamma}$. Leave the other weights as they are.
- End
- Output the classifier $\mathrm{MAJ}\left(h_1, \ldots, h_{t_0}\right)$. Assume t_0 is odd so there is no tie.

- Definition 5.4 ($\gamma-$ Weak learner on sample) : A $\gamma -$ *weak* learner is an algorithm that given examples, their labels, and a nonnegative real weight w_i on each example $\mathbf{x}_i$, produces a classifier that correctly labels a subest of examples with total weight at least $\left(\frac{1}{2} + \gamma\right) \sum_{i=1}^{n} w_i$.
- Theorem 5.21 : Let A be a $\gamma -$ *weak* learner for sample S. Then $t_0 = O\left(\frac{1}{\gamma^2} \log n\right)$ is sufficient so that the classifier $\mathrm{MAJ}\left(h_1, \ldots, h_{t_0}\right)$ produced by the boosting procedure has training error zero.

Outline

Some notations:

- $\mathcal{F}$: a class of real-valued functions $f_{\mathrm{w}} : \mathbb{R}^d \to \mathbb{R}$ where $\mathbf{w} = (w_1, \ldots, w_n)$ is a vector of parameters
- $h_{\mathbf{w}} = \{\mathbf{x} : f_{\mathbf{w}}(\mathbf{x}) \geq 0\}$.
- $\mathcal{H}_{\mathcal{F}} = \{h_{\mathrm{w}} : f_{\mathrm{w}} \in \mathcal{F}\}$.
- $L\left(f_{\mathbf{w}}(\mathbf{x}), c^*(\mathrm{x})\right)$: loss function describing the real-valued penalty we will associate with function $f_{\mathbf{w}}$ for its prediction on an example x whose true label is $c^*(\mathrm{x})$.

Stochastic Gradient Descent

- Given: staring point $\mathbf{w} = \mathbf{w}_{init}$ and learning rates $\lambda_1, \lambda_2, \lambda_3, \ldots$ (e.g.,$\mathbf{w}_{init} = 0$ and $\lambda_t = 1$ for all t, or $\lambda_t = 1/\sqrt{t}$).
- Consider a sequence of random examples $(\mathrm{x}_1, c^*(\mathrm{x}_1)), (\mathrm{x}_2, c^*(\mathrm{x}_2)), \ldots$
  - Given example $(\mathrm{x}_t, c^*(\mathrm{x}_t))$, compute the gradient $\nabla L(f_{\mathbf{w}}(\mathbf{x}_t), c^*(\mathbf{x}_t))$ of the loss of $f_{\mathbf{w}}(\mathbf{x}_t)$ with respect to the weights w. This is a vector in $\mathbb{R}^n$ whose ith component is $\frac{\partial L(f_{\mathbf{w}}(\mathbf{x}_t), c^*(\mathbf{x}_t))}{\partial w_i}$.
  - Update: $\mathbf{w} \leftarrow \mathbf{w} - \lambda_t \nabla L(f_{\mathbf{w}}(\mathbf{x}_t), c^*(\mathbf{x}_t))$

- Examples:
- Consider $n = d$, $f_{\mathbf{w}}(\mathrm{x}) = \mathbf{w}^T\mathbf{x}$, loss function $L(f_{\mathbf{w}}(\mathrm{x}), c^*(\mathrm{x})) = \max(0, -c^*(\mathrm{x}) f_{\mathrm{w}}(\mathrm{x}))$, $c^*(\mathrm{x}) \in \{-1, 1\}$.

Stochastic Gradient Descent

- Given: staring point $\mathbf{w} = \mathbf{w}_{init}$ and learning rates $\lambda_1, \lambda_2, \lambda_3, \ldots$ (e.g.,$\mathbf{w}_{init} = 0$ and $\lambda_t = 1$ for all t, or $\lambda_t = 1/\sqrt{t}$).
- Consider a sequence of random examples $\left(\mathrm{x}_1, c^*\left(\mathrm{x}_1\right)\right), \left(\mathrm{x}_2, c^*\left(\mathrm{x}_2\right)\right), \ldots$
  - Given example $\left(\mathrm{x}_t, c^*\left(\mathrm{x}_t\right)\right)$, compute the gradient $\nabla L\left(f_{\mathbf{w}}\left(\mathbf{x}_t\right), c^*\left(\mathbf{x}_t\right)\right)$ of the loss of $f_{\mathbf{w}}\left(\mathbf{x}_t\right)$ with respect to the weights w. This is a vector in $\mathbb{R}^n$ whose ith component is $\frac{\partial L\left(f_{\mathbf{w}}\left(\mathbf{x}_t\right), c^*\left(\mathbf{x}_t\right)\right)}{\partial w_i}$.
  - Update: $\mathbf{w} \leftarrow \mathbf{w} - \lambda_t \nabla L\left(f_{\mathbf{w}}\left(\mathbf{x}_t\right), c^*\left(\mathbf{x}_t\right)\right)$

- Examples:
- Consider $n = d$, $f_{\mathbf{w}}(\mathrm{x}) = \mathbf{w}^T\mathbf{x}$, loss function is hinge-loss $L\left(f_{\mathrm{w}}(\mathrm{x}), c^*(\mathrm{x})\right) = \max(0, 1 - c^*(\mathrm{x}) f_{\mathrm{w}}(\mathrm{x}))$

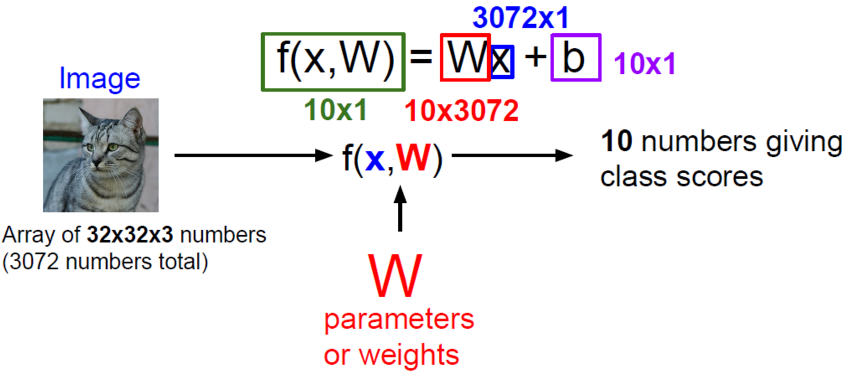
Outline

Linear classifier

50,000 training images
each image is **32x32x3**

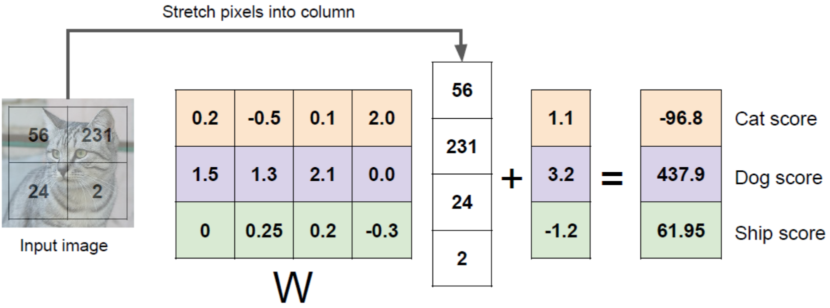
10,000 test images.

Linear classifier



Linear classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



- Define a loss function that measure our unhappiness with the scores across the training data.
- Start with a random W and find a W that minimizes the loss. We usually use SGD to optimize W.

- Given training examples $\mathrm{x}_1, \mathrm{x}_2, \ldots$ and the corresponding labels $c^*(\mathrm{x}_1), c^*(\mathrm{x}_2), \ldots$.(noted as $\mathrm{y}_1, \mathrm{y}_2, \ldots$)

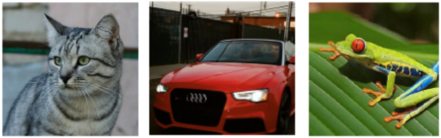
- Loss over the dataset is the average of loss over examples $L = \frac{1}{N}\sum_i L_i(f(x_i, W))$
- Denote $s = f(x; W) \overset{\text{e.g.}}{=} Wx$

Some common loss functions:

- *SVM* loss: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
- *Softmax* loss: $L_i = -\log P(Y = y_i | X = x_i) = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$

Softmax classifier (Multinomial Logistic Regression)

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities	
cat	**3.2**	→	**24.5**	→	**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

unnormalized log probabilities

probabilities

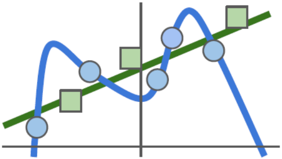
Regularization

- A way of preventing the model from overfitting

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data



Occam's Razor:
"Among competing hypotheses, the simplest is the best"
William of Ockham, 1285 - 1347

- So, the full loss is $L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$

Optimization

- Stochastic gradient descend (*SGD*)
- full loss $L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$
- gradient $\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i\left(x_i, y_i, W\right) + \lambda \nabla_W R(W)$
- Full sum of loss is too expensive. So, in practice, we approximate sum using a minibatch of examples. 32/63/128 are commonly used.

Backpropagation

Backpropagation: a simple example

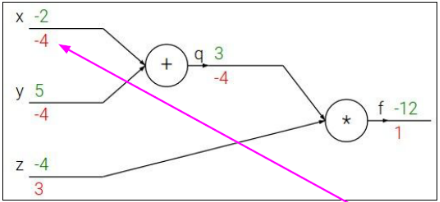
$$f(x, y, z) = (x + y)z$$

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

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

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

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



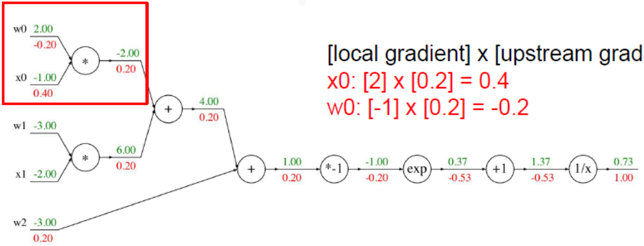
$$\boxed{\frac{\partial f}{\partial x}}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}}$$

Backpropagation

Another example: $f(w,x) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[local gradient] x [upstream gradient]
x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\Big|\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

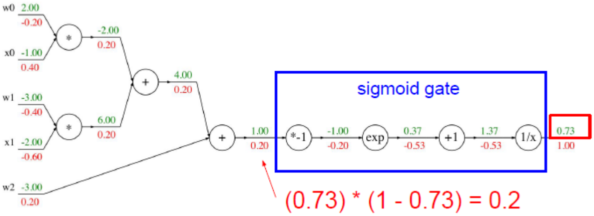
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \qquad\Big|\qquad f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Backpropagation

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

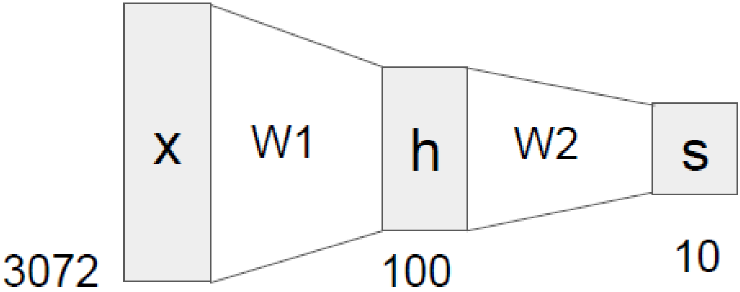
$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$ sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



Neural Network: Fully-connected layer

- Before: linear model $f = Wx$
- Now: 2-layer neural network $f = W_2 \max(0, W_1 x)$

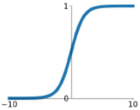


- hiden layer

Activation Function

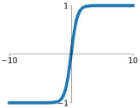
Sigmoid

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



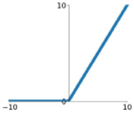
tanh

$\tanh(x)$



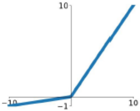
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

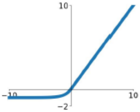


Maxout

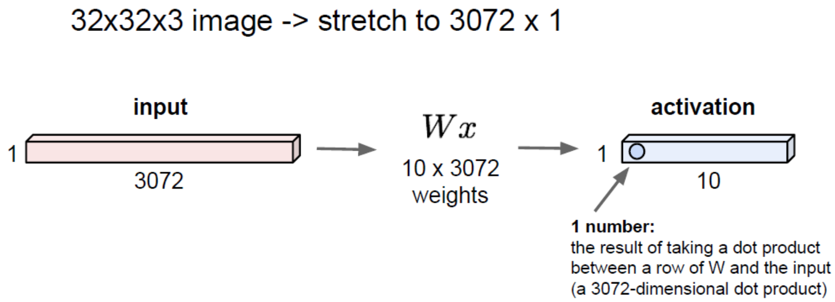
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

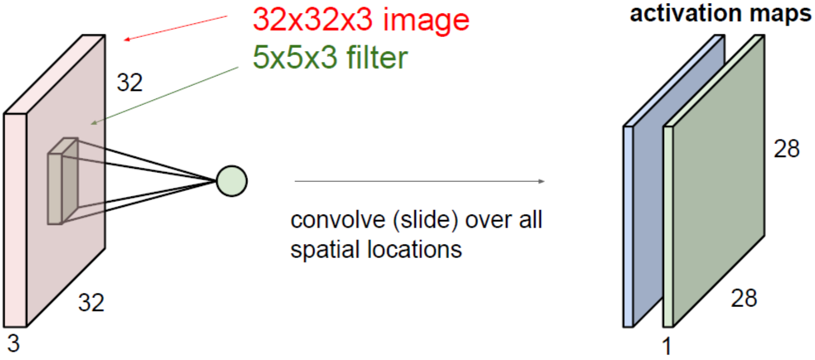
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Fully-connected layer

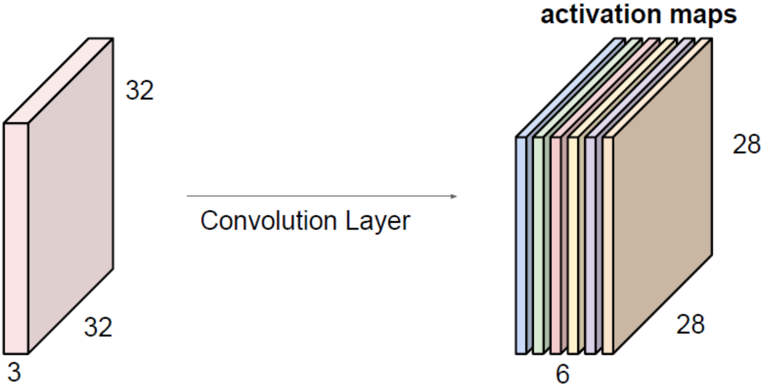


Convolutional Neural Network



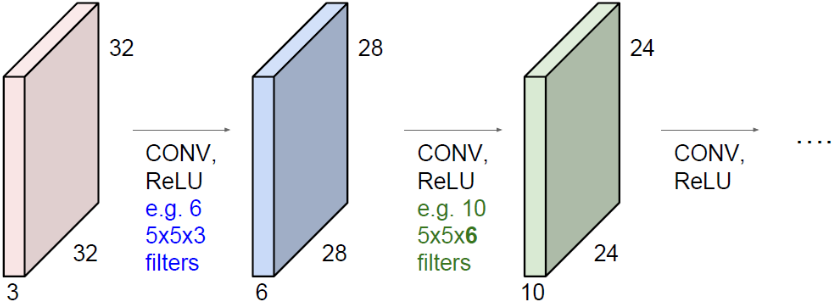
Convolutional Neural Network

- For example, if we have 6 filters, then we'll get 6 sperate activation map.



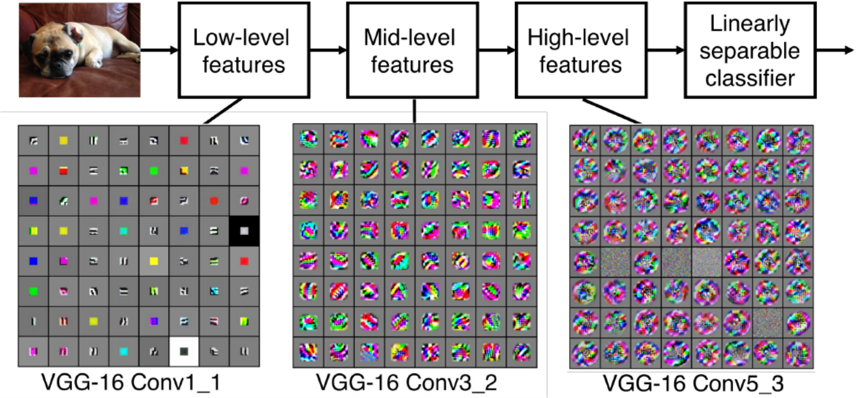
Convolutional Neural Network

- ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.



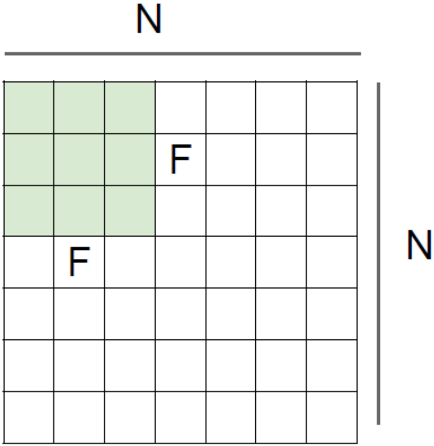
Convolutional Neural Network

- This Convolutional layers can be viewed as searching for features.



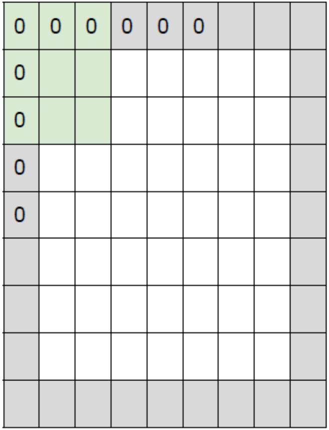
Zero padding

- output size is $(\mathrm{N}-\mathrm{F})/$ stride $+1$



Zero padding

- To maintain our output size, we introduce zero padding.
- For example, input 7×7 ,3×3 filter, applied with stride 1, pad with 1 pixel border

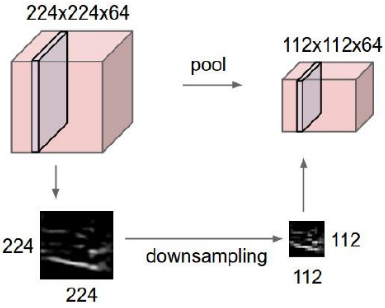


Convolutional Layer

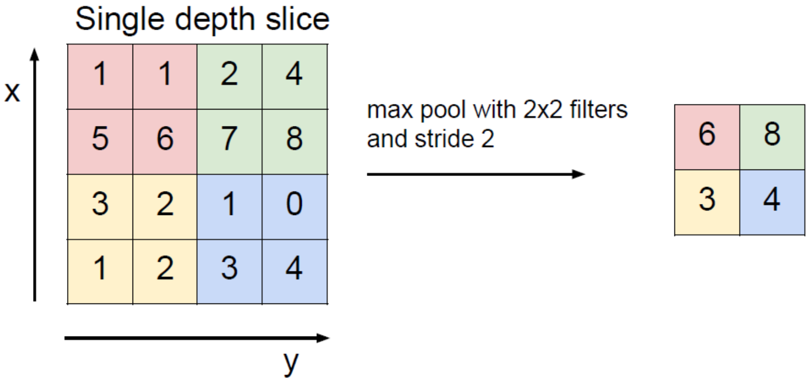
- Accept a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Numbers of filters K
  - their spatial extent F
  - the stride S
  - the amount of zero padding P
- Produces a volume of size $W_2 \times H_2 \times D_2$ where
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e.width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth since (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Pooling Layer

- makes the representations smaller and more manageable.
- operates over each activation map independently.



Max pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Pooling Layer

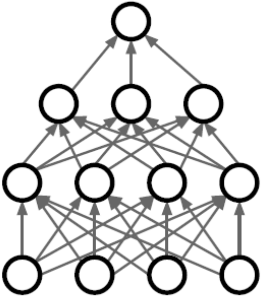
- Accept a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F
  - the stride S
- Produces a volume of size $W_2 \times H_2 \times D_2$ where
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layer

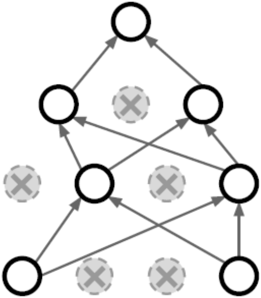
Avoid over fitting

- Over fitting is a major concern in deep learning since large networks can have hundreds of millions of weights. There are some ways to avoid it.
- Regularization.
- Drop out.
- Transfer learning.

Drop out

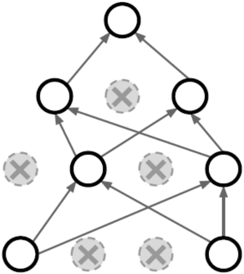
- In each forward pass, randomly set some neurons to zero. Probability of dropping is a hyperparameter; 0.5 is common.

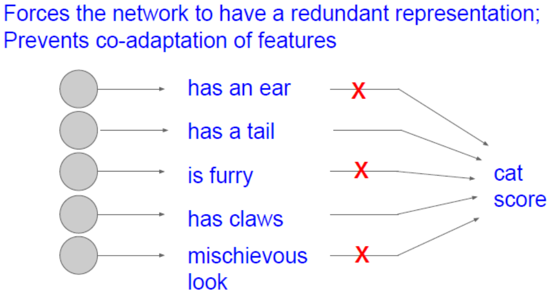




Drop out

- How can it be useful?
- It ensemble some of the models in the same model.





Transfer learning

- Train on a big dataset that has common features with your dataset. Called pretraining.
- Freeze the layers except the last layer and feed your small dataset to learn only the last layer.
- Not only the last layer maybe trained again, you can fine tune any number of layers you want based on the number of data you have.

Generative Adversarial Networks (GANs)

- GANs is a generative model.
- It involves two models. A generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G.
- The training procedure for G is to maximize the probability of D making a mistake.
- In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $1/2$ everywhere.

Generative Adversarial Networks (GANs)

- Some definitions:
  - p_g : the generator's distribution over data x
  - $p_{\mathbf{z}}(\mathbf{z})$: a prior on input noise variables
  - $G\left(z;\theta_g\right)$: a mapping to data space, generative model
  - $D(x)$: the probability that x came from the data rather than p_g, discriminative model
- We train D to maximize the probability of assigning the correct label to both training examples and samples from G.
- The D and G play the following two-players minimax game with value function $V(G,D)$:
- $\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))]$

GANs

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Update the generator by descending its stochastic gradient:

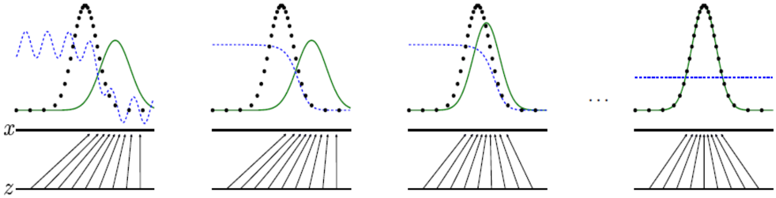
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GANs

- Proposition 1. For G fixed, the optimal discriminator D is: $D^*_G(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}$
- Theorem 2. The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value -log4.



(a) (b) (c) ... (d)

Outline

Semi-Supervised Learning

- Aim: To utilize the unlabeled data

Some definitions:

- D: distribution
- h: hypothesis
- $\chi(h, \mathcal{D})$: $\chi(h, \mathcal{D}) = 1$ means h is highly compatible with D
- $err_{\text{unl}}(h)$: $1 - \chi(h, \mathcal{D})$ unlabeled error rate of h

For example:

- Semi-Supervised SVMs, Co-Training, Graph-based methods

Semi-Supervised Learning

Theorem 5.23 *If $c^* \in \mathcal{H}$ then with probability at least $1-\delta$, for labeled set L and unlabeled set U drawn from $\mathcal{D}$, the $h \in \mathcal{H}$ that optimizes $\widehat{err}_{unl}(h)$ subject to $\widehat{err}(h) = 0$ will have $err_{\mathcal{D}}(h) \le \epsilon$ for*

$$|U| \ge \frac{2}{\epsilon^2}\left[\ln|\mathcal{H}| + \ln\frac{4}{\delta}\right], \text{ and } |L| \ge \frac{1}{\epsilon}\left[\ln|\mathcal{H}_{\mathcal{D},\chi}(err_{unl}(c^*) + 2\epsilon)| + \ln\frac{2}{\delta}\right].$$

Equivalently, for $|U|$ satisfying this bound, for any $|L|$, whp the $h \in \mathcal{H}$ that minimizes $\widehat{err}_{unl}(h)$ subject to $\widehat{err}(h) = 0$ has

$$err_{\mathcal{D}}(h) \le \frac{1}{|L|}\left[\ln|\mathcal{H}_{\mathcal{D},\chi}(err_{unl}(c^*) + 2\epsilon)| + \ln\frac{2}{\delta}\right].$$

Active Learning and Multi-Task learning

- Active learning: Take an active role in the selection of which examples are labeled.
- Multi-Task learning: Having many targets through a process