Lecture 20: neural networks

CS 475: Machine Learning

Raman Arora

April 17, 2017

Slides credit: Greg Shakhnarovich

Perceptron

- Consider binary classification task, $\mathcal{Y} = \{\pm 1\}$
- A very simple learning algorithm for linear classifiers $\widehat{y}_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + w_0)$:
  initialize $\mathbf{w} = \mathbf{0}$, $w_0 = 0$
  take one example $(\mathbf{x}_i, y_i)$ at a time
  if $y_i \left(\mathbf{w}^T\mathbf{x}_i + w_0\right) \leq 0$, update

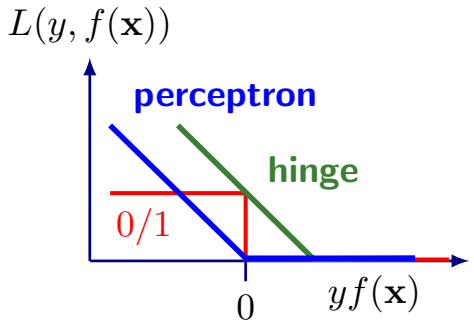
$$\mathbf{w} := \mathbf{w} + y_i\mathbf{x}_i, \quad w_0 := w_0 + y_i$$

  otherwise do nothing
  stop when all data are classified correctly
- Compare this to support vector machines; what is the loss?

Perceptron loss

- A mistake driven algorithm: updates weights only when making a mistake on the example
- What's the loss such that the (sub?)gradient is

$$\begin{cases} 0 & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i) > 0, \\ y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$



- We can modify perceptrons in various ways (regularize; use kernels; etc.)

Perceptron: analysis

- Assume data are linearly separable (otherise will never stop!)
- The final classifier:

$$\mathbf{w} = \sum_{t=1}^{T} \alpha_{t(i)} \mathbf{x}_{i(t)}$$

where T is the total number of iterations, $i(t)$ is the index of example used in t-th iteration, and $\alpha_{t(i)}$ is 0 or y_i, depending on what happened in t-th iteration.

- Let $\mathbf{w}$, w_0 be a linear separator, $\|\mathbf{w}\| = 1$, and the margin be γ. (we can always ensure $\|\mathbf{w}\| = 1$)
- Theorem (Novikoff, 1962): perceptron will converge after

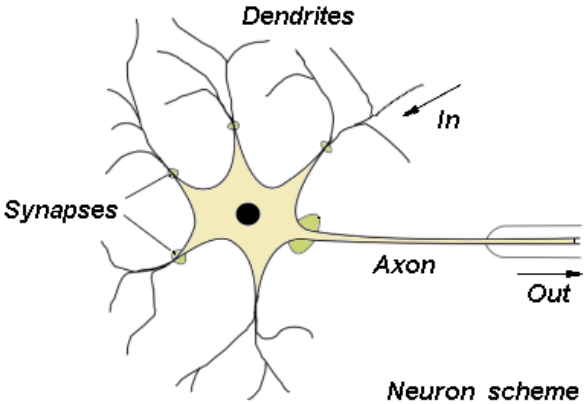
$$O\left(\frac{(\max_i \|\mathbf{x}_i\|)^2}{\gamma^2}\right)$$

Perceptron as a model of the brain?

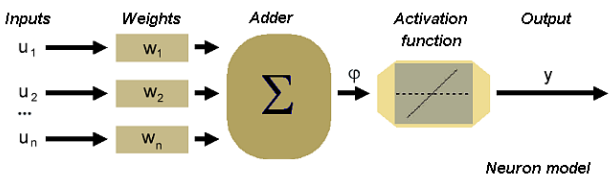
- Perceptron developed in 1950s
- Goal: pattern classification
- From "Mechanisation of Thought Process", 1959: "The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted"

Perceptron and neurons

- Real neurons (sort of)



- McCulloch-Pitts model, ca. 1943



source: `http://home.agh.edu.pl/~vlsi/AI`

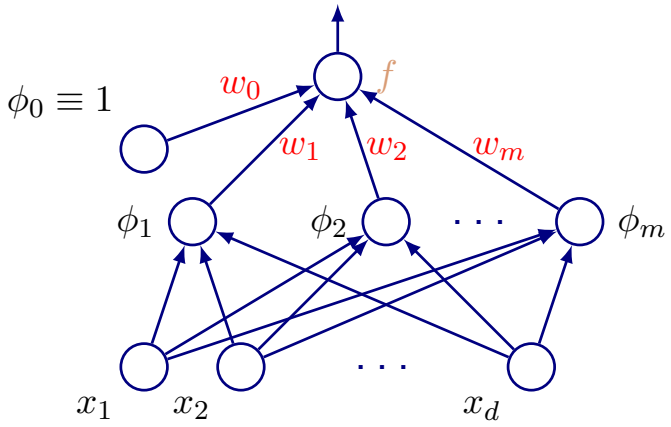
Neural networks

- General form of linear methods we have seen:

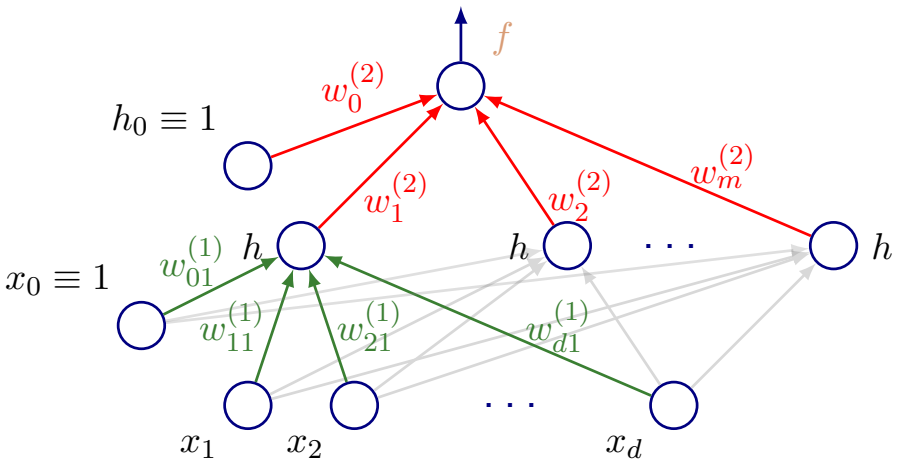
$$\hat{y}(\mathbf{x}; \mathbf{w}) \;=\; f\left(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})\right)$$

  logistic regression: $f(z) = (1 + \exp(z))^{-1}$,
  linear regression: $f(z) = z$.

- Representation as a *neural network*:



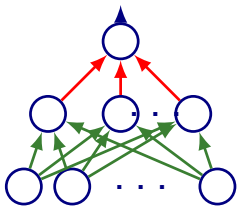
Two-layer network



- Idea: learn parametric features $\phi_j(\mathbf{x}) = h(\mathbf{w}_j{}^T \mathbf{x} + w_{0j})$ for some (possibly nonlinear) function h

Feed-forward networks

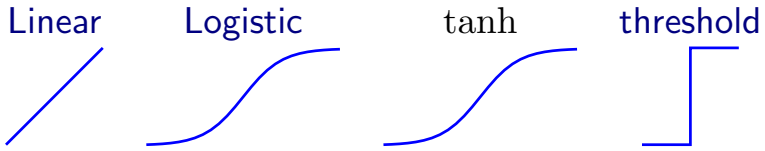
- Collect all the weights into a vector $\mathbf{w}$



- Feedforward operation, from input $\mathbf{x}$ to output $\hat{y}$:

$$\hat{y}(\mathbf{x}; \mathbf{w}) = f\left(\sum_{j=1}^{m} w_j^{(2)} h\left(\sum_{i=1}^{d} w_{ij}^{(1)} x_i + w_{0j}^{(1)}\right) + w_0^{(2)}\right)$$

- Common choices for f:



Training the network

- Error of the network on a training set:

$$L(X; \mathbf{w}) = \sum_{i=1}^{N} \frac{1}{2} \left(y_i - \hat{y}(\mathbf{x}_i; \mathbf{w})\right)^2$$

- Generally, no closed-form solution;
  resort to gradient descent
- Need to evaluate derivative of L on a single example
- Let's start with a simple linear model $\hat{y} = \sum_j w_j x_j$:

$$\frac{\partial L(\mathbf{x}_i)}{\partial w_j} = \underbrace{(\hat{y}_i - y_i)}_{\text{error}} x_{ij}.$$

Backpropagation

$$\hat{y}(\mathbf{x};\mathbf{w}) = f\left(\sum_{j=1}^{m} w_j^{(2)} h\left(\sum_{i=1}^{d} w_{ij}^{(1)} x_i + w_{0j}^{(1)}\right) + w_0^{(2)}\right)$$

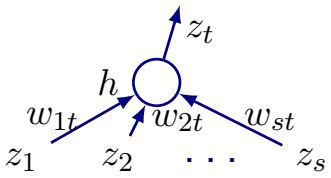
- Backpropagation: apply chain rule of derivation to

$$\frac{\partial L(\hat{y}, y)}{\partial w_j^{(k)}}$$

Backpropagation

- General unit activation in a multilayer network:

$$z_t = h\left(\sum_j w_{jt} z_j\right)$$



- Forward propagation: calculate for each unit $a_t = \sum_j w_{jt} z_j$
- The loss L depends on w_{jt} only through a_t:

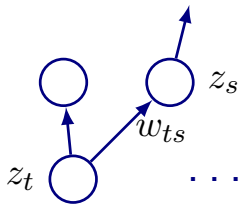
$$\frac{\partial L}{\partial w_{jt}} = \frac{\partial L}{\partial a_t}\frac{\partial a_t}{\partial w_{jt}} = \frac{\partial L}{\partial a_t} z_j$$

Backpropagation

$$\frac{\partial L}{\partial w_{jt}} = \delta_t z_j \qquad \delta_t = \frac{\partial L}{\partial a_t}$$

- Output unit with linear activation: $\delta_t = \hat{y} - y$
- Hidden unit $z_t = h(a_t)$ which sends inputs to units S:

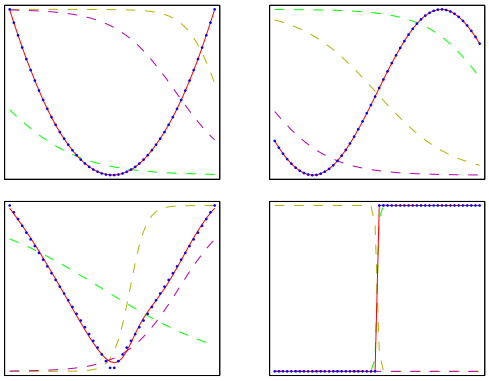
$$\begin{aligned}\delta_t &= \sum_{s \in S} \frac{\partial L}{\partial a_s} \frac{\partial a_s}{\partial a_t} \\ &= h'(a_t) \sum_{s \in S} w_{ts} \delta_s\end{aligned}$$



$$a_s = \sum_{j:j \to s} w_{js} h(a_j)$$

MLP as universal approximators

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!)
- Examples: 3 hidden units with $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$ activation



[from Bishop]

$$f(\mathbf{x}) = \mathbf{w}_2 \tanh(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + b$$

Deep vs shallow

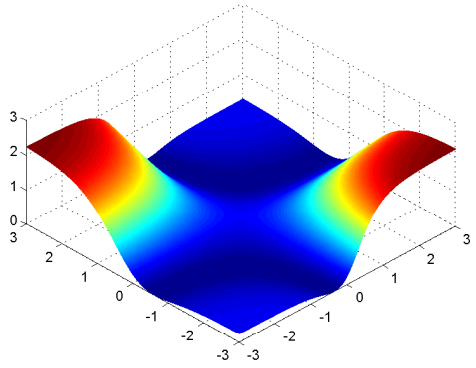
- What can we gain from depth?
- Example: parity of n-bit numbers, with AND, OR, NOT, XOR gates
- Trivial shallow architecture: express parity as DNF or CNF
  but need exponential number of gates!
- Deep architecture: a tree of XOR gates

Deep network loss optimization

- Example by Y. LeCun: the simplest 2-layer network ever

$$y = \tanh\left(w_1 \tanh(w_0 x)\right) \qquad x \in \{-.5, .5\}$$

- Loss surface as a function of w_0, w_1:
- We can expect highly non-linear, non-convex objectives



Classification networks

- Start with binary setup
- The final layer computes linear function of the previous layer

$$z_T = \mathbf{w}_T \mathbf{z}_{T-1} + b_T$$

  which we will interpret as the score of $y = 1$ for the input $\mathbf{x}$
- We can compute log-loss or hinge loss for this output
- This loss is backpropagated through the network
- The transformation in, e.g., logistic model $z_T \rightarrow p(y = 1 \,|\, \mathbf{x}) = \frac{1}{1+\exp(-z_T)}$ is not part of the network per se, only part of computation of the loss

Multi-class output

- Final layer consists of K units (number of classes)

$$z_{T,c} = \mathbf{w}_{T,c} \mathbf{z}_{T-1}$$

$$\mathbf{z}_T = \mathbf{W}_T \mathbf{z}_{T-1}$$

- The features (previous layer) are shared, but used with different weights
- Loss, e.g., softmax, computed on $\mathbf{z}_T$ and used in back-prop
- Normalization, exponentiation etc., not part of the feed-forward network, but part of back-prop

Multi-class with squared loss

- Encode C-class label by C-way binary vector $\mathbf{y}$
- Constraints: $\mathbf{y} \in \{0,1\}^K$, $\sum_c y_c = 1$
- Allow $\widehat{\mathbf{y}} \in [0,1]^K$, e.g., $\widehat{\mathbf{y}} = \sigma(\mathbf{z}_T)$
- Use square loss:

$$L(\mathbf{y}, \widehat{\mathbf{y}}) = \sum_{c=1}^{K} (\widehat{y}_c - y_c)^2$$

- Since exactly one class c is on in $\mathbf{y}$ this will drive $\widehat{y}_c$ up and all other $\widehat{y}_{c' \neq c}$ down

Multi-class vs multi-label

- What if we remove $\sum_c y_c = 1$?
- This is a *multi-label* scenario
- Example: object categories that correspond to an image; topics for a document; attributes for an image, e.g., {outdoors,night,has man made structures,raining}
- Loss on the top layer of the network:

$$L(\mathbf{y}, \widehat{\mathbf{y}}) = \sum_{c=1}^{K} (\widehat{y}_c - y_c)^2$$

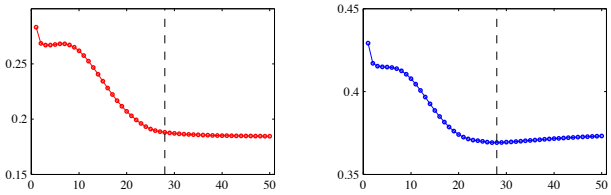
- Now we have K binary prediction tasks
- Note: we still share the features among them

Model complexity of MLP

- What determines model complexity for multilayer networks?
- Like with other models, number and norm of parameters is critical
- Follows from design of network *architecture*: depth/width of the network and complexity of units
- To prevent overfitting, can apply the usual tools: CV, regularization
- Traditionally, two form of regularization in neural networks
  - Weight decay:

$$\min_{\mathbf{w}} \left\{ \frac{1}{N} \sum_{i=1}^{N} L(y_i, \widehat{y}(\mathbf{x}_i; \mathbf{w})) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right\}$$

  - Early stopping

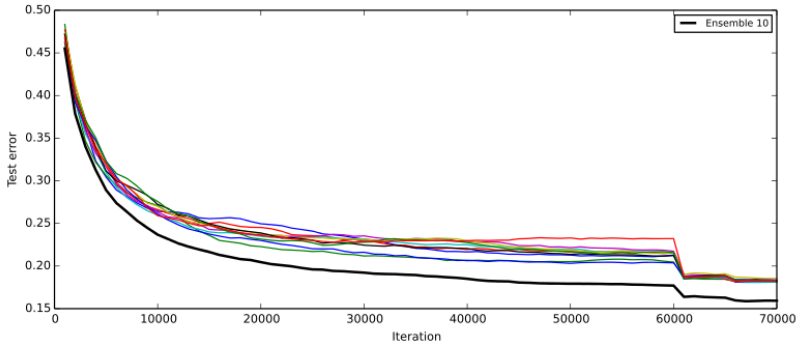


Learning rate

- Objective on example $(\mathbf{x}_t, y_t)$: $J(\mathbf{w}) = L(y_t, \widehat{y}(\mathbf{x}_t; \mathbf{w})) + R(\mathbf{w})$
- SGD update:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \Delta\mathbf{w}_t \qquad \Delta\mathbf{w}_t = -\eta_t \nabla_{\mathbf{w}} J$$

- Theoretically suggested schedule for η_t: $\eta_t = \eta_0/(1+\gamma t)$
- In practice, most common schedule: constant η_t, occasionally dropping when training loss plateus



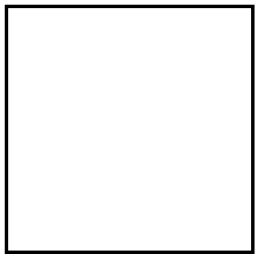
Momentum

- Vanilla SGD update at iter. t:

$$\Delta \mathbf{w}_t = -\eta_t \nabla_{\mathbf{w}} J$$

- Momentum:

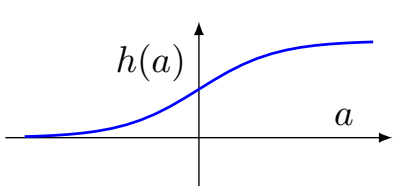
$$\Delta \mathbf{w}_t = \mu \Delta \mathbf{w}_{t-1} - \eta_t \nabla_{\mathbf{w}} J$$

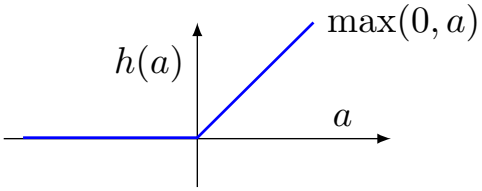


- Doesn't let the update vary too wildly
- Typical values of momentum are $\mu \approx 0.9$
- Second-order methods are rarely used on typically huge problems

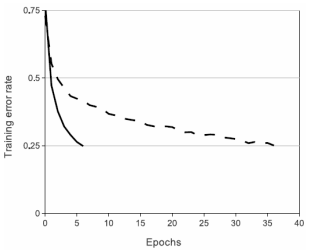
RELU

- Traditional activation function for a unit: sigmoid, e.g., $\tanh$ or logistic function





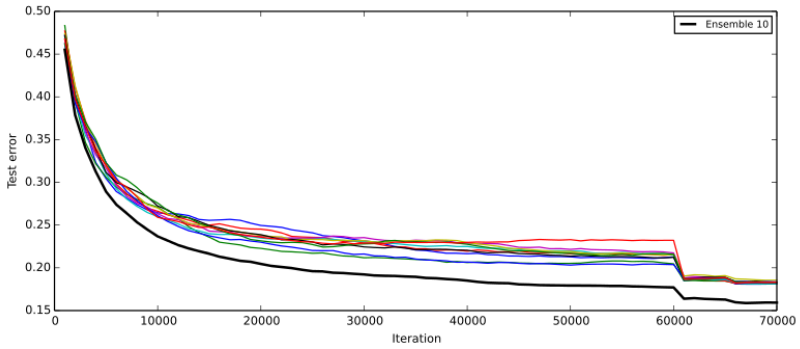
- Problem: vanishing gradient due to saturating nonlinearity
- **Re**ctified **l**inear **u**nit activation $\max(0, a)$
- Sometimes an order of magnitude speed-up



from Krizhevsky et al., 2012
solid: RELU
dashed: sigmoid

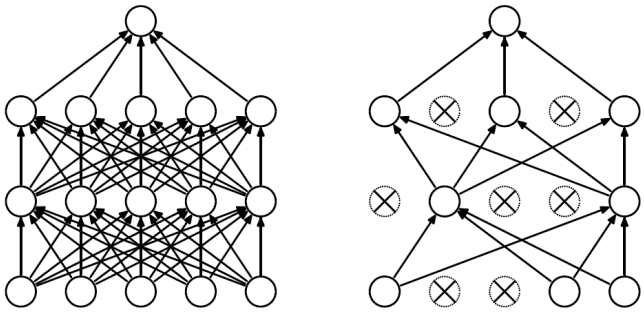
Ensembles of networks

- We may want to train multiple networks and somehow combine them
- Reduces variance (we have stochastic training of non-convex objective)
- Directly average the network weights? Terrible idea
- Averaging unit activations: equally bad
- Better idea: average the predictions
- Multiclass settings: output of network t is $\widehat{\mathbf{p}}_t = [\widehat{p}_t(y = 1), \ldots, \widehat{p}_t(K)]$, then use $\frac{1}{T} \sum_t \widehat{\mathbf{p}}_t$



Dropout

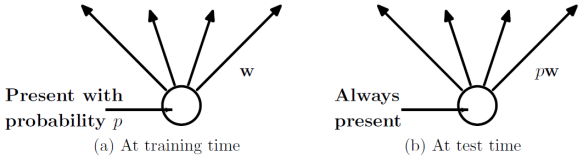
- Problem: fully connected units near the top of the network suffer from *co-adaptation*
- From Srivastava et al., 2014: "A motivation for dropout comes from a theory of the role of sex in evolution"
- Idea: during training, randomly remove a fraction of the units



(a) Standard Neural Net (b) After applying dropout.

Effect of dropout

- Training time: a unit has probability p to be dropped out
- Test time: multiply activation of unit by p



(a) At training time (b) At test time

- Intuition: sampling from a very large pool of network architectures
- Often very important in deep networks

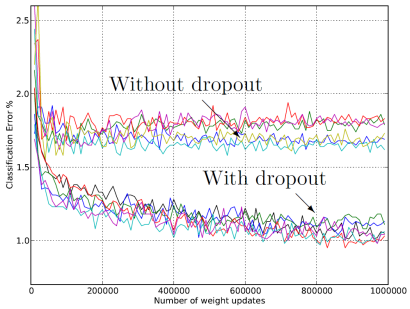
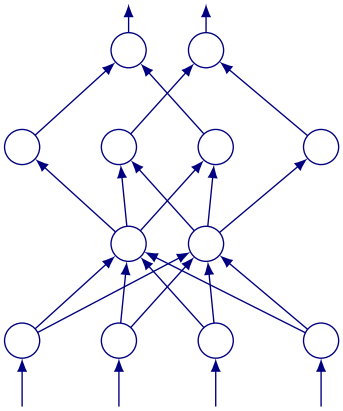


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Multi-layer network

- Each layer l computes activation vector $\mathbf{z}_l$
- Input layer: $\mathbf{z}_0 = \mathbf{x}$
- Feed-forward propagation: $\mathbf{z}_l = h(\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l)$

- This assumes connections between subsequent layers, but does not require full connections



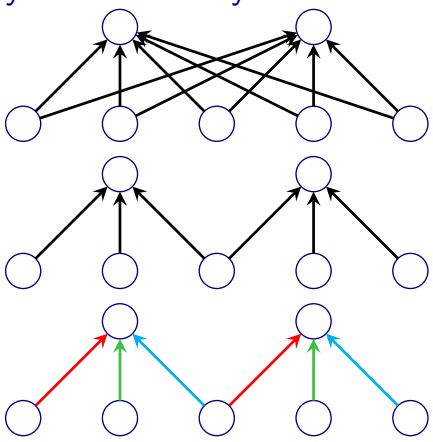
Network topology

- Usually should think in terms of *layer* topology
- Important types of layer connectivity:

fully connected

locally connected

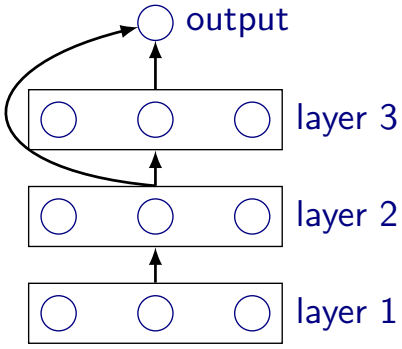
convolutional



- Receptive field of a unit: set of values in preceding layer that feed into it
- Feature map: set of values computed by units in a layer

Skip-layer connections

- Connections can skip layers
- Example: output depends directly on features from different levels in the hierarchy



- Backprop works as usual
- loss signal flows to L2 along two paths: through L3 and directly from output