Neural Networks: Learning (Backpropagation)

CS229: Machine Learning

Stanford University, Winter 2024
(Adapted from slides by Matgus Telgarsky and Alexander Schwing)

Goals of this lecture

- Understanding the forward and backward pass in deep networks
- Understand backpropagation in deep networks

Goals of this lecture

- Understanding the forward and backward pass in deep networks
- Understand backpropagation in deep networks

Reading material

- Course Notes, Section 7.3
- I. Goodfellow et al.; Deep Learning; Chapters 6-9

Lecture notation

$\phi(\cdot)$

Notation	Usage
$h(\cdot)$	Feature function; $h(\cdot)$ in the notes
$f(\cdot), F(\cdot)$	Prediction function; $h(\cdot)$ in the notes
$l(\cdot, \cdot)$	Loss function; $J(\cdot)$ in the notes
$\boldsymbol{w}, \boldsymbol{W}$	Model Parameters, θ in the notes
$\boldsymbol{x}^{(i)}, \boldsymbol{x}$	Input(s)
$y^{(i)}, y$	Label(s)
$\hat{y}$	Prediction; o in the notes
α_k	step size in decent methods
λ	Regularization parameter(s); C in the notes
$\sigma(\cdot)$	Activation function, nonlinearity

Recap: Our (regularized) learning framework:

$$\min_{\boldsymbol{w}} -\log P(y|x;\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \quad \equiv \quad \min_{\boldsymbol{w}} \ell\left(y, f(x)\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Recap: Our (regularized) learning framework:

$f(x) = w^T h(x)$

$$\min_{\boldsymbol{w}} - \log P(y|x;\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \quad \equiv \quad \min_{\boldsymbol{w}} \ell\left(y, f(x)\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

What are possible issues/limitations?

Recap: Our (regularized) learning framework:

$$\min_{\boldsymbol{w}} -\log P(y|x;\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \quad \equiv \quad \min_{\boldsymbol{w}} \ell\left(y, f(x)\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

What are possible issues/limitations?

Linearity in the feature space $h(x, y)$. Fix: use feature design. But still learning a model **linear** in the parameters $\boldsymbol{w}$

Recap: Our (regularized) learning framework:

$$\min_{\boldsymbol{w}} -\log P(y|x;\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \quad \equiv \quad \min_{\boldsymbol{w}} \ell\left(y, f(x)\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

What are possible issues/limitations?

Linearity in the feature space $h(x, y)$. Fix: use feature design. But still learning a model **linear** in the parameters $\boldsymbol{w}$

How to extend this?

Recap: Our (regularized) learning framework:

$$\min_{\boldsymbol{w}} -\log P(y|x;\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \quad \equiv \quad \min_{\boldsymbol{w}} \ell\left(y, f(x)\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

What are possible issues/limitations?

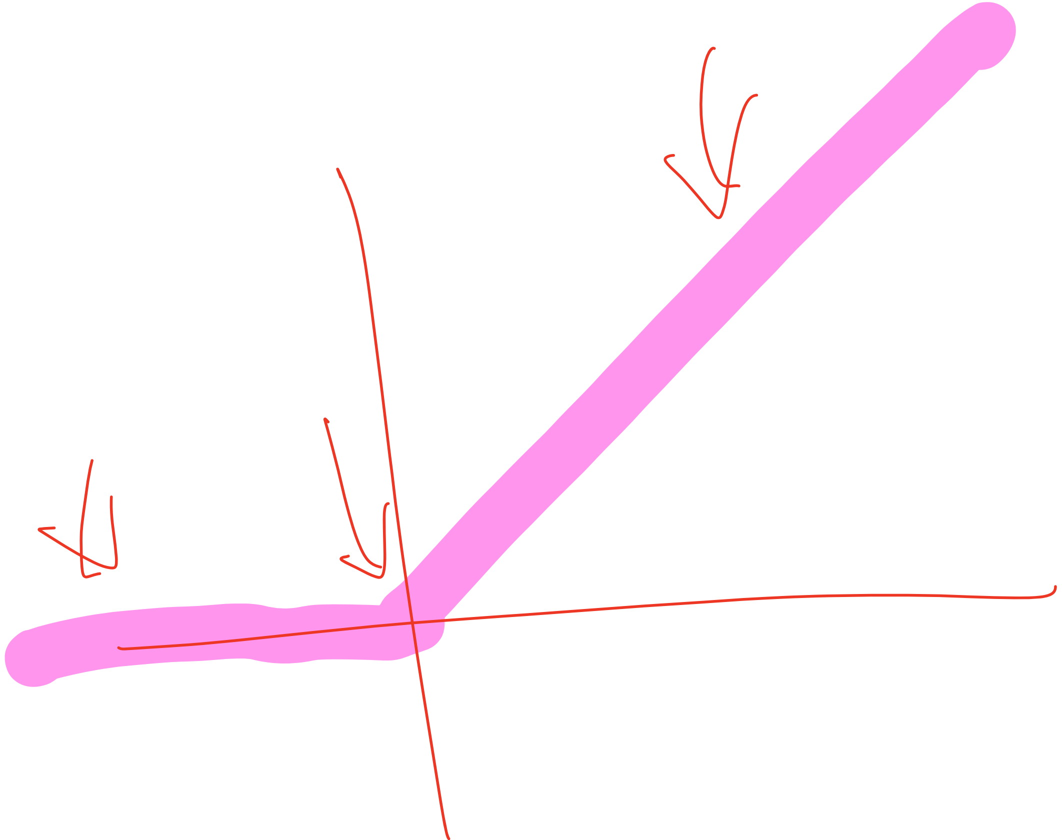
Linearity in the feature space $h(x, y)$. Fix: use feature design. But still learning a model **linear** in the parameters $\boldsymbol{w}$

How to extend this?

Replace $\boldsymbol{w}^T h(x)$ with a general function $f(x)$

Deep Learning:

What function $f(x)$ to choose?



Deep Learning:

sigmoid, ReLU, ...

$\sigma(w^T x)$

What function $f(x)$ to choose?

- Can choose any differentiable composite function

$$f(x; \boldsymbol{w}) = f_L(\boldsymbol{w}^L, f_{L-1}(\boldsymbol{w}^{L-1}, f_{L-2}(\ldots f_1(\boldsymbol{w}^1, x)\ldots)))$$

$\sigma(w^{L-1} f_{L-2})$

$$\sigma\left(\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix}\right) = \begin{bmatrix} \sigma(v_1) \\ \sigma(v_2) \\ \vdots \\ \sigma(v_m) \end{bmatrix}$$

Deep Learning:

What function $f(x)$ to choose?

- Can choose any differentiable composite function

$$f(x;\boldsymbol{w}) = f_L(\boldsymbol{w}^L, f_{L-1}(\boldsymbol{w}^{L-1}, f_{L-2}(\ldots f_1(\boldsymbol{w}^1, x)\ldots)))$$

- For deep learning, we choose the interleaved composition of linear functions and non-linear activations

$$f(x) = \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1\left(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]}\right)\cdots\right)$$

Optimization.

Regularized learning now takes the form

$$\min_{\boldsymbol{W}^{[1]},\ldots,\boldsymbol{W}^{[L]},\boldsymbol{b}^{[1]},\ldots,\boldsymbol{b}^{[L]}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1\left(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]}\right)\cdots\right)\right) + \lambda\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_2^2$$

Optimization.

Regularized learning now takes the form

$$\min_{\boldsymbol{W}^{[1]},\ldots,\boldsymbol{W}^{[L]},\boldsymbol{b}^{[1]},\ldots,\boldsymbol{b}^{[L]}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1\left(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]}\right)\cdots\right)\right) + \lambda\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_2^2$$

In general, resulting optimization is "harder" than linear regression

Optimization.

Regularized learning now takes the form

$$\min_{\boldsymbol{W}^{[1]},\ldots,\boldsymbol{W}^{[L]},\boldsymbol{b}^{[1]},\ldots,\boldsymbol{b}^{[L]}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1\left(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]}\right)\cdots\right)\right) + \lambda\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_2^2$$

In general, resulting optimization is "harder" than linear regression

Implications:

- Gradient-based optimization approaches is no longer guaranteed to find the global optimum
- Initialization of parameters matters

Optimization.

Regularized learning now takes the form

$$\min_{\boldsymbol{W}^{[1]},\ldots,\boldsymbol{W}^{[L]},\boldsymbol{b}^{[1]},\ldots,\boldsymbol{b}^{[L]}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1\left(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]}\right)\cdots\right)\right) + \lambda\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_2^2$$

In general, resulting optimization is "harder" than linear regression

Implications:

- Gradient-based optimization approaches is no longer guaranteed to find the global optimum
- Initialization of parameters matters
- Stochastic gradient descent works well in practice

Notation for Multiclass classification

Recall: for multiclass classification with $y \in [1, \ldots K]$, the prediction function

$$f(x) \in ?$$

Notation for Multiclass classification

Recall: for multiclass classification with $y \in [1, \dots K]$, the prediction function

$$f(x) \in \mathbb{R}^K,$$

$$\approx \begin{bmatrix} p(y=1|x) \\ \vdots \\ p(y=K|x) \end{bmatrix}$$

i.e., the prediction is a vector function.

Notation for Multiclass classification

Recall: for multiclass classification with $y \in [1, \ldots K]$, the prediction function

$$f(x) \in \mathbb{R}^K,$$

i.e., the prediction is a vector function.

Convenient notation: replace $F(\boldsymbol{w}, y, x) = [f(x)]_y$, i.e., the y^{th} index of $f(x)$

Notation for Multiclass classification

Recall: for multiclass classification with $y \in [1, \ldots K]$, the prediction function

$$f(x) \in \mathbb{R}^K,$$

i.e., the prediction is a vector function.

Convenient notation: replace $F(\boldsymbol{w}, y, x) = [f(x)]_y$, i.e., the y^{th} index of $f(x)$

$f_c(w_c, x)$

Implemented as:

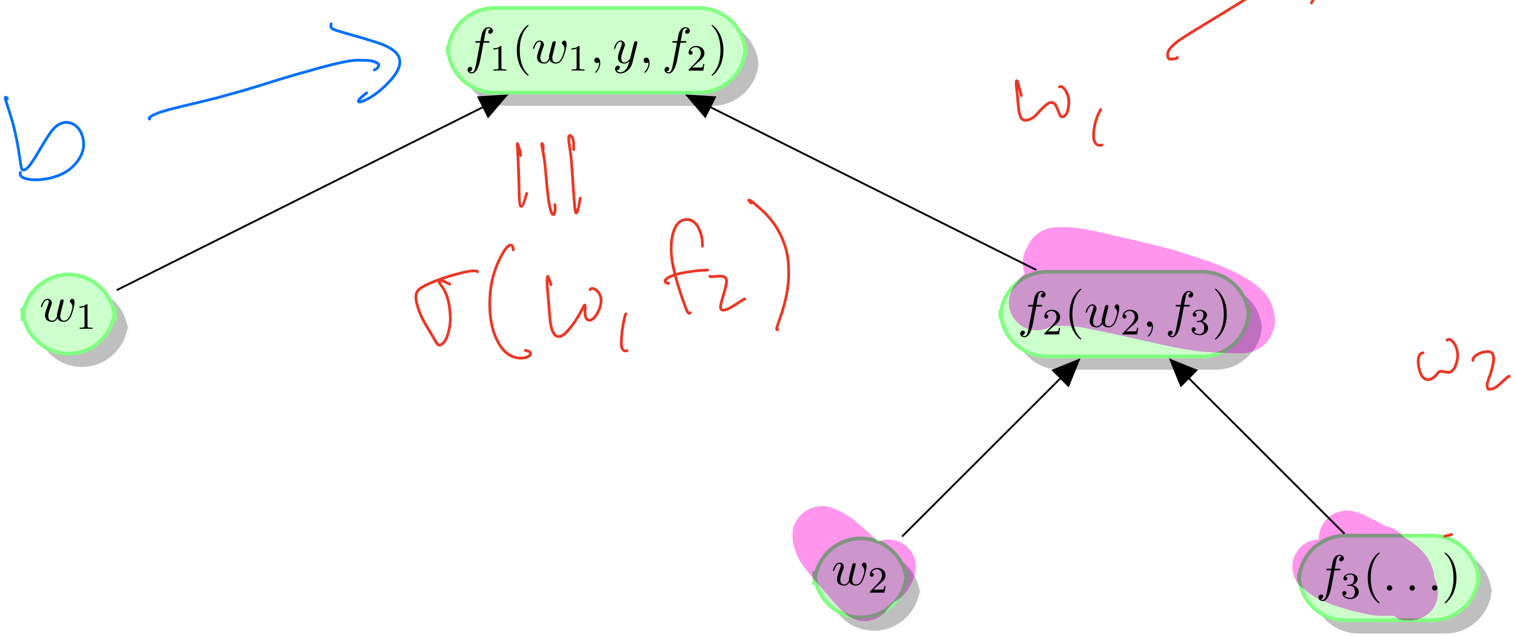
$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

$$\left[f_1(w_1, f_2(w_2, \ldots))\right]_y$$

Computation Graph Representation:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

Nodes are weights, data, and functions:



b

$\equiv \sigma(w_1, f_2)$

$f_1(w_1, y, f_2)$

w_1

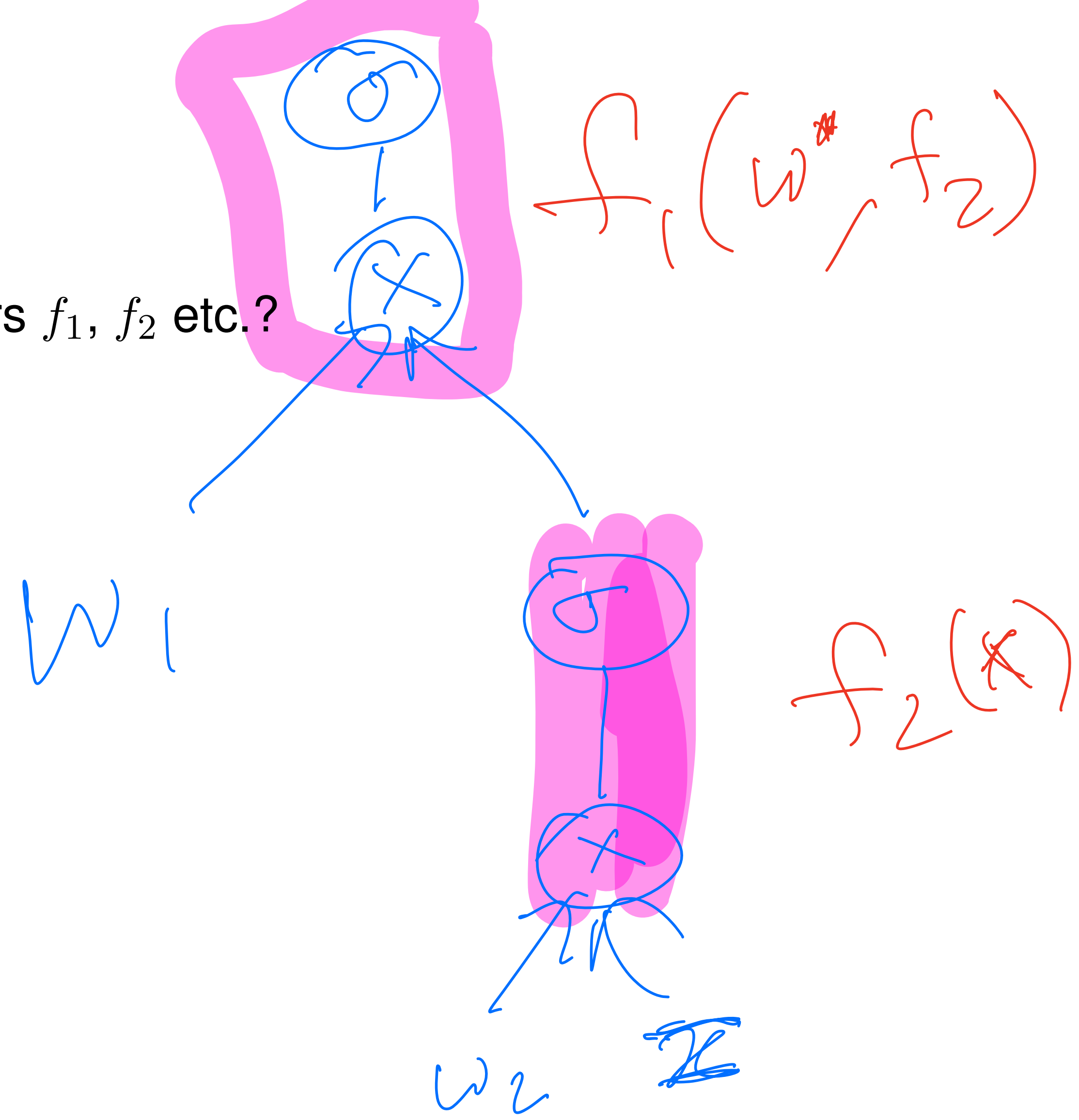
$f_2(w_2, x)$

w_2 x

$f_2(w, x)$

Internal representation used by deep net packages.

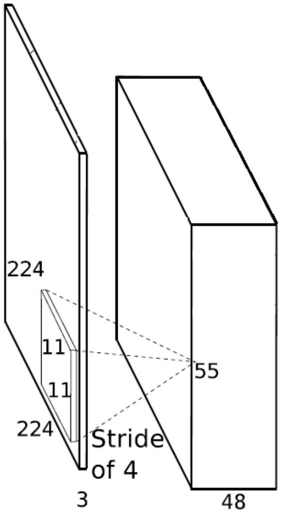
What are the individual functions/layers f_1, f_2 etc.?



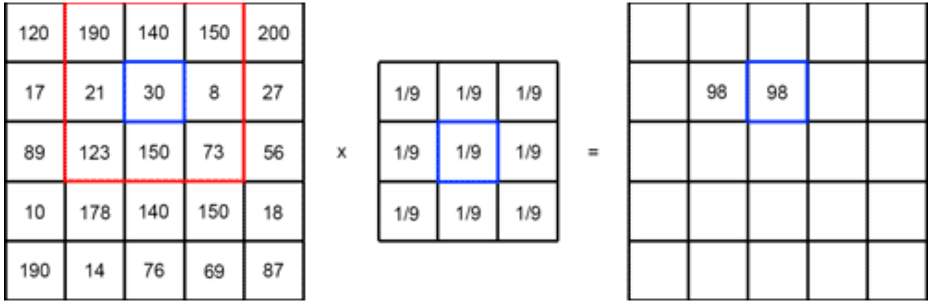
What are the individual functions/layers f_1, f_2 etc.?

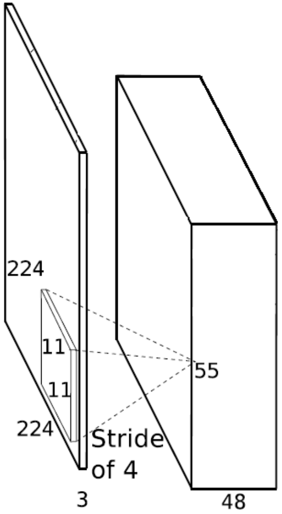
- Fully connected layer(s)
- Convolutions
- Univariate activations e.g. rectified linear units (ReLU): $\max\{0, x\}$
- Maximum-/Average pooling (sometimes called subsampling)
- Soft-max layer

Recall: Convolutions

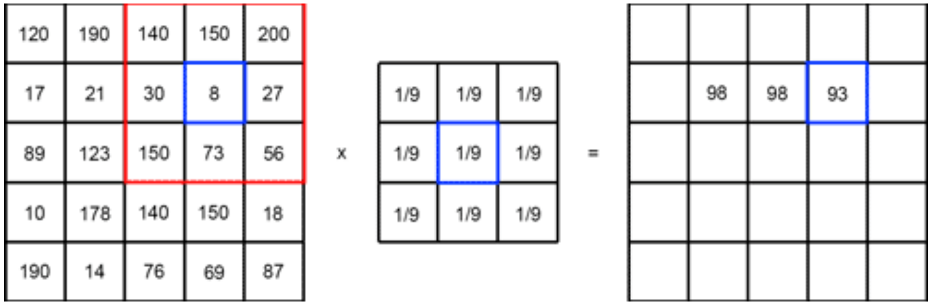


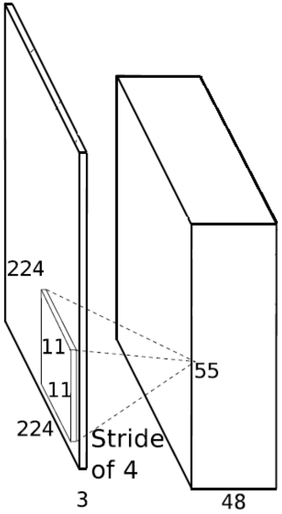
Recall: Convolutions





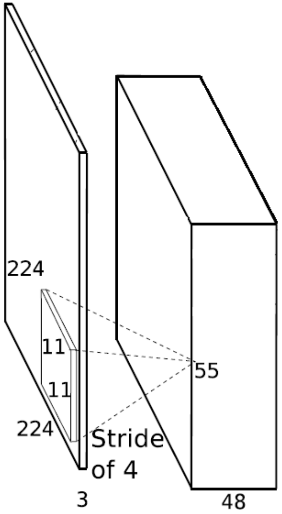
Recall: Convolutions



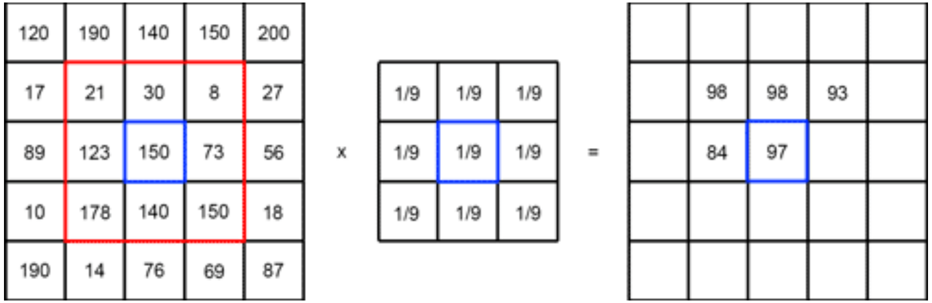


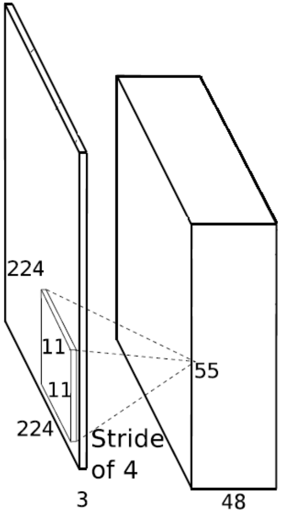
Recall: Convolutions

$$\begin{bmatrix} 120 & 190 & 140 & 150 & 200 \\ 17 & 21 & 30 & 8 & 27 \\ 89 & \boxed{123} & 150 & 73 & 56 \\ 10 & 178 & 140 & 150 & 18 \\ 190 & 14 & 76 & 69 & 87 \end{bmatrix} \times \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \boxed{1/9} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} & & & & \\ & 98 & 98 & 93 & \\ & \boxed{84} & & & \\ & & & & \\ & & & & \end{bmatrix}$$

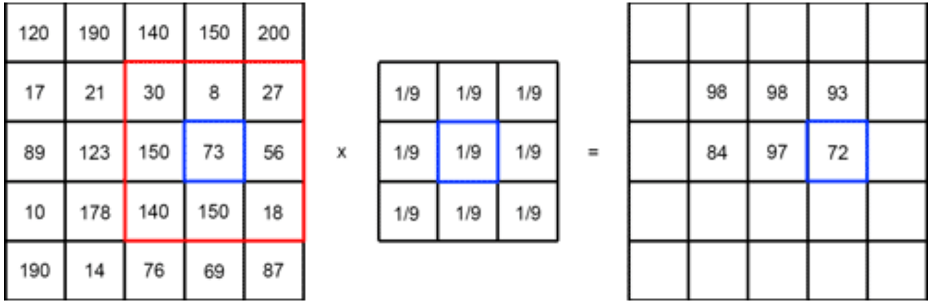


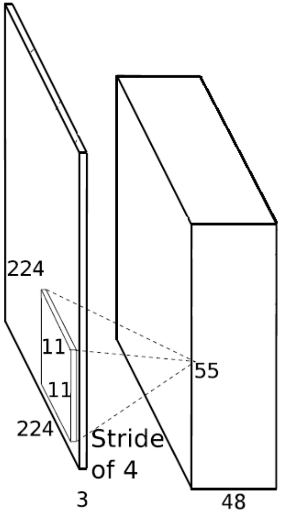
Recall: Convolutions



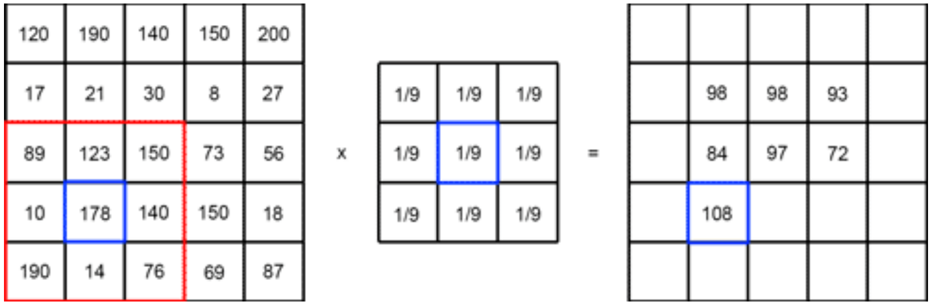


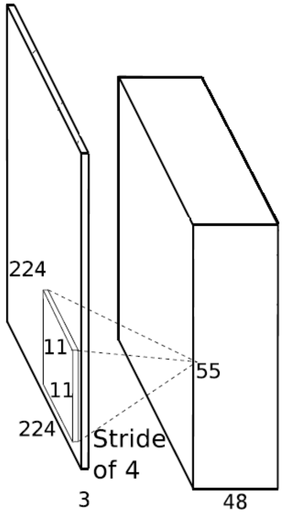
Recall: Convolutions





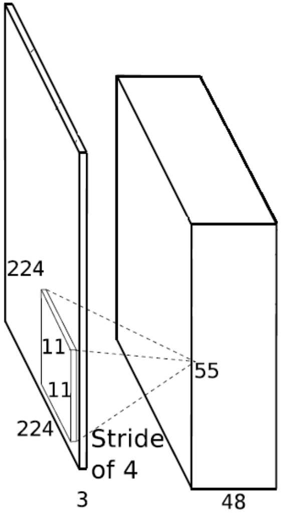
Recall: Convolutions





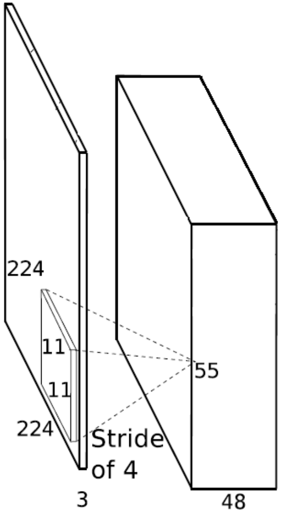
Recall: Convolutions

$$\begin{bmatrix} 120 & 190 & 140 & 150 & 200 \\ 17 & 21 & 30 & 8 & 27 \\ 89 & 123 & 150 & 73 & 56 \\ 10 & 178 & 140 & 150 & 18 \\ 190 & 14 & 76 & 69 & 87 \end{bmatrix} \times \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} & & & & \\ & 98 & 98 & 93 & \\ & 84 & 97 & 72 & \\ & 108 & 108 & & \\ & & & & \end{bmatrix}$$



Recall: Convolutions

$$
\begin{bmatrix} 120 & 190 & 140 & 150 & 200 \\ 17 & 21 & 30 & 8 & 27 \\ 89 & 123 & 150 & 73 & 56 \\ 10 & 178 & 140 & 150 & 18 \\ 190 & 14 & 76 & 69 & 87 \end{bmatrix}
\times
\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}
=
\begin{bmatrix} & & & & \\ & 98 & 98 & 93 & \\ & 84 & 97 & 72 & \\ & 108 & 108 & 91 & \\ & & & & \end{bmatrix}
$$



Recall: Convolutions

120	190	140	150	200
17	21	30	8	27
89	123	150	73	56
10	178	140	150	18
190	14	76	69	87

x

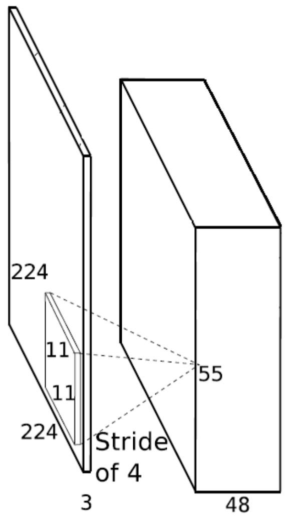
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

	98	98	93	
	84	97	72	
	108	108	91	

Trainable parameters w:

- Filters
- Bias
- Hyper-parameters (selected): width, height, depth, number, stride, padding



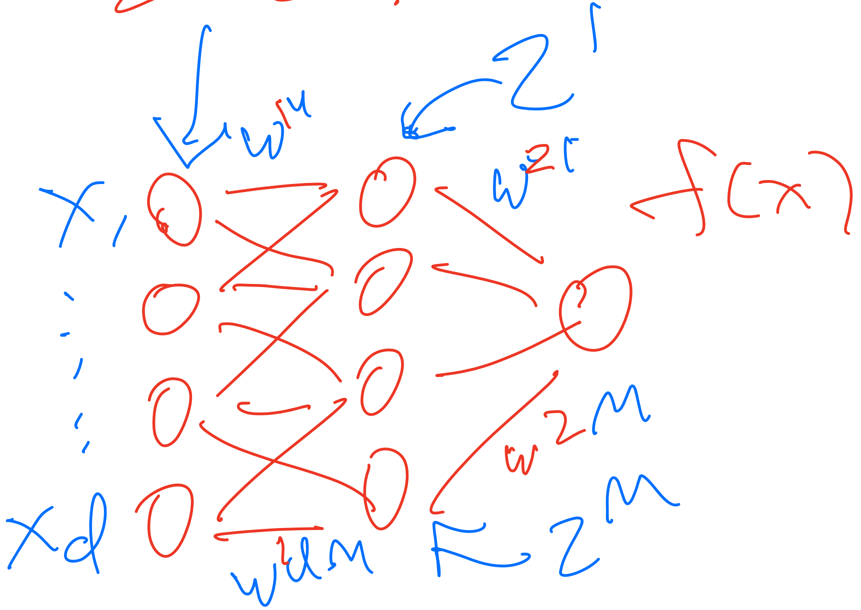
Rosenblatt, 1958

Example function architecture: Multilayer Perceptron (MLP)
The default, i.e., (multiple) fully connected layers
(often called MLP when used at the output).

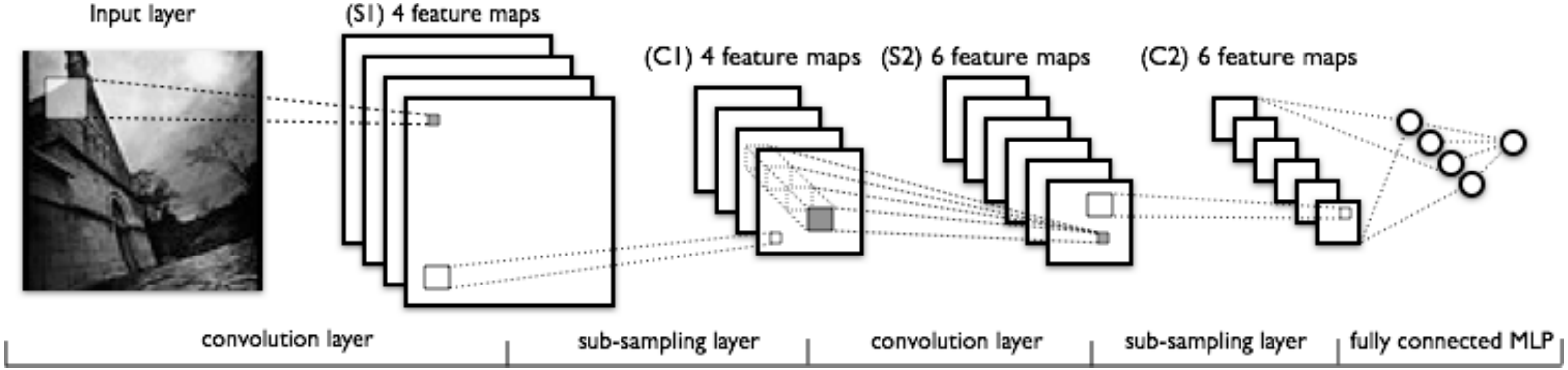
$$f(x) = \sigma\left(W^2 \overbrace{\sigma(W^1 x)}\right)$$

$x \in \mathbb{R}^d$
$z \in \mathbb{R}^m$
$f(x) \in \mathbb{R}$

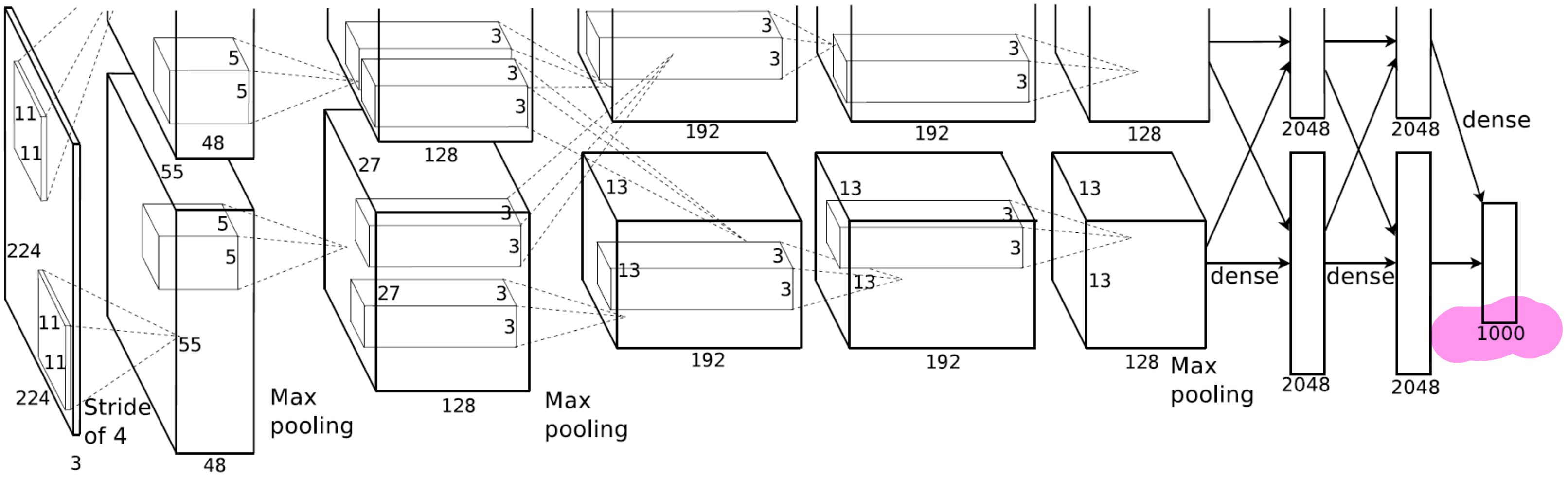
2-Layer



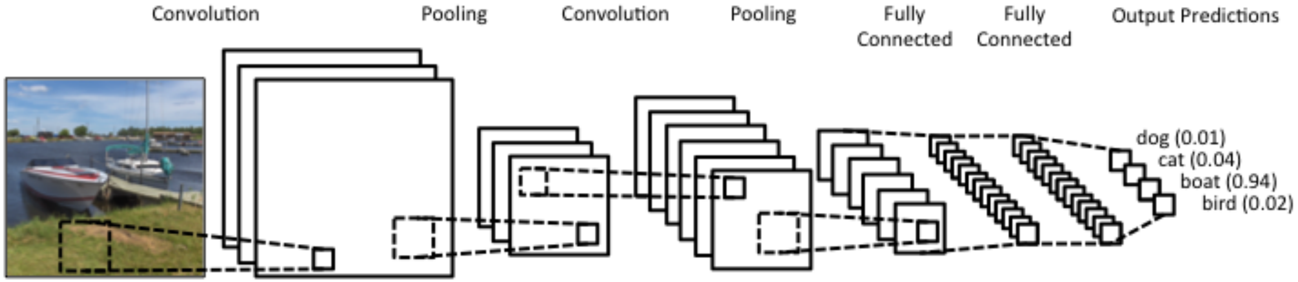
Example function architecture: LeNet



Example function architecture: AlexNet



Another deep net:



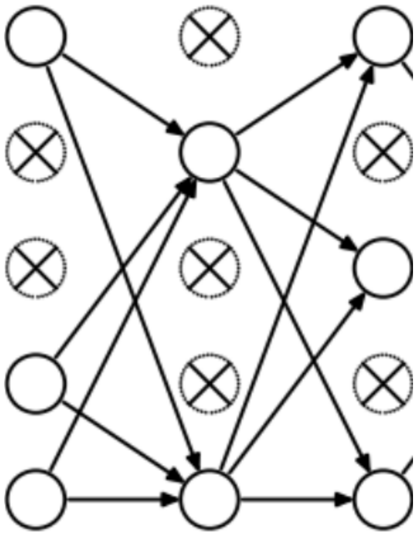
Note: These nets are structurally simple in that a layer's output is used as input for the next layer. This is not required.

Dropout:

Randomly set activations to zero

Trainable parameters w:

- None



Deep net training for multiclass classification (multiclass logistic loss):

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Deep net training for multiclass classification (multiclass logistic loss):

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

input
true
0/1

Equivalently, as regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \quad \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = \quad \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto \quad \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

Deep net training for multiclass classification (multiclass logistic loss):

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Equivalently, as regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

What is λ?

Deep net training for multiclass classification (multiclass logistic loss):

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Equivalently, as regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

What is λ? Weight decay (aka regularization constant)

Deep net training for multiclass classification (multiclass logistic loss):

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Equivalently, as regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = \quad \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto \quad \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

What is λ? Weight decay (aka regularization constant)

$$\min_{\boldsymbol{w}} \underbrace{\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2}_{\text{weight decay}} \underbrace{- \sum_{i\in[N]} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)})}_{\ell(y,F)}$$

Optimization Problem:

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

How to optimize this?

Optimization Problem:

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

How to optimize this?

Stochastic gradient descent: What was this again?

Use the Chain Rule! Gradient of

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is:

$$\frac{\cancel{2}\lambda w}{\cancel{2}} + \sum_i \left[\left(\frac{e^{F(\;)}}{\sum_{\hat{y}} e^{F(\;)}} \right) - \delta(\;) \right] \frac{\partial F(\;)}{\partial w}$$

Use the Chain Rule! Gradient of

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is:

$$\lambda \boldsymbol{w} + \sum_{i\in[N]} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

$$= \frac{e^F}{\sum e^F}$$

Use the Chain Rule! Gradient of

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is:

$$\lambda \boldsymbol{w} + \sum_{i\in[N]} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

- $p(\hat{y}|x) = \frac{\exp F(\boldsymbol{w},x,\hat{y})}{\sum_{\tilde{y}} \exp F(\boldsymbol{w},x,\tilde{y})}$ via soft-max which takes F as input

Use the Chain Rule! Gradient of

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in[N]} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is:

$$\lambda \boldsymbol{w} + \sum_{i\in[N]} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

- $p(\hat{y}|x) = \frac{\exp F(\boldsymbol{w},x,\hat{y})}{\sum_{\tilde{y}} \exp F(\boldsymbol{w},x,\tilde{y})}$ via soft-max which takes F as input
- $\frac{\partial F(\boldsymbol{w},x,\hat{y})}{\partial \boldsymbol{w}}$ via backpropagation

Backpropagation by example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, \overbrace{f_2(w_2, \underbrace{f_3(w_3, x)}_{z_2})}^{z_1}) \text{ with activations } \begin{cases} z_2 &= f_3(w_3, x) \\ z_1 &= f_2(w_2, z_2) \end{cases}$$

$$f_1(w_1, y, z_1)$$

Backpropagation by example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} z_2 &= f_3(w_3, x) \\ z_1 &= f_2(w_2, z_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

Backpropagation by example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x)))$$ with activations $$\begin{cases} z_2 & = & f_3(w_3, x) \\ z_1 & = & f_2(w_2, z_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3} =$$

Backpropagation by example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} z_2 &= f_3(w_3, x) \\ z_1 &= f_2(w_2, z_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

Backpropagation by example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} z_2 &= f_3(w_3, x) \\ z_1 &= f_2(w_2, z_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

Backpropagation by example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \quad \text{with activations} \quad \begin{cases} z_2 &= f_3(w_3, x) \\ z_1 &= f_2(w_2, z_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_2} =$$

Backpropagation by example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} z_2 &= f_3(w_3, x) \\ z_1 &= f_2(w_2, z_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_2} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial w_2}$$

$$\frac{\partial F}{\partial w_1}$$

Backpropagation by example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} z_2 &= f_3(w_3, x) \\ z_1 &= f_2(w_2, z_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

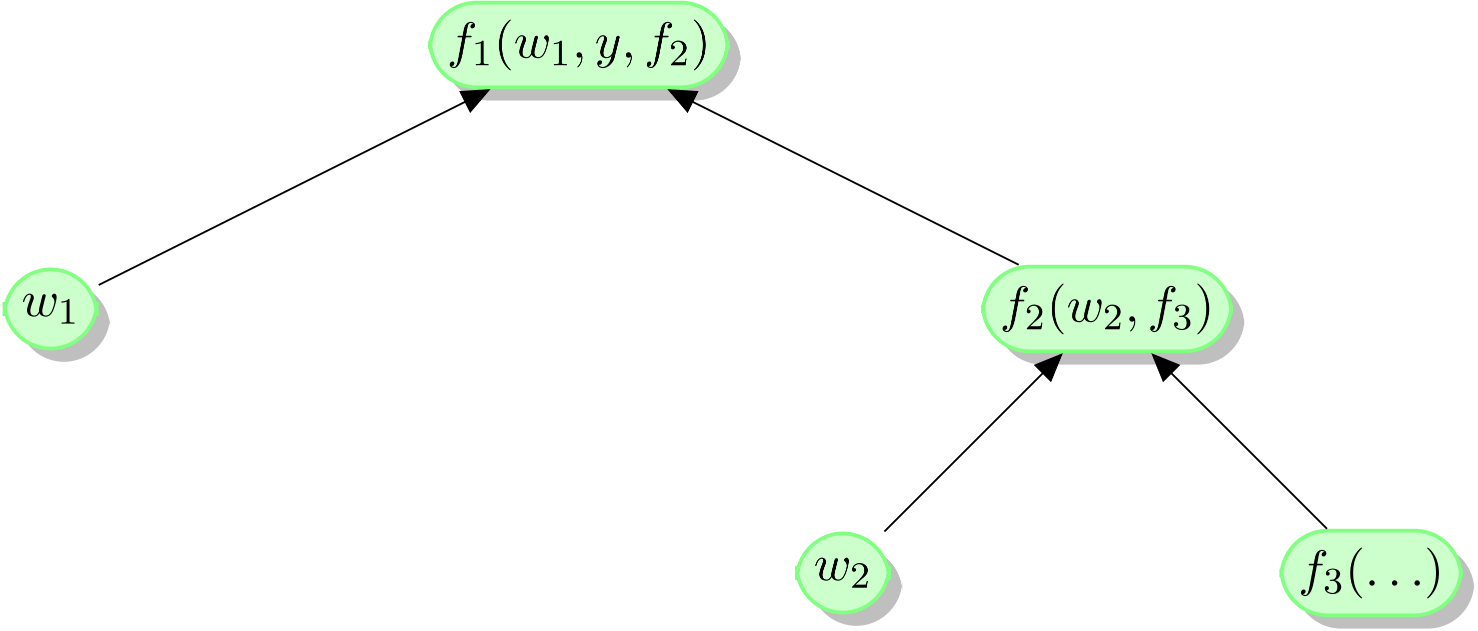
What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_2} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial w_2}$$

Generally: To avoid repeated computation, backpropagation on an acyclic graph.
Nodes in this graph are weights, data, and functions.

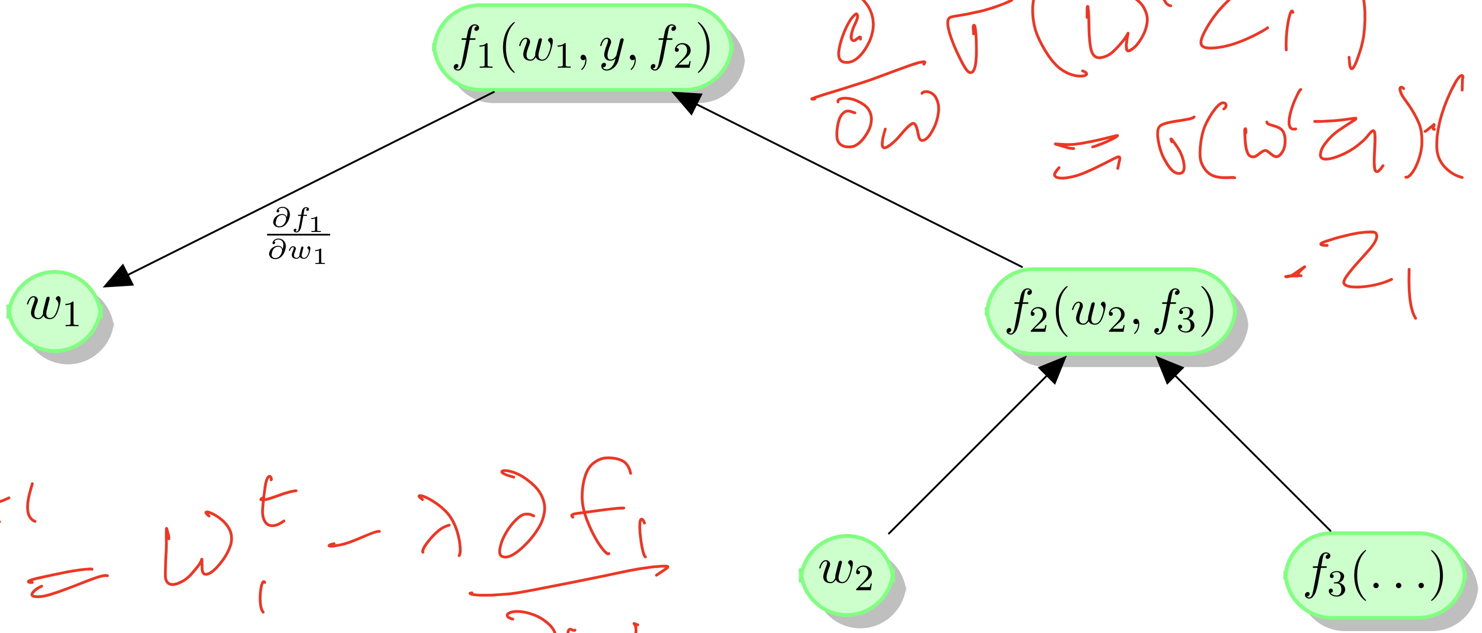
Composite function represented as acyclic graph: Forward Pass

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



Composite function represented as acyclic graph: Backward Pass

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



$$\frac{\partial}{\partial w}\sigma(w' z_1)$$

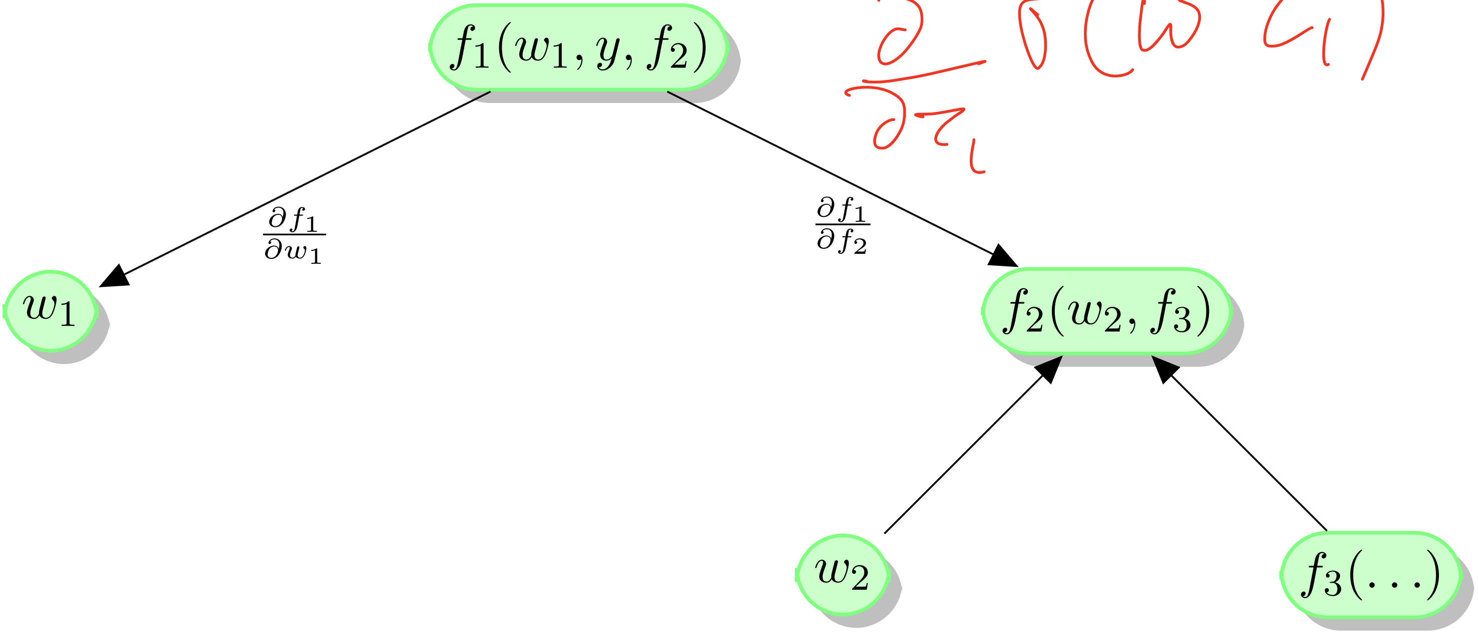
$$= \sigma(w' z_1)(1 - \sigma(w' z_1)) \cdot z_1$$

$$w_1^{t+1} = w_1^{t} - \lambda \frac{\partial f_1}{\partial w_1}$$

Composite function represented as acyclic graph: **Backward Pass**

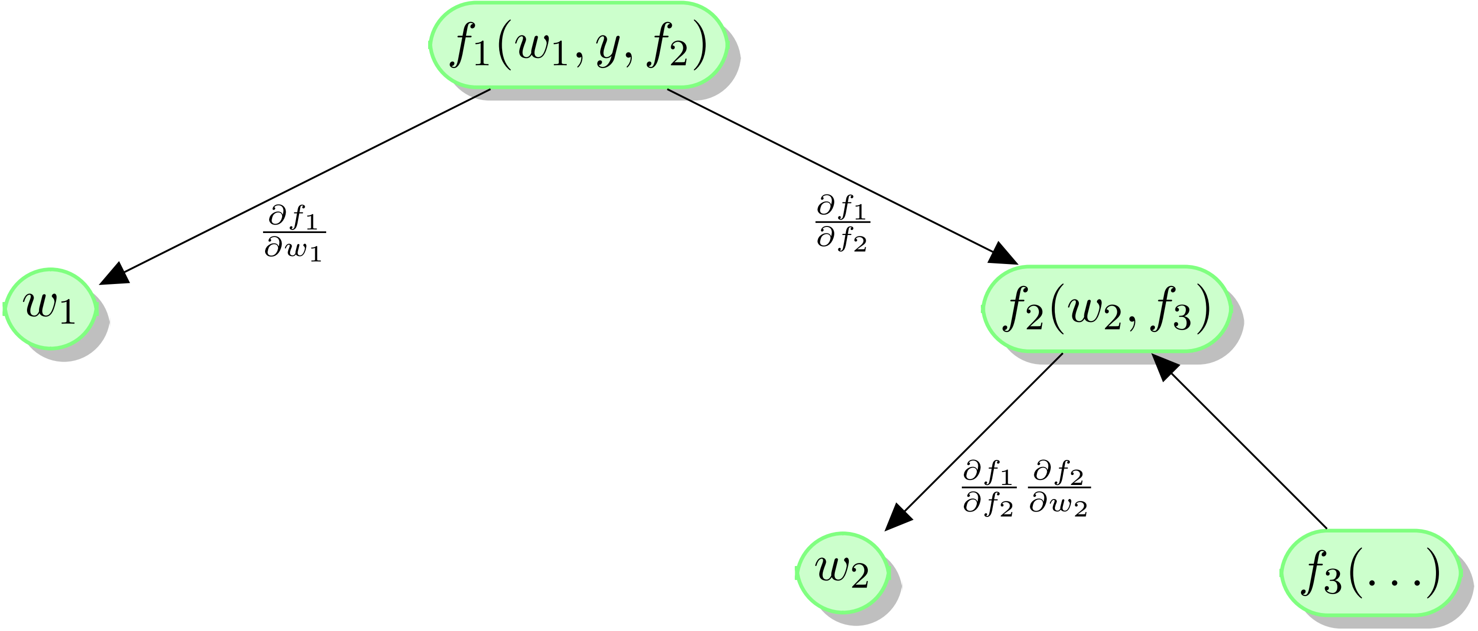
$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

$$\frac{\partial}{\partial z_1} \sigma(w' z_1)$$



Composite function represented as acyclic graph: Backward Pass

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

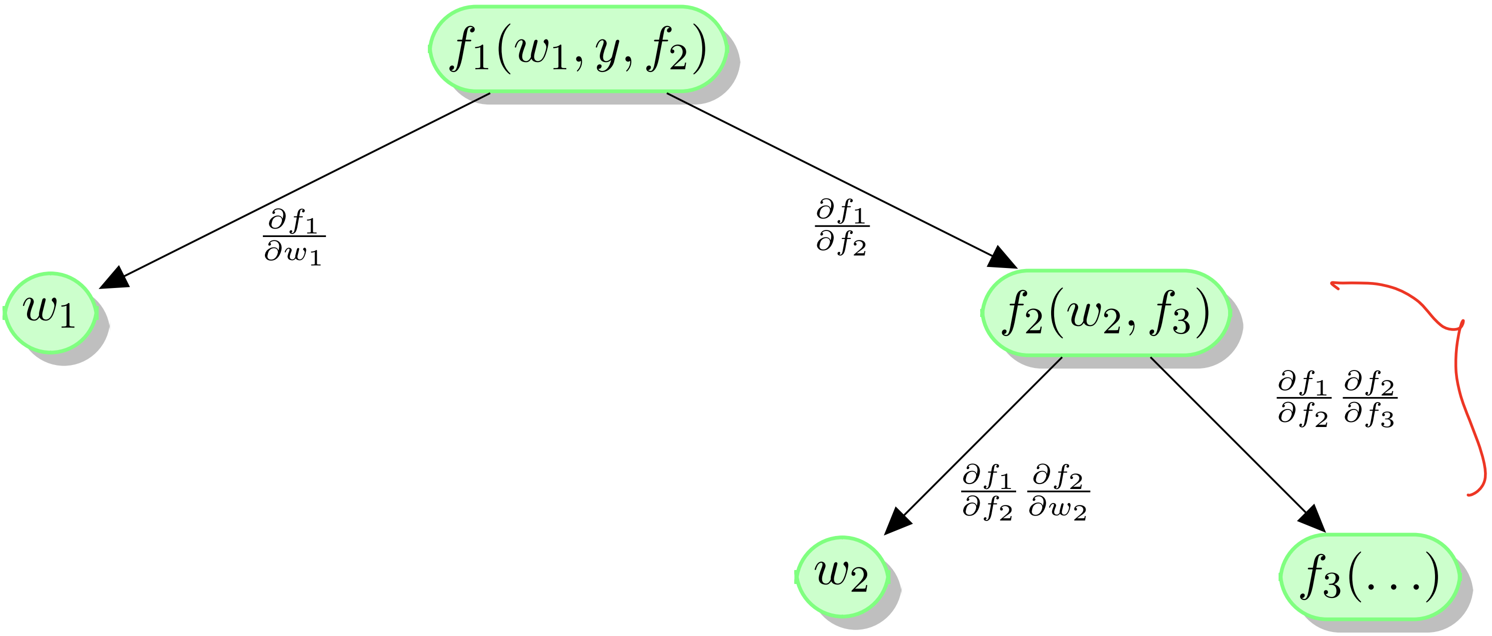


$\frac{\partial F}{\partial w_2}$

Composite function represented as acyclic graph:

Backward Pass

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



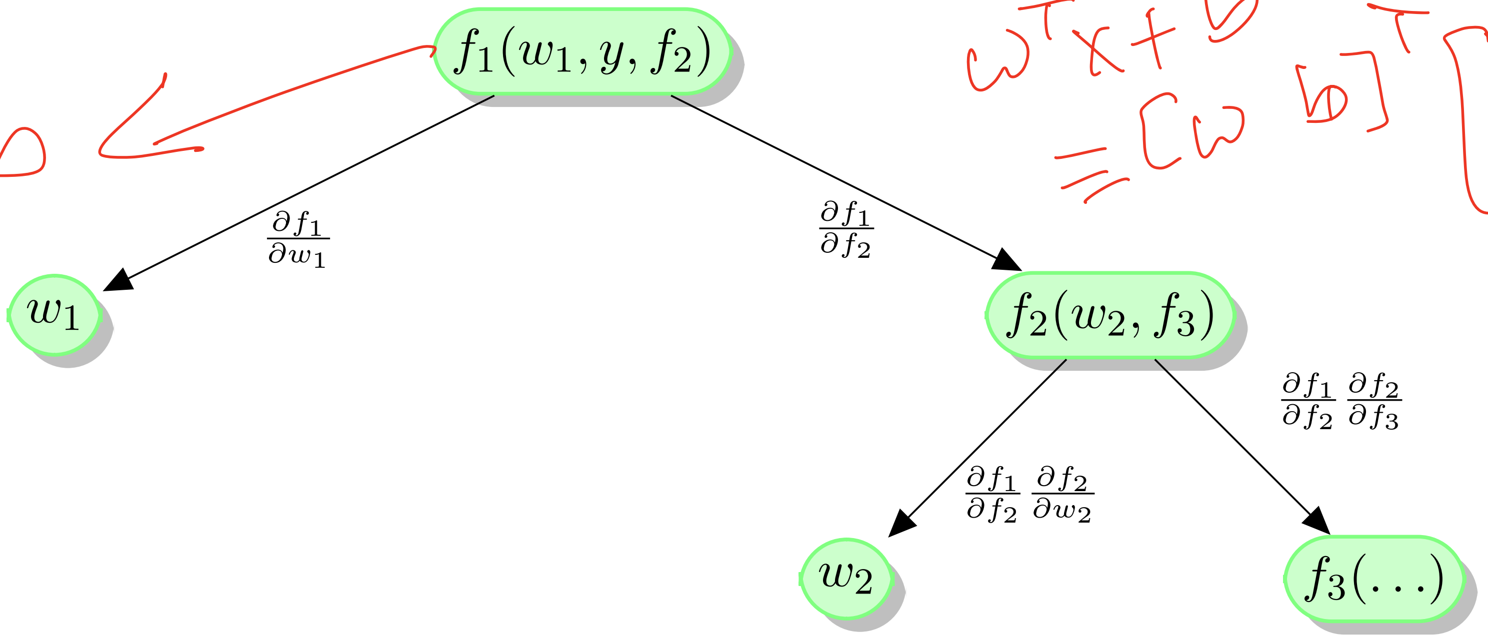
Composite function represented as acyclic graph: Backward Pass

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\dots)))$$

$w^T x + b$

$= [w \;\; b]^T \begin{bmatrix} x \\ 1 \end{bmatrix}$

b



Repeated use of chain rule for efficient computation of all gradients

Remark:

Can think of backpropagation as an efficient implementation of the chain rule on a computational graph.

Initialization matters:

Initialization matters:

- Not well understood in general

Initialization matters:

- Not well understood in general
- Heuristic: Random uniform

$$\text{Uniform}\left(-\frac{1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}}\right)$$

Initialization matters:

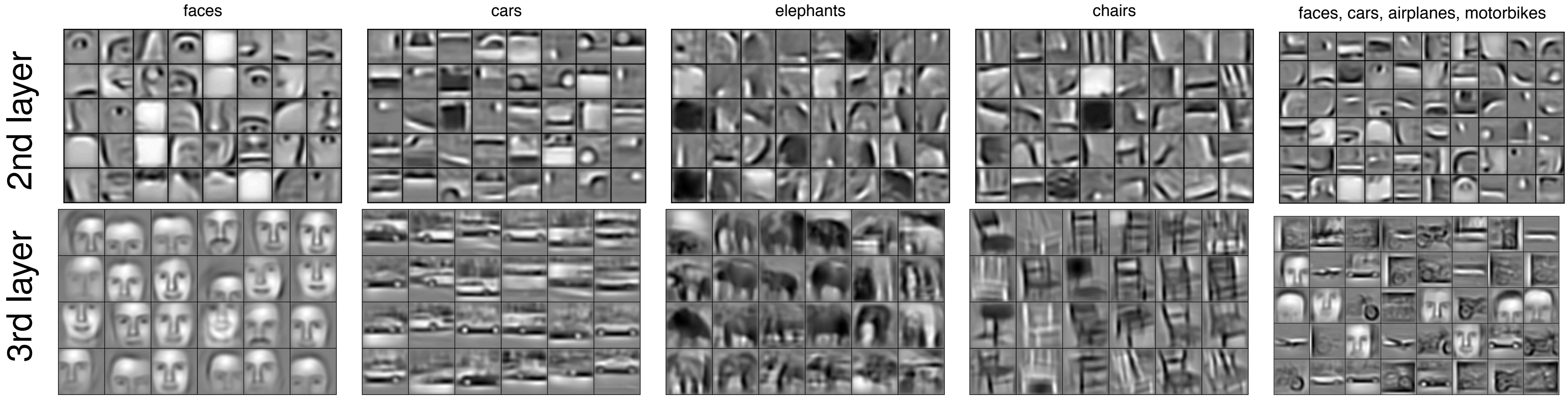
- Not well understood in general
- Heuristic: Random uniform

$$\text{Uniform}\left(-\frac{1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}}\right)$$

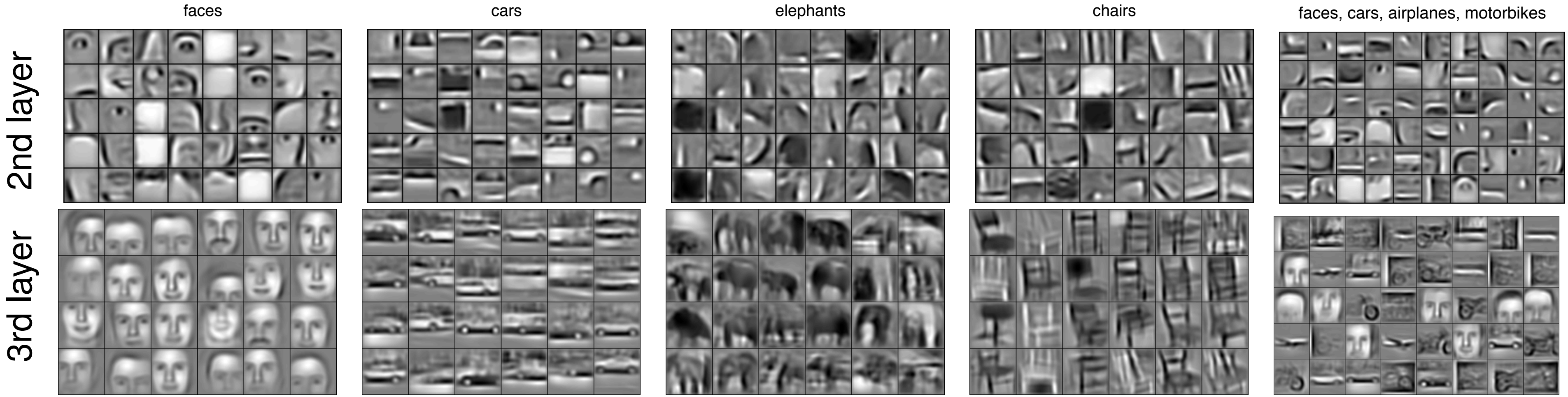
- Heuristic: Glorot and Bengio (2010)

$$\text{Uniform}\left(-\sqrt{\frac{6}{\text{fan in} + \text{fan out}}}, \sqrt{\frac{6}{\text{fan in} + \text{fan out}}}\right)$$

Example of feature transformations learned by deep networks



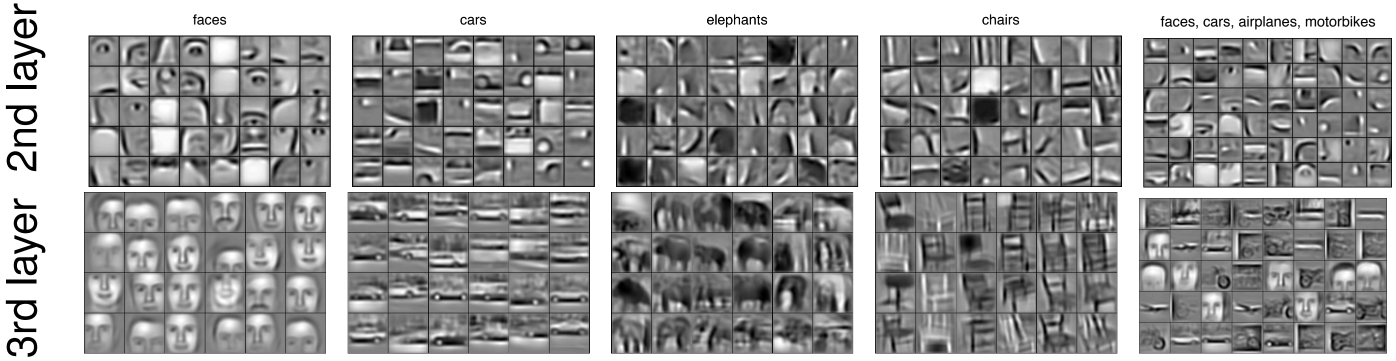
Example of feature transformations learned by deep networks



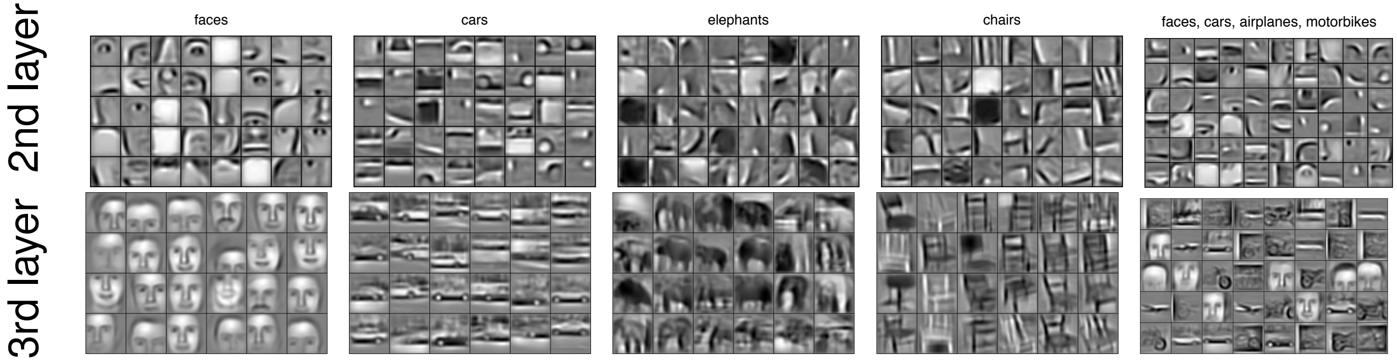
- **Remark:** A deep net with a single fully connected layer is a linear model

- Advantages of deep nets compared to using hand-crafted features:

- Advantages of deep nets compared to using hand-crafted features:
  - automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output

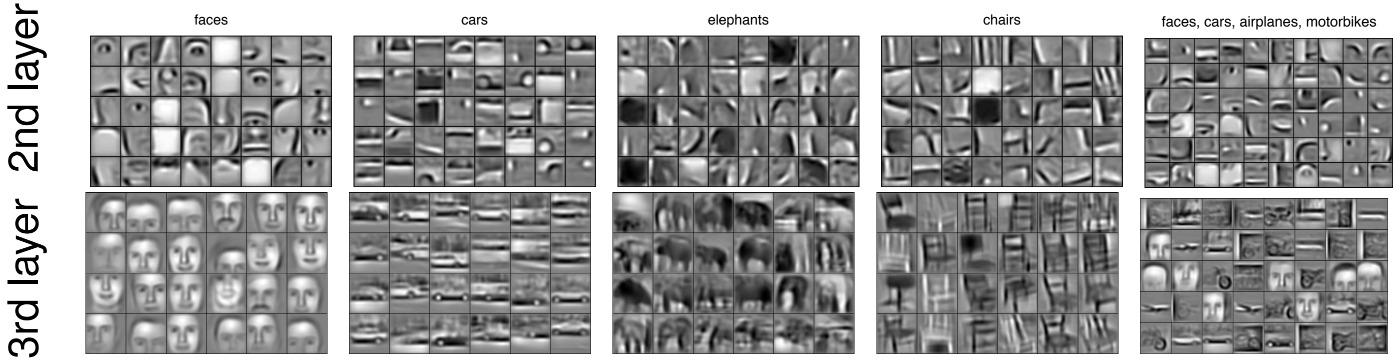


- Advantages of deep nets compared to using hand-crafted features:
  - automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output



- Disadvantage of deep nets compared to using hand-crafted features:

- Advantages of deep nets compared to using hand-crafted features:
  - automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output



- Disadvantage of deep nets compared to using hand-crafted features:
  - computationally demanding (routinely use GPUs)
  - require significant amounts of training data to perform well

Why this recent popularity:

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Algorithmic advances
- Sufficient evidence that it works

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Algorithmic advances
- Sufficient evidence that it works

This combination has led to significant performance improvements on many datasets (e.g., in language modeling, computer vision)

Algorithmic advances:

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation i.e., RELU instead of sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation i.e., RELU instead of sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation i.e., RELU instead of sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima

Algorithmic advances:

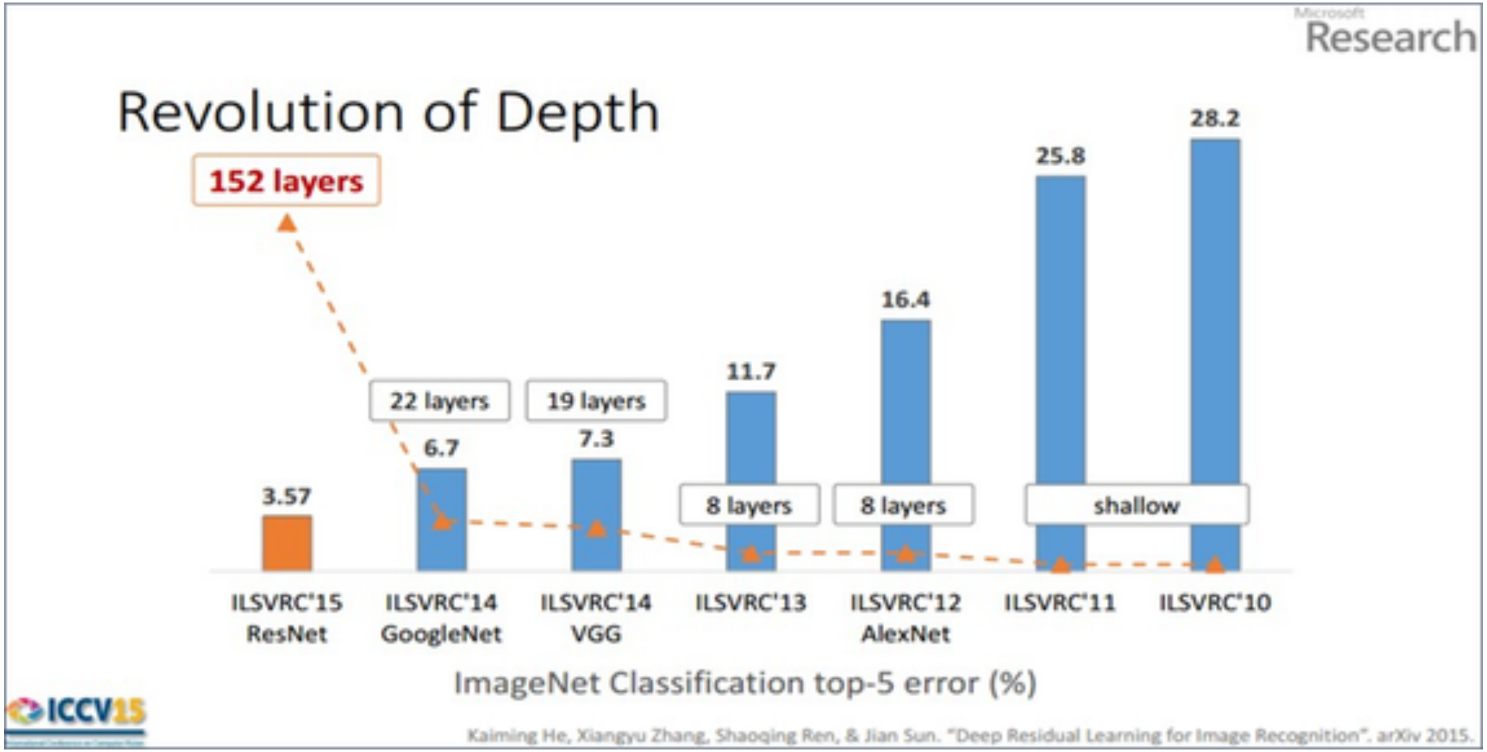
- Rectified linear unit ($\max\{0, x\}$) activation i.e., RELU instead of sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training
  - Normalizes data when training really deep nets
  - Normalize by subtracting mean and dividing by standard deviation

Imagenet Challenge:

Imagenet Challenge:

- A large dataset: 1.2M images, 1000 categories
- AlexNet was run on the GPU, i.e., sufficient computational resources
- Rectified linear units rather than sigmoid units simplify optimization

Results:


Microsoft
Research
Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15
ResNet
ILSVRC'14
GoogleNet
ILSVRC'14
VGG
ILSVRC'13
ILSVRC'12
AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Choices in deep learning packages:

Choices in deep learning packages:

- Use an appropriate loss function

Choices in deep learning packages:

- Use an appropriate loss function
- Design a composite function $F(\boldsymbol{w}, x, y)$

Choices in deep learning packages:

- Use an appropriate loss function
- Design a composite function $F(\boldsymbol{w}, x, y)$

Know what you are doing, i.e., know all the dimensions.

Loss functions:

- CrossEntropyLoss

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j])))
               = -x[class] + log(\sum_j exp(x[j]))
```

- NLLLoss (negative log-likelihood)

```
loss(x, class) = -x[class]
```

- MSELoss (mean squared error)

```
loss(x, y) = 1/n \sum_i |x_i - y_i|^2
```

- BCELoss (binary cross-entropy)

```
loss(o,t)=-1/n \sum_i i(t[i]*log(o[i])+(1-t[i])*log(1-o[i]))
```

- BCEWithLogitsLoss

```
loss(o,t)=-1/n\sum_i(t[i]*log(sigmoid(o[i]))
                +(1-t[i])*log(1-sigmoid(o[i])))
```

- L1Loss
- KLDivLoss

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Intended to be used in combination with 'LogSoftmax':

$$f_i(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Intended to be used in combination with ‘LogSoftmax’:

$$f_i(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

Why?

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Intended to be used in combination with ‘LogSoftmax’:

$$f_i(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

Why? Numerical robustness (‘log-sum-exp trick’)

$$\log \sum_j \exp x_j = c + \log \sum_j \exp \left(x_j - c\right)$$

Don’t try without, it **will** fail!

Popular architectures:

Popular architectures:

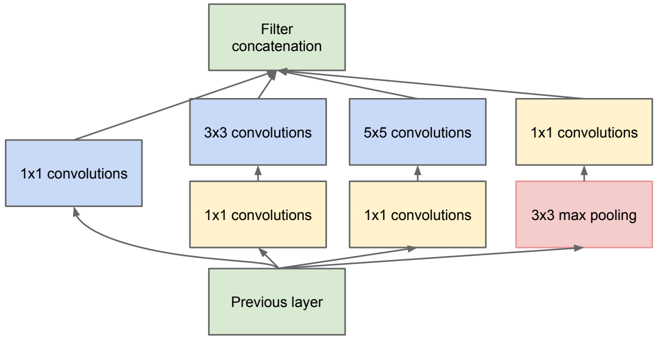
- LeNet
- AlexNet

Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)

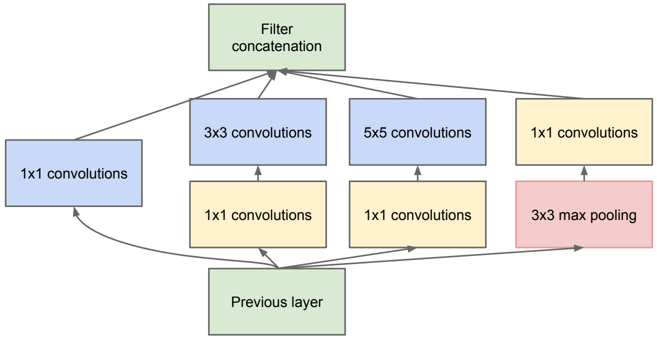
Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)
- GoogLeNet (inception module)

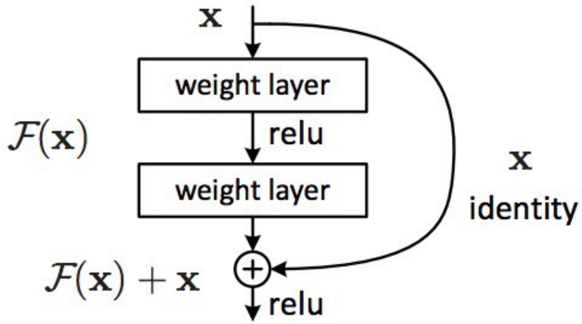


Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)
- GoogLeNet (inception module)



- ResNet (residual connections)



Quiz:

Quiz:

- What are deep nets?

Quiz:

- What are deep nets?
- How do deep nets relate to linear models?

Quiz:

- What are deep nets?
- How do deep nets relate to linear models?
- What is backpropagation in deep nets?

Quiz:

- What are deep nets?
- How do deep nets relate to linear models?
- What is backpropagation in deep nets?
- What components of deep nets do you know?

Quiz:

- What are deep nets?
- How do deep nets relate to linear models?
- What is backpropagation in deep nets?
- What components of deep nets do you know?
- What algorithm is used to train deep nets?

Important topics of this lecture

- Deep nets
- Backpropagation

Up next:

- More deep learning architectures