Multimodal Machine Learning

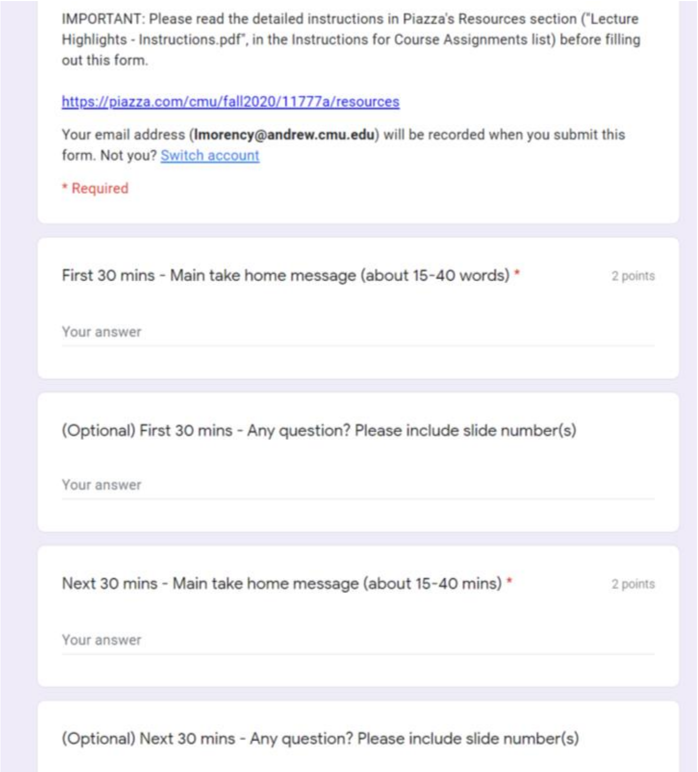
Lecture 2.2: Basic Concepts – Network Optimization

Louis-Philippe Morency

** Original course co-developed with Tadas Baltrusaitis.*
Spring 2021 edition taught by Yonatan Bisk

Administrative Stuff

Lecture Highlight Form



IMPORTANT: Please read the detailed instructions in Piazza's Resources section ("Lecture Highlights - Instructions.pdf", in the Instructions for Course Assignments list) before filling out this form.

https://piazza.com/cmu/fall2020/11777a/resources

Your email address (**lmorency@andrew.cmu.edu**) will be recorded when you submit this form. Not you? Switch account

* Required

First 30 mins - Main take home message (about 15-40 words) * 2 points

Your answer

(Optional) First 30 mins - Any question? Please include slide number(s)

Your answer

Next 30 mins - Main take home message (about 15-40 mins) * 2 points

Your answer

(Optional) Next 30 mins - Any question? Please include slide number(s)

Deadline: Today, Thursday at 11:59pm ET

Use your Andrew CMU email

- You will need to login using this address

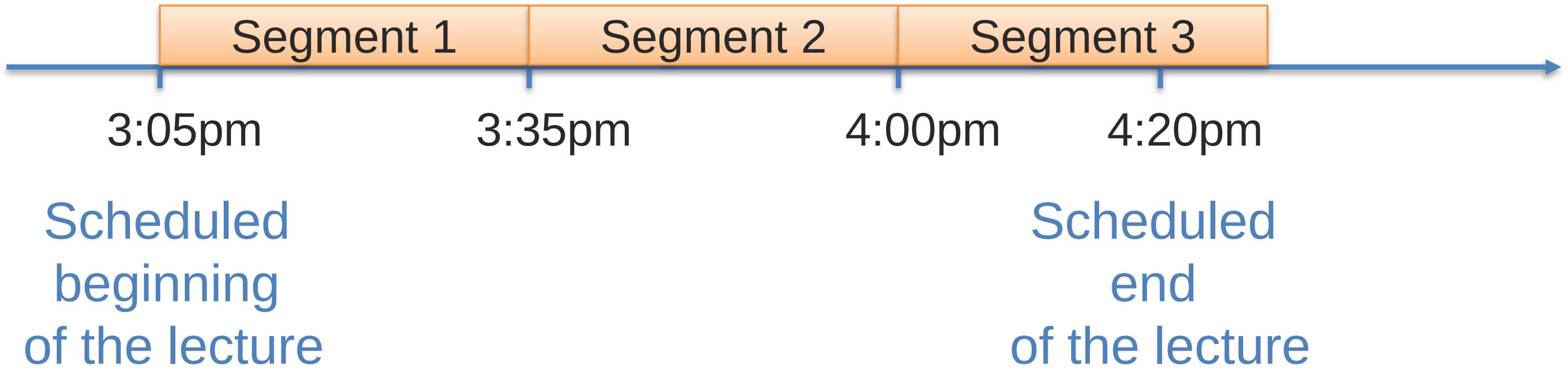
New form for each lecture

- Posted on Piazza’s Resources section

You should start taking notes now!

Contact us if you have any problem

Lecture Highlight Form - Segments



- Segment 1 starts at 3:05pm, even if the lecture starts slightly later.
- Segment 3 ends whenever the lecture ends
- Slides happening around the segment borders (+/- 5min of 3:35pm and 4:00pm) can be included in either neighboring segment.

Reading Assignments – Weekly Schedule

Four main steps for the reading assignments

1. Monday 8pm: Official start of the assignment
2. Wednesday 8pm: Select your paper
3. **Friday 8pm:** Post your summary
4. **Monday 8pm:** Post your extra comments (2 posts)

Team Matching Event – Today!

Today around 4:05pm ET

(later part of the lecture)

- Detailed instructions will be shared during lecture
- Event optional for students who already have a full team

Language Technologies Institute

Carnegie Mellon University

Multimodal Machine Learning

Lecture 2.2: Basic Concepts – Network Optimization

Louis-Philippe Morency

** Original course co-developed with Tadas Baltrusaitis.*
Spring 2021 edition taught by Yonatan Bisk

Lecture Objectives

- Learning neural networks
  - Optimization
  - Gradient computation
- Practical Deep Model Optimization
  - Adaptive Optimization Methods
  - Regularization
  - Co-adaptation
  - Multimodal Optimization

Basic Concepts:
Loss Function

Linear Classification: Loss Function

(or cost function or objective)

Scores $f(x_i; W)$

Label $y_i = 2\ (dog)$

→ **Loss** $L_i = ?$

Image x_i

(Size: 32*32*3)

0 (duck) ?	-12.3
1 (cat) ?	45.6
2 (dog) ?	98.7
3 (pig) ?	12.2
4 (bird) ?	-45.3

Multi-class problem

How to assign only one number representing how "unhappy" we are about these scores?

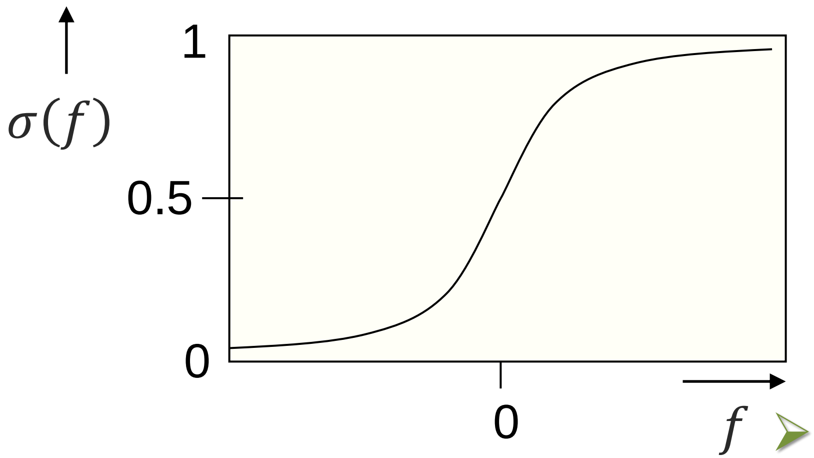
The loss function quantifies the amount by which the prediction scores deviate from the actual values.

A first challenge: how to normalize the scores?

First Loss Function: Cross-Entropy Loss

(or logistic loss)

Logistic function: $$\sigma(f) = \frac{1}{1 + e^{-f}}$$



➢ Score function

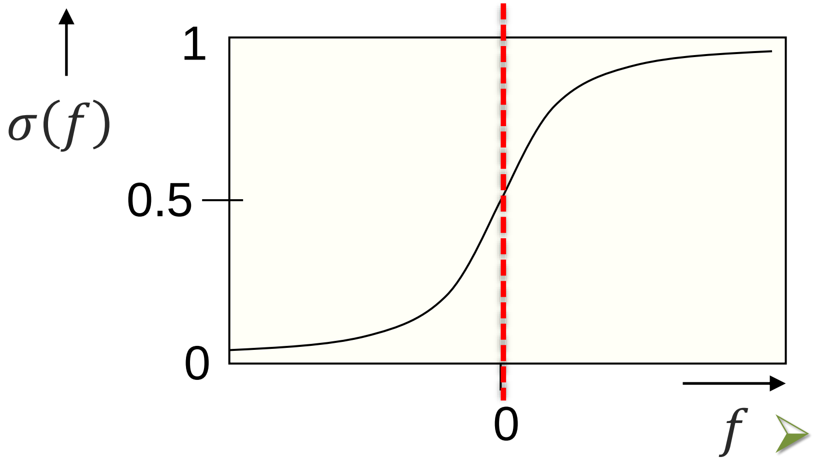
First Loss Function: Cross-Entropy Loss

(or logistic loss)

Logistic function: $$\sigma(f) = \frac{1}{1 + e^{-f}}$$

Logistic regression: (two classes) $$p(y_i = "dog"|x_i; w) \;\; = \sigma(w^T x_i)$$

= true
for two-class problem



➢ **Score function**

First Loss Function: Cross-Entropy Loss

(or logistic loss)

Logistic function: $$\sigma(f) = \frac{1}{1 + e^{-f}}$$

Logistic regression:
(two classes)

$$p(y_i = \text{"}dog\text{"} | x_i; w) \quad = \sigma(w^T x_i)$$

= true
for two-class problem

Softmax function:
(multiple classes)

$$p(y_i | x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

First Loss Function: Cross-Entropy Loss

(or logistic loss)

Cross-entropy loss:

$$L_i = \boxed{-\log\left(\boxed{\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}}\right)}$$

Softmax function

Minimizing the negative log likelihood.

Second Loss Function: Hinge Loss

(or max-margin loss or Multi-class SVM loss)

$$\boxed{L_i} = \boxed{\sum_{j \neq y_i}} \max(0, \boxed{f(x_i, W)_j - f(x_i, W)_{y_i}} + \Delta)$$

loss due to example i

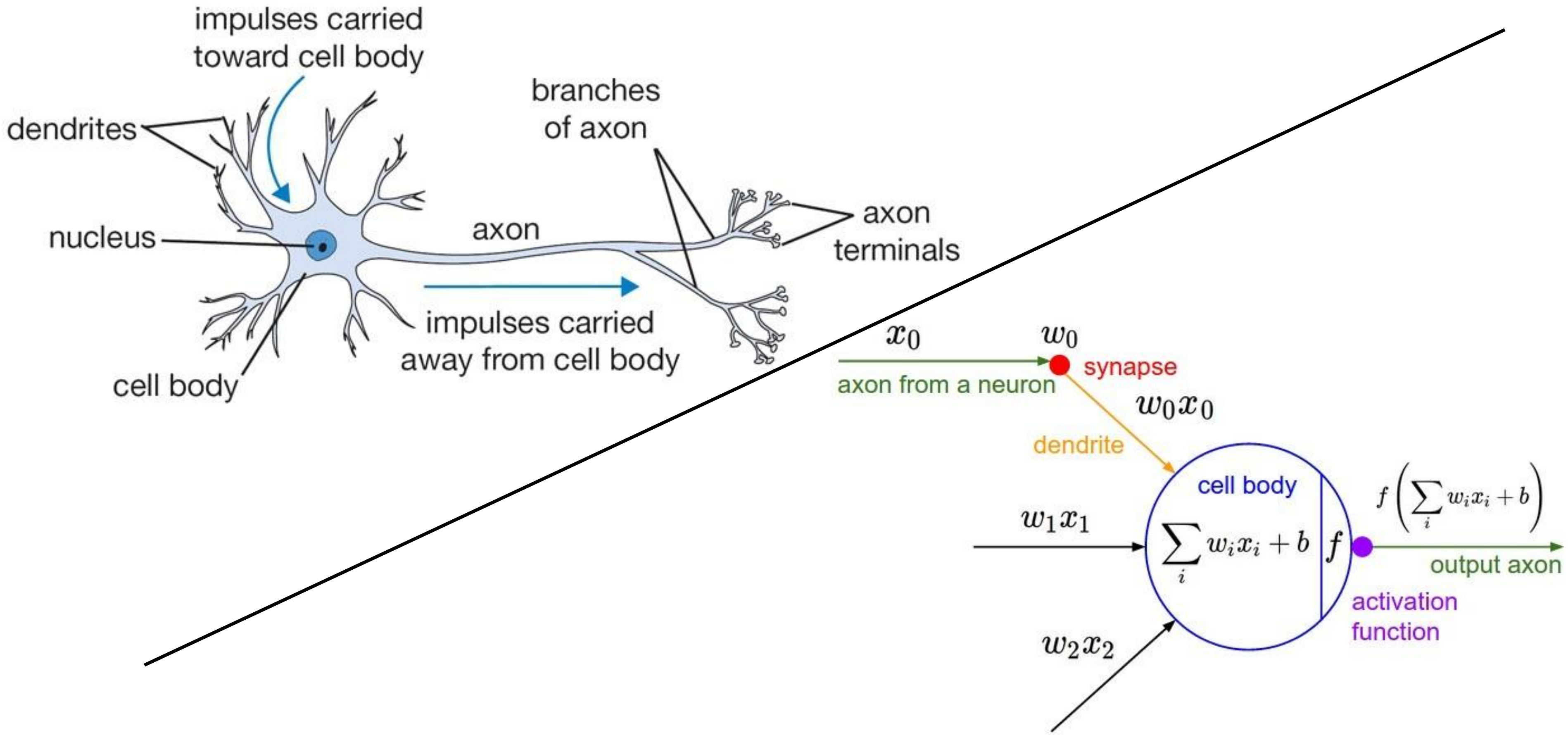
sum over all incorrect labels

difference between the correct class score and incorrect class score

Basic Concepts: Neural Networks

Neural Networks – inspiration

- Made up of artificial neurons



Neural Networks

Weighted sum followed by an activation function

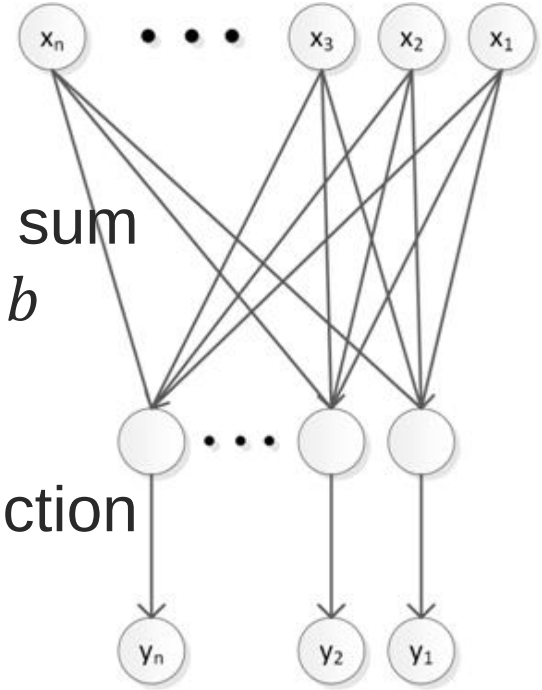
Input

Weighted sum

$Wx + b$

Activation function

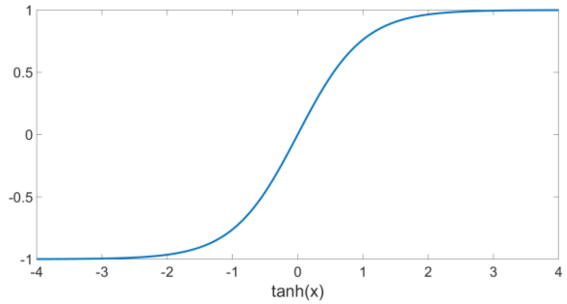
Output



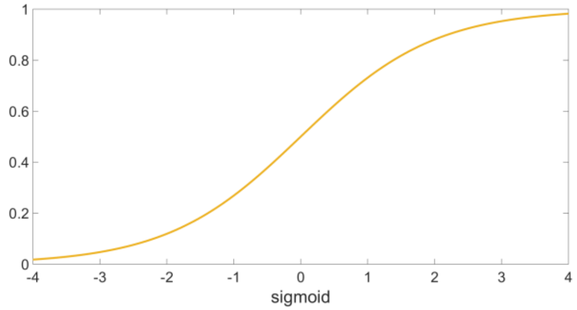
$$y = f(Wx + b)$$

Neural Networks – activation function

- $f(x) = \tanh(x)$

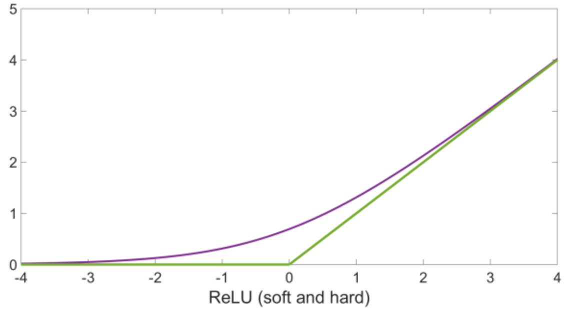


- Sigmoid - $f(x) = (1 + e^{-x})^{-1}$



- Linear – $f(x) = ax + b$

- ReLU $f(x) = \max(0, x) \sim \log(1 + \exp(x))$
  - Rectifier Linear Units
  - Faster training - no gradient vanishing
  - Induces sparsity



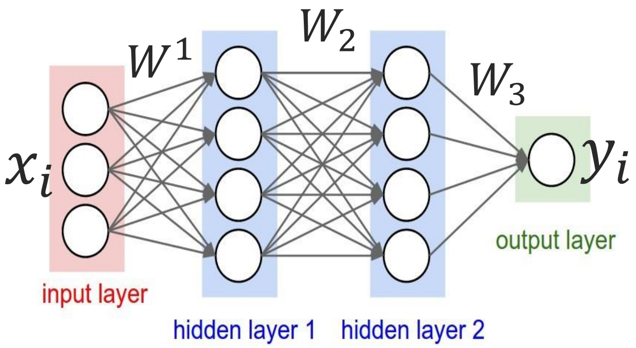
Multi-Layer Feedforward Network

Activation functions (individual layers)

$$f_{1;W_1}(x) = \sigma(W_1 x + b_1)$$

$$f_{2;W_2}(x) = \sigma(W_2 x + b_2)$$

$$f_{3;W_3}(x) = \sigma(W_3 x + b_3)$$



Score function

$$y_i = f(x_i) = f_{3;W_3}(f_{2;W_2}(f_{1;W_1}(x_i)))$$

Loss function (e.g., Euclidean loss)

$$L_i = (f(x_i) - y_i)^2 = (f_{3;W_3}(f_{2;W_2}(f_{1;W_1}(x_i))))^2$$

Optimization – Learning model parameters

Learning model parameters

We have our training data

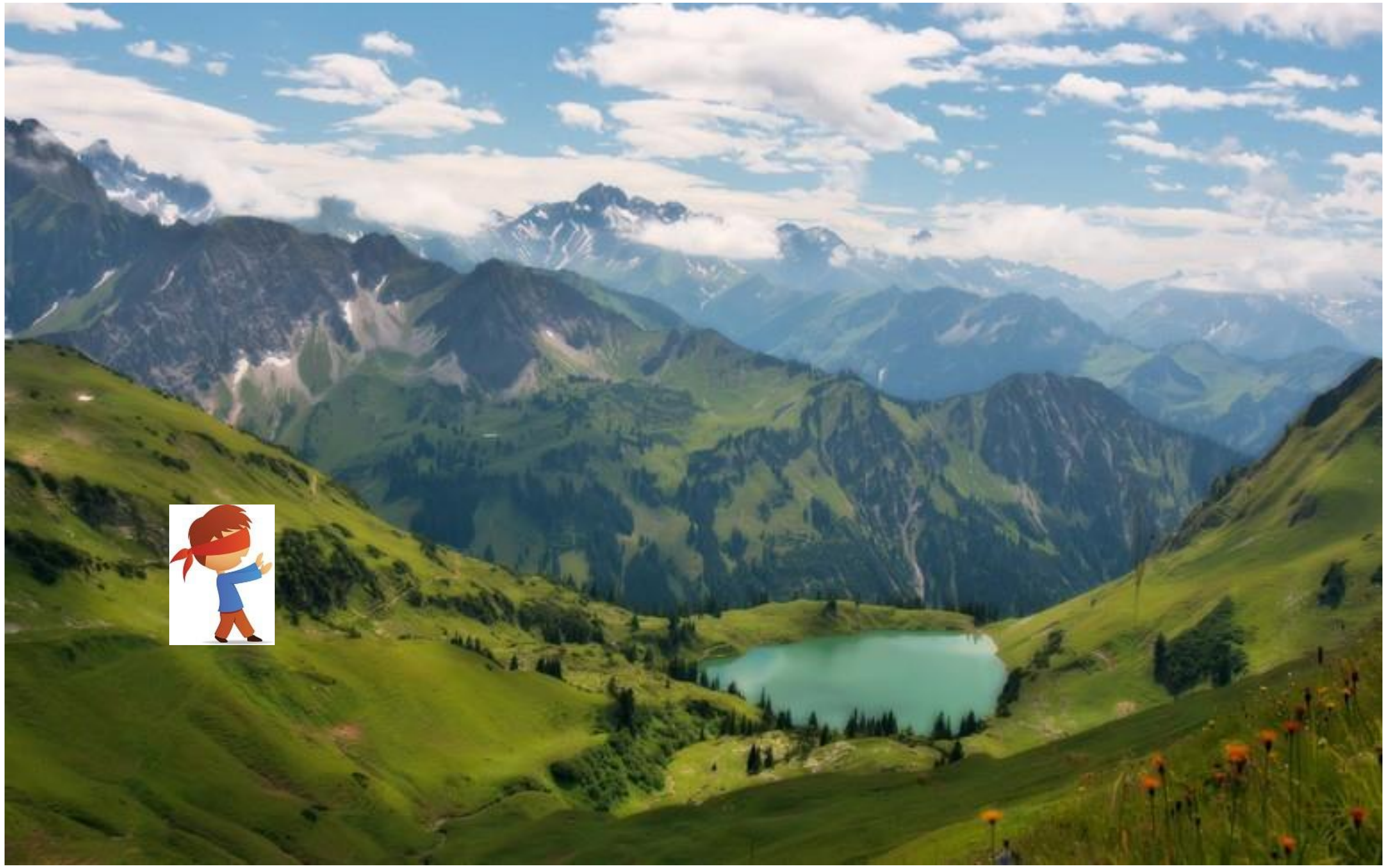
- $\mathrm{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ (e.g. images, videos, text etc.)
- $\mathrm{Y} = \{y_1, y_2, \ldots, y_n\}$ (labels)

We want to learn the W (weights and biases) that leads to best loss

$$\underset{W}{\operatorname{argmin}}[L(\mathrm{X}, \mathrm{Y}, W)]$$

The notation means find W for which $L(\mathrm{X}, \mathrm{Y}, W)$ has the lowest value

Optimization



Analytical gradient

If we know the function and it is **differentiable**

- Derivative/gradient is defined at every point in *f*
- Sometimes use differentiable approximations
- Some are locally differentiable

Examples:

$$f(x) = \frac{1}{1+e^{-x}};\ \frac{df}{dx} = (1-f(x))f(x)$$

$$f(x) = (x-y)^2;\ \frac{df}{dx} = 2(x-y)$$

How to follow the gradient

Many methods for optimization

- **Gradient Descent (actually the "simplest" one)**
- Newton methods (use Hessian – second derivative)
- Quasi-Newton (use approximate Hessian)
  - BFGS
  - LBFGS
  - Don't require learning rates (fewer hyperparameters)
  - But, do not work with stochastic and batch methods so rarely used to train modern Neural Networks

All of them look at the gradient

- Very few non gradient based optimization methods

Parameter Update Strategies

Gradient descent:

$$\theta^{(t+1)} = \theta^t - \epsilon_k \boxed{\nabla_\theta L}$$

Gradient of our loss function

New model parameters

Previous parameters

Learning rate at iteration *k*

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\boxed{\epsilon_\tau}$$

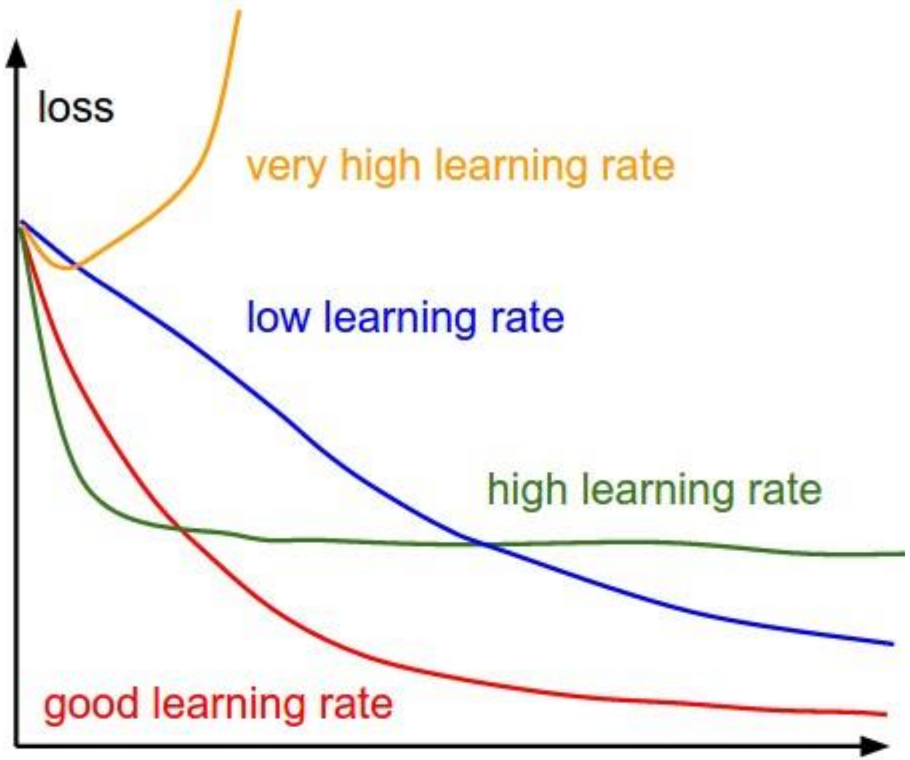
Decay learning rate linearly until iteration τ

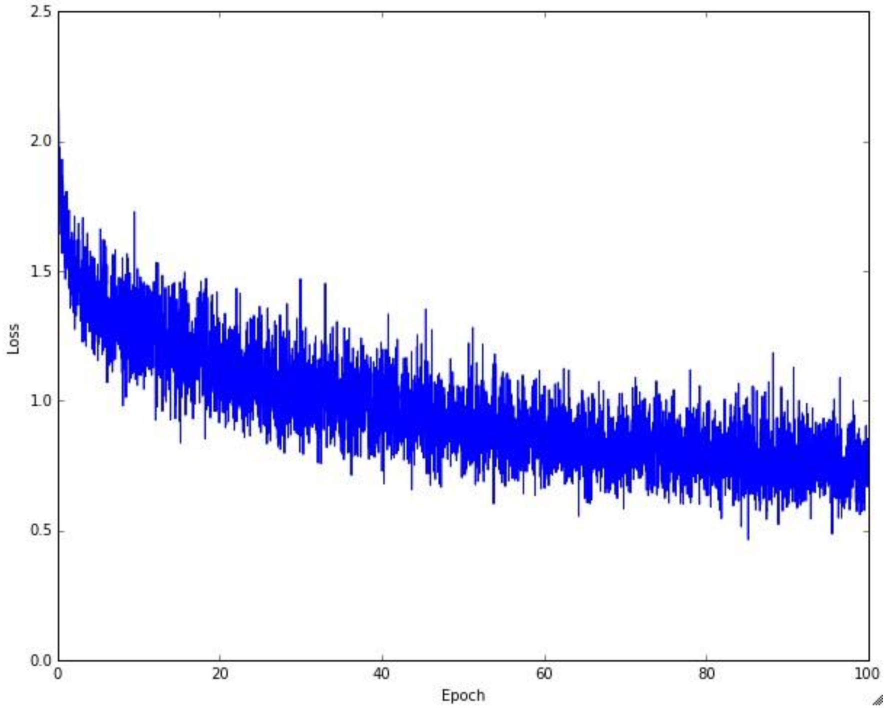
Learning rate at iteration *k*

Decay

Initial learning rate

Interpreting learning rates



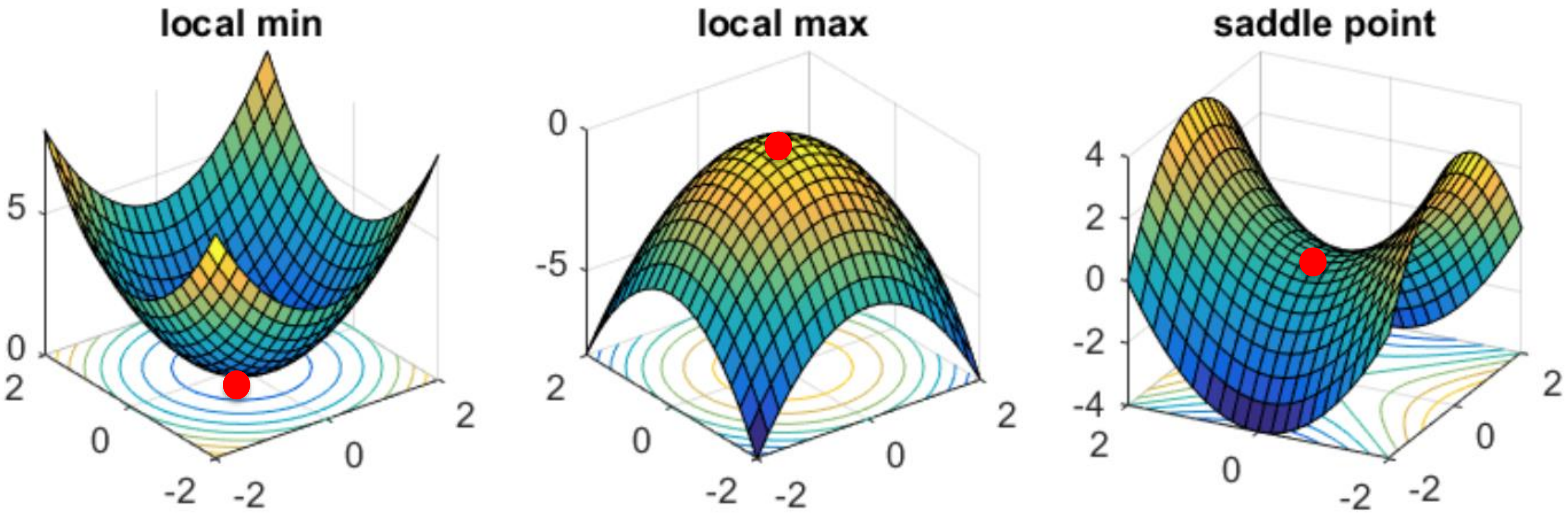


Optimization – Practical Guidelines

Optimization – Practical Guidelines

- **Adaptive Optimization Methods**
- Regularization
- Co-adaptation
- Pre-training

Critical Points



Detecting Saddles

One way to detect saddles:

- Calculate Hessian at point x
- If Hessian is indefinite you have a saddle for sure.
- If Hessian is not indefinite you really can't tell.

"My loss isn't changing"

- You are definitely close to a critical point
  - You may be in a saddle point
  - You may be in the local minima/maxima
- One trick: quickly check the surrounding
  - Best practical trick if Hessian is not indefinite.

Adaptive Learning Rate

Key Idea: Let neurons who just started learning have huge learning rate.

Adaptive Learning Rate is an active area of research:

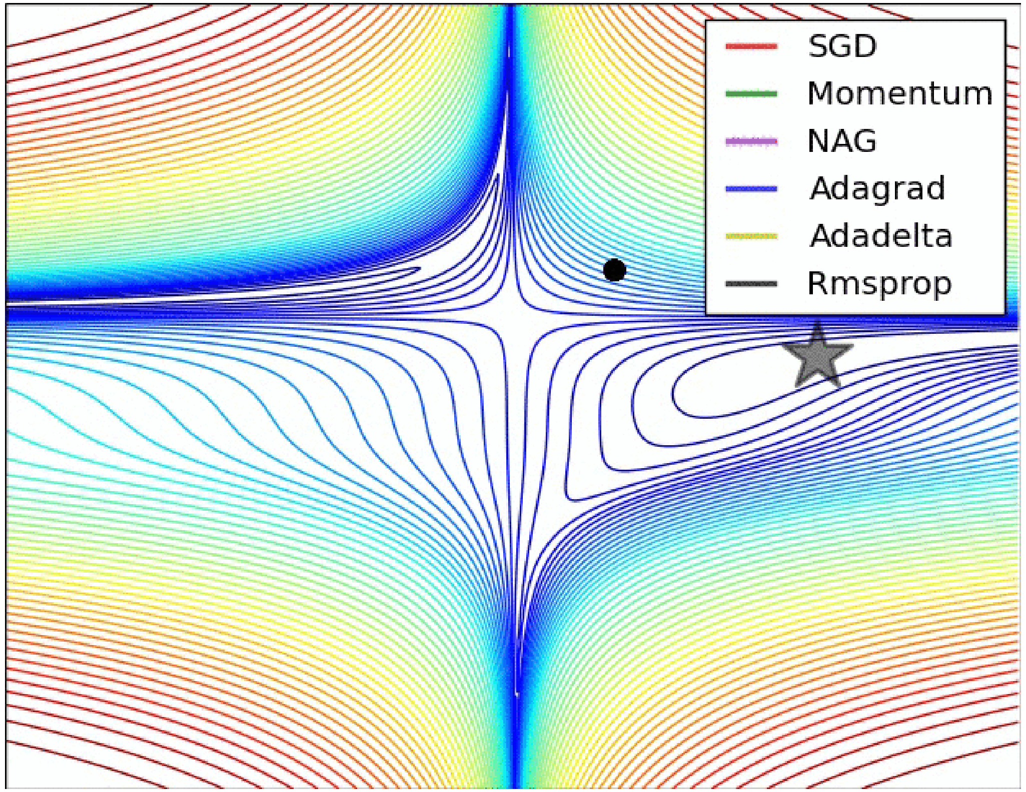
- Adadelta
- RMSProp

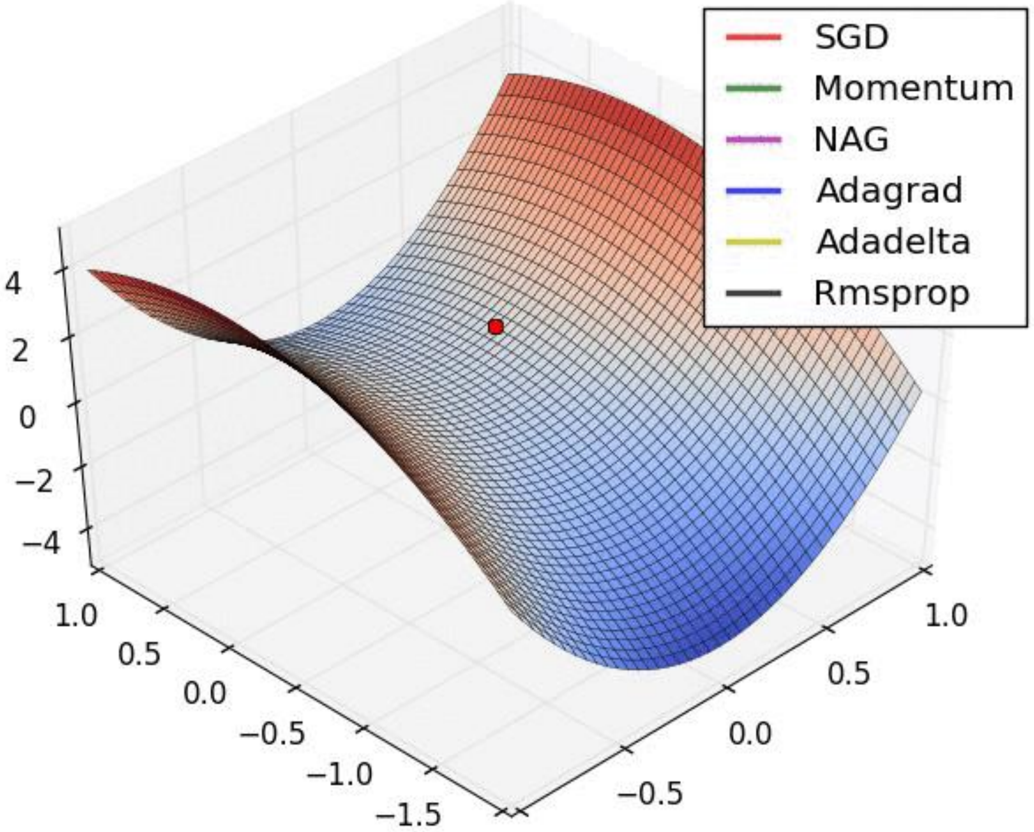
```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

- Adam

```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

Adaptive Learning Rate





Optimization – Practical Guidelines

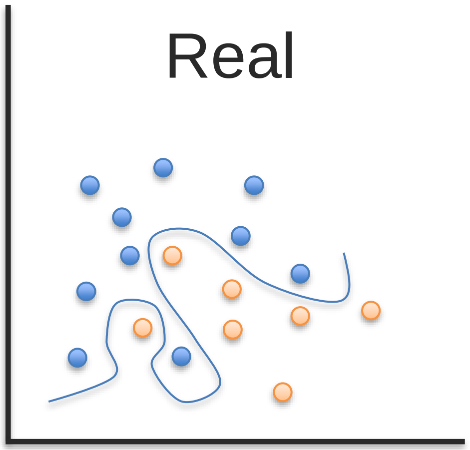
- Adaptive Optimization Methods
- **Regularization**
- Co-adaptation
- Pre-training

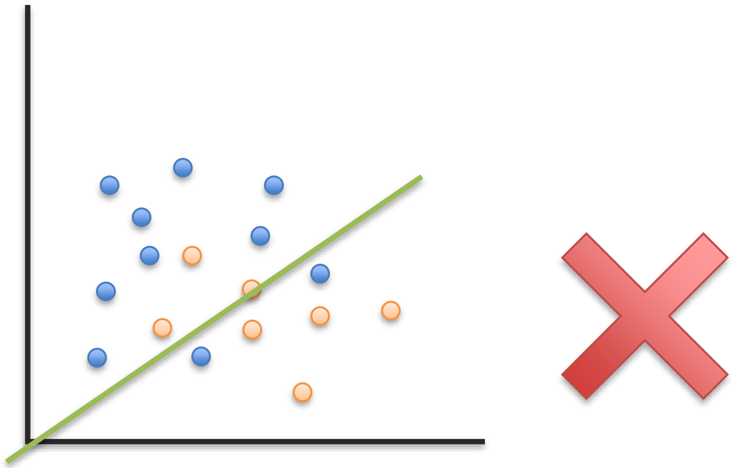
Bias-Variance

Problem of bias and variance

- Simple models are unlikely to find the solution to a hard problem, thus probability of finding the right model is low.

Not an issue these days!



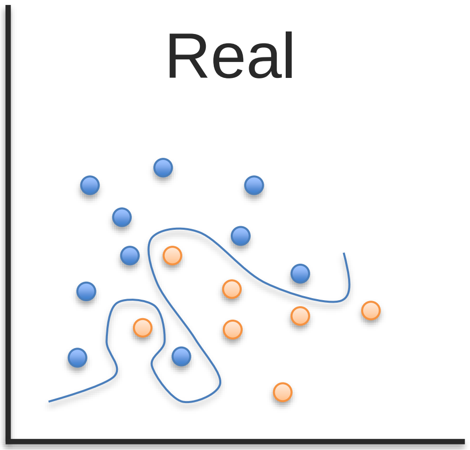


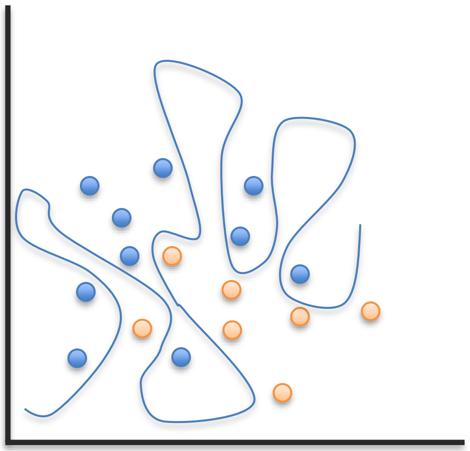
Bias-Variance

Problem of bias and variance

- Simple models are unlikely to find the solution to a hard problem, thus probability of finding the right model is low.
- Complex models find many solutions to a problem, thus probability of finding the right model is again low.

A big issue with deep learning!

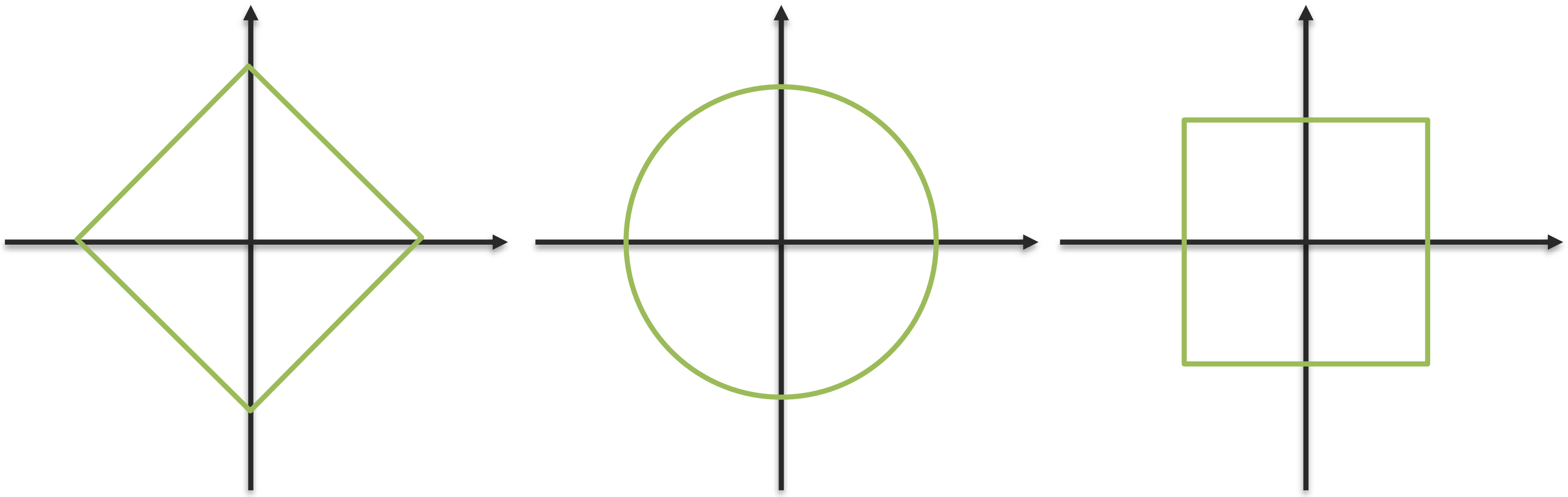




Parameter Regularization

Adding prior to the network parameters

- L^p Norms



L^1 $\quad$ L^2 $\quad$ L^∞

Minimize: $Loss(x;\theta) + \propto ||\theta||$

Parameter Regularization

Parameter Regularization

- L^1(Lasso) and L^2 (Ridge) are the most famous norms used.
  - Sometimes combined (Elastic)
- Other norms are computationally challenging.

Maximum a posteriori (MAP) estimation

- Having priors one the model parameters
- L^2 can be seen as a Gaussian prior on model parameters θ

Structural Regularization

Lots of models can learn everything.

- Go for simpler ones.

Occam's razor

Use task specific models:

- CNNs
- RecNNs
- LSTMs
- GRUs

Optimization – Practical Guidelines

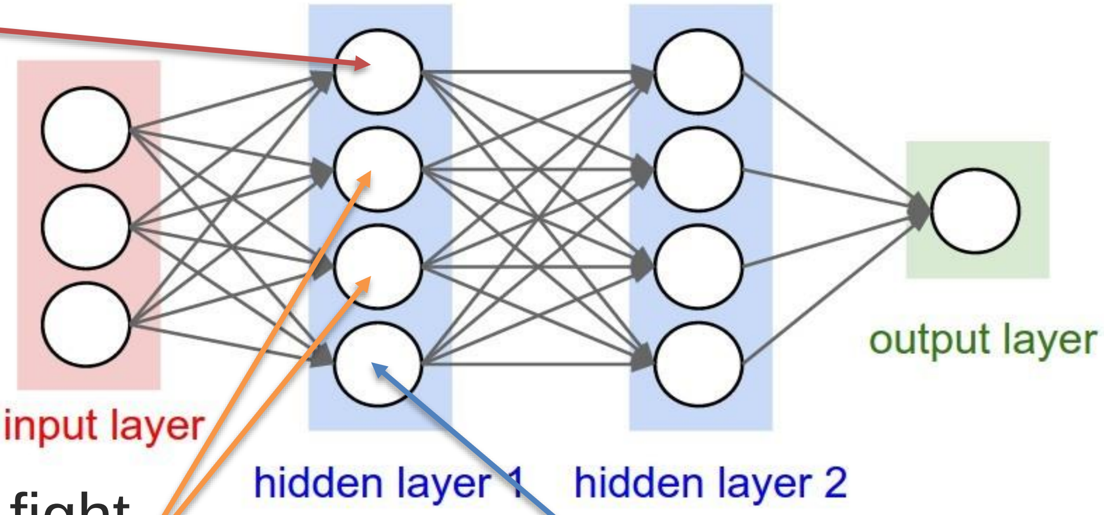
- Adaptive Optimization Methods
- Regularization
- **Co-adaptation**
- Pre-training

Co-Adaptation - Example

A neuron can learn something that is not useful:

1. It learn something useless
2. Other neurons learn to mitigate it.

Useless neuron

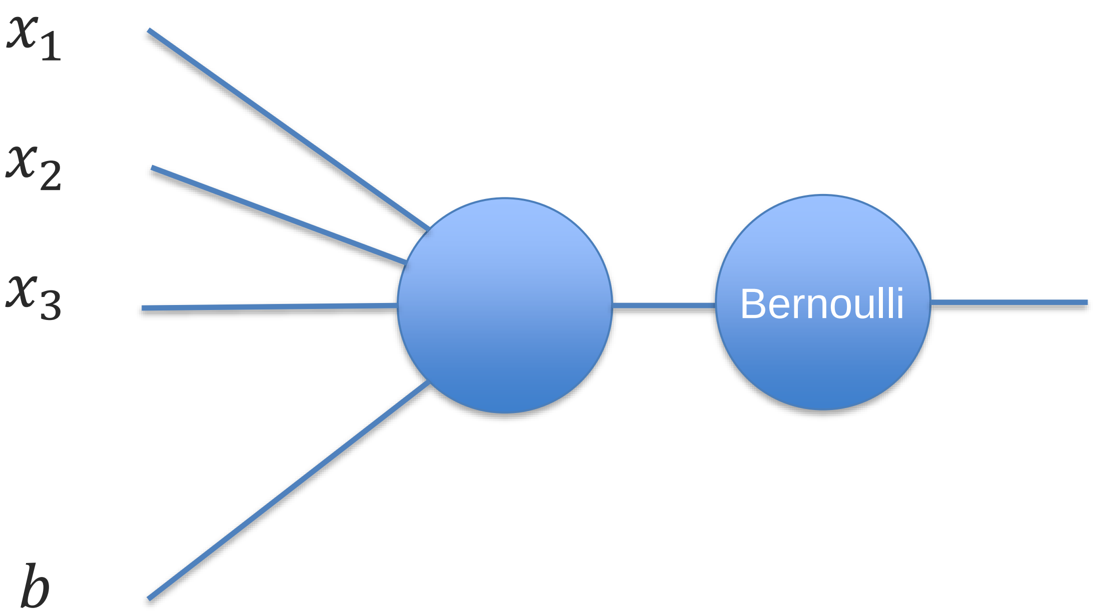


Learning to fight useless neuron

Actually learning something

Dropout

Simply multiply the output of a hidden layer with a mask of 0s and 1s (Bernoulli)



$$\begin{cases} p = 0.2 & y = 0 \\ p = 0.8 & y \end{cases}$$

Dropout

Forward step: multiply with a Bernoulli distribution per epoch, batch or sample point.

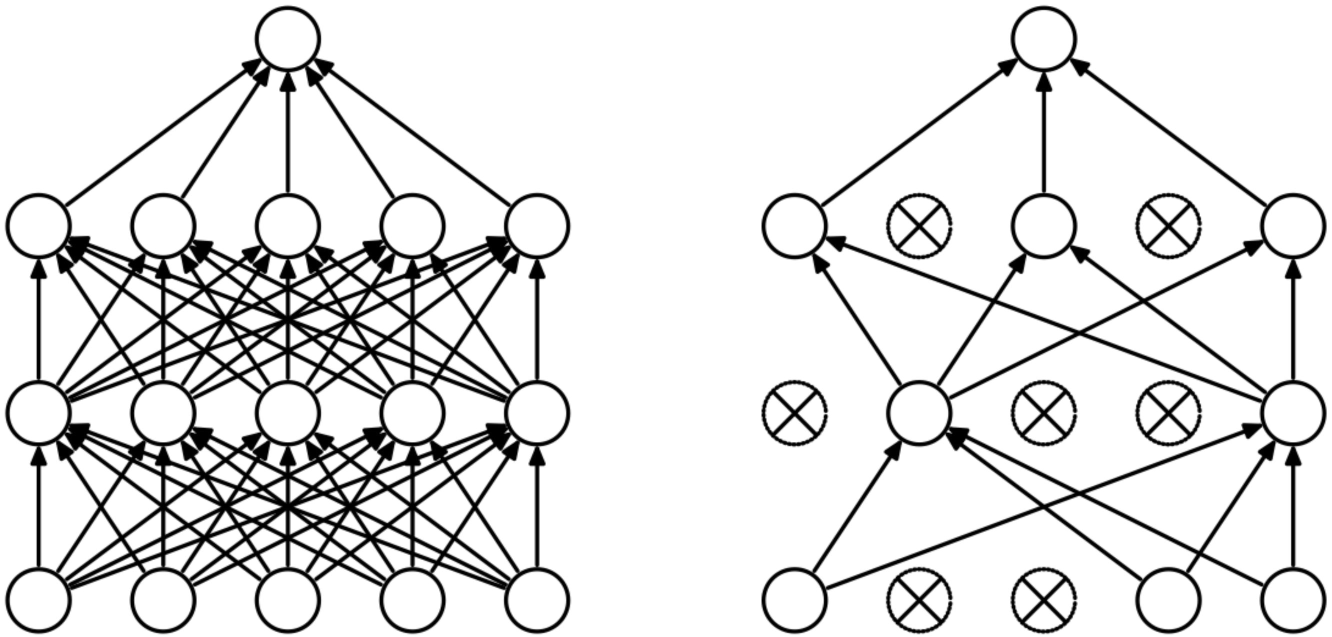
Backward step: just calculate the gradients same as before.

Question: some neurons are out of the network, so how does this work? All good? Nope!

Multiply the weights by $1 - p_i$

Dropout

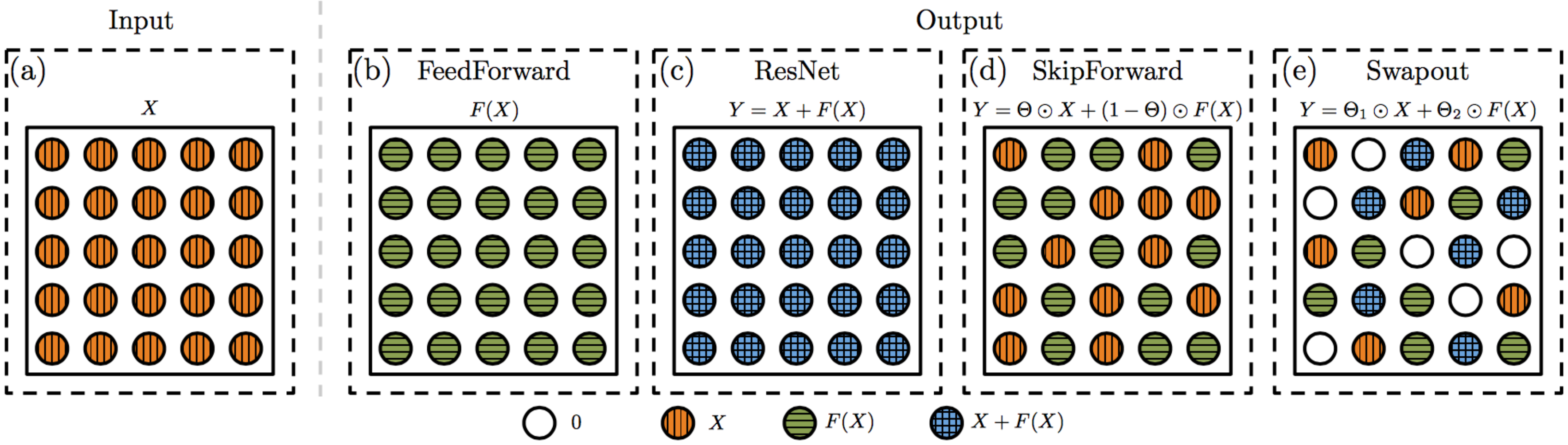
Stop co-adaptation + learn ensemble



Other variations

Gaussian dropout: instead of multiplying with a Bernoulli random variable, multiply with a Gaussian with mean 1.

Swapout: Allow skip-connections to happen

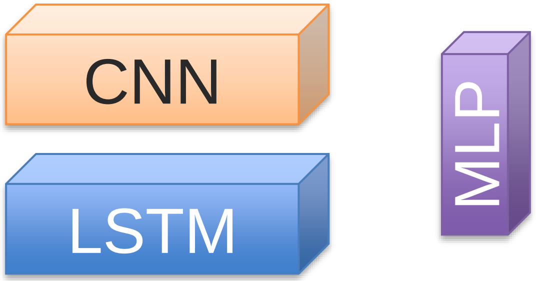


Optimization – Practical Guidelines

- Adaptive Optimization Methods
- Regularization
- Co-adaptation
- **Pre-training**

Optimization with Different Networks

Many multimodal problems are solved using different network architectures



Challenge: These networks may require different optimization strategies

Examples:

- CNNs work well with high decaying learning rate
- LSTMs work well with adaptive methods and normal SGD
- MLPs are very good with adaptive methods

Pre-Training

A potential solution for multimodal models:

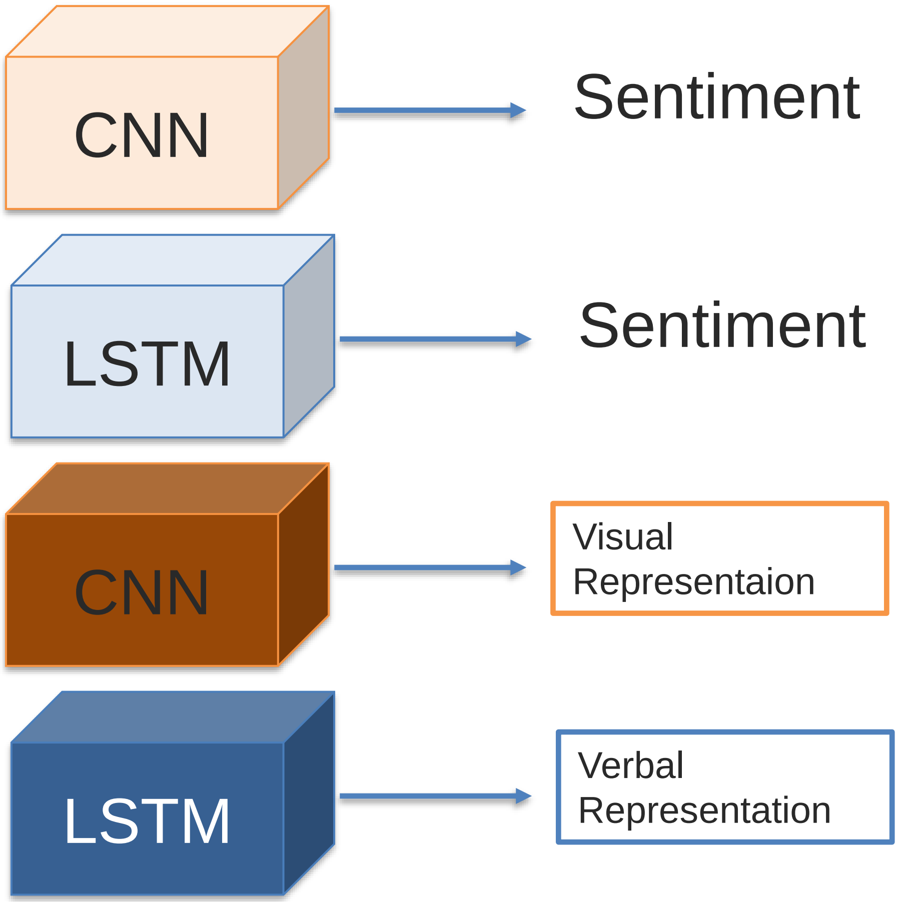
- Train each individual component of the model separately
- Put together and fine tune

An example: Multimodal Sentiment Analysis

Pre-training – Example (Multimodal Sentiment Analysis)

1. Unimodal Pre-training

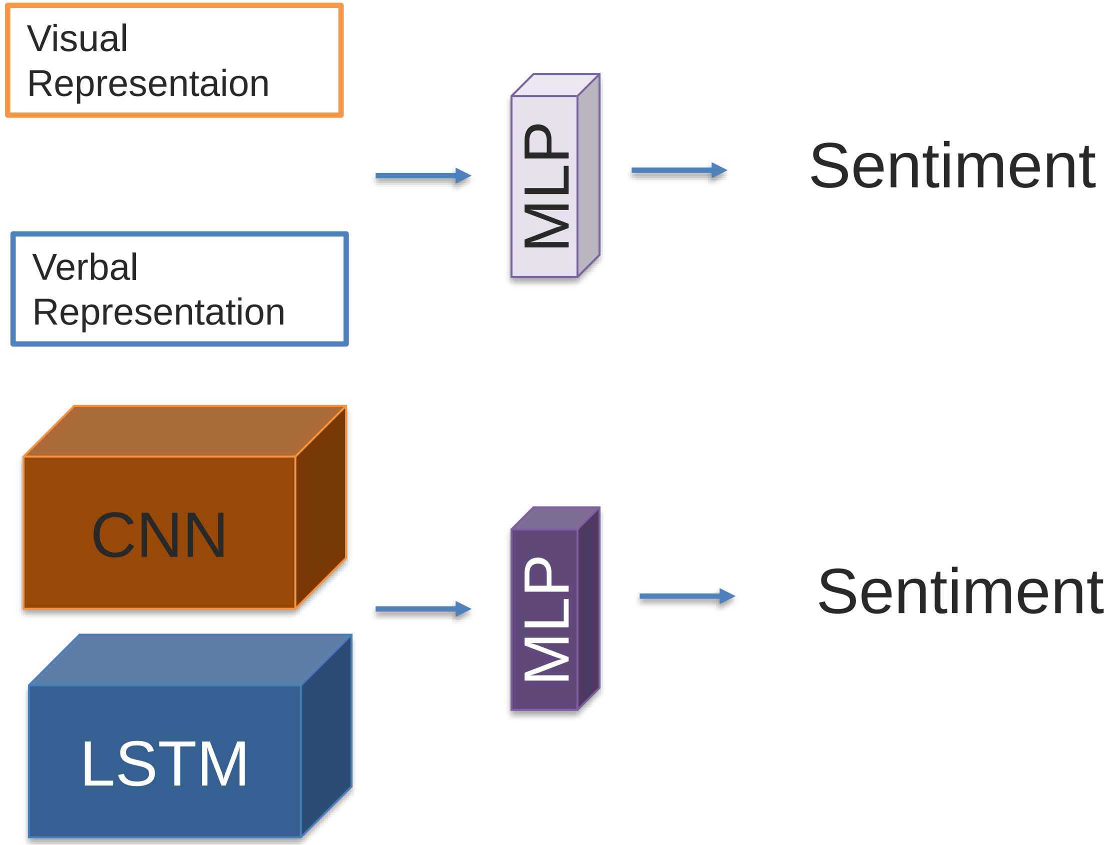
2. Encoding



Pre-training – Example (Multimodal Sentiment Analysis)

3. Fusion pre-training

4. Fine-tuning



Pre-training – Tricks

In the fine-tuning stage (4), it is better to not use adaptive methods such as Adam.

- Adam starts with huge momentum on all the networks parameters and can destroy the effects of pretraining.
- Simple SGD mostly helpful.

Initialization from other pre-trained models:

- VGG for CNNs
- Language models for RNNs
- Layer by layer training for MLPs

Team Matching Event