COMP 551 – Applied Machine Learning
Lecture 15: Neural Networks (cont'd)

Instructor: Joelle Pineau (*jpineau@cs.mcgill.ca)*

Class web page: *www.cs.mcgill.ca/~jpineau/comp551*

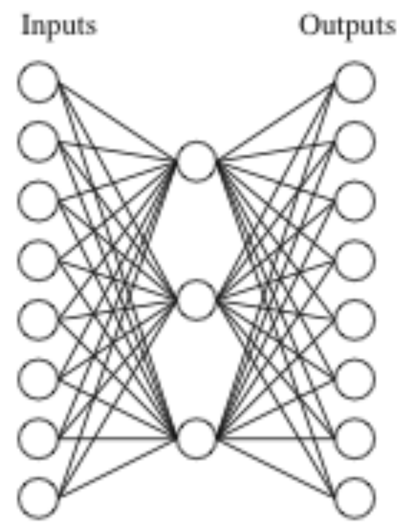
Learning the identity function

- Also called auto-regression.
- This a case of unsupervised learning.

Input		Output
10000000	$\longrightarrow$	10000000
01000000	$\longrightarrow$	01000000
00100000	$\longrightarrow$	00100000
00010000	$\longrightarrow$	00010000
00001000	$\longrightarrow$	00001000
00000100	$\longrightarrow$	00000100
00000010	$\longrightarrow$	00000010
00000001	$\longrightarrow$	00000001

Learning the identity function

- Neural network structure:



- Learned hidden layer weights:

(capture an alternate encoding of the data.)

Input		Hidden Layer				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Stochastic gradient descent for LMS loss

- Initialize all weights to small random numbers. — Initialization
- Repeat until convergence:
  - Pick a training example.
  - Feed example through network to compute output $o = o_{N+H+1}$.

Forward pass

  - For the output unit, compute the correction:

$$\delta_{N+H+1} \leftarrow o(1-o)(y-o)$$

  - For each hidden unit h, compute its share of the correction:

$$\delta_h \leftarrow o_h(1-o_h) w_{N+H+1,h} \delta_{N+H+1}$$

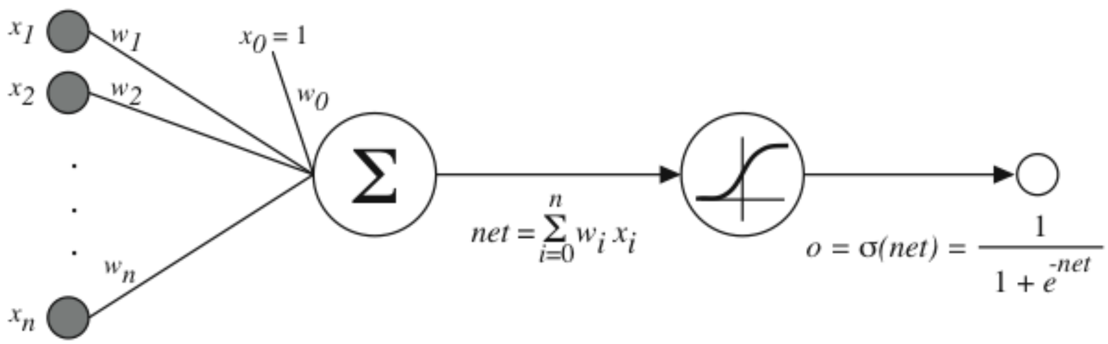
Backpro-pagation

  - Update each network weight:

$$w_{h,i} \leftarrow w_{h,i} + \alpha_{h,i} \delta_h x_{h,i}$$

Gradient descent

A family of sigmoid functions

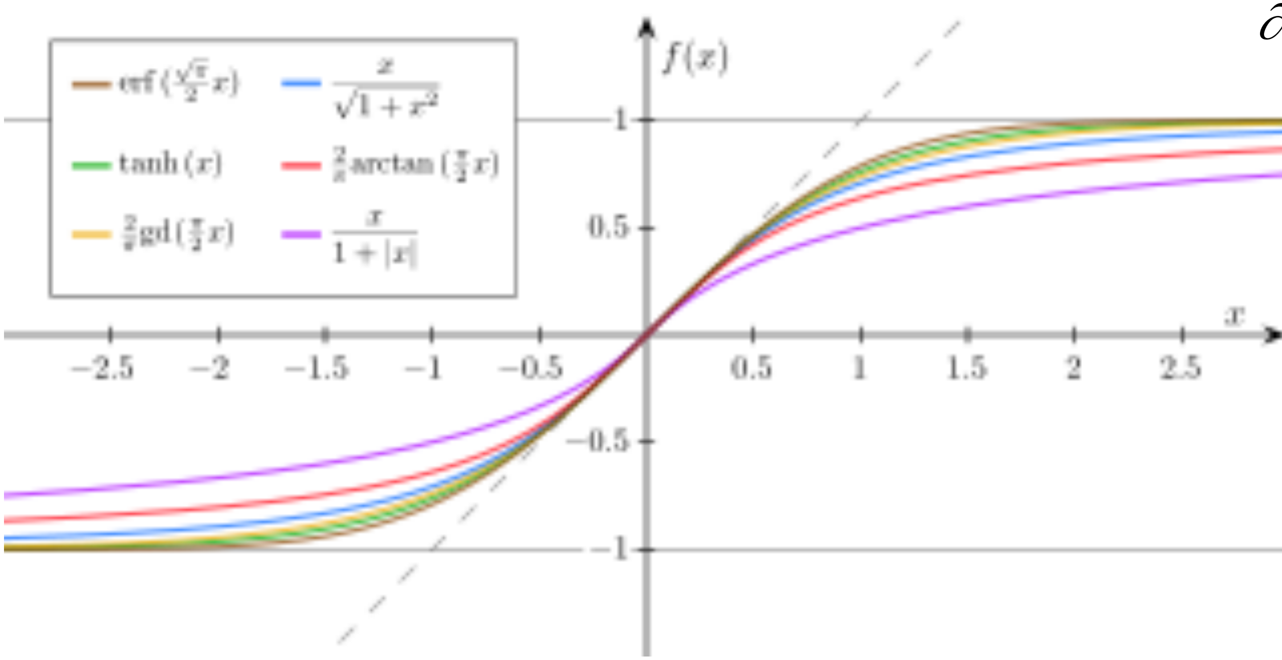


E.g.

$\sigma(z) = tanh(z)$

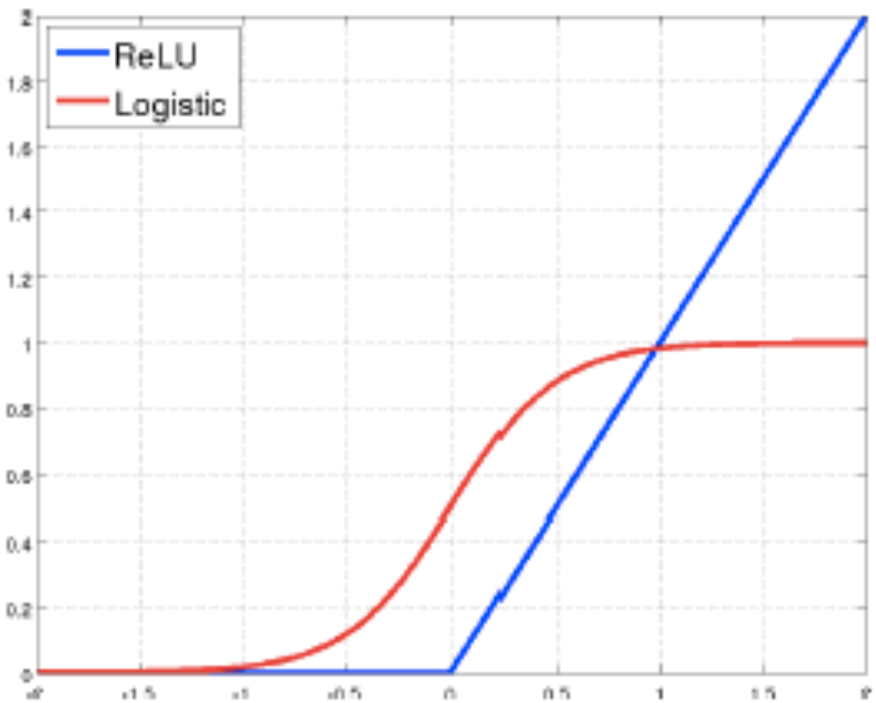
$tanh(z) = (e^z - e^{-z}) / (e^z + e^{-z})$

$\partial\sigma(z)/\partial z = 1-\sigma(z)^2$

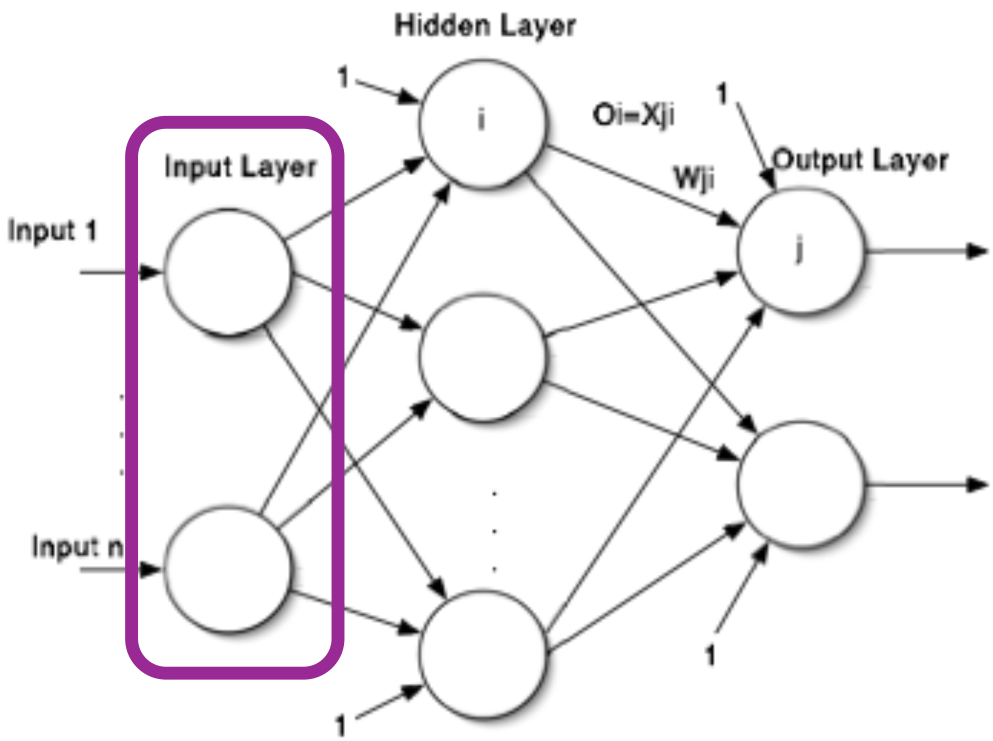


Rectified linear units

- Instead of using binary units, try *log(1+exp(Wx))*.
- Unit outputs linear function when input is positive, zero otherwise.
- Useful for speech processing and object recognition.



Encoding the input



Encoding the input: Discrete inputs

- Discrete inputs with *k* possible values are often encoded using a *1-hot* or *1-of-k* encoding:
  - *k* input bits are associated with the variable (one for each possible value).
  - For any instance, all bits are *0* except the one corresponding to the value found in the data, which is set to *1*.
  - If the value is missing, all inputs are set to *0*.

Encoding the input: Real-valued inputs

- Important to scale the inputs, so they have a common, reasonable range
- Standard transformation: normalize the data
  - To get mean=0, variance=1, subtract the mean and divide by the standard deviation
  - Works well if the data is roughly normal, but bad if we have outliers.

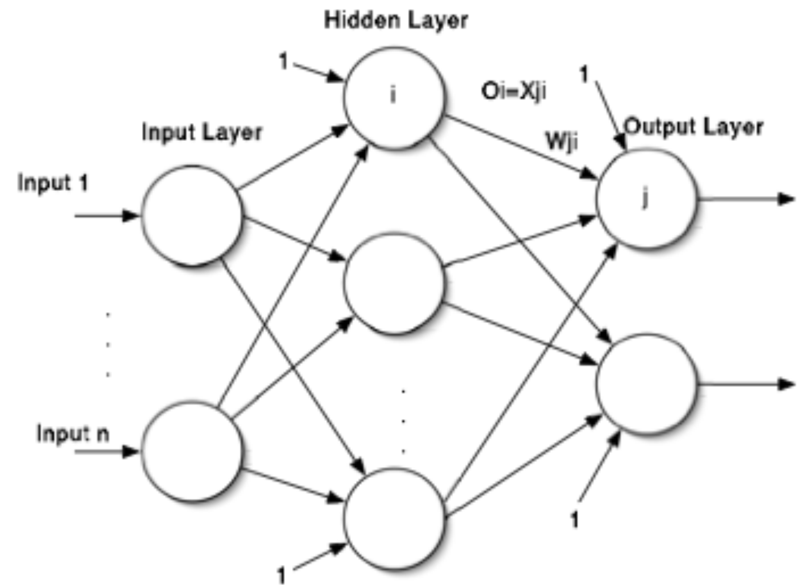
Encoding the input: Real-valued inputs

- Important to scale the inputs, so they have a common, reasonable range
- Standard transformation: normalize the data
  - To get mean=0, variance=1, subtract the mean and divide by the standard deviation
  - Works well if the data is roughly normal, but bad if we have outliers.

- Alternatives:
  - *1-to-n encoding*: discretize the variable into a given number of intervals *n*.
  - *Thermometer encoding*: like *1-to-n* but if the variable falls in the *i*=th interval, all bits *1..i* are set to 1.
  - The *thermometer encoding* is usually better than *1-to-n* encoding.

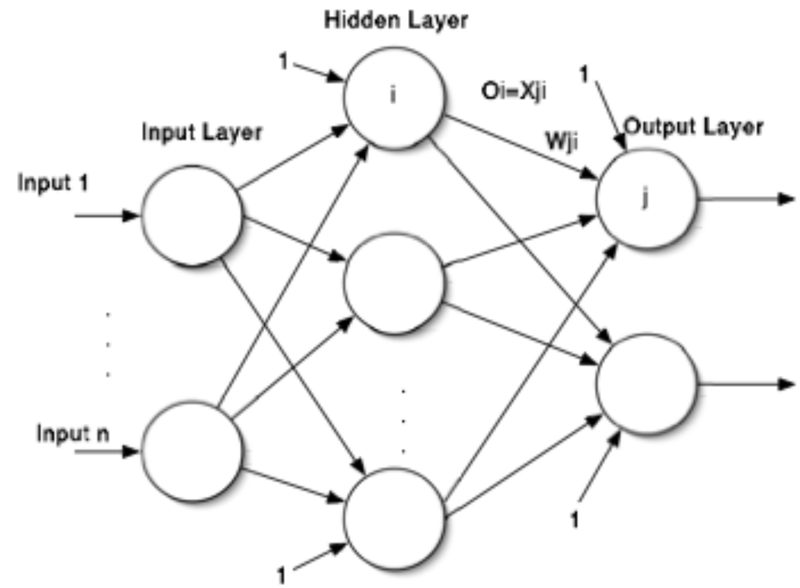
Encoding the output

- **Multi-class domains:**



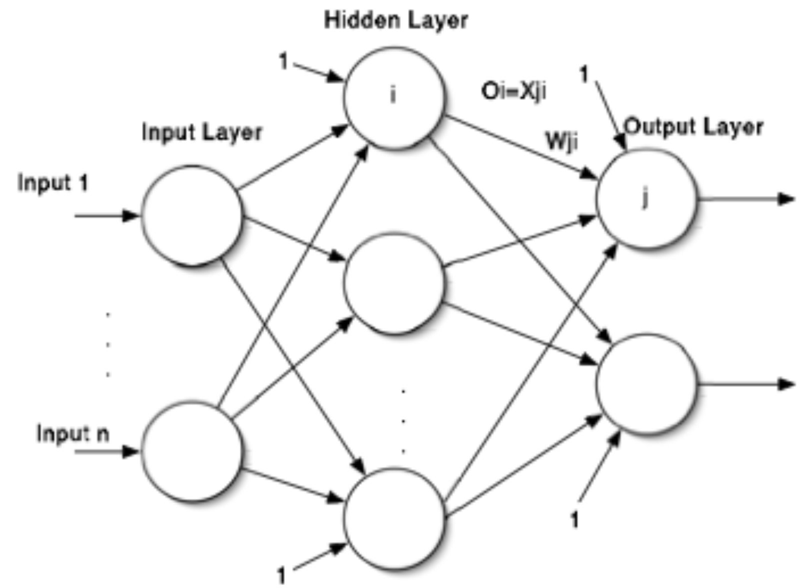
Encoding the output

- **Multi-class domains:**
  - Use a network with several output units: one per class
  - Compared to training multiple 1-vs-all classifiers, this allows shared weights at the hidden layers.



Encoding the output

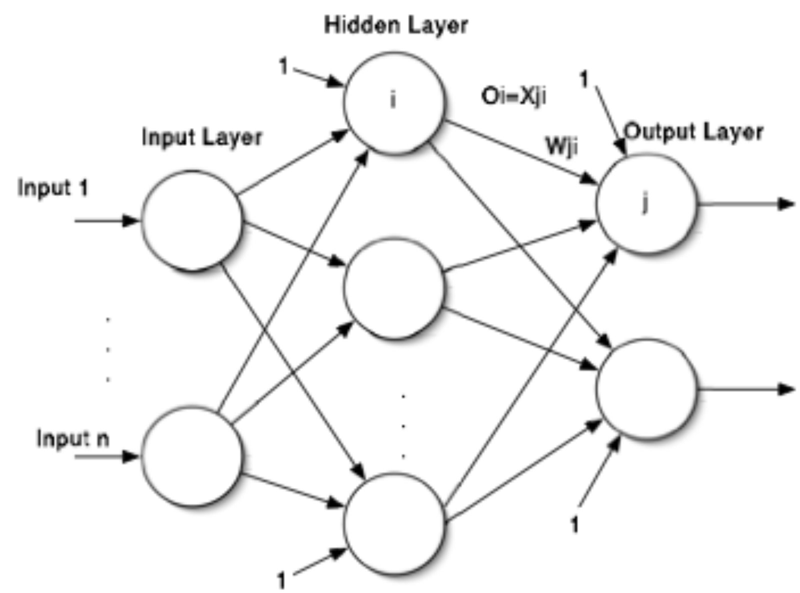
- **Multi-class domains:**
  - Use a network with several output units: one per class
  - Compared to training multiple 1-vs-all classifiers, this allows shared weights at the hidden layers.



- **Regression tasks:**

Encoding the output

- **Multi-class domains:**
  - Use a network with several output units: one per class
  - Compared to training multiple 1-vs-all classifiers, this allows shared weights at the hidden layers.



- **Regression tasks:**
  - Use a network with several output sigmoid units, corresponding to encoding of different output ranges of output value.
  - Use an output unit without a sigmoid function (i.e. just the weighted linear combination) to get full range of output values.

Network architecture

- Overfitting occurs if there are too many parameters compared to the amount of data available.
- Choosing the number of hidden units
  - Too few hidden units do not allow the concept to be learned.
  - Too many lead to slow learning and overfitting.
  - If the m inputs are binary, $\log m$ is a good heuristic choice.

Network architecture

- Overfitting occurs if there are too many parameters compared to the amount of data available.
- Choosing the number of hidden units
  - Too few hidden units do not allow the concept to be learned.
  - Too many lead to slow learning and overfitting.
  - If the m inputs are binary, log m is a good heuristic choice.
- Choosing the number of layers
  - Always start with **one** hidden layer.
  - Add one at a time, see if solution improves on validation set.

Convergence of backpropagation

- Backpropagation = gradient descent over **<u>all parameters</u>** in network.
- If the learning rate is appropriate, the algorithm is guaranteed to converge to a **<u>local minimum</u>** of the cost function.

Convergence of backpropagation

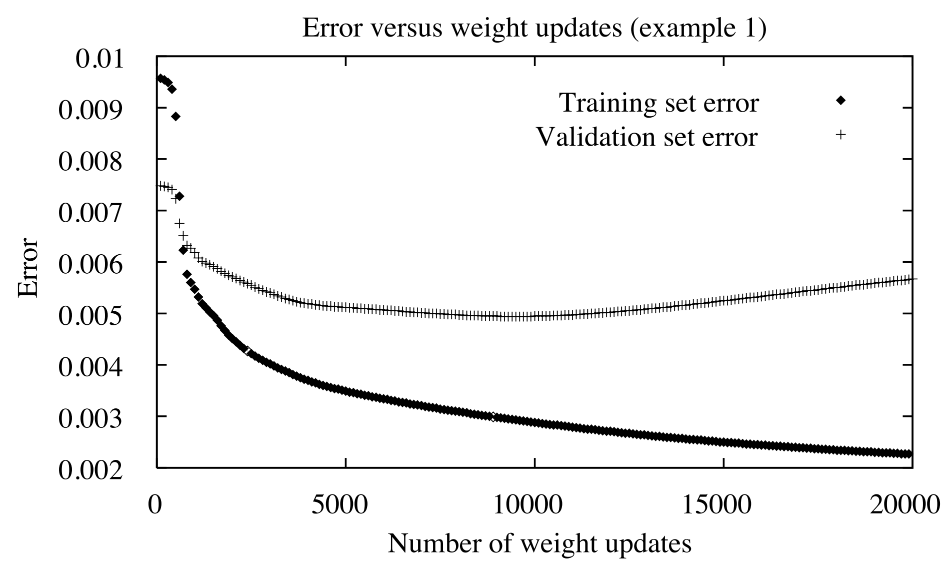
- Backpropagation = gradient descent over **all parameters** in network.
- If the learning rate is appropriate, the algorithm is guaranteed to converge to a **local minimum** of the cost function.
  - NOT the global minimum. (Can be much worse.)
  - There can be MANY local minimum.
  - Use random restarts = train multiple nets with different initial weights.
  - In practice, the solution found is often good (try a few parallel restarts).

Convergence of backpropagation

- Backpropagation = gradient descent over **all parameters** in network.
- If the learning rate is appropriate, the algorithm is guaranteed to converge to a **local minimum** of the cost function.
  - NOT the global minimum. (Can be much worse.)
  - There can be MANY local minimum.
  - Use random restarts = train multiple nets with different initial weights.
  - In practice, the solution found is often good (try a few parallel restarts).
- Training can take thousands of iterations - **VERY SLOW**! **But using network after training is very fast.**
- Can we find solution faster (i.e. in fewer iterations)?

Overtraining

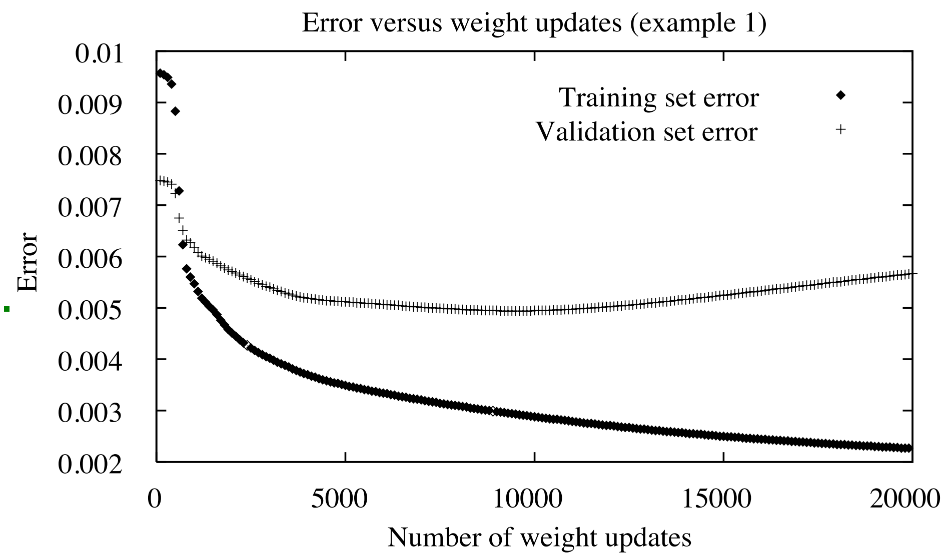
- Traditional overfitting is concerned with the number of parameters vs. the number of instances
- In neural networks: related phenomenon called overtraining occurs when weights take on large magnitudes, i.e. unit saturation
  - As learning progresses, the network has more active parameters.



Overtraining

- Traditional overfitting is concerned with the number of parameters vs. the number of instances
- In neural networks: related phenomenon called overtraining occurs when weights take on large magnitudes, i.e. unit saturation
  – As learning progresses, the network has more active parameters.

- Use validation set to decide when to stop training.
  Training horizon is a hyper-parameter.
- Regularization is also effective.



Regularization in neural networks

- Incorporate an L2 penalty: $J(w) = 0.5(y-h_w(x))^2 + 0.5\lambda w^T w$
  - Select λ with cross-validation.

- Can also use different penalties λ_1 , λ_2 for each layer.
  - Can be interpreted as a Bayesian prior over weights.

Choosing the learning rate

- Backprop is **very sensitive** to the choice of learning rate.
  - Too large ⇒ divergence.
  - Too small ⇒ VERY slow learning.
  - The learning rate also influences the ability to escape local optima.

- Very often, different learning rates are used for units in different layers. Hard to tune by hand!

- **Heuristic**: Track performance on validation set, when it stabilizes, divide learning rate by 2.

Optimization method: Adagrad

- Calculate adaptive learning rate per parameter.
- Intuition: Adapt learning rate depending on previous updates to that parameter.
  - Learn slowly for frequent features.
  - Learn faster for rare but informative features.
- Can add regularization term.

See: Duchi, Hazan, Singer (2011) *Adaptive subgradient methods for online learning and stochastic optimization*. JMLR.

Adding momentum

- On the t-th training sample, instead of the update:

$$\Delta w_{ij} \leftarrow \alpha_{ij} \delta_j x_{ij},$$

We do: $$\Delta w_{ij}(t) \leftarrow \alpha_{ij} \delta_j x_{ij} + \beta \Delta w_{ij}(t-1)$$

The <u>second term</u> is called <u>momentum</u>

Adding momentum

- On the t-th training sample, instead of the update:

$$\Delta w_{ij} \leftarrow \alpha_{ij}\delta_j x_{ij}$$

We do: $$\Delta w_{ij}(t) \leftarrow \alpha_{ij}\delta_j x_{ij} + \beta \Delta w_{ij}(t-1)$$

The <u>second term</u> is called <u>momentum</u>

Advantages:

- Easy to pass small local minima.
- Keeps the weights moving in areas where the error is flat.
- Increases the speed where the gradient stays unchanged.

Disadvantages:

- With too much momentum, it can get out of a global maximum!
- One more parameter to tune, and more chances of divergence.

More application-specific tricks

- If there is too little data, it can be **perturbed by random noise**; this helps escape local minima and gives more robust results.
  - In image classification and pattern recognition tasks, extra data can be generated, e.g., by applying transformations that make sense.

More application-specific tricks

- If there is too little data, it can be **perturbed by random noise**; this helps escape local minima and gives more robust results.
  - In image classification and pattern recognition tasks, extra data can be generated, e.g., by applying transformations that make sense.

- **Weight sharing** can be used to indicate parameters that should have the same value based on prior knowledge.
  - Each update is computed separately using backpropagation, then the tied parameters are updated with an average.

When to consider using NNs

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input).
- Output is discrete or real valued, or a vector of values.
- Possibly noisy data.
- Training time is not important.
- Form of target function is unknown.
- Human readability of result is not important.
- The computation of the output based on the input has to be fast.

Several applications

- Speech recognition and synthesis.
- Natural language understanding.
- Image classification, digit recognition.
- Financial prediction.
- Game playing strategies.
- Robotics.
- …..

In recent years, many state-of-the-art results obtained using **Deep Learning**.

Final notes

- What you should know:
  - Definition / components of neural networks.
  - Training by backpropagation.
  - Overfitting (and how to avoid it).
  - When to use NNs.
  - Some strategies for successful application of NNs.
- Project 2 peer review opening today. Due in 1 week.
- Additional information about neural networks:

  Video & slides from the Montreal Deep Learning Summer School:

  http://videolectures.net/deeplearning2017_larochelle_neural_networks/

  https://drive.google.com/file/d/0ByUKRdiCDK7-c2s2RjBiSms2UzA/view?usp=drive_web

  https://drive.google.com/file/d/0ByUKRdiCDK7-UXB1R1ZpX082MEk/view?usp=drive_web